P1954 [NOI2010] 航空管制 题解

返回题目

本题目已不接受新题解提交

当题目的题解数量、做题思路已足够丰富，题目过于简单，或处于月赛保护期时，题解提交入口会被关闭。

不要把题解发布在题目讨论区。

题解仅供学习参考使用

抄袭、复制题解，以达到刷 AC 率/AC 数量或其他目的的行为，在洛谷是严格禁止的。

洛谷非常重视**学术诚信**。此类行为将会导致您成为**作弊者**。 具体细则请查看洛谷社区规则。

提交题解前请务必阅读《洛谷主题库题解规范》。

8 篇题解 默认排序 按时间排序

SBofGaySchool 创建时间：2020-08-12 23:59:51 在 Ta 的博客查看

蒟蒻人生中的第一篇题解。

发现大佬们的题解都是用堆来做的贪心，那我就来一篇不用堆，纯 DFS 的题解吧。本题解极其详细，保证各位看官看完即看懂。

1. 朴素思路的产生

考虑题目中的两种条件：

- 航班之间的依赖顺序；
- 航班起飞的最晚时间；

如果只有第一种，那么是一道标准的**拓扑排序**问题；如果只有第二种条件，那么是一道标准的**贪心**问题。所以我们很自然地往**拓扑排序和贪心相结合**的思路上想。

2. 更新航班的 k 值

每个航班都有一个最晚起飞时间，即 k 值。这个值由第二种条件给出。但事实上，这个 k 值还受到第一种条件的约束。显然，若航班 B 依赖于航班 A（即 $A \to B$），则必有 $k[A] \leq k[B]-1$，所以航班 A 的真实 k 值可由如下表达式求出：$k[A]=min($题目中给出的 $k[A]$, 所有依赖于 A 的航班的 k 值 $-1)$。

我们通过一次 DFS 即可求解（实际上是一次 DAG 上的动态规划）。

3. 第一问的解

既然所有依赖关系中，后飞的航班 k 值现在一定大于先飞的航班，那么我们很容易想到一种合法安排：根据 k 值从大到小，从后往前安排航班。**即先安排 k 值为 n 的航班，再安排 k 值为 $n-1$ 的航班.....最后安排 k 值为 1 的航班。**

这样做一定是合法的。由于后飞的航班 k 值一定大于先飞的航班，所以所有依赖关系中，后飞的航班一定会安排在先飞的航班之后。故第一种条件一定被满足。

用反证法可证第二种条件一定被满足：若某航班不符合条件二，设其 k 值为 p，被安排的时间为 q，有 $p<q$。由于 k 值是从后往前排的，则该航班与之前所有航班（共 q 个）的 k 值都 $\leq p$；即有 q 个航班 k 值 $\leq p$。所以，我们要在 p 时间内起飞 q 架航班，而 $p<q$，与一定存在解不符。

实际上，这么做即为按照 k 值从小到大输出所有航班。此即第一问所求的一种可行解。

4. 第二问的解

第一问的做法实际上是从后往前，尽量让每一架航班都尽量晚飞。让航班 A 早飞，即让 A 和 A 的所有祖先（依赖关系中早飞的航班）尽量早飞，这等价于让其他航班尽量晚飞。

我们可以通过对反图进行一次 DFS，标记出 A 和 A 的祖先，然后仿照第一问，在不安排 A 和 A 的祖先的情况下，根据 k 值从大到小，从后往前安排其他航班。

由于其他航班是从后往前安排的，即**这些航班占据了可能的、最晚的起飞时间**。那么**空出来的起飞时间一定是可能情况下最早的**。而 A 最早的起飞时间，就是这些空出来的起飞时间中**最后一个**（其余空出来的时间必须安排 A 的祖先）。我们在从后往前安排其他航班的过程中即可顺便求解。

