

Contents

1	Data Structure	
1.1	Disjoint Set
2	Tree	
2.1	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
8	Graph	
8.1	Dijkstra
8.2	BellmanFord
8.3	FloydWarshall
8.4	Kruskal
8.5	Convex Hull

1 Data Structure

1.1 Disjoint Set

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 6; // number of nodes
4 int parent[n+10];
5 void init()
6 {
7     for(int i = 0; i < n; i++){
8         parent[i] = -1;
9     }
10 }
11 int find(int x)
12 {
13     int xParent = x;
14     while(parent[xParent] >= 0){
15         xParent = parent[xParent];
16     }
17     return xParent;
18 }
19 void unions(int x ,int y)
20 {
21     int xParent = find(x);
22     int yParent = find(y);
23     if(xParent != yParent){
24         if(parent[xParent] > parent[yParent]){
25             parent[xParent] += parent[yParent];
26             parent[yParent] = xParent;
27         }
28         else{
29             parent[yParent] += parent[xParent];
30             parent[xParent] = yParent;
31         }
32     }
33 }
```

2 Tree

2.1 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)
10 {
11     if(L == R) ST[vertex] = L;
12     else
13     {
14         int nL = vertex * 2, nR = vertex * 2 + 1;
15         ST_Build(ST, A, nL, L, (R - L) / 2);
16         ST_Build(ST, A, nR, L + (R - L) / 2 + 1, R);
17         int indexL = ST[nL], indexR = ST[nR];
18         int valueL = A[indexL], valueR = A[indexR];
19         ST[vertex] = valueL <= valueR ? indexL : indexR;
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 void update(vi &ST, vi &A, int x, int L, int R, int p,
50             int v)
51 {
52     // p is the index where you want to update
53     // v is the value will be update in A[p];
54     int mid = L + (R - L) / 2;
55     if(L == R) A[ST[x]] = v;
56     else
57     {
58         if(p <= mid) update(ST, A, x*2, L, mid, p, v);
59         else update(ST, A, x*2+1, mid+1, R, p, v);
60         ST[x] = (A[ST[x*2]] <= A[ST[x*2+1]]) ? ST[x*2] : ST
61             [x*2+1];
62     }
63 }
64 int main(int argc, char const *argv[])
65 {
66     ST_Creation(ST, A);
67     printf("%d\n", query(ST, A, 1, 0, n-1, 3, 7));
68     // query return the index
69     printf("%d\n", A[query(ST, A, 1, 0, n-1, 3, 7)]);
70     update(ST, A, 1, 0, n-1, 5, 18);
71     // query and update first to fifth parameter dont
72     // change
73     // ST, A, 1, 0, n-1
74     // last two would be
75     // query: the range(array index) you want to query

```

```

68 // update: fisrt is the index you want to update,
   // second is the value will be
69 return 0;
70 }

```

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    return (maxl + maxr);
21 }
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
   // arrp and arrq
2 int arrp[maxn], arrq[maxn];
3 int dp[maxn+5][maxn+5];
4 int p, q; // p is the length of arrp, q is the length
   // of arrq
5 void LCS()
6 {
7     memset(dp, 0, sizeof(dp));
8
9     for(int i = 1; i <= p; i++){
10         for(int j = 1; j <= q; j++){
11             if(arrp[i] == arrq[j]){

```

```

12         dp[i][j] = 1 + dp[i-1][j-1];
13     }
14     else{
15         dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
16     }
17 }
18 }
19 // dp[p][q] is the answer
20 }

```

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
   // whether the M can qualify the demand
7     else R = M - 1;
8 }
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
   // whether the M can qualify the demand
13     else L = M + 1;
14 }

```

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,
   // include it self
9     // example query[3] = arr[0] + arr[1] + arr[2] + arr
   // [3]
10     query[0] = arr[0];

```

```

11 for(int i = 1; i < maxn; i++){
12     query[i] = arr[i];
13     query[i] += query[i-1];
14 }
15 }
16 int RangeSumQuery(int s, int e)
17 {
18     //Prefix Sum Algorithm
19     if(s >= 1) return query[e] - query[s-1];
20     else return query[e];
21 }
22 int main(int argc, char const *argv[])
23 {
24     init();
25     int start = 2, end = 5;
26     printf("RangeSumQuery(%d, %d): %d\n", start, end,
27         RangeSumQuery(start, end));
28     return 0;
29 }

```

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             // bulid prefix query
16             for(int i = 0; i < n; i++){
17                 for(int j = 0; j < n; j++){
18                     query[i][j] = arr[i][j];
19                     if(i - 1 >= 0) query[i][j] += query[i-1][j];
20                     if(j - 1 >= 0) query[i][j] += query[i][j-1];
21                     if(i - 1 >= 0 && j - 1 >= 0) query[i][j] -=
22                         query[i-1][j-1];
23                 }
24             }
25             int temp;
26             int maximum = 0x80000000;
27             // find the maximum sum in any range
28             for(int i = 0; i < n; i++){
29                 for(int j = 0; j < n; j++){
30                     for(int k = i; k < n; k++){
31                         for(int t = j; t < n; t++){
32                             temp = query[k][t];
33                             if(i - 1 >= 0) temp -= query[i-1][t];
34                             if(j - 1 >= 0) temp -= query[k][j-1];
35                             if(i - 1 >= 0 && j - 1 >= 0) temp += query[
36                                 i-1][j-1];
37                             if(maximum < temp) maximum = temp;
38                         }
39                     }
40                 }
41             }
42             printf("%d\n", maximum);
43         }
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     memset(FenwickTree, 0, sizeof(FenwickTree));
15     // Notice that we start in 1
16     for(int i = 1; i <= maxn; i++){
17         int index = i;
18         FenwickTree[i] += arr[i-1];
19         int temp = arr[i-1];
20         while(index + ANDlowbit(index) <= maxn){
21             index += ANDlowbit(index);
22             FenwickTree[index] += temp;
23         }
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
  [maxDigit];
6 // aux size is depends on the max digit in sorting
7 int main(int argc, char const *argv[])
8 {
9     int n = 4;
10    // array index start with 1
11    memset(aux, 0, sizeof(aux));
12    for(int i = 1; i <= n; i++){
13        aux[unsorted[i]]++;
14    }
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 }

