目录

10.14 模拟赛

1. Traps

【问题描述】

这里有 n 个陷阱，你需要按照给出的顺序穿过这些陷阱，每个陷阱将会对你造成 ai 的伤害

你有至多 k 次机会跳过这些陷阱，可以避免你所跳过的陷阱给你造成的伤害，不过接下来的所有陷阱都会给你多造成 1 点伤害

跳过陷阱所造成的额外伤害会叠加，如果你当前已经跳过了 3 个陷阱，接下来的陷阱给你造成的伤害将会是 ai+3 现在需要你求出受到的最小伤害

【样例输入】

```
5
4 4
8 7 1 4
4 1
5 10 11 5
7 5
8 2 5 15 11 2 8
6 3
1 2 3 4 5 6
1 1
7
```

【样例输出】

```
0
21
9
6
0
```

题目链接：https://www.luogu.com.cn/problem/CF1684D

【题目分析】

假设 i 从 0 开始，分析跳过第 i 个陷阱的实际伤害影响，跳过第 i 个陷阱会减少 ai 的伤害，但会让后面的 n-i-1 个陷阱都增加 1 的伤害，因此跳过第 i 个陷阱实际上减免的伤害为 ai-(n-i-1)=ai-n+i+1。所以可以将实际减少的伤害排序，最大的 k 个一定是减少伤害最大的，而此时还需要考虑这 k 个陷阱互相影响的伤害。除了将 a 数组的和减去这 k 个陷阱的和，还需要减去多算的伤害，由于这 k 个陷阱跳过后不会受到其他陷阱的影响，因此在这 k 个陷阱中，第 x 个陷阱增加的 1 伤害对 x+1 到 k 这 k-x 个陷阱不起作用，因此多算的伤害为 k(k-1)/2。这样把上面的答案再减去这部分即可。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
void solve(){
    int n,k;
    cin>>n>>k;
    vector<int> a(n);
    int sum=0;
    for(int i=0;i<n;i++){
        cin>>a[i];sum+=a[i];
        a[i]-=n;
        a[i]+=i;
        a[i]++;
    }
```

```
    sort(a.rbegin(),a.rend());
    for(int i=0;i<k;i++)sum-=a[i];
    cout<<sum-(k-1)*k/2<<endl;
    return;
}
signed main()
{
   int t;
   cin>>t;
    for(int i=0;i<t;i++){
        solve();
    }
   return 0;
}
```

2. Sleepy Game

【问题描述】

一个人从起点(s)走,当他走不动的时候,如果走了奇数步，输出"Win",否则如果有环，输出"Draw"，否则输出"Lose"

【样例输入】

