Dynamic Programming

General Idea

- Problem can be divided into stages with a policy decision required at each stage. (Solution is a sequence of decisions)
- Each stage has a number of states associated with it.
- The effect of the policy decision at each stage is to transform the current state into a state in the next stage.
- Given a current state, the optimal policy for the remaining stages is independent from the policy adopted in the previous stages. (The decision at the current stage is based on the results of the previous stage, but decisions are independent)

Dynamic Programming

General Idea

"Principle of optimality" : A decision sequence can be optimal only if the decision sequence that takes us from the outcome of the initial decision is itself optimal.

$\rightarrow$ eliminates "not optimal" subsequences

- The solution procedure begins by finding the optimal policy for the last stage.(backward approach) Then, a recursive relationship that represents the optimal policy for each stage can be generated.

Greedy & Dynamic Programming

Greedy:

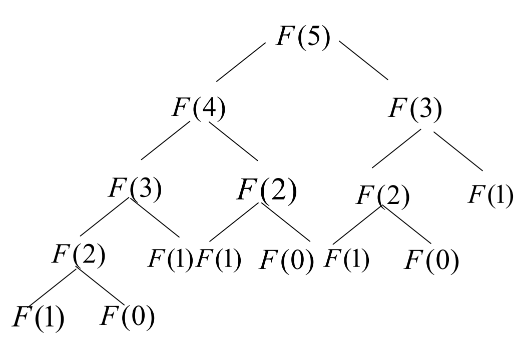
- Consider only 1 seq. of decisions

Dynamic Programming:

- Consider many seq. of decisions
- Try all possibilities

Divide and conquer: divide the problem into subproblems. Solve each subproblem independently and then combine the solutions.

Ex: $\begin{cases} F(0)=0 & n=0 \\ F(1)=1 & n=1 \\ F(n)=F(n-1)+F(n-2) & \forall n \geq 2 \end{cases}$ Assume n = 5



But **Dynamic Programming**: works on phases:

0 1 1 2 3 5 8 . . .

Based on the results in previous phase, make our decision for the current phase

Can be solved in a for-loop

F(0) = 0
F(1) = 1
For i ← 2 to n
   F(i) = F(i-1) + F(i-2);

1. Multistage Single-source Single-destination Shortest Path

The problem:

Given: *m* columns (or stages) of *n* nodes

assume edge only exist between stages

Goal: to find the shortest path from the source (0, 1) to the destination (*m*-1,1)

Stage number

Node number of that stage

The straight forward way:

Time complexity is $O(n^{m-2})$ as there are that many different paths.

The Dynamic Programming way:

Let $m(i, j)$ = length of the shortest path from the source (0,1) to (i, j)

$c(i, k, j)$=distance from (i, k) to $(i+1, j)$

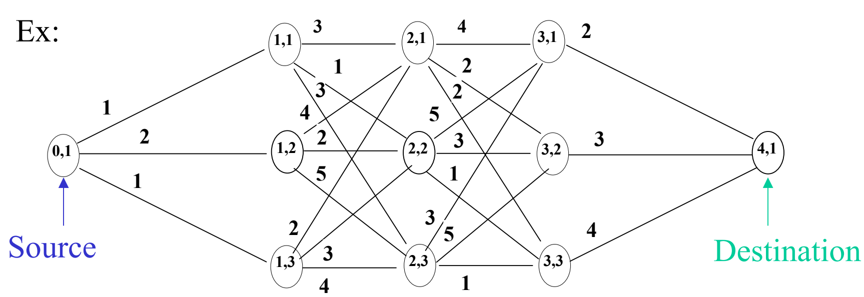
want to find $m(m\text{-}1, 1)$ ← Shortest path of source (0,1) → destination (m-1,1)

define the function equation:

$$m(i,j) = \min_{k=1}^{n}\{m(i-1,k) + c(i-1,k,j)\}$$

$$m(0,1) = 0$$

Ex:



m:

$node(j) \backslash column(i)$	0	1	2	3	4
1	0	1	3	7	7
2		2	2	5	
3		1	4	3	

← Stage #

$$m(2,1) = \min\left\{\overbrace{1+3}^{1}, \overbrace{2+4}^{2}, \overbrace{1+2}^{3}\right\} = 3_3$$

$$m(2,2) = \min\left\{\overbrace{1+1}^{1}, \overbrace{2+2}^{2}, \overbrace{1+3}^{3}\right\} = 2_1$$

$$m(2,3) = \min\left\{\overbrace{1+3}^{1}, \overbrace{2+5}^{2}, \overbrace{1+4}^{3}\right\} = 4_1$$

$$m(3,1) = \min\left\{\overbrace{3+4}^{1}, \overbrace{2+5}^{2}, \overbrace{4+3}^{3}\right\} = 7_{1,2,3}$$

$$m(3,2) = \min\left\{\overbrace{3+2}^{1}, \overbrace{2+3}^{2}, \overbrace{4+5}^{3}\right\} = 5_{1,2}$$

$$m(3,3) = \min\left\{\overbrace{3+2}^{1}, \overbrace{2+1}^{2}, \overbrace{4+1}^{3}\right\} = 3_2$$

$$m(4,1) = \min\left\{\overbrace{7+2}^{1}, \overbrace{5+3}^{2}, \overbrace{3+4}^{3}\right\} = 7_3$$

Remember the previous node of this shortest path

Can be used for constructing the shortest path at the end

The shortest path is: $(0,1) \to (1,1) \to (2,2) \to (3,3) \to (4,1)$

Time complexity is $O(mn)$ to compute $m(i, j)$, each $m(i, j)$ also takes $O(n)$ comparisons.
Total complexity is $O(mn^2)$.

