Contents

1 Basic

1.1 Map

```cpp
#include <map>
int main(){

    //declaration container and iterator
    map<string, string> mapStudent;
    map<string, string>::iterator iter;
    map<string, string>::reverse_iterator iter_r;

    //insert element
    mapStudent.insert(pair<string, string>("r000",
        "student_zero"));

    mapStudent["r123"] = "student_first";
    mapStudent["r456"] = "student_second";

    //traversal
    for(iter = mapStudent.begin(); iter !=
        mapStudent.end(); iter++)
            cout<<iter->first<<"
                "<<iter->second<<endl;
    for(iter_r = mapStudent.rbegin(); iter_r !=
        mapStudent.rend(); iter_r++)
            cout<<iter_r->first<<"
                "<<iter_r->second<<endl;

    //find and erase the element
    iter = mapStudent.find("r123");
    mapStudent.erase(iter);

    iter = mapStudent.find("r123");

    if(iter != mapStudent.end())
       cout<<"Find, the value is
           "<<iter->second<<endl;
    else
       cout<<"Do not Find"<<endl;

    return 0;
}
```

2 DataStructure

2.1 DisjointSet

```cpp
#define SIZE 10000

int arr[SIZE];

void init(int n) // give a initial length
{
  for(int i=0; i<n; i++)
    arr[i] = -1;
}

int find(int x)
{ // find the father point
  return arr[x] < 0 ? x : (arr[x] = find(arr[x])); //
      update every child to the other father
}

void Union(int x, int y)
{
  x = find(x);
  y = find(y);

  if(x == y)
    return;

  if(arr[x] <= arr[y])
  {
    arr[x] += arr[y];
    arr[y] = x;
  }
  else
  {
    arr[y] += arr[x];
    arr[x] = y;
  }
}
```

2.2 SegmentTree

```cpp
#define SIZE 100000

int st[SIZE];
int st_val[SIZE];

void st_build(int *st, int *st_val, int now, int ls,
    int rs)
{
  if(ls == rs)
    st[now] = st_val[ls];
  else
  {
    st_build(st, st_val, now*2, ls, (ls+rs)/2);
    st_build(st, st_val, now*2+1, (ls+rs)/2+1, rs);
    st[now] = max(st[now*2], st[now*2+1]);
  }
}

// ls and rs are query range, begin and end is whole
    st[] range
int query(int now, int ls, int rs, int begin, int end)
{
  int mid = (begin+end)/2;
  int ret = 0;

  if(ls <= begin && rs >= end)
    return st[now];

  // it is find max now (modify here)
  if(ls <= mid)
    ret = max(ret, query(now*2, ls, rs, begin, mid));

  if(rs > mid)
    ret = max(ret, query(now*2+1, ls, rs, mid+1,
        end));

  return ret;
}
```

2.3 AdjList

```cpp
#include <bits/stdc++.h>
using namespace std;

#define MAXN 100

//the size of matrix
int n;
//the number of edges
int m;
//declare the List with MAXN
vector<pair<int, int>> Adj[ MAXN ];

void inputList() {
    //weight
    int a, b, w;

    for (int i = 0; i < m; i++) {
        //input and store
        cin >> a >> b >> w;
        Adj[ a ].push_back(make_pair(w, b));
            //雙向圖
        Adj[ b ].push_back(make_pair(w, a));

    }
}

void printList() {
    printf("The adjacency list:\n");
    for (int i = 0; i < n; i++) {
        printf("[%d] --> ", i);
        for (auto p : Adj[ i ]) {
            printf("(%d, %d)  ---> ", p.first,
                p.second);
        }
        puts("NULL");
    }
}

int main() {
    cin >> n >> m;
    inputList();
    printList();
}
```

2.4 AdjMatrix

```cpp
#include <bits/stdc++.h>
using namespace std;

#define MAXN 100

//the size of matrix
int n;
//the number of edges
int m;
//declare the Matrix with MAXN
int Adj[ MAXN ][ MAXN ];

void inputMatrix() {
    int a, b;

    for (int i = 0; i < m; i++) {
        //input and store
        cin >> a >> b;
        Adj[ a ][ b ] = 1;
    }
}

void printMatrix() {
    printf("The adjacency matrix:\n");
    for (int i = 0; i < n; i++) {
        printf("|");
        for (int j = 0; j < n; j++) {
            printf("%3d ", Adj[ i ][ j ]);
        }
        puts("|");
    }
}

int main() {
    //initialize the matrix
    memset(Adj, 0, sizeof(Adj));  //初始化
    cin >> n >> m;
    inputMatrix();
    printMatrix();
}
```

3 Graph

3.1 Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;

#define MP make_pair
#define PII pair<int, int>
#define maxn 50000+5

int dis[maxn]; // 預設都是 INF
vector<PII > e[maxn]; // (連到的點， 邊的距離)

void dijk(int cur) // dijk(起點)
{
  int d;
  priority_queue<PII,vector<PII>,greater<PII>> q; //
      放 (距離, 點編號)，每次會拿距離最小的點出來
  q.push( MP(0, cur) );

  while (!q.empty())
  {
    tie(d, cur) = q.top();
    q.pop();
    if (dis[cur] != 1e9)
        continue; // 如果之前就拜訪過，無視

    dis[cur] = d;

    for (auto i: e[cur])
        if (dis[i.first] == 1e9)
        {
            q.push( MP(d+i.second, i.first) );
        }
    }
}

void init(void)
{
    fill(dis, dis+maxn, 1e9);

    for(int i = 0; i < maxn; i++)
    {
        e[i].clear();
    }
}
```

3.2 DFS

```cpp
#include <stdio.h>
#include <deque>
using namespace std;

