

Contents

1	Basic
1.1	Syntax
1.2	BigInteger
2	Data Structure
2.1	Disjoint Set
2.2	Segment Tree
3	Divide and Conquer
3.1	MaximumSubArray
4	Dynamic Programming
4.1	LCS
4.2	LIS
5	Search
5.1	Binary Search
6	Sequence
6.1	RSQ(Prefix Sum)
6.2	RSQ(2DPrefix Sum)
6.3	RSQ(Fenwick Tree)
7	Sorting
7.1	Counting Sort
7.2	Topology Sort
7.3	Topology Sort with DFS(check 有無環)
8	Graph
8.1	DFS I
8.2	DFS II
8.3	BFS
8.4	Dijkstra
8.5	SPFA
8.6	BellmanFord
8.7	FloydWarshall
8.8	Kruskal
8.9	Bipartite Matching
8.10	CLE Directed MST
8.11	Convex Hull
9	Number
9.1	Sieve
9.2	Power
9.3	Euler

1 Basic

1.1 Syntax

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(int argc, char const *argv[])
4 {
5     // String to Integer
6     char str[30] = {'-', '1', '2', '3', '4', '5', '\0'};
7     printf("%d\n", stoi(str));
8     // Integer to String
9     int x = 185;
10    char temp[30];
11    int base = 10;
12    itoa(x, temp, base);
13    printf("%s\n", temp);
14    // String to Double
15    char strd[30] = {'0', '.', '6', '0', '2', '2', '2', '9', '\0'};
16    printf("%Lf\n", stod(strd));
17    // Double to String
18    double y = 3.1415926;
19    string dstr = to_string(y);
20    cout << dstr << endl;
21    // String initialize
22    char null[30] = {'\0'};
23    char A[30];
24    strcpy(A, null);
25    // String Length
26    char strl[30] = {'H', 'E', 'L', 'L', 'O', '\0'};
27    printf("%d\n", strlen(strl));
28    return 0;

```

29 | }

1.2 BigInteger

```

1 import java.math.BigInteger;
2 import java.util.Scanner;
3 class Main {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         BigInteger n = input.nextBigInteger();
7         BigInteger m = input.nextBigInteger();
8         n.add(m); a.subtract(m); n.multiply(m); n.
9             divide(m); n.mod(m);
10        n.pow(m.intValue()); n.gcd(m); n.negate(); n.
11        abs();
12    }
13 }

```

2 Data Structure

2.1 Disjoint Set

```

5
5 1 const int n = 6; // number of nodes
5 2 int p[n+10];
5 3 void init()
6 4 {
6 5     for(int i = 0; i < n; i++){
7 6         p[i] = -1;
7 7     }
8 8 }
8 9 int find(int x){
10 10 int root, trail, lead;
9 11 for (root = x; p[root] >= 0; root = p[root]);
9 12 for (trail = x; trail != root; trail = lead) {
9 13     lead = p[trail];
14 14     p[trail] = root;
15 15 }
16 16 return root;
17 17 }
18 void uni(int x, int y)
19 {
20     int xRoot = find(x), yRoot = find(y);
21     if(xRoot != yRoot){
22         if(p[xRoot] > p[yRoot]){
23             p[xRoot] += p[yRoot];
24             p[yRoot] = xRoot;
25         }
26         else{
27             p[yRoot] += p[xRoot];
28             p[xRoot] = yRoot;
29         }
30     }
31 }

```

2.2 Segment Tree

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 8;
4 int B[n] = {18, 17, 13, 19, 15, 11, 20, 87};
5 typedef vector<int> vi;
6 vi A (B, B + 8);
7 vi ST;
8 void ST_Build(vi &ST, const vi &A, int vertex, int L,
9     int R)
9 {
10     if(L == R) ST[vertex] = L;
11     else
12     {
13         int nL = vertex * 2, nR = vertex * 2 + 1;