\# of nodes in the graph

$\therefore mn^2 = mn \cdot n = N \cdot n$

Let $N = m \times n$ nodes

max # of stages

max # of nodes in each stage

We have $mn^2 = mn \times n = N \times n$

Note: if use the Dijkstra's algorithm, the complexity is $O(N^2)$

Revisit Dynamic Programming

- The problem can be divided into stages.
- Need to make decisions at each stage.
- The decision for the current stage(based on the results of the previous stage) is independent from that of the previous stages,

  e.g. look at node 3,1 of the multistage shortest path problem, it chooses among 3 decisions from nodes 2,1 or 2,2 or 2,3 but it is independent from how node 2,1 (2,2 or 2,3) makes its decision.

  Note that we find shortest paths from the source to all three nodes 2,1 and 2,2 and 2,3, but some of them may not be used on the shortest path at all.
- But every decision must be optimal, so that the overall result is optimal. This is called "Principle of Optimality".

2. 0/1 Knapsack

The problem:

Given n objects, $p_1, p_2, \ldots, p_n$, profit,

knapsack of capacity M, $w_1, w_2, \ldots, w_n$, weight

Goal: find $x_1, x_2, \ldots, x_n$

s.t. $\sum_{i=1}^{n} p_i x_i$ is maximized subject to $\sum_{i=1}^{n} w_i x_i \le M$

where $x_i = 0$ or 1

The straight forward way:

The complexity is $O(2^n)$

Example: $n = 3$

$(p_1, \quad p_2, \quad p_3) = (1, \quad 2, \quad 5)$

$(w_1, \quad w_2, \quad w_3) = (2, \quad 3, \quad 4)$

$M = 6$

x_1	x_2	x_3	$\sum w_i x_i$	$\sum p_i x_i$
0	0	0	0	0
0	0	1	4	5
0	1	0	3	2
0	1	1	–	–
1	0	0	2	1
1	0	1	6	$\underline{6}$ ← *solution*
1	1	0	5	3
1	1	1	–	–

Does Greedy work?

This problem can't use Greedy (P_i / W_i ↓ order)

Counter-example: 3 objects with $M = 6$,

$(p_1, p_2, p_3) = (5, 3, 3)$

$(w_1, w_2, w_3) = (4, 3, 3)$

The Dynamic Programming way:

Define notation:

$f_i(X)$ = max profit generated from $x_1, x_2, \ldots, x_i$ subject to the capacity X

$$\begin{cases} f_0(X) = 0 \\ f_i(X) = \max\{\underbrace{f_{i-1}(X)}_{x_i = 0}, \underbrace{p_i + f_{i-1}(X - W_i)}_{x_i = 1}\} \end{cases}$$

Example:

$$(p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6) = (w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5 \quad w_6)$$
$$= (100 \quad 50 \quad 20 \quad 10 \quad 7 \quad 3)$$

$M = 165$

Question: to find $f_6(165)$

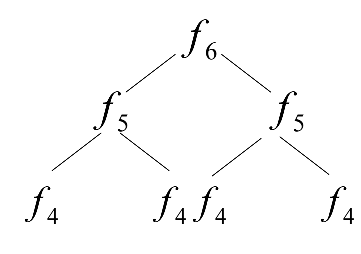
Use backward approach:

$$f_6(165) = \max\{f_5(165), f_5(162) + 3\} = \ldots$$
$$f_5(165) = \max\{f_4(165), f_4(158) + 7\} = \ldots$$

$\ldots$

The result:

$$(x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6) = (1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1)$$



Therefore, the complexity of 0/1 Knapsack is $O(2^n)$

$\longleftarrow M+1 \longrightarrow$

n+1

	0	1	…	$j\text{-}w_i$	…	j	…	M
0	0	0	0	0	0	0	0	0
1								
…								
$i\text{-}1$				$f_{i-1}(j-w_i)$		$f_{i-1}(j)$		
i				$+p_i$		$f_i(j)$		
…								
n								

$(M+1)(n+1)$ entries and constant time each $\Rightarrow O(Mn)$

Example: $M = 6$

	Object 1	Object 2	Object 3	Object 4
p_i	3	4	8	5
w_i	2	1	4	3

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3
2	0 +8	4	4	7	7	7	7
3	0	4	4	7 +5	8	12	12
4	0	4	4	7	9	12	12

Max profit is 12

By tracing which entries lead to this max-profit solution, we can obtain the optimal solution - Object 1,2 and 4.

