枚举优化与搜索 习题讲解

洛谷网校

基础衔接提高计划

www.luogu.com.cn

T494768 奇怪的按钮

小五拿到了一个奇怪的装置，上面有 3×3 共 9 个按钮。每个按钮有 凹/凸（用 0/1 表示）两种状态。奇怪的是，每次按下按钮不仅会切换当前按钮的状态，还会同时切换上下左右按钮的状态。作为强迫症患者的小五，希望能使得这些按钮全部变为凸状态（即全为 1）。现给出装置的初始状态，请你输出最少需要按多少次，能使所有按钮变为凸状态。

T494768 奇怪的按钮

- 最为简单的想法是递归枚举每一次按按钮的位置，直到所有的位置均为 1 。
- 当然，如果你真的尝试了这样做，你很可能发现程序陷入了无限，甚至跑不出样例。这是因为当我们对一个位置按了两次之后，其自身状态改变了两次，周围按钮的状态改变了两次，这样这些按钮状态本质上和没进行操作一样！这启发我们其实**每个按钮至多只能按一次**，这样我们递归枚举按了哪些按钮即可解决问题，复杂度 $O(2^9)$ 。（由于数据比较弱，可能朴素bfs也能通过此题）

T494768 奇怪的按钮

```
void act(int x,int y){
    a[x][y]^=1;
    a[x+1][y]^=1;
    a[x][y+1]^=1;
    a[x-1][y]^=1;
    a[x][y-1]^=1;
}
void dfs(int dep,int k){
    if(dep>9) return;
    if (k>=ans) return;
    int s=1;
    for(int i=1;i<=3;i++)
    for(int j=1;j<=3;j++)
        s&=a[i][j];
    if(s){
        ans=min(ans,k);
        return;
    }
```

```
    int x=dep/3+1,y=dep%3+1;
    dfs(dep+1,k); //不操作
    act(x,y); //对(x,y)操作
    dfs(dep+1,k+1);
    act(x,y); //再做一次恢复状态
    return;
}
int main(){
    for(int i=1;i<=3;i++)
        for(int j=1;j<=3;j++)
            cin>>a[i][j];
    dfs(0,0);
    cout<<ans<<endl;
}
```

T494790 淹园

小五的校园可以抽象成一个 n 行 m 列的矩阵。最初（第 0 小时）有 p 处积水，这些积水每小时会向上下左右四个方向蔓延，使其也成为积水处。现有 q 次询问，每次查询一个坐标 (x_i, y_i)，请回答其最早在多少小时变为积水处。

$1 \leq n, m \leq 500$, $1 \leq p, q \leq 10^5$

T494790 淹园

- 首先考虑最为直接的方法：我们发现对于一处积水，其使某位置变为积水的最短时间即为二者的曼哈顿距离（即$abs(x_i - x_j) + abs(y_i - y_j)$）。所以每次询问一个位置，我们暴力计算其距离每个初始积水处的曼哈顿距离并取其最小值，即为其最早变为积水的时间。复杂度为 $O(pq)$ 。
- 回忆我们上课讲过的$floodfill$算法，我们发现在积水蔓延的过程中，每个格子只有第一次被访问是有意义的，而较晚的访问其蔓延出去的时间都会比之前的访问晚，不会影响答案。故我们直接一开始将所有初始积水加入队列，每次从队首弹出一个位置，将其四周没有被加入过队列的位置加入队列，并更新该位置的访问时间（即答案）。这样，每个位置只会被入队一次，时间复杂度 $O(nm)$ 。

T494790 淹园

```
int dx[]={-1,1,0,0},dy[]={0,0,1,-1};
queue<pair<int,int>> que;
void bfs(){
    while(!que.empty()){
        pair<int,int> now=que.front();
        int u_x=now.first,u_y=now.second;
        que.pop();
        for(int i=0;i<4;i++){
            int v_x=u_x+dx[i],v_y=u_y+dy[i];
            if (v_x<=0||v_x>n||v_y<=0||v_y>m)
                continue;
            if (vis[v_x][v_y]) continue;
            vis[v_x][v_y]=1;
            dis[v_x][v_y]=dis[u_x][u_y]+1;
            que.push(make_pair(v_x,v_y));
        }
    }
}
```

```
int main(){
    n=read(),m=read(),p=read(),q=read();
    memset(vis,0,sizeof(vis));
    memset(dis,0,sizeof(dis));
    while(p--){
        int x=read(),y=read();
        vis[x][y]=1;
        que.push(make_pair(x,y));
    }
    bfs();
    while(q--){
        int x=read(),y=read();
        printf("%d\n",dis[x][y]);
    }
    return 0;
}
```

