Liam Knight
Yaseen Khan
Quan Nguyen
Justin Bowen

Team Teqnify 1894 Big-O Ananysis

Function #1: File Read (O(n + e))

Description: Reads the database in from file

Code:

```
void fileRead(stadium a[40], int loops)
{
	int key; +1 (creating variable)
	int key2; +1 (creating variable)
	int distance; +1 (creating variable)
	string teamName; +1 (creating variable)
	string stadiumName; +1 (creating variable)
	int seatingCapacity; +1 (creating variable)
	string location; +1 (creating variable)
	string playingSurface; +1 (creating variable)
	string League; +1 (creating variable)
	int dateOpened; +1 (creating variable)
	int DTCF; //Distance to center Field +1 (creating variable)
	string Typology; +1 (creating variable)
	string roofType; +1 (creating variable)
	fstream infile; +1 (creating variable)
	infile.open("Stadiums.txt"); +1 (Opening a file)

	for(int i = 1; i <= loops; i++) +n (reading n number of datasets into the database)
	{
		infile >> key; +1 (reading input)
		infile.ignore(10000, '\n'); +1 (clearing input buffer)
		getline(infile, teamName); +1 (reading input)
		getline(infile, stadiumName); +1(reading input)
		infile >> seatingCapacity; +1 (reading input)
		infile.ignore(1000, '\n'); +1 (clearing input buffer)
		getline(infile, location); +1 (reading input)
		getline(infile, playingSurface); +1(reading input)
		getline(infile, League); +1 (reading input)
		infile >> dateOpened; +1 (reading input)
		infile.ignore(1000, '\n'); +1 (clearing input buffer)
		infile >> DTCF; +1 (reading input)
		infile.ignore(1000, '\n'); +1 (clearing input buffer)
```

```
                getline(infile, Typology); +1 (reading input)
                getline(infile, roofType); +1 (reading input)

                a[i].setAll(key, teamName, stadiumName, seatingCapacity, location,
playingSurface, League, dateOpened, DTCF, Typology, roofType); +1 (setAll runs in O(1) time)
        }

        infile.close(); +1 (Closing the file)
        infile.open("Distances.txt"); +1 (opening a new file for the edges)
        stadiumName = ""; +1 (setting stadium name)


        for(int i = 1; i <= loops; i++) +n (looping through this n times)
        {
                infile >> key; +1 (reading input)
                infile.ignore(1000, '\n'); +1 (clearing input buffer)
                while(getline(infile, stadiumName)) +e (adding edges to the database)
                {
                        if(stadiumName == "") +1 (if statement)
                        {
                                Break; +1 (breaking from the loop)
                        }
                        infile >> key2; +1 (reading input)
                        infile.ignore(1000, '\n'); +1 (clearing input buffer)
                        infile >> distance; +1 (reading input)
                        infile.ignore(1000, '\n'); +1 (clearing input buffer)
                        a[i].addDistance(key2, distance, stadiumName); +1 (addDistance runs in O(1) time)
                }
        }
}
```

Total running time for fileRead: O(n + e) (n + e + 40)
where n is the total number of stadiums and e is the total number of edges

Function #2: Expand Data (O(n))

Description: Allows administrator to add data into the database via a file during runtime

Code:

```
void expandData(stadium a[40], string fileName, int &count)
{
        int key; +1 (creating variable)
        int key2; +1 (creating variable)
        int distance; +1 (creating variable)
```

```
string teamName; +1 (creating variable)
string stadiumName; +1 (creating variable)
int seatingCapacity; +1 (creating variable)
string location; +1 (creating variable)
string playingSurface; +1 (creating variable)
string League; +1 (creating variable)
int dateOpened; +1 (creating variable)
int DTCF; //Distance to center Field +1 (creating variable)
string Typology; +1 (creating variable)
string roofType; +1 (creating variable)
fstream infile; +1 (creating variable)
infile.open(fileName);+1 (opening a file)

        infile >> key; +1 (reading input)
        infile.ignore(10000, '\n'); +1 (clearing input buffer)
        getline(infile, teamName); +1 (reading input)
        getline(infile, stadiumName); +1 (reading input)
        infile >> seatingCapacity; +1 (reading input)
        infile.ignore(1000, '\n'); +1 (clearing input buffer)
        getline(infile, location);+1 (reading input)
        getline(infile, playingSurface); +1 (reading input)
        getline(infile, League); +1 (reading input)
        infile >> dateOpened; +1 (reading input)
        infile.ignore(1000, '\n'); +1 (clearing input buffer)
        infile >> DTCF;+1 (reading input)
        infile.ignore(1000, '\n'); +1 (clearing input buffer)
        getline(infile, Typology); +1 (reading input)
        getline(infile, roofType);+1 (reading input)

        a[key].setAll(key, teamName, stadiumName, seatingCapacity, location,
playingSurface, League, dateOpened, DTCF, Typology, roofType); +1 (setAll runs in O(1) time)
        while(getline(infile, stadiumName)) +n (Number of edges the stadium has)
        {
                if(stadiumName == "")  +1 (if statement)
                {
                        break;
                }
                infile >> key2; +1 (reading input)
                infile.ignore(1000, '\n'); +1 (clearing input buffer)
                infile >> distance; +1 (reading input)
                infile.ignore(1000, '\n'); +1 (clearing input buffer)
                a[key].addDistance(key2, distance, stadiumName); +1
                                                        (addDistance runs in O(1) time)
        }
```

```
                Count++; +1 (Arithmetic operation)
}
```

Total running time for expandData: O(n) (n+38) Where n is the total number of edges in the added vertice.

