CSE 3500

HW2 due on 11-8

team 2 on 11-15 @ 5PM

Exam 3 on 12.8. @ 3:30PM

DYNAMIC PROGRAMMING.

1. IDENTIFY A FUNCTION S.T. the solution to the problem at hand is the value of the function at a specific point.

2. WRITE A RECURRENCE Relation for the function

3. START FROM the base cases & compute the values of the function at as many points as needed and finally calculate its value at the point of interest.

0/1 KNAPSACK.

KNAPSACK CAPACITY $= m$.

INPUT: n objects:

p_1	p_2	$\ldots$	p_n	← PROFITS
w_1	w_2		w_n	← WEIGHTS.

Output: $x_1, x_2, \ldots, x_n$ s.t.

$x_i = 0$ OR 1, $\forall i$

$\sum_{i=1}^{n} w_i x_i \le m$, and $\sum_{i=1}^{n} p_i x_i$ IS MAXIMUM.

KNAP(i, j, y):

0/1 KNAPSACK PROBLEM using objects $O_i, O_{i+1}, \ldots, O_j$ when the knapsack capacity is y.

0/1 KNAPSACK: KNAP$(1, n, m)$.

let

① $f_i(y) \triangleq$ the Max. profit for KNAP$(1, i, y)$.

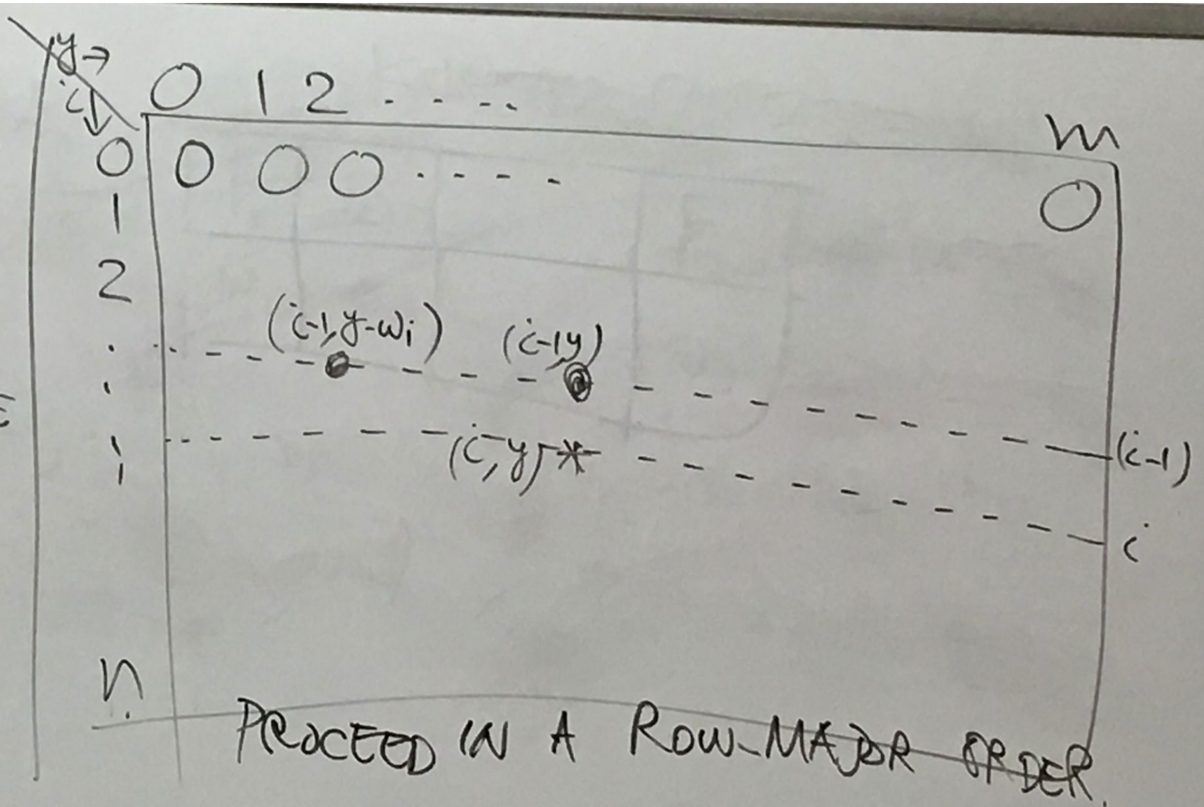
We want to compute: $f_n(m)$.

