|  |  |
| --- | --- |
| Design Document for Corporate Network | Abstract  This design document Contains details of implementation PS17-Corporate Network.  **Authors:**  **Vivek Biradar**  **Vikrant KUMAR Singh**  **PATIL Atul VILAS** |

**Introduction**

There are mainly two methods to implement graphical ADT which are adjacency matrix and adjacency list.

This design document has been created for problem statement 15 (Corporate network). As per problem solution was to be implemented to answer below questions.

1. List of unique companies and directors the agency has stored in the system.
2. List of companies associated with a director
3. List of directors associated with a company
4. Validate if two companies have a common director
5. Find if two companies can be connected through a network of directors and companies (you can assume that a director can be associated with max two companies)
6. If you start with a company and reach another company traversing through your graph, then they are connected.
7. Perform an analysis for the features above and give the running time in terms of input size: n.

We had to tackle this problem as graph-based problem and below solution has been provided according to that.

**Algorithm Selection**

There are mainly two methods to implement graphical ADT which are adjacency matrix and adjacency list. Selection of algorithm totally depends on problem and for our case adjacency list was preferable choice. We have opted for adjacency list method due to below reasons.

* **Adjacency lists are a compact way of representing only existing edges.**
* However, this comes at the cost of possibly slow lookup of specific edges. Since each list is as long as the degree of a vertex the worst-case lookup time of checking for a specific edge can become O(n), if the list is unordered. However, looking up the neighbours of a vertex becomes trivial, and for a sparse or small graph the cost of iterating through the adjacency lists might be negligible.
* **As we will be implementing BFS, it is always good to use adjacency list because nodes are in adjacent if we ae traversing**
* Memory usage depends on the number of edges (not number of nodes),  
  which might save a lot of memory if the adjacency matrix is sparse
* Finding the presence or absence of specific edge between any two nodes  
  is slightly slower than with the matrix O(k); where k is the number of neighbours nodes
* It is fast to iterate over all edges because you can access any node neighbours directly
* It is fast to add/delete a node; easier than the matrix representation
* It fast to add a new edge O(1)

**Implementation Architecture**

To establish relation between companies and directors we have followed below architecture. We have assumed all distinct companies and directors as nodes and related each node with their connected edge.

Maria Garcia

James Smith

ABCD CORP

SEAN MAXWELL

MARCUS CEASER

JUAN CARLOS

MARIA HERNANDEZ

MARIA GARCIA

HAHA LAUGH CORP

HAHA LAUGH CORP

HAHA LAUGH CORP

HAHA LAUGH CORP

HAHALAUGH CORP

ABCD Corp

HAHA LAUGH CORP

ABC Corp

SEAN MAXWELL

MARCUS CEASER

JUAN CARLOS

JAMES SMITH

MARIA GARCIA

MARIA HERNANDEZ

**Implementation Approach**

We have defined node Class with two attributes. One is name of attribute and second is type of attribute. Type are Company and Director. This has been done so that when we will have to filter companies and director then based on these attributes, we can filter those.

* First, we are reading the input file and based on that we are creating graph based on above architecture.
* Based on method given in problem statement we have implemented all to get answer to all questions.

**Time Complexity**

Below is the table where we have provided time complexity of all important methods which has been used in this implementation or the methods which has been used to answer question.

|  |  |
| --- | --- |
| **Method\_Name** | **Worst Time Complexity (Big O)** |
| readCompanyDirfile | O(n2) |
| displayAll | O(n) |
| displayCompanies | O(n2) |
| displayDirectors | O(n2) |
| findCommonDirector | O(n) |
| findRelatedCompany | O(n2) |