Assignment 2

Vivek Bansal (Id : 112044493)
CSE 548 : Analysis of Algorithms

Due Date : October 01, 2018

Ans 1. Super-Computer and Personal-Computer Job Scheduling Problem

Algorithm : Execute the jobs on the personal computers in the decreasing order of the jobs finishing times.

Correctness Proof : We will proof correctness of this algorithm using exchange argument. Let S be a schedule which doesn't contain jobs in the decreasing order of their finishing times. There must be 2 jobs J_i and J_j such that finishing time of job J_i is less than finishing time of job J_j, i.e. $f_i < f_j$ and job J_j schedules later than job J_i. Now we will proof there exists another schedule S' which is more optimal than schedule S and whose total finishing time is less than schedule S. We are swapping jobs J_i and J_j such that job J_j schedules before job J_j. On swapping these 2 jobs there is no effect on the other jobs finishing times. Now in new schedule S', job J_j finishes earlier than it would be in older schedule S as it schedules earlier in schedule S'. Also, the scheduler hands off the job J_i to its PC at the same time as the time of job J_j in the older schedule S. Since finishing time of job J_i is less than job J_j so it will finish earlier in this schedule. Thus, both the jobs finish earlier in this new schedule S', so total finishing time of schedule S' is less than schedule S and this solution is more optimal than S.

Time complexity : O(nLogn)

Ans 2. Minimize weighted completion times of a Scheduling Problem

Algorithm : Schedule the jobs in the decreasing order of w_i/t_i.

Correctness Proof : We will proof the correctness of this algorithm using exchange argument. Let S be the schedule which doesn't contain jobs in the decreasing order of w_i/t_i. There must be 2 jobs, job J_i and job J_j such that job J_i schedules earlier than job J_j and whose total completion weighted time is $W_i(T_i + \mathrm{T}) + W_j(T_i + T_j + \mathrm{T})$, where T is the completion time of the jobs preceding these pair of jobs. We will proof there exists another schedule S' whose total completion time is less than schedule S. We will swap 2 jobs, J_i

and J_j such that job J_j schedules earlier than job J_i in new schedule S'. Since the total completion times of jobs preceding these pair of jobs will not change, so that will still remain T. Total completion times after swapping these jobs is $W_j(T_j + T) + W_i(T_i + T_j + T)$. On subtracting completion times of schedule S from schedule S' and cancelling common terms, we will get difference as W_iT_j - W_jT_i. Dividing this whole term by T_iT_j, we get W_i/T_i - W_j/T_j. But since $W_j/T_j > W_i/T_i$, we get the difference as negative which proves that if we will schedule jobs in the decreasing order of w_i/t_i, we will get less completion times of the schedule.

Time Complexity : O(nLogn)

Ans 3. ***Variation of Interval Scheduling Problem***

Algorithm :
1. Let R be the set of given requests.
2. We will separate out the jobs which are to be scheduled in mid-night from the jobs to be scheduled during day-time. Whenever a job start time is in p.m. and end time is in a.m, that means it is a mid-night job and will put this job in set of mid-night jobs. Else, put the job in day-time jobs set. Let's say day-time job set is S1 and mid-night job set is S2.
3. Apply interval scheduling algorithm on the day-time jobs set (S1).
4. Sort the set S1 based on the increasing order of finishing times.
5. While set S1 has more intervals to remove :
    req = Keep picking the first request from S1 which has least ending time.
    Remove all the intervals from set S1 which conflict with request req, i.e. which are overlapping(whose starting time is less than ending time of request req) with request req.
6. Now set S1 contains only non-overlapping jobs. Iterate over mid-night job set (S2) and check if it overlapping with the jobs in set S1. If it is not overlapping, then add it to set S1.
7. Finally set S1 contains the jobs which are non-overlapping. This is our optimal solution.

Correctness Proof : By Induction
Let's say we are considering a request number r and have to decide whether greedy algorithm will pick this request or not. Also assume that r - 1 requests have been processed till now. We have set of jobs (G_1, G_2, ..., G_{r-1}) in our greedy solution(G) and set of jobs (J_1, J_2, ..., J_{r-1}) in our non-greedy solution(J).

1. Base case : When r = 1, greedy algorithm will pick only this job and it has least ending time. So induction base case holds true.
2. Induction Hypothesis : From induction hypothesis, we claim that our assumption holds true for r -1 requests. So we have :

$f(G_{r-1}) < f(J_{r-1})$ - (1)

Also, since the jobs are ordered according to starting and ending times as well in set J, so we have:

f(J_{r-1}) < s(J_r) - (2)

From (1) and (2), we have:

f(G_{r-1}) < s(J_r)

Now for request r, let's say we have f(G_r) > f(J_r). But since request J_r is available in the given intervals to pick (as it has more starting time than ending time of last selected greedy interval) and can be picked by the greedy algorithm so it will pick that request as it has least ending time instead of G_r which has more ending time. Thus it shows that greedy algorithm for this problem always pick that request first which has least ending time.

