# 1 BFS

```
int dist[MX_N];
vector<int> adjList[MX_N];

int main(){
    for(int i = 0; i < MX_N; i++)
        dist[i]=INF;
    queue<int> q;
    q.push(0);
    dist[0] = 0;
    while(!q.empty()){
        int u = q.front(); q.pop();
        int d = dist[u];
        for(int i : adjList[u]){
            if(dist[i]==INF){
                dist[i]=d+1;
                q.push(i);
            }
        }
    }
    return 0;
}
```

# 2 Convex Hull

```
typedef complex<double> point;

double cross(const point& a, const point& b){
    return real(a)*imag(b) - imag(a)*real(b);
}

bool ccw(const point& a,const point& b, const point& c){
    if(a==b || b==c || a==c)
        return false;
    point u = b-c;
    point v = b-a;
    double cr = cross(u,v);
    return cr > 0;
}

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;
}
```

# 3 Dijkstras

```
struct path {
    int u,d;
    path(int _u, int _d) : u(_u), d(_d) {}
    path(){}
    bool operator < (const path& p) const {
        return d > p.d;
    }
};
for(int i =0; i < N; ++i)
    dist[i] = INF;
dist[S] = 0;
priority_queue<path> q;
q.push(path(S,0));
while(!q.empty()){
    path p = q.top(); q.pop();
    u = p.u,d = p.d;
    if(dist[u] < d)
        continue;
    for(auto v : adjList[u]){
        nd = d + v.second;
        if(nd < dist[v.first]){
            dist[v.first] = nd;
            q.push(path(v.first,nd));
        }
    }
}
```

# 4 Fenwick

```
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;
}
```

# 5 Inversion Count

```
int N;
int a[MX_N];
long long cnt=0;

void mergesort(int L, int R){
    if(L>=R)
        return;
    int mid = (L+R)/2;
    mergesort(L,mid);
    mergesort(mid+1,R);
    int n[R-L+1];
    int i = 0;
    int lp = L;
    int rp = mid+1;
    while(rp<=R || lp<=mid){
        if(rp<=R && lp <= mid){
            if(a[rp]<a[lp])
                n[i]=a[rp],rp++,cnt+=((long long) (mid-lp+1));
            else
                n[i]=a[lp],lp++;
        } else if(rp<=R){
            n[i]=a[rp++];
        } else {
            n[i]=a[lp++];
        }
        i++;
    }
    for(int j= L; j <= R; j++)
        a[j]=n[j-L];
}
```

# 6 Maximum Flow

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

int N,M,S,T,f;
int res[MX_N][MX_N];
int graph[MX_N][MX_N];
int dist[MX_N];
int p[MX_N];

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;
    while(1){
        f=0;
        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));
}
```

# 7 MCBM

```
int S,T,N,n,m,s,v;
vector<int> adjList[MX_N];
int res[MX_N][MX_N];
bool vis[MX_N];

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(){
    S=0;
    N=n+m+2;
    T = N-1;
    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);
}
```

# 8 MST

```
typedef pair<int,int> ii;
int p[MX_N],M,N;

// UFDS for sets
bool connected(int a, int b){ return find(a)==find(b);}

struct edge {
    int x,y,w;
    edge(int _x, int _y, int _w) : x(_x), y(_y), w(_w) {}
    bool operator < (edge e) const {
        return w < e.w;
    }
};

int main(){
    for(int i = 0; i < N; i++)
        p[i]=i;
    vector<edge> eList;
    vector<ii> treeList;
    int u,v,w;
    for(int i = 0; i < M; i++){
        scanf("%d %d %d",&u,&v,&w);
        eList.push_back(edge(u,v,w));
    }
    sort(eList.begin(),eList.end());
    int cost = 0;
    int sz=N;
    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");
}
```

# 9 LCA

```
int sptable[MX_LG][MX_SZ];
int V,P;

int H[MX_VP],E[MX_SZ],L[MX_SZ],vind;

