GitGood Big-Oh

Database class

This class handles all of the functions that interact specifically with the SQL database that holds all of the information of the NFL teams. About half of these methods in the Database class have the performance of O(1) and O(n). The methods that are constant time are mainly methods that add/delete elements that are in the SQL database. The methods that have the performance of O(n) are ones that deal with getting all of the team information from the database.

* QVector<QString> GetAllTeams();
  + O(n)
  + This method retrieves all the teams in the database and uses a while loop to store the elements in a vector.
* QVector<QString> GetAFCTeams();
  + O(n)
  + This method retrieves the AFC teams in the database and uses a while loop to store the elements in a vector.
* QVector<QString> GetNationalConferenceTeams();
  + O(n)
  + This method retrieves the NFC teams in the database and uses a while loop to store the elements in a vector.
* void GetTeamBySeatingCapacity(…);
  + O(n)
  + This method retrieves the teams by seating capacity in the database and uses a while loop to store the elements in a vector.
* void GetTeamsSurfaceBySeating(…);
  + O(n)
  + This method retrieves the teams by surface type in the database and uses a while loop to store the elements in a vector.
* void GetPlayersByTeamname(…);
  + O(n)
  + This method retrieves the teams with their star players in the database and uses a while loop to store the elements in a vector.
* void GetNFLStadiums(…);
  + O(n)
  + This method retrieves the teams with NFL stadium in the database and uses a while loop to store the elements in a vector.
* void GetOpenStadiums(…);
  + O(n)
  + This method retrieves the teams with open stadiums in the database and uses a while loop to store the elements in a vector.
* bool AddLasVegas(…);
  + O(n)
  + This method loops through the list and adds Las Vegas to the database.
* bool AddSouvenir(…);
  + O(1)
  + Adds a new soubenir to the database
* void GetSouvenirs(…);
  + O(n)
  + Retrieves elements from database and loops through to add to a list.
* bool DeleteItem(…);
  + O(1)
  + Deletes an item from the database.
* bool UpdateItem(…);
  + O(1)
  + Updates an item from the database.
* double getItemPrice(…);
  + O(1)
  + Gets item price from a certain team from the database
* bool updateStadium(…);
  + O(1)
  + Updates a stadum from the database.
* bool updateSeatCap(…) ;
  + O(1)
  + Updates seating capacity for a staduimfrom the database.
* void GetAllTeamInfo(…);
  + O(n)
  + loops through query and adds teams information to vectors
* void GetOneTeamInfo(…)
  + int seatingSu O(n)
  + loops through query and adds teams information to vectors m();
* void getNodes(…);
  + O(n)
  + Loops through query and adds information to vectors.
* int getCount();
  + O(1)
  + Retrieves count of elements in database
* void getTeamsAndStadiums(…);
  + O(n)
  + Loops through query and adds information to vectors.
* QVector<int> TeamNamesToNodes(…);
  + O(n^2)
  + Has a loop within a loop to edit team information
* int GetTeamNumber(…);
  + O(n)
  + Loops through query and adds information to vectors.
* QString GetTeamName(…);
  + O(n)
  + Loops through query and adds information to vectors.

Graph class

The graph class encapsulates all of the methods that deal with graph traversals and anything else related to graphs. Some of the traversals that we used for this program are the Depth-First Search, Breadth-First Search, MST, and Dijkstra’s Algorithm. Most of these methods have a Big-Oh of O(n^2). The only methods that have a performance of O(n) are methods the DFS and finding the total distance.

* vector< vector< int > > incidentEdges(…);
  + O(n^2)
  + This has a loop within a loop due to an insertion sort
* bool IsAdjacentTo(…);
  + O(1)
  + Returns a bool if a city is adjacent to another city
* vector<int> DFS(…);
  + O(n)
  + Loops through all cities in the edge list
* vector< int > BFS(…);
  + O(n^2)
  + Has a loop within a loop
* float GetTotalDistanceTraveled() const;
  + O(1)
  + Returns the total distance
* void Dijkstra(…);
  + O(n^2)
  + Finds the shortest path for all vertices with a loop and updates the value of the adjacent vertices within a loop
* void MST(…);
  + O(n^2)
  + Update key value and parent index of the adjacent vertices of the picked vertex. Consider only those vertices which are not yet included in MST within a loop
* void fill(…);
  + O(n^2)
  + Fills the matrix using a loop within a loop

Login class

The login class is a fairly simple class. This class deals with logging in as an administrator of the program. There are only four methods in this class. These methods all have the same performance, which is O(1). These methods basically only deal with when buttons in the login pop up window are pressed.

