Data Structures and Algorithms for Competitive Programming

Eoin Davey

November 14, 2019

Contents

1 Graphs

Search

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;
}
```

Dijkstras

```
struct path {
    int u,d;
    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});
        }
    }
}
```

A*

```
struct path {
    int u, wght;
    bool operator<(const path& rhs) const {
        return wght > rhs.wght;
    }
};
int h(int x) { // heuristic function
    // Don't overestimate, should be 0 at sink
    return 0; // 0 for djikstras
}
for(int i =0; i < N; ++i)
    dist[i] = INF;
dis[S] = 0;
priority_queue<path> pq;
pq.push({s, h(s)});
while(!pq.empty()){
    path p = pq.top(); pq.pop();

    if(p.u == t)
        return p.wght;

    int d = p.wght - h(p.u);

    if(dis[p.u] < d)
        continue;

    for(auto v : adjList[p.u]) {
        int nd = d + v.second;

        if(dis[v.first] <= nd)
            continue;

        dis[v.first] = nd;

        pq.push({v.first , nd + h(v.first)});
    }
}
```

Bidirectional BFS

```
int bidirbfs(){
    queue<int> fq, rq;
    fq.push(S); dis[S] = 0;
    rq.push(T); disr[T] = 0;

    vector<int> proc;
    for(;;){
        if(fq.empty() && rq.empty())
            return -1; // disconnected
        if(!fq.empty()){
            int u = fq.front(); fq.pop();
            for(int v : adjList[u]){
                if(dis[v] <= dis[u] + 1)
```

```
                    continue;
                dis[v] = dis[u] + 1;
                fq.push(v);
            }
            vis[u] = true;
            proc.pb(u);
        }
        if(!rq.empty()){
            int u = rq.front(); rq.pop();
            for(int v : RadjList[u]){ // REVERSE adj
                if(disr[v] <= disr[u] + 1)
                    continue;
                disr[v] = disr[u] + 1;
                rq.push(v);
            }
            if(vis[u])
                break;
            proc.pb(u);
        }
    }

    int mn = INF;
    for(int u : proc)
        if(dis[u] != INF && disr[u] != INF)
            mn = min(mn, dis[u] + disr[u]);
    return mn;
}
```

Trees

MST

```
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");
}
```

LCA

```
/*
 * H[u] is first visit of u
 * E[x] is vertex at time x
 * L[x] is depth at time x
 * MEMSET H to -1
 * SET L to size of list * 2
 */
int 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 = 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]);
}
```

Centroid Decomposition

```
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;
}
```

SCC Tarjans

```
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++;
    }
}
```

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]);
    }
}
```

Network Flow

Edmond Karp Max Flow

```
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 Max Flow

```
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);
}
```

2 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;
}
```

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]++;
    }
}
```

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 mtable[u][k];
    return 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];
```

Segment Tree

```
int tree[MX_N*4 + 2];
int a[MX_N];
int N;

void construct(int p, int L, int R){
    if(L==R){
        tree[p] = 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] = a[tree[2*p]] < a[tree[2*p+1]] ? 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] = ind;
        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] = a[tree[2*p]] < a[tree[2*p+1]] ? 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;
    int lf = rmq(2*p,L,R,l,md);
    int rf = rmq(2*p+1,L,R,md+1,r);
    if(lf >= INF)
        return rf;
    if(rf >= INF)
        return lf;
    return a[lf] < a[rf] ? lf : rf;
}
```

Lazy Segment Tree

```
ll tree[4 * MX_N + 2];
ll d[4 * MX_N + 2];

void prop(int p, int l, int r){
    tree[p] += d[p] *(r-l+1);
    if(r>l){
        d[2*p]+=d[p];
        d[2*p + 1]+=d[p];
    }
    d[p] = 0;
}

ll rsq(int p, int ql, int qr, int l, int r) {
    prop(p, l, r);
    if(r < ql || l > qr)
        return 0;
    if(l >= ql && r <= qr)
        return tree[p];
    int md = (l+r)/2;
    return rsq(2*p, ql, qr, l, md)
        + rsq(2*p+1, ql, qr, md+1, r);
}

void rng_up(int p, int ql, int qr, ll v, int l, int r){
    prop(p,l,r);
    if(r < ql || l > qr)
        return;
    if(l>=ql && r <= qr){
        d[p]+=v;
        prop(p,l,r);
        return;
    }
    int md=(l+r)/2;
    rng_up(2*p, ql, qr, v, l, md);
    rng_up(2*p+1, ql, qr, v, md+1,r);
    tree[p] = tree[2*p] + tree[2*p+1];
}
```

