

ACM ICPC Code Notebook

UofT Blue 2018

November 7, 2018

Contents

Templates	3
C++	3
Java	3
Makefile	4
Algorithms	5
Graphs	5
AP & Bridges	5
Centroid Decomposition	5
Centroid Decomposition	5
Centroid Decomposition (Eoin)	6
Network Flow	6
Dinic's Blocking flow	6
Edmond Karp	7
Ford Fulkerson 1	7
Ford Fulkerson 2	7
Lowest Common Ancestor	8
Lowest Common Ancestor (Using RMQ)	8
Lowest Common Ancestor (Using Binary Lifting)	8
Minimum Spanning Tree (Kruskal's)	9
Strongly Connected Components (Tarjan's)	9
Bipartite Check	10
Dijkstras	10
Find Cycles	10
Arrays	11
Longest Increasing Subsequence	11
Data Structures	12
Fenwick Tree	12
Segment Trees	12
Segment Tree Lazy	12
Segment Tree (Eoin)	12

Iterative Segment Tree	13
Sparse Table	13
Union Find	13
Strings	14
KMP	14
KMP	14
KMP (Eoin)	14
Rabin Karp	15
Suffix Array	15
Manacher's Algorithm	15
Z function	16
Trie	16
Hash	17
Geometry	18
Convex Hull	18
Convex Hull	18
Convex Hull (Eoin)	18
Geometry Functions	19
Math	21
Binomial Coefficients	21
Ternary Search	21
Miller Rabin primality test	21
Fast fourier transform	21
Matrix Exponentiation	22
Math Tricks	22
Theorems	23

Templates

C++

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define ll long long int
6 #define inf 0x3f3f3f3f
7 #define pb push_back
8 #define mk make_pair
9 #define mt make_tuple
10 #define fi first
11 #define se second
12 #define ii pair<int, int>
13 #define all(x) (x).begin(), (x).end()
14 #define N 1000007 // 10e6 + 7
15
16 const double PI = acos(-1.0);
17
18
19 int main(int argc, char const *argv[]) {
20     //ios::sync_with_stdio(false);
21
22     return 0;
23 }
```

Java

```
1 import java.io.BufferedReader;
2 import java.io.Closeable;
3 import java.io.IOException;
4 import java.io.InputStreamReader;
5 import java.io.PrintWriter;
6 import java.util.StringTokenizer;
7 import java.math.*;
8 import java.text.*;
9 import java.util.*;
10
11 public class Icpc {
12
13     public static void main(String... args) throws Exception {
14         FastScan sc = new FastScan(new BufferedReader(new
15             InputStreamReader(System.in)));
16         PrintWriter pw = new PrintWriter(System.out);
17
18         sc.close();
19         pw.close();
20         System.exit(0);
21     }
22
23     static class FastScan implements Closeable {
24         private BufferedReader br;
25         private StringTokenizer tk;
26
27         public FastScan(BufferedReader br) {
28             this.br = br;
29         }
30
31         public int in() throws NumberFormatException, IOException
32         {
33             return Integer.parseInt(next());
34         }
35
36         public long ln() throws NumberFormatException, IOException
37         {
38             return Long.parseLong(next());
39         }
40     }
41 }
```

```

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 }

```

```

27 mt19937 rand((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(), rand) // this is better than random_shuffle
        !!!!!

```

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 rand((time_t)ts.tv_nsec); // NEW RAND THAT GOES TO 4*10^9, this
        is quicker than rand()
19  rand() % WHATEVER // to get the value
20
21  srand(time(NULL)) // but this is in seconds
22  mt19937 rand(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

```

Algorithms

Graphs

AP & Bridges

```
1 int dfs(int u, int p){
2     dfs_num[u] = dfs_low[u] = ++dfs_counter;
3     for(auto v : adjList[u]){
4         if(dfs_num[v]==0){
5             dfs(v, u);
6             if(dfs_low[v] >= dfs_num[u]){
7                 articulation[u]=true;
8             }
9             if(dfs_low[v] > dfs_num[u])
10                bridge = true;
11            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
12        } else if(v!=p)
13            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
14    }
15 }
16
17 int main(){
18     memset(dfs_num, 0, sizeof(dfs_num));
19     memset(dfs_low, 0, sizeof(dfs_low));
20     bridge=false;
21     dfs_counter=0;
22     dfs(0, -1);
23     for(int i = 0; i < N; ++i)
24         if(dfs_num[i]==0)
25             bridge=true;
26     puts(bridge ? "Yes" : "No");
27     return 0;
28 }
```

Centroid Decomposition

Centroid Decomposition

```
1 #include <bits/stdc++.h>
2 #define MAXN 100100
3 typedef long long ll;
4
5 using namespace std;
```

```
6
7 int n, sz[MAXN];
8 bool deleted[MAXN], vis[MAXN];
9 char ch[MAXN];
10 vector<int> g[MAXN];
11
12 void dfs(int x, int p){
13     if(vis[x]) return;
14     vis[x] = true;
15     sz[x] = 1;
16     for(auto i : g[x]){
17         if(i == p || deleted[i]) continue;
18         dfs(i, x);
19         sz[x] += sz[i];
20     }
21     //cout << x << " " << sz[x] << "\n";
22 }
23
24 int findCentroid(int x){
25     memset(vis, 0, sizeof(vis));
26     dfs(x, -1);
27     int p = -1, c = sz[x] / 2;
28     while(true){
29         bool found = false;
30         for(auto i : g[x]){
31             if(!deleted[i] && i != p && sz[i] > c){
32                 found = true;
33                 p = x;
34                 x = i;
35                 break;
36             }
37         }
38         if(!found) return x;
39     }
40 }
41
42 void decomp(int x, char c){
43     int cen = findCentroid(x);
44     ch[cen] = c;
45     deleted[cen] = true;
46     for(auto i : g[cen]){
47         if(deleted[i]) continue;
48         decomp(i, c + 1);
49     }
50 }
```

```

49     }
50 }
51
52 int main(){
53     #ifndef ONLINE_JUDGE
54         freopen("input.txt", "r", stdin);
55     #endif
56     ios_base::sync_with_stdio(false);
57     cin.tie(NULL);
58     memset(deleted, 0, sizeof(deleted));
59     cin >> n;
60     for(int i = 0; i < n - 1; i++){
61         int a, b;
62         cin >> a >> b;
63         g[a].push_back(b);
64         g[b].push_back(a);
65     }
66     //cout << findCentroid(1);
67     decomp(1, 'A');
68     for(int i = 1; i <= n; i++){
69         cout << ch[i] << " ";
70     }
71 }

