BZOJ JOI 2013~2014 春季training合宿 系列题解

4236　　JOIOJI

前缀和转化，对J-O+n 和 O-i+n hash一下即可

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;
#define rep(i,a,b) for(int i=a;i<=b;i++)
const int N = 2 * 100000 + 10, M = 1007;
typedef pair<int, int> node;
#define mp(a,b) make_pair(a,b)
#define x first
#define y second
struct hash{
    int h[M][M], l, pre[N], s[N];
    node q[N];
    hash(){
        l = 0;
    }
    int ins(node a, int s1){
        for(int i = h[a.x % M][a.y % M]; i; i = pre[i])
            if (q[i] == a) return s[i];
        q[++l] = a, s[l] = s1;
        pre[l] = h[a.x % M][a.y % M], h[a.x % M][a.y % M] = l;
        return s1;
    }
}h;
int main(){
    int n; scanf("%d",&n);
    int J = 0, O = 0, I = 0, ans = 0;
    h.ins(mp(n,n),0);
    rep(i,1,n){
        char c = getchar();
```

```
            while (c != 'J' && c != 'O' && c != 'I') c = getchar();
            if (c == 'J') J++;
            if (c == 'O') O++;
            if (c == 'I') I++;
            int tmp = h.ins(mp(J - O + n, O - I + n), i);
            if (i - tmp > ans) ans = i - tmp;
        }
        printf("%d\n",ans);
    }
```

4237　　稻草人

习惯上。。还是考虑左上一个点右下一个点的吧。。

考虑对单个点，有哪些在它左上的点可以和它构成满足要求的长方形

这些点显然是随着x的增大单调递增的

是否可以直接对坐标的pair排序，用set维护一个这样的随着x的增大单调递增的类似凸包结构呢？（就叫类凸包吧）

不行。。理由如图

（。）

[。]

。

。

x

其中x是我们考虑的那个点，我们(。）这个点和它是可以组成长方形的，因为［。］在x的后面。。

也就是说我们要维护的是x<A.x y<A.x的类凸包。。。是一个前缀类凸包！

似乎说可以线段树维护前缀类凸包Orz。。并不会

但是更简单的办法是应用cdq分治（和NOI购票类似啦。。）

cdq分治的核心思想是把每个点都受前面所有点影响变成后一半点都受前一半点影响，然后一起处理

这里x<A.x这样的显然就是每个x都受前面所有x的影响

而这里的难点又在于处理x<A.x，那么我们对x进行cdq分治。。

然后我们对点排序，按照y扫描，set/平衡树维护左边的一个类凸包，

对于右边一个点。。只能计入y>=右半边的所有点中在它的左边的最靠下的一个点.y的答案。。

嗯。。这个玩意的set按照y排序比较好。。

等等？按照y排序？那每次不都是从最后插入的吗。。？一个栈就搞定了囧

同样的办法处理"右半边的所有点中在它的左边的最靠下的一个点.y的答案"

每次插入时二分即可

复杂度O(nlog^2n)

由于二分t已经有一个log了。。可以放心使用sort。。。

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define rep(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
const int N = 2 * 100000 + 10;
typedef long long LL;
typedef pair<int, int> node;
#define mp(a,b) make_pair(a,b)
#define x first
#define y second
```

```
node t[N];
bool cmp(const node &a, const node &b){
    return a.y < b.y;
}
LL ans = 0;
node s[N], s2[N], t1[N], T2[N];
int top, t2;
void cdq(int l, int r){
    if (l == r) return;
    int mid = (l + r) >> 1;
    top = t2 = 0;
    int L = l, R = r;
    rep(i,L,R) if (t[i].x <= mid){
        while (top && s[top - 1].x < t[i].x) top--;
        s[top++] = t[i];
    }else{
        while (t2 && s2[t2 - 1].x > t[i].x) t2--;
        int my = (t2 == 0) ? -1 : s2[t2 - 1].y;
        s2[t2++] = t[i];

        int l = -1, r = top;//[l,r)
        while (l + 1 < r){
            int mid = (l + r) >> 1;
            if (s[mid].y < my) l = mid; else r = mid;
        }

        ans += top - 1 - l;
    }
    int l1 = 0, l2 = 0;
    rep(i,L,R) if (t[i].x <= mid) t1[++l1] = t[i]; else T2[++l2] = t[i];
    int l3 = L;
    rep(i,1,l1) t[l3++] = t1[i];
    rep(i,1,l2) t[l3++] = T2[i];
    cdq(L, mid), cdq(mid + 1, R);
}
int n;
void init(){
```

```
    scanf("%d",&n);
    rep(i,1,n) {
        int a, b;
        scanf("%d%d",&a,&b);
        t[i] = mp(a,b);
    }
    sort(t + 1, t + n + 1);
    rep(i,1,n) t[i].x = i;
    sort(t + 1, t + n + 1, cmp);
    rep(i,1,n) t[i].y = i;
}
int main(){
    init();
    cdq(1, n);
    printf("%lld\n",ans);
}
```

