Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 7

University of Southern California

Spring 2023

Dynamic Programming - 2

Reading: chapter 6

Exam - I

Date: Thursday March 2

Time: starts at 5pm

Locations: TBA

Practice Exam: posted

TA Review: February 28 and March 1

Closed book and notes
No internet searching
No talking to each other (chat, phone, messenger)

Concept of Dynamic Programming

Optimal substructure:

optimal solution to problem consists of optimal solutions to subproblems

Overlapping subproblems:

few subproblems in total, many recurring instances of each

Solve *bottom-up*, building a table of solved subproblems that are used to solve larger ones.

Discussion Problem 1

substring

A subsequence is palindromic if it reads the same left and right. Devise a DP algorithm that takes a string and returns the length of the longest palindromic subsequence (not necessarily contiguous).

For example, the string

QRAECCETCAURP

has several palindromic subsequences, RACECAR is one of them.

① How many substrings? $O(n^2)$

② How many subsequences? $O(2^n)$

input $(s[1.2.. n])$

Step 1. Subproblems.

Let $OPT[i,j]$ be the longest

Step 2. Recurrence

case 1: $s[i] = s[j]$

$OPT[i,j] = 2 + OPT[i+1, j-1]$

case 2: $s[i] \neq s[j]$

$OPT[i,j] = \max($
$OPT[i+1, j],$
$OPT[i, j-1])$

Base cases:

$OPT[i,i] = 1$

$OPT[i,j] = 2$, & $s[i] = s[j]$
then $j = i+1$

Example:

"AA"

$i: 0 \to 1$

$j: 1 \to 0$

Discussion Problem 2

You are to plan the fall 2023 schedule of classes. Suppose that you can sign up for as many classes as you want, and you'll have infinite amount of energy to handle all the classes, but you cannot have any time conflict between the lectures. Also assume that the problem reduces to planning your schedule of *one particular day*. Thus, consider one particular day of the week and all the classes happening on that day: $c_1, .., c_n$. Associated with each class c_i is a start time s_i and a finish time f_i and you also assign a score v_i to that class based on your interests and your program requirement. Assume $s_i < f_i$. You would like to choose a set of courses C for that day to *maximize* the total score. Devise an algorithm for planning your schedule.

① Input: bunch of classes

Sort classes: $c_1, c_2, \dots, c_n$ by finish time

② Dosing a DP algorithm:

2.1 Let $OPT[i]$ be the optimal solution for $c_1, c_2, \dots, c_i$

2.2 Recurrence.

$$OPT[i] = MAX\left(\begin{array}{l} OPT[i-1], \\ v_i + OP[\underset{p(i)}{\boxed{i-1}}] \end{array}\right) \quad ?$$

$$p(i) = \max_j (j \mid f_j < s_i)$$

2.3 Runtime: $O(n \cdot n)$

a) $O(n)$

[✓ b) $O(n^2)$]

c) $O(n^3)$

d) no idea :-)

[✓ e) $O(n \log n)$] correct answer

f) $O(n^2 \log n)$

run binary search

Static Optimal Binary Search Tree

Build a binary search tree which gives a minimum search cost, assuming we know the frequencies p_i with which data k_i is accessed. The tree cannot be modified after it has been constructed.

Want to build a binary search tree with minimum expected search cost:

