目录

1. 图论

0.费用流

求最大费用最大流直接边权变负数
求最小费用，要在 dis[t]>0break
Double 型费用流，加 eps 精度控制

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF=1e9+7;
const int maxn=5e3+5;
const int maxm=1e5+5;
int n,m,s,t,maxflow=0,lowflow;
struct node {
    int to,flow,next,dis;
} edge[maxm];
int
head[maxn],cnt,mincost,vis[maxn],dis[maxn],incf[maxn],pre[maxn]
;
void add(int u,int v,int flow,int dis) {
    edge[++cnt].next=head[u];
    edge[cnt].to=v;
    edge[cnt].flow=flow;
    edge[cnt].dis=dis;
    head[u]=cnt;
}
struct mincostMaxflow {
    void init() {
        mincost=maxflow=0;
        cnt=1;
        memset(head,0,sizeof head);
    }
    void addedge(int u,int v,int w,int dis) {
        add(u,v,w,dis);
        add(v,u,0,-dis);
    }
    bool spfa() {
        queue<int>q;
        memset(dis,0x3f,sizeof dis);
        memset(vis,0,sizeof vis);
        dis[s]=0,vis[s]=1,q.push(s);
        incf[s]=(1<<30);
        while(!q.empty()) {
            int u=q.front();
            q.pop();
            vis[u]=0;
            for(int i=head[u]; i; i=edge[i].next) {
                if(!edge[i].flow)
                    continue;
                int v=edge[i].to,dist=edge[i].dis;
```

```
                if(dis[v]>dis[u]+dist) {
                    dis[v]=dis[u]+dist;
                    incf[v]=min(incf[u],edge[i].flow);
                    pre[v]=i;
                    if(!vis[v])vis[v]=1,q.push(v);
                }
            }
        }
        if(dis[t]>1e9)return 0;
        return 1;
    }
    void Dinic() {
        while(spfa()) {
            int x=t,i;
            maxflow+=incf[t];
            mincost+=dis[t]*incf[t];
            while(x!=s) {
                i=pre[x];
                edge[i].flow-=incf[t];
                edge[i^1].flow+=incf[t];
                x=edge[i^1].to;
            }
        }
    }
} ac;

int main() {
    ac.init();
    scanf("%d%d%d%d",&n,&m,&s,&t);
    for(int i=1; i<=m; i++) {
        int u,v,w,dis;
        scanf("%d%d%d%d",&u,&v,&w,&dis);
        ac.addedge(u,v,w,dis);
    }
    ac.Dinic();
    printf("%d %d\n",maxflow,mincost);
}
```

1.最大流

Zkw 费用流常数小一点，或者时间戳优化匈牙利

最大匹配输出方案：

```
for(int u=1; u<=m; u++) {
        for(int i=head[u]; i; i=edge[i].next) {
```

```
            int v=edge[i].to,w=edge[i].w;
            if(edge[i].w==0&&edge[i^1].w==1&&v!=s)
                printf("%d %d\n",u,v);
        }
    }
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int INF=1e9+7;
const int N=3e3+5;
const int M=2e5+5;
int n,m,s,t,maxflow=0,lowflow;
struct node {
    int to,w,next;
} edge[M];
int head[N],cnt=1,dep[N],inque[N],cur[N];
void add(int u,int v,int w) {
    edge[++cnt].next=head[u];
    edge[cnt].to=v;
    edge[cnt].w=w;
    head[u]=cnt;
}
struct Maxflow {
    bool bfs() {
        queue<int>q;
        memset(dep,-1,sizeof dep);
        dep[s]=0;
        q.push(s);
        while(!q.empty()) {
            int u=q.front();
            q.pop();
            inque[u]=0;
            for(int i=head[u]; i; i=edge[i].next) {
                int v=edge[i].to,w=edge[i].w;
                if(dep[v]==-1&&w) {
                    dep[v]=dep[u]+1;
                    q.push(v);
                }
            }
        }
        return dep[t]!=-1;
    }
    int dfs(int u,int flow) {
        if(u==t) {
```

```
            return flow;
        }
        int rlow=0,used=0;
        for(int i=cur[u]; i; i=edge[i].next) {
            int v=edge[i].to,w=edge[i].w;
            if(w&&dep[v]==dep[u]+1) {
                if(rlow=dfs(v,min(flow,w))) {
                    used+=rlow;
                    flow-=rlow;
                    edge[i].w-=rlow;
                    edge[i^1].w+=rlow;
                    if(flow==0)
                        break;
                }
            }
        }
        if(!used)dep[u]=-1;
        return used;
    }
    int Dinic() {
        maxflow=0;
        while(bfs()) {
            for(int i=1; i<=t; i++)
                cur[i]=head[i];
            maxflow+=dfs(s,INF);
        }
        return maxflow;
    }
    void init() {
        cnt=1;
        memset(head,0,sizeof head);
    }
    void addedge(int u,int v,int w) {
        add(u,v,w);
        add(v,u,0);
    }
} ac;

