

Advanced topics on Algorithms

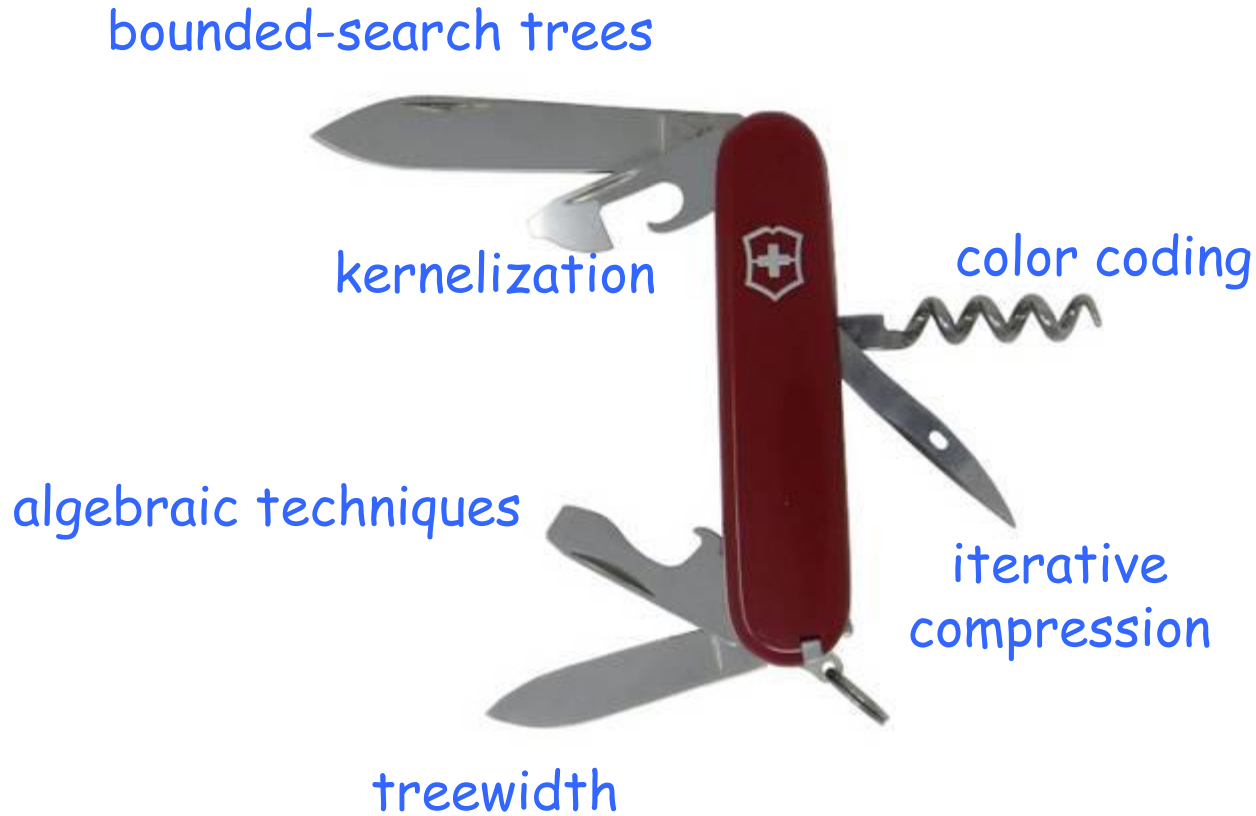
Luciano Gualà

www.mat.uniroma2.it/~guala/

Parameterized algorithms

Episode II

Toolbox (to show a problem is FPT)



color coding

k-Path

Input:

- a graph $G=(V,E)$
- a nonnegative integer k

question:

is there a simple path of k vertices

parameter: k

obs: NP-hard since it contains the Hamiltonian path as special case

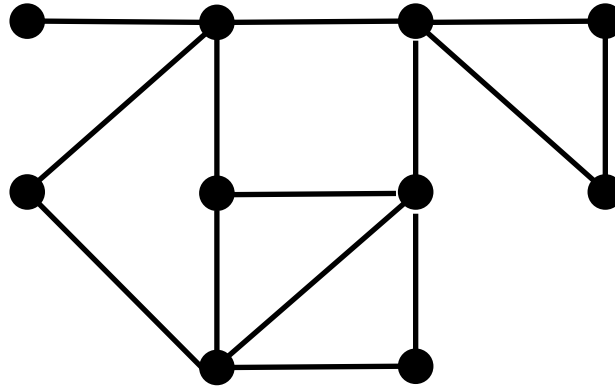
Theorem [Alon, Yuster, Zwick 1994]

k-Path can be solved in time $2^{O(k)} n^{O(1)}$.

previous best algorithms had running time $k^{O(k)} n^{O(1)}$.

