자료구조 과제(HW6)

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. Adjacency Matrix to Adjacency List

<Pseudo Code>

Variable: int** adj_matrix, int* visited, int num_of_vertex, node_pointer* graph

Function:

1. void dfs(FILE* fp, int v)

visited[v] = TRUE;

fprintf(fp, “%d”, v);

for(w=graph[v]; w; w=w->link)

    if(!visited[w->vertex])

        dfs(fp, w->vertex);

2. void connected(FILE* fp)

for i:=0 to num_of_vertex

    if(!visited[i])

        dfs(fp, i);

3. main function

open “input.txt”

get first line and it will be a ‘num_of_vertex’

graph = (node_pointer)malloc(sizeof (node_pointer) * num_of_vertex), each graph[i] = NULL

visited = (int*)malloc(sizeof (int) * num_of_vertex), each visited[i] = FALSE

dynamically allocate adj_matrix

get information from “input.txt” and save it into adj_matrix

close “input.txt”

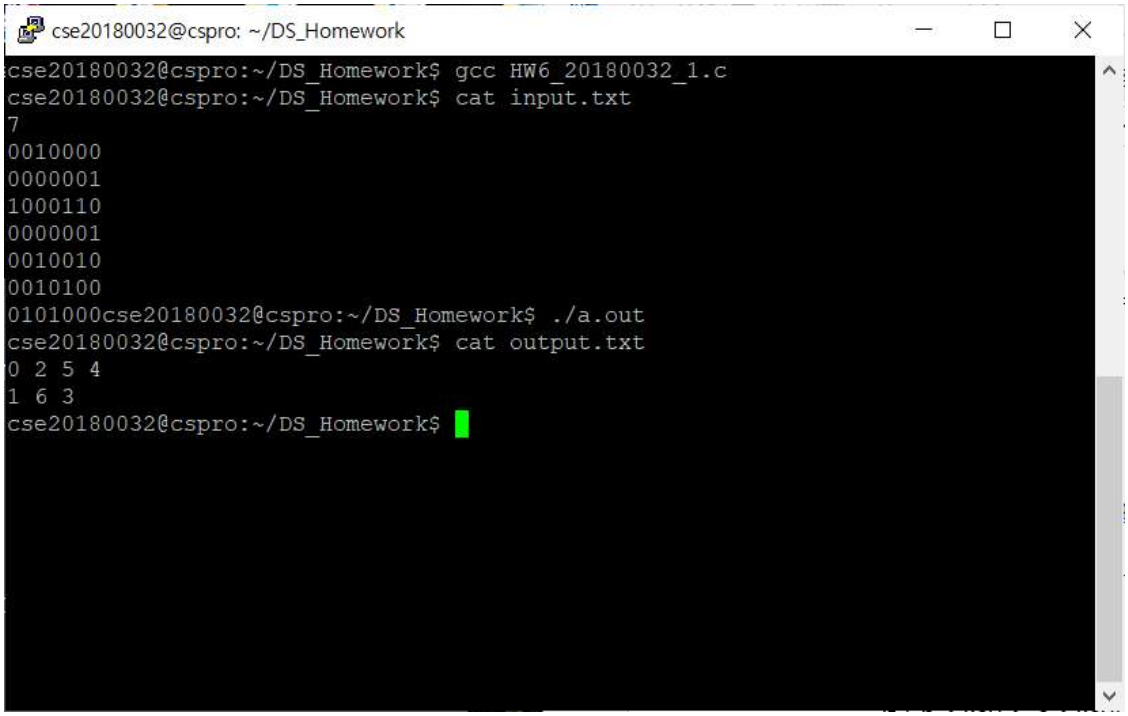
convert adj_matrix’s information into adjacency list

open “output.txt”

