Dynamic Programming

chandakmb@gmail.com, hodcs@rknec.edu
www.mbchandak.com

Topics:

Longest Common Subsequence

Optimal Binary Search Tree

0/1 Knapsack problem

Chained Matrix Multiplication

Travelling Salesman Problem

String Editing

Characteristics

- **Developed by Richard Bellman in 1950.**
- **To provide accurate solution with the help of series of decisions.**
- **Define a small part of problem and find out the optimal solution to small part.**
- **Enlarge the small part to next version and again find optimal solution.**
- **Continue till problem is solved.**
- **Find the complete solution by collecting the solution of optimal problems in bottom up manner.**

Characteristics

- Types of problems and solution strategies.
- **1. Problem can be solved in stages.**
- **2. Each problem has number of states**
- **3. The decision at a stage updates the state at the stage into the state for next stage.**
- **4. Given the current state, the optimal decision for the remaining stages is independent of decisions made in previous states.**
- **5. There is recursive relationship between the value of decision at a stage and the value of optimum decisions at previous stages.**

Characteristics

- How memory requirement is reduce:
- How recursion overheads are converted into advantage
- - By storing the results of previous n-2 computations in computation of n-1 stage and will be used for computation of “n” stage.
- Not very fast, but very accurate
- It belongs to smart recursion class, in which recursion results and decisions are used for next level computation. Hence not only results but decisions are also generated.
- **Final collection of results: Bottom Up Approach.**

Longest Common Subsequence

- Given two strings of characters, LCS is a method to find a longest subsequence either continuous or non-continuous and is common in both the strings. The subsequence generation starts from comparison of first character of both the given strings.
- *The two strings may or may not be of equal length.*
- Let “A” and “B” be two strings of length “m” and “n” respectively, and let “i” and “j” be two pointers to handle the two strings.
- Let matrix c[m,n] be the storage matrix to store the results of comparison related with two strings. The C[m,n] matrix will be handled using pointer “i” and “j”.

If i=0 and j=0, then c[i,j] = 0

If a[1..i] and b[1..j] is compared and a[i] is not equal to b[j]. Then the comparison will be either

In between

a[1..i-1] and b[1..j] or a[1..i] and b[1..j-1]
Then

c[i,j] = max[c[i-1,j], c[I,j-1]]

If a[1..i] and b[1..j] is compared and a[i] is equal to b[j].

Then c[i,j] = c[i-1,j-1] + 1

Example

STRING A = G D V E G T A B = G V C E K S T

i/j	0	1G	2V	3C	4E	5K	6S	7T
0	0/H	0/H	0/H	0/H	0/H	0/H	0/H	0/H
1G	0/H	**1/D**	1/S	1/S	1/S	1/S	1/S	1/S
2D	0/H	1/U	1/U	1/U	1/U	1/U	1/U	1/U
3V	0/H	1/U	**2/D**	2/S	2/S	2/S	2/S	2/S
4E	0/H	1/U	2/U	2/U	**3/D**	3/S	3/S	3/S
5G	0/H	1/D	2/U	2/U	3/U	3/U	3/U	3/U
6T	0/H	1/U	2/U	2/U	3/U	3/U	3/U	**4/D**
7A	0/H	1/U	2/U	2/U	3/U	3/U	3/U	**4/U**

Example:2

STRING A = X M J Y A U Z B = M Z J A W X U

i/j	0	1X	2M	3J	4Y	5A	6U	7Z
0	0/H	0/H	0/H	0/H	0/H	0/H	0/H	0/H
1M	0/H	1/U	1/D	1/S	1/S	1/S	1/S	1/S
2Z	0/H	1/U	1/U	1/U	1/U	1/U	1/U	2/D
3J	0/H	1/U	1/U	2/D	2/S	2/S	2/S	2/S
4A	0/H	1/U	1/U	2/U	2/U	3/D	3/S	3/S
5W	0/H	1/U	1/U	2/U	2/U	3/U	3/U	3/U
6X	0/H	1/D	1/U	2/U	2/U	3/U	3/U	3/U
7U	0/H	1/U	1/U	2/U	2/U	3/U	4/D	4/S

