# Developing new algorithms for parity games

Tom van Dijk
University of Twente, 2021

# OVERVIEW

## My background

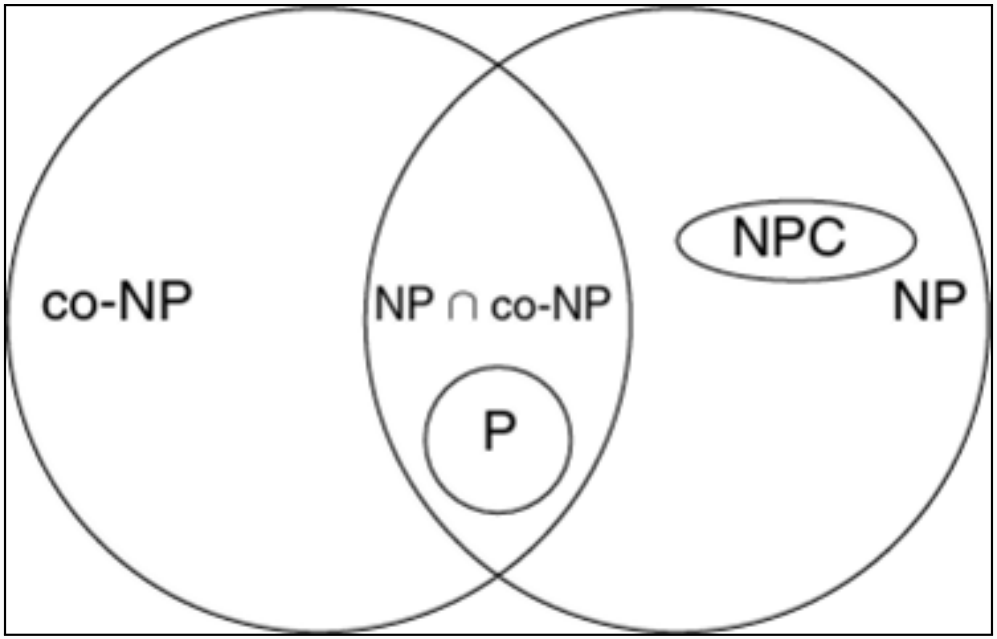
- 2003 – 2016: Enschede (BSc, MSc, PhD)
- 2016 – 2019: Linz (Postdoc)
- 2019 – now: Enschede (UD)
- PhD topic: parallelized binary decision diagrams & applications
- After PhD: mostly parity games and related topics

## Outline of the talk

- Motivation for parity games
- What are parity games
- What are some approaches to parity games
- How to develop new algorithms

# WHY PARITY GAMES?

- Want to work on algorithms? Find an open problem!
  `https://en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science`
- Famous open problem: P vs NP



# WHY PARITY GAMES?

## Parity games

- Are in NP ∩ co-NP
  a solution *and* a refutation can be checked in polynomial time
- But even in UP ∩ co-UP
  the unique solution/refutation can be checked in polynomial time
- Problems in NP ∩ co-NP are believed to be in P!
- Until 2017: all algorithms ran in exponential time.
- Now: various quasi-polynomial algorithms

$$\mathcal{O}(n^c) < \mathcal{O}(n^{\log n}) < \mathcal{O}(c^n)$$

- Goal: find a P-time algorithm or prove a >P lower bound

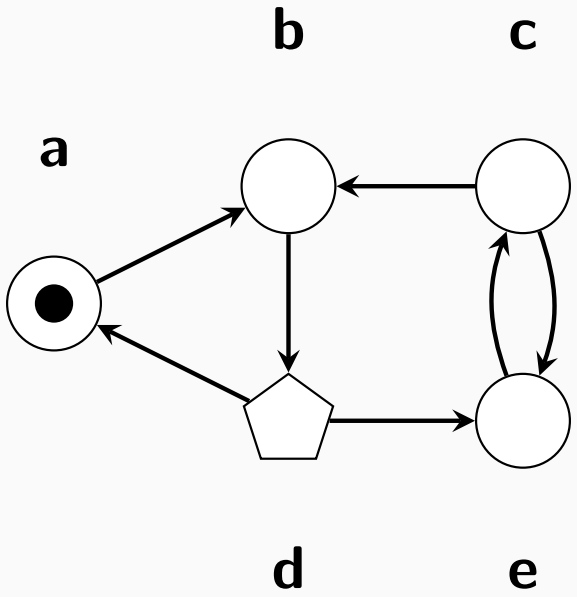
# WHY PARITY GAMES?

## Practical use of parity games

- Modal logic with nested least and greatest fixpoint operators
- Linear equivalence with modal $\mu$-calculus model-checking
  (CTL, LTL, CTL*, mCRL2, PBES, ...)
- Backend for LTL model checking and LTL synthesis
- However: practical parity games are all "easy" to solve!

# PARITY GAMES

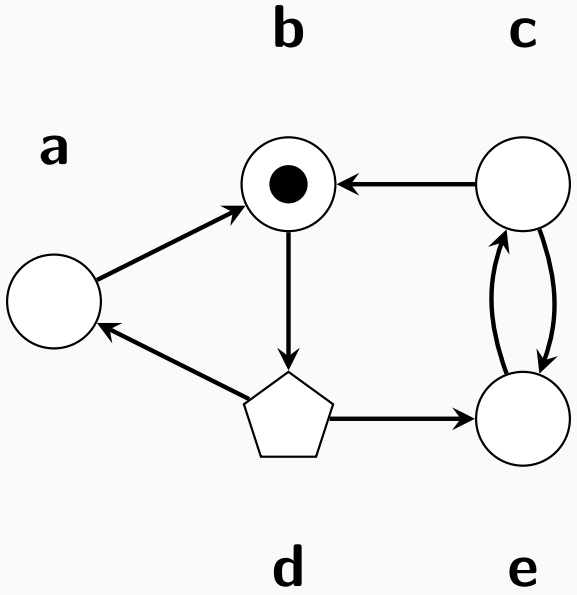
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a**

# PARITY GAMES

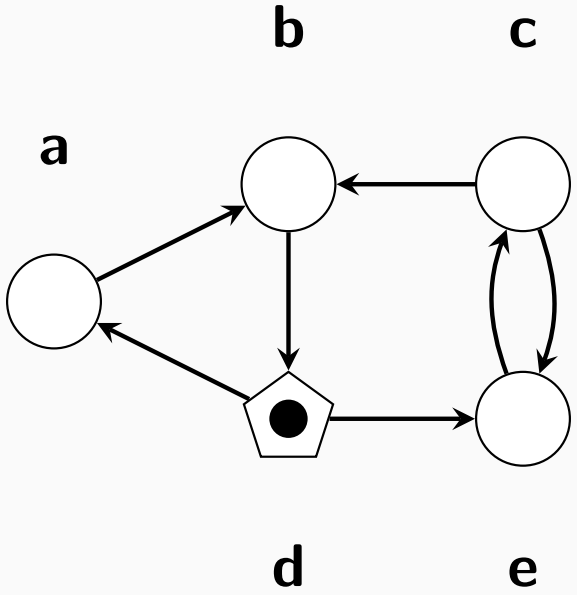
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b**

# PARITY GAMES

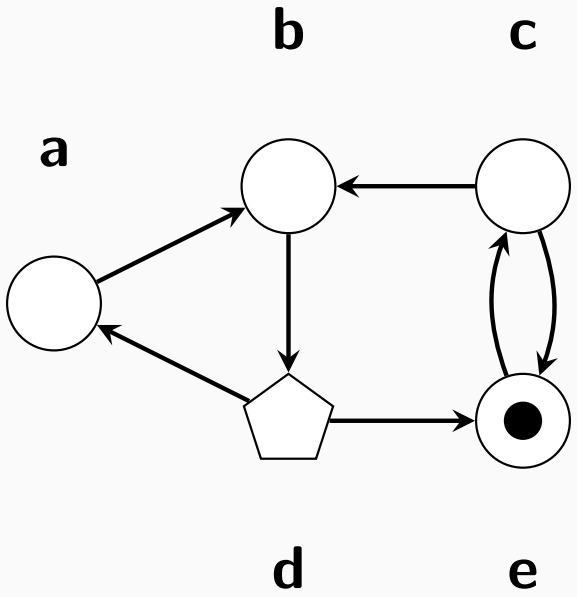
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d**

# PARITY GAMES

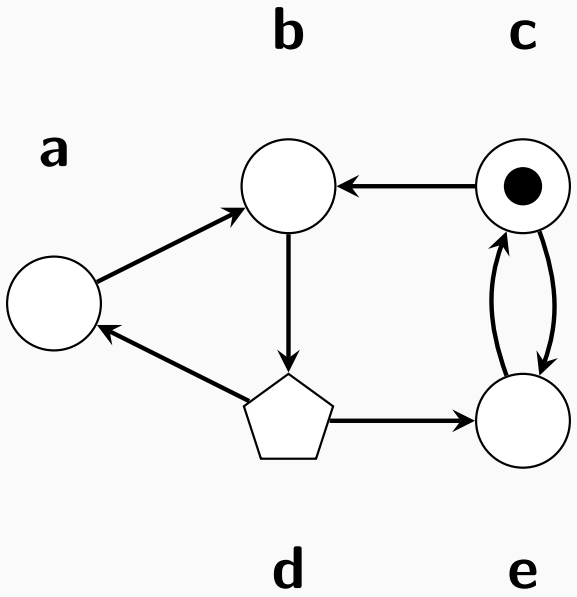
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e**

# PARITY GAMES

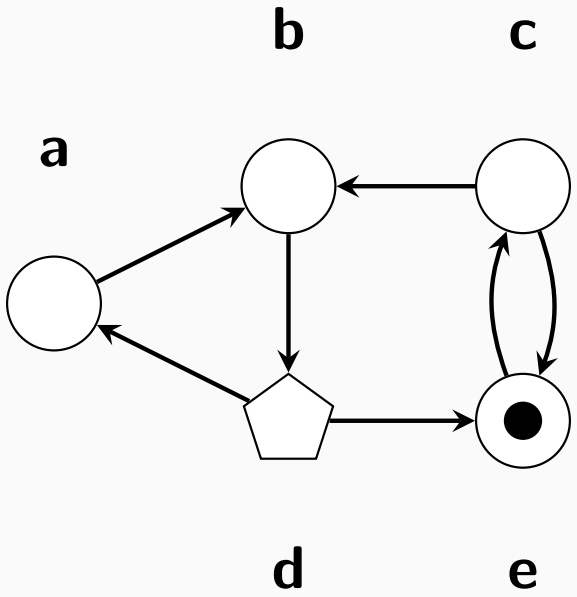
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c**

# PARITY GAMES

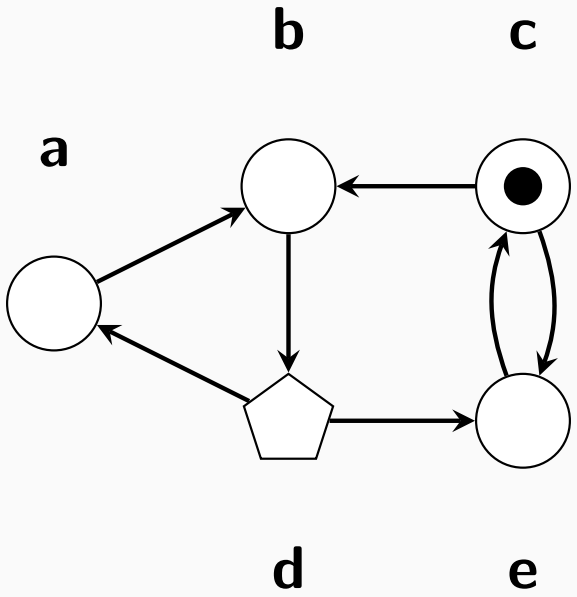
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e**

# PARITY GAMES

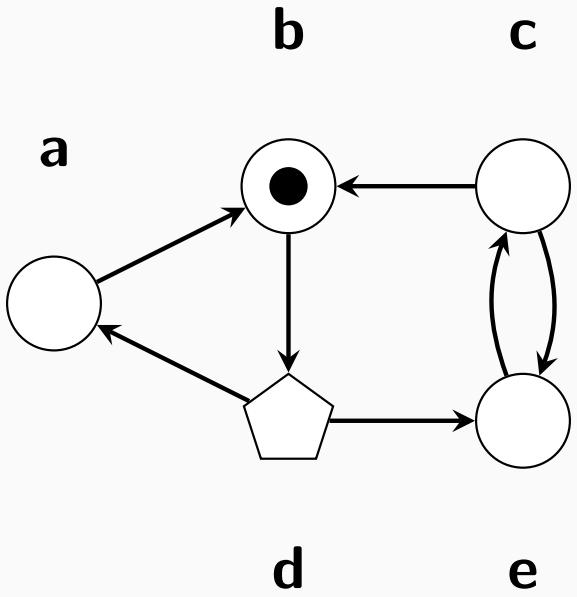
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c**

# PARITY GAMES

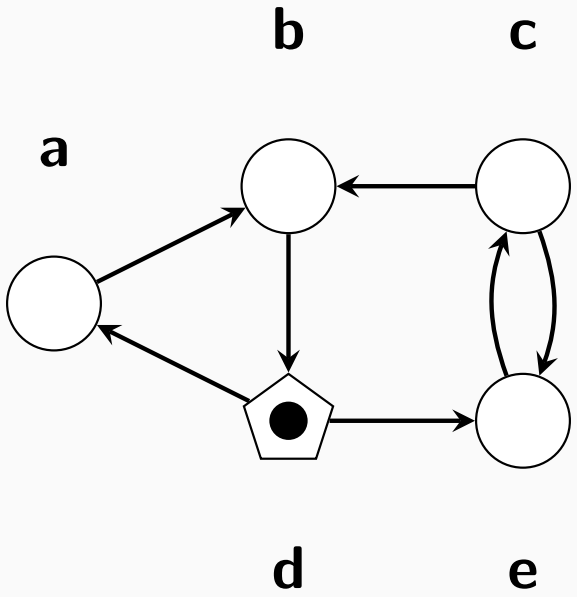
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b**

# PARITY GAMES

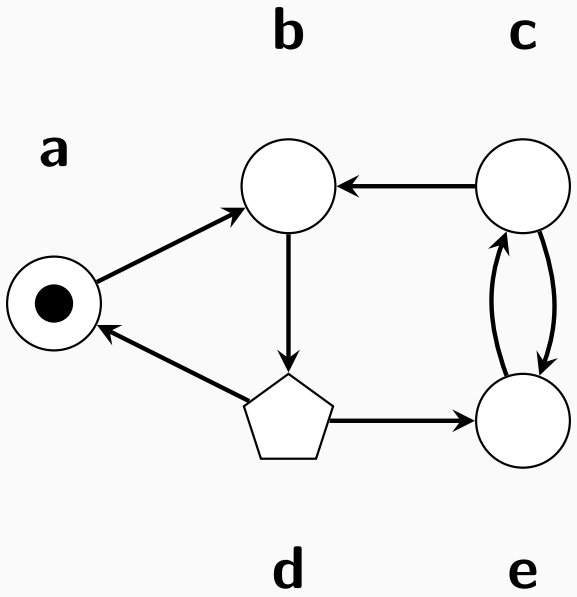
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b d**

# PARITY GAMES

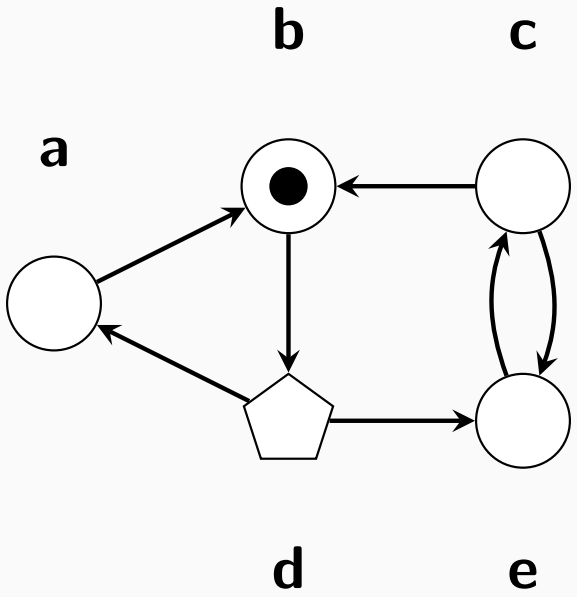
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b d a**

# PARITY GAMES

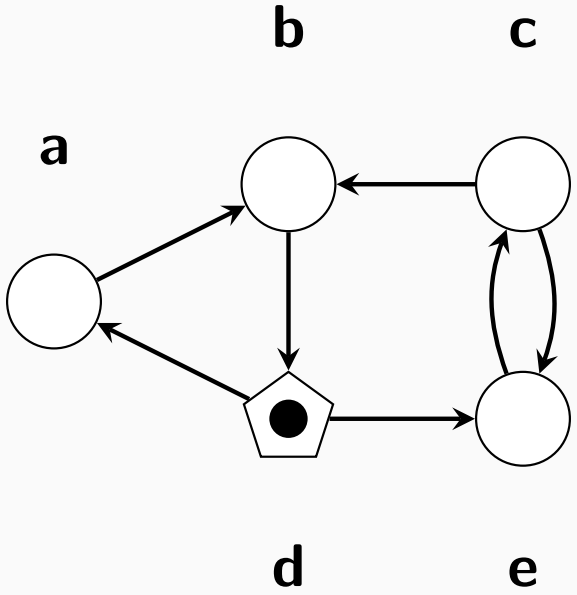
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b d a b**

# PARITY GAMES

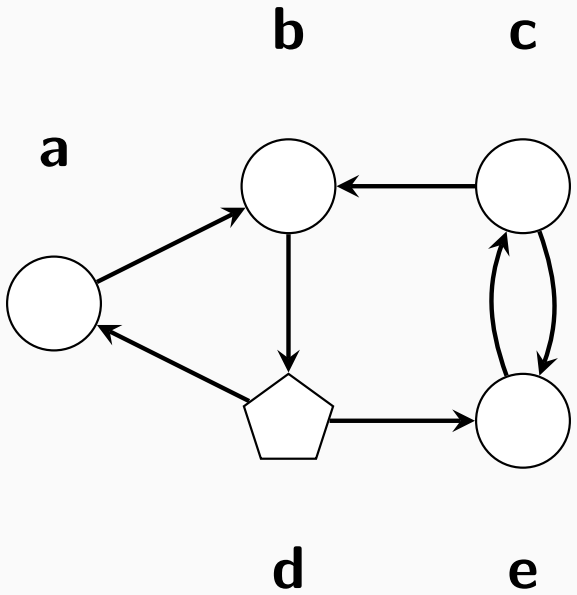
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




The play $\pi$: **a b d e c e c b** $(\mathbf{d\ a\ b})^{\omega}$

# PARITY GAMES

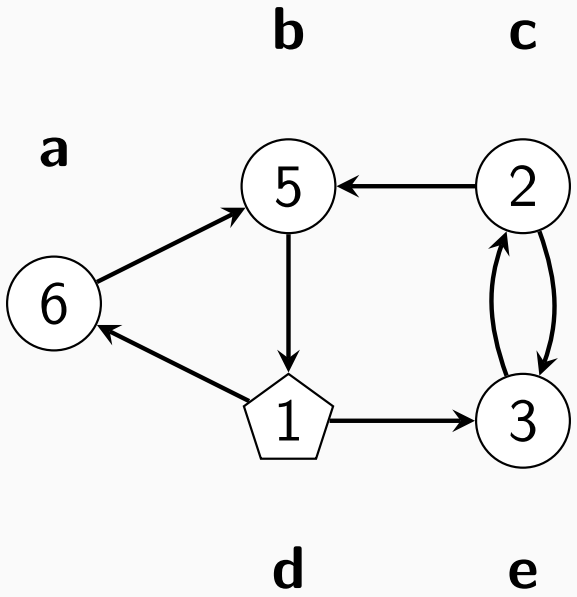
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




Who wins a play?

# PARITY GAMES

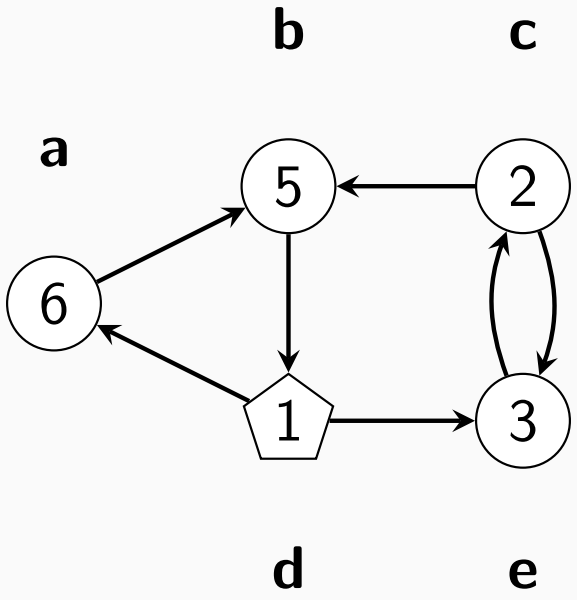
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




- Each vertex has a priority $\{0, 1, 2, \ldots, d\}$
- Player Even wins if the highest priority seen infinitely often is even

# PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor

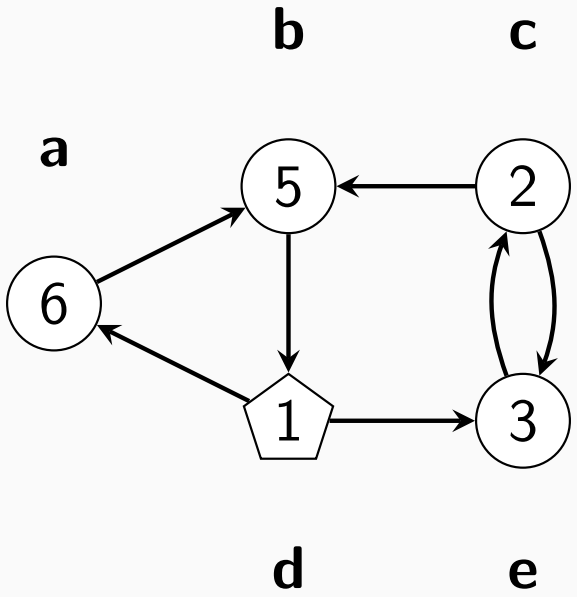



The play $\pi$: 6 5 1 3 2 3 2 5 $(1\ 6\ 5)^{\omega}$

Who wins this play?

# PARITY GAMES

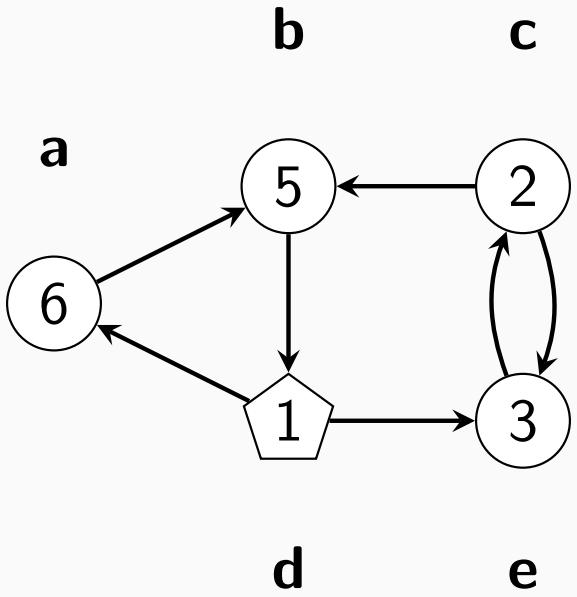
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




How do we determine who wins a vertex?

# PARITY GAMES

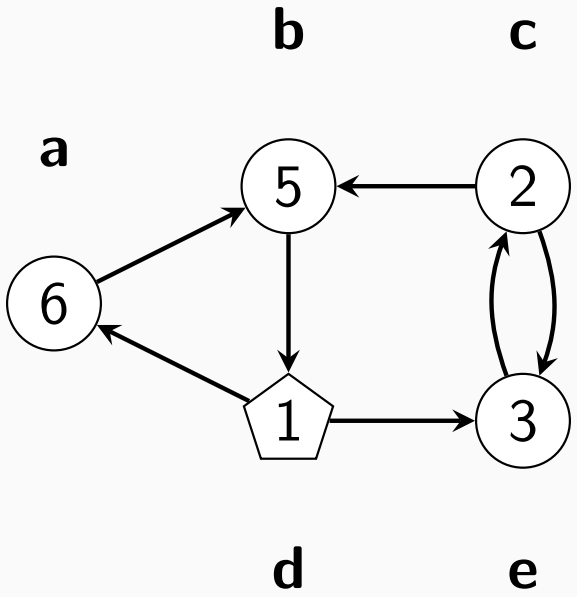
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




A player wins a vertex $v$ if they have a positional strategy $\sigma$ such that every play from $v$ consistent with $\sigma$ is won by that player.

# PARITY GAMES

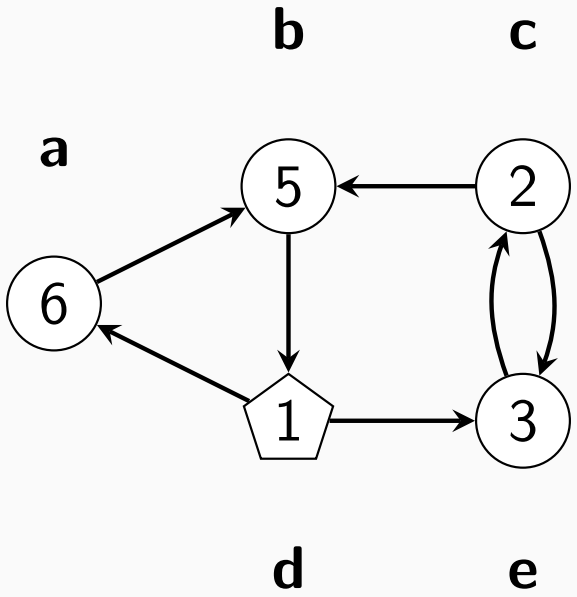
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




A player wins a vertex $v$ if they have a positional strategy $\sigma$ such that all cycles in the induced graph $\mathcal{G}[\sigma]$ are won by that player.

# PARITY GAMES

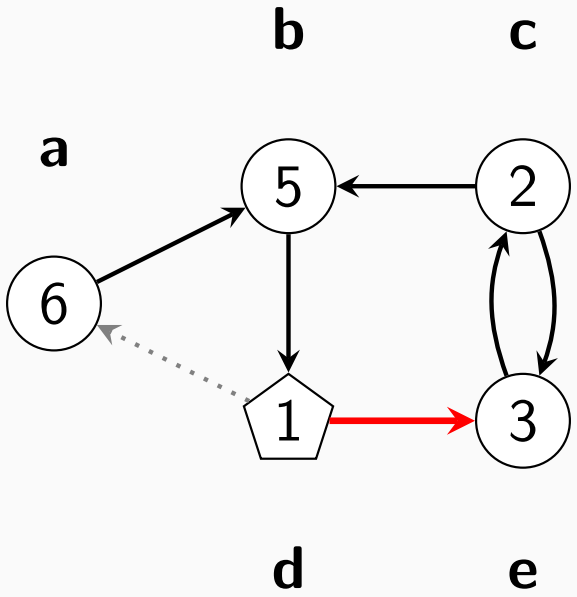
- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




Which vertices are won by which player?

