Caderninho do Sucesso

1 Grafo

DFS

```
void dfs(int v){
        cor[v]=1;
        for(int u:G[v])
                if(!cor[u]){
                        dist[u]=dist[v]+1;
                        dfs(u);
                }
}
```

DFS-ponte

```
void dfs(int u,int p){
        cor[u]=1;
        d[u]=low[u]=tempo++;
        for(int &v:G[u])
                if(!cor[v]){
                        dfs(v,u);
                        low[u]=min(low[u],low[v]);
                        if(low[v]>d[u])
                                //ponte encontrada se entrar aqui
                }else if(v!=p)
                        low[u]=min(low[u],d[v]);
}
```

BFS

```
void bfs(int v){
        queue<int> q;
        q.push(v);
        cor[v]=1;
        while(!q.empty()){
                int u=q.front();
                q.pop();
                for(int w:G[u])
                if(!cor[w]){
                        q.push(w);
                        cor[w]=1;
                        dist[w]=dist[u]+1;
                }
        }
}
```

Dijkstra-matriz

```
vector < pair< int, pii > > G[m][m];
int dist[m][m];
int Dijkstra(pair<int,int> v,pair<int,int> z){
        memset(dist,63,sizeof(dist));
        dist[v.first][v.second]=0;
        priority_queue< pair<int, pii > > pq;
        pq.push(make_pair(0,v));
        while(!pq.empty()){
                int uf = pq.top().second.first;
                int us = pq.top().second.second;
                int d = -pq.top().first;pq.pop();
                if(d>dist[uf][us]) continue;
                for(pair<int, pii > j:G[uf][us]){
                        int wf=j.second.first,ws=j.second.second,_d=j.first;
                        if(dist[wf][ws]>d+_d){
                                dist[wf][ws]=d+_d;
                                pq.push(make_pair(-dist[wf][ws],make_pair(wf,ws)));
                        }
                }
        }
        return dist[z.first][z.second];
}
```

Dijkstra

```
int dist[100002][3];// paridade da chegada no ponto,%3 (quantidade de arestas usadas)
//Dijkstra com paridade ate o destino, acessar dist[z][X] na main
void dijkstra(int v,int z){
        memset(dist,63,sizeof dist);
        priority_queue< pair<int,pair<int,int> > > pq;
        dist[v][0]=0;
        pq.push({0,{v,0}});
        while(!pq.empty()){
                int d=-pq.top().f;
                int u=pq.top().s.f;
                int p=pq.top().s.s;
                pq.pop();
                if(u==z) continue;
                if(d>dist[u][0] and d>dist[u][1] and d>dist[u][2]) continue;
                for(pair<int,int> j:G[u]){
                        int w=j.s,_d=j.f;
                        if(p==2){//max paridade
                                if(dist[w][0]>_d+dist[u][2]){
                                        dist[w][0]=_d+dist[u][2];
                                        pq.push({-dist[w][0],{w,0}});
                                }
                        }else{
                                if(dist[w][p+1]>_d+dist[u][p]){
                                        dist[w][p+1]=_d+dist[u][p];
                                        pq.push({-dist[w][p+1],{w,p+1}});
                                }
                        }
                }
        }
}
```

Floyd-Warshall

```
memset(dist,63,sizeof dist);
for(int k = 1;k <= n;k++)
        for(int i = 1;i <= n;i++)
                for(int j = 1;j <= n;j++)
                        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

FloodFill

```
auto MAT[N][M];
int dr[]={0,0,1,-1};
int dc[]={1,-1,0,0};
int floodfill(int r,int c,auto k,auto p){
        if(r>=N or r<0 or c>=M or c<0)
                return 0;
        if(MAT[r][c]!=p)
                return 0;
        int ans=1;
        for(int i=0;i<4;i++)
                ans+=floodfill(r+dr[di],c+dc[di],k,p);
        return ans;
}
```

Kuhn

