week7_lec1

Main Ideas

- Shortest reliable path
- Floyd Warshall Algorithm
- Independent Set in Trees

Shortest Reliable Path

Problem : Given a directed and weighted graph and two vertices ‘u’ and ‘v’ in it, find shortest path from ‘u’ to ‘v’ with exactly k edges on the path.

The graph is given as adjacency matrix representation where value of graph[i][j] indicates the weight of an edge from vertex i to vertex j and a value infinite indicates no edge from i to j.

For example, consider the following graph. Let source ‘u’ be vertex 0, destination ‘v’ be 3 and k be 2. There are two walks of length 2, the walks are {0, 2, 3} and {0, 1, 3}. The shortest among the two is {0, 2, 3} and weight of path is 3+6 = 9.

Solution :

Since it satisfies the conditions for dynamic programming, we can optimize the solution using dynamic Programming. The idea is to build a 3D table where first dimension is source, second dimension is destination, third dimension is number of edges from source to destination, and the value is count of walks. Like other Dynamic Programming problems, we fill the 3D table in bottom-up manner.

```
int shortestPath(int graph[][V], int u, int v, int k)
{

    int sp[V][V][k+1];

    for (int e = 0; e <= k; e++)
    {
        for (int i = 0; i < V; i++)  // for source
        {
            for (int j = 0; j < V; j++) // for destination
            {
```

```
                sp[i][j][e] = INF;


                if (e == 0 && i == j)
                    sp[i][j][e] = 0;
                if (e == 1 && graph[i][j] != INF)
                    sp[i][j][e] = graph[i][j];

                if (e > 1)
                {
                    for (int a = 0; a < V; a++)
                    {
                        if (graph[i][a] != INF && i != a &&
                            j!= a && sp[a][j][e-1] != INF)
                          sp[i][j][e] = min(sp[i][j][e], graph[i][a] +
                                                    sp[a][j][e-1]);
                    }
                }
            }
        }
    }
    return sp[u][v][k];
}
```

Floyd Warshall Algorithm

Problem : The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph.

Solution : Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm works for both the directed and undirected weighted graphs. But, it does not work for the graphs with negative cycles.

We initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices. For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.

1)k is not an intermediate vertex in shortest path from i to j. We keep the value of matrix[i][j] as it is.

2)k is an intermediate vertex in shortest path from i to j. We update the value of matrix[i][j] as matrix[i][k] + matrix[k][j] if matrix[i][j] > matrix[i][k] + matrix[k][j]

```
void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;

  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];

  // Adding vertices individually
  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
}
```

Independent Set in Trees

Problem : Given a Binary Tree, find size of the Largest Independent Set(LIS) in it. A subset of all tree nodes is an independent set if there is no edge between any two nodes of the subset.

Solution : A Dynamic Programming solution solves a given problem using solutions of subproblems in bottom up manner.

1) Optimal Substructure:

Let LISS(X) indicates size of largest independent set of a tree with root X.

```
     LISS(X) = MAX { (1 + sum of LISS for all grandchildren of X),
                     (sum of LISS for all children of X) }
```

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

```
int max(int x, int y)
{
    return (x > y) ? x : y;
}

int LISS(node *root)
{
    if (root == NULL)
    return 0;

    // Calculate size excluding the current node
    int size_excl = LISS(root->left) +
                    LISS(root->right);

    // Calculate size including the current node
    int size_incl = 1;
    if (root->left)
        size_incl += LISS(root->left->left) +
                     LISS(root->left->right);
    if (root->right)
        size_incl += LISS(root->right->left) +
                     LISS(root->right->right);

    return max(size_incl, size_excl);
}
```