CSCI570 Fall2023 HW2

Vinit Pitamber Motwani (USC ID: 8187180925)

September 2023

1. In the SGM building at USC Viterbi, there is a need to schedule a series of n classes on a day with varying start and end times. Each class is represented by an interval $[start\_time, end\_time]$, where $start\_time$ is the time when the class begins and end_time is when it concludes. Each class requires the exclusive use of a lecture hall.

   (a) To optimize resource allocation, devise an algorithm using binary heap(s) to determine the minimum number of lecture halls needed to accommodate all the classes without any class overlapping in scheduling. (7 points)

   (b) Analyze and state its worst-case time complexity in terms of n. (3 points)

   Solution :

   (a) **Algorithm**

      i. Initialize the $classes$ variable to 1

      ii. Then we sort the intervals based on $start\_time$ using min heap $Heap1$ (i.e Heap Sort).

      iii. Create another min heap $Heap2$ that will store the end_time of the classes

      iv. Remove the root $deleteMin()$ from $Heap1$ and add it's end time to $Heap2$

      v. While Heap1 is not empty

         A. Pop the smallest element(root) and compare it's $start\_time$ with $Heap2$ root node and keep adding the end_time of each deleted root node to $Heap2$

         B. If the $start\_time$ ($Heap1$ root node) $\leq$ end_time ($Heap2$ root node) increment the $classes$

         C. Else remove the root node from $Heap2$ by performing $deleteMin()$

      vi. Keep repeating the steps from v to viii

   (b) **Worst Case Time Complexity :** $\mathcal{O}(nlog(n))$

      i. Heap Sort will require $\mathcal{O}(nlog(n))$ time

ii. And within the while loop we will run $deleteMin()$ maximum of n times so this operation will also require $\mathcal{O}(nlog(n))$ time

iii. All the remaining comparison operations will take constant time

iv. So total time complexity = $\mathcal{O}(nlog(n) + nlog(n)) \approx \mathcal{O}(nlog(n))$

2. The Thomas Lord Department of Computer Science at USC Viterbi is working on a project to compile research papers from various departments and institutes across USC. Each department maintains a sorted list of its own research papers by publication date, and the USC researchers need to combine all these lists to create a comprehensive catalog sorted by publication date. With limited computing resources on hand, they are facing a tight deadline. To address this challenge, they are seeking the fastest algorithm to merge these sorted lists efficiently, taking into account the total number of research papers (m) and the number of departments (n)

   (a) Devise an algorithm using concepts of binary heap(s). (7 points)

   (b) Analyze and state its worst-case time complexity in terms of m and n. (3 points)

   Solution :

   (a) **Algorithm**

      i. Create a min heap of size n and initialize it with first element of each sorted list.

      ii. While min heap is not empty

         A. Pop the smallest element(root) from min heap

         B. Identify the source list index from which the element was taken.

         C. Add the next element from that list into min heap

      iii. Continue this process until all the elements in all the lists are processed

   (b) **Worst Case Time Complexity :** $\mathcal{O}(nlog(n) + mlog(n))$

      i. Initializing the min-heap with the first element from each of the n lists takes $\mathcal{O}(nlog(n))$ time.

      ii. The main loop iterates until all elements from all source lists are processed. As there are total m research papers In the worst case, this loop will run for m iterations.

      iii. In each iteration of the loop, we perform the following operations:

         A. Pop the smallest element from the min-heap $\mathcal{O}(log(n))$ time.

         B. Insert the next element from the source array into the min-heap $\mathcal{O}(log(n))$ time.

      iv. Since there are m iterations, the total time spent in this loop is $\mathcal{O}(mlog(n))$. So total becomes $\mathcal{O}(nlog(n) + mlog(n))$bfs

3. In an interstellar odyssey, a spaceship embarks on a journey from a celestial origin to a distant target star, equipped with an initial fuel capacity of $'currentFuel'$ units. Along the cosmic highway, there are space refueling stations represented as an array of $'spaceStations'$, each defined as $[distanceToStationFromOrigin, fuelCapacity]$. There are $'n'$ space stations between the celestial origin and the target star. The objective is to determine the minimum number of refueling stops required for the spaceship to reach the target star, which is located $'targetDistance'$ light-years away. The spaceship consumes one unit of fuel per light-year traveled. Upon encountering a space station, it can re- fuel completely by transferring all available $'fuelCapacity'$ units from the station. The challenge is to calculate the $'refuelStops'$ needed for a successful voyage to the target star or return -1 if reaching the destination remains unattainable with the available fuel resources