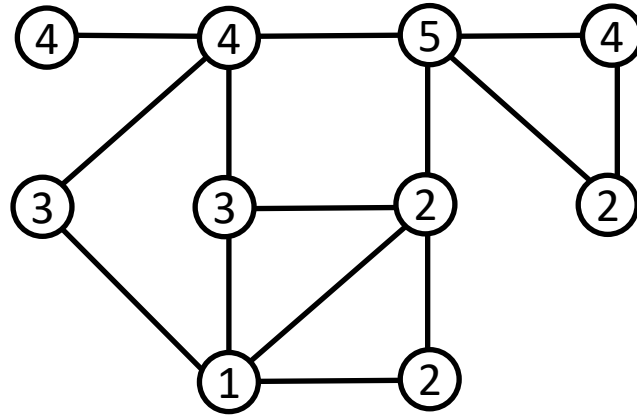
color coding

- assign colors from $\{1, \dots, k\}$ to vertices $V(G)$ uniformly and independently at random.



color coding

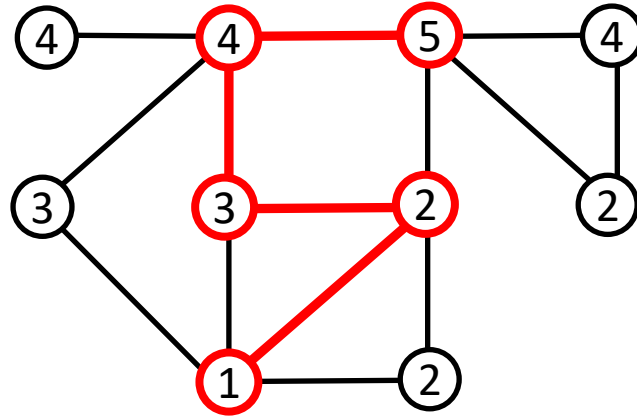
- assign colors from $\{1, \dots, k\}$ to vertices $V(G)$ uniformly and independently at random.



- check if there is a path colored 1-2-...-k and output YES or NO

color coding

- assign colors from $\{1, \dots, k\}$ to vertices $V(G)$ uniformly and independently at random.



- check if there is a path colored 1-2-...-k and output YES or NO

obs1: if there is no k-path: no path colored 1-2-...-k exists

➡ NO

obs2: if there is a k-path: there is some probability that this path is colored 1-2-...-k

probability of success: k^{-k} .

➡ YES with probability k^{-k} .

boosting the probability of success: independent repetitions

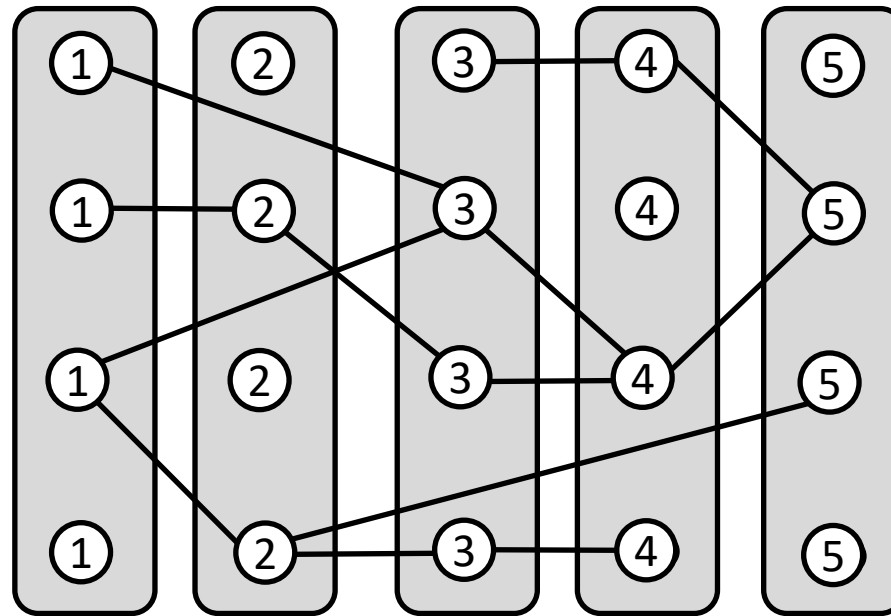
Useful fact

If the probability of success is at least p , then the probability that the algorithm does not say "YES" after $1/p$ repetitions is at most

