COLLATE

Algorithm Design and Analysis

Notes

UNIT - 2

SOURCE : NOTESHUB

UNIT-2

DYNAMIC Programming

→ It is a problem solving technique like Divide & Conquer.

→ It solves is used when the subproblems are not independent i.e they share same subproblems.

→ It solves each subproblem once & stores the result in a table, so that it can rapidly retrieved if needed.

→ It is often used for optimization problem; problem with many possible soln for which we want to find the optimal (best soln)

→ Dynamic prog. is reverse of Recursion. Recursion is a top down mechanism, we take a problem, split it up & solve the smaller problem.
It is a bottom up mechanism. — we solve small problems & then combine them to obtain a soln to bigger problem.

Characteristics

1. Optimal Substructure: If an optimal soln contain optimal sub soln, then a problem exhibit optimal substructure.

2. Overlapping Subproblems: When a recursive algo would visit the same subproblem repeatedly

then a problem has overlapping subproblems.

* If a problem has optimal substructure, then we can recursively define an optimal soln.

* If a problem has overlapping subproblem, then we can improve on a recursive implementation by computing each subproblem only once.

Appln.

→ Knapsack problem
→ Shortest path problems
→ Matrix Chain Multiplication
→ Longest Common Sequences etc.

⇒ Diff b/w Divide & Conquer & Dynamic prog

Divide & Conquer Algo	Dynamic Prog
1. It is a top-down algorithm.	1. It is a bottom-up algorithm
2. In this subproblems are independent.	2. Subproblems are not independent
3. It is simple as	3. It is often Quite complex

Compared to dynamic programming	Greedy
4. It can be used for any kind of problem	4. It can be used for optimization problems
5. only one decision sequence is ever generated	5. Many decision sequences are generated.
6. It splits a problem in to seperate subprob, Solve the subprob & Combine the result for a soln to the original problem. eg:- Quick sort, Binary search, Merge sort	6. It splits a problem into subproblems, some of which are common, Solves the subproblem & Combines the results for a soln. to the original problem. eg:- MCM, LCS.

```
OBST(p,q,n)
// Given n distinct identifiers a1 < a2 < ... an &
// probabilities p[i], 1 ≤ i ≤ n & q[i], 0 ≤ i ≤ n, this algo
// Computes the cost c[i,j] of OBST tij for identifiers
//  qi+1, ..., qj. It also computes r[i,j], the root of
// tij.
// w[i,j] is the weight of tij
{
 for i := 0 to n-1 do
  {
   // Initialize.
   w[i,i] := q[i]; r[i,i] := 0, c[i,i] = 0.0
   // optimal trees with one node

   w[i,i+1] := q[i] + q[i+1] + p[i+1];
   r[i,i+1] := i+1;
   c[i,i+1] := q[i] + q[i+1] + p[i+1];
  }
 w[n,n] := q[n]; r[n,n] := 0; c[n,n] := 0.0;

 for m := 2 to n do // find optimal trees with m
    nodes)
    for i := 0 to n-m do
     {
     j := i+m;
     w[i,j] := w[i,j-1] + p[j] + q[j];
     k := find (q,r,i,j);
```

// A Value of l in the range $r[i, j-1] \le l$
// $\le r[i+1, j]$ that minimizes $c[i, l-1] + c[l, j]$

$c[i,j] := w[i,j] + c[i, k-1] + c[k, j]$;
$r[i,j] := k$;
}
write ($c[0,n]$, $w[0,n]$, $r[0,n]$);
}

Algorithm Find (c, r, i, j)
{
min := ∞;
for $m := r[i, j-1]$ to $r[i+1, j]$ do
if ($c[i, m-1] + c[m, j] <$ min then
{ min := $c[i, m-1] + c[m, j]$; $l := m$; }
return l;
}

Q. Let $n=4$ and a_1 = do, a_2 = if, a_3 = int, a_4 = while. Let $P(1:4) = (3,3,1,1)$ & $q(0:4) = (2, 3, 1, 1, 1)$. The p's & q's are multiplied by 16 for convenience. Initially, we have $w(i,i) = q(i)$, $c(i,i) = 0$ & $r(i,i) = 0$, $0 \le i < 4$.