Function 3: DFS (O(n + e))

Description: run the DFS algorithm on the database.

Code:

```
void DFS(int v, bool visited[40], stadium a[40])
{
        int key; +1 (creating variable)
        int distance; +1 (creating variable)
        string stadium; +1 (creating variable)
   // Mark the current node as visited and
   // print it
   visited[v] = true; +1 (arithmetic operation)
   //a[v].printName();//Test line

   // Recur for all the vertices adjacent
   // to this vertex
  for (unsigned int i = 0; i != a[v].getDistance().size(); ++i) +e (loop for all of a stadiums edges)
  {
          getDistance(stadium, distance, key, i, a[v].getDistance()); +1 (getDistance runs in O(1))
       if (!visited[key])
       {
          DFS(key, visited, a); +n (we will recur for every vertex in the database)
       }
  }

}
```

Total running time for DFS: O(n + e) (n + e + 5) Where n is the total number of vertices and e is the total number of edges)

Function 4: BFS O(n + e)

Description: run the BFS algorithm on the database

Code:

```
void BFS(int s, bool visited[40], stadium a[40])
{

    // Create a queue for BFS
    list<int> queue; +1 (creating variable)
    int key; +1 (creating variable)
    int distance; +1 (creating variable)
    string stadium = "Chase Field"; +1 (Arithmetic operation)

    // Mark the current node as visited and enqueue it
    visited[s] = true; +1 (Arithmetic operation)
    queue.push_back(s); +1 (push_back runs in O(1) time)

    // 'i' will be used to get all adjacent
    // vertices of a vertex

    while(!queue.empty()) +n (We will loop for all vertices)
    {
        // Dequeue a vertex from queue and print it

        s = *queue.begin(); +1 (Arithmetic operation)
        queue.erase(queue.begin()); +1 (erase runs in O(1))
        //a[s].printName();//test line


        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (unsigned int i = 0; i != a[s].getDistance().size(); ++i) +e (loop for all edges in a vertice)
        {
            getDistance(stadium, distance, key, i, a[s].getDistance());
            if (visited[key] == false)
            {
                visited[key] = true; +1 (Arithmetic operation)
                queue.push_back(key); +1 (push_back runs in O(1) time)
            }
        }
    }
}
```

Total run time for BFS: O(n + e) (n + e + 10) where n is the number of vertices in the graph and e is the total number of edges in the graph

Function #5: Dijkstra (O(n^3))

Description: run Dijkstras algorithm on the database

Code:

```
void dijkstra(int src, bool whitelist[40], int edges[40][40], vector<string> DB)
{

    int dist[40]; // The calculation array.  dist[i] will hold the shortest +1 (creating variable)
    // distance from src to i
    vector<string> output; // our output vector, will contain the shortest distances in order +1
(creating variable)

    bool sptSet[40]; // sptSet[i] will be true if vertex i is included in shortest +1 (creating variable)
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < 40; i++) +n (loop n times)
        dist[i] = INT_MAX, sptSet[i] = false; +2 (Arithmetic)

    // Distance of source vertex from itself is always 0
    dist[src] = 0; +1 (Arithmetic)

    // Find shortest path for all vertices
    for (int count = 0; count < 40; count++) {  +n (loop n times)
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet); +n (minDistance runs in O(n) time)

        // Mark the picked vertex as processed
        sptSet[u] = true; +1 (Arithmetic)

        // Update dist value of the adjacent vertices of the picked vertex.
        for (unsigned int v = 0; v < 40; v++) *n (loop n times)
        {
            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]

            if (!sptSet[v] && edges[u][v] && dist[u] != INT_MAX && dist[u] + edges[u][v] < dist[v])
            +1 (if statement)
            {
                    if(!whitelist[v])
                    {
                            dist[v] = INT_MAX; +1 (Arithmetic)
```

```
                }
                else
                {
                        dist[v] = dist[u] + edges[u][v]; +1 (Arithmetic)
                }
        }
    }
}

//reset sptSet for the output vector
for(int i = 0; i < 40; i++)
{
        sptSet[i] = false; +1 (Arithmetic)
}
//entering stadium names in order of smallest distance-greatest distance
for(unsigned int i = 1; i < DB.size(); i++) +n (loop n times)
{
        int u = minDistance(dist,sptSet); +1 (Arithmetic)
        output.push_back(DB.at(u - 1)); +1 (push_back runs in O(1) time)
        sptSet[u] = true; +1 (Arithmetic)
}
/*/test code
for(unsigned int i = 0; i < output.size(); i++) +n (loop n times)
{
        cout << output.at(i) << endl; +1 (Arithmetic)
}*/

}
```

Total runtime for Dijkstra: O(n^2) (n*n + 3n + 15) Where n is the total number of vertices in the graph