```

```

14     ST_Build(ST, A, nL, L, L + (R - L) / 2);
15     ST_Build(ST, A, nR, L + (R - L) / 2 + 1, R);
16     int indexL = ST[nL], indexR = ST[nR];
17     int valueL = A[indexL], valueR = A[indexR];
18     ST[vertex] = valueL <= valueR ? indexL : indexR;
19 }
20 }
21
22 void ST_Creation(vi &ST, const vi &A)
23 {
24     int len = 4 * A.size();
25     ST.assign(len, 0);
26     ST_Build(ST, A, 1, 0, A.size()-1);
27 }
28 int query(vi &ST, const vi &A, int vertex, int L, int R
29           , int qL, int qR)
30 {
31     int temp, mid = (L + R) / 2;
32     if(qL <= L && R <= qR) return ST[vertex];
33     if(qR <= mid)
34     { //all we want at the left child
35         return query(ST, A, vertex * 2, L, mid, qL, qR);
36     }
37     if(qL > mid)
38     { // all we want at the right child
39         return query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
40                       qR);
41     }
42     return A[query(ST, A, vertex * 2, L, mid, qL, qR)] <=
43            A[query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
44                    qR)]
45            ? query(ST, A, vertex * 2, L, mid, qL, qR) :
46              query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
47                    qR);
48 }
49
50 void update(vi &ST, vi &A, int x, int L, int R, int p,
51            int v)
52 {
53     // p is the index where you want to update
54     // v is the value will be update in A[p];
55     int mid = L + (R - L) / 2;
56     if(L == R) A[ST[x]] = v;
57     else
58     {
59         if(p <= mid) update(ST, A, x*2, L, mid, p, v);
60         else update(ST, A, x*2+1, mid+1, R, p, v);
61         ST[x] = (A[ST[x*2]] <= A[ST[x*2+1]]) ? ST[x*2] : ST
62                [x*2+1];
63     }
64 }
65
66 int main(int argc, char const *argv[])
67 {
68     ST_Creation(ST, A);
69     printf("%d\n", query(ST, A, 1, 0, n-1, 3, 7));
70     // query return the index
71     printf("%d\n", A[query(ST, A, 1, 0, n-1, 3, 7)]);
72     update(ST, A, 1, 0, n-1, 5, 18);
73     // query and update first to fifth parameter dont
74     change
75     // ST, A, 1, 0, n-1
76     // last two would be
77     // query: the range(array index) you want to query
78     // update: first is the index you want to update,
79     second is the value will be
80     return 0;
81 }

```

3 Divide and Conquer

3.1 MaximumSubArray

```

1 #include <bits/stdc++.h>
2 using namespace std;

```

```

3 const int n = 16;
4 int arr[n] = {13, -3, -25, 20, -3, -16, -23,
5              18, 20, -7, 12, -5, -22, 15, -4, 7};
6
7 int findMaxCrossing(int left, int mid, int right){
8     int maxl = 0x80000000;
9     int sum = 0;
10    for(int i = mid; i >= left; i--){
11        sum += arr[i];
12        if(sum > maxl) maxl = sum;
13    }
14    int maxr = 0x80000000;
15    sum = 0;
16    for(int i = mid + 1; i <= right; i++){
17        sum += arr[i];
18        if(sum > maxr) maxr = sum;
19    }
20
21    return (maxl + maxr);
22 }
23
24 int findMaxSub(int left, int right)
25 {
26     if(left == right){
27         return arr[left];
28     }
29     else{
30         int mid = left + (right - left) / 2;
31         int maxl = findMaxSub(left, mid);
32         int maxr = findMaxSub(mid + 1, right);
33         int res = max(maxl, maxr);
34         res = max(res, findMaxCrossing(left, mid, right));
35         return res;
36     }
37 }
38
39
40 int main(int argc, char const *argv[])
41 {
42     printf("%d\n", findMaxSub(0, n-1));
43     return 0;
44 }

```

4 Dynamic Programming

4.1 LCS

```

1 const int maxn = 10000; // maxn is maximum length of
2 arrp and arrq
3 int arrp[maxn], arrq[maxn];
4 int dp[maxn+5][maxn+5];
5 int p, q; // p is the length of arrp, q is the length
6 of arrq
7 void LCS()
8 {
9     memset(dp, 0, sizeof(dp));
10
11     for(int i = 1; i <= p; i++){
12         for(int j = 1; j <= q; j++){
13             if(arrp[i] == arrq[j]){
14                 dp[i][j] = 1 + dp[i-1][j-1];
15             }
16             else{
17                 dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
18             }
19         }
20     }
21
22     // dp[p][q] is the answer
23 }

```

4.2 LIS

```

1 int LIS(vector<int>& s)
2 {
3     if (s.size() == 0) return 0;
4
5     vector<int> v;
6     v.push_back(s[0]);
7
8     for (int i = 1; i < s.size(); ++i)
9     {
10         int n = s[i];
11
12         if (n > v.back())
13             v.push_back(n);
14         else
15             *lower_bound(v.begin(), v.end(), n) = n;
16     }
17
18     return v.size();
19 }