② $f_i(y) = \text{Max}\left\{ f_{i-1}(y), f_{i-1}(y - w_i) + p_i \right\}$

Base Cases: $f_0(y) = 0 \ \forall y \geq 0.$

$f_0(y) = -\infty$ if $y < 0.$

ASSUME that the WEIGHTS ARE INTEGERS.



$$f_i(y) = \text{Max}\{f_{i-1}(y), f_{i-1}(y-w_i)+p_i\}.$$

Example:

$m = 5$

	1	2	3
PROFITS	4	5	12
WEIGHTS	2	3	4

$f_1(1) = \text{Max}\{f_0(1), f_0(1-2)+4\} = 0$

$f_1(2) = \text{Max}\{f_0(2), f_0(2-2)+4\} = 4.$

$f_1(3) = f_1(4) = 4;\ f_1(5) = 4$

$f_2(0) = 0;\ f_2(1) = \text{Max}\{f_1(1), f_1(1-3)+5\}$
$= 0.$

$f_2(2) = \text{Max}\{f_1(2), f_1(2-3)+5\} = 4.$

ALL PAIRS SHORTEST PATHS PROBLEM:

INPUT: A DIRECTED WEIGHTED GRAPH $G(V,E)$.

Output: The SHORTEST PATH FROM every node to EVERY OTHER NODE.

If there are no negative edges, we can run Dijkstra's alg. n times ($n = |V|$).

RUN TIME $= O(|V|(|V|+|E|)\log|V|)$

We can devise a DYNAMIC PROG. ALG. FOR the GENERAL CASE.

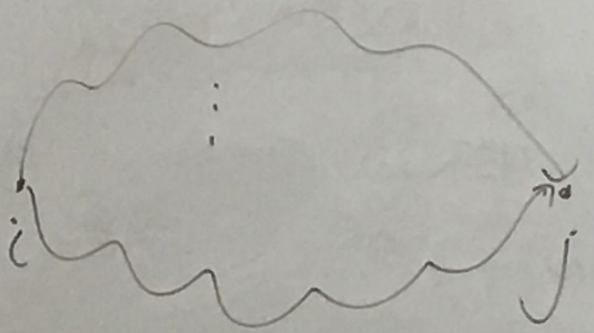
$V = \{1, 2, \cdots, n\}$.

(1) $A^k(i,j) \triangleq$ the LEAST PATH WEIGHT FROM AMONG ALL the i to j paths whose INTERMEDIATE NODES COME FROM $\{1, 2, \cdots, k\}$.

Note that we want to compute $A^n(i,j)$ $\forall i, j \in V$.

<u>INPUT</u>: $A^0(i,j)$ $\forall i, j \in V$.

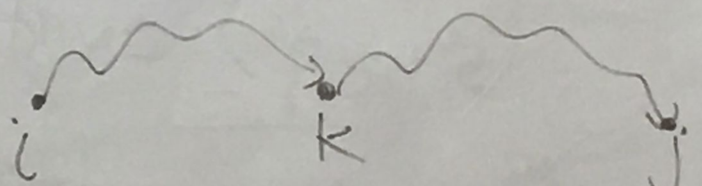
$A^k(i,j)$ $\{1,\dots,k\}$



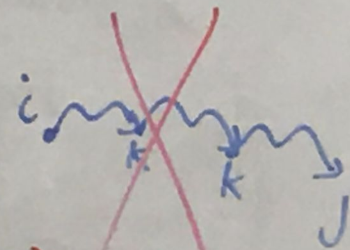
CASE 1: k IS NOT AN INTER-MEDIATE NODE

IN this case $A^k(i,j) = A^{k-1}(i,j)$.