# PARITY GAMES

- A parity game is played on a directed graph
- Two players: Even ● and Odd ⬟
- A play is an infinite path along the edges
- The owner of each vertex chooses the successor




Player Odd wins with strategy $\{\,\mathbf{d} \to \mathbf{e}\,\}$
Only two (Odd) cycles remain

## Solving a parity game

- Compute the winning regions $W_{\bigcirc}$ and $W_{\pentagon}$
- Compute the winning positional strategies $\sigma_{\bigcirc}$ and $\sigma_{\pentagon}$

# STUDYING A PROBLEM

## How to approach a problem?

First step is always to understand what algorithms already exist!

- Try implementing them
- Maybe you can already find optimizations
- Think about unique strengths/weaknesses
- Visualise the algorithms (backup slides)

# PARITY GAMES

## (Incomplete list of) published algorithms

| | | |
|---|---|---|
| McNaughton/Zielonka | $\mathcal{O}(e \cdot n^d)$, $\mathcal{O}(2^n)$ | 1998 |
| Small Progress Measures | $\mathcal{O}(d \cdot e \cdot (n/d)^{d/2})$ | 1998 |
| Strategy Improvement | $\mathcal{O}(n \cdot e \cdot 2^e)$ | 2000 |
| Dominion Decomposition | $\mathcal{O}(n^{\sqrt{n}})$ | 2006 |
| Big Step | $\mathcal{O}(e \cdot n^{d/3})$ | 2007 |
| Fixed Point: APT, BFL, DFI | $\mathcal{O}(n^d)$ | 2014 – 2021 |
| Priority Promotion | $\Omega(2^{\sqrt{n}})$ | 2016 |
| Quasi-Polynomial (several) | $\mathcal{O}(n^{6+\log d})$ | 2016 – 2021 |
| Tangle Learning | $\Omega(2^{\sqrt{n}})$ | 2018 – 2021 |

# OINK

- Modern (parallel) implementation of parity game algorithms
  - Strategy Improvement (parallel)
  - Distraction fixed point iteration (parallel)
  - Recursive algorithm (parallel attractor)
  - Priority promotion
  - Tangle learning
  - Small progress measures (with optimizations)
  - Succinct progress measures
  - Ordered progress measures
- The usual preprocessing algorithms
  - Inflation and compression
  - Removing self-loops and trivial winning regions
  - SCC decomposition
- `https://www.github.com/trolando/oink`
- Simple to use/extend library in C++

# OINK

## Easy to use

```
#include "oink.hpp"

pg::Game parity_game;
parity_game.parse_pgsolver(std::cin);

pg::Oink solver(parity_game);
solver.setSolver("tl");
solver.run();

parity_game.write_sol(std::cout);
```

## Easy to extend

- Implement Solver interface
- Add one line to `solvers.cpp`
- Comes with `text_solvers` program to quickly find bugs

# STUDYING A PROBLEM

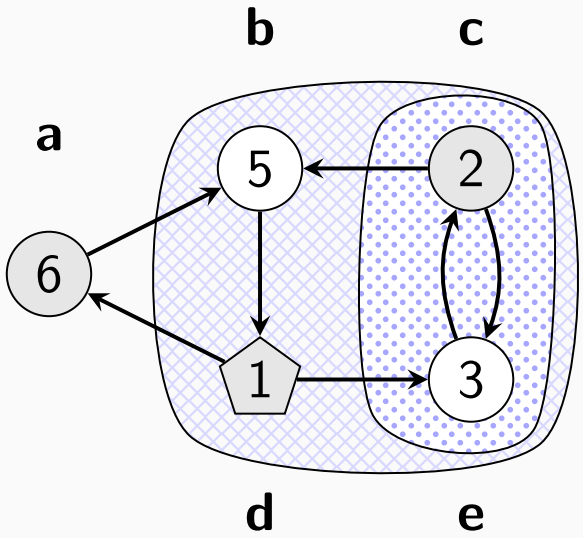
## How to approach a problem?

Next step is to understand how to delay an algorithm

- Tangles: subgames that are explored and analysed many times (exponentially often)
- Distractions: high value vertices that are misdirections
- Game of cat and mice: defeat the delays, then find new countergames, defeat those, repeat ad nauseam

## Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.




A game with a 5-tangle and a 3-tangle

# TANGLES

## Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

## Properties

- The opponent must escape (or lose inside the tangle)
- The opponent can freely choose any escape (strongly connected)
- A closed tangle (no escapes) is a winning region

# TANGLES

## Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

## Definition

A tangle is

- a pair $T = (U, \sigma)$ where
  - $U \subseteq V$ is a nonempty set of vertices
  - $\sigma \colon U_\alpha \to U$ is a strategy for player $\alpha := \mathrm{pr}(U) \bmod 2$

  such that
- player $\alpha$ wins all cycles in the induced subgame $\mathcal{G}[U, \sigma]$
- the induced subgame $\mathcal{G}[U, \sigma]$ is strongly connected

# TANGLES

## Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

## Tangle attractor

Because the opponent $\overline{\alpha}$ **must** escape the tangle, we can use tangles to attract all vertices of a tangle simultaneously.

Add to $Z$ (exhaustively):

- all vertices of the player $\alpha$ that *can* play to $Z$
- all vertices of opponent $\overline{\alpha}$ that *must* play to $Z$
- all vertices in an $\alpha$-tangle where all escapes lead to $Z$

# TANGLES

## Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

## Tangles are in every algorithm

- All algorithm implicitly reason about tangles:
  - if you conclude that the opponent cannot 'hide' in some subgame
  - then this **must** be because of tangles
- Known algorithms often explore the same tangle many times!
  (This can lead to repetitive behavior and exponential blowup!)
- **Hypothesis**: if a polynomial solution exists, then a polynomial solution exists that reasons about tangles
- **Research goal**: rewrite existing algorithms to use tangles explicitly

# TANGLES

## Tangle

A tangle is a (strongly connected) subgame for which one player has a strategy to win all plays that stay in the subgame.

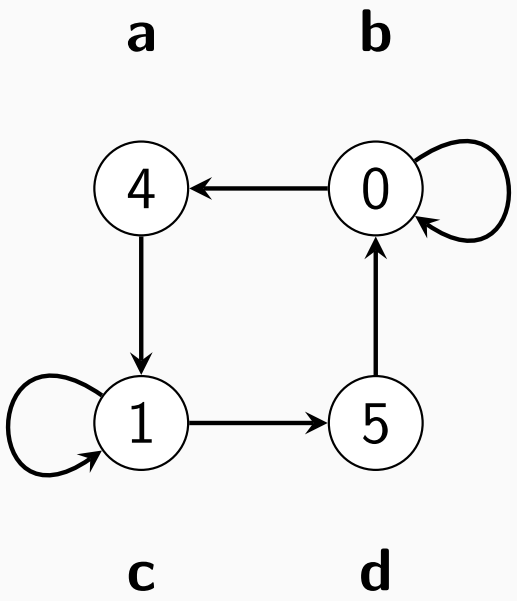
## Tangle learning

- Start with an empty set of tangles
- Repeatedly learn new tangles
  - Decompose the game using the tangle attractor
  - Learn new tangles using the decomposition
  - Repeat the above...
- When learning a closed tangle, we solve (part of) the game

## Tangles

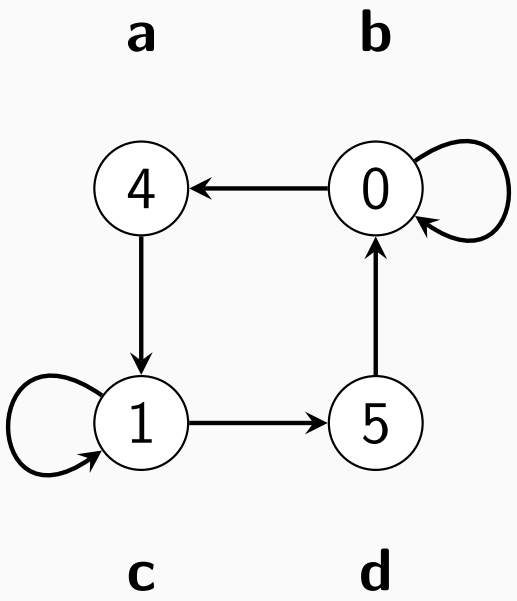
Many algorithms are hindered by tangles, which motivates my research into many different tangle learning algorithms: explicitly maintain a collection of discovered tangles, and use this knowledge to avoid repetitive behavior.

# DISTRACTIONS



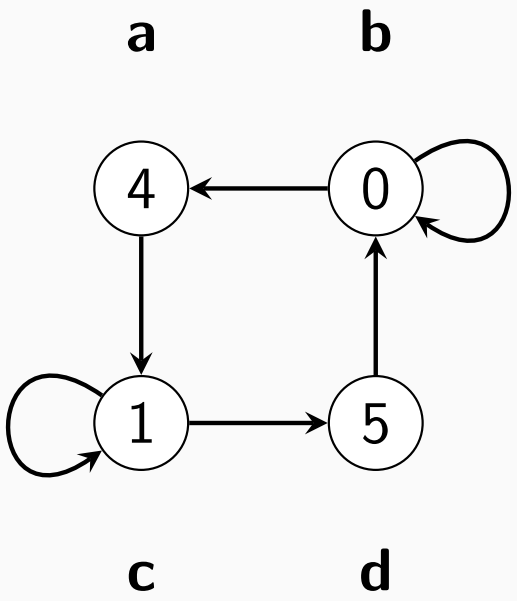

- **Find the winning strategy for player Even.**

# DISTRACTIONS



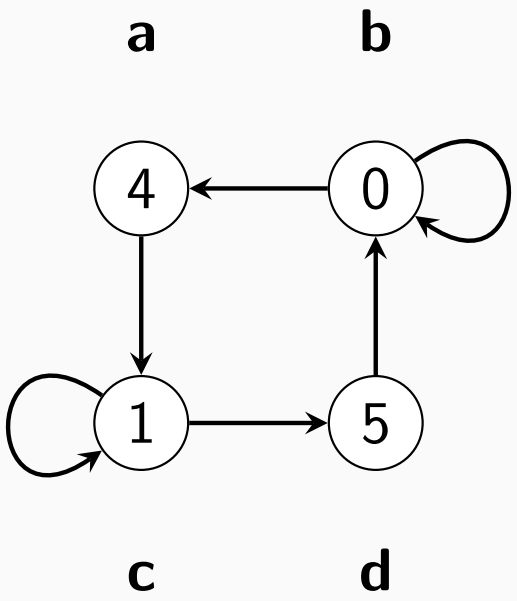

- **Find the winning strategy for player Even.**
- A random vertex inside the winning region?

# DISTRACTIONS



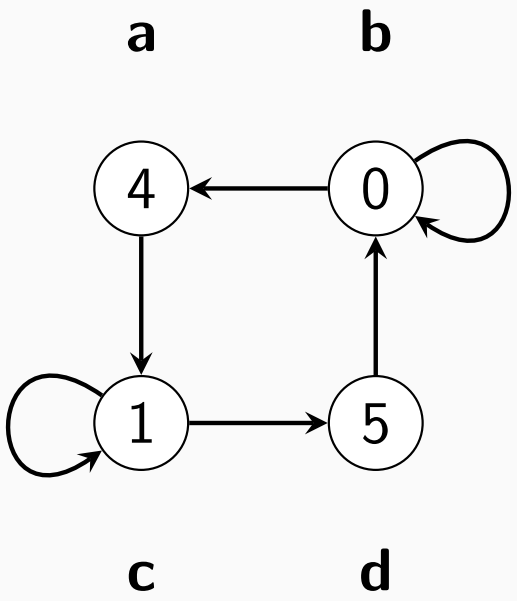

- **Find the winning strategy for player Even.**
- A random vertex inside the winning region?
  If you play from **c** to **c** you lose.

# DISTRACTIONS



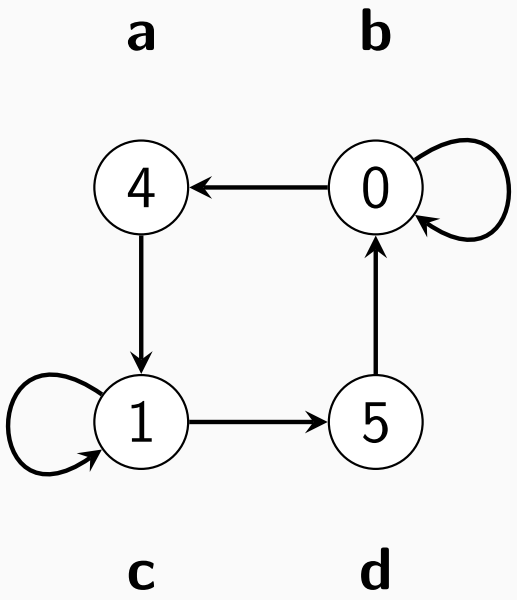

- **Find the winning strategy for player Even.**
- A random vertex inside the winning region?
  If you play from **c** to **c** you lose.
- Always play to the highest reachable even vertex?

# DISTRACTIONS




- **Find the winning strategy for player Even.**
- A random vertex inside the winning region?
  If you play from **c** to **c** you lose.
- Always play to the highest reachable even vertex?
  If you play from **b** to **a** you lose.

# DISTRACTIONS




- **Find the winning strategy for player Even.**
- A random vertex inside the winning region?
  If you play from **c** to **c** you lose.
- Always play to the highest reachable even vertex?
  If you play from **b** to **a** you lose.
- Vertex **a** is a distraction for player Even

# DISTRACTIONS

## Intuition

A distraction for $\alpha$ is a vertex with an $\alpha$-priority that the opponent $\overline{\alpha}$ can win if player $\alpha$ always tries to visit it.

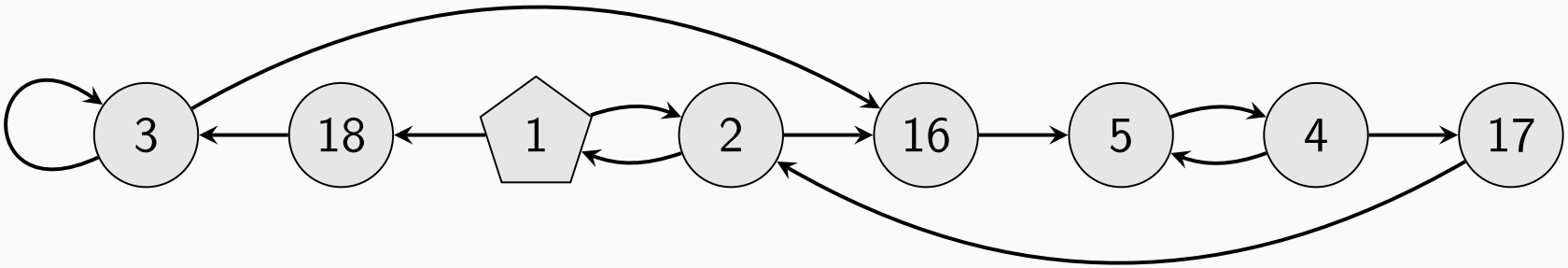
Player $\alpha$ must *sometimes/always* avoid the distraction.

## Definition

A distraction for player $\alpha$ is a vertex $v$ with an $\alpha$-priority $p$, such that if player $\alpha$ always plays to reach $v$ along paths of priorities $\leq p$, then player $\overline{\alpha}$ wins $v$ and all vertices that reach $v$.

(Along these paths to $v$, player $\overline{\alpha}$ either must or chooses to play to $v$.)

# DISTRACTIONS




Vertex **16** is a distraction.
Play **2** $\rightarrow$ **1** instead of **2** $\rightarrow$ **16** to win!
But also play **3** $\rightarrow$ **16**!

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible

**hat is a step?**

- FPI: set vertex as "distracting"
- REC: recursive calls
- PP: promotions
- TL: learned tangles
- *PM: increasing the value of a vertex

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?
  - Refine the "knowledge" of the solver until it solves (part of) the game

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?
  - Refine the "knowledge" of the solver until it solves (part of) the game

What is this knowledge?

- FPI: a vertex is now distracting, recompute lower game
- REC: a vertex is now distracting, recompute lower game
- PP: after the promotion, lower opponent vertices might be attracted
- TL: after learned tangles, lower opponent vertices might be attracted
- *PM: a vertex is slightly better/worse (possibly updated preference order)

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?
  - Refine the "knowledge" of the solver until it solves (part of) the game
- What "knowledge" do solvers have?

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?
  - Refine the "knowledge" of the solver until it solves (part of) the game
- What "knowledge" do solvers have?
  - How "preferable" is each vertex compared to other vertices
  - Attract to/play to "good" vertices and avoid "bad" vertices

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?
  - Refine the "knowledge" of the solver until it solves (part of) the game
- What "knowledge" do solvers have?
  - How "preferable" is each vertex compared to other vertices
  - Attract to/play to "good" vertices and avoid "bad" vertices
- What are *eventually* good vertices?
  - Vertices on good cycles (attracted to good priorities)
  - Vertices visited once on paths to good cycles
- What are *eventually* bad vertices?
  - Vertices on bad cycles (attracted to higher bad priorities)
  - Vertices visited once on paths to bad cycles

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many "steps" as possible
- What does a "step" accomplish?
  - Refine the "knowledge" of the solver until it solves (part of) the game
- What "knowledge" do solvers have?
  - How "preferable" is each vertex compared to other vertices
  - Attract to/play to "good" vertices and avoid "bad" vertices
- What are *eventually* good vertices?
  - Vertices on good cycles (attracted to good priorities)
  - Vertices visited once on paths to good cycles
- What are *eventually* bad vertices?
  - Vertices on bad cycles (attracted to higher bad priorities)
  - Vertices visited once on paths to bad cycles
- So delay the solver by delaying distinguishing good and bad vertices

# HARD PARITY GAMES

## What makes a parity game “hard to solve”?

- Require as many steps as possible

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many steps as possible
- That is: delay distinguishing "good" vertices from "bad" vertices

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many steps as possible
- That is: delay distinguishing "good" vertices from "bad" vertices
- That is: trick the solver to believe bad vertices are good and good vertices are bad
  - It **will** eventually discover the truth

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many steps as possible
- That is: delay distinguishing "good" vertices from "bad" vertices
- That is: trick the solver to believe bad vertices are good and good vertices are bad
  - It **will** eventually discover the truth
- Remember: when is a vertex bad?
  - It leads to a higher opponent priority
  - It leads to a region won by the opponent

  This must be nontrivial (otherwise it doesn't delay the solver)

# HARD PARITY GAMES

## What makes a parity game "hard to solve"?

- Require as many steps as possible
- That is: delay distinguishing "good" vertices from "bad" vertices
- That is: trick the solver to believe bad vertices are good and good vertices are bad
  - It **will** eventually discover the truth
- Remember: when is a vertex bad?
  - It leads to a higher opponent priority
  - It leads to a region won by the opponent

  This must be nontrivial (otherwise it doesn't delay the solver)
- Remember: when is a vertex good?
  - It leads to a higher good priority
  - It leads to a region won by us

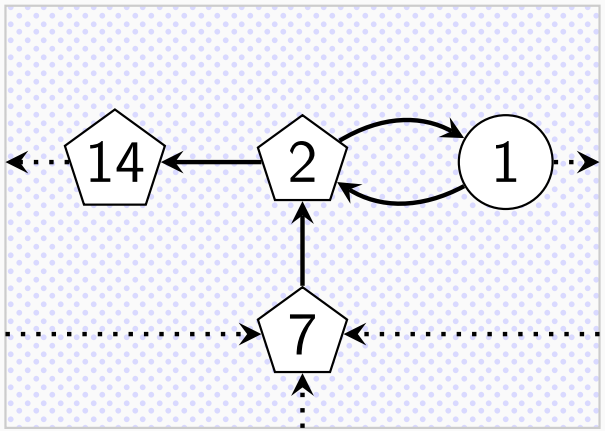
  This must be nontrivial (otherwise it doesn't delay the solver)

## A Parity Game Tale of Two Counters

- Context
  - Broad goal: develop polynomial parity games solution
  - Using concepts like tangles, distractions
  - Many types of algorithms that deal with distractions differently
  - If not polynomial, deepen understanding with *strong* counterexamples
- Contributions
  - Parameterized parity game family for:
    - Local fixed point algorithms (FPI, FPJ)
    - Recursive algorithm (Zielonka)
    - Priority promotion & variations
    - Tangle learning

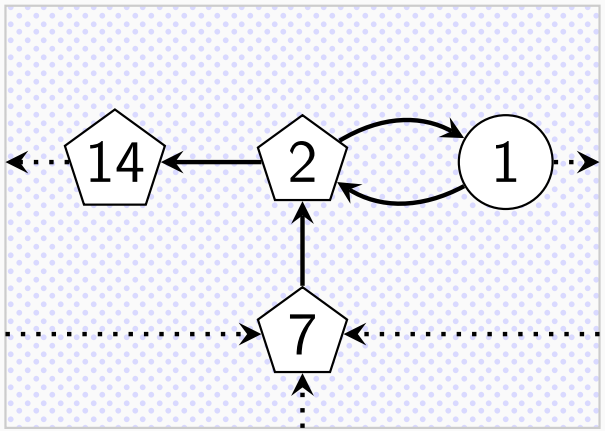
    All attractor-based algorithms that recognize a distraction by finding that the opponent can attract the distraction
  - Based on two interleaved binary counters of $N$ bits

# A BIT IN THE GAME



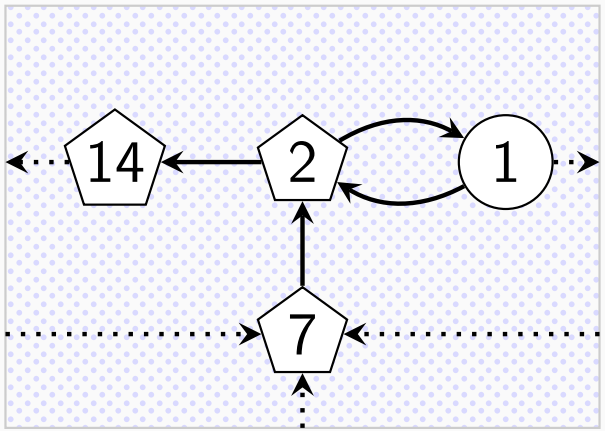

- This is a fragment of the counter: a bit
- 7 is a distraction for Odd

# A BIT IN THE GAME



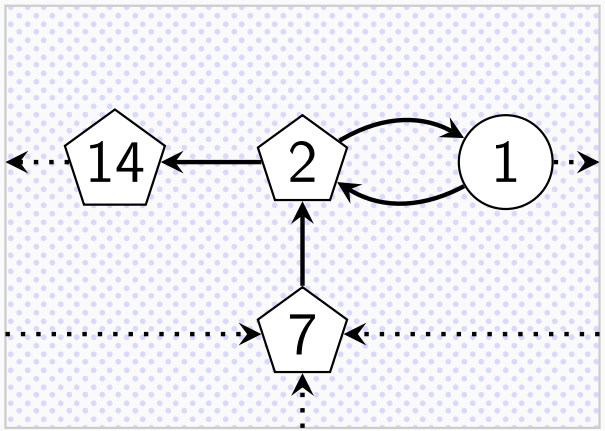

- This is a fragment of the counter: a bit
- 7 is a distraction for Odd
- Initially, the solver does not see the attractor to 14, so it believes 7 is bad for Even and good for Odd

# A BIT IN THE GAME



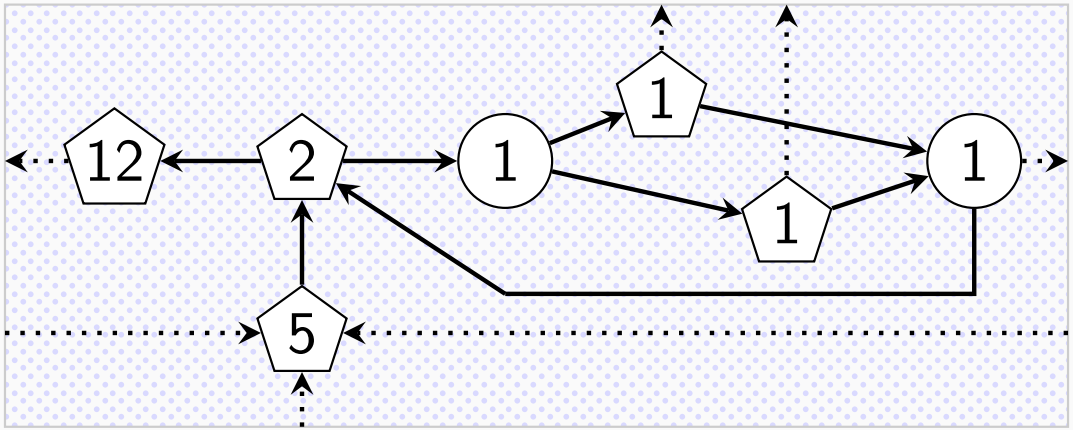

- This is a fragment of the counter: a bit
- 7 is a distraction for Odd
- Initially, the solver does not see the attractor to 14, so it believes 7 is bad for Even and good for Odd
- Paths from 1 lead to distractions for Even, so it does not play to 2

# A BIT IN THE GAME



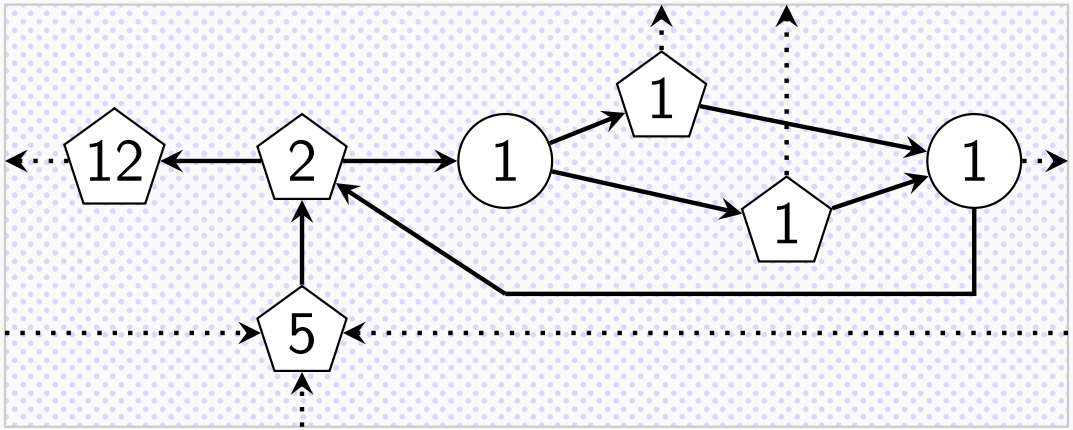

- This is a fragment of the counter: a bit
- 7 is a distraction for Odd
- Initially, the solver does not see the attractor to 14, so it believes 7 is bad for Even and good for Odd
- Paths from 1 lead to distractions for Even, so it does not play to 2
- After discovering that paths from 1 lead to distractions, now plays to 2 and discovers that 7 is a distraction for Odd

