Theoretical Assignment 3

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
April 11, 2022

Question 1

Bob is a competitive programmer participating in a global coding competition. Competition has N problems and Bob takes t_i time to solve problem i. Given an order in which Bob solves the problems, T_i denotes the time at which Bob submits problem i (no additional time is required to submit a problem). Each problem has an associated penalty weight p_i based on its difficulty to penalize late submission. Bob wants to minimize total penalty score of competition, $\sum_{i=1}^{N} T_i * p_i$.

(a) (15 points) Given t_i and p_i for all N problems, design an $\mathcal{O}(nlogn)$ algorithm to output order in which Bob should solve the problem to minimize his total penalty score.
(b) (5 points) Give the proof of correctness and analyze the time complexity of your algorithm.

Solution

(a) Psuedocode :

```
merge(a[], beg, mid, end)
{
    n1 = mid - beg + 1
    n2 = end - mid

    declare LeftArray[n1], RightArray[n2] //temporary arrays

    /* copy data to temp arrays */
    for (i from 0 to n1-1)
        LeftArray[i] = a[beg + i]
    for (j from 0 to n2-1)
        RightArray[j] = a[mid + 1 + j]

    i = 0, j = 0, k = beg

    while (i < n1 and j < n2)
    {
        if(LeftArray[i].r <= RightArray[j].r)
        {
            a[k] = LeftArray[i]
            i=i+1
        }
        else
        {
            a[k] = RightArray[j]
            j=j+1
        }
        k=k+1
    }
    while (i<n1)
```

```
    {
        a[k] = LeftArray[i]
        i=i+1
        k=k+1
    }

    while (j<n2)
    {
        a[k] = RightArray[j]
        j=j+1
        k=k+1
    }
}

mergeSort(a[], beg, end)
{
    if (beg < end)
    {
        mid = (beg + end) / 2
        mergeSort(a, beg, mid)
        mergeSort(a, mid + 1, end)
        merge(a, beg, mid, end)
    }
}

input arrays t[] and p[] of size N

Create an array X[] of size N and initialize it
//such that the ith index stores the tuple (index idx=i, ratio r=t[i]/p[i]).
for(i from 0 to N-1)
{
    X[i].idx = i
    X[i].r = t[i]/p[i]
}

mergeSort(X, 0, N - 1)

for(i from 0 to N-1)
    print X[i].idx
```

(b) Consider the optimal ordering of problems. Suppose we have a problem i that is followed by problem j in the optimal order. Consider swapping problems i and j.
Note that swapping the order of solving problems i and j does not alter the completion times for every other problem and only changes the completion times for i and j.

Completion times:

Before :	Problem i	$p_i * (C + t_i)$	Problem j	$p_j*(C+t_i+t_j)$
After :	Problem i	$p_i*(C+t_j+t_i)$	Problem j	$p_j * (C + t_j)$

T_i increases by t_j and T_j decreases by t_i. This means that our objective function $\sum_{i=1}^{N} T_i * p_i$ changes by $p_i t_j - p_j t_i$. Since we assumed our order was optimal originally, our objective function cannot decrease after swapping the problems. This means,

$$p_i t_j - p_j t_i \geq 0 \implies \frac{t_j}{p_j} \geq \frac{t_i}{p_i}$$

Therefore, we want to solve problems in increasing order of $\frac{t_i}{p_i}$, the ratio of the time to the penalty of each problem.
Since, the algorithm needs to sort the problems based on the ratio of time to penalty, its time complexity would be $\mathcal{O}(nlogn)$.

Theoretical Assignment 3

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
April 11, 2022

Question 2

There are N cities and M bidirectional roads connecting them. Each road connects exactly two cities, and there may be multiple roads connecting the same two cities. Each road i is d_i km long and has w_i waiting time due to traffic. Fuel consumption of car for travelling road i is given by, where A and B are known constants

$$F_i = A * d_i + B * W_i$$

Alex wants to travel across all N cities, but he uses an old car with limited fuel capacity. All N cities have a fuel station and there is no limit on how many times Alex can refuel his car at any fuel station. If the car travels through road i and car fuel capacity is less than F_i , then Alex will get stuck in the middle of the road, causing a delay. Alex wants to upgrade his car to the minimum possible fuel capacity such that he can travel between any two cities without any delay. Assume that the given graph of cities and roads is a connected graph.

(a) (15 points) Design an $\mathcal{O}(MlogN + MlogM)$ time algorithm to find the minimum fuel capacity of Alex's car given such that he can travel between any two cities for given value of A and B.

(b) (20 points) Alex is also famous business man, so time is very important to him. Alex will choose N-1 roads such that he can travel between any two cities. F is the minimum fuel capacity he needs to travel through these N-1 roads and W is the maximum waiting time of these N-1 roads. Design an $\mathcal{O}(MNlogN + MlogM)$ time algorithm to output these N-1 roads such that value of $F + W$ is minimum for given value of A and B.