int main() {
}
```

2. 最大权闭合子图

所谓闭合子图就是给定一个有向图，从中选择一些点组成一个点集 V。对于 V 中任意一个点，其后续节点

都仍然在 V 中。
因为我们完成一个工作需要一些前置技能，所以我们把工作向他所需要的前置技能建立一条边。每个技能向他的前置技能建一条边。那么我们可以发现。我们如果需要完成一个工作。那么该工作的所有后继边所连接的技能我们都要学会，正好对应最大闭合权子图的性质
建立源点和工作连一条边，容量为该工作所获得的赏金。建立汇点，每个技能和汇点建边，容量为该技能的花费。其他边容量设为 INF 即可。

3. 最小点覆盖，最小边覆盖，最大团，最大独立集

二分图的最小顶点覆盖
定义：假如选了一个点就相当于覆盖了以它为端点的所有边。最小顶点覆盖就是选择最少的点来覆盖所有的边。
方法：最小顶点覆盖等于二分图的最大匹配

二分图中最小边覆盖=顶点数-最小点覆盖（最大匹配）
最小边覆盖：实质是个边集，这个集合里的边能覆盖所有的点，最小边覆盖是满足这个要求的所有边集中边数最少的一个
这里顶点数等于总的顶点数，是二分图两边的顶点数，不是一边

二分图的最大独立集
定义：选出一些顶点使得这些顶点两两不相邻，则这些点构成的集合称为独立集。找出一个包含顶点数最多的独立集称为最大独立集。
方法：最大独立集=所有顶点数-最小顶点覆盖

二分图的最大团
定义：对于一般图来说，团是一个顶点集合，且由该顶点集合诱导的子图是一个完全图，简单说，就是选出一些顶点，这些顶点两两之间都有边。最大团就是使得选出的这个顶点集合最大。对于二分图来说，我们默认为左边的所有点之间都有边，右边的所有顶点之间都有边。那么，实际上，我们是要在左边找到一个顶点子集 X，在右边找到一个顶点子集 Y，使得 X 中每个顶点和 Y 中每个顶点之间都有边。
方法：二分图的最大团=补图的最大独立集。
补图的定义是：对于二分图中左边一点 x 和右边一点 y，若 x 和 y 之间有边，那么在补图中没有，否则有。
这个方法很好理解，因为最大独立集是两两不相邻，所以最大独立集的补图两两相邻。

4. 最小路径覆盖

定义：通俗点将，就是在一个有向图中，找出最少的路径，使得这些路径经过了所有的点。
最小路径覆盖分为最小不相交路径覆盖和最小可相交路径覆盖。
最小不相交路径覆盖：每一条路径经过的顶点各不相同。
最小可相交路径覆盖：每一条路径经过的顶点可以相同。
特别的，每个点自己也可以称为是路径覆盖，只不过路径的长度是 0。

DAG 的最小不相交路径覆盖

算法：把原图的每个点 V 拆成 Vx 和 Vy 两个点，如果有一条有向边 A->B，那么就加边 Ax−>By。这样就得到了一个二分图。那么最小路径覆盖=原图的结点数-新图的最大匹配数。

证明：一开始每个点都是独立的为一条路径，总共有 n 条不相交路径。我们每次在二分图里找一条匹配边就相当于把两条路径合成了一条路径，也就相当于路径数减少了 1。所以找到了几条匹配边，路径数就减少了多少。所以有最小路径覆盖=原图的结点数-新图的最大匹配数。

因为路径之间不能有公共点，所以加的边之间也不能有公共点，这就是匹配的定义。

说的拆点这么高深，其实操作起来超级超级简单，甚至没有操作。

简单的来说，每个顶点都能当成二分图中作为起点的顶点。

DAG 的最小可相交路径覆盖

算法：先用 floyd 求出原图的传递闭包，即如果 a 到 b 有路径，那么就加边 a->b。然后就转化成了最小不相交路径覆盖问题。

证明：为了连通两个点，某条路径可能经过其它路径的中间点。比如 1->3->4，2->4->5。但是如果两个点 a 和 b 是连通的，只不过中间需要经过其它的点，那么可以在这两个点之间加边，那么 a 就可以直达 b，不必经过中点的，那么就转化成了最小不相交路径覆盖。

5.KM

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int INF=0x3f3f3f3f;
const int MOD=1e9+7;
const int MAXN=505;
int adj[MAXN][MAXN],n;
int lx[MAXN],ly[MAXN],link[MAXN];
bool visx[MAXN],visy[MAXN];
bool dfs(int x) { // 匈牙利算法部分
    visx[x]=1;
    for(int i=1; i<=n; i++) {
        if(adj[x][i]==lx[x]+ly[i]&&visy[i]==0) {
            visy[i]=1;
            if(link[i]==-1||dfs(link[i])) {
                link[i]=x;
```

```
                return true;
            }
        }
    }
    return false;
}
int KM() {
    memset(link,-1,sizeof(link));
    for(int i=1; i<=n; i++) {
        lx[i]=-INF,ly[i]=0;
        for(int j=1; j<=n; j++) lx[i]=max(lx[i],adj[i][j]);
    }
    for(int k=1; k<=n; k++) { //匈牙利算法
        while(1)//对于每个左图的点,进行不断的查找最大权边,一旦发现
有
            // 某一个左图点不能匹配,要求下降一点;继续
        {
            memset(visy,0,sizeof visy);
            memset(visx,0,sizeof visx);
            if(dfs(k)) break;
            int minval=INF;
            for(int i=1; i<=n; i++) if(visx[i]) for(int j=1; j<=n;
j++) if(!visy[j]) minval=min(minval,lx[i]+ly[j]-adj[i][j]);
            if(minval==INF) return -1;
            for(int i=1; i<=n; i++) if(visx[i]) lx[i]-=minval;
            for(int i=1; i<=n; i++) if(visy[i]) ly[i]+=minval;
        }
    }
    int res=0;
    for(int i=1; i<=n; i++) {
        if(link[i]!=-1) res+=adj[link[i]][i];
    }
    return res;
}

int main() {
    while(~scanf("%d",&n)) {
        memset(adj,-1,sizeof(adj));
        for(int i=1; i<=n; i++)
            for(int j=1; j<=n; j++)
                scanf("%d",&adj[i][j]);
        printf("%d\n",KM());
    }
    return 0;
```

```
}
```

