

Contents

1	Basic
1.1	Map
2	DataStructure
2.1	DisjointSet
2.2	SegmentTree
2.3	AdjList
2.4	AdjMatrix
3	Graph
3.1	Dijkstra
3.2	DFS
3.3	BFS
4	Math
4.1	PrimeTable
4.2	TopologySort

1 Basic

1.1 Map

```

1 #include <map>
2 int main(){
3
4     //declaration container and iterator
5     map<string, string> mapStudent;
6     map<string, string>::iterator iter;
7     map<string, string>::reverse_iterator iter_r;
8
9     //insert element
10    mapStudent.insert(pair<string, string>("r000",
11        "student_zero"));
12
13    mapStudent["r123"] = "student_first";
14    mapStudent["r456"] = "student_second";
15
16    //traversal
17    for(iter = mapStudent.begin(); iter !=
18        mapStudent.end(); iter++)
19        cout<<iter->first<<"
20            "<<iter->second<<endl;
21    for(iter_r = mapStudent.rbegin(); iter_r !=
22        mapStudent.rend(); iter_r++)
23        cout<<iter_r->first<<"
24            "<<iter_r->second<<endl;
25
26    //find and erase the element
27    iter = mapStudent.find("r123");
28    mapStudent.erase(iter);
29
30    iter = mapStudent.find("r123");
31
32    if(iter != mapStudent.end())
33        cout<<"Find, the value is
34            "<<iter->second<<endl;
35    else
36        cout<<"Do not Find"<<endl;
37
38    return 0;
39 }

```

2 DataStructure

2.1 DisjointSet

```

1 #define SIZE 10000
2
3 int arr[SIZE];
4
5 void init(int n) // give a initial length

```

```

6 {
7     for(int i=0; i<n; i++)
8         arr[i] = -1;
9 }
10
11 int find(int x)
12 { // find the father point
13     return arr[x] < 0 ? x : (arr[x] = find(arr[x])); //
14         update every child to the other father
15 }
16
17 void Union(int x, int y)
18 {
19     x = find(x);
20     y = find(y);
21
22     if(x == y)
23         return;
24
25     if(arr[x] <= arr[y])
26     {
27         arr[x] += arr[y];
28         arr[y] = x;
29     }
30     else
31     {
32         arr[y] += arr[x];
33         arr[x] = y;
34     }
35 }

```

2.2 SegmentTree

```

1 #define SIZE 100000
2
3 int st[SIZE];
4 int st_val[SIZE];
5
6 void st_build(int *st, int *st_val, int now, int ls,
7     int rs)
8 {
9     if(ls == rs)
10         st[now] = st_val[ls];
11     else
12     {
13         st_build(st, st_val, now*2, ls, (ls+rs)/2);
14         st_build(st, st_val, now*2+1, (ls+rs)/2+1, rs);
15         st[now] = max(st[now*2], st[now*2+1]);
16     }
17
18     // ls and rs are query range, begin and end is whole
19     // st[] range
20     int query(int now, int ls, int rs, int begin, int end)
21     {
22         int mid = (begin+end)/2;
23         int ret = 0;
24
25         if(ls <= begin && rs >= end)
26             return st[now];
27
28         // it is find max now (modify here)
29         if(ls <= mid)
30             ret = max(ret, query(now*2, ls, rs, begin, mid));
31
32         if(rs > mid)
33             ret = max(ret, query(now*2+1, ls, rs, mid+1,
34                 end));
35
36         return ret;
37     }
38 }

```

2.3 AdjList

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define MAXN 100
5
6 //the size of matrix
7 int n;
8 //the number of edges
9 int m;
10 //declare the List with MAXN
11 vector<pair<int, int>> Adj[ MAXN ];
12
13 void inputList() {
14     //weight
15     int a, b, w;
16
17     for (int i = 0; i < m; i++) {
18         //input and store
19         cin >> a >> b >> w;
20         Adj[ a ].push_back(make_pair(w, b));
21         //雙向圖
22         Adj[ b ].push_back(make_pair(w, a));
23     }
24 }
25
26 void printList() {
27     printf("The adjacency List:\n");
28     for (int i = 0; i < n; i++) {
29         printf("[%d] --> ", i);
30         for (auto p : Adj[ i ]) {
31             printf("(%d, %d) ---> ", p.first,
32                 p.second);
33         }
34         puts("NULL");
35     }
36 }
37
38 int main() {
39     cin >> n >> m;
40     inputList();
41     printList();
42 }

```

2.4 AdjMatrix

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define MAXN 100
5
6 //the size of matrix
7 int n;
8 //the number of edges
9 int m;
10 //declare the Matrix with MAXN
11 int Adj[ MAXN ][ MAXN ];
12
13 void inputMatrix() {
14     int a, b;
15
16     for (int i = 0; i < m; i++) {
17         //input and store
18         cin >> a >> b;
19         Adj[ a ][ b ] = 1;
20     }
21 }
22
23 void printMatrix() {
24     printf("The adjacency matrix:\n");
25     for (int i = 0; i < n; i++) {
26         printf("/");
27         for (int j = 0; j < n; j++) {
28             printf("%3d ", Adj[ i ][ j ]);
29         }
30     }
31 }