5. 时间复杂度

由于本做法对于每个节点都求了一次 DFS，故时间复杂度为 $O(n(m+n))$，不开 $O2$ 耗时 $311ms$，较为高效。

6. 代码实现

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
#define MAXN 2005
#define MAXM 10005
int n, m;
int k[MAXN];
struct Edge
{
    int v, nxt;
} e[MAXM << 1];
int head[MAXN], rhead[MAXN], cnt = 1;
// 建图
void add(int u, int v)
{
    e[cnt].v = v;
    e[cnt].nxt = head[u];
    head[u] = cnt++;
    // 顺便存反边，方便后续反图上的DFS
    e[cnt].v = u;
    e[cnt].nxt = rhead[v];
    rhead[v] = cnt++;
}
int vis[MAXN], rvis[MAXN];
int num[MAXN][MAXN], ccnt[MAXN];
// DFS求航班真实k值
int dfs(int cur)
{
    if (vis[cur])
        return k[cur];
    vis[cur] = 1;
    for (int i = head[cur]; i; i = e[i].nxt)
    {
        int res = dfs(e[i].v);
        if (k[cur] > res - 1)
            k[cur] = res - 1;
    }
    // 将航班放到对应k值的航班集合中
    num[k[cur]][ccnt[k[cur]]++] = cur;
    return k[cur];
}
// 反图DFS标记航班及其祖先
void rdfs(int cur)
{
    rvis[cur] = 1;
    for (int i = rhead[cur]; i; i = e[i].nxt)
        if (!rvis[e[i].v])
            rdfs(e[i].v);
}
int main()
{
    scanf("%d %d", &n, &m);
    for (int i = 1; i <= n; i++)
        scanf("%d", &k[i]);
    while (m--)
    {
        int u, v;
        scanf("%d %d", &u, &v);
        add(u, v);
```

```
    }
    // 求所有航班真实k值
    for (int i = 1; i <= n; i++)
        dfs(i);
    // 按照k值从小到大，从前往后输出所有航班
    for (int i = 1; i <= n; i++)
        for (int j = 0; j < ccnt[i]; j++)
            printf("%d ", num[i][j]);
    printf("\n");
    // 求解每个航班的最早起飞时间
    for (int i = 1; i <= n; i++)
    {
        // 反图DFS标记祖先
        memset(rvis, 0, sizeof(rvis));
        rdfs(i);
        // p为当前所考虑的起飞时间
        int p = n;
        // 按照航班k值从大到小，从后往前安排所有其他航班
        for (int j = n; j >= 1; j--)
        {
            // 从后往前遇到的第一个无法安排其他航班的起飞时间
            // 即为空出来的最后一个起飞时间
            // 此即为所求
            if (p > j)
                break;
            for (int t = 0; t < ccnt[j]; t++)
                if (!rvis[num[j][t]])
                    p--;
        }
        // 输出航班i的最早起飞时间
        printf("%d ", p);
    }
    printf("\n");
    return 0;
}
```


Solystic 创建时间：2021-09-09 21:44:52 在 Ta 的博客查看

简单题。

首先看到每一个航班有 deadline，到某一个时刻开始就飞不了了，立刻想到时光倒流，变成每一个航班某一个时刻开始可以飞，限制变为某一个航班必须在另一个航班之后飞。

那么对于第一问，可以倒着枚举时间，维护一个航班集合表示枚举到当前时间时可以飞的航班，每一次随便抽出一个航班让它起飞，然后用类似拓扑排序的方式更新集合。

对于第二问，要求变为让某一个航班 i 尽量晚飞，思想类似，只是不能随便抽一个，如果航班集合中有不是 i 的航班，则让它飞；否则如果只有 i 可以飞了，就让 i 飞，得到答案。

可以直接使用队列维护集合，做第二问时判一下队尾，就可以做到总复杂度 $O(nm)$。

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

#define getchar() (p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<21,stdin),p1==p2)?EOF:*p1++)
char buf[(1<<21)+5], *p1 = buf, *p2 = buf;

inline int qread() {
    char c = getchar();
```

```
    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;
}

struct Edge {
    int to, nxt;
    Edge() {
        nxt = -1;
    }
};
Edge e[10005];
int n, m, hd[2005], pnt, k[2005], ans[2005], seq[2005], indgr[2005], tmp[2005], que2[2005];
vector <int> v[2005];

inline void AddEdge(int u, int v) {
    e[++pnt].to = v;
    e[pnt].nxt = hd[u];
    hd[u] = pnt;
    indgr[v]++;
}

inline void Read() {
    n = qread(); m = qread();
    for (int i = 1;i <= n;i++) v[k[i] = qread()].push_back(i);
    for (int i = 1;i <= m;i++) {
        int u = qread(), v = qread();
        AddEdge(v, u);
    }
}

inline void Solve1() {
    for (int i = 1;i <= n;i++) tmp[i] = indgr[i] + 1;
    queue <int> que;
    for (int t = n;t >= 1;t--) {
        int siz = v[t].size();
        for (int j = siz - 1;j >= 0;j--) {
            tmp[v[t][j]]--;
            if (!tmp[v[t][j]]) que.push(v[t][j]);
        }
        int u = que.front();
        que.pop();
        seq[t] = u;
        for (int i = hd[u];~i;i = e[i].nxt) {
            tmp[e[i].to]--;
            if (!tmp[e[i].to]) que.push(e[i].to);
        }
    }
    for (int i = 1;i <= n;i++) printf("%d ", seq[i]);
    puts("");
}

inline void Solve2(int u) {
    //printf("u=%d ", u);
    int qhd = 1, qtl = 1;
    for (int i = 1;i <= n;i++) tmp[i] = indgr[i] + 1;
```

```
    for (int t = n;t >= 1;t--) {
        int siz = v[t].size();
        for (int j = siz - 1;j >= 0;j--) {
            tmp[v[t][j]]--;
            if (!tmp[v[t][j]]) {
                que2[qtl++] = v[t][j];
                if (qhd < qtl - 1 && que2[qtl - 2] == u) swap(que2[qtl - 2], que2[qtl - 1]);
            }
        }
        //for (int i = qhd;i < qtl;i++) printf("%d ", que2[i]); puts("");
        int x = que2[qhd++];
        if (x == u) {
            printf("%d ", t);
            break;
        }
        for (int i = hd[x];~i;i = e[i].nxt) {
            tmp[e[i].to]--;
            if (!tmp[e[i].to]) {
                que2[qtl++] = e[i].to;
                if (qhd < qtl - 1 && que2[qtl - 2] == u) swap(que2[qtl - 2], que2[qtl - 1]);
            }
        }
    }
    //puts("");
}

int main() {
    memset(hd, -1, sizeof(hd));
    Read();
    Solve1();
    for (int i = 1;i <= n;i++) Solve2(i);
    return 0;
}
```