# A BIT IN THE GAME



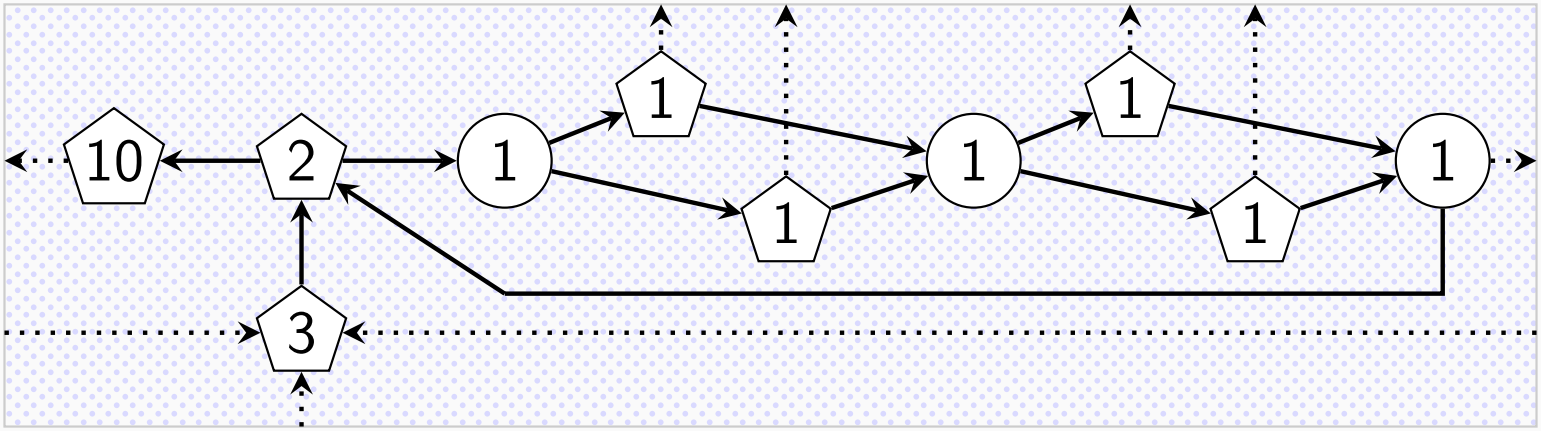

- This is another bit
- 5 is a distraction for Odd, only discovered when Even plays to 2, and paths from 5 now go via 12

# A BIT IN THE GAME



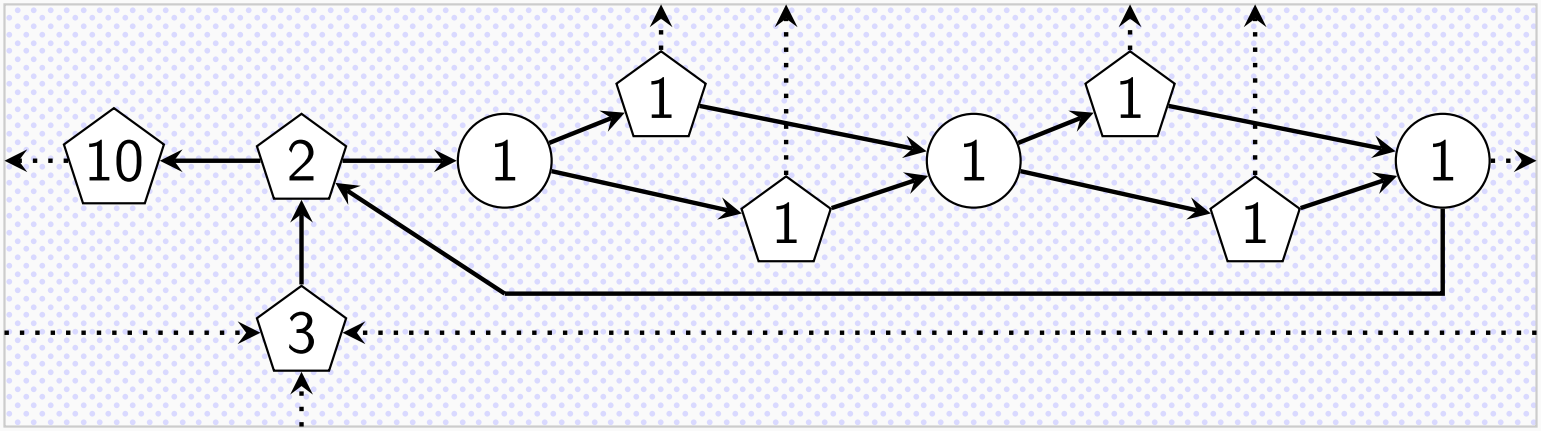

- This is another bit
- 5 is a distraction for Odd, only discovered when Even plays to 2, and paths from 5 now go via 12
- We now have 2 tangles
  - one "unstable" tangle that leads to a distraction
    initially Odd wants to avoid it, but afterwards Odd can escape there
  - one "stable" tangle that leads to an eventually-good vertex
    initially Odd can escape there, but afterwards Odd wants to avoid it

# A BIT IN THE GAME



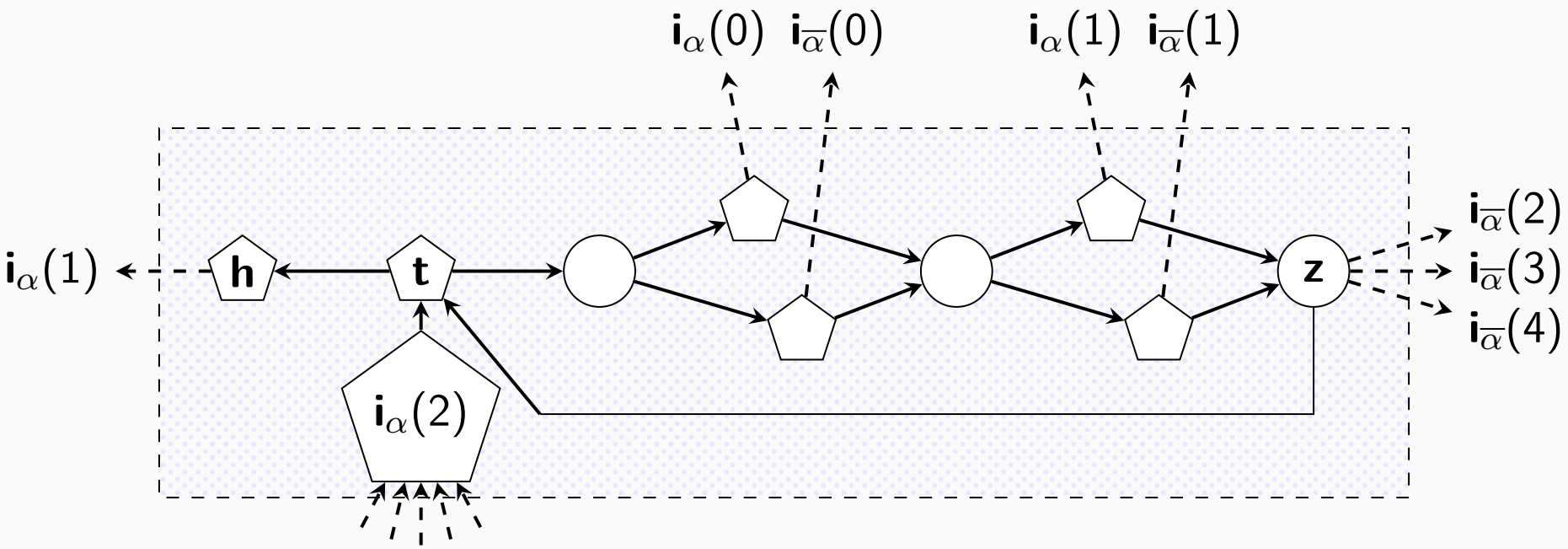

- This is another bit
- 3 is a distraction for Odd, only discovered when Even plays to 2, and paths from 3 now go via 10

# A BIT IN THE GAME



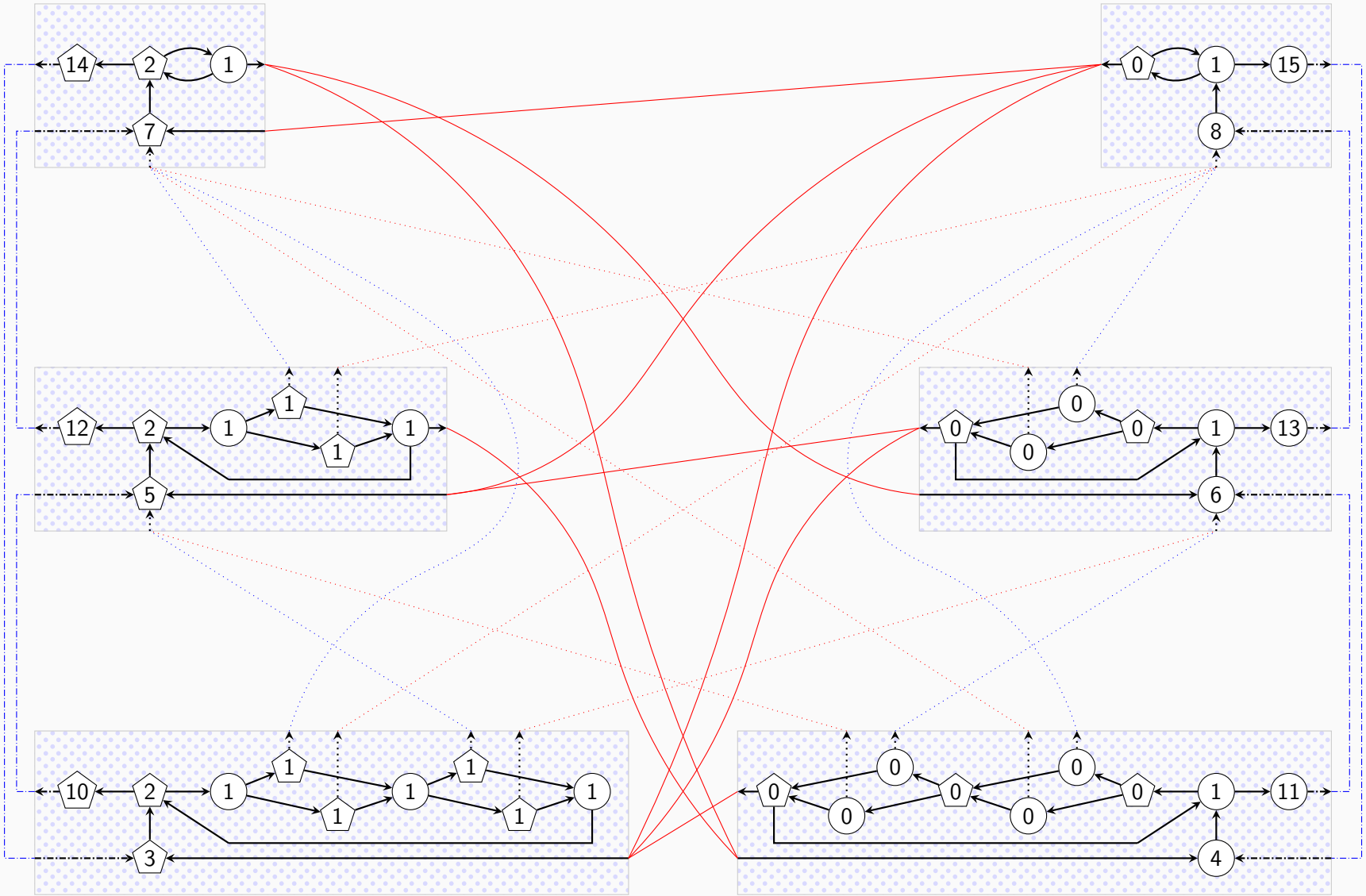

- This is another bit
- 3 is a distraction for Odd, only discovered when Even plays to 2, and paths from 3 now go via 10
- We now have 4 tangles
  - three "unstable" tangles with escapes to one/two distractions
  - one "stable" tangle with escapes to eventually-good vertices

# A BIT IN THE GAME




- We now say a bit is "set" when the solver believes $\mathbf{i}_\alpha$ is a distraction
- If higher bits (0, 1) of the opponent counter are set (they are now bad), the bit is "reset" because the opponent can now escape
- A bit is not set until **z** plays to **t** instead of the distractions
  That is: first learn lower bits of the opponent counter

# EXAMPLE 3-BIT GAME



# EXAMPLE 3-BIT GAME

| **Even** | **Odd** | **Event** |
|---|---|---|
| 000 | 000 | Initial state |
| 001 | 000 | Set Even 3 |
| 001 | 001 | Set Odd 3 |
| 011 | 000 | Set Even 2 (reset Odd 3) |
| 010 | 010 | Set Odd 2 (reset Even 3) |
| 011 | 010 | Set Even 3 |
| 011 | 011 | Set Odd 3 |
| 111 | 000 | Set Even 1 (reset Odd 2, 3) |
| 100 | 100 | Set Odd 1 (reset Even 2, 3) |
| 101 | 100 | Set Even 3 |
| 101 | 101 | Set Odd 3 |
| 111 | 100 | Set Even 2 (reset Odd 3) |
| 110 | 110 | Set Odd 2 (reset Even 3) |
| 111 | 110 | Set Even 3 |
| 111 | 111 | Set Odd 3 |

# SUMMARY

## Reflections

- Defeats all algorithms that rely only on opponent attraction to identify distractions.
- Until opponent attraction, the player is distracted and cannot find the distracted tangles.

- Other method: play to the best path, not just the best endpoint
  This is like progressive path values in progress measures
- "Recursive tangle learning" and "progressive tangle learning" (unpublished) use this method to identify distractions and can even use both methods to find distracted tangles.

# SUMMARY

## Reflections

- Defeats all algorithms that rely only on opponent attraction to identify distractions.
- Until opponent attraction, the player is distracted and cannot find the distracted tangles.

- Other method: play to the best path, not just the best endpoint
  This is like progressive path values in progress measures
- "Recursive tangle learning" and "progressive tangle learning" (unpublished) use this method to identify distractions and can even use both methods to find distracted tangles.

  *BUT VULNERABLE! (to a different version of the TC game)!*

# CONCLUSION

- Currently studying a “recursive tangle learning” variation that disables distractions
  - When we learn a tangle that was “behind” an open higher-value vertex, assume that it is a distraction

# CONCLUSION

- Currently studying a "recursive tangle learning" variation that disables distractions
  - When we learn a tangle that was "behind" an open higher-value vertex, assume that it is a distraction
- (But it removes more than just the distractions, so still not sure if polynomial)

# CONCLUSION

- Currently studying a "recursive tangle learning" variation that disables distractions
  - When we learn a tangle that was "behind" an open higher-value vertex, assume that it is a distraction
- (But it removes more than just the distractions, so still not sure if polynomial)
- Come up with requirements for an superpolynomial game / lemmata that prove that the algorithm is polynomial (essentially the same)
- Then try to construct a superpolynomial game and discover that it doesn't work, resulting in new lemmata, etc, etc.

# CONCLUSION

- Find an interesting open problem
- Study existing algorithms: implementing, visualising
- Study ways to delay algorithms
- For parity games: tangles and distractions
- Feel free to ask questions now.

# PARITY GAME ALGORITHMS

## Roughly two types

- Attractor-based
  Based on properties of *sets of vertices* computed with attractors.
  - Recursive algorithm (Zielonka, Parys, etc)
  - Priority promotion family
  - Tangle learning family
- Local value improvement
  Based on locally improving/updating the *value of individual vertices* by looking at their successors.
  - Monotonic values:
    - small progress measures
    - succinct progress measures (quasi-polynomial)
    - ordered progress measures (quasi-polynomial)
  - Semi/non-monotonic values:
    - strategy iteration and its numerous variations
    - nested fixed points iteration

## Attractor-based algorithms

- Recursive algorithm
- Priority promotion
- Tangle learning

## Attractor-based algorithms

- Recursive algorithm
  - Compute $\alpha$-attractor to highest priority vertices
  - Solve the subgame recursively
  - Now attracted to opponent? Then recompute remainder
- Priority promotion

- Tangle learning

## Attractor-based algorithms

- Recursive algorithm
  - Compute $\alpha$-attractor to highest priority vertices
  - Solve the subgame recursively
  - Now attracted to opponent? Then recompute remainder
- Priority promotion
  - Partition $\mathcal{G}$ into regions
  - Region locally closed? Promote (merge) with lowest escape and reset lower regions
  - Region globally closed? Return a dominion
- Tangle learning

## Attractor-based algorithms

- Recursive algorithm
  - Compute $\alpha$-attractor to highest priority vertices
  - Solve the subgame recursively
  - Now attracted to opponent? Then recompute remainder
- Priority promotion
  - Partition $\mathcal{G}$ into regions
  - Region locally closed? Promote (merge) with lowest escape and reset lower regions
  - Region globally closed? Return a dominion
- Tangle learning
  - Partition $\mathcal{G}$ into regions
  - Region locally closed? Learn new tangles
  - Closed tangles? Return as dominions
  - Repartition after learning new tangles

# ATTRACTOR COMPUTATION

## Playing from **A** to **B**

- How to show that a player $\alpha \in \{\bullet, \pentagon\}$ wins?

  or...

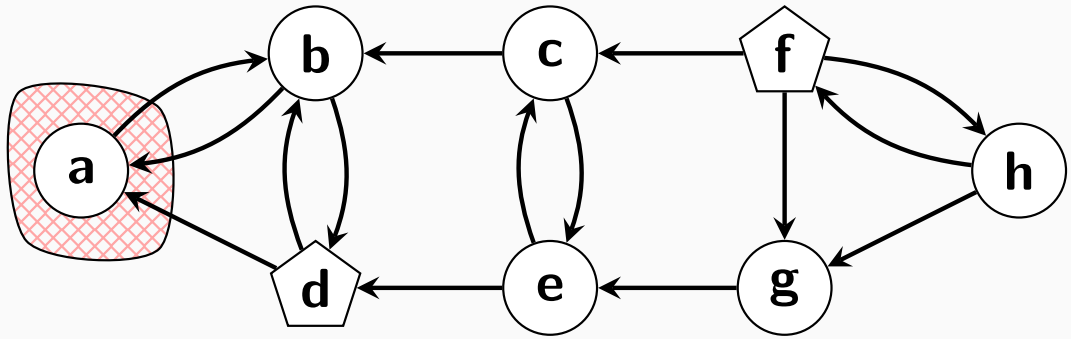
- How to show that the opponent $\overline{\alpha}$ must play to certain "bad" vertices?

## Attractor computation

- "Backward reachability but in a game"
- Given target set $Z$ and a player $\alpha$,
  compute all vertices from which player $\alpha$ can ensure arrival in $Z$
- Add to $Z$ (exhaustively):
  - all vertices of the player $\alpha$ that *can* play to $Z$
  - all vertices of opponent $\overline{\alpha}$ that *must* play to $Z$

## Example of attractor computation

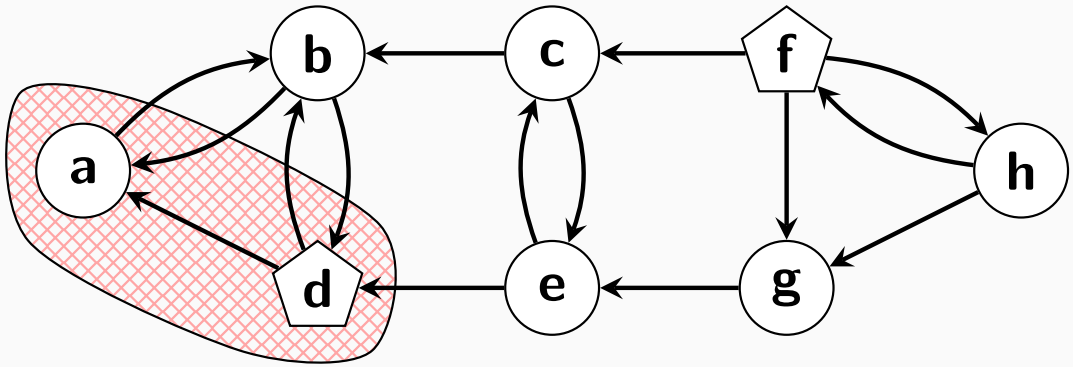
Computing the ⬟-attractor to **a**




Attractor set: $\{\mathbf{a}\}$
Can attract: **d** but not **b**

## Example of attractor computation

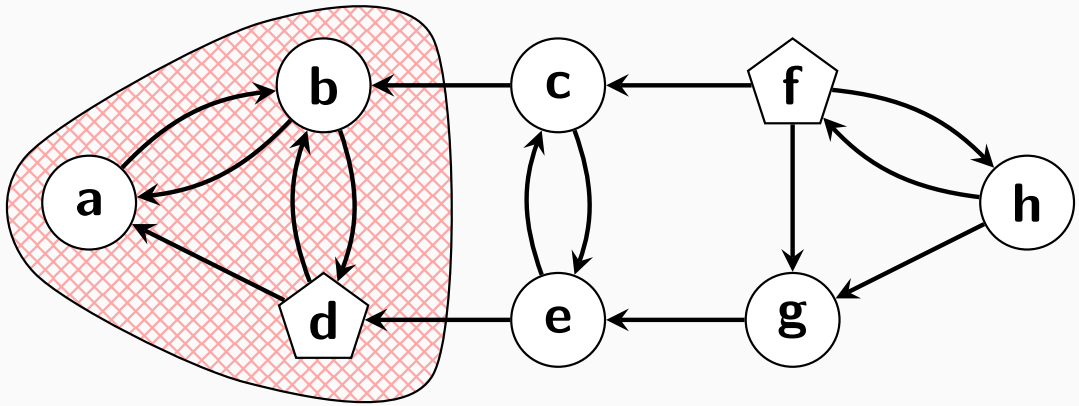
Computing the ⬟-attractor to **a**




Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$
Can attract: **b** but not **e**

## Example of attractor computation

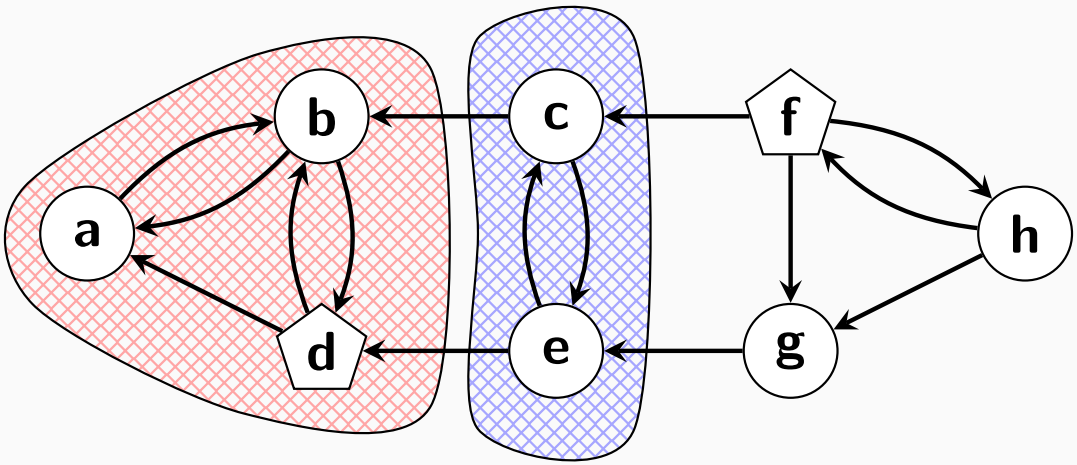
Computing the ⬟-attractor to **a**




Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$
Can attract: neither **c** nor **e**

## Example of attractor computation

Computing the ⬟-attractor to **a**

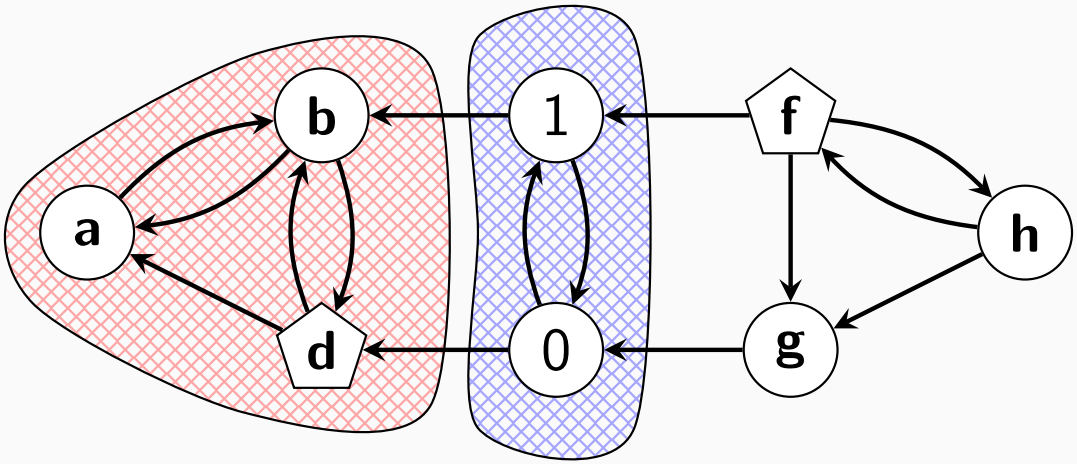



Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$

What if player ● cannot stay in $\{\mathbf{c}, \mathbf{e}\}$??