class GraphNode{
public:
  int id;
  deque<int>adjacency;
  int discovered;
```

```

  GraphNode(int _id){
    id = _id;
    discovered = 0;
  }
  void dump(){
    printf("Vertex: %d, I'm adjacent to: ", id);
    int i;
    for(i = 0; i<adjacency.size(); i++){
      printf("%d ", adjacency[i]);
    }
    printf("\n");
  };
};

deque<GraphNode*> NodeList;
void DFS(int vertex){
  GraphNode* curr = NodeList[vertex];
  curr->discovered = 1;
  printf("%d ", vertex);
  int i, next;
  for( i = 0; i < curr->adjacency.size(); i++){
    next = curr->adjacency[i];
    if(NodeList[next]->discovered == 0){
      DFS(next);
    }
  }
}

int main(){
  int node_count, edge_count;
  printf("Enter no of vertices: ");
  scanf(" %d", &node_count);
  printf("Enter no of edges: ");
  scanf(" %d", &edge_count);

  int i, j;
  for(i = 0; i<node_count; i++){
    NodeList.push_back( new GraphNode(i) );
  }

  printf("Enter %d pairs of verteices: \n",
      edge_count);
  for( j = 0; j < edge_count; j++ ){
    int node_1, node_2;
    scanf("%d %d", &node_1, &node_2);
    NodeList[node_1]->adjacency.push_back(node_2);
    NodeList[node_2]->adjacency.push_back(node_1);
  }

  for( i = 0; i < NodeList.size(); i++ ){
    NodeList[i]->dump();
  }

  printf("DFS Traversal: ");
  DFS(0);

  printf("\n");
  return 0;
}
```

3.3 BFS

```
#include <stdio.h>
#include <deque>
using namespace std;

class GraphNode{
public:
  int id;
  deque<int>adjacency;
  int discovered;

  GraphNode(int _id){
    id = _id;
    discovered = 0;
  }
  void dump(){
    printf("Vertex: %d, I'm adjacent to: ", id);
    int i;
    for(i = 0; i<adjacency.size(); i++){
      printf("%d ", adjacency[i]);
    }
    printf("\n");
  };
};

deque<GraphNode*> NodeList;

void BFS(int start){
  deque<int> que;
  que.push_back(start);
  NodeList[start]->discovered = 1;

  int curr_id, adj_id;
  GraphNode* curr_node;
  GraphNode* adj_node;

  while(que.size() != 0){
    curr_id = que[0];
    que.pop_front();
    printf("%d ", curr_id);
    curr_node = NodeList[curr_id];

    for(int i = 0; i < curr_node->adjacency.size();
        i++){
      adj_id = curr_node->adjacency[i];
      adj_node = NodeList[adj_id];
      if(adj_node->discovered == 0){
        adj_node->discovered = 1;
        que.push_back(adj_id);
      }
    }
  }
}

int main(){
  int node_count, edge_count;
  printf("Enter no of vertices: ");
  scanf(" %d", &node_count);
  printf("Enter no of edges: ");
  scanf(" %d", &edge_count);

  int i, j;
  for(i = 0; i<node_count; i++){
    NodeList.push_back( new GraphNode(i) );
  }

  printf("Enter %d pairs of verteices: \n",
      edge_count);
  for( j = 0; j < edge_count; j++ ){
    int node_1, node_2;
    scanf("%d %d", &node_1, &node_2);
    NodeList[node_1]->adjacency.push_back(node_2);
    NodeList[node_2]->adjacency.push_back(node_1);
  }

  for( i = 0; i < NodeList.size(); i++ ){
    NodeList[i]->dump();
  }

  printf("BFS Traversal: ");
  BFS(0);

  printf("\n");
  return 0;
}
```

4 Math

4.1 PrimeTable

```cpp
void primeTable()
{
    is_notp.reset();
    is_notp[0] = is_notp[1] = 1;
    for (int i = 2; i < N; i++)
    {
        if (!is_notp[i])
        {
            p.push_back(i);
        }
        for (int j = 0; j < (int)p.size() && i * p[j]
            < N; j++)
        {
            is_notp[i * p[j]] = 1;
            if (i % p[j] == 0)
            {
                break;
            }
        }
    }
}
```

4.2 TopologySort

```cpp
#include <bits/stdc++.h>
#define maxn 50005
using namespace std;
struct edge
{
    int t,next;
}   in[maxn*4];
//n vertex has n*4 maximum edges

int n,m,e,first[maxn],s[maxn],top;
//  first   紀錄是否有固定順序
//  s 紀錄順序

bool fail,ins[maxn],vis[maxn];
//  vis 是否訪問
//  ins 在做dfs的當下 那點是否被訪問過

void add(int x,int y)
{
    in[e].t=y;
    in[e].next=first[x];
    first[x]=e++;
}
void dfs(int cur)
{
    ins[cur]=vis[cur]=true;
    for(int i=first[cur]; ~i; i=in[i].next)
    {
        if(!vis[in[i].t])
            dfs(in[i].t);
        else if(ins[in[i].t])
            fail=true;
    }
    ins[cur]=false;
    s[top++]=cur;
}
int main(void)
{
    int x,y;
    while(cin >> n >> m)
    {
        //init
        e = 0;
        top = 0;
        fail = false;
        memset(first, -1, sizeof(first));
        memset(ins, false, sizeof(ins));
        memset(vis, false, sizeof(vis));

        for(int i = 1; i <= m; i++)
        {
            scanf("%d %d",&x,&y);
            add(x,y);
        }

        for(int i = 1; i <= n; i++)
            if(!vis[i])
                dfs(i);

        if(fail)
            puts("-1");
        else
            for(int i = top-1; i >= 0; i--)
                printf("%d\n",s[i]);
    }
    return 0;
}
```

加油!