Bilkent University

Department of Computer Engineering

CS202 Homework-4

Name: Alperen
Surname: Can
ID: 21601740
Section: 1

1) Minimize Cost – Prim's Algorithm

My implementation of the Prim's Algorithm runs in $O(|V|^3)$ worst case time, when the graph is dense and there are edges from every vertex to every other vertex.

```
int V = 0; // start from vertex 0
int mstCount = 0; // number of vertices in the MST so far
visited[V] = V; // set as visited
mst[mstCount] = V; // put inital vertex in the MST
mstCount++;

// Outer loop is O(|V|)
while(mstCount != numVertices){
    int to, from, leastDuration = INF;
    //(V+1)/2 times on average -> (1+2+...+V)/V = (V+1) /2 -> O((V+1)/2) = O(V)
    for(int i = 0; i < mstCount; i++){ // traverse vertices added to mst
        Airport *v = adjList[mst[i]].getHead()->next;
        // traverse and find the shortest duration in the discovered part of mst
        // Worst case O(V), a vertex can have edges to every other vertex
        while(v != 0){
            if (v -> getDuration() < leastDuration && visited[v->getID()] == -1){
                leastDuration = v -> getDuration();
                from = mst[i];
                to = v->getID();
            }
            v = v -> next;
        }
    }

    MST.addEdge(from, to, leastDuration);
    visited[to] = to; // mark the vertex visited
    mst[mstCount] = to; // add vertice to mst
    mstCount++; // increment the number of vertices in mst
}
```

Outer loop is required to add every vertex to MST, which is $O(|V|)$. The inner for loop traverses every vertex added to MST so far. On average it runs for (1+2+...+V) / V times which is V / 2 and therefore it has $O(|V|)$ complexity. The while loop traverses the edges for the vertices, if the graph is dense and every vertex has an edge to every other vertex, it runs in $O(|V|)$ time as well. Total run time: $O(|V|^3)$

2) Shortest Path – Dijkstra's Algorithm

My implementation of the Dijkstra's algorithm runs in $O(|V|^2)$ time.

