Color Coding and Chromatic Coding

Venkatesh Raman

The Institute of Mathematical Sciences

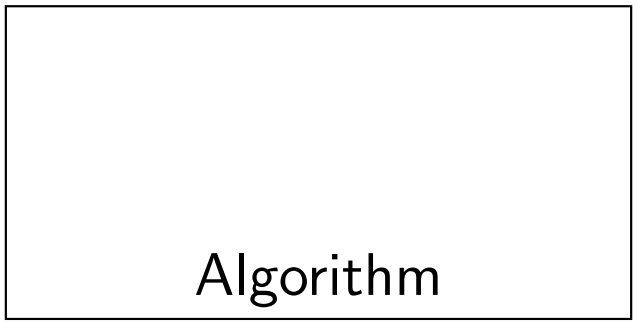
3 March 2014

Randomized Algorithms

Let $\Pi \subseteq \Sigma^*$ be a problem.

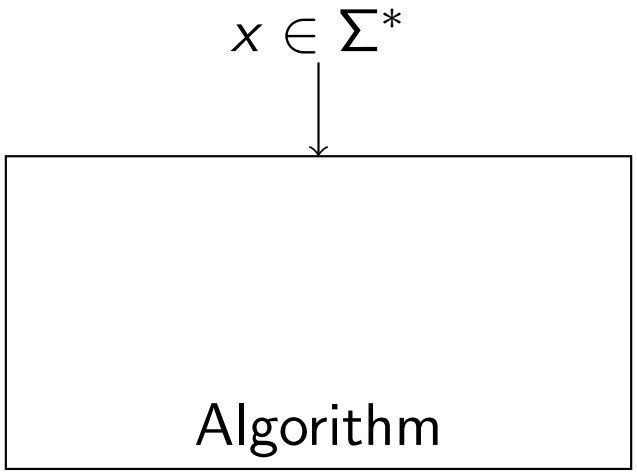
Randomized Algorithms

Let $\Pi \subseteq \Sigma^*$ be a problem.



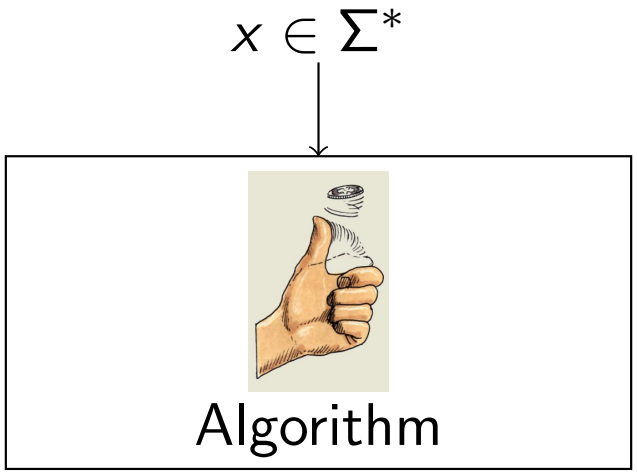
Randomized Algorithms

Let $\Pi \subseteq \Sigma^*$ be a problem.



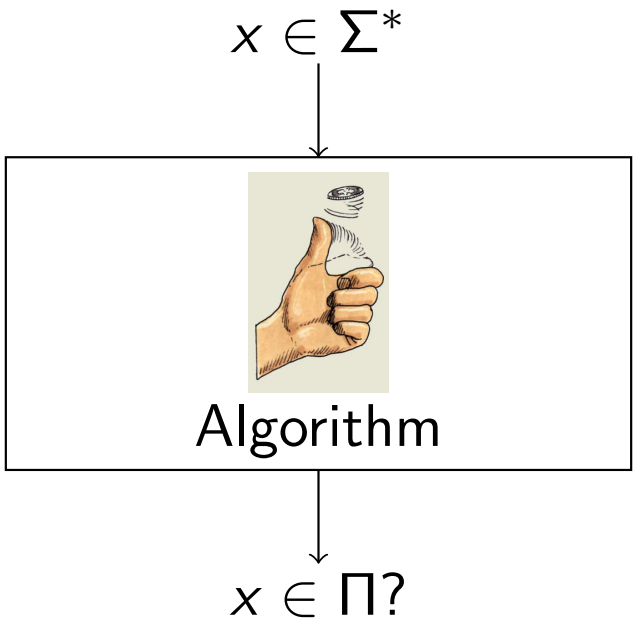
Randomized Algorithms

Let $\Pi \subseteq \Sigma^*$ be a problem.