$$\text{Expected Cost} = \sum_{i=1}^{n} p_i \underline{\text{depth}}(k_i)$$

$\text{depth}(\text{root}) = 1$

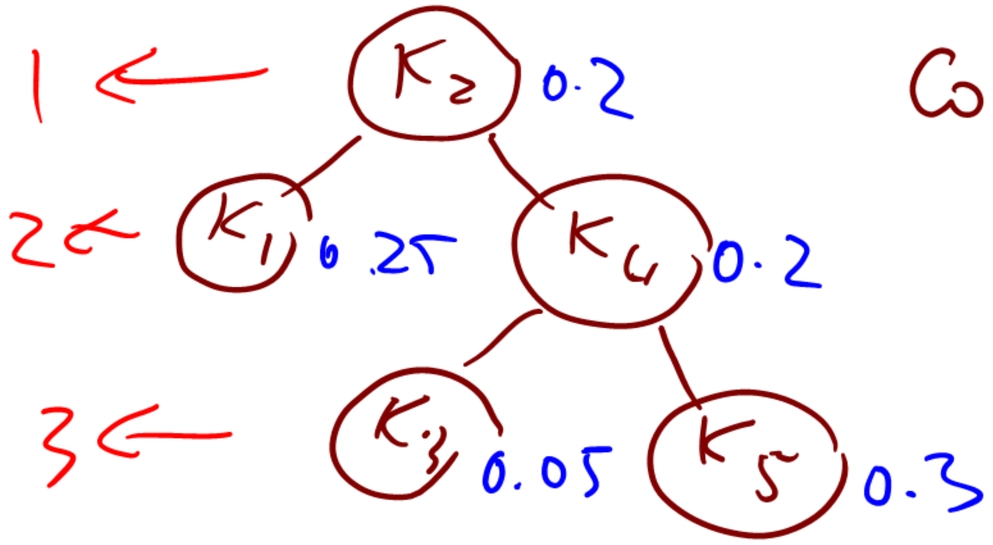
Example

Consider 5 items

$$k_1 < k_2 < k_3 < k_4 < k_5$$

and their search probabilities

$$p_1 = 0.25,\ p_2 = 0.2,\ p_3 = 0.05,\ p_4 = 0.2,\ p_5 = 0.3.$$



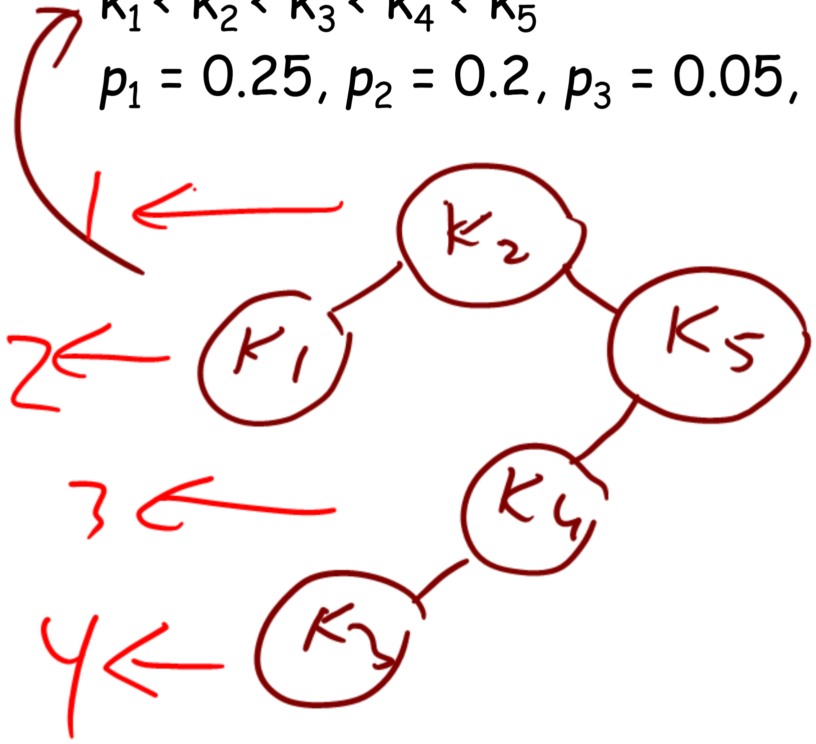
Cost = $0.2 \times 1 + 0.25 \times 2 + 0.2 \times 2 + 0.05 \times 3 + 0.3 \times 3 =$

$= 2.15$

Another possibility

$k_1 < k_2 < k_3 < k_4 < k_5$

$p_1 = 0.25,\ p_2 = 0.2,\ p_3 = 0.05,\ p_4 = 0.2,\ p_5 = 0.3$



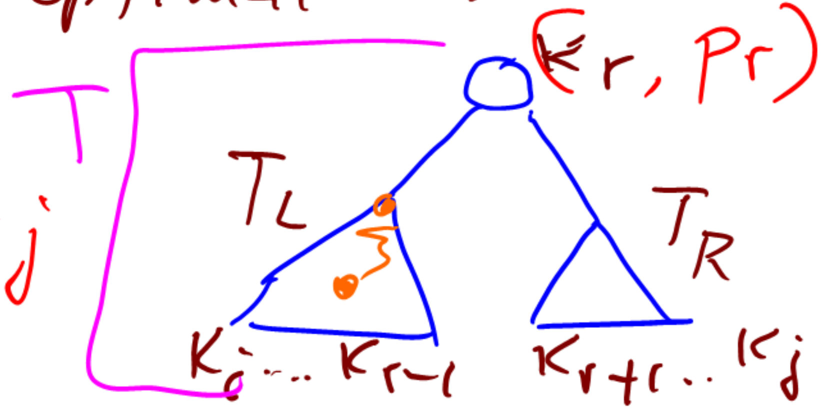
$\text{Cost} = 0.2 \times 1 + 0.25 \times 2 + 0.3 \times 2 + 0.2 \times 3 + 0.05 \times 4 = 2.1$

k = key

Optimal Substructure

Let OPT[i, j] be the optimal cost for $k_i, k_{i+1}, \ldots, k_j$.

