Assignment - 1

1 February 2021

190050023	Aquib Nawaz
190050030	Rajesh Dasari
190050051	Paavan Kumar
190050106	Sanjana Burman

Question 1

a)

Choose the course x that ends last, discard classes that conflict with x, and recurse.
This fails.
example:

```
|————————————————————————————————————————————————|
            |————————————|       |————————————|
   |——————————————————|                       |———|
```

Since this algorithm picks the lengthiest job which starts first and ends last but 3 jobs can be scheduled.

b)

Choose the course x that starts first, discard all classes that conflict with x, and recurse.
This fails.
example:

```
|————————————————————————————————————————————————|
            |————————————|       |————————————|
   |——————————————————|                       |———|
```

Since this algorithm picks the lengthiest job which starts first and ends last but 3 jobs can be scheduled.

c)

Choose the course x that starts last, discard all classes that conflict with x, and recurse.
This doesn't fail.
Proof:
Let A denote the jobs selected by the algorithm. A = $\{i_1, i_2, i_3, ...i_m\}$.
Let Opt denote the optimal selection. Opt = $\{j_1, j_2, j_3, ...j_k\}$.
Enough to show that |A| = |Opt|, i.e m = k.
Lemma: s(i_r) $\geq$ s(j_r) where s(.) is start time of the course/job.

For r=1, by the choice of i_1, s(i_1) $\geq$ s(j_1) Base case is true
Suppose s(i_{l-1}) $\geq$ s(j_{l-1}) Induction Hypothesis

When A picks its lth job, j_l is available for it to chose from.
Hence s(i_l) $\geq$ s(j_l) By Induction
If a job can be picked by Opt, it can also be picked by A at any given time.
This also shows that A gives the optimal solution.
Hence |A| $\geq$ |Opt|, but m cannot be greater than k, so m = k.

d)

Choose the course x with shortest duration, discard all classes that conflict with x, and recurse.
This fails.
Example

```
 |————————|  |——————————|        |——————————|
       |—————|        |——————————————————|
```

Starting with the shortest job, we can schedule only 2 classes but the optimal must have 3 classes scheduled.

e)

Choose a course x that conflicts with the fewest other courses, discard all classes that conflict with x, and recurse.
This fails.
Example:

```
 |————————|  |——————————|                |——————————|
 |————————|  |——————————|                |——————————|
 |————————|  |——————————|                |——————————|
 |————————|  |——————————|    |—————|
 |————————|  |—————|    |———————|    |—————————|    |————————|
```

If we run the algorithm we get the 3rd course on the third line which is undesirable as all the otpimal solutions do not contain this course . |Opt|=5

f)

If no classes conflict, choose them all. Otherwise, discard the course with longest duration and recurse.
This fails.
Example:

```
 |————————|  |——————————|
       |——————————————|          |————————————————————————|
```

According to the algorithm only 2 classes would be picked for this example but 3 must be picked according to optimal algorithm.

g)

If no classes conflict, choose them all. Otherwise, discard a course that conflicts with the most other courses and recurse.
This fails.
Example:

```
 |————————|  |——————————|        |——————————|
       |—————|        |——————————————————|
```

Since the choice of tie is completely arbitary, if the algorithm by chance picks the second course on the first row then we would get only two classes scheduled which fails to get the number 3 which is by an optimal algorithm, Hence this fails.

h)