```

Centroid Decomposition (Eoin)

```

1 void fill_sz(int u, int p){
2     sz[u] = 1;
3     for(int v : adjList[u]){
4         if(v==p || mkd[v])
5             continue;
6         fill_sz(v, u);
7         sz[u] += sz[v];
8     }
9 }
10
11 int get_centroid(int u, int n, int p){
12     for(int v : adjList[u]){
13         if(v==p || mkd[v])
14             continue;
15         if(sz[v] > n/2)
16             return get_centroid(v, n, u);
17     }
18     return u;
19 }
20
21 int decomp(int u){
22     fill_sz(u, -1);
23     int cent = get_centroid(u, sz[u], -1);
24     mkd[cent] = true;
25     for(int v : adjList[cent]){
26         if(mkd[v])
27             continue;
28         int r = decomp(v);
29         centP[r] = cent;

```

```

30     }
31     return cent;
32 }

```

Network Flow

Dinic's Blocking flow

```

1 /**
2     e-maxx's flow
3     Dinic algorithm
4     Complexity  $O(V^2 \cdot E)$  OR  $O(V \cdot \sqrt{E})$  for bipartite graphs!
5
6 */
7
8 struct edge {
9     int a, b, cap, flow;
10 };
11
12 int nodes, s, t, d[N], ptr[N], q[N]; // NEED TO set n (
13                                     // max of nodes), s source, t sink
14 vector<edge> e;
15 vector<int> g[N];
16
17 void add_edge(int a, int b, int cap) {
18     edge e1 = { a, b, cap, 0 };
19     edge e2 = { b, a, 0, 0 };
20     g[a].push_back((int) e.size());
21     e.push_back(e1);
22     g[b].push_back((int) e.size());
23     e.push_back(e2);
24 }
25
26 bool bfs() {
27     int qh=0, qt=0;
28     q[qt++] = s;
29     memset(d, -1, nodes * sizeof d[0]);
30     d[s] = 0;
31     while (qh < qt && d[t] == -1) {
32         int v = q[qh++];
33         for (size_t i=0; i<g[v].size(); ++i) {
34             int id = g[v][i],
35                 to = e[id].b;
36             if (d[to] == -1 && e[id].flow < e[id].cap) {
37                 q[qt++] = to;
38                 d[to] = d[v] + 1;
39             }
40         }
41     }
42     return d[t] != -1;
43 }
44
45 int dfs(int v, int flow) {
46     if (!flow) return 0;
47     if (v == t) return flow;

```

```

47     for (; ptr[v]<(int)g[v].size(); ++ptr[v]) {
48         int id = g[v][ptr[v]],
49             to = e[id].b;
50         if (d[to] != d[v] + 1) continue;
51         int pushed = dfs (to, min (flow, e[id].cap - e[id].flow));
52         if (pushed) {
53             e[id].flow += pushed;
54             e[id ^ 1].flow -= pushed;
55             return pushed;
56         }
57     }
58     return 0;
59 }
60
61 int dinic () {
62     int flow = 0;
63     for (;;) {
64         if (!bfs()) break;
65         memset (ptr, 0, nodes * sizeof ptr[0]);
66         while (int pushed = dfs (s, inf))
67             flow += pushed;
68     }
69     return flow;
70 }

```

Edmond Karp

```

1 void aug(int u, int minE){
2     if(u==S){ f=minE; return; }
3     if(p[u]!=u){
4         aug(p[u], min(minE, res[p[u]][u]));
5         res[p[u]][u]-=f;
6         res[u][p[u]]+=f;
7     }
8 }
9
10 int main(){
11     int mf=0;
12     for(;;){
13         f=0; //Global
14         for(int i = 0; i < N; i++)
15             dist[i]=INF, p[i]=i;
16         dist[S]=0;
17         queue<int> q; q.push(S);
18         while(!q.empty()){
19             int u = q.front(); q.pop();
20             if(u==T) break;
21             for(int i = 0; i < N; i++)
22                 if(res[u][i] > 0 && dist[i]==INF)
23                     dist[i]=dist[u]+1, p[i]=u, q.push(i);
24         }
25         aug(T, INF);
26         if(f==0) break;
27         mf+=f;
28     }

```

```

29     vector<ii> used;
30     for(int i = 0; i < N; i++)
31         for(int j = 0; j < N; j++)
32             if(graph[i][j] > 0 && res[i][j] < graph[i][j])
33                 used.push_back(make_pair(i, j));
34 }