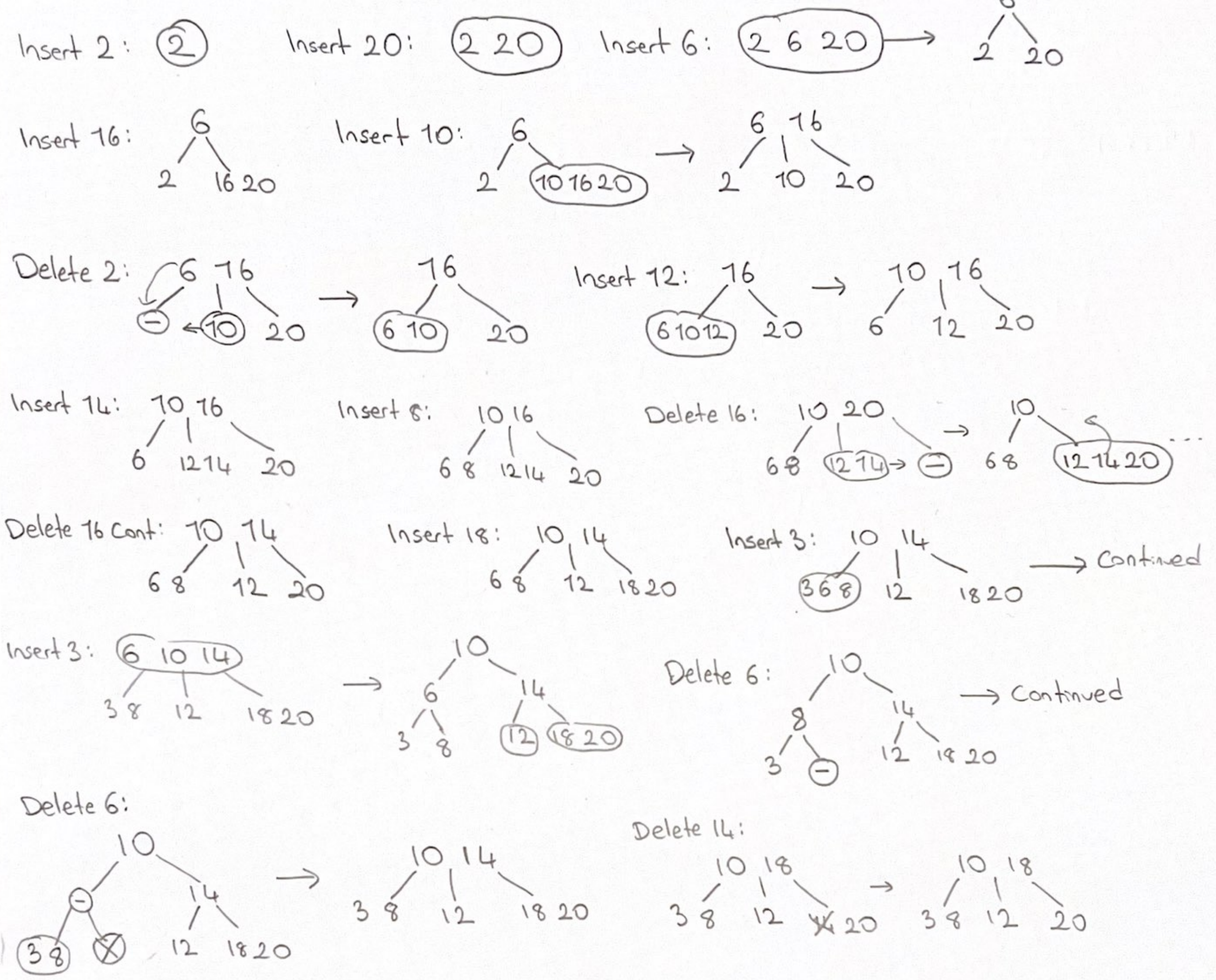
```
for(int step = 1; step < numVertices; step++){
    int v = findSmallestWeight(weight, vertexSet); // O(|V|) to find smallest weight
    vertexSet[v] = v; // add v to the vertexset
    Airport *uAirport = adjList[v].getHead();// search for all vertices u adjacent to v
    // O(|V|) in the worst case when there is edges to every vertex

     while(uAirport != 0){
        if(weight[uAirport->getID()] > weight[v] + uAirport->getDuration()){
            weight[uAirport->getID()] = weight[v] + uAirport->getDuration(); //update
            predecessor[uAirport->getID()] = v; // to indicate a path from v to u
        }
        uAirport = uAirport->next;
    }
}
```

Outer loop runs in $O(|V|)$ in order to find the shortest path to every other vertex starting from source. Helper function ***findSmallestWeight*** runs in $O(|V|)$ time which is a simple for loop over the weights array to find the smallest among them. The while loop visits the edges of a given vertex, it runs in $O(|V|)$ worst case time, when the graph is dense and there are edges to every other vertex from a vertex. Inside the for loop: $O(|V| + |V|) = O(|V|)$.

Total runtime: $O(|V|^2)$

Question 1) a)


Insert 2:
Insert 20:
Insert 6:
Insert 16:
Insert 10:
Delete 2:
Insert 12:
Insert 14:
Insert 8:
Delete 16:
Delete 16 Cont:
Insert 18:
Insert 3:
Continued
Insert 3:
Delete 6:
Continued
Delete 6:
Delete 14:

Question 1) b)

Insert 2: 2 Insert 20: 2 20 Insert 6: 2 6 20 Insert 16:

6
2 20
→
6
2 16 20

Insert 10:

6
2 10 16 20

Delete 2:

6 10
~~2~~ 16 20
→
10
6 16 20

Insert 12:

10
6 12 16 20

Insert 14: Split 4node

10 16
6 12 20
→
10 16
6 12,14 20

Insert 8:

10 16
6 8 12,14 20

Delete 16:

10 14 16
6,8 12 20
→
10 14
6,8 12 ~~16~~ 20

Insert 18:

10 14
6,8 12 18 20

Insert 3:

10 14
3,6,8 12 18 20

Delete 6:

10 14
3,8 12 18 20

Delete 14:

Swap with inorder succ.