CASE 2: k IS AN INTERMEDIATE NODE.



$$A^k(i,j) = A^{k-1}(i,k) + A^{k-1}(k,j)$$



NO NEGATIVE CYCLES

$$A^{k}(i,j) = \min\left\{ A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j) \right\}$$

START WITH A^{0},

COMPUTE A^{1}

Compute A^{2},

⋮

Compute $A^{n} \Rightarrow$ Output,

```
for i=1 to n do
    for j=1 to n do
        A[i,j]=Cost(i,j);
```

```
for k=1 to n do
    for i=1 to n do
        for j=1 to n do
            A[i,j]=Min{A[i,j],
                A[i,k]+A[k,j]}
```

RUN TIME = $O(n^3)$.

DIJKSTRA'S:

$$O\left(n(m+n)\log n\right)$$

$n=|V|$, $m=|E|$.

IF $m < \frac{n^2}{\log n}$ then Dijkstra's is better.

DYNAMIC PROG.

$$O(n^3).$$

IF $m > \frac{n^2}{\log n}$ DYN. PROG. IS BETTER.

V. WILLIAMS: (2014):

ALL PAIRS SHORTEST PATHS PROBLEM can be solved in

$$O\left(\frac{n^3}{2^{\Omega(\sqrt{\log n})}}\right) \text{ TIME.}$$

CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 18: October 27, 2016

Dynamic Programming

- Steps involved in a typical dynamic programming algorithm are:
  1. Identify a function such that the solution we are looking for is the value of this function at a specific point;
  2. Write a recurrence relation for this function; and
  3. Start from the base case values of the function. Use the recurrence relation to evaluate the function at as many points as needed before the point of interest is reached.

0/1 Knapsack Problem

- Input for this problem are n objects. Object i has a profit of p_i and a weight of w_i, for $1 \leq i \leq n$. We are also given a knapsack of capacity m. The problem is to compute $x_1, x_2, \ldots, x_n$ such that each x_i is either zero or one (for $1 \leq i \leq n$), $\sum_{i=1}^{n} w_i x_i \leq m$, and $\sum_{i=1}^{n} p_i x_i$ is maximum.
- Define Knap(i, j, y) to be a subproblem (of the 0/1 knapsack problem) as follows: Given objects $i, i+1, \ldots, j$ and a knapsack of capacity y, computer the maximum profit obtainable. We note that the 0/1 knapsack problem is nothing but Knap$(1, n, m)$.
- Here is a dynamic programming solution for Knap(i, j, y):
  1. Define $f_i(y)$ to be the solution to Knap$(1, i, y)$.
  2. In an optimal solution to Knap$(1, i, y)$ there are two possibilities: Object i is in the knapsack or the object i is not in the knapsack. If object i is not in the knapsack, then the optimal profit obtainable is $f_i(y) = f_{i-1}(y)$. If object i is in the knapsack, then the maximum profit achievable is $f_i(y) = f_{i-1}(y - w_i) + p_i$. Put together we get:
  $$f_i(y) = \max\{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\}. \tag{1}$$
  3. Now consider the case that the object weights are integers. We can use Equation 1 to compute $f_n(m)$ as follows. We have the following base cases: $f_0(y) = 0$ for all $y \geq 0$ and $f_i(y) = -\infty$ for all $y < 0$ and all i. Let M be a $(n+1) \times (m+1)$ matrix such that $M_{i,j} = f_i(j)$ for $0 \leq i \leq n$ and $0 \leq j \leq m$. Compute this matrix in a row major order. Row 0 and column 0 of this matrix have all zeros.

Note that $M_{i,j}$ depends on two entries in row $(i-1)$. If we proceed in a row major order, these two entries will be available when we are ready to compute $M_{i,j}$ and hence $M_{i,j}$ can be computed in $O(1)$ time (for each entry $M_{i,j}$).

Thus the entire algorithm takes a total of $O(mn)$ time.

An Example

- Consider the following instance of the 0/1 knapsack problem: There are three objects whose profits are $4, 5$, and 12, and whose weights are $2, 3$, and 4, respectively. The knapsack capacity is 5.

