T1chess

40分：由于每个位置向右向下字符都不同，为了使得答案最小，贪心地向更小的方向走即可。

100分：某一步有可能两个方向一样，此时我们用set或者queue维护每一步的所有可能的最小下标即可。复杂度$O(n*m)$或者$O(nmlogn)$。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int read()
{
    int x;scanf("%d",&x);return x;
}
int n,m,x,y;
char s[2010][2010],minn[2010];
set<pair<int,int>>o[5010];
int main()
{
    freopen("a.in","r",stdin);
    freopen("a.out","w",stdout);
    n=read();m=read();
    for(int i=1;i<=n;i++)
        scanf("%s",s[i]+1);
    o[1].insert({1,1});
    minn[0]=s[1][1];
    for(int i=1;i+2<=n+m;i++)
    {
        minn[i]='z';
        for(auto j:o[i])
        {
            x=j.first;
            y=j.second;
            if(x+1<=n)
                minn[i]=min(minn[i],s[x+1][y]);
            if(y+1<=m)
                minn[i]=min(minn[i],s[x][y+1]);
        }
        for(auto j:o[i])
        {
            x=j.first;
            y=j.second;
            if(x+1<=n&&s[x+1][y]==minn[i])
                o[i+1].insert({x+1,y});
            if(y+1<=m&&s[x][y+1]==minn[i])
                o[i+1].insert({x,y+1});
        }
    }
    for(int i=0;i+2<=n+m;i++)
        cout<<minn[i];
}
```

T2glass

注意到最优解可以永远向已经空了的杯子倒水，只从有水的向有水的杯子倒。因为我们可以在空杯子变空的那次倒水之前，把这次倒水插进去。

并且倒水建图来看一定是有向无环图，因为如果有环的话一定可以断开。

于是我们状压dp，$f[S]$表示当前有水状态为S时的最小花费。则枚举这个状态中有水的杯子向另一个有水的杯子里倒水进行更新即可。复杂度小于$O(2^N * N^2)$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int read()
{
    int x;scanf("%d",&x);return x;
}
int n,k,c[30][30];
int ans,cnt[2000010],f[2000010];
int main()
{
    freopen("b.in","r",stdin);
    freopen("b.out","w",stdout);
    n=read();k=read();
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            c[i][j]=read();
    memset(f,0x3f,sizeof(f));
    ans=1e9;
    for(int i=1;i<(1<<n);i++)
        cnt[i]=cnt[i/2]+(i&1);
    f[(1<<n)-1]=0;
    for(int i=(1<<n)-1;i;i--)
    {
        if(cnt[i]<=k)
            continue;
        for(int x=0;x<n;x++)
        {
            if((i&(1<<x))==0)
                continue;
            for(int y=0;y<n;y++)
            {
                if(x==y||(i&(1<<y))==0)
                    continue;
                f[i^(1<<x)]=min(f[i^(1<<x)],f[i]+c[x][y]);
            }
        }
    }
    for(int i=0;i<(1<<n);i++)
        if(cnt[i]==k)
            ans=min(ans,f[i]);
    cout<<ans;
}
```

T3card

为了获得严格递增子序列，我们需要将原串分成严格递减子序列和严格递增子序列，把他们连起来。

首先从后往前，$nlogn$地跑位置i向后能构成的最长严格递增子序列长度$f[i]$，最长严格递减子序列长度$g[i]$。

考虑枚举数字i是分界点，前面的不要。则能形成最长长度是$f[i]+g[i]-1$，最长长度即为$ans=max(f[i]+g[i]-1)$。

接下来需要计算方案数。考虑仍然枚举i，如果最长长度$ans=(f[i]+g[i]-1)$则他对答案的贡献是$2^{n-ans}*i$向后长为$f[i]$的最长严格递增子序列方案数$*i$向后长为$g[i]$的最长严格递减子序列方案数。可以使用树状数组来解决。

以向后最长严格递减子序列方案数距离，实际上是在求递增子序列方案数

我们需要知道在i位置之前，小于a_i的元素中的最大长度g_j。

该最大长度所对应的方案数之和$\sum cnt$

则长度为g_j+1，方案数为cnt。

可以使用线段树，维护l,r间最大值和最大值出现次数cnt

则合并时：

```
maxg[x]=max(maxg[x*2],maxg[x*2+1]);
cnt[x]=0;
if(maxg[x*2]==maxg[x])
    cnt[x]=cnt[x*2];
if(maxg[x*2+1]==maxg[x])
    cnt[x]=(cnt[x]+cnt[x*2+1])%mod;
```