6.HK

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=3005;
const int INF=3e7;
vector<int>G[maxn];
vector<int>::iterator it;
int matched[maxn],n,mp[maxn][maxn],n1,n2,m,k,x[maxn],y[maxn],T;
struct node {
    int x,y,v;
} a[maxn],b[maxn];
int um[maxn],vm[maxn];
int dx[maxn],dy[maxn];
int dis;
bool vis[maxn];//建立匹配关系
//bfs 寻找增广路径
bool SearchP(int n) {
    queue<int>q;
    dis = INF;
    memset(dx,-1,sizeof(dx));
    memset(dy,-1,sizeof(dy));
    for(int i=1; i<=n; ++i) {
        //u 中未匹配
        if(um[i]==-1) {
            q.push(i);
            dx[i] = 0;
        }
    }
    while(!q.empty()) {
        int u = q.front();
        q.pop();
        if(dx[u]>dis)
            break; //说明该增广路径长度大于 dis 还没有结束，等待下一
次 BFS 在扩充
        for(int i=0; i<(int)G[u].size(); ++i) {
            int v = G[u][i];
            if(dy[v]==-1) { //v 中尚未匹配的点
                dy[v] = dx[u] + 1;
                //对点进行分层（不标记是匹配点和未匹配点）
                if(vm[v]==-1) {
                    //得到本次 BFS 的最大遍历层次
```

```
                    dis = dy[v];
                } else {
                    dx[vm[v]] = dy[v] + 1;//v 是匹配点，继续延伸
                    q.push(vm[v]);
                }
            }
        }
    }
    return (dis!=INF);
}

bool dfs(int u) {
    for(int i=0; i<G[u].size(); ++i) {
        int v = G[u][i];
        if(!vis[v]&&dy[v]==dx[u]+1) {
            vis[v] = 1;
            if(vm[v]!=-1&&dy[v]==dis)
                continue;
            //层次(也就是增广路径的长度)大于本次查找的 dis,是 searchP
被 break 的情况，
            //也就是还不确定是否是增广路径，只有等再次调用 searchP()
在判断。
            if(vm[v]==-1||dfs(vm[v])) {
                vm[v] = u;
                um[u] = v;
                return true;
            }
        }
    }
    return false;
}

int Maxmatch() {
    int res = 0;
    memset(um,-1,sizeof(um));
    memset(vm,-1,sizeof(vm));
    while(SearchP(n)) {
        memset(vis,0,sizeof(vis));
        for(int i=1; i<=n; ++i) {
            //查找到一个增广路径，匹配数++
            if(um[i]==-1&&dfs(i)) {
                ++res;
            }
        }
```

```
    }
    return res;
}
int check(int i,int j,int T) {
    int
dis=(a[i].x-b[j].x)*(a[i].x-b[j].x)+(a[i].y-b[j].y)*(a[i].y-b[
j].y);
    int d=T*a[i].v;
    if(d*d>=dis) {
        return 1;
    }
    return 0;
}
void make_edge() {
    for(int i=1; i<=n; i++) {
        for(int j=1; j<=m; j++) {
            if(check(i,j,T))
                G[i].push_back(j);
        }
    }
}
int main() {
    int t,o=1;
    cin>>t;
    while(t--) {
        scanf("%d",&T);
        scanf("%d",&n);
        for(int i=1; i<=n; i++)
            scanf("%d%d%d",&a[i].x,&a[i].y,&a[i].v);
        scanf("%d",&m);
        for(int i=1; i<=m; i++)
            scanf("%d%d",&b[i].x,&b[i].y);
        make_edge();
        printf("Scenario #%d:\n%d\n\n",o++,Maxmatch());
        for(int i=1; i<=n; i++)
            G[i].clear();
    }
}
```

7.匈牙利最大匹配

