Algorithm Analysis

CSI 3344

Name: Maiqi Hou

1. void add_edges()

Description

This function is add edge between two vertices. Creating a directed graph.

Data Structure

I use the adjacency list idea to create a directed graph. Thus, for this function, I use the linked list to add edges.

Algorithm

```
void D_graph::add_edges(int mem1, int mem2) {
    adj_list[mem1].push_back(mem2);
}
```

Analysis

Input N	Mem1 and mem2
Basic Operation	adj_list[mem1].push_back(mem2);
Summation or Recurrence Relation	1

Worst Case Analysis

$T(n) = O(1)$

Best Case Analysis

$T(n) = \Omega(1)$

2. void explore()

Description

This function is find all vertices reachable from a particular vertex

Data Structure

In this function, I uses the list.

Algorithm

1. Check if the vertex has been visited.

2. Search all edges.

Analysis

Input N	Vertices(n)
Basic Operation	`list<int>::iterator neigbor;` `if (visited[n] == true)` `return;` `visited[n] = true;` `for (neigbor = adj_list[n].begin(); neigbor != adj_list[n].end(); neigbor++)` `if (!visited[*neigbor])` `explore(*neigbor, sta, visited, count, t);`
Summation or Recurrence Relation	$$\sum_{i=1}^{E} 1 = E$$ Where E = number of vertex edges(E)

Worst Case Analysis

T = O(E)

Best Case Analysis

T = Ω(E)

3. void dfs();

Description

This function's main purpose is to search all vertices of a directed graph.

After the graph reverse, also DFS again.

Data Structure

Using the bool array to check vertex.

Algorithm

1.Set all vertices of the graph that have not been visited.

2.Check if all vertices are visited. if a vertex does not visit, then exploring it.

Analysis

Input N	Number of vertices
Basic Operation	`for (int i = 0; i < vertex + 1; i++)` `visited[i] = false;` `for (int i = 1; i < vertex + 1; i++)` `if (visited[i] != true)` `Explore(i);`
Summation or Recurrence Relation	$\sum_{i=1}^{V} 1 = V$ Where v is number of vertices

Worst Case Analysis

T = O(V+E),where V = number of vertices and E is number of edges because the running time of exploring function is O(E)

Best Case Analysis

T = Ω(V),where V = number of vertices

4. rGraph()

Description

Reverse original graph direction

Data Structure

In this function, I use list to search and reverse original graph.

Algorithm

Vertex from 1 to V;

If V < number of vertices return

Else reverse all edges connect this vertex.

Increase one for V and call graph() again.

Analysis

Input N	Number of vertices(v)

Basic Operation	`if (v <= vertex)` `list<int>::iterator it;` `for (it = adj_list[v].begin(); it != adj_list[v].end(); it++) {` `rG.add_edges(*it, v);` `v++;` `rGraph(rG, v);`
Summation or Recurrence Relation	T = T(n+1) + O(n) Where V is number of vertices and E is number of edges

Worst Case Analysis

T = O(V+E), where V is number of vertices and E is number of edges

Best Case Analysis

T = Ω(V)

5. kosaraju()

Description

Computing strong components

Data Structure

Algorithm

DFS the directed graph

Revserse directed graph

DFS again reverse directed graph

Computing strong components

Analysis

Input N	
Basic Operation	`int D_graph::kosaraju() {` `int val = 1;                    //1` `stack<int> path;                //1` `dfs(val, path);                //O(V+E)` `D_graph reverse(vertex);        //1`

	`` rGraph(reverse, 1); //O(V+E) val = 0; //1 reverse.dfs(val, path); //(V+E) return val; } ```
Summation or Recurrence Relation	

Worst Case Analysis

T = O(V+E), becuase dfs function, rGraph function running time is O(v+E)

Best Case Analysis

T = Ω(V), because dfs function and rGraph function running time is Ω(V)