Contents

1 Basic

1.1 Syntax

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(int argc, char const *argv[])
{
  // String to Integer
  char str[30] = {'-', '1', '2', '3', '4', '5', '\0'};
  printf("%d\n", stoi(str));
  // Integer to String
  int x = 185;
  char temp[30];
  int base = 10;
  itoa(x, temp, base);
  printf("%s\n", temp);
  // String to Double
  char strd[30] = {'0', '.', '6', '0', '2', '2', '2', '9', '\0'};
  printf("%lf\n", stod(strd));
  // Double to String
  double y = 3.1415926;
  string dstr = to_string(y);
  cout << dstr << endl;
  // String initialize
  char null[30] = {'\0'};
  char A[30];
  strcpy(A, null);
  // String Length
  char strl[30] = {'H', 'E', 'L', 'L', 'O', '\0'};
  printf("%d\n", strlen(strl));
  return 0;
}
```

1.2 BigInteger

```cpp
import java.math.BigInteger;
import java.util.Scanner;
class Main {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        BigInteger n = input.nextBigInteger();
        BigInteger m = input.nextBigInteger();
        n.add(m); a.subtract(m); n.multiply(m); n.divide(m); n.mod(m);
        n.pow(m.intValue()); n.gcd(m); n.negate(); n.abs();
    }
}
```

2 Data Structure

2.1 Disjoint Set

```cpp
const int n = 6; // number of nodes
int p[n+10];
void init()
{
  for(int i = 0; i < n; i ++){
    p[i] = -1;
  }
}
int find(int x){
  int root, trail, lead;
  for (root = x; p[root] >= 0; root = p[root]);
  for (trail = x; trail != root; trail = lead) {
        lead = p[trail];
        p[trail]= root;
  }
  return root;
}
void uni(int x ,int y)
{
  int xRoot = find(x), yRoot = find(y);
  if(xRoot != yRoot){
    if(p[xRoot] > p[yRoot]){
      p[xRoot] += p[yRoot];
      p[yRoot] = xRoot;
    }
    else{
      p[yRoot] += p[xRoot];
      p[xRoot] = yRoot;
    }
  }
}
```

2.2 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

```
#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)

```
#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)

```
#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 Sorting

7.1 Counting Sort

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 50;
const int maxDigit = 1050;
int unsorted[maxn] = {0, 3, 7, 6, 5}, sorted[maxn], aux
    [maxDigit];
// aux size is depends on the max digit in sorting
int main(int argc, char const *argv[])
{
  int n = 4;
  // array index start with 1
  memset(aux, 0, sizeof(aux));
  for(int i = 1; i <= n; i++){
    aux[unsorted[i]]++;
  }
  for(int i = 1; i < maxDigit; i++){
    aux[i] += aux[i-1];
  }
  for(int i = n; i > 0; i--){
    sorted[aux[unsorted[i]]] = unsorted[i];
    aux[unsorted[i]]--;
  }
  for(int i = 1; i <= n; i++){
    printf("%d ", sorted[i]);
  }
  return 0;
}
```

7.2 Topology Sort

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100;
vector<int> ans;
vector<int> adj[maxn];
int refs[maxn];
int n = 5;

// refs 紀錄這個點被幾個邊連到
void TopologyOrder()
{
  for(int i = 0; i < n; i++){
    int s = 0;
    while(s < n && refs[s] != 0) {
      s++;
    }
    if(s == n) break;
    refs[s] = -1;
    ans.push_back(s);
    for(auto j : adj[s]){
      refs[j]--;
    }
  }
}
int main(int argc, char const *argv[])
{
  memset(refs, 0, sizeof(refs));
  ans.clear();
  // adj[from].push_back(to); refs[to]++;
  adj[4].push_back(1); refs[1]++;
  adj[1].push_back(3); refs[3]++;
  adj[1].push_back(0); refs[0]++;
  adj[2].push_back(0); refs[0]++;
  adj[3].push_back(0); refs[0]++;
  TopologyOrder();
  for(int i = 0; i < ans.size(); i++){
    if(i == ans.size()-1) printf("%d\n", ans[i]);
    else printf("%d ", ans[i]);
  }
  return 0;
}
```

8 Graph

8.1 DFS

```
//implement by adjcent list
//functional dfs
void dfs(int now, int fa, int layer){
  for (auto j : adj[now])
    if(j != fa ) dfs(j, now, layer + 1);
}
//stack dfs
stack<int> st;
bool vis[maxn];
memset(vis, false, sizeof(vis));
int src;
st.push(src);
while(!st.empty())
{
  int now = st.top(); st.pop();
```

```
    vis[now] = true;
    for(auto i : adj[now])
      if(!vis[i]) st.push(i);

}
```

8.2 BFS

```
queue<int> st;
bool vis[maxn];
memset(vis, false, sizeof(vis));
int src;
st.push(src);
while(!st.empty())
{
  int now = st.front(); st.pop();
    vis[now] = true;
  for(auto i : adj[now])
      if(!vis[i]) st.push(i);
}
```