Let x be the class with the earliest start time, and let y be the class with the second earliest start time.
• If x and y are disjoint, choose x and recurse on everything but x.
• If x completely contains y, discard x and recurse.
• Otherwise, discard y and recurse
This doesn't fail.
Proof:
Claim: If a course x completely contains other course y then there is a optimal schedule which excludes x.
Suppose there is a optimal schedule S such that $x \in S$
Since y is completely in S, any course that overlaps with y overlaps with x.
Now since no course in S-x must overlap with x [S is optimal]
Therefore no course in S-x will overlap with y [y is contained in x]
Therefore $|S-x+y| = |S|$.
Therefore S-x+y is optimal solution which doesn't contain x.
Claim: If two courses x and y are disjoint,(x and y satisfy conditions of problem) then optimal solution must contain x.
Suppose there is a Optimal solution S which doesn't include x.
$f(x) < f(y)$ [x and y are disjoint, f(.) is finish time of a job/course]
S+x doesn't have any conflicts [All other jobs have s(.) late than x and y.]
$|S+x| > |S|$ [contradiction]
Hence x must be present in optimal Solution.
Claim: If any of the two above don't happen then there exists a optimal solution with y excluded.
Suppose there is an optimal solution S such that $y \in S$
$f(y) > f(x)$ [y is not in x]
Any edge that overlaps with x also overlaps with y [$s(U-x,y) > s(y)$]
Therefore S-y+x has no conflicts.
Therefore $|S-y+x| = |S|$.
Therefore S-y+x is optimal solution which doesn't contain y.
As a direct consequence of these claims if we follow the steps we get an optimal solution since no claim is violated and the claims are mutually exclusive and spanning. This algorithm can also be viewed as the one discussed in class where we take the course with earliest finish time.

i)

If any course x completely contains another course, discard x and recurse. Otherwise, choose the course y that ends last, discard all classes that conflict with y, and recurse.
This doesn't fail.
Proof:
As a direct consequence of first claim proved above in **h)**, we can eliminate all the courses which contain other course as we can always find an optimal solution without the larger course.
Now Since after removal of all such courses, lets say we are left with set X.
Let y be the course with largest f(.), $s(X-y) < s(y)$ [Obvious due to the structure of X]
Now according the proof in **c)**, we can say that y is course with the latest start time, and hence we have a optimal solution including y.

Hence by this method, we can e sure of finding an optimal solution always for X.
Since an optimal Scheduling for X is also an optimal Scheduling for the given problem. This way doesn't fail.

Question 2

a)

Sort the jobs in non-increasing order of the duration required to correct the first part (i.e. $t_{i,1}$) and schedule as per this ordering.

Example:

S.no	$t_{i,1}$	$t_{i,2}$
1	6	3
2	1	7

According to this strategy, task 1 is scheduled first which means the teachers have to wait for 6 units before correcting subjective , therefore time taken = 6 + max(3,1+7) = 14 units.
According to the optimal solution, if task 2 was scheduled first then time taken = 1 + max(6+3,7) = 10 units.
Therefore this strategy fails.

b)

Sort the jobs in non-increasing order of the duration required to correct the second part(i.e. $t_{i,2}$) and schedule as per this ordering.
This doesn't fail.
Proof:
Claim:
Scheduling of task with non-decreasing order of their t_2 gives an optimal solution
Suppose there is a Optimal Schedule S such that y is scheduled just before x with $t_{x,2} > t_{y,2}$.
Let T(S) be the time of completion of the Schedule. an let K be the time at which execution of task y by computer started.
Now consider the Schedule S', where positions of x and y are swapped.
Let T1(S) = the time at which all the taks except x, y are completed. Also T1(S) = T1(S')
Then $T(S) = max(T1(S), K + t_{y,1} + t_{y,2}, K + t_{y,1} + t_{x,1} + t_{x,2})$
And, $T(S') = max(T1(S'), K + t_{x,1} + t_{x,2}, K + t_{y,1} + t_{x,1} + t_{y,2})$
As $K + t_{x,1} + t_{x,2} < K + t_{y,1} + t_{x,1} + t_{x,2}$ and $K + t_{y,1} + t_{x,1} + t_{y,2} < K + t_{y,1} + t_{x,1} + t_{x,2}$
$T(S') \leq T(S)$ But S is optimal so $T(S') = T(S)$ and hence S' is optimal.

c)

Sort the jobs in non-increasing order of the total time taken to finish correcting both the parts (i.e. $t_{i,1}$+ $t_{i,2}$) and schedule as per this ordering.