```

5 Search

5.1 Binary Search

```

1 int L = 0; // left boundary
2 int R = ans; // right boundary
3 // check using L = 3, R = 4, ans = 4
4 while(L < R){
5     int M = L + (R - L + 1) / 2; // left + half distance
6     if(ok(M)) L = M; // ok() method is to find
7     // whether the M can qualify the demand
8     else R = M - 1;
9 }
10 while(L < R){
11     int M = L + (R - L) / 2; // left + half distance
12     if(ok(M)) R = M; // ok() method is to find
13     // whether the M can qualify the demand
14     else L = M + 1;
15 }

```

6 Sequence

6.1 RSQ(Prefix Sum)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 10;
4 int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
5 int query[maxn];
6 void init()
7 {
8     // every query is the sum of all previos element,
9     // include it self
10    // example query[3] = arr[0] + arr[1] + arr[2] + arr
11    [3]
12    query[0] = arr[0];
13    for(int i = 1; i < maxn; i++){
14        query[i] = arr[i];
15        query[i] += query[i-1];
16    }
17 }
18 int RangeSumQuery(int s, int e)
19 {
20     //Prefix Sum Algorithm
21     if(s >= 1) return query[e] - query[s-1];
22     else return query[e];
23 }
24 int main(int argc, char const *argv[])
25 {
26     init();

```

```

25 int start = 2, end = 5;
26 printf("RangeSumQuery(%d, %d): %d\n", start, end,
27        RangeSumQuery(start, end));
28
29 return 0;

```

6.2 RSQ(2DPrefix Sum)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int arr[110][110];
4 int query[110][110];
5 int n;
6
7 int main(int argc, char const *argv[])
8 {
9     while(cin >> n){
10        // input
11        for(int i = 0; i < n; i++){
12            for(int j = 0; j < n; j++){
13                cin >> arr[i][j];
14            }
15        }
16        // bulid prefix query
17        for(int i = 0; i < n; i++){
18            for(int j = 0; j < n; j++){
19                query[i][j] = arr[i][j];
20                if(i - 1 >= 0) query[i][j] += query[i-1][j];
21                if(j - 1 >= 0) query[i][j] += query[i][j-1];
22                if(i - 1 >= 0 && j - 1 >= 0) query[i][j] -=
23                    query[i-1][j-1];
24            }
25        }
26        int temp;
27        int maximum = 0x80000000;
28        // find the maximum sum in any range
29        for(int i = 0; i < n; i++){
30            for(int j = 0; j < n; j++){
31                for(int k = i; k < n; k++){
32                    for(int t = j; t < n; t++){
33                        temp = query[k][t];
34                        if(i - 1 >= 0) temp -= query[i-1][t];
35                        if(j - 1 >= 0) temp -= query[k][j-1];
36                        if(i - 1 >= 0 && j - 1 >= 0) temp += query[
37                            i-1][j-1];
38                        if(maximum < temp) maximum = temp;
39                    }
40                }
41            }
42        }
43        printf("%d\n", maximum);
44    }
45    return 0;
46 }

```

6.3 RSQ(Fenwick Tree)

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 10;
4 int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
5 int FenwickTree[maxn];
6 int ANDlowbit(int src)
7 {
8     // src & -src will get the Lowbit
9     // example: 6 & -6 = 0110 & 1010 = 0010 = 2
10    return src & -src;
11 }
12 void init()
13 {

```

```

14 |
15 | memset(FenwickTree, 0, sizeof(FenwickTree));
16 | // Notice that we start in 1
17 | for(int i = 1; i <= maxn; i++){
18 |     int index = i;
19 |     FenwickTree[i] += arr[i-1];
20 |     int temp = arr[i-1];
21 |     while(index + ANDlowbit(index) <= maxn){
22 |         index += ANDlowbit(index);
23 |         FenwickTree[index] += temp;
24 |     }
25 | }
26 |
27 | void Modify(int src, int val)
28 | {
29 |     // Modify arr[src] to val
30 |     int gap = val - arr[src];
31 |     arr[src] = val;
32 |     int index = src + 1;
33 |     FenwickTree[index] += gap;
34 |     while(index + ANDlowbit(index) <= maxn){
35 |         index += ANDlowbit(index);
36 |         FenwickTree[index] += gap;
37 |     }
38 | }
39 | int SequenceQuery(int src)
40 | {
41 |     //src is the index of the array which we want to know
42 |     the Sequence Query
43 |     int res = FenwickTree[src];
44 |     int index = src;
45 |     while(index - ANDlowbit(index) > 0){
46 |         index -= ANDlowbit(index);
47 |         res += FenwickTree[index];
48 |     }
49 |     return res;
50 | }
51 | int RangeSumQuery(int s, int e)
52 | {
53 |     return SequenceQuery(e) - SequenceQuery(s - 1);
54 | }
55 | int main(int argc, char const *argv[])
56 | {
57 |     init();
58 |     int start = 2, end = 5;
59 |     // for Fenwick index is 3, 6 for array index is 2, 5
60 |     printf("RangeSumQuery(%d, %d): %d\n", start, end,
61 |         RangeSumQuery(start + 1, end + 1));
62 |     Modify(2, 5);
63 |     // Modify arr[2] from 3 to 5
64 |     printf("RangeSumQuery(%d, %d): %d\n", start, end,
65 |         RangeSumQuery(start + 1, end + 1));
66 |     return 0;
67 | }

```

7 Sorting

7.1 Counting Sort

