上海科技大学
ShanghaiTech University

CS240 Algorithm Design and Analysis

Lecture 15

FPT (fixed parameter tractable)

Quan Li
Fall 2023
2023.11.21

What you need to know

- **PSPACE.** Decision problems solvable in polynomial **space**
- **PSPACE problems**
  - QSAT
  - Planning
- **Theorem.** NP $\subseteq$ PSPACE $\subseteq$ EXPTIME
- **PSPACE-Complete.** Problem Y is PSPACE-complete if (i) Y is in PSPACE and (ii) for every problem X in PSPACE, $X \le_p Y$
- **PSPACE-Complete problems**
  - QSAT
  - Competitive Facility Location

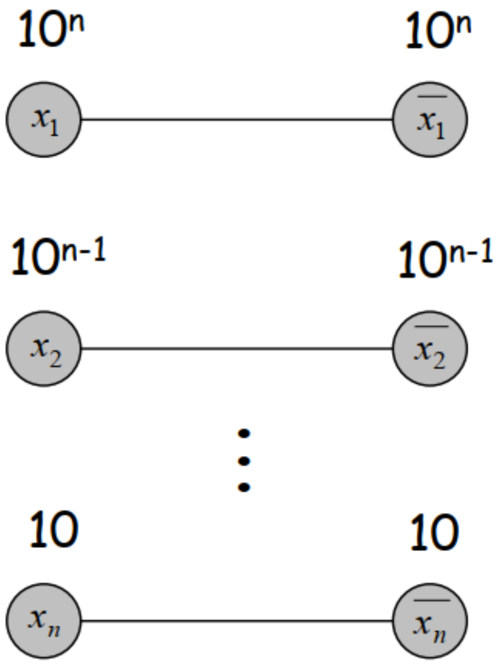
Competitive Facility Location

- **Claim.** COMPETITIVE-FACILITY is PSPACE-complete
- **Pf.**
  - To show that it's complete, we show that QSAT polynomial reduces to it.
  - Given an instance of QSAT, we construct an instance of COMPETITIVE-FACILITY such that player 2 can force a win iff QSAT formula is **false**

Competitive Facility Location

Assume n is odd

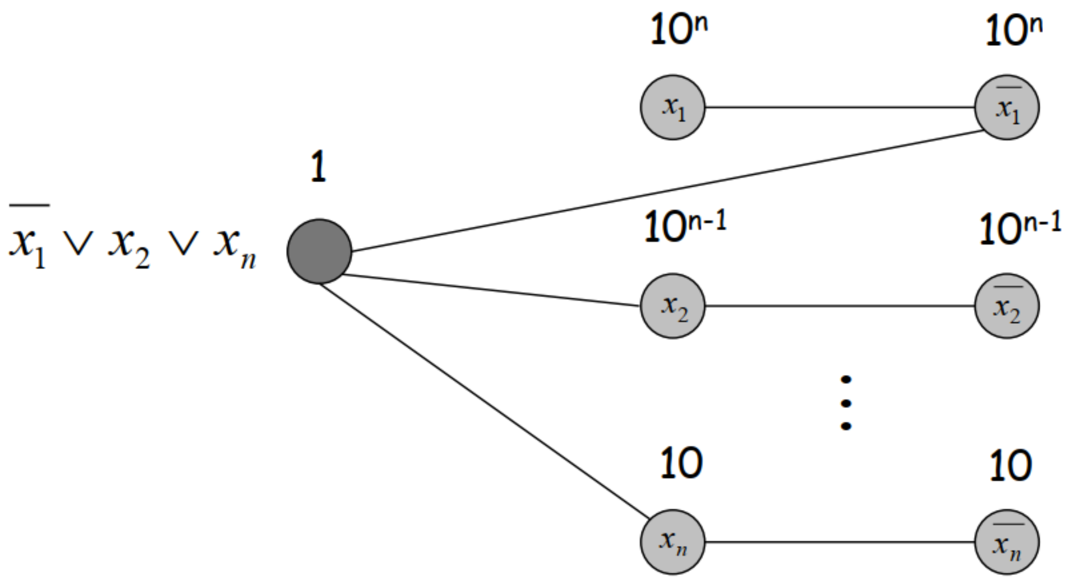
- **Construction.** Given instance $\Phi(X_1, \ldots, X_n) = C_1 \wedge C_2 \wedge \ldots C_k$ of QSAT
  - Include a node for each literal and its negation and connect them
    - At most one of X_i and its negation can be chosen
  - Choose a large constant c (e.g., $c \geq k+2$), and put weight c^{n-i+1} on literal x_i and its negation;
  - Set $B = c^{n-1} + c^{n-3} + \ldots + c^4 + c^2 + 1$
    - This ensures variables are selected in order $x_1, x_2, \ldots, x_n$
  - As is, player 2 will lose by 1 unit: $c^{n-1} + c^{n-3} + \ldots + c^4 + c^2$



