1 Bellman Ford's.cpp

```
#define MAX 2050
/// complexity V*E
/// 0 based
struct Edge{
    int a,b,w;
};
Edge adj[MAX];
int dist[MAX];
void init(int n){
    for(int i=0;i<n;i++){
        dist[i]=oo;
    }
    return ;
}
bool bellmanford(int s,int n,int m){
    dist[s]=0;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(dist[adj[j].a]==oo) continue;
            if(dist[adj[j].a]+adj[j].w<dist[adj[j].b]){
                dist[adj[j].b]=dist[adj[j].a]+adj[j].w;
            }
        }
    }
    for(int i=0;i<m;i++){
        if(dist[adj[i].b]>adj[i].w+dist[adj[i].a])
        return true;
    }
    return false;
}
int main(){
    int n,m,test,u,v,c,source,destination;
    scanf("%d",&test);
    while(test--){
        scanf("%d%d",&n,&m);
        init(n);
        for(int i=0;i<m;i++){
            scanf("%d%d%d",&u,&v,&c);
            adj[i].a=u;
            adj[i].b=v;
            adj[i].w=c;
        }
        if(bellmanford(0,n,m)) printf("possible\n");
        else printf("not possible\n");
    }
    return 0;
}
```

2 Blossom Algorithm.cpp

```
const int MAXN = 505; // number of elements.
int n; //n no. of vertices.
vector<int> g[MAXN];
int match[MAXN]; //stores the matcings
int p[MAXN]; //array of ancestors.
int base[MAXN]; //Node numbering after compression.
int q[MAXN]; //Queue
bool used[MAXN], blossm[MAXN];
int lca (int a, int b){
    bool used[MAXN] = { 0 };
// From the node a climb up to the roots,
//marking all even vertices
    for (;;){
        a = base[a];
        used[a] = true;
        if (match[a] == -1) break; // Got the root
        a = p[match[a]];
    }
// Climb from node b,
//until we find the marked vertex
    for (;;){
        b = base[b];
        if (used[b]) return b;
        b = p[match[b]];
```

```
    }
}
void mark_path (int v, int b, int children){
    while (base[v] != b){
        blossm[base[v]]=blossm[base[match[v]]]=true;
        p[v] = children;
        children = match[v];
        v = p[match[v]];
    }
}
int find_path (int root){
    mem(used,0);
    mem(p,-1);
    for (int i=0; i<n; ++i)
        base[i] = i;
    used[root] = true;
    int qh=0, qt=0;
    q[qt++] = root;
    while (qh < qt){
        int v = q[qh++];
        for (int i=0; i<g[v].size(); ++i){
            int to = g[v][i];
            if (base[v] == base[to]
                    || match[v] == to) continue;
            if (to == root || match[to] != -1
                    && p[match[to]] != -1){
                int curbase = lca (v, to);
                mem(blossm,0);
                mark_path (v, curbase, to);
                mark_path (to, curbase, v);
                for (int i=0; i<n; ++i)
                    if (blossm[base[i]]){
                        base[i] = curbase;
                        if (!used[i]){
                            used[i] = true;
                            q[qt++] = i;
                        }
                    }
            }
            else if (p[to] == -1){
                p[to] = v;
                if (match[to] == -1) return to;
                to = match[to];
                used[to] = true;
                q[qt++] = to;
            }
        }
    }
    return -1;
}
int graph_match()
{
    int ret = 0;
    mem(match,-1);
    for (int i=0; i<n; ++i)
        if (match[i] == -1){
            int v = find_path (i);
            if(v!=-1) ret++;
            while (v != -1){
                int pv = p[v], ppv = match[pv];
                match[v] = pv, match[pv] = v;
                v = ppv;
            }
        }
    return ret;
}
int main(){
    int i,j;
    scanf("%d",&n);
    while(scanf("%d %d",&i,&j)==2){
        i--,j--;
        g[i].psb(j);
        g[j].psb(i);
    }
    int ans = graph_match();
