Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG
Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems
Institute for Informatics

Advanced Algorithms

Winter term 2019/20

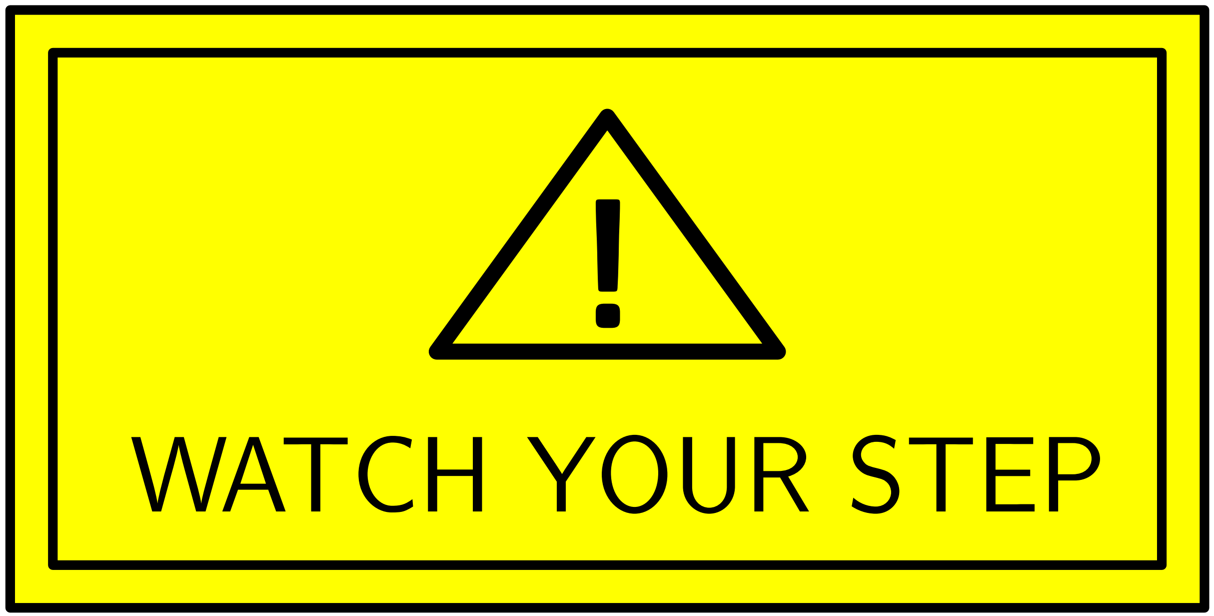
Lecture 6. Approaches using Randomisation

or: Color Coding and “Isolation” Lemmas

Source: **[Parameterized Algorithms: §3.3, 5, 5.1, 5.2, 11.2]**

(slides by Thomas van Dijk & Alexander Wolff)

Steven Chaplick and Alexander Wolff *Chair for Computer Science I*



In this lecture:

- Coloring $\neq$ **Graph**coloring.
- k-coloring of n elements:
  label each element with one number from $1..k$.

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] \leq \frac{1}{2}$

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: YES-instance → $\Pr[\text{YES}] > \frac{1}{2}$

NO-instance → $\Pr[\text{YES}] \leq \frac{1}{2}$

$$\text{SAT} \overset{?}{\in} \mathcal{PP}$$

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] \leq \frac{1}{2}$

$$\text{SAT} \overset{?}{\in} \mathcal{PP}$$

1. Randomly assign binary values to variables.
Return $\textsc{Yes}$ when the formula is satisfied.

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] \leq \frac{1}{2}$

$$\text{SAT} \overset{?}{\in} \mathcal{PP}$$

1. Randomly assign binary values to variables. Return $\textsc{Yes}$ when the formula is satisfied.

2. O.w.: answer $\textsc{Yes}$ or $\textsc{No}$ w/ equal probability

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] \leq \frac{1}{2}$

$$\text{SAT} \overset{?}{\in} \mathcal{PP}$$

1. Randomly assign binary values to variables.
Return $\textsc{Yes}$ when the formula is satisfied.

2. O.w.: answer $\textsc{Yes}$ or $\textsc{No}$ w/ equal probability

Thm: $NP \subseteq \mathcal{PP} \subseteq PSPACE$

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] \leq \frac{1}{2}$

Las Vegas ($\mathcal{ZPP}$), zero-error probabilistic polynomial time

Runtime: expected polynomial

Result: correct

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: YES-instance → $\Pr[\text{YES}] > \frac{1}{2}$

NO-instance → $\Pr[\text{YES}] \leq \frac{1}{2}$

Las Vegas ($\mathcal{ZPP}$), zero-error probabilistic polynomial time

Runtime: expected polynomial

Result: correct

Monte Carlo:

bounded-error prob. poly. time

$\mathcal{BPP}$

Runtime: polynomial

Result: YES-instance → $\Pr[\text{YES}] > \frac{2}{3}$

NO-instance → $\Pr[\text{YES}] \leq \frac{1}{3}$

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: YES-instance → $\Pr[\text{YES}] > \frac{1}{2}$

NO-instance → $\Pr[\text{YES}] \leq \frac{1}{2}$

Las Vegas ($\mathcal{ZPP}$), zero-error probabilistic polynomial time

Runtime: expected polynomial

Result: correct

Monte Carlo:

		$\mathcal{BPP}$ (bounded-error prob. poly. time)	$\mathcal{RP}$ (randomised poly. time)
Runtime:	polynomial		
Result:	YES-instance → $\Pr[\text{YES}] >$	$\frac{2}{3}$	$\frac{1}{2}$
	NO-instance → $\Pr[\text{YES}] \leq$	$\frac{1}{3}$	0

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: YES-instance → $\Pr[\text{YES}] > \frac{1}{2}$

NO-instance → $\Pr[\text{YES}] \leq \frac{1}{2}$

Las Vegas ($\mathcal{ZPP}$), zero-error probabilistic polynomial time

Runtime: expected polynomial

Result: correct

Monte Carlo:

bounded-error prob. poly. time

randomised poly. time

		$\mathcal{BPP}$	$\mathcal{RP}$	co-$\mathcal{RP}$
Runtime:	polynomial			
Result:	YES-instance → $\Pr[\text{YES}] >$	$\frac{2}{3}$	$\frac{1}{2}$	1
	NO-instance → $\Pr[\text{YES}] \leq$	$\frac{1}{3}$	0	$\frac{1}{2}$

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow$ $\Pr[\textsc{Yes}] \leq \frac{1}{2}$

Las Vegas ($\mathcal{ZPP}$), zero-error probabilistic polynomial time

Runtime: expected polynomial

Result: correct

Monte Carlo:

Runtime: polynomial

Result:		$\mathcal{BPP}$	$\mathcal{RP}$	co-$\mathcal{RP}$
$\textsc{Yes}$-instance $\rightarrow$	$\Pr[\textsc{Yes}] >$	$\frac{2}{3}$	$\frac{1}{2}$	1
$\textsc{No}$-instance $\rightarrow$	$\Pr[\textsc{Yes}] \leq$	$\frac{1}{3}$	0	$\frac{1}{2}$

bounded-error prob. poly. time

randomised poly. time

Thm: $\mathcal{ZPP} = \mathcal{RP} \cap$ co-$\mathcal{RP}$

Randomised Algorithms

Probabilistic Polynomial Time ($\mathcal{PP}$)

Runtime: polynomial

Result: $\textsc{Yes}$-instance $\rightarrow \Pr[\textsc{Yes}] > \frac{1}{2}$

$\textsc{No}$-instance $\rightarrow \Pr[\textsc{Yes}] \leq \frac{1}{2}$

Las Vegas ($\mathcal{ZPP}$), zero-error probabilistic polynomial time

Runtime: expected polynomial

Result: correct

Monte Carlo:

		bounded-error prob. poly. time	randomised poly. time	
		$\mathcal{BPP}$	$\mathcal{RP}$	co-$\mathcal{RP}$
Runtime:	polynomial			
Result:	$\textsc{Yes}$-instance $\rightarrow \Pr[\textsc{Yes}] >$	$\frac{2}{3}$	$\frac{1}{2}$	1
	$\textsc{No}$-instance $\rightarrow \Pr[\textsc{Yes}] \leq$	$\frac{1}{3}$	0	$\frac{1}{2}$

Thm: $\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}$

Amplification

$$\mathcal{RP}: \quad \begin{array}{l} \textsc{Yes}\text{-Instance} \rightarrow \Pr[\textsc{Yes}] \geq t \\ \textsc{No}\text{-Instance} \rightarrow \Pr[\textsc{Yes}] = 0 \end{array}$$

Amplification

$$\mathcal{RP}: \quad \begin{array}{l} \text{Yes-Instance} \rightarrow \Pr[\text{Yes}] \geq t \\ \text{No-Instance} \rightarrow \Pr[\text{Yes}] = 0 \end{array}$$

If an $\mathcal{RP}$-algorithm returns Yes, it is correct

Amplification

$\mathcal{RP}$: $\quad$ $\textsc{Yes}$-Instance $\rightarrow$ $\Pr[\textsc{Yes}] \geq t$
$\quad\quad\quad$ $\textsc{No}$-Instance $\rightarrow$ $\Pr[\textsc{Yes}] = 0$

If an $\mathcal{RP}$-algorithm returns $\textsc{Yes}$, it is correct

If an $\mathcal{RP}$-algorithm returns $\textsc{No}$,
it is incorrect with probability $\leq 1 - t$

Amplification

$\mathcal{RP}$: $\quad$ $\textsc{Yes}$-Instance $\rightarrow$ $\Pr[\textsc{Yes}] \geq t$
$\quad\quad\quad$ $\textsc{No}$-Instance $\rightarrow$ $\Pr[\textsc{Yes}] = 0$

If an $\mathcal{RP}$-algorithm returns $\textsc{Yes}$, it is correct

If an $\mathcal{RP}$-algorithm returns $\textsc{No}$,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return $\textsc{Yes}$ if every some returns $\textsc{Yes}$
Otherwise $\textsc{No}$

Amplification

$\mathcal{RP}$: $\textsc{Yes}$-Instance $\rightarrow$ Pr[$\textsc{Yes}$] $\geq t$
$\textsc{No}$-Instance $\rightarrow$ Pr[$\textsc{Yes}$] $= 0$

If an $\mathcal{RP}$-algorithm returns $\textsc{Yes}$, it is correct

If an $\mathcal{RP}$-algorithm returns $\textsc{No}$,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return $\textsc{Yes}$ if every some returns $\textsc{Yes}$
Otherwise $\textsc{No}$

Error Probability :

Amplification

$\mathcal{RP}$: Yes-Instance $\rightarrow$ Pr[Yes] $\geq t$
No-Instance $\rightarrow$ Pr[Yes] $= 0$

If an $\mathcal{RP}$-algorithm returns Yes, it is correct

If an $\mathcal{RP}$-algorithm returns No,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return Yes if every some returns Yes
Otherwise No

Error Probability : $(1 - t)^{1/t}$

Amplification

$\mathcal{RP}$: $\quad$ $\textsc{Yes}$-Instance $\rightarrow \Pr[\textsc{Yes}] \geq t$
$\quad\quad$ $\textsc{No}$-Instance $\rightarrow \Pr[\textsc{Yes}] = 0$

If an $\mathcal{RP}$-algorithm returns $\textsc{Yes}$, it is correct

If an $\mathcal{RP}$-algorithm returns $\textsc{No}$,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return $\textsc{Yes}$ if every some returns $\textsc{Yes}$
Otherwise $\textsc{No}$ $\quad\quad$ $x := 1/t$

Error Probability : $\quad (1-t)^{1/t} = (1 - \frac{1}{x})^x$

Amplification

$\mathcal{RP}$: $\textsc{Yes}$-Instance $\rightarrow \Pr[\textsc{Yes}] \geq t$
$\textsc{No}$-Instance $\rightarrow \Pr[\textsc{Yes}] = 0$

If an $\mathcal{RP}$-algorithm returns $\textsc{Yes}$, it is correct

If an $\mathcal{RP}$-algorithm returns $\textsc{No}$,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return $\textsc{Yes}$ if every some returns $\textsc{Yes}$
Otherwise $\textsc{No}$ $\quad x := 1/t$

Error Probability : $(1-t)^{1/t} = (1 - \frac{1}{x})^x < \frac{1}{e}$

Amplification

$\mathcal{RP}$: YES-Instance $\rightarrow$ $\Pr[\text{YES}] \geq t$
NO-Instance $\rightarrow$ $\Pr[\text{YES}] = 0$

If an $\mathcal{RP}$-algorithm returns YES, it is correct

If an $\mathcal{RP}$-algorithm returns NO,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return YES if every some returns YES
Otherwise NO $\quad x := 1/t$

Error Probability : $(1-t)^{1/t} = (1 - \frac{1}{x})^x < \frac{1}{e} < \frac{1}{2}$

Amplification

$\mathcal{RP}$: $\quad$ YES-Instance $\to$ Pr[YES] $\geq t$
NO-Instance $\to$ Pr[YES] $= 0$

If an $\mathcal{RP}$-algorithm returns YES, it is correct

If an $\mathcal{RP}$-algorithm returns NO,
it is incorrect with probability $\leq 1 - t$

Algorithm: Run the original algorithm $\lceil 1/t \rceil$ times
Return YES if every some returns YES
Otherwise NO $\qquad x := 1/t$

Error Probability : $(1 - t)^{1/t} = (1 - \frac{1}{x})^x < \frac{1}{e} < \frac{1}{2}$

(**Obs.:** Repeating $100 \cdot t^{-1}$ times $\rightsquigarrow$ error prob. $< 2^{-100}$.)