```
5 6
2 2 3
2 4 5
1 4
1 5
0
1
```

【样例输出】

```
Win
1 2 4 5
```

题目链接：https://www.luogu.com.cn/problem/CF936B

【题目分析】

题目要求判断从起点走若干奇数步能否到达没有出度的点，因此考虑从起点开始 dfs，判断有没有一次走到没有出度的点为奇数步即可。

然而 dfs 的实现还需要注意一些细节。首先，同一个点可能既可以奇数步到达，也可以偶数步到达，因此，我们需要用一个 vis[n][0/1]数组记录，用 0 表示该点能够用偶数步到达，用 1 表示该点能够奇数步到达。起点标记偶数步到达。

然后搜索时利用这个数组进行剪枝，及时停止走重复的路。当下一个点与当前点奇偶性不同的情况已被走过，就没有必要再走了，这样我们可以判断出“Win”的情况，而在这种情况不成立时，只需要从起点再跑一遍 tarjan 判环即可。

这里的易错点是奇环的存在，因此不能粗暴的判断当前点既被偶数步访问又被奇数步访问就停止搜索，因为可能是通过了一个奇环，这样会产生新的可能性。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
const int M=2e5+5;
struct edge{
```

```
    int to,next;
}e[M];
int n,m,cnt,head[N],st;
int chu[N],ans[N],len;
bool tag[N][3],win;
inline void add(int x,int y){
    e[++cnt].to=y;
    e[cnt].next=head[x];
    head[x]=cnt;
}
inline void dfs(int u,int check){//check : 0 偶数，1 奇数
    tag[u][check]=true;
    ans[++len]=u;
    if(win) return;
    for(int i=head[u];i;i=e[i].next){
        int v=e[i].to;
        if(tag[v][check^1]) continue;//若下一个点与当前点奇偶性不同的情况已经被走过
        if(!chu[v]&&(!check)){//若走到没有出度的点时步数为奇数步
            win=true;
            printf("Win\n");
            for(int j=1;j<=len;j++){
                printf("%d ",ans[j]);
            }
            printf("%d",v);
        }
        dfs(v,check^1);
        if(win) return;
    }
    len--;
}
int dfn[N],low[N],tot,cot;
bool insta[N],draw;
stack<int> s;
inline void paint(int u){
    s.pop();
    insta[u]=false;
}
inline void tarjan(int u){
    dfn[u]=low[u]=++tot;
    s.push(u);
    insta[u]=true;
    for(int i=head[u];i;i=e[i].next){
        int v=e[i].to;
        if(!dfn[v]){
            tarjan(v);
            low[u]=min(low[u],low[v]);
        }
        else if(insta[v]){
            low[u]=min(low[u],dfn[v]);
        }
    }
```

```
    if(dfn[u]==low[u]){
        cot++;
        int CNT=1;
        while(s.top()!=u){
            paint(s.top());
            CNT++;
        }
        paint(u);
        if(CNT>=2) draw=true;
    }
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1,k;i<=n;i++){
        scanf("%d",&k);
        chu[i]=k;
        for(int j=1,x;j<=k;j++){
            scanf("%d",&x);
            add(i,x);
        }
    }
    scanf("%d",&st);
    dfs(st,0);//标记奇偶
    if(win)
        return 0;
    else{
        tarjan(st);//找环
        if(draw){
            printf("Draw");
        }
        else printf("Lose");
    }
    return 0;
}
```

3. Skills

【问题描述】

L 在玩一个游戏。每个人有 n 个技能，每个技能都有一个等级，满级都是 A ，每个人的初始技能等级为非负整数 ai。

玩家的排名是由一个能力值决定的。这个能力值是以下 2 种值的和：

1.满级技能的个数（即 ai=A）乘上一个系数 cf

2.所有技能中最低的等级（即 minai）乘上一个系数 cm

现在 L 愿意花 m 单位钱币（可以不用完），每一单位钱币可以使一种技能升一级。现在请你帮他操作，使他的能力值在操作后最大，并输出操作后每种能力的等级。注意每种能力最多升到 A 级。

【样例输入】

```
3 3
101
001
110
```

【样例输出】

```
2
```

题目链接：https://www.luogu.com.cn/problem/CF613B

【题目分析】

设等于 A 的数的个数为 k，最小值为 min，那么答案为 k * cf + min * cm，那么答案只跟 k 与 min 有关，而满级个数与最小值是互斥的，并不能找到一个贪心策略同时维护这两个值。既然无法同时维护，那就考虑在某个值确定的情况下，另一个值的最优情况。

这种情况就是经典的枚举+二分的应用，确定两个互斥的信息，利用枚举维护一个信息，二分维护在第一个信息的情况下另一个信息。将等级从大到小排好序，那么在技能点够用的情况下，从前往后开始枚举技能点点满的个数，然后在这个情况下二分维护最小值，判断一遍合法性即可得到当前的最优值，判断合法性可以再利用二分或者前缀和来进行判断。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=100005;
int n,A,cf,cm,m,sum[N],ans=-1,rk,v,b[N];
struct node{
    int val,id;
}a[N];
bool cmp(const node &a,const node &b) {
    return a.val<b.val;
}
int find(int sum,int x) {
    int l=0,r=x,res;
    while(l<=r) {
        int mid=(l+r)>>1;
        if(a[mid].val<sum)res=mid,l=mid+1;
        else r=mid-1;
    }
    return res;
}
signed main() {
    scanf("%lld%lld%lld%lld%lld",&n,&A,&cf,&cm,&m);
    for(int i=1;i<=n;++i)
        scanf("%lld",&a[i].val),a[i].id=i;
    sort(a+1,a+n+1,cmp);
    for(int i=1;i<=n;++i)
        sum[i]=sum[i-1]+a[i].val;
    m+=sum[n];
    for(int i=1;i<=n+1;++i) {
        if(A*(n-i+1)+sum[i-1]>m)continue;
        int l=a[1].val,r=A,res;
        while(l<=r) {
            int mid=(l+r)>>1;
            int t=find(mid,i-1);
            if(A*(n-i+1)+mid*t+sum[i-1]-sum[t]<=m)res=mid,l=mid+1;
            else r=mid-1;
        }
```