```
//Minimum edge cover == Maximum Cardinality Bipartite Matching == MVC
//MIS(Maximum Independent Set)+MVC(max matching) = N(n s)
bool kuhn(int u)//max matching
{
        if(cor[u] == tempo)
                return 0;
        cor[u] = tempo;
        //random_shuffle(G[u].begin(), G[u].end(), [](int x){ return rand() % x; });
        for(const int &v : G[u])
                if(!b[v] or kuhn(b[v]))
                        return b[v] = u;
        return 0;
}
int main(){
        tempo = 1;
        int ans = 0;
        for(int i = 1; i <= na; i++)
                ans += kuhn(i), tempo++;
}
```

Kruskal

```
while(!pq.empty()){
        iii a=pq.top();
        pq.pop();
        if(find(a.s.f)!=find(a.s.s) ){
                join(a.s.f,a.s.s);
                mst.push_back(a); } }
```

LCA

```
int nivel[200002],ancestral[200002][20],table[200002][20];
vector< pair<int,int> > MST[200002];
void dfs(int v){
        for(pair<int,int> u:MST[v])
                if(nivel[u.second]==-1){
                        ancestral[u.second][0]=v;
                        table[u.second][0]=u.first;
                        nivel[u.second]=nivel[v]+1;
                        dfs(u.second);
                        sz[v]+=sz[u];
                }
        sz[v]++;
}
pair<int,int> LCA(int u,int v){
        if(nivel[u]<nivel[v]) swap(u,v);
        int m=0;
        for(int i=19;i>=0;i--)
                if(nivel[u]-(1<<i) >= nivel[v]){
                        m=max(m,table[u][i]);
                        u=ancestral[u][i];
                }
        if(u==v) return {v,m};
        for(int i=19;i>=0;i--)
                if(ancestral[u][i]!=-1 and ancestral[u][i]!=ancestral[v][i]){
                        m=max(m,max(table[u][i],table[v][i]));
                        u=ancestral[u][i],v=ancestral[v][i];
                }
        return {ancestral[u][0],max(m,max(table[u][0],table[v][0]))};
}
int main(){
        memset(nivel,-1,sizeof nivel);
        memset(pai,-1,sizeof pai);
        memset(ancestral,-1,sizeof ancestral);
        //input, grafo
        dfs(1);//grafo 1-idexado
        for(int i = 1; i<20; ++i)
                for(int j=1; j<=N; ++j)
                if(ancestral[j][i-1]!=-1){
                        ancestral[j][i] = ancestral[ancestral[j][i-1]][i-1];
                        table[j][i]=max(table[ancestral[j][i-1]][i-1],table[j][i-1]);
                }
        //Query -> elemento propagado na sparse table "table",k.second
        pair<int,int> k=LCA(Arestas[m].second.first,Arestas[m].second.second);
}//dist entre dois pontos com LCA nivel[a]+nivel[b]-2*nivel[lca], N-sz[aux1]-sz[aux2]
```

Kahn

```
vector<int> G[50002];
int grau[50002];
int main(){
        int N,M;
        cin >> N >> M;
        while(M--){
                int a,b;
                cin >> a >> b;
                G[a].push_back(b);
        grau[b]++;
    }
    vector<int> ts;
    set<int> S;
    for(int i=0;i<N;i++)
        if(!grau[i]) S.insert(i);
    int b=0;
    while(!S.empty()){ //Prioriza os menores valores
        int a=*S.begin();
        ts.push_back(a);
        S.erase(S.begin());
        for(auto &u:G[a]){
            grau[u]--;
            if(!grau[u])
                S.insert(u);
        }
    }
    if((int)ts.size()<N){
        cout << "*\n";
    }else{
        for(int i=0;i<ts.size();i++)
            cout << ts[i] << '\n';
    }
}
```

Fluxo Máximo