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\ S \subseteq V$ such that $\lvert S \rvert \leq k$ and $G[V \setminus S]$ is a forest?

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\, S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

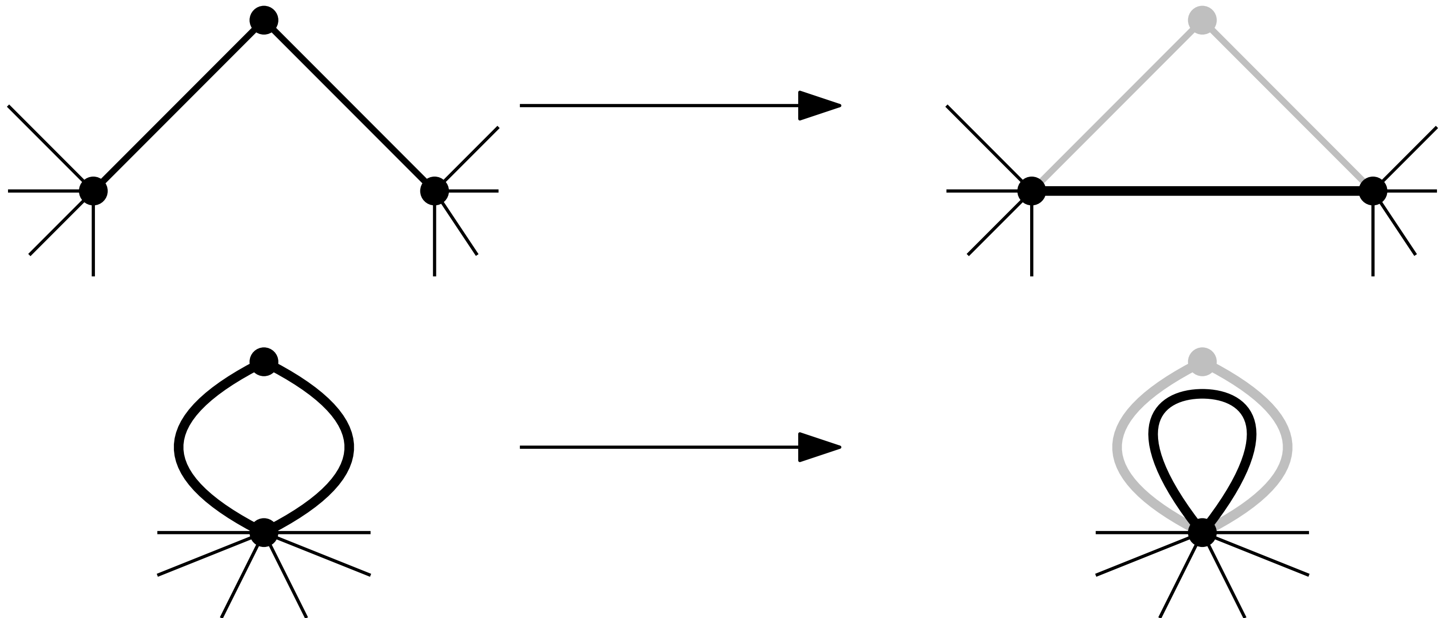
Reduction Rule: Delete vertices of degree < 2

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\, S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Reduction Rule: Delete vertices of degree < 2

Reduction Rule: "Bypass" each degree two vertex.



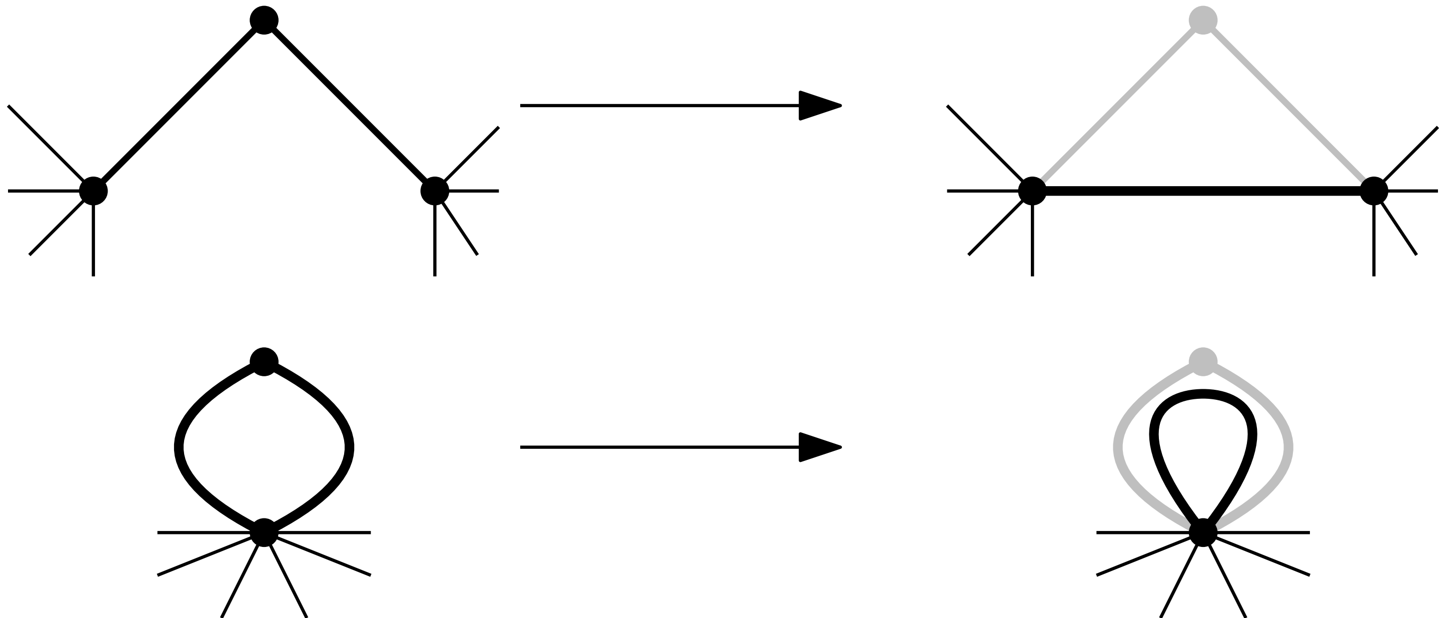
Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Reduction Rule: Delete vertices of degree < 2

Reduction Rule: "Bypass" each degree two vertex.

Reduction Rule: Put vertices incident to loops in FVS



Feedback Vertex Set

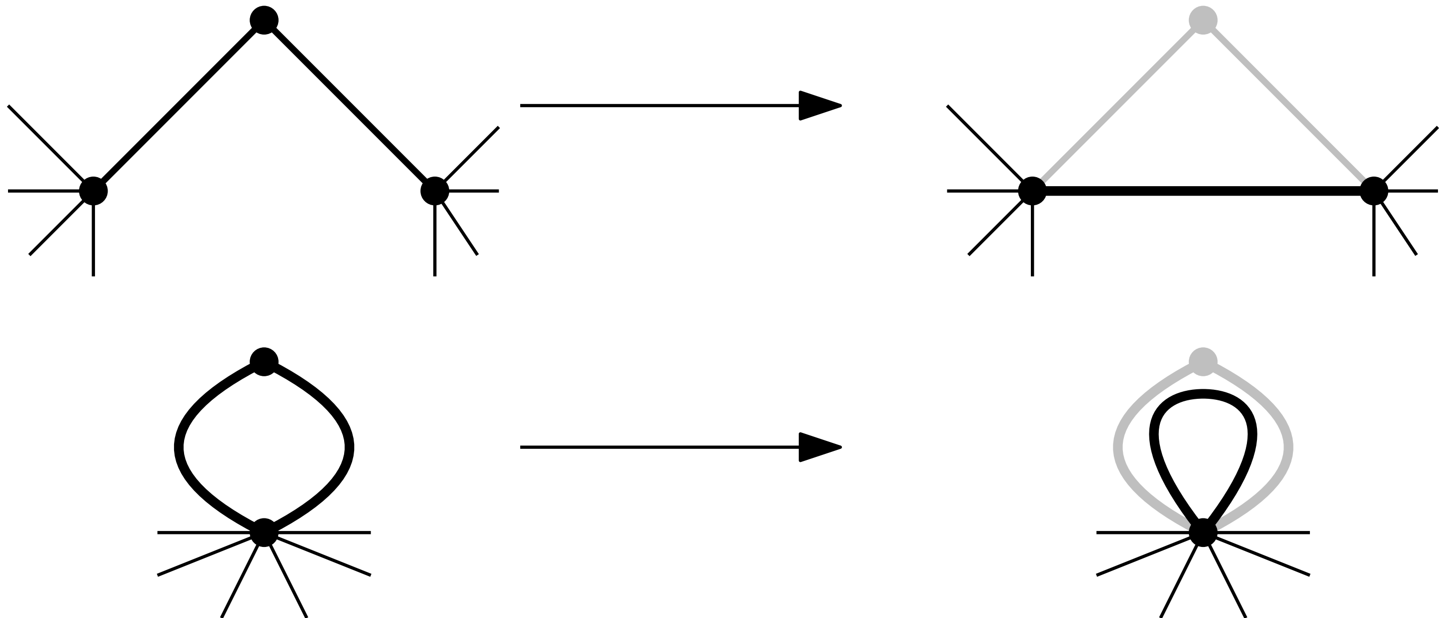
Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Reduction Rule: Delete vertices of degree < 2

Reduction Rule: "Bypass" each degree two vertex.

Reduction Rule: Put vertices incident to loops in FVS

Def.: If no rule applies, the graph is called *reduced*.



Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\, S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\ S \subseteq V$ such that $\lvert S \rvert \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

$$\leq \sum_{v \in W} \deg(v)$$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\, S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

$$3|W| \leq \sum_{v \in W} \deg(v)$$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\, S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

mindeg 3

$$3|W| \leq \sum_{v \in W} \deg(v)$$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\, S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

mindeg 3 ↓

$$3|W| \leq \sum_{v \in W} \deg(v) = |E_{S,W}| + 2|E_W|$$

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\, S \subseteq V$ such that $\lvert S\rvert \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

mindeg 3

$$3|W| \leq \sum_{v \in W} \deg(v) = |E_{S,W}| + 2|E_W|$$

$|E_W| < |W|$ since forest

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\ S \subseteq V$ such that $\lvert S\rvert \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

mindeg 3

$< |W|$ since forest

$$3|W| \leq \sum_{v \in W} \deg(v) = |E_{S,W}| + 2|E_W| < |E_{S,W}| + 2|W|$$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.
Let $E_W :=$ edges connecting vertices in W
Let $E_{S,W} :=$ edges connecting S and W

mindeg 3

$< |W|$ since forest

?

$$3|W| \leq \sum_{v \in W} \deg(v) = |E_{S,W}| + 2|E_W| < |E_{S,W}| + 2|W|$$

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\ S \subseteq V$ such that $\lvert S\rvert \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Let $E_W :=$ edges connecting vertices in W

Let $E_{S,W} :=$ edges connecting S and W

mindeg 3

$< \lvert W\rvert$ since forest

$$3\lvert W\rvert \leq \sum_{v \in W} \deg(v) = \lvert E_{S,W}\rvert + 2\lvert E_W\rvert < \lvert E_{S,W}\rvert + 2\lvert W\rvert$$

Lemma: If G is reduced, then $\lvert E_{S,W}\rvert \geq \lvert E_W\rvert$

(see also Lemma 5.1 in textbook)

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\ S \subseteq V$ such that $\lvert S\rvert \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $\lvert E_{S,W}\rvert \geq \lvert E_W\rvert$

Feedback Vertex Set

Given:	Graph $G = (V, E)$, number k
Question:	$\exists\ S \subseteq V$ such that $\lvert S \rvert \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $\lvert E_{S,W} \rvert \geq \lvert E_W \rvert$

Idea: Find some $v \in S$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $|E_{S,W}| \geq |E_W|$

Idea: Find some $v \in S$

Algorithm:

1. pick each $e \in E$ with equal prob.

2. pick $v \in e$ with equal prob.

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $|E_{S,W}| \geq |E_W|$

Idea: Find some $v \in S$

Algorithm:

1. pick each $e \in E$ with equal prob.

2. pick $v \in e$ with equal prob.

Success probability: at least

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k

Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $|E_{S,W}| \geq |E_W|$

E_S**?**

Idea: Find some $v \in S$

Algorithm:

1. pick each $e \in E$ with equal prob.

2. pick $v \in e$ with equal prob.

Success probability: at least

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $|E_{S,W}| \geq |E_W|$

E_S**?**

Idea: Find some $v \in S$

Algorithm:

1. pick each $e \in E$ with equal prob. $> 1/2$

2. pick $v \in e$ with equal prob.

Success probability: at least $1/2$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $|E_{S,W}| \geq |E_W|$

E_S**?**

Idea: Find some $v \in S$

Algorithm:

1. pick each $e \in E$ with equal prob.	$> 1/2$
2. pick $v \in e$ with equal prob.	$> 1/2$

Success probability: at least $1/2 \cdot 1/2 = 1/4$

Feedback Vertex Set

Given: Graph $G = (V, E)$, number k
Question: $\exists\ S \subseteq V$ such that $|S| \leq k$ and $G[V \setminus S]$ is a forest?

Def.: Let G be reduced, $S \subseteq V$ be an FVS, and $W := V \setminus S$.

Lemma: If G is reduced, then $|E_{S,W}| \geq |E_W|$

E_S**?**

Idea: Find some $v \in S$

Algorithm:

1. pick each $e \in E$ with equal prob.	$> 1/2$
2. pick $v \in e$ with equal prob.	$> 1/2$

Success probability: at least $1/2 \cdot 1/2 = 1/4$

Obs.: With prob. $\geq 1/4$, we find a node from an (unknown) optimal FVS

FVS: algorithm given k

1. `while` G is not empty:

FVS: algorithm given k

1. `while` G is not empty:

  2. Apply reduction rules

FVS: algorithm given k

1. `while` G is not empty:

  2. Apply reduction rules

  3. pick a vertex v via randomized proc. on last slide

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

  2. Apply reduction rules

  3. pick a vertex v via randomized proc. on last slide

  4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

2. Apply reduction rules

3. pick a vertex v via randomized proc. on last slide

4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

5. If $|S| > k$: return $\textsc{No}$

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

2. Apply reduction rules

3. pick a vertex v via randomized proc. on last slide

4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

5. If $|S| > k$: return No

6. Return Yes

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

> **2.** Apply reduction rules
>
> **3.** pick a vertex v via randomized proc. on last slide
>
> **4.** $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$
>
> **5.** If $|S| > k$: return NO

6. Return YES

Runtime:

Prob. of success:

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

> **2.** Apply reduction rules
>
> **3.** pick a vertex v via randomized proc. on last slide
>
> **4.** $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$
>
> **5.** If $|S| > k$: return No

6. Return Yes

Runtime:	$O(n+m)$
Prob. of success:	

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

> **2.** Apply reduction rules
>
> **3.** pick a vertex v via randomized proc. on last slide
>
> **4.** $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$
>
> **5.** If $|S| > k$: return No

6. Return Yes

Runtime:	$O(n+m)$
Prob. of success:	$> 1/4$

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty:

2. Apply reduction rules

3. pick a vertex v via randomized proc. on last slide

4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

5. If $|S| > k$: return No

6. Return Yes

Runtime: $O(n + m)$

Prob. of success: $> 1/4$

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty: $\leftarrow$ max k times

2. Apply reduction rules

3. pick a vertex v via randomized proc. on last slide

4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

5. If $|S| > k$: return No

6. Return Yes

Runtime:	$O(n+m)$	$O(k(n+m))$
Prob. of success:	$> 1/4$	

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty: $\leftarrow$ max k times

2. Apply reduction rules

3. pick a vertex v via randomized proc. on last slide

4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

5. If $|S| > k$: return No

6. Return Yes

Runtime:	$O(n+m)$	$O(k(n+m))$
Prob. of success:	$> 1/4$	$> 4^{-k}$

FVS: algorithm given k

0. $S \leftarrow \varnothing$

1. `while` G is not empty: $\leftarrow$ max k times

2. Apply reduction rules

3. pick a vertex v via randomized proc. on last slide

4. $S \leftarrow S \cup \{v\}$; $G \leftarrow G \setminus v$

5. If $|S| > k$: return No

6. Return Yes

Runtime:	$O(n+m)$	$O(k(n+m))$
Prob. of success:	$> 1/4$	$> 4^{-k}$

Thm: Feedback Vertex Set can be solved in $O(4^k \cdot k(n+m))$ time by a randomised algorithm

Longest Path

Given:	Graph $G = (V, E)$, number k
Question:	Does G contain a length k path?

(length := # edges)

Longest Path

Given:	Graph $G = (V, E)$, number k
Question:	Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

Longest Path

Given:	Graph $G = (V, E)$, number k
Question:	Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

↓

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Special Case:
Longest Path in acyclic graphs: Runtime?

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

↓

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Special Case:
Longest Path in acyclic graphs: Runtime? $O(m)$

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

↓

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Special Case:
Longest Path in acyclic graphs: Runtime? $O(m)$

1. Topological sort

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

↓

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Special Case:
Longest Path in acyclic graphs: Runtime? $O(m)$

1. Topological sort
2. Let $L(v)$:= longest path to v

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

↓

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Special Case:
Longest Path in acyclic graphs: Runtime? $O(m)$

1. Topological sort
2. Let $L(v)$:= longest path to v
3. "backwards" dynamic program

Longest Path

Given: Graph $G = (V, E)$, number k
Parameter: k
Question: Does G contain a length k path?

(length := # edges)

Thm: Longest Path is NP-complete

Thm: Longest Path can be solved in $O^*(2^n)$ time.

Special Case:
Longest Path in acyclic graphs: Runtime? $O(m)$

1. Topological sort
2. Let $L(v)$:= longest path to v
3. "backwards" dynamic program
4. Look for v with $L(v) = k$

Longest Path

Idea. Longest Path is easy on acyclic graphs

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Obs.: $\exists$ k-path in $\vec{G}$ $\rightarrow$ $\exists$ k-path in G

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Obs.: $\exists$ k-path in $\vec{G} \rightarrow \exists$ k-path in G

Obs.: Converse does not apply however ...

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Obs.: $\exists$ k-path in $\vec{G} \rightarrow \exists$ k-path in G

Obs.: Converse does not apply however ...

$\exists$ k-path in $G \rightarrow \Pr[\vec{G}$ has k-path$] > 0$.

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Obs.: $\exists$ k-path in $\vec{G} \rightarrow \exists$ k-path in G

Obs.: Converse does not apply however ...

$\exists$ k-path in $G \rightarrow \Pr[\vec{G}$ has k-path$] > 0$.

Now: Randomisied algorithm?

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

> **1.** pick random permutation π of V
>
> **2.** orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Obs.: $\exists$ k-path in $\vec{G} \rightarrow \exists$ k-path in G

Obs.: Converse does not apply however ...
$\exists$ k-path in $G \rightarrow \Pr[\vec{G}$ has k-path$] > 0.$

Now: Randomisied algorithm? **Runtime?**

Longest Path

Idea. Longest Path is easy on acyclic graphs

Plan: make G acyclic!

1. pick random permutation π of V
2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Obs.: $\exists$ k-path in $\vec{G} \rightarrow \exists$ k-path in G

Obs.: Converse does not apply however ...

$\exists$ k-path in $G \rightarrow \Pr[\vec{G}$ has k-path$] > 0$.

Now: Randomisied algorithm? **Runtime?**

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Obs.: Order of vertices $\notin p$ is irrelevant

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

π: 1 4 2 6 8 5 3 7

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

π: 1 4 2 6 8 5 3 7

p: $(5, 3, 6)$

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

π:	1 4 2 6 8 5 3 7
p:	$(5, 3, 6)$
$\pi_{/p}$:	1 4 2 $\times$ 8 $\times$ $\times$ 7

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

All have equal probability.

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

All have equal probability.

For two of them, p is a path in $\vec{G}$ (two correct)

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

All have equal probability.

For two of them, p is a path in $\vec{G}$ (two correct)

Thus $\Pr[p \in \vec{G} \mid \pi_{/p}] = \frac{2}{(k+1)!}$

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

All have equal probability.

For two of them, p is a path in $\vec{G}$ (two correct)

Thus $\Pr[p \in \vec{G} \mid \pi_{/p}] = \frac{2}{(k+1)!}$

$\rightsquigarrow \Pr[p \in \vec{G}] =$

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] =$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

All have equal probability.

For two of them, p is a path in $\vec{G}$ (two correct)

Thus $\Pr[p \in \vec{G} \mid \pi_{/p}] = \frac{2}{(k+1)!}$

$\rightsquigarrow \Pr[p \in \vec{G}] = \frac{2}{(k+1)!}$ (indep. sum over $\pi_{/p}$)

Randomised Orientation: Success Prob.

1. pick random permutation π of V

2. orient edges $\{u, v\}$ from u to v when $\pi(u) < \pi(v)$

Result → $\vec{G}$ (random variable!)

Lemma: Let p be a k-path in G. Then $\Pr[p \in \vec{G}] = \frac{2}{(k+1)!}$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

All have equal probability.

For two of them, p is a path in $\vec{G}$ (two correct)

Thus $\Pr[p \in \vec{G} \mid \pi_{/p}] = \frac{2}{(k+1)!}$

$\rightsquigarrow \Pr[p \in \vec{G}] = \frac{2}{(k+1)!}$ (indep. sum over $\pi_{/p}$) $\square$

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

  2. $\vec{G} \leftarrow$ random acyclic orientation of G

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

   2. $\vec{G} \leftarrow$ random acyclic orientation of G

   3. $p \leftarrow$ longest path in $\vec{G}$

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

2. $\vec{G} \leftarrow$ random acyclic orientation of G

3. $p \leftarrow$ longest path in $\vec{G}$

4. If $|p| \geq k$, return $\textsc{Yes}$.

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

2. $\vec{G} \leftarrow$ random acyclic orientation of G

3. $p \leftarrow$ longest path in $\vec{G}$

4. If $|p| \geq k$, return Yes.

5. Return No

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

- **2.** $\vec{G} \leftarrow$ random acyclic orientation of G
- **3.** $p \leftarrow$ longest path in $\vec{G}$
- **4.** If $|p| \geq k$, return $\textsc{Yes}$.

