目录

10.4 模拟赛

1. Post Lamps

【问题描述】

给定一条数轴，从 0 到 n-1 这 n 个位置都可以放置路灯，但有些地方存在障碍不能放，路灯具有功率设为 l，如果把一盏功率为 l 的路灯放到 x 位置，他能照亮 [x,x+l-1]，然后我们需要找到一种功率，也只能放这种功率的路灯，使得他能照亮整条数轴[0,n]，放置一次的贡献是 al，求最小成本。

【输入格式】

第一行输入三个整数 n，m，k（1≤k≤n≤10^6,0≤m≤n） 表示数轴长度，禁止放置的数量，灯的数量；

接下来输入一行 m 个整数 $s1,s2,\cdots,sn$(0≤s1<s2<s3<…<sm<n)，表示禁止放置的位置。

接下来输入一行 m 个整数 $a1,a2,\cdots,ak$(1≤ai≤1e6)，表示每个灯的成本。

【输出格式】

若无解则输出-1，否则输出最小成本。

【样例输入】

```
6 2 3
1 3
1 2 3
```

【样例输出】

```
6
```

题目链接：https://www.luogu.com.cn/problem/CF990E

【题目分析】

枚举功率，贪心地放置路灯，贪心策略就是每次都选择能覆盖到的范围内的最右边的非障碍的点，如果这个点等于原来的出发点就直接 return -1，因为说明不可行。这时候我们需要预处理出来每个点对应的它左边离他最近的那个非障碍的点，用于回跳。此时枚举复杂度需要计算，考虑极限情况，枚举总和为 n+n/2+n/3+…+n/k，为调和级数，总的复杂度为 nlogn，所以时间复杂度能够通过。

需要注意 0 处必须放置路灯，并且需要覆盖整个区间并非覆盖所有点。

【参考代码】

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int INF=1e6+5;
int n,m,k,s[INF],vis[INF],pre[INF],a[INF];
signed main()
{
    ios::sync_with_stdio(false);
    cin>>n>>m>>k;
    for (int i=1;i<=m;i++) cin>>s[i],vis[s[i]]=1;
    for (int i=0;i<n;i++)
        if (!vis[i]) pre[i]=i;
        else pre[i]=pre[i-1];
    int Max=0,res=0;
    for (int i=0;i<n;i++) {
        if (vis[i]) Max++;
        else Max=0;
        res=max(res,Max);
    }
    for (int i=1;i<=k;i++) cin>>a[i];
    if (vis[0]) {cout<<"-1\n";return 0;}
    int ans=1e18;
```

```
    for (int i=1;i<=k;i++) {
        if (i<=res) continue;
        int r=pre[0],sum=0;
        while (r<n) {
            sum++;
            r+=i;
            if (r>=n) break;
            r=pre[r];
        }
        ans=min(ans,sum*a[i]);
    }
    if (ans>1e17) cout<<"-1\n";
    else cout<<ans<<"\n";
    return 0;
}
```

2. Make It Equal

【问题描述】

有一个长度为 n 的序列。定义切割操作为把序列中所有大于等于 H（自定义）的数变为 H，定义其代价为操作前后序列中所有数的变化量的和。
定义一个“好的”切割操作为代价小于等于 K 的操作。
问至少需要多少次“好的”切割操作才可以使序列中的所有数的大小均相等。

【输入格式】

第一行两个整数，n 和 k，n<=2*10^5，n<=k<=10^9，表示序列长度和“好的”切割的代价
第二行 n 个整数，分别表示序列中每个元素的高度。

【输出格式】

最小切割操作。

【样例输入】

```
5 5
3 1 2 2 4
```

【样例输出】

```
2
```

题目链接：https://www.luogu.com.cn/problem/CF1065C

【题目分析】

桶排序+贪心前缀和，最优解总是使序列中的每个数都等于原来的最小数，首先利用桶排序，从高到低统计出每种高度正方体高度的个数，然后贪心地从高到低进行切割，那么题目就相当于每一次从最上层开始取连续的几层，且每一次取得总点数不超过 k，每次前缀和超过 k 时，ans 就要加一，最后还有切完没统计的，也要把 ans 加一。

【参考代码】

```
#include<bits/stdc++.h>
#define int long long
using namespace std;

const int maxn = 4e5 + 5;
int n, k, a[maxn], b[maxn], sum[maxn];
signed main() {
    scanf("%lld%lld", &n, &k);
    for (int i = 1; i <= n; i++) scanf("%lld", &a[i]);
    sort(a + 1, a + 1 + n);
```