```

```

30     puts("/");
31 }
32 }
33
34 int main() {
35     //initialize the matrix
36     memset(Adj, 0, sizeof(Adj)); //初始化
37     cin >> n >> m;
38     inputMatrix();
39     printMatrix();
40 }

```

3 Graph

3.1 Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define MP make_pair
5 #define PII pair<int, int>
6 #define maxn 50000+5
7
8 int dis[maxn]; // 預設都是 INF
9 vector<PII> e[maxn]; // (連到的點, 邊的距離)
10
11 void dijk(int cur) // dijk(起點)
12 {
13     int d;
14     priority_queue<PII, vector<PII>, greater<PII>> q; //
15     // 放 (距離, 點編號), 每次會拿距離最小的點出來
16     q.push( MP(0, cur) );
17
18     while (!q.empty())
19     {
20         tie(d, cur) = q.top();
21         q.pop();
22         if (dis[cur] != 1e9)
23             continue; // 如果之前就拜訪過, 無視
24
25         dis[cur] = d;
26
27         for (auto i : e[cur])
28             if (dis[i.first] == 1e9)
29             {
30                 q.push( MP(d+i.second, i.first) );
31             }
32     }
33 }
34
35 void init(void)
36 {
37     fill(dis, dis+maxn, 1e9);
38
39     for(int i = 0; i < maxn; i++)
40     {
41         e[i].clear();
42     }
43 }

```

3.2 DFS

```

1 #include <stdio.h>
2 #include <deque>
3 using namespace std;
4
5 class GraphNode{
6 public:
7     int id;
8     deque<int> adjacency;
9     int discovered;

```

```

10
11  GraphNode(int _id){
12      id = _id;
13      discovered = 0;
14  }
15  void dump(){
16      printf("Vertex: %d, I'm adjacent to: ", id);
17      int i;
18      for(i = 0; i < adjacency.size(); i++){
19          printf("%d ", adjacency[i]);
20      }
21      printf("\n");
22  };
23 };
24
25 deque<GraphNode*> NodeList;
26 void DFS(int vertex){
27     GraphNode* curr = NodeList[vertex];
28     curr->discovered = 1;
29     printf("%d ", vertex);
30     int i, next;
31     for( i = 0; i < curr->adjacency.size(); i++){
32         next = curr->adjacency[i];
33         if(NodeList[next]->discovered == 0){
34             DFS(next);
35         }
36     }
37 }
38
39 int main(){
40     int node_count, edge_count;
41     printf("Enter no of vertices: ");
42     scanf(" %d", &node_count);
43     printf("Enter no of edges: ");
44     scanf(" %d", &edge_count);
45
46     int i, j;
47     for(i = 0; i < node_count; i++){
48         NodeList.push_back( new GraphNode(i) );
49     }
50
51     printf("Enter %d pairs of vertices: \n",
52           edge_count);
53     for( j = 0; j < edge_count; j++ ){
54         int node_1, node_2;
55         scanf("%d %d", &node_1, &node_2);
56         NodeList[node_1]->adjacency.push_back(node_2);
57         NodeList[node_2]->adjacency.push_back(node_1);
58     }
59     for( i = 0; i < NodeList.size(); i++ ){
60         NodeList[i]->dump();
61     }
62
63     printf("DFS Traversal: ");
64     DFS(0);
65
66     printf("\n");
67     return 0;
68 }

```

3.3 BFS

```

1 #include <stdio.h>
2 #include <deque>
3 using namespace std;
4
5 class GraphNode{
6 public:
7     int id;
8     deque<int>adjacency;
9     int discovered;
10
11     GraphNode(int _id){
12         id = _id;
13         discovered = 0;

```