```

1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | const int maxn = 50;
4 | const int maxDigit = 1050;
5 | int unsorted[maxn] = {0, 3, 7, 6, 5}, sorted[maxn], aux
6 | [maxDigit];
7 | // aux size is depends on the max digit in sorting
8 | int main(int argc, char const *argv[])
9 | {
10 |     int n = 4;
11 |     // array index start with 1
12 |     memset(aux, 0, sizeof(aux));
13 |     for(int i = 1; i <= n; i++){
14 |         aux[unsorted[i]]++;
15 |     }

```

```

15 | for(int i = 1; i < maxDigit; i++){
16 |     aux[i] += aux[i-1];
17 | }
18 | for(int i = n; i > 0; i--){
19 |     sorted[aux[unsorted[i]]] = unsorted[i];
20 |     aux[unsorted[i]]--;
21 | }
22 | for(int i = 1; i <= n; i++){
23 |     printf("%d ", sorted[i]);
24 | }
25 | return 0;
26 | }

```

7.2 Topology Sort

```

1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | const int maxn = 100;
4 | vector<int> ans;
5 | vector<int> adj[maxn];
6 | int refs[maxn];
7 | int n = 5;
8 |
9 | // refs 紀錄這個點被幾個邊連到
10 | void TopologyOrder()
11 | {
12 |     for(int i = 0; i < n; i++){
13 |         int s = 0;
14 |         while(s < n && refs[s] != 0) {
15 |             s++;
16 |         }
17 |         if(s == n) break;
18 |         refs[s] = -1;
19 |         ans.push_back(s);
20 |         for(auto j : adj[s]){
21 |             refs[j]--;
22 |         }
23 |     }
24 | }
25 | int main(int argc, char const *argv[])
26 | {
27 |     memset(refs, 0, sizeof(refs));
28 |     ans.clear();
29 |     // adj[from].push_back(to); refs[to]++;
30 |     adj[4].push_back(1); refs[1]++;
31 |     adj[1].push_back(3); refs[3]++;
32 |     adj[1].push_back(0); refs[0]++;
33 |     adj[2].push_back(0); refs[0]++;
34 |     adj[3].push_back(0); refs[0]++;
35 |     TopologyOrder();
36 |     for(int i = 0; i < ans.size(); i++){
37 |         if(i == ans.size()-1) printf("%d\n", ans[i]);
38 |         else printf("%d ", ans[i]);
39 |     }
40 |     return 0;
41 | }

```

7.3 Topology Sort with DFS(有無環)

```

1 | const int maxn = 5000+50;
2 | vector<int> adj[maxn];
3 | stack<int> ans;
4 | int state[maxn];
5 | bool head[maxn];
6 | bool valid;
7 | int n, m;
8 |
9 | void dfs(int src)
10 | {
11 |     state[src] = 1;
12 |
13 |     for (auto next : adj[src])
14 |         if (!state[next]) dfs(next);

```

```

15     else if (state[next] == 1){
16         // 有環
17         valid = false;
18         return;
19     }
20
21     state[src] = 2;
22
23     ans.push(src);
24 }
25
26 void topology_sort()
27 {
28     for (int i = 0; i < n; i++){
29         // 從 (0 ~ n-1) 找一個頭沒有被任何人連到的開始
30         // 做dfs
31         if (valid && head[i]) dfs(i);
32     }
33
34     if (!valid)
35     {
36         cout << "Cycle!" << endl;
37         return;
38     }
39
40     while (!ans.empty())
41     {
42         cout << ans.top() << endl;
43         ans.pop();
44     }
45 }
46
47 int main()
48 {
49     cin >> n >> m;
50
51     memset(head, true, sizeof(head));
52     // make adjacent list
53     for (int i = 0; i < m; i++)
54     {
55         int a, b;
56         cin >> a >> b;
57
58         head[b] = false;
59         //
60         adj[a].push_back(b);
61     }
62
63     memset(state, 0, sizeof(state));
64     valid = true;
65     //如果 valid = false代表有還
66     topology_sort();
67
68     return 0;
69 }

```

8 Graph

8.1 DFS I