T494863 强迫症

小五攒下了总计 n 元资金，他准备将其全部拿来买薯片和可乐。其中薯片每袋售价 a 元，可乐每瓶售价 b 元。但是小五有强迫症，**他必须将手中的钱全部花完**，请你为他设计购买方案。当然，小五喜欢可乐，所以如果存在多种方案，请输出可乐瓶数最多的方案。

注意，可乐和薯片均不可分割，即必须购买非负整数份。

$1 \leq n, a, b \leq 10^7$

T494863 强迫症

- 考虑最为暴力的做法，因为必须购买整数份，所以我们可以直接枚举薯片和可乐的购买数量 x 和 y 。再判断是否满足 $ax+by=n$ 的条件。由于 $a,b\geq 1$ ，所以此处 x,y 均应当 $\leq n$ 。故只需枚举到 n ，时间复杂度 $O(n^2)$ 。
- 但我们注意到实际上并不需要同时枚举 x 和 y ，对于一个确定的 x ，我们可以由 $ax+by=n$ 得到 $by=n-ax$ ，即 $n-ax$ 应当是 y 的倍数（或为 0），所以我们可以直接通过 $(n-ax)\ mod\ y==0$ 来判定，时间复杂度为 $O(n)$ 。
- 当然，这道题还有更好的做法，相信同学们很快就将学到。

T494863 强迫症

```
int main(){
    cin>> n>> a>> b;
    for(int x=0; x<=n/a; x++)
        if( (n-x*a) %b == 0 )
            cout<< x<< " "<< (n-x*a)/b,exit(0);
    cout<<"NO";
    return 0;
}
```

T494849 坚持锻炼

小五放暑假了，准备开始假期锻炼。他的暑假为期 n 天，第 i 天可以从 x_i, y_i 两种运动中选择一种进行。而小五只会选择一段连续的日子锻炼，并且这些天所做的运动必须完全相同。请计算出一种方案，使得小五锻炼的日子尽可能得长。如果存在有多种方案，请输出运动编号较小的那个。

对于 100% 的数据，$1 \leq n \leq 10^5, 1 \leq x_i, y_i \leq 5$

T494849 坚持锻炼

- 考虑最朴素的做法，我们暴力枚举每个区间并检查区间内是否共有某项运动。复杂为$O(N^3)$或$O(N^2)$，无法通过此题。
- 进一步的，我们发现其实x_i, y_i的范围特别小。这启发我们其实可以将要做的运动作为枚举项。在枚举运动编号t之后，我们可以将那些包含t的日子记为1，不包含的日子记为0。这样问题就转化为了在一个$0/1$序列中找到最长的全为1的区间。这样我们直接顺序遍历序列，记录最长全1后缀，并不断更新答案即可。时间复杂度$O(|x_i|N)$。

T494849 坚持锻炼

```
int main(){
    scanf("%d",&n);
    for(int i=0;i<n;++i)scanf("%d%d",&a[i],&b[i]);
    for(int i=1;i<=5;i++){
        int cnt=0;
        for(int j=0;j<n;++j){
            if(a[j]==i||b[j]==i)cnt++;
            else cnt=0;
            if(cnt>maxx){
                maxx=cnt;
                now=i;
            }
        }
    }
    printf("%d %d\n",maxx,now);
    return 0;
}
```

P5002 专心OI - 找祖先

这个游戏会给出你一棵树，这棵树有 N 个节点，根结点是 R，系统会选中 M 个点 $P_1, P_2 \ldots P_M$，求有多少组点对 (u_i, v_i) 的最近公共祖先是 P_i。

$1 \leq R \leq N \leq 10000\,, 0 \leq M \leq 50000$

P5002 专心OI - 找祖先

- 考虑最简单的做法，我们对于每个询问 P_i，枚举每对点对 (u,v)，查询其最近公共祖先 $lca(u,v)$。若 $lca(u,v)=P_i$，则该次询问答案加一。
- 更进一步的，我们发现其实没必要对于每组询问都枚举一轮点对。只需考虑将每个点 x 的答案记录为 $ans[x]$。在枚举点对时使得 $ans[lca(u,v)]+=1$，最后输出询问处的答案即可。
- 利用上课时讲到的利用dfs序和st表求lca，即可做到$O(1)$查询lca，这样总时间复杂度$O(N^2)$，可以通过此题。（可通过只枚举 $u\le v$ 的点对减少枚举次数）
- 同学们会在接下来的课程会学到该题更优秀的做法。

