Contents

1 Data Structure

1.1 Disjoint Set

```cpp
#include <bits/stdc++.h>
using namespace std;
const int n = 6; // number of nodes
int parent[n+10];
void init()
{
  for(int i = 0; i < n; i ++){
    parent[i] = -1;
  }
}
int find(int x)
{
  int xParent = x;
  while(parent[xParent] >= 0){
    xParent = parent[xParent];
  }
  return xParent;
}
void unions(int x ,int y)
{
  int xParent = find(x);
  int yParent = find(y);
  if(xParent != yParent){
    if(parent[xParent] > parent[yParent]){
      parent[xParent] += parent[yParent];
      parent[yParent] = xParent;
    }
    else{
      parent[yParent] += parent[xParent];
      parent[xParent] = yParent;
    }
  }
}
```

2 Tree

2.1 Segment Tree

```cpp
#include <bits/stdc++.h>
using namespace std;
const int n = 8;
int B[n] = {18, 17, 13, 19, 15, 11, 20, 87};
typedef vector<int> vi;
vi A (B, B + 8);
vi ST;
void ST_Build(vi &ST, const vi &A, int vertex, int L,
    int R)
{
  if(L == R) ST[vertex] = L;
  else
  {
    int nL = vertex * 2, nR = vertex * 2 + 1;
    ST_Build(ST, A, nL, L, L + (R - L) / 2);
    ST_Build(ST, A, nR, L + (R - L) / 2 + 1, R);
    int indexL = ST[nL], indexR = ST[nR];
    int valueL = A[indexL], valueR = A[indexR];
    ST[vertex] = valueL <= valueR ? indexL : indexR;
  }
}

void ST_Creation(vi &ST, const vi &A)
{
  int len = 4 * A.size();
  ST.assign(len, 0);
  ST_Build(ST, A, 1, 0, A.size()-1);
}
int query(vi &ST, const vi &A, int vertex, int L, int R
    , int qL, int qR)
{
  int temp, mid = (L + R) / 2;
  if(qL <= L && R <= qR) return ST[vertex];
  if(qR <= mid)
  { //all we want at the left child
    return query(ST, A, vertex * 2, L, mid, qL, qR);
  }
  if(qL > mid)
  { // all we want at the right child
    return query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
        qR);
  }
  return A[query(ST, A, vertex * 2, L, mid, qL, qR)] <=
      A[query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
      qR)]
      ? query(ST, A, vertex * 2, L, mid, qL, qR) :
          query(ST, A, vertex * 2 + 1, mid + 1, R, qL,
          qR);

}
void update(vi &ST, vi &A, int x, int L, int R, int p,
    int v)
{
  // p is the index where you want to update
  // v is the value will be update in A[p];
  int mid = L + (R - L) / 2;
  if(L == R) A[ST[x]] = v;
  else
  {
    if(p <= mid) update(ST, A, x*2, L, mid, p, v);
    else update(ST, A, x*2+1, mid+1, R, p, v);
    ST[x] = (A[ST[x*2]] <= A[ST[x*2+1]]) ? ST[x*2] : ST
        [x*2+1];
  }
}
int main(int argc, char const *argv[])
{
  ST_Creation(ST, A);
  printf("%d\n", query(ST, A, 1, 0, n-1, 3, 7));
  // query return the index
  printf("%d\n", A[query(ST, A, 1, 0, n-1, 3, 7)]);
  update(ST, A, 1, 0, n-1, 5, 18);
  // query and update first to fifth parameter dont
      change
  // ST, A, 1, 0, n-1
  // last two would be
  // query: the range(array index) you want to query
  // update: fisrt is the index you want to update,
      second is the value will be
  return 0;
}
```

3 Divide and Conquer

3.1 MaximumSubArray

```cpp
#include <bits/stdc++.h>
using namespace std;
const int n = 16;
int arr[n] = {13, -3, -25, 20, -3, -16, -23,
        18, 20, -7, 12, -5, -22, 15, -4, 7};

int findMaxCrosing(int left, int mid, int right){
  int maxl = 0x80000000;
  int sum = 0;
  for(int i = mid; i >= left; i--){
    sum += arr[i];
    if(sum > maxl) maxl = sum;
  }
  int maxr = 0x80000000;
```

```
  sum = 0;
  for(int i = mid + 1; i <= right; i++){
    sum += arr[i];
    if(sum > maxr) maxr = sum;
  }

  return (maxl + maxr);
}

int findMaxSub(int left, int right)
{
  if(left == right){
    return arr[left];
  }
  else{
    int mid = left + (right - left) / 2;
    int maxl = findMaxSub(left, mid);
    int maxr = findMaxSub(mid + 1, right);
    int res = max(maxl, maxr);
    res = max(res, findMaxCrosing(left, mid, right));
    return res;
  }
}


int main(int argc, char const *argv[])
{
  printf("%d\n", findMaxSub(0, n-1));
  return 0;
}
```