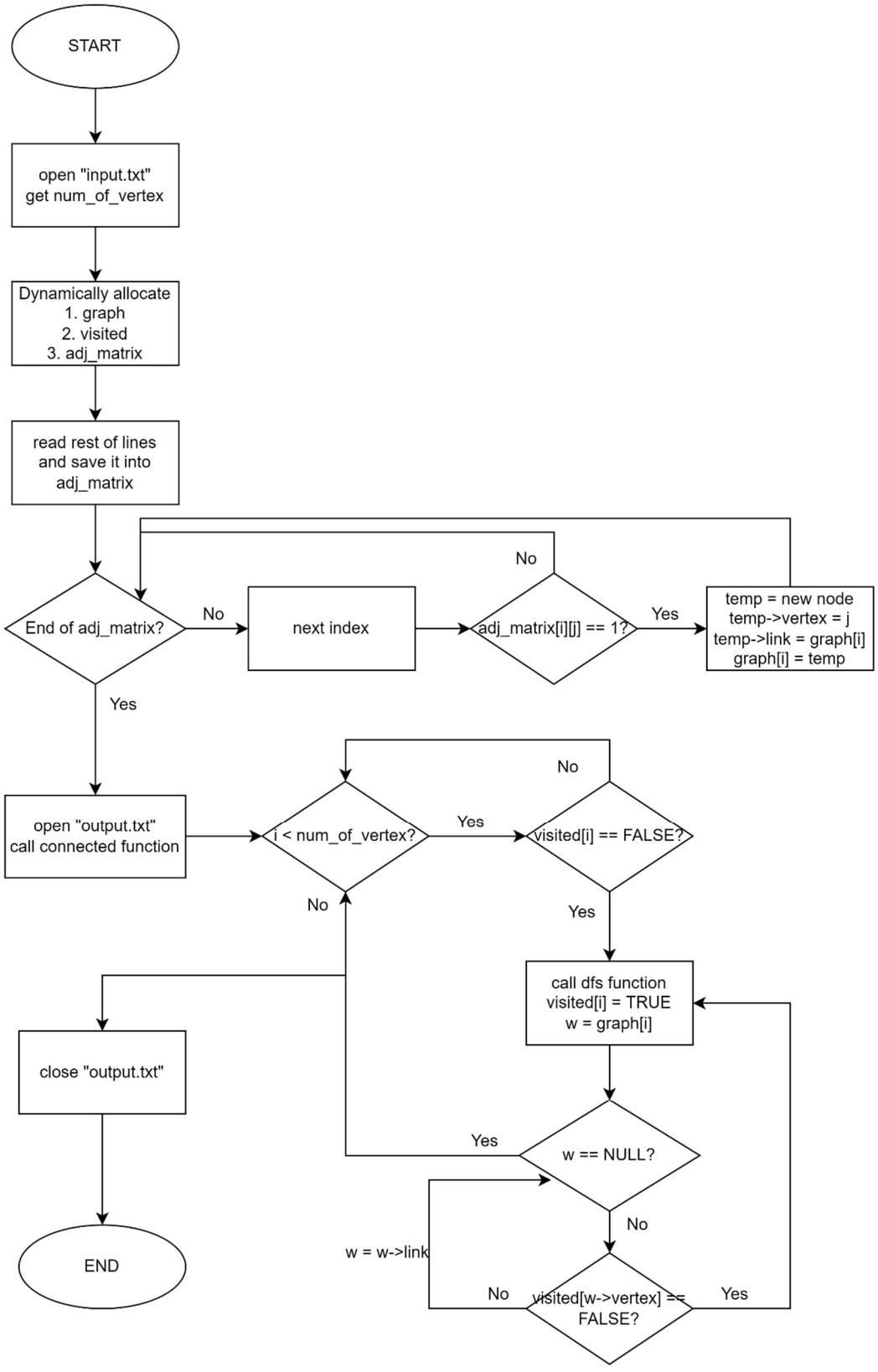
call connected function

close “output.txt”