```

1 //implement by adjacent list
2 //functional dfs
3 void dfs(int now, int fa, int layer){
4     for (auto j : adj[now])
5         if(j != fa ) dfs(j, now, layer + 1);
6 }
7 //stack dfs
8 stack<int> st;
9 bool vis[maxn];
10 memset(vis, false, sizeof(vis));
11 int src;
12 st.push(src);
13 while(!st.empty())
14 {

```

```

15     int now = st.top(); st.pop();
16     vis[now] = true;
17     for(auto i : adj[now])
18         if(!vis[i]) st.push(i);
19 }

```

8.2 DFS II

```

1 const int maxn = 10;
2 struct Node{
3     int d, f, color;
4     // d: discover time, f: finish time, color: 0 ==
5     // white, 1 == gray, 2 == black
6 };
7 vector<int> adj[maxn];
8 Node node[maxn];
9 int times;
10 void DFS(int src)
11 {
12     node[src].d = times++;
13     node[src].color = 1;
14     for(auto i : adj[src]){
15         if(node[i].color == 0) DFS(i);
16     }
17     node[src].color = 2;
18     node[src].f = times++;
19 }
20 void DFS_Start(int n, int sp)
21 {
22     for(int i = 0; i < n; i++){
23         node[i].color = 0;
24     }
25     times = 0;
26     DFS(sp);
27 }
28 int main(int argc, char const *argv[])
29 {
30     int n, m, x, y;
31     cin >> n >> m;
32     for(int i = 0; i < n; i++) adj[i].clear();
33     for(int i = 0; i < m; i++){
34         cin >> x >> y;
35         adj[x].push_back(y);
36     }
37     DFS_Start(6, 0);
38     for(int i = 0; i < n; i++){
39         printf("%d: d: %d f: %d color: %d\n", i, node[i].d,
40             node[i].f, node[i].color);
41     }
42     return 0;
43 }

```

8.3 BFS

```

1 queue<int> st;
2 bool vis[maxn];
3 memset(vis, false, sizeof(vis));
4 int src;
5 st.push(src);
6 while(!st.empty())
7 {
8     int now = st.front(); st.pop();
9     vis[now] = true;
10    for(auto i : adj[now])
11        if(!vis[i]) st.push(i);
12 }

```

8.4 Dijkstra

```

1 #define PII pair<int, int>
2 #define maxn 50000+5
3
4 int dis[maxn]; // 預設都是 INF
5 vector<PII> adj[maxn]; // (連到的點, 邊的距離)
6
7 void dijk(int src) // dijk(起點)
8 {
9     priority_queue<PII, vector<PII>, greater<PII>> q; // 放
        (點編號, 距離), 每次會拿距離最小的點出來
10    q.push(make_pair(src, 0));
11
12    while (!q.empty())
13    {
14        auto top = q.top(); q.pop();
15        printf("%d %d\n", top.first, top.second);
16        if (dis[top.first] != 1e9) continue; // 如果之前
            就拜訪過就continue
17
18        dis[top.first] = top.second;
19        for (auto i: adj[top.first]){
20            if (dis[i.first] == 1e9) q.push(make_pair(i.
                first, i.second + top.second));
21        }
22    }
23 }
24
25 void init(void)
26 {
27     fill(dis, dis+maxn, 1e9);
28
29     for(int i = 0; i < maxn; i++){
30         adj[i].clear();
31     }
32 }
33
34 int main(int argc, char const *argv[])
35 {
36     init();
37     // make adjcnet List
38     int sp = 0; // start point
39     dijk(sp);
40     // dis[x] store the distance from sp to x
41     return 0;
42 }

```

8.5 SPFA

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define INF 0x3f3f3f3f
5 const int maxn = 10000+5;
6
7 int n, m;
8 int dist[maxn], vis[maxn], out[maxn];
9 //dist = distance, vis = visit, out
10 vector<pair< int, int >> adj[maxn];
11
12 void init()
13 {
14     memset(dist, INF, sizeof(dist));
15     memset(vis, 0, sizeof(vis));
16     memset(out, 0, sizeof(out));
17     for(int i = 0; i <= n; i++){
18         adj[i].clear();
19     }
20 }
21
22 bool spfa(int sp, int n)
23 {
24     queue<int> q;
25     q.push(sp);
26
27     while(!q.empty())

