AOA Assignment 2

Sunny Shah - 112044068

February 13, 2019

Problem 1 - (Page 190 problem #7)

Solution

The optimal solution will be to sort the jobs in decreasing order of their F time and then process each job one by one.

Proof

Let us consider a optimal solution to the problem, where the n jobs are ordered as:
J_1 , J_2 , J_3 .... J_i , J_{i+1} .... J_n

Then for some J_i and J_{i+1} :
Let C be the time when J_i enters the supercomputer,
then job J_i will complete at (C + P_i + F_i) and job J_{i+1} will complete at (C + P_{i+1} + F_{i+1})

If the solution is optimal, then swapping J_i and J_{i+1} will take more time, i.e.

Max(P_i + F_i , P_i + P_{i+1} + F_{i+1}) < Max(P_i + F_i , P_i + P_{i+1} + F_{i+1})
Now, lets contradict the above solution statement and assume $F_{i+1} > F_i$

Then on the left side (Max(P_i + F_i , P_i + P_{i+1} + F_{i+1})) of the equation,

$P_i + P_{i+1} + F_{i+1} > P_i + F_i$ as $F_{i+1} > F_i$

therefore, it is enough to prove that the term $P_i + P_{i+1} + F_{i+1}$ will never be less than the right side(Max(P_i + F_i , P_i + P_{i+1} + F_{i+1})) of the equation to contradict the assumption made.

Thus, we just need to prove that following statements cannot be true:

1) $P_i + P_{i+1} + F_{i+1} < P_i + F_i$
i.e. $P_{i+1} + F_{i+1} < F_i$ which cannot be true as this contradicts our assumption that $F_{i+1} > F_i$

and

2) $P_i + P_{i+1} + F_{i+1} < P_i + P_{i+1} + F_{i+1}$
i.e. $P_i < 0$ which cannot be true as jobs can only have positive process time.

Thus, if $F_{i+1} > F_i$, the solution is not optimal and therefore, we will have to sort the jobs in decreasing order of their F time and then proces them one by one.

Problem 2 - (Page 190 problem #13)

Solution

Sort the jobs in decreasing order of the ratio of $\frac{w_i}{t_i}$ and then process them one by one to give the optimal solution i.e. minimize the weighted sum of the completion times $\sum_{i=1}^{n} w_i c_i$

Proof

Let us consider the client orders in the following order which is giving us the optimal solution:
J_1 , J_2 , J_3 .... J_i , J_{i+1} .... J_n
Let us say J_{i-1} complete at time C, then the completion time of J_i will be (C + t_i) and similarly for J_{i+1}, it will be (C + t_i + t_{i+1}).

Now, (C + t_i) + (C + t_i + t_{i+1}) is one part of the function of our minimization problem. Since, this order of the job is giving us the optimal answer, then

(C + t_i) * w_i + (C + t_i + t_{i+1}) * w_{i+1} ≤ (C + t_{i+1}) * w_{i+1} + (C + t_{i+1} + t_i) * w_i

i.e. (t_i) * w_i + (t_i + t_{i+1}) * w_{i+1} ≤ (t_{i+1}) * w_{i+1} + (t_{i+1} + t_i) * w_i

i.e. w_{i+1} * t_i ≤ w_i * t_{i+1}

i.e. $\frac{w_{i+1}}{t_{i+1}} \leq \frac{w_i}{t_i}$

i.e. $\frac{w_i}{t_i} \geq \frac{w_{i+1}}{t_{i+1}}$

Thus, we should sort the jobs in decreasing order of their $\frac{w_i}{t_i}$ ratio and then execute them one after another to mimimize the cost function.

Problem 3 - (Page 190 problem #17)

Solution

- Since, the jobs need to be scheduled for a day, we can never know what should be the start job for the day i.e. on a day start, the executing job can be the one that started before 12AM or the job that starts after 12AM.
- However, unlike a normal interval scheduling problem, where, start time is always lesser than the end time for a job, here, we cannot pick any particular job from the list of the jobs to start with, since it may or may not be the part of the optimal list of jobs.
- Thus, the only optimal solution would be to start with each job as start of the optimal job list and use a interval scheduling as normal starting from that job.
- However, in this case, to check if a job is eligible to be added to the list, we will have to ensure that the next job's runtime (start, end) range to not overlap with the current last element in the list of chosen jobs.
- One really important check that is required in this problem is to check that the new job to be added to not overlap with the first job in the list of the jobs chosen.
- Following is the algorithm to find the maximum non-overlapping jobs for the day:

```
=> Sort the jobs (s_1, e_1), (s_2, e_2) .... (s_n, e_n),
based on decreasing order of their end time.

let min_answer = inf

for i in range(1...n):

    start_of_first_job = s_i
    prev_s_i = s_i
    prev_e_i = e_i
    curr_answer = 0

    for j in range(i ... n+i):
        actual_job_id = j%n  # this is because we need to visit jobs in circular path

        if (start_actual_job_id, end_actual_job_id)
                do not overlap with
            (prev_s_i, prev_e_i)
                    and
            (end_actual_job_id<start_of_first_job) ):

                prev_s_i = start_actual_job_id
                prev_e_i = end_actual_job_id
                curr_answer++

    min_answer = min(min_answer, curr_answer)

return min_answer
```

