

Algorithms and Complexity

Lecture 7

Dynamic Programming and
Inclusion Exclusion

Overview of Today

- Dynamic Programming
 - Subset Sum / Knapsack approximation
 - Coloring
 - Traveling Salesman Problem
 - Steiner Tree
 - Weighted Independent Set on trees
- Inclusion Exclusion
 - Hamiltonian Cycle
 - k-coloring

Dynamic Programming (DP)

- The method works with a big table of data that has to be stored in memory.
- A relatively easy procedure computes new table entries using already computed table entries.
- This easy procedure is often so easy that we just write it down as a single formula, obtaining a *recurrence*.
- Other interpretation:
 - cleverly define subproblems,
 - solve easier subproblems first; store solutions
 - use solutions of easier subproblem to solve harder ones
 - the original problem also is a subproblem
 - “DFS/BFS with memoization”

Sawmill

- A sawmill resaws long planks (of length n) into shorter planks, and can sell a plank of length i for a fixed profit:

Length m	i	1	2	3	4	5	6	7	8	9	10
Profit €	p_i	1	5	8	9	10	17	17	20	24	30

- What is the most profit that can be made by resawing a plank of length n ?
- E.g., a plank of length 5
 - Is worth €10 uncut
 - Worth €10 cut into 1-4
 - Worth €13 cut into 2-3
 - Worth €8 cut into 1-1-1-2

Top Choice & Subproblem

- To design a DP algorithm, we formulate a local decision what we make about the solution first: *top choice*
- After making the top choice, a *subproblem* remains to be solved
- The original problem is also an instance of the subproblem

Top Choice & Subproblem

- To design a DP algorithm, we formulate a local decision what we make about the solution first: *top choice*
- After making the top choice, a *subproblem* remains to be solved

- Top choice: length of the first plank we cut
- Subproblem: maximize the profit of the remaining wood

Recurrence

- Let $R(n)$ denote the profit (revenue) that can be made by resawing a plank of length n
- We have that

$$R(n) = \begin{cases} 0, & n = 0 \\ & n > 0 \end{cases}$$

Recurrence

- Let $R(n)$ denote the profit (revenue) that can be made by resawing a plank of length n
- We have that

$$R(n) = \begin{cases} 0, & n = 0 \\ \max_{1 \leq i \leq n} p_i + R(n - i), & n > 0 \end{cases}$$

- We can compute $R(i)$ using $R(i - 1), R(i - 2), \dots, R(0)$

Subset Sum

- Given integers w_1, \dots, w_n, t find $X \subseteq \{1, \dots, n\}$ such that $\sum_{i \in X} w_i = t$.
- For $i = 1, \dots, n$ and $j = 1, \dots, t$ define $A[i, j]$ to be true iff $\exists X \subseteq \{1, \dots, i\}$ s.t. $\sum_{i \in X} w_i = j$. Want: $A[n, t]$
- Instance $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $t = 50$
- True/false: $A[0,0], A[2,0], A[3,5], A[6,12], A[4,20], A[12,50]$?

$$A[i, j] = \begin{cases} \text{false} & \text{if } i = 0, j > 0, \\ \text{true} & \text{if } i = 0, j = 0, \\ A[i - 1, j] & \text{if } i > 0, j < w_i, \\ A[i - 1, j] \vee A[i - 1, j - w_i] & \text{otherwise.} \end{cases}$$

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```


	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```


	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

- 1: $A[0, 0] \leftarrow \text{true}$
- 2: **for** $j = 1$ **to** t **do**
- 3: $A[0, j] \leftarrow \text{false}$
- 4: **for** $i = 1$ **to** n **do**
- 5: **for** $j = 1$ **to** t **do**
- 6: **if** $j < w_i$ **then**
- 7: $A[i, j] \leftarrow A[i - 1, j]$
- 8: **else**
- 9: $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$
- 10: **return** $A[n, t]$

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

```

1:  $A[0, 0] \leftarrow \text{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \text{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

- 1: $A[0, 0] \leftarrow \text{true}$
- 2: **for** $j = 1$ **to** t **do**
- 3: $A[0, j] \leftarrow \text{false}$
- 4: **for** $i = 1$ **to** n **do**
- 5: **for** $j = 1$ **to** t **do**
- 6: **if** $j < w_i$ **then**
- 7: $A[i, j] \leftarrow A[i - 1, j]$
- 8: **else**
- 9: $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$
- 10: **return** $A[n, t]$

	0	1	...	n
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
...				
t				?

Algorithm $\text{sss}(w_1, \dots, w_n, t)$

Output: Whether there exists $X \subseteq \{1, \dots, n\}$ with $\sum_{e \in X} w_e = t$.

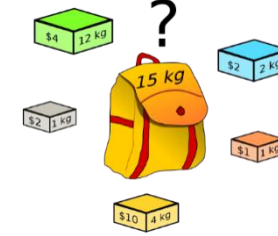
```