```
const int MAX = 1e4;
struct edge{
    int v, f, c;
    edge(){}
    edge(int _v, int _f, int _c){
        v = _v, f = _f, c = _c;
    }
};
vector<edge> edges;
vector<int> G[MAX];
int tempo = 1, cor[MAX], pai[MAX],N,M;
void add_edge(int u, int v, int cp, int rc){
    edges.push_back(edge(v, 0, cp));
    G[u].push_back(edges.size()-1);
    edges.push_back(edge(u, 0, rc));
    G[v].push_back(edges.size()-1);
}
int dfs(int s, int t, int f){
    if(s == t) return f;
    cor[s] = tempo;
    for(int e : G[s])
    if(cor[edges[e].v] < tempo and edges[e].c-edges[e].f > 0)
        if(int a = dfs(edges[e].v, t, min(f, edges[e].c-edges[e].f))){
            edges[e].f += a;
            edges[e^1].f -= a;
            return a;
        }
    return 0;
}
int MaxFlow(int s, int t){
    int mf = 0;
    while(int a = dfs(s, t, inf))
        mf += a, tempo++;
    return mf;
}
int main(){
    int s=0,t=1000;// pontos de base para a passagem do fluxo
    REP(i,N)
        add_edge(s,i+1,1,0);//pontos de entrada
    REP(i,M)
        add_edge(i+N+1,t,1,0);//pontos de saida
    REP(i,N)
        REP(j,M)
            if(abs((A[i])-B[j])<=1){
                add_edge(i+1,j+N+1,1,0);//pontos intermediarios
            }
    cout << MaxFlow(s,t) << '\n';//Chamada da funcao
}
```

Grafo fortemente Conexo

```
int N,M,t,dist[MAXN],gr[MAXN],ciclo,out[MAXN],cor[MAXN];
vi G[MAXN],GI[MAXN];
stack<int> s;
//OBS: o pedido da questao ira alterar os dados retirados das dfs`s
void dfs1(int v){
    cor[v]=1;
    for(int u:G[v])
        if(!cor[u])
            dfs1(u);
    dist[v]=t++;
    s.push(v);
}
void dfs2(int v,int di){
    cor[v]=0;
    gr[v]=ciclo;
    for(int u:GI[v])
        if(cor[u] and di>dist[u])
            dfs2(u,di);
}
int main(){
    cin >> N >> M;
    while(M--){
        int a,b;
        cin >> a >> b;
        G[b].pb(a);
        GI[a].pb(b);
    }
    FOR(i,1,N+1)
        if(!cor[i])
            dfs1(i);
    while(!s.empty()){
        int k=s.top();
        s.pop();
        if(cor[k])
            dfs2(k,dist[k]),ciclo++;
    }
}
```

2 Tecnicas de Programação

Busca Binária

```
binary_search(B.begin()+i,B.end(),B[i]+(s/3))
//              inicio         fim        valor
int i=0,j=N-1,meio=-1;
while(i<=j){
        meio=(i+j)/2;
        if(C[meio]==b) break;
        if(C[meio]>b) j=meio-1;
        else i=meio+1;
}
```

Merge Sort

```
int MergeSort(vi &A){
        if(A.size()<=1) return 0;
        vi B,C;
        REP(i,A.size()/2)
                B.pb(A[i]);
        FOR(i,A.size()/2,A.size())
                C.pb(A[i]);
        int ans=MergeSort(B)+MergeSort(C);
        B.pb(inf);
        C.pb(inf);
        int b=0,c=0;
        REP(i,A.size()){
                if(C[c]<B[b]){ A[i]=C[c++]; ans+=B.size()-b-1;
                }else A[i]=B[b++];
        }
        return ans;
}
```

Counting Sort

```
int vet[max];
int frec[max]; //vetor da frequencia dos elementos
void CoutingSort(){
        for(int i=0,c=0; i<max; i++)
                while(frec[i])
                        vet[c++]=i, frec[i]--;//coloca o valor no local certo
}
```

Max Sum

```
int max_sum(vector<int> &s){
        int resp=0,maior=0;
        for(int i=0;i<s.size();i++){
                maior=max(0,maior+s[i]);
                resp=max(resp,maior);
        }
        return resp;
}
```

LCS

