ACM ICPC Code Notebook

UofT Blue 2018

November 7, 2018

Contents

Templates

C++

```cpp
#include <bits/stdc++.h>

using namespace std;

#define ll long long int
#define inf 0x3f3f3f3f
#define pb push_back
#define mk make_pair
#define mt make_tuple
#define fi first
#define se second
#define ii pair<int, int>
#define all(x) (x).begin(), (x).end()
#define N 1000007 // 10e6 + 7

const double PI = acos(-1.0);


int main(int argc, char const *argv[]){
        //ios::sync_with_stdio(false);

        return 0;
}
```

Java

```java
import java.io.BufferedReader;
import java.io.Closeable;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.StringTokenizer;
import java.math.*;
import java.text.*;
import java.util.*;

public class Icpc {

        public static void main(String... args) throws Exception {
                FastScan sc = new FastScan(new BufferedReader(new
                    InputStreamReader(System.in)));
                PrintWriter pw = new PrintWriter(System.out);



                sc.close();
                pw.close();
                System.exit(0);
        }

        static class FastScan implements Closeable {
                private BufferedReader br;
                private StringTokenizer tk;

                public FastScan(BufferedReader br) {
                        this.br = br;
                }

                public int in() throws NumberFormatException, IOException
                    {
                        return Integer.parseInt(next());
                }

                public long ln() throws NumberFormatException, IOException
                     {
                        return Long.parseLong(next());
```

```
38                  }
39
40                  public double db() throws NumberFormatException,
                        IOException {
41                      return Double.parseDouble(next());
42                  }
43
44                  @Override
45                  public void close() throws IOException {
46                          tk = null;
47                          br.close();
48                  }
49
50                  public String next() throws IOException {
51                          while (tk == null || !tk.hasMoreTokens()) {
52                                  String line = br.readLine();
53                                  if (line == null)
54                                          return null;
55                                  tk = new StringTokenizer(line);
56                          }
57                          return tk.nextToken();
58                  }
59          }
60  }
```

Makefile

```
1  #all:
2   # g++ -std=c++0x $(f).cpp -o $(f) -O2
3
4
5
6
7  # TO DEBUG USE THIS:
8
9  all:
10         ./gen > outGen
11         cat outGen
12         ./a < outGen > outTest
13         cat outTest
14         ./tester < outTest
15         make all
16
17
18 mt19937 rrand((time_t)ts.tv_nsec); // NEW RAND THAT GOES TO 4*10^9, this
       is quicker than rand()
19 rrand() % WHATEVER // to get the value
20
21 srand(time(NULL)) // but this is in seconds
22 mt19937 rrand(time(NULL));
23
24 struct timespec ts;
25 clock_gettime(CLOCK_MONOTONIC, &ts);
26 srand((time_t)ts.tv_nsec);      // using nano-seconds instead of seconds
27 mt19937 rrand((time_t)ts.tv_nsec);
28
29 clock_t delay = SECONDS * CLOCKS_PER_SEC; // convert seconds to clock
       ticks
30 clock_t start = clock(); // get starting clock ticks
31 while((clock() - start) < delay){}
32
33 random_shuffle(all(v)) // Dont forget to srand()!!
34 shuffle(v.begin(), v.end(), rrand) // this is better than random_shuffle
       !!!!!
```

Algorithms

Graphs

AP & Bridges

```
int dfs(int u,int p){
    dfs_num[u] = dfs_low[u] = ++dfs_counter;
    for(auto v : adjList[u]){
        if(dfs_num[v]==0){
            dfs(v,u);
            if(dfs_low[v] >= dfs_num[u]){
                articulation[u]=true;
            }
            if(dfs_low[v] > dfs_num[u])
                bridge = true;
            dfs_low[u] = min(dfs_low[u],dfs_low[v]);
        } else if(v!=p)
            dfs_low[u] = min(dfs_low[u],dfs_num[v]);
    }
}

int main(){
    memset(dfs_num,0,sizeof(dfs_num));
    memset(dfs_low,0,sizeof(dfs_low));
    bridge=false;
    dfs_counter=0;
    dfs(0,-1);
    for(int i = 0; i < N; ++i)
        if(dfs_num[i]==0)
            bridge=true;
    puts(bridge ? "Yes" : "No");
    return 0;
}
```

Centroid Decomposition

Centroid Decomposition

```
#include <bits/stdc++.h>
#define MAXN 100100
typedef long long ll;

using namespace std;

int n, sz[MAXN];
bool deleted[MAXN], vis[MAXN];
char ch[MAXN];
vector<int> g[MAXN];

void dfs(int x, int p){
        if(vis[x]) return;
        vis[x] = true;
        sz[x] = 1;
        for(auto i : g[x]){
                if(i == p || deleted[i]) continue;
                dfs(i, x);
                sz[x] += sz[i];
        }
        //cout << x << " " << sz[x] << "\n";
}

int findCentroid(int x){
        memset(vis, 0, sizeof(vis));
        dfs(x, -1);
        int p = -1, c = sz[x] / 2;
        while(true){
                bool found = false;
                for(auto i : g[x]){
                        if(!deleted[i] && i != p && sz[i] > c){
                                found = true;
                                p = x;
                                x = i;
                                break;
                        }
                }
                if(!found) return x;
        }
}

void decomp(int x, char c){
        int cen = findCentroid(x);
        ch[cen] = c;
        deleted[cen] = true;
        for(auto i : g[cen]){
                if(deleted[i]) continue;
                decomp(i, c + 1);
```

```
        }
}

int main(){
    #ifndef ONLINE_JUDGE
                freopen("input.txt", "r", stdin);
        #endif
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        memset(deleted, 0, sizeof(deleted));
    cin >> n;
    for(int i = 0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    //cout << findCentroid(1);
    decomp(1, 'A');
    for(int i = 1; i <= n; i++){
        cout << ch[i] << " ";
    }
}
```

