CSE 548 Assignment

Divyansh Upreti (SBU ID: 112026646)

1 Prob 7

The overall problem is to schedule n distinct jobs $J_1, J_2, ...J_n$ on a supercomputer in an order such that overall completion time of the job is minimum. We will pick the jobs in order of decreasing duration/finishing time (f_i.

Algorithm 1 scheduleJobsOnSuperComputer()

1: begin
2: Order the jobs in order of decreasing duration/finishing time (f_i).
3: Schedule each job one by one on supercomputer starting from the job with highest finishing time
4: end

Time complexity = $\mathcal{O}(n * log(n)), n = number of jobs$ (because to sort jobs time taken will be $\mathcal{O}(n * log(n)$

We can prove the above solution is optimal using exchange argument. Suppose O is the optimal solution and S is our solution produced by above algorithm. We will convert O to S and will see that optimality doesn't break and hence our solution S is no worse than optimal solution or in other words, our solution is optimal solution.

We choose two adjacent jobs in O, O_i and O_j such that f_i is less than f_j (inversion). Let's swap these jobs and make new solution $O^{'}$ and compare with O. In $O^{'}$, O_j is scheduled first and also has lesser finishing time and so will be completed much before than in solution O. For the job O_i, the completion time will not impact the overall solution as it will schedule on completion of O_j and will complete in time equal to sum of completion time of both jobs like in solution O. Thus, $O^{'}$ is also optimal as it will give the same overall completion time. We will do similar inversions in O till we get to situation where jobs are arranged in decreasing order of finishing time. This will be nothing but our solution S and it will be optimal. Therefore, by exchange argument, our solution S is optimal.

2 Prob 13

Customers = 1,2,...n
For each customer, let completion time be $C_1, C_2, ...C_n$
Each i_{th} customer has a weight w_i
To minimize: $\sum_{i=1}^{n} w_i * C_i$

To minimize the above ratio, we can see that we need to order jobs in decreasing order of C_i (inversely proportional to the result so when time is less, schedule greater duration job first) and increasing order of w_i (customer's importance).

Combining the above two statements, we can say that we need to schedule jobs in decreasing order of w_i/C_i.

Algorithm 2 scheduleJobsForCustomers()

1: begin
2: Order the jobs in order of decreasing w_i/C_i.
3: Schedule each job one by one on the machine.
4: end

Time complexity = $\mathcal{O}(n * log(n))$, n = number of customers (because to sort, time taken will be $\mathcal{O}(n * log(n)$

We will prove the optimality with an exhcange argument similar to Problem 1. Let the optimal solution O contains one or more pairs of jobs i.e. O_i and O_j scheduled in 'inverted' order, i.e. w_i/C_i is less than w_j/C_j. We will pick one adjacent pair and invert them to make it similar to our solution (say S). Before inversion, say the sum for O_i was $w_i * t_i$ so, for O_j, the sum will be $W_j * (t_i + t_j)$. After inversion, the sum for O_j will be $w_j * t_j$ and so the sum for O_i will be $w_i * (t_j + t_i)$. If we subtract the addition of latter two results with sum of previous two, we will get the equation $w_i * t_j - w_j * t_i$. As w_j/t_j is greater than w_i/t_i, the resulting effect on optimal solution will not be much. This can be said for all such inverted pairs. And thus, if we rearrange all such pairs to make a situation similar to our solution without affecting optimal solution O, we can say that our solution S is optimal.

3 Prob 17

Given jobs $J_1, J_2, ...J_n$ to be run on a processor.
Accept as many jobs as possible out of them.
In this solution, the goal is to give the size of the set of maximum number of non-overlapping jobs.
Assumption - No two jobs have same start/end time.

We can do this by the following algorithm:

Algorithm 3 scheduleJobsForCustomers()

1: begin
2: Sort the jobs in order of increasing start time.
3: Maintain a dictionary for each job and at each time when a new job comes, put the new job in the dictionary of each ended job.
4: Output the size of the maximum dictionary item.
5: end

Time complexity = $\mathcal{O}(n^2)$, n = number of jobs
Space complexity = $\mathcal{O}(n)$, for additonal dictionary space for n jobs.

Essentially, we are seeing in the above algorithm, that each item can be scheduled with how many number of non overlapping jobs. If we remove each job from a schedule, we will have a result which is denoted by each dictionary item's values. Out of n such results, we will choose the result with maximum size and this will take $\mathcal{O}(n^2)$ time.

4 Prob 20

Given n towns.
$G = (V, E)$, V = Number of towns, E = Roads connecting towns.
Each edge cost = a_e, each a_e is unique
We need to find the minimum possible height path (winter optimal) over $G(V, E)$.
$E^{'}(E^{'} \subset E)$ denotes edges for such an optimal path.

- To prove: $G(V, E^{'})$, the minimum spanning tree of G with respect to edge weights a_e is a minimum altitude connected sub-graph.
  Proof: Suppose $G(V, E^{'})$ is not MST, but not the one with optimal path. So there must be some other tree $G(V, E^{*})(E^{*}$ = set of edges covering all nodes in the tree) with such a path having a lesser altitude edge. But since, $G(V, E^{'})$ is an MST with all weights such that the summation is least and also, each edge weight a_e is unique, $G(V, E^{'})$ can be the only MST for the graph G, hence the existence of other tree $G(V, E^{*})$ is a contradiction. Therefore, the only unique minimum altitude connected sub-graph is also the MST $G(V, E^{'})$.