```
//Dois vetores A e B
int dp(int a,int b){
        if(a>=N or b>=M) return 0;
        if(pd[a][b]!=-1) return pd[a][b];
        if(A[a]==B[b])
                return pd[a][b]=max(dp(a+1,b+1)+1,max(dp(a+1,b),dp(a,b+1)));
        return pd[a][b]=max(dp(a+1,b),dp(a,b+1));
}
//Um vetor A
int pd(int id,int v){
        if(id<0) return 0;
        if(dp[id][Map[v]]!=-1) return dp[id][Map[v]];
        if(A[id]<=v)
                return dp[id][Map[v]]=max(pd(id-1,A[id])+1,pd(id-1,v));
        return dp[id][Map[v]]=pd(id-1,v);
}
```

LIS

```
vector<char> lis(string &str){
        vector<char> pilha,resp;
        int pos[300002],pai[300002];
        for(int i=0;i<str.size();i++){
                vector<char>::iterator it=upper_bound(pilha.begin(),pilha.end(),str[i])
                                                //lower_bound -> elementos distintos
                int p=it-pilha.begin();
                if(it==pilha.end()) pilha.push_back(str[i]);
                else *it=str[i];
                pos[p]=i;
                if(p==0) pai[i]=-1;
                else pai[i]=pos[p-1];
        }
        int p=pos[pilha.size()-1];
        while(p>=0){
                resp.push_back(str[p]);
                p=pai[p];
        }
        reverse(resp.begin(),resp.end());
        return resp;
}
```

3 Programação Dinâmica

Max Sum 2D

```
for(int i=1;i<=N;i++)
        for(int j=1;j<=M;j++){
                scanf("%d",&A[i][j]);
                dp[i][j]=dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+A[i][j];
                for(int k=0;k<j;k++)
                        for(int l=0;l<i;l++)
                                saida=max(saida,dp[i][j]-dp[i][k]-dp[l][j]+dp[l][k]); }
```

DP-map

```
//Trabalho do Papa - codcad
#define inf 0x3f3f3f3f
pair<int,int> A[1010];
int pd[1010][1010],x,N;
map<int,int> Map;
pair<map<int,int>::iterator,bool > r;
bool cmp(pair<int,int> a,pair<int,int> b){
        return (a.second==b.second)?(a.first>b.first):(a.second>b.second);
}
int dp(int id,int p){
        if(p<0) return -inf;
        if(id>=N or !p) return 0;
    r=Map.insert({p,x});
    if(r.second) x++;
        if(pd[id][Map[p]]!=-1) return pd[id][Map[p]];
        return pd[id][Map[p]]=max(dp(id+1,min(p-A[id].first,
                                A[id].second-A[id].first))+1,dp(id+1,p));
}
int main(){
        memset(pd,-1,sizeof pd);
        cin >> N;
        for(int i=0;i<N;i++){
                cin >> A[i].first >> A[i].second;
        r=Map.insert({A[i].second,x});
        if(r.second) x++;
    }
        sort(A,A+N,cmp);
        cout << dp(0,1000001) << '\n';
}
```

DP-Tree

```
int dp(int v,bool flag){
        if(pd[v][flag]!=-1) return pd[v][flag];
        int cas1=0,cas2=0;
        if(flag) cas1=A[v];
        for(int u:G[v]){
                if(u!=pai[v]){
                        pai[u]=v;
                        cas1+=dp(u,false);
                        cas2+=dp(u,true);
                }
        }
        if(flag) return pd[v][flag]=max(cas1,cas2);
        return pd[v][flag]=cas2;
}
```

Sub-conjuntos

```
v[0]=1;
for(auto valor:Valores)
        FORI(int i=valor_MAX-valor;i>=0;i--)
                if(v[i])
                        v[i+valor]++;
```

Digit-DP