1:  $A[0, 0] \leftarrow \mathbf{true}$ 
2: for  $j = 1$  to  $t$  do
3:    $A[0, j] \leftarrow \mathbf{false}$ 
4: for  $i = 1$  to  $n$  do
5:   for  $j = 1$  to  $t$  do
6:     if  $j < w_i$  then
7:        $A[i, j] \leftarrow A[i - 1, j]$ 
8:     else
9:        $A[i, j] \leftarrow A[i - 1, j] \vee A[i - 1, j - w_i]$ 
10: return  $A[n, t]$ 

```

- $O(nt)$ time
- Not polynomial: t can be exponentially large since it is represented in binary!

Knapsack



- Given $w_1, \dots, w_n, b, v_1, \dots, v_n, t$,
find $X \subseteq \{1, \dots, n\}$ s.t. $\sum_{e \in X} v_e \geq t, \sum_{e \in X} w_e \leq b$.

- For $i = 0, \dots, n$ and $j = 0, \dots, V := \sum_e v_e$ let

$$A[i, j] = \min\{\sum_{e \in X} w_e : X \subseteq \{1, \dots, i\} \wedge \sum_{e \in X} v_e \geq j\}$$

- $A[i, j]$ is minimum weight of a subset realizing a given value

$$A[i, j] = \begin{cases} 0 & \text{if } i = 0, j \leq 0, \\ \infty & \text{if } i = 0, j > 0, \\ \min\{A[i - 1, j], A[i - 1, \max\{j - v_i, 0\}]\} & \text{if } i > 0. \end{cases}$$

- Can *find* in $O(nV)$ time set X maximizing

$$\sum_{e \in X} v_e \text{ s.t. } \sum_{e \in X} w_e \leq b:$$

- Let v be largest value s.t. $T[n, v] \leq b$
- Backtrack from $T[n, v]$ (Exercise 6.1)

Approximation scheme for Knapsack

- This DP is again pseudopolynomial
- We'll see a $O(n^3/\epsilon)$ time $(1 - \epsilon)$ -approximation for the maximization version of knapsack
- I.e., find in time $O(n^3/\epsilon)$ a set A such that:
 - $\sum_{i \in A} w_i \leq b$, and
 - for every X satisfying $\sum_{i \in X} w_i \leq b$
 - $\sum_{i \in A} v_i \geq (1 - \epsilon) \sum_{i \in X} v_i$

Approximation scheme for Knapsack

- Idea:
 1. Can replace instance $w_1, \dots, w_n, b, v_1 \cdot S, \dots, v_n \cdot S$, with $w_1, \dots, w_n, b, v_1, \dots, v_n$
 - Optimum stays the same! Useful if S large!
 2. Round values to multiples of S

Approximation for Knapsack

- L4 runs in time

$$O\left(n \sum_i \frac{\tilde{v}_i}{S}\right) = O\left(n \sum_i \frac{v_i}{S}\right) = O\left(n^2 \left(\frac{v_{\max}}{S}\right)\right) = O\left(\frac{n^3}{\epsilon}\right)$$

- Output A satisfies $\sum_{e \in A} w_e \leq b$
- It remains to compare A with X that reaches OPT

Algorithm apxKnapsack($w_1, \dots, w_n, v_1, \dots, v_n, b, \epsilon$)

assumes $\max_i w_i \leq b$

Output: A s.t. $\sum_{e \in A} w_e \leq b \wedge \sum_{e \in A} v_e \geq (1 - \epsilon)OPT$

$OPT = \max\{\sum_{e \in X} v_e : \sum_{e \in X} w_e \leq b\}$

1: let $v_{\max} = \max_i v_i$, let $S = \lfloor \frac{\epsilon v_{\max}}{n} \rfloor$

2: **for all** $i = 1, \dots, n$ **do**

3: $\tilde{v}_i = S \lfloor \frac{v_i}{S} \rfloor$

4: **return** optKnapsack($w_1, \dots, w_n, \tilde{v}_1/S, \dots, \tilde{v}_n/S, b$)

Approximation for Knapsack

- Suppose that X is optimal solution (value is $\sum_{i \in X} v_i$)
- In rounded instance, value of X is
$$\sum_{i \in X} S \lfloor v_i/S \rfloor \geq \sum_{i \in X} v_i - S$$