Centroid Decomposition (Eoin)

```
void fill_sz(int u, int p){
    sz[u] = 1;
    for(int v : adjList[u]){
        if(v==p || mkd[v])
            continue;
        fill_sz(v,u);
        sz[u]+=sz[v];
    }
}

int get_centroid(int u, int n, int p){
    for(int v : adjList[u]){
        if(v==p || mkd[v])
            continue;
        if(sz[v] > n/2)
            return get_centroid(v, n, u);
    }
    return u;
}

int decomp(int u){
    fill_sz(u, -1);
    int cent = get_centroid(u, sz[u], -1);
    mkd[cent] = true;
    for(int v : adjList[cent]){
        if(mkd[v])
            continue;
        int r = decomp(v);
        centP[r] = cent;
    }
    }
    return cent;
}
```

Network Flow

Dinic's Blocking flow

```
/**
        e-maxx's flow
        Dinic algorithm
        Complexity O(V^2*E) OR O(Vsrqt(E)) for bipartite graphs!

*/

struct edge {
        int a, b, cap, flow;
};

int nodes, s, t, d[N], ptr[N], q[N];                    // NEED TO set n (
    max of nodes), s source, t sink
vector<edge> e;
vector<int> g[N];

void add_edge (int a, int b, int cap) {
        edge e1 = { a, b, cap, 0 };
        edge e2 = { b, a, 0, 0 };
        g[a].push_back ((int) e.size());
        e.push_back (e1);
        g[b].push_back ((int) e.size());
        e.push_back (e2);
}

bool bfs() {
        int qh=0, qt=0;
        q[qt++] = s;
        memset (d, -1, nodes * sizeof d[0]);
        d[s] = 0;
        while (qh < qt && d[t] == -1) {
                int v = q[qh++];
                for (size_t i=0; i<g[v].size(); ++i) {
                        int id = g[v][i],
                                to = e[id].b;
                        if (d[to] == -1 && e[id].flow < e[id].cap) {
                                q[qt++] = to;
                                d[to] = d[v] + 1;
                        }
                }
        }
        return d[t] != -1;
}

int dfs (int v, int flow) {
        if (!flow)  return 0;
        if (v == t)  return flow;
```

```
        for (; ptr[v]<(int)g[v].size(); ++ptr[v]) {
                int id = g[v][ptr[v]],
                        to = e[id].b;
                if (d[to] != d[v] + 1)  continue;
                int pushed = dfs (to, min (flow, e[id].cap - e[id].flow));
                if (pushed) {
                        e[id].flow += pushed;
                        e[id^1].flow -= pushed;
                        return pushed;
                }
        }
        return 0;
}

int dinic() {
        int flow = 0;
        for (;;) {
                if (!bfs())  break;
                memset (ptr, 0, nodes * sizeof ptr[0]);
                while (int pushed = dfs (s, inf))
                        flow += pushed;
        }
        return flow;
}
```

Edmond Karp

```
void aug(int u, int minE){
    if(u==S){ f=minE; return; }
    if(p[u]!=u){
        aug(p[u],min(minE,res[p[u]][u]));
        res[p[u]][u]-=f;
        res[u][p[u]]+=f;
    }
}

int main(){
    int mf=0;
    for(;;){
        f=0;//Global
        for(int i = 0; i < N; i++)
            dist[i]=INF,p[i]==i;
        dist[S]=0;
        queue<int> q; q.push(S);
        while(!q.empty()){
            int u = q.front(); q.pop();
            if(u==T) break;
            for(int i = 0; i < N; i++)
                if(res[u][i] > 0 && dist[i]==INF)
                    dist[i]=dist[u]+1, p[i]=u, q.push(i);
        }
        aug(T,INF);
        if(f==0) break;
        mf+=f;
    }
    vector<ii> used;
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
            if(graph[i][j] > 0 && res[i][j] < graph[i][j])
                used.push_back(make_pair(i,j));
}
```

Ford Fulkerson 1

```
int ff(int u, int minE){
    if(u==T)
        return minE;
    vis[u]=true;
    for(auto i : adjList[u]){
        if(!vis[i] && res[u][i] > 0){
            if(int f = ff(i, min(minE,res[u][i]))){
                res[u][i] -= f;
                res[i][u] += f;
                return f;
            }
        }
    }
    return 0;
}

int main(){
    int mf = 0;
    while(1){
        memset(vis,0,sizeof(vis));
        int f = ff(S,INF);
        if(f==0)
            break;
        mf+=f;
    }
    printf("%d\n",mf);
}
```

Ford Fulkerson 2

```
#include <bits/stdc++.h>
#define MAXN 3000
typedef long long ll;

using namespace std;

int n;
int g[MAXN][MAXN], rg[MAXN][MAXN], parent[MAXN];

bool bfs(int source, int sink){
        bool visited[MAXN];
        memset(visited, 0, sizeof(visited));
        queue<int> q;
        q.push(source);
        visited[source] = true;
        parent[source] = -1;
```

```
        while(!q.empty()){
                int i = q.front();
                q.pop();
                for(int j = 0; j < MAXN; j++){
                        if(!visited[j] && rg[i][j] > 0){
                                q.push(j);
                                parent[j] = i;
                                visited[j] = true;
                        }
                }
        }
        return visited[sink];
}

int maxFlow(int source, int sink){
        for(int i = 0; i < MAXN; i++){
                for(int j = 0; j < MAXN; j++){
                        rg[i][j] = g[i][j];
                }
        }
        int max_flow = 0;
        while(bfs(source, sink)){
                int path_flow = 99999999;
                for(int i = sink; i != source; i = parent[i]){
                        int j = parent[i];
                        path_flow = min(path_flow, rg[j][i]);
                }
                for(int i = sink; i != source; i = parent[i]){
                        int j = parent[i];
                        rg[j][i] -= path_flow;
                        rg[i][j] += path_flow;
                }
                max_flow += path_flow;
        }
        return max_flow;
}

int main(){
    #ifndef ONLINE_JUDGE
                freopen("input.txt", "r", stdin);
        #endif
        ios_base::sync_with_stdio(false);
        for(int i = 0; i < MAXN; i++){
                for(int j = 0; j < MAXN; j++){
                        rg[i][j] = g[i][j] = 0;
                }
        }
}
```