```

Ford Fulkerson 1

```

1 int ff(int u, int minE){
2     if(u==T)
3         return minE;
4     vis[u]=true;
5     for(auto i : adjList[u]){
6         if(!vis[i] && res[u][i] > 0){
7             if(int f = ff(i, min(minE, res[u][i]))){
8                 res[u][i] -= f;
9                 res[i][u] += f;
10                return f;
11            }
12        }
13    }
14    return 0;
15 }
16
17 int main(){
18     int mf = 0;
19     while(1){
20         memset(vis, 0, sizeof(vis));
21         int f = ff(S, INF);
22         if(f==0)
23             break;
24         mf+=f;
25     }
26     printf("%d\n", mf);
27 }

```

Ford Fulkerson 2

```

1 #include <bits/stdc++.h>
2 #define MAXN 3000
3 typedef long long ll;
4
5 using namespace std;
6
7 int n;
8 int g[MAXN][MAXN], rg[MAXN][MAXN], parent[MAXN];
9
10 bool bfs(int source, int sink){
11     bool visited[MAXN];
12     memset(visited, 0, sizeof(visited));
13     queue<int> q;
14     q.push(source);
15     visited[source] = true;
16     parent[source] = -1;

```



```

17     while(!q.empty()){
18         int i = q.front();
19         q.pop();
20         for(int j = 0; j < MAXN; j++){
21             if(!visited[j] && rg[i][j] > 0){
22                 q.push(j);
23                 parent[j] = i;
24                 visited[j] = true;
25             }
26         }
27     }
28     return visited[sink];
29 }
30
31 int maxFlow(int source, int sink){
32     for(int i = 0; i < MAXN; i++){
33         for(int j = 0; j < MAXN; j++){
34             rg[i][j] = g[i][j];
35         }
36     }
37     int max_flow = 0;
38     while(bfs(source, sink)){
39         int path_flow = 99999999;
40         for(int i = sink; i != source; i = parent[i]){
41             int j = parent[i];
42             path_flow = min(path_flow, rg[j][i]);
43         }
44         for(int i = sink; i != source; i = parent[i]){
45             int j = parent[i];
46             rg[j][i] -= path_flow;
47             rg[i][j] += path_flow;
48         }
49         max_flow += path_flow;
50     }
51     return max_flow;
52 }
53
54 int main(){
55     #ifndef ONLINE_JUDGE
56         freopen("input.txt", "r", stdin);
57     #endif
58     ios_base::sync_with_stdio(false);
59     for(int i = 0; i < MAXN; i++){
60         for(int j = 0; j < MAXN; j++){
61             rg[i][j] = g[i][j] = 0;
62         }
63     }
64 }

```

Lowest Common Ancestor

Lowest Common Ancestor (Using RMQ)

```

1  /*
2   * H[u] is first visit of u

```

```

3   * E[x] is vertex at time x
4   * L[x] is depth at time x
5   */
6   void vis(int u, int d){
7       H[u]=vind;
8       E[vind] = u;
9       L[vind++] = d;
10      for(auto i : adjList[u]){
11          if(H[i]!=-1)
12              continue;
13          vis(i,d+1);
14          E[vind] = u;
15          L[vind++] = d;
16      }
17  }
18
19  int LCA(int u, int v){
20      if(H[u] > H[v]){
21          int t = u;
22          u = v;
23          v = t;
24      }
25      //run some range min query on L
26      //between H[u] and H[v]
27      int ind = rmq(H[u],H[v]);
28      return E[ind];
29  }
30
31  int dist(int u, int v){
32      int a = H[u];
33      int b = H[v];
34      int ind = LCA(u,v);
35      return abs(L[H[ind]]-L[a])
36             + abs(L[H[ind]]-L[b]);
37  }

```

Lowest Common Ancestor (Using Binary Lifting)

```

1  #include <bits/stdc++.h>
2  #define MAXN 100100
3  typedef long long ll;
4
5  using namespace std;
6
7  int n, m, s[MAXN], depth[MAXN], anc[MAXN][40];
8  vector<int> g[MAXN];
9  bool vis[MAXN];
10
11  int dfs(int x, int d, int p){
12      vis[x] = true;
13      depth[x] = d;
14      s[x] = 1;
15      anc[x][0] = p;
16      for(int i = 1; pow(2, i) <= d; i++){
17          anc[x][i] = anc[anc[x][i-1]][i-1];

```

```

18     }
19     for(int i = 0; i < g[x].size(); i++){
20         if(vis[g[x][i]]) continue;
21         s[x] += dfs(g[x][i], d + 1, x);
22     }
23
24     return s[x];
25 }
26
27 int walk(int x, int d){
28     int i = 0;
29     while(d){
30         if(d & 1) x = anc[x][i];
31         d /= 2;
32         i++;
33     }
34     //cout << "\n";
35     return x;
36 }
37
38 int lca(int x, int y){
39     //cout << x<<y;
40
41     if(depth[x] < depth[y]) y = walk(y, depth[y] - depth[x]);
42     if(depth[x] > depth[y]) x = walk(x, depth[x] - depth[y]);
43     //cout << x<<y;
44     if(x == y) return x;
45     for(int i = 30; i >= 0; i--){
46         if(depth[x] >= pow(2, i) && anc[x][i] != anc[y][i]){
47             return lca(anc[x][i], anc[y][i]);
48         }
49     }
50     return anc[x][0];
51 }
52
53 int main(){
54     ios_base::sync_with_stdio(false);
55     cin >> n;
56     for(int i = 0; i < n - 1; i++){
57         int a, b;
58         cin >> a >> b;
59         g[a].push_back(b);
60         g[b].push_back(a);
61     }
62     dfs(1, 0, -1);
63     cin >> m;
64     for(int i = 0; i < m; i++){
65         int a, b;
66         cin >> a >> b;
67         if(depth[a] > depth[b]) swap(a, b);
68         if(a == b) cout << n;
69         else{
70             int l = lca(a, b);
71             int d = -2 * depth[l] + depth[a] + depth[b];
72             if(d % 2) cout << "0";