The root k_r ! $i \le r \le j$



Compute the cost

$$\text{Cost} = OPT[i,j] = 1 \times P_r + \sum_{s=i}^{r-1} P_s \cdot \text{depth}_T(k_s) + \sum_{s=r+1}^{j} P_s \cdot \text{depth}_T(k_s)$$

$$\boxed{\mathrm{depth}_T(k_s)} = 1 + \mathrm{depth}_{T_L}(k_s)$$

$$= 1 + \mathrm{depth}_{T_R}(k_s)$$

$$= p_r + \sum_{s=i}^{r-1} p_s\left(\boxed{1} + \underbrace{\mathrm{depth}_{T_L}(k_s)}\right) \leftarrow$$

$$+ \sum_{s=r+1}^{j} p_s\left(\boxed{1} + \underbrace{\mathrm{depth}_{T_R}(k_s)}\right) =$$

$$= p_i + p_{i+1} + \ldots + p_{r-1} + p_r + p_{r+1} + \ldots + p_j +$$

$$+ \boxed{\sum_{s=i}^{r-1} p_s \cdot \mathrm{depth}_{T_L}(k_s)} + \;= \mathrm{cost}(T_L)$$

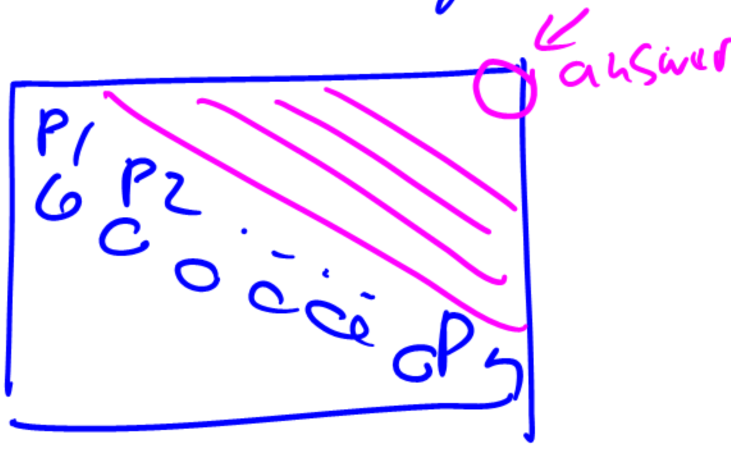
$$+ \sum_{s=r+1}^{j} p_s \cdot \mathrm{depth}_{T_R}(k_s) = \mathrm{cost}(T_R)$$

Recurrence Relation

$$OPT(i,j) = p_i + \dots + p_j + \underbrace{cost(T_L) + cost(T_R)}_{\text{rec. calls}}$$

$$OPT(i,j) = \min_{i \le r \le j} \left(p_i + \dots + p_j + OPT(i, r-1) + OPT(r+1, j) \right)$$

$OPT[i,j]$



Base cases:

$OPT(i,i) = p_i$

$OPT(\underline{i, i-1}) = 0$?

Filling up the table

```
array p = [$p_1$, $p_2$, …, $p_n$]
set OPT[i, i-1] = 0, for $1 \le i \le n$
set OPT[i, i] = $p_i$, for $1 \le i \le n$

for(k = 1; k < n; k++)
   for(i = 1; i <= n-k, i++)
      j = i + k;
     OPT[i, j] =$p_i$+…+$p_j$+ min$_r$(OPT[i,r-1]+OPT[r+1,j];
                               $(i \le r \le j)$
return OPT[1,n];
```