<Test Examples>



<Flowchart>


START
open "input.txt"
get num_of_vertex
Dynamically allocate
1. graph
2. visited
3. adj_matrix
read rest of lines
and save it into
adj_matrix
End of adj_matrix?
No
next index
No
adj_matrix[i][j] == 1?
Yes
temp = new node
temp->vertex = j
temp->link = graph[i]
graph[i] = temp
Yes
open "output.txt"
call connected function
i < num_of_vertex?
Yes
No
visited[i] == FALSE?
No
Yes
call dfs function
visited[i] = TRUE
w = graph[i]
close "output.txt"
w == NULL?
Yes
No
w = w->link
visited[w->vertex] ==
FALSE?
No
Yes
END

2. MST by Kruskal's algorithm

<Pseudo Code>

Variable: count(number of edges), edge* edge_set(array that save edge's information), int** adj_matrix(save info from input.txt as adjacency matrix), int* visited(whether the vertex is visited or not), int num_of_vertex(number of vertex), node_pointer* graph(save info from input.txt as adjacency list), node_pointer* group(MST made by Kruskal algorithm), int* parent(notice whether edges are in same set)

Function: set_union, sort_edge, dfs, Kruskal, main

1. void set_union (int i, int j)

if(parent[i] < parent[j])

    change values into parent[i] if they are same as parent[j]

else

    change values into parent[j] if they are same as parent[i]

2. void sort_edge( )

for i=0 to count -1

    min = edge_set[i].weigh, index = i

    for j = i+1 to count

        if (edge_set[j].weight < min) min = edge_set[j].weight, index = j

    if (index != i) //something changes

    swap edge_set[i] with edge_set[index](which is minimum)

3. void dfs(FILE* fp, int v)

visited[v] = TRUE;

fprintf(fp, “%d”, v);

for(w=graph[v]; w; w=w->link)

    if(!visited[w->vertex])

        dfs(fp, w->vertex);

4. int Kruskal( )

for i=0 to count

    if (edge_set[i].initial and edge_set[i].terminal are in different set) continue;

    set_union(edge_set[i].initial, edge_set[i].terminal)

    temp = new node, temp->vertex = edge_set[i].terminal,

    link to group[edge_set[i].initial]

    temp = new node, temp->vertex = edge_set[i].initial

    link to group[edge_set[i].terminal]

    cost += edge_set[i].weight //accumulate tree’s cost

return cost

5. main function

open “input.txt” and get first line which is a ‘num_of_vertex’

graph = (node_pointer)malloc(sizeof (node_pointer) * num_of_vertex), each graph[i] = NULL

group = (node_pointer)malloc(sizeof (node_pointer) * num_of_vertex), each group[i]= NULL

visited = (int*)malloc(sizeof (int) * num_of_vertex), each visited[i] = FALSE

dynamically allocate adj_matrix and edge_set

parent = (int*)malloc(sizeof(int)*num_of_vertex), each parent[i] = i

get information from “input.txt” and save it into adj_matrix

close “input.txt”