```
int dp[20][1000][2];
//varicoes apenas no retorno e no segundo parametro
int digitDP(int idx, int sum, int can, vector<int> &digit){
        if(idx == (int)digit.size())
                return sum%mod;
        if(dp[idx][sum][can] != -1)
                return dp[idx][sum][can];
        int ans = 0;
        for(int i = 0; i < 10; i++)
          if(can or i <= digit[idx])
            ans =(ans+ digitDP(idx+1,sum+i,can or i<digit[idx], digit)) % mod;
        return dp[idx][sum][can] = ans % mod;
}
int query(int x){
        memset(dp, -1, sizeof(dp));
    vector<int> digit;
    while(x){
        digit.push_back(x%10);
        x /= 10;
    }
    reverse(digit.begin(), digit.end());
    return digitDP(0, 0, 0, digit);
}
```

Kadane 2D

```
int A[N+1][N+1],pd[N+1][N+1];
for(int i=1;i<=N;i++)
        for(int j=1;j<=N;j++){
                scanf("%lld",&A[i][j]);
                pd[i][j]=pd[i][j-1]+A[i][j];
        }
int ans=0;
for(int i=1;i<=N;i++)
        for(int j=i+1;j<=N;j++){
                int sum=0;
                for(int k=1;k<=N;k++){
                        sum += pd[k][j] - pd[k][i-1];
                        if(sum<0) sum = 0;
                        ans = max(ans, sum); } }
```

Caixeiro Viajante

```
int tsp(int bitmask,int id){ //O((2^n)*(n^2))
        if(memo[bitmask][id]!=-1)
                return memo[bitmask][id];
        if(bitmask==((1<<N)-1))
                return dist[id][0];
        int ans=INT_MAX;
        for(int i=0;i<N;i++)
                if(!(bitmask&(1<<i)))
                        ans=min(ans,tsp((bitmask|(1<<i)),i)+dist[id][i]);
        return memo[bitmask][id]=ans; }
```

Knapsack

```
int peso[MAXobj],valor[MAXobj],tab[MAXobj][MAXpeso];
int knapsack(int obj, int aguenta){
        if(tab[obj][aguenta]>=0)
                return tab[obj][aguenta];
        if(obj==N or !aguenta)
                return tab[obj][aguenta]=0;
        int nao_coloca=knapsack(obj+1, aguenta);
        if(peso[obj]<=aguenta){
                int coloca=valor[obj]+knapsack(obj+1, aguenta-peso[obj]);
                return tab[obj][aguenta]=max(coloca, nao_coloca);
        }
        return tab[obj][aguenta]=nao_coloca;
}
```

4 Estrutura de Dados

Union-Find

```
int find(int x){
        return (pai[x]==-1)?x:pai[x]=find(pai[x]);
}
void join(int x,int y){
        x=find(x);
        y=find(y);
        pai[x]=y;
}
```

BIT

```
int bit[100000],N;
int update(int x,int v){
        while(x<=N){
                bit[x]+=v;
                x+=(x&-x);
        }
}
int sum(int x){
        int s=0;
        while(x>0){
                s+=bit[x];
                x-=(x&-x);
        }
        return s;
}
```

BIT2D

```
int bit[100000][100000],N,M;
int sum(int x,int y){
        int resp=0;
        for(int i=x;i>0;i-=(i&-i))
                for(int j=y;j>0;j-=(j&-j))
                        resp+=bit[i][j];
        return resp;
}
void update(int x,int y,int val){
        for(int i=x;i<N;i+=(i&-i))
                for(int j=y;j<M;j+=(j&-j))
                        bit[i][j]+=val;
}
```

Segment Tree