Solution

(a) The given problem can be modelled into a graph with the cities as the vertices of the graph and roads as the edges connecting two vertices. The Fuel consumption F_i will be the weight of edge i.
Thus, for finding the minimum fuel capacity to travel all cities, we need to find the maximum edge weight in the minimum spanning tree of the graph.
We find the Minimum Spanning tree using Kruskal's algorithm with Union-Find path compression method.

Algorithm :

```
Sort the edges of the graph G based on their weights Fi

create a MST tree (set of edges)
MST = NULL

//Create disjoint set for each vertex
for each v in G.V do
        MAKE-SET(v)

declare variable max_fuel = -infinity

for (each edge (u,v) in G.E)
{
    // take that edge in MST if it does form a cycle
    if( FIND-SET(u) != FIND-SET(v) )
    {
        MST = MST U {(u, v)} U {(v, u)}
        w = weight(u,v)
        max_fuel = max( max_fuel, w )
        UNION(FIND-SET(u), FIND-SET(v))
    }
}

return max_fuel
```

(b) Algorithm :

```
Sort the edges of the graph G based on their weights Fi+Wi=(A*di+(B+1)*Wi)

create an empty MST tree (set of edges)
MST = NULL

//Create disjoint set for each vertex
for each v in G.V do
        MAKE-SET(v)


for (each edge (u,v) in G.E)
{
    // take that edge in MST if it does form a cycle
    if( FIND-SET(u) != FIND-SET(v) )
    {
        MST = MST U {(u, v)} U {(v, u)}
        print (u,v)
        UNION(FIND-SET(u), FIND-SET(v))
    }
}
```

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
April 11, 2022

Question 3

Company XYZ has decided to end the work from home policy and is observing strong protest from the employees. XYZ has decided to allow all M employees to enter and exit the office at any time indexed from 1 to N. XYZ have stored entry and exit times of employees in arrays EntryTime and ExitTime of size M and the management wants to observe how many employees are present inside the office premises at any given index of time. (EntryTime[i] $<$ ExitTime[i] and employee is considered inside the office for t $\geq$ EntryTime[i] and t $\leq$ ExitTime[i])

(a) (10 points) Design an $\mathcal{O}(MlogN + QlogN)$ algorithm to answer Q queries of management, i.e. output number of employees present inside the office at the given time index. (Hint: Follow multi increment problem)
(b) (15 points) Optimize above approach to design an efficient $\mathcal{O}(M + N + Q)$ algorithm to answer Q queries of management.

Solution

(a) Pseudocode:

```
Define an Array T of size 2N-1 and initialise it with all zeroes

MultiIncrement(i,j)
{
    i = (n-1) + i
    j = (n-1) + j
    T[i] = T[i] + 1
    if(j>i)
    {
        T[j] = T[j] + 1
        while( floor((i-1)/2) != floor((j-1)/2) )
        {
            if(i%2=0)
                T[i+1] = T[i+1] + 1
            if(j%2=1)
                T[j-1] = T[j-1] + 1
            i = floor((i-1)/2)
            j = floor((j-1)/2)
        }
    }
}

Report(i)
{
    i = (n-1) + i
    val = 0
    while(i>0)
    {
        val = val + T[i]
        i = floor((i-1)/2)
    }
    return val
}

main()
{
    input arrays EntryTime[M] and ExitTime[M]
    for (i from 0 to M-1)
        MultiIncrement(EntryTime[i]-1, ExitTime[i]-1)

    input Q  //(number of queries)
    for (i from 0 to Q-1)
    {
        input ti
        noOfEmployees = Report(ti-1)
        print noOfEmployees
    }
}
```

(b) Pseudocode:

```
Define integer array T[] size N and initialise it with all zeroes

input arrays EntryTime[M] and ExitTime[M]

Define an integer array sum of size N and initialise it with all zeroes

for (i from 0 to M-1)
{
    sum[EntryTime[i]-1] = sum[EntryTime[i]-1] + 1

    if (ExitTime[i]) < M)
        sum[ExitTime[i]] = sum[ExitTime[i]] - 1
}

T[0] = T[0] + sum[0];
for (i from 0 to N-1)
{
    sum[i] = sum[i] + sum[i - 1]
    T[i] = T[i] + sum[i]
}


input Q  //(number of queries)
for (i from 0 to Q-1)
{
    input ti
    print T[ti-1]
}
```

Theoretical Assignment 2

Kaushik Raj V Nadar
Roll No.: 200499
ESO207A: Data Structures and Algorithms
April 11, 2022

Question 4

Company XYZ is now fed up of tantrums of the employees and has decided to only retain K of the existing M developers. Each developer has two characteristics - Talent and Skills both stored in arrays EmpTalent and EmpSkills of size M respectively.
Efficiency of the group of any K developers is defined as

$$(\sum_{i=1}^{k} EmpSkills[i]) * min_{i \in 1 \ldots K}(EmpTalent[i])$$

Help management to retain K employees such that they maximize the efficiency of the group of retained employees.