5. Return $\textsc{No}$

Runtime: $O^*(k!)$ iterations

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

  2. $\vec{G} \leftarrow$ random acyclic orientation of G

  3. $p \leftarrow$ longest path in $\vec{G}$

  4. If $|p| \geq k$, return $\textsc{Yes}$.

5. Return $\textsc{No}$

Runtime: $O^*(k!)$ iterations   each $O(m)$ time

Randomised Orientation: Algorithm

Algorithm

1. Repeat $(k+1)!/2$ times:

  2. $\vec{G} \leftarrow$ random acyclic orientation of G

  3. $p \leftarrow$ longest path in $\vec{G}$

  4. If $|p| \geq k$, return Yes.

5. Return No

Runtime: $O^*(k!)$ iterations each $O(m)$ time

Thm: A randomised algorithm can solve Longest Path in $O^*(k! \cdot n)$ time

LONGEST PATH : attempt 2

Obs. LONGEST PATH is easy on acyclic graphs.

Longest Path : attempt 2

Obs. Longest Path is easy on acyclic graphs.

Color vertices with $(k+1)$ colors

$(k$-path has $k+1$ vertices$)$

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

LONGEST PATH : attempt 2

Obs. LONGEST PATH is easy on acyclic graphs.

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

Color vertices with $(k + 1)$ colors

(k-path has $k + 1$ vertices)

Def.: A path is *colorful*, when each vertex has a different color.

LONGEST PATH : attempt 2

Obs. LONGEST PATH is easy on acyclic graphs.

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

Color vertices with $(k+1)$ colors

(k-path has $k+1$ vertices)

Def.: A path is *colorful*, when each vertex has a different color.

Obs.: Colorful paths are "easy"

LONGEST PATH : attempt 2

Obs. LONGEST PATH is easy on acyclic graphs.

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

Color vertices with $(k+1)$ colors

(k-path has $k+1$ vertices)

Def.: A path is *colorful*, when each vertex has a different color.

Obs.: Colorful paths are "easy"

Part 1: Finding a colorful path is easy

LONGEST PATH : attempt 2

Obs. LONGEST PATH is easy on acyclic graphs.

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

Color vertices with $(k+1)$ colors
(k-path has $k+1$ vertices)

Def.: A path is *colorful*, when each vertex has a different color.

Obs.: Colorful paths are "easy"

Part 1: Finding a colorful path is ~~easy~~

Longest Path : attempt 2

Obs. Longest Path is easy on acyclic graphs.

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

Color vertices with $(k+1)$ colors
(k-path has $k+1$ vertices)

Def.: A path is *colorful*, when each vertex has a different color.

Obs.: Colorful paths are “easy”

Part 1: Finding a colorful path is ~~easy~~ FPT in k.

Longest Path : attempt 2

Obs. Longest Path is easy on acyclic graphs.

Color vertices with $(k+1)$ colors

$\neq$ **Graph**coloring!

vertices with equal color might not be adjacent

(k-path has $k+1$ vertices)

Def.: A path is *colorful*, when each vertex has a different color.

Obs.: Colorful paths are "easy"

Part 1: Finding a colorful path is ~~easy~~ FPT in k.

Part 2: $\exists$ k-path in G $\rightarrow$ good prob. of a colorful path

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof:

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Consider perm. π, but ignore the elements of $p \rightsquigarrow \pi_{/p}$

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

There are $(k+1)!$ ways to complete $\pi_{/p}$ to some π'.

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

For two of them, p is a path in $\vec{G}$

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

Of these, $k!$ are colorful

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then Pr[p is coloful] $>$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

Of these, $k!$ are colorful

Thus Pr[p is colorful] $=$

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

Of these, $k!$ are colorful

Thus $\Pr[p \text{ is colorful}] = \frac{k!}{k^k}$

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] >$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

Of these, $k!$ are colorful

Thus $\Pr[p \text{ is colorful}] = \frac{k!}{k^k} > \left(\frac{k}{e}\right)^k / k^k$

Stirling:
$k! \geq \sqrt{2\pi} k^{k+\frac{1}{2}} e^{-k}$

Random Coloring: Success Prob.

Lemma: Let c be a random k-coloring of V, and p be a $(k-1)$-path.

Then $\Pr[p \text{ is coloful}] > e^{-k}$

Proof: Fix the colors of the nodes outside of p

We get k^k different colorings of p

Each with equal probability

Of these, $k!$ are colorful

Thus $\Pr[p \text{ is colorful}] = \frac{k!}{k^k} > \left(\frac{k}{e}\right)^k / k^k = e^{-k}$

□

Stirling:
$k! \geq \sqrt{2\pi} k^{k+\frac{1}{2}} e^{-k}$

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Table entries:

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Table entries:

For a subset S of our colors, and vertex u:
Path(S, u) = true if and only if there is an S-colorful path ending at u

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Table entries:

For a subset S of our colors, and vertex u:
$\textbf{Path}(S, u) =$ true if and only if there is an S-colorful path ending at u

Recurrence:

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Table entries:

For a subset S of our colors, and vertex u:
$\mathbf{Path}(S, u) =$ true if and only if there is an S-colorful path ending at u

Recurrence:

$$\mathbf{Path}(S, u) =$$

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Table entries:

For a subset S of our colors, and vertex u:
Path$(S, u) =$ true if and only if there is an S-colorful path ending at u

Recurrence:

$$\textbf{Path}(S, u) = \begin{cases} \bigvee_{uv \in E(G)} \textbf{Path}(S \setminus c(u), v), & \text{if } c(u) \in S \\ \text{false}, & \text{otherwise} \end{cases}$$

Finding Colorful Paths

Approach 1: dynamic program

Given c colored graph G

Table entries:

For a subset S of our colors, and vertex u:
Path(S, u) = true if and only if there is an S-colorful path ending at u

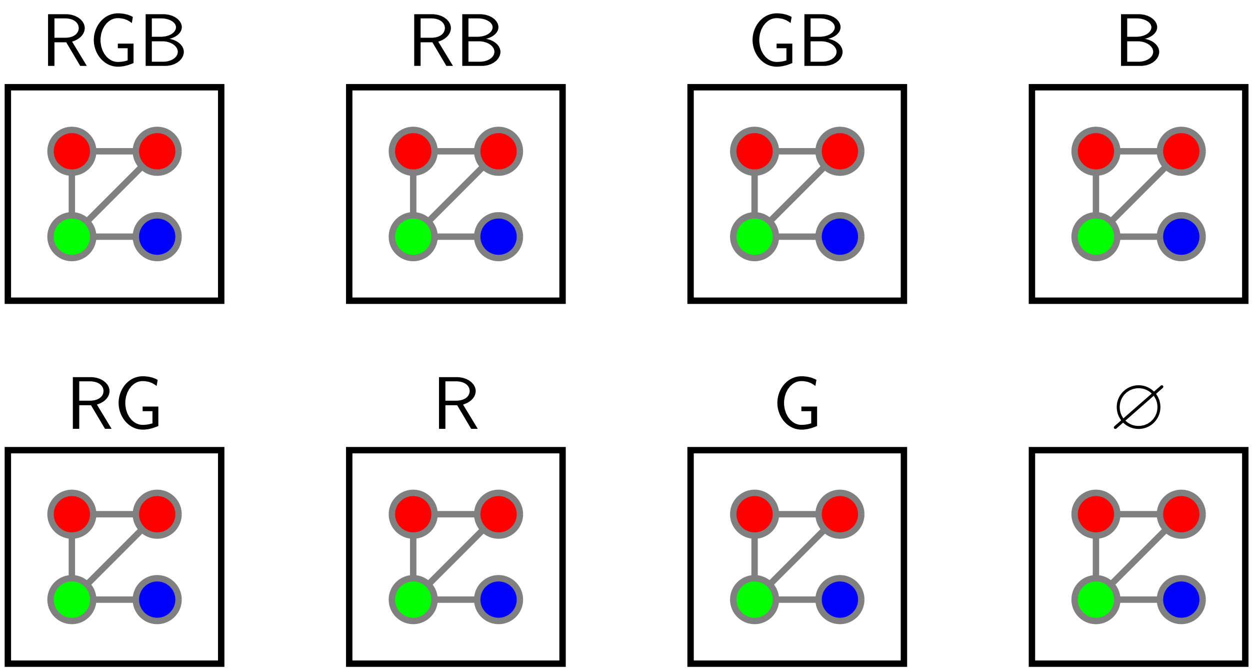
Recurrence:

$$\textbf{Path}(S, u) = \begin{cases} \bigvee_{uv \in E(G)} \textbf{Path}(S \setminus c(u), v), & \text{if } c(u) \in S \\ \text{false}, & \text{otherwise} \end{cases}$$

Runtime?

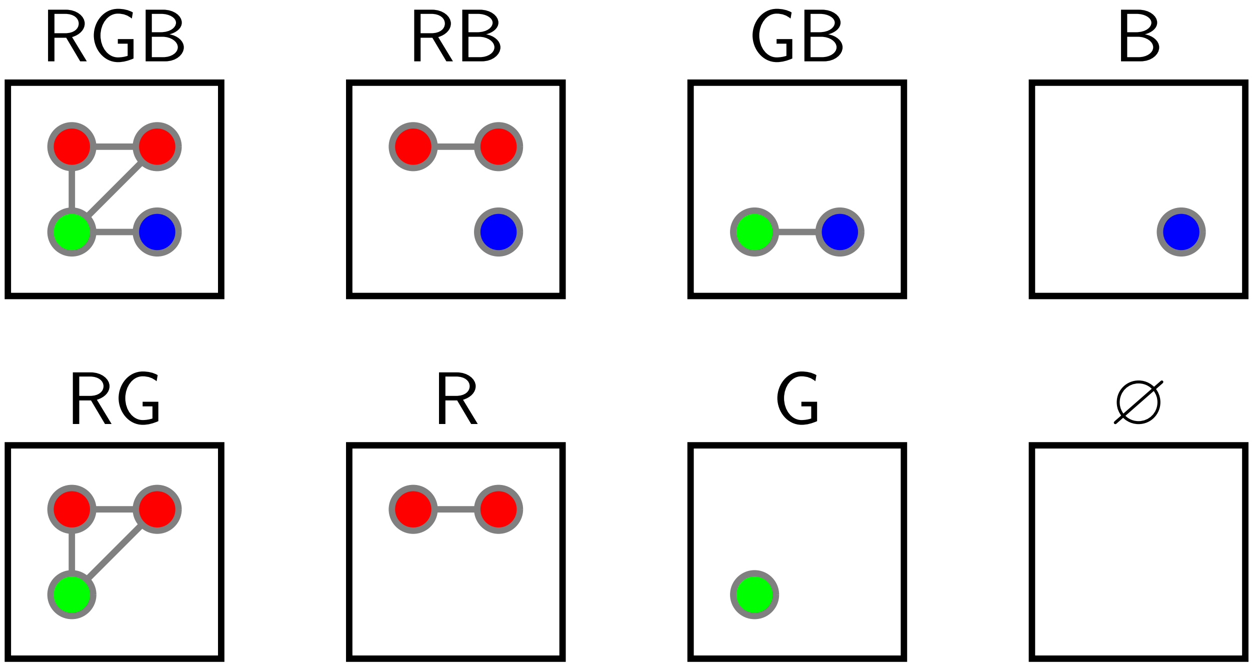
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