    printf("%d\n",ans*2);
```

```
    for(i=0; i<n; i++)
        if(match[i]>-1){
            printf("%d %d\n",i+1,match[i]+1);
            match[match[i]] = -1;
        }
    return 0;
}
```

3 Dinic's-Maxflow.cpp

```
///V^2*E Complexity
///number of augment path * (V+E)
///Base doesn't matter

const int INF = 2000000000;
const int MAXN = 100;///total nodes
const int MAXM = 10000;///total edges

int N,edges;
int last[MAXN],prev[MAXM],head[MAXM];
int Cap[MAXM],Flow[MAXM];
int dist[MAXN];
int nextEdge[MAXN];///used for keeping track of next edge of
    ith node

queue<int> Q;

void init(int N){
    edges=0;
    memset(last,-1,sizeof(int)*N);
}

//cap=capacity of edges , flow = initial flow
inline void addEdge(int u,int v,int cap,int flow){
    head[edges]=v;
    prev[edges]=last[u];
    Cap[edges]=cap;
    Flow[edges]=flow;
    last[u]=edges++;
    head[edges]=u;
    prev[edges]=last[v];
    Cap[edges]=0;
    Flow[edges]=0;
    last[v]=edges++;
}

inline bool dinicBfs(int S,int E,int N){
    int from=S,to,cap,flow;
    memset(dist,0,sizeof(int)*N);
    dist[from]=1;
    while(!Q.empty()) Q.pop();
    Q.push(from);
    while(!Q.empty()){
        from=Q.front();Q.pop();
        for(int e=last[from];e>=0;e=prev[e]){
            to=head[e];
            cap=Cap[e];
            flow=Flow[e];
            if(!dist[to] && cap>flow){
                dist[to]=dist[from]+1;
                Q.push(to);
                ///Important
                if(to==E) return true;
                ///Need to be sure
            }
        }
    }
    return (dist[E]!=0);
}

inline int dfs(int from,int minEdge,int E){
    if(!minEdge) return 0;
    if(from==E) return minEdge;
    int to,e,cap,flow,ret;
    for(;nextEdge[from]>=0;nextEdge[from]=prev[e]){
```

```
        e=nextEdge[from];
        to=head[e];
        cap=Cap[e];
        flow=Flow[e];
        if(dist[to]!=dist[from]+1) continue;
        ret=dfs(to,min(minEdge,cap-flow),E);
        if(ret){
            Flow[e]+=ret;
            Flow[e^1]-=ret;
            return ret;
        }
    }
    return 0;
}

int dinicUpdate(int S,int E){
    int flow=0;
    while(int minEdge = dfs(S,INF,E)){
        if(minEdge==0) break;
        flow+=minEdge;
    }
    return flow;
}

int maxFlow(int S,int E,int N){
    int totFlow=0;
    while(dinicBfs(S,E,N)){
        for(int i=0;i<=N;i++) nextEdge[i]=last[i];/// update
            last edge of ith node
        totFlow+=dinicUpdate(S,E);
    }
    return totFlow;
}

int main(){
    return 0;
}
```

4 Extend Euclid.cpp

```
int sign_(ll a,ll b){
    if(a<0 && b>0) return -1;
    if(a>0 && b<0) return -1;
    return 1;
}

ll Floor(ll a,ll b){
    ll F=a/b + (!(a%b == 0))*(sign_(a,b) < 0?-1:0);
    return F;
}

ll Ceil(ll a,ll b){
    ll C=a/b + (!(a%b == 0))*(sign_(a,b) < 0?0:1);
    return C;
}

ll GCD(ll a,ll b){
    if(b==0) return a;
    return GCD(b,a%b);
}

ll EGCD(ll a,ll b,ll &X,ll &Y){
    if(b==0){
        X=1;
        Y=0;
        return a;
    }
    ll x=-(a/b),PX,r;
    r=EGCD(b,a%b,X,Y);
    PX=X;
    X=Y;
    Y=(Y*x)+(PX);
    return r;
}
```

```
vector<pair<ll,ll>> find_any_solution(ll a,ll b,ll c){
    ll x0,y0,x,y;
    ll g=EGCD(a,b,x0,y0);
    if(c%g) return vector<pair<ll,ll>>();
    vector<pair<ll,ll>> retSol;
    x=(c/g)*x0;
    y=(c/g)*y0;
    retSol.psb(mp(x,y));
    return retSol;
}

//ax+by=c
//x=x1+(b/g)*t
//y=y1-(a/g)*t

vector<pair<ll,ll>> find_all_solution_in_range(ll a,ll b,ll c
    ,ll mina,ll maxa,ll minb,ll maxb){
    ll x0,y0,x1,y1,x,y;
    ll g=EGCD(a,b,x0,y0);
    if(g && c%g) return vector<pair<ll,ll>>();
    vector<pair<ll,ll>> retSol;
    if(a==0 && b==0){
        if(c==0)
            for(ll T1=mina;T1<=maxa;T1++)