- Thus, loss is at most $nS \leq n \frac{\epsilon \cdot v_{max}}{n} \leq \epsilon \cdot v_{max} \leq \epsilon \cdot OPT$
($OPT \geq v_{max}$)
- Value of rounded instance is $\geq OPT - \epsilon \cdot OPT$

Algorithm `apxKnapsack`($w_1, \dots, w_n, v_1, \dots, v_n, b, \epsilon$)

assumes $\max_i w_i \leq b$

Output: A s.t. $\sum_{e \in A} w_e \leq b \wedge \sum_{e \in A} v_e \geq (1 - \epsilon)OPT$

$OPT = \max\{\sum_{e \in X} v_e : \sum_{e \in X} w_e \leq b\}$

1: let $v_{max} = \max_i v_i$, let $S = \lfloor \frac{\epsilon \cdot v_{max}}{n} \rfloor$

2: **for all** $i = 1, \dots, n$ **do**

3: $\tilde{v}_i = S \lfloor \frac{v_i}{S} \rfloor$

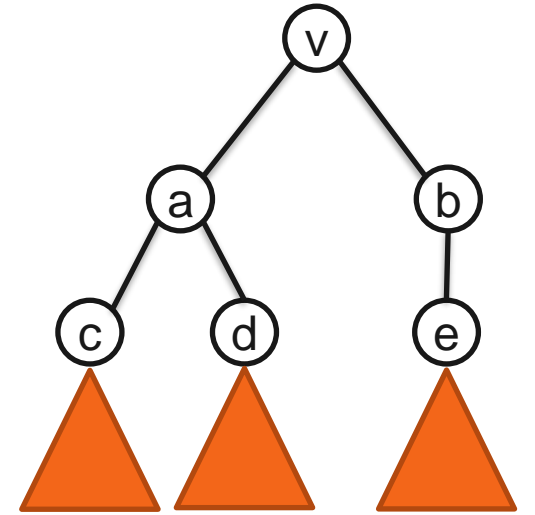
4: **return** `optKnapsack`($w_1, \dots, w_n, \tilde{v}_1/S, \dots, \tilde{v}_n/S, b$)

Weighted Independent Set in Trees

- Many NP-hard problem are easy on trees
- In weighted independent set the vertices have weights and we want to find a max. weight ind. set.
- Let T be a rooted tree, $T[v]$ the subtree rooted at $v \in V$. Define $A[v]$ as the max weight of an ind. set of $T[v]$, then

$$A[v] = \begin{cases} \omega(v), & \text{if } T[v] \text{ is a single node,} \\ \max \left\{ \sum_{c \in \text{ch}(v)} A[c], \omega(v) + \sum_{c_1 \in \text{ch}(v)} \sum_{c_2 \in \text{ch}(c_1)} A[c_2] \right\}, & \text{otherwise.} \end{cases}$$

- And we can compute $A[\text{root}]$ in $O(n)$ time.

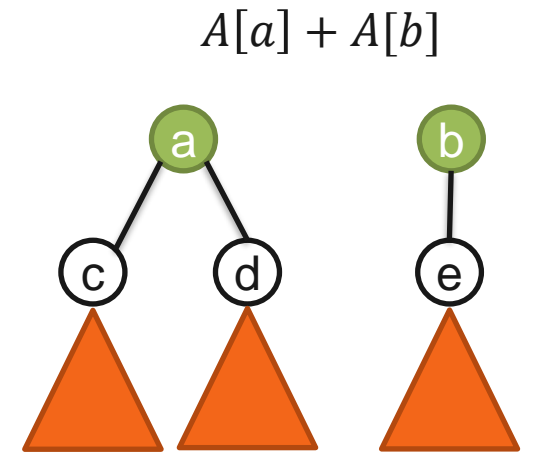


Weighted Independent Set in Trees

- Many NP-hard problem are easy on trees
- In weighted independent set the vertices have weights and we want to find a max. weight ind. set.
- Let T be a rooted tree, $T[v]$ the subtree rooted at $v \in V$. Define $A[v]$ as the max weight of an ind. set of $T[v]$, then

$$A[v] = \begin{cases} \omega(v), & \text{if } T[v] \text{ is a single node,} \\ \max \left\{ \sum_{c \in \text{ch}(v)} A[c], \omega(v) + \sum_{c_1 \in \text{ch}(v)} \sum_{c_2 \in \text{ch}(c_1)} A[c_2] \right\}, & \text{otherwise.} \end{cases}$$

- And we can compute $A[\text{root}]$ in $O(n)$ time.

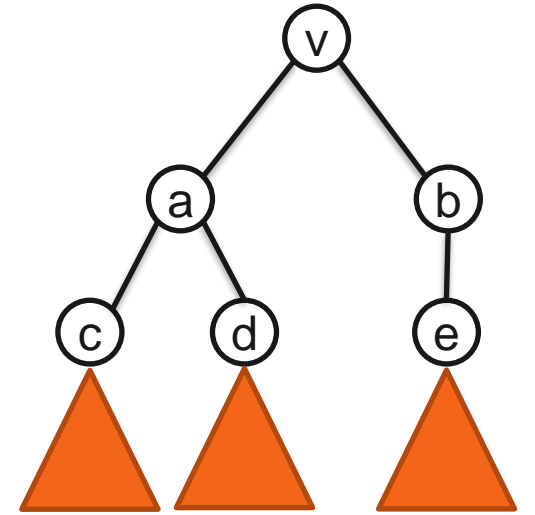


Weighted Independent Set in Trees

- Many NP-hard problem are easy on trees
- In weighted independent set the vertices have weights and we want to find a max. weight ind. set.
- Let T be a rooted tree, $T[v]$ the subtree rooted at $v \in V$. Define $A[v]$ as the max weight of an ind. set of $T[v]$, then

$$A[v] = \begin{cases} \omega(v), & \text{if } T[v] \text{ is a single node,} \\ \max \left\{ \sum_{c \in \text{ch}(v)} A[c], \omega(v) + \sum_{c_1 \in \text{ch}(v)} \sum_{c_2 \in \text{ch}(c_1)} A[c_2] \right\}, & \text{otherwise.} \end{cases}$$

- And we can compute $A[\text{root}]$ in $O(n)$ time.



Weighted Independent Set in Trees

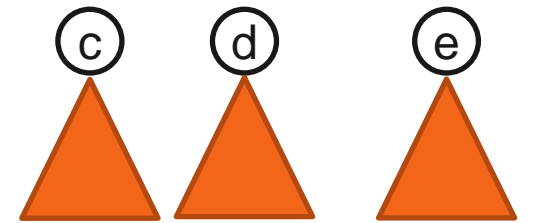
- Many NP-hard problem are easy on trees
- In weighted independent set the vertices have weights and we want to find a max. weight ind. set.
- Let T be a rooted tree, $T[v]$ the subtree rooted at $v \in V$. Define $A[v]$ as the max weight of an ind. set of $T[v]$, then