```

```

73         else{
74             if(depth[a] == depth[b]) cout << s[1] - s[walk(b,
75                 d / 2 - 1)] - s[walk(a, d / 2 - 1)];
76             else cout << s[walk(b, d / 2)] - s[walk(b, d / 2 -
77                 1)];
78         }
79         cout << "\n";
80     }
81 }

```

Minimum Spanning Tree (Kruskal's)

```

1 struct edge {
2     int x,y,w;
3     bool operator < (edge e) const {
4         return w < e.w;
5     }
6 };
7
8 int main(){
9     vector<edge> eList; //Input
10    for(int i = 0; i < N; i++)// Set up UFDS
11        p[i]=i;
12    vector<ii> treeList;
13    sort(eList.begin(), eList.end());
14    int cost = 0;
15    int sz=N;
16    int u,v,w;
17    for(const auto &i : eList){
18        v=i.x; u=i.y; w=i.w;
19        if(!connected(u,v)){
20            join(u,v);
21            treeList.push_back({min(u,v), max(u,v)});
22            sz--;
23            cost+=w;
24        }
25    }
26    if(sz!=1)
27        puts("Impossible");
28 }

```

Strongly Connected Components (Tarjan's)

```

1 typedef pair<int, int> ii;
2
3 int N,M;
4 vector<int> adjList[MXN];
5 int dfs_num[MXN], dfs_low[MXN];
6 bool vis[MXN];
7 stack<int> scc;
8 int dfsCounter=1;
9 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 }

```

```

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<
19         pair<int, int> > > pq;
20     pq.push({0, x});
21     dist[x] = 0;
22     while(!pq.empty()){
23         pair<int, int> v = pq.top();
24         pq.pop();
25         for(int i = 0; i < g[v.second].size(); i++){
26             pair<int, int> u = g[v.second][i];
27             if(dist[v.second] + u.second < dist[u.first])
28                 pq.push({dist[v.second] + u.second, u.first});
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

```
1 int tree[MXN];
2 int N;
3 int lsOne(int i){ return i&(-i); }
4 void update(int k,int v){
5     for(; k<MXN; k+=lsOne(k))
6         tree[k]+=v;
7 }
8 int query(int k){
9     int cnt=0;
10    for(; k; k-=lsOne(k)){
11        cnt+=tree[k];
12    }
13    return cnt;
14 }
```

Segment Trees

Segment Tree Lazy

```
1 typedef long long ll;
2 typedef pair<int, int> ii;
3 const int INF = 0x3f3f3f3f;
4 const double PI = acos(-1.0);
5
6 struct SegT {
7     vector<ll> seg, lazy;
8     int n;
9
10    SegT () {}
11
12    SegT (int n) {
13        this->n = n;
14        seg.resize(4*n + 1);
15        lazy.resize(4*n + 1);
16    }
17
18    void prop (int r, int i, int j) {
19
20        seg[r] += lazy[r] * (j-i+1);
21    }
```

```
22        if (i != j) {
23            lazy[2*r] += lazy[r];
24            lazy[2*r + 1] += lazy[r];
25        }
26
27        lazy[r] = 0;
28    }
29
30    int a, b;
31    ll update (int r, int i, int j, ll val) {
32        prop (r, i, j);
33        if (j < a or i > b) return 0LL;
34
35        if (i >= a and j <= b) {
36            lazy[r] += val;
37            prop (r, i, j);
38            return seg[r];
39        } else {
40            int mid = (i + j)/2;
41            ll L = update (2*r, i, mid, val);
42            ll R = update (2*r + 1, mid + 1, j, val);
43            return L + R;
44        }
45    }
46
47    ll update (int l, int r, ll val) {
48        a = l; b = r;
49        return update (1, 0, n - 1, val);
50    }
51
52    ll query (int l, int r) {
53        return update (1, r, 0);
54    }
55
56 };
```

Segment Tree (Eoin)

```
1 int tree[MXN*4];
2 int a[MXN];
3 int N;
4
5 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

```
1  /*
2      border = proper prefix that is suffix
3      p[i] = length of longest border of prefix of length i, s[0...i-1]
4  */
5
6  typedef long long ll;
7  typedef pair<int, int> ii;
8  const int INF = 0x3f3f3f3f;
9  const double PI = acos(-1.0);
10
11 const int N = 1e6 + 6;
12 int pi[N];
13 string p, t;
14
15 void pre () {
16     p += '#';
17
18     pi[0] = pi[1] = 0;
19     for (int i = 2; i <= (int)p.size(); i++) {
20         pi[i] = pi[i-1];
21
22         while (pi[i] > 0 and p[pi[i]] != p[i-1])
23             pi[i] = pi[pi[i]];
24
25         if (p[pi[i]] == p[i-1])
26             pi[i]++;
27     }
28 }
29
30 void report (int at) {
31
32 }
33
34 void KMP () {
35     pre ();
36
37     int k = 0;
38     int m = p.size() - 1;
```

```
39
40     for (int i = 0; i < (int)t.size(); i++) {
41         while (k > 0 and p[k] != t[i])
42             k = pi[k];
43
44         if (p[k] == t[i])
45             k++;
46         if (k == m)
47             report (i - m + 1);
48     }
49
50 }
51
52 int main (void) {
53     ios_base::sync_with_stdio(false);
54
55     return 0;
56 }
```