Convex Hull Trick

```
// Convex hull trick, simpler if can sort insertions and queries
struct CvxHullTrickSimple {
    vector<ll> A;
    vector<ll> B;

    int ptr;

    cvxH():ptr(0){}

    // insert a descending
    void addLine(ll a, ll b){ // intersection of (A[len-2],B[len-2]) with
        (A[len-1],B[len-1]) must lie to the left of intersection of (A[len
        -1],B[len-1]) with (a,b)
```

```
        while (sz(A) >= 2 && (B[sz(B) - 2] - B[sz(B) - 1]) * (a - A[sz(A) - 1]) >= (B[sz(B) - 1] - b) * (A[sz(A) - 1] - A[sz(A) - 2])) {
            A.pop_back();
            B.pop_back();
        }
        A.pb(a);
        B.pb(b);
    }

    // query x ascending
    ll minValue(ll x) {
      ptr = min(ptr, sz(A) - 1);
      while (ptr + 1 < sz(A) && A[ptr + 1] * x + B[ptr + 1] <= A[ptr] * x + B[ptr]) {
        ++ptr;
      }
      return A[ptr] * x + B[ptr];
    }
};

struct CvxHullOpt {
    // Fully dynamic variant for use if can't guarantee
    // insertion or query order
    // stores lower envelope, negate lines and function to get upper
    static const ll qV = -(1LL<<50); //hacky special value
    struct line {
        ll a,b;
        mutable double xLeft;
        bool operator<(const line& l) const {
            if(l.a != qV)
                return a < l.a;
            return xLeft > l.xLeft;
        }
    };
    multiset<line> hull;

    ll xcomp(const line& a, const line& b, const line& c){
        //  < 0 => AxB < AxC
        //  > 0 => AxB > AxC
        return (a.a-c.a)*(b.b - a.b) - (c.b - a.b)*(a.a-b.a);
    }

    double xin(const line& a, const line& b){
        return (b.b - a.b)/(1.0*(a.a-b.a));
    }

    bool bad(set<line>::iterator y){
        auto z = next(y);
        if(y==hull.begin()){
            if(z==hull.end())
                return false;
            return y->a == z->a && y->b >= z->b;
        }
        auto x = prev(y);
        if(z==hull.end())
            return x->a == y->a && x->b <= y->b;
        return xcomp(*x, *y, *z) < 0;
    }

    void addLine(ll a, ll b){ // add line a*x + b
        CvxHullOpt::line l{a,b};
        auto y = hull.insert(l);
        if(bad(y)) { hull.erase(y); return;}
        while(next(y) != hull.end() && bad(next(y)))
            hull.erase(next(y));
        while(y!=hull.begin() && bad(prev(y)))
            hull.erase(prev(y));
        if(next(y)==hull.end())
            y->xLeft = -INF;
        else
            y->xLeft = xin(*y, *next(y));
        if(y != hull.begin())
            prev(y)->xLeft = xin(*prev(y), *y);
    }

    ll eval(ll x){
        auto l = hull.lower_bound({qV, 0, x});
        return l->a*x + l->b;
    }
};
```

3 Geometry

Convex Hull

```
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 Axioms

```
typedef complex<double> pt;
typedef complex<double> vec;
typedef vector<pt> pgon;
typedef struct { pt p,q; } lseg;
struct circ{ pt c; double r; };
struct rect{ pt p,q;}; // X(p) <= X(q) and Y(p) <= Y(q)

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 ab = 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);
}
//intersection pts of two circles
```