## Example of attractor computation

Computing the ⬠-attractor to **a**




Attractor set: $\{\mathbf{a}\}$, $\{\mathbf{a}, \mathbf{d}\}$, $\{\mathbf{a}, \mathbf{b}, \mathbf{d}\}$

*Every play inside the blue area is won by ⬠!*

# ATTRACTOR COMPUTATION

In game $\mathcal{G}$, attract to $Z$ as player $\alpha$ from the vertices $R$.
Update the strategy in $\sigma$ for:

- new vertices in $Z$ of player $\alpha$
- vertices in $Z$ of player $\alpha$ that can now play back to $Z$

```
1  def attract(𝒢, Z, R, α, σ):
2      Q ← Z                                  // initialize with all target vertices
3      while Q ≠ ∅ :
4          v ← take(Q)                                  // take a v ∈ Q
5          foreach u : v ∈ E(u) do                  // check incoming edges
6              if u ∈ R ∧ u ∉ Z ∧ (u ∈ V_α ∨ E(u) ⊆ Z) :        // attract?
7                  Z ← Z ∪ {u}
8                  Q ← Q ∪ {u}
9              if u ∈ Z ∩ V_α ∧ u ∉ σ : σ[u] ← v          // set strategy?
10     return Z, σ
```

# ZIELONKA'S RECURSIVE ALGORITHM

## Outline

- Recursive algorithm by McNaughton 1993, Zielonka 1998
- Decomposition of the game into regions
- Initially assume highest vertex to be "good" for the priority player
- Recompute recursively if this was not the case

- Recent variations run in quasi-polynomial time
  Parys 2019, Lehtinen 2019, Jurdzinski 2020
- A new variation uses the justification graph
  Lapauw 2020

# ZIELONKA'S RECURSIVE ALGORITHM

```
1  def zielonka(𝒢):
2      if 𝒢 = ∅ :
3          return ∅, ∅                                   // empty game
4      α ← pr(𝒢) mod 2                                   // winner of highest priority
5      Z ← pr^{-1}(pr(𝒢))                                // vertices of highest priority
6      A ← attract(𝒢, Z, V, α)                           // attracted to highest priority
7      W●, W⬟ ← zielonka(𝒢 \ A)                          // recursive solution
8      B ← attract(𝒢, W_ᾱ, V, ᾱ)                         // check if opponent attracts
9      if B = W_ᾱ :
10         W_α ← W_α ∪ A                                 // A is won by α
11     else:
12         W●, W⬟ ← zielonka(𝒢 \ B)                      // recompute remainder
13         W_ᾱ ← W_ᾱ ∪ B                                 // B is won by ᾱ
14     return W●, W⬟
```

# ZIELONKA'S RECURSIVE ALGORITHM

## Computing strategy

- Strategy is computed by attractor
  - Every attracted $\alpha$-vertex $u$ to some $v$ in the set: strategy is $u \to v$
- Special case: $\alpha$-vertices of the original target set
  - Pick any successor in winning region as strategy
- Implementation: use only a single strategy array, reset the strategy of highest priority vertices before attracting

# ZIELONKA'S RECURSIVE ALGORITHM

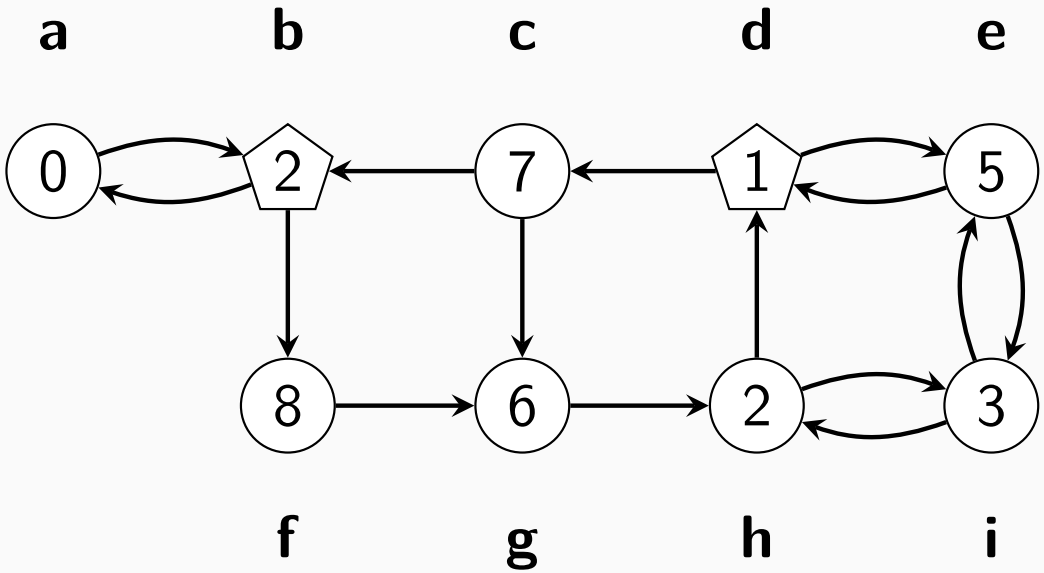
```
1  def zielonka(𝒢):
2      if 𝒢 = ∅ :
3          return ∅, ∅                                          // empty game
4      α ← pr(𝒢) mod 2                                  // winner of highest priority
5      Z ← pr^{-1}(pr(𝒢))                              // vertices of highest priority
6      A, σ_A ← attract(𝒢, Z, V, α, ∅)                 // attracted to highest priority
7      W_●, W_⬟, σ_●, σ_⬟ ← zielonka(𝒢 \ A)                    // recursive solution
8      B, σ_ᾱ ← attract(𝒢, W_ᾱ, V, ᾱ, σ_ᾱ)             // check if opponent attracts
9      if B = W_ᾱ :
10         W_α ← W_α ∪ A                                          // A is won by α
11         σ_α ← σ_α ∪ σ_A ∪ ((z ∈ Z) ↦ pick(E(z) ∩ W_α))
12     else:
13         W_●, W_⬟, σ_●, σ_⬟ ← zielonka(𝒢 \ B)            // recompute remainder
14         W_ᾱ ← W_ᾱ ∪ B                                          // B is won by ᾱ
15         σ_ᾱ ← σ_ᾱ ∪ σ_B
16     return W_●, W_⬟, σ_●, σ_⬟
```

# ZIELONKA'S RECURSIVE ALGORITHM

## Another perspective

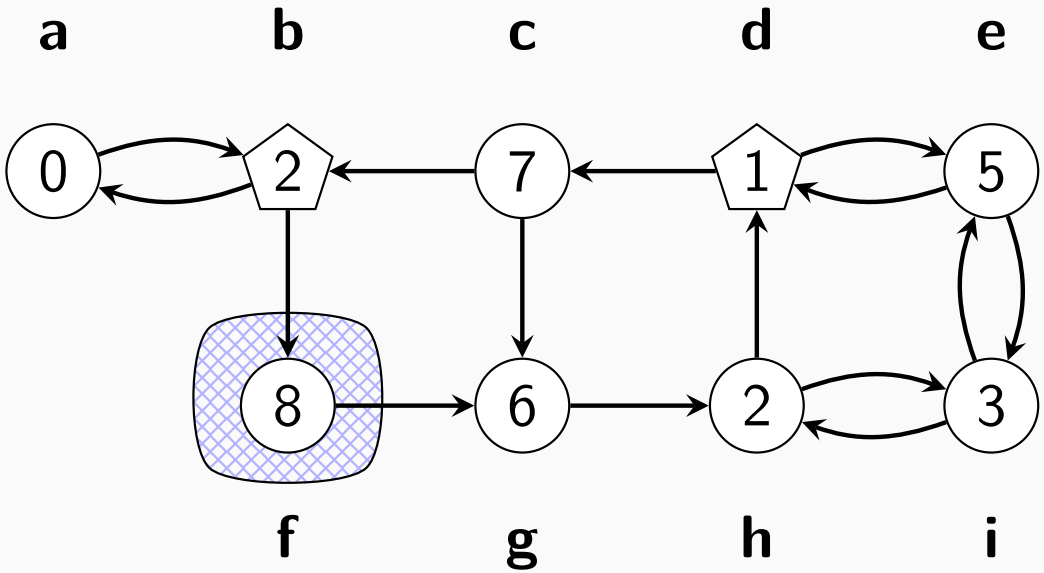
- Decompose top-down into regions
- Going back up, check if every "top vertex" is a distraction by attracting to the opponent's lower regions
- If so, reset and recompute the player's lower regions

# ZIELONKA'S ALGORITHM



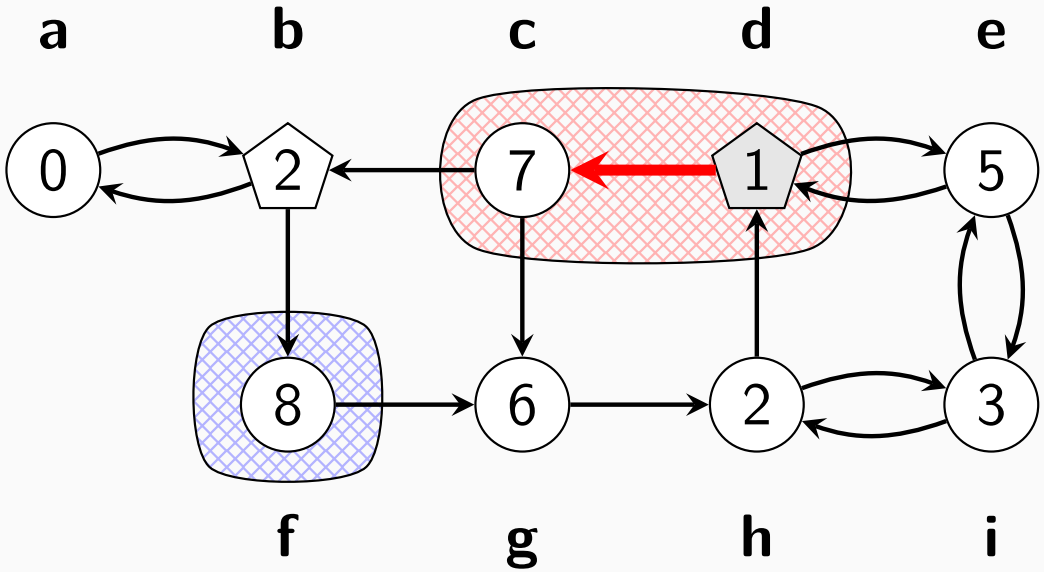

- We start by attracting to 8 for player Even.

# ZIELONKA'S ALGORITHM



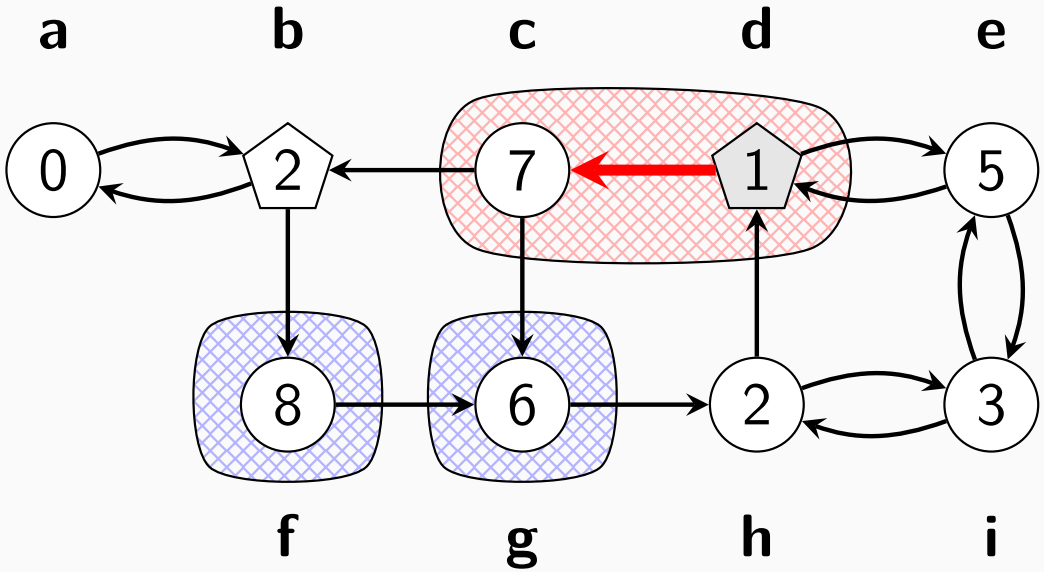

- After region 8 (player Even).
- Continue (recursively) with region 7.

# ZIELONKA'S ALGORITHM



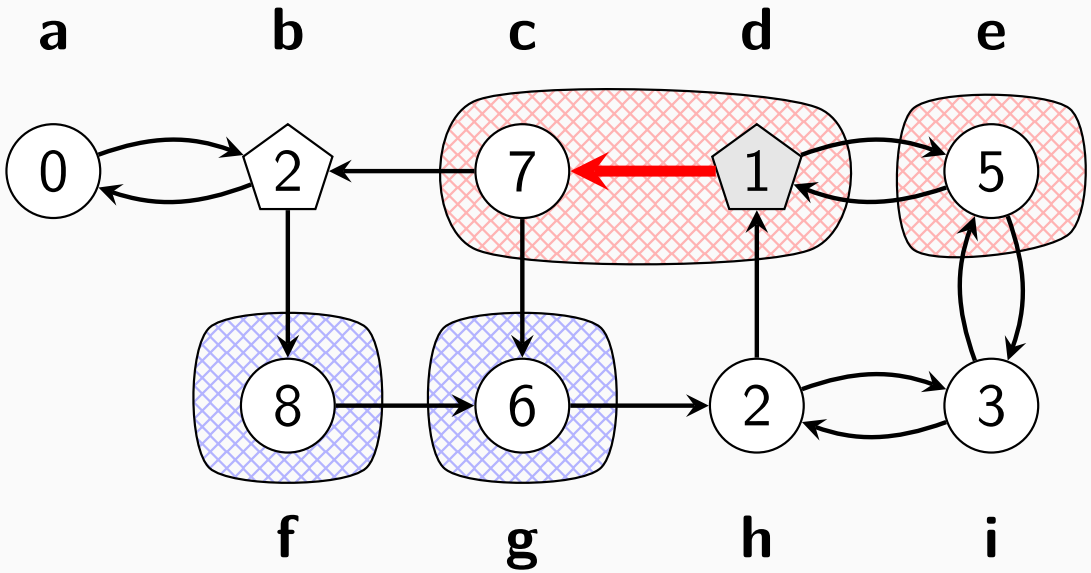

- After regions 8 (player Even) and 7 (player Odd).
- Continue (recursively) with region 6.

# ZIELONKA'S ALGORITHM



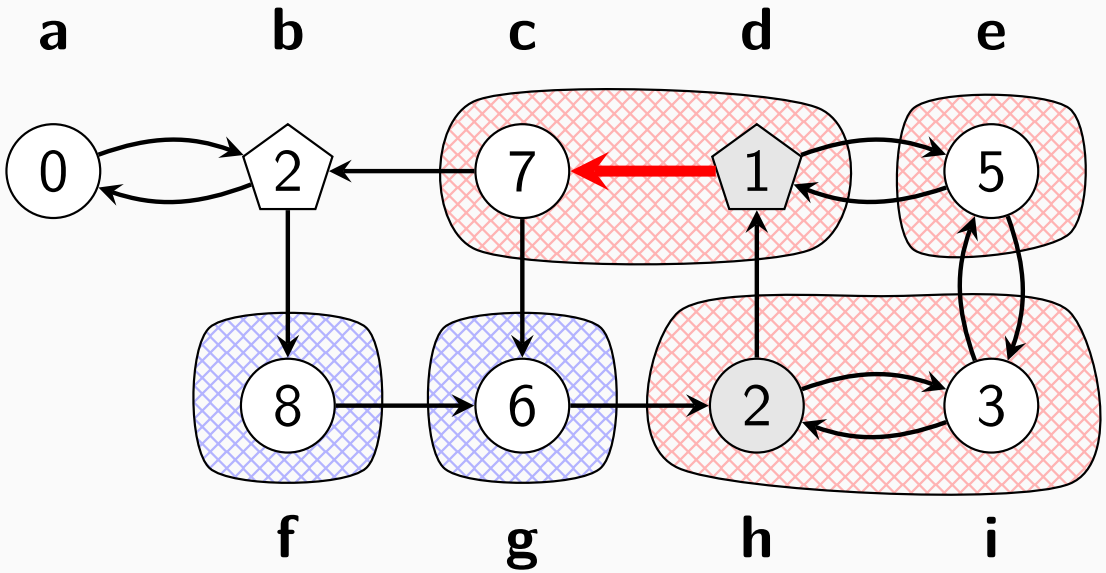

- After regions 8, 7 and 6.
- Continue (recursively) with region 5.

# ZIELONKA'S ALGORITHM



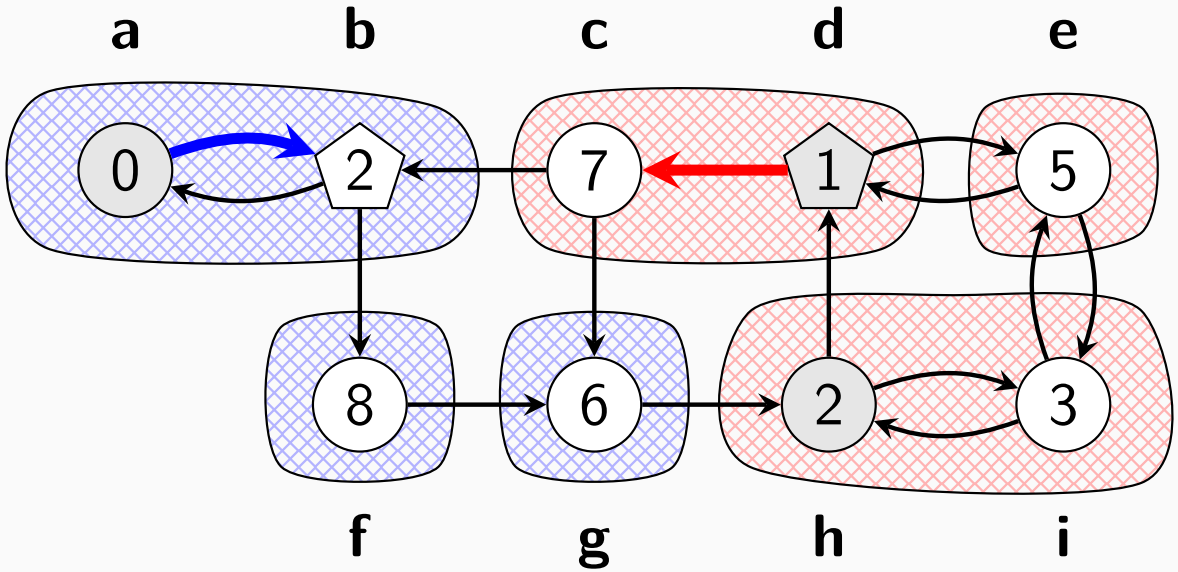

- After regions 8, 7, 6 and 5.
- Continue (recursively) with region 3.

# ZIELONKA'S ALGORITHM



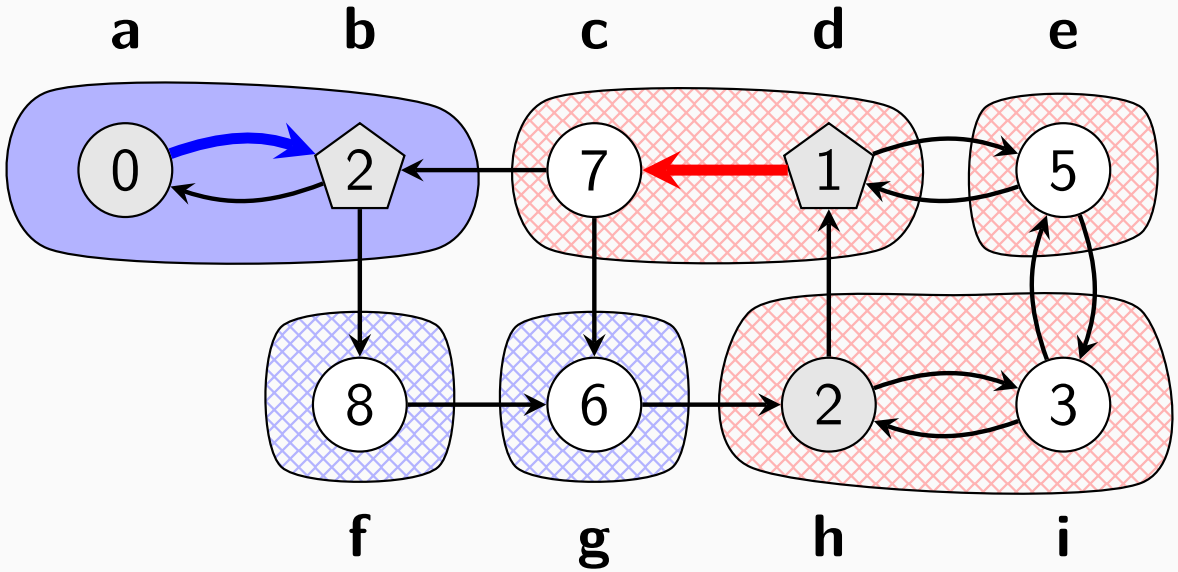

- After regions 8, 7, 6, 5 and 3.
- Now remains just region 2.

# ZIELONKA'S ALGORITHM



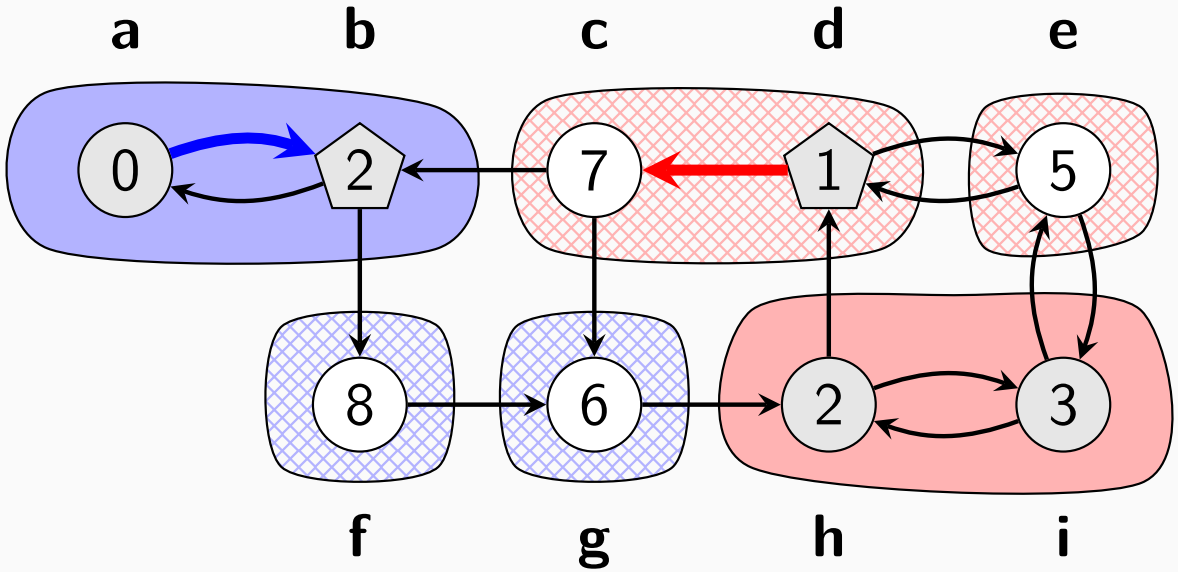

- Game is partitioned fully, now go up in the recursion.
- Up in region 2, does the lower opponent's winning region attract?
- Region 2: no (because the subgame is empty).

# ZIELONKA'S ALGORITHM



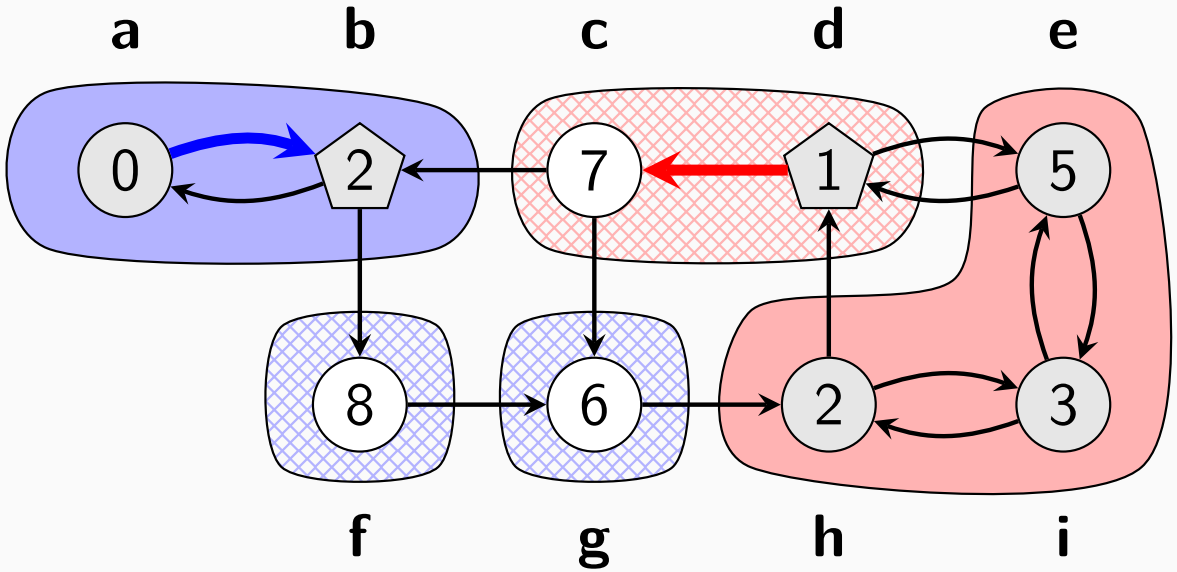

- Up in region 3: does the lower Even region attract from region 3?

# ZIELONKA'S ALGORITHM



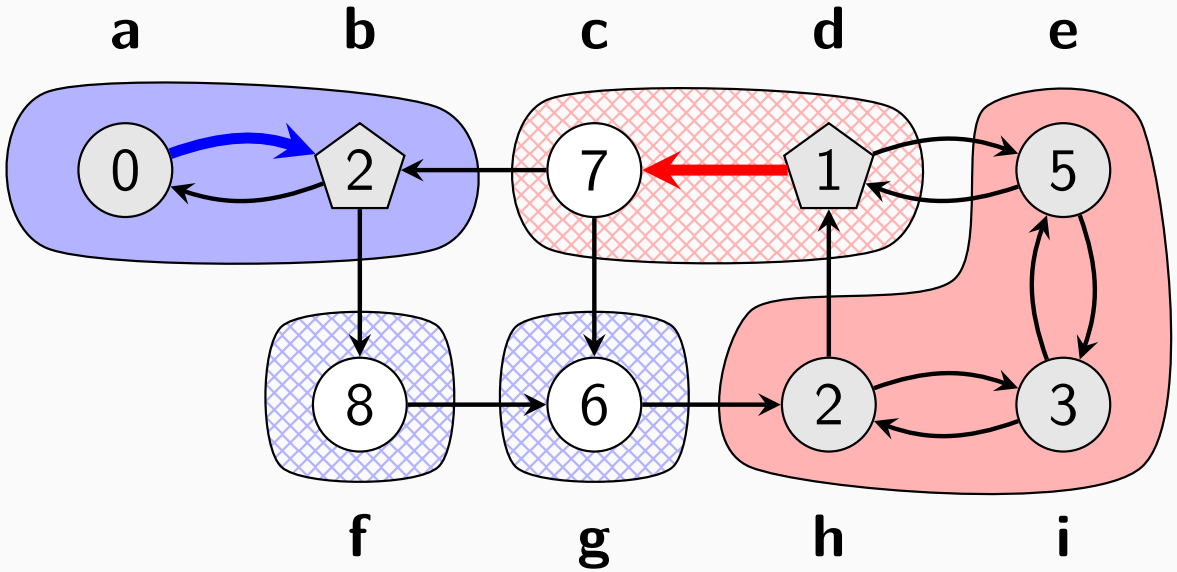

- Up in region 5: does the lower Even region attract from region 5?

# ZIELONKA'S ALGORITHM



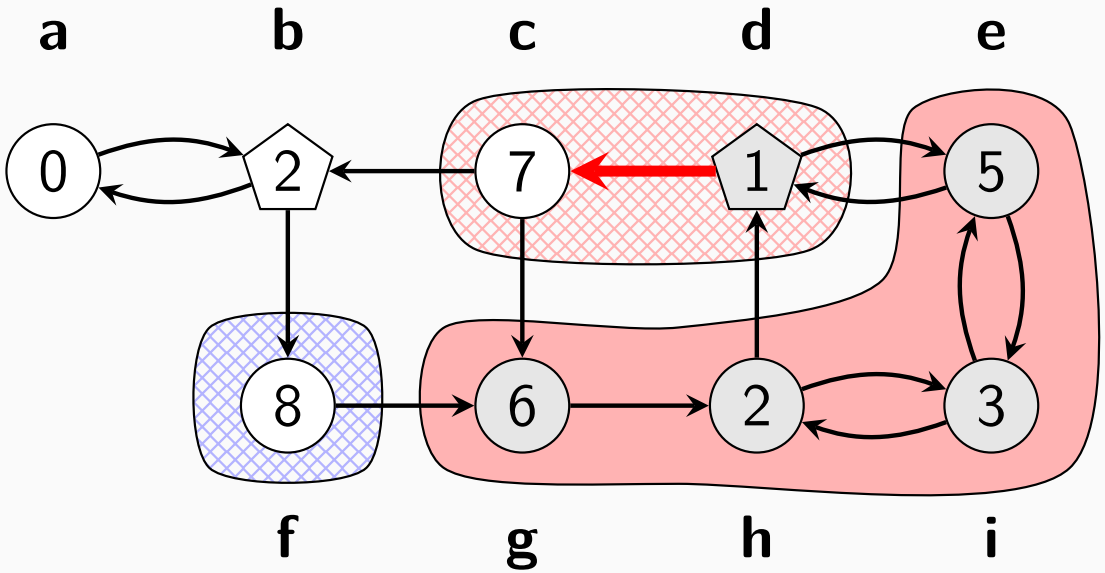

- Up in region 6: does the lower Odd region attract from 6?