                for(ll T2=minb;T2<=maxb;T2++)
                    retSol.psb(mp(T1,T2));
        return retSol;
    }
    else if(a==0){
        if(c/b>=minb&&c/b<=maxb)
            for(ll T=mina;T<=maxa;T++)
                retSol.psb(mp(T,c/b));
        return retSol;
    }
    else if(b==0){
        if(c/a>=mina&&c/a<=maxa)
            for(ll T=minb;T<=maxb;T++)
                retSol.psb(mp(c/a,T));
        return retSol;
    }
    x1=(c/g)*x0;
    y1=(c/g)*y0;
    ll minT1,maxT1, minT2,maxT2,minT,maxT,a1,b1;
    a1=b/g;
    b1=a/g;
    minT1=Ceil(mina-x1,a1);
    maxT1=Floor(y1-minb,b1);
    minT2=Ceil(y1-maxb,b1);
    maxT2=Floor(maxa-x1,a1);
    minT=max(minT1,minT2);
    maxT=min(maxT1,maxT2);
    for(ll T=minT;T<=maxT;T++){
        x=x1+a1*T;
        y=y1-b1*T;
        if(x<mina||x>maxa || y<minb||y>maxb) continue;
        retSol.psb(mp(x1+a1*T , y1-b1*T));
    }
    return retSol;
}

ll count_all_solution_in_range(ll a,ll b,ll c,ll mina,ll maxa,
    ll minb,ll maxb){
    ll x0,y0,x1,y1,x,y;
    ll g=EGCD(a,b,x0,y0);
    if(g && c%g) return 0;
    if(a==0 && b==0){
        if(c==0) return (maxa-mina+1)*(maxb-minb+1);
        return 0;
    }
    else if(a==0){
        if(c/b>=minb&&c/b<=maxb) return (maxa-mina+1);
        return 0;
    }
    else if(b==0){
        if(c/a>=mina&&c/a<=maxa) return (maxb-minb+1);
```

```
        return 0;
    }
    x1=(c/g)*x0;
    y1=(c/g)*y0;
    ll minT1,maxT1,minT2,maxT2,minT,maxT,a1,b1;
    a1=b/g;
    b1=a/g;
    minT1=Ceil(mina-x1,a1);
    maxT2=Floor(maxa-x1,a1);
    minT2=Ceil(y1-maxb,b1);
    maxT1=Floor(y1-minb,b1);
    minT=max(minT1,minT2);
    maxT=min(maxT1,maxT2);
    return max(maxT-minT+1,0ll);
}

int main(){
    ll x,y;
    ll a,b,c,mina,maxa,minb,maxb,tmp;
    cin>>a>>b;
    EGCD(a,b,x,y);
    deb(x,y);
    return 0;
    int sa,sb;
    vector<pair<ll,ll>> solution;
    while(SF("%lld %lld %lld %lld %lld %lld %lld",&a,&b,&c,&
        mina,&maxa,&minb,&maxb)==7){
        sa=1;
        sb=1;
        if(a<0){
            a=-a;
            tmp=mina;
            mina=-maxa;
            maxa=-tmp;
            sa=-1;
        }
        if(b<0){
            b=-b;
            tmp=minb;
            minb=-maxb;
            maxb=-tmp;
            sb=-1;
        }
        ll totSol = count_all_solution_in_range(a,b,c,mina,
            maxa,minb,maxb);
        PF("Total solution : %lld\n",totSol);
        solution=find_all_solution_in_range(a,b,c,mina,maxa,
            minb,maxb);
        int n=SZ(solution);
        for(int i=0;i<n;i++){
            solution[i].fs*=sa;
            solution[i].sc*=sb;
            deb("Solution",i+1,":","X =",solution[i].fs,"Y =",
                solution[i].sc);
        }
    }
    return 0;
}
```

5 Hashing.cpp

```
const int MAX1 = 100010;
ll P1=59272331ll;
ll P2=84592337ll;
int ID[555];
ll PowerP1[MAX1];
ll PowerP2[MAX1];
ll InvP1[MAX1];
ll InvP2[MAX1];
ll HashStr1P1[MAX1];
ll HashStr1P2[MAX1];
ll HashStr2P1[MAX1];
ll HashStr2P2[MAX1];
void MakePowerInv(int n,int B){
    PowerP1[0]=1;
    PowerP2[0]=1;
```

```
    InvP1[0]=1;
    InvP2[0]=1;
    ll IP1=BigMod(B,P1-2,P1);
    ll IP2=BigMod(B,P2-2,P2);
    for(int i=1;i<=n;i++){
        PowerP1[i]=(PowerP1[i-1]*B)%P1;
        PowerP2[i]=(PowerP2[i-1]*B)%P2;
        InvP1[i]=(InvP1[i-1]*IP1)%P1;
        InvP2[i]=(InvP2[i-1]*IP2)%P2;
    }
    return ;
}

void MakeHashTable(int n){
    for(int i=0;i<n;i++){
        HashStr1P1[i]=(HashStr1P1[i]*PowerP1[i] + (i?
            HashStr1P1[i-1]:0))%P1;
        HashStr1P2[i]=(HashStr1P2[i]*PowerP2[i] + (i?
            HashStr1P2[i-1]:0))%P2;

        HashStr2P1[i]=(HashStr2P1[i]*PowerP1[i] + (i?
            HashStr2P1[i-1]:0))%P1;
        HashStr2P2[i]=(HashStr2P2[i]*PowerP2[i] + (i?