然后散步的时候想到了怎么优化到O(nlogn)

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define rep(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
const int N = 2 * 100000 + 10;
typedef long long LL;
typedef pair<int, int> node;
#define mp(a,b) make_pair(a,b)
#define x first
#define y second
node t[N];
bool cmp(const node &a, const node &b){
    return a.y < b.y;
}
LL ans = 0;
node s[N], s2[N], t1[N], T2[N];
```

```
int top, t2;
int f[N], id[N];
inline int find(int x){
    return (f[x] == x || f[x] == -1) ? f[x] : f[x] = find(f[x]);
}
void cdq(int l, int r){
    if (l == r) return;
    int mid = (l + r) >> 1;
    top = t2 = 0;
    int L = l, R = r;
    rep(i,L,R) f[i] = i;
    rep(i,L,R) if (t[i].x <= mid){
        while (top && s[top - 1].x < t[i].x) top--, f[s[top].x] = top ?
s[top - 1].x : -1;
        s[top++] = t[i]; id[t[i].x] = top - 1;
    }else{
        while (t2 && s2[t2 - 1].x > t[i].x) t2--;

        int tt = t2 == 0 ? -1 : find(s2[t2 - 1].x);
        tt = tt == -1 ? -1 : id[tt];

        s2[t2++] = t[i];

        f[t[i].x] = top ? s[top - 1].x : -1;

        ans += top - 1 - tt;
    }
    int l1 = 0, l2 = 0;
    rep(i,L,R) if (t[i].x <= mid) t1[++l1] = t[i]; else T2[++l2] = t[i];
    int l3 = L;
    rep(i,1,l1) t[l3++] = t1[i];
    rep(i,1,l2) t[l3++] = T2[i];
    cdq(L, mid), cdq(mid + 1, R);
}
int n;
void init(){
    scanf("%d",&n);
```

```
        rep(i,1,n) {
            int a, b;
            scanf("%d%d",&a,&b);
            t[i] = mp(a,b);
        }
        sort(t + 1, t + n + 1);
        rep(i,1,n) t[i].x = i;
        sort(t + 1, t + n + 1, cmp);
        rep(i,1,n) t[i].y = i;
    }
    int main(){
        init();
        cdq(1, n);
        printf("%lld\n",ans);
    }
```

这类题目无论一维二维都是一样。。。"每个点都受前面所有点影响变成后一半点都受前一半点影响"。。尤其是求这种前缀凸包里面特别好用。。。

4238 电压

简化一下题意。。求的是哪些边满足把u-v两个点合并之后，原图变成一个二分图。。。

考虑原图即二分图的显然是桥的个数

受这个启发取一棵生成树，考虑每条非树边对答案的影响即可

注意dfs树只有返祖边，也就是说每条非树边都是返祖边。。lca即一个端点，覆盖的时候直接通过树上差分打标记实现。。

一个点事合法的必须满足:1.fa=0 2.good = 0 3. bad = bad_cnt 才是可选的

（顺便说一句，这个办法也可以用来求无向图的桥

```
    #include <iostream>
    #include <cstdio>
    #define rep(i,a,b) for(int i=a;i<=b;i++)
```