Randomized Algorithms

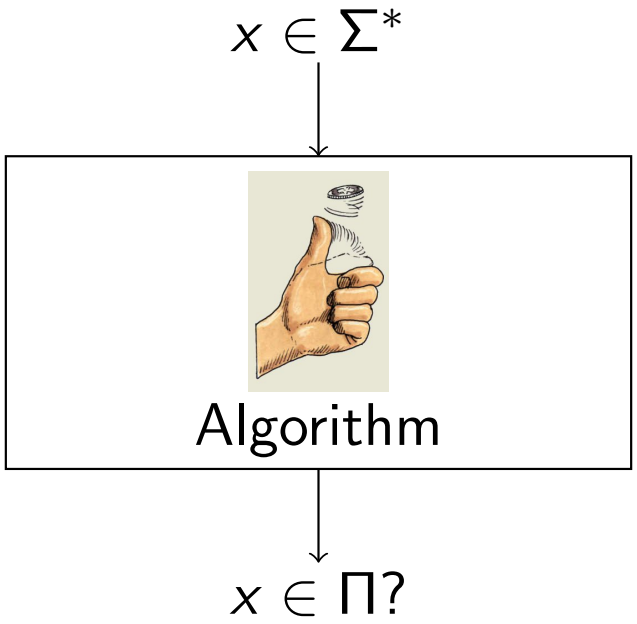
Let $\Pi \subseteq \Sigma^*$ be a problem.



(Output correct answer with good probability)

Randomized Algorithms

Let $\Pi \subseteq \Sigma^*$ be a problem.



(Output correct answer with good probability)

Here we are interested in randomized algorithms for parameterized problems taking running time $f(k)|x|^{O(1)}$ with constant success probability.

Randomized Algorithm for UFVS

Definition (Feedback Vertex Set)

Let $G = (V, E)$ be an undirected graph. $S \subseteq V$ is called feedback vertex set if $G \setminus S$ is a forest.

Randomized Algorithm for UFVS

Definition (Feedback Vertex Set)

Let $G = (V, E)$ be an undirected graph. $S \subseteq V$ is called feedback vertex set if $G \setminus S$ is a forest.

Undirected Feedback Vertex Set (UFVS)

Input:	An undirected graph $G = (V, E)$ and a positive integer k
Parameter:	k
Question:	Does there exists a feedback vertex set of size at most k

Randomized Algorithm for UFVS

Do the following preprocessing rules

Randomized Algorithm for UFVS

Do the following preprocessing rules

- Delete degree ≤ 1 vertices.

Randomized Algorithm for UFVS

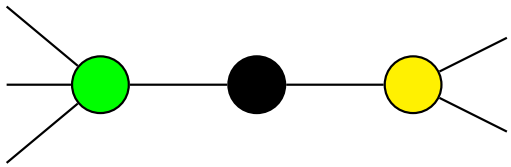
Do the following preprocessing rules

- Delete degree ≤ 1 vertices.
- Short circuit degree 2 vertices.

Randomized Algorithm for UFVS

Do the following preprocessing rules

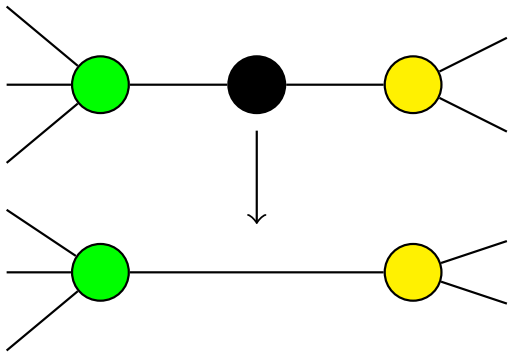
- Delete degree ≤ 1 vertices.
- Short circuit degree 2 vertices.



