**Assignment - 7**

**Answer to the Question No. – 1**

1. The given example is,

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Si | 1 | 3 | 0 | 5 | 3 | 5 | 6 | 8 | 8 | 2 | 12 |
| Fi | 4 | 5 | 6 | 7 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |
| Active time | 3 | 2 | 6 | 2 | 6 | 4 | 4 | 3 | 4 | 12 | 4 |

Now, if we choose the activity that will be active the least amount of time, then the solution for this greedy algorithm will be {A2, A8, A11}. But this is not the optimal solution. If we choose the optimal solution, then the result will be {A1, A4, A8, A11}.

1. If we sort the activities in decreasing order of start time then it will give another optimal solution.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Si | 12 | 2 | 8 | 8 | 6 | 5 | 3 | 5 | 0 | 3 | 1 |
| Fi | 16 | 14 | 12 | 11 | 10 | 9 | 9 | 7 | 6 | 5 | 4 |

The optimal solution will be {A1, A3, A8, A11}.

OptimalGreedySolution (S, f){

n = length(S);

A = {ai};

k = 1;

for j = 2 to n {

if f(j) <= S(k) { A = A U {aj}; k = j; }

}

return A;

}

**Theorem:** Consider any non-empty sub-problem Sk and let aj denote the activity in Sk with the last activity to start. Then aj is included in some maximum size subset of compatible activities of Sk.

To prove this algorithm is correct we need to prove the above theorem.

**Proof:** let Ak be an optimal solution for Sk and let ap be the activity in Ak with last start time. If ap = aj we are done.

So, assume ap != aj. Now, consider the set Ak’ = Ak \ {ap} U {aj}

Ak \ {ap}

ap

aj

Here all the activity in Ak has finish time earlier than the start time of ap and the start time of aj is later than the start time of ap. so, Ak’ is feasible and Ak’ = Ak.

So, Ak’ is optimal and the algorithm is correct.

**Answer to the Question No. – 2**

Initially the number of in-degrees of each node: a = 0, b = 2, c =1, d = 1, e =1, f = 2.

So the valid topological sorts are –

1. a -> b -> c -> d -> e -> f
2. a -> b -> d -> c -> e -> f
3. a -> b -> d -> e -> c -> f
4. a -> d -> e -> b -> c -> f
5. a -> d -> b -> c -> e -> f
6. a -> d -> b -> e -> c -> f

**Answer to the Question No. – 3**

bool dfs(int node, int parent) {

visited[node] = 1;

for (int i = 0; i < n; i++) {

if (graph[node][i]) {

if (!visited[i]) {

if (dfs(i, node)) return true;

}

else if (i != parent) return true;

}}

return false;

}

bool hasCycle() {

for (int i = 0; i < n; i++) visited[i] = 0;

for (int i = 0; i < n; i++) {

if (!visited[i]) {

if (dfs(i, -1)) return true;

}}

return false;

}

The DFS explores each edge once. If it encounters a visited node that is not the direct parent, a cycle is detected. Here the time complexity is O(n + m) where n is the number of nodes and m is the number of edges.

**Answer to the Question No. – 4**

**Proof:** suppose, our connected undirected graph G, with edges of unique weights, has two distinct and acceptable minimum spanning tree (MST) T and T’ and let t = W(T) = W(T’).

Next consider the overlap of T and T’. In this overlap we will see some cycle c with k edges, where k-1 edges are from T and k-1 edges from T’. Consider the heaviest edge e on this cycle and assume without loss of generality that e is definitely in T. Because all edge weights are unique for any given cycle from our original graph, an MST will not use the heaviest weighted edge in this cycle. Hence e does not belong to any MST, contradicting T is an MST.

Example:

1

2

5

7

9

From the above graph we can see that every edge has distinct weight. The MST will have the edges with weight {1, 2, 5, 7}, as 9 is the largest weight on the other edge we cannot take this edge in MST. So there cannot be any other MST with the edge weight 9 and only one unique MST is possible whether we use Prim’s or Krushkal’s algorithm to find the MST.