gyh20 创建时间：2021-05-18 19:35:52 在 Ta 的博客查看

做题时偶然想到的一个复杂度更优的做法，写一下题解。

首先第一问，建反图做类似于 HNOI2015 菜肴制作的贪心，即可构造一种方案。

第二问大家很多 $O(nm+n^2\log n)$ 的题解都是之后枚举一个点强制不选，贪心的选其他点，重新做一次拓扑排序，直到出现不合法情况说明必须使用当前点，每次拓扑排序是 $O(m+n\log n)$ 的。

可以发现，假设当前枚举点是 x，那么所有在 DAG 上可以由 x 到达的点都永远无法到达，但是对于其他点的相对拓扑序没有改变，可直接枚举原来的拓扑序，再强制删掉由 x 可达的点。

而找到 DAG 上可以由 x 到达的点这一步可以用 bitset 优化，做到总复杂度 $O(n^2+\frac{nm}{\omega})$，是目前的 Rank 1。

```
#include<bits/stdc++.h>
#define re register
using namespace std;
int n,p[2002],head[2002],cnt,m,a[2002],d[2002];
struct edge{int to,next;}e[10002];
bitset<2002>B[2002];
inline void add(re int x,re int y){e[++cnt]=(edge){y,head[x]},head[x]=cnt,++d[y];}
struct node{int x,y;bool operator <(const node a)const{return y<a.y;}};
priority_queue<node>q;
```

```
inline int calc(re int z){
    cnt=0;
    for(re int i=1;i<=n;++i){
        re int x=p[i];
        if(B[x][z])continue;
        ++cnt;
        if(a[x]<=n-cnt)return n-cnt+1;
    }
    return n-cnt;
}
inline int read(){
    re int t=0;re char v=getchar();
    while(v<'0')v=getchar();
    while(v>='0')t=(t<<3)+(t<<1)+v-48,v=getchar();
    return t;
}
signed main(){
    n=read(),m=read();
    for(re int i=1;i<=n;++i)a[i]=read();
    for(re int i=1,x,y;i<=m;++i)x=read(),y=read(),add(y,x);
    for(re int i=1;i<=n;++i)if(!d[i])q.push((node){i,a[i]});cnt=0;
    while(!q.empty()){
        re int x=q.top().x;q.pop();p[++cnt]=x;
        for(re int i=head[x];i;i=e[i].next)if(!(--d[e[i].to]))q.push((node){e[i].to,a[e[i].to]});
    }
    for(re int i=1;i<=n;++i){
        re int x=p[i];
        B[x][x]=1;
        for(re int j=head[x];j;j=e[j].next)B[e[j].to]|=B[x];
    }
    for(re int i=n;i;--i)printf("%d ",p[i]);puts("");
    for(re int i=1;i<=n;++i)printf("%d ",calc(i));
}
```


ywy_c_asm 创建时间：2018-10-23 17:15:29 在 Ta 的博客查看

线段树大法好

首先那个第一问就是建反图然后用堆拓扑排序，这个另外两篇题解已经说过了。这里重点介绍一下第二问的线段树做法。

然后第二问我们要枚举每个点i，我们想让这个点的位置尽量靠前，注意到我们现在已经有了一个可行的序列，我们考虑在这个序列上进行贪心的调整。

很显然，在原DAG上（这里的DAG指的都是反图）拓扑序在i之前的点必须还在i前面，我们干脆就把这个子序列拿出来（这个点是子序列的最后一个），然后一位一位的让他们尽量往前插入剩下的序列中，首先这样贪心是对的，假如在最终构造出的序列中，这个子序列中的某一个可以往前放而且不会造成非法的影响，可以让后面的这个子序列的点都往前移，答案会变得更优。而且这样肯定不会影响DAG拓扑序的关系，因为剩下的点的拓扑序一定不会在他们中的某个点之前（否则就会在这个子序列里了），但是会影响到剩下的点的最靠后位置的限制。