Randomized Algorithm for UFVS

Do the following preprocessing rules

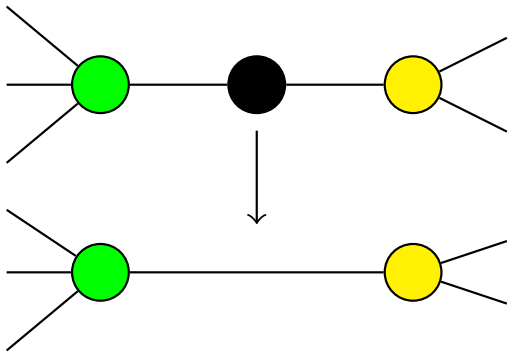
- Delete degree ≤ 1 vertices.
- Short circuit degree 2 vertices.



Randomized Algorithm for UFVS

Do the following preprocessing rules

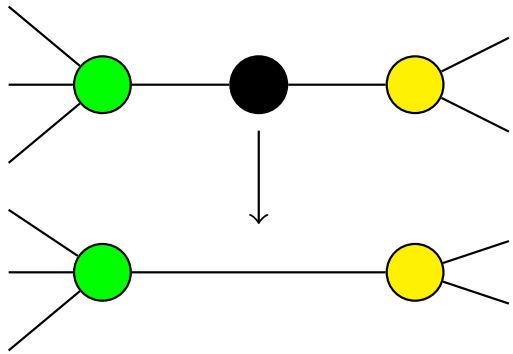
- Delete degree ≤ 1 vertices.
- Short circuit degree 2 vertices.
- Add vertex x to FVS, if x has self loop.



Randomized Algorithm for UFVS

Do the following preprocessing rules

- Delete degree ≤ 1 vertices.
- Short circuit degree 2 vertices.
- Add vertex x to FVS, if x has self loop.



Now we can assume every vertex in G has degree ≥ 3.

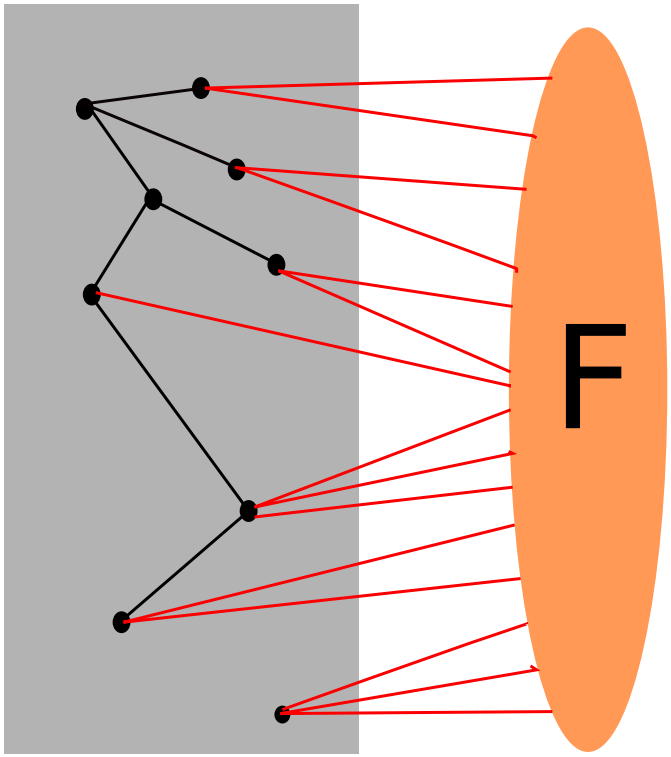
Randomized Algorithm for UFVS

Claim : If G is graph with minimum degree ≥ 3, then number of edges incident to any *FVS* F is $\geq \frac{E(G)}{2}$.

Randomized Algorithm for UFVS

Claim : If G is graph with minimum degree ≥ 3, then number of edges incident to any *FVS* F is $\geq \frac{E(G)}{2}$.

