4. Dynamic Programming

Dynamic programming is a method of solving problems exhibiting the properties of overlapping sub problems and optimal substructure that takes much less time than naive methods. The term was originally used in the 1940s by Richard Bellman to describe the process of solving problems where one needs to find the best decisions one after another.

It is used when the solution to a problem can be viewed as the result of a sequence of decisions. It avoid duplicate calculation in many cases by keeping a table of known results and fills up as sub instances are solved.

It follows a bottom-up technique by which it start with smaller and hence simplest sub instances. Combine their solutions to get answer to sub instances of bigger size until we arrive at solution for original instance.

Dynamic programming differs from Greedy method because Greedy method makes only one decision sequence but dynamic programming makes more than one decision.

Principle of Optimality
An optimal sequence of decisions has the property that whatever the initial state and decisions are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

4.1 Multistage Graph

A multistage graph is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets v_i, $1 \leq i \leq k$. If <u, v> is an edge in E, then $u \in v_i$ and $v \in v_{i+1}$ for some i, $1 \leq i \leq k$. Sets v_i and v_k is such that $| v_i | = | v_k | = 1$. Let s and t are vertices if v_i and v_k respectively such that s is the source and t is the sink. c(i,j) is the cost of edge <i,j>. Cost of the path from s to t is the sum of cost of all edges on the path. Multi stage graph problem is to find a minimum cost path from s to t. Every path from s to t starts at stage 1, goes to stage 2, etc. eventually terminates in stage k.

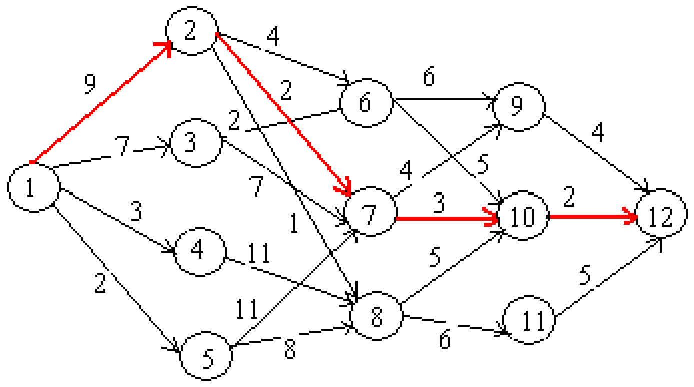
- Every path from s to t is the result of a sequence of k-1 decisions.
- i^{th} decision involves determining which vertex in v_{i+1} $1 \leq i \leq k-2$ is to be on the path.

Let p(i,j) is the minimum cost path from j in v_i to vertex t, and cost (i,j) be its cost.
According to forward approach

$$\text{Cost}(i, j) = \min_{\substack{l \in v_{i+1} \\ <j,l> \in v_i}} \left\{ c(j, l) + \text{cost}(i+1, l) \right\}$$

Since cost (k-1, j) = c(j. t) if $<j,t> \in E$ and
cost (k-1, j) = ∞ if $<j,t> \notin E$
we can solve the above equation. First calculate cost(k-2, j) for all $j \in v_{k-2}$, then calculate cost (k-3, j) for all $j \in v_{k-3}$.etc, finally calculate cost(1,s).

Example 4.1
Consider the directed graph given below.



cost (3, 6) = min{6 + cost(4,9), 5 + cost(4,10)} = 7
cost (3, 7) = min{4 + cost(4,9), 3 + cost(4,10)} = 5
cost (3, 8) = min{5 + cost(4,10), 6 + cost(4,11)} = 7

cost (2, 2) = min{4 + cost(3,6) , 2 + cost(3,7), 1 + cost(3,8)} = 7
cost (2, 3) = min{1 + cost(3,6), 7 + cost(3,7)} = 9
cost (2, 4) = min{11 + cost(3,8)} = 18
cost (2, 5) = min{11 + cost(3, 7), 8 + cost(3,8)} =15

cost (1, 1) = min{9 + cost(2, 2), 7 + cost(2, 3) , 3 + cost(2, 4), 2 + cost(2, 5)} = **16**

