306-07-关系型容器

Set，Multiset，Iterator

Iterator：迭代器

我们可以发现所谓一些数据结构比如说数组和链表，它们都有一些相似的性质。我们看下面两个例子：

1. 数组：定义数组$int\ a[10]$，第一个元素的指针为a，第二个元素的指针为$a+1$，第三个元素的指针为$a+2$，等等、
2. 链表：对于一个链表$list<int>\ mylist;$，它的储存方式是链式储存，内存里的地址不是连续的，而是分散的，它只能用$next$或者$last$来访问元素。

为了统一这两种储存方式的指针，我们引入了一种更加高级的指针，叫做$iterator$，现在定义一个$iterator$：$std::set<int>::iterator\ iter$，这种指针可以支持以下的操作：

1. $iter++$：将$iter$指向下一个元素的地址
2. $iter--$：将$iter$指向上一个元素的地址
3. $*iter$：获得$iter$指针所指向地址所储存的值

Set / Multiset

Set是指集合，它有集合所拥有的性质：元素的唯一性。而$Multiset$则没有前面所说的性质，会存在形如：$\{0,0,1,1,2\}$这样的集合。

这两种数据结构是默认会进行升序排列，用的是类似于平衡二叉搜索树。

以下是一种使用$iterator$的例子：

```cpp
#include <iostream>
#include <set>
#include <algorithm>

int main() {
    std::set<int> s;
    s.insert(2); s.insert(1); s.insert(10);
    std::set<int>::iterator iter = s.end();
    iter --;
    std::cout << *iter << std::endl; // 10
    iter --;
    std::cout << *iter << std::endl; // 2
    iter --;
    std::cout << *iter << std::endl; // 1
}
```

通过上面的例子，不难发现，$end()$指向的是不存在于set的一个地址，是最后一个元素后的一个地址。

注：

几乎所有set的函数和返回值返回的都是$iterator$。

所有的$lower_bound()$和$upper_bound()$都是遵循左闭右开，即$[a,b)$的形式

举例：

1. 对于集合$\{1,2,3,4,5,6,8,9\}$

   $*lower\_bound(7) = *upper\_bound(7) = 8$
2. 对于数组$\{1,2,3,3,3,3,3,4\}$

   $*upper\_bound(3) - *lower\_bound(3) = count(3) = 5$
3. 对于集合$\{1,2,3,4,5,6,7,8,9\}$

   $lower\_bound(10) = upper\_bound(10) = set.end()$

P1607 [USACO09FEB]庙会班车Fair Shuttle

题目描述

逛逛集市，兑兑奖品，看看节目对农夫约翰来说不算什么，可是他的奶牛们非常缺乏锻炼——如果要逛完一整天的集市，他们一定会筋疲力尽的。所以为了让奶牛们也能愉快地逛集市，约翰准备让奶牛们在集市上以车代步。但是，约翰木有钱，他租来的班车只能在集市上沿直线跑一次，而且只能停靠$N(1 \leq N \leq 20000)$个地点（所有地点都以1到N之间的一个数字来表示）。现在奶牛们分成$K(1 \leq K \leq 50000)$个小组，第i 组有$M_i(1 \leq M_i \leq N)$头奶牛，他们希望从S_i跑到$T_i(1 \leq S_i < T_i \leq N)$。

由于班车容量有限，可能载不下所有想乘车的奶牛们，此时也允许小里的一部分奶牛分开乘坐班车。约翰经过调查得知班车的容量是$C(1 \leq C \leq 100)$，请你帮助约翰计划一个尽可能满足更多奶牛愿望的方案。