进行线段树单点修改和区间询问即可。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int read()
{
    int x;scanf("%d",&x);return x;
}
int n,m,a[200010],b[200010],v[200010],len,f[200010],g[200010],cntf[200010],cntg[200010];
int ans,cnt,mod=1e9+7;
pair<int,int>G[1600010];
ll quick(ll x,ll y)
{
    ll t=1;
    while(y)
        t=t*2%mod,y--;
    return t;
}
pair<int,int>merge(pair<int,int> a,pair<int,int> b)
{
    if(b.first==a.first)
```

```
        a.second=(a.second+b.second)%mod;
    if(b.first>a.first)
        a=b;
    return a;
}
pair<int,int> askg(int x,int l,int r,int tl,int tr)
{
    if(tl<=l&&r<=tr)
        return G[x];
    int mid=(l+r)/2;
    if(tr<=mid)
        return askg(x*2,l,mid,tl,tr);
    if(tl>mid)
        return askg(x*2+1,mid+1,r,tl,tr);
    return merge(askg(x*2,l,mid,tl,tr),askg(x*2+1,mid+1,r,tl,tr));
}
void add(int x,int l,int r,int d,pair<int,int> v)
{
    if(l==r)
    {
        G[x]=merge(G[x],v);
        return ;
    }
    int mid=(l+r)/2;
    if(d<=mid)
        add(x*2,l,mid,d,v);
    else
        add(x*2+1,mid+1,r,d,v);
    G[x]=merge(G[x*2],G[x*2+1]);
}

int main()
{
    freopen("c.in","r",stdin);
    freopen("c.out","w",stdout);
    n=read();
    for(int i=1;i<=n;i++)
        b[i]=a[i]=read();
    sort(b+1,b+1+n);
    m=unique(b+1,b+1+n)-b-1;
    m++;
    b[m]=2e9;
    for(int i=1;i<=n;i++)
        a[i]=lower_bound(b+1,b+1+m,a[i])-b;
    add(1,0,m,0,{0,1});
    for(int i=n;i>=1;i--)
    {
        pair<int,int>t=askg(1,0,m,0,a[i]-1);
        f[i]=t.first+1;
        cntf[i]=t.second;
        add(1,0,m,a[i],{f[i],cntf[i]});
    }
    memset(G,0,sizeof(G));
    add(1,0,m,m,{0,1});
    for(int i=n;i>=1;i--)
    {
        pair<int,int>t=askg(1,0,m,a[i]+1,m);
        g[i]=t.first+1;
```

```
            cntg[i]=t.second;
            add(1,0,m,a[i],{g[i],cntg[i]});

        }
        for(int i=1;i<=n;i++)
            ans=max(ans,f[i]+g[i]-1);
        ll tt=quick(2,n-ans);
        for(int i=1;i<=n;i++)
            if(f[i]+g[i]-1==ans)
                cnt=(cnt+tt*cntg[i]%mod*cntf[i]%mod)%mod;
        cout<<ans<<' '<<cnt;
    }

```

注意到这个过程只求前缀后缀，单点修改，可以使用树状数组来优化。

```
#include<bits/stdc++.h>
using namespace std;
#define mod 1000000007
typedef long long ll;
int n,A[200005],B[200005],pn;
int minn,maxx,len,ans;
ll pow2[200005],kdf;
struct node{
    int a;ll b;node(){a=b=0;}
}tree[2][200005];
node dp[200005],dp2[200005];
node add(node A,node B){
    if(A.a==B.a)A.b=(A.b+B.b)%mod;
    if(B.a>A.a)A=B;
    return A;
}
void Insert(int x,int kdf,node D){
    while(x<=pn&&x>0){
        tree[kdf][x]=add(tree[kdf][x],D);
        x=x+(kdf?-1:1)*(x&(-x));
    }
}
node Query(int x,int kdf){
    node res;
    while(x<=pn&&x>0){
        res=add(res,tree[kdf][x]);
        x=x+(kdf?1:-1)*(x&(-x));
    }
    return res;
}
int main()
{
    freopen("c.in","r",stdin);
    scanf("%d",&n);pn=n;
    for(int i=1;i<=n;i++)scanf("%d",&A[i]);
    for(int i=1;i<=n;i++)B[i]=A[i];
    sort(B+1,B+n+1);pn=unique(B+1,B+n+1)-B;
    for(int i=1;i<=n;i++)A[i]=lower_bound(B+1,B+pn,A[i])-B;
    pow2[0]=1;
    for(int i=1;i<=n;i++)pow2[i]=(pow2[i-1]*2ll)%mod;
    for(int i=n;i>=1;i--){
```