Soln. $n = 4$
a_1 = do
a_2 = if
a_3 = int
a_4 = while.

i \ j	0	1	2	3	4
0	$w_{00}=2$ $c_{00}=0$ $r_{00}=0$	$w_{01}=3$ $c_{01}=0$ $r_{01}=0$	$w_{02}=1$ $c_{02}=0$ $r_{02}=0$	$w_{03}=1$ $c_{03}=0$ $r_{03}=0$	$w_{04}=1$ $c_{04}=0$ $r_{04}=0$
1	$w_{11}=8$ $c_{11}=8$ $r_{11}=1$	$w_{12}=7$ $c_{12}=7$ $r_{12}=2$	$w_{23}=3$ $c_{23}=3$ $r_{23}=3$	$w_{34}=3$ $c_{34}=3$ $r_{34}=4$	
2	$w_{02}=12$ $c_{02}=19$ $r_{02}=1$	$w_{13}=9$ $c_{13}=12$ $r_{13}=2$	$w_{24}=5$ $c_{24}=8$ $r_{24}=3$		
3	$w_{03}=14$ $c_{03}=25$ $r_{03}=2$	$w_{14}=11$ $c_{14}=19$ $r_{14}=2$			
4	$w_{04}=16$ $c_{04}=32$ $r_{04}=2$				

$$w(i,j) = p(j) + q(j) + w(i,j-1)$$

$$w(0,1) = p(1) + q(1) + w(0,1-1)$$
$$3 + 3 + w(0,0)$$
$$3 + 3 + 2$$
$$= 8$$

$$C(0,1) = ?$$

$$C[i,j] := w[i,j] + c[i,k-1] + c[k,j];$$

$$C(0,1) := w[0,1] + (c[i,k-1] + c[k,j])$$

Complexity.

We have to Compute $c(i,j)$ for $(j-i) = 1, 2 \cdots n$.

When $j-i = m$. there are $n-m+1$, $c(i,j's)$ to Compute.

Computation of each of these $c(i,j)$'s requires us to find the minimum of m quantities.

$\therefore$ Each $c[i,j)$ can be computed in $O(m)$.

Total time for all $c(i,j)$ with $j-i=m$

$\therefore O(nm-m^2)$

Total time to evaluate all $c(i,j's)$ & $r(i,j's)$

$$\therefore \sum_{1 \le m \le n} (nm - m^2) = O(n^3)$$

0/1 Knapsack

In knapsack problem, a thief robbing a store & can carry a max weight of 'w' in to their knapsack

→ There are 'n' items & i^{th} item weighs w_i & is worth v_i / p_i. Problem is what thief should take.

→ In 0/1 knapsack, items may not be broken in to smaller pieces, so a thief may decide either to take an item or leave it but may not a fraction of an item. Since 0/1 knapsack problem exhibits optimal substructure property, therefore only dynamic programming algorithm

$$C[i,w] = \begin{cases} 0 & \text{if } i=0 \text{ or } w=0 \\ C[i-1, w] & \text{if } w_i > 0 \\ \max\{[v_i + C[i-1, w-w_i], C[i-1, w]]\} & \text{if } i>0 \text{ \& } w \geq w_i \end{cases}$$

It means, if one picks item i, then takes v_i value & then can choose from items $w-w_i$ & get $C[i-1, w-w_i]$ additional value; On the other hand, if he decides not to take item i, then can choose from item 1, 2, -- i-1 up to the weight limit 'w' & get $C[i-1, w]$ value.

Algo takes an input the max. weight w, the no. of items n, & two sequences.

$V = \langle v_1, v_2 \ldots v_n \rangle$ & $w = \langle w_1, w_2, \ldots w_n \rangle$

It stores the $c[i, j]$ values in the table. i.e a 2-D array $c[0..n, 0..w]$

$c[n, w]$ = max. value that can be picked into the knapsack

Eg:- find optimal soln for 0/1 knapsack $n = 3$

$(w_1, w_2, w_3) = (2, 3, 3)$

$(p_1, p_2, p_3) = (1, 2, 4)$

& $m = 6$.