```
int Find(int u) {
    int len=G[u].size();
    for(int i=0; i<len; i++) {
```

```
        int v=G[u][i];
        if(!vis[v]) {
            vis[v]=1;
            if(!matched[v]||Find(matched[v])) {
                matched[v]=u;
                return 1;
            }
        }
    }
    return 0;
}
int Maxmatch() {
    memset(matched,0,sizeof matched);
    int cnt=0;
    for(int i=1; i<=n2; i++) {
        memset(vis,0,sizeof vis);///可以直接时间戳优化
        if(Find(i))
            cnt++;
    }
    return cnt;
}
```

8.2-SAT

输出解集的时候，我们要注意，tarjan 给连通分量标号的拓扑序是反序，拓扑序越大，扑序越大的点其所在强联通分量编号越小，我们选择编号小的就行了。

1.Top 序任意解

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=2e6+6;
const int mod=1e9+7;
vector<int>G[N];
stack<int>s;
int n,m,x,y,a,b,tot,colnum;
int dfn[N],col[N],vis[N],low[N];
void tarjan(int u) {
    dfn[u]=low[u]=++tot;
    vis[u]=1,s.push(u);
    for(auto v:G[u]) {
        if(!dfn[v])
```

```
            tarjan(v),low[u]=min(low[u],low[v]);
        else if(vis[v])
            low[u]=min(low[u],dfn[v]);
    }
    if(low[u]==dfn[u]) {
        ++colnum;
        while(true) {
            int v=s.top();
            s.pop();
            vis[v]=0,col[v]=colnum;
            if(u==v)break;
        }
    }
}
void add(int u,int v) {
    G[u].push_back(v);
}
int main() {
    scanf("%d%d",&n,&m);
    for(int i=1; i<=m; i++) {
        scanf("%d%d%d%d",&a,&x,&b,&y);
        if(x==0&&y==0) {
            add(a+n,b);
            add(b+n,a);
        } else if(x==0&&y==1) {
            add(b,a);
            add(a+n,b+n);
        } else if(x==1&&y==0) {
            add(b+n,a+n);
            add(a,b);
        } else if(x==1&&y==1) {
            add(b,a+n);
            add(a,b+n);
        }
    }
    for(int i=1; i<=n*2; i++) {
        if(!dfn[i])tarjan(i);
    }
    for(int i=1; i<=n; i++) {
        if(col[i]==col[i+n]) {
            printf("IMPOSSIBLE\n");
            return 0;
        }
    }
```

```
    printf("POSSIBLE\n");
    for(int i=1; i<=n; i++) {
        if(col[i]<col[i+n])printf("0 ");
        else printf("1 ");
    }
    printf("\n");
}
```

2. 字典序最小解

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 8006;
int n, m, stak[maxn], top, col[maxn<<1];
vector<int> G[maxn<<1];
int mir(int x) {
    return x&1? x+1: x-1;
}
bool paint(int x) {
    if(col[x])
        return col[x] == 1;
    col[x] = 1;
    col[mir(x)] = 2;
    stak[++top] = x;
    for(unsigned int i = 0; i < G[x].size(); ++i) {
        int id = G[x][i];
        if(!paint(id))
            return 0;
    }
    return 1;
}

bool work() {
    for(int i = 1; i <= (n<<1); ++i)
        if(!col[i]) {
            top = 0;
            if(!paint(i)) {
                while(top)  col[stak[top]] = col[mir(stak[top])] =
0, top--;
                if(!paint(mir(i)))
                    return 0;
            }
        }
    return 1;
```