KMP (Eoin)

```
1 vector<int> buildFailure(string s){
2     vector<int> T(n+1,0);
3     T[0]=-1;
4     int j = 0;
5     for(int i = 1; i < s.size();++i){
6         if(s[i]==s[j]){
7             T[i]=T[j];
8             j++;
9         } else{
10            T[i] = j;
11            j = T[j];
12            while(j >= 0 && s[i]!=s[j])
13                j = T[j];
14            j++;
15        }
16    }
17    T[s.size()] = j;
18    return T;
19 }
20 vector<int> search(string W, string S){
21     auto T=buildFailure(W);
22     vector<int> p;
```

```

23     int k = 0;
24     int j = 0;
25     while(j < S.size()){
26         if(W[k]==S[j]){
27             k++; j++;
28             if(k==W.size()){
29                 p.push_back(j-k);
30                 k = T[k];
31             }
32         }else{
33             k = T[k];
34             if(k < 0)
35                 j+=1, k+=1;
36         }
37     }
38     return p;
39 }

```

Rabin Karp

```

1  // Looks for a pattern s in text t in O(n+m) time.
2  // Returns where the occurrences are
3
4  vector<int> rabin_karp(string const& s, string const& t) {
5      const int p = 31;
6      const int m = 1e9 + 9;
7      int S = s.size(), T = t.size();
8
9      vector<long long> p_pow(max(S, T));
10     p_pow[0] = 1;
11     for (int i = 1; i < (int)p_pow.size(); i++)
12         p_pow[i] = (p_pow[i-1] * p) % m;
13
14     vector<long long> h(T + 1, 0);
15     for (int i = 0; i < T; i++)
16         h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
17     long long h_s = 0;
18     for (int i = 0; i < S; i++)
19         h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;
20
21     vector<int> occurrences;
22     for (int i = 0; i + S - 1 < T; i++) {
23         long long cur_h = (h[i+S] + m - h[i]) % m;
24         if (cur_h == h_s * p_pow[i] % m)
25             occurrences.push_back(i);
26     }
27     return occurrences;
28 }

```

Suffix Array

```

1 void countingSort(int k){
2     int i, sum, maxi=max(300,N);

```

```

3     memset(c,0,sizeof(c));
4     for(i = 0; i < N; i++)
5         c[i+k < N ? RA[i+k] : 0]++;
6     for(i=sum=0; i < maxi; i++){
7         int t = c[i];
8         c[i]=sum;
9         sum+=t;
10    }
11    for(i = 0; i < N; i++)
12        tempSA[c[SA[i]+k < N
13            ? RA[SA[i]+k]: 0]++] = SA[i];
14    for(i=0; i < N; i++)
15        SA[i]=tempSA[i];
16 }
17
18 int main(){
19     for(int i = 0; i < N; i++)
20         SA[i]=i, RA[i]=input[i];
21     int r;
22     for(int k = 1; k < N; k <= 1){
23         countingSort(k);
24         countingSort(0);
25         tempRA[SA[0]]=r=0;
26         for(int i = 1; i < N; i++){
27             tempRA[SA[i]]
28                 =(RA[SA[i]]==RA[SA[i-1]]
29                 && RA[SA[i]+k]==RA[SA[i-1]+k]
30                 ? r:++r);
31         }
32         for(int i = 0; i < N; i++)
33             RA[i]=tempRA[i];
34     }
35     return 0;
36 }

```

Manacher's algorithm for longest palindromic substring

```

1  /* Manacher's algorithm O(N), time and memory, algorithm to find
2  * longest palindromic substring
3  *
4  * Transform initial string t into s,
5  * putting separators between characters
6  *
7  * Build vector p[], where p[i] is the length of the
8  * palindrome centered at s[i]
9  *
10 * Works for both, odd and even length
11 * s: # a # b # a #                # a # a #
12 * p: 0 1 0 3 0 1 0                0 1 2 1 0
13 *
14 * p built in O(N) using the fact that elements can be symmetric
15 * given some center and p[center]:

```



```

16  * If we are in i and center c, i_mirror = c - (i - c), if p[i_mirror]
17  * fits in center + p[center], p[i] is p[i_mirror], else we need to
18  * check real value of p[i]
19  * If we call the border center + p[center], r. Its easy to see
20  * r is only increased, achieving the O(N) time complexity
21  *
22  * Longest palindromic substring is the maximum element in p
23  *
24  */
25
26 typedef long long ll;
27 typedef pair<int, int> ii;
28 const int INF = 0x3f3f3f3f;
29 const double PI = acos(-1.0);
30
31 const int N = 1e6 + 5;
32 int p[2*N + 2];
33
34 int main (void) {
35     ios_base::sync_with_stdio(false);
36
37     string s, t;    cin >> t;
38     s += "#";
39     for (auto c : t) {
40         s += c;
41         s += '#';
42     }
43
44     int n = s.size();
45     int c = 0, r = 0;
46     for (int i = 0; i < n; i++) {
47         int i_mirror = c - (i - c);
48
49         if (i <= r)
50             p[i] = min (p[i_mirror], r - i);
51         else
52             p[i] = 0;
53
54         while (i - 1 - p[i] >= 0 and i + 1 + p[i] < n and s[i + 1 - p[i]] == s[i + 1 + p[i]]) {
55             p[i]++;
56         }
57
58         if (i + p[i] > r) {
59             c = i;
60             r = i + p[i];
61         }
62     }
63
64     int len = 0, center = 0;
65     for (int i = 0; i < n; i++)
66         if (p[i] > len) {
67             len = p[i];
68             center = i;
69         }

```

```

70
71     /* not tested */
72     string res;
73     for (int i = 0; i < n; i++)
74         if (i >= center - len and i <= center + len and s[i] != '#')
75             res += s[i];
76
77     /* */
78
79     cout << len << endl;
80     cout << res << endl;
81
82     return 0;
83 }

