Dynamic Programming

Dynamic Programming Algorithms

Longest Common Subsequence

Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

- Recurrence Relation:

$$\text{LCS}(i, j) = \begin{cases} 0; & \text{if } i = 0 \text{ (or) } j = 0 \\ 1 + LCS(i-1, j-1); & \text{if } x[i-1] = y[j-1] \\ \max\,[LCS(i-1, j),\ LCS(i, j-1)]; & \text{if } x[i] \neq y[j] \end{cases}$$

Time complexity: by **Brute force** = $O(2^m)$, by **Dynamic programming** = $O(m \times n)$.

Space complexity = $O(mn)$; where m and n are length of two given sequences.

0/1 Knapsack Problem

Given weights and values of n items, select items to put items in a Knapsack of capacity W to maximise the total value in the Knapsack. You cannot break an item, either pick the complete item, or don't pick it (0 to 1 property).

- $$0/1\ \text{KS}(M, N) = \begin{cases} 0; & \text{if } M = 0 \text{ or } N = 0 \\ 0/1\ \text{KS}(M, N-1); & \text{if } w[n] > M \\ \max \begin{Bmatrix} 0/1\ \text{KS}(M - W[n], N-1) + P[n] \\ 0/1\ \text{KS}(M, N-1) \end{Bmatrix}; & \text{otherwise} \end{cases}$$

Time complexity = $O(MN)$: If M value is very large then it behaves like an exponential and **NP-complete problem**.

Travelling Salesperson Problem

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. In other words find the minimum cost Hamiltonian cycle.

$$TSP(A, R) = \begin{cases} C(A,S) \text{ if } R = \phi \\ \min\{(C[A,K] + TSP(K, R-K)) \forall K \in R\} \end{cases}$$

Time:

Time complexity: without dynamic programming = $O(n^n)$, **with** dynamic programming = $O(2^n . n^2)$

Space complexity: without dynamic programming = $O(n^2)$, **with** dynamic programming = $O(n^n)$.

It is one of the NP Hard problem.

atrix Chain Multiplication

Given a sequence of matrices, find the most efficient order to multiply ese matrices together in order to minimise the number of multiplications.

$$MCM = \begin{cases} 0 & \text{if } i = j \\ \min\begin{cases} mcm(i, K) + mcm(K+1, j) + Pi \times Pj \times Pk \\ \text{where } i \le K < j \text{ or} \\ i \le K \le j-1 \end{cases} \end{cases}$$

Number of ways to multiply matrix: $T(n) = \sum_{i=1}^{n-1} T(i).T(n-i)$

Time complexity: without dynamic programing = $O(n^n)$, **with** dynamic programing = $O(n^3)$.

Space complexity: without dynamic programing = $O(n)$, **with** dynamic programing = $O(n^2)$.

Sum of Subset Problem

Find if there is a subset of the given set, sum of whose elements is equal to given sum.

- **Recurrence Relation:**

$$SoS(M, N, S) = \begin{cases} \text{return}(S); & M = 0 \\ return(-1); & N = 0 \\ SoS(M, N-1, S); & \text{if } w[N] > M \\ \text{Min}\begin{Bmatrix} SoS(M - w[N], N-1, S \cup w[N]) \\ SoS(M, N-1, S) \end{Bmatrix}; & \text{otherwise} \end{cases}$$

Time complexity by Brute force = $O(MN)$

Floyd-Warshall's: All Pair Shortest Path

For finding shortest paths between all pairs of vertices in a weighted graph with positive or negative edge weights (but with no negative cycles).

- $A^k(i, j)$ = the min cost required to go from i to j by considering all intermediate vertices are numbered not greater than k.

$$A^0(i, j) = C(i, j)$$

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

Time complexity = $O(n^3)$ with heap = $O(n^2 \log n)$.

- Warshall's algorithm will not work for negative edge cycle.

Optimal Binary Search Tree

Given a sorted array keys[0.. $n - 1$] of search keys and an array frequency[0.. $n-1$] of frequency counts, where frequency[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

- **It is lexically ordered tree:**

$$\text{cost}(i, j) = \min_{i<k\leq j} \{\text{cost}(i, k-1) + \text{cost}(k, j) + w(i, j)\}$$

$$w(i, j) = p(j) + q(i) + w(i, j-1)$$
$$w(i, i) = q(i) \text{ and cost}(i, i) = 0$$

$$\text{cost} = \underbrace{\sum_{i}^{n} p_i \times \text{level } a_i}_{\text{(successful)}} + \underbrace{\sum_{i}^{n+1} q_i\,(\text{level } E_i - 1)}_{\text{(unsuccessful)}}$$

Time complexity is $O(n^3)$.

Multistage Graph

A Multistage graph is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only.

Task is to find shortest path from source to destination using the special property of multistage graph.

Shortcut: Travels back from destination to source with every time min cost selection.

$$\underset{\text{stage, vertex}}{MSG\ (\overset{}{S_i}\downarrow,\ V_j\downarrow)} = \begin{cases} 0 \text{ if } s_i = F \,\&\&\, V_j = 0 \\ \min\begin{cases}((V_a, K) + MSG(s_i + s, K)) \\ \forall K \in s_i + 1 \,\&\&\, (V_j, K) \in E\end{cases}\end{cases}$$

Time complexity: without dynamic programing = O(2^n), **with** dynamic programing = O($V + E$).

Space complexity: without dynamic programing = O(v^2), **with** dynamic programing = O(v^2).

■■■