```

```

28 {
29     int u = q.front(); q.pop();
30     vis[u] = 0; // pop point
31     out[u]++;
32     if(out[u] > n) return false; // negative cycle
        occurs
33
34     for(int j = 0; j < adj[u].size(); j++){
35         int v = adj[u][j].first; // first is point,
            second is weight
36         if(dist[v] > dist[u] + adj[u][j].second){
37             dist[v] = dist[u] + adj[u][j].second;
38             if(vis[v]) continue;
39
40             vis[v] = 1; //push point
41             q.push(v);
42         }
43     }
44 }
45 return true;
46 }
47
48 int main(int argc, char const *argv[])
49 {
50     // n nodes and m edges
51     scanf("%d%d", &n, &m);
52     init();
53     // make adjacent List
54     int a, b, w;
55     for(int i = 0; i < m; i++){
56         scanf("%d%d%d", &a, &b, &w);
57         adj[a].push_back(make_pair(b, w));
58     }
59     int sp = 0; // start point
60     dist[sp] = 0; vis[sp] = 1;
61     if(spfa(sp, n))
62         for (int i = 0; i < n; i++) printf("dist %d: %d\n",
            i, dist[i]);
63     else printf("can't reach.\n");
64     return 0;
65 }

```

8.6 BellmanFord

```

1 int main(int argc, char const *argv[])
2 {
3     //initialize dis[] with 1e9
4     //make an adjecnt List
5     call bellman_ford(src);
6     return 0;
7 }
8
9 void bellman_ford(int src)
10 {
11     dis[src] = 0; //initialize source
        with distance 0
12     for (int k = 0; k < n - 1; ++k){ //do n-1
        times
13         for (int i = 0; i < n; ++i){
14             for(auto j : v[i]){
15                 if(dis[i] != 1e9) dis[j] = min(dis[j], dis[i] +
                    w[i][j]);
16             }
17         }
18     }
19 }
20 bool negativeCycle()
21 {
22     for(i = 0; i < n; ++i){
23         for(auto j : v[i]){
24             if(dis[j] > dis[i] + w[i][j]) return true //has
                negative cycle
25         }
26     }
27     return false;
28 }

```

8.7 FloydWarshall

```

1 //dis[i][j] is the distance of node i to node j
2 int dis[n+5][n+5];
3 void init()
4 {
5     memset(dis, 0x3f, sizeof(dis));
6     for(int i = 0; i < n; i++) d[i][i] = 0;
7 }
8 void floyd(){
9     for (int k = 0; k < n; ++k)
10         for(int i = 0; i < n; ++i)
11             for(int j = 0; j < n; ++j)
12                 dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
13 }
14 int main(int argc, char const *argv[])
15 {
16     //If we got n nodes, Label from 0 to (n-1)
17     init();
18     //Set the dis
19     floyd();
20 }

```

8.8 Kruskal

```

1 const int maxn = 1000+5;
2 struct Edge
3 {
4     int from, to;
5     double cost;
6     bool operator<(const Edge other){
7         return cost < other.cost;
8     }
9 }E[maxn*maxn];
10 int p[maxn];
11 vector<Edge> G[maxn];
12 int find(int x){
13     int root, trail, lead;
14     for (root = x; p[root] >= 0; root = p[root]);
15     for (trail = x; trail != root; trail = lead) {
16         lead = p[trail];
17         p[trail] = root;
18     }
19     return root;
20 }
21 bool uni(int x, int y)
22 {
23     int xRoot = find(x), yRoot = find(y);
24     if(xRoot != yRoot){
25         if(p[xRoot] > p[yRoot]){
26             p[xRoot] += p[yRoot];
27             p[yRoot] = xRoot;
28         }
29         else{
30             p[yRoot] += p[xRoot];
31             p[xRoot] = yRoot;
32         }
33         return true;
34     }
35     else return false;
36 }
37 double kruskal(int n, int m)
38 {
39     // n is the numbers of node, m is the numbers of edge
40     for(int i = 0; i <= n; i++){
41         G[i].clear();
42         p[i] = -1;
43     }
44     sort(E, E + m);
45     double ans = 0;
46     int edge_cnt = 0;
47     for(int i = 0; i < m; i++){
48         if(uni(E[i].from, E[i].to)){