```
            node p=Query(A[i]+1,1),s;
            s.a=1,s.b=1;
            if(!p.a)dp[i]=s,Insert(A[i],1,s);
            else p.a++,dp[i]=p,Insert(A[i],1,p);
        }
        for(int i=n;i>=1;i--){
            node p=Query(A[i]-1,0),s;
            s.a=1,s.b=1;
            if(!p.a)dp2[i]=s,Insert(A[i],0,s);
            else p.a++,dp2[i]=p,Insert(A[i],0,p);
        }
        for(int i=1;i<=n;i++)ans=max(ans,dp[i].a+dp2[i].a-1);
        for(int i=1;i<=n;i++)
        if(dp[i].a+dp2[i].a-1==ans)kdf=(kdf+((1ll*dp[i].b*dp2[i].b)%mod)*pow2[n-ans]%mod)%mod;
        printf("%d %lld",ans,kdf);
    }
```

T4 godnumber

题意：求给定区间内包含给定字符串的数字之和。

看到数据规模，容易想到 ==数位DP==，并且涉及到 ==多模式串匹配==，我们可以使用 ==AC自动机==。

我们需要考虑 **如果若干位任选**，我们该怎么统计答案。 首先根据状态机类DP的套路，我们至少需要两个维度 $dp_{i,j}$ 表示当前位在 j 号节点，还能填 i 个数的答案。但是这样无法与我们已经有的状态建立联系。已经有的状态是指 **数位DP从前往后进行的过程中，会确定一些位上的数字，这些确定的数字构成了当前的状态。**对于本题而言，已经确定的数字会影响 那些字符串已经包含 和 当前在自动机的那个节点上。所以我们新加两维，设 $dp_{i,j,mask1,mask2}$ 表示当前在 j 号节点，还能填 i 个数字，当前的字符串匹配情况是 $mask1$，目标匹配情况是 $mask2$ 的所有填数方案中 **最后形成的数字之和。**

考虑转移：那么就是对于当前状态，我们枚举第一个数字填什么，设填了 c， 那么我们让节点跳到 $tr[j][c]$， 还能填的数字还有 $i-1$个，并且当前的状态为 $mask1 \mid e[tr[j][c]]$。 $e[tr[j][c]]$ 代表当前节点匹配的字符串的状态。

那么 $dp_{i,j,mask1,mask2} = \sum_{k=0}^{9} dp_{i-1,tr[j][k],mask1 \mid e[tr[j][k]],mask2} + num_{i-1,tr[j][k],mask1 \mid e[tr[j][k]],mask2} * 10^{i-1} * k$ 。

$num_{i,j,mask1,mask2}$ 表示当前节点位于 j 还能填 i 个数字，当前状态是 $mask1$，目标状态是 $mask2$ 的填数方案数， 那么 num 的转移与 dp 类似，也是枚举第一个填什么。

需要注意的是，我们要判断 $mask1 \mid e[tr[j][k]]$ 是否为 $mask2$ 的子集。

预处理出来 dp 数组后，进行数位DP就非常简单了，我们记录当前已经确定的数字的大小 lst， 和当前已经包含的字符串状态 $mask$，当前的节点 now，以及答案 res。每次确定一位后计算答案就行了。

需要注意的是输入会爆 $longlong$，我们直接输入字符串，然后求 $DP(r)-DP(l)$，然后暴力检验 l 是否为神数就好了。

CODE：