题解

对于这道题，很显然是一道贪心，而且我们必须将其考虑为有反悔机制的贪心。首先在每一站，进行一下的判断：

1. 将在车上的奶牛可以下车的下车
2. 让所有在此站点上车的奶牛上车
3. 如果超过数量，将最远目的地的奶牛赶下车

```cpp
#include <iostream>
#include <set>
#include <algorithm>

using namespace std;
typedef long long ll;

const int maxn = 500005;

// 一组牛
struct group {
    ll s, t, m;
    group() {}
    group(ll s, ll t, ll m) : s(s), t(t), m(m) {}
    // set内部按照终点站顺序排序
    friend bool operator < (const group &a, const group &b) {
        return a.t < b.t;
    }
} cows[maxn];
ll k, n, c, sum_on_car, ans;

// 在车上的牛的组
multiset<group> cow_set;

// 按照起点站顺序排序
bool cmp(group a, group b) {
```

```cpp
    if (a.s == b.s) {
        if (a.t == b.t) {return a.m < b.m;}
        else {return a.t < b.t;}
    } else {return a.s < b.s;}
}

int main() {
    // 读
    cin >> k >> n >> c;
    for (int i = 1; i <= k; i ++)
        cin >> cows[i].s >> cows[i].t >> cows[i].m;
    sort(cows + 1, cows + 1 + k, cmp);
    // 从第一站开始遍历上车
    for (int i = 1, j = 0; i <= n; i ++) {
        multiset<group>::iterator begin_iter = cow_set.begin();
        // 到站下车（终点站顺序排序）
        while (begin_iter -> t == i) {
            sum_on_car -= begin_iter -> m;
            cow_set.erase(begin_iter);
            begin_iter = cow_set.begin();
        }
        // 全部上车
        for (int t = j + 1; t <= k && cows[t].s == i; t ++) {
            j ++;
            cow_set.insert(cows[t]);
            sum_on_car += cows[t].m;
            ans += cows[t].m;
        }
        // 人数超标，删最远的牛
        while (sum_on_car > c) {
            ll delta = sum_on_car - c;
            multiset<group>::iterator iter = cow_set.end();
            iter --;
            ll all_cow_farthest = iter -> m;
            cow_set.erase(iter);
            if (delta >= all_cow_farthest) {
                sum_on_car -= all_cow_farthest;
                ans -= all_cow_farthest;
            } else {
                ll cow_tmp_s = iter -> s, cow_tmp_t = iter -> t,
cow_tmp_m = iter -> m;
                cow_set.insert(group(cow_tmp_s, cow_tmp_t,
all_cow_farthest - delta));
                sum_on_car -= delta;
                ans -= delta;
            }
        }
    }
    cout << ans << endl;
```

```
74      return 0;
75  }
```

P2869 [USACO07DEC]美食的食草动物Gourmet Grazers

题目

约翰的奶牛对食物越来越挑剔了。现在，商店有M 份牧草可供出售，奶牛食量很大，每份牧草仅能供一头奶牛食用。第i 份牧草的价格为Pi，口感为Qi。约翰一共有N 头奶牛，他要为每头奶牛订购一份牧草，第i 头奶牛要求它的牧草价格不低于Ai，口感不低于Bi。请问，约翰应该如何为每头奶牛选择牧草，才能让他花的钱最少?

题解

对奶牛的价格要求和草的价格要求进行排序，排序后可以得到如下：

$$(A_1,B_1),(A_2,B_2),\cdots,(A_N,B_N),\ A_i\le A_j,\ 1\le i<j\le N \tag{1}$$
$$(P_1,Q_1),(P_2,Q_2),\cdots,(P_M,Q_M),\ P_i\le P_j,\ 1\le i<j\le M \tag{2}$$

从第一项开始遍历，若(P_1,Q_1)可以使得$A_x<P_1<A_{x+1}$，那么将$(B_1,1),\cdots,(B_x,x)$放入S，因为这样可以将奶牛的需求按照品质排序，每次该牧草所给的奶牛的序号应该为t，其中$Q_t\le Q_1<Q_{t+1}$，即$t=S.lower_bound(Q_1)-1$，但是我们会发现这存在问题，因为$lower_bound$如果找不到，会定位在比Q_1大的那个数上，所以这里可以使用技巧：将S进行降序排序（原来是升序），只需要将每个数乘上-1即可，取出来的时候也乘-1。对于其后考虑的牧草，可以发现$P_i\ge P_1$（已经排好序了），所以在S中的奶牛只需要关心品质，无需关心价格。

综上所述我们可以发现是需要维护一个储存奶牛的$multiset$，这其中在判断草是否可以用有一个技巧：直接使用迭代器
$std::multiset<int>::iterator\ \ iter=cow\_set.lower\_bound(Q_i)$。对于这个迭代器，由于放入集合中的是降序排列，我们只需要操作迭代器就可以确定一个草可不可以用。现在我们假设：一个草的品质低于现在集合中奶牛所需的最低品质，很显然这个时候返回的$iter$是不在集合当中的（因为默认找更大的一个），此时一定有
$iter==cow\_set.end()$（根据$end()$的定义）。