10 18
3,8 12 14 20
→

Delete 14

10 18
3,8 12 20

Question 2) Linear Probing

a)

0	1	2	3	4	5	6	7	8	9	10	11	12
26		54		17	69	45	58	32	60			64

Succesful Search: Try 45, 64, 54, 17, 69, 58, 32, 60, 26

45: 6 | 64: 12 | 54: 2 | 17: 4 | 69: 4,5 | 58: 6,7 | 32: 6,7,8 | 60: 8,9 | 26: 0

Average Probes = (1+1+1+1+2+2+3+2+1) / 9 = 14/9 = 1.55 probes

Unsuccesful Search: Try 0,1,2,3,4,5,6,7,8,9,10,11,12

0: 0,1 | 1: 1 | 2: 2,3 | 3: 3 | 4: 4,5,6,7,8,9,10 | 5: 5,6,7,8,9,10 | 6: 6,7,8,9,10

7: 7,8,9,10 | 8: 8,9,10 | 9: 9,10 | 10: 10 | 11: 11 | 12: 12

Avg Probes = (2+1+2+1+7+6+5+4+3+2+1+1+1) / 13 = 36/13

= 2.76 probes

b) Quadratic Probing

0	1	2	3	4	5	6	7	8	9	10	11	12
26		54		17	69	45	58	60		32		64

Succesful Search: Try 45, 64, 54, 17, 69, 58, 32, 60, 26

45: 6 | 64: 12 | 54: 2 | 17: 4 | 69: 4, 5 | 58: 6, 7 | 32: 6, 7, 10 | 60: 8 | 26: 0

Avg Probes = (1+1+1+1+2+2+3+1+1) = 13/9 = 1.44 probes

Unsuccesful Search: Try 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

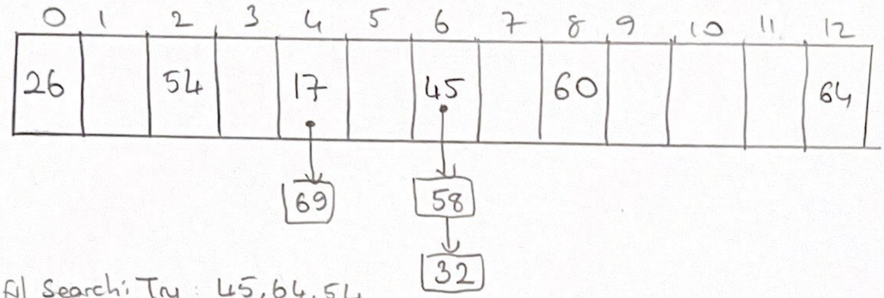
0: 0, 1 | 1: 1 | 2: 2, 3 | 3: 3 | 4: 4, 5, 8, 0, 7, 3 | 5: 5, 6, 9 | 6: 6, 7, 10, 2, 9 | 7: 7, 8, 11

8: 8, 9 | 9: 9 | 10: 10, 11 | 11: 11 | 12: 12, 1, 3

Avg Probes: (2+1+2+1+6+3+5+3+2+1+2+1+3)/13 = 32/13 = 2.46 Probes

c) Seperate Chaining

i→0 : head
i→1 : first node
i→2 : Second node
⋮



Succesful Search: Try : 45, 64, 54, 17, 69, 58, 32, 60, 26

45: 6→0 | 64: 12→0 | 54: 2→0 | 17: 4→0 | 69: 4→0, 4→1 | 58: 6→0, 6→1 | 32: 6→0, 6→1, 6→2 | 60: 8→0 | 26: 0→0

Avg Probes: (1+1+1+1+2+2+3+1+1)/9 = 13/9 = 1.44 probes

Unsuccesful Search: Try 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

0: 0→0, 0→1 | 1: 1→0 | 2: 2→0, 2→1 | 3: 3→0 | 4: 4→0, 4→1, 4→2 | 5: 5→0 | 6: 6→0, 6→1, 6→2, 6→3 | 7: 7→0 | 8: 8→0, 8→1 | 9: 9→0 | 10: 10→0 | 11: 11→0 | 12: 12→0, 12→1

Avg Probes = (2+1+2+1+3+1+4+1+2+1+1+1+2)/13 = 22/13 = 1.69 probes