Lowest Common Ancestor

Lowest Common Ancestor (Using RMQ)

```
/*
 * H[u] is first visit of u
 * E[x] is vertex at time x
 * L[x] is depth at time x
 */
void vis(int u, int d){
    H[u]=vind;
    E[vind] = u;
    L[vind++] = d;
    for(auto i : adjList[u]){
        if(H[i]!=-1)
            continue;
        vis(i,d+1);
        E[vind] = u;
        L[vind++] = d;
    }
}

int LCA(int u, int v){
    if(H[u] > H[v]){
        int t = u;
        u = v;
        v = t;
    }
    //run some range min query on L
    //between H[u] and H[v]
    int ind = rmq(H[u],H[v]);
    return E[ind];
}

int dist(int u, int v){
    int a = H[u];
    int b = H[v];
    int ind = LCA(u,v);
    return abs(L[H[ind]]-L[a])
        + abs(L[H[ind]]-L[b]);
}
```

Lowest Common Ancestor (Using Binary Lifting)

```
#include <bits/stdc++.h>
#define MAXN 100100
typedef long long ll;

using namespace std;

int n, m, s[MAXN], depth[MAXN], anc[MAXN][40];
vector<int> g[MAXN];
bool vis[MAXN];

int dfs(int x, int d, int p){
        vis[x] = true;
        depth[x] = d;
        s[x] = 1;
        anc[x][0] = p;
        for(int i = 1; pow(2, i) <= d; i++){
                anc[x][i] = anc[anc[x][i - 1]][i - 1];
```

```
        }
        for(int i = 0; i < g[x].size(); i++){
                if(vis[g[x][i]]) continue;
                s[x] += dfs(g[x][i], d + 1, x);
        }

        return s[x];
}

int walk(int x, int d){
        int i = 0;
        while(d){
                if(d & 1) x = anc[x][i];
                d /= 2;
                i++;
        }
        //cout << "\n";
        return x;
}

int lca(int x, int y){
        //cout << x<<y;

        if(depth[x] < depth[y]) y = walk(y, depth[y] - depth[x]);
        if(depth[x] > depth[y]) x = walk(x, depth[x] - depth[y]);
        //cout << x<<y;
        if(x == y) return x;
        for(int i = 30; i >= 0; i--){
                if(depth[x] >= pow(2, i) && anc[x][i] != anc[y][i]){
                        return lca(anc[x][i], anc[y][i]);
                }
        }
        return anc[x][0];
}

int main(){
        ios_base::sync_with_stdio(false);
    cin >> n;
    for(int i = 0; i < n - 1; i++){
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    dfs(1, 0, -1);
    cin >> m;
    for(int i = 0; i < m; i++){
        int a, b;
        cin >> a >> b;
        if(depth[a] > depth[b]) swap(a, b);
        if(a == b) cout << n;
        else{
                int l = lca(a, b);
                int d = -2 * depth[l] + depth[a] + depth[b];
                if(d % 2) cout << "0";
                else{
                        if(depth[a] == depth[b]) cout << s[1] - s[walk(b,
                            d / 2 - 1)] - s[walk(a, d / 2 - 1)];
                        else cout << s[walk(b, d / 2)] - s[walk(b, d / 2 -
                            1)];
                }
        }
        cout << "\n";
    }

}
```

Minimum Spanning Tree (Kruskal's)

```
struct edge {
    int x,y,w;
    bool operator < (edge e) const {
        return w < e.w;
    }
};

int main(){
    vector<edge> eList; //Input
    for(int i = 0; i < N; i++)// Set up UFDS
        p[i]=i;
    vector<ii> treeList;
    sort(eList.begin(),eList.end());
    int cost = 0;
    int sz=N;
    int u,v,w;
    for(const auto &i : eList){
        v=i.x; u=i.y; w=i.w;
        if(!connected(u,v)){
            join(u,v);
            treeList.push_back({min(u,v),max(u,v)});
            sz--;
            cost+=w;
        }
    }
    if(sz!=1)
        puts("Impossible");
}
```

Strongly Connected Components (Tarjan's)

```
typedef pair<int, int> ii;

int N,M;
vector<int> adjList[MX_N];
int dfs_num[MX_N],dfs_low[MX_N];
bool vis[MX_N];
stack<int> scc;
int dfsCounter=1;
int sccIdx=1;
```

```
10
11 map<int, int> sccMap;
12
13 void tarjans(int u){
14     scc.push(u);
15     vis[u]=true;
16
17     dfs_low[u]=dfs_num[u]=dfsCounter++;
18
19     for(int i = 0; i < adjList[u].size(); i++){
20         int v = adjList[u][i];
21         if(dfs_num[v]==0){
22             tarjans(v);
23             dfs_low[u]=min(dfs_low[u],dfs_low[v]);
24         } else if(vis[v]){
25             dfs_low[u]=min(dfs_low[u],dfs_num[v]);
26         }
27     }
28     if(dfs_low[u]==dfs_num[u]){
29         while(1){
30             int v = scc.top(); scc.pop();
31             sccMap[v]=sccIdx;
32             vis[v]=false;
33             if(v==u)
34                 break;
35         }
36         sccIdx++;
37     }
38 }
```

Bipartite Check

```
1  int n;
2  vector<vector<int>> adj;
3
4  vector<int> side(n, -1);
5  bool is_bipartite = true;
6  queue<int> q;
7  for (int st = 0; st < n; ++st) {
8      if (side[st] == -1) {
9          q.push(st);
10         side[st] = 0;
11         while (!q.empty()) {
12             int v = q.front();
13             q.pop();
14             for (int u : adj[v]) {
15                 if (side[u] == -1) {
16                     side[u] = side[v] ^ 1
17                     q.push(u);
18                 } else {
19                     is_bipartite &= side[u] != side[v];
20                 }
21             }
22         }
23     }
24 }
25
26 cout << (is_bipartite ? "YES" : "NO") << endl;
```