代码：

```
1  #include <iostream>
2  #include <algorithm>
3  #include <set>
4
5  using namespace std;
6  typedef long long ll;
```

```cpp

const int maxn = 100005;

ll n, m;
pair<ll, ll> cow[maxn], grass[maxn];
multiset<ll> cow_set;

int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i ++)
        cin >> cow[i].first >> cow[i].second;
    for (int i = 1; i <= m; i ++)
        cin >> grass[i].first >> grass[i].second;
    sort(cow + 1, cow + 1 + n);
    sort(grass + 1, grass + 1 + m);

    ll ans = 0;
    for (int grass_i = 1, cow_i = 0; grass_i <= m; grass_i ++) {
        ll now_grass_cost = grass[grass_i].first;
        ll now_grass_quality = grass[grass_i].second;
        for (int i = cow_i + 1; i <= n && cow[i].first <= now_grass_cost; i ++) {
            // 符号：将序排列
            cow_set.insert(-cow[i].second);
            cow_i ++;
        }
        set<ll>::iterator iter = cow_set.lower_bound(-now_grass_quality);
        // 草品质达到
        if (iter != cow_set.end()) {
            cow_set.erase(iter);
            ans += now_grass_cost;
        }
    }
    // 如果还有牛剩下来 -> 这些牛的要求没有被满足 -> 答案为-1
    if (!cow_set.empty()) ans = -1;
    cout << ans << endl;
}
```

P5021 赛道修建[NOIP2018-D1T3]

题目描述

C城将要举办一系列的赛车比赛。在比赛前，需要在城内修建m条赛道。

C城一共有n个路口，这些路口编号为$1,2,\ldots,n$，有 n-1n-1n-1 条适合于修建赛道的双向通行的道路，每条道路连接着两个路口。其中，第 iii 条道路连接的两个路口编号为a_i和b_i，该道路的长度为l_i。借助这 n-1n-1n-1 条道路，从任何一个路口出发都能到达其他所有的路口。

一条赛道是一组互不相同的道路$e_1,e_2,\ldots,e_k$，满足可以从某个路口出发，依次经过 道路$e_1,e_2,\ldots,e_k$（每条道路经过一次，不允许调头）到达另一个路口。一条赛道的长度等于经过的各道路的长度之和。为保证安全，要求每条道路至多被一条赛道经过。

目前赛道修建的方案尚未确定。你的任务是设计一种赛道修建的方案，使得修建的m条赛道中长度最小的赛道长度最大d（即m条赛道中最短赛道的长度尽可能大）

输入格式

输入文件第一行包含两个由空格分隔的正整数n,m，分别表示路口数及需要修建的赛道数。

接下来$n-1$行，第i行包含三个正整数a_i,b_i,l_i，表示第i条适合于修建赛道的道 路连接的两个路口编号及道路长度。保证任意两个路口均可通过这$n-1$条道路相互到达。每行中相邻两数之间均由一个空格分隔。

输出格式

输出共一行，包含一个整数，表示长度最小的赛道长度的最大值。

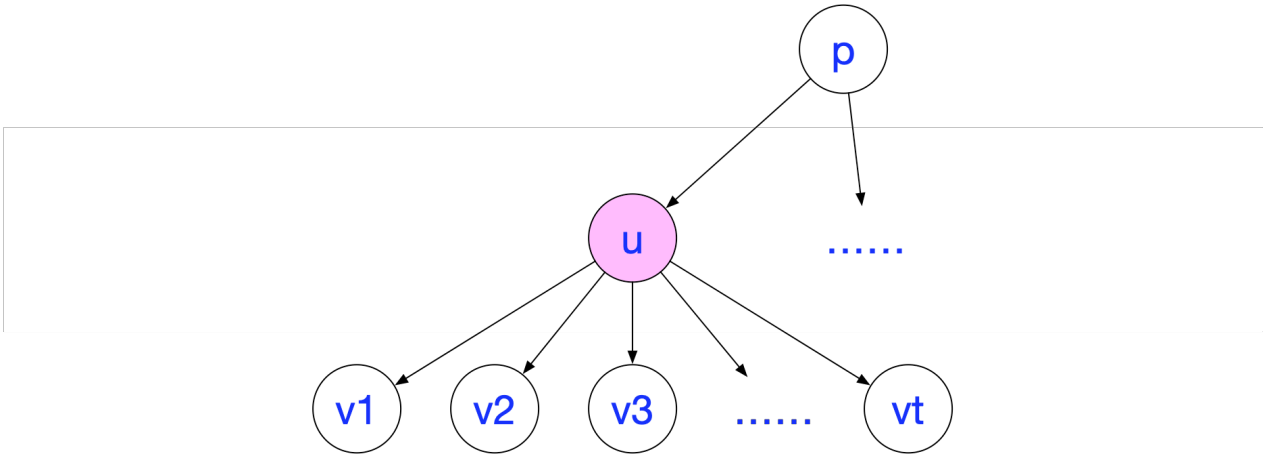
题解

首先可以有题目的条件得到这是一颗树（所有的路口都互相连通），而我们就要考虑的问题是最短的赛道，最长可以达到的长度。由于这里暴力枚举肯定会超时，所以采用二分答案的策略。