```

Z function

```

1  /*          {0, if i = 0
2  z[i] = {length longest common prefix of s and s[i...n-1]
3  */
4
5  typedef long long ll;
6  typedef pair<int, int> ii;
7  const int INF = 0x3f3f3f3f;
8  const double PI = acos(-1.0);
9
10 const int N = 2e5 + 5;
11 string s;
12 int z[N];
13
14 void go () {
15     int l = 0, r = 0;
16     int n = s.size();
17     memset (z, 0, sizeof z);
18
19     for (int i = 1; i < n; i++) {
20         if (i <= r)
21             z[i] = min (z[i-l], r - i + 1);
22         while (z[i] + i < n and s[z[i] + i] == s[i + z[i]])
23             z[i]++;
24         if (r < i + z[i] - 1) {
25             l = i;
26             r = i + z[i] - 1;
27         }
28     }
29 }

```

Trie

```

1 struct node {
2     node * children[26];
3     int count;
4     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
45 // + jlen - 1]
46 // not tested if ilen != jlen
47 bool comp (string &s, int i, int ilen, int j, int jlen) {
48     int bot = 0, top = min (ilen, jlen) - 1;
49     int id = -1;
50
51     while (bot <= top) {
52         int mid = (bot + top)>>1;
53
54         ii pi = ii(h[0].query(i, i + mid), h[1].query(i, i + mid));
55         ;
56         ii pj = ii(h[0].query(j, j + mid), h[1].query(j, j + mid));
57         ;
58         if (pi == pj) {
59             bot = mid + 1;
60             id = mid;
61         } else {
62             top = mid - 1;
63         }
64     }
65
66     if (id == min (ilen, jlen) - 1) {
67         if (ilen != jlen)
68             return ilen < jlen;
69         return i < j;
70     }
71
72     return s[i + id + 1] < s[j + id + 1];
73 }
74
75 const ll mod[2] = {10000000007, 10000000009};

```

Geometry

Convex Hull

Convex Hull

```
1 // Implementation of Andrew's monotone chain 2D convex hull algorithm.
2 // Asymptotic complexity:  $O(n \log n)$ .
3
4 typedef double coord_t; // coordinate type
5 typedef double coord2_t; // must be big enough to hold  $2 * \max(|\text{coordinate}|)^2$ 
6
7 struct Point {
8     coord_t x, y;
9     Point() {}
10    Point(coord_t xx, coord_t yy){
11        x=xx, y=yy;
12    }
13    bool operator <(const Point &p) const {
14        return x < p.x || (x == p.x && y < p.y);
15    }
16 };
17
18 // 2D cross product of OA and OB vectors, i.e. z-component of their 3D
19 // cross product.
20 // Returns a positive value, if OAB makes a counter-clockwise turn,
21 // negative for clockwise turn, and zero if the points are collinear.
22 coord2_t cross(const Point &O, const Point &A, const Point &B){
23     return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
24 }
25
26 // Returns a list of points on the convex hull in counter-clockwise order.
27 // Note: the last point in the returned list is the same as the first one.
28 vector<Point> convex_hull(vector<Point> P){
29     size_t n = P.size(), k = 0;
30     if (n <= 3) return P;
31     vector<Point> H(2*n);
32
33     // Sort points lexicographically
34     sort(P.begin(), P.end());
35
36     // Build lower hull
37     for (size_t i = 0; i < n; ++i) {
```

```
37         while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--; //
38         // Remove the last "=" if you want to get max points in
39         // the hull
40         H[k++] = P[i];
41     }
42
43     // Build upper hull
44     for (size_t i = n-1; i > 0; --i) {
45         while (k >= t && cross(H[k-2], H[k-1], P[i-1]) <= 0) k--;
46         // Remove the last "=" if you want to get max points
47         // in the hull
48         H[k++] = P[i-1];
49     }
50
51     H.resize(k-1);
52     return H;
53 }
54
55 /*
56  * Note that this is using double. Its better to use long long because
57  * double might be TLE!
58 */
```

Convex Hull (Eoin)

```
1 int main(){
2     for(int i = 0; i < N; i++){
3         perm[i]=i;
4     }
5     sort(perm, perm+N,
6         [](int a, int b){
7             const point &pa = V[a];
8             const point &pb = V[b];
9             if (real(pa)!=real(pb))
10                 return real(pa) < real(pb);
11             return imag(pa) < imag(pb);
12         });
13     vector<int> L; vector<int> U;
14     for(int i = 0; i < N; i){
15         int t = L.size();
16         if(t >= 2 && !ccw(V[L[t-2]], V[L[t-1]], V[perm[i]]))
17             L.pop_back();
18         else
```

```

19         L.push_back(perm[i++]);
20     }
21     for(int i = N-1; i >=0;){
22         int t = U.size();
23         if(t >= 2 && !ccw(V[U[t-2]],V[U[t-1]],V[perm[i]]))
24             U.pop_back();
25         else
26             U.push_back(perm[i--]);
27     }
28     vector<int> hull;
29     for(int i = 0; i < L.size()-1; ++i)
30         hull.push_back(L[i]);
31     for(int i = 0; i < U.size()-1; ++i)
32         hull.push_back(U[i]);
33     return 0;
34 }