```
using namespace std;
const int N = 100000 + 10, M = 200000 * 2 + 10;
int n, m, a[M], b[M];
struct edge{
    int to, pre;
}e[M];
int u[N], l = 1;
#define v e[i].to
#define reg(i,a) for(int i=u[a];i;i=e[i].pre)
void ins(int a, int b){
    e[++l] = (edge) {b, u[a]}, u[a] = l;
}
int dep[N], good_cnt = 0, bad_cnt = 0, vis[N], good[N], bad[N];
void dfs(int x, int fa){
    vis[x] = 1;good[x] = bad[x] = 0;
    reg(i,x) if (i ^ fa ^ 1){
        if (!vis[v])
            dep[v] = dep[x] + 1, dfs(v, i), good[x] += good[v], bad[x] +=
bad[v];
        else if (dep[v] <= dep[x]){
            if ((dep[x] ^ dep[v]) & 1){
                good_cnt++;
                good[x]++, good[v]--;
            }else{
                bad_cnt++;
                bad[x]++, bad[v]--;
            }
        }
    }
}
int t[N];
int main(){
    scanf("%d%d",&n,&m);
    rep(i,1,m) {
        scanf("%d%d",&a[i],&b[i]);
        ins(a[i],b[i]);
        if (a[i] != b[i]) ins(b[i],a[i]);
```

```
    }
    rep(i,1,n) if (!vis[i]) dep[i] = 0, dfs(i, 0), t[i] = 1; else t[i] =
0;
    int ans = 0;
    rep(i,1,n) if (!t[i] && !good[i] && bad[i] == bad_cnt) ans++;
    if (bad_cnt == 1) ans++;
    printf("%d\n",ans);
    return 0;
}
```

4239: 巴士走读

显然等价于求出我们想乘坐每辆巴士的最迟出发时间，对于起点在1的巴士。。显然就是巴士的出发时间，然后每辆巴士由在它出发之前到达出发点的每辆巴士更新。。复杂度不对？并不。。直接对出发点的到达车次前缀维护最迟出发时间嘛（毕竟都是'从1在yi之前到这个站的最迟出发时间"是一个简单的max）

所以就是按照<到达地,到达时间,编号>排序之后，按照<到达时间,编号>顺序更新每辆车即可（到达时间先于它的出发时间的肯定更早于出发时间）

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define max(a,b) ((a) < (b) ? (b) : (a))
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define dep(i,a,b) for(int i=a;i>=b;i--)
#define get(L) ((l < r && y[l] <= L) ? s[l] : -1)
#define upd() s[x] = (head[b[i].b] == x) ? f[x] : max(s[x - 1], f[x]);
#define binary(a,L) for(l=head[a], r=head[a + 1]; l + 1 < r; mid = (l + r)
>> 1, (y[mid] <= L ? (l = mid) : (r = mid)))
using namespace std;
const int M = 300000 + 10;
struct bus{
    int a, b, x, y, num;
}b[M];
bool cmp1(const bus &a, const bus &b){
    return a.b < b.b || (a.b == b.b && a.y < b.y) || (a.a == b.b && a.y ==
```

```
b.y && a.num < b.num);
}
bool cmp2(const bus &a, const bus &b){
    return a.y < b.y ||  (a.y == b.y && a.num < b.num);
}
int y[M], f[M], s[M], head[M];
int main(){
    int n, m; scanf("%d%d",&n,&m);
    rep(i,1,m) scanf("%d%d%d%d",&b[i].a, &b[i].b, &b[i].x, &b[i].y),
b[i].num = i;

    sort(b + 1, b + m + 1, cmp1);
    rep(i,1,m) b[i].num = i, y[i] = b[i].y;
    b[0].b = 0;
    rep(i,1,m) if (b[i].b != b[i - 1].b) head[b[i].b] = i;
    head[n + 1] = m + 1;
    dep(i,n,1) if (head[i] == 0) head[i] = head[i + 1];

    sort(b + 1, b + m + 1, cmp2);
    rep(i,1,m) f[i] = s[i] = -1;
    rep(i,1,m){
        int x = b[i].num;
        if (b[i].a == 1) {
            f[x] = b[i].x;
            if (head[b[i].b] == x) s[x] = f[x]; else s[x] = max(s[x - 1],
f[x]);
        }else{
            int l,r,mid;//[l,r)
            binary(b[i].a, b[i].x);
            f[x] = get(b[i].x);
            upd();
        }
    }
    int Q; scanf("%d",&Q);
    rep(i,1,Q) {
        int L; scanf("%d",&L);
        int l, r, mid;
```

```
            binary(n, L);
            printf("%d\n",get(L));
        }
        return 0;
    }
```

popoqqq大爷说有这么一种做法：

"将二元组<站点，时间>看做一个点，那么点数是O(m)的

每辆巴士连一条边，每个点向下一个时间连边，然后将所有边反向

枚举终点站的每一个时间，加入队列广搜，得到最晚多久到达1号站点可以在这个时间到达终点站

然后对于每个询问去数组中二分即可 "