convert adj_matrix’s information into adjacency list and make edge set

call sort_edge function to sorting edges

call Kruskal fuction and get the total cost of MST

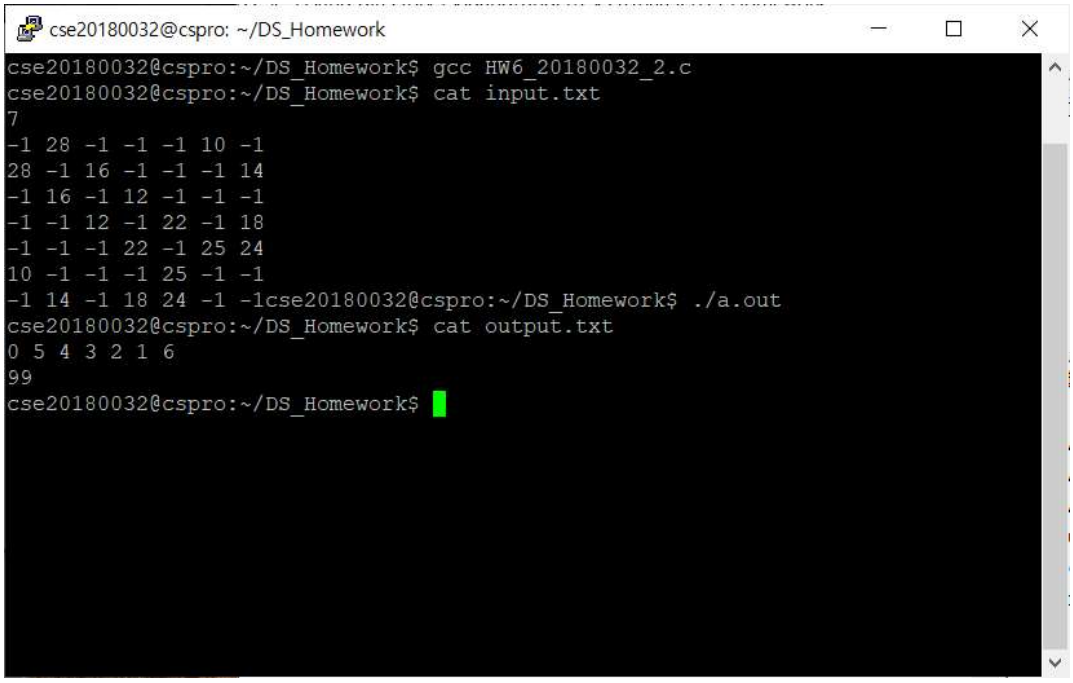
open “output.txt”