$$(1-p)^{1/p} \leq (e^{-p})^{1/p} = 1/e \approx 0.38$$

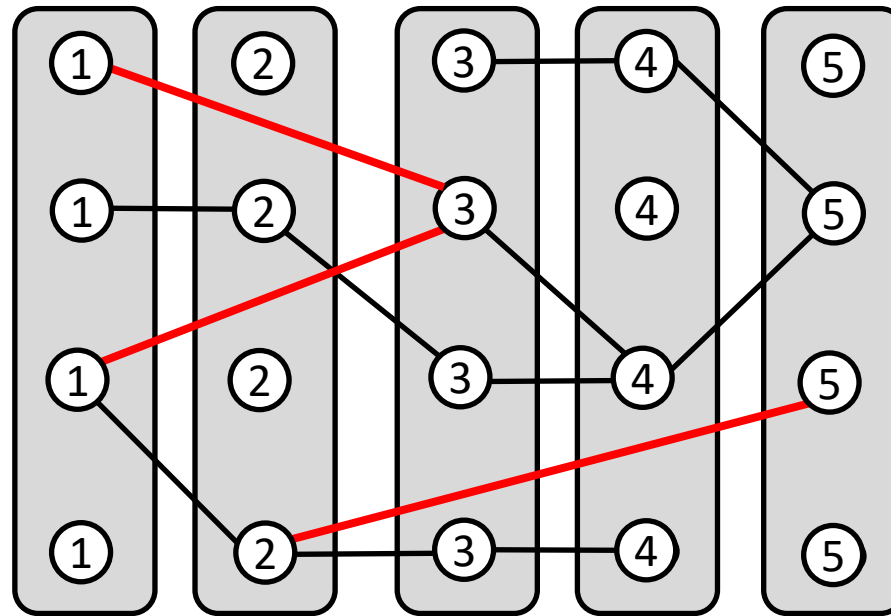
- Thus, if $p \geq k^{-k}$ then error probability is at most $1/e$ after k^k repetitions
 - repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant
- example:** trying $100 k^k$ random colorings, the probability of a wrong answer is at most $(1/e)^{100}$.

Finding a path colored 1-2-...-k



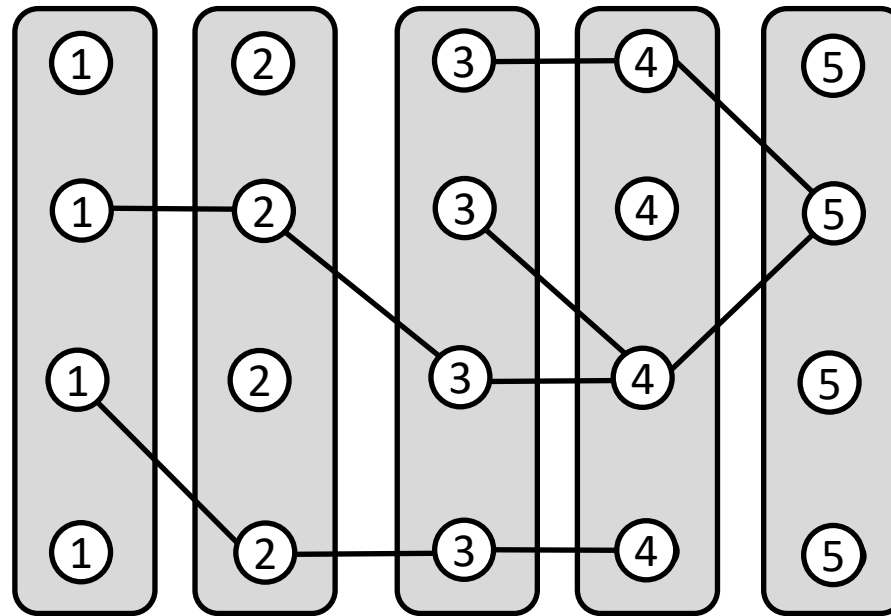
- edges connecting nonadjacent color classes are removed
- the remaining edges are directed towards the larger class
- all we need to check is if there is a directed path from class 1 to class k