Let d(I, j) denotes value of l that minimizes c(j, l) + cost(i+1, l).
Hence in the above example

d(3,6) = 10	d(2,2) = 7	d(1,1) = 2
d(3, 7) =10	d(2,3) = 6	
d(3,8) = 10	d(2,4) = 8	
	d(2,5) = 8	

Algorithm 4.1
In the algorithm vertices are assigned index in the order of stages. Hence vertices in stage1 have indices less than in stage2 and so on. Hence first subscript in cost, p and d which is denoting the stage number is omitted.

```
Void FGraph( graph G, int k, int n, int p[])
// n vertices are indexed in the oreder of stages. E is set of edges and c[i][j] is cost of <i,j>
// and p[i:k] is the minimum cost path.
{
    float cost[10];
    int d[10], r, j;
    for(j = n-1; j>=n; j--)  // compute cost[j]
    {
            Let r be a vetex such that <j,r> is an edge in G and c[j][r] + cost[r] is minimum;
            cost[j] = c[j][r] + cost[r];
            d[j] = r;
    }
    //Find a minimum-cost path
    p[1] = 1;        p[k] = n;
    for(j=2; j<=k; j++)
            p[j] = d[p[j-1]];
}
```

Time Complexity
If G is represented by adjacency lists, then time taken to find r is equal to degree of the vertex j. If G has |E| edges, the number of times for loop executes is (|V| + |E|). The complexity of last for loop is Θ(k). Hence the time complexity of the algorithm is Θ(|V| + |E|).

4.2 All pairs Shortest Path

Let G=(V,E) be a directed graph with n vertices.The cost of the vertices are represented by an adjacency matrix such that $1 \leq i \leq n$ and $1 \leq j \leq n$

cost(i, i) = 0,
cost(i, j) is the length of the edge <i, j> if $<i, j> \in E$ and
cost(i, j) = ∞ if $<i, j> \notin E$.

The graph allows edges with negative cost value, but negative valued cycles are not allowed. All pairs shortest path problem is to find a matrix A such that A[i][j] is the length of the shortest path from i to j.

Consider a shortest path from i to j, $i \neq j$. The path originates at i goes through possibly many vertices and terminates at j. Assume that there is no cycles. If there is a cycle we can remove them without increase in the cost. Because there is no cycle with negative cost.
Initially we set A[i][j] = c[i][j].
Algorithm makes n passes over A. Let A_0, A_1, .. A_n represent the matrix on each pass.
Let $A^{k-1}[i,j]$ represent the smallest path from i to j passing through no intermediate vertex greater than k-1. This will be the result after k-1 iterations. Hence kth iteration explores whether k lies on optimal path. A shortest path from I to j passes through no vertex greater than k, either it goes through k or does not.
If it does , $A^k[i,j] = A^{k-1}[i,k] + A^{k-1}[k,j]$
If it does not, then no intermediate vertex has index greater than k-1. Then $A^k[i,j] = A^{k-1}[i,j]$

Combining these two conditions we get
$A^k[i,j] = \min\{ A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j]\}$, $k>=1$ $(A^0[i,j]= cost(i, j))$

Algorithm 4.2

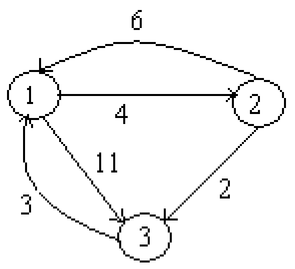
```
Void AllPaths ( float cost[][], float A[][], int n)
{
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            A[i][j] = cost[i][j];

    for (int k=1; k<=n; k++)
        for (i=1; i<=n; i++)
            for (j=1; j<=n; j++)
                    A[i][j] = min{A[i][j], A[i][k] + A[k][j]);
}
```

Time complexity of the algorithm is $O(n^3)$.

Example 4.2

Consider the directed graph given below.



Cost adjacency matrix for the graph is as given below

$$\begin{bmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{bmatrix}$$

Copy the cost values to the matrix A. So we have A_0 as

$$\begin{array}{c|ccc} A_0 & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & \infty & 0 \end{array}$$

Matrix A after each iteration is as given below.

$$\begin{array}{c|ccc} A_1 & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 11 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array} \qquad \begin{array}{c|ccc} A_2 & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 6 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array} \qquad \begin{array}{c|ccc} A_3 & 1 & 2 & 3 \\ \hline 1 & 0 & 4 & 6 \\ 2 & 5 & 0 & 2 \\ 3 & 3 & 7 & 0 \end{array}$$

4.3 The Traveling Salesman problem

Problem: A salesman spends his time in visiting cities cyclically. In his tour he visits each city just once, and finishes up where he started. In which order should he visit to minimize the distance traveled.