Example:

S.no	$t_{i,1}$	$t_{i,2}$
1	6	1
2	1	2

Using this greedy approach, we are bound to do Task 1 first which means the teachers have to wait for 6 units before correcting subjective, Total time of completion is 6 + max(1,1+2)= 9 units
But if computer does task 2 first then time of completion is 1 + max(1+6,2) = 8 units [more optimal]
Therefore this strategy fails.

d)

Sort the jobs in non-decreasing order of the duration required to correct the first part (i.e. $t_{i,1}$) and schedule as per this ordering.

Example:

S.no	$t_{i,1}$	$t_{i,2}$
1	6	7
2	4	8

Using this greedy approach, we are bound to do Task 1 first which means the teachers have to wait for 6 units before correcting subjective, Total time of completion is 6 + max(7,4+8) = 18 units
But if computer does Task 2 first then time of completion is 4 + max(6+7,8) = 17 units [more optimal]
Therefore this strategy fails.

e)

Sort the jobs in non-decreasing order of the duration required to correct the second part(i.e. $t_{i,2}$) and schedule as per this ordering.

Example:

S.no	$t_{i,1}$	$t_{i,2}$
1	6	3
2	4	8

Using this greedy approach, we are bound to do Task 1 first which means the teacher have to wait for 6 units before correcting subjective, Total time of completion is 6+max(3,4+8) = 18 units
But if computer does Task 2 first then time of completion is 4 + 6+3,8 = 13 units [mpre optimal]
Therefore this strategy fails.

Question 3

Input: Directed Acyclic graph $G = (V, E)$ with edge costs in adjacency list representation.
Output: True or False

```
Let $E = \{e_1, e_2, ...e_n\}$ such that $\forall i < j d(e_i) \leq d(e_j)$ {$E$ is the sequence of vertex of given graph with their deadlines in non-decreasing order.}
$T \leftarrow 0$ {gives the current time}
for $v \in V$ do
  $color(v) \leftarrow WHITE$
end for
for $i \leq n$ do
  if $color(e_i) = WHITE$ then
    Let $A = ancestors(e_i)$ {Where ancestor(v) is sequence of all nodes including $e_i$ from which a path to $e_i$ exists satisfying precedence order.}
    Compute $A$
    for $w \in A$ do
      if $color(w) = WHITE$ then
        $T \leftarrow T + t(w)$
        $color(w) \leftarrow BLACK$
        if $T > d(w)$ then
          return False
        end if
```

```
            end if
        end for
    end if
end for
return True
```

In the algorithm we are executing task with minimum deadline and its ancestor first in the given precedence order.

Claim

If there exist a schedule S of task such that all the tasks can be executed before deadline and let d_i be the minimum of all deadlines then there exist another schedule S' in which all the ancestors of i^{th} task and i^{th} task is executed first satisfying precedence order which also satisfy all the deadlines.

Proof
Let S be $\{s_1, s_2, s_3, ..., s_j, i, s_{j+1}, ..., s_{n-1}\}$.
Also $ancestors(i)$ in precedence order is a subsequence of $\{s_1, s_2, s_3, ..., s_j\}$
Now from above we can say that $C = \sum_{k=1}^{j} t_{s_k} + t_i \leq d_i$ (as the schedule satisfy deadline)

Let sequence $T = \{s_i | s_i \notin ancestors(i)\}$ with the same preceding order.
Then we can write $S' = ancestors(i)i, T, \{s_{j+1}, ..., s_{n-1}\}$. where XY stands for concatenation of two sequence $XandY$

Now for any task l before s_{j+1} in S' time at completion of l^{th} task is C_l.
$C_l \leq C$ as t_ks are positive for all k.
Also $C_l \leq C \leq d_i \leq d_l$ as d_i is minimum.
Hence $C_l \leq d_l$ and task l is completed before its deadline.
Since the order of task after and including s_{j+1} has not changed and total time elapsed at the starting of s_{j+1} task is same due to same tasks executed(in both S and S' till then) their deadline is also satisfied by S'.
Also since $ancestors(i)$ sequence is satisfying precedence order and the tasks in T are in same order as in S S' preserves precedence order.

