ADA ASSIGNMENT:
IMPLEMENT ANY 5 PROBLEMS

HARSH SHETH, 185560
SHUBHAM JAIN, 185550
BHANU RANA, 185559

November 30, 2020

1 PERSONAL DETAILS

1.1 GROUP NO: 17

- **HARSH SHETH**
  - **Email: ycdtharsh@gmail.com**
  - **Roll No: 185560**
  - **Contact: +91 9407394270**
- **SHUBHAM JAIN**
  - **Email: sjn574472@gmail.com**
  - **Roll No: 185550**
  - **Contact: +91 8580410883**
- **BHANU RANA**
  - **Email: bhanupratap1312@gmail.com**
  - **Roll No: 185559**
  - **Contact: +91 8529580897**

2 CODE AND SNAPSHOTS

2.1 DFS and BFS

Listing 1: dfs_bfs.cpp file

```
1  #include <iostream>
2  #include <bits/stdc++.h>
3  #include "queue.h"
4  #define MAXNODE 100
5  #define UNDISCOVERED -1
6  #define DISCOVERED    0
7  #define PROCESSED     1
8  #define NOPARENT     -1
9  using namespace std;
10
11 int *state;
12 int *parent;
13 int *entry_time;
14 int *exit_time;
15 int dfs_time;
16
17 struct edgenode{
18   int adj;
19   int weight;
20   edgenode *next;
21 };
22
23 struct graph{
24   // Adjacency List(Array of linked lists)
25   edgenode *edges[MAXNODE+1];
26   // OUTDEGREE OF EACH NODE
27   int outdegree[MAXNODE+1];
28   int nvertices;
```

```
29    int nedges;
30    bool directed;
31  };
32
33  void initialize_graph(graph *g, bool directed){
34    // INITIALLY THERE ARE NO VERTICES
35    g->nvertices = 0;
36    // THEREFORE NO EDGES ALSO
37    g->nedges = 0;
38    g->directed = directed;
39    for(int i = 0; i < MAXNODE+1; i++){
40      // ADJACENCY LIST WILL BE NULL INITIALLY
41      g->edges[i] = NULL;
42      // OUTDEGREE WILL BE ZERO AS NO NODES ARE PRESENT
43      g->outdegree[i] = 0;
44    }
45    return;
46  }
47
48  void insert_edge(graph *g, int x, int y, bool directed, int weight=0){
49    // TEMPERORY EDGENODE
50    edgenode *e = new edgenode;
51    e->weight = weight;
52    e->adj = y;
53    // INSERT IN THE FRONT OF THE EDGE LIST
54    e->next = g->edges[x];
55    g->edges[x] = e;
56
57    if(!directed)
58      insert_edge(g, y, x, true, weight);
59    else
60      g->nedges++;
61    return;
62  }
```

```
63
64 void build_graph(graph *g, bool directed){
65   int n, m;
66   char x, y;
67   initialize_graph(g, directed);
68   cout << "Enter no. of vertices and edges: ";
69   cin >> n >> m;
70   if(n == 0) { exit(0); }
71   // TOTAL NO OF NODES IN GRAPH
72   g->nvertices = n;
73   for(int i = 0; i < m; i++){
74     cout << "\nEnter edge to be inserted: \n source vertex: ";
75     cin >> x;
76     cout << "Destination vertex: ";
77     cin >> y;
78     insert_edge(g, ctoi(x) , ctoi(y), directed);
79   }
80   return;
81 }
82
83 void print_graph(graph *g){
84   prettify();
85   edgenode *ptr = new edgenode;
86   for(int i = 0; i < g->nvertices; i++){
87     cout << "\nEdgelist of node "; putchar(i + 'A'); cout << ": ";
88     ptr = g->edges[i];
89     while(ptr != NULL){
90       cout << "{"; putchar(ptr->adj + 'A'); cout << "}, ";
91       ptr = ptr->next;
92     }
93   }
94   prettify();
95   return;
96 }
```

```
97
98  // BFS of graph
99  void bfs(graph *g, int source){
100   // DISCOVERED
101   state[source] = DISCOVERED;
102   Enqueue(source);
103   int vertex;
104   while((vertex = Dequeue()) != -1){
105     cout << "\nCurrent Vertex is: " << itoc(vertex);
106     edgenode *ptr = g->edges[vertex];
107     while(ptr != NULL){
108       if(state[ptr->adj] == UNDISCOVERED){
109         cout << "\n-> \tcurrent edges is: {"
110           << itoc(vertex) << ", " << itoc(ptr->adj) << "}";
111         state[ptr->adj] = DISCOVERED;
112         parent[ptr->adj] = vertex;
113         Enqueue(ptr->adj);
114       }
115       ptr = ptr->next;
116     }