P5002 专心OI - 找祖先

```
void dfs(int u,int f){
    fa[u]=f,siz[u]=1;
    dfn[u]=++cnt;dep[u]=dep[f]+1;
    st[0][dfn[u]]=u;
    for(auto v:g[u]){
        if (v==f) continue;
        dfs(v,u); siz[u]+=siz[v];
    }
}
int ask(int l,int r){
    int len=r-l+1,bit=log_2(len);
    int x=st[bit][l],y=st[bit][r-(1<<bit)+1];
    if (dep[x]<dep[y]) return fa[x];
    else return fa[y];
}
int main(){
    n=read(),s=read(),m=read();
    for(int i=1;i<n;i++){
        int u=read(),v=read();
        g[u].push_back(v);g[v].push_back(u);
    }
    dfs(s,0);
    for(int i=1;i<=20;i++)
        for(int j=1;j+(1<<i)-1<=n;j++){
            int x=st[i-1][j];
            int y=st[i-1][j+(1<<(i-1))];
            if (dep[x]<dep[y]) st[i][j]=x;
            else st[i][j]=y;
        }
    for(int i=1;i<=n;i++) ans[i]=1;
    for(int i=1;i<=n;i++)
        for(int j=i+1;j<=n;j++){
            int u=i,v=j;
            if (dfn[u]>dfn[v]) swap(u,v);
            ans[ask(dfn[u]+1,dfn[v])]+=2;
        }
    for(int i=1;i<=m;i++)
        printf("%d\n",ans[read()]);
    return 0;
}
```

P2329 [SCOI2005] 栅栏

农夫约翰需要 n 块特定规格的木材，可是木材店老板说他这里只剩下 m 块大规格的木板了。不过约翰可以购买这些木板，然后切割成他所需要的规格。而且约翰有一把神奇的锯子，用它来锯木板，不会产生任何损失，也就是说长度为 10 的木板可以切成长度为 8 和 2 的两个木板。

你的任务：给你约翰所需要的木板的规格，还有木材店老板能够给出的木材的规格，求约翰最多能够得到多少他所需要的木板。

$m \leq 50, n \leq 1000$，木材规格小于 32767 。

P2329 [SCOI2005] 栅栏

- 首先我们考虑最朴素的做法。直接递归枚举每个约翰需要的木板切割自哪块大木材（或者未获得该木板），并从中统计切割出木板最多的方案。但这种做法的复杂度（约 $O(m^n)$）与目标数据范围相差甚远，只是简单的剪枝无法通过。
- 这时候，我们可以发现一个简单的结论：我们的最终方案一定可以只选择最短的那些木板，并且如果我们能获取 k 块木板，则一定可以获得 k' $(k' \leq k)$ 块木板。这样，我们可以通过二分答案 k，将问题转化为是否能够从这些木材中切割出最短的那 k 块木板。

P2329 [SCOI2005] 栅栏

- 这时候我们就可以考虑对搜索进行剪枝了。由于此处我们已将问题转为判定问题，故基本考虑可行性剪枝。
- 首先，如果我们剩下的木材的长度和已经小于需要的木板的长度和，则一定不可能切割成功，可以结束分支。
- 接着，我们可以考虑切割过程中“浪费”的部分。如果一段木材的长度小于最短的木板的长度，则其一定不可能被利用，我们可以将他的长度从木材长度和中扣除掉，从而得到“可利用的木材长度和”，帮助减少搜索时间。
- 最后，注意枚举顺序。从较长的木板开始枚举可以更早的引出矛盾从而剪枝，进而大幅优化效率。

