Dynamic Programming

It is typically applied to optimization problems

Optimization problems – the solutions are plenty with each solution there is an associated value-it can be cost or profit

The solution that maximizes profit or minimizes cost is known as optimal solution

Solving Optimization problem is to get optimal solution

A greedy approach may not always give an optimal solution for example 0/1 knapsack problem

A brute force approach is generating all solutions and then finding amongst them the one having lowest cost or highest profit.

This will require lot of effort

Effort can be reduced in two ways

- In obtaining solutions a strategy similar to divide and conquer strategy can be used

  Divide problem into subproblems, find the solution of subproblems and combine to get a solution of main problem

  Since the process is carried out to get all possible solutions, there is possiblity of overlapping subproblems, same subproblem a part of two or more subproblems

  The solving of same subproblem repetatively can be avoided by storing its value once it is computed and using all over again
- Avoid solving those subproblems which are no way anywhere near optimal solution by attaching value with each subproblem

  Solve only those subproblems which contain subproblems having best value

Matrix-Chain Multiplication problem

Given a sequence(chain) (A_1,A_2,……A_n) of n matrices , the product $A_1A_2A_3…A_n$ is to be computed.

There is already an algorithm to compute the product of two matrices which is to be repeatedly applied

The matrix multiplication algorithm for two matrices of order nxm by mxl involves nxmxl multiplications which is the cost of multiplying two matrices

The cost of multiplying n matrices will depend on how the chain is parenthesised

Ex. N=3 (A_1,A_2,A_3) A_1=10x100 A_2=100x10 A_3=10x100

1 [[A_1 A_2] A_3]

The product [A_1,A_2] involves 10x100x10=10000 multiplications giving a matrix which is 10x10

The product of this 10x10 matrix with A3 involves 10x10x100 multiplications

Total multiplications= 10000+10000=20000

2 $[A_1[A_2 A_3]]$

The product $[A_2,A_3]$ involves 100x10x100=100000 multiplications giving a matrix which is 100x100

The product of A_1 with this 100x100 matrix involves 10x100x100 multiplications

Total multiplications= 100000+100000=200000

Thus the way the chain is parenthesized decides the cost of evaluating the product

The optimal solution to Matrix chain multiplication problem is that parenthesization that gives minimum cost (minimum number of multiplications)

Given a set of n matrices there can be many ways of parenthesizing it

1 $[[A_1 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots A_7][A_8 \ldots\ldots\ldots\ldots\ldots\ldots A_{16}]]$

$[[[A_1 \ldots\ldots \; A_4][A_5 \ldots\ldots A_7][[A_8 \ldots A_{10}][A_{11} \ldots A_{16}]]$

2 $[[A_1 \ldots A_2][A_3 \ldots \ldots\ldots\ldots\ldots\ldots \ldots\ldots\ldots\ldots\ldots A_{16}]]$

$[[A_1 \ldots A_2][A_3 .. [[A_5 \qquad A_7][[A_8 \ldots A_{10}][A_{11} \ldots A_{16}]]$

Dynamic Programming can be applied to those problems which exhibit optimal substructure property

A problem exhibits optimal substructure property if an optimal solution to the problem contains within it optimal solution to sub problems

Development of Dynamic programming algorithm can be broken into four steps

1. Characterize the optimal solution in terms of optimal solution of sub problems

Let us denote By A_{ij} the matrix obtained from evaluating the product $A_i A_{i+1} \ldots . A_j$ in an optimal fashion

Thus A_{ij} denotes optimal solution of a sub problem and the optimal solution we are interested in is A_{1n}

A_{1n} is decomposed into two sub problems A_{1k} and A_{k+1n}

Unless A_{1k} and A_{k+1n} is optimal , A_{1n} will not be optimal

Thus the optimal solution is characterized in terms of optimal solution of sub problems

2. Recursively define the value of an optimal solution

The value should be such that it can be used as an optimization criteria to discard sub problems

Let mij denote the minimum number of multiplications required to compute matrix chain multiplication Aij

We define mij recursively

m_{ij} =0 if i=j

m_{11} is the number of multiplications for A_{11} $[A_1]$ =0

Since A_{ij} can be expressed as $[A_{ij}]=[[A_{ik}][A_{k+1j}]]$

There are several ways we can choose k , $i \le k < j$

Total multiplications for A_{ik} are M_{ik} and similarly for A_{k+1j} are M_{k+1j}