```
vector<pt> insct(const circ& a, const circ& b){
    vector<pt> o;
    double dist = abs(a.c - b.c);
    if(dist > a.r + b.r)
        return o; //none, don't touch
    if(abs(a.r-b.r) > dist)
        return o; //none, inside
    if(abs(dist - (a.r + b.r)) < EPS){ // one intersect
        pt p = a.c + (a.r/dist)*(b.c-a.c);
        o.pb(p);
        return o;
    }
    double delta = (sq(dist) + (a.r-b.r)*(a.r+b.r))/(2.0*dist);
    pt cent = a.c + (delta/dist)*(b.c-a.c);

    double h = sqrt(sq(a.r) - sq(delta));

    pt dVec = (b.c - a.c)/dist;
    o.pb(cent + h*pt(0,1)*dVec);
    o.pb(cent + h*pt(0,-1)*dVec);
    return o;
}

// intersection of two rectangles, sets none to true if no overlap
rect overlap(const rect& a, const rect& b, bool& none){
    rect r;
    if(X(a.p) > X(b.q) || Y(a.p) > Y(b.q)
        || X(b.p) > X(a.q) || Y(b.p) > Y(a.q)){
        none=true;
        return r;
    }
    r.p = {max(X(a.p),X(b.p)), max(Y(a.p),Y(b.p))};
    r.q = {min(X(a.q),X(b.q)), min(Y(a.q),Y(b.q))};
    return r;
}
```

4 Strings

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;
}
```

Trie

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

void insert(node* nd, char *s){
    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  }
```

KMP

```
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  }
```

5 Algorithms

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--;
        }
    }
}
```

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);
}
```

6 Maths

Miller Rabin

```
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;
}
```

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);
}
```

Gaussian Elimination

```
/*
 * mat is augmented matrix
 * e.g 3x + 4y = 2 is [3,4,2]
 */
void gauss(double mat[MX_N][MX_N+1], double ans[MX_N], int n){
    int i,j,k,l;double t;

    for(j = 0; j < n-1; ++j){
        l = j;
        for(i = j+1; i < n; ++i){
            if(fabs(mat[i][j]) > fabs(mat[l][j]))
                l=i;
        }
        for(k = j; k <= n; ++k)
            t=mat[j][k],mat[j][k]=mat[l][k],mat[l][k]=t;
        for(i = j+1; i < n; ++i)
            for(k = n; k >= j; --k)
                mat[i][k] -= mat[j][k] * mat[i][j] / mat[j][j];
    }

    for(j = n-1; j >= 0; --j){
        for(t=0.0, k = j+1;k<n;++k)
            t += mat[j][k] * ans[k];
        ans[j] = (mat[j][n] - t) / mat[j][j];
    }
}
```

Ternary Search

```
double ternary_search(double l, double r) { //maximises
    while (r - l > EPS) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);
}
```

Matrix Exponential

```
/* 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;
        }
}
```

Exponents

```
// a^b % md
ll modExp(ll a, int b, ll mod){
    if(b==0)
        return 1LL;
    if(b%2==0){
        ll y = modExp(a, b/2, mod);
        return (y*y)%mod;
    }
    return (a*modExp(a,b-1,mod))%mod;
}

// a*b^-1 % mod
// mod should be prime
ll modDiv(ll a, ll b, ll mod) {
    return (a * modExp(b, mod-2, mod))%mod;
}
```

Theorems

Burnside's Lemma

Let G be a finite group that acts on a set X. For each g let X^g denote the set of elements in X by g.

$$X^g = \{\ x \in X \mid g.x = x\}$$

Then the number of orbits is as follows

$$|X/G| = \frac{1}{|G|}\sum_{g\in G}|X^g|$$

Polya's enumeration theorem

Let X be a finite set and let G be a group of permutations of X. Let Y be a finite set of colors so that Y^X is the set of colored arrangements of X. Then the number of orbits of Y^X under G is

$$|Y^X/G| = \frac{1}{|G|}\sum_{g\in G}|Y|^{c(g)}$$

where $c(g)$ is the number of cycles in permutation g.