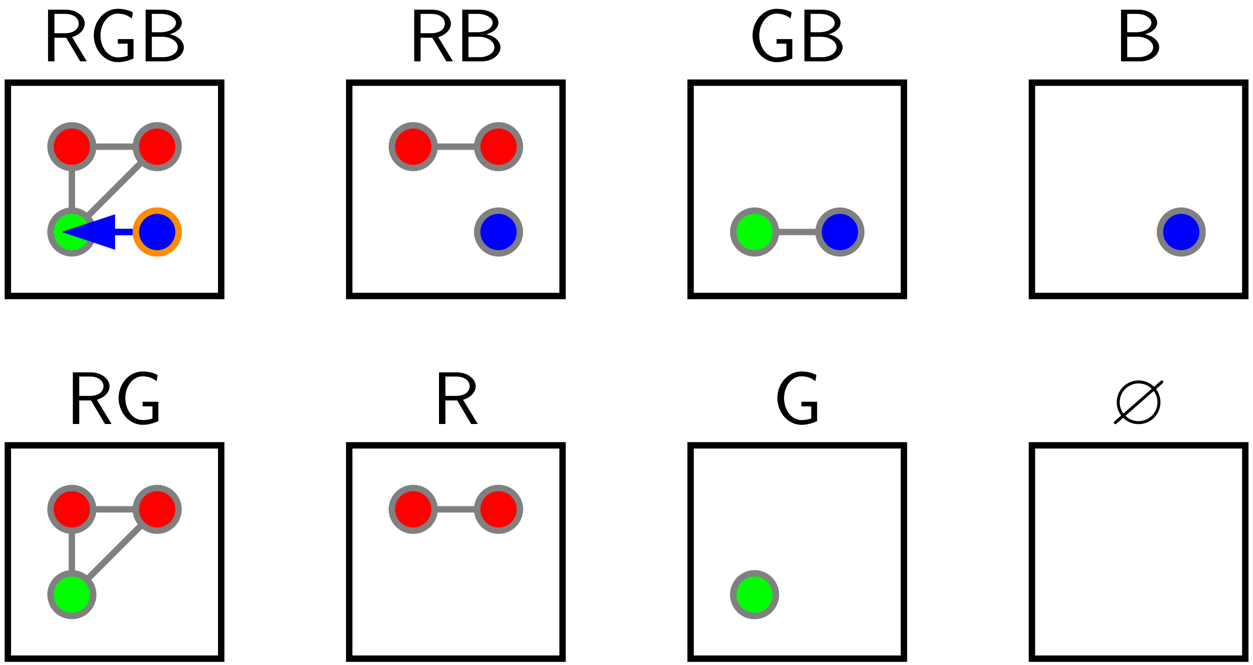
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



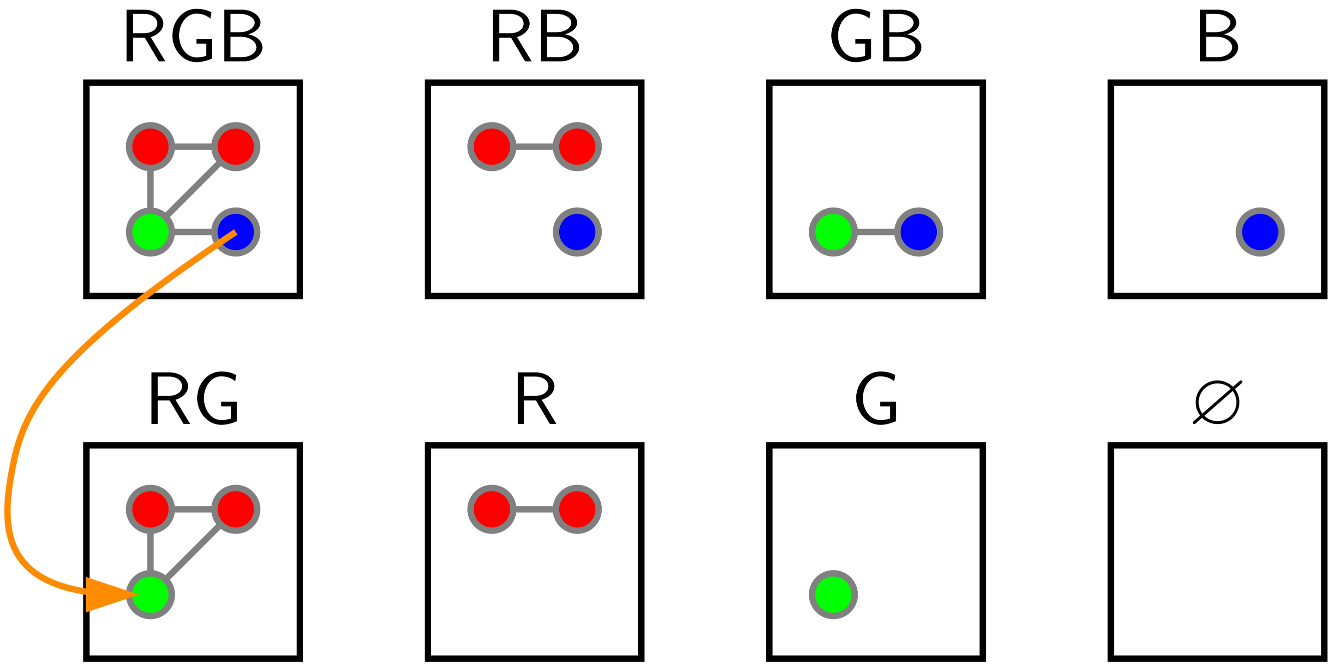
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



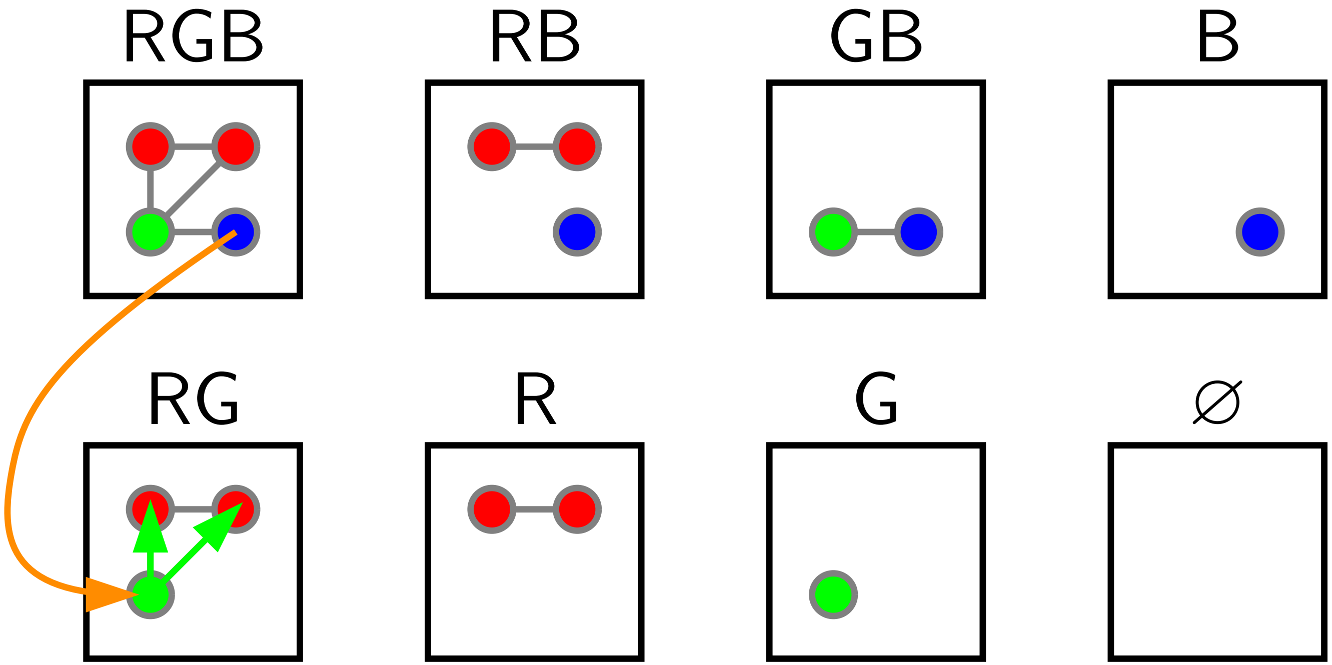
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



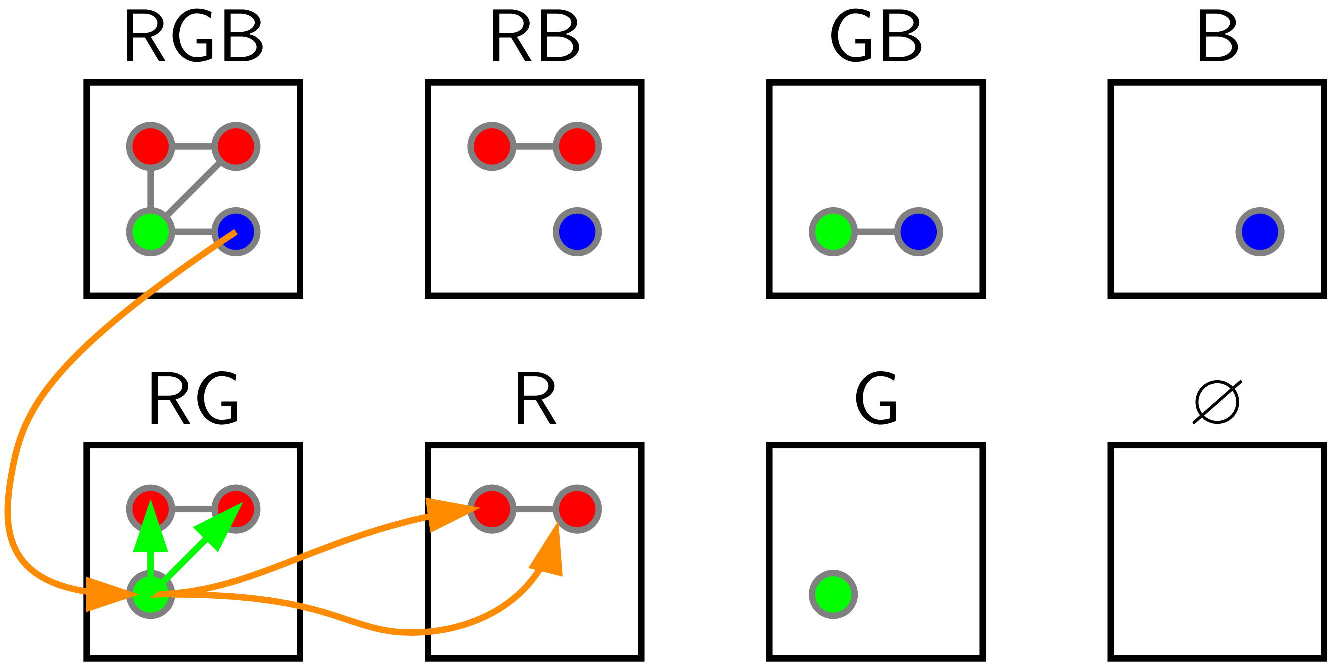
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



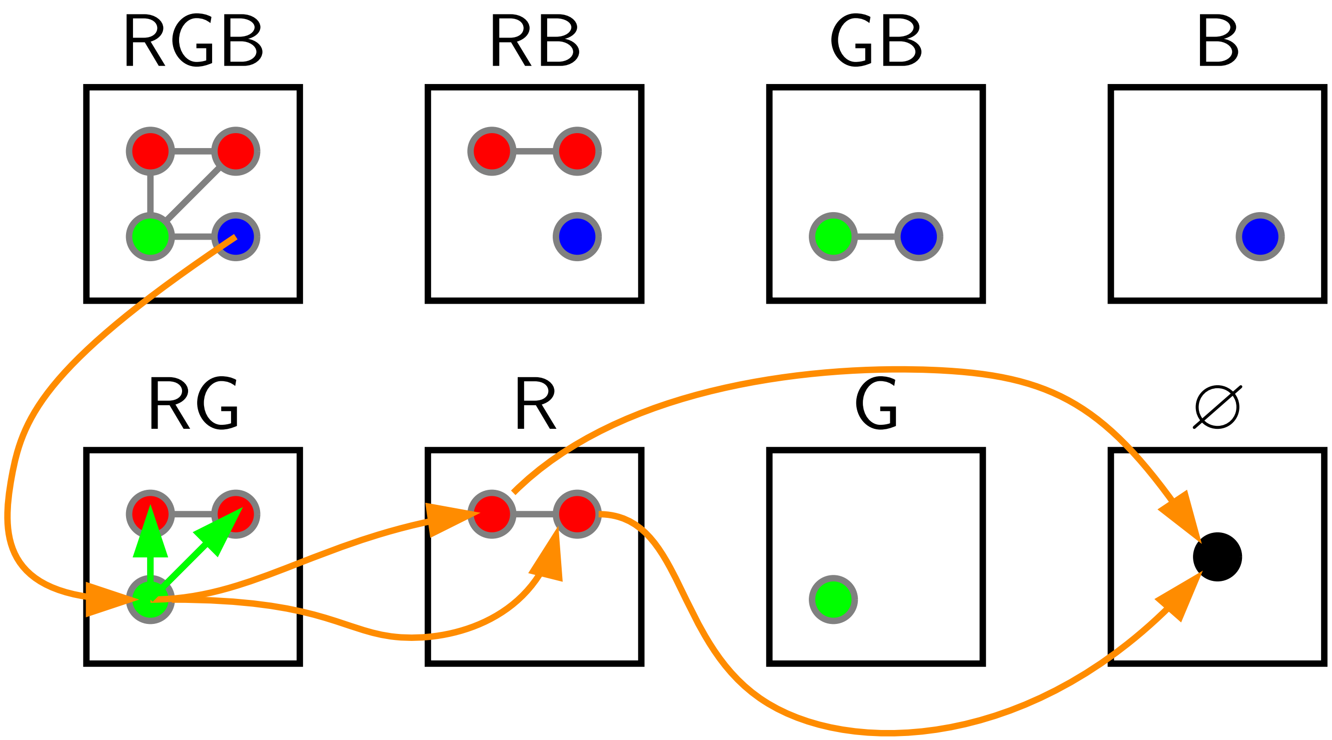
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



