More Dynamic Programming

Common Subproblems

- Opt(i) - Opt solution using $x_1,..,x_i$. (eg LIS, longest path).
- Opt(i,j) - Opt solution using $x_i,...,x_j$. (eg RNA)
- Opt(i,j) - Opt solution using $x_1,...,x_i$ and $y_1,...,y_j$. (eg Edit distance)
- Opt(r) - Opt solution using subtree rooted at r. (eg Vertex cover on trees).

Longest increasing subsequence

Given: sequence of numbers

Goal: find longest increasing subsequence

41 , 22 , 9 , 15 , 23 , 39 , 21 , 56 , 24 , 34 , 59 , 23 , 60 , 39 , 87 , 23 , 90

Longest increasing subsequence

Given: sequence of numbers

Goal: find longest increasing subsequence

41 , 22 , **9** , **15** , **23** , 39 , 21 , 56 , **24** , **34** , **59** , 23 , **60** , 39 , **87** , 23 , **90**

longest increasing subsequence: length 9

Longest increasing subsequence

Given: sequence of numbers $x_1,..,x_n$

Goal: find longest increasing subsequence

41 , 22 , 9 , 15 , 23 , 39 , 21 , 56 , 24 , 34 , 59 , 23 , 60 , 39 , 87 , 23 , 90

Subproblems: l(j) - length of longest increasing subseq. ending at j.

Longest increasing subsequence

Given: sequence of numbers $x_1,..,x_n$

Goal: find longest increasing subsequence

41 , 22 , 9 , 15 , 23 , 39 , 21 , 56 , 24 , 34 , 59 , 23 , 60 , 39 , 87 , 23 , 90

Subproblems: l(j) - length of longest increasing subseq. ending at j.

Observation: if longest inc. sub. ending at j is $x_{i1},x_{i2},...,x_i,x_j$ then $l(j) = l(i)+1$

Longest increasing subsequence

Given: sequence of numbers $x_1,..,x_n$

Goal: find longest increasing subsequence

41 , 22 , 9 , 15 , 23 , 39 , 21 , 56 , 24 , 34 , 59 , 23 , 60 , 39 , 87 , 23 , 90

Subproblems: l(j) - length of longest increasing subseq. ending at j.

Observation: if longest inc. sub. ending at j is $x_{i1},x_{i2},...,x_i,x_j$ then $l(j) = l(i)+1$

Claim: $l(j) = \begin{cases} 1 & \text{if } x_i \geq x_j\text{, for all } i<j \\ 1+\max\limits_{i:\ i<j,\ x_i<x_j} l(i) & \text{else} \end{cases}$

Longest increasing subsequence

Subproblems: l(j) - length of longest increasing subseq. ending at j.

Claim:
$$l(j) = \begin{cases} 1 & \text{if } x_i \geq x_j \text{, for all } i<j \\ 1+\max\limits_{i:\, i<j,\, x_i<x_j} l(i) & \text{else} \end{cases}$$

Algorithm:

for $j=1,\ldots,n$
 if $x_i \geq x_j$, for all $i<j$, set $l(j) = 1$
 else, set $l(j) = 1+\max\limits_{i:\, i<j,\, x_i<x_j} l(i)$

output $\max\limits_{j} l(j)$

Running time

$O(n^2)$

All pairs shortest path in directed graph with no negative cycles.