```
    if (a[1] == a[n]) {
        printf("0");
        return 0;
    }
    int pos = 1;
    for (int i = 1; i <= n; i++) {
        while (pos <= 200000 && pos <= a[i]) {
            sum[pos] = (n - i + 1);
            pos++;
        }
    }
    for (int i = 1; i <= 200000; i++) b[i] = sum[200000 - i + 1];
    pos = 1;
    int ans = 0, cnt = 0;
    while (pos <= 200000) {
        if (cnt + b[pos] <= k) {
            cnt += b[pos];
            pos++;
            if (pos == 200001) ans++;
        }
        else {
            ans++;
            if (b[pos] == n) break;
            cnt = b[pos];
            pos++;
        }
    }
    printf("%lld", ans);
    return 0;
}
```

3. Present

【问题描述】

Little Beaver 的信息学教师要过生日了，他种了 n 朵花想当作礼物，但是一段时间后他发现花不长了，他认为送特别小的花是不好的行为，就想了一种方案：

每一天，他会对于一个区间[i,i+w-1]进行一种特殊的浇水，让该区间内所有花的高度都增加 1。

现在，距离生日还有 m 天，他想让花的最小高度最大。

一个长度为 n 的序列 a ，你有 m 次操作的机会，每次操作是将其中连续的 w 个元素增加 1 。最大化最终序列的最小值。

【样例输入】

```
6 2 3
2 2 2 2 1 1
```

【样例输出】

```
2
```

题目链接：https://www.luogu.com.cn/problem/CF460C

【题目分析】

看到最大化最终序列的最小值，考虑二分。但直接模拟操作时间复杂度明显无法通过，考虑贪心操作，当存在一个小于目标值的数 x，就一定要把它加到 x，此时为了不浪费操作次数，可以直接把当前这个数作为左端点累加到 x，区间操作用差分数

组或者线段树/树状数组来实现，但复杂度都满足题目要求。有些位置不能作为长度为 w 的区间的左端点，此时需要操作完成后，再把[n-w+1,n]这个区间内的数值加上剩余的操作次数即可。

操作完后判断是否还存在不满足条件的位置，用于二分 check 判断。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
long long N, M, W;
//h 为高度，dis 为差分
long long h[100010], dis[100010];
//判断
bool check(long long H)//H 为目标最小高度
{
    long long cur = 0, dsum = 0;
    for (int i = 1;i <= N;i++)
    {
        cur += dis[i];//遍历
        if (cur < H)//需要浇水
        {
            dsum += H - cur;//增加天数
          //更改
            dis[i] += H - cur;
          //注意越界问题
            if(i + W <= N) dis[i + W] -= H - cur;
            cur += H - cur;//dis 加了，cur 当然也要加了
        }
        //for (int i = 1;i <= N;i++) printf("%d ",dis[i]);
        //printf("\n");
    }
    return dsum <= M;
}
int main()
{
    scanf("%lld%lld%lld", &N, &M, &W);
    long long l = 111111111, r = -1;
    for (int i = 1;i <= N;i++)
    {
        scanf("%lld", &h[i]);
       //开差分数组
        dis[i] = h[i] - h[i - 1];
       //l 要保证合法，取最小
        l = min(h[i], l);
       //r 要保证不合法，取最大，下面再加（天数+1）
        r = max(h[i], r);
    }
    //for (int i = 1;i <= N;i++) printf("%d ",dis[i]);
    //printf("\n");
    r += M + 1;
   //二分
    while (l < r - 1)
    {
        long long m = (l + r) / 2;
```

```
        if (check(m)) l = m;//合法
        else r = m;//不合法
       //重新计算差分数组（check 已经将其更改）
        for (int i = 1;i <= N;i++)
            dis[i] = h[i] - h[i - 1];
    }
    printf("%lld", l);//输出合法值
    return 0;
}
```

4. New Year Tree

【问题描述】

给出一棵 n 个节点的树，根节点为 1。每个节点上有一种颜色 ci
m 次操作。操作有两种：
1 u c：将以 u 为根的子树上的所有节点的颜色改为 c。
2 u：询问以 u 为根的子树上的所有节点的颜色数量。

【样例输入】

```
7 10
1 1 1 1 1 1
1 2
1 3
1 4
3 5
3 6
3 7
1 3 2
2 1
1 4 3
2 1
1 2 5
2 1
1 6 4
2 1
2 2
2 3
```

【样例输出】

```
2
3
4
5
1
2
```

题目链接：https://www.luogu.com.cn/problem/CF620E

【题目分析】

首先直接递归修改子树很明显不可能通过，需要考虑如何在 logn 的复杂度内进行维护，所以要与线段树与树状数组产生联系，但树上子树的修改怎么能用数据结构维护，是需要处理的点。

dfs 序可以帮助我们将树上问题转化为区间问题，dfs 序是从根节点开始进行先序遍历访问的结点顺序，将每次访问的结点编号输出即可，这样对于每个有子树的结点，都会经过两次，一次 in 一次 out，而 in 和 out 之间的所有结点，即为当前结点的子树，这样就成功地把一棵树转换为了线性结构。