Dijkstras

```
1  #include <bits/stdc++.h>
2  #include <utility>
3  #define MAXN 505
4
5  using namespace std;
6
7  typedef long long ll;
8  typedef pair<int, int> ii;
9  int n;
10
11 vector<pair<int, int> > g[MAXN];
12 int dist[MAXN];
13
14 void dijkstra(int x){
15         for(int i = 0; i < n; i++){
16                 dist[i] = 99999999;
17         }
18         priority_queue<pair<int, int>, vector<pair<int, int> >, greater<
               pair<int, int> > > pq;
19         pq.push({0, x});
20         dist[x] = 0;
21         while(!pq.empty()){
22                 pair<int, int> v = pq.top();
23                 pq.pop();
24                 for(int i = 0; i < g[v.second].size(); i++){
25                         pair<int, int> u = g[v.second][i];
26                         if(dist[v.second] + u.second < dist[u.first])
27                                 pq.push({dist[u.first] = dist[v.second] +
                                       u.second, u.first});
28                 }
29         }
30 }
31
32 int main(){
33     #ifndef ONLINE_JUDGE
34                 freopen("input.txt", "r", stdin);
35         #endif
36         ios_base::sync_with_stdio(false);
37     //cin >> n;
38 }
```

Find Cycles

```
1  int n;
2  vector<vector<int>> adj;
3  vector<char> color;
4  vector<int> parent;
5  int cycle_start, cycle_end;   // In O(M)
```

```
6
7  bool dfs(int v) {
8      color[v] = 1;
9      for (int u : adj[v]) {
10         if (color[u] == 0) {
11             parent[u] = v;
12             if (dfs(u))
13                 return true;
14         } else if (color[u] == 1) {
15             cycle_end = v;
16             cycle_start = u;
17             return true;
18         }
19     }
20     color[v] = 2;
21     return false;
22 }
23
24 void find_cycle() {
25     color.assign(n, 0);
26     parent.assign(n, -1);
27     cycle_start = -1;
28
29     for (int v = 0; v < n; v++) {
30         if (dfs(v))
31             break;
32     }
33
34     if (cycle_start == -1) {
35         cout << "Acyclic" << endl;
36     } else {
37         vector<int> cycle;
38         cycle.push_back(cycle_start);
39         for (int v = cycle_end; v != cycle_start; v = parent[v])
40             cycle.push_back(v);
41         cycle.push_back(cycle_start);
42         reverse(cycle.begin(), cycle.end());
43
44         cout << "Cycle found: ";
45         for (int v : cycle)
46             cout << v << " ";
47         cout << endl;
48     }
49 }
```

Arrays

Longest Increasing Subsequence

```
1  int ls[MX_N];
2  int L[MX_N];
3  int I[MX_N];
4
5  void nlogn(){
6      for(int i = 1; i < N+1; ++i)
7          I[i]=INF;
8      I[0] = -INF;
9      int mx = 1;
10     for(int i = 0; i < N; ++i){
11         int ind = lower_bound(I,I+N+1,ls[i]) - I;
12         I[ind] = ls[i];
13         L[i] = ind;
14         mx = max(mx,ind);
15     }
16     int prv = INF;
17     vector<int> out;
18     for(int i = N-1; i >= 0; --i){
19         if(ls[i] < prv && L[i]==mx){
20             out.push_back(ls[i]);
21             prv = ls[i];
22             mx--;
23         }
24     }
25 }
```

Data Structures

Fenwick Tree

```
int tree[MX_N];
int N;
int lsOne(int i){ return i&(-i); }
void update(int k,int v){
    for(; k<MX_N; k+=lsOne(k))
        tree[k]+=v;
}
int query(int k){
    int cnt=0;
    for(; k; k-=lsOne(k)){
        cnt+=tree[k];
    }
    return cnt;
}
```

Segment Trees

Segment Tree Lazy

```
typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);

struct SegT {
        vector <ll> seg, lazy;
        int n;

        SegT () {}

        SegT (int n) {
                this->n = n;
                seg.resize(4*n + 1);
                lazy.resize(4*n + 1);
        }

        void prop (int r, int i, int j) {

                seg[r] += lazy[r] * (j-i+1);

                if (i != j) {
                        lazy[2*r] += lazy[r];
                        lazy[2*r + 1] += lazy[r];
                }

                lazy[r] = 0;
        }

        int a, b;
        ll update (int r, int i, int j, ll val) {
                prop (r, i, j);
                if (j < a or i > b)     return 0LL;

                if (i >= a and j <= b) {
                        lazy[r] += val;
                        prop (r, i, j);
                        return seg[r];
                } else {
                        int mid = (i + j)/2;
                        ll L = update (2*r, i, mid, val);
                        ll R = update (2*r + 1, mid + 1, j, val);
                        return L + R;
                }
        }

        ll update (int l, int r, ll val) {
                a = l;  b = r;
                return update (1, 0, n - 1, val);
        }

        ll query (int l, int r) {
                return update (l, r, 0);
        }

};
```

Segment Tree (Eoin)

```
int tree[MX_N*4];
int a[MX_N];
int N;

void construct(int p, int L, int R){
```

```
6       if(L==R){
7           tree[p] = a[L];
8           return;
9       }
10      if(R<L)
11          return;
12      int md = (L+R)/2;
13      construct(2*p,L,md);
14      construct(2*p+1,md+1,R);
15      tree[p] = min(tree[2*p],tree[2*p+1]);
16  }
17
18  void update(int p, int L, int R, int ind,int v){
19      if(L==R){
20          a[ind] = v;
21          tree[p] = v;
22          return;
23      }
24      int md = (L+R)/2;
25      if(ind <= md)
26          update(2*p,L,md,ind,v);
27      else
28          update(2*p+1,md+1,R,ind,v);
29      tree[p] = min(tree[2*p],tree[2*p+1]);
30  }
31
32  int rmq(int p, int L, int R, int l, int r){
33      if(r < L || l > R)
34          return INF;
35      if(l>=L && r<=R)
36          return tree[p];
37      int md = (l+r)/2;
38      return min(rmq(2*p,L,R,l,md),rmq(2*p+1,L,R,md+1,r));
39  }
```

Iterative Segment Tree

```
1   int t[2*N], n; // When debugging, the prob is most likely that you have
        multiple n's. need this one here!
2
3   int query(int l, int r){ // This r is exclusive!
4           int ans=0;
5           for(l+=n, r+=n; l<r; l>>=1, r>>=1){
6                   if(l&1)ans+=t[l++];
7                   if(r&1)ans+=t[--r];
8           }
9           return ans;
10  }
11
12  void update(int p, int v){
13          for(t[p+=n]+=v; p>1; p>>=1){
14                  t[p>>1]=t[p]+t[p^1];
15          }
16  }
```