```
#include<bits/stdc++.h>
```

```cpp
// dp[i][j][mask1][mask2] 表示还能走i步，当前节点在j，状态由mask1 变成 mask2 的数值和  可以有前导0
#define N 105
#define LL long long
#define mod 998244353
using namespace std;// num[i][j][mask1][mask2] 表示还能走i步，当前节点在j，状态由mask1 变成 mask2 的方案数  可以有前导0
int n, tot, tr[N][10], e[N], fail[N], nn;
char str[N][N], l[N], r[N];
LL dp[N][N][1 << 5][1 << 5], num[N][N][1 << 5][1 << 5];
LL Pow(LL x, LL y){
    LL res = 1, k = x % mod;
    while(y){
        if(y & 1) res = (res * k) % mod;
        y >>= 1;
        k = (k * k) % mod;
    }
    return res % mod;
}
void ins(int id, char *str){
    int len = strlen(str + 1);
    int p = 0;
    for(int i = 1; i <= len; i++){
        if(!tr[p][str[i] - '0']) tr[p][str[i] - '0'] = ++tot;
        p = tr[p][str[i] - '0'];
    }
    e[p] |= (1 << (id - 1));
}
void build_ac(){
    queue< int > q;
    for(int i = 0; i < 10; i++)
       if(tr[0][i]) q.push(tr[0][i]);
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(int i = 0; i < 10; i++){
            if(tr[u][i]) fail[tr[u][i]] = tr[fail[u]][i], e[tr[u][i]] |= (e[fail[tr[u][i]]]), q.push(tr[u][i]);
            else tr[u][i] = tr[fail[u]][i];
        }
    }
}
void pre_work(){
    for(int i = 0; i <= tot; i++){
        for(int j = 0; j < (1 << nn); j++){
            num[0][i][j][j] = 1LL;
        }
    }
    for(int i = 1; i <= 100; i++){//步数为阶段
        for(int j = 0; j <= tot; j++){//枚举初始点
            for(int mask1 = 0; mask1 < (1 << nn); mask1++){
                for(int mask2 = mask1; mask2 < (1 << nn); mask2++){
                    if((mask2 & mask1) != mask1) continue;//结尾一定要包含自己
                    for(int c = 0; c < 10; c++){
                        int nxt = tr[j][c];
                        if((mask2 & e[nxt]) != e[nxt]) continue;//必须走包含的点
                        dp[i][j][mask1][mask2] = (((dp[i][j][mask1][mask2] + dp[i - 1][nxt][mask1 | e[nxt]][mask2])) % mod + (((num[i - 1][nxt][mask1 | e[nxt]][mask2] * Pow(10LL, 1LL * (i - 1))) % mod * (1LL * c) % mod))) % mod;
                        num[i][j][mask1][mask2] = (num[i][j][mask1][mask2] + num[i - 1][nxt]
```

```
[mask1 | e[nxt]][mask2]) % mod;
                    }
                }
            }
        }
    }
}
LL DP(char *str){
    vector< int > nums;
    int len = strlen(str + 1);
    for(int i = len; i >= 1; i--) nums.push_back(str[i] - '0');
    LL res = 0, lst = 0;
    int mask = 0, now = 0;
    for(int i = nums.size() - 1; i >= 0; i--){//规定第一位不能为0
        int x = nums[i];
        if(i == nums.size() - 1){//第一位
            for(int j = 1; j < x; j++){//填j
                int tc = tr[now][j], nmask = (mask | e[tc]);
                LL tnum = (lst * 10LL + j);
                res = (res + ((((tnum * Pow(10LL, 1LL * i)) % mod) * num[i][tc][nmask][(1 <<
nn) - 1]) % mod) + dp[i][tc][nmask][(1 << nn) - 1]) % mod;
            }
        }
        else{
            for(int j = 0; j < x; j++){//填j
                int tc = tr[now][j], nmask = (mask | e[tc]);
                LL tnum = (lst * 10LL + j);
                res = (res + ((((tnum * Pow(10LL, 1LL * i)) % mod) * num[i][tc][nmask][(1 <<
nn) - 1]) % mod) + dp[i][tc][nmask][(1 << nn) - 1]) % mod;
            }
        }
        now = tr[now][x];
        mask |= e[now];
        lst = (lst * 10LL + x) % mod;
        if(i == 0 && (mask == ((1 << nn) - 1))) res = (res + lst) % mod;
    }
    for(int i = nums.size() - 1; i >= 1; i--){//枚举最高位
        for(int j = 1; j <= 9; j++){
            int c = tr[0][j], mask = e[tr[0][j]];
            res = (res + ((((Pow(10LL, 1LL * (i - 1)) * (1LL * j)) % mod) * num[i - 1][c][mask]
[(1 << nn) - 1]) % mod) + dp[i - 1][c][mask][(1 << nn) - 1]) % mod;
        }
    }
    return res;
}
LL check(char *S){
    LL num = 0, f = 1; int len = strlen(S + 1);
    for(int i = 1; i <= len; i++) num = (num * 10LL + (S[i] - '0')) % mod;
    for(int i = 1; i <= nn; i++){
        int tl = strlen(str[i] + 1);
        bool ff = 0;
        for(int j = 1; j <= len; j++){
            if(len - j + 1 < tl) break;
            if(S[j] == str[i][1]){
                bool fff = 1;
                for(int k = 1; k < tl; k++){
                    if(S[j + k] != str[i][k + 1]){
                        fff = 0;
```

```
                        break;
                    }
                }
                if(fff){
                    ff = 1;
                    break;
                }
            }
        }
        if(!ff) f = 0;
    }
    return num * f;
}
int main(){
    freopen("d.in", "r", stdin);
    freopen("d.out", "w", stdout);
    scanf("%d%s%s", &nn, l + 1, r + 1);
    for(int i = 1; i <= nn; i++){
        scanf("%s", str[i] + 1);
        ins(i, str[i]);
    }
    build_ac();
    pre_work();
    cout << (((DP(r) - DP(l)) % mod + mod) % mod + check(l)) % mod<< endl;
    return 0;
}
```