Given: directed graph, (possibly negative) edge weights

Goal: find shortest path between every two vertices

Bellman-Ford algorithm can do this in time $O(n^2m)$

All pairs shortest path in directed graph with weighted edges

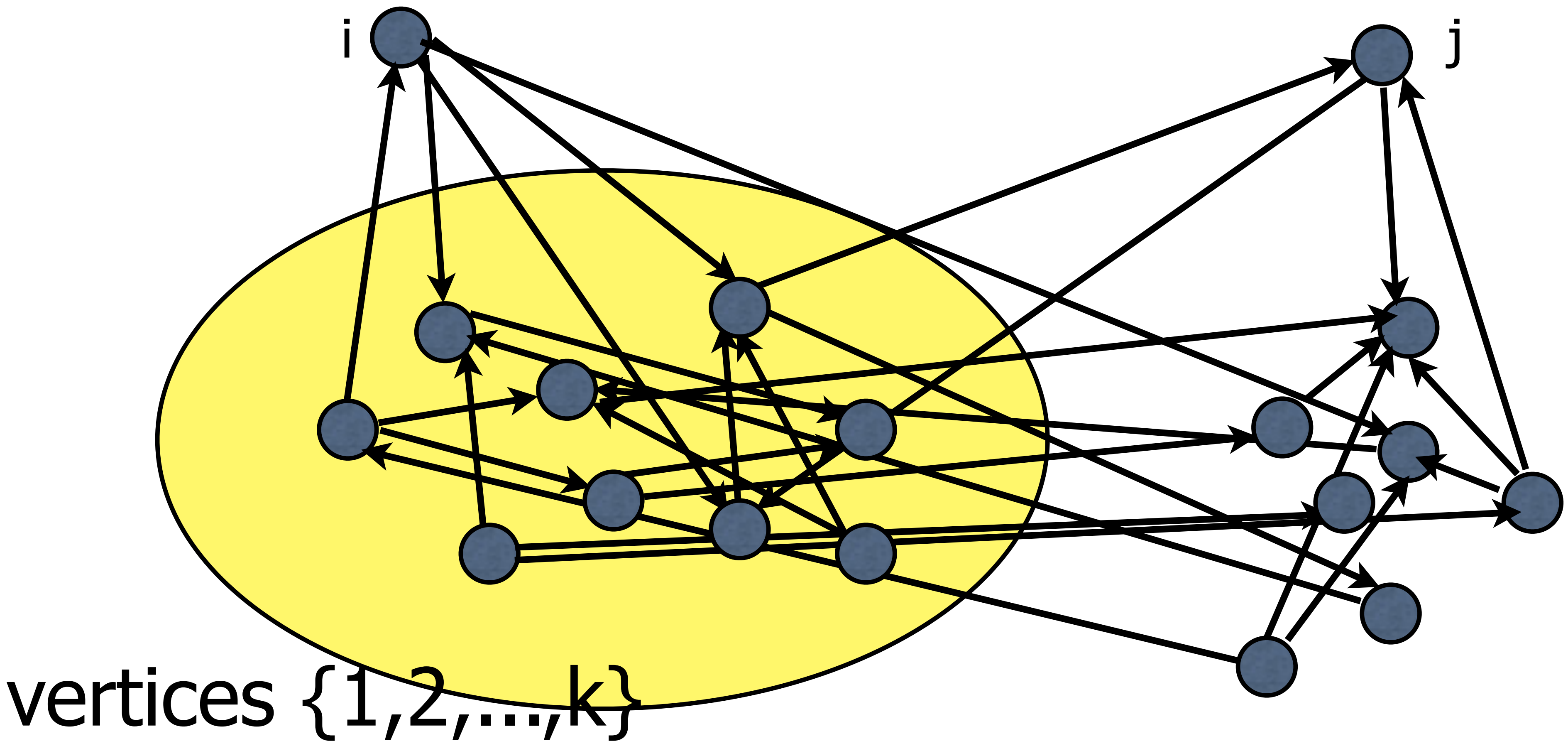
Given: directed graph, (possibly negative) edge weights

Goal: find shortest path between every two vertices

Subproblems: d(i,j,k) - length of shortest path that starts at i, ends at j and visits only {1,2,...,k} in the middle.