# ZIELONKA'S ALGORITHM



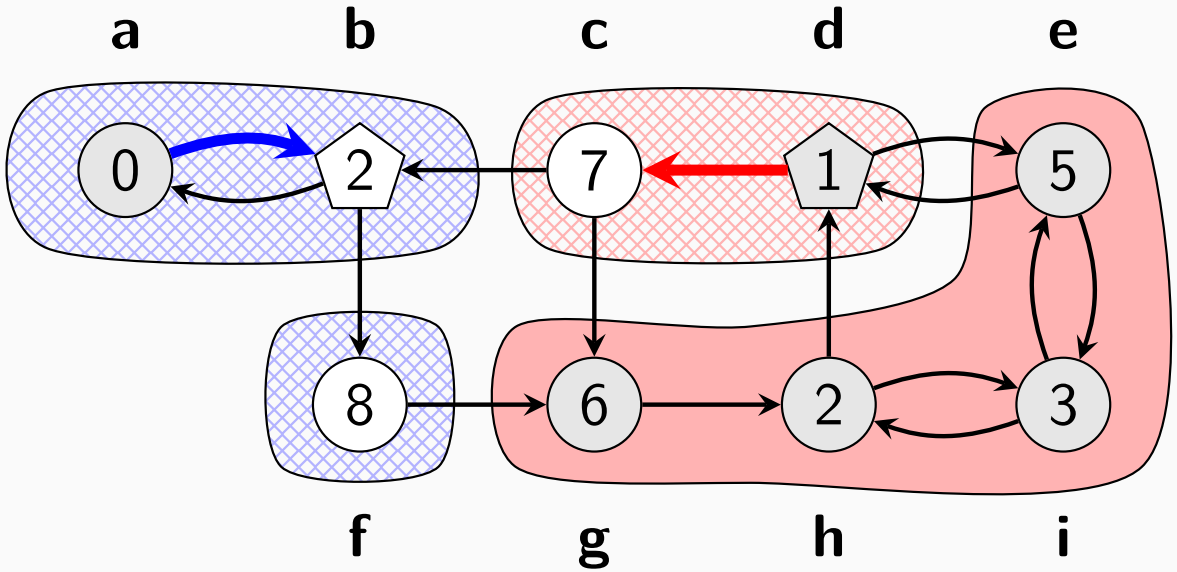

- Up in region 6: does the lower Odd region attract from 6?
- Yes: the lower Odd region attracts vertex **g**.
- So: attract to Odd, reset and recompute lower.

# ZIELONKA'S ALGORITHM



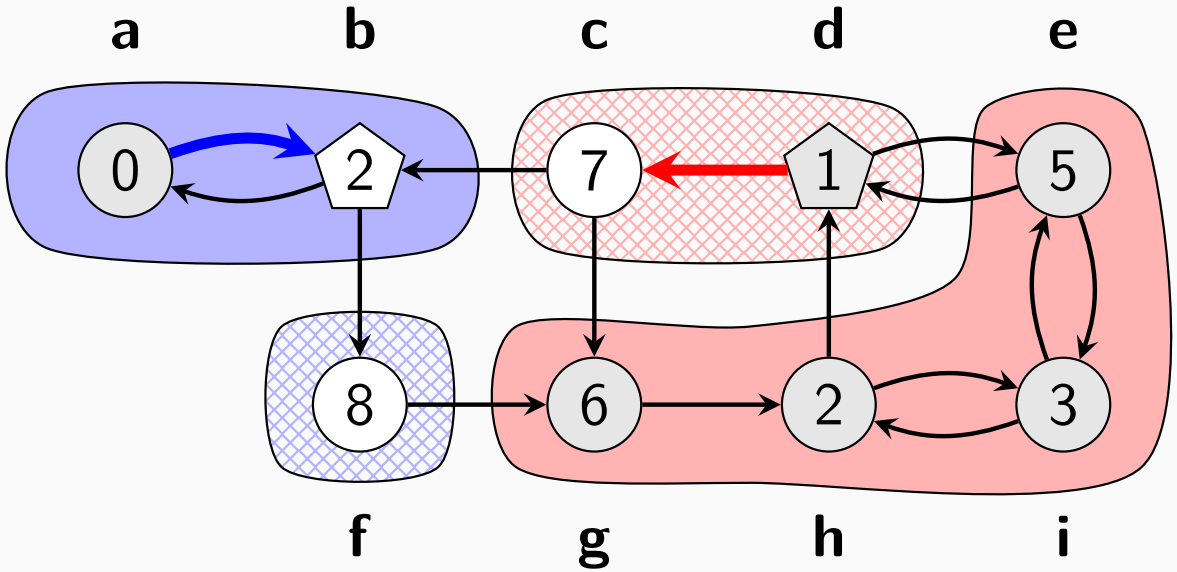

- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.

# ZIELONKA'S ALGORITHM



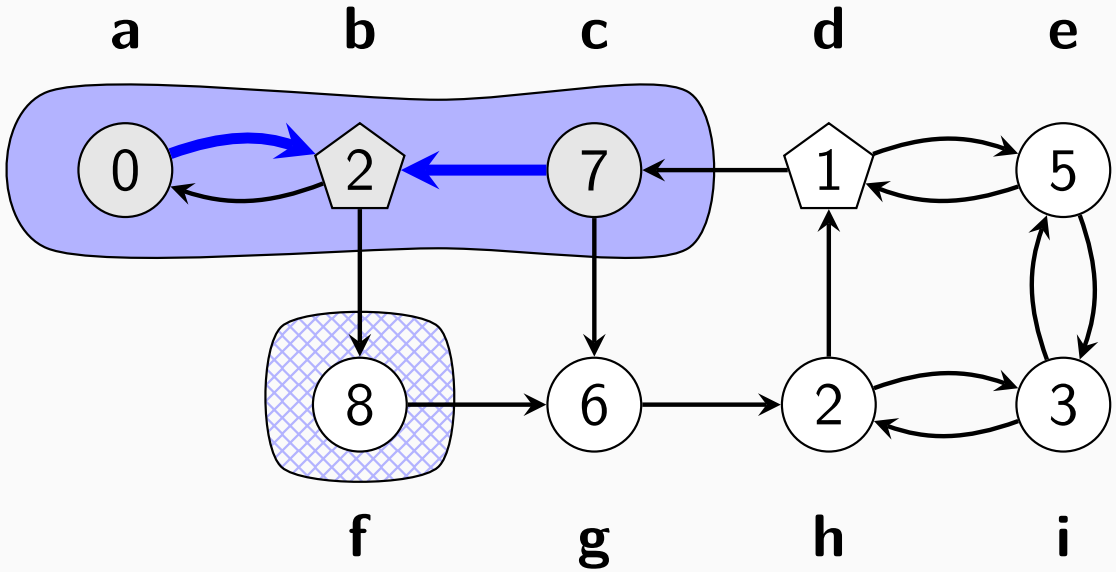

- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.
- Recompute region 2...

# ZIELONKA'S ALGORITHM



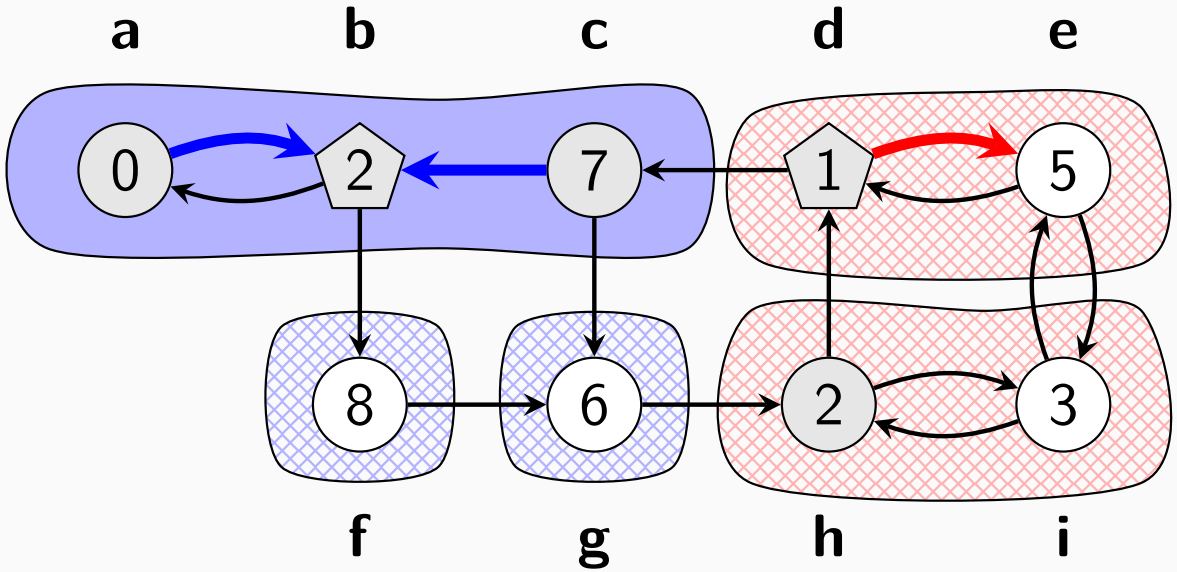

- Vertex **g** is attracted to the Odd region.
- So now recompute the (remainder of the) lower regions of Even.
- Recompute region 2... not attracted to lower Odd...
- Next: region 7: does the lower Even region attract from 7?
- Yes, the lower Even region attracts vertex **c**.
- Attract to Even, reset lower!

# ZIELONKA'S ALGORITHM



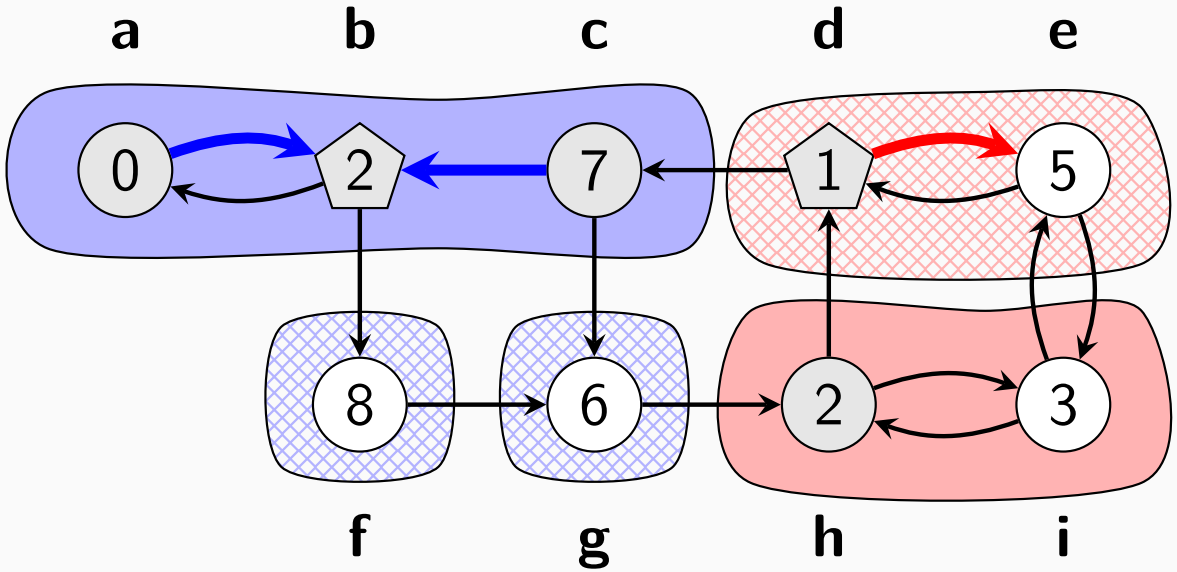

- Vertex **c** is attracted to the Even region.
- Recompute the remainder of the lower regions of Odd.

# ZIELONKA'S ALGORITHM



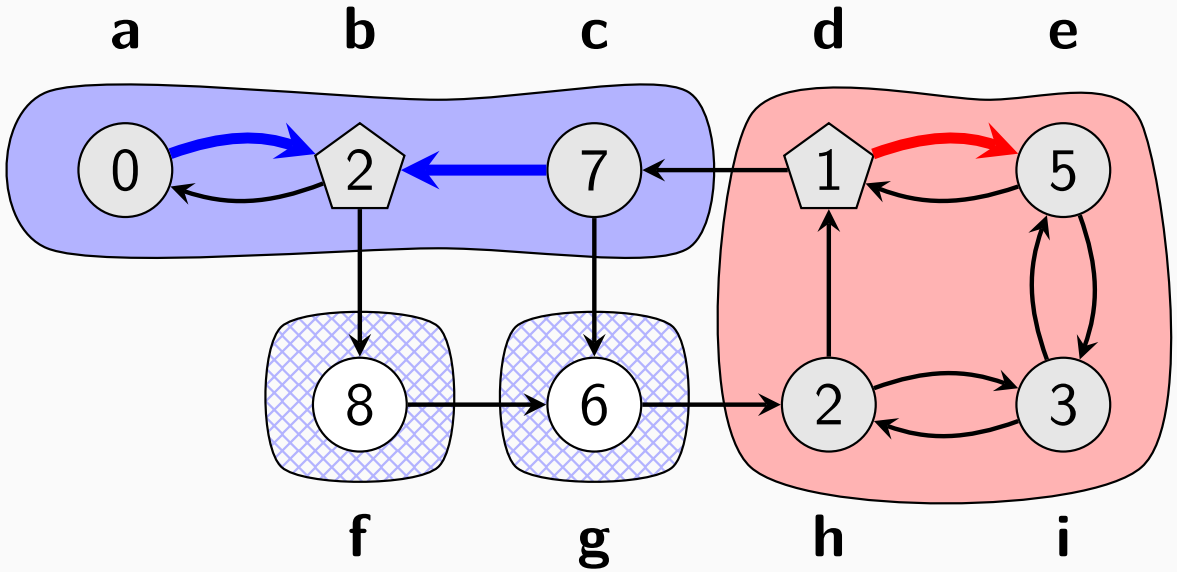

- Partition the remainder into regions 6, 5 and 3.
- Up in region 3: no attraction from Even.
- Up in region 5: no attraction from Even.
- Up in region 6: the lower Odd regions attract **g** again!

# ZIELONKA'S ALGORITHM



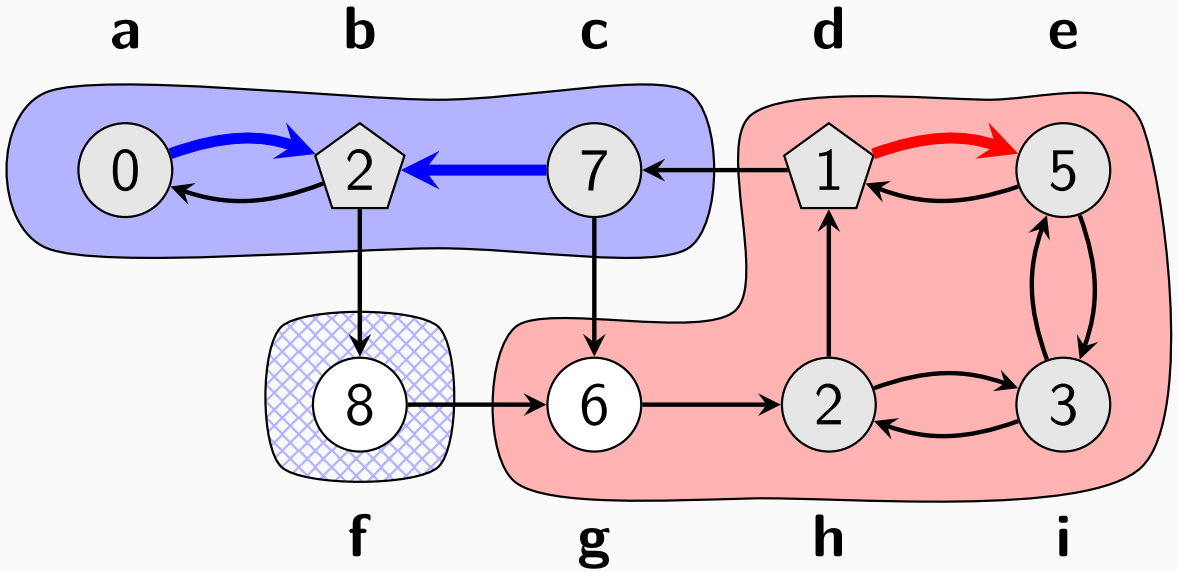

- Partition the remainder into regions 6, 5 and 3.
- Up in region 3: no attraction from Even.
- Up in region 5: no attraction from Even.
- Up in region 6: the lower Odd regions attract **g** again!

# ZIELONKA'S ALGORITHM



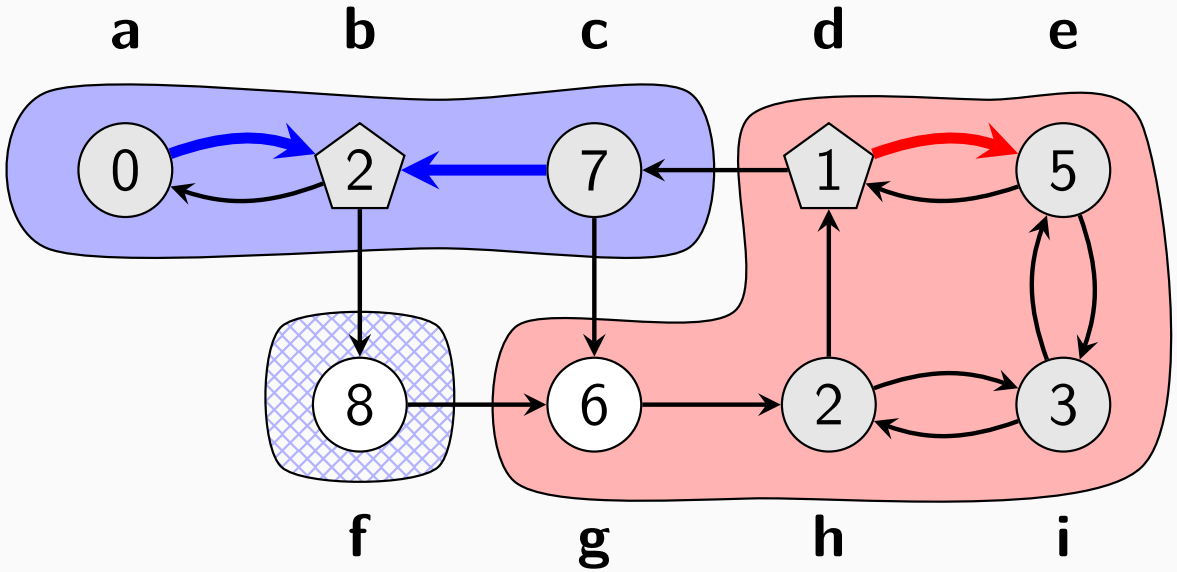

- Region 6: now the Odd region attracts vertex g again.
- Attract to Odd, reset the lower game.

# ZIELONKA'S ALGORITHM



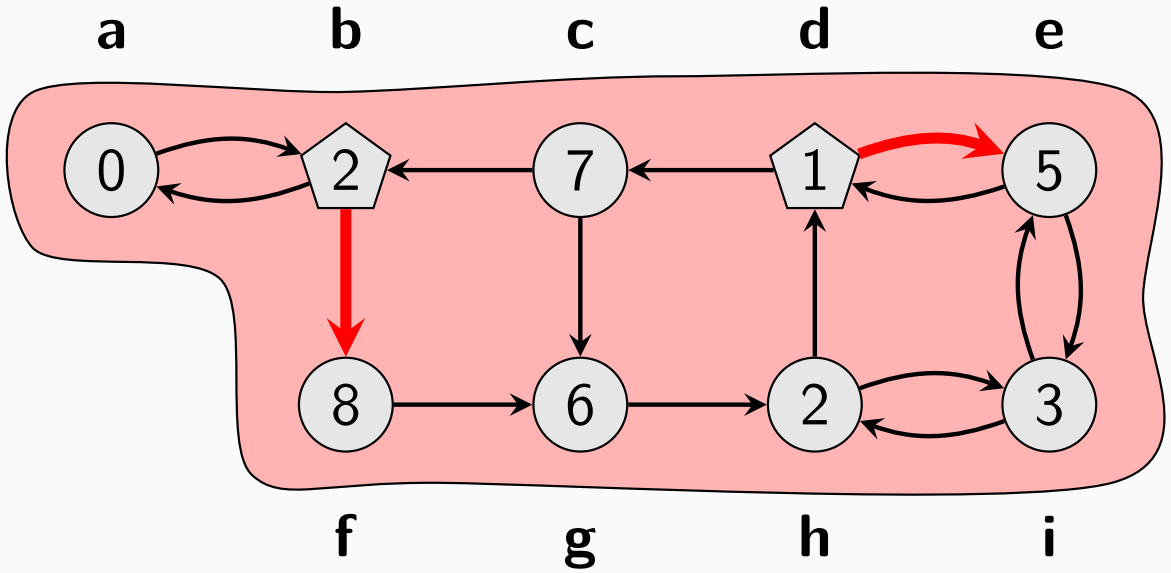

- Vertex **g** is attracted to the Odd region.
- Recursive game of 6 is empty.
- Up to region 8, does Odd now attract 8?
- Yes: vertex **f** is attracted to the Odd region.
- Attracting to Odd now attracts all remaining vertices...

# ZIELONKA'S ALGORITHM




- Vertex **g** is attracted to the Odd region.
- Recursive game of 6 is empty.
- Up to region 8, does Odd now attract 8?

# ZIELONKA'S ALGORITHM




- Final result, entire game won by player Odd.

## Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set $M$
  Meaning: how much player Odd "prefers" a vertex

## Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set $M$
  Meaning: how much player Odd "prefers" a vertex
- Special top element $\top \in M$
  Meaning: definitely won by Odd

## Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set $M$
  Meaning: how much player Odd "prefers" a vertex
- Special top element $\top \in M$
  Meaning: definitely won by Odd
- An edge $(v, u)$ is $\mu$-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$

  Meaning: increase value when we see odd priorities, until $\top$

## Outline 1/3

- Every vertex has a value a.k.a. measure from an ordered set $M$
  Meaning: how much player Odd "prefers" a vertex
- Special top element $\top \in M$
  Meaning: definitely won by Odd
- An edge $(v, u)$ is $\mu$-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$

  Meaning: increase value when we see odd priorities, until $\top$
- A mapping $\mu \colon V \to M$ is an Even progress measure if:
  - for every even-controlled vertex: some edge is $\mu$-progressive
  - for every odd-controlled vertex: all edges are $\mu$-progressive
  - and it is the smallest $\mu$ where this condition holds

  Meaning: Even chooses lowest escape; Odd chooses best strategy

## Outline 2/3

- A mapping $\mu\colon V \to M$ is an Even progress measure if:
  - every even-controlled vertex: some edge is $\mu$-progressive
  - every odd-controlled vertex: all edges are $\mu$-progressive
  - and it is the smallest $\mu$ where this condition holds
- An edge $(v, u)$ is $\mu$-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$

## Outline 2/3

- A mapping $\mu\colon V \to M$ is an Even progress measure if:
  - every even-controlled vertex: some edge is $\mu$-progressive
  - every odd-controlled vertex: all edges are $\mu$-progressive
  - and it is the smallest $\mu$ where this condition holds
- An edge $(v, u)$ is $\mu$-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$
- Even progress measure is a least fixed point
  - Start with $\mu = V \to \bot$ (the lowest element of $M$)
  - Repeatedly "lift" value $\mu(v)$ to locally repair the condition
    - Even-controlled: find lowest progressive successor (the escape for Even)
    - Odd-controlled: find highest progressive successor (the strategy for Odd)

## Outline 2/3

- A mapping $\mu\colon V \to M$ is an Even progress measure if:
  - every even-controlled vertex: some edge is $\mu$-progressive
  - every odd-controlled vertex: all edges are $\mu$-progressive
  - and it is the smallest $\mu$ where this condition holds
- An edge $(v, u)$ is $\mu$-progressive if:
  - if $\mathsf{pr}(v)$ is even and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) = \mu(u) = \top$
- Even progress measure is a least fixed point
  - Start with $\mu = V \to \bot$ (the lowest element of $M$)
  - Repeatedly "lift" value $\mu(v)$ to locally repair the condition
    - Even-controlled: find lowest progressive successor (the escape for Even)
    - Odd-controlled: find highest progressive successor (the strategy for Odd)
- After computing the Even progress measure:
  - all $\top$ won by Odd, all non-$\top$ won by Even
  - strategy for Even: choose any $\mu$-progressive edge

## Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with $n_p$ the number of vertices with priority $p$
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.

## Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with $n_p$ the number of vertices with priority $p$
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.
- A total order $<$ which is lexicographic:

$$\begin{aligned} \langle 1,0,0,0 \rangle &< \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle &< \langle 4,3,0,0 \rangle \end{aligned}$$

## Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with $n_p$ the number of vertices with priority $p$
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.
- A total order $<$ which is lexicographic:

$$\begin{aligned} \langle 1,0,0,0 \rangle &< \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle &< \langle 4,3,0,0 \rangle \end{aligned}$$

- A $p$-truncation keeps only elements $r_i$ with $i \geq p$:

$$\begin{aligned} \langle 1,2,3,2 \rangle|_2 &= \langle 1,2,3 \rangle \\ \langle 1,2,3,2 \rangle|_5 &= \langle 1,2 \rangle \\ \langle 1,2,3,2 \rangle|_8 &= \langle \rangle \end{aligned}$$

## Outline 3/3

- Measures are tuples $\langle r_7, r_5, r_3, r_1 \rangle$ (with highest odd priority 7)
- Each $r_p = [0..n_p]$ with $n_p$ the number of vertices with priority $p$
- Example: $M = (\{0\} \times \{0,1,2\} \times \{0\} \times \{0,1\}) \cup \{\top\}$ if there are 2 vertices of priority 5 and 1 vertex of priority 1.
- A total order $<$ which is lexicographic:

$$\begin{aligned} \langle 1,0,0,0 \rangle &< \langle 1,0,0,1 \rangle \\ \langle 4,2,10,5 \rangle &< \langle 4,3,0,0 \rangle \end{aligned}$$

- A $p$-truncation keeps only elements $r_i$ with $i \geq p$:

$$\begin{aligned} \langle 1,2,3,2 \rangle|_2 &= \langle 1,2,3 \rangle \\ \langle 1,2,3,2 \rangle|_5 &= \langle 1,2 \rangle \\ \langle 1,2,3,2 \rangle|_8 &= \langle \rangle \end{aligned}$$

- Additional notation: $m_1 >_p m_2 \Leftrightarrow m_1|_p > m_2|_p$
  Compare $p$-truncated measures (or: compare until $r_p$)

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m, p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{aligned} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) &= \end{aligned}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{aligned} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) &= \end{aligned}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m, p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \langle 3,2,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{array}{lcl} \texttt{Prog}(\langle 3,2,3,2\rangle, 0) & = & \langle 3,2,3,2\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 1) & = & \langle 3,2,3,3\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 2) & = & \langle 3,2,3,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 3) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 4) & = & \langle 3,2,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 5) & = & \langle 3,3,0,0\rangle \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 6) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 7) & = & \\ \texttt{Prog}(\langle 3,2,3,2\rangle, 8) & = & \end{array}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m, p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{aligned}
\texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \langle 3,2,3,2\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \langle 3,2,3,3\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \langle 3,2,3,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \langle 3,2,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \langle 3,0,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 8) &=
\end{aligned}$$

## The Prog function

Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

$$\begin{aligned}
\texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \langle 3,2,3,2\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \langle 3,2,3,3\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \langle 3,2,3,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \langle 3,2,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \langle 3,0,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \top \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 8) &=
\end{aligned}$$

## The Prog function

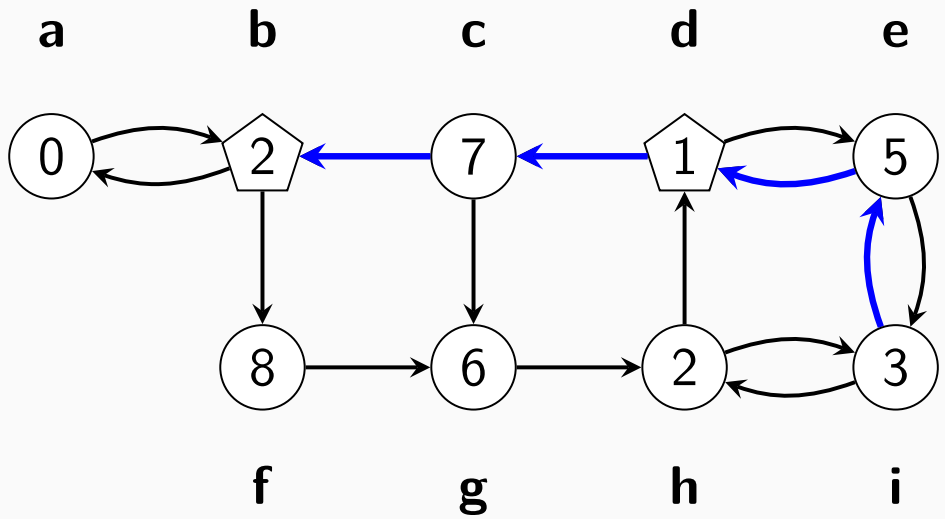
Playing from a vertex with priority $p$ to a vertex with measure $m$ yields:

$$\texttt{Prog}(m,p) := \begin{cases} \min\{\, m' \in M \mid m' \geq_p m \,\} & p \text{ is even} \\ \min\{\, m' \in M \mid m' >_p m \,\} & p \text{ is odd} \end{cases}$$

How to compute: truncate, then (if odd) increase by 1 and carry-over.
Example (with highest value 3 for all elements):

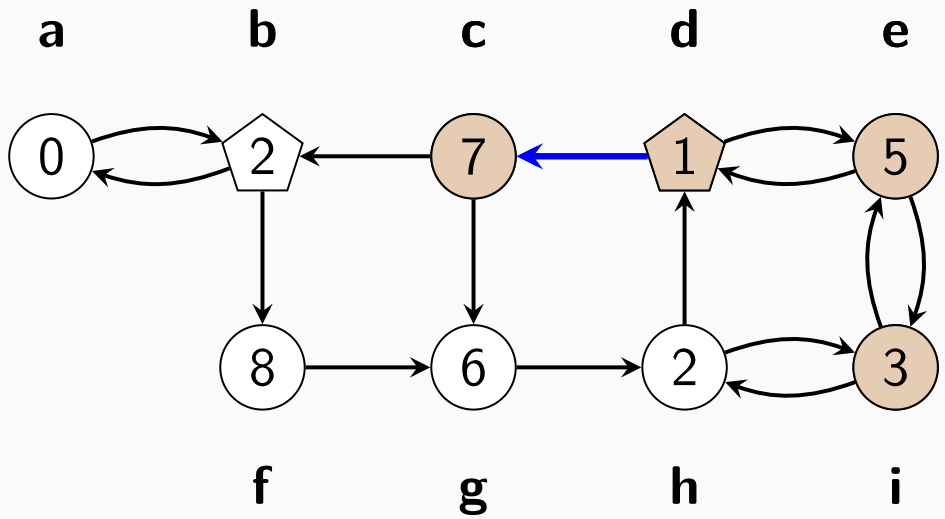
$$\begin{aligned}
\texttt{Prog}(\langle 3,2,3,2\rangle, 0) &= \langle 3,2,3,2\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 1) &= \langle 3,2,3,3\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 2) &= \langle 3,2,3,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 3) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 4) &= \langle 3,2,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 5) &= \langle 3,3,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 6) &= \langle 3,0,0,0\rangle \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 7) &= \top \\
\texttt{Prog}(\langle 3,2,3,2\rangle, 8) &= \langle 0,0,0,0\rangle
\end{aligned}$$

# SMALL PROGRESS MEASURES



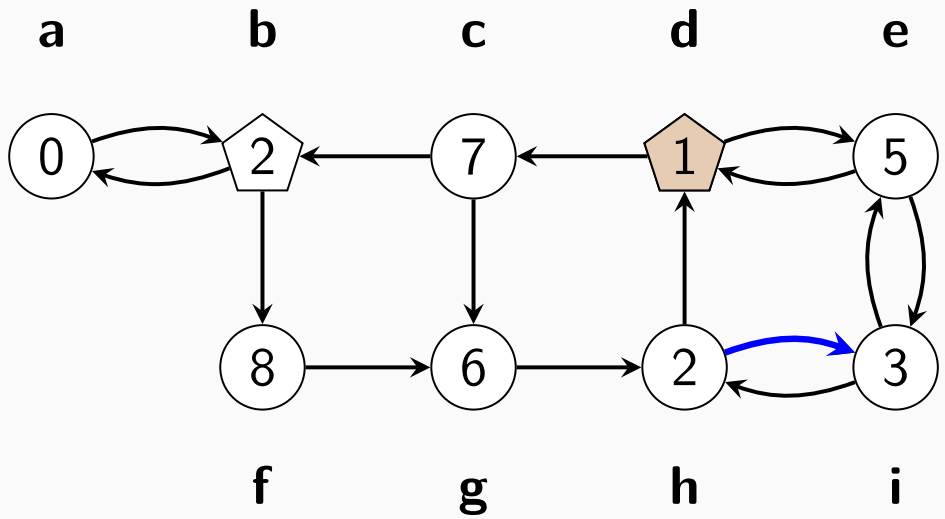

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 0,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,1\rangle$ | 1 |
| **e** | $\langle 0,0,-,-\rangle$ | to | $\langle 0,1,-,-\rangle$ | 5 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,1,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



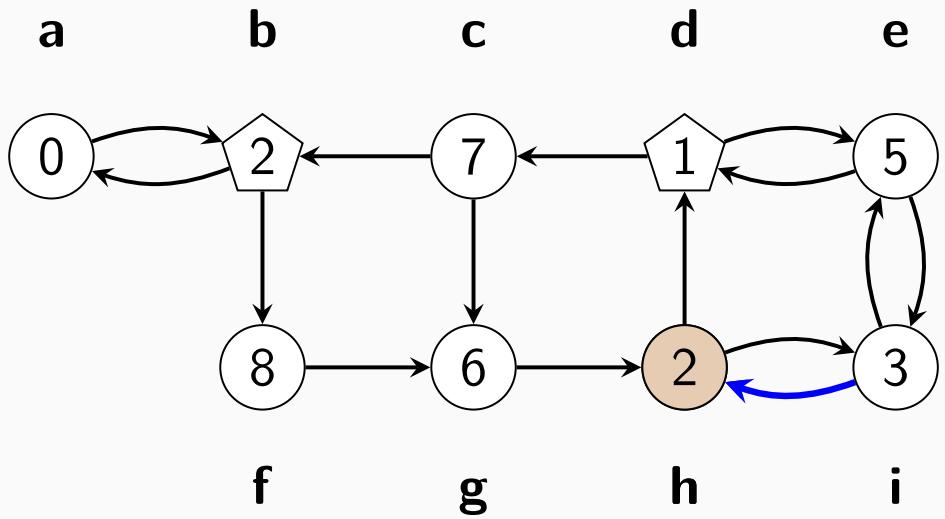

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 0,1,-,-\rangle$ | to | $\langle 0,1,-,-\rangle$ | 5 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **i** | $\langle 0,0,1,-\rangle$ | to | $\langle 0,0,1,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



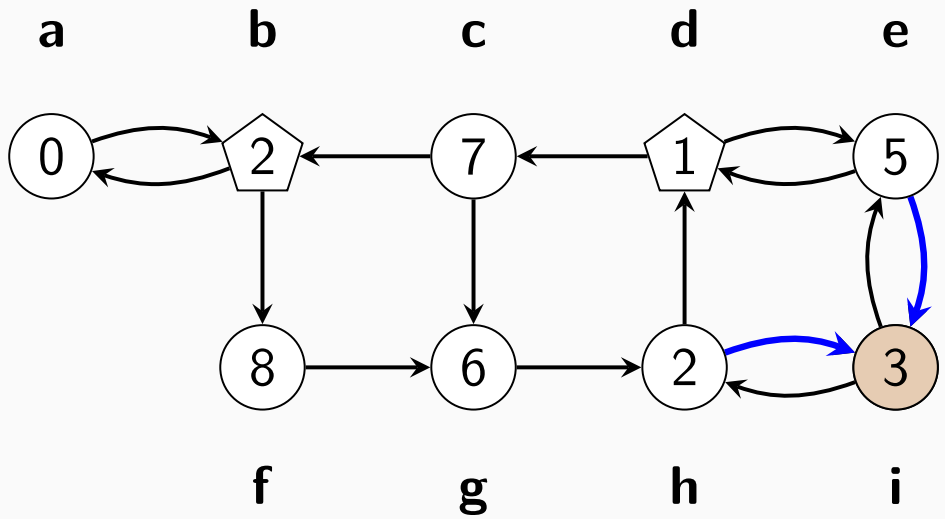

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 0,1,-,-\rangle$ | to | $\langle 0,1,-,-\rangle$ | 5 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,1,-\rangle$ | 23 |
| **i** | $\langle 0,0,1,-\rangle$ | to | $\langle 0,0,1,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



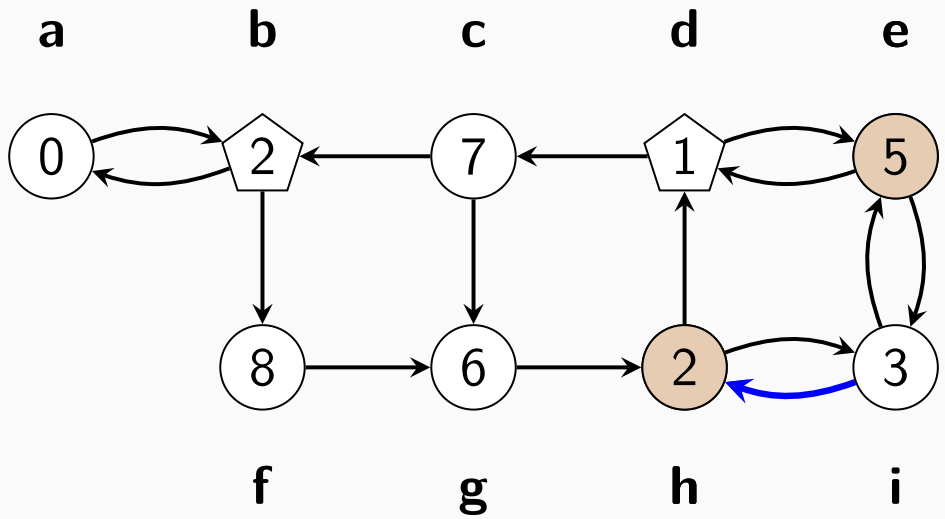

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 0,1,-,-\rangle$ | to | $\langle 0,1,-,-\rangle$ | 5 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,1,-\rangle$ | to | $\langle 0,0,1,-\rangle$ | 23 |
| **i** | $\langle 0,0,1,-\rangle$ | to | $\langle 0,1,0,-\rangle$ | 323 |

# SMALL PROGRESS MEASURES



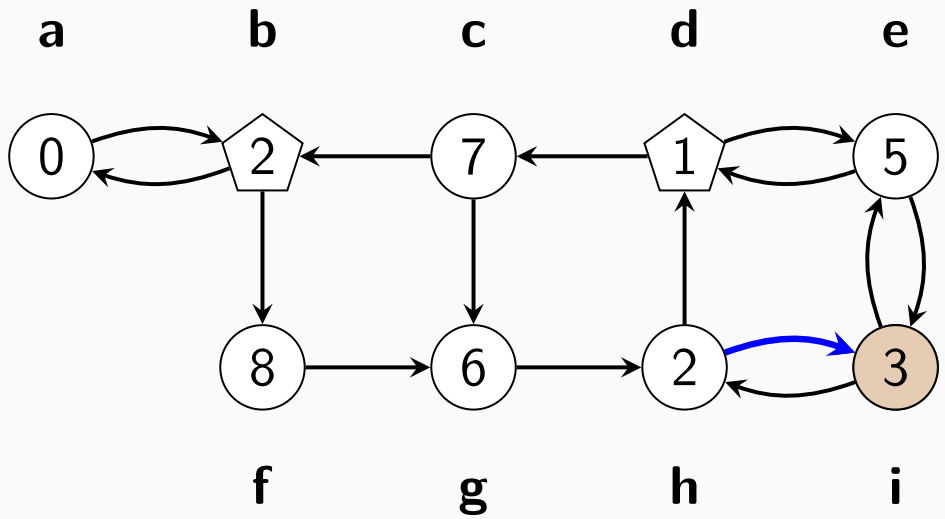

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 0,1,-,-\rangle$ | to | $\langle 1,0,-,-\rangle$ | 5323 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,1,-\rangle$ | to | $\langle 0,1,0,-\rangle$ | 2323 |
| **i** | $\langle 0,1,0,-\rangle$ | to | $\langle 0,1,0,-\rangle$ | 323 |

# SMALL PROGRESS MEASURES



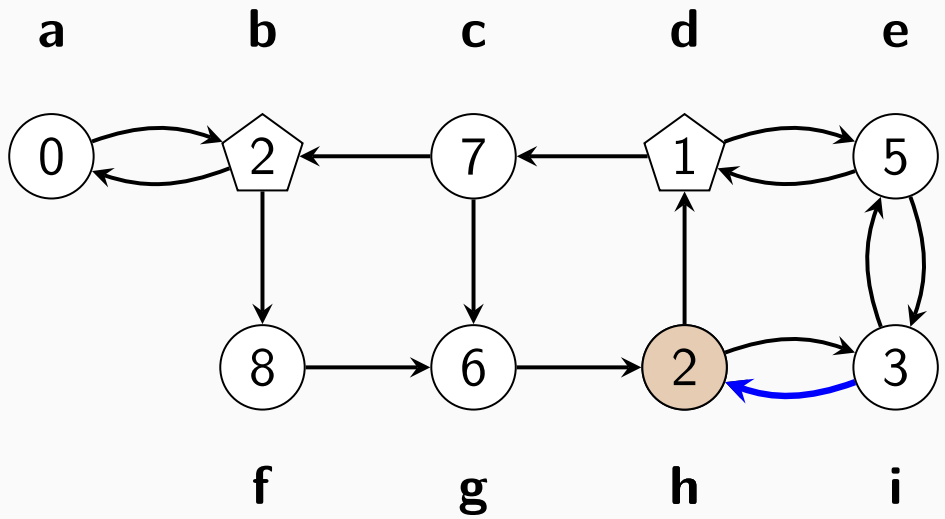

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 1,0,-,-\rangle$ | to | $\langle 1,0,-,-\rangle$ | 5323 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,1,0,-\rangle$ | to | $\langle 0,1,0,-\rangle$ | 2323 |
| **i** | $\langle 0,1,0,-\rangle$ | to | $\langle 0,1,1,-\rangle$ | 32323 |

# SMALL PROGRESS MEASURES



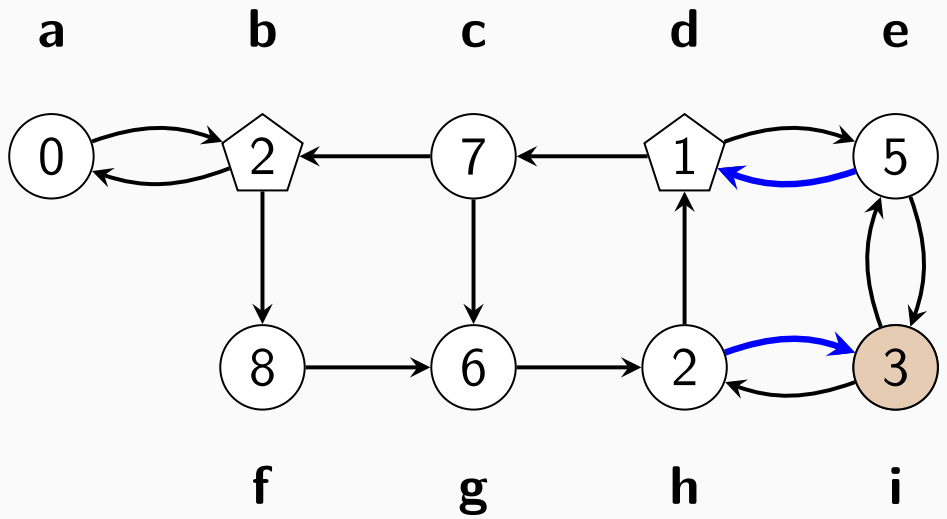

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 1,0,-,-\rangle$ | to | $\langle 1,0,-,-\rangle$ | 5323 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,1,0,-\rangle$ | to | $\langle 0,1,1,-\rangle$ | 232323 |
| **i** | $\langle 0,1,1,-\rangle$ | to | $\langle 0,1,1,-\rangle$ | 32323 |

# SMALL PROGRESS MEASURES



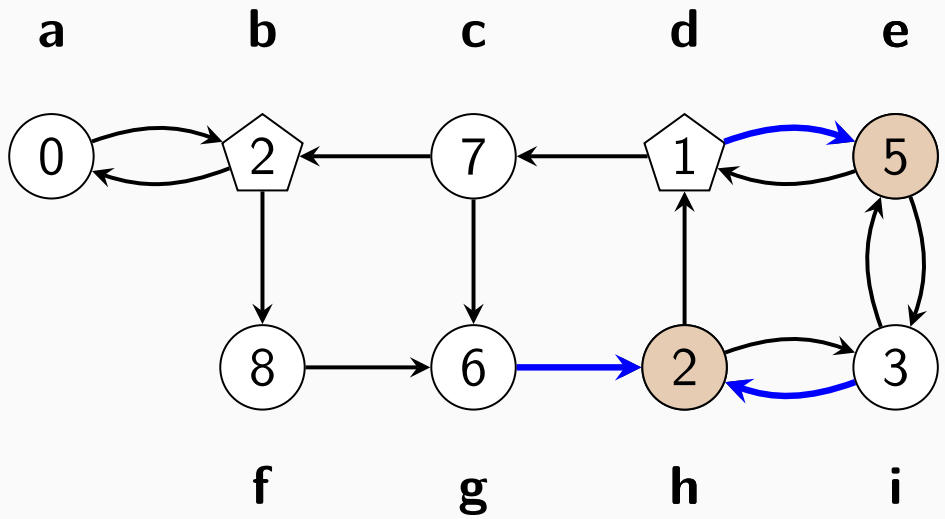

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 1,0,-,-\rangle$ | to | $\langle 1,0,-,-\rangle$ | 5323 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,1,1,-\rangle$ | to | $\langle 0,1,1,-\rangle$ | 232323 |
| **i** | $\langle 0,1,1,-\rangle$ | to | $\langle 1,0,0,-\rangle$ | 3232323 |

# SMALL PROGRESS MEASURES



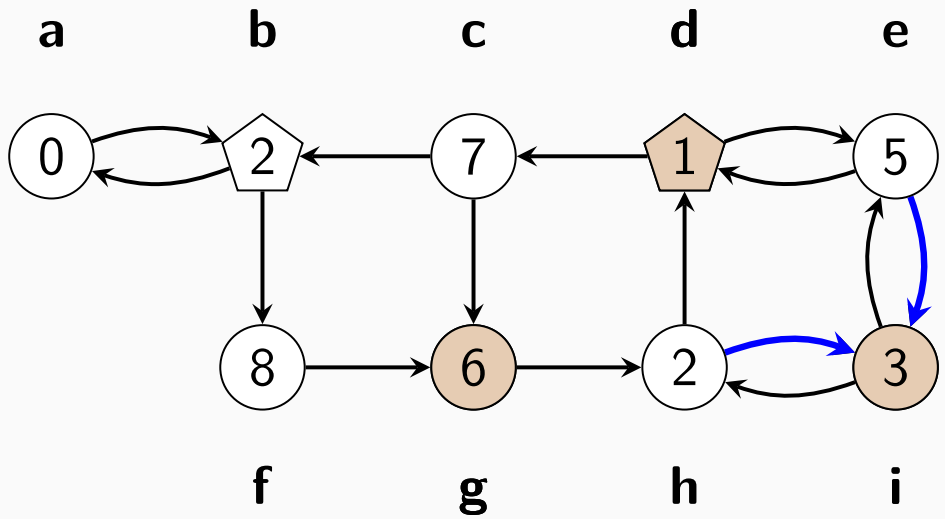

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,0,0,1\rangle$ | 17 |
| **e** | $\langle 1,0,-,-\rangle$ | to | $\langle 1,1,-,-\rangle$ | 517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 0,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,1,1,-\rangle$ | to | $\langle 1,0,0,-\rangle$ | 23232323 |
| **i** | $\langle 1,0,0,-\rangle$ | to | $\langle 1,0,0,-\rangle$ | 3232323 |

# SMALL PROGRESS MEASURES



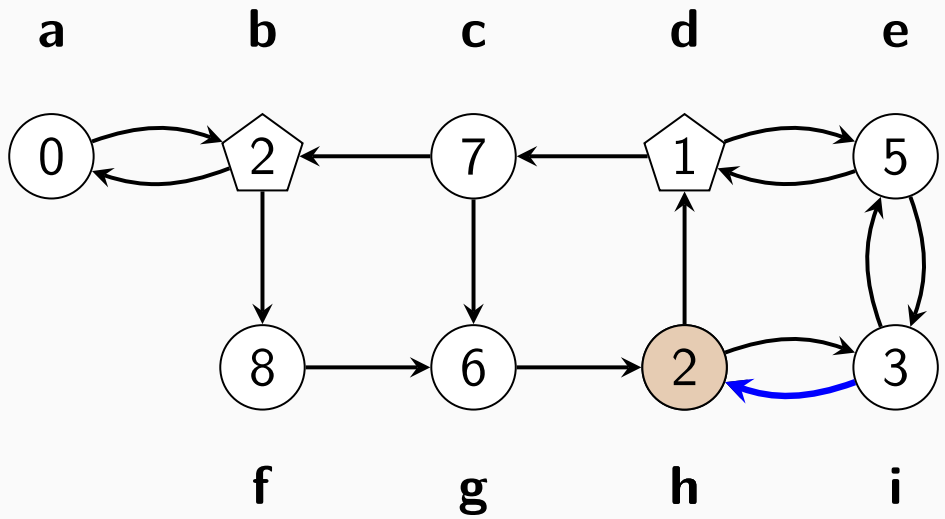

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,0,0,1\rangle$ | to | $\langle 1,1,0,1\rangle$ | 1517 |
| **e** | $\langle 1,1,-,-\rangle$ | to | $\langle 1,1,-,-\rangle$ | 517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,0,0,-\rangle$ | to | $\langle 1,0,0,-\rangle$ | 23232323 |
| **i** | $\langle 1,0,0,-\rangle$ | to | $\langle 1,0,1,-\rangle$ | 323232323 |

# SMALL PROGRESS MEASURES



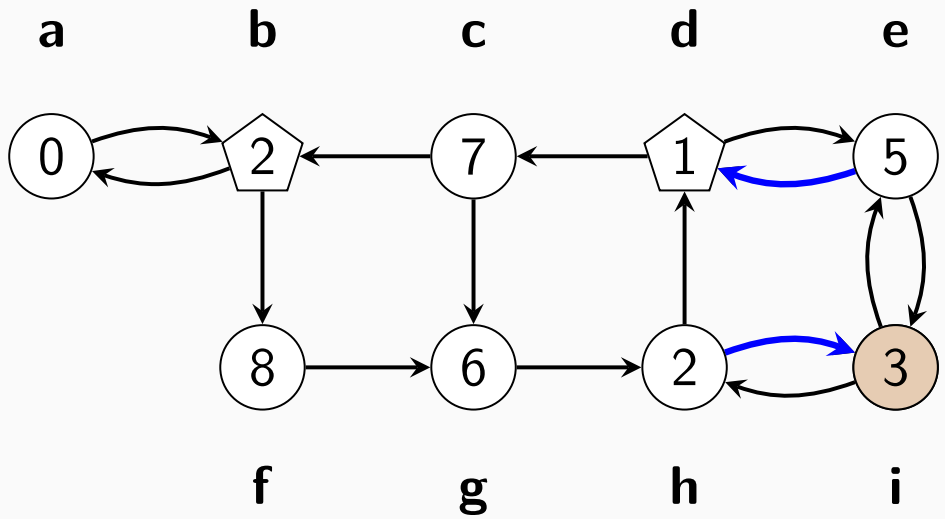

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,1,0,1\rangle$ | to | $\langle 1,1,0,1\rangle$ | 1517 |
| **e** | $\langle 1,1,-,-\rangle$ | to | $\langle 1,1,-,-\rangle$ | 5323232323 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,0,0,-\rangle$ | to | $\langle 1,0,1,-\rangle$ | 2323232323 |
| **i** | $\langle 1,0,1,-\rangle$ | to | $\langle 1,0,1,-\rangle$ | 323232323 |