4 Dynamic Programming

4.1 LCS

```
const int maxn = 10000; // maxn is maximum length of
    arrp and arrq
int arrp[maxn], arrq[maxn];
int dp[maxn+5][maxn+5];
int p, q; // p is the length of arrp, q is the length
    of arrq
void LCS()
{
  memset(dp, 0, sizeof(dp));

  for(int i = 1; i <= p; i++){
    for(int j = 1; j <= q; j++){
      if(arrp[i] == arrq[j]){
        dp[i][j] = 1 + dp[i-1][j-1];
      }
      else{
        dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
      }
    }
  }
  // dp[p][q] is the answer
}
```

4.2 LIS

```
int LIS(vector<int>& s)
{
    if (s.size() == 0) return 0;

    vector<int> v;
    v.push_back(s[0]);

    for (int i = 1; i < s.size(); ++i)
    {
        int n = s[i];

        if (n > v.back())
            v.push_back(n);
        else
            *lower_bound(v.begin(), v.end(), n) = n;
    }

    return v.size();
}
```

5 Search

5.1 Binary Search

```
int L = 0;     //  left boundary
int R = ans;   // right boundary
// check using L = 3, R = 4, ans = 4
while(L < R){
  int M = L + (R - L + 1) / 2;  // left + half distance
  if(ok(M)) L = M;      // ok() method is to find
      whether the M can qualify the demand
  else R = M - 1;
}

while(L < R){
  int M = L + (R - L) / 2;  // left + half distance
  if(ok(M)) R = M;      // ok() method is to find
      whether the M can qualify the demand
  else L = M + 1;
}
```

6 Sequence

6.1 RSQ(Prefix Sum)

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 10;
int arr[maxn] = {5, -2, 3, 10, -7, 1, -4, 8, -9};
int query[maxn];
void init()
{
  // every query is the sum of all previos element,
      include it self
  // example query[3] = arr[0] + arr[1] + arr[2] + arr
      [3]
  query[0] = arr[0];
  for(int i = 1; i < maxn; i++){
    query[i] = arr[i];
    query[i] += query[i-1];
  }
}
int RangeSumQuery(int s, int e)
{
  //Prefix Sum Algorithm
  if(s >= 1) return query[e] - query[s-1];
  else return query[e];
}
int main(int argc, char const *argv[])
{
  init();
  int start = 2, end = 5;
  printf("RangeSumQuery(%d, %d): %d\n", start, end,
      RangeSumQuery(start, end));

  return 0;
}
```

6.2 RSQ(2DPrefix Sum)

```cpp
#include <bits/stdc++.h>
using namespace std;
int arr[110][110];
int query[110][110];
int n;

int main(int argc, char const *argv[])
{
  while(cin >> n){
    // input
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++)
        cin >> arr[i][j];
    }
    // bulid prefix query
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
        query[i][j] = arr[i][j];
        if(i - 1 >= 0) query[i][j] += query[i-1][j];
        if(j - 1 >= 0) query[i][j] += query[i][j-1];
        if(i - 1 >= 0 && j - 1 >= 0) query[i][j] -=
            query[i-1][j-1];
      }
    }

    int temp;
    int maximum = 0x80000000;
    // find the maximum sum in any range
    for(int i = 0; i < n; i++){
      for(int j = 0; j < n; j++){
        for(int k = i; k < n; k++){
          for(int t = j; t < n; t++){
            temp = query[k][t];
            if(i - 1 >= 0) temp -= query[i-1][t];
            if(j - 1 >= 0) temp -= query[k][j-1];
            if(i - 1 >= 0 && j - 1 >= 0) temp += query[
                i-1][j-1];
            if(maximum < temp) maximum = temp;
          }
        }
      }
    }
    printf("%d\n", maximum);


  }
  return 0;
}
```

6.3 RSQ(Fenwick Tree)

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

  memset(FenwickTree, 0, sizeof(FenwickTree));
  // Notice that we start in 1
  for(int i = 1; i <= maxn; i++){
    int index = i;
    FenwickTree[i] += arr[i-1];
    int temp = arr[i-1];
    while(index + ANDlowbit(index) <= maxn){
      index += ANDlowbit(index);
      FenwickTree[index] += temp;
    }
  }
}
void Modify(int src, int val)
{
  // Modify arr[src] to val
  int gap = val - arr[src];
  arr[src] = val;
  int index = src + 1;
  FenwickTree[index] += gap;
  while(index + ANDlowbit(index) <= maxn){
    index += ANDlowbit(index);
    FenwickTree[index] += gap;
  }
}
int SequenceQuery(int src)
{
  //src is the index of the array which we want to know
       the Sequence Query
  int res = FenwickTree[src];
  int index = src;
  while(index - ANDlowbit(index) > 0){
    index -= ANDlowbit(index);
    res += FenwickTree[index];
  }
  return res;
}
int RangeSumQuery(int s, int e)
{
  return SequenceQuery(e) - SequenceQuery(s - 1);
}
int main(int argc, char const *argv[])
{
  init();
  int start = 2, end = 5;
  // for Fenwick index is 3, 6 for array index is 2, 5
  printf("RangeSumQuery(%d, %d): %d\n", start, end,
      RangeSumQuery(start + 1, end + 1));
  Modify(2, 5);
  // Modify arr[2] from 3 to 5
  printf("RangeSumQuery(%d, %d): %d\n", start, end,
      RangeSumQuery(start + 1, end + 1));
  return 0;
}
```