Finding a path colored 1-2-...-k



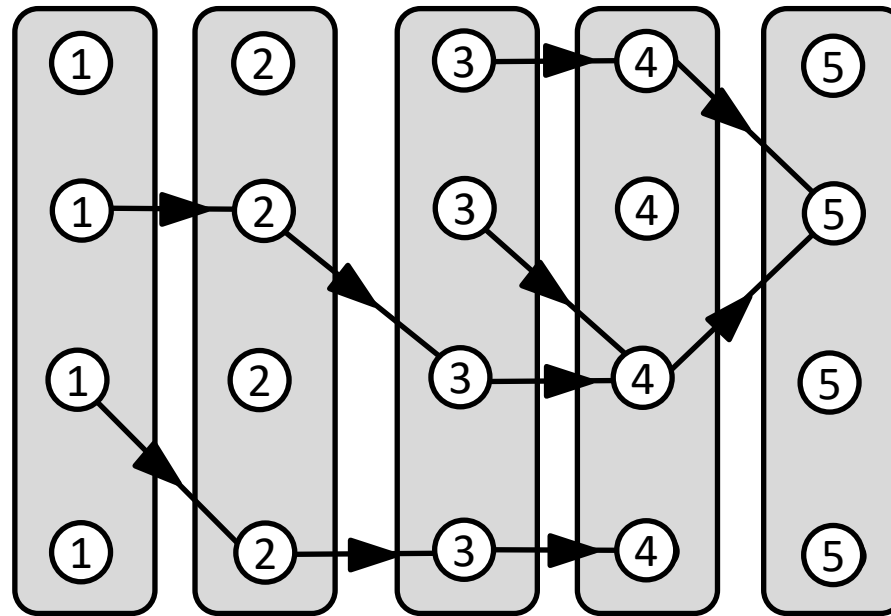
- edges connecting nonadjacent color classes are removed
- the remaining edges are directed towards the larger class
- all we need to check is if there is a directed path from class 1 to class k

Finding a path colored 1-2-...-k



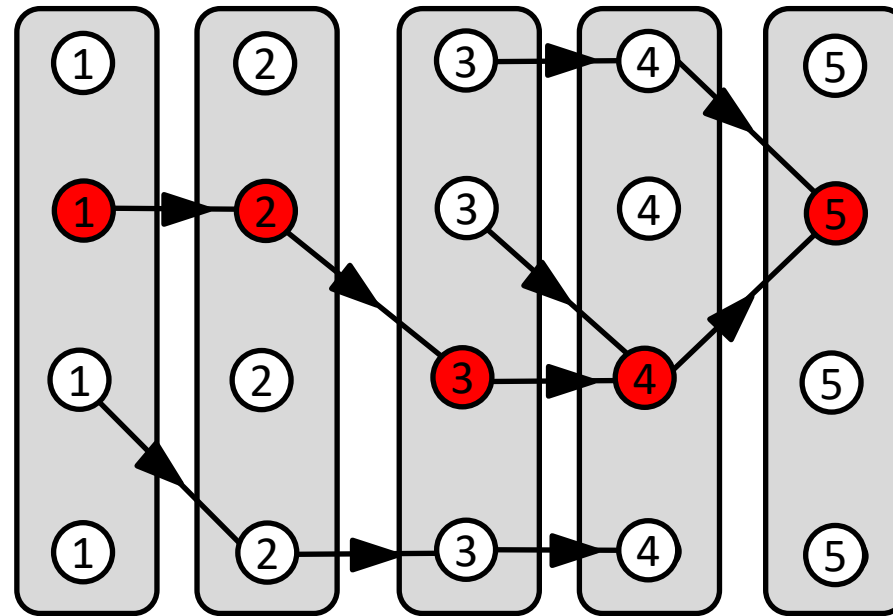
- edges connecting nonadjacent color classes are removed
- the remaining edges are directed towards the larger class
- all we need to check is if there is a directed path from class 1 to class k

Finding a path colored 1-2-...-k



- edges connecting nonadjacent color classes are removed
- the remaining edges are directed towards the larger class
- all we need to check is if there is a directed path from class 1 to class k