Algorithm:

- Assumptions: Let the two strings be present in array “a” size “m” and array “b” size “n” handled using pointers “i“ and “j”.
- The resultant array will be c[m,n] : one instance c[i,j]. The array “c” will be structure: c[i,j].val and **c[i,j].dir = “u”, “s”, “d” and “h”**

```
Algorithm lcs (a,b: c)
{
   for i = 0 to m do
     for j = 0 to n do
      if a[i]=0 or b[j]=0
{
     c[i,j].val = 0;
     c[i,j].dir=‘h’
}
else
{
  if (a[i] ≠ b[j])
    c[i,j].val = max [c[i-1, j].val, c[i,j-1].val]
     if(c[i-1, j].val >= c[i,j-1].val)
       c[i,j].dir = ‘u’
     else
       c[i,j].dir=‘s’
  else
     c[i,j].val = c[i-1,j-1].val  + 1
     c[i,j].dir = ‘d’
}
```

Algorithm:

- Printing LCS: Assumptions: Let “A” be the given array.
- Let “C” be the array containing the details of LCS

```
Algorithm print_lcs (a,c,i,j)
{
   if (i=0 or j=0)
      return;
   if(c[i,j] = ‘d’
   {
     print_lcs(a,c,i-1,j-1)
     print(a[i])
   }
   else
   {
      if(c[i,j] = ‘u’
      print_lcs(a,c,i-1,j)
     else
      print_lcs(a,c,i,j-1)
   }
```

Travelling Salesman Problem: (Self Study topic)

- Principle: To find out cycle in a given graph by visiting all the vertices once and reaching back to source vertex.
- Constraint: A vertex or an edge can be only visited only once.
- The total length of cycle should be minimum.
- Applied on complete graph (a matrix with only diagonal element as zero).

Formulation

- Formula used:
- g(i,S) = min {Cij + g(j – ***(S - {j})***)
- j€S
- ***In this formula:***
- “i" represents source vertex
- “j” represents the next vertex used as intermediate.
- “S” represents the vertices other than “i" and if out of these vertices “j” vertex is used as next intermediate, then remaining vertices left will be ***S - {j}.***

Example

0	10	15	20
5	0	9	10
6	13	0	12
8	8	9	0

1			*	1
1		**		1
1	***			1
1				1

Example

- The path will consist of FIVE vertices: in which FIRST and LAST vertex will be SAME (SOURCE)

- Step: 1: To perform computation with zero intermediates an reach to Source, i.e., for position one place before the SOURCE

- Possible vertices which can be used are 2, 3, 4
- g(2, *) = C21 = 5
- g(3,*) = C31 = 6
- g(4,*) = C41 = 8

Example:

- **Step 2:**
- Vertex 2 at pre-to-pre-final position

g(2, {3}) = C23 + g(3,*) = 9+6=15

g(2, {4}) = C24 + g(4,*) = 10 + 8 = 18

- Vertex 3 at pre-to-pre-final position

g(3, {2}} = {C32 + g(3, *) } = 13 + 5 = 18

g(3, {4}} = {C34 + g(4, *) } = 12 + 8 = 20

- Vertex 4 at pre-to-pre-final position

g(4, {2}} = {C42 + g(2, *) } = 8 + 5 = 13

g(4, {3}} = {C43 + g(3, *) } = 9 + 6 = 15

Example:

- **Step 3: This step will allow two intermediates**

g (2, {3,4}) = $\min\{C23 + g\{3,4\}\ ,\ C24 + g\{4,3\}) = 25$

g(3, {2,4}) = $\min\{C32 + g\{2,4\}\ ,\ C34 + g\{4,2\}) = 25$

g (4, {2,3}) = $\min\{C42 + g\{2,3\}\ ,\ C43 + g\{3,2\})$ = 23

Step 4: The path from the source vertex

g (1, {2,3,4} = min{C12 + g{2,{3,4}}, C13 + g{3, {2,4}}, C14 + g{4, {2,3}}}

= min {10+25, 15+25, 30+23} = 35

Final Path:

1	2	4	3	1