Competitive Facility Location

- **Construction.** Given instance $\Phi(X_1, \ldots, X_n) = C_1 \wedge C_2 \wedge \ldots C_k$ of QSAT
  - Given player 2 one last move on which she can try to win
  - For each clause C_j, add node with value 1 and an edge to each of its literals
  - Player 2 can make last move iff truth assignment defined alternately by the players failed to satisfy some clause, i.e.,

$$\forall x_1\ \exists x_2\ \forall x_3\ \exists x_4 \ldots \exists x_{n-1}\ \forall x_n\ \neg\Phi(x_1, \ldots, x_n)$$
$$\Leftrightarrow\ \neg\ \exists x_1\ \forall x_2\ \exists x_3\ \forall x_4 \ldots\ \forall x_{n-1}\ \exists x_n\ \Phi(x_1, \ldots, x_n)$$



Coping with NP-Completeness

- **Q.** Suppose I need to solve an NP-complete problem. What should I do?
- **A.** Theory says you're unlikely to find poly-time algorithm.

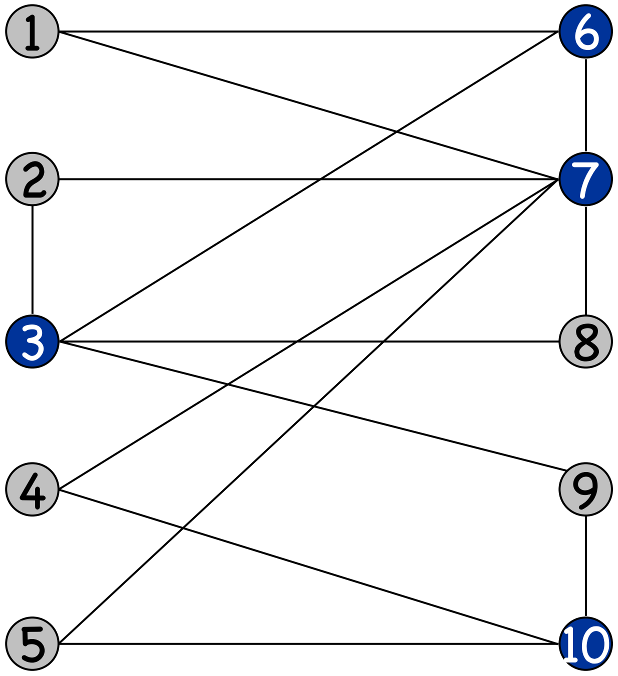
- **Must sacrifice at least one of three desired features.**
  - Solve problem in polynomial time.
  - Solve problem to optimality.
  - Solve arbitrary instances of the problem.

- **This lecture.** Solve some special cases of NP-complete problems that arise in practice.

Finding Small Vertex Covers

Vertex Cover

- **Problem definition:** Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge (u, v), either $u \in S$, or $v \in S$, or both



$$k = 4$$
$$S = \{3, 6, 7, 10\}$$

Finding Small Vertex Covers

- **Q.** What if k is small?
- **Brute force.** $O(kn^{k+1})$.
  - Try all $C(n,k) = O(n^k)$ subsets of size k.
  - Takes $O(kn)$ time to check whether a subset is a vertex cover.
- **Goal.** Limit exponential dependency on k, e.g., to $O(2^k n)$.
- **Ex.** $n = 1{,}000,\ k = 10$.
  - **Brute force:** $kn^{k+1} = 10^{34} \Rightarrow$ infeasible.
  - **Better:** $2^k n = 10^6 \Rightarrow$ feasible.