   (a) Devise an algorithm using concepts of binary heap(s). (7 points)

   (b) Analyze and state its worst-case time complexity in terms of n. (3 points)

   Solution :

   (a) **Algorithm**

      i. $currentFuel$ corresponds to the current distance that can be covered

      ii. Create a max heap that will store the fuel capacity of each $spaceStations$

      iii. While current distance is less than target

         A. We compare current distance with every $spaceStations$ distance from origin

         B. If its greater then we add that stations $fuelCapacity$ to max heap

         C. If its less than that means we have reached the maximum distance and we cannot go further so we need to stop at any one of the stations with highest $fuelCapacity$

         D. Pop the highest element(root) from the max heap and add it to current distance

         E. Increment the stop variable

      iv. Continue this process until we reach the target

   (b) **Worst Case Time Complexity :** $\mathcal{O}(nlog(n))$

      i. In the worst case we will add all the stations $fuelCapacity$ to max heap

      ii. So for getting the highest element it will take $\mathcal{O}(log(n))$ time

      iii. And as there will be n operations in worst case total time complexity will be $\mathcal{O}(nlog(n))$

4. You are tasked with performing operations on binomial min-heaps using a sequence of numbers. Follow the steps below:

   (a) Create a binomial min-heap H1 by inserting the following numbers from left to right: 3, 1, 13, 9, 11, 5, 7, 15. (2 points)

   (b) Perform one deleteMin() operation on H1 to obtain H2. (2 points)

   (c) Create another binomial min-heap H3 by inserting the following numbers from left to right: 8, 12, 4, 2. (2 points)

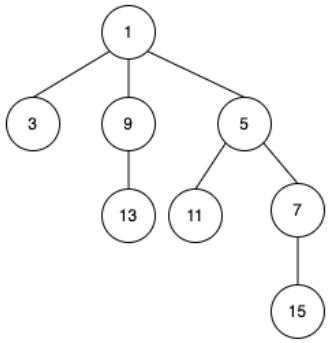
   (d) Merge H2 and H3 to form a new binomial heap, H4. (2 points)

   (e) Perform two deleteMin() operations on H4 to obtain H5. (2 points)

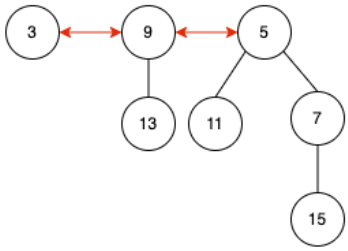
   Note: It is optional to show the intermediate steps in your submission. Only the five final binomial heaps (H1, H2, H3, H4, and H5) will be considered for grading. So, please ensure that you clearly illustrate your final binomial heaps (H1, H2, H3, H4, and H5). You can use online tools like draw.io for drawing these heaps.

   Solution :

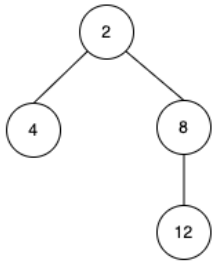
   (a) **Binomial min-heap H1**



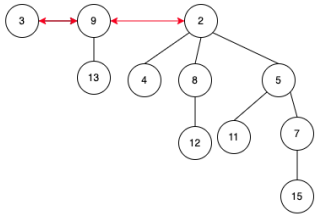
   (b) **Binomial min-heap H2**



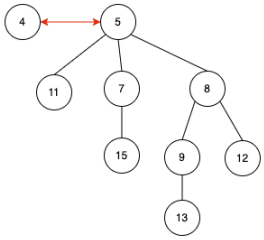
(c) **Binomial min-heap H3**



(d) **Binomial min-heap H4**



(e) **Binomial min-heap H5**



5. If we have a k-th order binomial tree (B_k), which is formed by joining two B_{k-1} trees, then when we remove the root of this k-th order binomial tree, it results in k binomial trees of smaller orders. Prove by mathematical induction. (10 points)

   Solution :

   Base Case : If $k = 1$ then binomial tree B_1 will have only two node and if we remove the root node we get one binomial tree (i.e B_0) it satisfies the statement

Inductive Hypothesis : Assume that B_{k-1} without its root node consist of $k-1$ trees of smaller orders.

