CS 345: Algorithms II

Submitted By:
Anirudh Kumar (Y9088)
Chandra Prakash (Y9181)

Assignment 3

1 Solution-1

assignEdgeWeight(S)
{
    $v \leftarrow S$
    $visited[v] \leftarrow true$
    $numPath[v] \leftarrow 0$
    for each vertex x such that $(v, x) \in E$
        if($visited[x] == false$)
            $W[v, x] = numPath[v]$
            if($outDeg[x] == 0$)
                $numPath[x] \leftarrow 1$
            else
                assignEdgeWeight(x)
            $numPath[v] \leftarrow numPath[v] + numPath[x]$
        else
            $W[v, x] = numPath[v]$
            $numPath[v] \leftarrow numPath[v] + numPath[x]$
}

Proof of Algorithm:
Induction:

Base Case: Only one edge is present i.e (u, v)
$W[u, v] = 0$
$numPath[v] = 1$
$numPath[u] = 0 + numPath[v] = 1$

Induction Step
Let there be a graph G with $inDeg[u] = 0$, $numPath[u] = m$, $outDeg[v] = 0$ and all pathIDs are distinct and less than m. Another node w is added to G such that $inDeg[w] = 0$ and has n outgoing edges to the vertices of graph G namely $\{u_1, u_2, ..u_i...u_n\}$.
Number of paths from w to v is given by

$$numPath[w] = \sum_{i=1}^{n} numPath[u_i]$$

Now, consider an edge (w, u_i) and the path u_i to v. Let $W[w, u_i] = x$ then pathID of this path is x+pathID(u_i, v). Then, by our algorithm $W[w, u_i] = x = \sum_{j=1}^{i-1} numPath[u_j]$. Therefore, $x < numPath[w]$ and pathID(u_i, v) is already less than $numPath[u_i]$(by induction).Therefore, pathID of each path less than the number of paths.
Now, we need to show that the new pathIDs are distinct.Let there be two paths (w, u_i, v) and (w, u_j, v) with same pathID then,

$$W[w, u_i] + \text{pathID}(u_i, v) = W[w, u_j] + \text{pathID}(u_j, v)$$

Without the loss of generality let u_j be reached before u_i in the loop. Then,
$W[w, u_i] = W[w, u_j] + \sum_{k=j}^{i-1} numPath[u_k]$.Therefore $W[w, u_i] \geq W[w, u_j] + numPath[u_j]$.
We also have pathID$(u_j, v) < numPath[u_j]$(by induction).Therefore
$W[w, u_i]$ + pathID(u_i, v) ¿ $W[w, u_j]$ + pathID(u_j, v)

Thus we have assigned unique pathIDs from $0 \to N - 1$ with each edge having integral edge weight.

2 Solution-2

Algorithm 1 Optimising projects

function OPTIMAL(N, H)
  for $i \leftarrow 0$ to H **do**
    Optimal[0][i] $\leftarrow 0$
  end for
  for $i \leftarrow 0$ to N **do**
    Optimal[i][0] $\leftarrow 0$
  end for
  $id \leftarrow 1$
  for $n \leftarrow 1$ to N **do**
    for $h \leftarrow 1$ to H **do**
      Optimal[n][h] $\leftarrow \max_{i=0}^{h}(f_{id}[i]$ + Optimal$[n-1][h-i])$
      Let the maximum be achieved for i
      Schedule Project id for time i
      $id \leftarrow id + 1$
    end for
  end for
end function

2.1 Explanation

The above dynamic programming algorithm maintains an $O(NH)$ array Optimal such that Optimal$[n][h]$ represents the maximum total grade points for n projects and h hours to work on these projects. The optimal scheduling for n projects and h hours can be represented by the following recurrence:

$$\text{Optimal}[n][h] = \max_{i=0}^{h}(f_{id}[i] + \text{Optimal}[n-1][h-i])$$

This recurrence essentially states that if we schedule a project for i hours and then recursively calculate the optimal scheduling for the remaining projects in time $h-i$, and maximise the total grade points over all i, we will get the optimal scheduling strategy for n projects. This is exactly what the above algorithm does. It initiates the first row and column with 0 (0 projects $\rightarrow$ 0 grade points, 0 hours $\rightarrow$ 0 grade points). For each n, h the values required are Optimal$[n-1][0]$ to Optimal$[n-1][0]$ so they are already computed.

2.2 Time Complexity

The algorithm requires O(NH) space for storing the matrix. The time complexity is O(NH^2), as for calculting the maximum we require O(H) time and this is done for each entry in matrix.