Let A_1 be $p_0 x p_1$ A_2 be $p_1 x p_2$ ………….A_n be $p_{n-1} x p_n$

The total multiplication will be

$$m_{ik}+m_{k+1j} + p_{i-1} x p_k x p_j$$

$$m_{ij} = \begin{cases} 0 & \text{if } i=j \\ \min_{i \le k < j}(m_{ik}+m_{k+1j} + p_{i-1} * p_k * p_j) & \text{if } i \ne j \end{cases}$$

3. Compute the values in bottom up fashion and storing them so that they can be reused

We will use a nxn table to store the calculated costs mij

Ex n=5 $A_1 = 5x10$ A_2 10x10 $A_3 = 10x5$ $A_4 = 5x20$ $A_5=20x5$

$(p_o,p_1,p_2,p_3,p_4,p_5)=(5,10,10,5,20,5)$

We are interested in calculating m_{1n} which is minimum number of multiplications required to calculate A_{1n}

We go in bottom up manner first calculating m_{11}

$m_{ij}=0$ if i=j

for i= 1 to n do m[i][i]=0

$m_{12} = \min_{1\leq k < 2}(m_{ik}+m_{k+1j} + p_{i-1} \times p_k \times p_j)$

$= m_{11}+m_{22}+p_o x p_1 x p_2=0+0+5x10x10=500$

$m_{23}=m_{22}+m_{33}+p_1 x p_2 x p_3=0+0+10x10x5=500$

$m_{34}=m_{33}+m_{44}+p_2 x p_3 x p_4=0+0+10x5x20=1000$

$m_{45}=m_{44}+m_{55}+p_3 x p_4 x p_5=0+0+5x20x5=500$

m_{ij}	1	2	3	4	5
1	0	500	750		
2		0	500	1500	
3			0	1000	
4				0	500
5					0

$(p_o,p_1,p_2,p_3,p_4,p_5)$
$=(5,10,10,5, 20, 5)$

$$m_{13}=\min_{1\leq k<3}(m_{ik}+m_{k+1j} + p_{i-1}xp_kxp_j)$$

$$=\min\begin{cases} m_{11}+m_{23} +p_oxp_1xp_3=0+500+5x10x5 \\ m_{12}+m_{33}+p_0xp_2xp_3=500+0+5x10x5 \end{cases}$$

$$= \min(750,750)=750$$

$$m_{24}=\min_{2\leq k<4}(m_{ik}+m_{k+1j} + p_{i-1}xp_kxp_j)$$

$$=\min\begin{cases} m_{22}+m_{34} +p_1xp_2xp_4=0+1000+10x10x20 \\ m_{23}+m_{44}+p_1xp_3xp_4=500+0+10x5x20 \end{cases}$$

$$= \min(3000,1500)=1500$$

m_{ij}	1	2	3	4	5
1	0	500	750	1250	
2		0	500	1500	
3			0	1000	750
4				0	500
5					0

$(p_o,p_1,p_2,p_3,p_4,p_5) = (5,10,10,5,20,5)$

$m_{35} = \min_{3 \le k < 5}(m_{ik} + m_{k+1j} + p_{i-1} x p_k x p_j)$

$$= \min \begin{cases} m_{33}+m_{45} + p_2xp_3xp_5 = 0+500+10x5x5 \\ m_{34}+m_{55}+p_2xp_4xp_5 = 1000+0+10x20x5 \end{cases}$$

$= \min(750,2000) = 750$

$m_{14} = \min_{1 \le k < 4}(m_{ik} + m_{k+1j} + p_{i-1} x p_k x p_j)$

$$= \min \begin{cases} m_{11}+m_{24} + p_0xp_1xp_4 = 0+1500+5x10x20 \\ m_{12}+m_{34} + p_0xp_2xp_4 = 500+1000+5x10x20 \\ m_{13}+m_{44}+p_0xp_3xp_4 = 750+0+5x5x20 \end{cases}$$

$= \min(2500,2500,1250) = 1250$

m_{ij}	1	2	3	4	5
1	0	500	750	1250	1375
2		0	500	1500	1250
3			0	1000	750
4				0	500
5					0