Inductive Step : We need to prove that statement holds true for binomial trees of order k i.e B_k

As per our definition B_k is formed by joining two B_{k-1} trees. The joining happens at root node of one of the two trees. So when we remove the root node of the resulting tree we get

(a) One B_{k-1} tree

(b) One B_{k-1} tree without the root node

By Induction Hypothesis we have that B_{k-1} without its root node consist of $k-1$ binomial trees. So, this plus one B_{k-1} tree adds up to k binomial trees of small orders **Hence Proved**

6. Given a weighted undirected graph with all distinct edge costs. Design an algorithm that runs in $\mathcal{O}(V+E)$ to determine if a particular edge e is contained in the minimum spanning tree of the graph. Pseudocode is not required, and you can use common graph algorithms such as DFS, BFS, and Minimum Spanning Tree Algorithms as subroutines without further explanation. You are not required to prove the correctness of your algorithm. (10 points)

Solution :

Algorithm : If there exists a cycle in a graph than the maximum edge of that cycle will not be part of MST, so we can just use DFS traversal.

(a) Perform a DFS traversal of the graph, starting from any vertex. During this traversal, maintain information about the parent of each visited vertex to keep track of the edges in the DFS tree.

(b) During the DFS traversal, if you encounter a vertex that is already marked as visited (but not its parent), it means you've found a back edge i.e the cycle is present.

(c) For each cycle you encounter, keep track of the edge with the maximum weight within that cycle.

(d) Once the DFS traversal is complete, we will get the edge with the maximum weight within each of the cycles.

(e) Check if any of these edges is equal to e if it is than e will not be in the MST of the graph.

(f) Else it will be present in the MST

Time Complexity: As we are using DFS traversal the time complexity will be $\mathcal{O}(V+E)$

7. Given a weighted undirected graph with all distinct edge costs and $E = V + 10$. Design an algorithm that outputs the minimum spanning tree of the graph and runs in $\mathcal{O}(V)$. Pseudocode is not required, and you can use common graph algorithms such as DFS, BFS, and Minimum Spanning Tree Algorithms as subroutines without further explanation. You are not required to prove the correctness of your algorithm. (10 points)

   Solution :

   Algorithm : Prim's Algorithm

   (a) Initialize keys of all vertices as infinite and parent of every vertex as -1.

   (b) Create an empty priority queue (min heap) pq. Every item of pq is a pair(weight, vertex). Weight (or key) is used as first item of pair as first item is by default used to compare two pairs.

   (c) Create an boolean array inMST[] and initialize all vertices as false(not part of MST)

   (d) Insert source vertex into pq and make its key as 0.

   (e) While pq is not empty

      i. Extract minimum key vertex from pq and let the extracted vertex be u.

      ii. Include u in MST using inMST[u] = true.

      iii. Loop through all adjacent of u and do following for every vertex(v)

         A. If inMST[v] = false and key[v] > weight(u, v)

         B. Update key of v, i.e key[v] = weight(u, v)

         C. Insert v into the pq

         D. parent[v] = u

   (f) Print MST edges using parent array.

   Time Complexity: As we have used Prim's Algorithm whoes time complexity is $\mathcal{O}(Vlog(V)+Elog(V))$ but as it is given that $E = V+10$ and also in min heap at any particular time there will be constant set of vertices so all the heap operations will be of constant time so our time complexity reduces down to $\mathcal{O}(V * 1 + (V + 10) * 1)$ which is $\mathcal{O}(V)$

8. There are N people with the i-th person's weight being w_i. A boat can carry at most two people under the max weight limit of $M \geq max_i w_i$. Design a greedy algorithm that finds the minimum number of boats that can carry all N people. Pseudocode is not required, and you can assume the weights are sorted. Use mathematical induction to prove that your algorithm is correct. (10 points)

   Solution :

   Algorithm :

(a) Sort the people based on weight

(b) Initialize *boat* variable to 0

(c) Initialize two pointers as $left$ and $right$ where $left$ points to start of array and $right$ at the end of array

(d) While $left \leq right$ do the following

   i. If the sum of people at $left$ and $right$ is less than equal to max weight M that can be carried by boat, that means a boat can be shared by two people. Increment the $left$ and decrement $right$ to next people. Also increment the *boat* variable

   ii. If the sum is greater than max weight than we decrement the $right$ pointer as only one person can take the boat and it should be the person with max weight and also increment the *boat* variable

(e) Continue the process until $left$ less than $right$ and return the *boat* variable

Correctness of Algorithm using Mathematical Induction

Base Case ($N = 1$): If there is only one person than he will require only one boat and algorithm will also return 1 which is correct.