那么我们就要在不影响剩下序列的位置限制的情况下做这个贪心，我们现在已经把这个子序列拿掉了，就暂且让剩下的序列的位置就是现在的位置，显然这一定是合法的因为并没有往后移。如果我们发现一个地方之后的东西的位置并没有到达他们的限制，那么可以在这个地方之后插入一个东西，我们找到最靠前的这样一个位置就行了。

很显然，在合法的前提下每一项的当前位置pos_i都是不超过限制lim_i的，那么我们可以维护最小的$lim_i - pos_i$，如果他是0的话说明有个地方已经达到限制，我们不能在他前面插东西了，所以我们要找到序列上最靠

右的$lim_i - pos_i = 0$的位置之后插就行了（当然要是没有的话就在最前面插就行了），注意到插入一个数会让后面的当前位置$+1$，也就是说$lim_i - pos_i$会-1，这相当于一个区间减，我们用线段树做这个东西就行了，维护一下区间内的最小值与最靠右的最小值的位置，复杂度$O(nm + n^2logn)$（解释一下，我们$O(n)$枚举每个点统计答案，然后$O(m)bfs$找出拓扑序在i之前的所有点也就是这个子序列，再$O(n)$的扫描这个被拿出的子序列，加上线段树之后每次操作$O(logn)$）。注意序列后面的空位置我们把他们设为inf就行了。

上代码~

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<queue>
#define ls(_o) (_o<<1)
#define rs(_o) ((_o<<1)|1)
using namespace std;
inline int get() {//我的垃圾快读
    int n=0;
    char c;
    while((c=getchar())||23333) {
        if(c>='0'&&c<='9')break;
        if(c=='-')goto s;
    }
    n=c-'0';
    while((c=getchar())||23333) {
        if(c>='0'&&c<='9')n=n*10+c-'0';
        else return(n);
    }
s:
    while((c=getchar())||23333) {
        if(c>='0'&&c<='9')n=n*10-c+'0';
        else return(n);
    }
}
typedef struct _n {//这个结构体是用来进行第一问的堆拓扑排序的
    int pt;
    int cnt;//点的限制
    _n(int a,int b) {
        pt=a;
        cnt=b;
    }
    friend bool operator <(const _n &a,const _n &b) {//限制靠后的优先
        return(a.cnt<b.cnt);
    }
} node;
priority_queue<node> que2;//堆（不要问我que1在哪我也不知道……）
int lim[2001];//限制
int ints[2001];//我们得到的可行序列
int tmp[2001];//拿掉那个子序列的序列
int deg[2001];//点的入度
typedef struct _b {
    int dest;
    int nxt;
} bian;
bian memchi[20001];//边
int gn=1;
inline void add(int s,int t) {
    memchi[gn].dest=t;
    memchi[gn].nxt=heads[s];
    heads[s]=gn;
    gn++;
}
int que[2001];//用于bfs的队列
```

```
unsigned char bv[2001];//bfs的时候记录是否已经到达过
int heads[2001];
typedef struct _res {//这个结构体把线段树上维护的最小值与最小值位置封装了起来
    int minn;//最小值
    int pos;//最靠右的最小值位置
    friend _res operator +(const _res &a,const _res &b) {
        if(a.minn<b.minn)return(a);
        return(b);
    }
} res;
int adds[100001];//减法标记
res data[100001];//维护的信息
inline void up(int tree) {//上放
    data[tree]=data[ls(tree)]+data[rs(tree)];
}
inline void down(int tree) {//标记下传
    if(adds[tree]) {
        int cjr=adds[tree];
        adds[tree]=0;
        adds[ls(tree)]+=cjr;
        adds[rs(tree)]+=cjr;
        data[ls(tree)].minn-=cjr;
        data[rs(tree)].minn-=cjr;
    }
}
void add(int rl,int rr,int l,int r,int tree) {//区间减
    if(rl==l&&rr==r) {
        adds[tree]++;
        data[tree].minn--;
        return;
    }
    int mid=(l+r)>>1;
    down(tree);
    if(rl>mid)add(rl,rr,mid+1,r,rs(tree));
    else {
        if(rr<=mid)add(rl,rr,l,mid,ls(tree));
        else {
            add(rl,mid,l,mid,ls(tree));
            add(mid+1,rr,mid+1,r,rs(tree));
        }
    }
    up(tree);
}
res query(int rl,int rr,int l,int r,int tree) {//查询区间最小值是啥，在哪
    if(rl==l&&rr==r)return(data[tree]);
    int mid=(l+r)>>1;
    down(tree);
    if(rl>mid)return(query(rl,rr,mid+1,r,rs(tree)));
    if(rr<=mid)return(query(rl,rr,l,mid,ls(tree)));
    return(query(rl,mid,l,mid,ls(tree))+query(mid+1,rr,mid+1,r,rs(tree)));
}
void build(int l,int r,int tree) {//建树
    adds[tree]=0;//注意清空标记
    if(l==r) {
        data[tree].minn=tmp[l]-l;//维护的是lim_i-pos_i
        data[tree].pos=l;
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,ls(tree));
    build(mid+1,r,rs(tree));
    up(tree);
}
int main() {
```