```

14     }
15     void dump(){
16         printf("Vertex: %d, I'm adjacent to: ", id);
17         int i;
18         for(i = 0; i < adjacency.size(); i++){
19             printf("%d ", adjacency[i]);
20         }
21         printf("\n");
22     };
23 };
24
25 deque<GraphNode*> NodeList;
26
27 void BFS(int start){
28     deque<int> que;
29     que.push_back(start);
30     NodeList[start]->discovered = 1;
31
32     int curr_id, adj_id;
33     GraphNode* curr_node;
34     GraphNode* adj_node;
35
36     while(que.size() != 0){
37         curr_id = que[0];
38         que.pop_front();
39         printf("%d ", curr_id);
40         curr_node = NodeList[curr_id];
41
42         for(int i = 0; i < curr_node->adjacency.size();
43             i++){
44             adj_id = curr_node->adjacency[i];
45             adj_node = NodeList[adj_id];
46             if(adj_node->discovered == 0){
47                 adj_node->discovered = 1;
48                 que.push_back(adj_id);
49             }
50         }
51     }
52
53     int main(){
54         int node_count, edge_count;
55         printf("Enter no of vertices: ");
56         scanf(" %d", &node_count);
57         printf("Enter no of edges: ");
58         scanf(" %d", &edge_count);
59
60         int i, j;
61         for(i = 0; i < node_count; i++){
62             NodeList.push_back( new GraphNode(i) );
63         }
64
65         printf("Enter %d pairs of vertices: \n",
66               edge_count);
67         for( j = 0; j < edge_count; j++ ){
68             int node_1, node_2;
69             scanf("%d %d", &node_1, &node_2);
70             NodeList[node_1]->adjacency.push_back(node_2);
71             NodeList[node_2]->adjacency.push_back(node_1);
72         }
73
74         for( i = 0; i < NodeList.size(); i++ ){
75             NodeList[i]->dump();
76         }
77
78         printf("BFS Traversal: ");
79         BFS(0);
80
81         printf("\n");
82         return 0;
83     }

```

4 Math

4.1 PrimeTable

```

1 void primeTable()
2 {
3     is_notp.reset();
4     is_notp[0] = is_notp[1] = 1;
5     for (int i = 2; i < N; i++)
6     {
7         if (!is_notp[i])
8         {
9             p.push_back(i);
10        }
11        for (int j = 0; j < (int)p.size() && i * p[j]
12              < N; j++)
13        {
14            is_notp[i * p[j]] = 1;
15            if (i % p[j] == 0)
16            {
17                break;
18            }
19        }
20    }

```

```

46     memset(first, -1, sizeof(first));
47     memset(ins, false, sizeof(ins));
48     memset(vis, false, sizeof(vis));
49
50     for(int i = 1; i <= m; i++)
51     {
52         scanf("%d %d",&x,&y);
53         add(x,y);
54     }
55
56     for(int i = 1; i <= n; i++)
57         if(!vis[i])
58             dfs(i);
59
60     if(fail)
61         puts("-1");
62     else
63         for(int i = top-1; i >= 0; i--)
64             printf("%d\n",s[i]);
65 }
66 return 0;
67 }

```

4.2 TopologySort

```

1 #include <bits/stdc++.h>
2 #define maxn 50005
3 using namespace std;
4 struct edge
5 {
6     int t,next;
7 } in[maxn*4];
8 //n vertex has n*4 maximum edges
9
10 int n,m,e,first[maxn],s[maxn],top;
11 // first 紀錄是否有固定順序
12 // s 紀錄順序
13
14 bool fail,ins[maxn],vis[maxn];
15 // vis 是否訪問
16 // ins 在做dfs的當下 那點是否被訪問過
17
18 void add(int x,int y)
19 {
20     in[e].t=y;
21     in[e].next=first[x];
22     first[x]=e++;
23 }
24 void dfs(int cur)
25 {
26     ins[cur]=vis[cur]=true;
27     for(int i=first[cur]; ~i; i=in[i].next)
28     {
29         if(!vis[in[i].t])
30             dfs(in[i].t);
31         else if(ins[in[i].t])
32             fail=true;
33     }
34     ins[cur]=false;
35     s[top++]=cur;
36 }
37 int main(void)
38 {
39     int x,y;
40     while(cin >> n >> m)
41     {
42         //init
43         e = 0;
44         top = 0;
45         fail = false;

```



加油!