$(p_o,p_1,p_2,p_3,p_4,p_5)$
$=(5,10,10,5,20,5)$

$$m_{25}=\min_{2\le k<5}(\ m_{ik}+m_{k+1j}+p_{i-1}xp_kxp_j)$$

$$=\min\begin{cases} m_{22}+m_{35}+p_1xp_2xp_5=0+750+10x10x5 \\ m_{23}+m_{45}+p_1xp_3xp_5=500+500+10x5x5 \\ m_{24}+m_{55}+p_1xp_4xp_5=1500+0+10x20x5 \end{cases}$$

$$=\min(1250,1250,2550)=1250$$

$$m_{15}=\min_{1\le k<5}(\ m_{ik}+m_{k+1j}+p_{i-1}xp_kxp_j)$$

$$=\min\begin{cases} m_{11}+m_{25}+p_0xp_1xp_5=0+1250+5x10x5 \\ m_{12}+m_{35}+p_0xp_2xp_5=500+750+5x10x5 \\ m_{13}+m_{45}+p_0xp_3xp_5=750+500+5x5x5 \\ m_{14}+m_{55}+p_0xp_4xp_5=1250+0+5x20x5 \end{cases}$$

$$=\min(1500,1500,1375,1750)=1375$$

m_{ij}	1	2	3	4	5
1	0	500	750	1250	1375
2		0	500	1500	1250
3			0	1000	750
4				0	500
5					0

```
    Algorithm MatrixChain(P,M,n)
{       for i= 1 to n do M[i][i]=0//multiplications of size 1
        for l=2 to n //multiplications of size l
            { for i = 1 to n -l +1 // number of multiplications
                { j=i+l-1
             M[i][j]=∞
                  for k= i to j-1
                   {q=M[i][k]+M[k+1][j]+p[i-1]xp[k]xp[j]
                  if q<M[i][j] then M[i][j]=q }}}
  return m}
```

The running time of the algorithm is $O(n^3)$

Space complexity $\theta(n^2)$ space to store the values

4. Construct an optimal solution from computed information

apart from value of m we must also store the value of k

m_{ij}	1	2	3	4	5
1	0	500 1	750 1	1250 3	1375 3
2		0	500 2	1500 3	1250 2
3			0	1000 3	750 3
4				0	500 4
5					0

[A1….A5]

3

[[A1…A3][A4…A5]]

1 4

[[[A1][A2A3]][A4A5]]

Print optimal parents

- print_optimal_parents(S, i, j)

```
{ if i = j
then print "Ai"
else
print " [ "
print_optimal_parents(S, i , S[ i , j ])
print_optimal_parents(S,S[ i , j ] + 1 , j )
print " ] "
}
```

```
Algorithm MatrixChain(P,M,S,n)
{         for i= 1 to n do
              { S[i][j]=0 M[i][i]=0    }
          for l=2 to n //multiplications of size l
              { for i = 1 to n -l +1 // number of multiplications
                     { j=i+l-1
                M[i][j]=∞
                         for k= i to j-1
                          {q=M[i][k]+M[k+1][j]+p[i-1]xp[k]xp[j]
                         if q<M[i][j] then
                           {M[i][j]=q
                            S[i][j]=k
                            }
                          }  }
                }
  return m}
```

Additional space of $\theta(n^2)$ for S

Once Sij are calculated the optimal solution can be calculated using following recursive procedure

```
Algorithm MatrixChainMultiply(A,S,i,j)
If  i < j then
    { x=MatrixChainMultiply(A,S,i,s[i][j])
    y=MatrixChainMultiply(A,S, s[i][j]+1,j)
    return MatrixMultiply[x,y] // return” [“+X,Y+”]”
        }
else return Ai // [Ai]
}
```

MCM (A,S,1,5)

[[[A1][[A2][A3]]] [[A4][A5]]]

MCM (A,S,1,3)

[[A1][[A2][A3]]]

MCM (A,S,4,5)

[[A4][A5]]

MM (A,S,1,1)

[A1]

MCM (A,S,2,3)

[[A2][A3]]

MCM (A,S,2,2)

[A2]

MCM (A,S,3,3)

[A3]

MCM (A,S,4,4)

[A4]

MCM (A,S,5,5)

[A5]

0/1 Knapsack Problem

The solution is a set of values $x_1, x_2, \ldots. x_n$ such that

$\sum p_i x_i$ is maximized and $\sum w_i x_i \leq m$

1. Let Knap(i, j, y) denote the knapsack problem with solution $x_i, \ldots.. x_j$ with $\sum p_i x_i$ is maximized and $\sum w_i x_i \leq y$