```
struct SegTreeLazy{
        int tree[400000]={0},lazy[400000]={0},arr[100000];// tree e lazy 4*tam_arr
        void build(int node,int left,int right){
                if(left==right){
                        tree[node]=arr[left];
                        return ;
                }
                int mid=(left+right)/2;
                build(2*node,left,mid);
                build(2*node+1,mid+1,right);
                tree[node]= tree[2*node]+tree[2*node+1];
        }
        void update(int node,int left,int right,int l,int r,int value){
                if(lazy[node]){
                        tree[node]=(right-left+1)*lazy[node];
                        if(right!=left){
                                lazy[2*node]=lazy[node];
                                lazy[2*node+1]=lazy[node];
                        }
                        lazy[node]=0;
                }
                if(left>r or l>right)
                        return ;
                if(left>=l and r>=right){
                        tree[node]=(right-left+1)*value;
                        if(right!=left){
                                lazy[2*node]=value;
                                lazy[2*node+1]=value;
                        }
                        return ;
                }
                int mid=(left+right)/2;
                update(2*node,left,mid,l,r,value);
                update(2*node+1,mid+1,right,l,r,value);
                tree[node]=tree[2*node]+tree[2*node+1];
        }
        int sum(int node,int l,int r,int left,int right){
                if(lazy[node]){
                        tree[node]=(right-left+1)*lazy[node];
                        if(right!=left){
                                lazy[2*node]=lazy[node];
                                lazy[2*node+1]=lazy[node];
                        }
                        lazy[node]=0;
                }
                if(left>r or l>right)
                        return 0;
                if(left>=l and r>=right)
                        return tree[node];
                int mid=(left+right)/2;
                return sum(2*node,l,r,left,mid)+sum(2*node+1,l,r,mid+1,right);
        }
};
typedef struct SegTreeLazy seg;
void pointupdate(int node,int left,int right,int idx,double value){
        if(left==right){
                tree[node]=value;
                return ;
        }
        int mid=(left+right)/2;
        if(left<=idx and idx<=mid)
                pointupdate(2*node,left,mid,idx,value);
        else
                pointupdate(2*node+1,mid+1,right,idx,value);
        tree[node]=tree[2*node]+tree[2*node+1];
}
```

Sparse Table

```
int table[MAXN][MAXN],arr[MAXN];
void buildSparseTable(int N){
        for(int i=0;i<N;i++)
                table[i][0]=arr[i];
        for(int j=1;(1LL<<j)<=N;j++)
                for(int i=0;(i+(1LL<<j))<=N;i++)
                        table[i][j]=min(table[i][j-1],table[i+(1LL<<(j-1))][j-1]);
}
int query(int l,int r){
        int j=log2(r-l+1);
        return min(table[l][j] , table[r-(1LL<<j)+1][j]);
}
```

5 Matemática

Crivo

```
int primos[m+1]; //m = valor maximo desejado
void crivo(){
        primos[1]=1;
        for(int i=2;i<=m;i++)
                if(primos[i]==0){
                        for(int j=2;i*j<=m;j++)
                                primos[i*j]++;
                }
}
```

Exponenciação Rápida

```
long long fast_expo(long long base,long long e){
        if(e==0) return 1;
        ll ans=fast_expo(base,e/2);
        ans= (ans*ans)%mod;
        if(e%2) ans=(ans*base)%mod;
        return ans;
}
```

Divisibilidade 11

```
int k=1;
for(int i=str.size()-1;i>=0;i--,k++){
        if(k%2==1) impar+=(int)(str[i]-'0');
        else par+=(int)(str[i]-'0');
}
if(impar<par){
        int a=abs(impar-par);
        double k= (double)a/11.0;
        impar+=(11*ceil(k));
}
cout << ((((impar-par)%11)==0)?"S\n":"N\n");
```

Inverso Multiplicativo

```
/// TIP /// inv(x) = x^(m-2) mod m ??? if m is prime and x<m
var phi(var n){
        auto f = fatorar(n);
        var res = 1;
        for(auto x: f){
                var fator = x.fi; var exp = x.se;
                res *= fexp(fator,exp-1);
                res *= fator-1;
        }
        return res;
}
var inv(var x, var mod){
        if(__gcd(x,mod)!=1) return -1;
        var _phi = phi(mod) - 1;
        return fexp(x,_phi,mod);
}
```

6 String

KMP