3. Chained Matrix Multiplication

The problem:

Given $\underset{[d_0 \times d_1]}{M_1} \times \underset{[d_1 \times d_2]}{M_2} \times \ldots \times \underset{[d_{r-1} \times d_r]}{M_r}$

(r matrices)

Goal:

find minimum cost of computing

$M_1 \times M_2 \times \ldots \times M_r$

The idea: Define $C(i, j)$ = cost of computing $M_i \times M_{i+1} \times \ldots \times M_j$

We want $C(1,r)$ be minimum

$$C(i, i) = 0 \quad \forall i$$

$$C(i, i+1) = d_{i-1} \times d_i \times d_{i+1}$$

(the total # of elementary multiplications)

Sequence matrices from i to j:

$$\underbrace{\underset{i}{M_i} \times \underset{i+1}{M_{i+1}} \times \underset{i+2}{M_{i+2}} \times \underset{\ldots}{\ldots} \times \underset{k}{M_k}}_{C(i,k)} \times \underbrace{\underset{k+1}{M_{k+1}} \underset{\ldots}{\ldots} \times \underset{j-1}{M_{j-1}} \times \underset{j}{M_j}}_{C(k+1,\, j)}$$

k is divide point, $i \le k \le j-1$

so, we can divide sequence matrices from i to j into two part with divide point k, $i \le k \le j-1$

$$\begin{array}{ccccccc|ccccc} M_i & \times & M_{i+1} & \times & \dots & \times & M_k & M_{k+1} & \times & \dots & \times & M_j \\ [d_{i-1}\times d_i] & & & & & & [d_{k-1}\times d_k] & [d_k \times d_{k+1}] & & & & [d_{j-1}\times d_j] \end{array}$$

$$[d_{i-1} \times d_k] \qquad\qquad [d_k \times d_j]$$

A **B**

Define the functional equation:

$$C(i,j) = \min_{i \le k \le j-1} \{ C(i,k) + C(k+1,j) + \underbrace{d_{i-1} \times d_k \times d_j}_{\text{Cost of multiplying A and B}} \}$$

Example:

$$\begin{array}{ccccccc} M_1 & \times & M_2 & \times & M_3 & \times & M_4 \\ [3\times 8] & & [8\times 2] & & [2\times 5] & & [5\times 4] \end{array}$$

size 1 $\leftarrow \left[C(i,i) = 0 \qquad \forall i \right.$

size 2 $\leftarrow \left[\begin{array}{l} C(1,2) = 3\times 8\times 2 = 48 \\ C(2,3) = 8\times 2\times 5 = 80 \\ C(3,4) = 2\times 5\times 4 = 40 \end{array} \right.$

$$\text{size } 3 \leftarrow \begin{cases} C(1,3) = \min_{k=1,\underline{2}} \{(C(1,1) + C(2,3) + 3 \times 8 \times 5), \underbrace{(C(1,2) + C(3,3) + 3 \times 2 \times 5}_{\text{Smallest}}\} \\ \quad = 78 \\ C(2,4) = \min_{k=\underline{2},3} \{\underbrace{(C(2,2) + C(3,4) + 8 \times 2 \times 4)}_{\text{Smallest}}, (C(2,3) + C(4,4) + 8 \times 5 \times 4\} \\ \quad = 104 \end{cases}$$

48 + 40 + 24

$$\text{size } 4 \leftarrow C(1,4) = \min_{k=1,\underline{2},3} \begin{Bmatrix} (C(1,1) + C(2,4) + 3 \times 8 \times 4), \\ \boxed{(C(1,2) + C(3,4) + 3 \times 2 \times 4),} \\ (C(1,3) + C(4,4) + 3 \times 5 \times 4 \end{Bmatrix} = 112$$

It has the smallest cost when $k = 2$.
Remember this k value and it can be used for obtaining the optimal solution.

Algorithm:

procedure MinMult (r, D, P) // r, D - inputs, P -output

begin
 for $i \leftarrow 1$ to r do
  $C(i, i) = 0$

 for $d \leftarrow 1$ to $(r-1)$ do
  for $i \leftarrow 1$ to $(r-d)$ do

$$\begin{cases} j \leftarrow i + d; \\ C(i,j) = \min_{i \le k \le j-1} \{C(i,k) + C(k+1,j) + D(i-1) \times D(k) \times D(j)\}; \end{cases}$$

end;

Analysis:

size 1	$j = i + 0$	0	0
size 2	$j = i + 1$	$r - 1$	1
size 3	$j = i + 2$	$r - 2$	2
. . .	. . .	. . .	. . .
size r	$j = i + (r - 1)$	$r - (r - 1) = 1$	r - 1

# of C(i, j)'s is $O(r^2)$

For computing each C(i, j): $O(r)$ *time*

Total complexity is $O(r^3)$