Proof:



H=G - F

;

Randomized Algorithm for UFVS

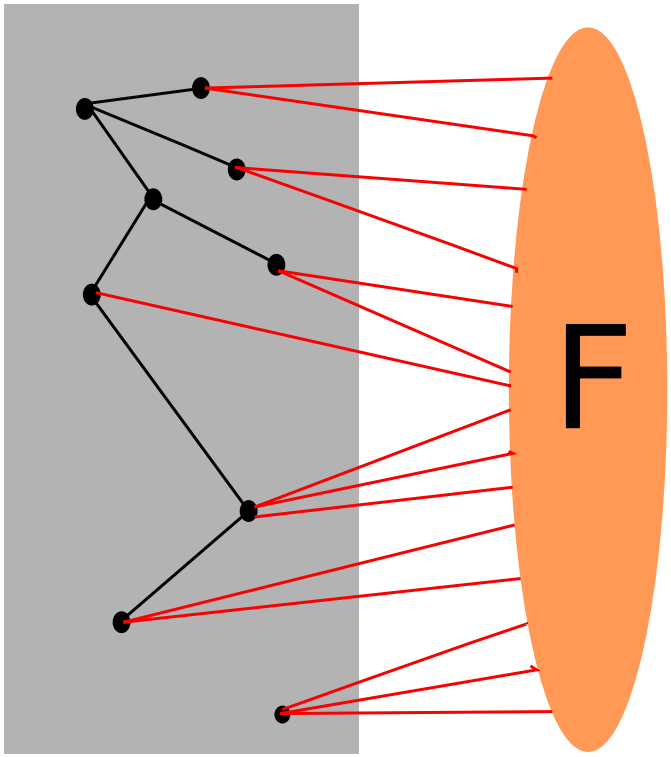
Claim : If G is graph with minimum degree ≥ 3, then number of edges incident to any *FVS* F is $\geq \frac{E(G)}{2}$.

Proof: Let

$V_{\leq 1}$ = set of degree ≤ 1 vertices in H,

V_2 = set of degree 2 vertices in H, and

$V_{\geq 3}$ = set of degree ≥ 3 vertices in H.



H=G - F

;

Randomized Algorithm for UFVS

Claim : If G is graph with minimum degree ≥ 3, then number of edges incident to any *FVS* F is $\geq \frac{E(G)}{2}$.

Proof: Let

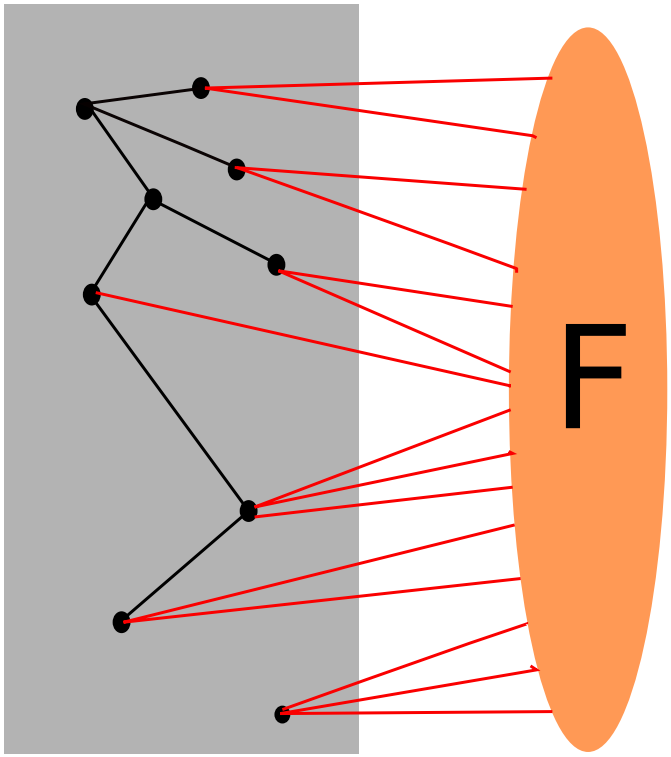
$V_{\leq 1}$ = set of degree ≤ 1 vertices in H,

V_2 = set of degree 2 vertices in H, and

$V_{\geq 3}$ = set of degree ≥ 3 vertices in H.