8.3 Dijkstra

```
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;
  }
}
```

8.4 SPFA

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

#define INF 0x3f3f3f3f;
const int maxn = 10000+5;

int n, m;
int dist[maxn], vis[maxn], out[maxn];
vector<pair<int, int>> adj[maxn];

void init()
{
  memset(dist, INF, sizeof(dist));
  memset(vis, 0, sizeof(vis));
  memset(out, 0, sizeof(out));
  for(int i = 0; i <= n;  i++){
    adj[i].clear();
  }
}

bool spfa(int sp, int n)
{
  queue<int> q;
  q.push(sp);

  while(!q.empty())
  {
    int u = q.front(); q.pop();
    vis[u] = 0;
    out[u]++;
    if(out[u] > n) return false; // negative cycle
        occurs

    for(int j = 0; j < adj[u].size(); j++){
      int v = adj[u][j].first;
      if(dist[v] > dist[u] + adj[u][j].second){
        dist[v] = dist[u] + adj[u][j].second;
        if(vis[v]) continue;

        vis[v] = 1;
        q.push(v);
      }
    }
  }
  return true;
}

int main(int argc, char const *argv[])
{
  // n nodes and m edges
  scanf("%d%d", &n, &m);
  init();
  // make adjcent list
  int a, b, w;
  for(int i = 0; i < m; i++){
    scanf("%d%d%d", &a, &b, &w);
    E[a].push_back(make_pair(b, w));
    E[b].push_back(make_pair(a, w));
  }
  int sp = 0; // start point
  dist[sp] = 0; vis[sp] = 1;
  if(spfa(sp, n)) printf("%d\n", dist[n-1]);
  else printf("can't reach.\n");
  return 0;
}
```

8.5 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;
}
```

8.6 FloydWarshall

```
//dis[i][j] is the distance of node i to node j
int dis[n+5][n+5];
void init()
{
  memset(dis, 0x3f, sizeof(dis));
  for(int i = 0; i < n; i++) d[i][i] = 0;
}
void floyd(){
  for (int k = 0; k < n; ++k)
    for(int i = 0; i < n; ++i)
      for(int j = 0; j < n; ++j)
        dis[i][j] = dis[j][i] = min(dis[i][j],  dis[i][k] + dis[k][j]);
}
int main(int argc, char const *argv[])
{
  //If we got n nodes, label from 0 to (n-1)
  init();
  //Set the dis
  floyd();
}
```

8.7 Kruskal

```
const int maxn = 1000+5;
struct Edge
{
  int from, to;
  double cost;
  bool operator<(const Edge other){
    return cost < other.cost;
  }
}E[maxn*maxn];
int p[maxn];
vector<Edge> G[maxn];
int find(int x){
  int root, trail, lead;
  for (root = x ; p[root] >= 0; root = p[root]);
  for (trail = x ; trail != root; trail = lead) {
        lead = p[trail];
        p[trail]= root;
  }
  return root;
}
bool uni(int x ,int y)
{
  int xRoot = find(x), yRoot = find(y);
  if(xRoot != yRoot){
    if(p[xRoot] > p[yRoot]){
      p[xRoot] += p[yRoot];
      p[yRoot] = xRoot;
    }
    else{
      p[yRoot] += p[xRoot];
      p[xRoot] = yRoot;
    }
    return true;
  }
  else return false;
}
double kruskal(int n, int m)
{
  // n is the numbers of node, m is the numbers of edge.
  for(int i = 0; i <= n; i++){
    G[i].clear();
    p[i] = -1;
  }
  sort(E, E + m);
  double ans = 0;
  int edge_cnt = 0;
  for(int i = 0; i < m; i++){
    if(uni(E[i].from, E[i].to)){
      int from = E[i].from, to = E[i].to;
      ans += E[i].cost;
      G[from].push_back(Edge{from, to, E[i].cost});
      G[to].push_back(Edge{to, from, E[i].cost});
      if(++edge_cnt == n-1) break;
    }
  }
  if(edge_cnt == n-1) return ans;
  else return -1;// means can't found spanning tree
}
// find max segment in MST graph
int maxcost[maxn][maxn];
vector<int> visited;
void dfs(int pre, int now, int w){
  for(auto x : visited){
    maxcost[x][now] = maxcost[now][x] = max(w, maxcost[pre][x]);
  }
  visited.push_back(now);
  for(auto i : G[now]){
    if(pre != i.to) dfs(i.from, i.to, i.cost);
  }
}
void findMaxPtah(int sp, int ep){
  memset(maxcost, 0,  sizeof(maxcost));
  visited.clear();
  dfs(-1, sp, 0);
}
```