```

```

49             int from = E[i].from, to = E[i].to;
50             ans += E[i].cost;
51             G[from].push_back(Edge{from, to, E[i].cost});
52             G[to].push_back(Edge{to, from, E[i].cost});
53             if(++edge_cnt == n-1) break;
54         }
55     }
56     if(edge_cnt == n-1) return ans;
57     else return -1; // means can't found spanning tree
58 }
59 // find max segment in MST graph
60 int maxcost[maxn][maxn];
61 vector<int> visited;
62 void dfs(int pre, int now, int w){
63     for(auto x : visited){
64         maxcost[x][now] = maxcost[now][x] = max(w, maxcost[
65             pre][x]);
66     }
67     visited.push_back(now);
68     for(auto i : G[now]){
69         if(pre != i.to) dfs(i.from, i.to, i.cost);
70     }
71 }
72 void findMaxPtah(int sp, int ep){
73     memset(maxcost, 0, sizeof(maxcost));
74     visited.clear();
75     dfs(-1, sp, 0);
76 }

```

8.9 Bipartite Matching

```

1 const int maxn = 500+5;
2 int W[maxn][maxn], n;
3 int Lx[maxn], Ly[maxn];
4 int Lef[maxn];
5 bool S[maxn], T[maxn];
6 bool match(int i)
7 {
8     S[i] = true;
9     for (int j = 1; j <= n; ++j)
10     {
11         if(Lx[i] + Ly[j] == W[i][j] && !T[j])
12         {
13             T[j] = true;
14             if(!Lef[j] || match(Lef[j]))
15             {
16                 Lef[j] = i;
17                 return true;
18             }
19         }
20     }
21     return false;
22 }
23 void update()
24 {
25     int a = 0x3f3f3f3f;
26     for(int i = 1; i <= n; i++)
27     {
28         if(S[i])
29         {
30             for(int j = 1; j <= n; j++)
31             {
32                 if(!T[j]) a = min(a, Lx[i] + Ly[j] - W[i][j]);
33             }
34         }
35     }
36     for(int i = 1; i <= n; i++)
37     {
38         if(S[i]) Lx[i] -= a;
39         if(T[i]) Ly[i] += a;
40     }
41 }
42 void KM()
43 {
44     for (int i = 1; i <= n; ++i)

```



```

46 {
47     Lef[i] = Lx[i] = Ly[i] = 0;
48     for(int j = 1; j <= n; j++){
49         Lx[i] = max(Lx[i], W[i][j]);
50     }
51 }
52 for (int i = 1; i <= n; ++i)
53 {
54     for(;;){
55         for(int j = 1; j <= n; j++){
56             S[j] = T[j] = 0;
57         }
58         if(match(i)) break;
59         else update();
60     }
61 }
62 }
63 }
64 }
65 int main(int argc, char const *argv[])
66 {
67     for(int i = 1; i <= n; i++){
68         for(int j = 1; j <= n; j++){
69             scanf("%d", &W[i][j]);
70         }
71     }
72
73     KM();
74     int ans = 0;
75
76     for(int i = 1; i <= n; i++){
77         ans += Ly[i];
78         ans += Lx[i];
79     }
80
81     for(int i = 1; i <= n; i++){
82         if(i != n) printf("%d ", Lx[i]);
83         else printf("%d\n", Lx[i]);
84     }
85
86     for(int i = 1; i <= n; i++){
87         if(i != n) printf("%d ", Ly[i]);
88         else printf("%d\n", Ly[i]);
89     }
90
91     printf("%d\n", ans);
92     return 0;
93 }

```

8.10 CLE Directed MST

```

1 const int maxn = 60+5;
2 const int INF = 0x3f3f3f3f;
3 struct Edge
4 {
5     int from, to, cost;
6 };
7 Edge E[maxn * maxn], e[maxn * maxn];
8 int n, m, c;
9 int in[maxn], pre[maxn], id[maxn], vis[maxn];
10 int CLE(int root, int n, int m)
11 {
12     int res = 0;
13     while(1)
14     {
15         for(int i = 0; i < n; i++){
16             in[i] = INF;
17         }
18         //Find in edge
19         for(int i = 0; i < m; i++){
20             int from = e[i].from, to = e[i].to;
21             if(from != to && e[i].cost < in[to]){
22                 in[to] = e[i].cost;
23                 pre[to] = from;
24             }
25         }