```

Geometry Functions

```

1  typedef complex<double> pt;
2  typedef complex<double> vec;
3  typedef vector<pt> pgon;
4  typedef struct { pt p,q; } lseg;
5  double cross(const vec& a, const vec &b){
6      return x(a)*y(b)-y(a)*x(b);
7  }
8  //cross product of (b-a) and (c-b), 0 is collinear
9  int orientation(const pt& a,
10     const pt& b, const pt& c){
11     double v = cross(b-a,c-b);
12     if(abs(v-0.0)<EPS)
13         return 0;
14     return v > 0 ? 1 : 2;
15 }
16 //Line segment intersection
17 bool intersects(const lseg& a, const lseg& b){
18     if(a.q == b.p || b.q == a.p)
19         return false;
20     if(orientation(a.p,a.q,b.p)
21        !=orientation(a.p,a.q,b.q)
22        && orientation(b.p,b.q,a.p)
23        != orientation(b.p,b.q,a.q))
24         return true;
25     return false;
26 }
27 //Area of polygon
28 double area(const pgon& p){
29     double area = 0.0;
30     for(int i = 1; i < p.size(); ++i)
31         area+=cross(p[i-1],p[i]);
32     return abs(area)/2.0;
33 }
34 //If a->b->c is a counterclockwise turn
35 double ccw(const point& a, const point& b,

```

```

36     const point& c){
37     if(a==b || b==c || a==c)
38         return false;
39     point relA = b-a;
40     point relC = b-c;
41     return cross(relA,relC) >= 0.0;
42 }
43 //Returns if point p is in the polygon poly
44 bool inPoly(const pgon& poly, const pt& p){
45     for(int i = 0; i < poly.size()-1; i++){
46         if(!ccw(poly[i],p,poly[i+1]))
47             return false;
48     }
49     return true;
50 }
51 //Distance from p to line (a,b)
52 double distToLine(const pt& p, const pt& a,
53     const pt &b){
54     vec ap = p-a;
55     vec ab = b-a;
56     double u = dot(ap,ab)/dot(ab,ab);
57     //Ignore for non-line segment
58     if(u < 0.0) //Closer to a
59         return abs(a-p);
60     if(u > 1.0) //Closer to b
61         return abs(b-p);
62     pt c = a+ab*u; // This is the point
63     return abs(c-p);
64 }

```

```

1  double area(vector<Point>v){ // Return the area of the convex hull in O
2      (n).
3      double ret=0.0;
4      int n=v.size();
5      for(int i=0; i<v.size(); i++){
6          ret+=v[i].x*(v[(i+1+n)%n].y-v[(i-1+n)%n].y);
7      }
8      return abs(ret/2);
9  }
10 double perimeter(vector<Point> v){ // Return the perimeter of the
11     convex hull in O(n).
12     double ans=0.0;
13     v.pb(v[0]);
14     for(int i=0; i<v.size()-1; i++){
15         ans+=sqrt((v[i].x-v[i+1].x)*(v[i].x-v[i+1].x)+(v[i].y-v[i
16             +1].y)*(v[i].y-v[i+1].y));
17     }
18     return ans;
19 }
19 Point rotate(Point c, Point po, double ang){ // Rotate po, around c by
20     ang.
21     /*
22      * 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

```
1  ll ncrmem[MXN][MXN];
2
3  ll ncr(int n, int r){
4      if(n==0)
5          return r==0;
6      if(r==0)
7          return 1;
8      if(ncrmem[n][r] != -1)
9          return ncrmem[n][r];
10     return ncrmem[n][r] = ncr(n-1, r-1) + ncr(n-1, r);
11 }
```

Ternary Search

```
1  double ternary_search(double l, double r) {
2      double eps = 1e-9;           //set the error limit here
3      while (r - l > eps) {
4          double m1 = l + (r - l) / 3;
5          double m2 = r - (r - l) / 3;
6          double f1 = f(m1);        //evaluates the function at m1
7          double f2 = f(m2);        //evaluates the function at m2
8          if (f1 < f2)
9              l = m1;
10         else
11             r = m2;
12     }
13     return f(l);                  //return the maximum of f(x) in [l, r]
14 }
```

Miller Rabin primality test

```
1  void factor(ll x, ll& e, ll& k){
2      while(x%2LL==0LL){
3          x/=2LL;
4          ++e;
5      }
6      k = x;
7  }
8
```

```
9  //increase x for higher certainty, 5 works well
10 bool is_prime(ll n, int x){
11     if(n&2LL==0 || n==1LL)
12         return false;
13     if(n==2 || n==3 || n==5 || n==7)
14         return true;
15     ll e, k;
16     factor(n-1,e,k);
17     while(x-->0){
18         ll a = (rand())%(n-5LL) + 2LL;
19         ll p = mod_exp(a,k,n);
20         if(p==1LL || p==n-1LL)
21             continue;
22         bool all_fail = true;
23         for(int i = 0; i < e-1; ++i){
24             p = mod_exp(p, 2, n);
25             if(p==n-1LL){
26                 all_fail = false;
27                 break;
28             }
29         }
30         if(all_fail)
31             return false;
32     }
33     return true;
34 }
```

Fast fourier transform

```
1  /* emaxx implementation */
2  /* Multiplication with arbitrary modulus
3   * use ntt if mod is prime and can be written as 2**k * c + 1
4   * if not, use Chinese Remainder Theorem
5   * or transform A(x) = A1(x) + A2(x)*c decompose into A(x)/c and A(x)%c
6   * B(x) = B1(x) + B2(x)*c
7   * where c ~= sqrt(mod)
8   * A * B = A1*B1 + c*(A1*B2 + A2*B1) * c**2(A2*B2)
9   * with all values < sqrt(mod) subpolynomials have coefficients <
10    mod * N after fft multiply decreasing changes of rounding error
11 * */
12 const double PI=acos(-1);
13
```