Induction Hypothesis : Assume that algorithm is true for N people with weights $w_1, ..., w_N$

Induction Step : Prove that algorithm is true for $N + 1$ people with weights $w_1, ..., w_{N+1}$

(a) According to our algorithm $N+1$ heaviest person with weight w_{N+1} will be paired with some one with less weight so that sum of both of person is less than or equal to max weight M of boat or $N+1$ person can take the boat by himself as well.

(b) For the rest of the people with weights $w_1, ..., w_N$ we have assumed in Induction Hypothesis that our algorithm is valid

(c) Therefore, our algorithm correctly determines the minimum number of boats needed for $N + 1$ people.

Hence Proved our algorithm is correct

9. Given $N > 1$ integer arrays with each array having at most M numbers, you are asked to select two numbers from two distinct arrays. Your goal is to find the maximum difference between the two selected numbers among all possible choices. Provide an algorithm that finds it in $\mathcal{O}(NM)$ time. Pseudocode is not required, and you can use common operations for arrays, such as min and max, without further explanation. Prove that your algorithm is correct. You may find proof by contradiction helpful when proving the correctness. (10 points)

Solution :

Algorithm :

(a) Initialize two variables $maxNum$ and $minNum$ to negative infinity and positive infinity respectively

(b) Traverse through each array (from 1 to N)

  i. Find the maximum value in current array

  ii. Find the minimum value in current array

  iii. Compare the obtained maximum value with $maxNum$ and update it based on whichever value is greater

  iv. Compare the obtained minimum value with $minNum$ and update it based on whichever value is smaller

(c) Calculate the maximum difference by $maxNum - minNum$

Time Complexity: Time Complexity of this algorithm will be $\mathcal{O}(NM)$ as there are N arrays and each array has M elements so our algorithm will run for $N * M$ times

Proving the correctness of algorithm using Contradiction

(a) Assume that there exists an optimal solution which will give more maximum difference than our algorithm

(b) Let the pairs of optimal solution be x and y

(c) so $x - y > maxNum - minNum$

(d) The above relation implies $x > maxNum$ and $y < minNum$

(e) But it contradict the definition of $maxNum$ as it the maximum value obtained after traversing all the arrays and $minNum$ is the minimum value obtained after traversing all the arrays

(f) In both cases, we have reached a contradiction. Therefore, our assumption that there exists an optimal solution with a higher maximum difference is incorrect.

Hence Proved our algorithm is correct

10. There are N cities (city 1, to city N) and some flights between these cities. Specifically, there is a direct flight from every city i to city $2i$ (no direct flight from city $2i$ to city i) and another direct flight from every city i to city $i-1$ (no direct flight from city $i-1$ to city i). Given integers a and b, determine if there exists a sequence of flights starting from city a to city b. If so, find the minimum number of flights required to fly from city a to city b. For example,when N =10,a=3,and b=9,the answer is 4 and the corresponding flights are 3→6→5→10→9. You are not required to prove the correctness of your algorithm. (10 points)

Solution :

Approach 1 : Using Gready Approach

```
city ← input()
src ← input()
des ← input()
flight ← 0
if des == city then
    print("NoFlight")
else
    i ← des
    while i ≠ src and i ≤ city do
        if i ≤ src then
            i ← i + 1
        else
            if i%2 == 0 then
                i ← i/2
            else
                i ← i + 1
            end if
        end if
        flight ← flight + 1
    end while
    print(flight)
end if
```

Approach 2 : Using BFS

We can consider cities as nodes and direct flight from one city to another city as edges that will result in unweighted directed graph. So for finding the shortest path from one node to another we will run BFS by considering each edge weight as 1 (same weight for all edges)

Algorithm :

(a) Initialize a queue

(b) Initialize a parent vector and distance vector

(c) Initialize a vector visited(to keep track of visited nodes)

(d) Enqueue the source vertex(start city) in queue and mark it visited in visited vector and mark its parent as -1

(e) While Queue is not empty

  i. Dequeue the vertex from the queue

  ii. Traverse through all the neighbours of that vertex

  iii. If they are not visited mark them as visited and enqueue them in queue

  iv. Update their distance by adding 1 to the distance of vertex (dis[u] = dis[v]+1)

  v. Update their parent to vertex (par[u] = v)

(f) Now for finding minimum flight path from one city to another

(g) First we will check if visited of destination city is false that means there exists no path

(h) Else we will traverse the parent vector starting from destination city and we will stop when we get -1 (parent of start city)

(i) Reverse the obtained path to get path from start city to destination city