- **Remark.** If k is a constant, then the algorithm is poly-time
- If k is a small constant, then it's also practical
- **Parameterized complexity.** FPT (fixed parameter tractable) with respect to some parameter k is the class of problems solvable in time $f(k) \cdot \mathrm{poly}(n)$

Finding Small Vertex Covers

- **Claim.** Let (u, v) be an edge of G. G has a vertex cover of size $\leq k$ iff at least one of $G - \{u\}$ and $G - \{v\}$ has a vertex cover of size $\leq k - 1$.

delete v and all incident edges

- **Pf.** $\Rightarrow$
  - Suppose G has a vertex cover S of size $\leq k$.
  - S contains either u or v (or both). Assume it contains u.
  - $S - \{u\}$ is a vertex cover of $G - \{u\}$.

- **Pf.** $\Leftarrow$
  - Suppose S is a vertex cover of $G - \{u\}$ of size $\leq k - 1$.
  - Then $S \cup \{u\}$ is a vertex cover of G. ▪

- **Claim.** If G has a vertex cover of size k, it has $\leq$ k(n-1) edges
- **Pf.** Each vertex covers at most n – 1 edges

Finding Small Vertex Covers: Algorithm

- **Claim.** The following algorithm determines if G has a vertex cover of size $\leq k$ in $O(2^k n)$ time

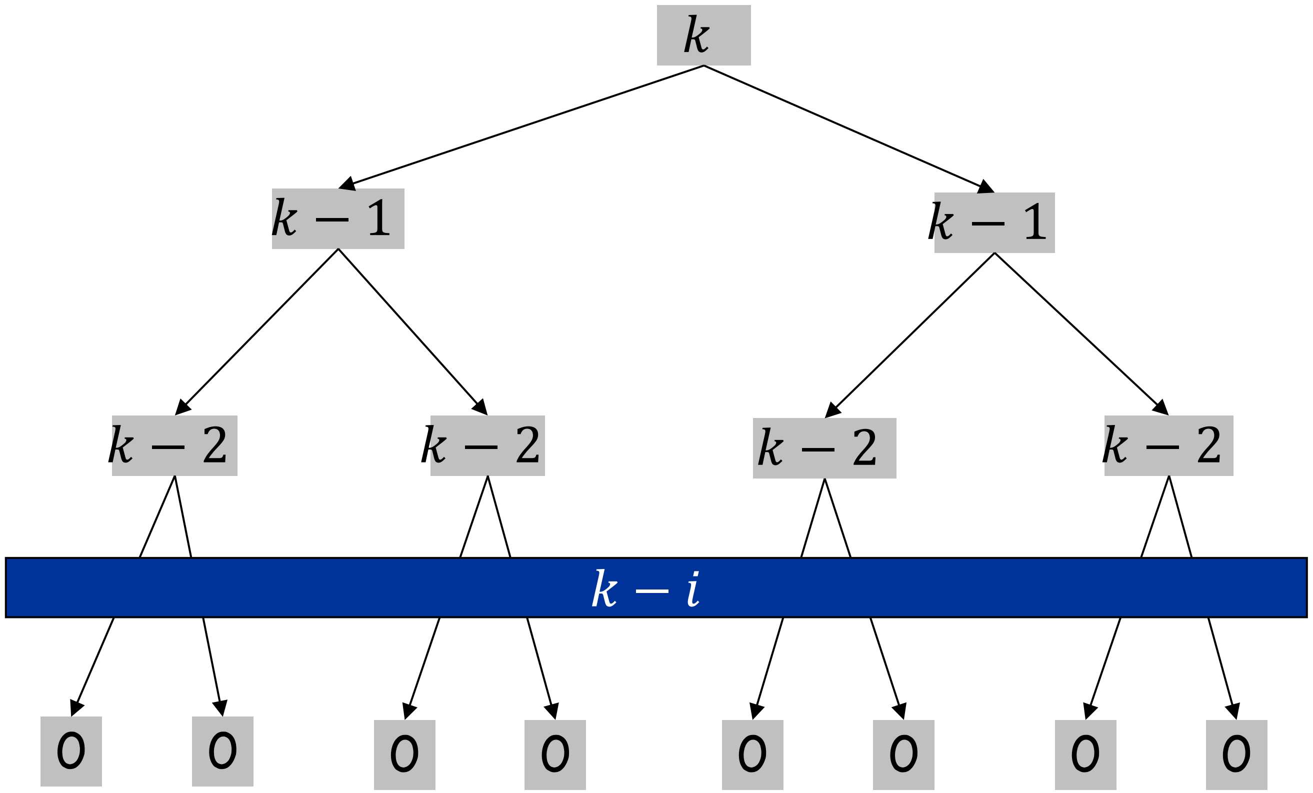
```
Vertex-Cover(G, k) {
   if (G contains no edges)    return true
   if (G contains ≥ kn edges) return false

   let (u, v) be any edge of G
   a = Vertex-Cover(G - {u}, k-1)
   b = Vertex-Cover(G - {v}, k-1)
   return a or b
}
```