Time Complexity:

The given algorithm tries to apply the interval scheduling algorithm starting from each interval. Thus, it will perform the same n operations for each of the n intervals.

Thus, the overall complexity of the algorithm is $O(n^2)$

Problem 4 - (Page 190 problem #20)

Solution

(a) - The minimum spanning tree of G, with respect to the edge weights a_e, is a minimum-altitude connected subgraph

Proof

- A minimum spanning tree is one where, the sum of all the edges of the tree is minimum.
- Let us consider a minimum spanning tree MST, that is not a minimum altitude tree MAT. Thus MAT contains a path E(u,v) that is not present in MST.
- Now, let was consider the path $(u^{'}, v)$ in MST, which according to our claim has height greater than height of E.
- MST did not include E, for the possible reason that it was causing a cycle. It means that MST might have explored and found nodes u,v being part of the same connected sub-tree. But if we remove the edge $(u^{'}, v)$ of higher altitude and replace it with (u,v), keeping the graph still connected. By doing this, we have found a spanning tree with minimum weight sum than MST.
- Thus, this contradicts the cliam that the considered MST was not the optimal minimum spanning tree.

(b) - A subgraph (v,$E^{'}$) is a minimum-altitude connected subgraph if and only if it contains the edges of the minimum spanning tree

Proof

- Let us assume that there is a MST and MAT, such that MAT does not contain an edge $E(u, v)$ present in MST.
- If MAT contains edge $E^{'}$ between (u,v), then we know, that E' height will be greater than e, since E is a part of MST.
- Thus, the altitude between u,v is greater in MAT as height($E^{'}$) > height(E)).
- Therefore, MAT s not the optimal altitude subtree. Thus, it should include the edge E in order to become a optimal minimum-altitude subgraph.

Problem 5 - (Page 190 problem #25)

Give a polynomial-time algorithm that takes the distance function d and produces a hierarchical metric, with the following properties.

(i) τ is consistent with d

Solution

- The most consistent way to build τ would be to make $\tau(P_i, P_j) = 0$ for all i,j
- But if we assign the value of 0 to all $\tau(P_i, P_j)$, then it will not meet the second property, as our below mentioned solution will produce consistent τ which has positive values. We can build such a rooted tree along with a consistent τ function using the Kruskal's Minimum Spanning Tree Algorithm.
- Given $d(P_i, P_j)$, we first sort it in ascending order. Now instead of choosing the edge P_i, P_j and directly joining them, we create a parent node P_{ij} and set its height to $d(P_i, P_j)$
- But, if either P_i or P_j or both of them are already part of any subtree, then we join their ancester with a new parent node, whose height will be $d(P_i, P_j)$
- Thus, $\tau(P_i, P_j)$ will still be $d(P_i, P_j)$ for every edge (P_i, P_j) chosen by kruskals algorithm n building MST.
- Consider (P_i, P_j) where we haven't chosen their direct path P_i, P_j as part of MST, then $\tau(P_i, P_j)$ will be less than $d(P_i, P_j)$, since Krushkal's algorithm would have already chosen $d(P_i, P_j)$ if it really was lower than any path from P_i to P_j
- Therefore, the τ build using the proposed method and Kruskal's MST algorithm is consistent.

