Dynamic Programming

Chapter 8

Dynamic Programming

- It was invented by a US mathematician Richard Bellman in the 1950’s as a general method for optimizing multistage decision processes.
- The word programming refers to planning.
- It is a technique for solving problems that have **overlapping subproblems**.
- Ex: Generating Fibonacci numbers
- These subproblems arise from a recurrence relating a solution to a given problem with solutions to its smaller subproblems of same type. Rather than solving overlapping subproblems again and again, **dynamic programming suggests solving each of the smaller problem only once and recording the results in a table from which we can then obtain a solution to the original problem.**

Computing Binomial Coefficient

- Example for applying dynamic programming for non optimization problem.
- It is denoted by C(n,k) or is the number of combinations of k elements from an n-element set ($0 \le k \le n$).
- The name binomial coefficient comes from the participation of these numbers in the binomial formula
- $C(n,k) = c(n-1, k-1) + c(n-1, k)$ for $n > k > 0$

  and $c(n,0) = c(n,n) = 1$

Record the binomial coefficients in a table of n+1 rows and k+1 columns (sum of the elements in previous row, previous column and previous row, same column)

Computing Binomial Coefficient

```
Algorithm Binomial (n, k)
//Computes C(n,k) by the dynamic programming algorithm.
//Input: A pair of non- negative integers n >= k >= 0
//Output: The value of C(n, k)
for (i ← 0 to n)
 for (j ← 0 to min(i, k))
    if (j == 0 || j == i)
       C[i][j] ← 1;
    else
       C[i][j] ← C[i - 1][j - 1] + C[i - 1][j];
return C[n][k]
```

Warshall's Algorithm

- Computes the **transitive closure of a directed graph**.
- The transitive closure of a directed graph with n vertices can be defined as the **n-by-n boolean matrix T= {tij}** where the element in the ith row and the jth column is **1 if there exists a nontrivial directed path** from the ith vertex to the jth vertex.
- Transitive closure can also be generated with DFS and BFS. Performing the traversal starting at the ith vertex
- This method traverses the same digraph several times and the efficiency decreases. **Warshall's algo improves the efficiency in finding transitive closure.**

Warshall's Algorithm

- It constructs the transitive closure of a digraph with n vertices thru **a series of n-by-n boolean matrices**
- $R^{(0)},\ldots,R^{(k-1)},R^{(k)},\ldots R^{(n)}$
- To generate the elements of $R^{(k)}$ from $R^{(k-1)}$ use the following method.
- **If an element rij is 1 in $R^{(k-1)}$ it remains 1 in $R^{(k)}$.**
- **If an element rij is 0 in $R^{(k-1)}$ it has to be changed to 1 in $R^{(k)}$ if and only if the element in its row i and column k and the element in its column j and row k are both 1's in $R^{(k-1)}$.**

Warshall’s Algorithm

Algorithm Warshall(A[1…n,1…n])

//Implements Warshall’s algorithm for computing the transitive closure.

//Input : The adjacency matrix A of digraph with n vertices.

//Output : The transitive closure of the digraph.

```
R^(0) ← A
    for k ← 1 to n do
        for i ← 1 to n do
            for j ← 1 to n do
                R^(k)[i][j] ← R^(k-1)[i][j] or R^(k-1)[i][k] and R^(k-1)[k][j]
return R^(n)
```

Floyd's Algorithm

- **The All-pairs shortest-paths problem asks to find the distances from each vertex to all other vertices.**
- We use a *distance matrix D (A n-by-n matrix that holds the lengths of shortest paths)*
- The element dij in the ith row and jth column of the matrix indicates the length of the shortest path from the ith vertex to the jth vertex.
- Its inventor is R Floyd.
- It is applicable to both undirected and directed weighted graphs.

Floyd’s Algorithm

- It computes the distance matrix thru a series of n-by-n matrices D(0),…,D(k-1),D(k),…D(n)
- Each of these matrices contains the lengths of the shortest paths with constraints on the path.
- The element in the ith row and the jth column of matrix D(k) (k=0,…,n) is equal to the length of the shortest path among all paths from the ith vertex to the jth vertex with each intermediate vertex numbered not higher than k.
- D(0) is the weight matrix of the graph.
- D(n) contains the lengths of the shortest paths among all paths that can use all n vertices as intermediate.

Floyd’s Algorithm

Algorithm Floyd (W[1…n,1…n])

//Implements Floyd’s Algorithm for all-pairs shortest-paths problem.

//Input : The weight matrix W of a graph.

//Output: The distance matrix of the shortest path’s lengths.

```
D← W
For k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i][j] ← min( D[i][j] , D[i][k] + D[k][j] )
return D
```

Memory Function

Algorithm MFKnapsack (i,j)

//Implements the Memory Function method for knapsack problem.

//Input: Non- negative integers I an j indicating the no. of first items being considered and
//the knapsack capacity.

//Output : Value of the optimal subset of first i items.

//Note : Weights array and values array are passed as global variables.

//Table entities V[1…n,1…w] are initialized with -1.

```
if v[i,j]<0
    if j<weights[i]
            value<-MFKnapsack (i-1,j)
    else
            value<-max{  MFKnapsack(i-1,j) , Values[i]+MFKnapsack(i-1,j-weight(i) }
            V[i,j] = value
return V[i,j]
```

The End

Thank You