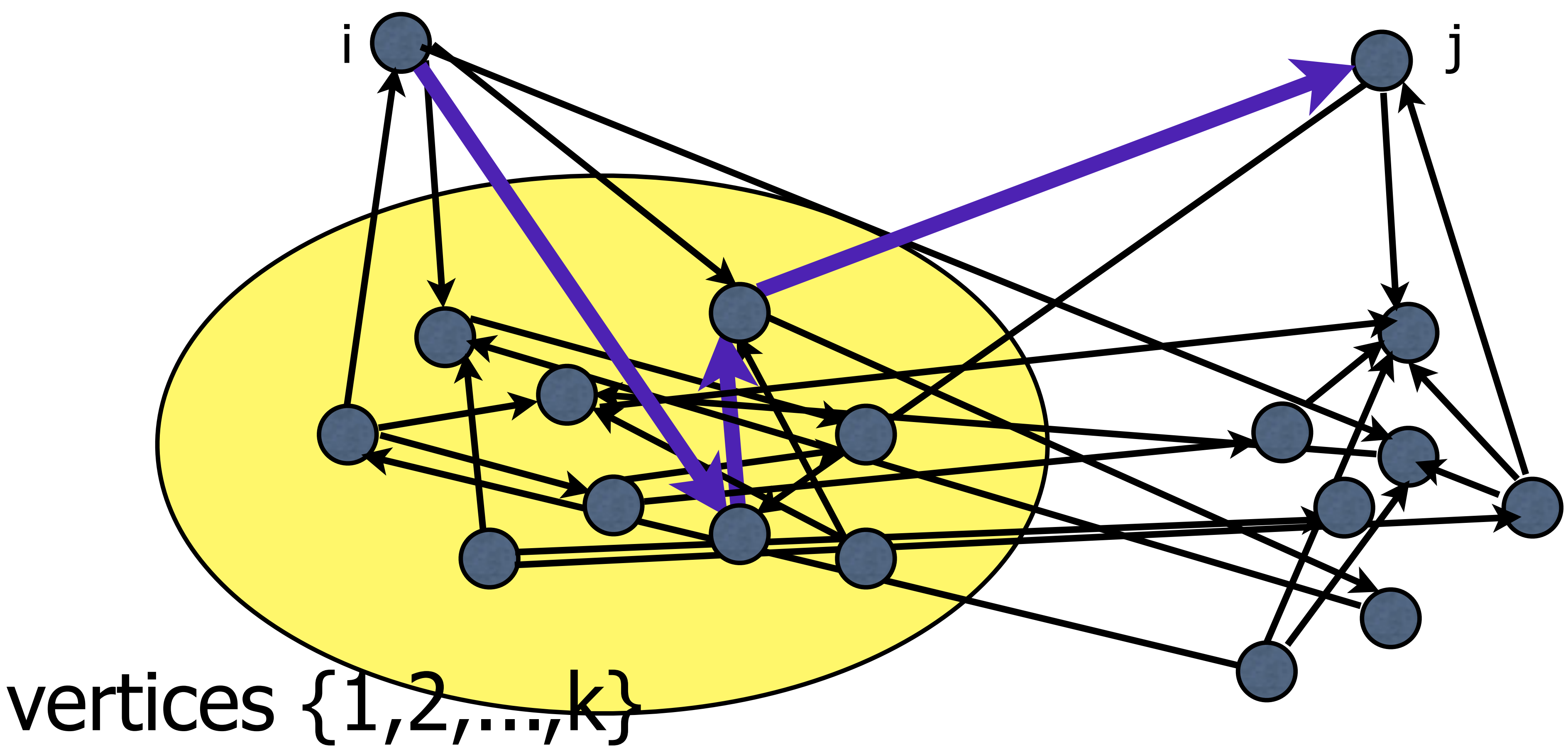
Goal: find shortest path between every two vertices

Subproblems: d(i,j,k) - length of shortest path that starts at i, ends at j and every other vertex on path is in {1,2,...,k}.