$W = 6$.

item i	p_i	w_i
1	1	2
2	2	3
3	4	3

i \ w	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1
2	0	0	1	2	2	3	3
3	0	0	1	4	4	5	6

for $w = 0$ to W, $i = 1$ to n

$c[0, w] = 0$ & $c[i, 0] = 0$

for $i = 1$, $w_i = w_1 = 2$, $w = 1$

wt of item

item included.

for $i=1$, $w_i = 2$ (given wt. to be included) $w = 1$ (1kg wt.)

$w_i > w$

$2 > 1$

$c[i-1, w] = c[1-1, 1]$

$c[1, 1] \Rightarrow c[0, 1] = 0$

$\boxed{c[1,1] = 0}$

IInd,

$i=1$, $w_i = 2$, $w = 2$

$w_i \not> w$.

$c[1,2] = \max[c[i-1, w], v_i + c[i-1, w-w_i]]$

$= \max[c[1-1, 2], 1 + c[0, 0]]$

$= \max[c[0, 2], 1 + c[0,0]]$

$= \max[0, 1+0]$

$= 1$

IIIrd,

$i=1$, $w_i = 2$, $w = 3$

$w_i \not> w$.

$c[1,3] = \max[c[i-1, w], v_i + c[i-1, w-w_i]]$

$= \max[c[1-1, 3], v_1 + c[0, 3-2]]$

$= \max[c[0,3], 1 + c[0,1]]$

$= \max[0, 1+0]$

$= 1$

Ity $i=1$, $w_i=2$, $w=4$

$w_i \not> w$.

$C[1,4] = \max[C[i-1,w], v_i + C[i-1, w-w_i]]$

$= \max[C[0,4], v_1 + C[1-1, 4-2]$

$= \max[0, 1 + C[0,2]$

$= 1$

IIIy $i=1$, $w_i=2$, $w=5$

$w_i \not> w$.

$C[1,5] = \max[C[i-1,w], v_i + C[i-1, w-w_i]]$

$= \max[C[1-1,5], v_1 + C[1-1, 5-2]$

$= \max[C[0,5], 1 + C[0,3]$

$= 1$

IIIy $C[1,6] = 1$

Now:

When $i=2$, $w_i=3$, $w=1$

Now $w_i > w$

$C[i,w] = C[2,1] = C[i-1,w]$

$= C[2-1,1]$

$C[1,1]$

$= 0$

Now, $i=2$, $w_i=3$, $w=2$

$w_i > w$

$C[i,w] = C[i-1,w]$

$C[2,2] = C[1,2]$

$= 1$

Now, $i = 2$, $w_i = 3$, $w = 3$

$w_i \neq w$.

$$C[i, w] = \max\{C[i-1, w], v_i + C[i-1, w - w_i]\}$$

$$C[2,3] = \max\{C[1,3], v_2 + C[1, 3-3]\}$$

$$= \max\{1, 2 + C[1,0]\}$$

$$\max\{1, 2\}$$

$$= 2$$

Now $i = 2$, $w_i = 3$, $w = 4$.

$w_i \neq w$.