# SMALL PROGRESS MEASURES



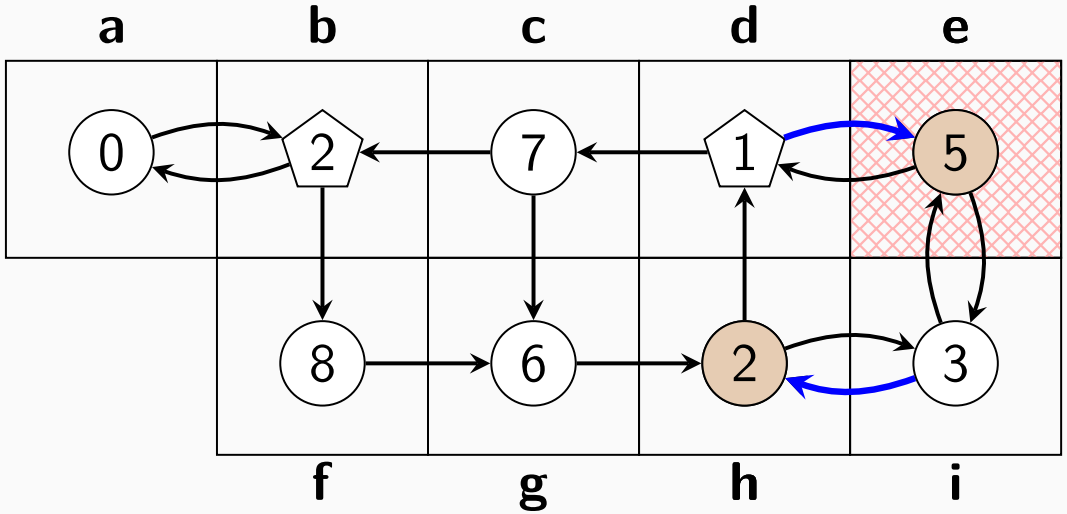

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,1,0,1\rangle$ | to | $\langle 1,1,0,1\rangle$ | 1517 |
| **e** | $\langle 1,1,-,-\rangle$ | to | $\langle 1,1,-,-\rangle$ | 5323232323 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,0,1,-\rangle$ | to | $\langle 1,0,1,-\rangle$ | 2323232323 |
| **i** | $\langle 1,0,1,-\rangle$ | to | $\langle 1,1,0,-\rangle$ | 32323232323 |

# SMALL PROGRESS MEASURES



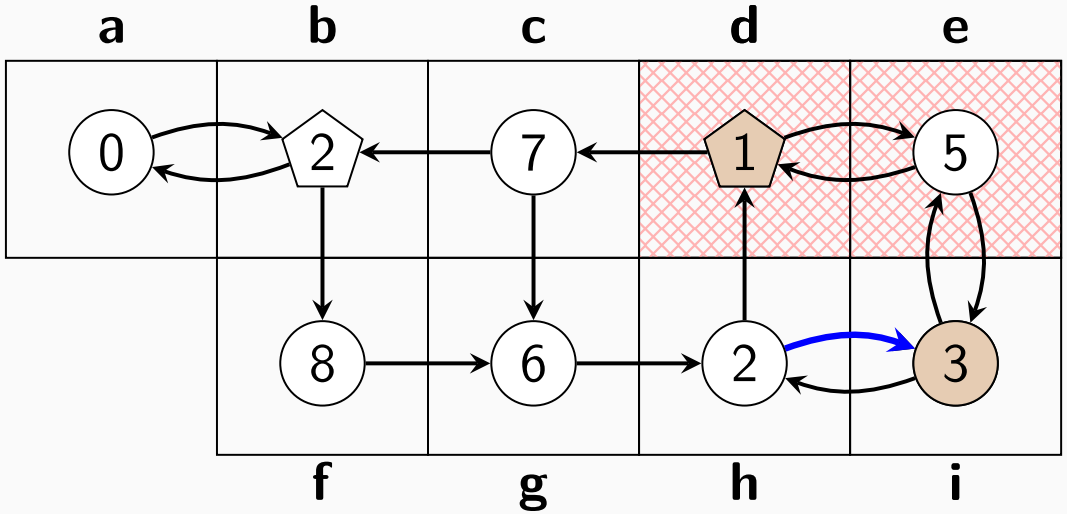

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,1,0,1\rangle$ | to | $\langle 1,1,0,1\rangle$ | 1517 |
| **e** | $\langle 1,1,-,-\rangle$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,0,1,-\rangle$ | to | $\langle 1,1,0,-\rangle$ | 232323232323 |
| **i** | $\langle 1,1,0,-\rangle$ | to | $\langle 1,1,0,-\rangle$ | 32323232323 |

# SMALL PROGRESS MEASURES



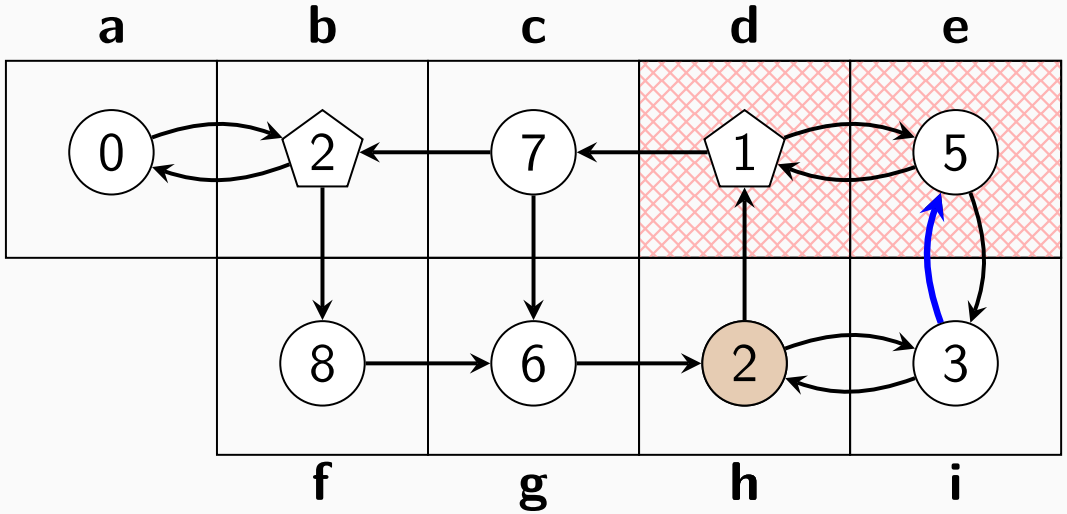

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\langle 1,1,0,1\rangle$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,1,0,-\rangle$ | to | $\langle 1,1,0,-\rangle$ | 232323232323 |
| **i** | $\langle 1,1,0,-\rangle$ | to | $\langle 1,1,1,-\rangle$ | 3232323232323 |

# SMALL PROGRESS MEASURES



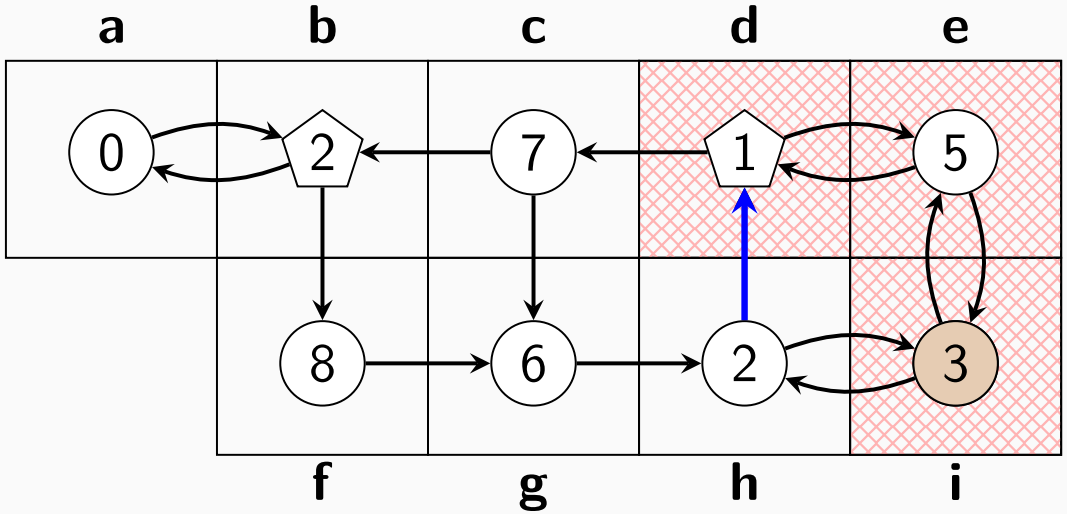

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,0,1,-\rangle$ | to | $\langle 1,1,1,-\rangle$ | 23232323232323 |
| **i** | $\langle 1,1,1,-\rangle$ | to | $\langle 1,1,1,-\rangle$ | 3232323232323 |

# SMALL PROGRESS MEASURES



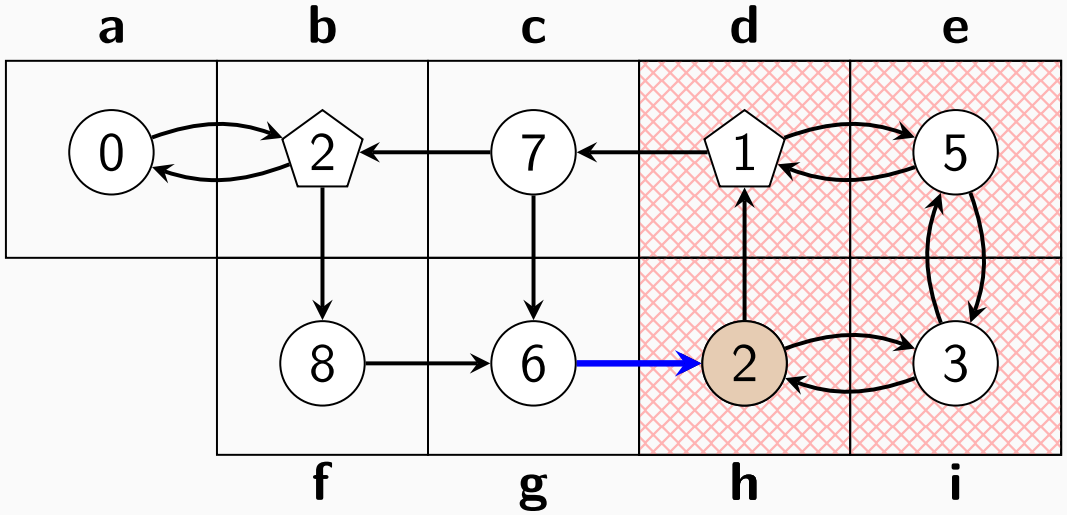

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,1,1,-\rangle$ | to | $\langle 1,1,1,-\rangle$ | 23232323232323 |
| **i** | $\langle 1,1,1,-\rangle$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES



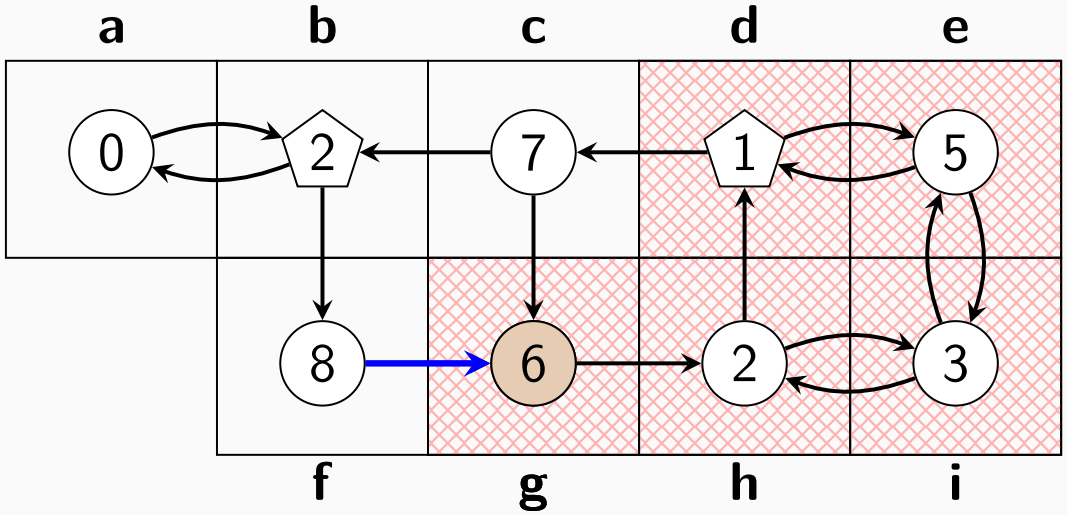

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 623232323 |
| **h** | $\langle 1,1,1,-\rangle$ | to | $\top$ | 2151517 |
| **i** | $\top$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES



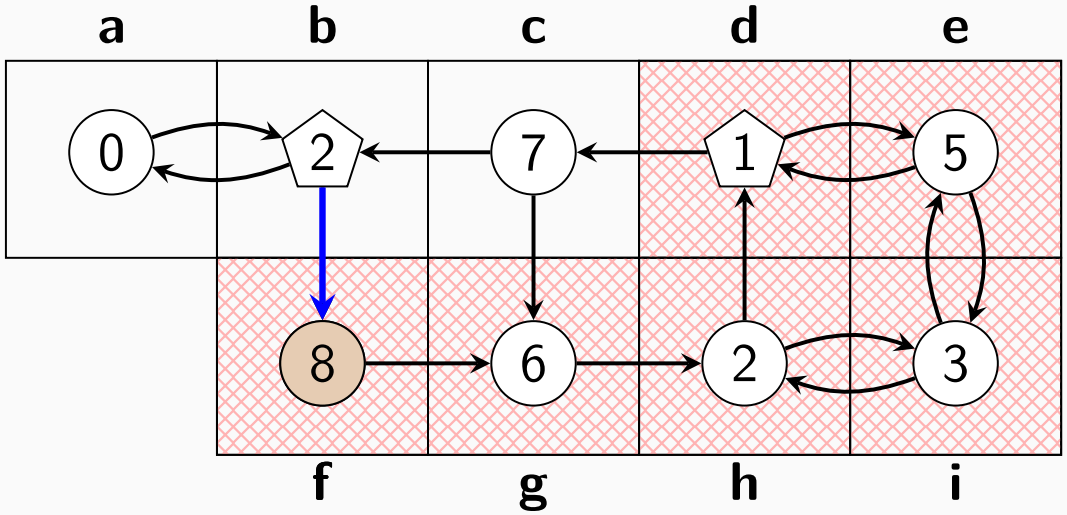

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\langle -,-,-,-\rangle$ | 8 |
| **g** | $\langle 1,-,-,-\rangle$ | to | $\top$ | 62151517 |
| **h** | $\top$ | to | $\top$ | 2151517 |
| **i** | $\top$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES



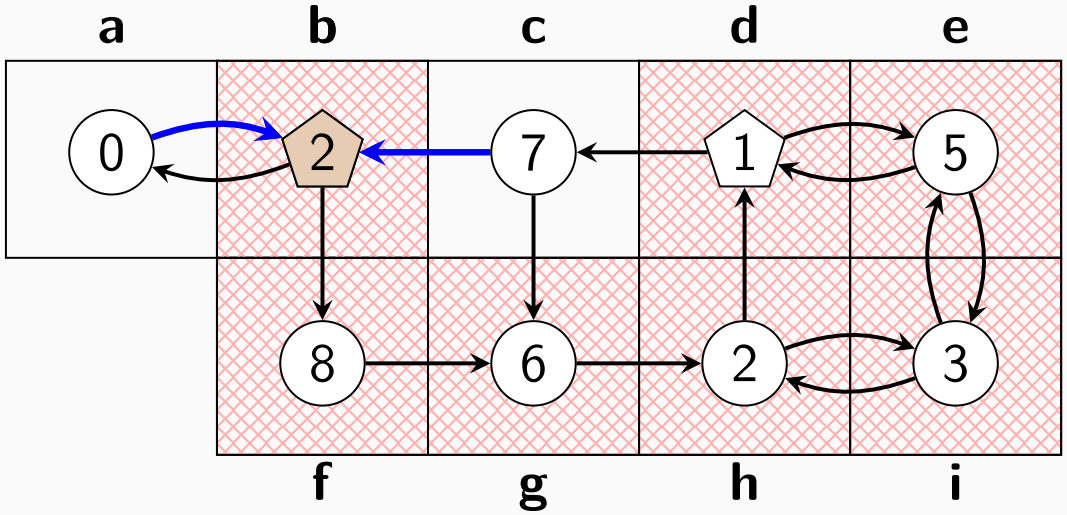

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\langle 0,0,0,-\rangle$ | 2 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\langle -,-,-,-\rangle$ | to | $\top$ | 862151517 |
| **g** | $\top$ | to | $\top$ | 62151517 |
| **h** | $\top$ | to | $\top$ | 2151517 |
| **i** | $\top$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES



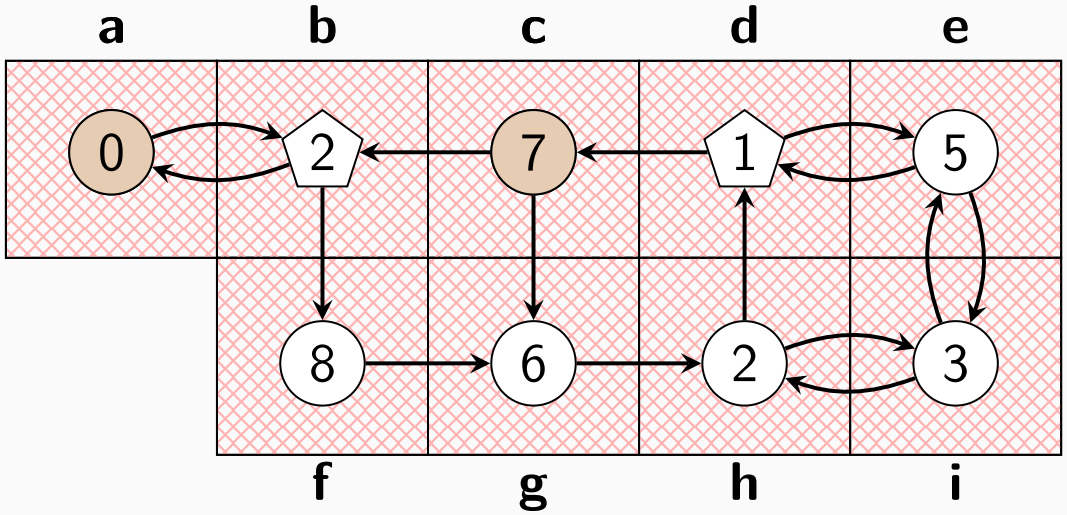

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\langle 0,0,0,0\rangle$ | 0 |
| **b** | $\langle 0,0,0,-\rangle$ | to | $\top$ | 2862151517 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\langle 1,-,-,-\rangle$ | 7 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\top$ | to | $\top$ | 862151517 |
| **g** | $\top$ | to | $\top$ | 62151517 |
| **h** | $\top$ | to | $\top$ | 2151517 |
| **i** | $\top$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES



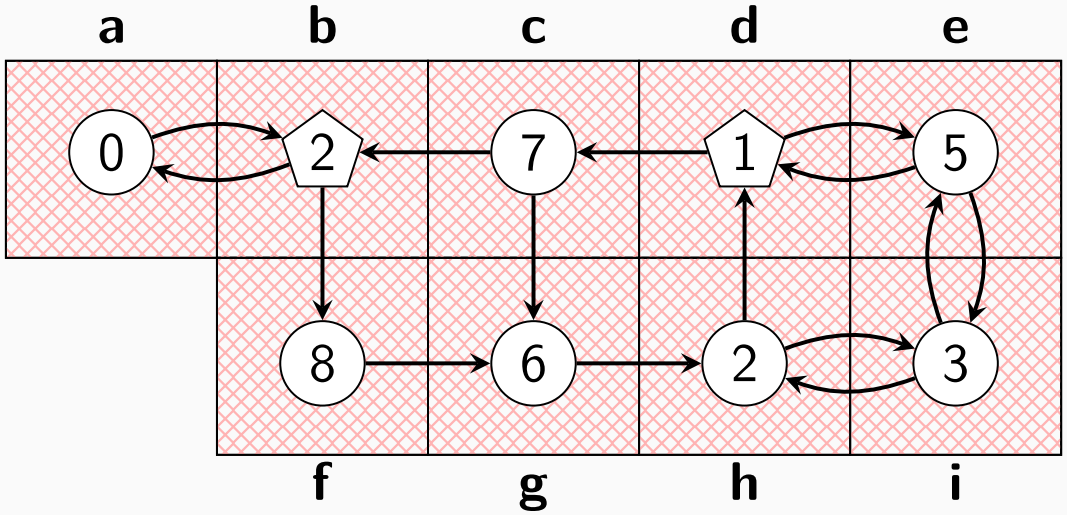

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0\rangle$ | to | $\top$ | 02862151517 |
| **b** | $\top$ | to | $\top$ | 2862151517 |
| **c** | $\langle 1,-,-,-\rangle$ | to | $\top$ | 762151517 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\top$ | to | $\top$ | 862151517 |
| **g** | $\top$ | to | $\top$ | 62151517 |
| **h** | $\top$ | to | $\top$ | 2151517 |
| **i** | $\top$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES




| | | | | |
|---|---|---|---|---|
| **a** | $\top$ | to | $\top$ | 02862151517 |
| **b** | $\top$ | to | $\top$ | 2862151517 |
| **c** | $\top$ | to | $\top$ | 762151517 |
| **d** | $\top$ | to | $\top$ | 151517 |
| **e** | $\top$ | to | $\top$ | 51517 |
| **f** | $\top$ | to | $\top$ | 862151517 |
| **g** | $\top$ | to | $\top$ | 62151517 |
| **h** | $\top$ | to | $\top$ | 2151517 |
| **i** | $\top$ | to | $\top$ | 351517 |

# SMALL PROGRESS MEASURES



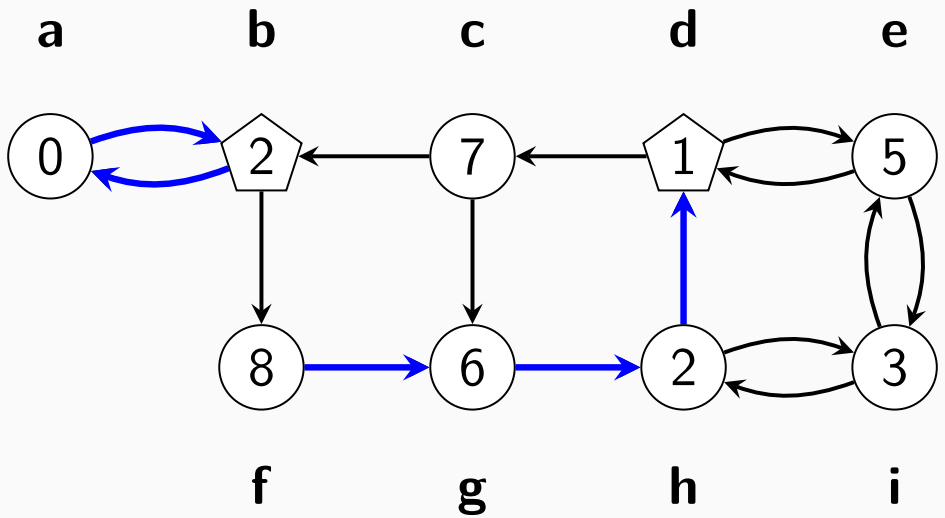

- All vertices are won by Odd
- No strategy for Even

## Odd progress measure

To get a strategy for Odd, we need an Odd progress measure

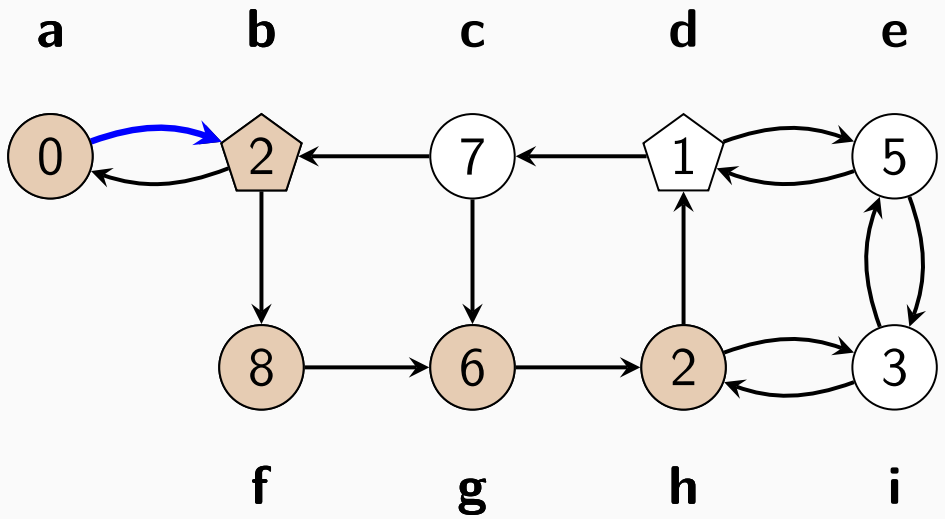
- Now we record how good a vertex is for player Even;
- $\top$ means won by player Even
- A mapping $\mu\colon V \to M$ is an Odd progress measure if:
  - every even-controlled vertex: all edges are $\mu$-progressive
  - every odd-controlled vertex: some edge is $\mu$-progressive
- An edge $(v, u)$ is now $\mu$-progressive if:
  - if $\mathsf{pr}(v)$ is odd and $\mu(v) \geq_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is even and $\mu(v) >_{\mathsf{pr}(v)} \mu(u)$
  - if $\mathsf{pr}(v)$ is even and $\mu(v) = \mu(u) = \top$

# SMALL PROGRESS MEASURES



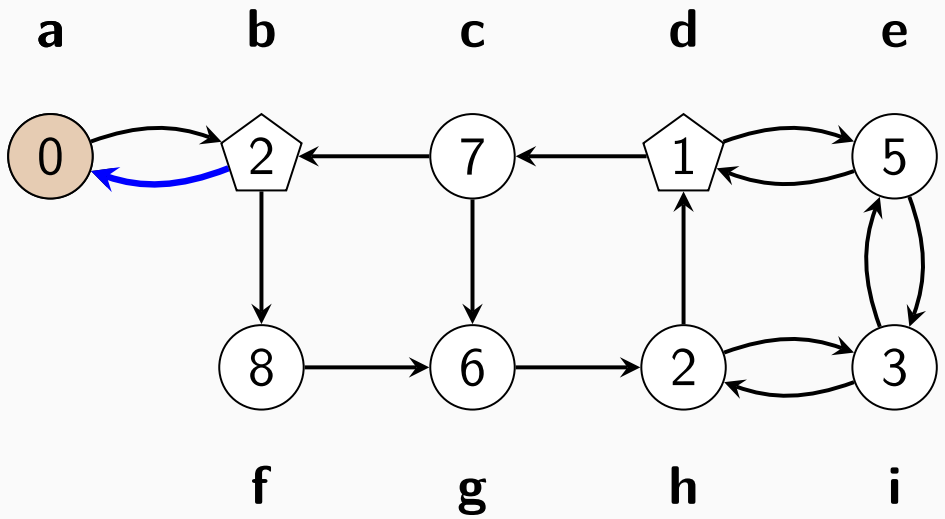

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0,0\rangle$ | to | $\langle 0,0,0,0,1\rangle$ | 0 |
| **b** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



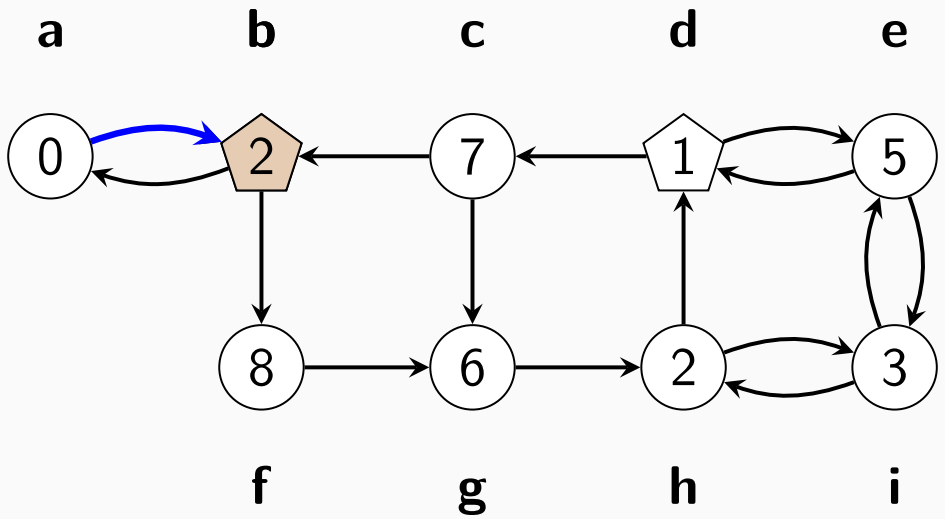

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,0,1\rangle$ | to | $\langle 0,0,0,1,1\rangle$ | 02 |
| **b** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