```
}

int main() {
    while(scanf("%d%d", &n, &m) != EOF) {
        for(int i = 1; i <= m; ++i) {
            int x, y;
            scanf("%d%d", &x, &y);
            G[x].push_back(mir(y));
            G[y].push_back(mir(x));
        }
        if(work()) {
            for(int i = 1; i <= (n<<1); ++i)
                if(col[i] == 1)
                    printf("%d\n", i);
        } else
            printf("NIE\n");
        for(int i = 1; i <= (n<<1); ++i)
            G[i].clear(), col[i] = 0;
    }
}
```

9.第 K 短路

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>
using namespace std;
const int maxn = 5010;
const int maxm = 400010;
const int inf = 2e9;
int n, m, s, t, k, u, v, ww, H[maxn], cnt[maxn];
int cur, h[maxn], nxt[maxm], p[maxm], w[maxm];
int cur1, h1[maxn], nxt1[maxm], p1[maxm], w1[maxm];
bool tf[maxn];
void add_edge(int x, int y, double z) {
    cur++;
    nxt[cur] = h[x];
    h[x] = cur;
    p[cur] = y;
    w[cur] = z;
}
void add_edge1(int x, int y, double z) {
    cur1++;
```

```
    nxt1[cur1] = h1[x];
    h1[x] = cur1;
    p1[cur1] = y;
    w1[cur1] = z;
}
struct node {
    int x, v;
    bool operator<(node a) const {
        return v + H[x] > a.v + H[a.x];
    }
};
priority_queue<node> q;
struct node2 {
    int x, v;
    bool operator<(node2 a) const {
        return v > a.v;
    }
} x;
priority_queue<node2> Q;
int main() {
    scanf("%d%d%d%d%d", &n, &m, &s, &t, &k);
    while(m--) {
        scanf("%d%d%d", &u, &v, &ww);
        add_edge(u, v, ww);
        add_edge1(v, u, ww);
    }
    for(int i = 1; i <= n; i++) H[i] = inf;
    Q.push({t, 0});
    while(!Q.empty()) {
        x = Q.top();
        Q.pop();
        if(tf[x.x]) continue;
        tf[x.x] = true;
        H[x.x] = x.v;
        for(int j = h1[x.x]; j; j = nxt1[j]) Q.push({p1[j], x.v +
w1[j]});
    }
    q.push({s, 0});
    while(!q.empty()) {
        node x = q.top();
        q.pop();
        cnt[x.x]++;
        if(x.x == t && cnt[x.x] == k) {
            printf("%d\n", x.v);
```

```
            return 0;
        }
        if(cnt[x.x] > k) continue;
        for(int j = h[x.x]; j; j = nxt[j]) q.push({p[j], x.v +
w[j]});
    }
    printf("-1\n");
    return 0;
}
```

10. LCA

```
namespace lca {
const int MAXN=1e5+5;
const int DEG=20;
struct Edge {
    int v,nxt;
} E[MAXN*2];
int head[MAXN],tot;
void addedge(int u,int v) {
    E[tot].v=v;
    E[tot].nxt=head[u];
    head[u]=tot++;
}
void init() {
    tot=0;
    memset(head,-1,sizeof(head));
}
int fa[MAXN][DEG];//fa[i][j]表示结点 i 的第 2^j 个祖先
int deg[MAXN];//深度数组
void BFS(int root) {
    queue<int> Q;
    deg[root]=0;
    fa[root][0]=root;
    Q.push(root);
    while(!Q.empty()) {
        int tmp=Q.front();
        Q.pop();
        for(int i=1; i<DEG; i++)
            fa[tmp][i]=fa[fa[tmp][i-1]][i-1];
        for(int i=head[tmp]; ~i; i=E[i].nxt) {
            int v=E[i].v;
            if(v==fa[tmp][0])continue;
            deg[v]=deg[tmp]+1;
```

```
            fa[v][0]=tmp;
            Q.push(v);
        }
    }
}
int LCA(int u,int v) {
    if(deg[u]>deg[v])swap(u,v);
    int hu=deg[u],hv=deg[v];
    int tu=u,tv=v;
    for(int det=hv-hu,i=0; det; det>>=1,i++)
        if(det&1)
            tv=fa[tv][i];
    if(tu==tv)return tu;
    for(int i=DEG-1; i>=0; i--) {
        if(fa[tu][i]==fa[tv][i])
            continue;
        tu=fa[tu][i];
        tv=fa[tv][i];
    }
    return fa[tu][0];
}
}
```

2. 字符串

1.后缀数组

```
///后缀数组
///https://xminh.github.io/2018/02/27/%E5%90%8E%E7%BC%80%E6%95%B0%E7%BB%84-%E6%9C%80%E8%AF%A6%E7%BB%86(maybe)%E8%AE%B2%E8%A7%A3.html

#include<iostream>
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<set>
#include<ctime>
#include<vector>
#include<cmath>
#include<algorithm>
#include<map>
```

```
using namespace std;
typedef long long ll;
const int N=1e6+5;
int n,p,q,k,m;
int v[N],height[N];
int sa[2][N],rk[2][N];
char a[N];
///sa[p][i]表示排名为 i 的起始下标
///rk[p][i]表示起始下标为 i 的排名
///LCP(i,j)=suff(sa[i]),suff(sa[j]);
///height[i]表示 LCP(i,i-1);
struct SA {
    void calsa(int sa[N],int rk[N],int SA[N],int RK[N]) {
        for(int i=1; i<=n; i++)v[rk[sa[i]]]=i;
        for(int i=n; i; i--)
            if(sa[i]>k)
                SA[v[rk[sa[i]-k]]--]=sa[i]-k;
        for(int i=n-k+1; i<=n; i++)SA[v[rk[i]]--]=i;
        for(int i=1; i<=n; i++)

RK[SA[i]]=RK[SA[i-1]]+(rk[SA[i-1]]!=rk[SA[i]]||rk[SA[i]+k]!=rk
[SA[i-1]+k]);
    }
    void getsa() {
        memset(v,0,sizeof(v));
        memset(rk,0,sizeof(rk));
        p=0,q=1;
        for(int i=1; i<=n; i++)v[a[i]]++;
        for(int i=1; i<=m; i++)v[i]+=v[i-1];
        for(int i=1; i<=n; i++)
            sa[p][v[a[i]]--]=i;
        for(int i=1; i<=n; i++)

rk[p][sa[p][i]]=rk[p][sa[p][i-1]]+(a[sa[p][i-1]]!=a[sa[p][i]])
;
        for(k=1; k<n; k<<=1,swap(p,q))
            calsa(sa[p],rk[p],sa[q],rk[q]);
    }
    void geth() {
        k=0;
        for(int i=1; i<=n; i++)
            if(rk[p][i]==1)height[rk[p][i]]=0;
            else {
                int j=sa[p][rk[p][i]-1];
```

```
                while(a[i+k]==a[j+k])k++;
                height[rk[p][i]]=k;
                if(k>0)k--;
            }
    }
    void init() {
        m=305;
        getsa();
        geth();
    }
} SA;
int main() {
    cin>>n;
    scanf("%s",a+1);
    n=strlen(a+1);
    SA.init();
    ll ans=0;
    for(int i=1; i<=n; i++)
        printf("%d ",height[i]);
    printf("\n");
    for(int i=1; i<=n; i++)
        printf("%d ",sa[p][i]);
    printf("\n");
    for(int i=1; i<=n; i++)
        printf("%d ",rk[p][i]);
    printf("\n");
}
```