- **Pf.**
  - Correctness follows previous two claims.
  - There are $\leq 2^{k+1}$ nodes in the recursion tree; each invocation takes $O(n)$ time. ▪

Finding Small Vertex Covers: Recursion Tree

$$T(n,k) \le \begin{cases} 1, & \text{if } k = 0, \\ 2T(n,k-1) + cn, & \text{if } k > 0. \end{cases} \quad \Rightarrow \quad T(n,k) \le 2^{k+1} cn$$



Solving NP-hard Problems on Trees

Independent set on trees

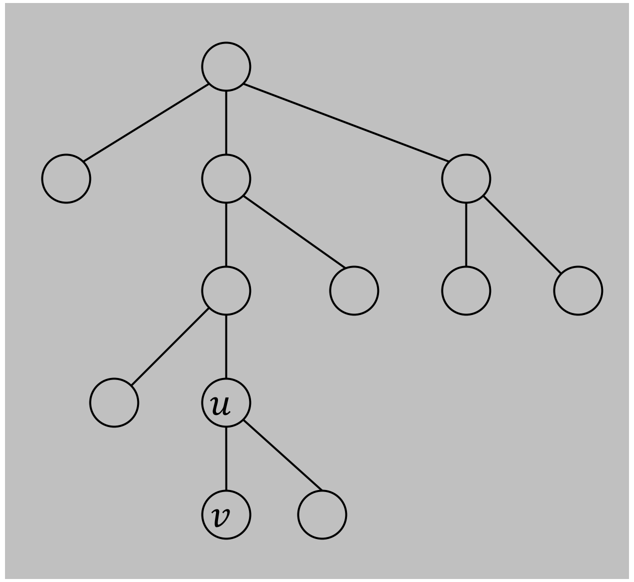
上海科技大学 ShanghaiTech University

- **Problem definition.** Given a **tree**, find a maximum cardinality subset of nodes such that no two share an edge.

- **Def.** A leaf is a node with degree 1.

- **Key observation.** If v is a leaf, there exists a maximum size independent set containing v.

- **Pf.** (exchange argument)
  - Consider a max size independent set S.
  - If $v \in S$, we're done.
  - If $u \notin S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow S$ not maximum.
  - If $u \in S$ and $v \notin S$, then $S \cup \{v\} - \{u\}$ is independent. ▪



立志 成才 报国 裕民

Independent Set on Trees: Greedy Algorithm

- **Theorem.** The following greedy algorithm finds a maximum cardinality independent set in forests (and hence trees)

```
Independent-Set-In-A-Forest(F):
   S ← ∅
   while F has at least one edge do
      Let e = (u,v) be an edge such that v is a leaf
      Add v to S
      Delete nodes u and v from F, and all edges
         incident to them.
   Add all remaining vertices to S
   return S
```

- **Pf.** Correctness follows from the previous key observation. ▪

- **Remark.** Can implement in O(n) time by considering nodes in postorder

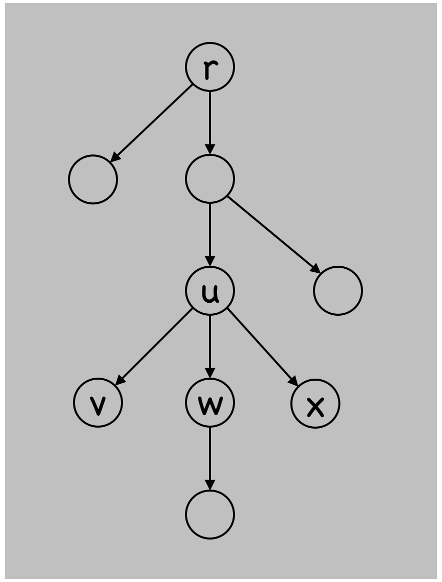
Weighted Independent Set on Trees

上海科技大学 ShanghaiTech University

- **Problem definition.** Given a tree and node weights $w_v > 0$, find an independent set S that maximizes $\sum_{v \in S} w_v$.
- **Note:** Greedy doesn't work anymore.