$$A[v] = \begin{cases} \omega(v), & \text{if } T[v] \text{ is a single node,} \\ \max \left\{ \sum_{c \in \text{ch}(v)} A[c], \omega(v) + \sum_{c_1 \in \text{ch}(v)} \sum_{c_2 \in \text{ch}(c_1)} A[c_2] \right\}, & \text{otherwise.} \end{cases}$$

- And we can compute $A[\text{root}]$ in $O(n)$ time.



$$\omega(v) + A[c] + A[d] + A[e]$$



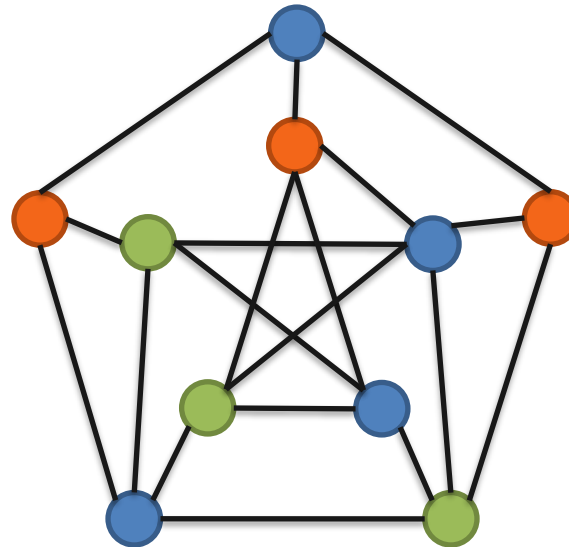
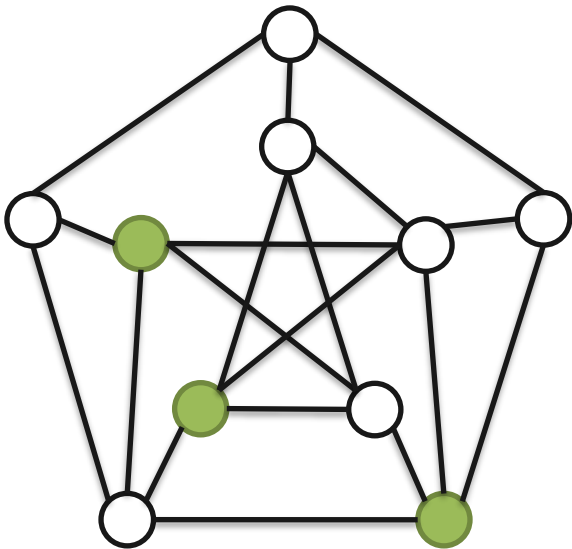
k-coloring in $O^*(3^n)$

- Previous lecture: saw brute force for 3-coloring in $O^*(3^n)$ time; improved to $O^*(2^n)$ time using smarter branching
- Now: k -coloring for arbitrary k in $O^*(3^n)$

k-coloring in $O^*(3^n)$

- Define $A_k[X] = [G[X] \text{ is } k\text{-colorable}]$

$$A_k[X] = \begin{cases} \text{true,} & \text{if } k = 1, X \text{ is an independent set of } G \\ \text{false,} & \text{if } k = 1, X \text{ is not an independent set of } G \\ \bigvee_{Y \subseteq X} A_{k-1}[X \setminus Y] \wedge A_1[Y], & \text{otherwise.} \end{cases}$$



k-coloring in $O^*(3^n)$

- Define $A_k[X] = [G[X] \text{ is } k\text{-colorable}]$

$$A_k[X] = \begin{cases} \text{true,} & \text{if } k = 1, X \text{ is an independent set of } G \\ \text{false,} & \text{if } k = 1, X \text{ is not an independent set of } G \\ \bigvee_{Y \subseteq X} A_{k-1}[X \setminus Y] \wedge A_1[Y], & \text{otherwise.} \end{cases}$$

Algorithm kcolor($G = (V, E), k$)

Output: Whether G is k -colorable.

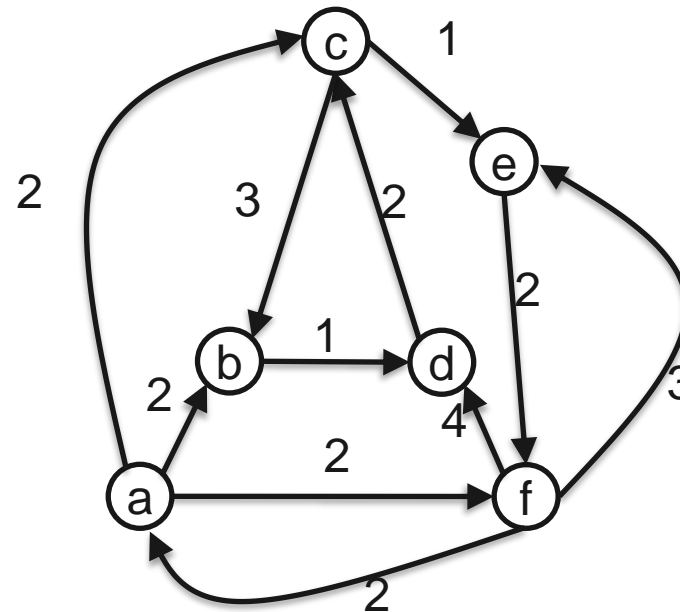
```
1: for  $X \subseteq V$  do
2:   if  $X$  is an independent set of  $G$  then set  $A_1[X] = \text{true}$  else set  $A_1[X] = \text{false}$ 
3: for  $l = 2$  to  $k$  do
4:   for  $X \subseteq V$  do
5:     set  $A_l[X] = \text{false}$ 
6:     for  $Y \subseteq X$  do
7:       if  $A_{l-1}[X \setminus Y] \wedge A_1[Y]$  then set  $A_l[X] = \text{true}$ 
8: return  $A_k[V]$ 
```

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$$

- $\text{poly}(n) \sum_{i=1}^n \binom{n}{i} 2^i = O^*(3^n)$ time by bin. thm.

Traveling Salesman Problem

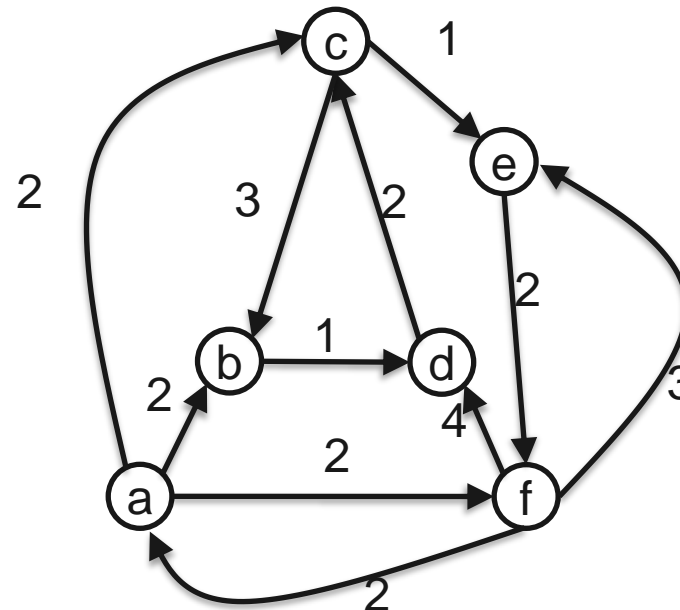
- Notation:
 - $\omega: E \rightarrow \mathbb{N}$, for $X \subseteq E$, $\omega(X) = \sum_{e \in X} \omega(e)$.
 - Path P : sequence of distinct consecutively adjacent vertices.
 - $V[P], E[P]$ are used vertices and edges
 - TSP: given $G = (V, E)$, $\omega: E \rightarrow \mathbb{N}$. Let $s \in V$.
 - Find path P from s to t with $V[P] = V$ minimizing $\omega(E[P]) + \omega(t, s)$



Traveling Salesman Problem

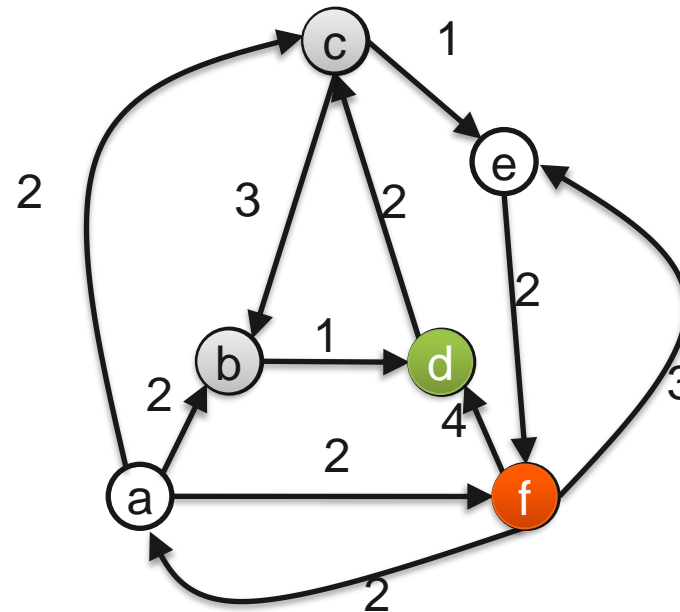
- Notation:
 - $\omega: E \rightarrow \mathbb{N}$, for $X \subseteq E$, $\omega(X) = \sum_{e \in X} \omega(e)$.
 - Path P : sequence of distinct consecutively adjacent vertices.
 - $V[P], E[P]$ are used vertices and edges
 - TSP: given $G = (V, E)$, $\omega: E \rightarrow \mathbb{N}$. Let $s \in V$.
 - Find path P from s to t with $V[P] = V$ minimizing $\omega(E[P]) + \omega(t, s)$