Runtime Complexity-? $O(n^3)$

Example

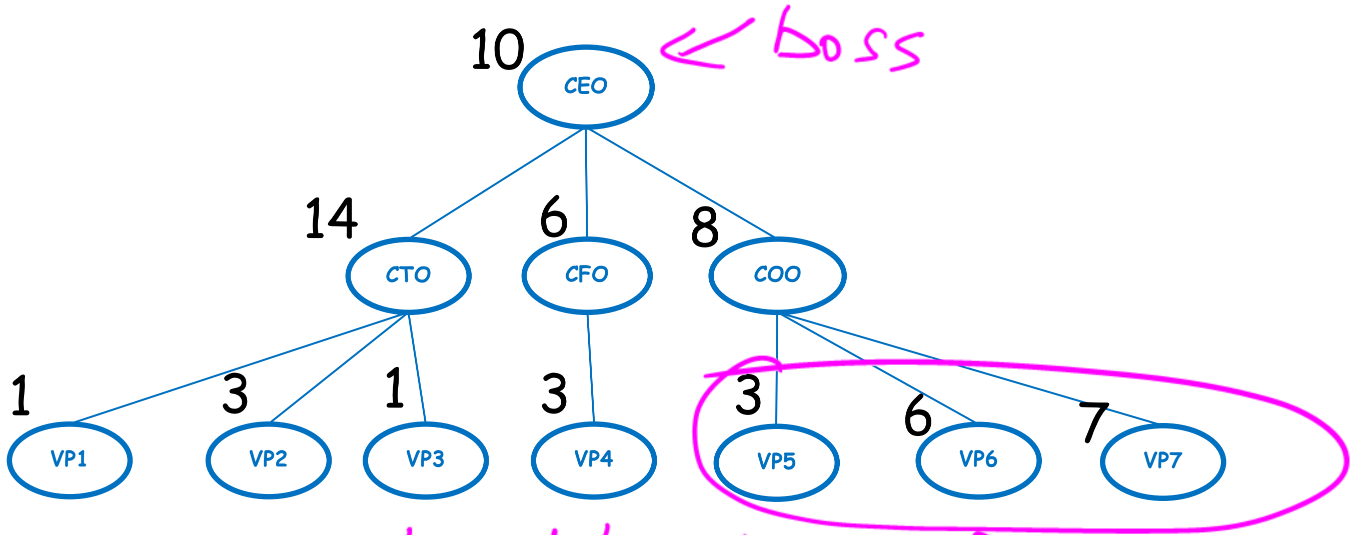
n = 5

(prob,value)=(0.1,5), (0.3,6), (0.9,4), (0.3,3), (0.1,8)

		0	1	2	3	4	5
(0.1, 5)	1	0	0.1	0.5			
(0.3, 6)	2		0	0.3			
(0.9, 4)	3			0	0.9		
(0.3, 3)	4				0	0.3	
(0.1, 8)	5					0	0.1

Discussion Problem 3

Suppose you are organizing a company party. The corporation has a tree-like ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee j has a value v_j (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.



Shall we invite the boss?