            HashStr2P2[i-1]:0))%P2;
    }
    return ;
}
char Str1[MAX1];
char Str2[MAX1];
int solve(int n,int k,int &st){
    vector<pair<int,int> > mpp;
    vector<pair<int,int> > :: const_iterator it;
    ll hash1,hash2;
    for(int i=0;i<n-k+1;i++){
        hash1=(HashStr1P1[i+k-1]-(i==0?0:HashStr1P1[i-1]))*
            InvP1[i];
        hash1%=P1;
        if(hash1<0) hash1+=P1;
        hash2=(HashStr1P2[i+k-1]-(i==0?0:HashStr1P2[i-1]))*
            InvP2[i];
        hash2%=P2;
        if(hash2<0) hash2+=P2;

        mpp.psb(mp(hash1,hash2));
    }
    sort(all(mpp));
    for(int i=0;i<n-k+1;i++){
        hash1=(HashStr2P1[i+k-1]-(i==0?0:HashStr2P1[i-1]))*
            InvP1[i];
        hash1%=P1;
        if(hash1<0) hash1+=P1;
        hash2=(HashStr2P2[i+k-1]-(i==0?0:HashStr2P2[i-1]))*
            InvP2[i];
        hash2%=P2;
        if(hash2<0) hash2+=P2;
        if(binary_search(all(mpp),mp((int)hash1,(int)hash2))){
            st=i;
            return true;
        }
    }
    return false;
}

int main(){
    int idx;
    int n,test;
    SF("%d",&n);
    SF("%s",&Str1);
    SF("%s",&Str2);
    idx=0;
    clrall(ID,-1);
    for(int i=0;i<n;i++){
        if(ID[Str1[i]]==-1) ID[Str1[i]]=idx++;
        if(ID[Str2[i]]==-1) ID[Str2[i]]=idx++;
        HashStr1P1[i]=HashStr1P2[i]=ID[Str1[i]];
```

```
        HashStr2P1[i]=HashStr2P2[i]=ID[Str2[i]];
    }
    MakePowerInv(n,idx);
    MakeHashTable(n);
    int lo=1,hi=n,mid,res=0,st=-1;
    while(lo<=hi){
        mid=(lo+hi)>>1;
        if(solve(n,mid,st)){
            lo=mid+1;
            res=mid;
        }
        else hi=mid-1;
    }
    if(st==-1) PF("\n");
    else{
        for(int i=st;i<st+res;i++) PF("%c",Str2[i]);
        PF("\n");
    }
    return 0;
}
```

6 ConvexHullTrick.cpp

```
const int MAX = 1000100;
//pointer=0,last=0 should be made initially
ll M[MAX],C[MAX]; //y=mx+c we need only m(slope) and c(
    constant)
//Returns true if either line l1 or line l3 is always better
    than line l2
bool bad(int l1,int l2,int l3){
        /*
        intersection(l1,l2) has x-coordinate (c1-c2)/(m2-m1)
        intersection(l1,l3) has x-coordinate (c1-c3)/(m3-m1)
        set the former greater than the latter, and cross-
            multiply to
        eliminate division
        */
        //if the query x values is non-decreasing (reverse(>
            sign) for vice verse)
        return (C[l3]-C[l1])*(M[l1]-M[l2])<=(C[l2]-C[l1])*(M[
            l1]-M[l3]);
}

//Adding should be done serially
//If we want minimum y coordinate(value) then maximum valued m
    should be inserted first
//If we want maximum y coordinate(value) then minimum valued m
    should be inserted first
void add(long long m,long long c,int &last){
        //First, let's add it to the end
        M[last]=m;
        C[last++]=c;
        //If the penultimate is now made irrelevant between
            the antepenultimate
        //and the ultimate, remove it. Repeat as many times as
             necessary
        //in short convex hull main convex hull tecnique is
            applied here
        while(last>=3&&bad(last-3,last-2,last-1)){
                M[last-2]=M[last-1];
                C[last-2]=C[last-1];
                last--;
        }
}

//Returns the minimum y-coordinate of any intersection between
    a given vertical
//line(x) and the lower/upper envelope(pointer)
//This can only be applied if the query of vertical line(x) is
    already sorted
//works better if number of query is huge
long long query(long long x,int &pointer,int last){
        //If we removed what was the best line for the
            previous query, then the
        //newly inserted line is now the best for that query