call dfs function to traverse MST made by Kruskal algorithm

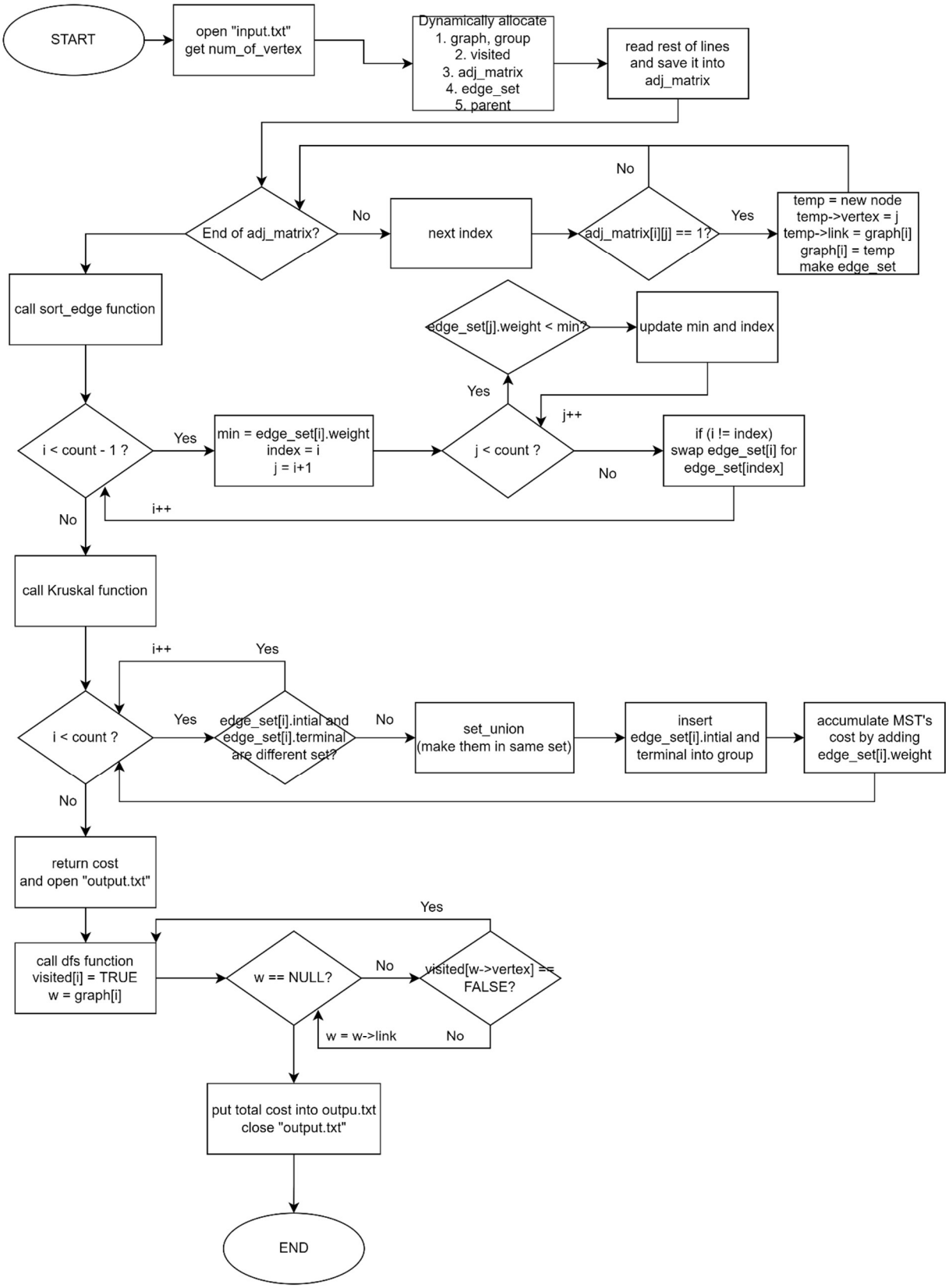
put total cost of MST into output.txt

close “output.txt”

<Test Examples>



<Flowchart>


START
open "input.txt"
get num_of_vertex
Dynamically allocate
1. graph, group
2. visited
3. adj_matrix
4. edge_set
5. parent
read rest of lines
and save it into
adj_matrix
End of adj_matrix?
No
next index
adj_matrix[i][j] == 1?
Yes
No
temp = new node
temp->vertex = j
temp->link = graph[i]
graph[i] = temp
make edge_set
call sort_edge function
edge_set[j].weight < min?
update min and index
Yes
j++
i < count - 1 ?
Yes
min = edge_set[i].weight
index = i
j = i+1
j < count ?
No
if (i != index)
swap edge_set[i] for
edge_set[index]
No
i++
call Kruskal function
i++
Yes
i < count ?
Yes
edge_set[i].intial and
edge_set[i].terminal
are different set?
No
set_union
(make them in same set)
insert
edge_set[i].intial and
terminal into group
accumulate MST's
cost by adding
edge_set[i].weight
No
return cost
and open "output.txt"
Yes
call dfs function
visited[i] = TRUE
w = graph[i]
w == NULL?
No
visited[w->vertex] ==
FALSE?
w = w->link
No
put total cost into outpu.txt
close "output.txt"
END

3. MST by Prim's algorithm

<Pseudo Code>

Variable: count(number of connected group), int** adj_matrix(save info from input.txt as adjacency matrix), int* visited(whether the vertex is visited or not), int num_of_vertex(number of vertex), node_pointer* graph(save info from input.txt as adjacency list), node_pointer* group(MST made by Prim algorithm)

Function: Prim, main function

1. Prim

```
for i=0 to num_of_vertex
        if(!visited[i])
                visited[i] = TRUE, root = new node, root->vertex = i, group[count] = root
                ptr=graph[i], index = i
                while(1)
                        find minimum cost edge
                        move to the edge whose cost is minimum
                        if ( all components in connected set are traversed )
                                root-> link = NULL, break;
                        index = ptr->vertex; temp = new node,
                        temp->vertex = index, temp->weight=min, insert temp to root
                        visited[index] = TRUE
                        ptr = graph[index] // move to next node
        count++
```

