Complete Search:

Killed[i][j];//rat attac, see the problem backward

for(int i=x-d ; i<=x+d ; i++)

for(int j=y-d ; j<=y+d ; j++)

killed[i][j]+=w;

==================================

Divide and coquer:

While(hi-low<eps){//bsearch ,(hi==low)

mi=(low+hi)/2;

if(f(mid)>0) low=mi; else hi=mi;

}

==================================

Greedy:

while(num!=n)//work reduction

if((num/2>=n) && (num/2)\*wlin>=wlog)

num/=2; cost+=wlog;

else{ num--; cost+=wlin;

==================================

LIS:

for(int i=0; i<n; i++) {//lis, print path

st.insert(a[i]); it=st.find(a[i]);

if(it!=st.begin()){it--; path[i]=(it->a2); it++;}

//else path[i]=i;

it++;

if(it!=st.end()){st.erase(it);}

i=i;

}

void print(int i){//print path, lis

if(i==0 || i==path[i]){cout << a[i].a1 << endl; retu;}

print(path[i]); cout << a[i].a1 << endl; }

//lis[i]+lds[j]-1;

----------------------------------

for(int i=0 ; i<n ; i++){ dp[i]=1;

for(int j=0 ; j<i ; j++){

if(a[i]>a[j] && dp[i]<dp[j]+1){

dp[i]=dp[j]+1;

} } }

LCS:

dp[MAX][MAX]={0};

for(int i=1 ; i<=n ; i++){

for(int j=1 ; j<=n ; j++){

if(a[i-1]==b[j-1]) dp[i][j]=dp[i-1][j-1]+1;

else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

}}

cout << dp[n][n] << endl;

==================================

# Coin Change

int8 solve(int8 w, int8 coin){//count the ways

if(w<0) return 0; if(w==0) return 1; int8 cnt=0;

for(int8 i=4 ; i>=0 ; i--)

if(c[i]<=c[coin]) cnt+=solve(w-c[i],i);

# Maximum Sum:

//Subrectangle

for(int i=1 ; i<n ; i++)//preprocess

for(int j=0 ; j<n ; j++)

a[i][j]+=a[i-1][j];

int Max=0, ans=0;

for(int k=0 ; k<n ; k++){//calc

for(int i=0 ; i<n-k ; i++){ Max=0;

for(int j=0 ; j<n ; j++){

if(Max<0) Max=a[i+k][j]-a[i][j];

else Max+=a[i+k][j]-a[i][j];

if(Max>ans) ans=Max;

} } }

//sub array, finsh and start point

p ans=p(-1,0,0); int sum=0,id=1;

for(int i=1 ; i<n ; i++){

if(sum<0){sum=0; id=i;}

sum+=a[i];

p tmp=p(sum,id,i+1); ans=Max(ans,tmp);

}

============================================

0-1knapsack:

int calc(int w, int idx){

if(w<0) return INF; if(idx==0) return w;

int Min=min(calc(w-a[i],idx-1),clac(w,idx-1)

return d[w][idx]=Min;

============================================

Non Classical:

int calc(int left, int right){

if(left+1==right) return 0; int ans=INF;

for(int i=left+1 ; i!=right ; i++){

ans=min(ans,calc(left, i)+

calc(i,right)+a[right]-a[left]);

}

Return d[left][right]

==================================

bitmask

bit&(1<<i) // bit i is 0 or 1

bit|(1<<i) // set bit i to 1

bit^(1<<i) // toggle bit i

x & ( x – 1) // check if x is a power of 2

string stmp; bitset<12> tmp; //Debuging

tmp=bit; stmp=tmp.to\_string();

==================================

Dp on Tree:

dp(u,selected){ int ans=0;

if(selected==true)

eoreach v is children u {ans+=dp(v,false)}

else for each v {ans+=max(dp(v,false),dp(v,true)}

Advanced Dp:

ACORN: We Don’t Need To Consider dp[height][tree]

dp[height] is enough

==================================

Euclid Equ:

//base condition for ax+by=c is c%gcd(a,b)==0

void extendedEuclid(int a, int b){

if(b==0){x=1; y=0; d=a; return;}

extendedEuclid(b,a%b);

int x1=y; int y1=x-(a/b)\*y; x=x1; y=y1;

(x1,y1) is the base solution for ax+by=gcd(a,b)

Ax+by=c => (c/gcd(a,b))(ax+by=gcd(a,b)=>correct (x1,y1)

Other solution: x=x1+(b/d)n && y=y1-(a/d)n

positive solutions x>0 && y>0 (above equs)

==================================

Catalan:

Catalan(n+1)=(catalan(n)\*(2n+2)\*(2n+1))/((n+1)\*(n+2))

/\*n!\*/Catalan(n)=2n!/(n!\*n!\*(n+1)), Catalan(1)=1;

000 001 002 003 004 005 006

001 001 002 005 014 042 132

WRONG ANSWER:

============================================

Never Use float, Mybe you must use unsigned char or unsigned long long

--------------------------------------------

Never Initialize variables outside of main loop

--------------------------------------------

Harja cin debug mizari Yeja benevis, check kon Aya pakeshon mikoni bara judge kardan.

--------------------------------------------

Size Araye Ro 10ta Bishtar Az Chizi Ke Niaze Begir (runtime)

--------------------------------------------

Format of Endline, Last Endline or Endline in between

Check Kon to Soal Gofte Blank between ya blank after

--------------------------------------------

Check kon Baraye Halataii ke bayad 0 ya minimum halat chap koni,

--------------------------------------------

for(int cnt=1; cnt<n && cnt!=a[i] ; cnt++)

vaghti index Az 1 Shoro Meshe, Bayad Ta Khode N bere,

for(int cnt=1; cnt=<n && cnt!=a[i] ; cnt++)

--------------------------------------------

while(cin >> qs >> qlev && qs!=0 && qlev!=0){

input terminates with 0 0;

while(cin >> qs >> qlev){

if(qs==0 && qlev==0) break;

--------------------------------------------

while(cin >> n >> m){

if( !n && !m ) break;

while(cin >> n >> m && n && m){

TIMELIMIT:

--------------------------------------------

Use introsort, bitset,

--------------------------------------------

for(i=n-1 ; i>0 ;)

for(j=i-1; j>=0 && a[i]>0 ;)

if(a[i]>a[j]){ a[j]=0; j--; }

Age Bana Be Dalaieli Fore Dovomi Ejra Nashe (j<0 || a[i]==0) Mioftim To Loop Binahaiat;

for(i=n-1 ; i>0 && j>=0 ;){

CPP TECHNIQUES:

//swipping array elements (1 2 3) => (3 2 1)

for(int j=0 ; j<=n/2 ; j++) swap(stack[j],stack[n-j])

//next permution, cmp

while(next\_permutation(str.begin(),str.end(),cmp))

//sort alphbatically

class cmp{public:

bool operator()(string a, string b){

for(int i=0 ; i<min(a.length(),b.length()) ; i++){

if(tolower(a[i])!=tolower(b[i])) return tolower(a[i])<tolower(b[i]);

if(a[i]!=b[i]) return a[i]<b[i];

}

return 0; }};

if(a[i]<=0){i--; continue;}

POINTS:

For getting a matrix of characters that bounds are doesn’t specify in problem use getline(cin,string) and sstream;

--------------------------------------------

Avale kar hame rah haii ke be zehnet mirese ro benevis ba’d ke yeki az rahhat be bombast khor az rah haye dg estefade kon,

Test Cases:

Don’t forget same testing, (1,2),(1,2)

--------------------------------------------

Lower bounds of test cases:

for(int i=0 ; i<v.size() ; i++){

if(i!=0) cout << " ";

cout << v[i]; }

if(v.size()==0) cout << 0 << endl;

OTHER:

--------------------------------------------

Shortest path on Tree:

dist[v][i]+dist[v][j]–2\*dist[v][LCA(i, j)]

--------------------------------------------

ax+by=c, d=GCD(a,b), d|c==0:

int x,y,d;

void extendedEuclid(int a, int b){

if(b==0){x=1; y=0; d=a; return;}

extendedEuclid(b,a%b);

int x1=y; int y1=x-(a/b)\*y;

x=x1; y=y1;

}

============================================

Fenwick Tree

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

#define LOGSZ 17

#define TREE\_SIZE (1<<LOGSZ)

#define T\_TYPE int

T\_TYPE tree[TREE\_SIZE + 10];//start index is 1

// add new\_val to value at pos

void fen\_set(int pos , int new\_val)

{

while(pos <= TREE\_SIZE){

tree[pos] += new\_val;

pos += (pos & -pos);

}

}

//get cumulative sum up to and ((including)) pos

T\_TYPE fen\_get(int pos)

{

T\_TYPE total = 0;

while(pos){

total += tree[pos];

pos -= (pos & -pos);

}

return total;

}

// get largest value with cumulative sum less than or equal to x;

// for smallest, pass x-1 and add 1 to result

int getind(int x) {

int idx = 0, mask = TREE\_SIZE; //(must be a power of 2)

while(mask && idx < TREE\_SIZE) {

int t = idx + mask;

if(x >= tree[t]) {

idx = t;

x -= tree[t];

}

mask >>= 1;

}

return idx;

}

int main()

{

fill(tree,tree+TREE\_SIZE + 1 , 0);

for(int i = 1 ; i <= 20 ; ++i)

fen\_set(i,i);

for(int i = 1 ; i <= 20 ; ++i)

cout << fen\_get(i) << endl;

cout << "=======================" << endl;

cout << getind(54) << endl;

return 0;

}

LCA

#include<vector>

#include<iostream>

using namespace std;

#define MAX\_LOG 10

#define MAX\_NODE (1<<MAX\_LOG)

vector<int> children[MAX\_NODE];

int A[MAX\_NODE][MAX\_LOG+1]; // A[i][j] is the

int L[MAX\_NODE]; // L[i] is the distance between node i and the root

int num\_nodes, log\_num\_nodes, root;

int lb(unsigned int n)

{

if(n==0)

return -1;

int p = 0;

if (n >= 1<<16) { n >>= 16; p += 16; }

if (n >= 1<< 8) { n >>= 8; p += 8; }

if (n >= 1<< 4) { n >>= 4; p += 4; }

if (n >= 1<< 2) { n >>= 2; p += 2; }

if (n >= 1<< 1) { p += 1; }

return p;

}

void DFS(int i, int l)

{

L[i] = l;

for(int j = 0; j < children[i].size(); j++)

DFS(children[i][j], l+1);

}

int LCA(int p, int q)

{

// ensure node p is at least as deep as node q

if(L[p] < L[q])

swap(p, q);

// "binary search" for the ancestor of node p situated on the same level as q

for(int i = log\_num\_nodes; i >= 0; i--)

if(L[p] - (1<<i) >= L[q])

p = A[p][i];

if(p == q)

return p;

// "binary search" for the LCA

for(int i = log\_num\_nodes; i >= 0; i--)

if(A[p][i] != -1 && A[p][i] != A[q][i])

{

p = A[p][i];

q = A[q][i];

}

return A[p][0];

}

int main(int argc,char\* argv[])

{

// read num\_nodes, the total number of nodes

log\_num\_nodes=lb(num\_nodes);

for(int i = 0; i < num\_nodes; i++)

{

int p;

A[i][0] = p;

if(p != -1)

children[p].push\_back(i);

else

root = i;

}

// precompute A using dynamic programming

for(int j = 1; j <= log\_num\_nodes; j++)

for(int i = 0; i < num\_nodes; i++)

if(A[i][j-1] != -1)

A[i][j] = A[A[i][j-1]][j-1];

else

A[i][j] = -1;

// precompute L

DFS(root, 0);

return 0;

}

KD-Tree

// A straightforward, but probably sub-optimal KD-tree implmentation that's

// probably good enough for most things (current it's a 2D-tree)

//

// - constructs from n points in O(n lg^2 n) time

// - handles nearest-neighbor query in O(lg n) if points are well distributed

// - worst case for nearest-neighbor may be linear in pathological case

#include <iostream>

#include <vector>

#include <limits>

#include <cstdlib>

#include<algorithm>

using namespace std;

// number type for coordinates, and its maximum value

typedef long long ntype;

const ntype sentry = numeric\_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D

struct point {

ntype x, y;

point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}

};

bool operator==(const point &a, const point &b)

{

return a.x == b.x && a.y == b.y;

}

// sorts points on x-coordinate

bool on\_x(const point &a, const point &b)

{

return a.x < b.x;

}

// sorts points on y-coordinate

bool on\_y(const point &a, const point &b)

{

return a.y < b.y;

}

// squared distance between points

ntype pdist2(const point &a, const point &b)

{

ntype dx = a.x-b.x, dy = a.y-b.y;

return dx\*dx + dy\*dy;

}

// bounding box for a set of points

struct bbox

{

ntype x0, x1, y0, y1;

bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

// computes bounding box from a bunch of points

void compute(const vector<point> &v) {

for (int i = 0; i < v.size(); ++i) {

x0 = min(x0, v[i].x); x1 = max(x1, v[i].x);

y0 = min(y0, v[i].y); y1 = max(y1, v[i].y);

}

}

// squared distance between a point and this bbox, 0 if inside

ntype distance(const point &p) {

if (p.x < x0) {

if (p.y < y0) return pdist2(point(x0, y0), p);

else if (p.y > y1) return pdist2(point(x0, y1), p);

else return pdist2(point(x0, p.y), p);

}

else if (p.x > x1) {

if (p.y < y0) return pdist2(point(x1, y0), p);

else if (p.y > y1) return pdist2(point(x1, y1), p);

else return pdist2(point(x1, p.y), p);

}

else {

if (p.y < y0) return pdist2(point(p.x, y0), p);

else if (p.y > y1) return pdist2(point(p.x, y1), p);

else return 0;

}

}

};

// stores a single node of the kd-tree, either internal or leaf

struct kdnode

{

bool leaf; // true if this is a leaf node (has one point)

point pt; // the single point of this is a leaf

bbox bound; // bounding box for set of points in children

kdnode \*first, \*second; // two children of this kd-node

kdnode() : leaf(false), first(0), second(0) {}

~kdnode() { if (first) delete first; if (second) delete second; }

// intersect a point with this node (returns squared distance)

ntype intersect(const point &p) {

return bound.distance(p);

}

// recursively builds a kd-tree from a given cloud of points

void construct(vector<point> &vp)

{

// compute bounding box for points at this node

bound.compute(vp);

// if we're down to one point, then we're a leaf node

if (vp.size() == 1) {

leaf = true;

pt = vp[0];

}

else {

// split on x if the bbox is wider than high (not best heuristic...)

if (bound.x1-bound.x0 >= bound.y1-bound.y0)

sort(vp.begin(), vp.end(), on\_x);

// otherwise split on y-coordinate

else

sort(vp.begin(), vp.end(), on\_y);

// divide by taking half the array for each child

// (not best performance if many duplicates in the middle)

int half = vp.size()/2;

vector<point> vl(vp.begin(), vp.begin()+half);

vector<point> vr(vp.begin()+half, vp.end());

first = new kdnode(); first->construct(vl);

second = new kdnode(); second->construct(vr);

}

}

};

// simple kd-tree class to hold the tree and handle queries

struct kdtree

{

kdnode \*root;

// constructs a kd-tree from a points (copied here, as it sorts them)

kdtree(const vector<point> &vp) {

vector<point> v(vp.begin(), vp.end());

root = new kdnode();

root->construct(v);

}

~kdtree() { delete root; }

// recursive search method returns squared distance to nearest point

ntype search(kdnode \*node, const point &p)

{

if (node->leaf) {

// commented special case tells a point not to find itself

// if (p == node->pt) return sentry;

// else

return pdist2(p, node->pt);

}

ntype bfirst = node->first->intersect(p);

ntype bsecond = node->second->intersect(p);

// choose the side with the closest bounding box to search first

// (note that the other side is also searched if needed)

if (bfirst < bsecond) {

ntype best = search(node->first, p);

if (bsecond < best)

best = min(best, search(node->second, p));

return best;

}

else {

ntype best = search(node->second, p);

if (bfirst < best)

best = min(best, search(node->first, p));

return best;

}

}

// squared distance to the nearest

ntype nearest(const point &p) {

return search(root, p);

}

}

// --------------------------------------------------------------------------

// some basic test code here

int main()

{

// generate some random points for a kd-tree

vector<point> vp;

for (int i = 0; i < 10000; ++i) {

vp.push\_back(point(rand()%100000, rand()%100000));

}

kdtree tree(vp);

// query some points

for (int i = 0; i < 10; ++i) {

point q(rand()%100000, rand()%100000);

cout << "Closest squared distance to (" << q.x << ", " << q.y << ")"

<< " is " << tree.nearest(q) << endl;

}

return 0;

}

# Segment Tree

\* Operation #1: Increment the elements within range [i, j] with value val

\* Operation #2: Get max element within range [i, j]

\* Build tree: build\_tree(1, 0, N-1)

\* Update tree: update\_tree(1, 0, N-1, i, j, value)

\* Query tree: query\_tree(1, 0, N-1, i, j)

\* Actual space required by the tree = 2\*2^ceil(log\_2(n)) - 1

\*/

#include<iostream>

#include<algorithm>

using namespace std;

#include<string.h>

#include<math.h>

#define N 20

#define MAX (1+(1<<6)) // Why? :D

#define inf 0x7fffffff

int arr[N];

int tree[MAX];

int lazy[MAX];

void build\_tree(int node, int a, int b) {

if(a > b) return; // Out of range

if(a == b) { // Leaf node

tree[node] = arr[a]; // Init value

return;

}

build\_tree(node\*2, a, (a+b)/2); // Init left child

build\_tree(node\*2+1, 1+(a+b)/2, b); // Init right child

tree[node] = max(tree[node\*2], tree[node\*2+1]); // Init root value

}

/\*\*

\* Increment elements within range [i, j] with value value

\*/

void update\_tree(int node, int a, int b, int i, int j, int value) {

if(lazy[node] != 0) { // This node needs to be updated

tree[node] += lazy[node]; // Update it

if(a != b) {

lazy[node\*2] += lazy[node]; // Mark child as lazy

lazy[node\*2+1] += lazy[node]; // Mark child as lazy

}

lazy[node] = 0; // Reset it

}

if(a > b || a > j || b < i) // Current segment is not within range [i, j]

return;

if(a >= i && b <= j) { // Segment is fully within range

tree[node] += value;

if(a != b) { // Not leaf node

lazy[node\*2] += value;

lazy[node\*2+1] += value;

}

return;

}

update\_tree(node\*2, a, (a+b)/2, i, j, value); // Updating left child

update\_tree(1+node\*2, 1+(a+b)/2, b, i, j, value); // Updating right child

tree[node] = max(tree[node\*2], tree[node\*2+1]); // Updating root with max value

}

/\*\*

\* Query tree to get max element value within range [i, j]

\*/

int query\_tree(int node, int a, int b, int i, int j) {

if(a > b || a > j || b < i) return -inf; // Out of range

if(lazy[node] != 0) { // This node needs to be updated

tree[node] += lazy[node]; // Update it

if(a != b) {

lazy[node\*2] += lazy[node]; // Mark child as lazy

lazy[node\*2+1] += lazy[node]; // Mark child as lazy

}

lazy[node] = 0; // Reset it

}

if(a >= i && b <= j) // Current segment is totally within range [i, j]

return tree[node];

int q1 = query\_tree(node\*2, a, (a+b)/2, i, j); // Query left child

int q2 = query\_tree(1+node\*2, 1+(a+b)/2, b, i, j); // Query right child

int res = max(q1, q2); // Return final result

return res;

}

int main() {

for(int i = 0; i < N; i++) arr[i] = 1;

build\_tree(1, 0, N-1);

memset(lazy, 0, sizeof lazy);

update\_tree(1, 0, N-1, 0, 6, 5); // Increment range [0, 6] by 5. here 0, N-1 represent the current range.

update\_tree(1, 0, N-1, 7, 10, 12);

update\_tree(1, 0, N-1, 10, N-1, 100);

cout << query\_tree(1, 0, N-1, 0, N-1) << endl; // Get max element in range [0, N-1]

return 0;

}

Bellman-Ford

#include<iostream>

#include<vector>

using namespace std;

#define FR(i,s,e) for(int i =(s);i<= (e); ++i)

#define MAX\_VER 1000

#define INVALID 9999999

#define INVALID\_PAR -1

#define INIT 0

#define Dist\_t int

#define weight\_t int

#define Edge pair<int,weight\_t>

struct Node{

int id;

//additional info

};

vector<Dist\_t> dist(MAX\_VER);

vector<bool> on\_negCycle(MAX\_VER);

vector<Edge> adjlist[MAX\_VER];

vector<int> parent(MAX\_VER);//SP Tree

void BF\_DFS(int s)

{

on\_negCycle[s] = true;

for(vector<Edge>::iterator it = adjlist[s].begin() ; it != adjlist[s].end() ; ++it){

int v = it->first;

if(!on\_negCycle[v]){

BF\_DFS(v);

}

}

}

//Change to meet problem needs

bool Relax(int u,int v,int w)

{

if(dist[u] == INVALID)//first node is still invalid

return false;

if(dist[v] == INVALID){

dist[v] = dist[u] + w;

parent[v] = u;

return true;

}

if(dist[v] > dist[u] + w){

dist[v] = dist[u] + w;

parent[v] = u;

return true;

}

return false;

}

//if graph contains negative cycles reachable from s

//it will mark all nodes on neg. cycles & return false

//otherwise it will return true

//O(VE)

bool BellmanFord(int s,int v\_num)

{

dist[s] = (Dist\_t)INIT;

bool well\_path = true;

FR(iter,1,v\_num - 1){

FR(u,0,v\_num - 1){

for(vector<Edge>::iterator e\_it = adjlist[u].begin();e\_it!=adjlist[u].end() ; ++e\_it){

int v = e\_it->first;

weight\_t w = e\_it->second;//use additional info

if(Relax(u,v,w)){

//do sth => usually nothing is needed to be done

}

else if(dist[v] == dist[u] + w){

//problem dependant for example to count or optimize sth

}

}

}

}

//check for neg cycles

on\_negCycle.resize(v\_num);

fill(on\_negCycle.begin(),on\_negCycle.end(),false);

FR(u,0,v\_num - 1){

for(vector<Edge>::iterator e\_it = adjlist[u].begin() ; e\_it != adjlist[u].end() ; ++e\_it){

int v = e\_it->first;

weight\_t w = e\_it->second;

//if the graph is undirected then be carful about negative edges

// which will cause neg. cycle between two endpoints of the edge

if(Relax(u,v,w)){

if(!on\_negCycle[v]){

BF\_DFS(v);//mark all nodes on the cycle

}

well\_path = false;

}

}

}

return well\_path;

}

int main()

{

int v,e;

cin >> v >> e;

//init graph & tables

FR(i,0,v-1) adjlist[i].clear();

dist.resize(v);

fill(dist.begin(),dist.end(),INVALID);

parent.resize(v);

fill(parent.begin(),parent.end(),INVALID\_PAR);

//reading the graph

FR(i,1,e){

int x,y,w;

cin >> x >> y >> w;

//for bidirectional edge push (y,x) either

adjlist[x].push\_back(Edge(y,w));

}

bool ok = BellmanFord(0,v);

return 0;

}

Dijkstra

#include<vector>

#include<set>

#include<iostream>

#include<algorithm>

using namespace std;

#define FR(i,s,e) for(int i = (s) ; i <= (e) ; ++i)

#define weight\_t int

#define DIST int

#define Edge pair<int,weight\_t>

#define MAX\_VER 10000

#define INVALID 99999999

#define INIT\_VAL 0

#define INVALID\_PAR -1

struct Node{

int id;

//additional info

};

vector<Node> node\_list;

vector<Edge> adjlist[MAX\_VER];

vector<int> parent;//reconstruct the path => SP Tree

vector<DIST> dist;//distance from source

//O(ElgV)

void Dijkstra(int s)

{

set<pair<DIST,int>> pq;

dist[s] = (DIST)INIT\_VAL;

parent[s] = INVALID\_PAR;

pq.insert(pair<DIST,int>(dist[s],s));

while(!pq.empty()){

DIST d\_u = pq.begin()->first;

int u = pq.begin()->second;

pq.erase(pq.begin());

for(vector<Edge>::iterator it = adjlist[u].begin() ; it != adjlist[u].end() ; ++it){

DIST v = it->first;

weight\_t w = it->second;//problem dependant => use additional info

if(dist[v] == INVALID){//first visit of a node

dist[v] = d\_u + w;

parent[v] = u;

pq.insert(pair<DIST,int>(dist[v],v));

}

else if(d\_u + w < dist[v]){

pq.erase(pair<DIST,int>(dist[v],v));

dist[v] = d\_u + w;

parent[v] = u;

pq.insert(pair<DIST,int>(dist[v],v));

}

else if(d\_u + w == dist[v]){

//problem dependant!!!

//for example for counting or optimizing sth

}

}

}

}

int main()

{

int v,e;

cin >> v >> e;

//zero indexed

FR(i,0,v-1) adjlist[i].clear();

FR(i,1,e){

int x,y,w;

cin >> x >> y >> w;

adjlist[x].push\_back(Edge(y,w));

}

//Init.

parent.resize(v);

dist.resize(v);

fill(dist.begin(),dist.end(),INVALID);

Dijkstra(0);

return 0;

}

Slow Prim

#define FR(i,s,e) for(int i=(s);i<= (e) ; ++i)

#define MAX\_V 1000

#define weight\_t int

#define Edge\_P pair<int,weight\_t>

#define INF\_EDGE 99999999

weight\_t adjmtx[MAX\_V][MAX\_V];

vector<Edge\_P> mst\_parent;

//O(V^2) => for small problems

weight\_t Slow\_Prim(int root,int v\_num)

{

weight\_t mst\_weight = 0;

vector<weight\_t> dist(v\_num);

fill(dist.begin(),dist.end(),INF\_EDGE);

vector<bool> in\_tree(v\_num);

fill(in\_tree.begin(),in\_tree.end(),false);

mst\_parent.resize(v\_num);

//fill parent

dist[root] = 0;

mst\_parent[root] = Edge\_P(-1,0);

int selected\_node = 0;

while(true){

//min. selection

int min\_index = -1;

weight\_t min\_dist = INF\_EDGE;

FR(i,0,v\_num-1){

if(!in\_tree[i] && dist[i] < min\_dist){

min\_dist = dist[i];

min\_index = i;

}

}

if(min\_index == -1) break;

in\_tree[min\_index] = true;

++selected\_node;

mst\_weight += mst\_parent[min\_index].second;

if(selected\_node == v\_num) break;//tree is compelete

//updating the neighbors

FR(i,0,v\_num-1){

if(!in\_tree[i] && adjmtx[min\_index][i] != INF\_EDGE){

if(dist[i] > adjmtx[min\_index][i]){

dist[i] = adjmtx[min\_index][i];

mst\_parent[i] = Edge\_P(min\_index,adjmtx[min\_index][i]);

}

}

}

}

//assert(selected\_node == v\_num);

return mst\_weight;

}

Dinic

// Adjacency list implementation of Dinic's blocking flow algorithm.

// This is very fast in practice, and only loses to push-relabel flow.

// Running time:

// O(|V|^2 |E|)

// INPUT:

// - graph, constructed using AddEdge()

// - source

// - sink

// OUTPUT:

// - maximum flow value

// - To obtain the actual flow values, look at all edges with

// capacity > 0 (zero capacity edges are residual edges).

#include <cmath>

#include <vector>

#include <iostream>

#include <queue>

using namespace std;

const int INF = 2000000000;

struct Edge {

int from, to, cap, flow, index;

Edge(int from, int to, int cap, int flow, int index) :

from(from), to(to), cap(cap), flow(flow), index(index) {}

};

struct Dinic {

int N;

vector<vector<Edge> > G;

vector<Edge \*> dad;

vector<int> Q;

Dinic(int N) : N(N), G(N), dad(N), Q(N) {}

void AddEdge(int from, int to, int cap) {

G[from].push\_back(Edge(from, to, cap, 0, G[to].size()));

if (from == to) G[from].back().index++;

G[to].push\_back(Edge(to, from, 0, 0, G[from].size() - 1));

}

long long BlockingFlow(int s, int t) {

fill(dad.begin(), dad.end(), (Edge \*) NULL);

dad[s] = &G[0][0] - 1;

int head = 0, tail = 0;

Q[tail++] = s;

while (head < tail) {

int x = Q[head++];

for (int i = 0; i < G[x].size(); i++) {

Edge &e = G[x][i];

if (!dad[e.to] && e.cap - e.flow > 0) {

dad[e.to] = &G[x][i];

Q[tail++] = e.to;

}

}

}

if (!dad[t]) return 0;

long long totflow = 0;

for (int i = 0; i < G[t].size(); i++) {

Edge \*start = &G[G[t][i].to][G[t][i].index];

int amt = INF;

for (Edge \*e = start; amt && e != dad[s]; e = dad[e->from]) {

if (!e) { amt = 0; break; }

amt = min(amt, e->cap - e->flow);

}

if (amt == 0) continue;

for (Edge \*e = start; amt && e != dad[s]; e = dad[e->from]) {

e->flow += amt;

G[e->to][e->index].flow -= amt;

}

totflow += amt;

}

return totflow;

}

long long GetMaxFlow(int s, int t) {

long long totflow = 0;

while (long long flow = BlockingFlow(s, t))

totflow += flow;

return totflow;

}

};

Max Unweighted Bipartite matching

// Running time: O(|E| |V|) -- often much faster in practice

//INPUT: w[i][j] = edge between row node i and column node j

//OUTPUT: mr[i] = assignment for row node i, -1 if unassigned

//mc[j] = assignment for column node j, -1 if unassigned

// function returns number of matches made

#include <vector>

using namespace std;

typedef vector<int> VI;

typedef vector<VI> VVI;

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {

for (int j = 0; j < w[i].size(); j++) {

if (w[i][j] && !seen[j]) {

seen[j] = true;

if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {

mr[i] = j;

mc[j] = i;

return true;

}

}

}

return false;

}

int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {

mr = VI(w.size(), -1);

mc = VI(w[0].size(), -1);

int ct = 0;

for (int i = 0; i < w.size(); i++) {

VI seen(w[0].size());

if (FindMatch(i, w, mr, mc, seen)) ct++;

}

return ct;

}

Kruskal

#include<algorithm>

#include<vector>

#include<iostream>

#include<functional>

#include<assert.h>

#include<queue>

using namespace std;

#define INVALID\_PAR -1

#define FR(i,s,e) for(int i = (s) ; i <= (e) ; ++i)

vector<int> s\_parent;

vector<int> rnk;

#define MAX\_VER 10000

#define weight\_t int

#define P pair<int,int>

#define Edge pair<weight\_t,P>

vector<Edge> edge\_list;

vector<bool> in\_tree;

vector<pair<int,weight\_t>> mst\_adjlist[MAX\_VER];

int Find(int p)

{

int x = p;

while(s\_parent[x] != INVALID\_PAR){

x = s\_parent[x];

}

int y = p;

while(y != x){//path compression

int tmp = s\_parent[y];

s\_parent[y] = x;

y = tmp;

}

return x;

}

bool Union(int p , int q)

{

int par\_p = Find(p);

int par\_q = Find(q);

if(par\_p == par\_q)

return false;//already joint

//union by rank

if(rnk[par\_q] > rnk[par\_p])

swap(par\_p,par\_q);

if(rnk[par\_p] == rnk[par\_q])

++rnk[par\_p];

s\_parent[par\_q] = par\_p;

return true;//successfully joint

}

//O(ElgV)

weight\_t MST\_Kruskal(int v\_num,vector<Edge> &edges)

{

int selected\_edge = 0;

int e\_num = edges.size();

weight\_t mst\_weight = 0;

in\_tree.resize(e\_num);

fill(in\_tree.begin(),in\_tree.end(),false);

//init union-find

s\_parent.resize(v\_num);

rnk.resize(v\_num);

fill(s\_parent.begin(),s\_parent.end(),INVALID\_PAR);

fill(rnk.begin(),rnk.end(),0);

//sort the edges(use compare function for more complicated sorting)

//or use priority\_queue for best spanning tree!

sort(edges.begin(),edges.end());

FR(i,0,v\_num-1) mst\_adjlist[i].clear();

FR(i,0,e\_num-1){

Edge &e = edges[i];

int x = e.second.first;

int y = e.second.second;

weight\_t w = e.first;//set w=-w again if we are finding Max ST

if(Union(x,y)){

in\_tree[i] = true;

++selected\_edge;

mst\_weight += w;

//add the edge to the tree

mst\_adjlist[x].push\_back(P(y,w));

mst\_adjlist[y].push\_back(P(x,w));

if(selected\_edge == v\_num - 1)//the tree is compelete

break;

}

}

//if(selected\_edge < v - 1) Graph is not connected

//assert(selected\_edge == v - 1);

return mst\_weight;

}

int main()

{

int v,e;

while(true){

cin >> v >> e;

if(!v && !e) break;

edge\_list.clear();

FR(i,0,e-1){

int x , y;

weight\_t w;

cin >> x >> y >> w;

//for max. ST set w = -w

edge\_list.push\_back(Edge(w,P(x,y)));

}

//be careful that after the execution of mst\_kruskal

//the order of edge\_list will be different

cout << MST\_Kruskal(v,edge\_list) << endl;

}

return 0;

}

Global Min Cut

// Adjacency matrix implementation of Stoer-Wagner min cut algorithm.

// Running time:O(|V|^3)

// INPUT:

// graph, constructed using AddEdge()

// OUTPUT:

//(min cut value, nodes in half of min cut)

#include <cmath>

#include <vector>

#include <iostream>

using namespace std;

typedef vector<int> VI;

typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {

int N = weights.size();

VI used(N), cut, best\_cut;

int best\_weight = -1;

for (int phase = N-1; phase >= 0; phase--) {

VI w = weights[0];

VI added = used;

int prev, last = 0;

for (int i = 0; i < phase; i++) {

prev = last;

last = -1;

for (int j = 1; j < N; j++)

if (!added[j] && (last == -1 || w[j] > w[last])) last = j;

if (i == phase-1) {

for (int j = 0; j < N; j++) weights[prev][j] += weights[last][j];

for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][j];

used[last] = true;

cut.push\_back(last);

if (best\_weight == -1 || w[last] < best\_weight) {

best\_cut = cut;

best\_weight = w[last];

}

} else {

for (int j = 0; j < N; j++)

w[j] += weights[last][j];

added[last] = true;

}

}

}

return make\_pair(best\_weight, best\_cut);

}

ARTICULATION POINTS + BRIDGES

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

#define DFS\_WHITE -1

vector<vector<int>> AdjList;

vector<int> dfs\_num;

vector<int> dfs\_parent;// distinguish backedge

vector<int> dfs\_low;//loop head

vector<bool> articulation\_vertex;

int dfsNumberCounter, dfsRoot, rootChildren;

void init(int n) {

dfsNumberCounter = 0;

dfs\_low.assign(n, 0);

dfs\_num.assign(n, DFS\_WHITE);

dfs\_parent.assign(n, -1);

articulation\_vertex.assign(n, false);

}

void articulationPointAndBridge(int u) {

dfs\_low[u] = dfs\_num[u] = dfsNumberCounter++;

for (int j = 0; j < (int)AdjList[u].size(); j++) {

int v = AdjList[u][j];

if (dfs\_num[v] == DFS\_WHITE) {

dfs\_parent[v] = u;

//if rootChildren <= 1 -> root is an art. point

if (u == dfsRoot) rootChildren++;

articulationPointAndBridge(v);

if (dfs\_low[v] >= dfs\_num[u])

articulation\_vertex[u] = true;

if (dfs\_low[v] > dfs\_num[u])

printf("Edge(%d,%d)is a bridge\n", u, v);

dfs\_low[u] = min(dfs\_low[u], dfs\_low[v]);

}

else if (v != dfs\_parent[u])

dfs\_low[u] = min(dfs\_low[u], dfs\_num[v]);

}

}

Eulerian Path

struct Edge;

typedef list<Edge>::iterator iter;

struct Edge

{

int next\_vertex;

iter reverse\_edge;

Edge(int next\_vertex)

:next\_vertex(next\_vertex){ }

};

const int max\_vertices = ;

int num\_vertices;

list<Edge> adj[max\_vertices]; // adjacency list

vector<int> path;

void find\_path(int v)

{

while (adj[v].size() > 0)

{

int vn = adj[v].front().next\_vertex;

adj[vn].erase(adj[v].front().reverse\_edge);

adj[v].pop\_front();

find\_path(vn);

}

path.push\_back(v);

}

void add\_edge(int a, int b)

{

adj[a].push\_front(Edge(b));

iter ita = adj[a].begin();

adj[b].push\_front(Edge(a));

iter itb = adj[b].begin();

ita->reverse\_edge = itb;

itb->reverse\_edge = ita;

}

Min Cost Matching

//////////////////////////////////////////////

// Min cost bipartite matching via shortest augmenting paths

// This is an O(n^3) implementation of a shortest augmenting path

// algorithm for finding min cost perfect matchings in dense

// second.

//cost[i][j] = cost for pairing left node i with right node j

//Lmate[i] = index of right node that left node i pairs with

//Rmate[j] = index of left node that right node j pairs with

// The values in cost[i][j] may be positive or negative. To perform

// maximization, simply negate the cost[][] matrix.

//////////////////////////////////////////////

#include <algorithm>

#include <cstdio>

#include <cmath>

#include <vector>

using namespace std;

typedef vector<double> VD;

typedef vector<VD> VVD;

typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {

int n = int(cost.size());

// construct dual feasible solution

VD u(n);

VD v(n);

for (int i = 0; i < n; i++) {

u[i] = cost[i][0];

for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);

}

for (int j = 0; j < n; j++) {

v[j] = cost[0][j] - u[0];

for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);

}

Lmate = VI(n, -1);

Rmate = VI(n, -1);

int mated = 0;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (Rmate[j] != -1) continue;

if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {

Lmate[i] = j;

Rmate[j] = i;

mated++;

break;

}

}

}

VD dist(n);

VI dad(n);

VI seen(n);

// repeat until primal solution is feasible

while (mated < n) {

// find an unmatched left node

int s = 0;

while (Lmate[s] != -1) s++;

// initialize Dijkstra

fill(dad.begin(), dad.end(), -1);

fill(seen.begin(), seen.end(), 0);

for (int k = 0; k < n; k++)

dist[k] = cost[s][k] - u[s] - v[k];

int j = 0;

while (true) {

// find closest

j = -1;

for (int k = 0; k < n; k++) {

if (seen[k]) continue;

if (j == -1 || dist[k] < dist[j]) j = k;

}

seen[j] = 1;

// termination condition

if (Rmate[j] == -1) break;

// relax neighbors

const int i = Rmate[j];

for (int k = 0; k < n; k++) {

if (seen[k]) continue;

const double new\_dist = dist[j] + cost[i][k] - u[i] - v[k];

if (dist[k] > new\_dist) {

dist[k] = new\_dist;

dad[k] = j;

}

}

}

// update dual variables

for (int k = 0; k < n; k++) {

if (k == j || !seen[k]) continue;

const int i = Rmate[k];

v[k] += dist[k] - dist[j];

u[i] -= dist[k] - dist[j];

}

u[s] += dist[j];

// augment along path

while (dad[j] >= 0) {

const int d = dad[j];

Rmate[j] = Rmate[d];

Lmate[Rmate[j]] = j;

j = d;

}

Rmate[j] = s;

Lmate[s] = j;

mated++;

}

double value = 0;

for (int i = 0; i < n; i++)

value += cost[i][Lmate[i]];

return value;

}

Max Flow Min Cost

// forward and reverse edges separately (so you can set cap[i][j] !=

// cap[j][i]). For a regular max flow, set all edge costs to 0.

// Running time, O(|V|^2) cost per augmentation

// max flow: O(|V|^3) augmentations

// min cost max flow: O(|V|^4 \* MAX\_EDGE\_COST) augmentations

// INPUT:

// - graph, constructed using AddEdge()

// - source

// - sink

// OUTPUT:

// - (maximum flow value, minimum cost value)

// - To obtain the actual flow, look at positive values only.

#include <cmath>

#include <vector>

#include <iostream>

using namespace std;

typedef vector<int> VI;

typedef vector<VI> VVI;

typedef long long L;

typedef vector<L> VL;

typedef vector<VL> VVL;

typedef pair<int, int> PII;

typedef vector<PII> VPII;

const L INF = numeric\_limits<L>::max() / 4;

struct MinCostMaxFlow {

int N;

VVL cap, flow, cost;

VI found;

VL dist, pi, width;

VPII dad;

MinCostMaxFlow(int N) :

N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),

found(N), dist(N), pi(N), width(N), dad(N) {}

void AddEdge(int from, int to, L cap, L cost) {

this->cap[from][to] = cap;

this->cost[from][to] = cost;

}

void Relax(int s, int k, L cap, L cost, int dir) {

L val = dist[s] + pi[s] - pi[k] + cost;

if (cap && val < dist[k]) {

dist[k] = val;

dad[k] = make\_pair(s, dir);

width[k] = min(cap, width[s]);

}

}

L Dijkstra(int s, int t) {

fill(found.begin(), found.end(), false);

fill(dist.begin(), dist.end(), INF);

fill(width.begin(), width.end(), 0);

dist[s] = 0;

width[s] = INF;

while (s != -1) {

int best = -1;

found[s] = true;

for (int k = 0; k < N; k++) {

if (found[k]) continue;

Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);

Relax(s, k, flow[k][s], -cost[k][s], -1);

if (best == -1 || dist[k] < dist[best]) best = k;

}

s = best;

}

for (int k = 0; k < N; k++)

pi[k] = min(pi[k] + dist[k], INF);

return width[t];

}

pair<L, L> GetMaxFlow(int s, int t) {

L totflow = 0, totcost = 0;

while (L amt = Dijkstra(s, t)) {

totflow += amt;

for (int x = t; x != s; x = dad[x].first) {

if (dad[x].second == 1) {

flow[dad[x].first][x] += amt;

totcost += amt \* cost[dad[x].first][x];

}

else {

flow[x][dad[x].first] -= amt;

totcost -= amt \* cost[x][dad[x].first];

}

}

}

return make\_pair(totflow, totcost);

}

};

STRONGLY CONNECTED

static ArrayList<Integer>[] w;

static int[] low;

static int[] nums;

static boolean[] instack;

static Stack<Integer> stack = new Stack<Integer>();

static int counter;//dar main 0 she, ghable seda zadane scc

static void scc(int start) {

low[start] = nums[start] = counter++;

instack[start] = true;

stack.push(start);

for (int i : w[start]) {

if (nums[i] == -1) scc(i);

if (instack[i])

low[start] = Math.min(low[i], low[start]);

}

if (nums[start] == low[start]) {

// yek scc peyda shode ke dar stack hast

}

}

Geometry 1

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <string>

#include <cstring>

#include <cmath>

#include <vector>

#include <queue>

#include <stack>

#include <algorithm>

using namespace std;

const double eps = 1e-8;

const double PI = acos(-1.0);

struct Point

{

double x, y;

Point(double x = 0, double y = 0) : x(x), y(y) { }

bool operator < (const Point& a) const

{

if(a.x != x) return x < a.x;

return y < a.y;

}

};

typedef Point Vector;

struct Line

{

Point P;

Vector v;

double ang;

Line() {}

Line(Point P, Vector v) : P(P), v(v) { ang = atan2(v.y, v.x); }

bool operator < (const Line& L) const

{

return ang < L.ang;

}

};

Vector operator + (Vector A, Vector B) { return Vector(A.x+B.x, A.y+B.y); }

Vector operator - (Point A, Point B) { return Vector(A.x-B.x, A.y-B.y); }

Vector operator \* (Vector A, double p) { return Vector(A.x\*p, A.y\*p); }

Vector operator / (Vector A, double p) { return Vector(A.x/p, A.y/p); }

int dcmp(double x)

{

if(fabs(x) < eps) return 0; else return x < 0 ? -1 : 1;

}

bool operator == (const Point& a, const Point &b)

{

return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y) == 0;

}

double Dot(Vector A, Vector B) { return A.x\*B.x + A.y\*B.y; }

double Length(Vector A) { return sqrt(Dot(A, A)); }

double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }

double Cross(Vector A, Vector B) { return A.x\*B.y - A.y\*B.x; }

double Area2(Point A, Point B, Point C) { return fabs(Cross(B-A, C-A)) / 2; }

Vector Rotate(Vector A, double rad)

{

return Vector(A.x\*cos(rad)-A.y\*sin(rad), A.x\*sin(rad) + A.y\*cos(rad));

}

Point GetLineIntersection(Point P, Vector v, Point Q, Vector w)

{

Vector u = P-Q;

double t = Cross(w, u) / Cross(v, w);

return P+v\*t;

}

bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)

{

double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);

double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);

return dcmp(c1) \* dcmp(c2) < 0 && dcmp(c3) \* dcmp(c4) < 0;

}

bool OnSegment(Point p, Point a1, Point a2)

{

return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) < 0;

}

double PolygonArea(Point\* p, int n)

{

double area = 0;

for(int i = 1; i < n-1; i++)

area += Cross(p[i]-p[0], p[i+1]-p[0]);

return area/2;

}

double PointDistanceToLine(Point P, Point A, Point B)

{

Vector v1 = B-A, v2 = P-A;

return fabs(Cross(v1, v2)) / Length(v1);

}

double PointDistanceToSegment(Point P, Point A, Point B)

{

if(A == B) return Length(P-A);

Vector v1 = B-A, v2 = P-A, v3 = P-B;

if(dcmp(Dot(v1, v2) < 0)) return Length(v2);

else if(dcmp(Dot(v1, v3) > 0)) return Length(v3);

else return fabs(Cross(v1, v2)) / Length(v1);

}

int isPointInPolygon(Point p, Point \*poly, int n)

{

int wn = 0;

for(int i = 0; i < n; i++)

{

const Point& p1 = poly[i], p2 = poly[(i+1)%n];

if(p == p1 || p == p2 || OnSegment(p, p1, p2)) return -1;

int k = dcmp(Cross(p2-p1, p-p1));

int d1 = dcmp(p1.y - p.y);

int d2 = dcmp(p2.y - p.y);

if(k > 0 && d1 <= 0 && d2 > 0) wn++;

if(k < 0 && d2 <= 0 && d1 > 0) wn--;

}

if(wn != 0) return 1;

return 0;

}

Vector Normal(Vector A)

{

double L = Length(A);

return Vector (-A.y/L, A.x/L);

}

double Dist2(Point p1, Point p2)

{

return (p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y);

}

double RotatingCalipers(Point \*P, int n)

{

if(n == 1) return 0;

if(n == 2) return Dist2(P[0], P[1]);

P[n] = P[0];

double ans = 0;

for(int u = 0, v = 1; u < n; u++)

{

for(;;)

{

double diff = Cross(P[u+1]-P[u], P[v+1]-P[v]);

if(diff <= 0)

{

ans = max(ans, Dist2(P[u], P[v]));

if(diff == 0) ans = max(ans, Dist2(P[u], P[v+1]));

break;

}

v = (v + 1) % n;

}

}

return ans;

}

bool OnLeft(Line L, Point p)

{

return Cross(L.v, p-L.P) > 0;

}

Point GetLineIntersection2(const Line &a, const Line &b)

{

Vector u = a.P-b.P;

double t = Cross(b.v, u) / Cross(a.v, b.v);

return a.P+a.v\*t;

}

int HalfPlaneIntersection(Line\* L, int n, Point\* poly)

{

sort(L, L+n);

int first, last;

Point \*p = new Point[n];

Line\* q = new Line[n];

q[first=last=0] = L[0];

for(int i = 1; i < n; i++)

{

while(first < last && !OnLeft(L[i], p[last-1])) last--;

while(first < last && !OnLeft(L[i], p[first])) first++;

q[++last] = L[i];

if(fabs(Cross(q[last].v, q[last-1].v)) < eps)

{

last--;

if(OnLeft(q[last], L[i].P)) q[last] = L[i];

}

if(first < last) p[last-1] = GetLineIntersection2(q[last-1], q[last]);

}

while(first < last && !OnLeft(q[first], p[last-1])) last--;

if(last - first <= 1) return 0;

p[last] = GetLineIntersection2(q[last], q[first]);

int m = 0;

for(int i = first; i <= last; i++) poly[m++] = p[i];

return m;

}

vector<Point> CutPolygon(const vector<Point> &poly, Point A, Point B)

{

vector<Point> newpoly;

int n = poly.size();

for(int i = 0; i < n; i++)

{

Point C = poly[i], D = poly[(i+1)%n];

if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push\_back(C);

if(dcmp(Cross(B-A, C-D)) != 0)

{

Point ip = GetLineIntersection(A, B-A, C, D-C);

if(OnSegment(ip, C, D)) newpoly.push\_back(ip);

}

}

return newpoly;

}

Convex Hull

#include <algorithm>

#include <vector>

using namespace std;

typedef int coord\_t; // coordinate type

typedef long long coord2\_t; // must be big enough to hold 2\*max(|coordinate|)^2

struct Point {

coord\_t x, y;

bool operator <(const Point &p) const {

return x < p.x || (x == p.x && y < p.y);

}

};

coord2\_t cross(const Point &O, const Point &A, const Point &B)

{

return (A.x - O.x) \* (B.y - O.y) - (A.y - O.y) \* (B.x - O.x);

}

// Returns a list of points on the convex hull in counter-clockwise order.

// Note: the last point in the returned list is the same as the first one.

vector<Point> convex\_hull(vector<Point> P)

{

int n = P.size(), k = 0;

vector<Point> H(2\*n);

// Sort points lexicographically

sort(P.begin(), P.end());

// Build lower hull

for (int i = 0; i < n; i++) {

while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;

H[k++] = P[i];

}

// Build upper hull

for (int i = n-2, t = k+1; i >= 0; i--) {

while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;

H[k++] = P[i];

}

H.resize(k);

return H;

}

int main()

{

vector<Point> h,p(1000000);//,p(6);

srand(time(nullptr));

for(int i = 0 ; i < 100000 ; ++i){

p[i] = Point(rand() , rand());

if(!(i % 1000)) srand(time(nullptr));

}

clock\_t start = clock();

h = convex\_hull(p);

for(int i = 0 ; i < h.size() ; ++i){

cout << "(" << h[i].x << "," << h[i].y << ")" << endl;

}

}

Geometry 2

#include <iostream>

#include <vector>

#include <cmath>

#include <cassert>

using namespace std;

double INF = 1e100;

double EPS = 1e-12;

#define M\_PI acos(-1)

struct PT {

double x, y;

PT() {}

PT(double x, double y) : x(x), y(y) {}

PT(const PT &p) : x(p.x), y(p.y) {}

PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }

PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }

bool operator<(const PT &p) const { return (x!=p.x ? x<p.x : y<p.y ); }

PT operator \* (double c) const { return PT(x\*c, y\*c ); }

PT operator / (double c) const { return PT(x/c, y/c ); }

};

double dot(PT p, PT q) { return p.x\*q.x+p.y\*q.y; }

double dist2(PT p, PT q) { return dot(p-q,p-q); }

double cross(PT p, PT q) { return p.x\*q.y-p.y\*q.x; }

// rotate a point CCW or CW around the origin

PT RotateCCW90(PT p) { return PT(-p.y,p.x); }

PT RotateCW90(PT p) { return PT(p.y,-p.x); }

PT RotateCCW(PT p, double t) {

return PT(p.x\*cos(t)-p.y\*sin(t), p.x\*sin(t)+p.y\*cos(t));

}

// project point c onto line through a and b

// assuming a != b

PT ProjectPointLine(PT a, PT b, PT c) {

return a + (b-a)\*dot(c-a, b-a)/dot(b-a, b-a);

}

// project point c onto line segment through a and b

PT ProjectPointSegment(PT a, PT b, PT c) {

double r = dot(b-a,b-a);

if (fabs(r) < EPS) return a;

r = dot(c-a, b-a)/r;

if (r < 0) return a;

if (r > 1) return b;

return a + (b-a)\*r;

}

// compute distance from c to segment between a and b

double DistancePointSegment(PT a, PT b, PT c) {

return sqrt(dist2(c, ProjectPointSegment(a, b, c)));

}

// compute distance between point (x,y,z) and plane ax+by+cz=d

double DistancePointPlane(double x, double y, double z,

double a, double b, double c, double d)

{

return fabs(a\*x+b\*y+c\*z-d)/sqrt(a\*a+b\*b+c\*c);

}

// determine if lines from a to b and c to d are parallel or collinear

bool LinesParallel(PT a, PT b, PT c, PT d) {

return fabs(cross(b-a, c-d)) < EPS;

}

bool LinesCollinear(PT a, PT b, PT c, PT d) {

return LinesParallel(a, b, c, d)

&& fabs(cross(a-b, a-c)) < EPS

&& fabs(cross(c-d, c-a)) < EPS;

}

// determine if line segment from a to b intersects with

// line segment from c to d

bool SegmentsIntersect(PT a, PT b, PT c, PT d) {

if (LinesCollinear(a, b, c, d)) {

if (dist2(a, c) < EPS || dist2(a, d) < EPS ||

dist2(b, c) < EPS || dist2(b, d) < EPS) return true;

if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)

return false;

return true;

}

if (cross(d-a, b-a) \* cross(c-a, b-a) > 0) return false;

if (cross(a-c, d-c) \* cross(b-c, d-c) > 0) return false;

return true;

}

// compute intersection of line passing through a and b

// with line passing through c and d, assuming that unique

// intersection exists; for segment intersection, check if

// segments intersect first

PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {

b=b-a; d=c-d; c=c-a;

assert(dot(b, b) > EPS && dot(d, d) > EPS);

return a + b\*cross(c, d)/cross(b, d);

}

// compute center of circle given three points

PT ComputeCircleCenter(PT a, PT b, PT c) {

b=(a+b)/2;

c=(a+c)/2;

return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));

}

// determine if point is in a possibly non-convex polygon (by William

// Randolph Franklin); returns 1 for strictly interior points, 0 for

// strictly exterior points, and 0 or 1 for the remaining points.

// Note that it is possible to convert this into an \*exact\* test using

// integer arithmetic by taking care of the division appropriately

// (making sure to deal with signs properly) and then by writing exact

// tests for checking point on polygon boundary

bool PointInPolygon(const vector<PT> &p, PT q) {

bool c = 0;

for (int i = 0; i < p.size(); i++){

int j = (i+1)%p.size();

if ((p[i].y <= q.y && q.y < p[j].y ||

p[j].y <= q.y && q.y < p[i].y) &&

q.x < p[i].x + (p[j].x - p[i].x) \* (q.y - p[i].y) / (p[j].y - p[i].y))

c = !c;

}

return c;

}

// determine if point is on the boundary of a polygon

bool PointOnPolygon(const vector<PT> &p, PT q) {

for (int i = 0; i < p.size(); i++)

if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)

return true;

return false;

}

// compute intersection of line through points a and b with

// circle centered at c with radius r > 0

vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {

vector<PT> ret;

b = b-a;

a = a-c;

double A = dot(b, b);

double B = dot(a, b);

double C = dot(a, a) - r\*r;

double D = B\*B - A\*C;

if (D < -EPS) return ret;

ret.push\_back(c+a+b\*(-B+sqrt(D+EPS))/A);

if (D > EPS)

ret.push\_back(c+a+b\*(-B-sqrt(D))/A);

return ret;

}

// compute intersection of circle centered at a with radius r

// with circle centered at b with radius R

vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {

vector<PT> ret;

double d = sqrt(dist2(a, b));

if (d > r+R || d+min(r, R) < max(r, R)) return ret;

double x = (d\*d-R\*R+r\*r)/(2\*d);

double y = sqrt(r\*r-x\*x);

PT v = (b-a)/d;

ret.push\_back(a+v\*x + RotateCCW90(v)\*y);

if (y > 0)

ret.push\_back(a+v\*x - RotateCCW90(v)\*y);

return ret;

}

double ComputeSignedArea(const vector<PT> &p) {

double area = 0;

for(int i = 0; i < p.size(); i++) {

int j = (i+1) % p.size();

area += p[i].x\*p[j].y - p[j].x\*p[i].y;

}

return area / 2.0;

}

double ComputeArea(const vector<PT> &p) {

return fabs(ComputeSignedArea(p));

}

PT ComputeCentroid(const vector<PT> &p) {

PT c(0,0);

double scale = 6.0 \* ComputeSignedArea(p);

for (int i = 0; i < p.size(); i++){

int j = (i+1) % p.size();

c = c + (p[i]+p[j])\*(p[i].x\*p[j].y - p[j].x\*p[i].y);

}

return c / scale;

}

// tests whether or not a given polygon (in CW or CCW order) is simple

bool IsSimple(const vector<PT> &p) {

for (int i = 0; i < p.size(); i++) {

for (int k = i+1; k < p.size(); k++) {

int j = (i+1) % p.size();

int l = (k+1) % p.size();

if (i == l || j == k) continue;

if (SegmentsIntersect(p[i], p[j], p[k], p[l]))

return false;

}

}

return true;

}

Closest Points

#include <cstdio>

#include <vector>

#include <algorithm>

#include <set>

#include <cstdlib>

#include <cmath>

#include <iostream>

using namespace std;

#define P pair< double , double >

#define x first

#define y second

#define dist( a , b ) (( a.x - b.x )\*( a.x - b.x ) + ( a.y - b.y )\*( a.y - b.y ))

#define maxn 10003

#define maxv 40003

#define inf 10000

#define eps 1e-9

P points[maxn];

int main()

{

int n;

while( scanf("%d" , &n ) != (-1) && n )

{

for( int i = 0 ; i < n ; i++ )

scanf("%lf %lf" , &points[i].x , &points[i].y );

sort( points , points + n );

set<P> activeSet;

activeSet.insert( P( points[0].y , points[0].x ) );

double h = inf;

double ans = inf\*inf;

int leftInd = 0;

for( int i = 1 ; i < n ; i++ )

{

for( ; leftInd < i ; leftInd++ )

if( ( points[i].x - points[leftInd].x - eps )< h ) break;

else activeSet.erase( P( points[leftInd].y , points[leftInd].x ) );

set<P>::iterator s = activeSet.lower\_bound( P( points[i].y - h -eps , 0 ) );// greater or equal

set<P>::iterator e = activeSet.upper\_bound( P( points[i].y + h +eps, maxv ) );// strictly greater

for( ; s != e ; s++ )

ans = min( ans, dist( points[i] , P( s->y , s->x ) ) );

h = sqrt(ans) + eps;

activeSet.insert( P( points[i].y , points[i].x ) );

}

ans = sqrt( ans );

if( (int)( ( ans\*inf)+0.5 ) < inf\*inf )

printf("%.4lf\n" , ans );

else

printf("INFINITY\n");

}

return 0;

}

Number Theory

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

typedef vector<int> VI;

typedef pair<int,int> PII;

// return a % b (positive value)

int mod(int a, int b) {

return ((a%b)+b)%b;

}

// computes gcd(a,b)

int gcd(int a, int b) {

int tmp;

while(b){a%=b; tmp=a; a=b; b=tmp;}

return a;

}

// computes lcm(a,b)

int lcm(int a, int b) {

return a/gcd(a,b)\*b;

}

// returns d = gcd(a,b); finds x,y such that d = ax + by

int extended\_euclid(int a, int b, int &x, int &y) {

int xx = y = 0;

int yy = x = 1;

while (b) {

int q = a/b;

int t = b; b = a%b; a = t;

t = xx; xx = x-q\*xx; x = t;

t = yy; yy = y-q\*yy; y = t;

}

return a;

}

// finds all solutions to ax = b (mod n)

VI modular\_linear\_equation\_solver(int a, int b, int n) {

int x, y;

VI solutions;

int d = extended\_euclid(a, n, x, y);

if (!(b%d)) {

x = mod (x\*(b/d), n);

for (int i = 0; i < d; i++)

solutions.push\_back(mod(x + i\*(n/d), n));

}

return solutions;

}

// computes b such that ab = 1 (mod n), returns -1 on failure

int mod\_inverse(int a, int n) {

int x, y;

int d = extended\_euclid(a, n, x, y);

if (d > 1) return -1;

return mod(x,n);

}

// Chinese remainder theorem (special case): find z such that

// z % x = a, z % y = b. Here, z is unique modulo M = lcm(x,y).

// Return (z,M). On failure, M = -1.

PII chinese\_remainder\_theorem(int x, int a, int y, int b) {

int s, t;

int d = extended\_euclid(x, y, s, t);

if (a%d != b%d) return make\_pair(0, -1);

return make\_pair(mod(s\*b\*x+t\*a\*y,x\*y)/d, x\*y/d);

}

// Chinese remainder theorem: find z such that

// z % x[i] = a[i] for all i. Note that the solution is

// unique modulo M = lcm\_i (x[i]). Return (z,M). On

// failure, M = -1. Note that we do not require the a[i]'s

// to be relatively prime.