117     state[vertex] = PROCESSED;
118   }
119   prettify();
120   for(int i = 0; i < g->nvertices; i++){
121     cout << "\nParent of vertex: " << itoc(i) << "is: " << itoc(parent
122   }
123   return;
124 }
125
126 void process_edge(int x, int y){
127   cout << "\n\t->Current edge is: " << itoc(x) << ", " << itoc(y);
128   return;
129 }
130
```

```
131 void initialize_arrays(graph *g){
132   for(int i = 0; i < g->nvertices; i++){
133     state[i] = UNDISCOVERED;
134     parent[i] = -1; // No Parent
135     entry_time[i] = exit_time[i] = 0;
136   }
137   dfs_time = 0;
138 }
139
140 // DFS of graph
141 void dfs(graph *g, int source){
142   int vertex = source;
143   // true if node has undiscovered neighbours
144   bool undiscovered_adj = false;
145   stack<int> dfs_stack;
146   state[source] = DISCOVERED;
147   dfs_stack.push(source);
148   edgenode *ptr = new edgenode;
149   ptr = g->edges[source];
150   entry_time[source] = ++dfs_time;
151   cout << "\nCurrent Vertex is: " << itoc(vertex)
152     << " with entry time: " << entry_time[vertex];
153   do{
154     while(ptr != NULL){
155       if(state[ptr->adj] == UNDISCOVERED){
156         process_edge(vertex, ptr->adj);
157         state[ptr->adj] = DISCOVERED;
158         entry_time[ptr->adj] = ++dfs_time;
159         parent[ptr->adj] = vertex;
160         dfs_stack.push(ptr->adj);
161         vertex = ptr->adj;
162         cout << "\nCurrent Vertex is: " << itoc(vertex)
163           << " with entry time: " << entry_time[vertex];
164         ptr = g->edges[ptr->adj];
```

```
165         }else{
166           ptr = ptr->next;
167         }
168       }
169       ptr = g->edges[vertex = dfs_stack.top()];
170       if(exit_time[vertex] == 0){
171         edgenode *newptr = g->edges[vertex];
172         while(newptr != NULL){
173           if(state[newptr->adj] == UNDISCOVERED){
174             undiscovered_adj = true;
175             break;
176           }
177           newptr = newptr->next;
178         }
179         if(!undiscovered_adj){
180           exit_time[vertex] = ++dfs_time;
181           state[vertex] = PROCESSED;
182           cout << "\n exit time of vertex: "
183             << itoc(vertex)<< "is: " << exit_time[vertex];
184         }
185       }
186       if(!undiscovered_adj)
187         dfs_stack.pop();
188       undiscovered_adj = false;
189     }while(!dfs_stack.empty());
190     return;
191 }
192
193 int main(){
194     graph *new_graph = new graph;
195     char ch;
196     int source;
197     cout << "Press 'y' or 'n'\n";
198     cout << "Is graph directed: ";
```

```
199     cin >> ch;
200     build_graph(new_graph, (ch == 'y') ? true : false);
201     print_graph(new_graph);
202     cout << "\n Enter the source node: ";
203     cin >> ch;
204     source = ctoi(ch);
205     if(source >= new_graph->nvertices || source < 0){
206       cout << "No such node found";
207       return -1;
208     }
209     state = new int[new_graph->nvertices];
210     parent = new int[new_graph->nvertices];
211     entry_time = new int[new_graph->nvertices];
212     exit_time = new int[new_graph->nvertices];
213     // BFS
214     initialize_arrays(new_graph);
215     bfs(new_graph, source);
216     for(int i = 0; i < new_graph->nvertices; i++){
217       if(state[i] == UNDISCOVERED){
218         bfs(new_graph, i);
219       }
220     }
221     // DFS
222     initialize_arrays(new_graph);
223     dfs(new_graph, source);
224     for(int i = 0; i < new_graph->nvertices; i++){
225       if(state[i] == UNDISCOVERED){
226         dfs(new_graph, i);
227       }
228     }
229     return 0;
230 }
```