Goal: find shortest path between every two vertices

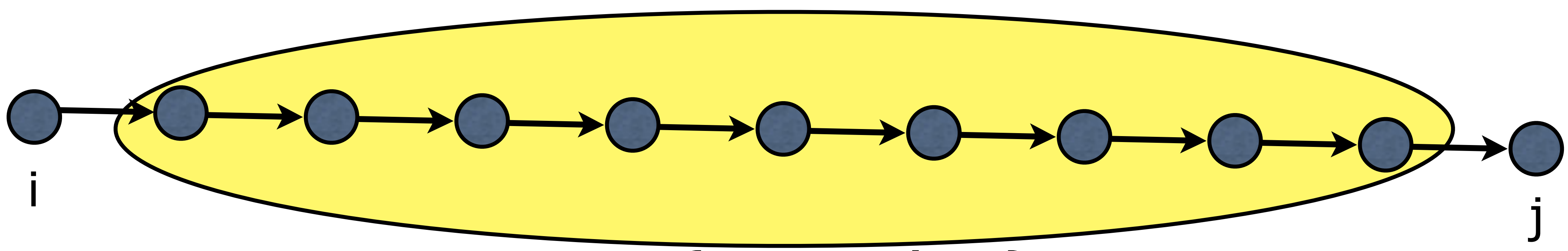
Subproblems: d(i,j,k) - length of shortest path that starts at i, ends at j and every other vertex on path is in {1,2,...,k}.



Subproblems: d(i,j,k) - length of shortest path that starts at i, ends at j and every other vertex on path is in {1,2,...,k}.

Observation:
if shortest path for d(i,j,k) does not visit k, then

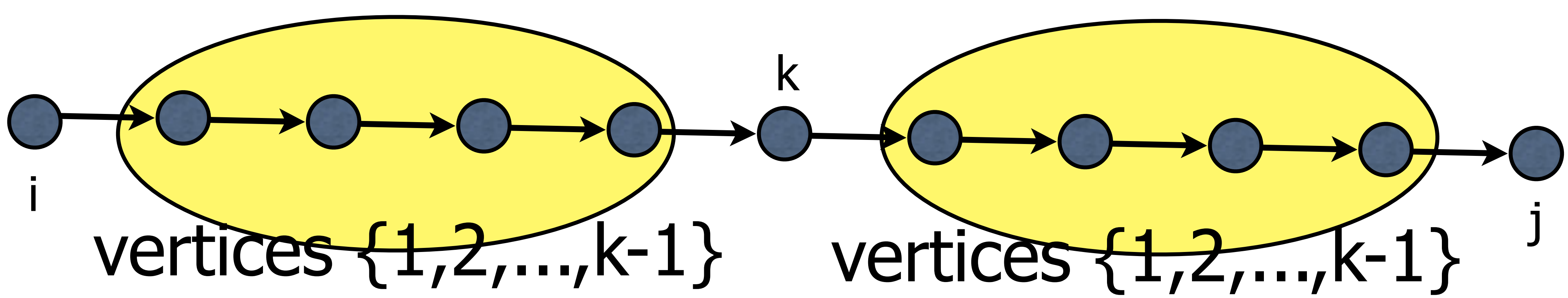
$$d(i,j,k) = d(i,j,k-1).$$



vertices {1,2,...,k-1}

Otherwise,

$$d(i,j,k) = d(i,k,k-1) + d(k,j,k-1)$$



Subproblems: d(i,j,k) - length of shortest path that starts at i, ends at j and every other vertex on path is in {1,2,...,k}.

Claim: d(i,j,k) = min{d(i,j,k-1), d(i,k,k-1)+d(k,j,k-1)}

Algorithm:

```
for all i,j=1,...,n
    set d(i,j,0) = weight of edge (i,j)

for k=1,...,n
    for all i,j=1,...,n
        set d(i,j,k) = min{d(i,j,k-1),d(i,k,k-1)+d(k,j,k-1)}
```

<u>Running time</u> $O(n^3)$

Traveling Salesperson Problem

Given: n cities, and the pairwise distances d_{ij}

Goal: find shortest tour that visits every city at least once

Traveling Salesperson Problem

Given: n cities, and the pairwise distances d_{ij}

Goal: find shortest tour that visits every city at least once

Brute force search algorithm: $n! \sim 2^{n\log n}$ time.