Time Complexity : O($nLogn$)
Proof of Time Complexity : It will take O(n) time to make 2 sets S1 and S2 and O(nLogn) time for sorting and another O(n) time to check overlapping intervals with night-time jobs.

Ans 4. ***Minimum Altitude Connected Subgraph***

a) Conjecture 1 : The minimum spanning tree of graph G is a minimum-altitude connected subgraph.

Let G = (V,E) is the graph and T is the spanning tree created from graph G.
Now, we assume that our MST, T is not a minimum-altitude connected subgraph. This means that the height of the path P between pair (i,j) in graph G is strictly lower than height of the path P' between pair (i,j) in the MST, T. Let the highest altitude length in path P' is e. Path P doesn't contain edge e but the union of path P and P', i.e. (PUP') forms a cycle C which contains e. So, e is the edge in the cycle with the greatest altitude. From cycle property, e can't belong to MST, T. Thus our assumption of T not being minimum-altitude connected subgraph is incorrect and it is a minimum-altitude connected subgraph.

b) Conjecture 2 : A minimum-altitude connected subgraph contains the edges of minimum spanning tree.

Let G = (V,E) is the graph and T is the spanning tree created from graph G.
Now we assume that the minimum-altitude connected subgraph doesn't contain all edges of MST, T. Let e = (u,v) be an edge which is in MST, T but not in minimum-altitude connected subgraph. Now if we remove edge e from MST, T, then spanning tree is divided into 2 components S and V-S. By the cut property, e is the minimum length edge connecting 2 components S and V-S. Also, any path from u to v in minimum-altitude connected subgraph must pass from component S to V-S at some point of time. If it can't use e then

its height must be greater than a_e . So only edge e is used to connect 2 components and our assumption of minimum-altitude connected subgraph doesn't contain all edges of MST is not valid. It should contain all edges of MST.

Ans 5. ***Hierarchical Metric***

a) Run Kruskal's algorithm to compute σ. When we are adding an edge connecting 2 points in 2 components c_i and c_j, so by the cut property this edge length should be minimum. Let's say this edge length to be e. Add ancestor of these points at the height of length e. Whenever we will consider edge between these 2 points again, they are already part of one component and ancestor was already added at the minimum height, so σ is consistent with d since $\sigma <$ d.

b) Let's assume $\sigma'(p_i,p_j) > \sigma(p_i,p_j)$, it means there exists some edge between Pi and Pj whose length is less than the length when the components got connected in Kruskal' algorithm. But since Kruskal's algorithm add edges in increasing order it is not possible to add an edge of more length when the edge of lesser length is possible between these 2 points. So it is a contradiction and $\sigma'(p_i,p_j) <= \sigma(p_i,p_j)$.

Ans 6. ***Minimization Spanning Tree Problem***

Explanation : Let's say we have a sorted order of edges in the minimum spanning tree. Since edges are a function of time and they are quadratic in nature, our minimum spanning tree changes only when 2 edges change their relative order in the sorted order. This only happens when 2 parabolas corresponding to edges intersect each other. Also, if we have n number of edges so parabolas can intersect atmost n^2 times. So we have to make n^2 intervals and run Kruskal's algorithm on each of them and find the best possible solution among them.

Algorithm:
1. Find out intersecting points of parabolas and divide these points into n^2 intervals so that sorted order on them remains same.
2. For each interval, run Kruskal's algorithm to generate a spanning tree. There will be time t at which cost of spanning tree is minimum for that interval. We find that time t by differentiation over all functions in that interval and then run Kruskal's algorithm on that interval.
3. Find the minimum among n^2 generated spanning trees to give the optimal solution.

Ans 7. ***Modified Kruskal's Algorithm***

a) 1. Modified Kruskal's algorithm suggests that if an edge is not yet added, then add it to the connected component.

2. If it is already added , then add it only if $l_e < d_{uv}/3$.
3. In the spanning tree H, there can be a path between 2 nodes u and v of atmost length $3l_e$.
4. Replace all the edges e(u,v) in G with those paths P(u,v) whose length is atmost $3l_e$. Also, we will keep removing the loops during the construction of the spanning tree.
5. Doing summation of these edges will give us a graph whose path length is atmost $3d_{uv}$.

b) 1. Let's say graph H contains a cycle of length greater than 4. But, since we have $l_e < d_{uv}/3$, so we can't add more edges to it. It shows that graph can't have edges more than n^2.
2. If we have a graph with n number of vertices, and each vertex has $\sqrt{n}$ neighbours, so if we will add all neighbours it will add more than n which is not possible.
3. f(n) is the number of max nodes possible with no cycle of length 4. So we have:

$$\text{f(n)} <= \text{f(n-1)} + \sqrt{n}$$

$$\lim_{n\to\infty} \frac{f(n)}{n^2} = 0$$