The Knapsack problem is represented by Knap(1,n,m)

If $y_1 y_2 \ldots. y_n$ is optimal solution of Knap(1,n,m) and

If $y_1=0$ then $y_2 \ldots y_n$ must be optimal solution of knap(2,n,m) and

If $y_1=1$ then $y_2..y_n$ must be optimal solution of knap(2,n,m-w_1)

Generalizing

If $y_i y_{i+1} \ldots. y_n$ is optimal solution of Knap(i, n, m) and

If $y_i=0$ then $y_{i+1} \ldots y_n$ must be optimal solution of knap(i+1,n,m) and If $y_i=1$ then $y_{i+1}..y_n$ must be optimal solution of knap(i+1,n,m-w_i)

Thus the optimal solution of problem can be expressed in terms of optimal solution to subproblem

Alternatively

If y1y2….yj is optimal solution of Knap(1,j,m) and

If yj=0 then y1…yj-1 must be optimal solution of knap (1,j-1,m) and If yj=1 then y1..yj-1 must be optimal solution of knap(1,j-1,m-w_j)

2. Let $g_j(y)$ be the value of the optimal solution to the knapsack problem knap(j+1,n,y) which is the profit earned

$g_0(m)$ is the value of optimal solution to knap(1,n,m)

$g_n(m)$ is the value of optimal solution to knap(n+1,n,m)

$g_n(y) = 0$

To avoid knapsack capacity to be considered as negative

$g_n(y)= -\infty$ if $y <0$ and $g_n(y)= 0$ if $y \geq 0$

$g_i(y)= \max \{ g_{i+1}(y) , g_{i+1}(y-w_i)+p_i\}$

Alternatively let $f_j(y)$ be the value of optimal solution to the knapsack problem knap(1,j,y)

$f_0(y)=0$ $y\geq0$ $f_0(y) = -\infty$ $y<0$ $f_j(y)= \max\{ f_{i-1}(y), f_{i-1}(y-w_i)+p_i\}$

3. Computing the values in bottom up manner

Consider the knapsack instance $n=4$, $m=20$

$(w1,w2,w3,w4)=(16,12,8,6)$ and

$(p1,p2,p3,p4)=(32,20,14,9)$

Traveling Salesperson Problem

The traveling Salesperson problem is to find a tour of minimum cost

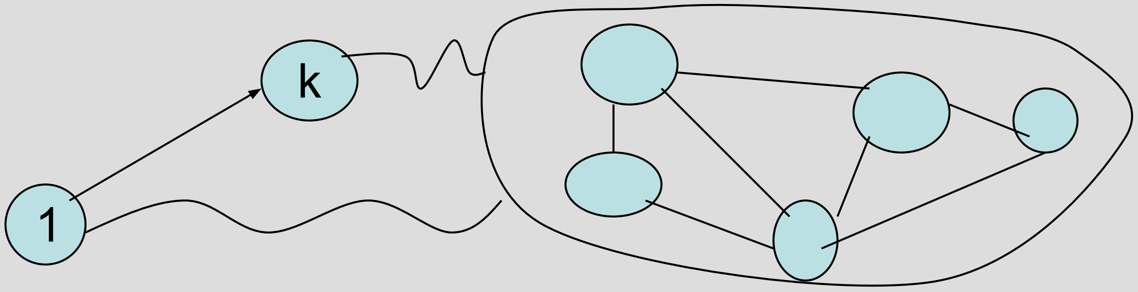
Let $G=(V,E)$ be a directed graph with $|V|=n > 0$ and edge costs c_{ij} where $c_{ij} > 0$ for i and j and $c_{ij} = -\infty$ if (i,j) not in E.

A tour of G is a directed simple cycle that includes every vertex in V.

The cost of tour is sum of the cost of edges on the tour

Without loss of generality , assume tour starts and ends at vertex 1.