After a ancestors and the tasks are executed another subgraph can be produced by removing those vertex.
As subgraph of DAG is also an DAG so above claim can be used again.
Using the above claim we can see that if the algorithm returns false then contrapositive of the claim says there doesn't exist any such schedule S.
Also if the algorithm returns True then the order in which tasks are executed in the algorithm is one such schedule.

Time Analysis
The sequence E can be calculated in $O(|V|log|V|)$ time
Also sequence A for all vertices can be calculated in $O(|V| + |E|)$ time using dfs by maintaining a ancestor map for each parent and modifying the map sequentially by adding the current parent to ancestor map of parent
Also next for loop can have maximum time complexity of $O(|V|^2)$
From here we can see the time complexity of algorithm is $\mathbf{O}(|\mathbf{V}|^2)$ which is polynomial.

Question 4

a)

Input: undirected connected graph G = (V,E) with edge costs in adjacency list representation.
Output: T representing the minimal edge set such that there are no cycles in the graph
Algorithm:

```
Let E' = $\{e_1, e_2, ....e_n\}$ be the set of edges sorted in decreasing order of c(.).
Q $\leftarrow \phi$, i $\leftarrow$ 1
while i $\leq$ n do
    if Q + $e_i$ doesn't have a cycle then
        Q $\leftarrow$ Q + $e_i$
    end if
    i $\leftarrow$ i + 1
end while
Output G-Q
```

Proof:
Let an essential edge,e be defined as an edge which when removed the Graph G becomes disconnected.
Claim: Any such minimal set T doesn't contain an essential edge.
Suppose the set P contains an essential edge e
Since removal of e from the graph did not remove any cycle.
This is because the graph,G is split into two components with total number of cycles same
N(G) = E - V + n(G) [N(.) is the number of cycles, n(.) is number of connected components]
N(G') = E' - V + n(G') = E - 1 - V + n(G) + 1 = N(G) [E' = E - 1, n(G') = n(G) + 1]
Therfore , P-{e} is a set which satisfies all the properties [contradiction]

CLaim: If Q is the maxmium spanning Tree, then Q + T = E
E-T consists of all the essential edges and is therefore connected. [by claim above]
E-T is spanning since it is connected.
E-T doesn't have any cycles. [def. of T']
E-T is a spanning tree.
By def, T is the minimum set that can be removed to make it acyclic
So, E-T is a Maximum Spanning Tree.

We therefore use a modified version of **kruskal's algorithm** where edges are sorted in descending order, which is a direct consequence of cut property.

Time Complexity
Since this is modified Kruskal's algorithm, the time complexity remians same since the change is only in sorting the order ascending/descending.
This can be done in the same time.
Therefore this has time complexity **O(E**`log`**V)**

b)

Input: undirected connected graph G = (V,E) with edge costs in adjacency list representation and a specified subset of the vertices R.
Output: P representing the minimum weight set of edges such that the graph G' = (R,P) is connected.
Algoritm:

```
T $\leftarrow \phi$, S $\leftarrow$ R
while |S| < |V| do
```

```
        Let $E_s$ = { (v,w) | v $\in$ S and w $\in$ V - S }
        Compute $E_s$
        Let $\hat{e}$ $\leftarrow$ $\texttt{argmin}_{e\in E_s}$ {c(e)}
        S $\leftarrow$ S + {w}
        T $\leftarrow$ T + {$\hat{e}$}
end while
Output T
```