Finding a path colored 1-2-...-k

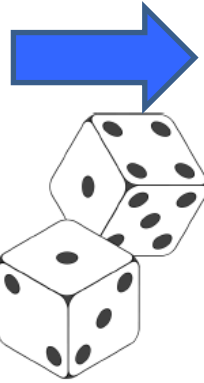


- edges connecting nonadjacent color classes are removed
- the remaining edges are directed towards the larger class
- all we need to check if there is a directed path from class 1 to class k

color coding

k-Path

color coding
success probability
 k^{-k}

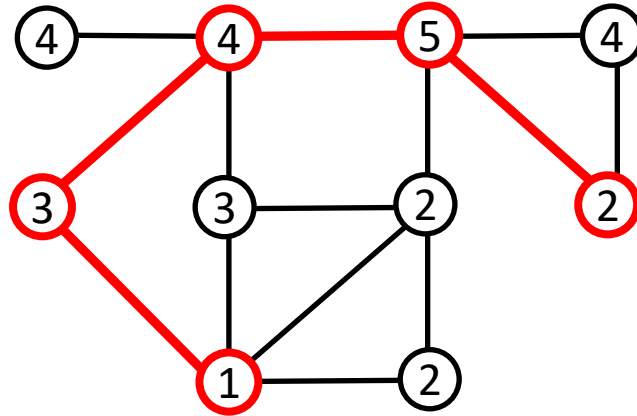


finding a
1-...-k
colored path

polynomial time
solvable

improved color coding

- assign colors from $\{1, \dots, k\}$ to vertices $V(G)$ uniformly and independently at random.



- check if there is a **colorful path** (each color appears exactly once) and output YES or NO

obs1: if there is no k -path: no colorful path exists

➡ NO

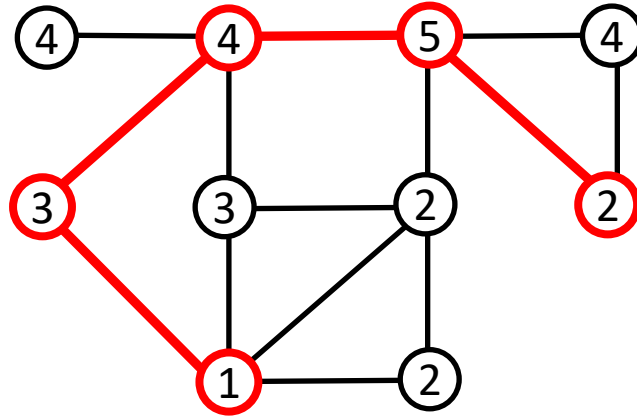
obs2: if there is a k -path: there is some probability that it is colorful

probability of success:

$$\frac{k! k^{n-k}}{k^n} = \frac{k!}{k^k} \geq \frac{(k/e)^k}{k^k} = e^{-k} \quad \text{➡ YES with probability } e^{-k}.$$

improved color coding

- assign colors from $\{1, \dots, k\}$ to vertices $V(G)$ uniformly and independently at random.



- repeating the algorithm $100 e^k$ times, the probability of a wrong answer is at most $(1/e)^{100}$.

how to find a colorful path?

- try all permutations: $k! n^{O(1)}$ time
- dynamic programming: $2^k n^{O(1)}$ time

finding a colorful path

subproblems:

$v \in V$, non-empty subset $S \subseteq \{1, \dots, k\}$

$\text{Path}(v, S)$: is there a path P ending at v such that each color of S appears in P exactly once and no other color appears in P ?

obs1: There is a colorful path iff $\text{Path}(v, \{1, \dots, k\}) = \text{TRUE}$ for some v

obs2: # of subproblems $2^k n$

finding a colorful path

subproblems:

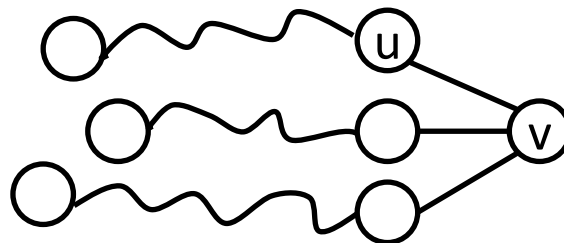
$v \in V$, non-empty subset $S \subseteq \{1, \dots, k\}$

$\text{Path}(v, S)$: is there a path P ending at v such that each color of S appears in P exactly once and no other color appears in P ?