修改子树就是把 dfs 序区间内进行修改，考虑颜色状态数不超过 64，所以可以用二进制来进行状态压缩，把颜色压成二进制数值然后用这个数值来进行区间值的维护，统计当前二进制中为 1 的位数即可，这里可以用 lowbit 快速查找。

【参考代码】

```
#include<bits/stdc++.h>
#define int long long
typedef unsigned long long ull;
const int Maxn=400010;
using namespace std;

struct edge{
    int v,nxt;
}e[Maxn<<2];int head[Maxn],cnt;//用链式前向星存边

void adg(int u,int v){//加边操作
    e[++cnt].v=v,e[cnt].nxt=head[u],head[u]=cnt;
}

int pos[Maxn],t;//pos 用来存放 dfs 序
pair<int,int>tim[Maxn];//用来维护子树信息 tim[i].second 代表这棵子树的最后一个元素

void dfs(int x,int f){
    pos[++t]=x;//存储 dfs 序
    tim[x].first=t;//这棵子树的根
    for(int i=head[x];i;i=e[i].nxt){//遍历这个节点的边
        int tmp=e[i].v;//存放下个节点
        if(tmp==f)continue;//不能往回走
        dfs(tmp,x);
    }
    tim[x].second=t;//这棵子树的最后一个元素
}

#define ls p<<1//偷个懒，事先设定好
#define rs p<<1|1//和 p*2+1 一样
ull c[Maxn];//原序列

struct Stree{
    int l,r;ull dat,chg;
    #define l(x) St[x].l//同样，方便写代码，也方便理解（从哪里偷来的自信）
    #define r(x) St[x].r
    #define d(x) St[x].dat
    #define c(x) St[x].chg
}St[Maxn<<2];//因为最后一层有空节点，所以要开四倍
//进入最可爱的线段树部分
void build(int p,int l,int r){//建树
    l(p)=l,r(p)=r;//确定子树范围
    if(l==r){d(p)=1ll*c[pos[l]];return;}//如果区间长度为一，代表到了最底层，返回
    int mid=(l+r)>>1;
    build(ls,l,mid);//建立左子树
    build(rs,mid+1,r);//建立右子树
    d(p)=d(ls)|d(rs);//上传信息，由于要统计颜色个数，所以用不进位加法（或|），代表有这种颜色
}
```

```
void spread(int p){//下传标记，注意是直接修改，不是增加。
    if(c(p)){//如果有标记
        d(ls)=c(p),d(rs)=c(p);//直接修改子树值
        c(ls)=c(p),c(rs)=c(p);//把标记下传给字数
        c(p)=0ll;//取消标记
    }
}

void change(int p,int l,int r,int d){//区间修改操作
    if(l(p)>=l&&r(p)<=r){
        d(p)=d;c(p)=d;return;
    }//如果覆盖区间，直接打标记，降低复杂度
    spread(p);//否则先下传标记
    int mid=(l(p)+r(p))>>1;
    if(l<=mid)change(ls,l,r,d);//如果左子树有重叠，修改左子树
    if(r>mid) change(rs,l,r,d);//同理可证
    d(p)=d(ls)|d(rs);//上传信息，线段树每部操作都要顺便上传信息
}

ull ask(int p,int l,int r){//询问操作
    if(l(p)>=l&&r(p)<=r){
        return d(p);
    }//如果完全覆盖，就直接返回
    spread(p);//否则先下传标记
    int mid=(l(p)+r(p))>>1;
    ull val=0;//一开始没有任何颜色
    if(l<=mid)val|=ask(ls,l,r);//如果左子树有重叠，统计左子树
    if(r>mid) val|=ask(rs,l,r);//如果右子树有重叠，统计右子树
    d(p)=d(ls)|d(rs);//顺便上传信息
    return val;//返回颜色
}

signed main(){
    int n,m;
    scanf("%lld%lld",&n,&m);

    for(int i=1,tmp;i<=n;i++){
        scanf("%lld",&tmp);c[i]=(1ll<<tmp);
    }

    int x,y;
    for(int j=1;j<=n-1;j++){
        scanf("%lld%lld",&x,&y);
        adg(x,y),adg(y,x);
    }

    dfs(1,0);build(1,1,n);

    #define lowbit(x) x&-x
    int t,v,l,r;ull ch;
```

```
    for(int k=1,tmp;k<=m;k++){
        scanf("%lld",&t);
        if(t==2){
            scanf("%lld",&v);
            l=tim[v].first,r=tim[v].second;
            ull temp=ask(1,l,r);int ans=0;
            for(;temp;temp-=lowbit(temp))ans++;//统计答案，lowbit 是最后一位唯一的数位
            printf("%lld\n",ans);
        }else{
            scanf("%lld%lld",&v,&tmp);
            l=tim[v].first,r=tim[v].second,ch=(1ll<<tmp);
            change(1,l,r,ch);
        }
    }
    return 0;
}
```