- **Observation.** If (u, v) is an edge such that v is a leaf node, then either OPT includes u, or it includes all leaf nodes incident to u.

- **Dynamic programming solution.** Root tree at some node, say r.
  - $OPT_{in}(u)$ = max weight independent set of subtree rooted at u, containing u.
  - $OPT_{out}(u)$ = max weight independent set of subtree rooted at u, not containing u.



children(u) = { v, w, x }

- $OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$

- $OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max\{OPT_{in}(v), OPT_{out}(v)\}$

立志成才报国裕民

Independent Set on Trees: DP Algorithm

- **Theorem.** The dynamic programming algorithm finds a maximum weighted independent set in trees in $O(n)$ time.

can also find independent set itself (not just value)

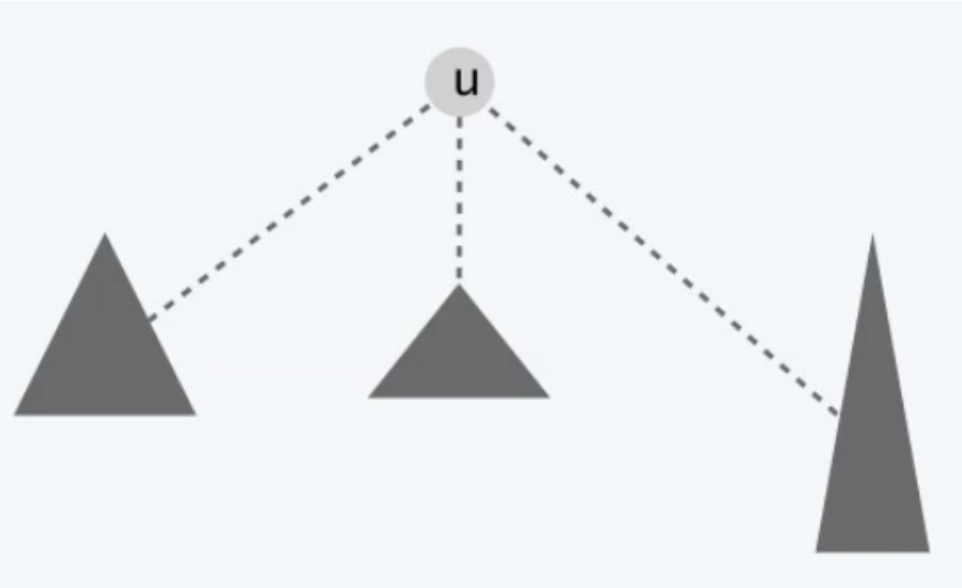
```
Weighted-Independent-Set-In-A-Tree(T):
   Root the tree at a node r
   for each node u of T in postorder
      if u is a leaf
         M_in[u] ← w_u
         M_out[u] ← 0
      else
         M_in[u] ← Σ_{v∈children(u)} M_out[v] + w_u
         M_out[u] ← Σ_{v∈chilren(u)} max(M_out[v], M_in[v])
   return max(M_in[r], M_out[r])
```

ensures a node is visited after all its children

- **Pf.** Takes O(n) time since we visit nodes in postorder and examine each edge exactly once. ▪

Context

- **Independent set on trees.** This structured special case is tractable because we can find a node that **breaks the communication** among the subproblems in different subtrees



- **Graphs of bounded tree width.** Elegant generalization of trees that:
  - Captures a rich class of graphs that arise in practice
  - Enables decomposition into independent pieces

Circular Arc Covering

Wavelength-Division Multiplexing

Background. More than one communication links can share the same portion of a fiber optic cable, provided they are transmitted using different wavelengths.