7 Graph

7.1 Dijkstra

```cpp
int maxn = ;
int w[maxn][maxn], dis[maxn];
vector<int> v[maxn];
struct Node
{
  int node, weight;
  Node(int _n, int _w){
    node = _n;
    weight = _w;
  }
  bool operator<(Node const other)const{
    return weight > other.weight;
  }
};
void dijkstra(int src)
{
  priority_queue<Node> pq;
  pq.push(Node(src, 0));
  while(!pq.empty())
  {
    auto top = pq.top();
    pq.pop();
    if(dis[top.node] != 1e9) continue;
    for(auto i : v[top.node]){
      pq.push(Node(i, top.weight + w[top.node][i]));
    }
    dis[top.node] = top.weight;
```

```
  }
}
```

7.2 BellmanFord

```
int main(int argc, char const *argv[])
{
  //initialize dis[] with 1e9
  //make an adjecnt list
  call bellman_ford(src);
  return 0;
}

void bellman_ford(int src)
{
  dis[src] = 0;                  //initialize source with distance 0
  for (int k = 0; k < n - 1; ++k){        //do n-1 times
    for (int i = 0; i < n; ++i){
      for(auto j : v[i]){
        if(dis[i] != 1e9) dis[j] = min(dis[j], dis[i] + w[i][j]);
      }
    }
   }
}
bool negativeCycle()
{
  for(i = 0; i < n; ++i){
    for(auto j : v[i]){
      if(dis[j] > dis[i] + w[i][j]) return true //has negative cycle
    }
  }
  return false;
}
```

7.3 FloydWarshall

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

7.4 Kruskal

```
#include <bits/stdc++.h>
using namespace std;
struct Edge
{
  int from, to, weight;
  bool operator< (const Edge other) const{
    return weight < other.weight;
  }
};
int n, m; // n is number of nodes, m is number of edges
Edge edge[25000+10];
int parent[1000+10];
void init()
{
  for(int i = 0; i < n; i ++){
    parent[i] = -1;
  }
}
int find(int x)
{
  int xParent = x;
  while(parent[xParent] >= 0){
    xParent = parent[xParent];
  }
  return xParent;
}
bool connect(int x ,int y)
{
  int xParent = find(x);
  int yParent = find(y);
  if(xParent != yParent){
    parent[xParent] = yParent;
    return true;
  }
  else return false;
}
int main(int argc, char const *argv[])
{
  while(cin >> n >> m)
  {
    if(n == 0 && m == 0) break;
    for(int i = 0; i < m; i++){
      cin >> edge[i].from >> edge[i].to >> edge[i].weight;
    }
    init();
    sort(edge, edge + m); // Kruskal need to sort the edge by thier weight
    int minCost = 0; // minimum spanning tree cost
    for(int i = 0; i < m; i++){
      if(connect(edge[i].from, edge[i].to)){
        minCost += edge[i].weight;
      }
    }
    printf("%d\n", minCost);
  }
  return 0;
}
```

7.5 Convex Hull

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

struct point{
  int x;
  int y;
  int d;
}p[600],ch[600];

int dist(point a, point b) {
  return (a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y);
}//若點的angle一樣，則比較遠的點

bool find_small_vertex(point a, point b) {
  return (a.y < b.y) || (a.y == b.y && a.x < b.x);
}

int cross(point o, point a, point b) {
  return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}

bool compare_angle(point a, point b){
  double c = cross( p[0], a, b );
  if ( !c ) return a.d < b.d;
  else return c > 0;
}

void GrahamScan(int k){
  sort(p+0, p+k, find_small_vertex);
  for(int i=1; i<k; i++){
    p[i].d = dist(p[0], p[i]);
  }
  sort(p+1, p+k, compare_angle);

  int m=0;
```

```
  for(int i=0; i<k; i++){
    while(m>=2 && cross(ch[m-2], ch[m-1], p[i]) <= 0){
      m--;
    }
    ch[m++] = p[i];
  }
  // Convex Hull find m nodes and print them out
  printf("%d\n", m+1);
  for(int j=0; j<m; j++){
    printf("%d %d\n", ch[j].x, ch[j].y);
  }
  printf("%d %d\n", ch[0].x, ch[0].y);
}
```