(a) (5 points) Design an brute force $\mathcal{O}(M^2)$ algorithm to output the maximum efficiency of K employees retained and prove its correctness.
(b) (15 points) Design an efficient $\mathcal{O}(MlogM)$ algorithm to output the maximum efficiency of K employees retained and prove its correctness.

Solution

(a) Pseudocode:

```
create an array X1[] of size M, which
stores the tuple (idx, EmpSkill, EmpTalent) at the ith index

for( i from 0 to M-1)
{
    X1[i].idx = i
    X1[i].EmpSkill = EmpSkills[i]
    X1[i].EmpTalent = EmpTalent[i]
}

create another array X2[] of size M such that
X2[] = X1[]

Sort the array X1 in the increasing order of X1[i].EmpTalent
Sort the array X2 in the decreasing order of X2[i].EmpSkill

define max_eff = -infinity

for( i from 0 to M-1)
{
    count = 1
    sum = X[i].EmpSkill
    for( j from 0 to M-1)
    {
        if(X1[i].idx != X2[j].idx and X2[j].EmpTalent>=X1[i].EmpTalent)
```

```
            {
                sum = sum + X2[j].EmpSkill
                count = count + 1
            }
            if(count==k)
                break;
        }

        if(count == k)
            max_eff = max(max_eff, sum * X[i].EmpTalent)
    }

    print max_eff
```

Proof of Correctness :
We first encapsulate the EmpSkills and EmpTalent of each employee, along with their id idx, under a single array-like data structure X1.
We create a duplicate X2 of this array X1.
We then sort the array X1 by EmpTalent in increasing order and sort X2 by EmpSkill in decreasing order. Now, we iterate over X1 and for each element in X1, we iterate over X2 and add X2.EmpSkill to the sum. As the EmpSkill is in decreasing order in X2, the employees closest to the starting index will have higher EmpSkill.
We also check the id and EmpTalent while iterating to get the correct result. In this way, we find the maximum efficiency by brute-forcing and finding the efficiency for each minimum value of EmpTalent.
Sorting takes $\mathcal{O}(MlogM)$ time. Since We find the efficiency for each value of EmpTalent, this takes $\mathcal{O}(M^2)$ time. Hence, the overall time complexity for this algorithm will be $\mathcal{O}(M^2)$.
Since two arrays are used, the Space Complexity will be $\mathcal{O}(M)$.

(b) Pseudocode:

```
create an array X[] of size M, which
stores the tuple (EmpSkill, EmpTalent) at the ith index

Sort X1 in increasing order of X[i].EmpTalent

Create an empty stack st
for( i from 0 to M-1 )
    st.push(X[i])

Create an Empty Binary min heap H to store EmpSkill
sum=0
for( i from 0 to k-1 )
{
    top = st.top()
    H.insert(top.EmpSkill)
    st.pop()
    sum = sum + top.EmpSkill
}

max_eff = sum*top.EmpTalent

while(stack st is not empty)
{
    top = st.top()
    if(top.EmpSkill > H.getMin())
    {
        sum = sum + top.EmpSkill - H.getMin()
        H.extractMin()
        H.insert(top.EmpSkill)
    }
    max_eff = max( max_eff, sum*top.EmpTalent)
    st.pop()
}

print max_eff
```

Proof of Correctness :
Like the previous method, we encapsulate the EmpSkill and EmpTalent data into a single array-like data structure. We then sort this array X by EmpTalent in increasing order. We then push the array elements into the stack such that the employee with the highest EmpTalent comes at the top. Then, we form a binary min-heap to store the EmpSkill values. Now, we pop the stack elements and insert these elements into the binary heap until the size of the binary heap becomes k. Then, we check whether the EmpSkill value of the top element is greater than the minimum value of the Binary heap. If yes, we remove the minimum element of the binary heap, insert the top element of the stack, and pop it out. We simultaneously calculate the sum while this process happens and multiply the EmpTalent of the top element of the stack to get the efficiency. We repeat this process until the stack is not empty. In this way, we find the Maximum efficiency.

This algorithm works because as we need to maximize the efficiency, we always try to Maximize the EmpSkill values and check the efficiency for each value of Minimum EmpTalent. Since the EmpTalent is traversed in decreasing order, we don't lose out on the correct answer of maximum efficiency even if we miss some EmpTalent due to the EmpSkill of the employee being smaller.
Sorting takes $\mathcal{O}(MlogM)$ time. Creation of stack takes $\mathcal{O}(M)$ time. Inserting one element into a Binary heap of size n takes $\mathcal{O}(logn)$ time; hence, for large values of M, insertion into the binary heap will eventually take $\mathcal{O}(logM)$ time. In the worst case, insertion into the Binary heap happens for each element in the stack of size M; hence the time complexity for all elements will become $\mathcal{O}(MlogM)$. Therefore, the overall time complexity will be $\mathcal{O}(MlogM)$.
As we have an array, a stack and a binary heap, the space complexity in this case is $\mathcal{O}(M)$.