```

```
        if (pointer>=last)
                pointer=last-1;
        //Any better line must be to the right, since query
            values are
        //non-decreasing
    // Min Value wanted... (reverse(> sign) for max value)
        while (pointer<last-1 && M[pointer+1]*x+C[pointer+1]<=
            M[pointer]*x+C[pointer])
                pointer++;
        return M[pointer]*x+C[pointer];
}

//for any kind of query(sorted or not) it can be used
//it works because of the hill property
//works better if number of query is few
long long bs(int st,int end,long long x,int last){
    int mid=(st+end)/2;
    // Min Value wanted... (reverse(> sign) for max value)
    if(mid+1<last && M[mid+1]*x+C[mid+1]<M[mid]*x+C[mid])
        return bs(mid+1,end,x,last);
    // Min Value wanted... (reverse(> sign) for max value)
    if(mid-1>=0 && M[mid-1]*x+C[mid-1]<M[mid]*x+C[mid]) return
        bs(st,mid-1,x,last);
    return M[mid]*x+C[mid];
}


int main(){
    return 0;
}
```

7 KnuthTrick.cpp

```
const int MAX = 1020;
ll dp[MAX][MAX];
int mid[MAX][MAX];
ll pos[MAX];

ll knuthTrick(int n){
    for(int s=0;s<=n;s++){
        for(int L=0;L+s<=n;L++){
            int R=L+s;
            if(s<2){
                dp[L][R]=0;
                mid[L][R]=L;
                continue;
            }
            int midLeft = mid[L][R-1];
            int midRight = mid[L+1][R];
            ll cost = pos[R]-pos[L];
            ll cur;
            dp[L][R]=(1ll<<60);
            for(int M=midLeft;M<=midRight;M++){
                cur = dp[L][M]+dp[M][R]+cost;
                if(dp[L][R]>cur){
                    dp[L][R] = cur;
                    mid[L][R] = M;
                }
            }
        }
    }
    return dp[0][n];
}

int main()
{
    int n;
    ll m;
    while(cin>>m>>n){
        pos[0]=0;
        pos[n+1]=m;
        for(int i=1;i<=n;i++){
            cin>>pos[i];
        }
        cout<<knuthTrick(n+1)<<"\n";
    }
```

```
    return 0;
}
```

8 Histogram.cpp

```
/**
1 based indexing
*/
const int MAX = 50100;
int val[MAX];
int l[MAX],r[MAX];
int n;
void find_left_right(){
    for(int i=1;i<=n;i++) l[i]=r[i]=i;
    for(int i=1;i<=n;i++){
        while(l[i]>1 && val[i]<=val[l[i]-1]){
            l[i]=l[l[i]-1];
        }
    }
    for(int i=n;i>0;i--){
        while(r[i]<n && val[i]<=val[r[i]+1]){
            r[i]=r[r[i]+1];
        }
    }
    return ;
}

int main()
{
    return 0;
}
```

9 2D-Seg+lazy.cpp

```
/// 1 based
/// find sum with lazyProp
const int MAX = 120;
int stree[75536];
int lazy[75536];
bool grid[MAX][MAX];
int update(int,int,int,int,int,int,int,int,int,int);
int point_query(int,int,int,int,int,int,int);
void relax(int,int,int,int,int,int);

/// x1,y1,mx,my
/// x1,my+1,mx,y2
/// mx+1,y1,x2,my
/// mx+1,my+1,x2,y2

void relax(int idx,int x1,int y1,int x2,int y2,int val){
    int mx=(x1+x2)>>1,my=(y1+y2)>>1,nidx=(idx<<2);
    stree[idx]+=(x2-x1+1)*(y2-y1+1)*val;
    if(!(x1==x2 && y1==y2)){
        lazy[nidx]+=val;
        lazy[nidx|1]+=val;
        lazy[nidx|2]+=val;
        lazy[nidx|3]+=val;
    }
    lazy[idx]=0;
    return ;
}

int update(int idx,int x1,int y1,int x2,int y2,int ux1,int uy1
    ,int ux2,int uy2,int val){
    if(x1>x2 || y1>y2) return 0;///invalid
    if(lazy[idx]){
        relax(idx,x1,y1,x2,y2,lazy[idx]);
    }
    if((ux1<=x1&&ux2>=x2) && (uy1<=y1&&uy2>=y2))///inside the
        given rectangle.{
        relax(idx,x1,y1,x2,y2,val);
        return stree[idx];
    }
    if(x2<ux1||x1>ux2) return stree[idx];///outside the given
```

```
        rectangle.
    if(y2<uy1||y1>uy2) return stree[idx];///outside the given
        rectangle.