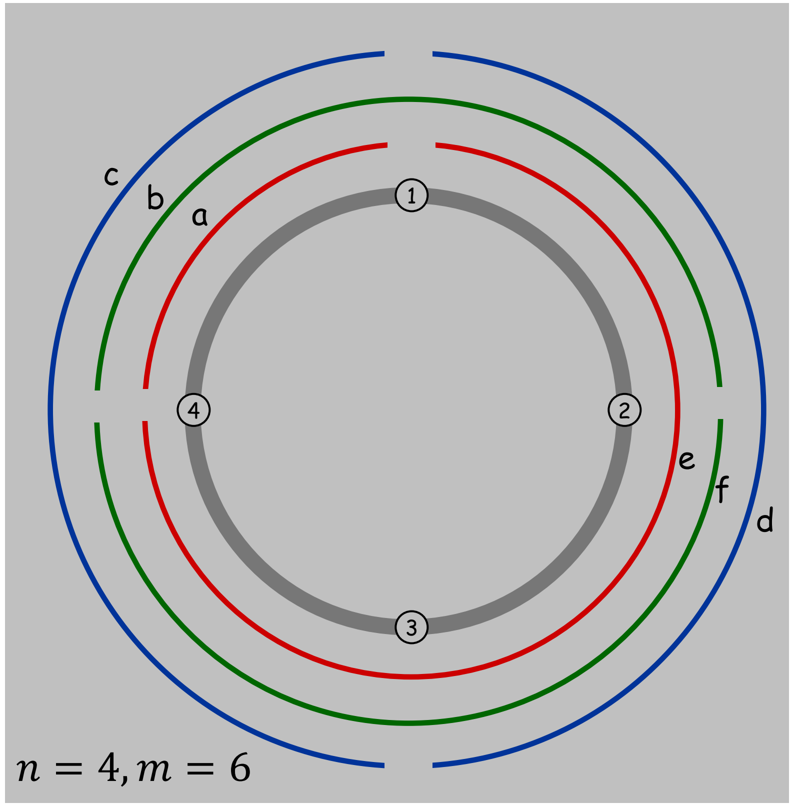
Problem definition. Given a graph $G = (V, E)$, and m paths $p_1, p_2, \ldots, p_m$, assign each path a color so that any two paths that share an edge must have different colors. The goal is to use as few colors as possible.

Ring topology. Consider the special case is when graph is a cycle on n nodes.

Bad news. NP-complete, even on rings.

Brute force. Can determine if k colors suffice in $O(k^m)$ time by trying all k-colorings.

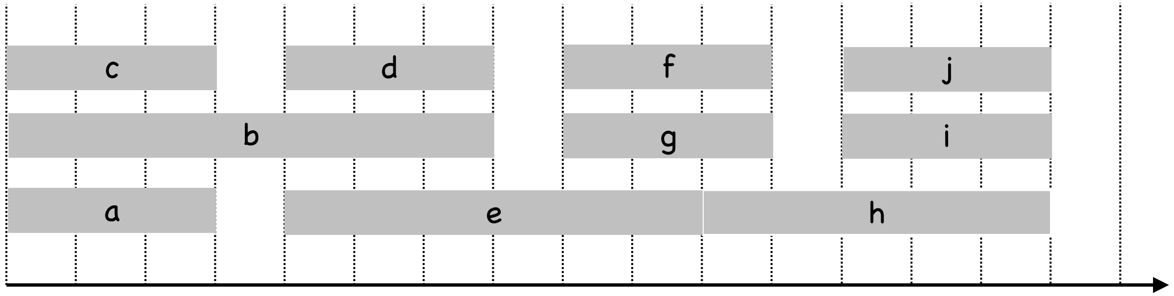
Goal. $f(k) \cdot \text{poly}(m, n)$ time.



Review: Interval Coloring

Interval coloring. Greedy algorithm finds coloring such that number of colors equals depth of schedule.

