CSE548/AMS542 Fall 2013 Analysis of Algorithms

Homework 2

Himanshu Shah
ID : 109324380

Himanshu Shah
109324380

Homework 2

<u>Solution 1 Problem 12 :</u>

Consider a set of n people whom we'll denote by P_1, P_2, P_3, ….., P_n. For each such person we will assign two nodes namely $P_{i_{death}}$ and $P_{i_{birth}}$ where i is the number of the person in the list. We will generate a graph using these node, and will put an edge between these nodes as per the following condition:

- If P_1 and P_2 have lifespans overlapped at least partially then we will have a directed edge from $P_{1_{birth}}$ to $P_{2_{birth}}$.
- If P_1 died before P_2 then there will be a directed edge from $P_{1_{death}}$ to $P_{2_{birth}}$.
- There is always an edge from $P_{1_{birth}}$ to $P_{1_{death}}$.

Now, by checking whether the generated directed graph is a DAG, we can know if the facts are correct or not. This is because, if the facts are not correct then there will be a case where in A will be born before B; B will be born before C, and C will be born before A, thus resulting in a cycle in the directed graph.

If the graph does not contain a cycle, then it can be used to calculate possible birth and death dates of each person. We know that the observations are made on the people who lived over the past 200 years.

Hence by searching for a node that has no incoming edges, assigning that node the earliest date and then giving later dates to other nodes in succession will give us the required results.

Solution 2:

The algorithm for this problem can be given as follows:
We will be using two arrays. Array - 1 to store a sorted list of segments and Array - 2 to store the segments that have been stabbed.

Step -1 : Calculate the length of each segment from the values given by X_L and X_R , and arrange the segments in an array in increasing order of their length.

Step - 2 : For each element in an array, check if it is already stabbed by referring to the Array - 2. Perform step 2 for next element in the array. If not perform step -3

Step - 3 :
Find all the X_L s and X_R s points of other intervals that are inside the interval selected.
Select the point s from these X_L s and X_R s such that they stab the maximum number of intervals that are not yet stabbed,
Add that point s to set P.
check step 2 for next element in the Array-1.

The running time of this algorithm will be the sum of the running time of each of the steps.
Running time for steps:
Step -1 : Here the running time is dominated primarily by the sorting function which is $O(n \log n)$.
Step - 2 : Here the running time is $O(1)$ as it is just a single comparison.
Step -3 : The step for searching the X_L and X_R is the most influencing factor and is $O(n^n)$.
However, if we sort the values of all X_L and X_R then, this time is reduced to $O(n)$.

So the upper bound to the running time of this algorithm is $O(n \log n)$.

Proof: Let us consider that OPT is the optimal solution for this problem.
Now we will prove that the set of points P generated by the algorithm is the same as the set P' generated by our algorithm.

Now as these points in the set P are minimum and each point stabs the line segments at least once, it isn't wrong to say that these points do lie on the segments with the smallest length.

Also, in order to get a set of minimum points, one needs to select a point on the smallest intervals such that the point stabs maximum number of segments.
We know that all such possible points are either over the entire region of the chosen line segment or they are bounded by X_L or X_R points.
Hence if the point in set P is either from theses options. Therefore, it implies that our algorithm detects the same points as the optimal solution.

Problem 3:
A) Problem #:7
Answer:
Consider n jobs $J_1, J_2, \ldots..J_n$ and for each job $J_i = p_i + f_i$ where p_i is the time taken by the task on the super computer and f_i is the time taken by the task on desktop computer.

Now, in order to solved the algorithm. we will sort the jobs by the decreasing order of f_i and select jobs from this list one by one until all the jobs are done.

So algorithm is :

For each job J_i sort the jobs in decreasing order of f_i

Select the job with the highest value in f_i and then the successive item, so on and so forth.

Running time:

The running time of this algorithm depends mainly on the sorting function and we can do that in polynomial time. So the time complexity of this algorithm is polynomial in n.

Proof:
As per the question, the processing done on desktop computer happens in parallel, where as the processing done on the super computer is in serialized form.

