Dynamic Programming

A dynamic-programming algorithm solves each subsubproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time it solves each subsubproblem.

We typically apply dynamic programming to ***optimization problems***. Such problems can have many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. We call such a solution *an* optimal solution to the problem, as opposed to *the* optimal solution,

since there may be several solutions that achieve the optimal value.

When developing a dynamic-programming algorithm, we follow a sequence of four steps:

1. Characterize the structure of an optimal solution.

2. Recursively define the value of an optimal solution.

3. Compute the value of an optimal solution, typically in a bottom-up fashion.

4. Construct an optimal solution from computed information.

Steps 1–3 form the basis of a dynamic-programming solution to a problem. If we

need only the value of an optimal solution, and not the solution itself, then we

can omit step 4. When we do perform step 4, we sometimes maintain additional

information during step 3 so that we can easily construct an optimal solution.

In dynamic programming an optimal sequence of decisions is obtained by making explicit appeal to the principle of optimality.

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

Comparison

For divide and conquer (top down), the subproblems are independent so we can solve them in any order.

For greedy algorithms (bottom up), we can always choose the "right" subproblem by a greedy choice.

In dynamic programming, we solve many subproblems and store the results: not all of them will contribute to solving the larger problem. Because of optimal substructure, we can be sure that at least some of the subproblems will be useful.

All Pairs Shortest Path(Floyd-Warshall Algorithm)

Let G = (V, E) be a directed graph with n vertices. Let cost be a cost adjacency matrix for G such that cost(i, i)= 0,1≤ i ≤ n. Then cost(i, j) is the length (or cost) of edge{i, j} if (i, j)Є E(G) and cost(i, j)= ∞ if i ≠ j and (i, j) ∉ E(G).The all-pairs shortest-path problem is to determine a matrix A such that A(i, j)is the length of a shortest path from i to j. Require cost(i, j)≥ 0, for every edge(i, j),we only require that G have no cycles with negative length. Note that if we allow G to contain a cycle of negative length, then the shortest path between any two vertices on this cycle has length -∞.

Let us examine a shortest i to j path in G, i ≠ j. This path originates at vertex i and goes through some intermediate vertices(possibly none)and terminates at vertex j. If k is an intermediate vertex on this shortest path, then the sub paths from i to k and from k to j must be shortest paths from i to k and k to j, respectively

Once this decision has been made, we need to find two shortest paths, one from i to k and the other from k to j. Neither of these may go through a vertex with index greater than k – 1. Using $A^k(i, j)$ to represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain

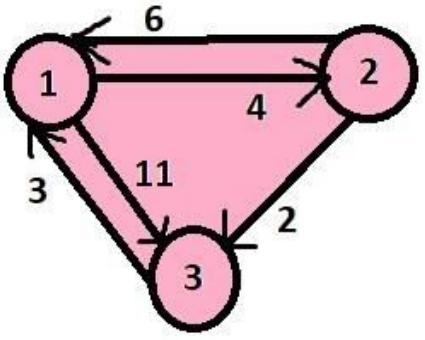
$$\mathbf{A(i, j)= \min \{\min_{1\le k \le n}\{A^{k-1}(i, k) + A^{k-1}(k, j)\},cost(i, j)\}}$$

Considering the above formula we also get

$\mathbf{A^k(i, j)= \min \{A^{k-1}(i, j), \{A^{k-1}(i, k) + A^{k-1}(k, j)\}\}}$ where 1 ≤ k ≤ n.

Example

Consider the following graph to find out all pairs shortest path using Floyd- Warshall algorithm.



Solution:

Adjacency matrix of the above graph represent by A^0 of the following where k=0. It is also called cost matrix.

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

A(1, 1) = A(2, 2) = A(3, 3) = 0

Consider k=1

By applying Floyd-Warshall algorithm we will get the following using

$A^k(i, j)= \min \{A^{k-1}(i, j), \{A^{k-1}(i, k) + A^{k-1}(k, j)\}\}$

where i is the source vertex, j is the destination vertex and k is the intermediate vertex. So the path will be i → k → j.

$A^1(1, 2) = \min\{A^0(1, 2), \{A^0(1, 1) + A^0(1, 2)\}\}$

$= \min\{4, \{0 + 4\}\}$

$= \min\{4, 4\}$

$= 4$

Similarly you solve $A^1(1, 3)$, $A^1(2, 1)$, $A^1(2, 3)$, $A^1(3, 1)$ now

$A^1(3, 2) = \min\{A^0(3, 2), \{A^0(3, 1) + A^0(1, 2)\}\}$

$= \min\{\infty, \{3 + 4\}\}$

$= \min\{\infty, 7\}$

$= 7$

therefore

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	**7**	0

Now consider k=2

you solve $A^2(1, 2)$ now

$A^2(1, 3) = \min\{A^1(1, 3), \{A^1(1, 2) + A^1(2, 3)\}\}$

$= \min\{11, \{4 + 2\}\}$

$= \min\{11, 6\}$

$= 6$

Similarly you solve $A^2(2, 1)$, $A^2(2, 3)$, $A^2(3, 1)$, $A^2(3, 2)$ we will get the following

A^2	1	2	3
1	0	4	**6**
2	6	0	2
3	3	7	0

Now consider k=3

you solve $A^3(1, 2)$, $A^3(1, 2)$ now

$A^3(2, 1) = \min\{A^2(2, 1), \{A^2(2, 3) + A^2(3, 1)\}\}$

$= \min\{6, \{2 + 3\}\}$

$= \min\{6, 5\}$

$= 5$

Similarly you solve $A^3(2, 3)$, $A^3(3, 1)$, $A^3(3, 2)$ we will get the following

A^3	1	2	3
1	0	4	**6**
2	**5**	0	2
3	3	7	0

The above table represent the optimal solution

Algorithm

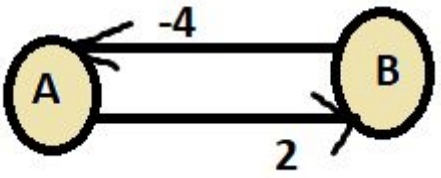
```
AlgorithmAllPaths(cost, A, n)
 // cost[1:n, 1:n] is the cost adjacency matrix of a graph with
// n vertices; A[i, j] is the cost of a shortest path from vertex
// i to vertex j. cost[i, i] = 0.0 for 1 ≤ i ≤ n.
```

```
{
        for i =1 to n do
        for j =1 to n do
                A[i, j] = cost[i, j];              //Copy cost into A.
        for k =1 to n do
                for i =1 to n do
                        for j =1 to n do
                                A[i, j] =min(A[i, j],A[i, k]+A[k, j]);
}
```

Time Complexity

In the above algorithm there are three nested loops therefore total time complexity of Floyd-Warshall algorithm is $O(n^3)$.

Note: Floyd-Warshall algorithm will not work if the graph has negative edge cycle(i.e. a cycle whose edges sum to a negative value) like following.



This is necessary to ensure that shortest paths consist of a finite number of edges. In the above graph the length of the shortest path from vertex A to vertex B is -∞. The length of the path A, B, A, B, A, B,.....