```
int main() {
    int n=get(),m=get();
    for(register int i=1; i<=n; i++)lim[i]=get();
    for(register int i=1; i<=m; i++) {
        int s=get(),t=get();
        deg[s]++;
        add(t,s);
    }
    register int ptr=n;
    for(register int i=1; i<=n; i++) {
        if(!deg[i])que2.push(_n(i,lim[i]));
    }
    while(!que2.empty()) {//拓扑排序
        int me=que2.top().pt;
        que2.pop();
        ints[ptr]=me;//注意要反着存序列
        ptr--;
        for(register int i=heads[me]; i; i=memchi[i].nxt) {
            deg[memchi[i].dest]--;
            if(!deg[memchi[i].dest])que2.push(_n(memchi[i].dest,lim[memchi[i].dest]));
        }
    }
    for(register int i=1; i<=n; i++)printf("%d ",ints[i]);
    cout<<endl;
    for(register int a=1; a<=n; a++) {//枚举点
        memset(bv,0,sizeof(bv));
        ptr=1;
        bv[a]=1;
        register int head=0,tail=1;
        que[0]=a;
        do {//bfs
            int me=que[head];
            head++;
            for(register int i=fheads[me]; i; i=memchi[i].nxt) {
                if(bv[memchi[i].dest])continue;
                bv[memchi[i].dest]=1;
                que[tail]=memchi[i].dest;
                tail++;
            }
        } while(head<tail);
        for(register int i=1; i<=n; i++) {
            if(!bv[ints[i]])tmp[ptr]=lim[ints[i]],ptr++;//把这个子序列拿出来，重构剩下的序列
        }
        for(register int i=ptr; i<=n; i++)tmp[i]=123456789;//空位置设为inf
        build(0,n,1);
        ptr=1;//维护当前在哪插
        int cnt=0;//统计子序列长度
        for(register int i=1; i<=n; i++) {
            if(bv[ints[i]]) {
                res w=query(ptr,n,0,n,1);
                int dst;
                if(w.minn!=0)dst=ptr;//后面没有达到限制的就接着在这插
                else dst=w.pos+1;//否则在最后一个达到限制的后面插
                add(dst-1,n,0,n,1);//在这里插，把后面都区间减
                ptr=dst;
                cnt++;
            }
        }
        printf("%d ",ptr-1+cnt);
    }
    return(0);
}
```

asuldb 创建时间：2018-10-14 13:33:25 在 Ta 的博客查看

关于拓扑排序的反建图还是一个非常套路的东西

比如说[HNOI2015]菜肴制作

我们希望使得某一个东西在拓扑序中出现的尽可能早，这个时候就可以建出一张反图来，使得这个东西在反图中的拓扑序尽量靠后，从而使得其出现的尽可能地早

这是为什么呢，比如说我们希望1出现的尽可能早，直接在正图上开一个小根堆来进行拓扑排序显然错的

为什么呢，因为这样只能使字典序尽可能小，而不是使得1尽可能靠前

但是我们建出反图来呢，显然反图上按照上述方法来跑的话会使得反向字典序最小，而最小的反向字典序一定是1最靠后的

这道题可以建出一个反图来，从而使得限制条件变成了每一个航班在拓扑序中的位置$> n - k[i]$，于是我们直接用小根堆来维护$n - k[i]$就好了

至于第二问，在每一个点入度为0可以进堆的时候，我们先不让它进去，而是等某一个元素不满足条件的时候，这个时候就是应该进堆了

代码

```
#include<iostream>
#include<queue>
#include<cstdio>
#include<cstring>
#define re register
#define mp std::make_pair
#define maxn 2005
struct E
{
    int v,nxt;
}e[10005];
int head[maxn],r[maxn],d[maxn];
int c[maxn];
int ans[maxn],tot;
int n,m,num;
typedef std::pair<int,int> pii;
std::priority_queue<pii,std::vector<pii>,std::greater<pii> > q;
inline void add_edge(int x,int y)
{
    e[++num].v=y;
    e[num].nxt=head[x];
    head[x]=num;
}
inline int read()
{
    char c=getchar();
    int x=0;
    while(c<'0'||c>'9') c=getchar();
    while(c>='0'&&c<='9')
        x=(x<<3)+(x<<1)+c-48,c=getchar();
    return x;
}
inline int work(int x)
{
    tot=0;
    while(!q.empty()) q.pop();
```