2. main function

open “input.txt” and get first line which is a ‘num_of_vertex’

graph = (node_pointer)malloc(sizeof (node_pointer) * num_of_vertex), each graph[i] = NULL

group = (node_pointer)malloc(sizeof (node_pointer) * num_of_vertex), each group[i]= NULL

visited = (int*)malloc(sizeof (int) * num_of_vertex), each visited[i] = FALSE

dynamically allocate adj_matrix

get information from “input.txt” and save it into adj_matrix

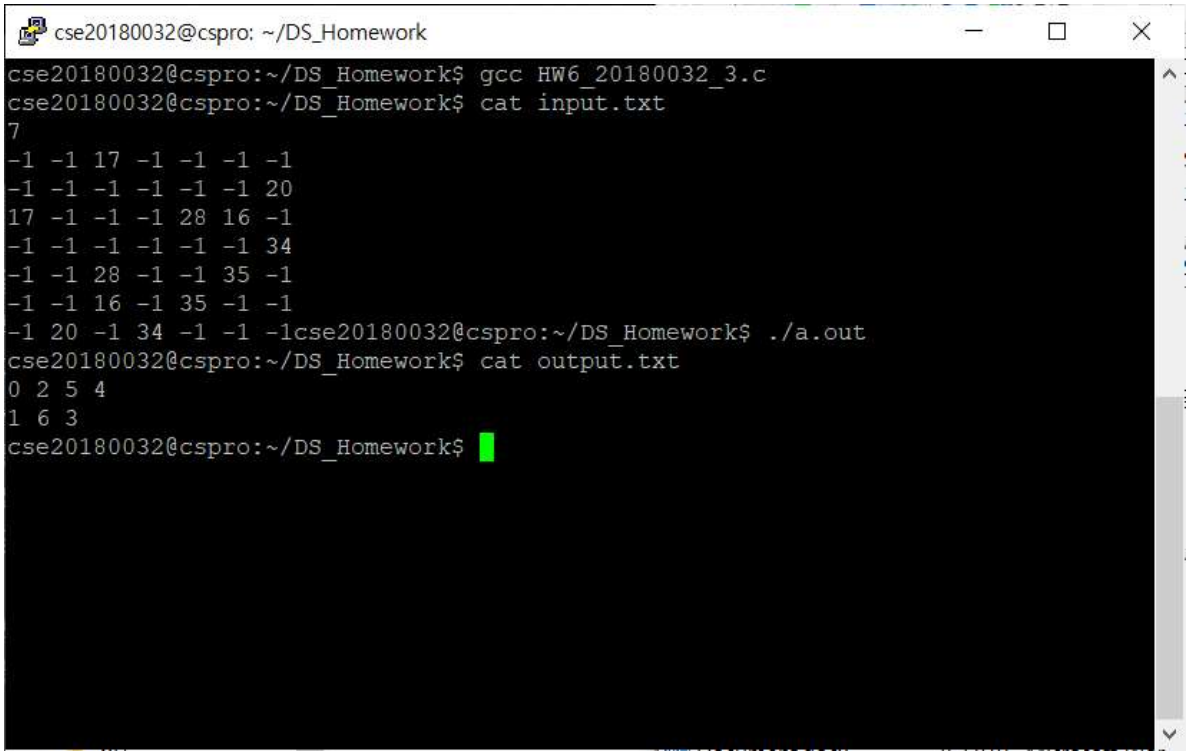
close “input.txt”