Sparse Table

```
1   inline int rmq(int u, int v){
2       if(u > v)
3           return -2000000000;
4       int k=(int)floor(log2((double)(v-u+1)));
5       if(r[mtable[u][k]]>
6               r[mtable[v-(1<<k)+1][k]])
7           return mtable[u][k];
8       return mtable[v-(1<<k) + 1][k];
9   }
10
11  for(int i = 0; i < N; i++)
12      mtable[i][0] = i;
13  for(int j = 1; (1 << j) <= N; j++)
14      for(int i = 0; i + (1<<j)-1 < N; ++i)
15          if(r[mtable[i][j-1]]
16                  >r[mtable[i+(1<<(j-1))][j-1]])
17              mtable[i][j]= mtable[i][j-1];
18          else
19              mtable[i][j]=mtable[i+(1<<(j-1))][j-1];
```

Union Find

```
1   /**
2           Union find algorithm
3           Complexity O(log n) for Join or Find.
4   */
5
6   int pai[N];
7
8   void init(int n){
9           for(int i=1; i<=n; i++){
10                  pai[i]=i;
11          }
12  }
13
14  int find(int i){
15          if(pai[i]==i)return i;
16          return pai[i]=find(pai[i]);
17  }
18
19  int join(int a, int b){
20          a=find(a);
21          b=find(b);
22          pai[a]=pai[b];
23  }
```

Strings

KMP

KMP

```
/*
        border = proper prefix that is suffix
        p[i] = length of longest border of prefix of length i, s[0...i-1]
*/

typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);

const int N = 1e6 + 6;
int pi[N];
string p, t;

void pre () {
        p += '#';

        pi[0] = pi[1] = 0;
        for (int i = 2; i <= (int)p.size(); i++) {
                pi[i] = pi[i-1];

                while (pi[i] > 0 and p[pi[i]] != p[i-1])
                        pi[i] = pi[pi[i]];

                if (p[pi[i]] == p[i-1])
                        pi[i]++;
        }
}

void report (int at) {

}

void KMP () {
        pre ();

        int k = 0;
        int m = p.size() - 1;

        for (int i = 0; i < (int)t.size(); i++) {
                while (k > 0 and p[k] != t[i])
                        k = pi[k];

                if (p[k] == t[i])
                        k++;
                if (k == m)
                        report (i - m + 1);
        }

}

int main (void) {
        ios_base::sync_with_stdio(false);

        return 0;
}
```

KMP (Eoin)

```
vector<int> buildFailure(string s){
    vector<int> T(n+1,0);
    T[0]=-1;
    int j = 0;
    for(int i = 1; i < s.size();++i){
        if(s[i]==s[j]){
            T[i]=T[j];
            j++;
        } else{
            T[i] = j;
            j = T[j];
            while(j >= 0 && s[i]!=s[j])
                j = T[j];
            j++;
        }
    }
    T[s.size()] = j;
    return T;
}
vector<int> search(string W, string S){
    auto T=buildFailure(W);
    vector<int> p;
```

```
    int k = 0;
    int j = 0;
    while(j < S.size()){
        if(W[k]==S[j]){
            k++; j++;
            if(k==W.size()){
                p.push_back(j-k);
                k = T[k];
            }
        }else{
            k = T[k];
            if(k < 0)
                j+=1, k+=1;
        }
    }
    return p;
}
```

Rabin Karp

```
// Looks for a pattern s in text t in O(n+m) time.
// Returns where the occurences are

vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();

    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;

    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;

    vector<int> occurences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurences.push_back(i);
    }
    return occurences;
}
```

Suffix Array

```
void countingSort(int k){
    int i,sum,maxi=max(300,N);
    memset(c,0,sizeof(c));
    for(i = 0; i < N; i++)
        c[i+k < N ? RA[i+k] : 0]++;
    for(i=sum=0; i < maxi; i++){
        int t = c[i];
        c[i]=sum;
        sum+=t;
    }
    for(i = 0; i < N; i++)
        tempSA[c[SA[i]+k < N
            ? RA[SA[i]+k]: 0]++] = SA[i];
    for(i=0; i < N;i++)
        SA[i]=tempSA[i];
}

int main(){
    for(int i = 0; i < N; i++)
        SA[i]=i,RA[i]=input[i];
    int r;
    for(int k = 1; k < N; k <<= 1){
        countingSort(k);
        countingSort(0);
        tempRA[SA[0]]=r=0;
        for(int i = 1; i < N; i++){
            tempRA[SA[i]]
                =(RA[SA[i]]==RA[SA[i-1]]
                && RA[SA[i]+k]==RA[SA[i-1]+k]
                ? r:++r);
        }
        for(int i = 0; i < N; i++)
            RA[i]=tempRA[i];
    }
    return 0;
}
```

Manacher's algorithm for longest palindromic substring

```
/* Manacher s algorithm O(N), time and memory, algorithm to find
 * longest palindromic substring
 *
 * Transform initial string t into s,
 * puting separators between characters
 *
 * Build vector p[], where p[i] is the length of the
 * palindrome centered at s[i]
 *
 * Works for both, odd and even length
 * s: # a # b # a #             # a # a #
 * p: 0 1 0 3 0 1 0             0 1 2 1 0
 *
 * p built in O(N) using the fact that elements can be simetric
 * given some center and p[center]:
```

```
 * If we are in i and center c, i_mirror = c - (i - c), if p[i_mirror]
 * fits in center + p[center], p[i] is p[i_mirror], else we need to
 * check real value of p[i]
 * If we call the border center + p[center], r. Its easy to see
 * r is only increased, achieving the O(N) time complexity
 *
 * Longest palindromic substring is the maximum element in p
 *
 */

typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);

const int N = 1e6 + 5;
int p[2*N + 2];

int main (void) {
        ios_base::sync_with_stdio(false);

        string s, t;    cin >> t;
        s += "#";
        for (auto c : t) {
                s += c;
                s += '#';
        }

        int n = s.size();
        int c = 0, r = 0;
        for (int i = 0; i < n; i++) {
                int i_mirror = c - (i - c);

                if (i <= r)
                        p[i] = min (p[i_mirror], r - i);
                else
                        p[i] = 0;

                while (i - 1 - p[i] >= 0 and i + 1 + p[i] < n and s[i + 1
                    + p[i]] == s[i - 1 - p[i]]) {
                        p[i]++;
                }

                if (i + p[i] > r) {
                        c = i;
                        r = i + p[i];
                }
        }

        int len = 0, center = 0;
        for (int i = 0; i < n; i++)
                if (p[i] > len) {
                        len = p[i];
                        center = i;
                }

        /* not tested */
        string res;
        for (int i = 0; i < n; i++)
                if (i >= center - len and i <= center + len and s[i] != '#
                    ')
                        res += s[i];
        /* */

        cout << len << endl;
        cout << res << endl;

        return 0;
}
```