$$C[i, w] = \max\{C[i-1, w], v_i + C[i-1, w - w_i]\}$$

$$C[2,4] = \max\{C[1,4], v_2 + C[1,1]$$

$$= \max\{1, v_2 + 0\}$$

$$= \max\{1, 2\}$$

$$= 2$$

Eg. $C[2,5]$

$w_i = 3$, $w = 5$

$w_i \neq w$

$$C[i, w] = \max\{C[i-1, w], v_i + C[i-1, w - w_i]\}$$

$$= \max\{C[1,5], v_2 + C[1,2]\}$$

$$= \max\{1, 2 + 1\}$$

$$\max\{1, 3\}$$

$$= 3$$

Ily. $C[2,6] = 3$

Now, when $i = 3$, $w_i = 3$, $w = 1$

$w_i > w$

$C[i,w] = C[i-1, w]$

$= C[3-1, 1]$

$C[3,1] = C[2,1]$

$= 0$

Now,

$i = 3$, $w_i = 3$, $w = 2$

$3 > 2$

$C[3,2] = C[3-1, 2]$

$= C[2,2]$

$= 1$

Now $i = 3$, $w = 3$, $w_i = 3$ $V_3 = 4$

$w_i \neq w$

$C[i,w] = \max\{[C(i-1,w), V_i + C[i-1, w-w_i]\}$

$C[3,3] = \max\{C[2,3], V_3 + C[2,0]\}$

$= \max\{2, 4+0\}$

$= \max 2, 4$

$C[3,3] \Rightarrow 4$

Ily. $C[3,4] = 4$

$C[3,5] = 5$

$C[3,6] = 6$

Last Value $i=3$, $\omega=6$,

so,

$C_{max} = C[i, \omega] = C[3,6] \Rightarrow 6$

Now we Construct the optimal soln for finding the items that make the max. Value.

Let $i=n$, $k=W$

i.e $i=3$, $k=6$

if $C[i,k] \neq C[i-1,k]$

then mark i^{th} item as 1

$k = k - \omega_i$

$i = i-1$

else.

mark the i^{th} item as 0.

$i = i-1$

Here.

$C[3,6] = C[2,6]$

$6 \neq 3$

So mark 3rd item as 1

$k = 6 - \omega_3 = 6-3$

$= 3$

& $i = 3-1$

$= 2$

Now.

$C[2,3] = C[1,3]$

$2 \neq 1$

∴ so mark 2nd item as 1

Now $k = 3 - \omega_2$

$3 - 3$

$= 0$

So, we only use 2nd & 3rd items to find

max. profit not 1st object ∴ optimal sol^n is (0, 1, 1)

⇒ FLOYD - WARSHALL Algo

It is also known as Roy-floyd algo, since Bernard Roy described it in 1959 is a graph analysis algo for finding shortest path in a weighted directed graph.

→ Single execution of the algo will find the shortest path b/ω all pairs of vertices.

→ It does so in $\Theta(V^3)$ time where V is the no. of vertices in the graph

→ Negative weight edges may be present, but we shall assume that there are no negative weight cycles.

→ The algo considers the "intermediate vertices of the shortest path, where an intermediate vertex of a simple path $p = \langle v_1, v_2 \dots v_m \rangle$ is any vertex of p. other than v_1 or v_m i.e any vertex in the set $\{v_2, v_3 \dots v_{m-1}\}$

→ Let the vertices of G be $V = \{1, 2 \dots n\}$ & Consider a subset $\{1, 2, \dots k\}$ of vertices for some k.

→ for any pair of vertices $i, j \in V$, Consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2 \dots k\}$ Let 'p' be the min. weight path from among them.

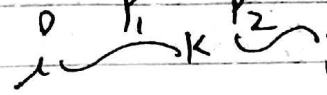
→ Floyd Warshall algo gives a rel'ship b/ω path p' & Shortest path from i to j with all intermediate vertices in the set $\{1, 2 \dots k-1\}$.

→ Relⁿ ship depends on whether or not k is an intermediate vertex of path p.

→ If 'k' is not an intermediate vertex of path p, then all intermediate vertices of path p are in set {1, 2 - - k-1}

→ If 'k' is an ~~int~~ intermediate vertex of path 'p'

then



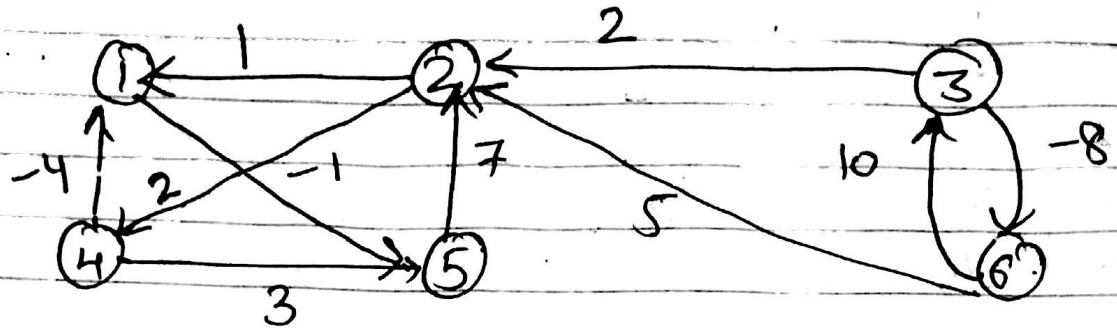
→ Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2 - - k.}