    int mx=(x1+x2)>>1,my=(y1+y2)>>1,nidx=(idx<<2);
    int ret=0;
    ret+=update(nidx,x1,y1,mx,my,ux1,uy1,ux2,uy2,val);///upper
         left side
    ret+=update(nidx|1,x1,my+1,mx,y2,ux1,uy1,ux2,uy2,val);///
        upper right side
    ret+=update(nidx|2,mx+1,y1,x2,my,ux1,uy1,ux2,uy2,val);///
        lower left side
    ret+=update(nidx|3,mx+1,my+1,x2,y2,ux1,uy1,ux2,uy2,val);//
        /lower left side
    stree[idx]=ret;
    return stree[idx];
}

int point_query(int idx,int x1,int y1,int x2,int y2,int x,int
    y){
    if(x1>x2 || y1>y2) return 0;///invalid
    if(lazy[idx]){
        relax(idx,x1,y1,x2,y2,lazy[idx]);
    }
    if((x==x1&&x==x2) && (y==y1&&y==y2))///inside the given
        rectangle.{
        return stree[idx];
    }
    if(x2<x||x1>x) return 0;///outside the given rectangle.
    if(y2<y||y1>y) return 0;
    int mx=(x1+x2)>>1,my=(y1+y2)>>1,nidx=(idx<<2);
    int ret=0;
    ret+=point_query(nidx,x1,y1,mx,my,x,y);///upper left side
    ret+=point_query(nidx|1,x1,my+1,mx,y2,x,y);///upper right
        side
    ret+=point_query(nidx|2,mx+1,y1,x2,my,x,y);///lower left
        side
    ret+=point_query(nidx|3,mx+1,my+1,x2,y2,x,y);///lower left
         side
    return ret&1;
}

void printGrid(int n,int m){
    int cur;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cur=grid[i][j]^point_query(1,1,1,n,m,i,j);
            PF("%d",cur);
        }
        puts("");
    }
}

int main(){
    int test,n,m,r,c;
    char ch;
    while(SF("%d %d %d %d",&n,&m,&r,&c)==4 && (n+m+r+c)){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++){
                SF(" %c",&ch);
                grid[i][j]=(bool) (ch-'0');
            }
        }
        int res=0;
        clrall(stree,0);
        clrall(lazy,0);
        bool cur;
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++){
                cur=grid[i][j]^point_query(1,1,1,n,m,i,j);
                if(cur==false) continue;
                if(i+r>n+1 || j+c>m+1){
                    res=-1;
                    i=n+1;
                    j=m+1;
                    continue;
                }
```

```
                res++;
                update(1,1,1,n,m,i,j,i+r-1,j+c-1,1);
            }
        }
        PF("%d\n",res);
    }
    return 0;
}
```

10 Mo's Algo.cpp

```
///1 based
const int MAX = 200010;
struct data{
    int l,r,Size,id;
    data(){}
    data(int l,int r,int Size,int id):l(l),r(r),Size(Size),id(
        id) {}
};

bool comp(const data& a,const data& b){
    if(a.l/a.Size==b.l/b.Size) return a.r<b.r;
    return (a.l/a.Size<b.l/b.Size);
}
data info[MAX];
ll a[MAX];
ll qans[MAX];
int freq[MAX*10];
void add(int p,ll &ans){
    ans+=(a[p]*((freq[a[p]]<<1ll) + 1));
    freq[a[p]]++;
    return ;
}

void remove(int p,ll &ans){
    freq[a[p]]--;
    ans-=(a[p]*((freq[a[p]]<<1ll) + 1));
    return ;
}

void solve(){
    int n,q;
    SF("%d %d",&n,&q);
    for(int i=0;i<n;i++) SF("%I64d",&a[i+1]);
    int Size = 1+(int)sqrt(n);
    for(int i=0;i<q;i++){
        SF("%d %d",&info[i].l,&info[i].r);
        info[i].Size=Size;
        info[i].id=i;
    }
    sort(info,info+q,comp);
    int lp=1,rp=0,lq,rq;
    ll ans=0;
    for(int i=0;i<q;i++){
        lq=info[i].l;
        rq=info[i].r;
        while(lp<lq) { remove(lp,ans); lp++; }
        while(rp<rq) { rp++; add(rp,ans); }
        while(rp>rq) { remove(rp,ans); rp--; }
        while(lp>lq) { lp--; add(lp,ans); }
        qans[info[i].id]=ans;
    }
    for(int i=0;i<q;i++) PF("%I64d\n",qans[i]);
}

int main()
{
    solve();
    return 0;
}
```

11 TreeDp.cpp

```
const int MAX = 100500;
const int M = 1000000007;
ll dprec[MAX][3];
```

```
vector<ll> dpgen[MAX][3];
vector<int> adj[MAX];
bool isBlack[MAX];
ll rec(int,int,int,int);
ll gen(int,int,int,int,int);
ll gen(int u,int par,int edgeId,int haveBlack,int needBlack){
    if(edgeId==SZ(adj[u])) return ((ll) haveBlack);