```
int N,M,Q,arr[10002];
string str1,str2,str;
void build(){
        int i=0,j=1;
        while(j<M){
                if(str2[i]==str2[j])
                        arr[j]= ++i;
                else{
                        i=0;
                        if(str2[i]==str2[j])
                                arr[j]=++i;
                }
                j++;
        }
}
int matching(){
        int i=0,j=0,cont=0;
        while(j<str.size()){
                if(str2[i]==str[j]) i++,j++;
                else if(i) i=arr[i-1];
                else j++;
                if(i==M)//matching na posicao return j-M
                        cont++;//quantidade de matching's
        }
        return cont;// ou -1, nao encontrado
}
int main(){
        cin >> str1 >> str2;
        build();
        while(Q--){
                int a,b;
                cin >> a >> b;
                str=str1.substr(a-1,(b-a)+1);
                cout << matching() << '\n';
        }
}
```

7 Geometria

Resumo

```
//Distancia entre dois pontos
double dist=sqrt((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2));
```

```
//Equacao da Circunferencia (Centro (a,b) e Raio r)
(x-a)^2+(y-b)^2=r^2
//Check se um circulo esta dentro do outro
if(dist<=(R1-R2))
//Check se um circulo esta fora do outro
if(dist>=(R1+R2))
//Condicao de existencia de um triangulo de lados A,B,C
if((abs(A-B)<C and C<(A+B)) and (abs(A-C)<B and B<(A+C))
   and (abs(B-C)<A and A<(B+C)))
//Formulas para um triangulo com lados a,b,c
   Semi-Perimetro => p = (a+b+c)/2
             Area => A = sqrt(p(p-a)(p-b)(p-c))
             Area => A = bc.sin(alpha)/2
           Altura => h = 2A/b
    Raio Inscrito => r = A/p
Raio Curcunscrito => R = (abc)/(4A)
```

Pontos e Retas

```
#define x first
#define y second.first
#define z second.second
typedef pair<double,pair<double,double> > point;
```

```
//[x,y,1] -> ponto
//[x,y,z] -> reta
```

```
// a*x+b*y+c*z=0
```

```
point reta(point a,point b){
        point resp;
        resp.x = a.y*b.z-a.z*b.y;
        resp.y = a.z*b.x-a.x*b.z;
        resp.z = a.x*b.y-a.y*b.x;
        return resp;
}
point intercessao(point a,point b){
        point i = reta(a,b);
        if(i.z!=0){//reduz ao ponto, i.z==0 -> retas paralelas
                i.x/=i.z;
                i.y/=i.z;
                i.z/=i.z;
        }
        return i;
}
reta1 = reta(p1,p2);
intercessao1= intercessao(reta1,reta2);
//se existe intercessao no intervalor p1 p2
if(intercessao1.y>0.0 and intercessao1.y<min(p1.y,p2.y) and
intercessao1.x>min(p1.x,p2.x) and intercessao1.x<max(p1.x,p2.x))
```

Convex Hull

```
#define X first
#define Y second
typedef pair<int, int> ii;
int cross(ii O, ii A, ii B){
        return (((A.X - O.X) * (B.Y - O.Y)) - ((A.Y - O.Y) * (B.X - O.X)));
}
vector<ii> ConvexHull(vector<ii> P){
        if(P.size() <= 1) return P;
        vector<ii> H(2*P.size());
        int k = 0;
        sort(P.begin(), P.end());
        //lower hull
        for(int i = 0; i < P.size(); i++){
                while(k >= 2 and cross(H[k-2], H[k-1], P[i]) < 0) k--;
                H[k++] = P[i];
        }
        //upper hull
        for(int i = P.size()-2, l = k + 1; i >= 0; i--){
                while(k >= l and cross(H[k-2], H[k-1], P[i]) < 0) k--;
                H[k++] = P[i];
        }
        H.resize(k-1);
        return H;
}
int main(){
        int n, x, y;
        vector<ii> P;
        cin >> n;
        while(n--){
                cin >> x >> y;
                P.push_back({x, y});
        }
        vector<ii> H = ConvexHull(P);
        for(int i = 0; i < H.size(); i++)
                cout << H[i].X << '␣' << H[i].Y << '\n';
}
```

8 Extra

Limites