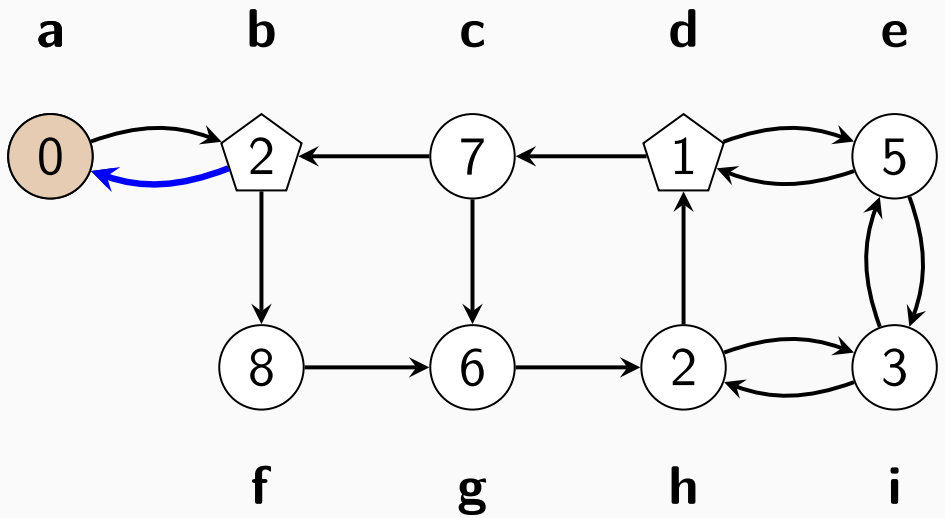

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,1,1\rangle$ | to | $\langle 0,0,0,1,1\rangle$ | 02 |
| **b** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,2,-\rangle$ | 202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



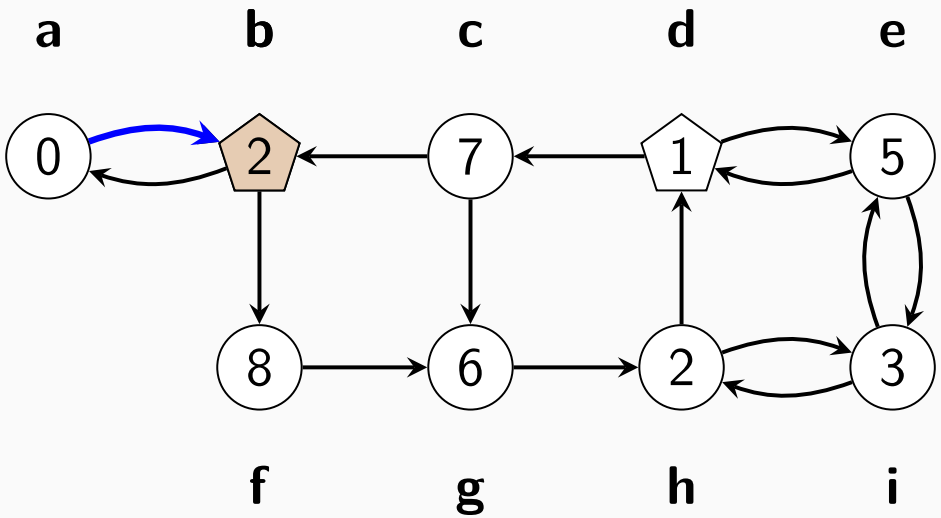

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,1,1\rangle$ | to | $\langle 0,0,0,2,1\rangle$ | 0202 |
| **b** | $\langle 0,0,0,2,-\rangle$ | to | $\langle 0,0,0,2,-\rangle$ | 202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



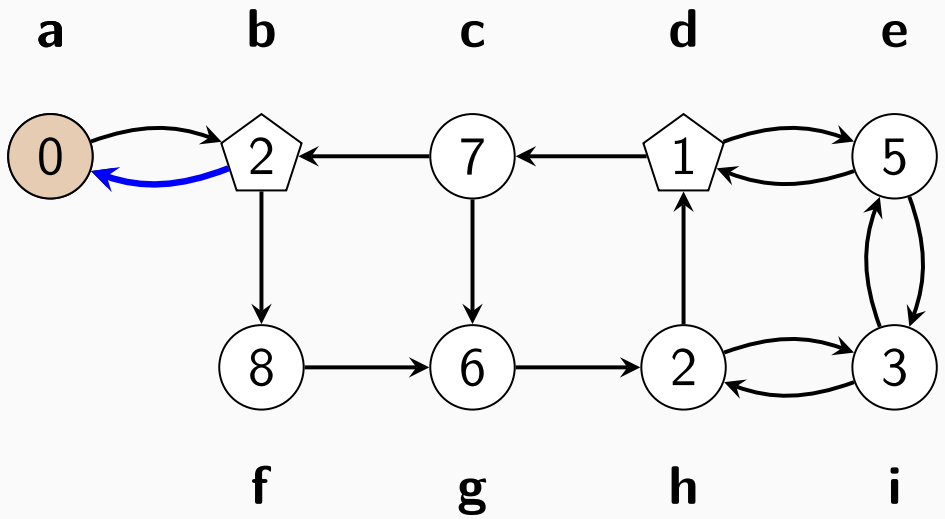

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,2,1\rangle$ | to | $\langle 0,0,0,2,1\rangle$ | 0202 |
| **b** | $\langle 0,0,0,2,-\rangle$ | to | $\langle 0,1,0,0,-\rangle$ | 20202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



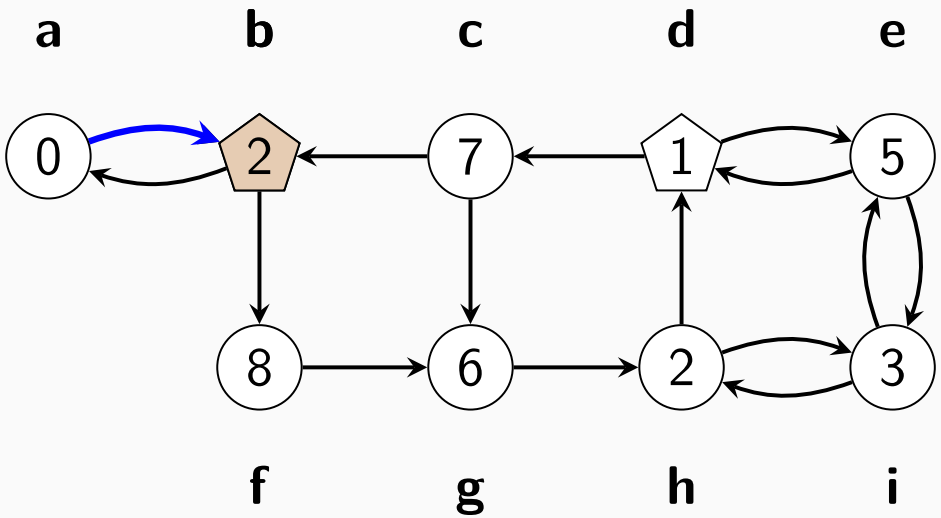

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,0,0,2,1\rangle$ | to | $\langle 0,1,0,0,1\rangle$ | 020202 |
| **b** | $\langle 0,1,0,0,-\rangle$ | to | $\langle 0,1,0,0,-\rangle$ | 20202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



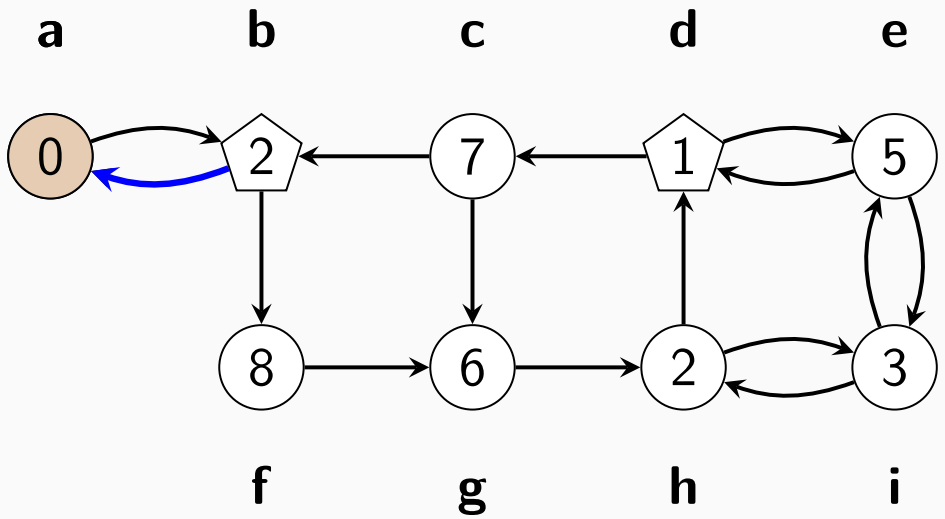

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,1,0,0,1\rangle$ | to | $\langle 0,1,0,0,1\rangle$ | 020202 |
| **b** | $\langle 0,1,0,0,-\rangle$ | to | $\langle 0,1,0,1,-\rangle$ | 2020202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



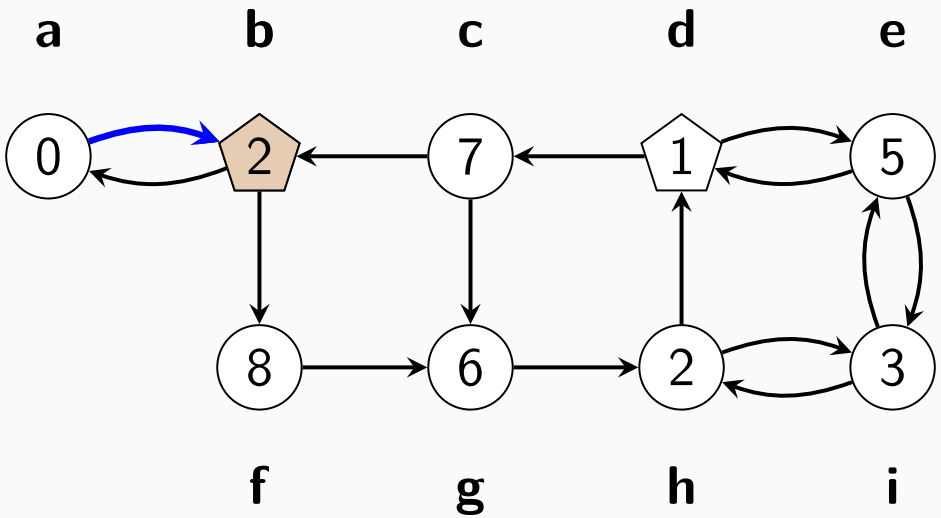

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,1,0,0,1\rangle$ | to | $\langle 0,1,0,1,1\rangle$ | 02020202 |
| **b** | $\langle 0,1,0,1,-\rangle$ | to | $\langle 0,1,0,1,-\rangle$ | 2020202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



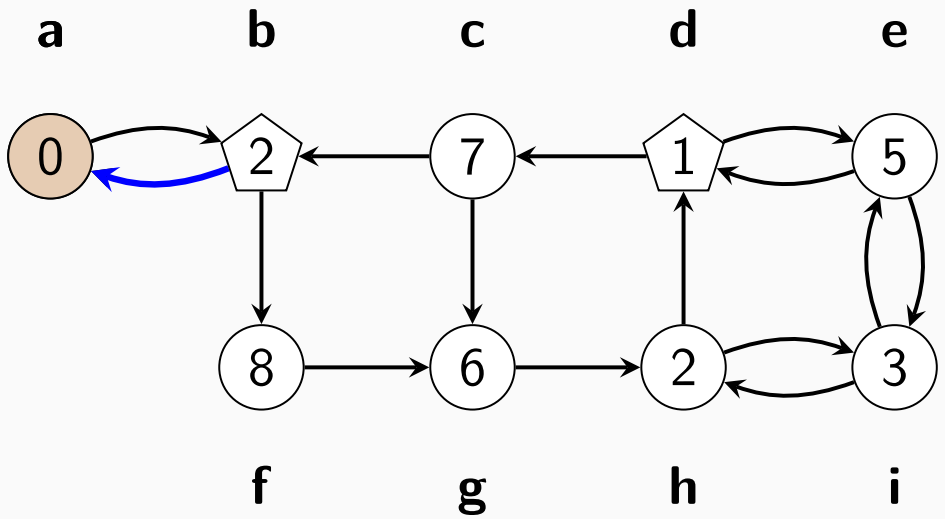

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,1,0,1,1\rangle$ | to | $\langle 0,1,0,1,1\rangle$ | 02020202 |
| **b** | $\langle 0,1,0,1,-\rangle$ | to | $\langle 0,1,0,2,-\rangle$ | 202020202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



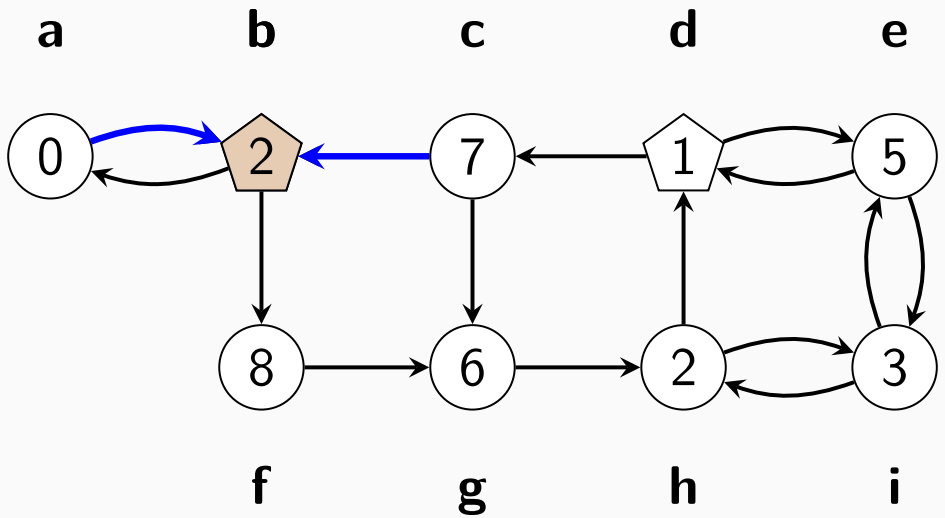

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,1,0,1,1\rangle$ | to | $\langle 0,1,0,2,1\rangle$ | 0202020202 |
| **b** | $\langle 0,1,0,2,-\rangle$ | to | $\langle 0,1,0,2,-\rangle$ | 202020202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



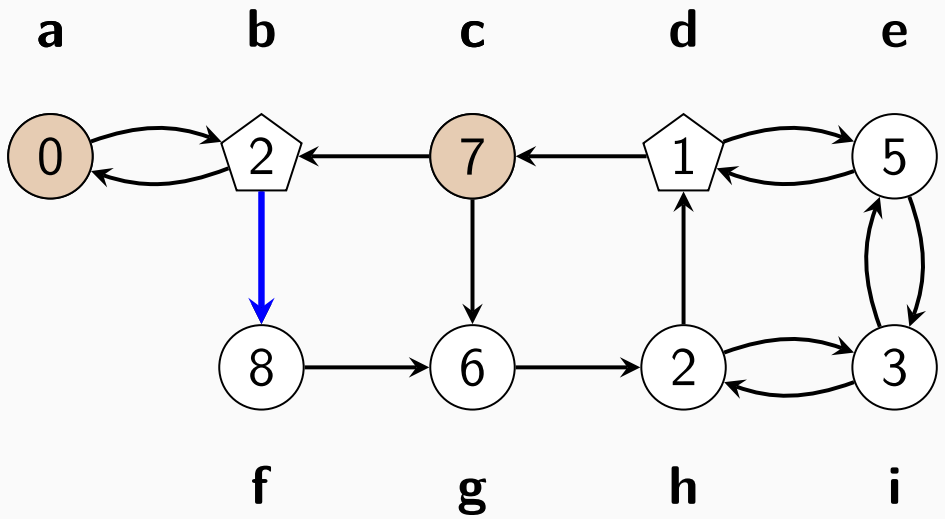

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,1,0,2,1\rangle$ | to | $\langle 0,1,0,2,1\rangle$ | 0202020202 |
| **b** | $\langle 0,1,0,2,-\rangle$ | to | $\langle 1,0,0,0,-\rangle$ | 20202020202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 0,-,-,-,-\rangle$ | 7 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



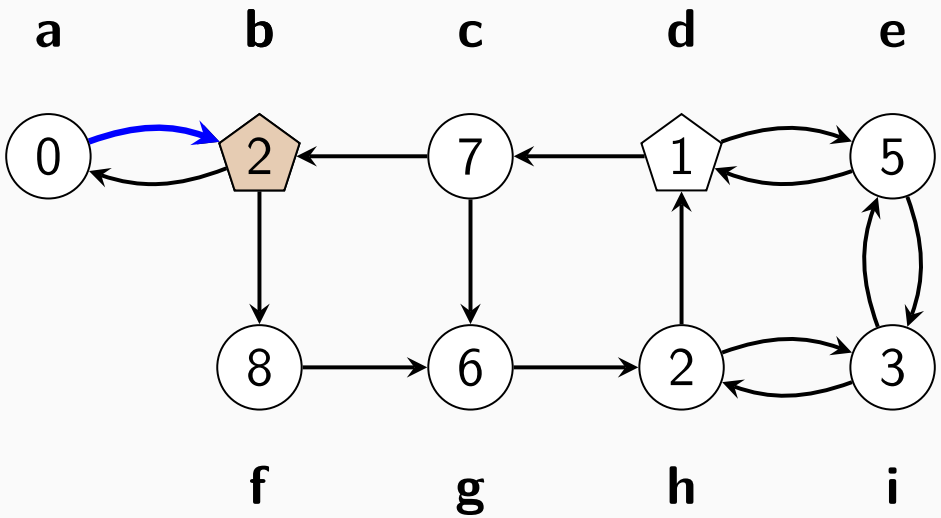

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 0,1,0,2,1\rangle$ | to | $\langle 1,0,0,0,1\rangle$ | 02020202020 |
| **b** | $\langle 1,0,0,0,-\rangle$ | to | $\langle 1,0,0,0,-\rangle$ | 20202020202 |
| **c** | $\langle 0,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 720202020202 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



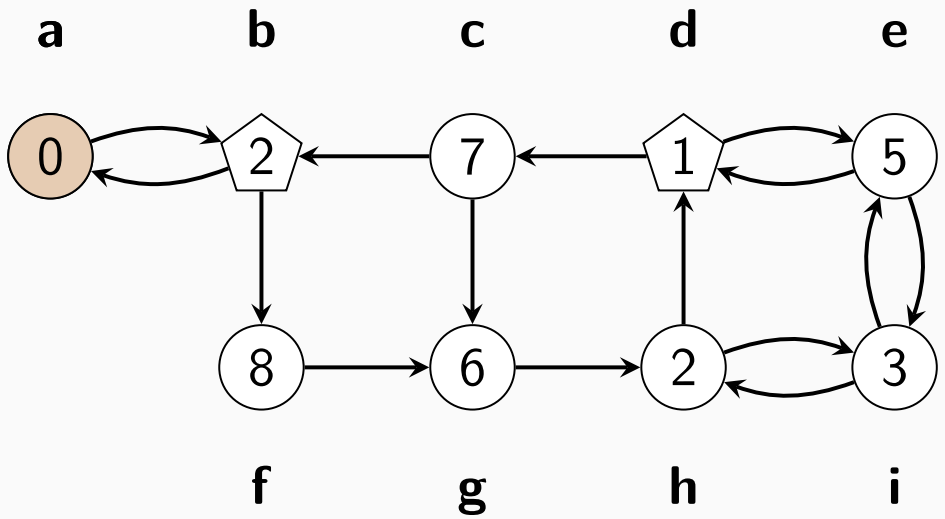

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 1,0,0,0,1\rangle$ | to | $\langle 1,0,0,0,1\rangle$ | 020202020202 |
| **b** | $\langle 1,0,0,0,-\rangle$ | to | $\langle 1,0,0,1,-\rangle$ | 28 |
| **c** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 728 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES



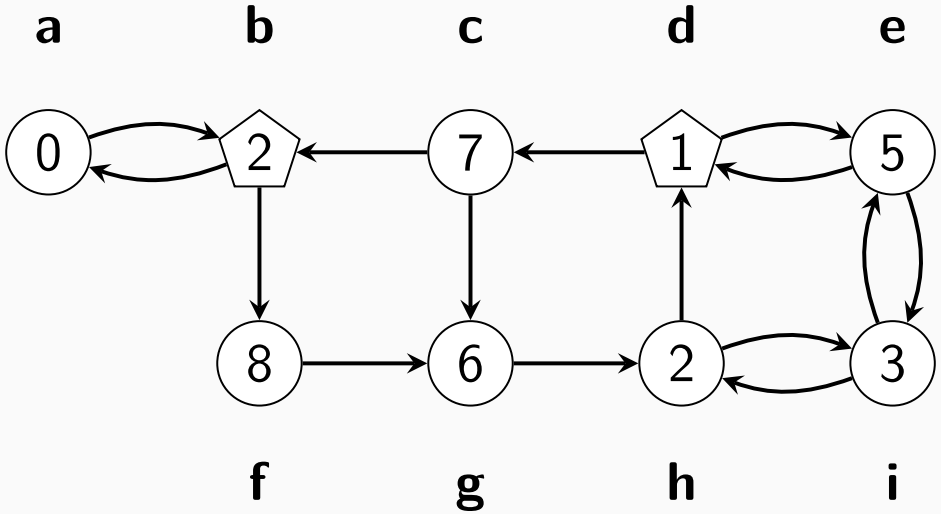

| | | | | |
|---|---|---|---|---|
| **a** | $\langle 1,0,0,0,1\rangle$ | to | $\langle 1,0,0,1,1\rangle$ | 028 |
| **b** | $\langle 1,0,0,1,-\rangle$ | to | $\langle 1,0,0,1,-\rangle$ | 28 |
| **c** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 728 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES




| | | | | |
|---|---|---|---|---|
| **a** | $\langle 1,0,0,1,1\rangle$ | to | $\langle 1,0,0,1,1\rangle$ | 028 |
| **b** | $\langle 1,0,0,1,-\rangle$ | to | $\langle 1,0,0,1,-\rangle$ | 28 |
| **c** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 728 |
| **d** | $\langle 0,0,0,0,-\rangle$ | to | $\langle 0,0,0,0,-\rangle$ | 1 |
| **e** | $\langle 0,0,-,-,-\rangle$ | to | $\langle 0,0,-,-,-\rangle$ | 5 |
| **f** | $\langle 1,-,-,-,-\rangle$ | to | $\langle 1,-,-,-,-\rangle$ | 8 |
| **g** | $\langle 0,1,-,-,-\rangle$ | to | $\langle 0,1,-,-,-\rangle$ | 6 |
| **h** | $\langle 0,0,0,1,-\rangle$ | to | $\langle 0,0,0,1,-\rangle$ | 2 |
| **i** | $\langle 0,0,0,-,-\rangle$ | to | $\langle 0,0,0,-,-\rangle$ | 3 |

# SMALL PROGRESS MEASURES




| | |
|---|---|
| **a** | $\langle 1, 0, 0, 1, 1 \rangle$ |
| **b** | $\langle 1, 0, 0, 1, - \rangle$ |
| **c** | $\langle 1, -, -, -, - \rangle$ |
| **d** | $\langle 0, 0, 0, 0, - \rangle$ |
| **e** | $\langle 0, 0, -, -, - \rangle$ |
| **f** | $\langle 1, -, -, -, - \rangle$ |
| **g** | $\langle 0, 1, -, -, - \rangle$ |
| **h** | $\langle 0, 0, 0, 1, - \rangle$ |
| **i** | $\langle 0, 0, 0, -, - \rangle$ |

- No vertices are won by Even
- Strategy for Odd
  - from **b** to **f**
    (to **a** not $\mu$-progressive!)
  - from **d** to **e**
    (to **c** not $\mu$-progressive!)

# SMALL PROGRESS MEASURES

## Operational interpretation [Gazda, Willemse, 2015]

- $p$-dominated stretches: how often is priority $p$ encountered before a higher priority
- Example: play 0010212023214265 6201 corresponds to $\langle 2, 1, 3, 2 \rangle$
  - priority 0 is seen $2\times$ before a higher priority
  - priority 2 is seen $3\times$ before a higher priority
  - priority 4 is seen $1\times$ before a higher priority
  - priority 6 is seen $2\times$ before a higher priority

## Operational interpretation [Van Dijk, 2018]

- Value represents the "statistic of the best continuation"
- *Notice the "overflow mechanism"!* If priority $p$ is seen more than $n_p$ times, there must be a cycle! If priority $p$ overflows, our optimal path contains a cycle of priority $p$. Keep increasing the measure until the opponent is forced to escape...

# SMALL PROGRESS MEASURES

```
1  def spm(𝒢, α):
2      μ ← V ↦ ⟨0, ..., 0⟩
3      Q ← {v ∈ V | pr(v) mod 2 = ᾱ}          // use a queue or a stack
4      while Q ≠ ∅ :
5          v ← take(Q)                          // take a v ∈ Q
6          if μ(v) < Lift(μ, v) :               // can be lifted?
7              μ[v] ← Lift(μ, v)                // update μ(v)
8              Q ← Q ∪ E^{-1}(v)                // add predecessors to Q
9      W_ᾱ ← {v | μ(v) = ⊤}
10     W_α ← {v | μ(v) ≠ ⊤}
11     σ_α ← (W_α ∩ V_α) ↦ pick({u ∈ E(v) | μ(v) = Prog(μ(u), pr(v))})
12     return W_α, W_ᾱ, σ_α
```

$$\mathrm{Lift}(\mu, v) := \begin{cases} \min_{(v,w)\in E} \mathrm{Prog}(\mu(w), \mathrm{pr}(v)) & v \in V_\alpha \\ \max_{(v,w)\in E} \mathrm{Prog}(\mu(w), \mathrm{pr}(v)) & v \in V_{\overline{\alpha}} \end{cases}$$

$$\mathrm{Prog}(m, p) := \begin{cases} \min\{m' \in M \mid m' \geq_p m\} & p \text{ is } \alpha\text{'s priority} \\ \min\{m' \in M \mid m' >_p m\} & p \text{ is } \overline{\alpha}\text{'s priority} \end{cases}$$

# SMALL PROGRESS MEASURES

## Implementation notes

- use a queue or stack to store "to do" vertices
- after lifting a vertex, add its predecessors to the queue (only once!)
  - a simple bit array can take care of this
- when lifting an $\overline{\alpha}$ **priority vertex** to $\top$, decrease $n_p$ by 1
- the most difficult part is implementing $<$ and `Prog`