感觉这样的按照时间拆点还是很巧妙的。。就是对每个站点的时间离散化之后只有O(m)个时刻，直接在时刻之间连边，表示一个可行则另一个也可行。。然后跑bfs

（其实不反向，倒序枚举起点的每个时间就可以，bfs中每个节点显然只会被访问一次，每次访问n的时候记录一下即可。。。就相当于在时间、站点中对可行的染色floodfill。。

BZOJ 4240 有趣的家庭菜园

考虑结果序列，一定是先递增再递减。。再考虑最小的元素，它肯定在两边的一边，找比较短的一边把它丢过去即可

一个数往左往右交换的代价就是那个方向上比它大的数的个数

BIT维护一下咯。。。一个实现的小trick是，如果先按照h排序然后BIT维护编号就不用离散化了，不过要小心相同的h，相同的h的话，一定有一种方案它们之间没有交换。。所以这个大于是h严格大于

```
    #include <iostream>
    #include <cstdio>
    #include <algorithm>
    #define min(a,b) ((a) < (b) ? (a) : (b))
```

```cpp
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define dep(i,a,b) for(int i=a;i>=b;i--)
using namespace std;
const int N = 300000 + 10;
typedef pair<int, int> node;
#define h first
#define num second
int s[N], n;
void add(int x){
    for(; x <= n; x += (-x) & x) s[x]++;
}
int sum(int x){
    int ans = 0;
    for(; x; x -= (-x) & x) ans += s[x];
    return ans;
}
node a[N];
typedef long long LL;
int main(){
    LL ans = 0;
    scanf("%d",&n);
    rep(i,1,n) scanf("%d",&a[i].h), a[i].h = -a[i].h, a[i].num = i;
    sort(a + 1, a + n + 1);
    a[n + 1].h = 0;
    int tt = 0;
    rep(i,1,n){
        int t = sum(a[i].num);
        ans += min(t, tt - t);
        if (a[i + 1].h != a[i].h) {
            dep(j,i,1) {if (a[j].h == a[i].h) add(a[j].num); else break;}
            tt = i;
        }
    }
    printf("%lld\n",ans);
    return 0;
}
```

4241: 历史研究

如果我们使用莫队算法。。那么就是单点修改，查询总体的max。。

注意这里查询max是总体的max。。我们要利用好这个性质。。

比如。。我们可以用一个priority_queue来维护这个max。。。但这只是卡常数。。

另外一个在总体max上用到的就是。。值域线段树。。然后使劲往右走。。（当然这里没用。。但是。。。值域块状数组可以啊！

有两个优化的办法。。。

一个是分块而不莫队。。

这题和区间众数类似（如果每个每次加重要程度改成加1就是区间众数），自然可以分成sqrt(n)段，预处理[i..j]段的答案和每个数出现了多少次，然后对零碎的更新

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#define max(a,b) ((a)>(b)?(a):(b))
#define rep(i,a,b) for(int i=a;i<=b;i++)
using namespace std;
const int N = 100015, M = 350;
typedef long long LL;
LL f[M][M];
int a[N], bel[N], cnt[M][N], b[N], n, q, l, L[M], R[M], c[N];
int find(int x){
    int l = 1, r = n + 1;//[l,r)
    while (l + 1 < r){
        int mid = (l + r) >> 1;
        if (b[mid] <= x) l = mid; else r = mid;
    }
    return l;
}
```

