Nevil Ladani
npl130030@utdallas.edu

Algorithms Project

Variation of Activity Selection Problem

Project Description:

Consider the activity selection problem with N activities from class. Activity a_i has start time s_i, finish time f_i, and profit of P_i if it is scheduled. An activity is *short* if its duration is at most 4 (i.e., a_i is short if f_i-s_i <= 4), and long otherwise. Consider the problem of finding a set S of compatible activities to generate maximum total profit, with the additional restriction that S must have more short activities than long ones.

Input format:

The input has 1 or more problem instances. Each instance has the following format. The first line has N, the number of activities. The next N lines have 3 integers (start, finish, profit) each. The activities are not given in any order.

Approach:

We approach the problem in a similar way as activity selection problem with some difference. We consider total number of activities as 'n'. In activity selection problem we sorted the activities by finish time. Using dynamic programming we kept track of maximum profit for a each case of first i activities (where i goes from 1 to n) with finish time in ascending order. We used the profit stored for first i-1 activities to help us get maximum profit for first i activities. At the end we get maximum profit for first n activities.

The difference is:

-The activities are differentiated into short activities and long activities. Short activities have running time < 5. Long activities have running time >5.

- Instead of just keeping track of profit for first i activities, we keep track of max profit for all case of difference between number of short activities and number of long activities.

- We maintain a 2d matrix (number of activities, difference between number of short and long activities) which is n X 2n. As the difference goes from -n to n.

- We also keep track of previous activity used in a similar fashion.

- We used all the profits stored for first i-1 activities in the row i-1 of matrix, to help us get maximum all cases of maximum profit for first i activities. At the end we get all cases of maximum profit for first n activities.

- We take the cases of maximum profit for n activities in which number of short activities >number of long activities to get the result.

Algorithm:

Input:

A - Activity list with start time, finish time and profit
Output - Maximum profit and all the activities considered.

```
asprun(A, n)
{
        short activities by finish time.
        define SLprofit[n][2n] // matrix to store profit. (Number of activities X Difference)
        define Pre[n][2n] // matrix to store previous activity considered.
        for i ← 1 to 2n+1 do
                SLprofit[1, i] ← -1 // To identify the matrix cells where no profit values exist.
                if  i == n
                        SLprofit[0, i] ← 0 //initialize the profit for 0 short 0 long as 0.
        for i ← 1 to n do
                for  j ← 1 to 2n+1
                        SLprofit[i,j]  ← SLprofit[i-1,j]
                        Pre[i,j] ← -1

                c = i-1
                while f_c > s_j do
                        c ← c-1

                if  (f_i - s_i <5) then
                        for k ← 2 to 2n+1
                                if  ((SLprofit [i,k] <SLprofit [c,k-1] + p_i ) AND (SLprofit [c,k-1] != -1))
                                        SLprofit [i,k] ← SLprofit[c,k-1] + p_i
                                        Pre[i,k] ← c //previous
                else
                        for k ← 1 to 2n
                                if  ((SLprofit [i,k] <SLprofit [c,k+1] + p_i) AND (SLprofit [c,k+1] != -1))
                                        SLprofit [i,k] ← SLprofit[c,k + 1] + p_i
                                        Pre[i,k] ← c //previous

        index=0;
        pMax=0; // maximum profit
        for  (i=n+1 to 2n+1) do // Max profit for n activities with number of short >number of long
        {
                if(pMax<SLprofit[n-1][i])
                {
                        pMax = SLprofit[n-1][i];
                        index = i;
                }
        }
        //printing the max output
```

```
        Print ("Profit = "+pMax)
        Print ("Excess Small = " + (r-n))
        k ← n
        //printing the activities considered along with the indication of its being big or small
        while (k>=0)
        {
                if(Pre[k][r] != -1)
                {
                        if(s_k -  f_k <5)
                        {
                                Print("[S]" , s_k , f_k , p_k )
                                k = Pre[kr]
                                r←r-1

                        }
                        else
                        {

                                Print("[B]", s_k , f_k , p_k)
                                k ← Pre[k,r]
                                r= r+1
                        }
                }
                else
                        k ← k-1
        }
}
```

Proof of Correctness:

We have all the activities sorted by finish time.

Proof of recursion by induction

Base Case:

For n = 1 activities. Two possible cases:

- It's a short activity: Then consider it and the max profit is same as profit of that.
- It's a long activity: It is not considered and the profit is zero.

Inductive hypothesis:

For n<= k. We have the maximum profit recorded in a matrix [number of activities considered(k) X difference between short and long(2k)] for i number of activities considered all the difference between number of short activities and number of long activities. Where i varies from 1 to k.

Inductive Step:

For n= k+1. Two possible cases:

- Activity [k+1] is short.

Step 1- We copy the maximum profits for all difference from row k (k number of activities).

Step 2 - We find the i^{th} row (i number of activities) whose highest finish time (i_i)is less than start time (s_{k+1}) of activity k+1. Which means all its activities are compatible with activity k+1. Where i goes from 1 to k.

Step 3:As we have the profit for all the cases of difference for each i< =k. We can add the p_{k+1} to all the p_i one at a time for each difference and check if its greater than current values. The maximum profit for each case of difference is recorded for row k+1(considering k+1 activities) according to the difference.

At the end of the recursion we profits for all the difference. We take the maximum profit out of all the cases where number of short>number of long

Time Complexity:

We have 2 for loops nested which runs for n and 2n times.
$O(n) = n^2$

Program:

We have used java to program the solution.
We have used eclipse as our IDE.

All the classes used:
asprun.java - Main class
Activity.java - class to define activity.
ActivityComparator.java - class to sort activity to finish time.