Number of edges between H and F,

$$E(H, F) \geq 2V_{\leq 1} + V_2$$



H=G - F

;

Randomized Algorithm for UFVS

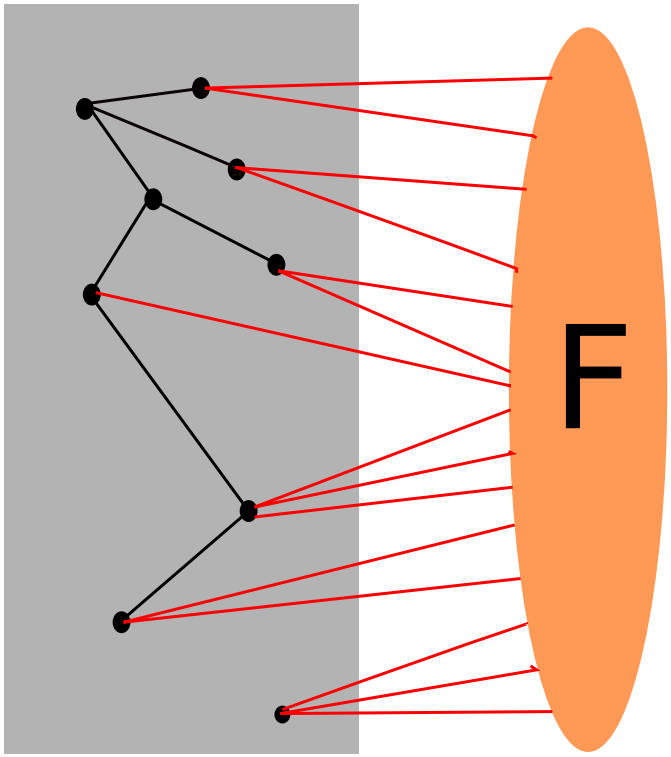
Claim : If G is graph with minimum degree ≥ 3, then number of edges incident to any *FVS* F is $\geq \frac{E(G)}{2}$.

Proof: Let

$V_{\leq 1}$ = set of degree ≤ 1 vertices in H,
V_2 = set of degree 2 vertices in H, and
$V_{\geq 3}$ = set of degree ≥ 3 vertices in H.

Number of edges between H and F,

$$\begin{aligned} E(H,F) &\geq 2V_{\leq 1} + V_2 \\ &> V_{\leq 1} + V_2 + V_{\geq 3} \ (\because V_{\leq 1} > V_{\geq 3}) \end{aligned}$$



H=G - F

;

Randomized Algorithm for UFVS

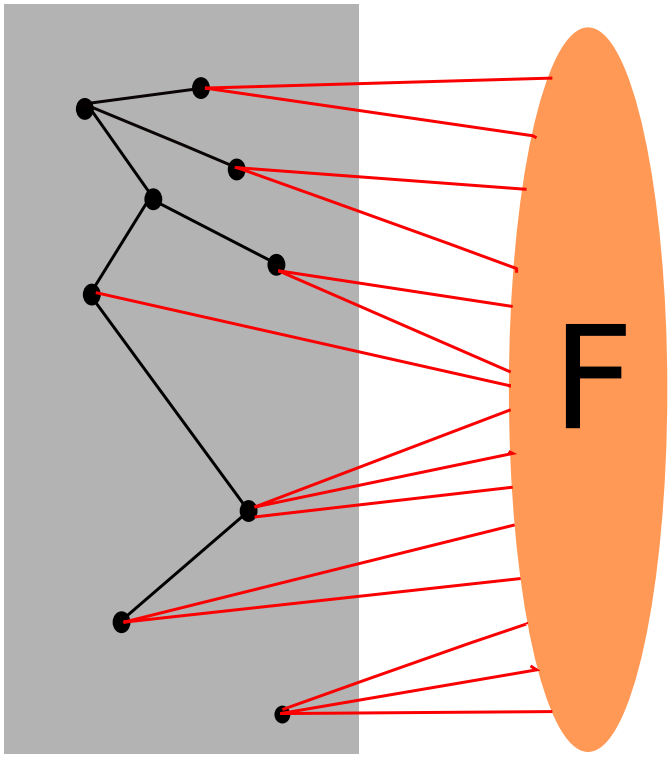
Claim : If G is graph with minimum degree ≥ 3, then number of edges incident to any *FVS* F is $\geq \frac{E(G)}{2}$.