Traveling Salesperson Problem

Given: n cities, and the pairwise distances d_{ij}

Goal: find shortest tour that visits every city at least once

Brute force search: $n! \sim 2^{n\log n}$ time.

Subproblems: $T(v,S)$ - length of shortest tour that visits all cities of the set S and ends at v.

Given: n cities, and the pairwise distances d_{ij}

Goal: find shortest tour that visits every city at least once

Subproblems: T(v,S) - length of shortest tour that visits all cities of the set S and ends at v.

Observation:
if shortest tour for T(v,S) visits city u right before v, then

$$T(v,S) = T(u,S-v) + d_{uv}$$

Given: n cities, and the pairwise distances d_{ij}

Goal: find shortest tour that visits every city at least once

Subproblems: T(v,S) - length of shortest tour that visits all cities of the set S and ends at v.

Algorithm:

```
for v=1,...,n
    set T(v,{v}) = 0
for k=2,...,n
    for all sets of cities S, |S|=k
        for all v in S
            set T(v,S) = min          T(u,S-v)+d_uv
                         u in S-v
```

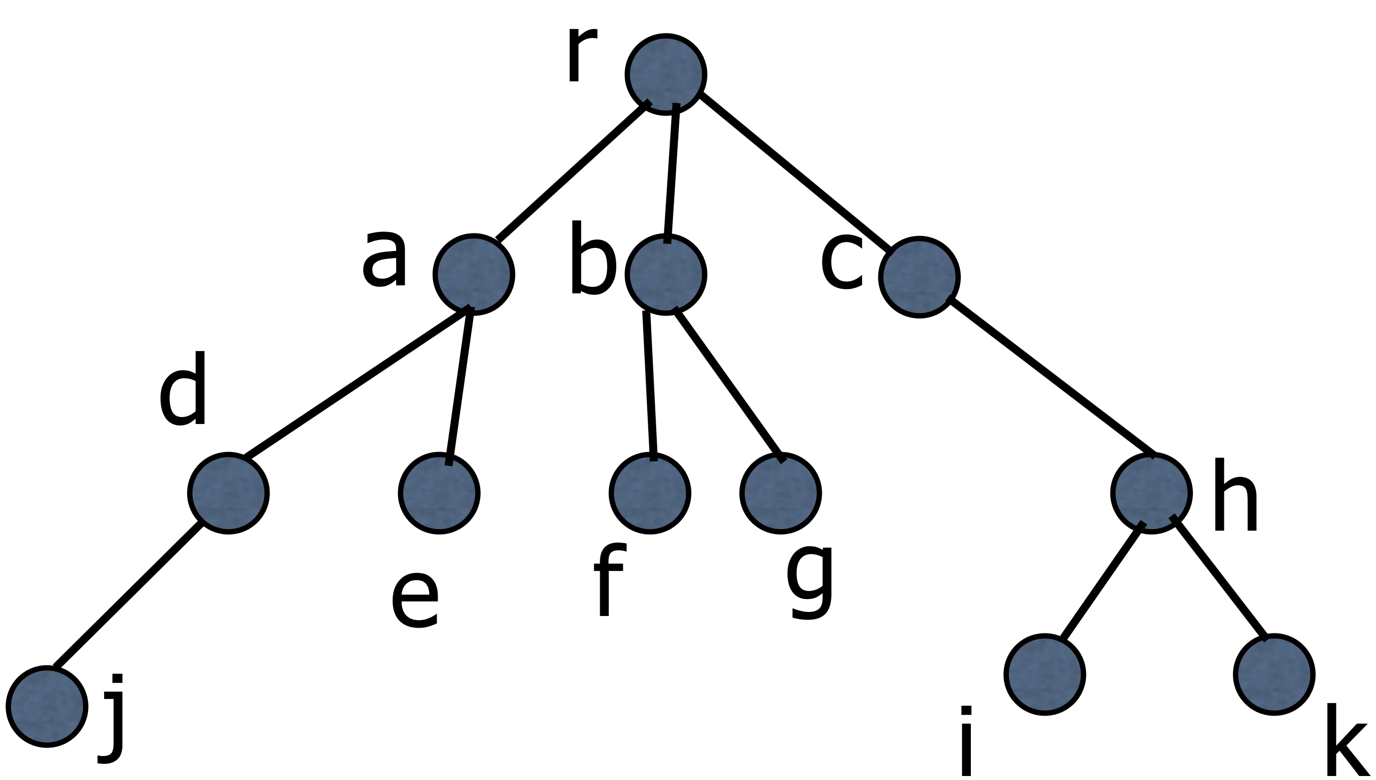
Running time

$O(n^2\, 2^n)$

Vertex Cover on Acyclic Graphs

Given: A tree

Goal: find smallest vertex cover (vertices that touch all edges)

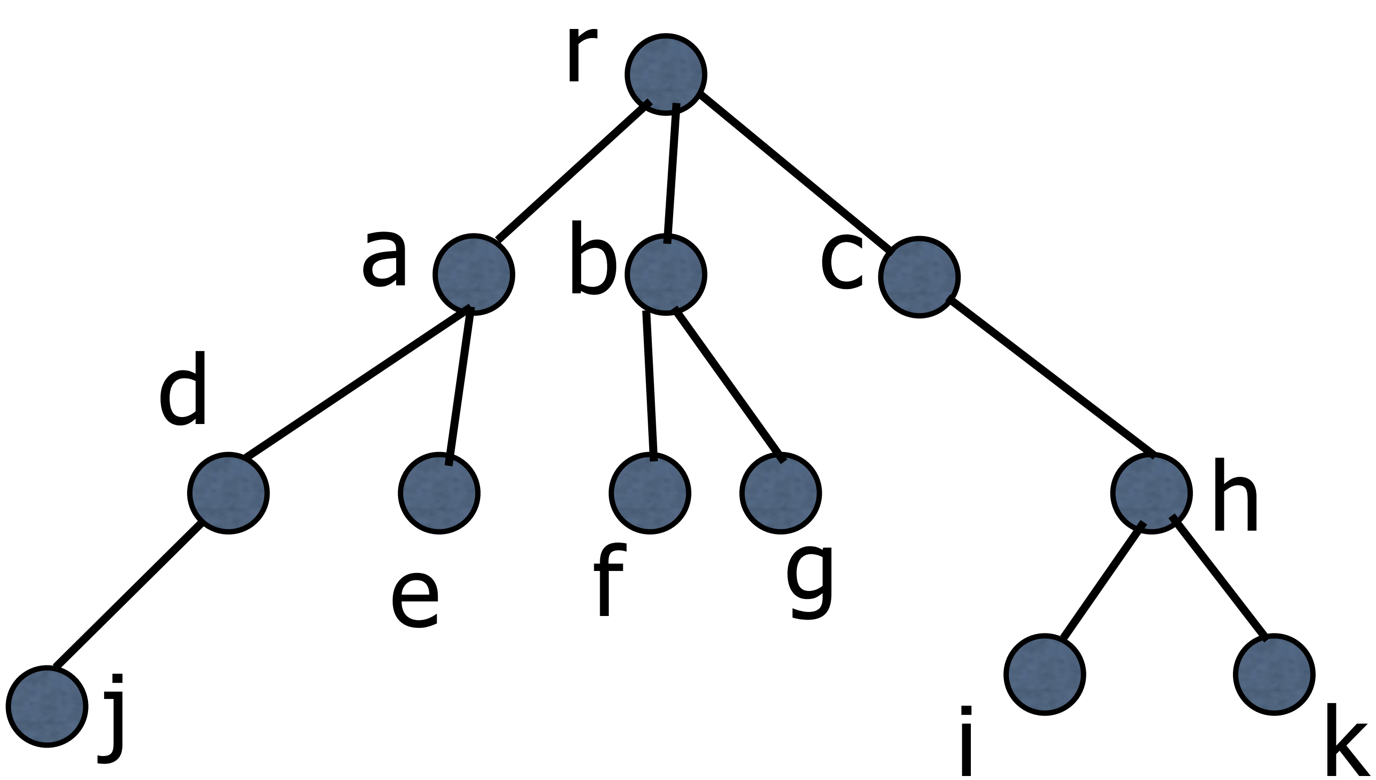


Vertex Cover on Acyclic Graphs

Given: A tree

Goal: find smallest vertex cover (vertices that touch all edges)

Subproblems: V(q) - size of smallest vertex cover of subtree rooted at q.