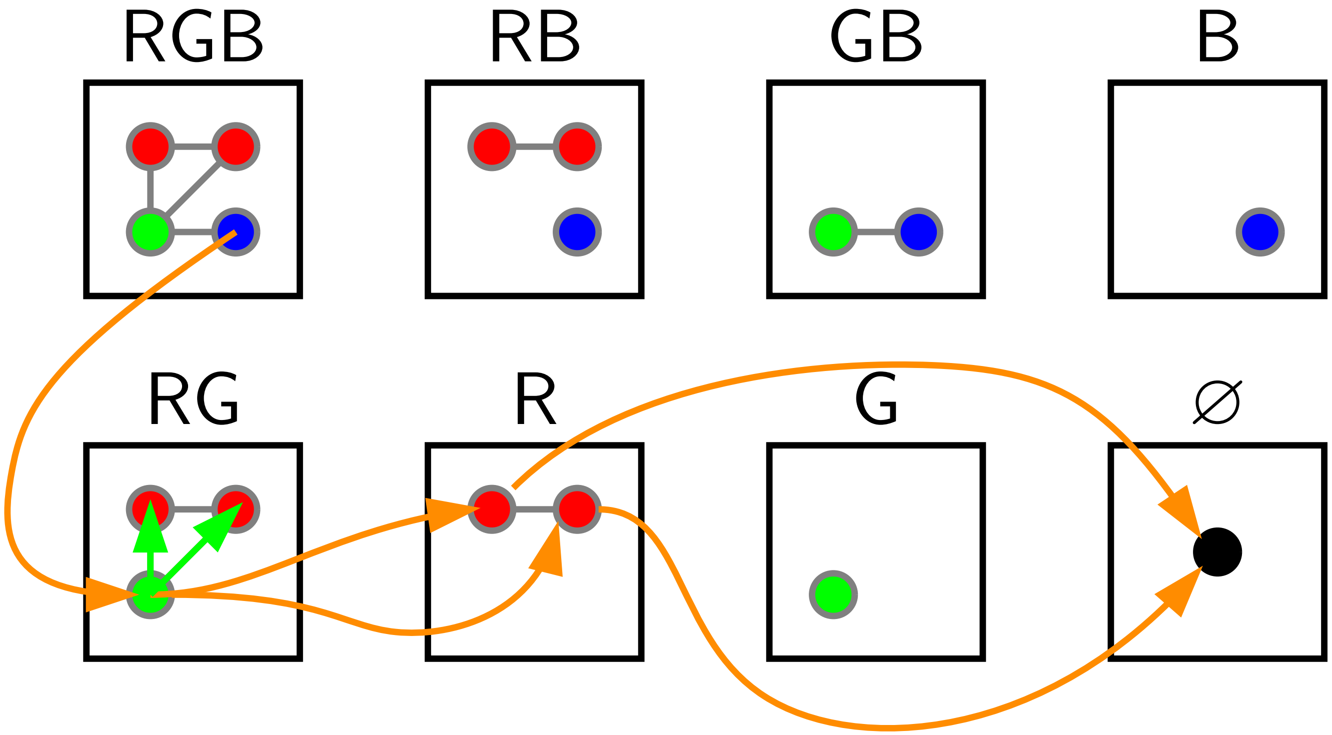
Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



Finding Colorful Paths

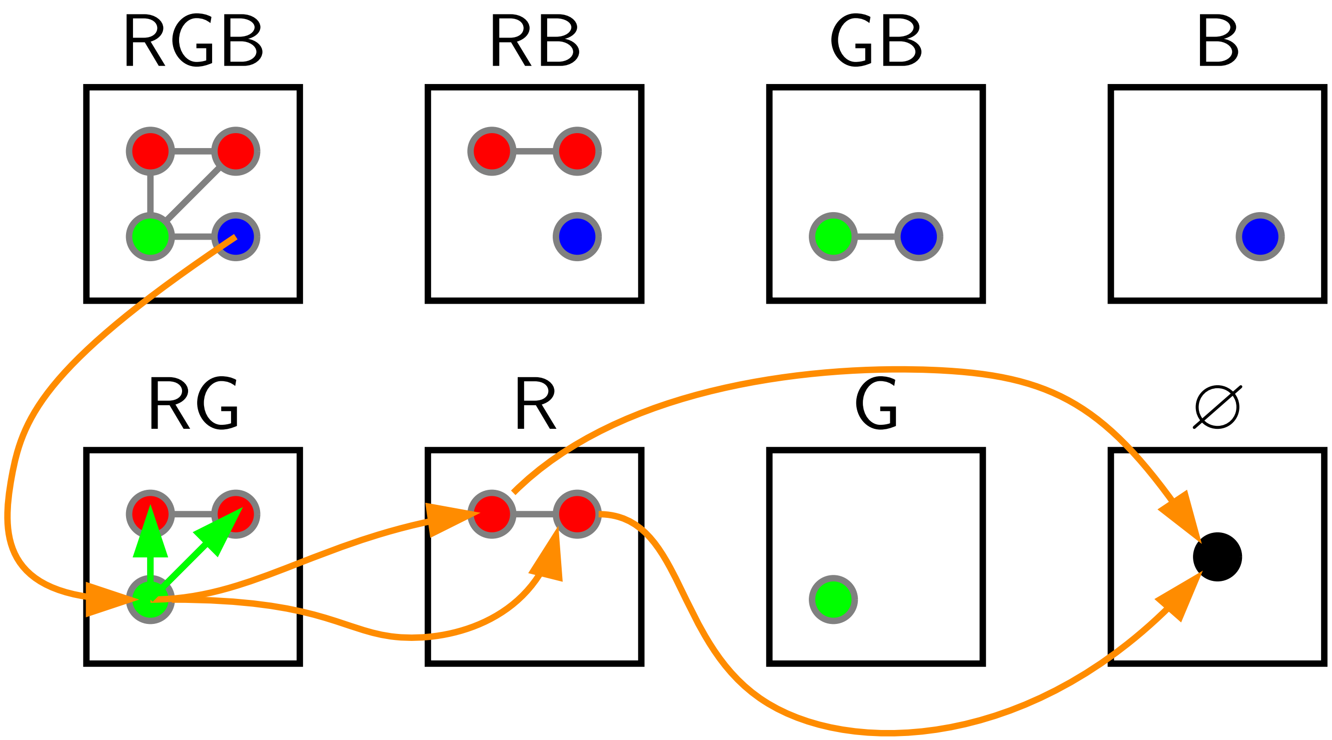
Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



How big is this graph?

Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...

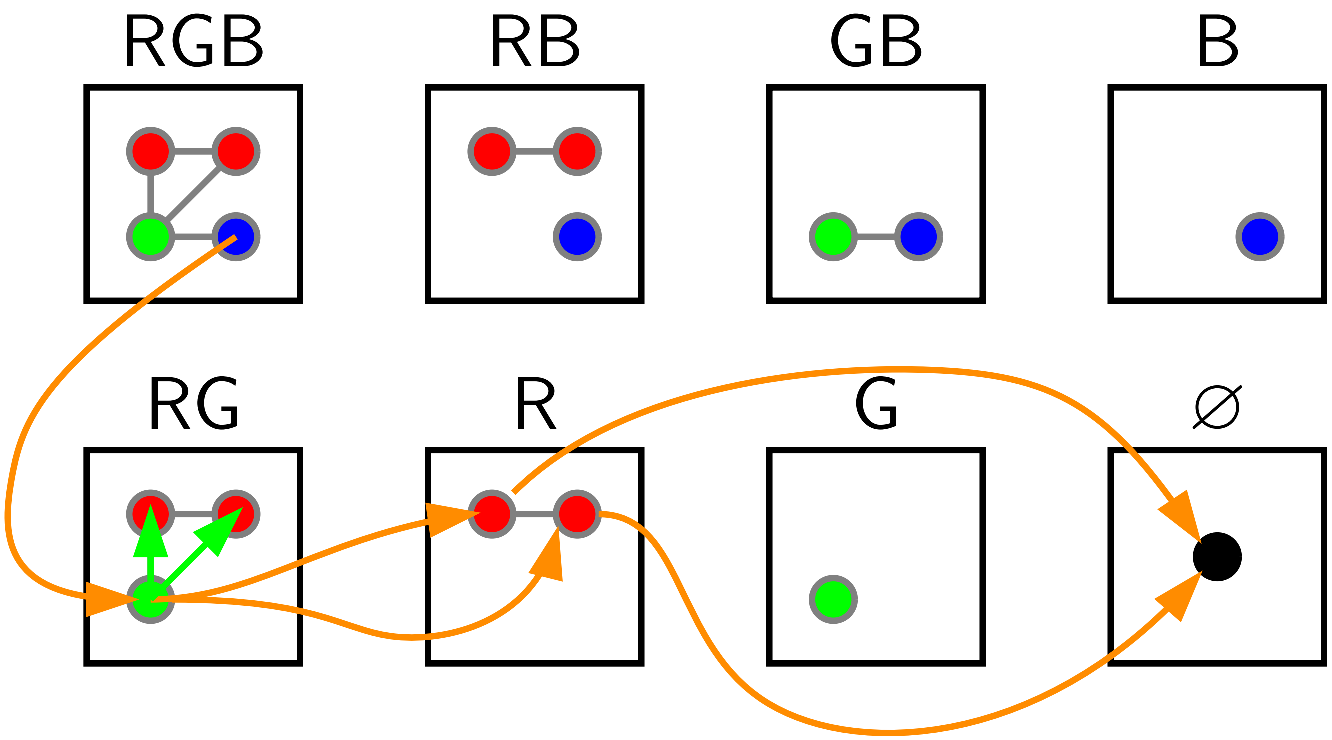


How big is this graph? $O(2^k \cdot n)$ vertices $O(2^k \cdot m)$ edges

Runtime:

Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...