```

14 typedef complex<double> base;
15
16 void fft (vector<base> & a, bool invert) {
17     int n=(int) a.size();
18     for (int i=1, j=0; i<n; ++i) {
19         int bit=n>>1;
20         for (;j>=bit; bit>>=1)
21             j-=bit;
22         j+=bit;
23         if(i<j)
24             swap(a[i], a[j]);
25     }
26     for (int len=2; len<=n; len<<=1) {
27         double ang = 2*PI/len * (invert ? -1 : 1);
28         base wlen(cos(ang), sin(ang));
29         for (int i=0; i<n; i+=len) {
30             base w(1);
31             for (int j=0; j<len/2; ++j) {
32                 base u=a[i+j], v=a[i+j+len/2]*w;
33                 a[i+j]=u+v;
34                 a[i+j+len/2]=u-v;
35                 w*=wlen;
36             }
37         }
38     }
39     if (invert)
40         for(int i=0; i<n; ++i)
41             a[i]/=n;
42 }
43
44 // a, b => coefs to multiply, res => resulting coefs
45 // a[0], b[0], res[0] = coef x^0
46 // Doesnt work with negative coefs
47 void multiply(const vector<int> & a, const vector<int> & b, vector<int> &
48 res) {
49     vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());
50     size_t n=1;
51     while (n<max(a.size(), b.size())) n<<=1;
52     fa.resize(n), fb.resize(n);
53     fft (fa, false);  fft (fb, false);
54     for (size_t i=0; i<n; ++i)
55         fa[i]*=fb[i];
56     fft (fa, true);
57     res.resize (n);
58     // avoid precision errors, mess up with negative values of coefs
59     for(size_t i=0; i<n; ++i)
60         res[i]=int(fa[i].real() + 0.5);
61 }

```

Matrix Exponentiation

```

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

```

```

3 const int mod=1;
4
5 /* c=a*b */
6 void mu(ll a[][NMAT], ll b[][NMAT], ll c[][NMAT], int _n) {
7     for(int i=0; i<_n; i++)
8         for(int j=0; j<_n; j++) {
9             c[i][j]=0;
10            for(int h=0; h<_n; h++) {
11                c[i][j]+=(a[i][h]*b[h][j])%mod;
12                c[i][j]%=mod;
13            }
14        }
15 }
16 /*returns ans=mat^b*/
17 void power(ll ans[][NMAT], ll mat[][NMAT], ll b, int _n) {
18     ll tmp[NMAT][NMAT];
19     for(int i=0; i<_n; i++)
20         for(int j=0; j<_n; j++)
21             ans[i][j]=i==j;
22     while(b) {
23         if(b&1) {
24             mu(ans, mat, tmp, _n);
25             for(int i=0; i<_n; i++)
26                 for(int j=0; j<_n; j++)
27                     ans[i][j]=tmp[i][j];
28         }
29         mu(mat, mat, tmp, _n);
30         for(int i=0; i<_n; i++)
31             for(int j=0; j<_n; j++)
32                 mat[i][j]=tmp[i][j];
33         b>>=1;
34     }
35 }

```

Math Tricks

```

1 ll fexp(ll a, int x, ll mod){ // Fast exponenciation returns a^x
2     % mod
3     if(x==0)return 1ll;
4     if(x%2==0){
5         ll y=fexp(a, x/2, mod);
6         return (y*y)%mod;
7     }
8     return (a*fexp(a, x-1, mod))%mod;
9 }
10 ll divv(ll a, ll b, ll mod){ // Division with mod returns a/b % mod
11     return (a*fexp(b, mod-2, mod))%mod;
12 }
13
14 ll f[N];
15
16 ll fat(ll a, ll mod){ // Calculates factorial and stores in f %
17     mod

```

```

17     if(a<=1)return 1;
18     return f[a]?f[a]:(f[a]=(a*fat(a-1, mod))%mod);
19 }
20
21 ll choose(ll n, ll k, ll mod){ // Returns n choose k % mod
22     return divv(fat(n, mod), (fat(k, mod)*fat(n-k, mod))%mod, mod)%mod;
23 }
24
25 ll gcd(ll a, ll b){ // Greatest common divisor
26     return b?gcd(b, a%b):a;
27 }
28
29 ll lcm(ll a, ll b){ // Least common multiple
30     return (a*b)/gcd(a, b);
31 }
32
33 /* Fast factorization */
34
35 int p[N];
36
37 void start_fast(int MAX){ // Runs O(nlog(n)) Needs to be called to
38     use fast_fact or ammount_of_divisors.
39     for(int i=2; i<=MAX; i++){
40         if(p[i]==0){
41             for(int j=i; j<=MAX; j+=i){
42                 p[j]=i;
43             }
44         }
45     }
46 }
47
48 vector<int>fast_fact(int x){ // Fast factorization in O(log2(x))
49     vector<int>ret;
50     while(x>1){
51         ret.pb(p[x]);
52         x/=p[x];
53     }
54     return ret;
55 }
56
57 int amount_of_divisors(int x){ // Calculate the ammount of divisors of a
58     number in O(log2(x)) assume already ran start_fast.
59     if(x==1)return 1;
60     vector<int>v=fast_fact(x);
61     int ret=1, curr=2;
62     for(int i=1; i<v.size(); i++){
63         if(v[i]==v[i-1])curr++;
64         else{
65             ret*=curr;
66             curr=2;
67         }
68     }
69     return ret*curr;

```

```

69 }

```

Theorems

```

1 Picks theorem
2 Given a certain lattice polygon with non-zero area.
3
4 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.
5
6 Then, the Pick's formula states:
7
8  $S = I + \frac{B}{2}$ 
9
10
11 Burnsidess Lemma
12 Let G be the finite group of operations we can perform on X
13 The number of orbits of X is the average of the number of fixed points for
  each g in G
14 G must be closed

```