    int state = (haveBlack*2+needBlack);
    ll ret=dpgen[u][state][edgeId];
    if(ret!=-1) return ret;
    int v=adj[u][edgeId];
    if(v==par) return gen(u,par,edgeId+1,haveBlack,needBlack);
    ret=dpgen[u][state][edgeId]=0;
    ret=(rec(v,u,0,0)*gen(u,par,edgeId+1,haveBlack,needBlack))
        %M;
    if(!haveBlack){
        ret=(ret+(rec(v,u,0,1)*gen(u,par,edgeId+1,1,0))%M)%M;
        ret=(ret+(rec(v,u,1,0)*gen(u,par,edgeId+1,0,1))%M)%M;
    }
    else{
        ret=(ret+(rec(v,u,1,0)*gen(u,par,edgeId+1,haveBlack,
            needBlack))%M)%M;
    }
    dpgen[u][state][edgeId]=ret;
    return ret;

}

ll rec(int u,int par,int haveBlack,int needBlack){
    if(haveBlack && isBlack[u]) return 0;
    int state = (haveBlack*2+needBlack);
    ll &ret = dprec[u][state];
    if(ret!=-1) return ret;
    ret=0;
    int curHaveBlack=(haveBlack|isBlack[u]);
    ret = (gen(u,par,0,curHaveBlack,!curHaveBlack))%M;
    return ret;
}
```

```
int main(){
    int n,v;
    SF("%d",&n);
    for(int i=0;i<n-1;i++){
        SF("%d",&v);
        adj[i+1].psb(v);
        adj[v].psb(i+1);
    }
    for(int i=0;i<n;i++){
        for(int j=0;j<SZ(adj[i]);j++){
            dpgen[i][0].psb(-1);
            dpgen[i][1].psb(-1);
            dpgen[i][2].psb(-1);
        }
    }
    clrall(dprec,-1);
    for(int i=0;i<n;i++) SF("%d",&isBlack[i]);
    PF("%I64d\n",rec(0,0,0,0));
    return 0;
}
```

12 TrieLoop.cpp

```
#define lim 3000100 //total number of characters

struct trie{
  int adj[2];
  trie(){
      clrall(adj,-1);
  }
};
int nodeindx;
trie node[lim];
int n,m,cnt;
void initialize(){
    nodeindx=0;
```

```
    node[nodeindx]=trie();
}

void maketrie( string &text){
    int now,nextnode=0;
    for(int indx=0; text[indx]!='\0' ; indx++){
        now = (text[indx]-'a');
        if(node[nextnode].adj[now]==-1){
            node[nextnode].adj[now]=++nodeindx;///child
                created
            node[nodeindx]=trie();
        }
        else cnt++;
        nextnode = node[nextnode].adj[now];
    }
    return ;
}
int main()
{
    int prv,n,test,cas=0;
    string s;
    cin>>test;
    while(test--)
    {
        initialize()
        cin>>s;
        maketrie(s);
    }
    return 0;
}
```

13 BIT.cpp

```
///0 based indexing
const int MAX = 100050;
const int M = 1000000007;
int tree[MAX],bitN;
void init (int n){
        bitN = n;
        clrall(tree,0);
}
///for min or max use result = min/max(result,tree[r]);
inline int query (int r){
        int result = 0;
        for (; r >= 0; r = (r & (r+1)) - 1){
        result = (result + tree[r]);
        if(result>=M) result-=M;
    }
        return result;
}

inline int query (int l, int r){
        return ((query (r) - query (l-1))%M + M)%M;
}
///for min or max use tree[i] = min/max(tree[i],delta);
inline void update (int i, int delta){
    for (; i < bitN; i = (i | (i+1))){
        tree[i] = (tree[i] + delta);
        if(tree[i]>=M) tree[i]-=M;
    }
}
int val[MAX],idx[MAX];
inline bool comp(const int &a,const int &b){
    if(val[a]==val[b]) return a>b;
    return val[a]<val[b];
}
int main(){
    int test,cas=0,value,tmp,ans;
    SF("%d",&test);
    while(test--){
        init(bitN);
        SF("%d",&bitN);
        for(int i=0;i<bitN;i++) SF("%d",&val[i]);
        for(int i=bitN-1;i>-1;i--) idx[i]=i;
        sort(idx,idx+bitN,comp);
        ans=0;
```

```
        for(int i=0;i<bitN;i++){
            value=query(idx[i]-1);
            ans+=value+1;
            if(ans>=M) ans-=M;
            update(idx[i],(value+1));
        }
        PF("Case %d: %d\n",++cas,ans);
    }
    return 0;
}
```

14 TernarySearch3D.cpp

```
struct point{
    double x,y,z;
    point(){}
    point(double x,double y,double z):x(x),y(y),z(z){}
    void input(){
        SF("%lf %lf %lf",&x,&y,&z);
        return ;
    }
};

inline double getDist(point A,point B){