```

8 Graph

8.1 Dijkstra

```

1 int maxn = ;
2 int w[maxn][maxn], dis[maxn];
3 vector<int> v[maxn];
4 struct Node
5 {
6     int node, weight;
7     Node(int _n, int _w){
8         node = _n;
9         weight = _w;
10    }
11    bool operator<(Node const other) const{
12        return weight > other.weight;
13    }
14 };
15 void dijkstra(int src)
16 {
17     priority_queue<Node> pq;
18     pq.push(Node(src, 0));
19     while(!pq.empty())
20     {
21         auto top = pq.top();
22         pq.pop();
23         if(dis[top.node] != 1e9) continue;
24         for(auto i : v[top.node]){
25             pq.push(Node(i, top.weight + w[top.node][i]));
26         }
27         dis[top.node] = top.weight;
28     }
29 }

```

8.2 BellmanFord

```

1 int main(int argc, char const *argv[])
2 {
3     //initialize dis[] with 1e9
4     //make an adjacent list
5     call bellman_ford(src);
6     return 0;
7 }
8
9 void bellman_ford(int src)
10 {
11     dis[src] = 0; //initialize source
12     //with distance 0
13     for (int k = 0; k < n - 1; ++k){ //do n-1
14         //times
15         for (int i = 0; i < n; ++i){
16             for(auto j : v[i]){
17                 if(dis[i] != 1e9) dis[j] = min(dis[j], dis[i] +
18                     w[i][j]);
19             }
20         }
21     }
22 }
23 bool negativeCycle()
24 {
25     for(i = 0; i < n; ++i){
26         for(auto j : v[i]){
27             if(dis[j] > dis[i] + w[i][j]) return true //has
28                 negative cycle
29         }
30     }
31     return false;
32 }

```

8.3 FloydWarshall

```

1 //dis[i][j] is the distance of node i to node j
2 for (int k = 0; k < n; ++k)
3     for(int i = 0; i < n; ++i)
4         for(int j = 0; j < n; ++j)
5             dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);

```

8.4 Kruskal

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct Edge
4 {
5     int from, to, weight;
6     bool operator< (const Edge other) const{
7         return weight < other.weight;
8     }
9 };
10 int n, m; // n is number of nodes, m is number of edges
11 Edge edge[25000+10];
12 int parent[1000+10];
13 void init()
14 {
15     for(int i = 0; i < n; i++){
16         parent[i] = -1;
17     }
18 }
19 int find(int x)
20 {
21     int xParent = x;
22     while(parent[xParent] >= 0){
23         xParent = parent[xParent];
24     }
25     return xParent;
26 }
27 bool connect(int x, int y)
28 {
29     int xParent = find(x);
30     int yParent = find(y);
31     if(xParent != yParent){
32         parent[xParent] = yParent;
33         return true;
34     }
35     else return false;
36 }
37 int main(int argc, char const *argv[])
38 {
39     while(cin >> n >> m)
40     {
41         if(n == 0 && m == 0) break;
42         for(int i = 0; i < m; i++){
43             cin >> edge[i].from >> edge[i].to >> edge[i].weight;
44         }
45         init();
46         sort(edge, edge + m); // Kruskal need to sort the
47                                // edge by thier weight
48         int minCost = 0; // minimum spanning tree cost
49         for(int i = 0; i < m; i++){
50             if(connect(edge[i].from, edge[i].to)){
51                 minCost += edge[i].weight;
52             }
53         }
54         printf("%d\n", minCost);
55     }
56     return 0;
57 }

```

8.5 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
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
20         - o.x);
21 }
22
23 bool compare_angle(point a, point b){
24     double c = cross( p[0], a, b );
25     if ( !c ) return a.d < b.d;
26     else return c > 0;
27 }
28
29 void GrahamScan(int k){
30     sort(p+0, p+k, find_small_vertex);
31     for(int i=1; i<k; i++){
32         p[i].d = dist(p[0], p[i]);
33     }
34     sort(p+1, p+k, compare_angle);
35
36     int m=0;
37     for(int i=0; i<k; i++){
38         while(m>=2 && cross(ch[m-2], ch[m-1], p[i]) <= 0){
39             m--;
40         }
41         ch[m++] = p[i];
42     }
43     // Convex Hull find m nodes and print them out
44     printf("%d\n", m+1);
45     for(int j=0; j<m; j++){
46         printf("%d %d\n", ch[j].x, ch[j].y);
47     }
48     printf("%d %d\n", ch[0].x, ch[0].y);
49 }

```