People in buildings

Attempted by: **896** / Accuracy: **65%** / Maximum Score: **50** / ★★★☆☆ 28 Votes

Tag(s): Algorithms, Backtracking, Backtracking Basic, Hard

Brief ideas are described here:

- Binary Search / Ternary Search to find local minima
- Binary Search with Maximum flow to assign at least points
- Assume only input points as safe house centres and try over them.
- Fixing a set of safe houses by randomly picking the circumcentre of 3 points and apply heuristics.

****

Updated problem name

People in buildings

Updated problem statement

There people in your city. An is associated with each of the citizens and numbered from to . You know the exact coordinates of the location where each citizen lives. Your task is to move all these people to a building. You can build a building at any time. The building will be sustainable only if at least people are moved to the building.

You are required to move each of the citizens by using the minimum possible time. People can move one unit distance per unit time and the distance of two locations in the 2D-plane is equal to their Euclidean distance.

Your task is to determine a solution so that the required time to move all the citizens to the building is minimized.

Input format

- First line: A single inetger denoting the number of citizens
- Next line contains a single integer denoting the minimum number of people required in a building
- Next lines: Contains two integers and where the of these lines denote the coordinates of the citizen with

Output format

The format of the output is as follows:

- First line: Print that denotes the number of buildings that you can build in your solution.
- lines:
  - of these lines contain a single integer that denotes the number of citizens moved to the building
  - of these lines contain the 's of the citizens that are moved to the safe house, in any order.

Note:

- You are not required to print the minimum time that is required to move all the citizens to buildings.
- You are not required to print the coordinates of the buildings' locations.
- You are not required to minimize that denotes the number of buildings that are used in your solution.
- Your score will be calculated automatically based on the explained solution.
- The test data is randomly generated.

11 LIVE EVENTS

Constraints

IS THIS EDITORIAL HELPFUL?

Yes, it's helpful

No, it's not helpful

3 developer(s) found this editorial helpful.

Author Solution by Anik Sarker

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100005;
const int LOG = 18;
const int MAXSz = MAXN * LOG * LOG;

const int LimN = 100005;
const int LimQ = 200005;
const int LimK = 100005;


int A[MAXN];
queue<int> q;
int Size[MAXSz];
int Child[MAXSz][2];

bool Check(int x, int b) {return (x>>b)&1;}
void Initialize() {for(int i=0; i<MAXSz; i++) q.push(i);}
int CreateNode(){
    assert(q.size() >= 1);
    int x = q.front(); q.pop();
    Child[x][0] = Child[x][1] = -1;
    Size[x] = 0; return x;
}
void FreeNode(int node) {q.push(node); return;}
int Sz(int node) {return node == -1 ? 0 : Size[node];}
```

```

void Insert(int root, int n){
    int Cur = root;
    for(int i = LOG-1; i>=0; i--){
        bool ID = Check(n,i);
        if(Child[Cur][ID] == -1) Child[Cur][ID] = CreateNode();
        Cur = Child[Cur][ID];
        Size[Cur]++;
    }
}

void Delete(int root, int n){
    int Cur = root;
    int Prev = -1;

    for(int i = LOG-1; i>=0; i--){
        bool ID = Check(n,i);
        assert(Child[Cur][ID] != -1);

        Prev = Cur;
        Cur = Child[Cur][ID];
        Size[Cur]--;
        if(Size[Cur] == 0) FreeNode(Cur), Child[Prev][ID] = -1;
    }
}

struct SegmentTree{
    #define Left (node*2)
    #define Right (node*2+1)
    #define mid ((lo+hi)/2)

    int root[MAXN*5];
    vector<int> Qnode;

    void build(int node, int lo, int hi){
        root[node] = CreateNode();
        for(int x = lo; x <= hi; x++) Insert(root[node], A[x]);
        if (lo == hi) return;
        build(Left,lo,mid);
        build(Right,mid+1,hi);
    }

    void update(int node, int lo, int hi, int i, int val, bool Type){
        if(lo>hi) return;
        else if(lo>i || hi<i) return;

        if(Type) Insert(root[node],val);
        else Delete(root[node],val);
        if(lo == hi) return;