The problem an be represented using a graph. Let G=(V,E) be a directed graph with cost c_{ij}, $c_{ij} > 0$ for all $<i\ j> \in E$ and $c_{ij} = \infty$ for all $<j,t> \notin E$. $|V| = n$ and $n>1$. We know that tour of a graph includes all vertices in V and cost of tour is the sum of the cost of all edges on the tour. Hence the traveling salesman problem is to minimize the cost.

Application

The traveling salesman problem can be correlated to many problems that we find in the day to day life. For example, consider a production environment with many commodities manufactured by same set of machines. Manufacturing occur in cycles. In each production cycle n different commodities are produced. When machine changes from product *i* to product *j*, a cost C_{ij} is incurred. Since products are manufactured cyclically, for the change from last commodity to the first a cost is incurred. The problem is to find the optimal sequence to manufacture the products so that the production cost is minimum.

We know that the tour of the simple graph starts and ends at vertex1. Every tour consist of an edge <i, k> for some $k \in V-\{1\}$ and a path from k to 1. Path from k to 1 goes through each vertex in V- {1,k} exactly once. If tour is optimal, path from k to 1 muat be shortest k to 1 path going through all vertices in V-{1,k}. Hence the principle of optimality holds.
Let g(i, S) be length of shortest path starting at i, going through all vertices in S ending at 1. Function g(1, V-{1}) is the optimal salesman tour. From the principle of optimality

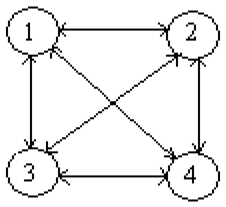
$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{C_{1k} + g(k, V-\{1,k\})\} \quad \text{—— (1)}$$

$$\text{for } i \notin S \quad g(1, S) = \min_{j \in S} \{C_{ij} + g(j, S-\{j\})\} \quad \text{———— (2)}$$

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

Use eq.(1) to get g(i, S) for |S| =1. Then find g(i, S) with |S|=2 and so on.

Example 4.3
Consider the directed graph with the cost adjacency matrix given below.



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

$g(2, \varphi) = C_{21} = 5$
$g(3, \varphi) = C_{31} = 6$
$g(4, \varphi) = C_{41} = 8$

$g(2, \{3\}) = C_{23} + g(3, \varphi) = 15$
$g(2, \{4\}) = C_{24} + g(4, \varphi) = 18$
$g(3, \{2\}) = C_{32} + g(2, \varphi) = 18$
$g(3, \{4\}) = C_{34} + g(4, \varphi) = 20$
$g(4, \{2\}) = C_{42} + g(2, \varphi) = 13$
$g(4, \{3\}) = C_{43} + g(3, \varphi) = 15$

$g(2, \{3,4\}) = \min\{C_{23} + g(3, \{4\}),\ C_{24} + g(4, \{3\})\} = 25$
$g(3, \{2,4\}) = \min\{C_{32} + g(2, \{4\}),\ C_{34} + g(4, \{2\})\} = 25$
$g(4, \{2,3\}) = \min\{C_{42} + g(2, \{3\}),\ C_{43} + g(3, \{2\})\} = 23$

$g(1, \{2,3,4\}) = \min\{C_{12} + g(2, \{3,4\}),\ C_{13} + g(3, \{2,4\}),\ C_{14} + g(4, \{2,3\})\}$
$\qquad = \min\{35, 40, 43\} = 35$

Let J(i, S) represent the value of j which is the value of g(i, S) that minimizes the right hand side of equ(2). Then J(1, {2,3,4}) =2, which infers that the tour starts from 1 goes to

2. Since J(2,{3,4}) = 4, we know that from 2 it goes to 4. J(4, {3}) = 3. Hence the next vertex is 3. Then we have the optimal sequence as **1,2,4,3,1**.