```
limite do int -> (1<<31)-1 = 2147483647 = 2*10^9
limite unsigned int -> (1<<32)-1 = 4294967295 = 4*10^9
limite do long long -> (1LL<<63)-1 = 9223372036854775807 = 9*10^(18)
limite unsigned long long -> (1LL<<64)-1 = 18446744073709551615 = 10^(19)
```

Bits

```
int couting_bits(int N){
        int i;
        for(i=0;N;i++)
                N&=(N-1);
        return i;
}
__builtin_popcount(int N);
__builtin_popcountll(ll N);
//Portas Logicas
and 1 & 1 = 1, 0 & X = 0
or 1 | X = 1
xor 1 ^ 0 = 1, X ^ X = 0
Conjunto |=(1<<i); //inserir elemento
& intersecao de dois conjuntos
| uniao de dois conjuntos
```

MO’s Algorithm

```
//Questao do CF - Powerful array - 86/D
#define int long long
int N,Q,arr[200002],block,sum,OC[1000002],out[200002];
struct _Query{
        int i,L,R;//i -> indice da query, L -> left, R -> Right
        /*o indece serve para retornar para a ordem original dos pedidos*/
};
typedef struct _Query Query;
void add(int p){//Funcao pra add na resposta do range (varia)
        sum+=(p*((OC[p]<<1)+1));
        OC[p]++;
}
void rem(int p){//Funcao pra remover na resposta do range (varia)
        OC[p]--;
        sum-=(p*((OC[p]<<1)+1));
}
bool cmp(Query a,Query b){//sqrt Decomposition
        if(a.L/block!=b.L/block)
                return a.L < b.L;
        if((a.L/block)&1)
                return a.R<b.R;
        return a.R>b.R;
}
void MoAlgorithm(Query A[],int Q){
        int left=0,right=-1;
        for(int i=0;i<Q;i++){
                Query &q=A[i];
                while(left<q.L) rem(arr[left++]);
                while(left>q.L) add(arr[--left]);
                while(right<q.R) add(arr[++right]);
                while(right>q.R) rem(arr[right--]);
                out[q.i]=sum;//Criar o vetor out[] evita um Q*logQ
        }
}
main(){
        scanf("%lld␣%lld",&N,&Q);
        for(int i=0;i<N;i++)
                scanf("%lld",&arr[i]);
        block=(int)sqrt(N);
        Query A[Q];
        for(int i=0;i<Q;i++){
                scanf("%lld␣%lld",&A[i].L,&A[i].R);
                A[i].L--,A[i].R--;
                A[i].i=i;
        }
        sort(A,A+Q,cmp);//ordena em blocos de sqrt(N)
        MoAlgorithm(A,Q);
        for(int i=0;i<Q;i++)
                printf("%lld\n",out[i]);
}
```

SQRT Decomposition

```
int F[100002][320],A[100002],block;//[valMax][sqrt(N)]
void update(int id,int W){
        F[A[id]][id/block]--;
        F[W][id/block]++;
        A[id]=W;
}
int query(int x,int y,int W){
        int x1=x/block,y1=y/block,out=0;
        for(int i=x1+1;i<y1;i++)
                out+=F[W][i];
        if(x1==y1){
                for(int i=x;i<=y;i++)
                        if(A[i]==W)
                                out++;
                return out;
        }
        for(int i=x;(i/block)==x1;i++)
                if(A[i]==W)
                        out++;
        for(int i=y;(i/block)==y1;i--)
                if(A[i]==W)
                        out++;
        return out;
}
void build(){
        for(int i=1;i<=N;i++)
                F[A[i]][i/block]++;
}
```