```
int main(){
    scanf("%d%d",&n,&q);
    rep(i,1,n) scanf("%d",&a[i]), b[i] = a[i];
    sort(b + 1, b + n + 1);
    rep(i,1,n) a[i] = find(a[i]);

    int len = sqrt(n), m;
    if (len == 0) len++;
    rep(i,1,n) {
        bel[i] = (i - 1) / len + 1;
        if (i % len == 1) L[bel[i]] = i;
        if (i % len == 0) R[bel[i]] = i;
    }
    m = bel[n], R[m] = n;


    rep(i,1,n) cnt[bel[i]][a[i]]++;
    rep(i,2,m) rep(j,1,n) cnt[i][j] += cnt[i - 1][j];

    rep(i,1,m)
        rep(j,i,m){
            f[i][j] = f[i][j - 1];
            rep(k,L[j],R[j]){
                int t = a[k];
                LL t1 = (long long)(cnt[j][t] - cnt[i - 1][t]) * b[t];
                f[i][j] = max(t1, f[i][j]);
            }
        }


    rep(i,1,q){
        int l, r; scanf("%d%d",&l,&r);
        int l1 = bel[l], r1 = bel[r];
        LL ans = 0;
        if (l1 == r1){
            rep(i,l,r) {
                int t = a[i];
```

```
                c[t]++, ans = max(1LL * c[t] * b[t], ans);
            }
            printf("%lld\n",ans);
            rep(i,l,r) c[a[i]]--;
        }else{
            r1--;
            ans = f[l1 + 1][r1];
            rep(i,l,R[l1]) {
                int t = a[i];
                c[t]++, ans = max(1LL * (c[t] + cnt[r1][t] - cnt[l1][t]) *
b[t], ans);
            }
            rep(i,R[r1] + 1, r){
                int t = a[i];
                c[t]++, ans = max(1LL * (c[t] + cnt[r1][t] - cnt[l1][t]) *
b[t], ans);
            }
            printf("%lld\n",ans);
            rep(i,l,R[l1]) c[a[i]]--;
            rep(i,R[r1] + 1,r) c[a[i]]--;
        }
    }
    return 0;
}
```

一个是莫队的基础上优化。。

莫队算法，考虑如何快速维护最大的重要度。

考虑到答案一共只有O(n)种本质不同的取值，于是可以先通过O(nlogn)的排序处理出这些值的大小关系，并将这些值离散化，同时对每种事件的每个出现次数维护两个指针pre和nxt，分别表示出现次数减少或增加一后是第几小。

然后对这些取值进行分块，每块维护该块内有哪些值出现过。

显然，修改是O(1)的。

查询的时候从后往前扫描，遇到第一个有数字的块就在该块内从后往前扫描，时间复杂度O(√n)。

于是总的复杂度为O((n+q)√n)。

这。。。就是值域块状数组。。"然后使劲往右走。。”。。。。。。。。。

这里需要明确的是。。假设单次修改复杂度为

$$O(f(n))$$

莫队算法中是保证了所有修改的总复杂度是

$$O\left((m\sqrt{n}+n\sqrt{n})f(n)\right)$$

询问实际上只需要O(m)次，（修改多是因为两个区间间的转移需要N^0.5次修改（当然是均摊））

假设单次询问复杂度是

$$O(g(n))$$

那么总询问复杂度是

$$O(m*g(n))$$

所以啊。。这里用块状数组这种询问O(sqrt(n))修改O(1)的东西十分合适

然而TLE了。。TAT自带大常数。。怎么办。。。

```
//TLE TAT
#include <cmath>
#include <cstdio>
#include <iostream>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;
typedef long long LL;
const int N = 100000 + 10, M = 950;
int a[N], t[N], pos[N], n, cl;
LL b[N], c[N];
inline int find(LL x){
    int l = 1, r = cl + 1;
    while (l + 1 < r){
        int mid = (l + r) >> 1;
        if (c[mid] <= x) l = mid; else r = mid;
    }
    return l;
}

struct qry{
    int a, b, c, num;
}Q[N];

int ans[N];

inline bool cmp(const qry &a, const qry &b){
    return (a.c < b.c) || (a.c == b.c && a.b < b.b);
}

int bel[N], sum[M], s[N], block_cnt, L[M], R[M];
inline void add(int x){
    sum[bel[x]]++;
    s[x]++;
}
```