```
        if(ans<res*cm+(n-i+1)*cf) {
            ans=res*cm+(n-i+1)*cf;
            rk=i,v=res;
        }
    }
    printf("%lld\n",ans);
    for(int i=1;i<rk;++i)a[i].val=max(a[i].val,v);
    for(int i=rk;i<=n;++i)a[i].val=A;
    for(int i=1;i<=n;++i)b[a[i].id]=a[i].val;
    for(int i=1;i<=n;++i)printf("%lld ",b[i]);
    return 0;
}
```

4. 505

【问题描述】

给出一个 n×m 01 矩阵，如果每个长宽都为偶数的正方形子矩阵内 1 的个数都为奇数，则这是一个“好的”矩阵。如果能把矩阵改成“好的”，问最少改多少个数。如果不能，输出−1。

【样例输入】

```
3 3
101
001
110
```

【样例输出】

```
2
```

题目链接：https://www.luogu.com.cn/problem/CF1391D

【题目分析】

观察题目数据范围，发现 n*m 是在 1e6 范围内的，这样二维数组无法直接存储，但观察题目情况分类讨论，考虑小范围数据，由于当 n>=4 并且 m>=4 时，4 个奇数一定是偶数，所以不存在所有偶矩阵都为奇数个 1 的情况，这样排除了无解情况，二维数组依然能够存储。

接下来只需要判断 n<4 的情况即可。当 n=1 时，一定是好矩阵，直接输出即可，当 n=2 时，此时矩阵只有两列，只需要考虑两列的 m-1 个 2×2 的区域即可，而某一列是否修改是取决于前一列中 1 的个数的，所以枚举前两列的修改情况即可。

当 n=3 时，考虑状压 dp，设 dp[i][j]表示在第 i 列 j 状态下时所需要修改的最小次数，这样枚举前一列所有合法的状态 k，根据 k 向 j 转移即可，计算出第 i 列修改为 j 状态所需要的次数 change（i，j）。

则 dp[i][j] = min(dp[i][j],dp[i-1][k]+change(i,j)。整个状态只有 8 种，所以时间复杂度为 O(n)，常数为 16 倍。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define M 20
#define N 1000050
using namespace std;
int n,m,mp[5][N];
int sta[M],cnt[M],tot;
char s[N];
int f[N][M],ans;
void init()
{
```

```
    for(int i=0;i<(1<<n);i++)
    {
        sta[++tot]=i; int u=i;
        while(u){cnt[tot]+=((u%2)^0);u>>=1;}
    }
}
int change(int id,int y)
{
    int ans=0,status=sta[id];
    for(int i=n;i>=1;i--,status>>=1)
        ans+=((status%2)^mp[i][y]);
    return ans;
}
void dp()
{
    init(); memset(f,0x3f,sizeof f);
    for(int i=1;i<=tot;i++) f[1][i]=change(i,1);
    for(int i=2;i<=m;i++)
        for(int j=1;j<=tot;j++)
            for(int k=1;k<=tot;k++)
            {
                int a=sta[j],b=sta[k];
                int last=a%2+b%2,now,flag=1;
                a>>=1; b>>=1;
                for(int l=n-1;l>=1;l--)
                {
                    now=a%2+b%2;
                    a>>=1; b>>=1;
                    if((last+now)%2==0) flag=0;
                    last=now;
                }
                if(flag) f[i][j]=min(f[i][j],f[i-1][k]+change(j,i));
            }
}
int main()
{
    scanf("%d%d",&n,&m);
    if(n>=4){printf("-1");return 0;}
    if(n==1){printf("0"); return 0;}
    for(int i=1;i<=n;i++)
    {
        scanf("%s",s+1);
        for(int j=1;j<=m;j++)
            mp[i][j]=s[j]-'0';
    }
    dp(); ans=n*m;
    for(int i=1;i<=tot;i++) ans=min(ans,f[m][i]);
    printf("%d",ans);
    return 0;
}
```