- $P=(a,b,d,c)$
 - is a path from a to c
 - $V[P]=\{a,b,d,c\}$
 - $E[P] =\{ab,bd,dc\}$
 - $\omega(P) = 2+1+2 = 5$



Traveling Salesman Problem

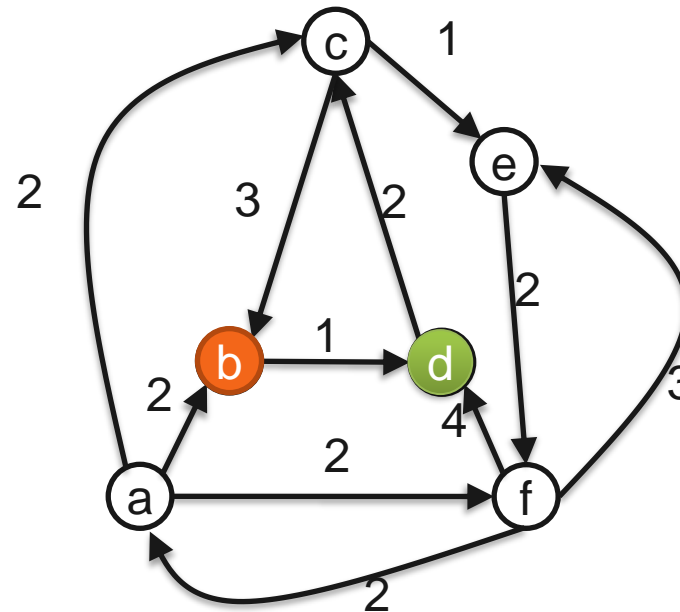
- Notation:
 - $\omega: E \rightarrow \mathbb{N}$, for $X \subseteq E$, $\omega(X) = \sum_{e \in X} \omega(e)$.
 - Path P : sequence of distinct consecutively adjacent vertices.
 - $V[P], E[P]$ are used vertices and edges
 - TSP: given $G = (V, E)$, $\omega: E \rightarrow \mathbb{N}$. Let $s \in V$.
 - Find path P from s to t with $V[P] = V$ minimizing $\omega(E[P]) + \omega(t, s)$
- $s=d$
 - only b, f are candidates for t



Traveling Salesman Problem

- Notation:
 - $\omega: E \rightarrow \mathbb{N}$, for $X \subseteq E$, $\omega(X) = \sum_{e \in X} \omega(e)$.
 - Path P : sequence of distinct consecutively adjacent vertices.
 - $V[P], E[P]$ are used vertices and edges
 - TSP: given $G = (V, E)$, $\omega: E \rightarrow \mathbb{N}$. Let $s \in V$.
 - Find path P from s to t with $V[P] = V$ minimizing $\omega(E[P]) + \omega(t, s)$

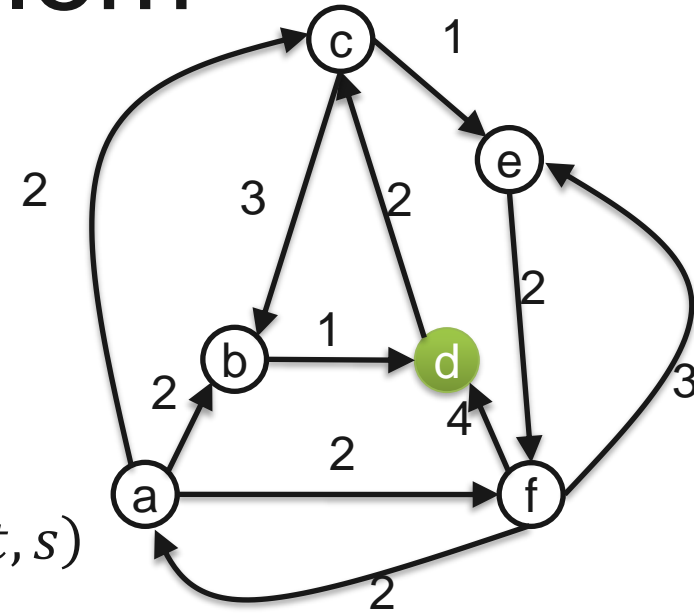
- $s=d$
 - only b, f are candidates for t
 - $P=(d, c, e, f, a, b)$; $\omega(P) = 9$
 - $\omega(E[P]) + \omega(t, s) = 10$



Traveling Salesman Problem

- Notation:

- $\omega: E \rightarrow \mathbb{N}$, for $X \subseteq E$, $\omega(X) = \sum_{e \in X} \omega(e)$.
- Path P : sequence of distinct consecutively adjacent vertices.
- $V[P], E[P]$ are used vertices and edges
- TSP: given $G = (V, E)$, $\omega: E \rightarrow \mathbb{N}$. Let $s \in V$.
- Find path P from s to t with $V[P] = V$ minimizing $\omega(E[P]) + \omega(t, s)$



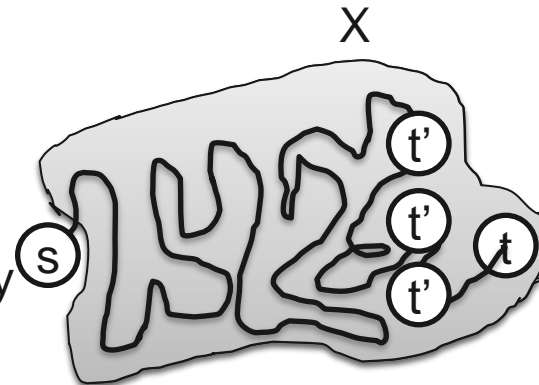
- $A_t[X] = \min\{\omega(E[P]): P \text{ path from } s \text{ to } t, \text{ and } V[P] \setminus s = X\}$
 - Only defined for $t \in X, X \subseteq V \setminus s$
 - $A_b[\{c, b\}] = 5$
 - $A_a[\{c, e, f, a\}] = 7$

Traveling Salesman Problem

- $A_t[X] = \min\{\omega(E[P]): P \text{ path from } s \text{ to } t, \text{ and } V[P] \setminus s = X\}$

$$A_t[X] = \begin{cases} \infty, & \text{if } |X| = 1 \text{ and } (s, t) \notin E, \\ \omega(s, t), & \text{if } |X| = 1 \text{ and } (s, t) \in E, \\ \min_{t' \in N^-(t) \cap X} A_{t'}[X \setminus t] + \omega(t', t), & \text{otherwise.} \end{cases}$$

- If $|X| = 1$, only allowed path is (s, t)
- If $|X| > 1$:
- To formally argue the equality:
 - Inequalities in both direction:
 - LHS \geq RHS: every path is built in this way
 - LHS \leq RHS: adding edge t', t to path obtained from $A_{t'}[X \setminus t]$ gives new path.



Traveling Salesman Problem

- $A_t[X] = \min\{\omega(E[P]): P \text{ path from } s \text{ to } t, \text{ and } V[P] \setminus s = X\}$

$$A_t[X] = \begin{cases} \infty, & \text{if } |X| = 1 \text{ and } (s, t) \notin E, \\ \omega(s, t), & \text{if } |X| = 1 \text{ and } (s, t) \in E, \\ \min_{t' \in N^-(t) \cap X} A_{t'}[X \setminus t] + \omega(t', t), & \text{otherwise.} \end{cases}$$

Algorithm tsp($G = (V, E), \omega$)