$|S|=1$ (base case)

$\text{Path}(v, S) = \text{TRUE}$ iff $S = \{\text{col}(v)\}$

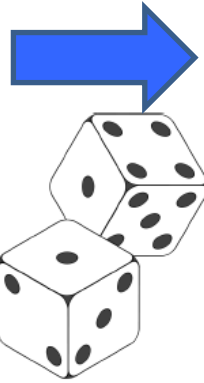
$|S| > 1$

$$\text{Path}(v, S) = \begin{cases} \text{OR}_{(u,v) \in E} \text{Path}(u, S - \{\text{col}(v)\}) & \text{if } \text{col}(v) \in S \\ \text{FALSE} & \text{otherwise} \end{cases}$$


color coding

k-Path

color coding
success probability
 e^{-k}



finding a
colorful path

solvable in
 $2^k n^{O(1)}$ time

Theorem

There is a randomized algorithm for k-Path that runs in time $(e2)^k n^{O(1)}$ that either reports a failure or find a path of k vertices. Moreover, the algorithm finds a solution of a YES-instance with constant probability.

Kernelization

a $2k$ -vertex kernel for VC
based on linear programming

an Integer Linear Programming (ILP) formulation of VC

LP-relaxation

minimize $\sum_{v \in V} x_v$

minimize $\sum_{v \in V} x_v$

subject to $x_u + x_v \geq 1 \quad e=(u,v) \in E$

subject to $x_u + x_v \geq 1 \quad e=(u,v) \in E$

$x_v \in \{0,1\} \quad v \in V$

$x_v \geq 0 \quad v \in V$

relax with $x_v \geq 0$ & $x_v \leq 1$

redundant

a feasible solution is a fractional VC

OPT_f : cost of the min fractional VC

$OPT_f \leq OPT$

Let x be an optimal fractional solution.

$$V_0 = \{ v \in V : x_v < \frac{1}{2} \}$$

$$V_{0.5} = \{ v \in V : x_v = \frac{1}{2} \}$$

$$V_1 = \{ v \in V : x_v > \frac{1}{2} \}$$

Theorem (Nemhauser-Trotter)

There is a minimum vertex cover S of G such that

$$V_1 \subseteq S \subseteq V_1 \cup V_{0.5}$$

proof

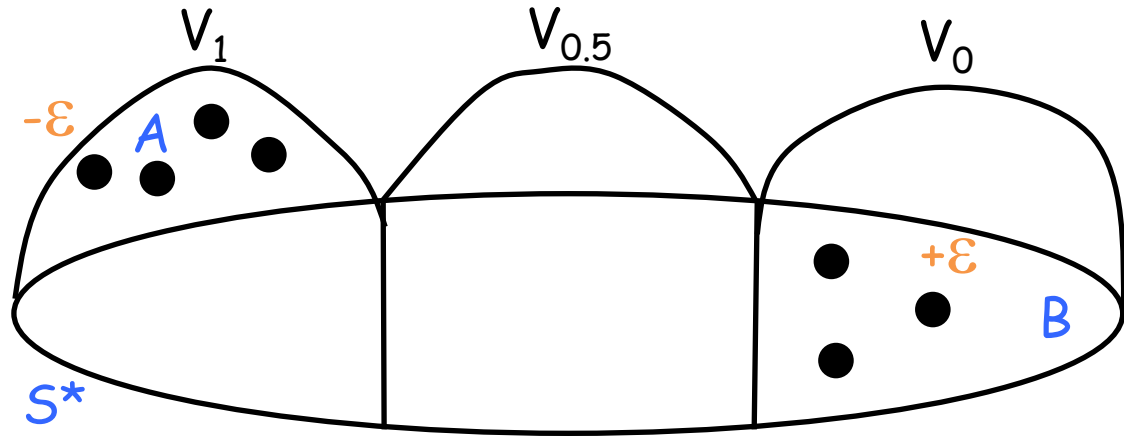
Let S^* be a minimum VC Let $S = (S^* \setminus V_0) \cup V_1$

S is a VC since every adjacent vertex of V_0 must be in V_1

claim: S is minimum assume by contradiction that $|S| > |S^*|$

$$\Rightarrow \underbrace{|S^* \cap V_0|}_B < \underbrace{|V_1 \setminus S^*|}_A$$