```
inline void del(int x){
    sum[bel[x]]--;
    s[x]--;
}

#define dep(i,a,b) for(int i=a;i>=b;i--)
inline int qry(){
    dep(i,block_cnt,1) if (sum[i])
        dep(j,R[i],L[i]) if (s[j]) return j;
}

int cnt[N];
inline void op(int a, int x){
    if (cnt[a] > 0) del(b[pos[a] + cnt[a]]);
    cnt[a]+=x;
    if (cnt[a] > 0) add(b[pos[a] + cnt[a]]);
}
inline int read(){
    char c = getchar();
    int x = 0, f = 1;
    while (c < '0' || c > '9') {if (c == '-') f = -1; c = getchar();}
    while (c >= '0' && c <= '9') x = x * 10 + c - '0', c = getchar();
    return x * f;
}
int main(){
    int q; scanf("%d%d",&n,&q);
    rep(i,1,n) a[i] = read(), b[i] = a[i];
    sort(b + 1, b + n + 1);
    rep(i,1,n) if (b[i] == b[i - 1]) c[i] = c[i - 1] + b[i]; else c[i] =
b[i], t[i] = 1;
    rep(i,1,n) b[i] = c[i];

    sort(c + 1, c + n + 1);
    cl = unique(c + 1, c + n + 1) - c - 1;
    rep(i,1,n) a[i] = find(a[i]);

    rep(i,1,n) {
```

```
        b[i] = find(b[i]);
        if (t[i]) pos[b[i]] = i - 1;
    }

    int m = sqrt(cl) + 1;
    rep(i,1,cl) {
        bel[i] = (i - 1) / m + 1;
        if (i % m == 1) L[bel[i]] = i;
        if (i % m == 0) R[bel[i]] = i;
    }
    R[bel[cl]] = cl;
    block_cnt = bel[cl];

    m = sqrt(q) + 1;
    rep(i,1,q) scanf("%d%d",&Q[i].a, &Q[i].b), Q[i].c = Q[i].a / m,
Q[i].num = i;
    sort(Q + 1, Q + q + 1, cmp);

    int l = 1, r = 0;
    rep(i,1,q){
        int L = Q[i].a, R = Q[i].b;
        while (r < R) r++, op(a[r], 1);
        while (r > R) op(a[r], -1), r--;
        while (l < L) op(a[l], -1), l++;
        while (l > L) l--, op(a[l], 1);
        ans[Q[i].num] = qry();
    }

    rep(i,1,q) printf("%lld\n",c[ans[i]]);
    return 0;
}
```

4242: 水壶

给定一个图，q个询问，每次问你两点间最大权值最小的路径。。。

求一个最小生成树然后树上倍增即可

一个问题是wh稍微有点大，这个不能*log

求个虚树可以。。但是太麻烦

Po姐说，"把每栋建筑物扔进队列跑BFS，对于每块空地记录这块空地是被哪栋建筑物搜到的，以及距离是多少 "

我们考虑这个网格图（每条边的边权都是1）的最小生成树。。

显然直接把边按任意顺序添加进去就可以。。。。

我们的做法就是。。取一个bfs形成的图出来。。由于每个点的第一次访问只有一次，只有第一次访问才会对四周连边，所以边数是O(4wh)的。。

为什么是对的呢。。。

考虑有n个点连到一个空格。。这可以产生n^2条边。。但是有一堆环。。。。

我们讨论每个三元环。。。A—X < B—X < C—X

A—B = A—X + B—X

A—C = A—X + C—X

B—C= B—X+ C—X > A—C > A—B

所以这个三元环上B---C是没有必要建出来的（环上最大边）。。。只要把所有到这个空格的建筑的边都连到离这个空格最近的建筑上即可。。

同时显然。。如果A - —X— —Y—C B---X---Y—C 其中 A—X<B—X, 并且AB先在X点会和的

那么BC > AC 并且 BC >AB ， 所以说这里就不用建BC了

所以如果一个点已经访问，就直接第一次连到访问它的那个点，不用再继续拓展了

一道sb题纠结这么多TAT。。。感觉自己药丸啊。。