        update(Left,lo,mid,i,val,Type);
        update(Right,mid+1,hi,i,val,Type);
    }

    void splitRange(int node, int lo, int hi, int i, int j){
        if(lo>hi) return;
        else if(lo>j || hi<i) return;
        if(lo>=i && hi <= j) {Qnode.push_back(root[node]); return;}

        splitRange(Left,lo,mid,i,j);
        splitRange(Right,mid+1,hi,i,j);
    }

    int queryRange(int n, int i, int j, int k){
        Qnode.clear();
        splitRange(1,1,n,i,j);

        int Ans = 0;
        for(int i=LOG-1; i>=0; i--){
            int Total = 1<<i;
            vector <int> New;
            int Present, Missing;

            Present = 0;
            for(auto root : Qnode) Present += Sz(Child[root][0]);
            Missing = Total - Present;

            if(Missing >= k){
                for(int root : Qnode){
                    if(Child[root][0] != -1){
                        New.push_back(Child[root][0]);
                    }
                }
                Qnode = New;
            }
            else{
                Ans |= (1<<i);
                k -= Missing;
                for(int root : Qnode){
                    if(Child[root][1] != -1){
                        New.push_back(Child[root][1]);
                    }
                }
                Qnode = New;
            }
        }
        assert(k == 1);
        return Ans;
    }
};

SegmentTree tree;
int main(){
    Initialize();

    int n;
    scanf("%d",&n);
    assert(1 <= n && n <= LimN);

    for(int i=1;i<=n;i++){
        scanf("%d",&A[i]);
        assert(0 <= A[i] && A[i] < n);
    }

    tree.build(1,1,n);

    int q;
    scanf("%d",&q);
    assert(1 <= q && q <= LimQ);

    int LastAns = 0;
    for(int i=1;i<=q;i++){
        int tp;
        scanf("%d",&tp);
        assert(1 <= tp && tp <= 2);

        if(tp == 1){
            int l,r;
            scanf("%d %d",&l,&r);
            l ^= LastAns; r ^= LastAns;

            assert(1 <= l && l <= r && r <= n);

            if(l == r) continue;
            tree.update(1,1,n,l,A[l],0);
            tree.update(1,1,n,r,A[r],0);
            swap(A[l],A[r]);
            tree.update(1,1,n,l,A[l],1);
            tree.update(1,1,n,r,A[r],1);
        }
        else{
            int l,r,k;
            scanf("%d %d %d",&l,&r,&k);
            l ^= LastAns; r ^= LastAns; k ^= LastAns;

            assert(1 <= l && l <= r && r <= n);
            assert(1 <= k && k <= LimK);

            LastAns = tree.queryRange(n,l,r,k);\
            printf("%d\n",LastAns);
        }
    }

    sort(A + 1, A + n + 1);
    for(int i=1; i<=n; i++) assert(A[i] == i-1);
}
```

COMMENTS (3)

SORT BY: Relevance

Join Discussion...

Cancel Post 11

LIVE EVENTS

Rushikesh Kulkarni 10 days ago

Where are you using the coordinates of people points for finding the nearest safe house? Can you please describe the program?

▲ 1 vote ● Reply ● Message ● Permalink

James Ross Audcent 9 days ago

Good question - I too don't see how this answers the question. I'm very new here but I don't see any use of euclidian distance or time calculation and I think the program would group ids 1 and 2 together even if they were on opposite corners of the grid?

▲ 1 vote ● Reply ● Message ● Permalink

Vlad D 8 days ago

this was an approximate problem. The code above simply returns a valid solution not necessary a good one.

▲ 0 votes ● Reply ● Message ● Permalink

+1-650-461-4192

contact@hackerearth.com

Resources

Tech Recruitment Blog

Product Guides

Developer hiring guide

Engineering Blog

Developers Blog

Developers Wiki

Competitive Programming

Start a Programming Club

Practice Machine Learning

Solutions

Assess Developers

Conduct Remote Interviews

Assess University Talent

Organize Hackathons

Company

About Us

Press

Careers

Service & Support

Technical Support

Contact Us

11 LIVE EVENTS