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 = H[u];
        H[u] = H[v];
        H[v] = t;
    }
    // Segment Tree or sparse table for RMQ
    int ind = rmq(L,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]);
}

int main(){
    memset(H,-1,sizeof(H));
    vis(0,0);
    construct(vind,L);
    return 0;
}
```

# 10 Segment Tree

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

void construct(int p, int L, int R){
    if(L==R){
        tree[p] = a[L];
        return;
    }
    if(R<L)
        return;
    int md = (L+R)/2;
    construct(2*p,L,md);
    construct(2*p+1,md+1,R);
    tree[p] = min(tree[2*p],tree[2*p+1]);
}

void update(int p, int L, int R, int ind,int v){
    if(L==R){
        a[ind] = v;
        tree[p] = v;
        return;
    }
    int md = (L+R)/2;
    if(ind <= md)
        update(2*p,L,md,ind,v);
    else
        update(2*p+1,md+1,R,ind,v);
    tree[p] = min(tree[2*p],tree[2*p+1]);
}

int rmq(int p, int L, int R, int l, int r){
    if(r < L || l > R)
        return INF;
    if(l>=L && r<=R)
        return tree[p];
    int md = (l+r)/2;
    return min(rmq(2*p,L,R,l,md),rmq(2*p+1,L,R,md+1,r));
}
```

# 11 RectInHist

```
int R,C;
char board[MX_RC][MX_RC];
int h[MX_RC][MX_RC];

int perim(int l, int w){
    if(l==0 || w==0)
        return 0;
    return 2*l + 2*w;
}

int main(){
    for(int i = 0; i < R; i++){
        int run=0;
        for(int j = 0; j < C; j++){
            run = (board[i][j] == '.' ? run+1:0);
            h[i][j] = run;
        }
    }
    int mx = 0;
    for(int j = 0; j < C; j++){
        stack<int> s;
        for(int i = 0; i < R; i++){
            if(s.empty() || h[i][j] > h[s.top()][j])
                s.push(i);
            else if(h[i][j] < h[s.top()][j]){
                while(!s.empty() && h[i][j] < h[s.top()][j]){
                    int l = h[s.top()][j]; s.pop();
                    int pm = perim(l, (s.empty() ? i : i-s.top()-1));
                    mx = max(mx,pm);
                }
                s.push(i);
            } else if(h[i][j] == h[s.top()][j]) {
                s.pop();
                s.push(i);
            }
        }
        while(!s.empty()){
            int l = h[s.top()][j]; s.pop();
            int pm = perim(l, s.empty() ? R : R - s.top()-1);
            mx = max(mx,pm);
        }
    }
    printf("%d\n",mx-1);
}
```

# 12 SCC Tarjans

```
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;

map<int, int> sccMap;

void tarjans(int u){
    scc.push(u);
    vis[u]=true;

    dfs_low[u]=dfs_num[u]=dfsCounter++;

    for(int i = 0; i < adjList[u].size(); i++){
        int v = adjList[u][i];
        if(dfs_num[v]==0){
            tarjans(v);
            dfs_low[u]=min(dfs_low[u],dfs_low[v]);
        } else if(vis[v]){
            dfs_low[u]=min(dfs_low[u],dfs_num[v]);
        }
    }
    if(dfs_low[u]==dfs_num[u]){
        while(1){
            int v = scc.top(); scc.pop();
            sccMap[v]=sccIdx;
            vis[v]=false;
            if(v==u)
                break;
        }
        sccIdx++;
    }
}
```

# 13 NlogN LIS

```
int ls[MX_N];
int L[MX_N];
int I[MX_N];

void nlogn(){
    for(int i = 1; i < N+1; ++i)
        I[i]=INF;
    I[0] = -INF;
    int mx = 1;
    for(int i = 0; i < N; ++i){
        int ind = lower_bound(I,I+N+1,ls[i]) - I;
        I[ind] = ls[i];
        L[i] = ind;
        mx = max(mx,ind);
    }
    int prv = INF;
    vector<int> out;
    for(int i = N-1; i >= 0; --i){
        if(ls[i] < prv && L[i]==mx){
            out.push_back(ls[i]);
            prv = ls[i];
            mx--;
        }
    }
}
```