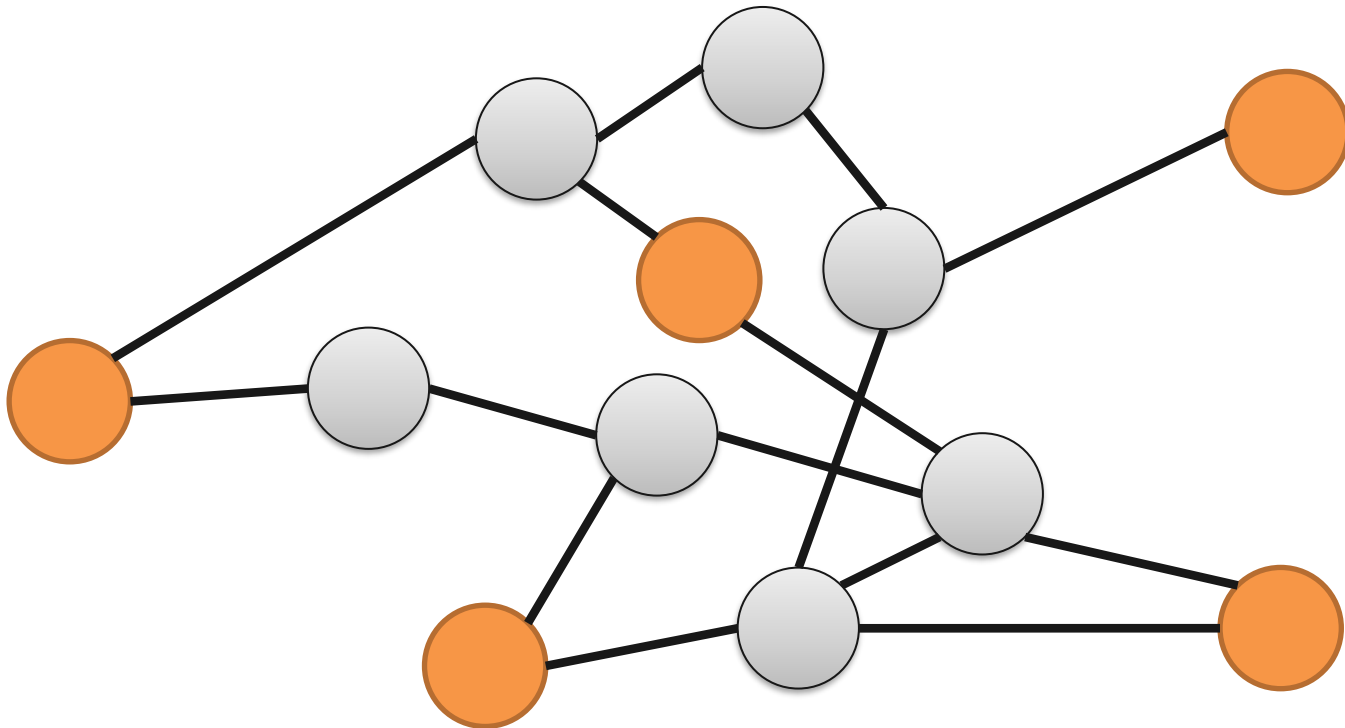
Output: The minimum $\omega(E[C])$ over all cycles of G (e.g., cycles C satisfying $V[C] = V$).

- 1: Pick an arbitrary vertex $s \in V$
- 2: Initiate a table $A_t[X]$ for every $X \subseteq V \setminus s$ and $t \in X$.
- 3: Set $A_t[\{t\}] = \omega(s, t)$ for every $t \in V$, where $\omega(s, t) = \infty$ if $(s, t) \notin E$.
- 4: **for** $i = 2$ to n **do**
- 5: **for** $t \in V \setminus s$ and $X \subseteq V \setminus s$ such that $|X| = i$ and $t \in X$ **do**
- 6: Set $A_t[X] = \min_{t' \in N^-(t) \cap X} A_{t'}[X \setminus t] + \omega(t', t)$.
- 7: **return** $\min_{t \in N^-(V)} A_t[X] + \omega(t, s)$.

- Gives $O^*(2^n)$ time.

Steiner Tree in $O^*(3^k)$ time

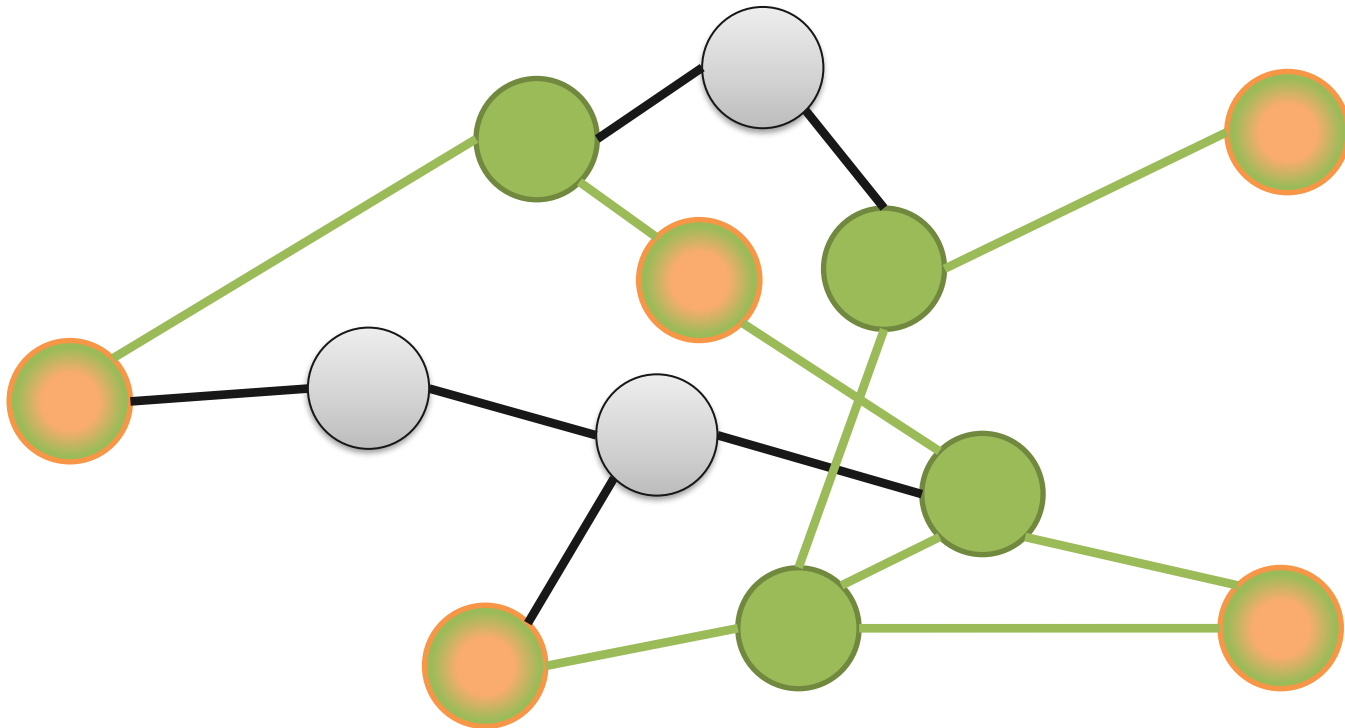
- Given graph $G = (V, E)$ set of terminals $T \subseteq V$, int l
find $T \subseteq S \subseteq V$ with $G[S]$ connected and $|S| \leq l$. $k = |T|$



$l = 9$

Steiner Tree in $O^*(3^k)$ time

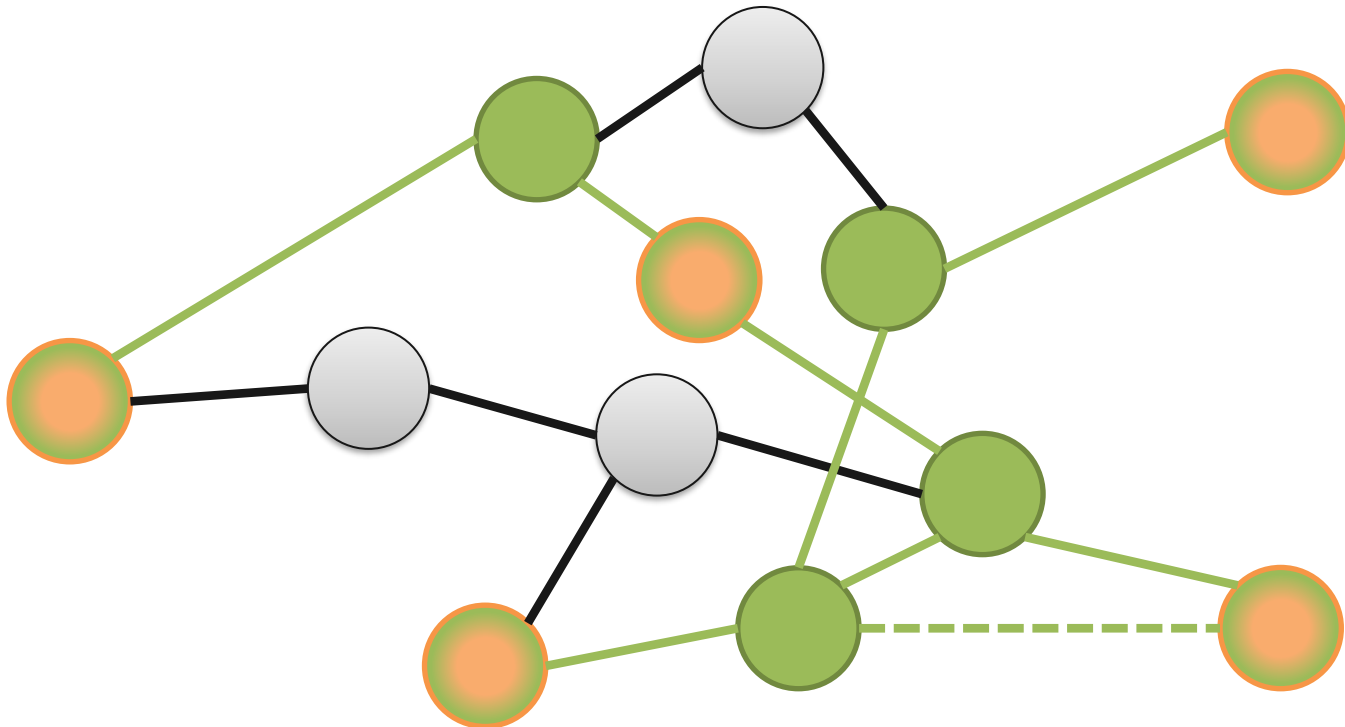
- Given graph $G = (V, E)$ set of terminals $T \subseteq V$, int l
find $T \subseteq S \subseteq V$ with $G[S]$ connected and $|S| \leq l$. $k = |T|$



$l = 9$

Steiner Tree in $O^*(3^k)$ time

- Given graph $G = (V, E)$ set of terminals $T \subseteq V$, int l find $T \subseteq S \subseteq V$ with $G[S]$ connected and $|S| \leq l$. $k = |T|$
- Why called Steiner *tree*? Equivalent to finding subtree (S, E') (e.g. a tree with $S \subseteq V, E' \subseteq E,$) of G with $|E'| = l - 1$



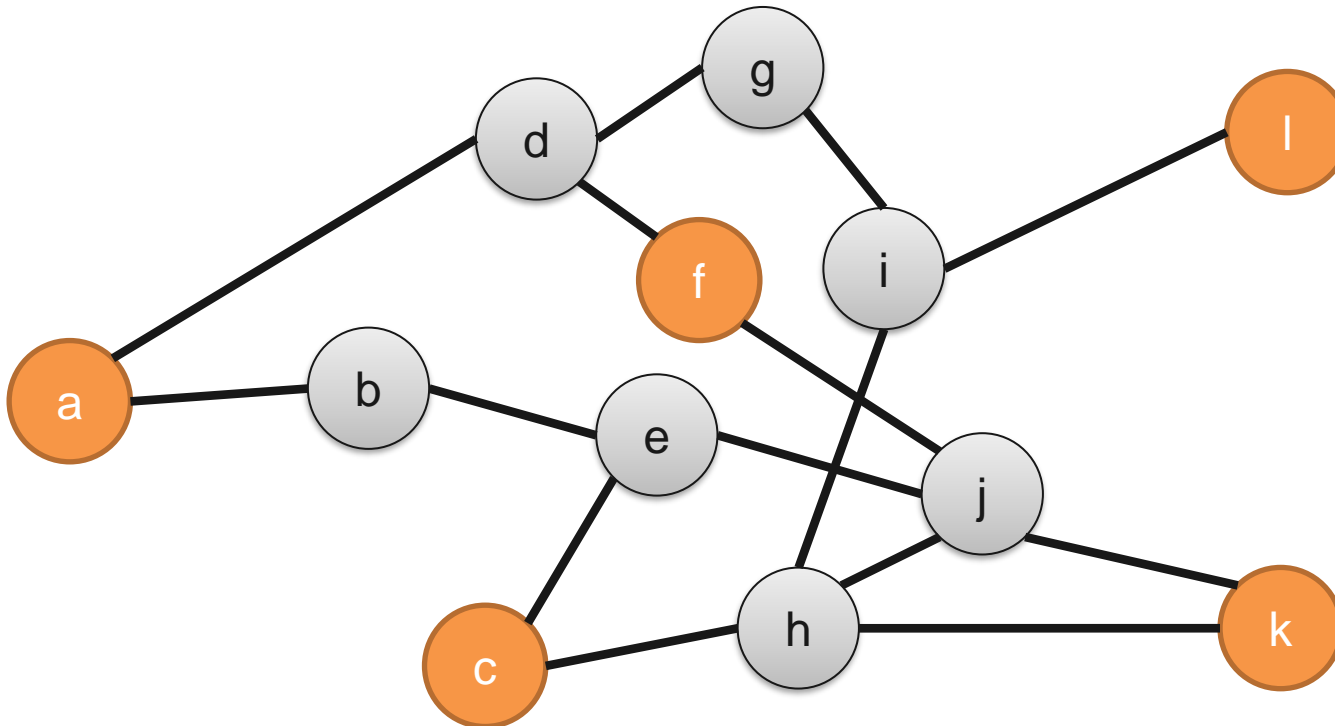
$$l = 9$$

Steiner Tree in $O^*(3^k)$ time

For vertex r , integer j and $X \subseteq T \setminus r$ define:

$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } \exists S \subseteq V : |S| \leq j \wedge X \cup r \subseteq S \wedge G[S] \text{ is connected.} \\ \text{false} & \text{otherwise.} \end{cases}$$

- $A_{d,2}[\{f\}], A_{g,4}[\{k\}], A_{c,7}[\{f, l, k\}]$



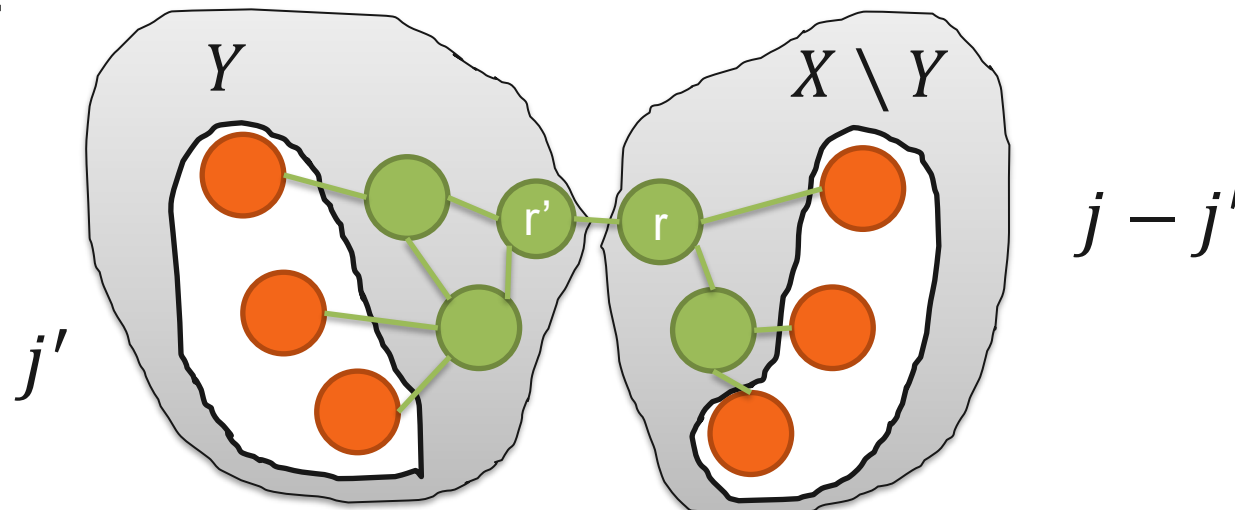
Steiner Tree in $O^*(3^k)$ time

For vertex r , integer j and $X \subseteq T \setminus r$ define:

$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } \exists S \subseteq V : |S| \leq j \wedge X \cup r \subseteq S \wedge G[S] \text{ is connected.} \\ \text{false} & \text{otherwise.} \end{cases}$$

$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } j \geq 1, X = \emptyset \\ \text{false} & \text{if } j \leq 1 \\ \bigvee_{Y \subseteq X} \bigvee_{r' \in N(r)} \bigvee_{j'=1}^j A_{r',j'}[Y \setminus r'] \wedge A_{r,j-j'}[X \setminus Y] & \text{otherwise.} \end{cases}$$

- Intuitively:



Steiner Tree in $O^*(3^k)$ time

For vertex r , integer j and $X \subseteq T \setminus r$ define:

$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } \exists S \subseteq V : |S| \leq j \wedge X \cup r \subseteq S \wedge G[S] \text{ is connected.} \\ \text{false} & \text{otherwise.} \end{cases}$$