Listing 2: queue.h file

```
1  #include <iostream>
2  using namespace std;
3
4  char itoc(int num){
5    return char(num)+'A';
6  }
7
8  int ctoi(char ch){
9    if(isalpha(ch)){
10     return toupper(ch)-'A';
11   }
12   return ch-'0';
13 }
14
15 struct Queue{
16   int info;
17   Queue *link;
18 };
19
20 void prettify(){
21   cout << "\n**********************\n";
22   return;
23 }
24
25 Queue *FRONT = NULL;
26 Queue *REAR = NULL;
27 void Enqueue(int vertex){
28   Queue *ptr = new Queue;
29   ptr->link = NULL;
30   ptr->info = vertex;
31   if(FRONT == NULL){
32     FRONT = ptr;
33     REAR = ptr;
34   }else{
```

```
35      REAR->link = ptr;
36      REAR = ptr;
37    }
38    return;
39  }
40
41  int Dequeue(){
42    if(FRONT == NULL) {
43      return -1;
44    }else{
45      int vertex;
46      Queue *temp = new Queue;
47      vertex = FRONT->info;
48      temp = FRONT->link;
49      FRONT->link = NULL;
50      FRONT = temp;
51      return vertex;
52    }
53  }
```

2.2 HEAPSORT

Listing 3: heapsort.cpp file

```
#include <iostream>
#define tab '\t'
using namespace std;

void print_array(int *arr, int n)
{
        for(int i = 0; i < n; ++i)
                cout << arr[i] << tab;
        cout << endl;
}

void swap(int &a, int &b)
{
        int temp = a;
        a = b;
        b = temp;
}

void adjust(int *tree, int n, int ptr)
{
        int left = ptr * 2 + 1, right = ptr * 2 + 2;
        int item = tree[ptr];
        while(right < n)
        {
                if(item >= tree[left] && item >= tree[right])
                {
                        tree[ptr] = item;
                        return;
                }
                else if(tree[left] > tree[right])
                {
```

```
32                          tree[ptr] = tree[left];
33                          ptr = left;
34                  }
35                  else
36                  {
37                          tree[ptr] = tree[right];
38                          ptr = right;
39                  }
40                  left = ptr*2 + 1;
41                  right = ptr*2 + 2;
42          }
43          if(left == n-1 && tree[left] > item)
44          {
45                  tree[ptr] = tree[left];
46                  ptr = left;
47          }
48          tree[ptr] = item;
49  }
50
51  void heap_sort(int *arr, int n)
52  {
53          for(int i = 0; i < n; ++i)
54                  adjust(arr, n, n-i-1);
55          for(int i = 0; i < n; ++i)
56          {
57                  swap(arr[0], arr[n - i - 1]);
58                  adjust(arr, n - i - 1, 0);
59          }
60  }
61
62  int main()
63  {
64          cout << "Enter the length of the array" << endl;
65          int n;
```

```
66          cin >> n;
67          int *arr = new int[n];
68          for(int i = 0; i < n; ++i)
69                  cin >> arr[i];
70
71          heap_sort(arr, n);
72          print_array(arr, n);
73          return 0;
74  }
```