2.给定 n 个值域在 [0,1000000] 的整数，

请求出最长的出现了至少 k 次的子串。

solution1:RMQ+SA

子串=后缀的前缀.

首先考虑：最长的子串的子串一定也出现了至少 K 次

那么我们就可以二分 len,

每次检查长度=len 的子串是否出现了 K 次。

height[i]=lcp(i，i-1）那么 height[i]>=len 不就

代表后缀的前缀出现了两次吗?

首先我们幻想一下 height[i]数组的大小趋势大致是这个样子

大中小大小大中小,分段有序

那么我们要查询相邻的 K 段最小的那个 height[i]取最大值就是答案

solution2:二分+SA

基于上一个 solution,我们考虑这个 check 函数怎么写，
我们记录一段区间> =len 的数量，大于 k-1 就行。

```
bool judge(int len) {
    int cnt=0;
    for(int i=1; i<=n; i++) {
        if(height[i]<len)
            cnt=0;
        else cnt++;
        if(cnt>=K-1)return 1;
    }
    return 0;
}
void solve1() {
    int l=1,r=n,ans=0;
    while(l<=r) {
        int mid=(l+r)/2;
        if(judge(mid))
            ans=mid,l=mid+1;
        else
            r=mid-1;
    }
    cout<<ans<<endl;
}
void init() {
    for(int i=1; i<=n; i++) {
        dp[i][0]=height[i];
    }
    int up=log2(n);
    for(int j=1; j<=up; j++) {
        for(int i=1; i+(1<<j)-1<=n; i++) {
            dp[i][j]=min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
        }
    }
}
int query(int x,int y) {
    int k=log2(y-x+1);
    return min(dp[x][k],dp[y-(1<<k)+1][k]);
}
void solve2() {
    init();
    int ans=0;
    for(int i=1; i<=n; i++) {
```

```
        ans=max(ans,query(i,i+K-1-1));
    }
    cout<<ans<<endl;
}
int main() {
    cin>>n>>K;
    for(int i=1; i<=n; i++)
        scanf("%d",&a[i]),d[i]=a[i];
    sort(d+1,d+n+1);
    int res=unique(d+1,d+n+1)-d-1;
    for(int i=1; i<=n; i++)
        a[i]=lower_bound(d+1,d+res+1,a[i])-d;
    SA.init();
    solve1();
    solve2();
}
```

2.kmp:

1.next 数组：子串的前缀和后缀最大长度，也就是子串后缀与子串的最大匹配长度。
2.kuangbin 模板上的 kmp_pre 求 next 数组优化版，部分 next[i]=-1,可能会在有些题目上出错，慎用.
3.周期串： （i%(i-next[i])==0）最小周期(i-next[i]),循环节（i/(i-next[i])）；
4.前缀在原串中出现的次数 dp[i]=dp[NEXT[i]]+1;第 i 个字符结尾的前缀数等于以第 next[i]个字符为结尾的前缀数加上它自己本身。
5.fill(cnt,cnt+m+1,1); for(int i=m; i>=1; i--) cnt[nt[i]]+=cnt[i];

3.e-kmp:

1.extend[i],S[i-len]与 T 的前缀的最大匹配长度，next[i],T[i-len]与 T 的前缀的最大匹配长度，
i+extend[i]==len 表示 S 在 i 之后的后缀与 T 的前缀相匹配
2.上面那个 dp 方程可以直接用自身匹配的 next 求和
3.判断前缀或者后缀是回文串：后缀：a->reverse->b，求 a 当作母串的 extend 数组 i+extend[i]==len 就是回文串。前缀：reverse 之后 b 当作母串求 extend2 同样判断就行 .翻转||往后接一个相同的串。