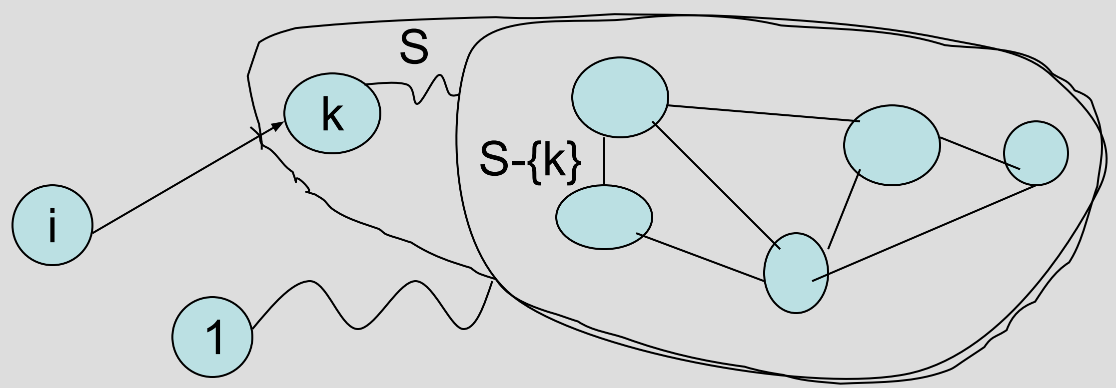
Every tour consists of an edge (1,k) for some k in V-{1} and a path from vertex k to 1 going through each vertex in V-{1,k} exactly once



1 Let us denote by T(i ,s) the problem of starting at vertex i and traveling through all vertices in set S and then reaching back to 1 as the tour should always end at 1 .

The optimal solution to T(i ,S) will involve going from i to some vertex k and then travelling through alll vertices in S-{k} in an optimal manner which is the optimal solution to the problem T(k,S-{k})

Thus the problem satisfies optimal substructure property



2 let g(i, S) denote the cost of the optimal tour starting from i, visiting all vertices in s and ending at 1.

The TSP problem is to get g(1,S)

If S is empty ,g(i,φ) will be the cost of the tour starting at i visiting no vertices as S is empty and reaching 1. Thus the cost is of directly moving from i to 1 i.e. c_{i1}

$g(i, \varphi)= c_{i1} \quad 1 \le i \le n$

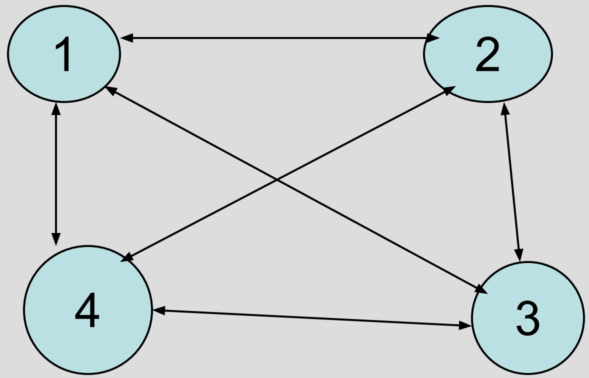
If S is not empty . The cost of moving from i to some k is c_{ik} and then g(k, S-{k}) is the optimal cost of reaching 1 after travelling through remaining vertices.

We can choose that k in S which gives the minimum cost

$$g(i,S)= \min_{k \in S} \{ c_{ik} + g(k, S-\{k\})$$

3 . Compute the values in bottom up manner

Consider a TSP instance given by following graph and cost matrix



$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

$g(2, \varphi)= c21=5$

$g(3, \varphi)= c31=6$

$g(4, \varphi) = c41 =8$

Next consider singleton sets {2} {3} and {4}

$g(i,S)= \min_{k \, \varepsilon \, S} \{ c_{ik} +g (k, S-\{k\})$

$g(2, \{3\})= \min_{k \, \varepsilon \{3\}} (c_{2k} + g(k, \{3\}-\{3\})= c_{23}+g(3, \varphi)$

$= 9+6=15$

$g(2, \{4\}) = \min_{k\ \varepsilon\{4\}} (c_{24} + g(k, \{4\}-\{4\}) = c_{24}+g(4, \varphi)$

$= 10+8=18$

$g(3, \{2\}) = \min_{k\ \varepsilon\{2\}} (c_{3k} + g(k, \{2\}-\{2\}) = c_{32}+g(2, \varphi)$

$= 13+5=18$

$g(3, \{4\}) = \min_{k\ \varepsilon\{4\}} (c_{3k} + g(k, \{4\}-\{4\}) = c_{34}+g(4, \varphi)$