Proof:
This is a modified version of **Prim's Algorithm** where we start with the given vertex set R instead of any arbitrary vertex.
To argue about its correctness,
By the cut property each time we are adding the minimum cost edge to the set T of all other possible edges, Hence this edge must be a part of the minimum spanning tree.
This is essentially like considering the set R to be a single vertex since we are not interested in adding edges between R.
So, we are required to find the Minimum spanning Tree considering R to be a single vertex, The modification done to the algorithm does exactly that because no edge is added between vertices of R and we get the min spanning Tree for this new Graph
There is a path from every edge in V-R to R
Suppose at some stage the vertex set is S = R+S'
At this stage if we add an edge e acc. to algorithm, if e was from R to V-R, then we are done. Suppose not, e was from V-R to V-R, then by induction hypothesis v in V-R was already connected to a vertex in R, so there is a path from w to R
Hence at every stage we are adding a min cost edge such that there is a path from V-R to R
Hence the algorithm is correct.
Time Complexity:
Since this being a modified version of **Prim's Algorithm**, The time complexity is just the same since only the initial conditions are changed.
Time complexity will be (V-R)$\texttt{log}$V + E$\texttt{log}$ V
There the time complexity of this algorithm is **O((E+V)$\texttt{log}$V)**

c)

Input: undirected connected graph G = (V,E) with edge costs in adjacency list representation.
Output: p[1..|V|], representing the set of edges in second MST in G
Algorithm:

```
Let E' = $\{e_1, e_2, ....e_n\}$ be the set of edges sorted in increasing order of weight.
Q $\leftarrow$ $\phi$, T $\leftarrow$ $\phi$, i $\leftarrow$ 1, c($\phi$) $\leftarrow$ $\infty$
while i $\leq$ n do
    if Q + $e_i$ doesn't have a cycle then
        Q $\leftarrow$ Q + $e_i$
    end if
    i $\leftarrow$ i + 1
end while
i $\leftarrow$ 1, E' = E - Q , m $\leftarrow$ |E-Q|
for $e \in$ Q do
    while i $\leq$ m do
        if Q + $e_i$ - $e$ doesn't have a cycle then
```

```
            **if** c(Q + $e_i$ - $e$) < c(T) **then**
                T ← Q + $e_i$ - $e$
            **end if**
        **end if**
        i ← i + 1
    **end while**
    i ← 1
**end for**
Output T
```

Let Q and T be the first and second MSTs respectively of the Graph G.
Claim: Q and T differ in only one edge, i.e a edge in Q is replaced by some other edge to get T.
Proof: |Q| = |T| = |V| - 1 [Trees]

Suppose Q and T differ in two or more edges.
This means that Q-T has two or more edges. Let e_1 be the edge with minimum weight in Q-T.
Consider the Graph T+e_1, This graph has a cycle C.
This cycle C must have some other edge e_2,$e_2 \in$ T-Q. [Otherwise Q would have a cycle]
Suppose w(e_1) > w(e_2) [Assumption]
Consider the Graph Q+e_2, This graph has a cycle C'.
This cycle C' must have some other edge e_1',$e_1' \in$ Q-T. [Otherwise T would have a cycle]
The Graph, Q'= Q+e_2-e_1 is a spanning Tree. [Since the cycle is broken]
We have w(e_1') < w(e_2) [Q' is not MST, therefore edge cost will be more]
From the assumption, we get w(e_1') < w(e_2) < w(e_1) [contradiction]
Therefore w(e_2) > w(e_1) [weight of edges are distinct]
Therefore the Tree,T'= Q+e_2-e_1' would be spanning [The edge cost is lower than T]
T' differs from Q by only one edge and has w(T') < w(T)
Therefore T' is second MST.

We can use **Kruskal's Algorithm** to find Q.
And By using E'-Q which is already sorted in terms of weight of edges.
For each edge e ∈ Q, we try to find a MST excluding the edge e and including the set of edges Q - e.
Of all the MST's obtained we output which has the smallest cost/weight.

Time Analysis

Kruskal's Algorithm - O(E$\log$V)
For each excluded edge finding a MST consumes O(E-V+1)
For all the edges it takes O((V-1)(E-V+1))
Total Time - O(E$\log$V+(V-1)(E-V+1)) = **O(EV)**

The Time complexity of this algorithm can be further reduced to **O(E`log`V)** by finding the maximal edge on the cycle in `log`V using Least common ancestor and other least common ancestor like data structures.
We found this in `https://codeforces.com/blog/entry/9570/`