```
    for(re int i=1;i<=n;i++) r[i]=c[i];
    for(re int i=1;i<=n;i++)
    if(!r[i]) q.push(mp(n-d[i],i));
    while(!q.empty())
    {
        int k=q.top().second;
        q.pop();
        if(k==x) continue;
        if(n-tot>d[k]) return n-tot;
        tot++;
        for(re int i=head[k];i;i=e[i].nxt)
        {
            r[e[i].v]--;
            if(!r[e[i].v]) q.push(mp(n-d[e[i].v],e[i].v));
        }
    }
    return n-tot;
}
int main()
{
    n=read(),m=read();
    for(re int i=1;i<=n;i++)
        d[i]=read();
    int x,y;
    for(re int i=1;i<=m;i++)
        x=read(),y=read(),add_edge(y,x),r[x]++;
    for(re int i=1;i<=n;i++) c[i]=r[i];
    for(re int i=1;i<=n;i++)
        if(!r[i]) q.push(mp(n-d[i],i));
    while(!q.empty())
    {
        int k=q.top().second;
        q.pop();
        ans[++tot]=k;
        for(re int i=head[k];i;i=e[i].nxt)
        {
            r[e[i].v]--;
            if(!r[e[i].v]) q.push(mp(n-d[e[i].v],e[i].v));
        }
    }
    for(re int i=n;i;i--)
        printf("%d ",ans[i]);
    puts("");
    for(re int i=1;i<=n;i++)
        printf("%d ",work(i));
    return 0;
}
```


pufanyi 创建时间：2019-02-17 20:23:02 在 Ta 的博客查看

成功抢到 `luogu` 最劣解+ `bzoj` 最劣解（至少我提交的时候是这样）……

题意是给你一张拓扑图，求出一个拓扑序使得第i个点在第k_i个位置之前。先构造一组解，然后输出每个点可以到的最小的位置。

构造一组解很简单，建个反图之后按k_i为关键字排序一下，从小到大一个一个遍历即可。因为如果k_i小的航班都没有开出，开k_i大的显然没有意义。

```
inline void shan(int now) // 遍历now点，但在遍历now点时显然先要遍历now之前（即反图之后）的点
{
    for(int i = ff[now]; i; i = ee[i].nxt)
    {
        int to = ee[i].to; // ee是反图中的边
        if(!vis[to])
            shan(to);
    }
    printf("%d ", now);
    vis[now] = true;
}

int main()
{
    for(int i = 1; i <= n; ++i)
        kkk[i] = mp(k[i], i); // #define mp make_pair
    sort(kkk + 1, kkk + n + 1);
    for(int i = 1; i <= n; ++i)
        if(!vis[kkk[i].second])
            shan(kkk[i].second);
}
```

然后我们发现如果给出的k序列无解，那么输出的序列一定不合法（怎么可能合法？），然后我们发现可以二分。

复杂度？$O(nm\log n)$……显然T，T了4个点。

然后开始$O(松)$卡常。

然后发现它过了。

~~虽然我卡了一年……~~

下面是卡完的代码：

```
// luogu-judger-enable-o2 开O2还是最劣解……
#include <cstdio>
#include <cstring>
#include <queue>
#include <iostream>
#include <algorithm>

using namespace std;

const int maxn = 2005;
const int maxm = 10005;

int n, m;

// 快读快输显然得加

inline char gc()
{
    static char sxd[1 << 16], *sss = sxd, *ttt = sxd;
    return (sss == ttt) && (ttt = (sss = sxd) + fread(sxd, 1, 1 << 16, stdin), sss == ttt) ? EOF : *sss++;
}

#define dd c = gc()
inline int read(int &x)
{
    char dd;
    x = 0;
```

```
    bool f = false;
    for(; !isdigit(c); dd)
    {
        if(c == '-')
            f = true;
        if(c == EOF)
            return -1;
    }
    for(; isdigit(c); dd)
        x = (x << 1) + (x << 3) + (c ^ 48);
    if(f)
        x = -x;
    return 1;
}
#undef dd

inline void write(register int x)
{
    int c[10];
    *c = 0;
    while(x)
    {
        c[++(*c)] = x % 10;
        x /= 10;
    }
    if(!(*c))
        x = 1;
    while(*c)
        putchar(c[(*c)--] | 48);
    putchar(' ');
}

struct pii // 并不知道手打pair会不会快一点
{
    int first, second;
    inline bool operator < (const pii& other) const
    {
        return this->first < other.first;
    }
};

pii kkk[maxn];