Proof: Let

$V_{\leq 1}$ = set of degree ≤ 1 vertices in H,
V_2 = set of degree 2 vertices in H, and
$V_{\geq 3}$ = set of degree ≥ 3 vertices in H.

Number of edges between H and F,

$$\begin{aligned} E(H,F) &\geq 2V_{\leq 1} + V_2 \\ &> V_{\leq 1} + V_2 + V_{\geq 3} \ (\because V_{\leq 1} > V_{\geq 3}) \\ &= V(H) > E(H). \end{aligned}$$



H=G - F

;

Randomized Algorithm for UFVS

Algorithm

Initially we set $S \leftarrow \emptyset$. Now run the following steps k times.

1: Apply preprocessing rules and create an equivalent instance (G', k')

2: Pick an edge e, u.a.r (i.e with probability $\frac{1}{E(G')}$)

3: Choose an end point of e, u.a.r (i.e with probability $\frac{1}{2}$) into S and delete it from the graph.

Now if S is a FVS, output S, otherwise output No.

Analysis

If the input instance is No instance Pr[Algorithm output No] $= 1$.
Let the input instance is Yes instance and F is a FVS od size $\leq k$.

$$\begin{aligned} \Pr[\text{Choosing an edge incident to F in step 2}] &\geq \frac{1}{2} \\ \Pr[\text{Chosen vertex is in F in step 3}] &\geq \frac{1}{2} \cdot \frac{1}{2} \\ \Pr[\text{S=F}] &\geq \frac{1}{4^k} \end{aligned}$$

Now we repeat the algorithm 4^k times.

$\Pr[\text{Algorithm fails in all repetitions}] \leq \left(1 - \frac{1}{4^k}\right)^{4^k} \leq \frac{1}{e}$.

$\Pr[\text{Algorithm succeed at least once}] \geq 1 - \frac{1}{e} \geq \frac{1}{2}$

Color Coding

- This technique is introduced by Alon et al. (1995)
- This technique is used to solve constant treewidth k-sized subgraph isomorphism problem.

Color Coding

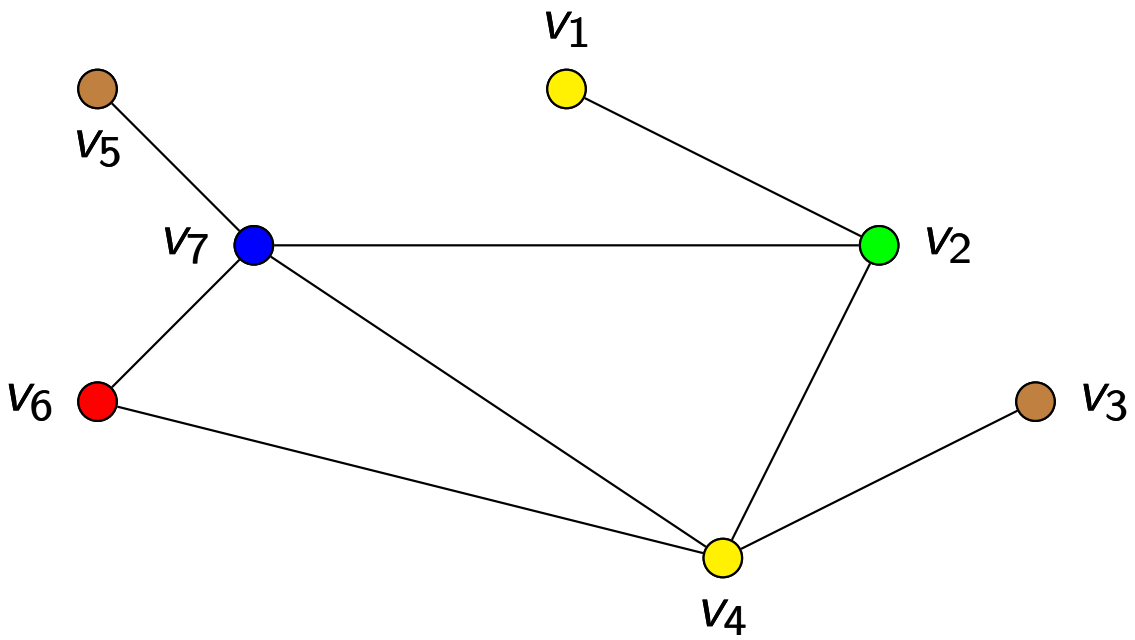
- This technique is introduced by Alon et al. (1995)
- This technique is used to solve constant treewidth k-sized subgraph isomorphism problem.
- We illustrate this technique by applying to k-PATH