```

```

26 //Check in edge
27 for(int i = 0; i < n; i++){
28     if(i == root) continue;
29     if(in[i] == INF) return -1;
30 }
31
32 int num = 0;
33 memset(id, -1, sizeof(id));
34 memset(vis, -1, sizeof(vis));
35 in[root] = 0;
36
37 //Find cycles
38 for(int i = 0; i < n; i++){
39     res += in[i];
40     int v = i;
41     while(vis[v] != i && id[v] == -1 && v != root)
42     {
43         vis[v] = i;
44         v = pre[v];
45     }
46     if(v != root && id[v] == -1)
47     {
48         for(int j = pre[v]; j != v; j = pre[j]){
49             id[j] = num;
50         }
51         id[v] = num++;
52     }
53 }
54 //No cycle
55 if(num == 0) break;
56 for(int i = 0; i < n; i++){
57     if(id[i] == -1) id[i] = num++;
58 }
59 //Grouping the vertices
60 for(int i = 0; i < m; i++){
61     int from = e[i].from, to = e[i].to;
62     e[i].from = id[from]; e[i].to = id[to];
63     if(id[from] != id[to]) e[i].cost -= in[to];
64 }
65 n = num;
66 root = id[root];
67 }
68 return res;
69 }
70 int main(int argc, char const *argv[])
71 {
72     int n, m;
73     // n nodes and m edges
74     scanf("%d%d", &n, &m);
75     for(int i = 0; i < m; i++){
76         scanf("%d%d%d", &E[i].from, &E[i].to, &E[i].cost);
77     }
78     int sp = 0; // start point
79     int ans = CLE(sp, n, m);
80     if(ans == -1) printf("No Directed Minimum Spanning Tree.\n");
81     else printf("%d\n", ans);
82     return 0;
83 }

```

8.11 Convex Hull

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 struct point{
5     int x;
6     int y;
7     int d;
8 }p[600],ch[600];
9
10 int dist(point a, point b) {
11     return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
12 }//若點的angle一樣，則比較遠的點

```


9.3 Euler

```

13
14 bool find_small_vertex(point a, point b) {
15     return (a.y < b.y) || (a.y == b.y && a.x < b.x);
16 }
17
18 int cross(point o, point a, point b) {
19     return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x
        - o.x);
20 }
21
22 bool compare_angle(point a, point b){
23     double c = cross( p[0], a, b );
24     if ( !c ) return a.d < b.d;
25     else return c > 0;
26 }
27
28 void GrahamScan(int k){
29     sort(p+0, p+k, find_small_vertex);
30     for(int i=1; i<k; i++){
31         p[i].d = dist(p[0], p[i]);
32     }
33     sort(p+1, p+k, compare_angle);
34
35     int m=0;
36     for(int i=0; i<k; i++){
37         while(m>=2 && cross(ch[m-2], ch[m-1], p[i]) <= 0){
38             m--;
39         }
40         ch[m++] = p[i];
41     }
42     // Convex Hull find m nodes and print them out
43     printf("%d\n", m+1);
44     for(int j=0; j<m; j++){
45         printf("%d %d\n", ch[j].x, ch[j].y);
46     }
47     printf("%d %d\n", ch[0].x, ch[0].y);
48 }

```

```

1 const int maxn = 50000;
2 int F[maxn+5];
3 void Euler(){
4     memset(F, 0, sizeof(F));
5     F[1] = 1;
6     for(int i=2; i<maxn; i++){
7         if(!F[i]){
8             for(int j=i; j<maxn; j+=i){
9                 if(!F[j]) F[j] = j;
10                F[j] = F[j] / i*(i-1);
11            }
12        }
13    }
14 }

```

9 Number

9.1 Sieve

```

1 const int maxn = 500+10;
2 bool visit[maxn];
3 int primes[maxn];
4 int sieve(int src)
5 {
6     memset(visit, false, sizeof(visit));
7     for(int i = 2; i <= sqrt(src + 0.5); i++){
8         if(!visit[i]){
9             for(int j = i * i; j <= src; j += i){
10                visit[j] = true;
11            }
12        }
13    }
14    int cnt = 0;
15    for(int i = 2; i <= src; i++){
16        if(!visit[i]) primes[cnt++] = i;
17    }
18    return cnt;
19 }

```

9.2 Power

```

1 double Power(double x, int n)
2 {
3     if (n == 0) return 1.00;
4     if (n == 1) return x;
5     double ans = Power(x, n / 2);
6     if (n % 2 == 0) return ans * ans;
7     else if (n < 0) return ans * ans / x;
8     else return ans * ans * x;
9 }

```