```
#include <iostream>
```

```cpp
#include <cstdio>
#include <algorithm>
#include <vector>
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define dep(i,a,b) for(int i=a;i>=b;i--)
using namespace std;
const int N = 2000 + 10, M = 200000 + 10;
char mp[N][N];
typedef pair<int, int> node;
#define x first
#define y second
#define mp(a,b) make_pair(a,b)
int dx[5], dy[5];
node bd[M], q[N * N], v[N][N];
int r = 0;
void init(){
    dx[0] = 0, dy[0] = 1;
    dx[1] = 1, dy[1] = 0;
    dx[2] = 0, dy[2] = -1;
    dx[3] = -1,dy[3] = 0;
}

vector<node> edge[N * N];
#define pb(a) push_back(a)

int n, m, p, Q;
void bfs(){
    int l = 0;
    while (l < r){
        node a = q[l++];
        rep(i,0,3){
            node b = mp(a.x + dx[i], a.y + dy[i]);
            if (b.x && b.x <= n && b.y && b.y <= m && mp[b.x][b.y] != '#')
{
                node &va = v[a.x][a.y], &vb = v[b.x][b.y];
                if (vb == mp(0,0))
                    vb = mp(va.x + 1, va.y), q[r++] = b;
```

```
                else if (va.y != vb.y)
                edge[va.x + vb.x].pb(mp(va.y, vb.y));//be careful with the
definition of the distance
            }
        }
    }
}

struct edg{
    int to, pre, c;
}e[M * 2];
int u[M], l = 0;
void ins(int a, int b, int c){
    e[++l] = (edg){b, u[a], c}, u[a] = l;
}
void insert(int a, int b, int c){
    ins(a,b,c), ins(b,a,c);
}
#define v e[i].to
#define ec e[i].c
#define reg(i,a) for(int i=u[a];i;i=e[i].pre)

int f[M], fa[M], vis[M], R[M];
int find(int x){
    return f[x] == x ? x : f[x] = find(f[x]);
}
void kruskal(){
    rep(i,1,p) f[i] = i;
    rep(i,0,n * m)
        for(vector<node>::iterator it = edge[i].begin();it !=
edge[i].end();it++){
            int fa = find((*it).x), fb = find((*it).y);
            if (fa != fb) insert((*it).x,  (*it).y, i), f[fa] = fb;
        }
}

int dep[M];
```

```
void dfs(int x){
    vis[x] = 1;
    reg(i,x) if (!vis[v]) dep[v] = dep[x] + 1, fa[v] = x, R[v] = ec,
dfs(v);
}

int g[M][20], g1[M][20];

#define max(a,b) ((a) < (b) ? (b) : (a))
int qry(int a, int b){
    if (find(a) != find(b)) return -1;
    if (dep[a] < dep[b]) swap(a,b);
    int ans = 0;
    dep(i,19,0) if (g[a][i] && dep[g[a][i]] >= dep[b]) ans = max(ans,
g1[a][i]), a = g[a][i];
    dep(i,19,0) if (g[a][i] != g[b][i]) ans = max(ans, g1[a][i]), ans =
max(ans, g1[b][i]), a = g[a][i], b = g[b][i];
    if (a != b) ans = max(ans, R[a]), ans = max(ans, R[b]);
    return ans;
}
#undef v

int main(){
    init();
    scanf("%d%d%d%d",&n,&m,&p,&Q);
    rep(i,1,n) scanf("%s",mp[i] + 1);
    int a, b;
    rep(i,1,p) scanf("%d%d",&a,&b), bd[i].x = a, bd[i].y = b, q[r++] =
mp(a,b), v[a][b] = mp(0, i);
    bfs();
    kruskal();
    rep(i,1,p) if (vis[i] == 0) dep[i] = 1, dfs(i);
    rep(i,1,p) if (fa[i]) g[i][0] = fa[i], g1[i][0] = R[i];
    rep(j,1,19)
        rep(i,1,p)
            g[i][j] = g[g[i][j - 1]][j - 1], g1[i][j] = max(g1[g[i][j -
1]][j - 1], g1[i][j - 1]);
```