# 14 AP & Bridges

```
#include<bits/stdc++.h>
using namespace std;
const int MX_N = 101;
vector<int> adjList[MX_N];
int N;
int dfs_num[MX_N],dfs_low[MX_N];
int dfs_counter;
bool bridge;

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(){
    int P,x,y;
    while(scanf("%d %d",&N,&P),N||P){
        for(int i = 0; i < N; ++i)
            adjList[i].clear();
        memset(dfs_num,0,sizeof(dfs_num));
        memset(dfs_low,0,sizeof(dfs_low));
        bridge=false;
        dfs_counter=0;
        for(int i = 0; i < P; ++i){
            scanf("%d %d",&x,&y);
            adjList[x].push_back(y);
            adjList[y].push_back(x);
        }
        dfs(0,-1);
        for(int i = 0; i < N; ++i)
            if(dfs_num[i]==0)
                bridge=true;
        puts(bridge ? "Yes" : "No");
    }
    return 0;
}
```

# 15 Sparse Table

```
inline int rmq(int u, int v){
    if(u > v)
        return -2000000000;
    int k = (int) floor(log2((double) (v-u+1)));
    if(r[mtable[u][k]] > r[mtable[v-(1<<k) + 1][k]])
        return r[mtable[u][k]];
    return r[mtable[v-(1<<k) + 1][k]];
}

for(int i = 0; i < N; i++)
    mtable[i][0] = i;
for(int j = 1; (1 << j) <= N; j++)
    for(int i = 0; i + (1<<j)-1 < N; ++i)
        if(r[mtable[i][j-1]] > r[mtable[i + (1 << (j-1))][j-1]])
            mtable[i][j] = mtable[i][j-1];
        else
            mtable[i][j] = mtable[i+(1<<(j-1))][j-1];
```

# 16 Suffix Array

```
char * buff;
int RA[MX_N],SA[MX_N],tempRA[MX_N],tempSA[MX_N],N,c[MX_N];

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(){
    buff = new char[100011];
    char * doubled = new char[MX_N];
    cin.getline(buff,MX_N);
    N=strlen(buff);
    for(int i = 0; i < N; i++)
        SA[i]=i,RA[i]=buff[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];
    }
    delete buff;
    return 0;
}
```

# 17 Trie

```
struct node {
    node * children[26];
    int count;
    node(){
        memset(children,0,sizeof(children));
        count=0;
    }
};

void insert(node* nd, char *s){
    if(*s){
        if(!nd->children[*s-'a'])
            nd->children[*s-'a']=new node();
        insert(nd->children[*s-'a'],s+1);
    }
    nd->count++;
}

int count(node* nd, char *s){
    if(*s){
        if(!nd->children[*s-'a'])
            return 0;
        return count(nd->children[*s-'a'],s+1);
    } else {
        return nd->count;
    }
}

int main(){
    node * trie = new node();
    int N; scanf("%d",&N);
    char * buff = new char[40];
    for(int i = 0; i < N; i++){
        scanf("%s",buff);
        printf("%d\n",count(trie,buff));
        insert(trie,buff);
    }
    return 0;
}
```

# 18 UFDS

```
int find(int u){ return p[u] = (p[u] == u ? u : find(p[u])); }

inline void join(int a, int b){
    pa = find(a);
    pb = find(b);
    if(pa!=pb){
        if(rank[pa] < rank[pb]){
            ni = pb;
            pb = pa;
            pa = ni;
        }
        p[pb] = pa;
        if(rank[pa]==rank[pb])
            rank[pa]++;
    }
}
```

# 19 KMP

```
vector<int> buildFailure(string s){
    int n = s.size();
    vector<int> T(n+1,0);
    T[0]=-1;
    int j = 0;
    for(int i = 1; i < n;++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[n] = j;
    return T;
}

vector<int> search(string W, string S){
    int N = S.size();
    auto T=buildFailure(W);
    vector<int> p;
    int k = 0;
    int j = 0;
    while(j < N){
        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;
}
```

# 20 vimrc

```
set nocompatible
set autoindent " always set autoindenting on
set cindent
filetype indent on
filetype plugin on

set backup
set undofile

set history=50
set laststatus=2
imap jj <ESC>
nnoremap <CR> :noh<CR><CR>
set wildmenu

" Tabs"
set tabstop=8
set softtabstop=0
set expandtab
set shiftwidth=4
set smarttab
```