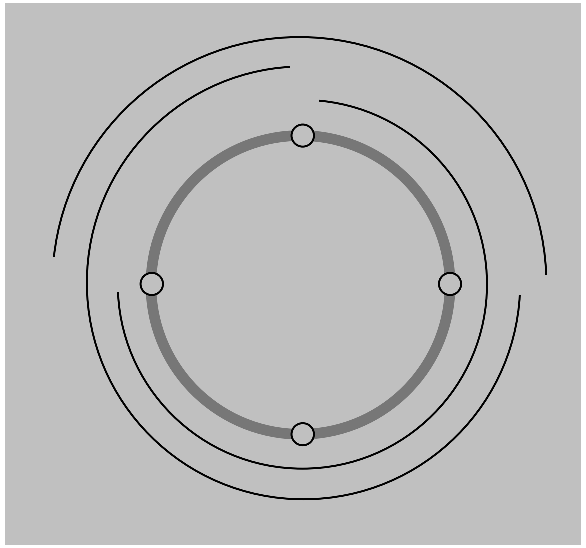
maximum number of intervals at one location

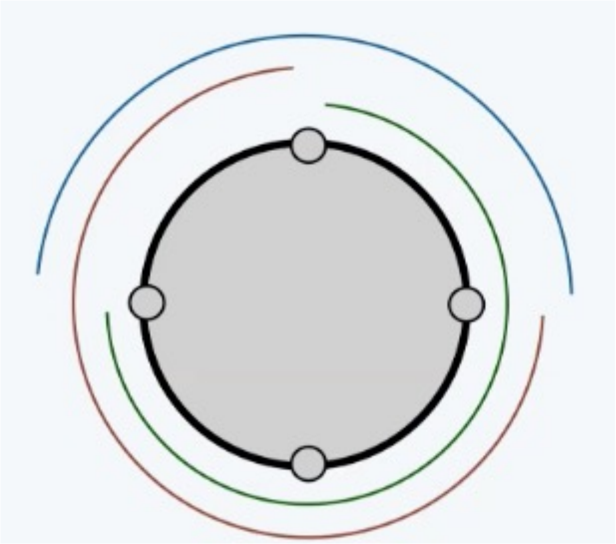


Circular arc coloring.

- Weak duality: Number of colors ≥ depth.
- Strong duality does not hold
- But the two may not be equal.

max depth = 2
min colors = 3

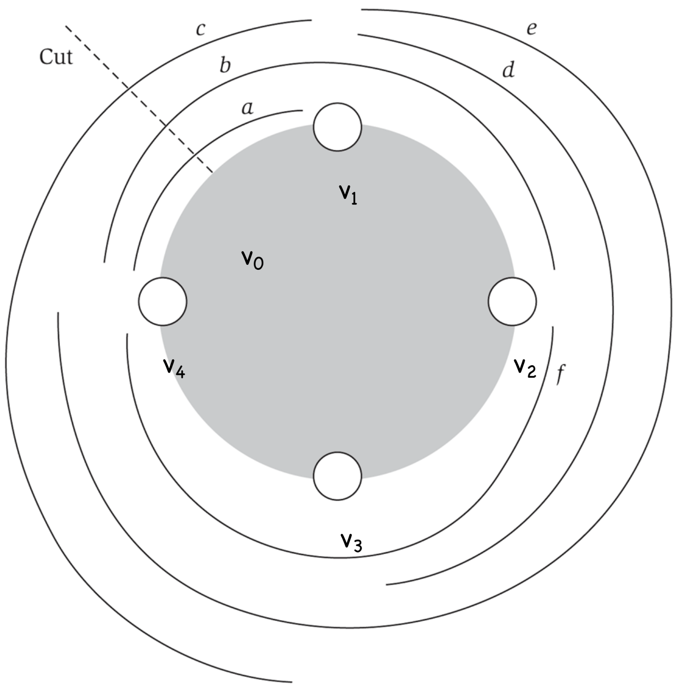




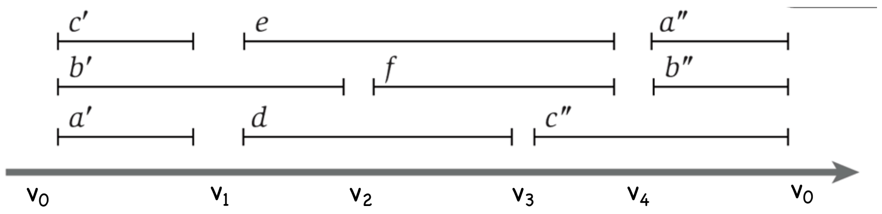
(Almost) Transforming Circular Arc Coloring to Interval Coloring

Circular arc coloring. Given a set of m arcs with depth $d \leq k$, can the arcs be colored with k colors?