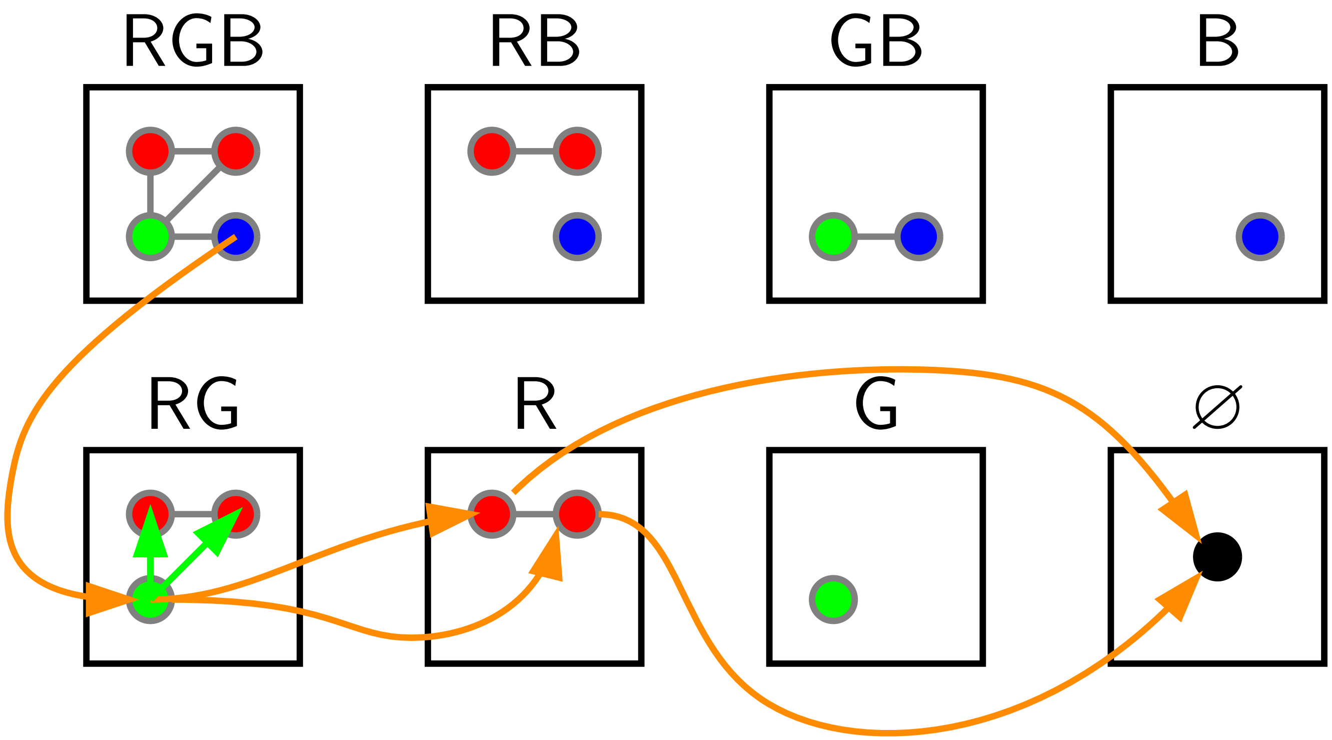


How big is this graph? $O(2^k \cdot n)$ vertices $O(2^k \cdot m)$ edges

Runtime: $O(2^k \cdot m)$ since...

Finding Colorful Paths

Approach 2: For each subset S of the colors, create a copy G_S where G_S contains the vertices colored S and the edges are ...



How big is this graph? $O(2^k \cdot n)$ vertices $O(2^k \cdot m)$ edges

Runtime: $O(2^k \cdot m)$ since... graph is acyclic :)

LONGEST PATH: colorful algorithm

Algorithm

1. repeat e^k times:

LONGEST PATH: colorful algorithm

Algorithm

1. repeat e^k times:

  2. pick a random k-coloring c of V

Longest Path: colorful algorithm

Algorithm

1. repeat e^k times:

2. pick a random k-coloring c of V

3. If there is a colorful path, return Yes

LONGEST PATH: colorful algorithm

Algorithm

1. repeat e^k times:

  2. pick a random k-coloring c of V

  3. If there is a colorful path, return YES

4. Return NO

LONGEST PATH: colorful algorithm

Algorithm

1. repeat e^k times:

  2. pick a random k-coloring c of V

  3. If there is a colorful path, return YES

4. Return NO

Runtime: e^k iterations each $O(2^k \cdot m)$ time

Longest Path: colorful algorithm

Algorithm

1. repeat e^k times:

  2. pick a random k-coloring c of V

  3. If there is a colorful path, return Yes

4. Return No

Runtime: e^k iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

LONGEST PATH: colorful algorithm

Algorithm

1. repeat e^k times:

  2. pick a random k-coloring c of V

  3. If there is a colorful path, return YES

4. Return NO

Runtime: e^k iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm

Algorithm

let us recall the purpose of this

1. repeat e^k times:

  2. pick a random k-coloring c of V

  3. If there is a colorful path, return YES

4. Return NO

Runtime: e^k iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm

Algorithm

1. repeat e^k times:

2. pick a random k-coloring c of V

3. If there is a colorful path, return YES

4. Return NO

let us recall the purpose of this

guarantees (randomised):
k-path in G $\rightarrow$ c colorful path

Runtime: e^k iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm

Algorithm

1. repeat for each coloring $c \in \mathcal{C}$:

3. If there is a colorful path, return YES

4. Return NO

Runtime: ~~e^k~~ $|\mathcal{C}|$ iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm

Algorithm What property of $\mathcal{C}$ do we need?

1. repeat for each coloring $c \in \mathcal{C}$:

3. If there is a colorful path, return YES

4. Return NO

Runtime: $|\mathcal{C}|$ ~~e^k~~ iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm

Algorithm What property of $\mathcal{C}$ do we need?

1. repeat for each coloring $c \in \mathcal{C}$:

sufficient: $\forall S \subseteq V$ with $|S| = k$: $\exists c \in \mathcal{C}$: S is colorful

3. If there is a colorful path, return YES

4. Return NO

$|\mathcal{C}|$

Runtime: ~~e^k~~ iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm deterministic

Algorithm What property of $\mathcal{C}$ do we need?

1. repeat for each coloring $c \in \mathcal{C}$:

sufficient: $\forall S \subseteq V$ with $|S| = k$: $\exists c \in \mathcal{C}$: S is colorful

3. If there is a colorful path, return YES

4. Return NO

Runtime: ~~e^k~~ $|\mathcal{C}|$ iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm deterministic

Algorithm

What property of $\mathcal{C}$ do we need?

1. repeat for each coloring $c \in \mathcal{C}$:

sufficient: $\forall S \subseteq V$ with $|S| = k$: $\exists c \in \mathcal{C}$: S is colorful

3. If there is a colorful path, return YES

4. Return NO

Thm [§5.6]: There is $\mathcal{C}$ with this property and $|\mathcal{C}| \in 2^{O(k)} \log n$ so that $\mathcal{C}$ can be produced in $O(|\mathcal{C}|)$ time.

Runtime: ~~e^k~~ $|\mathcal{C}|$ iterations each $O(2^k \cdot m)$ time

total: $O(\ (2e)^k \cdot m\ ) \subset O(5.43657^k \cdot m)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm deterministic

Algorithm What property of $\mathcal{C}$ do we need?

1. repeat for each coloring $c \in \mathcal{C}$:

sufficient: $\forall S \subseteq V$ with $|S| = k$: $\exists c \in \mathcal{C}$: S is colorful

3. If there is a colorful path, return YES

4. Return NO

Thm [§5.6]: There is $\mathcal{C}$ with this property and $|\mathcal{C}| \in 2^{O(k)} \log n$ so that $\mathcal{C}$ can be produced in $O(|\mathcal{C}|)$ time.

Runtime: ~~e^k~~ $|\mathcal{C}|$ iterations each $O(2^k \cdot m)$ time

total: $O(\alpha^k \cdot m \log n)$ time

Thm: There is a randomised algorithm that solves LONGEST PATH in $O^*(5.44^k \cdot m)$ time.

LONGEST PATH: colorful algorithm deterministic

What property of $\mathcal{C}$ do we need?

Algorithm

1. repeat for each coloring $c \in \mathcal{C}$:

sufficient: $\forall S \subseteq V$ with $|S| = k$: $\exists c \in \mathcal{C}$: S is colorful

3. If there is a colorful path, return YES

4. Return NO

Thm [§5.6]: There is $\mathcal{C}$ with this property and $|\mathcal{C}| \in 2^{O(k)} \log n$ so that $\mathcal{C}$ can be produced in $O(|\mathcal{C}|)$ time.

Runtime: ~~e^k~~ $|\mathcal{C}|$ iterations each $O(2^k \cdot m)$ time

total: $O(\alpha^k \cdot m \log n)$ time

Thm: There is an ~~randomised~~ algorithm that solves LONGEST PATH in $O^*(\alpha^k \cdot m \log n)$ time

Color Coding: User's Guide

Given: Graph G

Question: Is H a(n induced) subgraph of G (graph H, $|H| =$

Color Coding: User's Guide

Given: Graph G

Question: Is H a(n induced) subgraph of G (graph H, $|H| =$

1. Randomly color vertices

Color Coding: User's Guide

Given: Graph G
Question: Is H a(n induced) subgraph of G (graph H, $|H| = $

1. Randomly color vertices

2. Show that Pr[copy of H is colorful] $\geq 1/f(k)$

Color Coding: User's Guide

Given: Graph G
Question: Is H a(n induced) subgraph of G (graph H, $|H| = $

1. Randomly color vertices

2. Show that Pr[copy of H is colorful] $\geq 1/f(k)$

3. Find colorful copy of H in FPT-time

Color Coding: User's Guide

Given: Graph G
Question: Is H a(n induced) subgraph of G (graph H, $|H| = $

1. Randomly color vertices

2. Show that Pr[copy of H is colorful] $\geq 1/f(k)$

3. Find colorful copy of H in FPT-time

4. repeat $O(\ f(k)\ )$ times

Color Coding: User's Guide

Given: Graph G
Question: Is H a(n induced) subgraph of G (graph H, $|H| =$

1. Randomly color vertices
2. Show that Pr[copy of H is colorful] $\geq 1/f(k)$
3. Find colorful copy of H in FPT-time
4. repeat $O(\ f(k)\ )$ times

(**5.** Derandomise)

Longest Path: Approach 3 (sketch)

#Multilabeled Walks

Given: Graph $G = (V, E)$, vertex $v \in V$, number k, edge labels $\lambda(e) \in \Lambda = \{1..k\}$

Question: How many walks in G start at v, have length k, but don't use an edge label twice?

Longest Path: Approach 3 (sketch)

#Multilabeled Walks

Given: Graph $G = (V, E)$, vertex $v \in V$, number k, edge labels $\lambda(e) \in \Lambda = \{1..k\}$

Question: How many walks in G start at v, have length k, but don't use an edge label twice?

Algorithm:

Longest Path: Approach 3 (sketch)

#Multilabeled Walks

Given: Graph $G = (V, E)$, vertex $v \in V$, number k, edge labels $\lambda(e) \in \Lambda = \{1..k\}$

Question: How many walks in G start at v, have length k, but don't use an edge label twice?

Algorithm: dynamic program

Recurrence:

Longest Path: Approach 3 (sketch)

#Multilabeled Walks

Given: Graph $G = (V, E)$, vertex $v \in V$, number k, edge labels $\lambda(e) \in \Lambda = \{1..k\}$

Question: How many walks in G start at v, have length k, but don't use an edge label twice?

Algorithm: dynamic program

Recurrence: $A(S, u)$ with $S \subseteq \Lambda, u \in V$

Runtime:

Longest Path: Approach 3 (sketch)

#Multilabeled Walks

Given: Graph $G = (V, E)$, vertex $v \in V$, number k, edge labels $\lambda(e) \in \Lambda = \{1..k\}$

Question: How many walks in G start at v, have length k, but don't use an edge label twice?

Algorithm: dynamic program

Recurrence: $A(S, u)$ with $S \subseteq \Lambda, u \in V$

Runtime: $O(2^k n^2)$

Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return YES **?!**

4. Otherwise: return NO

Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return YES

4. Otherwise: return NO

Lemma: Non-simple walks are counted evenly.