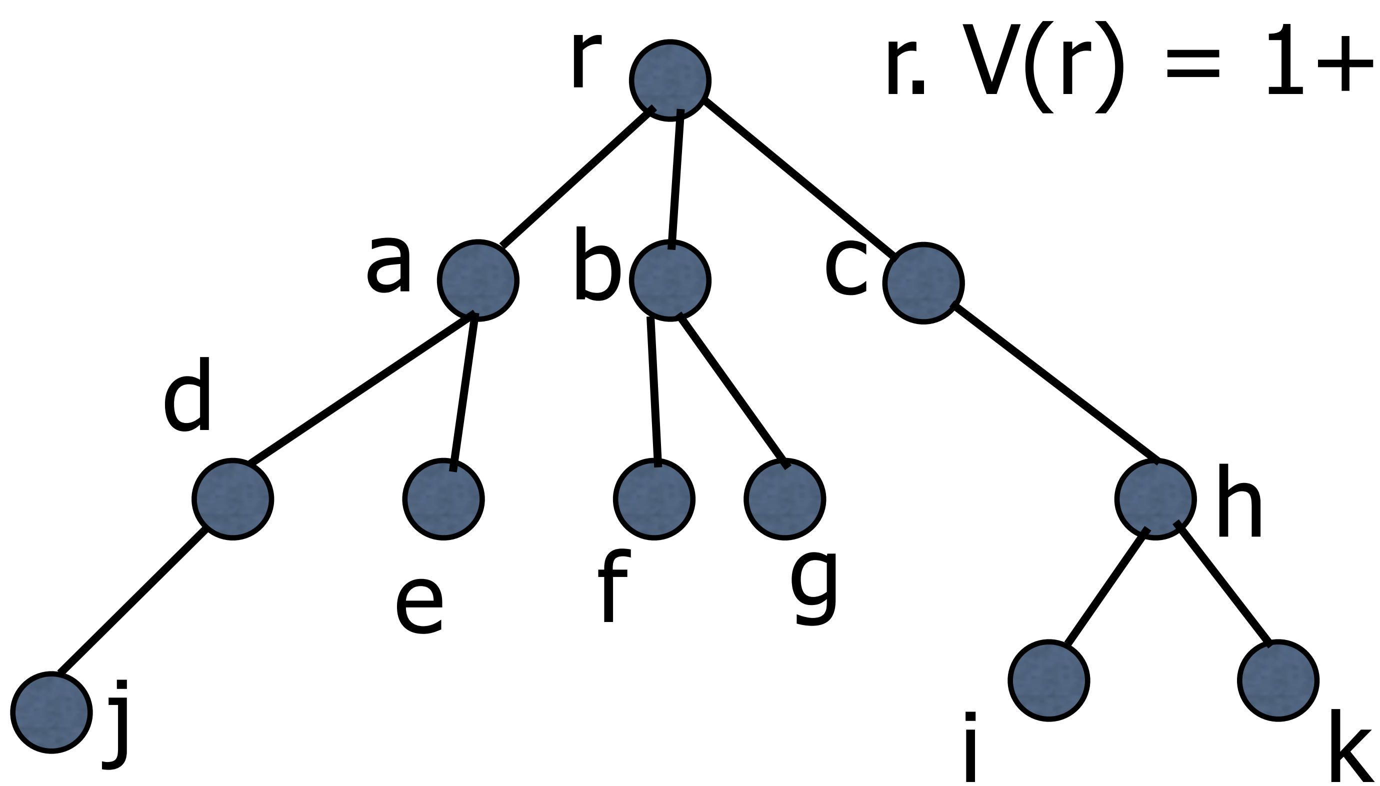


Vertex Cover on Acyclic Graphs

Subproblems: V(r) - size of vertex cover at subtree rooted at r.