struct EDGE
{
    int to, nxt;
} ee[maxm]; // 注意是反图的边，原图的边似乎不用建

int du[maxn];
int first[maxn];
int ff[maxn];
int dz[maxn];

inline void add_edge(register int from, register int to)
{
    static int cnt = 0;
    ee[++cnt].nxt = ff[to];
    ff[to] = cnt;
    ee[cnt].to = from;
}

int k[maxn];
int top;
```

```
int vis[maxn];

inline void shan(register int now) // 回答第一个问题
{
    for(register int i = ff[now]; i; i = ee[i].nxt)
        if(!vis[ee[i].to])
            shan(ee[i].to);
    write(now);
    dz[now] = ++top; // 记录一下每个点至少可以在那个时刻被遍历，这样缩小的二分的范围
    vis[now] = true;
}

pii kx[maxn];
bool viss[maxn];
int X, KK;

inline bool shann(register int now)
{
    for(register int i = ff[now]; i; i = ee[i].nxt)
        if(!viss[ee[i].to])
            if(!shann(ee[i].to))
                return false;
    if(++top > ((now != X) ? k[now] : KK)) // 直接边跑边判断，常数应该会小一点
        return false;
    return viss[now] = true;
}

inline bool pan(register int x, register int kk)
{
    KK = kk;
    register int now = 0;
    for(register int i = 1; i <= n; ++i)
    {
        kx[i] = kkk[i];
        if(kx[i].second == x)
        {
            kx[i].first = kk;
            now = i;
        }
    }
    pii T;
    // 本来下面一段只是一句sort，但那个常数是在太大了，只能插排
    while(now > 1 && kx[now].first < kx[now - 1].first)
    {
        T = kx[now];
        kx[now] = kx[now - 1];
        kx[now - 1] = T;
        now--;
    }
    while(now <= n && kx[now].first > kx[now + 1].first)
    {
        T = kx[now];
        kx[now] = kx[now + 1];
        kx[now + 1] = T;
        now++;
    }
    memset(viss, 0, sizeof(viss));
    top = 0;
    for(register int i = 1; i <= n; ++i)
        if(!viss[kx[i].second])
            if(!shann(kx[i].second))
                return false;
    return true;
```

```
}

inline int solve(const register int x)
{
    X = x;
    register int l = 1, r = dz[x], mid; // 二分，r直接就是上面的dz了，即第一个问题x在那个位置
    while(l < r)
    {
        mid = (l + r) >> 1;
        if(!pan(x, mid))
            l = mid + 1;
        else
            r = mid;
    }
    return r;
}

int main()
{
    read(n), read(m);
    for(register int i = 1; i <= n; ++i)
    {
        read(k[i]);
        kkk[i].first = k[i];
        kkk[i].second = i;
    }
    sort(kkk + 1, kkk + n + 1);
    int f, t;
    for(register int i = 1; i <= m; ++i)
    {
        read(f), read(t);
        add_edge(f, t);
    }
    for(register int i = 1; i <= n; ++i)
        if(!vis[kkk[i].second])
            shan(kkk[i].second);
    puts("");
    for(register int i = 1; i <= n; ++i)
        write(solve(i));
    return 0;
}
```


Ryo_Yamada 创建时间：2021-09-16 19:33:08 在 Ta 的博客查看

对于限制条件 第 i 架飞机起飞序号 $\leq k_i$，我们怎么看怎么觉得别扭。考虑倒着枚举时间，即时光倒流。两类限制条件就变为了

- 第 i 架飞机在 k_i 之前不得起飞（倒序，使时间 i 在 $i-1$ 前面。）
- a 必须在 b 后起飞。

这样我们就可以一遍拓扑解决任务一。对于任务二贪心选取每次起飞的航班，即为在能不选就不选 i 的情况下进行拓扑。直接 n 遍拓扑求答案即可。

对于起飞的时间限制，可以将入度为零但暂时没有到起飞时间限制的航班编号放到优先队列里，每次弹出可以起飞的航班编号。

时间复杂度 $O(n^2 \log n + nm)$。

Code:

```
def(N, int, 2e3 + 5)

int n, m;
int k[N];
vector<int> e[N];
int deg[N], ans[N];
int Deg[N];
int lst[N];

void topo() {
    priority_queue<pii> q;
    queue<int> Q;
    rep(i, 1, n) if(!deg[i]) {
        if(k[i] == n) Q.push(i);
        else q.push(mp(k[i], i));
    }
    per(i, n, 1) {
        //assert(Q.size());
        while(!q.empty() && q.top().fi >= i) {
            Q.push(q.top().se);
            q.pop();
        }
        int u = Q.front();
        Q.pop();
        ans[i] = u;
        for(int v : e[u]) {
            --deg[v];
            if(!deg[v]) {
                if(k[v] >= i) Q.push(v);
                else q.push(mp(k[v], v));
            }
        }
    }
    rep(i, 1, n) printf("%d%c", ans[i], " \n"[i == n]);
}

void solve(int x) {
    rep(i, 1, n) deg[i] = Deg[i];
    priority_queue<pii> q;
    queue<int> Q;
    rep(i, 1, n) if(!deg[i]) {
        if(k[i] == n) Q.push(i);
        else q.push(mp(k[i], i));
    }
    per(i, n, 1) {
        //assert(Q.size());
        while(!q.empty() && q.top().fi >= i) {
            Q.push(q.top().se);
            q.pop();
        }
        int u = Q.front();
        Q.pop();
        if(u == x) {
            if(!Q.empty()) {
                Q.push(x);
                u = Q.front();
                Q.pop();
            }
            else {
                printf("%d ", i);
                return ;
            }
```

```
        }

        for(int v : e[u]) {
            --deg[v];
            if(!deg[v]) {
                if(k[v] >= i) Q.push(v);
                else q.push(mp(k[v], v));
            }
        }
    }
}

int main() {
    qread(n, m);
    rep(i, 1, n) qread(k[i]);
    rep(i, 1, m) {
        int u, v;
        qread(u, v);
        ++deg[u];
        e[v].pb(u);
    }
    rep(i, 1, n) Deg[i] = deg[i];
    topo();
    rep(i, 1, n) solve(i);
    puts("");
    return 0;
}
```


Sor4 创建时间：2017-06-22 14:40:47 在 Ta 的博客查看

这题的思想是很好的，正图难做的情况下考虑返图。

第一问比较好做，拓扑排序就可以了。

第二问的话，思想非常巧妙，首先因为是反图，那么我们就是要找一个点在返图中最晚什么时候起飞。这样的话，我们首先还是按拓扑排序的思想，当我们考虑一个点的时候，首先不管它(即使它没有度了也不放进去)，然后当什么时候我们发现其它的点已经不满足情况了，那么就是这个点必须要出现的位置，也就是正图中的最早位置。

```
#include <queue>
#include <cstdio>
#include <algorithm>
#define Rep( i , _begin , _end ) for(int i=(_begin),i##_END=(_end);i<=(i##_END);i++)
#define For( i , _begin , _end ) for(int i=(_begin),i##_END=(_end);i!=(i##_END);i++)
#define Lop( i , _begin , _end ) for(int i=(_begin),i##_END=(_end);i>=(i##_END);i--)
#define Dnt( i , _begin , _end ) for(int i=(_begin),i##_END=(_end);i!=(i##_END);i--)
using std :: max;
using std :: min;
const int maxx = 2000 + 25;
const int maxm = 10000 + 25;
typedef std :: pair<int,int> iii;
std :: priority_queue<iii> quq;
int n,m,x,y,z,num,tot,cnt;
int idx[maxx],c[maxx],idx1[maxx],ans[maxx];
int head[maxm],nxt[maxm<<1],to[maxm<<1];
namespace top_sort{
    void Ins(int x,int y){
        to[++num] = y;nxt[num] = head[x];head[x] = num;
```

```
        idx[y] ++;
    }
    int Get(int x){
        while(!quq.empty()) quq.pop();
        Rep( i , 1 , n ) idx1[i] = idx[i];
        Rep( i , 1 , n ) if(i != x && !idx1[i]) quq.push(iii(c[i],i));
        Lop( i , n , 1 ){
            if(quq.empty()) return i;
            iii tmp = quq.top();quq.pop();
            if(tmp.first < i) return i;
            for(int k = head[tmp.second];k;k = nxt[k]){
                idx1[to[k]] --;
                if(to[k] != x && !idx1[to[k]]) quq.push(iii(c[to[k]],to[k]));
            }
        }
    }
    void GG(){
        Rep( i , 1 , n ) idx1[i] = idx[i];
        Rep( i , 1 , n ) if(!idx1[i]) quq.push(iii(c[i],i));
        Lop( i , n , 1 ){
            iii tmp = quq.top();quq.pop();
            ans[i] = tmp.second;
            for(int k = head[tmp.second];k;k = nxt[k]){
                idx1[to[k]] --;
                if(!idx1[to[k]]) quq.push(iii(c[to[k]],to[k]));
            }
        }
    }
}
using namespace top_sort;
int main(){
    scanf("%d%d",&n,&m);
    Rep( i , 1 , n ) scanf("%d",&c[i]);
    while( m-- ){
        scanf("%d%d",&x,&y);
        Ins(y,x);
    }
    GG();
    Rep( i , 1 , n ) printf("%d ",ans[i]);
    putchar(10);
    Rep( i , 1 , n ) printf("%d ",Get(i));
    return 0;
}
```