4.最小表示法

字符串最小表示法：

```
int getmin(char *s)
```

```
{
    int n=cnt;
    int i=0,j=1,k=0,t;
    while(i<n && j<n && k<n)
    {
        t=s[(i+k)%n]-s[(j+k)%n];
        if (!t) k++;
        else
        {
            if (t>0) i+=k+1;
            else j+=k+1;
            if (i==j) j++;
            k=0;
        }
    }
    return i<j?i:j;
}
int getmax(char *s)
{
    int n=cnt;
    int i=0,j=1,k=0,t;
    while(i<n && j<n && k<n)
    {
        t=s[(i+k)%n]-s[(j+k)%n];
        if (!t) k++;
        else
        {
            if (t>0) j+=k+1;
            else i+=k+1;
            if (i==j) j++;
            k=0;
        }
    }
    return i<j?i:j;
}
```

5.字典树

```
struct trie {
  int nex[100000][26], cnt;
  bool exist[100000];  // 该结点结尾的字符串是否存在

  void insert(char *s, int l) {  // 插入字符串
    int p = 0;
```

```
  for (int i = 0; i < l; i++) {
    int c = s[i] - 'a';
    if (!nex[p][c]) nex[p][c] = ++cnt;  // 如果没有，就添加结点
    p = nex[p][c];
  }
  exist[p] = 1;
}
bool find(char *s, int l) {  // 查找字符串
  int p = 0;
  for (int i = 0; i < l; i++) {
    int c = s[i] - 'a';
    if (!nex[p][c]) return 0;
    p = nex[p][c];
  }
  return exist[p];
}};
```

3. 博弈

约瑟夫问题：

问题描述

> n 个人标号 $0, 1, \cdots, n-1$ 。逆时针站一圈，从 0 号开始，每一次从当前的人逆时针数 k 个，然后让这个人出局。问最后剩下的人是谁。

简单优化

寻找下一个人的过程可以用线段树优化。具体地，开一个 $0, 1, \cdots, n-1$ 的线段树，然后记录区间内剩下的人的个数。寻找当前的人的位置以及之后的第 k 个人可以在线段树上二分做。

线性算法

设 $J_{n,k}$ 表示规模分别为 n, k 的约瑟夫问题的答案。我们有如下递归式

$$J_{n,k} = (J_{n-1,k} + k) \bmod n$$

这个也很好推。你从 0 开始数 k 个，让第 $k-1$ 个人出局后剩下 $n-1$ 个人，你计算出在 $n-1$ 个人中选的答案后，再加一个相对位移 k 得到真正的答案。这个算法的复杂度显然是 $\Theta(n)$ 的。

```
int josephus(int n, int k) {
  int res = 0;
  for (int i = 1; i <= n; ++i) res = (res + k) % i;
  return res;
}
```

对数算法

对于 k 较小 n 较大的情况，本题还有一种复杂度为 $\Theta(k \log n)$ 的算法。

考虑到我们每次走 k 个删一个，那么在一圈以内我们可以删掉 $\left\lfloor \frac{n}{k} \right\rfloor$ 个，然后剩下了 $n - \left\lfloor \frac{n}{k} \right\rfloor$ 个人。这时我们在第 $\left\lfloor \frac{n}{k} \right\rfloor \cdot k$ 个人的位置上。而你发现这个东西它等于 $n - n \bmod k$。于是我们继续递归处理，算完后还原它的相对位置。得到如下的算法：

```
int josephus(int n, int k) {
  if (n == 1) return 0;
  if (k == 1) return n - 1;
  if (k > n) return (josephus(n - 1, k) + k) % n;  // 线性算法
  int res = josephus(n - n / k, k);
  res -= n % k;
  if (res < 0)
    res += n;  // mod n
  else
    res += res / (k - 1);  // 还原位置
  return res;
}
```

可以证明这个算法的复杂度是 $\Theta(k \log n)$ 的。我们设这个过程的递归次数是 x，那么每一次问

博弈

1．巴什博奕：只有一堆 n 个物品，两个人轮流从中取物，规定每次最少取一个，最多取 m 个，最后取光者为胜。

2.威佐夫博弈（Wythoff Game）：
有两堆各若干的物品，两人轮流从其中一堆取至少一件物品，至多不限，或从两堆中同时取相同件物品，规定最后取完者胜利。
直接说结论了，若两堆物品的初始值为（x，y），且 x<y，则另 z=y-x；
记 w=（int）[（（sqrt（5）+1）/2）*z]；
若 w=x，则先手必败，否则先手必胜。

3.尼姆博弈指的是这样一个博弈游戏：有任意堆物品，每堆物品的个数是任意的，双方轮流从中取物品，每一次只能从一堆物品中取部分或全部物品，最少取一件，取到最后一件物品的人获胜。
结论就是：把每堆物品数全部异或起来，如果得到的值为 0，那么先手必败，否则先手必胜。