2.3 KNAPSACK

Listing 4: knapsack.cpp file

```
#include <iostream>
#define tab '\t'
using namespace std;

struct object
{
        int profit;
        int weight;
        double ratio;
        object(int p = 0, int w = 0, int r = 0)
                : profit(p), weight(w), ratio(r) {}
};

class knapsack
{
        int total_weight;
        double total_profit;
        object * arr;
        int n;
        void scan_objects();
        void sort();
        public:
        knapsack();
        void maximize();
        void display();
        ~knapsack() {delete [] arr;}
};

knapsack :: knapsack()
{
        total_profit = 0;
```

```
32          cout << "Enter the weight of the bag" << endl;
33          cin >> total_weight;
34          cout << "Enter the no. of total Objects" << endl;
35          cin >> n;
36          arr = new object[n];
37          scan_objects();
38  }
39
40  void knapsack :: scan_objects()
41  {
42          for(int i = 0; i < n; ++i)
43          {
44                  cout << "Enter the weight of object:" << tab;
45                  cin >> arr[i].weight;
46                  cout << "Enter the profit of object:" << tab;
47                  cin >> arr[i].profit;
48                  arr[i].ratio = (double) arr[i].profit / arr[i].weight;
49          }
50  }
51
52  void swap(object &a, object &b)
53  {
54          object temp = a;
55          a = b;
56          b = temp;
57  }
58
59  void adjust(object *tree, int n, int ptr)
60  {
61          int left = ptr * 2 + 1, right = ptr * 2 + 2;
62          object item = tree[ptr];
63          while(right < n)
64          {
65                  if(item.ratio >= tree[left].ratio
```

```
66                                      && item.ratio >= tree[right].ratio)
67                      {
68                              tree[ptr] = item;
69                              return;
70                      }
71                      else if(tree[left].ratio
72                                      > tree[right].ratio)
73                      {
74                              tree[ptr] = tree[left];
75                              ptr = left;
76                      }
77                      else
78                      {
79                              tree[ptr] = tree[right];
80                              ptr = right;
81                      }
82                      left = ptr*2 + 1;
83                      right = ptr*2 + 2;
84              }
85              if(left == n-1 && tree[left].ratio
86                              > item.ratio)
87              {
88                      tree[ptr] = tree[left];
89                      ptr = left;
90              }
91              tree[ptr] = item;
92      }
93
94      void knapsack :: sort()
95      {
96              for(int i = 0; i < n; ++i)
97                      adjust(arr, n, n-i-1);
98              for(int i = 0; i < n; ++i)
99              {
```

```
100                     swap(arr[0], arr[n - i - 1]);
101                     adjust(arr, n - i - 1, 0);
102             }
103     }
104
105
106     void knapsack :: maximize()
107     {
108             sort();
109             double sum = 0, diff = 0, w = 0;
110             double *x = new double[n];
111             for(int i = n - 1; i >= 0; --i)
112             {
113                     diff = total_weight - sum;
114                     w = arr[i].weight;
115                     if(diff >= w)
116                             x[i] = 1;
117                     else
118                             x[i] = diff / w;
119                     total_profit += x[i] * arr[i].profit;
120                     sum += x[i] * w;
121             }
122             display();
123             delete [] x;
124     }
125
126     void knapsack :: display()
127     {
128             for(int i = 0; i < n; ++i)
129             {
130                     cout << "Profit : " << arr[i].profit << tab
131                             << "Weight : " << arr[i].weight << tab
132                             << "Ratio : " << arr[i].ratio << endl;
133             }
```

```
134             cout << endl << "Maximum profit : "
135                     << total_profit << endl;
136 }
137
138 int main()
139 {
140             knapsack a;
141             a.maximize();
142             return 0;
143 }
```

2.4 N-QUEEN

Listing 5: nqueen.cpp file

```
#include<iostream>
using namespace std;
int grid[10][10];

void print(int n) {
  for (int i = 0;i <= n-1; i++) {
    for (int j = 0;j <= n-1; j++) {

      cout <<grid[i][j]<< " ";
    }
    cout<<endl;
  }
  cout<<endl;
  cout<<endl;
}

bool isSafe(int col, int row, int n) {
  //check for same column
  for (int i = 0; i < row; i++) {
    if (grid[i][col]) {
      return false;
    }
  }
  //check for upper left diagonal
  for (int i = row,j = col;i >= 0 && j >= 0; i--,j--) {
    if (grid[i][j]) {
      return false;
    }
  }
  //check for upper right diagonal
  for (int i = row, j = col; i >= 0 && j < n; j++, i--) {
```

```
32      if (grid[i][j]) {
33        return false;
34      }
35    }
36    return true;
37  }
38
39  bool solve (int n, int row) {
40    if (n == row) {
41      print(n);
42      return true;
43    }
44    //variable res is use for possible backtracking
45    bool res = false;
46    for (int i = 0;i <=n-1;i++) {
47      if (isSafe(i, row, n)) {
48        grid[row][i] = 1;
49        //recursive call solve(n, row+1) for next queen (row+1)
50        //if res ==false then backtracking will occur
51        res = solve(n, row+1) || res;
52        //by assigning the grid[row][i] = 0
53
54        grid[row][i] = 0;
55      }
56    }
57    return res;
58  }
59
60  int main()
61  {
62    ios_base::sync_with_stdio(false);
63    cin.tie(NULL);
64    int n;
65    cout<<"Enter the number of queen"<<endl;
```

```
66    cin >> n;
67    for (int i = 0;i < n;i++) {
68      for (int j = 0;j < n;j++) {
69        grid[i][j] = 0;
70      }
71    }
72    bool res = solve(n, 0);
73    if(res == false) {
74      //if there is no possible solution
75      cout << -1 << endl;
76    } else {
77      cout << endl;
78    }
79    return 0;
80  }
```