8.8 Bipartite Matching

```
const int maxn = 500+5;
int W[maxn][maxn], n;
int Lx[maxn], Ly[maxn];
int Lef[maxn];
bool S[maxn], T[maxn];
bool match(int i)
{
  S[i] = true;
  for (int j = 1; j <= n; ++j)
  {
    if(Lx[i] + Ly[j] == W[i][j] && !T[j])
    {
      T[j] = true;
      if(!Lef[j] || match(Lef[j]))
      {
        Lef[j] = i;

        return true;
      }
    }
  }
  return false;
}
void update()
{
  int a = 0x3f3f3f3f;
  for(int i = 1; i <= n; i++)
  {
    if(S[i])
    {
      for(int j = 1; j <= n; j++)
      {
        if(!T[j]) a = min(a, Lx[i] + Ly[j] - W[i][j]);
      }
```

```
    }
  }
  for(int i = 1; i <= n; i++)
  {
    if(S[i]) Lx[i] -= a;
    if(T[i]) Ly[i] += a;
  }
}
void KM()
{
  for (int i = 1; i <= n; ++i)
  {
    Lef[i] = Lx[i] = Ly[i] = 0;
    for(int j = 1; j <= n; j++){
      Lx[i] = max(Lx[i], W[i][j]);
    }
  }
  for (int i = 1; i <= n; ++i)
  {
    for(;;){
      for(int j = 1; j <= n; j++){
        S[j] = T[j] = 0;
      }
      if(match(i)) break;
      else update();
    }


  }
}
int main(int argc, char const *argv[])
{
  for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
      scanf("%d", &W[i][j]);
    }
  }

  KM();
  int ans = 0;

  for(int i = 1; i <= n; i++){
    ans += Ly[i];
    ans += Lx[i];
  }

  for(int i = 1; i <= n; i++){
    if(i != n) printf("%d ", Lx[i]);
    else printf("%d\n", Lx[i]);
  }

  for(int i = 1; i <= n; i++){
    if(i != n) printf("%d ", Ly[i]);
    else printf("%d\n", Ly[i]);
  }

  printf("%d\n", ans);
  return 0;
}
```

8.9 CLE Directed MST

```
const int maxn = 60+5;
const int INF = 0x3f3f3f3f;
struct Edge
{
  int from, to, cost;
};
Edge E[maxn * maxn], e[maxn * maxn];
int n, m, c;
int in[maxn], pre[maxn], id[maxn], vis[maxn];
int CLE(int root, int n, int m)
{
  int res = 0;
  while(1)
  {
    for(int i = 0; i < n; i++){
      in[i] = INF;
    }
    //Find in edge
    for(int i = 0; i < m; i++){
      int from = e[i].from, to = e[i].to;
      if(from != to && e[i].cost < in[to]){
        in[to] = e[i].cost;
      }
      pre[to] = from;
    }
    //Check in edge
    for(int i = 0; i < n; i++){
      if(i == root) continue;
      if(in[i] == INF) return -1;
    }

    int num = 0;
    memset(id, -1, sizeof(id));
    memset(vis, -1, sizeof(vis));
    in[root] = 0;

    //Find cycles
    for(int i = 0; i < n; i++){
      res += in[i];
      int v = i;
      while(vis[v] != i && id[v] == -1 && v != root)
      {
        vis[v] = i;
        v = pre[v];
      }
      if(v != root && id[v] == -1)
      {
        for(int j = pre[v]; j != v; j = pre[j]){
          id[j] = num;
        }
        id[v] = num++;
      }
    }
    //No cycle
    if(num == 0) break;
    for(int i = 0; i < n; i++){
      if(id[i] == -1) id[i] = num++;
    }
    //Grouping the vertices
    for(int i = 0; i < m; i++){
      int from = e[i].from, to = e[i].to;
      e[i].from = id[from]; e[i].to = id[to];
      if(id[from] != id[to]) e[i].cost -= in[to];
    }
    n = num;
    root = id[root];
  }
  return res;
}
int main(int argc, char const *argv[])
{
  int n, m;
  // n nodes and m edges
  scanf("%d%d", &n, &m);
  for(int i = 0; i < m; i++){
    scanf("%d%d%d%d", &E[i].from, &E[i].to, &E[i].cost)
        ;
  }
  int sp = 0; // start point
  int ans = CLE(sp, n, m);
  if(ans == -1) printf("No Directed Minimum Spanning
      Tree.\n");
  else printf("%d\n", ans);
  return 0;
}
```

8.10 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);
}
```

9 Number

9.1 Sieve

```
const int maxn = 500+10;
bool visit[maxn];
int primes[maxn];
int sieve(int src)
{
  memset(visit, false, sizeof(visit));
  for(int i = 2; i <= sqrt(src + 0.5); i++){
    if(!visit[i]){
      for(int j = i * i; j <= src; j += i){
        visit[j] = true;
      }
    }
  }
  int cnt = 0;
  for(int i = 2; i <= src; i++){
    if(!visit[i]) primes[cnt++] = i;
  }
  return cnt;
}
```

9.2 Power

```
double Power(double x, int n)
{
    if (n == 0) return 1.00;
    if (n == 1) return x;
    double ans = Power(x, n / 2);
    if (n % 2 == 0) return ans * ans;
    else if (n < 0) return ans * ans / x;
    else return ans * ans * x;
}
```

9.3 Euler

```
const int maxn = 50000;
int F[maxn+5];
void Euler(){
  memset(F, 0, sizeof(F));
  F[1] = 1;
  for(int i=2; i<maxn; i++){
    if(!F[i]){
      for(int j=i; j<maxn; j+=i){
        if(!F[j]) F[j] = j;
        F[j] = F[j] / i*(i-1);
      }
    }
  }
}
```