(ii) if τ' is any other hierarchical metric consistent with d, then, $\tau'(P_i, P_j) \leq \tau(P_i, P_j)$ for each pair of points P_i and P_j

- To prove, $\tau'(P_i, P_j) \leq \tau(P_i, P_j)$, it is enough to prove that there is no consistent hierarchical metric such that $\tau'(P_i, P_j) > \tau(P_i, P_j)$
- Since, kruskal's algorithm will start with the minimum weight edge. Then the first edge(P_i, P_j) choosed by kruskal is the minimum distance between them.
- This means that if there were 2 nodes P_i, P_j joined directly to their parent P_{ij} based on their direct path $d(P_i, P_j)$(which was also minimum), then $\tau(P_i, P_j) = d(P_i, P_j)$.
- However, τ' cannot join them as it will contradict $\tau' < \tau$
- But $\tau'(P_i, P_j) > \tau(P_i, P_j)$ AND we know that $\tau(P_i, P_j) = d(P_i, P_j)$, then it is clear that $\tau'(P_i, P_j) > d(P_i, P_j)$.
- But this will make τ' inconsistent as it is higher than the distance between P_i, P_j.
- This proves that there cannot be any τ', such that $\tau'(P_i, P_j) > \tau(P_i, P_j)$
- Also, if τ' starts with a different pair of points, then sometimes in future it will have to join the minimum direct path (P_i, P_j), since it has to maintain $\tau'(P_i, P_j) \leq d(P_i, P_j)$.
- This would mean that least common ancestor of (P_i, P_j), to have a lower height than its child, which contradicts the consistency property that $h(u) >= h(v)$ where u is parent of v in the graph.
- Therefore, all the consistent τ' will have to be less than the τ we described using the Kruskals MST approach.

Problem 6 - (Page 190 problem #26)

Solution:

- Given a list of edges E ≡ (e,w), it will have the same minimum spanning tree as another list of edges $E^{'} \equiv (e^{'}, w^{'})$, if and only if after sorting $E^{'}$ and E based on their weights, the order of e_i is same in both sorted list of edges.
- Therefore, to find the minimum spanning tree in a graph, where the edge weights change over time, we will have to find number of different spanning trees that can exist in the graph.
- This is to say that we need to find the number of distinct Edge lists E.
- Now, the edge weights of any edges is a function of type $y = ax^2 + bx + c$, which is parabolic in nature. Also, all of this parabolas will be upward facing parabolas as a¿0
- If we plot a parabola for any 2 edges using their respective $y = ax^2 + bx + c$ cost function, we find that they will intersect only at 2 places.
- Just after their point of intersection, the order of these edges will flip in the sorted order of edge list, giving rise to a new spanning tree.
- If all the parabola of each edge intersect at the same times (t_1, t_2), then there are only 3 different spanning trees possible. One before t_1, one between t_1 and t_2 and 3rd after t_3.
- But, on the worst case, the maximum number of spanning trees will be possible if at any time only 2 parabolas meet. Thus, we can have at max (2 * $\binom{N}{2}$) (where, N = number of edges in graph), which comes out to be ((N) * (N-1)).
- This means that we will have to find all the N*(N-1) edge lists and check if we have found the minimum spanning trees for any of them. This, we can find by finding the point of intersection for each pair of edge, which can be done in $O(N^2)$.
- Let us say that, we have found the following times t_i of intersections:
  0, t_1, t_2, t_3, ... , t_n
- Then, we will have to check for minimum spanning tree between each interval, rather than at each point as at the point the edge weights are same for 2 edges.
- Thus, we will can find the MST in a polynomial time using the above approach.

Problem 7 - (Page 190 problem #31)

a) Prove that for every pair of nodes u, v $\in$ V, the length of the shortest u-v path in H is at most three times the length of the shortest u-v path in G.

Solution:

- Consider there is already a edge existing between nodes u, v in H of path length x.
- But, we found a new edge between u, v of length y, such that $3 * y < x$
- Since, we are traversing the edges in increasing order of their edge weight.
- Thus, for $3 * y < x$ to be true, path with cost x should consists of atleast 4 edges.
- Also, if there was a edge between u, v with a distance lower than x, then it would have been already traversed prior to y, as it will be both:
  1) before y in sorted order and
  2) satisfy the constraint $3 * l_e < d_{u,v}$
- Now that we, have proven that x should have atleast 4 paths to get replaced by a y where (3*y < x), it is clear that, x consisting of 3 or less edges will never get replaced by any edge y in the edge list. This is sufficient to prove that x will be only at most 3 times the equivalent shortest distance in G, since for paths having more than 3 edges, there is a fair chance for another edge connecting the endpoints of x reduce their distance.

(b) Despite its ability to approximately preserve shortest-path distances, the subgraph H produced by the algorithm cannot be too dense. Let f(n) denote the maximum number of edges that can possibly be produced as the output of this algorithm, over all n-node input graphs with edge lengths. Prove that

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

Solution: