

1 STL

Listing 1: sorts.cpp

```
#include <stdio.h>
#include <vector>
#include <queue>
#include <functional>
#include <algorithm>
#include <string>
#include <iostream>
using namespace std;

struct{
    int profit, spendings;
}typedef biz;

bool sortBizs(const biz &a, const biz &b){
    return a.profit - a.spendings < b.profit - b.spendings;
}

int main(){
    int i;

    int intarray[100];
    intarray[0] = 3;
    intarray[1] = 5;
    intarray[2] = 1;
    intarray[3] = 0;
    sort(intarray, intarray + 4);
    for(i=0; i<4; i++) printf("%d\n", intarray[i]);
    putchar('\n');
    sort(intarray, intarray + 4, greater<int>());
    for(i=0; i<4; i++) printf("%d\n", intarray[i]);
    putchar('\n');

    vector<string> stringvector;
    stringvector.push_back("ancel");
    stringvector.push_back("coxo");
    stringvector.push_back("nabo");
    sort(stringvector.begin(), stringvector.end());
    for(i=0; i<3; i++) cout << stringvector[i] << endl;
    putchar('\n');

    biz bizs[5];
    bizs[2].profit = 15;
    bizs[2].spendings = 3;
    bizs[0].profit = 3;
    bizs[0].spendings = 5;
    bizs[1].profit = 10;
    bizs[1].spendings = 5;
    sort(bizs, bizs + 3, sortBizs);
    for(i=0; i<3; i++)
        printf("biz: profit: %d, spendings: %d\n", bizs[i].profit, bizs[i].spendings);

    return 0;
}
```

Listing 2: pair.cpp

```
#include <utility>
#include <stdio.h>
using namespace std;

int main(){
    pair<int, double> p = pair<int, double>(3, 2.3);
    printf("%d %lf\n", p.first, p.second);
}
```

Listing 3: bsearch.cpp

```

#include <iostream>
#include <algorithm>
#include <vector>

int main () {
    int myints[] = {10,20,30,30,20,10,10,20};
    std::vector<int> v(myints,myints+8);

    std::sort (v.begin(), v.end());

    std::vector<int>::iterator low,up;
    low = std::lower_bound (v.begin(), v.end(), 20);
    up = std::upper_bound (v.begin(), v.end(), 20);

    std::cout << "lower_bound at position " << (low- v.begin()) << '\n';
    std::cout << "upper_bound at position " << (up - v.begin()) << '\n';

    return 0;
}

```

Listing 4: compare.cpp

```

#include <stdio.h>
#include <queue>
using namespace std;

struct{
    int x, y;
}typedef Point;

bool operator<(const Point& a, const Point& b) {
    if(a.y < b.y)
        return true;
    else if(a.y == b.y)
        return a.x < b.y;
    return false;
}

priority_queue<Point> pq;

int main(){
    Point p;
    p.x = 3; p.y = 2; pq.push(p);
    p.x = 3; p.y = 4; pq.push(p);
    p.x = 3; p.y = 3; pq.push(p);
    p.x = 2; p.y = 3; pq.push(p);
    p.x = 1; p.y = 4; pq.push(p);
    p.x = 1; p.y = 3; pq.push(p);
    while(!pq.empty()){
        p = pq.top();
        printf("%d %d\n", p.y, p.x);
        pq.pop();
    }
    /*
        4 1
        4 3
        3 3
        3 1
        3 2
        2 3
    */
}

```

2 Others

Listing 5: bitset.cpp

```
// BITSET IMPLEMENTATION

#include <stdio.h>

void printset(unsigned int set, int size){
    int i;
    for(i=0; i<size; i++){
        printf("%d", set%2);
        set = set >> 1;
    }
    putchar('\n');
}

int isset(unsigned int set, int p){
    return (set >> p) & 0x1;
}

int setp(unsigned int set, int pos){
    set = set | (0x1 << pos);
    return set;
}

int unsetp(unsigned int set, int pos){
    set = set ^ (0x1 << pos);
}
```

3 Graphs

Listing 6: ArticulationPoints.cpp

```
// Count articulation points of a graph

#include <stdio.h>
#include <string.h>
#include <vector>

using namespace std;

struct{
    vector<int> edges;
    int dfs;
    int low;
}typedef Node;

int n;
Node graph[805];
bool vis[805];

int d;
int INF=100000;
int best, count;

int min(int a, int b){
    return a < b ? a : b;
}

void dfs(int node){
    int i, neigh;
    vis[node] = true;
    graph[node].dfs = d++;
    graph[node].low = graph[node].dfs;
    for(i=0; i<(int)graph[node].edges.size(); i++){
        neigh = graph[node].edges[i];
        if(!vis[neigh]){
            dfs(neigh);
            graph[node].low = min(graph[node].low, graph[neigh].low);
            if(graph[node].dfs>1){
                if(graph[neigh].low >= graph[node].dfs && graph[node].edges.size() >
                    1)
                    count++;
            }else{
                if(graph[neigh].dfs > 2){
                    count++;
                }
            }
        }else{
            graph[node].low = min(graph[node].low, graph[neigh].dfs);
        }
    }
}

int main(){
    int i, j, v;

    while(true){
        scanf("%d", &n);
        if(n==0)
            break;
        printf("%d\n", n);
        for(i=0; i<n; i++){
            scanf("%d", &j);
            printf("%d\n", j);
            while(getchar() != '\n'){
```

```

        scanf("%d", &v);
        graph[j].edges.push_back(v);
        graph[v].edges.push_back(j);
        printf("%d ", v);
    }
    putchar('\n');
}

d=1;
memset(vis, false, sizeof(vis));
count=0;
for(i=1; i<=n; i++)
    if(!vis[i])
        dfs(i);

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

```

Listing 7: BipartiteMatching.cpp

```

// Week Problem K - Distributing gifts between friends, each friend has different likings and there
// are quantities of each gift (Bipartite Matching with Max Flow(Edmonds-Karp) )

#include <stdio.h>
#include <vector>
#include <queue>
#include <string.h>
#include <algorithm>

using namespace std;

vector<int> graph[1110];
int dist[1110][1110];

int bfs(int st, int end){
    int tree[1110];
    bool vis[1110];
    int neigh, top, i;
    memset(vis, false, sizeof(vis));
    queue<int> q;
    q.push(st);
    tree[st] = st;
    while(!q.empty()){
        top = q.front();
        q.pop();
        if(top == end){
            break;
        }
        for(i=0; i<(int)graph[top].size(); i++){
            neigh = graph[top][i];
            if(dist[top][neigh] > 0 && !vis[neigh]){
                vis[neigh] = true;
                q.push(neigh);
                tree[neigh] = top;
            }
        }
    }
    if(top != end){
        return -1;
    }else{
        int mi = 1000;
        int node = top;
        while(node != st){
            mi = min(dist[tree[node]][node], mi);
            node = tree[node];
        }
        node = top;
    }
}

```

```

        while (node != st){
            dist[tree[node]][node] -= mi;
            dist[node][tree[node]] += mi;
            node = tree[node];
        }
        return mi;
    }
}

int main(){
    int n, m, st = 0, end;
    int i, j, c, np, g;
    scanf("%d %d", &m, &n);
    memset(dist, 0, sizeof(dist));
    for(i=1; i<=m; i++){
        graph[0].push_back(i);
        graph[i].push_back(0);
        dist[0][i] = 1;
        dist[i][0] = 0;
    }
    for(i=1; i<=m; i++){
        scanf("%d", &np);
        for(j=0; j<np; j++){
            scanf("%d", &g);
            graph[i].push_back(g + m);
            graph[g + m].push_back(i);
            dist[i][g+m] = 1;
            dist[g+m][i] = 0;
        }
    }
    end = n+m+1;
    for(i=m+1; i<=m+1+n; i++){
        scanf("%d", &dist[i][end]);
        dist[end][i] = 0;
        graph[i].push_back(end);
        graph[end].push_back(i);
    }

    int total=0;
    while(1){
        c = bfs(st, end);
        if(c == -1){
            printf("%d\n", total);
            break;
        }
        total += c;
    }

    return 0;
}

```

Listing 8: MST.cpp

```

// Connect disconnected graphs with weighted edges - the weight is the distance between each node (
// each node has x, y coordinates)
// Calculate the MST

struct Edge{
    int a, b;
    double w;
    bool operator< (const struct Edge& other) const{
        return other.w < w;
    }
};

int coord[755][2];
int parents[755];

```

```

int parentU(int a){
    if(parents[a] != a)
        parents[a] = parentU(parents[a]);
    return parents[a];
}
int findu(int a, int b){
    return parentU(a) == parentU(b);
}
void uni(int a, int b){
    parents[parentU(a)] = parents[parentU(b)];
}
double dist(int i, int j){
    return sqrt( (double)pow((double)coord[i][0]-coord[j][0], 2) + (double)pow((double)coord[i][1]-coord[j][1], 2) );
}

int main(){
    int n, m, i, j, n1, n2;
    double cost;
    while(scanf("%d", &n) != EOF){
        for(i=1; i<=n; i++){
            scanf("%d %d", &coord[i][0], &coord[i][1]);
            parents[i] = i;
        }
        cost=0;
        scanf("%d", &m);
        for(i=0; i<m; i++){
            scanf("%d %d", &n1, &n2);
            if(!findu(n1, n2)){
                uni(n1, n2);
            }
        }
        priority_queue<Edge> q;
        Edge ed;
        for(i=1; i<=n; i++){
            for(j=i+1; j<=n; j++){
                ed.a = i;
                ed.b = j;
                ed.w = dist(ed.a, ed.b);
                q.push(ed);
            }
        }
        while(!q.empty()){
            ed = q.top(); q.pop();
            if(!findu(ed.a, ed.b)){
                uni(ed.a, ed.b);
                cost += dist(ed.a, ed.b);
            }
        }
        printf("%.2lf\n", cost);
    }
}

```

Listing 9: unionfind.cpp

```

// UNION-FIND IMPLEMENTATION

#include <stdio.h>

int parent(int v){
    if(uf[v] == uf[uf[v]]){
        return uf[v];
    }
    uf[v] = parent(uf[v]);
    return uf[v];
}

void uni(int v1, int v2){
    int p1 = parent(v1), p2 = parent(v2);
    uf[p1] = p2;
}

```

```
}  
  
inline int find(int v1, int v2){  
    return parent(v1) == parent(v2);  
}
```


4 Geometry

Listing 10: AreaPerimeterIntersection.cpp

```
// Calculates area and perimeter of union of rectangles.

struct {
    int x1, y1, x2, y2;
}typedef Rect;

bool garden[2005][2005];
Rect rects[1005];
int xs[2005];
int ys[2005];
int mx[32670];
int my[32670];

int main(){
    int n, i=1, x, y, xi, xf, yi, yf;
    //bool tx[33000]; bool ty[33000];
    xs[0] = 0;
    ys[0] = 0;
    while(scanf("%d %d %d %d", &rects[i].x1, &rects[i].y1, &rects[i].x2, &rects[i].y2)!= EOF ){
        xs[2*i] = rects[i].x1;
        xs[2*i+1] = rects[i].x2;
        ys[2*i] = rects[i].y1;
        ys[2*i+1] = rects[i].y2;
        i++;
    }
    n = i;
    sort(xs, xs + 2*n);
    sort(ys, ys + 2*n);
    for(i=0; i<2*n; i++){
        mx[xs[i]] = i;
        my[ys[i]] = i;
    }
    memset(garden, 0, sizeof(garden));
    for(i=0; i<n; i++){
        xi = mx[rects[i].x1];
        xf = mx[rects[i].x2];
        yi = my[rects[i].y1];
        yf = my[rects[i].y2];
        //printf("%d %d %d %d\n", xi, yi, xf, yf);
        for(y=yi; y<yf; y++)
            memset(garden[y] + xi, true, xf-xi);
    }
    int a=0, p=0;
    for(x=1; x<2*n+1; x++){
        for(y=1; y<2*n+1; y++){
            //printf("%d", garden[y][x]);
            if(garden[y][x]){
                a += (xs[x+1] - xs[x]) * (ys[y+1] - ys[y]);

                if(!garden[y][x-1])
                    p += ys[y+1] - ys[y];
                if(!garden[y][x+1])
                    p += ys[y+1] - ys[y];
                if(!garden[y-1][x])
                    p += xs[x+1] - xs[x];
                if(!garden[y+1][x])
                    p += xs[x+1] - xs[x];
            }
        }
    }
    printf("%d %d\n", a, p);

    return 0;
}
```

Listing 11: CW+Intersect.cpp

```

// Art Gallery Problem - Solution each diagonal must intersect all diagonals.

#include <stdio.h>
#include <vector>
#include <string.h>

using namespace std;

struct {
    int x, y;
}typedef Point;

bool cmp(const Point &a, const Point &b){
    if(a.y < b.y)
        return true;
    else if(a.y == b.y)
        return a.x < b.x;
    return false;
}

Point points[15];
int dp[10][10];
int n;

inline bool cw(int a, int b, int c){
    return (points[b].x * points[c].y + points[c].x * points[a].y + points[a].x * points[b].y)
        - (points[b].x * points[a].y + points[c].x * points[b].y + points[a].x * points[c].y) > 0;
}

inline int intersect(int a, int b, int c, int d){
    return cw(a,c,d) != cw(b,c,d) && cw(a,b,c) != cw(a,b,d);
}

int intersect_inside(int p1, int p2){
    int i, j;
    if(dp[p1][p2])
        return 1;
    for(i=0; i<n; i++){
        for(j=i+2; j<n+i-1 && j%n > i; j++){
            if(p1 != i && p2 != i && p1 != j && p2 != j){
                //printf("%d %d %d %d\n", p1, p2, i, j);
                if(intersect(p1, p2, i, j)){
                    dp[i][j] = 1;
                    return 1;
                }
            }
        }
    }
    return 0;
}

int main(){
    int i, j, r;
    while(true){
        scanf("%d", &n);
        if(n==0)
            break;
        for(i=0; i<n; i++)
            scanf("%d %d", &points[i].x, &points[i].y);

        r=0;
        memset(dp, 0, sizeof(dp));
        for(i=0; i<n; i++){
            for(j=i+2; j<n+i-1 && j%n > i; j++){
                r = intersect_inside(i, j%n);
                if(r == 0){
                    break;
                }
            }
        }
    }
}

```

```

        }
        if(r==0)
            break;
    }

    if(r == 0)
        printf("Yes\n");
    else
        printf("No\n");
}
return 0;
}

```

Listing 12: GrahamScanAlgorithm.txt

```

Graham Scan algorithm
Step 1: Find the bottom-most point p0
Step 2:
    Sort the points in counterclockwise order of the polar angle wrt p0.
    (use CCW test in the comparison function)
Step 3: Push p 0 , p 1 onto stack S
Step 4: For i = 2 to n
    While CCW(S(before_top),S(top),p i ) = False
        Pop S.
    Push p i onto S.

```