Z function

```
/*          {0, if i = 0
    z[i] =  {length longest commom prefix of s and s[i...n-1]
*/

typedef long long ll;
typedef pair<int, int> ii;
const int INF = 0x3f3f3f3f;
const double PI = acos(-1.0);

const int N = 2e5 + 5;
string s;
int z[N];

void go () {
        int l = 0, r = 0;
        int n = s.size();
        memset (z, 0, sizeof z);

        for (int i = 1; i < n; i++) {
                if (i <= r)
                        z[i] = min (z[i-l], r - i + 1);
                while (z[i] + i < n and s[z[i] + i] == s[z[i]])
                        z[i]++;
                if (r < i + z[i] - 1) {
                        l = i;
                        r = i + z[i] - 1;
                }
        }
}
```

Trie

```
struct node {
    node * children[26];
    int count;
    node(){
```

```
5           memset(children,0,sizeof(children));
6           count=0;
7       }
8   };
9
10  void insert(node* nd, char *s){
11      if(*s){
12          if(!nd->children[*s-'a'])
13              nd->children[*s-'a']=new node();
14          insert(nd->children[*s-'a'],s+1);
15      }
16      nd->count++;
17  }
18
19  int count(node* nd, char *s){
20      if(*s){
21          if(!nd->children[*s-'a'])
22              return 0;
23          return count(nd->children[*s-'a'],s+1);
24      } else {
25          return nd->count;
26      }
27  }
```

Hash

```
1   typedef long long ll;
2   typedef pair<ll, ll> ii;
3
4   ll md (ll x, ll mod) {
5           x %= mod;
6           if (x < 0)
7                   return x + mod;
8           return x;
9   }
10
11  struct Hash {
12          const ll base = 31;
13          ll mod, *h, *pot;
14          string s;
15
16          Hash () {}
17
18          void build (string s, ll mod) { // O(n)
19                  this->mod = mod;
20                  this->s = s;
21                  h = new ll [s.size() + 2];
22                  pot = new ll [s.size() + 2];
23
24                  h[0] = s[0] - 'a';
25                  for (int i = 1; i < (int)s.size(); i++)
26                          h[i] = (h[i-1]*base + s[i] - 'a')%mod;
27
28                  pot[0] = 1LL;
29                  for (int i = 1; i < (int)s.size(); i++)
30                          pot[i] = (pot[i-1] * base)%mod;
31          }
32
33          ll query (int l, int r) { // O(1)
34                  ll R = h[r], L = 0;
35
36                  if (l)
37                          L = (h[l-1] * pot[r - l + 1])%mod;
38
39                  return md (R - L, mod);
40          }
41
42  } h[2];
43
44  // returns if s[i, i + ilen - 1] is lexicographically smaller than s[j, j
        + jlen - 1]
45  // not tested if ilen != jlen
46  bool comp (string &s, int i, int ilen, int j, int jlen) {
47          int bot = 0, top = min (ilen, jlen) - 1;
48          int id = -1;
49
50          while (bot <= top) {
51                  int mid = (bot + top)>>1;
52
53                  ii pi = ii(h[0].query(i, i + mid), h[1].query(i, i + mid))
                        ;
54                  ii pj = ii(h[0].query(j, j + mid), h[1].query(j, j + mid))
                        ;
55                  if (pi == pj) {
56                          bot = mid + 1;
57                          id = mid;
58                  } else {
59                          top = mid - 1;
60                  }
61          }
62
63          if (id == min (ilen, jlen) - 1) {
64                  if (ilen != jlen)
65                          return ilen < jlen;
66                  return i < j;
67          }
68
69          return s[i + id + 1] < s[j + id + 1];
70  }
71
72  const ll mod[2] = {1000000007, 1000000009};
```

Geometry

Convex Hull

Convex Hull

```
// Implementation of Andrew's monotone chain 2D convex hull algorithm.
// Asymptotic complexity: O(n log n).

typedef double coord_t;    // coordinate type
typedef double coord2_t;  // must be big enough to hold 2*max(|coordinate|)^2

struct Point {
        coord_t x, y;
        Point(){}
        Point(coord_t xx, coord_t yy){
                x=xx, y=yy;
        }
        bool operator <(const Point &p) const {
                return x < p.x || (x == p.x && y < p.y);
        }
};

// 2D cross product of OA and OB vectors, i.e. z-component of their 3D cross product.
// Returns a positive value, if OAB makes a counter-clockwise turn,
// negative for clockwise turn, and zero if the points are collinear.
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){
        size_t n = P.size(), k = 0;
        if (n <= 3) return P;
        vector<Point> H(2*n);

        // Sort points lexicographically
        sort(P.begin(), P.end());

        // Build lower hull
        for (size_t i = 0; i < n; ++i) {
                while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--; // Remove the last "=" if you want to get max points in the hull
                H[k++] = P[i];
        }

        // Build upper hull
        for (size_t i = n-1, t = k+1; i > 0; --i) {
                while (k >= t && cross(H[k-2], H[k-1], P[i-1]) <= 0) k--;
                // Remove the last "=" if you want to get max points in the hull
                H[k++] = P[i-1];
        }

        H.resize(k-1);
        return H;
}

/*
 *  Note that this is using double. Its better to use long long because double might be TLE!
*/
```