k-PATH

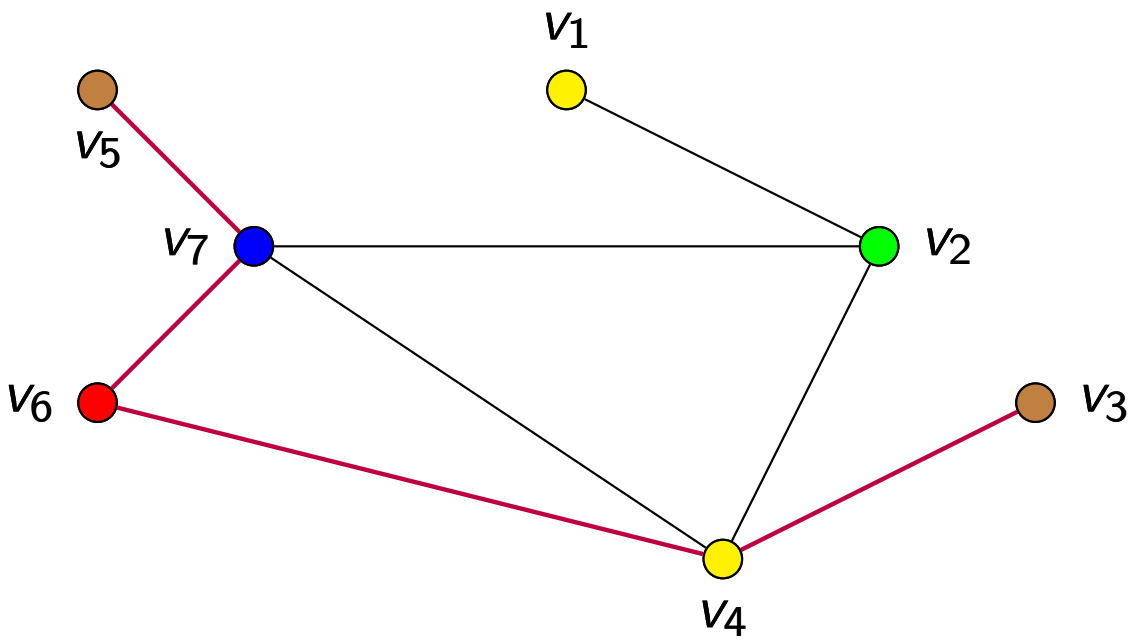
k-PATH

Input:	An undirected graph G and a positive integer k
Parameter:	k
Question:	Does there exists a path on k vertices in G

Colorful path



Colorful path



colorful path on 5 vertices in the graph G ?

Algorithm for k-PATH

Algorithm for k-PATH

- Color each vertex of the input graph G, u.a.r using from the set of k colors, $[k]$
- Check whether there exists a colorful k-path in the colored graph and output YES/NO accordingly

Algorithm for k-PATH

- Color each vertex of the input graph G, u.a.r using from the set of k colors, $[k]$
- Check whether there exists a colorful k-path in the colored graph and output YES/NO accordingly

Running Time : Time to check colorful k-path $\times$ $n^{O(1)}$

Algorithm for k-PATH

- Color each vertex of the input graph G, u.a.r using from the set of k colors, $[k]$
- Check whether there exists a colorful k-path in the colored graph and output YES/NO accordingly

Running Time : Time to check colorful k-path $\times$ $n^{O(1)}$

Probability of success

- If (G, k) is a NO instance, the probability of success is 1.
- If (G, k) is an YES instance, the probability of success is at least $\frac{k!}{k^k}$.

DP for Checking colorful k-path

We introduce $2^k \cdot |V(G)|$ Boolean variables:

$x(v, C) = \text{TRUE}$ for some $v \in V(G)$ and $C \subseteq [k]$

$\Updownarrow$

There is path ends at v where each color in C appears exactly once and no other color appears.