    double dx=(A.x-B.x);
    double dy=(A.y-B.y);
    double dz=(A.z-B.z);
    return (double) sqrt(dx*dx + dy*dy + dz*dz);
}
double Ternary_Search(point A,point B,point P){
    point LowP,HigP;
    double distL,distH;
    double res=0.0;
    int cnt=48;
    while(cnt--){
        LowP.x=(2.0*A.x+B.x)/3.0;
        LowP.y=(2.0*A.y+B.y)/3.0;
        LowP.z=(2.0*A.z+B.z)/3.0;
        HigP.x=(A.x+2.0*B.x)/3.0;
        HigP.y=(A.y+2.0*B.y)/3.0;
        HigP.z=(A.z+2.0*B.z)/3.0;
        distL=getDist(LowP,P);
        distH=getDist(HigP,P);
        if(distL<distH){
            B=HigP;
            res=distL;
        }
        else{
            A=LowP;
            res=distH;
        }
    }
    return res;
}
int main(){
    int test,cas=0;
    SF("%d",&test);
    point A,B,P;
    double res;
    while(test--){
        A.input();
        B.input();
        P.input();
        res=Ternary_Search(A,B,P);
        PF("Case %d: %.12lf\n",++cas,res);
    }
    return 0;
}
```

15 SuffixArray*.cpp

```
#define MAX 100000

string text;
int revSA[MAX],SA[MAX];
```

```
int cnt[MAX] , nxt[MAX];
bool bh[MAX],b2h[MAX];
int LCP[MAX];

bool cmp(const int &i,const int &j){
    return text[i]<text[j];
}
void sortFirstChar(int n)
{
    /// sort for the first char ...
    for(int i =0 ; i<n ; i++) SA[i] = i;
    sort(SA,SA+n ,cmp);
    ///indentify the bucket ........
    for(int i=0 ; i<n ; i++){
        bh[i] = (i==0 || text[SA[i]]!=text[SA[i-1]]);
        b2h[i] = false;
    }
    return;
}

int CountBucket(int n){
    int bucket = 0;
    for(int i =0 ,j; i<n ; i=j){
        j = i+1;
        while(j<n && bh[j]==false) j++;
        nxt[i] = j;
        bucket++;
    }
    return bucket;
}

void SetRank(int n){
    for(int i = 0 ; i<n ; i=nxt[i]){
        cnt[i] = 0;
        for(int j =i ; j<nxt[i] ; j++){
            revSA[SA[j]] = i;
        }
    }
    return;
}

void findNewRank(int l,int r,int step){
    for(int j = l ; j<r ; j++){
        int pre = SA[j] - step;
        if(pre>=0){
            int head = revSA[pre];
            revSA[pre] = head+cnt[head]++;
            b2h[revSA[pre]] = true;
        }
    }
    return;
}

void findNewBucket(int l,int r,int step){
    for(int j = l ; j<r ; j++){
        int pre = SA[j] - step;
        if(pre>=0 && b2h[revSA[pre]]){
            for(int k = revSA[pre]+1 ; b2h[k] && !bh[k] ; k++)
                b2h[k] = false;
        }
    }
    return;
}

void buildSA(int n){
    ///start sorting in logn step ...
    sortFirstChar(n);
    for(int h =1 ; h<n ; h<<=1){
        if(CountBucket(n)==n) break;
        SetRank(n);
        /// cause n-h suffix must be sorted
        b2h[revSA[n-h]] = true;
        cnt[revSA[n-h]]++;

        for(int i = 0 ; i<n ; i=nxt[i]){
            findNewRank(i,nxt[i] , h);
```

```
            findNewBucket(i , nxt[i] , h);
        }
        ///set the new sorted suffix array ...
        for(int i =0 ; i<n ; i++){
            SA[revSA[i]] = i;
            bh[i] |= b2h[i]; ///new bucket ....
        }
    }
    return;
}

void buildLCP(int n){
    int len = 0;
    for(int i = 0 ;i<n ; i++) revSA[SA[i]] = i;
    for(int i =0 ; i< n ; i++){
        int k = revSA[i];
        if(k==0){
            LCP[k] = 0;
            continue;
        }
        int j = SA[k-1];
        while(text[i+len]==text[j+len]) len++;
        LCP[k] = len;
        if(len) len--;
    }
    return;
}
void printSA(){
    for(int i=0;i<SZ(text);i++) printf("%2d ",SA[i]),cout<<
        text.substr(SA[i])<<endl;
    puts("");
    for(int i=1;i<SZ(text);i++) printf("%2d\n",LCP[i]);
    puts("");
    return ;
}
int main(){
    string a,b;
    int n,p,q;
    int tcase,cas=1;
    scanf("%d",&tcase);
    while(tcase--){
        cin>>a>>b;
        text=a+"$"+b;
        buildSA(SZ(text));
        buildLCP(SZ(text));
        printSA();
    }
    return 0;
}
```