Now we would like to submit tasks that take the longest time in processing on a desktop computer first, so that by the time rest of the jobs are being processed, this one gets the maximum time for execution and other tasks can run simultaneously.
For example: Consider two tasks: J_1 and J_2. $J_1 = p_1 + f_1$ and $J_2 = p_2 + f_2$ let $f_1 = x - 1$ and $f_2 = x + 1$ and let p_1 and p_2 are arbitrary values.

Now as we can see, if J_2 is scheduled earlier than J_1, the waiting time will decrease. This is because, the time taken by the jobs to get processed on the super computer will always be the same which is $P_1 + P_2$ doesn't matter the order. Hence proved.

B) Problem #:12
Answer:

Sub question : a)
CLAIM : There exists a valid schedule if and only if each stream i satisfies $b_i \leq \mathrm{rt}_i$
Ans: **False**
Proof:
Consider two streams with rate of transmission $x - 1$ and $x + 1$. The permissible rate is given by x. Let each of the two streams take 1 seconds each, so $t_1 = 1$ and $t_2 = 1$. Here as t_1 and t_2 are 1 sec, the rate of transmission becomes the number of bits $b_1 = x - 1$ and $b_2 = x + 1$.

So as per the claim, each of the streams need to satisfy the condition $b_i \leq \mathrm{rt}_i$ which stream 2 does not however, there exists a schedule, $\{\mathrm{stream}_1, \mathrm{stream}_2\}$ which satisfies the ultimate condition denoted by (*).
Hence, proved.

Sub question : b)

Considering n streams. Stream i consists of a total of b_i bits that need to be sent, at a constant rate, over a period of t_i seconds.

Our algorithm will be :
Step 1: First calculate the data rate for each stream by dividing the number of total bits b_i per stream to the time interval t_i, this will give rate r_i for each stream.

Step 2 : Now sort the video stream in the increasing order of bit rates.

Step 3 : This sorted array becomes the schedule for the video stream.
- Check if the condition : For each natural number t > 0, the total number of bits you send over the time interval from 0 to t cannot exceed rt holds, true.
- Select the video stream if the condition holds true ; and check for the next element in the sorted array.
- If the condition is violated; then there exists no schedule.

Running time:
The running time of this algorithm depends mainly on the sorting function and we can do that in polynomial time. So the time complexity of this algorithm is polynomial in n.

Proof:
As per the problem, one cannot exceed at any given point, the number of total bits sent by a specified value. So, in order to make sure that our schedule follows this, we will choose to take streams that have the least bit rates so that we can use the difference while sending later streams that have a higher rate.

Answer : C) Problem #:17

This problem can be solved by choosing the job with the earliest finishing time first strategy. However, given that some jobs run though out the midnight making the time interval circular, we need to first find a suitable point that has minimum number of tasks scheduled and then proceed with the earliest finishing time strategy.

The following algorithm consists of two main steps:

Step -1 : Find a suitable initial starting time from the time range of 24 hours such that it has the minimum number of scheduled jobs. This can be done by searching for an ideal time [when none of the jobs are scheduled] and if no such time is possible then searching for a time that has the minimum number of jobs to be scheduled.

This can be precisely known from the starting and ending points of the jobs submitted and will take $O(n^2)$ time.

Step - 2 : Once we have a starting point, we need to sort the jobs by increasing finishing time starting from the time we have discovered in the previous step.

We can now select jobs that have the earliest finishing time from the sorted array. Consecutively, to select the next jobs, we will select the next one from the sorted array if it does not conflict with the job already selected.

The main cost involved in this step is that of sorting which can be done in $O(\mathrm{n \log n})$ time.

Thus, we can create an optimal schedule in a total of $O(n^n + \mathrm{n \log n}) = O(n^n)$ time interval which is basically polynomial time in n.

Explanation:
The method of finding the starting point is done so that minimum numbers of events get affected and still we can find an optimum scheduling solution for selecting jobs.
The strategy for scheduling jobs is valid because it accepts jobs that free the resources as soon as possible, leaving us with higher probability of scheduling other jobs in the spare duration.