Convex Hull (Eoin)

```
int main(){
    for(int i = 0; i < N; i++){
        perm[i]=i;
    }
    sort(perm,perm+N,
            [](int a, int b){
                const point &pa = V[a];
                const point &pb = V[b];
                if(real(pa)!=real(pb))
                    return real(pa) < real(pb);
                return imag(pa) < imag(pb);
            });
    vector<int> L; vector<int> U;
    for(int i = 0; i < N;){
        int t = L.size();
        if(t >= 2 && !ccw(V[L[t-2]],V[L[t-1]],V[perm[i]]))
            L.pop_back();
        else
```

```
            L.push_back(perm[i++]);
    }
    for(int i = N-1; i >=0;){
        int t = U.size();
        if(t >= 2 && !ccw(V[U[t-2]],V[U[t-1]],V[perm[i]]))
            U.pop_back();
        else
            U.push_back(perm[i--]);
    }
    vector<int> hull;
    for(int i = 0; i < L.size()-1; ++i)
        hull.push_back(L[i]);
    for(int i = 0; i < U.size()-1; ++i)
        hull.push_back(U[i]);
    return 0;
}
```

Geometry Functions

```
typedef complex<double> pt;
typedef complex<double> vec;
typedef vector<pt> pgon;
typedef struct { pt p,q; } lseg;
double cross(const vec& a, const vec &b){
    return x(a)*y(b)-y(a)*x(b);
}
//cross product of (b-a) and (c-b), 0 is collinear
int orientation(const pt& a,
        const pt& b, const pt& c){
    double v = cross(b-a,c-b);
    if(abs(v-0.0)<EPS)
        return 0;
    return v > 0 ? 1 : 2;
}
//Line segment intersection
bool intersects(const lseg& a, const lseg& b){
    if(a.q == b.p || b.q == a.p)
        return false;
    if(orientation(a.p,a.q,b.p)
            !=orientation(a.p,a.q,b.q)
            && orientation(b.p,b.q,a.p)
            != orientation(b.p,b.q,a.q))
        return true;
    return false;
}
//Area of polygon
double area(const pgon& p){
    double area = 0.0;
    for(int i = 1; i < p.size(); ++i)
        area+=cross(p[i-1],p[i]);
    return abs(area)/2.0;
}
//If a->b->c is a counterclockwise turn
double ccw(const point& a, const point& b,
        const point& c){
    if(a==b || b==c || a==c)
        return false;
    point relA = b-a;
    point relC = b-c;
    return cross(relA,relC) >= 0.0;
}
//Returns if point p is in the polygon poly
bool inPoly(const pgon& poly, const pt& p){
    for(int i = 0; i < poly.size()-1; i++){
        if(!ccw(poly[i],p,poly[i+1]))
            return false;
    }
    return true;
}
//Distance from p to line (a,b)
double distToLine(const pt& p, const pt& a,
        const pt &b){
    vec ap = p-a;
    vec ap = b-a;
    double u = dot(ap,ab)/dot(ab,ab);
    //Ignore for non-line segment
    if(u < 0.0) //Closer to a
        return abs(a-p);
    if(u > 1.0) //Closer to b
        return abs(b-p);
    pt c = a+ab*u; // This is the point
    return abs(c-p);
}
```

```
double area(vector<Point>v){    // Return the area of the convex hull in O(n).
        double ret=0.0;
        int n=v.size();
        for(int i=0; i<v.size(); i++){
                ret+=v[i].x*(v[(i+1+n)%n].y-v[(i-1+n)%n].y);
        }
        return abs(ret/2);
}

double perimeter(vector<Point> v){      // Return the perimeter of the convex hull in O(n).
        double ans=0.0;
        v.pb(v[0]);
        for(int i=0; i<v.size()-1; i++){
                ans+=sqrt((v[i].x-v[i+1].x)*(v[i].x-v[i+1].x)+(v[i].y-v[i+1].y)*(v[i].y-v[i+1].y));
        }
        return ans;
}

Point rotate(Point c, Point po, double ang){    // Rotate po, around c by ang.
        /*
         * C is the center of the rotation
```

```
22            * Po is the point to be rotated by ang in rad .
23            * Ang is the angle in radians to be rotated counter-clockwise.
24            */
25           Point p;
26           p.x=c.x+(po.x-c.x)*cos(ang)-(po.y-c.y)*sin(ang);
27           p.y=c.y+(po.x-c.x)*sin(ang)-(po.y-c.y)*cos(ang);
28           return p;
29   }
```

Math

Binomial Coefficients

```
ll ncrmem[MX_N][MX_N];

ll ncr(int n, int r){
        if(n==0)
                return r==0;
        if(r==0)
                return 1;
        if(ncrmem[n][r] != -1)
                return ncrmem[n][r];
        return ncrmem[n][r] = ncr(n-1, r-1) + ncr(n-1, r);
}
```

Ternary Search

```
double ternary_search(double l, double r) {
    double eps = 1e-9;              //set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);      //evaluates the function at m1
        double f2 = f(m2);      //evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);                    //return the maximum of f(x) in [l, r]
}
```

Miller Rabin primality test

```
void factor(ll x, ll& e, ll& k){
    while(x%2LL==0LL){
        x/=2LL;
        ++e;
    }
    k = x;
}

//increase x for higher certainty, 5 works well
bool is_prime(ll n, int x){
    if(n&2LL==0 || n==1LL)
        return false;
    if(n==2 || n==3 || n==5 || n==7)
        return true;
    ll e, k;
    factor(n-1,e,k);
    while(x-->0){
        ll a = (rand())%(n-5LL) + 2LL;
        ll p = mod_exp(a,k,n);
        if(p==1LL || p==n-1LL)
            continue;
        bool all_fail = true;
        for(int i = 0; i < e-1; ++i){
            p = mod_exp(p, 2, n);
            if(p==n-1LL){
                all_fail = false;
                break;
            }
        }
        if(all_fail)
            return false;
    }
    return true;
}
```