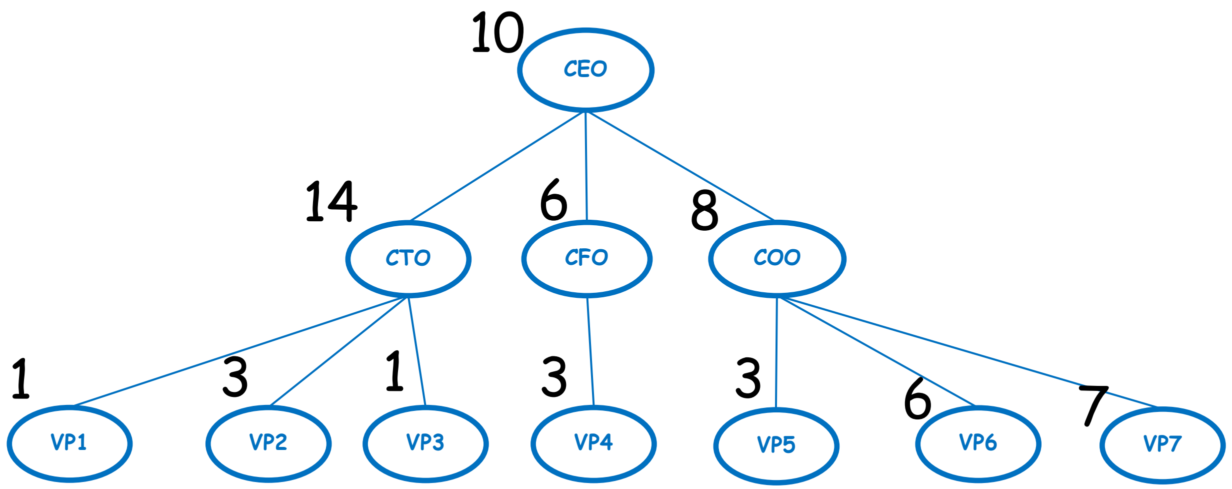
① we invite the boss

score $= 10 + (1+3+1+3+\overset{24}{3}+6+7) = 34$

② we do not invite the boss

score $= 28$

$14 + 6 + (3+6+7) = 36$



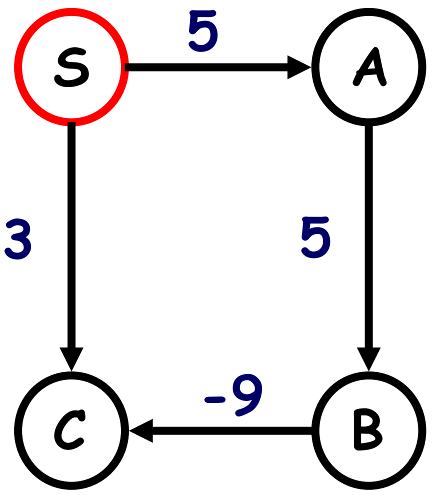
① Let OPT[r] be the max value of a subtree rooted at r.

② $OPT[r] = \max\left(\sum_{\text{child}} [OPT], \; \sum_{\text{grandchild}} [OPT] + V_{\text{root}}\right)$

Base case:

OPT[NULL] = 0

The Shortest Path Problem



Dijkstra's greedy algorithm does not work on graphs with negative weights.

How can we use Dynamic Programming to find the shortest path? We need to somehow defined ordered subproblems, otherwise we may get an exponential runtime.

Intuition

Consider the path (with k edges)

$$v = w_0, w_1, \ldots, w_{k-1}, w_k = u.$$

To have an optimal substructure the following path (k-1 edges)

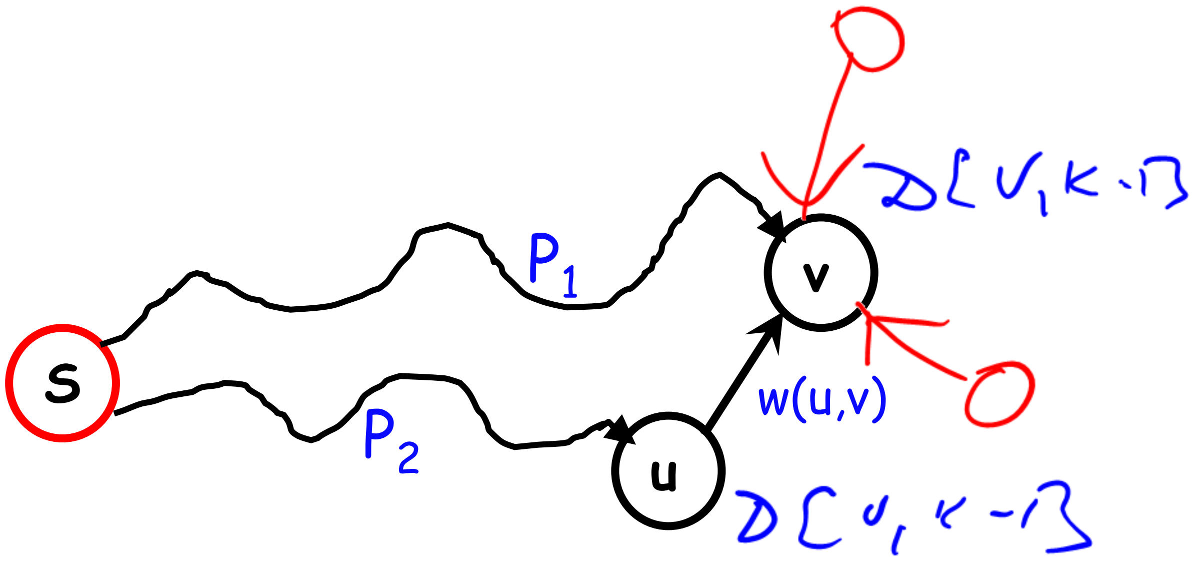
$$v = w_0, w_1, \ldots, w_{k-1}$$

must be a shortest path to w_{k-1}.

Thus, we will be counting *the number of edges* in the shortest path. This is how we order subproblems.

D[v, k] denotes the length of the shortest path from s to v that uses at most k edges.