$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } j \geq 1, X = \emptyset \\ \text{false} & \text{if } j \leq 1 \\ \bigvee_{Y \subseteq X} \bigvee_{r' \in N(r)} \bigvee_{j'=1}^j A_{r',j'}[Y \setminus r'] \wedge A_{r,j-j'}[X \setminus Y] & \text{otherwise.} \end{cases}$$

- The base cases are easy
- For the third case again consider two directions:
 - \rightarrow : $A_{r,j}[X]$ implies some S .
 - S contains r and neighbor r' since in third case
 - Can find S_1, S_2 and connecting Y to r' and $X \setminus Y$ to r
 - \leftarrow : $A_{r',j'}[X], A_{r,j-j'}[X \setminus Y]$ imply some sets S_1, S_2 ,
 - $S = S_1 \cup S_2$ implies $A_{r,j}[X]$

Steiner Tree in $O^*(3^k)$ time

For vertex r , integer j and $X \subseteq T \setminus r$ define:

$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } \exists S \subseteq V : |S| \leq j \wedge X \cup r \subseteq S \wedge G[S] \text{ is connected.} \\ \text{false} & \text{otherwise.} \end{cases}$$

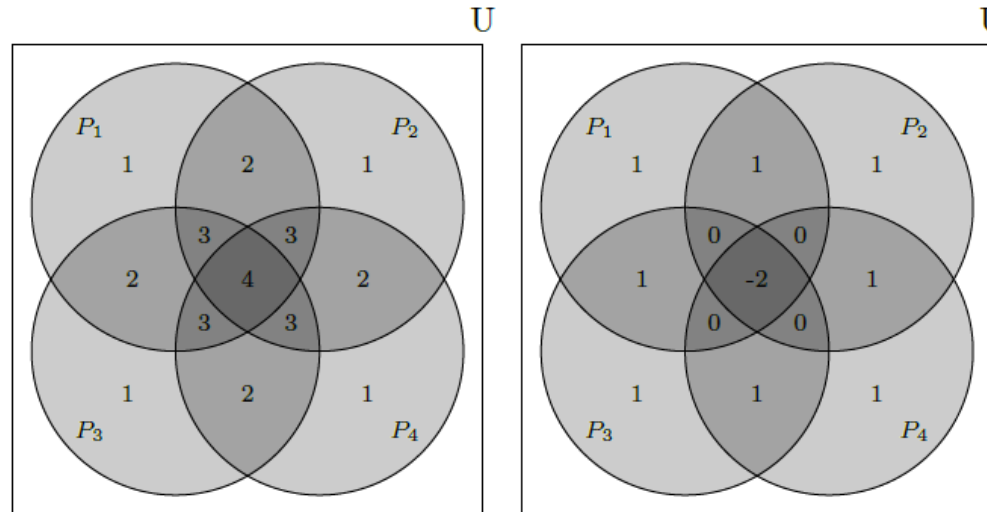
$$A_{r,j}[X] = \begin{cases} \text{true} & \text{if } j \geq 1, X = \emptyset \\ \text{false} & \text{if } j \leq 1 \\ \bigvee_{Y \subseteq X} \bigvee_{r' \in N(r)} \bigvee_{j'=1}^j A_{r',j'}[Y \setminus r'] \wedge A_{r,j-j'}[X \setminus Y] & \text{otherwise.} \end{cases}$$

- Pick an arbitrary $r \in T$
- Evaluating and reading off the answer $A_{r,l}[T \setminus r]$ results in an $O^*(3^k)$ time algorithm (by binomial theorem again).

Inclusion Exclusion

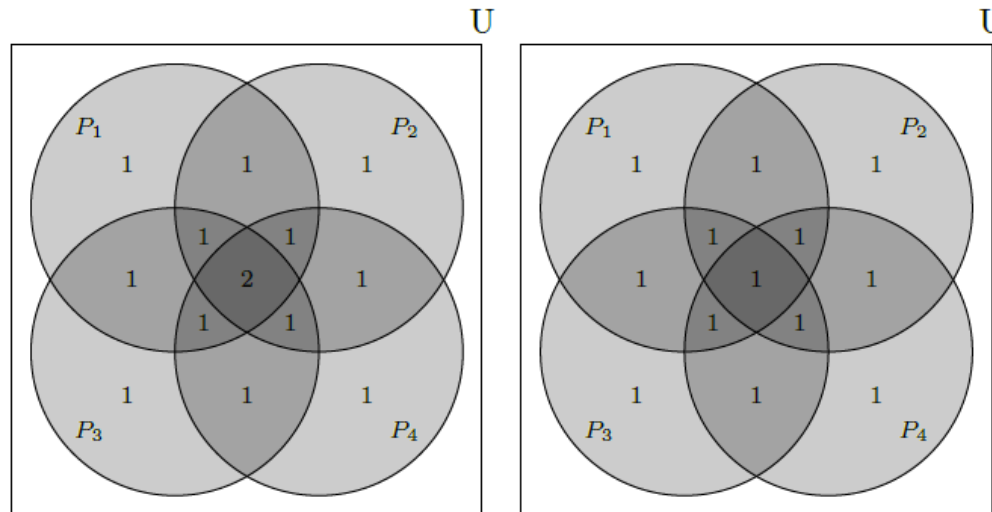
- Classical version: express the size of a union as size of an intersection: if $P_1, P_2 \subseteq U$
$$|P_1 \cup P_2| = |P_1| + |P_2| - |P_1 \cap P_2|$$
- We are often interested in intersections:
$$|P_1 \cap P_2| = |U| - |\overline{P_1}| - |\overline{P_2}| + |\overline{P_1 \cap P_2}|$$
- Generalizes to n sets.

Computing $|P_1 \cup P_2 \cup P_3 \cup P_4|$



(a) $\sum_i |P_i|$

(b) $-\sum_{i < j} |P_i \cap P_j|$



(c) $+\sum_{i < j < k} |P_i \cap P_j \cap P_k|$

(d) $-\sum_{i < j < k < l} |P_i \cap P_j \cap P_k \cap P_l|$

Inclusion Exclusion Formula

- If $P_1, \dots, P_n \subseteq U, \bar{P}_i = U \setminus P_i, \bigcap_{i \in \emptyset} \bar{P}_i = U,$

$$\left| \bigcap_{i=1}^n P_i \right| = \sum_{F \subseteq \{1, \dots, n\}} (-1)^{|F|} \left| \bigcap_{i \in F} \bar{P}_i \right|.$$

Inclusion Exclusion Formula

- If $P_1, \dots, P_n \subseteq U, \bar{P}_i = U \setminus P_i, \bigcap_{i \in \emptyset} \bar{P}_i = U,$

$$\left| \bigcap_{i=1}^n P_i \right| = \sum_{F \subseteq \{1, \dots, n\}} (-1)^{|F|} \left| \bigcap_{i \in F} \bar{P}_i \right|.$$

- Proof: for $e \in U,$ let $p(e) = \{i: e \in P_i\}$

$$\begin{aligned} \sum_{F \subseteq \{1, \dots, n\}} (-1)^{|F|} \left| \bigcap_{i \in F} \bar{P}_i \right| &= \sum_{F \subseteq \{1, \dots, n\}} (-1)^{|F|} |\{e \in U : p(e) \cap F = \emptyset\}| \\ &= \sum_{e \in U} \sum_{F \subseteq \{1, \dots, n\} \setminus p(e)} (-1)^{|F|}. \end{aligned}$$

- $\sum_{Y \subseteq X} (-1)^{|Y|}$ is 1 if $X = \emptyset$ and 0 otherwise by bin thm

