week5_lec2

Main Ideas

- Dynamic Programming
    - Shortest Paths in Directed Acyclic Graphs (DAGs)
    - Longest Path in Directed Acyclic Graphs (DAGs)
- Longest Increasing Subsequence

Dynamic Programming

It is an algorithmic technique for solving a problem by recursively breaking it down into simpler subproblems and using the fact that the optimal solution to the overall problem depends upon the optimal solution to its individual subproblems.

Can be applied in following scenarios :

- Optimal Substructure - able to break the problem into smaller problems of similar type
- Overlapping Subproblems - The result of subproblem once solved, should be needed again.

Shortest Path in Directed Acyclic Graphs

Problem : Given a weighted directed acyclic graph and a source vertex in the graph, find the shortest paths from given source to all other vertices.

Solution

1) Initialize dist[] = {INF, INF, ….} and dist[s] = 0 where s is the source vertex.

2) Create a topological order of all vertices.

3) Do following for every vertex u in topological order.

```
if (dist[v] > dist[u] + weight(u, v))
    dist[v] = dist[u] + weight(u, v)
```

Time complexity of topological sorting is O(V+E).

After finding topological order, the algorithm process all vertices and for every vertex, it runs a loop for all adjacent vertices.

Total adjacent vertices in a graph is O(E). So the inner loop runs O(V+E) times.

Therefore, overall time complexity of this algorithm is O(V+E).

Longest Path in Directed Acyclic Graphs

Finding the longest path is not possible. The possibility of a cycle means that the solution to longest would be not useful.

Longest Increasing Subsequence

Problem : The input is a sequence of of numbers $a_1, a_2, ..., a_n$. A subsequence is $a_{i_1}, a_{i_2}, ..., a_{i_k}$ such that $1 \leq i_1 \leq i_2 \leq ... \leq i_k$. An increasing subsequence is one where the a values are increasing. Find such a sequence.

Solution : Create a DAG of all permissible transitions. If a node i exists for each a_i then an edge (i, j)exists if $i < j$ and $a_i < a_j$ i.e. if a_i can come before a_j in an increasing subsequence.

$L(j)$ is the length of the longest path ending at a node j. In other word, it is the LIS ending at j.

Then, L(i) can be recursively written as:

```
L(i) = 1 + max( L(j) ) where 0 < j < i and arr[j] < arr[i]; or
L(i) = 1, if no such j exists.
```

Time Complexity : Exponential