```
    rep(i,1,Q){
        scanf("%d%d",&a,&b);
        printf("%d\n",qry(a,b));
    }
}
```

4243: 交朋友

对于双向边连接的一个连通分量。。显然可以通过任意多次会议使得它们两两之间有双向边。。把这样一个分量并到一起

剩下有向边的计数检查所有给定边即可。。。

考虑怎么并。。只需要考虑:

1.有向边和有向边导致的会议

这个直接把每个点连出的边到的所有点并一起即可

2.有向边和无向边导致的会议

这玩意告诉我们。。。只要一个分量有超过两点。。就可以和另外一个分量并起来。。。预处理对已经开过会的点多源BFS即可。。

论分类讨论思想的重要性。。。

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define rep(i,a,b) for(int i = a; i <= b; i++)
using namespace std;

const int N = 200000 + 10;
int f[N], s[N];
int find(int x){
    return f[x] == x ? x : f[x] = find(f[x]);
}
```

```
void Union(int a, int b){
    int fa = find(a), fb = find(b);
    if (fa != fb){
        s[fa] += s[fb];
        f[fb] = fa;
    }
}

int n, m;
struct edge{int to, pre;}e[N * 4];
int u[N], l = 0;
void ins(int a, int b){e[++l] = (edge){b, u[a]}, u[a] = l;}
#define v e[i].to
#define reg(i,a) for(int i=u[a];i;i=e[i].pre)

struct E{int a,b,c;}ed[N * 2];
bool operator < (const E &a, const E &b){
    return (a.a < b.a) || (a.a == b.a && a.b < b.b) || (a.a == b.a && a.b
== b.b && a.c < b.c);
}

bool vis[N];
int q[N];

void bfs(){
    int l = 0, r = 0;
    rep(i,1,n) if (vis[i]) q[r++] = i;
    while (l < r){
        int x = q[l++];
        reg(i,x) if (vis[v]) Union(x, v); else {
            vis[v] = 1, q[r++] = v;
            Union(x, v);
        }
    }
}
```

```
int main(){
    scanf("%d%d",&n,&m);
    int a, b;
    rep(i,1,n) s[i] = 1, f[i] = i, vis[i] = 0;
    rep(i,1,m) {
        scanf("%d%d",&a,&b);
        if (a < b) ed[i] = (E){a, b, 0}; else ed[i] = (E){b, a, 1};
    }
    sort(ed + 1, ed + m + 1);

    rep(i,1,m){
        if (i < m && ed[i].a == ed[i + 1].a && ed[i].b == ed[i + 1].b)
            Union(ed[i].a, ed[i].b), vis[ed[i].a] = vis[ed[i].b] = 1;
        if (ed[i].c == 1) ins(ed[i].b, ed[i].a); else ins(ed[i].a,
ed[i].b);
    }
    rep(x,1,n){
        int flag = 0, cnt = 0;
        reg(i,x){
            if (flag) Union(v, flag); else flag = v;
            cnt++;
        }
        if (cnt > 1) reg(i,x) vis[v] = 1;
    }

    bfs();
    long long ans = 0;
    rep(i,1,n) if (f[i] == i) ans += 1LL * s[i] * (s[i] - 1);
    rep(i,1,m){
        int a = ed[i].a, b = ed[i].b;
        if (find(a) != find(b)) ans++;
    }
    printf("%lld\n",ans);
    return 0;
}
```

4247: 挂饰

考虑一种挂的方案。。肯定是每次挂Al >= 1的啦。。我们可以之间把Ai排序，背包搞定

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define rep(i,a,b) for(int i=a;i<=b;i++)
#define dep(i,a,b) for(int i=a;i>=b;i--)
using namespace std;
const int N = 2000 + 10, INF = 2000000010;
int a[N], b[N], f[N][N], q[N], Q;
int main(){
    int n; scanf("%d",&n);
    int l = 0, Q = 0;
    rep(i,1,n){
        int x, y;
        scanf("%d%d",&x,&y); if (x == 0) q[++Q] = y; else a[++l] = x, b[l]
= y;
    }
    sort(q + 1, q + Q + 1);
    f[0][0] = -INF, f[0][1] = 0; rep(i,2,n) f[0][i] = -INF;
    rep(i,1,l){
        rep(j,0,n) f[i][j] = f[i - 1][j];
        rep(j,1,n)
            if (f[i - 1][j] != -INF){
                int k = j - 1 + a[i]; if (k > n) k = n;
                f[i][k] = max(f[i - 1][j] + b[i], f[i][k]);
            }
    }
    int j = Q, aj = 0, ans = 0;
    rep(i,0,n){
        if (j && i && q[j] > 0) aj += q[j], j--;
        if (f[l][i] != -INF) ans = max(ans, aj + f[l][i]);
    }
    printf("%d\n",ans);
}
```