$$\sum_{y \subseteq X} (-1)^{|Y|} = \sum_{i=0}^{|X|} \binom{|X|}{i} (-1)^i = (1 - 1)^{|X|}$$

k-coloring in $O^*(2^n)$ time

- Equivalently: can we cover the vertices with k independent sets?
- Let $U = \{(I_1, \dots, I_k): I_i \text{ ind. set of } G\}$
- Let $P_v = \{(I_1, \dots, I_k) \in U: v \in I_i, \text{ for some } i\}$
- $|\bigcap_{v \in V} P_v| = \# \text{coverings with } k \text{ ind. sets}$

$$|\bigcap_{i=1}^n P_i| = \sum_{F \subseteq \{1, \dots, n\}} (-1)^{|F|} |\bigcap_{i \in F} \overline{P}_i|.$$

$|\bigcap_{i \in F} \overline{P}_i| = \# \text{ind. Sets in } G[V \setminus F] \text{ to the power } k.$

k-coloring in $O^*(2^n)$ time

- For $X \subseteq G$, let $i[X] = \#ind. sets in G[X]$
- Use DP to compute $i[X]$ for every $X \subseteq V$

$$i[X] = \begin{cases} 1 & X = \emptyset \\ i[X \setminus v] + i[X \setminus N[v]] & \text{if } v \in X. \end{cases}$$

- Given this table we can evaluate the formula

$$\sum_{F \subseteq V} (-1)^{|F|} i[V \setminus F]^k$$

- In time $O^*(2^n)$.

Hamiltonian Cycle in $O^*(2^n)$ time and polynomial space

- Notation:
 - *Walk*: sequence P of vertices p_1, \dots, p_l s.t $(p_i, p_{i+1}) \in E$
 - *Length* of $P = l - 1$ (# of edges)
 - *From s to t* : $p_1 = s, p_l = t$, *cyclic* if $p_1 = p_l$
- HC: cyclic walk of length n visiting all vertices.

Hamiltonian Cycle in $O^*(2^n)$ time and polynomial space

- Notation:
 - *Walk*: sequence P of vertices p_1, \dots, p_l s.t $(p_i, p_{i+1}) \in E$
 - *Length* of $P = l - 1$ (# of edges)
 - *From s to t* : $p_1 = s, p_l = t$, *cyclic* if $p_1 = p_l$
- HC: cyclic walk of length n visiting all vertices.
- $U =$ all cyclic walks of length n in G
- $P_v =$ cyclic walks visiting $v \Rightarrow |\bigcap_{v \in V} P_v| = \#HC$

Hamiltonian Cycle in $O^*(2^n)$ time and polynomial space

- Notation:

- *Walk*: sequence P of vertices p_1, \dots, p_l s.t. $(p_i, p_{i+1}) \in E$
- *Length* of $P = l - 1$ (# of edges)
- *From s to t* : $p_1 = s, p_l = t$, *cyclic* if $p_1 = p_l$

- HC: cyclic walk of length n visiting all vertices.

- $U =$ all cyclic walks of length n in G

- $P_v =$ cyclic walks visiting $v \Rightarrow |\bigcap_{v \in V} P_v| = \#HC$

$$|\bigcap_{i=1}^n P_i| = \sum_{F \subseteq \{1, \dots, n\}} (-1)^{|F|} |\bigcap_{i \in F} \overline{P}_i|.$$

$|\bigcap_{i \in F} \overline{P}_i|$ equals #cyclic walks of length n in $G[V \setminus F]$

Hamiltonian Cycle in $O^*(2^n)$ time and polynomial space

- Notation:

- *Walk*: sequence P of vertices p_1, \dots, p_l s.t. $(p_i, p_{i+1}) \in E$
- *Length* of $P = l - 1$ (# of edges)
- *From s to t* : $p_1 = s, p_l = t$, *cyclic* if $p_1 = p_l$

- $w_F(s, t, k)$ = #length k walks from s to t avoiding F

$$w_F(s, t, k) = \begin{cases} 0 & \text{if } k = 0 \text{ and } s \neq t \\ 1 & \text{if } k = 0 \text{ and } s = t, \\ \sum_{t' \in N^-(t) \setminus F} w_F(s, t', k - 1) & \text{otherwise.} \end{cases}$$

- #cyclic walks of length n avoiding F can be computed as $\sum_{s \in V \setminus F} w_F(s, s, n)$