- To prove: $G(V, E^{'})$, sub-graph of G is is a minimum altitude connected sub-graph iff it contains edges $E^{'}$ of the minimum spanning tree .
  Proof: Suppose $G(V, E^{'})$ contains an edge e^{*} that is not in MST. Hence, weight $a_{e^{*}}$ will be greater than an edge in MST. From first proof, we have shown that MST of graph will be minimum altitude connected sub-graph. However, if the minimum altitude connected sub-graph has a sum greater than any other path in the graph, it no longer remains a minimum altitude path. Hence it is contradiction that the graph contains such an heavier edge e^{*}. Therefore, the only edges it contains are the edges of MST.

5 Prob 25

$P = P_1, P_2, P_3, ...P_n$
Given hierarchical metric τ :
A distance function $d(p_i, p_j)$ for each pair of points p_i, p_j
$\tau(P_i, P_j)$ = Least common ancestor v of leaves P_i and P_j.
$\tau(P_i, P_j) = h_v$

- To Prove that for all pairs in P:
  $\tau(P_i, P_j) \leq d(P_i, P_j)$
  Proof: Case 1: When building the connected components, the least common ancestor of p_i and p_j is either of the two points. In this case $\tau(P_i, P_j) = d(P_i, P_j)$. The minimum spanning tree that connects components while building tree can have a case that the two components where p_i and p_j lie, get connected by an edge comprising two points itself. Hence, there will be equality in that case.
  Case 2: When building the connected components, the least common ancestor of p_i and p_j is a node that connects the two components containing p_i and p_j with minimum lengths. In this case, $\tau(P_i, P_j) < d(P_i, P_j)$ because it it wasn't so, then minimum spanning tree algorithm would have already picked $d(P_i, P_j)$ for connecting the two components containing p_i and p_j.
  Hence, $\tau(P_i, P_j) \leq d(P_i, P_j)$ is always true

- For any other $\tau^{'}$,
  Proof: $\tau^{'}(P_i, P_j) \leq \tau(P_i, P_j)$
  Suppose there is another $\tau^{'}(P_i, P_j)$ that exists and is consistent with distance function $d(p_i, p_j)$ for each pair of points p_i, p_j. Assume $\tau^{'}(P_i, P_j) > \tau(P_i, P_j)$ (which we will contradict). If it is so, then while creating $\tau^{'}(P_i, P_j)$, the MST algorithm must have picked up a connecting edge of length smaller than it picked while creating $\tau(P_i, P_j)$. But this is not possible, as MST algorithm is correct and always picks the smallest edge. Hence, this is a contradictory statement, Therefore, $\tau^{'}(P_i, P_j)$ should always be $\leq \tau(P_i, P_j)$

6 Problem 26

For a graph $G = (V, E)$
Each edge e $\in$ E is time varying with $f_e(t)$
Cost of MST = $C_G(t)$
Each $f_e(t) = a_e t^2 + b_e t + c, a_e > 0$ (Parabolic functions)
We need to find a value of t at which $C_G(t)$ will be minimum.
Approach to solve: The edges in the problem have time varying edge functions (Parabolic curves). We can find the value of t at which $C_G(t)$ will be minimum using following steps:

We will need to run Kruskal's algorithm in various stages. We will plot the edges curves on time axis. We will use Kruskal's algorithm, which has an important property that it provides constant results(even if time is varying) unless there is a change in the order of values of edge costs. In our case, this change will happen only when the two parabolic curves intersect - after which the opposite situation will hold true (the bigger curve will be smaller and the smaller curve will have larger value).

So we cannot employ Kruskal algorithm in one pass. However, we can employ Kruskal algorithm in the intervals when there was no change in the sorted order. Hence for all such intervals, in the duration before the curves intersect, we will run our Kruskal algorithm in each interval to find the optimum quadratic function. This function we will use to find the minimum time t and return this minimum time.

Implementation of algorithm could not be attempted.

7 Prob 31

Given, $G = (V, E)$.
$H = (V, F)$
Spanning subgraph H built using Kruskal algorithm variant which adds an edge to H if no edge between verices exists, or adds an edge to H if an edge already exists but the length of new edge added satisfies: $3 * l_e <$ already existing length

To prove: For every pair of $(u, v) \in V$, length of shortest u-v path in H is $\leq$ three times length of shortest path of u-v in G.
Proof: This proof directly follows from the property that the algorithm makes sub-graph H by adding edges if the lengths of new edges added satisfy: $3 * l_e <$ already existing edge length. Suppose, there is such a short length edge that existed between two vertices in G and it also is present in H. Now if a second edge comes, it can be added iff the condition ($3 * l_e <$ already existing edge length) holds true. Therefore, for every pair of $(u, v) \in V$, length of shortest u-v path in H is $\leq$ three times length of shortest path of u-v in G.

To prove: if $f(n)$ is max number of edges that can be produced as output of above algorithm, then if n is number of nodes of graph G, $\lim_{n\to\infty} \frac{f(n)}{n^2} = 0$
Proof: Could not be attempted.