PII chinese\_remainder\_theorem(const VI &x, const VI &a) {

PII ret = make\_pair(a[0], x[0]);

for (int i = 1; i < x.size(); i++) {

ret = chinese\_remainder\_theorem(ret.second, ret.first, x[i], a[i]);

if (ret.second == -1) break;

}

return ret;

}

// computes x and y such that ax + by = c; on failure, x = y =-1

void linear\_diophantine(int a, int b, int c, int &x, int &y) {

int d = gcd(a,b);

if (c%d) {

x = y = -1;

} else {

x = c/d \* mod\_inverse(a/d, b/d);

y = (c-a\*x)/b;

}

}

long conquer\_fibonacci\_lgN(long n){

long i, h, j, k, t;

i = h = 1;

j = k = 0;

while (n > 0) {

if (n % 2 == 1){

t = j \* h;

j = i \* h + j \* k + t;

i = i \* k + t;

}

t = h \* h;

h = 2 \* k \* h + t;

k = k \* k + t;

n = (long)n / 2;

}

return j;

}

Gauss-Jordan elimination

// Uses:

// (1) solving systems of linear equations (AX=B)

// (2) inverting matrices (AX=I)

// (3) computing determinants of square matrices

// Running time: O(n^3)

// INPUT: a[][] = an nxn matrix

// b[][] = an nxm matrix

//

// OUTPUT: X = an nxm matrix (stored in b[][])

// A^{-1} = an nxn matrix (stored in a[][])

// returns determinant of a[][]

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;

typedef double T;

typedef vector<T> VT;

typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {

const int n = a.size();

const int m = b[0].size();

VI irow(n), icol(n), ipiv(n);

T det = 1;

for (int i = 0; i < n; i++) {

int pj = -1, pk = -1;

for (int j = 0; j < n; j++) if (!ipiv[j])

for (int k = 0; k < n; k++) if (!ipiv[k])

if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j; pk = k; }

if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }

ipiv[pk]++;

swap(a[pj], a[pk]);

swap(b[pj], b[pk]);

if (pj != pk) det \*= -1;

irow[i] = pj;

icol[i] = pk;

T c = 1.0 / a[pk][pk];

det \*= a[pk][pk];

a[pk][pk] = 1.0;

for (int p = 0; p < n; p++) a[pk][p] \*= c;

for (int p = 0; p < m; p++) b[pk][p] \*= c;

for (int p = 0; p < n; p++) if (p != pk) {

c = a[p][pk];

a[p][pk] = 0;

for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] \* c;

for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] \* c;

}

}

for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {

for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);

}

return det;

}

int main() {

const int n = 4;

const int m = 2;

double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };

double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };

VVT a(n), b(n);

for (int i = 0; i < n; i++) {

a[i] = VT(A[i], A[i] + n);

b[i] = VT(B[i], B[i] + m);

}

double det = GaussJordan(a, b);

// expected: 60

cout << "Determinant: " << det << endl;

// expected: -0.233333 0.166667 0.133333 0.0666667

// 0.166667 0.166667 0.333333 -0.333333

// 0.233333 0.833333 -0.133333 -0.0666667

// 0.05 -0.75 -0.1 0.2

cout << "Inverse: " << endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

cout << a[i][j] << ' ';

cout << endl;

}

// expected: 1.63333 1.3

// -0.166667 0.5

// 2.36667 1.7

// -1.85 -1.35

cout << "Solution: " << endl;

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++)

cout << b[i][j] << ' ';

cout << endl;

}}

LONGEST PALINDROME O(N)

int longest\_palindrome(char \*text, intn) {

int rad[2 \* n], i, j, k;

for (i = 0, j = 0; i < 2 \* n; i += k, j = max(j - k, 0)) {

while (i - j >= 0 && i + j + 1 < 2 \* n&&text[(i - j) / 2] == text[(i + j + 1) / 2]) ++j;

rad[i] = j;

for (k = 1; i - k >= 0 && rad[i] - k >= 0 && rad[i - k] != rad[i] - k; ++k)

rad[i + k] = min(rad[i - k], rad[i] - k);

}

return \*max\_element(rad, rad + 2 \* n); // ret. centre of the longest palindrome

}

KMP

#include <iostream>

#include <string>

#include <vector>

using namespace std;

typedef vector<int> VI;

void buildTable(string& w, VI& t)

{

t = VI(w.length());

int i = 2, j = 0;

t[0] = -1; t[1] = 0;

while (i < w.length())

{

if (w[i - 1] == w[j]) { t[i] = j + 1; i++; j++; }

else if (j > 0) j = t[j];

else { t[i] = 0; i++; }

}

}

int KMP(string& s, string& w)

{

int m = 0, i = 0;

VI t;

buildTable(w, t);

while (m + i < s.length())

{

if (w[i] == s[m + i])

{

i++;

if (i == w.length()) return m;

}

else

{

m += i - t[i];

if (i > 0) i = t[i];

}

}

return s.length();

}

LIS O(NlgN)

// Given a list of numbers of length n, this routine extracts a

// longest increasing subsequence.

//

// Running time: O(n log n)

//

// INPUT: a vector of integers

// OUTPUT: a vector containing the longest increasing subsequence

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

typedef vector<int> VI;

typedef pair<int, int> PII;

typedef vector<PII> VPII;

#define STRICTLY\_INCREASNG

VI LongestIncreasingSubsequence(VI v) {

VPII best;

VI dad(v.size(), -1);

for (int i = 0; i < v.size(); i++) {

#ifdef STRICTLY\_INCREASNG

PII item = make\_pair(v[i], 0);

VPII::iterator it = lower\_bound(best.begin(), best.end(), item);

item.second = i;

#else

PII item = make\_pair(v[i], i);

VPII::iterator it = upper\_bound(best.begin(), best.end(), item);

#endif

if (it == best.end()) {

dad[i] = (best.size() == 0 ? -1 : best.back().second);

best.push\_back(item);

}

else {

dad[i] = dad[it->second];

\*it = item;

}

}

VI ret;

for (int i = best.back().second; i >= 0; i = dad[i])

ret.push\_back(v[i]);

reverse(ret.begin(), ret.end());

return ret;

}

ig Addition

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#define MAX 1000

void reverse(char \*from, char \*to ){

int len=strlen(from);

int l;

for(l=0;l<len;l++)

to[l]=from[len-l-1];

to[len]='\0';

}

void call\_sum(char \*first, char \*sec, char \*result)

{

char F[MAX], S[MAX], Res[MAX];

int f,s,sum,extra,now;

f=strlen(first);

s=strlen(sec);

reverse(first,F);

reverse(sec,S);

for(now=0,extra=0;(now<f && now<s);now++){

sum=(F[now]-'0') + (S[now]-'0') + extra;

Res[now]=sum%10 +'0';

extra= sum/10;

}

for(;now<f;now++){

sum=F[now] + extra-'0';

Res[now]=sum%10 +'0';

extra=sum/10;

}

for(;now<s;now++){

sum=F[now] + extra-'0';

Res[now]=sum%10 +'0';

extra=sum/10;

}

if(extra!=0)

Res[now++]=extra+'0';

Res[now]='\0';

if(strlen(Res)==0) strcpy(Res,"0");

reverse(Res,result);

}

int main(){

char fir[MAX],sec[MAX],res[MAX];

while(scanf("%s%s",&fir,&sec)==2){

call\_sum(fir,sec,res);

int len=strlen(res);

for(int i=0;i<len;i++) printf("%c",res[i]);

printf("\n");

}

return 0;

}

Big Multiplication

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#define MAX 1000

void reverse(char \*from, char \*to )

{

int len=strlen(from);

int l;

for(l=0;l<len;l++)

to[l]=from[len-l-1];

to[len]='\0';}

void call\_mult(char \*first,char \*sec,char \*result)

{

char F[MAX],S[MAX],temp[MAX];

int f\_len,s\_len,f,s,r,t\_len,hold,res;

f\_len=strlen(first);

s\_len=strlen(sec);

reverse(first,F);

reverse(sec,S);

t\_len=f\_len+s\_len;

r=-1;

for(f=0;f<=t\_len;f++)

temp[f]='0';

temp[f]='\0';

for(s=0;s<s\_len;s++){

hold=0;

for(f=0;f<f\_len;f++){

res=(F[f]-'0')\*(S[s]-'0') + hold+(temp[f+s]-'0');

temp[f+s]=res%10+'0';

hold=res/10;

if(f+s>r)

r=f+s;

}

while(hold!=0){

res=hold+temp[f+s]-'0';

hold=res/10;

temp[f+s]=res%10+'0';

if(r<f+s)

r=f+s;

f++;

}

}

for(;r>0 && temp[r]=='0';r--);

temp[r+1]='\0';

reverse(temp,result);

}

int main(){

char fir[MAX],sec[MAX],res[MAX];

while(scanf("%s%s",&fir,&sec)==2){

call\_mult(fir,sec,res);

int len=strlen(res);

for(int i=0;i<len;i++)

printf("%c",res[i]);

printf("\n");

}

return 0;

}

Earth Coordinates

const double R = 6371009.00 ; // radius of earth / sphere

// great Circle in Sphere / points must be in radian / distance between 2 pnts on earth

double spherical\_distance(double lat1,double lon1,double lat2,double lon2) {

double dlon = lon2 - lon1;

double dlat = lat2 - lat1;

double a = pow((sin(dlat/2)),2) + cos(lat1) \* cos(lat2) \* pow(sin(dlon/2), 2);

double c = 2 \* atan2(sqrt(a), sqrt(1-a));

double d = R \* c;

double e = sqrt(2.\*R\*R \*(1-cos(c)));

return round(d-e+EPS);

}