接下来就是考虑二分的目标，很明显，应该二分m。

考虑二分时的$check$函数：

考虑下面这种转移的形式，对于一个节点：



对于每一个节点，我们想要把其子树加起来可以达到m的去掉（dfs返回的是去不掉的最长路径值），可以两两匹配或者直接去掉。所以这个时候，我们需要一种数据结构支持下面两种操作：

1. 可以进行排序
2. 可以进行$lower_bound$操作进行查找

由此，我们想到了使用stl中的$multiset$这种容器。

下面为AC代码：

```
#include <iostream>
#include <set>
using namespace std;

const int maxn = 100005;

typedef long long ll;
ll head[maxn];

struct edge {
    ll to, next, w;
} g[maxn*2];

ll ecnt = 2, t = 0, num = 0, n, m;

void add_edge(ll u, ll v, ll w) {
    g[ecnt] = (edge) {v, head[u], w};
    head[u] = ecnt ++;
}

ll dfs(ll u, ll p) {
    multiset<ll> s;
    for (ll e = head[u]; e != 0; e = g[e].next) {
        ll v = g[e].to;
        if (v != p) {
            ll alpha = dfs(v, u) + g[e].w;
            if (alpha < t)
                s.insert(alpha);
            else
                num --;
        }
    }
    ll ret = 0;
    while (!s.empty()) {
        multiset<ll>::iterator iter_1 = s.begin();
        ll x = *iter_1; s.erase(iter_1);
        multiset<ll>::iterator iter_2 = s.lower_bound(t - x);
        if (iter_2 != s.end())
```

```
            s.erase(iter_2), num --;
        else
            ret = max(ret, x);
    }
    return ret;
}

int main() {
    cin >> n >> m;
    ll sum = 0;
    for (ll i = 1; i < n; i ++) {
        ll u, v, w; cin >> u >> v >> w;
        add_edge(u, v, w); add_edge(v, u, w);
        sum += w;
    }
    ll r = sum / m + 1;
    ll l = 0;
    while (r - l > 1) {
        t = (r + l) / 2;
        num = m;
        dfs(1, 0);
        if (num > 0)
            r = t;
        else
            l = t;
    }
    cout << l << endl;
    return 0;
}
```

ACOJ-1664 纯种奶牛

题目大意

n头奶牛排成了一条直线，其中第iii头的奶牛品种为a_i。约翰希望在队伍里选出一段尽量长的区间，使得这段区间里的所有奶牛都是相同品种的。为此他可以淘汰一些品种的奶牛，所谓淘汰一个品种的奶牛，就是让该品种的奶牛全部离开队伍，而剩余的奶牛仍然保持在队伍里的顺序。

约翰最多可以淘汰k种奶牛，请问约翰应该淘汰哪些品种的奶牛，然后选择哪一段区间，才能让纯种奶牛的区间尽量长?

题解

这道题和Jessica's Reading Problem非常类似，可以被称作为“大不小步法”。即可以有两个指针用来维护一个区间，而这个区间的维护条件就是确保这个区间始终有不多于$k+1$种元素，元素越多越好，而每次更新答案的方法就是更新这个区间的第一个元素在这个区间的元素数量，这样每一种元素其实都可以被更新到。而之所以要有$k+1$种元素，是因为可以去掉k个元素使得它们连续，那么换而言之就是有一个区间有$k+1$个元素，而数量就是其中任意一个元素的数量（最大最优）。而中间过程中用来维护元素个数的类似于桶的数据结构，就可以使用stl的map来完成（因为数据会达到$1e9$所以数组肯定会爆炸）

AC代码：

```
#include <iostream>
#include <map>

using namespace std;
typedef long long ll;

const int maxn = 1000005;
ll a[maxn];
int n, k;
int kind = 0;

map<ll, int> cnt;

void inc(int &j) {
  if (cnt[a[j]]++ == 0)
    kind ++;
}

void dec(int &j) {
    if (--cnt[a[j]] == 0)
        kind --;
}

void adv(int &i) {
    if (--cnt[a[i ++]] == 0)
        kind --;
}

int main() {
    cin >> n >> k;
    for (int i = 1; i <= n; i ++)
        cin >> a[i];
    int j = 1; int ans = 0;
    for (int i = 1; i <= n; adv(i)) {
        while (j <= n) {
            inc(j);
            if (kind > k + 1) {
```

```
                dec(j); break;
            } else j ++;
        }
        ans = max(ans, cnt[a[i]]);
    }
    cout << ans << endl;
    return 0;
}
```