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\left(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\right) & \text{if } k \geq 1 \end{cases}$$

Algo FLOYD-WARSHALL (W)

```
1. n ← rows[W]
2. D⁰ ← W
3. for k ← 1 to n                    } n
4.   do for i ← 1 to n              } n²      ⇒ O(n³)
5.     do for j ← 1 to n             } n³
6.       do d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))  →(1)
7. return D^(n)
```

Q.



Sol^n

$D^{(0)}$ =

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	∞	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	3	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

$D^{(1)}$ =

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	-5	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

$D^{(2)}$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	3	2	0	4	2	-8
4	-4	∞	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	5	0

$D^{(3)}$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	3	2	0	4	2	-8
4	-4	∞	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	6	5	10	7	5	0

$D^{(4)}$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	-2	0	∞	2	-3	∞
3	0	2	0	4	-1	-8
4	-4	∞	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

$D^{(5)}$

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	0	2	0	4	-1	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

$D^{(6)}$

	1	2	3	4	5	6
1	0	6	∞	8	-1	∞
2	-2	0	∞	2	-3	∞
3	-5	-3	0	-1	-6	-8
4	-4	2	∞	0	-5	∞
5	5	7	∞	9	0	∞
6	3	5	10	7	2	0

The shortest path from 3 to 1 is

$3 \rightarrow 6 \rightarrow 2 \rightarrow 4 \rightarrow 1$ & is -5 units.

Binomial Coefficient

It is a non-optimization problem but can be solved using dynamic programming.

We use the notation $\binom{n}{k}$ (reads 'n chooses k') or $C(n,k)$

i.e. $(a+b)^n = C(n,0)a^n + \cdots + C(n,k)a^{n-k}b^k + \cdots C(n,n)b^n$

or

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

These variables or numbers are called Binomial Coefficients due to their appearance in the binomial expansion.

Special Case

when $a = b = 1$;

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

Recursive formula for Computation

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{for all integers } n, k \text{ with } 1 \le k \le n-1.$$

with initial / boundary values.

$$\binom{n}{0} = \binom{n}{n} = 1 \quad \text{for all integers } n \ge 0$$

→ formula follows from considering the set $\{1, 2, \ldots n\}$ & Counting seperately
(a) the k-element grouping that include a particular set element say "i" in every group.
(b) all the k^{th} groupings that don't include "i";

⇒ Multiplicative formula

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} = \frac{n(n-1)(n-2)\cdots(n-(k-1))}{k(k-1)(k-2)\cdots 1}$$

$$= \prod_{i=1}^{k} \frac{n-(k-i)}{i}$$

$$= \prod_{i=1}^{k} \frac{n+1-i}{i},$$

Where numerator of the 1st func^n $(n^{\underline{k}})$ is expressed as a falling factorial power.
→ Numerator gives the no. of ways to select a sequence of k-distinct objects, retaining the order of selection from a set of n-objects
→ Denominator counts the no. of distinct sequences that define the same k-combination when order is disregarded

⇒ factorial formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \le k \le n$$

Where $n!$ → factorial of n.

Generalization & Connection to the Binomial Series

The Multiplicative formula allows the definition of binomial coefficients to be extended by replacing 'n' by an arbitrary number 'α' or even an element of any commutative ring in which all +ve integers are invertible.

$$\binom{\alpha}{k} = \frac{\alpha^{\underline{k}}}{k!} = \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-k+1)}{k(k-1)(k-2)\cdots 1} \quad \text{for } k \in \mathbb{N}$$

& Arbitrary 'α'