Case 1: Cover realizing V(r) does not contain r. Then it must contain children(r).
V(r)= #children(r) + sum over grandchilren g V(g)

Case 2: Cover realizing V(r) does contain r. V(r) = 1+sum over children c V(c)



Vertex Cover on Acyclic Graphs

Subproblems: V(r) - size of vertex cover at subtree rooted at r.

Case 1: Cover realizing V(r) does not contain r. Then it must contain children(r).
V(r)= #children(r) + sum over grandchilren g V(g)

Case 2: Cover realizing V(r) does contain r.
V(r) = 1+sum over children c V(c)

Rough Algorithm:

Running time
O(n)

For each vertex r, in decreasing order of depth, set

$$V(r) = \min\{\#\text{children}(r)+\sum_{g,\ \text{grandchild of } r} V(g),\ 1+\sum_{c,\ \text{child of } r} V(c)\}$$

Chain Matrix Multiplication

Given: n matrices $M_1,M_2,\ldots,M_n$

Goal: compute product $M_1,M_2,\ldots,M_n$ (in what order should we multiply?)

Example: To compute VWXYZ we could multiply V((WX)(YZ)) or (V(W(XY)))Z or ...

Basic operations: multiplying (a by b) matrix with (b by c) matrix gives (a by c) matrix in abc time.

Chain Matrix Multiplication

Given: n matrices $M_1, M_2, \ldots, M_n$

Goal: compute product $M_1, M_2, \ldots, M_n$ (in what order should we multiply?)

Example: To compute VWXYZ we could multiply V((WX)(YZ)) or (V(W(XY)))Z or ...

Basic operations: multiplying (a by b) matrix with (b by c) matrix gives (a by c) matrix in abc time.

Subproblems: C(i,j) - time to compute $M_i M_{i+1} \ldots M_j$

Given: n matrices $M_1,\ldots,M_n$, i'th matrix of size (m_i by m_{i+1})

Goal: compute product $M_1,M_2,\ldots,M_n$ (in what order should we multiply?)

Basic operations: multiplying (a by b) matrix with (b by c) matrix gives (a by c) matrix in abc time.

Subproblems: C(i,j) - time to compute $M_iM_{i+1}\ldots M_j$

Observation: If the final multiplication in optimal solution is between $(M_i\ldots M_k)(M_{k+1}\ldots M_j)$, then
$C(i,j) = C(i,k)+C(k,j)+ n_in_{k+1}n_j$.

Basic operations: multiplying (a by b) matrix with (b by c) matrix gives (a by c) matrix in abc time.

Subproblems: $C(i,j)$ - time to compute $M_i M_{i+1} \ldots M_j$

Observation: If the final multiplication in optimal solution is between $(M_i \ldots M_k)(M_{k+1} \ldots M_j)$, then
$C(i,j) = C(i,k)+C(k+1,j)+ m_i m_{k+1} m_j$.

Algorithm:

Running time
$O(n^3)$

for $i=1,2,\ldots,n-1$, set $C(i,i)=0$

for $s=1,2,\ldots,n-1$, $i=1,\ldots,n-1$

$$\text{set } C(i,i+s)=\min_{i<k<s} C(i,k)+C(k+1,i+s)+m_i m_{k+1} m_{i+s}$$

Common Subproblems

- Opt(i) - Opt solution using $x_1,..,x_i$. (eg LIS, longest path).
- Opt(i,j) - Opt solution using $x_i,...,x_j$. (eg RNA)
- Opt(i,j) - Opt solution using $x_1,...,x_i$ and $y_1,...,y_j$. (eg Edit distance)
- Opt(r) - Opt solution using subtree rooted at r. (eg Vertex cover on trees).