4.斐波那契博弈：
有一堆物品，两人轮流取物品，先手最少取一个，至多无上限，但不能把物品取完，之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件，取走最后一件物品的人获胜。
结论是：先手胜当且仅当 n 不是斐波那契数（n 为物品总数）

4. 计算几何

1.判断圆与矩形相交

```
ll pow2(int x)
{
    return 1ll*x*x;
}
bool up(int i)
{
    if(op[i]==1)
        return b[i]+c[i]>=y;
    return d[i]>=y;
}
bool down(int i)
{
    if(op[i]==1)
```

```
        return b[i]-c[i]<=0;
    return b[i]<=0;
}
bool inarc(int i,int j)///j 是圆  角在圆内
{
    return pow2(a[i]-a[j])+pow2(b[i]-b[j])<=pow2(c[j])||
           pow2(a[i]-a[j])+pow2(d[i]-b[j])<=pow2(c[j])||
           pow2(c[i]-a[j])+pow2(b[i]-b[j])<=pow2(c[j])||
           pow2(c[i]-a[j])+pow2(d[i]-b[j])<=pow2(c[j]);

}
bool in(int i,int j)///j 是圆   圆心在矩形内
{
    return (a[j]>=a[i]&&a[j]<=c[i])&&(b[j]>=b[i]&&b[j]<=d[i]);
}
bool line1(int i,int j)///j 是圆 矩形左或者右边在圆内
{
    if(a[j]>=a[i]&&a[j]<=c[i])
        return 0;
    if(a[j]<a[i]&&(b[j]>=b[i]&&b[j]<=d[i]))
        return a[i]-a[j]<=c[j];
    if(a[j]>c[i]&&(b[j]>=b[i]&&b[j]<=d[i]))
        return a[j]-c[i]<=c[j];
    return 0;
}
bool line2(int i,int j)///j 是圆  矩形上或者下边在圆内
{
    if(b[j]>=b[i]&&b[j]<=d[i])
        return 0;
    if(b[j]<b[i]&&(a[j]>=a[i]&&a[j]<=c[i]))
        return b[i]-b[j]<=c[j];
    if(b[j]>d[i]&&(a[j]>=a[i]&&a[j]<=c[i]))
        return b[j]-d[i]<=c[j];
    return 0;
}


bool cross(int i,int j)
{
    if(op[i]==1&&op[j]==1)///两圆
        return pow2(a[i]-a[j])+pow2(b[i]-b[j])<=pow2(c[i]+c[j]);
    if(op[i]==2&&op[j]==2)///两矩形
        return
min(c[i],c[j])>=max(a[i],a[j])&&min(d[i],d[j])>=max(b[i],b[j])
```

```
;
    if(op[i]==1)
        swap(i,j);
    ///i 是矩形，j 是圆，相交
    ///1.圆心在矩形里面
    ///2.矩形一个角在园里面
    ///3.矩形的一个边（左右上下）在园里面
    //return line1(i,j)||line2(i,j)||point(i,j);
    return in(i,j)||inarc(i,j)||line1(i,j)||line2(i,j);
}
```

叉乘三角形面积

```
//(x0,y0), (x1,y1), (x2, y2)是逆时针返回值为正，顺时针返回值为负

double area(double x0, double y0, double x1,double y1, double
x2,double y2){
   return (x0*y1 + x2*y0 + x1*y2 - x2*y1 - x0*y2 - x1*y0 ) / 2;

}
```

JAVA 小数高精度四舍五入

可以看到舍入模式有很多种BigDecimal.ROUND_XXXX_XXX, 具体都是什么意思呢

```
BigDecimal result51 = num22.divide(num12, scale: 3,BigDecimal.ROUND_UP);
BigDecimal result52 = num22.divide(num12, scale: 3,BigDecimal.ROUND_DOWN);
BigDecimal result53 = num22.divide(num12, scale: 3,BigDecimal.ROUND_CEILING);
BigDecimal result54 = num22.divide(num12, scale: 3,BigDecimal.ROUND_FLOOR);
BigDecimal result55 = num22.divide(num12, scale: 3,BigDecimal.ROUND_HALF_UP);
BigDecimal result56 = num22.divide(num12, scale: 3,BigDecimal.ROUND_HALF_DOWN);
BigDecimal result57 = num22.divide(num12, scale: 3,BigDecimal.ROUND_HALF_EVEN);
BigDecimal result58 = num22.divide(num12, scale: 2,BigDecimal.ROUND_UNNECESSARY);
```

计算1÷3的结果（最后一种ROUND_UNNECESSARY在结果为无限小数的情况下会报错）

```
除法ROUND_UP：0.334
除法ROUND_DOWN：0.333
除法ROUND_CEILING：0.334
除法ROUND_FLOOR：0.333
除法ROUND_HALF_UP：0.333
除法ROUND_HALF_DOWN：0.333
除法ROUND_HALF_EVEN：0.333

Process finished with exit code 0
```

快速乘法 O1

```
inline ll ksc(ll x,ll y,ll mod)
{
    return (x*y-(ll)((long double)x/mod*y)*mod+mod)%mod;
}
```