  We start with $i = 1$ in Equation 1: $f_1(1) = \max\{f_0(1), f_0(1-2)+4\} = \max\{0, -\infty\} = 0$. $f_1(2) = \max\{f_0(2), f_0(2-2)+4\} = \max\{0, 4\} = 4$. Likewise, $f_1(3) = f_1(4) = f_1(5) = 4$.

  $f_2(0) = 0$. $f_2(1) = \max\{f_1(1), f_1(1-3)+5\} = \max\{0, -\infty\} = 0$. $f_2(2) = \max\{f_1(2), f_1(2-3)+5\} = 4$. $f_2(3) = \max\{f_1(3), f_1(3-3)+5\} = \max\{4, 5\} = 5$. $f_2(4) = \max\{f_1(4), f_1(4-3)+5\} = \max\{4, 5\} = 5$. $f_2(5) = \max\{f_1(5), f_1(5-3)+5\} = \max\{4, 4+5\} = 9$.

  $f_3(0) = 0$. $f_3(1) = \max\{f_2(1), f_2(1-4)+12\} = \max\{0, -\infty\} = 0$. $f_3(2) = \max\{f_2(2), f_2(2-4)+12\} = \max\{4, -\infty\} = 4$. $f_3(3) = \max\{f_2(3), f_2(3-4)+12\} = \max\{5, -\infty\} = 5$. $f_3(4) = \max\{f_2(4), f_2(4-4)+12\} = \max\{5, 12\} = 12$. Likewise we realize that $f_3(5) = 12$ and the final answer is 12.

All Pairs Shortest Paths (APSP) Problem

- Input for the APSP problem is a weighted directed graph $G(V, E)$. The problem is to find the shortest path between every pair of nodes in the graph.

- In the context of path problems in graphs, we typically assume that there are no negative cycles in the graph.

- If there are no negative edges in G, we can use Dijkstra's algorithm to solve the APSP problem. The idea is to invoke Dijkstra's algorithm multiple times, once for each node in the graph as the source. In this case, the run time will be $O(|V|(|V|+|E|)\log|V|)$.

- We can employ dynamic programming to solve the APSP problem (even when the graph has negative edges) as follows. Let $V = \{1, 2, 3, \ldots, n\}$.

  1. For any two nodes i and j in V, define $A^k(i,j)$ to be the weight of the shortest i to j path from among all the i to j paths for which the intermediate nodes come from $\{1, 2, 3, \ldots, k\}$. Input for the APSP problem is $A^0(i,j)$ for $i, j \in V$. We want to compute $A^n(i,j)$, for $i, j \in V$.

2. To derive a recurrence relation for $A^k(i,j)$ note that the shortest i to j path whose intermediate nodes come from $\{1,2,3,\ldots,k\}$ either does not have k as an intermediate node or it has k as an intermediate node. If k is not an intermediate node, then, $A^k(i,j) = A^{k-1}(i,j)$. If k is an intermediate node, then $A^k(i,j) = A^{k-1}(i,k) + A^{k-1}(k,j)$. Put together, we get:

$$A^k(i,j) = \min\{A^{k-1}(i,j),\ A^{k-1}(i,k) + A^{k-1}(k,j)\}. \tag{2}$$

3. We can use Equation 2 to compute $A^n(i,j)$ starting from $A^0(i,j)$. A pseudocode follows.

for $i = 1$ **to** n **do**
    for $j - 1$ **to** n **do**
        $A[i,j] = cost(i,j)$;
for $k = 1$ **to** n **do**
    for $i = 1$ **to** n **do**
        for $j = 1$ **to** n **do**
            $A[i,j] = \min\{A[i,j],\ A[i,k] + A[k,j]\}$;

- Clearly, the run time of the above algorithm is $O(n^3)$.

- If there are no negative edges in the graph, Dijkstra's algorithm can be used to solve the APSP problem in $O(n(m+n)\log n)$ time where $m = |E|$. Dijkstra's algorithm will be faster than the dynamic programming algorithm when $m < \frac{n^2}{\log n}$. Otherwise, the dynamic programming algorithm will be faster.

- In 2014, V. Williams has shown that the APSP problem can be solved in $O\left(\frac{n^3}{2^{\Omega(\sqrt{\log n})}}\right)$ time.