Fast fourier transform

```
/* emaxx implementation */
/* Multiplication with arbitrary modulos
 *    use ntt if mod is prime and can be written as 2**k * c + 1
 *    if not, use Chinese Reminder Theorem
 *    or transform A(x) = A1(x) + A2(x)*c decompose into A(x)/c and A(x)%c
 *                 B(x) = B1(x) + B2(x)*c
 *        where c ~= sqrt(mod)
 *        A * B = A1*B1 + c*(A1*B2 + A2*B1) * c**2(A2*B2)
 *        with all values < sqrt(mod) subpolynomials have coefficientes <
    mod * N after fft multiply decreasing changes of rounding error
 * */

const double PI=acos(-1);
```

```
typedef complex<double> base;

void fft (vector<base> & a, bool invert) {
        int n=(int) a.size();
        for (int i=1, j=0; i<n; ++i) {
                int bit=n>>1;
                for (;j>=bit;bit>>=1)
                        j-=bit;
                j+=bit;
                if(i<j)
                        swap(a[i],a[j]);
        }
        for (int len=2; len<=n; len<<=1) {
                double ang = 2*PI/len * (invert ? -1 : 1);
                base wlen(cos(ang), sin(ang));
                for (int i=0; i<n; i+=len) {
                        base w(1);
                        for (int j=0; j<len/2; ++j) {
                                base u=a[i+j], v=a[i+j+len/2]*w;
                                a[i+j]=u+v;
                                a[i+j+len/2]=u-v;
                                w*=wlen;
                        }
                }
        }
        if (invert)
                for(int i=0;i<n;++i)
                        a[i]/=n;
}

// a, b => coefs to multiply,  res => resulting coefs
// a[0], b[0], res[0] = coef x^0
// Doesnt work with negative coefs
void multiply (const vector<int> & a, const vector<int> & b, vector<int> &
     res) {
        vector<base> fa (a.begin(), a.end()),  fb (b.begin(), b.end());
        size_t n=1;
        while (n<max(a.size(),b.size())) n<<=1;
        n<<=1;
        fa.resize(n),fb.resize(n);
        fft (fa,false);  fft(fb,false);
        for (size_t i=0; i<n; ++i)
                fa[i]*=fb[i];
        fft (fa, true);
        res.resize (n);
        // avoid precision errors, mess up with negative values of coefs
        for(size_t i=0; i<n; ++i)
                res[i]=int(fa[i].real() + 0.5);
}
```

Matrix Exponentiation

```
typedef long long ll;
const int NMAT=2;
const int mod=1;

/* c=a*b */
void mu(ll a[][NMAT], ll b[][NMAT], ll c[][NMAT], int _n) {
        for(int i=0;i<_n;i++)
                for(int j=0;j<_n;j++) {
                        c[i][j]=0;
                        for(int h=0;h<_n;h++) {
                                c[i][j]+=(a[i][h]*b[h][j])%mod;
                                c[i][j]%=mod;
                        }
                }
}
/*returns ans=mat^b*/
void power(ll ans[][NMAT], ll mat[][NMAT], ll b, int _n) {
        ll tmp[NMAT][NMAT];
        for(int i=0;i<_n;i++)
                for (int j = 0; j<_n; j++)
                        ans[i][j]=i==j;
        while(b) {
                if(b&1) {
                        mu(ans, mat, tmp, _n);
                        for(int i=0;i<_n;i++)
                                for(int j=0;j<_n;j++)
                                        ans[i][j]=tmp[i][j];
                }
                mu(mat, mat, tmp, _n);
                for(int i=0;i<_n;i++)
                        for(int j=0;j<_n;j++)
                                mat[i][j]=tmp[i][j];
                b>>=1;
        }
}
```

Math Tricks

```
ll fexp(ll a, int x, ll mod){            // Fast exponenciation returns a^x
    % mod
        if(x==0)return 1ll;
        if(x%2==0){
                ll y=fexp(a, x/2, mod);
                return (y*y)%mod;
        }
        return (a*fexp(a, x-1, mod))%mod;
}

ll divv(ll a, ll b, ll mod){     // Division with mod returns a/b % mod
        return (a*fexp(b, mod-2, mod))%mod;
}

ll f[N];

ll fat(ll a, ll mod){            // Calculates factorial and stores in f %
    mod
```

```
        if(a<=1)return 1;
        return f[a]?f[a]:(f[a]=(a*fat(a-1, mod))%mod);
}

ll choose(ll n, ll k, ll mod){  // Returns n choose k % mod
        return divv(fat(n, mod), (fat(k, mod)*fat(n-k, mod))%mod, mod)%mod
            ;
}

ll gcd(ll a, ll b){     // Greatest common divisor
        return b?gcd(b, a%b):a;
}

ll lcm(ll a, ll b){     // Least common multiple
        return (a*b)/gcd(a, b);
}


/* Fast factorization */

int p[N];

void start_fast(int MAX){       // Runs O(nlog(n)) Needs to be called to
    use fast_fact or ammount_of_divisors.
        for(int i=2; i<=MAX; i++){
                if(p[i]==0){
                        for(int j=i; j<=MAX; j+=i){
                                p[j]=i;
                        }
                }
        }
}

vector<int>fast_fact(int x){    // Fast factorization in O(log2(x))
        vector<int>ret;
        while(x>1){
                ret.pb(p[x]);
                x/=p[x];
        }
        return ret;
}

int amount_of_divisors(int x){  // Calculate the ammount of divisors of a
    number in O(log2(x)) assume already ran start_fast.
        if(x==1)return 1;
        vector<int>v=fast_fact(x);
        int ret=1, curr=2;
        for(int i=1; i<v.size(); i++){
                if(v[i]==v[i-1])curr++;
                else{
                        ret*=curr;
                        curr=2;
                }
        }
        return ret*curr;
}
```

Theorems

```
Picks theorem
Given a certain lattice polygon with non-zero area.

We denote its area by S, the number of points with integer coordinates
    lying strictly inside the polygon by I and the number of points lying
    on polygon sides by B.

Then, the Pick's formula states:

S=I+B/2   1


Burnsides Lemma
Let G be the finite group of operations we can perform on X
The number of orbits of X is the average of the number of fixed points for
     each g in G
G must be closed
```