* bool getValid();
  + O(1)
  + Returns a bool to check if login password is valid
* void on\_buttonBox\_accepted();
  + O(1)
  + Runs in constant time
* void on\_buttonBox\_rejected();
  + O(1)
  + Runs in constant time

Mainwindow class

The Mainwindow class is a very extensive class. This class encapsulates basically all of the main functionality of the program such as, outputting all the teams, filling/clearing tables within widgets, and administrator capabilities that deal with editing the database through the program. For the most part, most of these methods have a performance of O(n) or O(n^2). The methods that have the performance of O(n) are basically the ones that fill in the tables with data onto the program. The methods with O(n^2) are the ones that deal with dijkstra and souvenirs.

* void on\_comboBox\_TeamInfo\_currentIndexChanged(int index);
  + O(n)
  + Creates a table by looping through vector and displaying team information.
* void on\_loginButton\_clicked();
  + O(1)
  + Runs in constant time.
* void on\_AddLV\_Button\_clicked();
  + O(n)
  + Add las vegas to database and loops through to add souvenirs to the team.
* void on\_AdminTeamSouvCombo\_currentTextChanged(const QString &arg1);
  + O(n)
  + Creates a table to view teams to modify by looping through list.
* void on\_NewItemAddButton\_clicked();
  + O(1)
  + Adds an item to database in constant time.
* void on\_AdminDeleteSouv\_clicked();
  + O(1)
  + Deletes an item from database in constant time.
* void on\_AdminUpdateSouv\_clicked();
  + O(1)
  + Updates souvenir in constant time.
* void on\_AdminItemCombo\_currentTextChanged(const QString &arg1);
  + O(1)
  + fills the double combo box for price update to the price of the item selected in constant time
* void on\_updateStadNutton\_clicked();
  + O(1)
  + Updates stadium name and seating capacity in constant time.
* void on\_InfoTeamCombobox\_currentIndexChanged(const QString &arg1);
  + O(n)
  + loops through each name in vector and fills table based on choice in combo box
* void on\_ChangeTeamInfo\_Button\_clicked();
  + O(1)
  + Gives capability to change team info in constant time.
* void on\_NextStadium\_Button\_clicked();
  + O(n^2)
  + Displays souvenirs by using a loop within a loop
* void on\_VisitAll\_Button\_clicked();
  + O(n)
  + Calls on FillTripLabels which runs in O(n).
* void on\_TestButton\_clicked();
  + O(n)
  + Loops the list to test and display dikstra and MST output
* void on\_VisitSelected\_Button\_clicked();
  + O(n)
  + Loops through the teams list
* void on\_buttonBox\_rejected();
  + O(1)
  + Runs in constant time by changing the TripsWidget index
* void on\_buttonBox\_accepted();
  + O(n^2)
  + Loops through the dijkstra list and within that it loops again through he team numbers list
* void on\_BackButton\_clicked();
  + O(1)
  + Runs in constant time by changing the TripsWidget index
* void ClearTable(QTableWidget \*table);
  + O(n)
  + Loops through list to clear table
* void fillTeamComboBoxs(); //fill all team combo boxes up
  + O(n)
  + loops through the team list to add to combo box
* void fillAdminTeamTable();//fill team table in admin tab
  + O(n)
  + Loops through to fill up table with team information
* void fillGraph();
  + O(n)
  + Calls getTeamsAndStadiums that runs in O(n) time
* void FillTripLabels(QString name);
  + O(n)
  + inserts the teams souvenirs along with their prices tothe table by using loop.
* void refreshMST();
  + O(n)
  + uses loop to refresh MST

Map class

This class is an implementation of the map class. It has basic functionalities such as inserting, erasing, clearing list, checking if list is empty, and the size of the list. All, but one method runs in constant time (O(1)). The erase methods has a performance of O(n) because it has to search for the key to erase in the list.

* int size() const; //returns size of map
  + O(1)
  + Retrieves the size of the list in constant time
* bool empty() const; // returns if vector is empty or not
  + O(1)
  + retrieves bool to see if list is empty in constant time
* void insert(QString key, int value); //inserts key and value to map
  + O(1)
  + Inserts element into list in constant time
* void erase(QString key); //erases element with key passed in
  + O(n)
  + Searches key to delete in list by looping through the list
* vector<node> GetList(); //returns map list
  + O(1)
  + Returns list in constant time
* void clearList();
  + O(1)
  + Clears the list from its elements in constant time