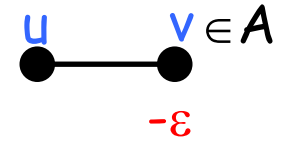
$$y_v = \begin{cases} x_v - \varepsilon & \text{if } v \in A \\ x_v + \varepsilon & \text{if } v \in B \\ x_v & \text{otherwise} \end{cases}$$



$$\varepsilon = \min\{|x_v - \frac{1}{2}| : v \in V_0 \cup V_1\}$$

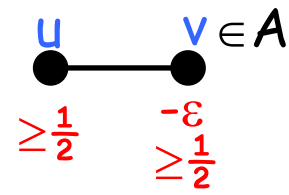
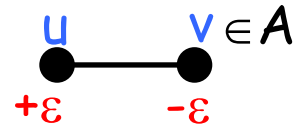
claim:

- y is strictly better than x
 - y is feasible
- } contradicts optimality of x



interesting case:
 u or $v \in A$

since S^* is a VC then $u \in B$ or $u \in S^* \setminus B$



kernelization

compute an optimal fractional solution x of the LP-relaxation for the VC instance (G,k) . Define $V_0, V_{0.5}, V_1$ as before.

if $\sum_{v \in V} x_v > k$ conclude that (G,k) is a No-instance.

Otherwise, greedily pick V_1 in the VC, delete vertices in V_1 and V_0 (and all their incident edges).

The new instance is $(G'=G-(V_1 \cup V_0), k'=k-|V_1|)$.

Theorem

k -Vertex Cover admits a kernel of at most $2k$ vertices.

proof

(G,k) is a YES-instance iff (G',k') is a YES-instance

$$|V(G')| = |V_{0.5}| = \sum_{v \in V_{0.5}} 2x_v \leq 2 \sum_{v \in V} x_v \leq 2k$$



...in the next episode...

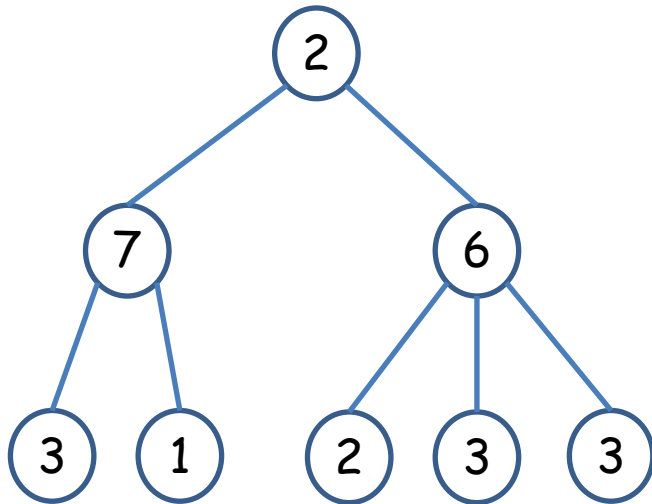
The party problem

problem: invite people to a party

maximize: total fun factor of the invited people

constraint: everyone should be having fun

➔ do not invite a colleague and his direct boss at the same time!



input: a tree with weights on the nodes

goal: an independent set of maximum total weight

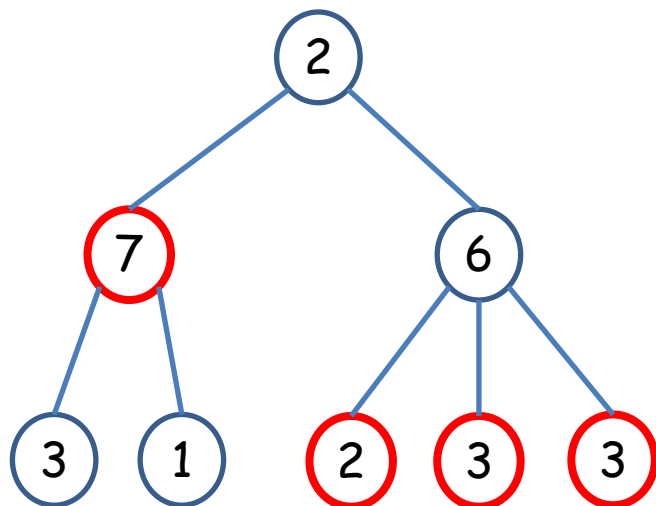
The party problem

problem: invite people to a party

maximize: total fun factor of the invited people

constraint: everyone should be having fun

➔ do not invite a colleague and his direct boss at the same time!



input: a tree with weights on the nodes

goal: an independent set of maximum total weight

OPT= 15

Exercise: give a polynomial time algorithm for it