DP for Checking colorful k-path

Clearly, $x(v, \{color(v)\}) = \text{TRUE}$. Recurrence for vertex v with color r:

$$x(v, C) = \bigvee_{uv \in E(G)} x(u, C \setminus \{r\})$$

If we know every $x(v, C)$ with $|C| = i$, then we can determine every $x(v, C)$ with $|C| = i + 1$. All the values can be determined in time $O(2^k \cdot |E(G)|)$.

There is a colorful path ends at $t \iff x(t, [k]) = \text{TRUE}$ for some t.

Analysis for k-PATH

The algorithm for k-PATH has

- Running time $2^k n^{O(1)}$

Analysis for k-PATH

The algorithm for k-PATH has

- Running time $2^k n^{O(1)}$
- Success probability
  - at least $\frac{k!}{k^k}$ if (G, k) is an YES instance
  - 1 if (G, k) is an NO instance

Analysis for k-PATH

The algorithm for k-PATH has

- Running time $2^k n^{O(1)}$
- Success probability
  - at least $\frac{k!}{k^k}$ if (G, k) is an YES instance
  - 1 if (G, k) is an NO instance

Now run the algorithm $\frac{k^k}{k!}(\leq e^k)$ times and output YES if at least once we get an YES answer, otherwise output NO.

Analysis for k-PATH

The algorithm for k-PATH has

- Running time $2^k n^{O(1)}$
- Success probability
  - at least $\frac{k!}{k^k}$ if (G, k) is an YES instance
  - 1 if (G, k) is an NO instance

Now run the algorithm $\frac{k^k}{k!}(\leq e^k)$ times and output YES if at least once we get an YES answer, otherwise output NO.

- Probability of failure $\leq \left(1 - \frac{k!}{k^k}\right)^{k^k/k!} \leq e^{-1}$

Analysis for k-PATH

The algorithm for k-PATH has

- Running time $2^k n^{O(1)}$
- Success probability
  - at least $\frac{k!}{k^k}$ if (G, k) is an YES instance
  - 1 if (G, k) is an NO instance

Now run the algorithm $\frac{k^k}{k!}(\leq e^k)$ times and output YES if at least once we get an YES answer, otherwise output NO.

- Probability of failure $\leq \left(1 - \frac{k!}{k^k}\right)^{k^k/k!} \leq e^{-1}$

k-PATH can be solved in randomized $(2e)^k n^{O(1)}$ time, with constant success probability.

Derandomization

- Suppose we have a list of colorings $col_1, col_2, \ldots, col_m$ such that for any $S \subseteq V$ with $|S| = k$ there exists i, col_i is one-to-one on S, then instead of random coloring we can use these list of colorings to get a deterministic algorithm.

Derandomization

- Suppose we have a list of colorings $col_1, col_2, \ldots, col_m$ such that for any $S \subseteq V$ with $|S| = k$ there exists i, col_i is one-to-one on S, then instead of random coloring we can use these list of colorings to get a deterministic algorithm.
- Such a list of colorings is called an (n, k)-*family of perfect hash functions*

Derandomization

- Suppose we have a list of colorings $col_1, col_2, \ldots, col_m$ such that for any $S \subseteq V$ with $|S| = k$ there exists i, col_i is one-to-one on S, then instead of random coloring we can use these list of colorings to get a deterministic algorithm.
- Such a list of colorings is called an (n, k)-*family of perfect hash functions*
- There exists an (n, k)-*family of perfect hash functions* of size $e^k k^{O(\log k)} \log^2 n$ and can be constructed in time linear in the output size

Derandomization

- Suppose we have a list of colorings $col_1, col_2, \ldots, col_m$ such that for any $S \subseteq V$ with $|S| = k$ there exists i, col_i is one-to-one on S, then instead of random coloring we can use these list of colorings to get a deterministic algorithm.
- Such a list of colorings is called an (n, k)-*family of perfect hash functions*
- There exists an (n, k)-*family of perfect hash functions* of size $e^k k^{O(\log k)} \log^2 n$ and can be constructed in time linear in the output size

k-PATH can be solved deterministically in $(2e)^k k^{O(\log k)} n^{O(1)}$ time.

Chromatic Coding

Thank You