convert adj_matrix’s information into adjacency list

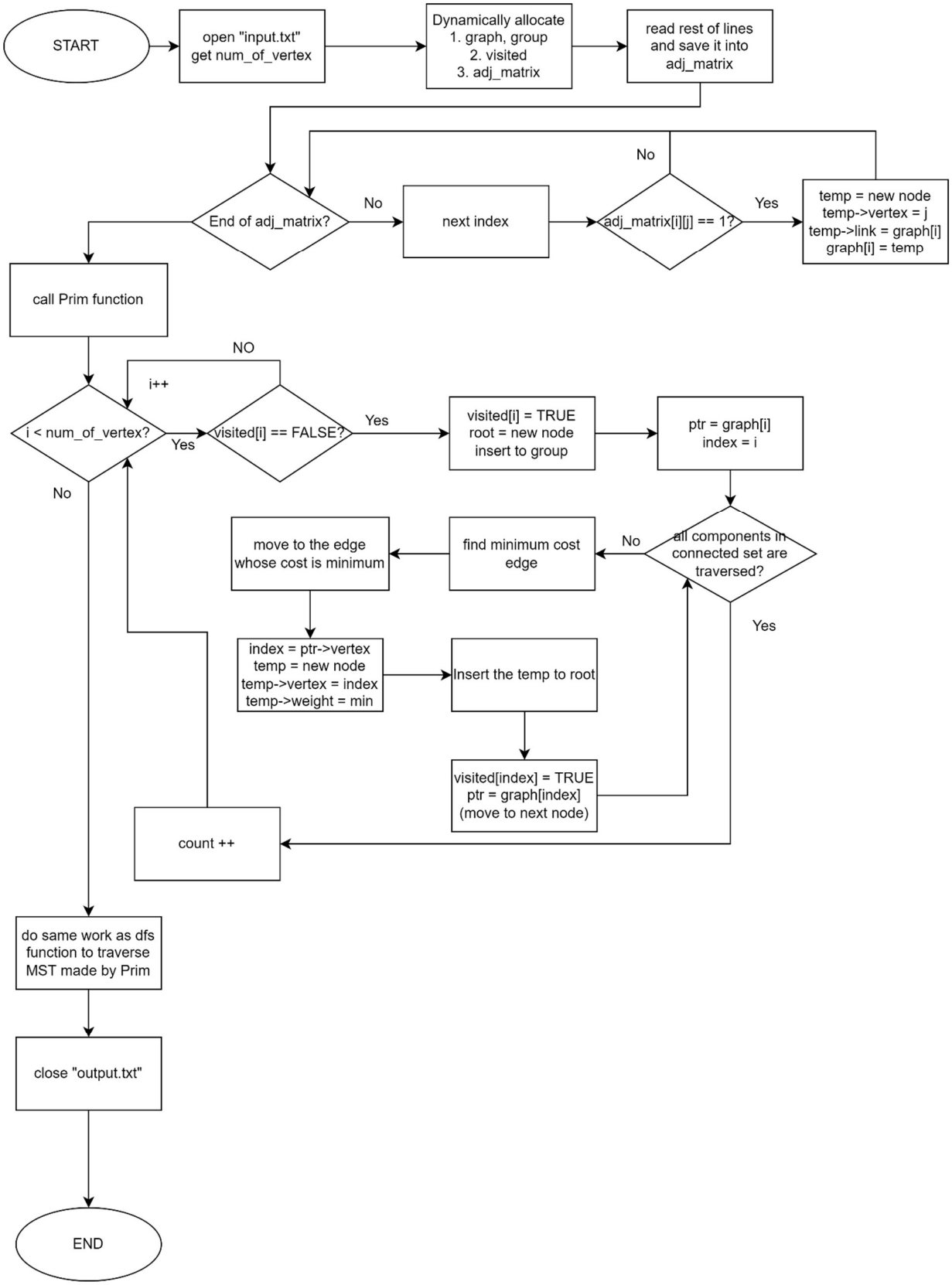
call Prim function and open “output.txt”

call dfs function to traverse MST made by Prim algorithm and close “output.txt”

<Test Examples>



<Flowchart>


START
open "input.txt"
get num_of_vertex
Dynamically allocate
1. graph, group
2. visited
3. adj_matrix
read rest of lines
and save it into
adj_matrix
End of adj_matrix?
No
next index
No
adj_matrix[i][j] == 1?
Yes
temp = new node
temp->vertex = j
temp->link = graph[i]
graph[i] = temp
call Prim function
NO
i++
i < num_of_vertex?
Yes
visited[i] == FALSE?
Yes
visited[i] = TRUE
root = new node
insert to group
ptr = graph[i]
index = i
No
all components in
connected set are
traversed?
No
find minimum cost
edge
move to the edge
whose cost is minimum
Yes
index = ptr->vertex
temp = new node
temp->vertex = index
temp->weight = min
Insert the temp to root
visited[index] = TRUE
ptr = graph[index]
(move to next node)
count ++
do same work as dfs
function to traverse
MST made by Prim
close "output.txt"
END