Counting Multilabeled Walks: Algorithm

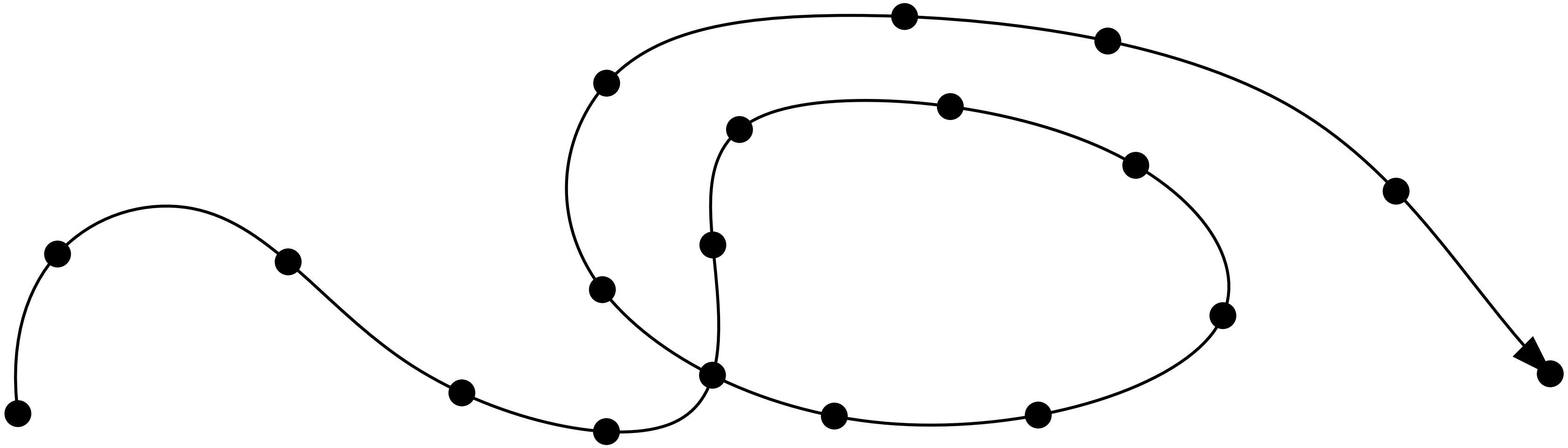
1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return YES

4. Otherwise: return NO

Lemma: Non-simple walks are counted evenly.



Counting Multilabeled Walks: Algorithm

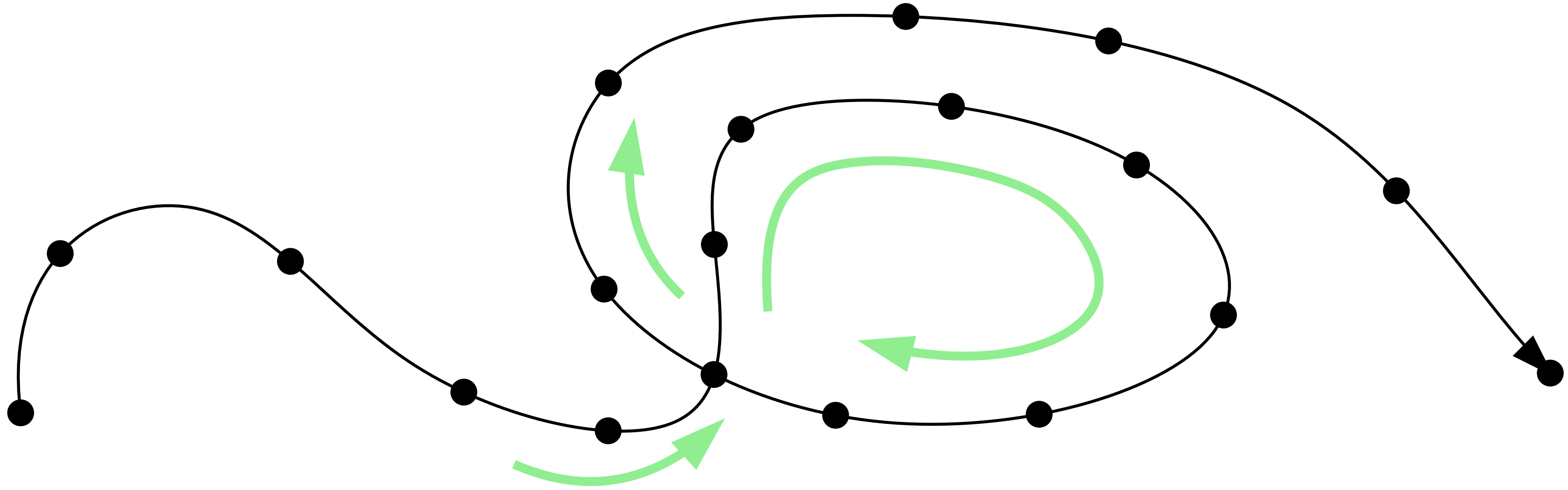
1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return Yes

4. Otherwise: return No

Lemma: Non-simple walks are counted evenly.



Counting Multilabeled Walks: Algorithm

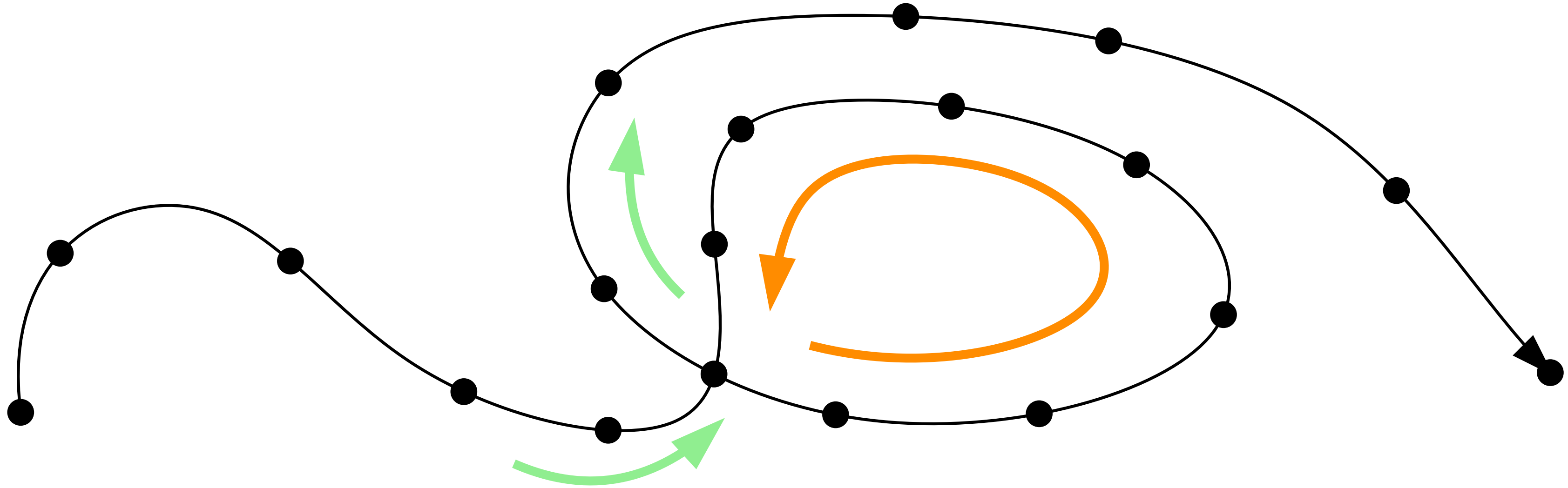
1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return $\textsc{Yes}$

4. Otherwise: return $\textsc{No}$

Lemma: Non-simple walks are counted evenly.



Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return YES $\leftarrow$ correct!

4. Otherwise: return NO

Lemma: Non-simple walks are counted evenly.

Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return YES $\leftarrow$ correct!

4. Otherwise: return NO

Lemma: Non-simple walks are counted evenly.

Problem: What happens if the number of k-paths is even?

An “Isolation” Lemma

Parameterized Algorithms Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

An “Isolation” Lemma

Parameterized Algorithms Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

with probabilty at least $1 - n/N$ we have:

$$\text{argmin}_{S \in \mathcal{F}} \sum_{v \in S} \omega(v) \text{ is unique.}$$

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

with probabilty at least $1 - n/N$ we have:

$$\text{argmin}_{S \in \mathcal{F}} \sum_{v \in S} \omega(v) \text{ is unique.}$$

not exam material

Proof: Let $\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} \omega(S) - \min_{S \in \mathcal{F}, x \in S} \omega(S \setminus \{x\})$

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

with probabilty at least $1 - n/N$ we have:

$$\text{argmin}_{S\in\mathcal{F}} \sum_{v\in S} \omega(v) \text{ is unique.}$$

not exam material

Proof: Let $\alpha(x) = \min_{S\in\mathcal{F}, x\notin S} \omega(S) - \min_{S\in\mathcal{F}, x\in S} \omega(S \setminus \{x\})$

since $\alpha(x)$ does not depend on $\omega(x)$: $\Pr[\alpha(x) = \omega(x)] \leq 1/N$.

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

with probabilty at least $1 - n/N$ we have:

$$\text{argmin}_{S \in \mathcal{F}} \sum_{v \in S} \omega(v) \text{ is unique.}$$

not exam material

Proof: Let $\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} \omega(S) - \min_{S \in \mathcal{F}, x \in S} \omega(S \setminus \{x\})$

since $\alpha(x)$ does not depend on $\omega(x)$: $\Pr[\alpha(x) = \omega(x)] \leq 1/N$.

Thus: $\Pr[\exists x \in U : \alpha(x) = \omega(x)] \leq n/N$.

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

with probabilty at least $1 - n/N$ we have:

$$\text{argmin}_{S \in \mathcal{F}} \sum_{v \in S} \omega(v) \text{ is unique.}$$

not exam material

Proof: Let $\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} \omega(S) - \min_{S \in \mathcal{F}, x \in S} \omega(S \setminus \{x\})$

since $\alpha(x)$ does not depend on $\omega(x)$: $\Pr[\alpha(x) = \omega(x)] \leq 1/N$.

Thus: $\Pr[\exists x \in U \colon \alpha(x) = \omega(x)] \leq n/N$.

Suppose that $A \neq B \in \mathcal{F}$ are both minimum.

Now $\exists x \in U \colon \alpha(x) = \omega(B) - (\omega(A) - \omega(x)) = \omega(x)$.

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$

with probabilty at least $1 - n/N$ we have:

$$\text{argmin}_{S \in \mathcal{F}} \sum_{v \in S} \omega(v) \text{ is unique.}$$

not exam material

Proof: Let $\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} \omega(S) - \min_{S \in \mathcal{F}, x \in S} \omega(S \setminus \{x\})$

since $\alpha(x)$ does not depend on $\omega(x)$: $\Pr[\alpha(x) = \omega(x)] \leq 1/N$.

Thus: $\Pr[\exists x \in U\colon \alpha(x) = \omega(x)] \leq n/N$.

Suppose that $A \neq B \in \mathcal{F}$ are both minimum.

Now $\exists x \in U\colon \alpha(x) = \omega(B) - (\omega(A) - \omega(x)) = \omega(x)$.

This has probability at most n/N. $\square$

An "Isolation" Lemma

Parameterized Algorithms
Lemma 11.5

Lemma: Let $\mathcal{F}$ be a family of subsets of U, with $|U| = n$.

Indep. at random, assign each $x \in U$ weight $\omega(x)$ from $\{1..N\}$ 1..2n

with probabilty at least $1 - n/N$ (1/2) we have:

$$\text{argmin}_{S \in \mathcal{F}} \sum_{v \in S} \omega(v) \text{ is unique.}$$

not exam material

Proof: Let $\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} \omega(S) - \min_{S \in \mathcal{F}, x \in S} \omega(S \setminus \{x\})$

since $\alpha(x)$ does not depend on $\omega(x)$: $\Pr[\alpha(x) = \omega(x)] \leq 1/N$.

Thus: $\Pr[\exists x \in U \colon \alpha(x) = \omega(x)] \leq n/N$.

Suppose that $A \neq B \in \mathcal{F}$ are both minimum.

Now $\exists x \in U \colon \alpha(x) = \omega(B) - (\omega(A) - \omega(x)) = \omega(x)$.

This has probability at most n/N. □

Counting Multilabeled Walks: Algorithm

1. Copy each edge k times and apply labels $1..k$

2. $a \leftarrow$ number of multilabeled k-walks

3. If a is odd, return $\textsc{Yes}$ $\leftarrow$ correct!

4. Otherwise: return $\textsc{No}$

Lemma: Non-simple walks are counted evenly.

Problem: What happens if the number of k-paths is even?

Solution: Isolation Lemma gives edge weights (with $\mathcal{F} =$ k-paths in G), such that a *k-path of minimum weight* is unique. Then we just expand DP to count weighted multilabled walks.