P2329 [SCOI2005] 栅栏

```
int wast=0,sum=0;
bool check(int now,int need){
    if (now==0)return 1;
    if (need>sum-wast)return 0;
    for(int i=m;i>=1;i--){
        if (b[now]>a[i])continue;
        a[i]-=b[now];
        if(a[i]<b[1])wast+=a[i];
        int res=check(now-1,need);
        if (a[i]<b[1])wast-=a[i];
        a[i]+=b[now];
        if (res) return 1;
    }
    return 0;
}
int main(){
    m=read();
    for(int i=1;i<=m;i++)
        a[i]=read(),sum+=a[i];
    n=read();
    for(int i=1;i<=n;i++)
        b[i]=read();
    sort(a+1,a+m+1);sort(b+1,b+n+1);
    if(b[1]>a[m]){puts("0"); return 0;}
    for(int i=1;i<=n;i++)
        c[i]=c[i-1]+b[i];
    while(c[n]>sum)n--;
    int l=1,r=n,ans;
    while(l<=r){
        int mid=(l+r)/2;
        if (check(mid,c[mid]))
            ans=mid,l=mid+1;
        else r=mid-1;
    }
    cout<<ans<<endl;
    return 0;
}
```

P1146 硬币翻转

在桌面上有一排硬币，共 N 枚，每一枚硬币均为正面朝上。现在要把所有的硬币翻转成反面朝上，规则是每次可翻转任意 $N-1$ 枚硬币（正面向上的被翻转为反面向上，反之亦然）。求一个最短的操作序列（将每次翻转 $N-1$ 枚硬币成为一次操作）

$1 \leq N \leq 100$，N 为偶数

P1146 硬币翻转

- 如果你尝试过使用递归枚举每一步操作，将发现同样会陷入无限递归，因为对一个位置做两次操作，对硬币状态无影响。这启发我们最多对一个位置做一次操作，总共最多做 N 次 操作。
- 更进一步的，我们可以将每次操作替换为：**将所有硬币翻一次面，再将指定位置的硬币翻一次面**。设总共进行了 k （$k \leq N$）次操作，则等价于先将所有硬币翻了 k 次面，再选 k 个硬币翻面。

P1146 硬币翻转

我们按 k 的奇偶性讨论：

- 若 k 为奇数，则 k 次全部翻转后所有硬币已经反面朝上，而奇数次单个硬币反转必然使得某个位置被翻回正面朝上，故不可能。
- 若 k 为偶数，则 k 次全部翻转后所有硬币仍然全部正面朝上，接着用 k（$k \leq N$）次单个硬币翻转使得所有硬币反面朝上，有且只有当 $k = N$（注意到 N 为偶数）时，通过对每个位置均进行一次操作得到。故我们得到方法为对每一个位置均进行一次操作即可。

P1146 硬币翻转

```
int main(){
    int n=read();
    printf("%d\n",n);
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            if (i!=j) a[j]^=1;
            printf("%d",a[j]);
        }
        puts("");
    }
    return 0;
}
```

P1003 [NOIP2011 提高组] 铺地毯

为了准备一个独特的颁奖典礼，组织者在会场的一片矩形区域（可看做是平面直角坐标系的第一象限）铺上一些矩形地毯。一共有 n 张地毯，编号从 1 到 n 。现在将这些地毯按照编号从小到大的顺序平行于坐标轴先后铺设，后铺的地毯覆盖在前面已经铺好的地毯之上。

地毯铺设完成后，组织者想知道覆盖地面某个点的最上面的那张地毯的编号。注意：在矩形地毯边界和四个顶点上的点也算被地毯覆盖。地毯的左下角的坐标设为 (a,b) 。

$1 \le n \le 10^4$，$0 \le a,b \le 10^5$

P1003 [NOIP2011 提高组] 铺地毯

- 考虑较为暴力的做法，我们直接模拟铺地毯的过程。开一个二维数组记录地毯覆盖的状态。每新加入一个地毯，我们将其对应的矩形区域内全部赋值为他的id，这样该二维数组中每个位置都记录其上最后铺设的（即最上层的）地毯的id，查询时直接输出即可。但由于内存限制，这种做法无法通过。
- 注意到其实查询操作只有一次，我们不需要追求 $O(1)$ 查询。考虑按先后顺序枚举每个地毯，如果该地毯覆盖最终的查询位置，则更新最终答案。因为我们从前到后枚举，所以最终答案一定是最上层的地毯。时间复杂度 $O(n)$。

P1003 [NOIP2011 提高组] 铺地毯

```
int main(){
    int n=read();
    for(int i=1;i<=n;i++)
        a[i]=read(),b[i]=read(),g[i]=read(),k[i]=read();
    int x=read(),y=read();
    int ans=-1;
    for(int i=1;i<=n;i++)
        if ((x>=a[i])&&(x<=a[i]+g[i])
            &&(y>=b[i])&&(y<=b[i]+k[i])) ans=i;
    cout<<ans<<endl;
    return 0;
}
```