The Bellman-Ford Algorithm



Case 1: $D[v,k] = D[v,k-1]$

Case 2: $D[v,k] = \min\limits_{u = \text{adj}} \left(D[u,k-1] + w(u,v) \right)$

$$D[v,k] = \mathrm{MIN}\left(D[v,k-1],\ \min_{u=\text{adj.}}\left(D[u,k-1] + w(u,v) \right) \right)$$

$1 \leq k \leq V-1$

$$D[s,k] = 0 \quad , \quad D[v,0] = \underline{\infty}\ !$$

Runtime $= O(\qquad)$

Implementation

D[v,k] denotes the length of the shortest path from s to v that uses at most k edges.

```
D[v,0] = INFINITY; v != s
D[s,k] = 0; for all k

for k=1 to V-1:
  for each v in V:
    for each edge (u,v) in E:
       D[v, k] = min(D[v,k-1] , w(u,v) + D [u,k-1])
```

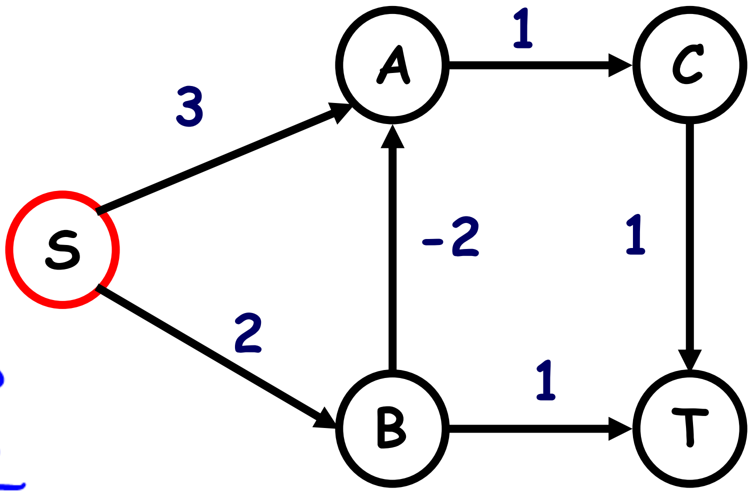
V (answer to slide 26)

O(E)

Runtime - ? $O(V \times V \times E) = O(\cancel{V^2} E)$

$O(V \cdot E)$

Example



$k=1$

$D[A,1]=3$

$D[B,1]=2$

$k=2$ $D[A,2]=\min(3, 2+(-2))=0$

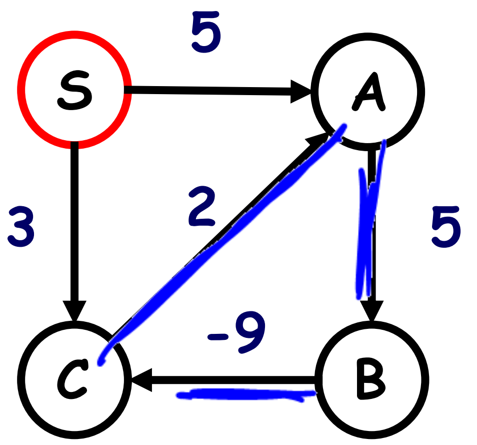
$D[C,2]=\min(\infty, 3+1)=4$

$D[T,2]=3$

$k=3$ $D[C,3]=2-2+1=1$

$k=4$ $D[T,4]=(2-2+1)+1=2$

How would you apply the Bellman-Ford algorithm to find out if a graph has a negative cycle?



ill-poised problem

Discussion Problem 4

There are n trading posts along a river numbered n, n–1..., 1. At any of the posts you can rent a canoe to be returned at any other post downstream. (It is impossible to paddle against the river). For each possible departure point i and each possible arrival point $j < i$, the cost of a rental is $C[i, j]$. However, it can happen that the cost of renting from i to j is higher than the total costs of a series of shorter rentals. In this case you can return the first canoe at some post k between i and j and continue your journey in a second (and, maybe, third, fourth . . .) canoe. There is no extra charge for changing canoes in this way. Give a dynamic programming algorithm to determine the minimum cost of a trip by canoe from each possible departure point i to each possible arrival point j.

① Let OPT[i, j] be the optimal cost of reaching from i to j.

②

$$OPT[i,j] = \min_{K} \left(C[i,K] + OPT[K,j] \right)$$

$i \leq K \leq j$

$$OPT[i,i] = 0$$

③ Runtime — $O(n^3)$

Can you do it in $O(n^2)$?

Let OPT[i] be the optimal cost of reaching from i to point 1.