Equivalent problem. Cut the ring between nodes v_1 and v_n. The arcs can be colored with k colors iff the intervals can be colored with k colors in such a way that those "sliced" arcs have the same color.



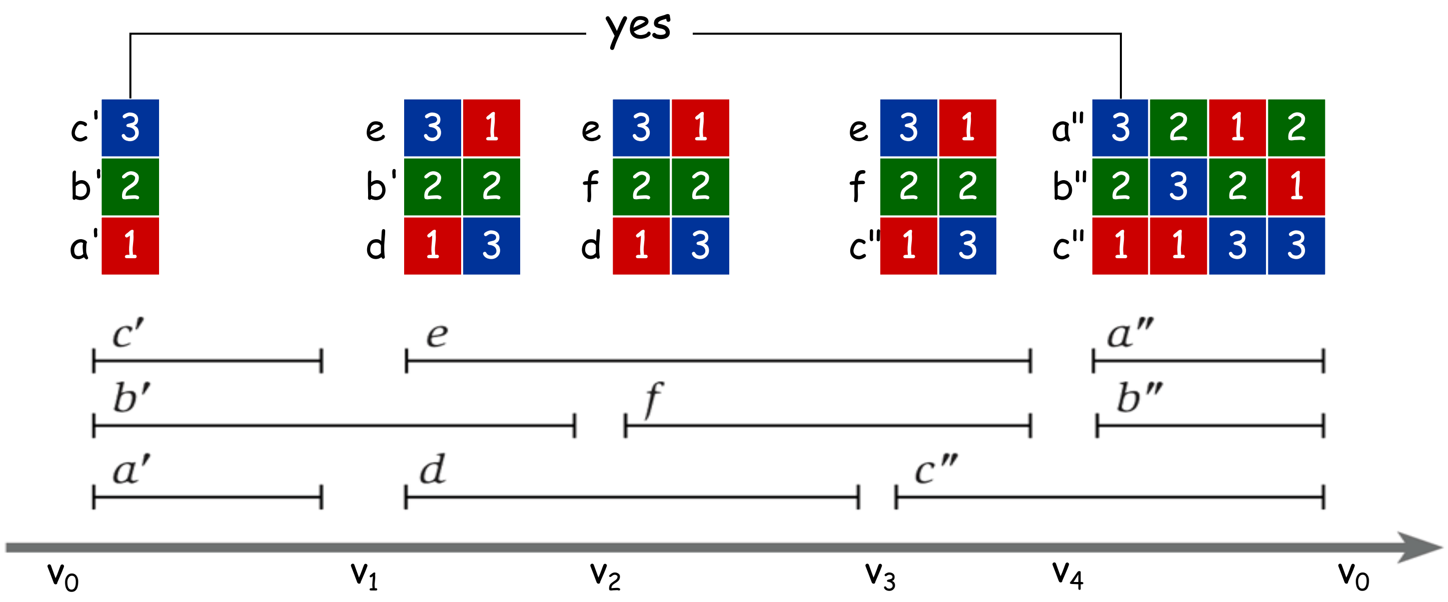
colors of a', b', and c' must correspond
to colors of a'', b'', and c''



Circular Arc Coloring: Dynamic Programming Algorithm

Dynamic programming algorithm.

- Assign distinct color to each interval which begins at cut node v_0.
- At each node v_i, some intervals may finish, and others may begin.
- Enumerate all k-colorings of the intervals through v_i that are consistent with the colorings of the intervals through v_{i-1}.
- The arcs are k-colorable iff some coloring of intervals ending at cut node v_0 is consistent with original coloring of the same intervals.



Circular Arc Coloring: Running Time

Running time. $O(k! \cdot n)$

- n phases of the algorithm.
- Bottleneck in each phase is enumerating all consistent colorings.
- There are at most k intervals through v_i, so there are at most $k!$ colorings to consider.

Remark. This is $\text{poly}(n)$ time if $k = O(\log n \,/\, \log\log n)$

This algorithm is practical for small values of k (say k = 10) even if the number of nodes n (or paths) is large

Next Time:
Midterm
Review and Recitation