$= 12+8=20$

$g(4, \{2\}) = \min_{k\ \varepsilon\{2\}} (c_{4k} + g(k, \{2\}-\{2\}) = c_{42}+g(2, \varphi)$

$= 8+5=13$

$g(4, \{3\}) = \min_{k\ \varepsilon\{3\}} (c_{4k} + g(k, \{3\}-\{3\}) = c_{43}+g(3, \varphi)$

$= 9+6=15$

Next , Compute g(i,S) with S containing 2 elements

$g(2,\{3,4]) = \min(c_{23}+g(3,\{4\}), c_{24}+g(4,\{3\}))$

$= \min(9+20, 10+15) = \min(29,25) = 25$

$g(3,\{2,4]) = \min(c_{32}+g(2,\{4\}), c_{34}+g(4,\{2\}))$

$= \min(13+18, 12+13) = \min(31,25) = 25$

$g(4,\{2,3]) = \min(c_{42}+g(2,\{3\}), c_{43}+g(3,\{2\}))$

$= \min(8+15, 9+18) = \min(23,27) = 23$

Finally

$g(1, \{2,3,4\})$

$=\min(c_{12}+g(2,\{3,4\}),c_{13}+g(3,\{2,4\}), c_{14}+g(4,\{2,3\}))$

$= \min (10+25, 15+25 ,20+23)=\min(35,40,43\}=35$

4. The tour can be constructed if we also store with each $g(i,S)$, the value j that minimizes the right hand side say $J(i,S)$

$J(1,\{2,3,4\})=2$ The tour starts from 1 and goes to 2

$J(2,\{3,4\})=4$ from 2 go to 4 , from 4 to 3 and back to 1

Let N be the number of g(i,s)'s to be computed and stored

For each value of S there are n-1 choices of i. The number of sets S of size k not including 1 and I is ${}^{n-2}C_k$

$$N = \sum_{k=0}^{n-2} n-1 \left({}^{n-2}C_k\right) = n-1 \sum_{k=0}^{n-2} \left({}^{n-2}C_k\right) = (n-1)\, 2^{\,n-2}$$

The space complexity of the algorithm is $O(n2^n)$ as all the computed values need to be stored

Since for finding minimum, comparisons are required time complexity is $O(n^2 2^n)$

It is better than enumerating all n! permutations to chose the best one but space complexity is very high

String Editing

- Given two strings X=x1,x2,…,xn Y=y1,y2,..,ym where xi's and yj's are elements of finite set of symbols called Alphabet.
- We want to transform X into Y using a sequence of edit operations on X.
- These edit operations are insert, delete and change (symbol of X into another).
- There is cost associated with each operation.
- The cost of sequence of operations is sum of the costs of individual operations in the sequence.

- The problem of string editing is to find a minimum cost sequence of edit operations that will transform X into Y.
- Let $D(x_i)$ = the cost of deleting x_i from X
- $I(y_j)$ = the cost of inserting yj into X
- $C(x_i,y_j)$=the cost of changing xi of X into yj.
- Define cost(i,j) = minimum cost of edit sequence for transforming x1,x2,…, x_i to y1,y2,…yj where ,0≤ i≤ n and 0≤ j≤ m.
- Cost(i,j) = 0 if i=j=0
- To find cost(m,n) = minimum cost of transforming X into Y

- If $j=0$ $i>0$ transform X to Y by sequence of delete operations

  $$cost(i,0) = cost\ (i-1,0)+D(xi)$$

- If $i=0$ $j>0$ transform X to Y by sequence of insert operations

  $$cost(0,j) = cost\ (0,j-1)+I(yj)$$

- If $i \neq 0$ $j \neq 0$ one of the three ways can be used

i. Transform x1,x2,…,xi-1 into y1,y2,…,yj using a minimum cost edit sequence and then delete xi. $cost(i,j) = cost(i-1,j)+D(xi)$

ii. Transform $x1,x2,\dots,xi-1$ into $y1,y2,\dots,yj-1$ using a minimum cost edit sequence and then change xi to yj. $cost(i,j) = cost(i-1,j-1)+C(xi,yj)$

iii. Transform $x1,x2,\dots,xi$ into $y1,y2,\dots,yj-1$ using a minimum cost edit sequence and then insert yj. $cost(i,j) = cost(i,j-1)+I(yj)$