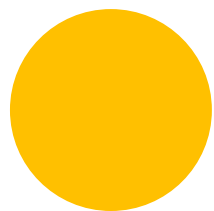
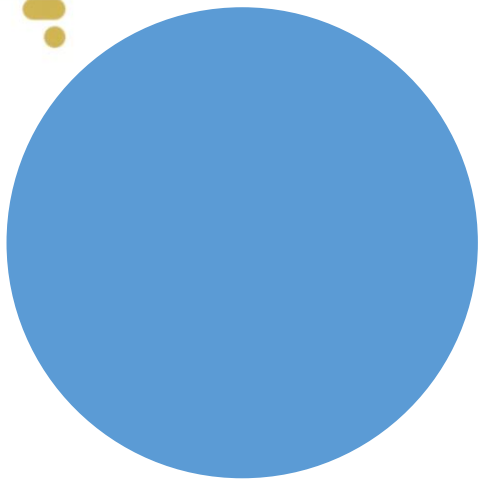


# 马上开始

35tang-C++竞赛系列三阶课程





# 《 35tang-C++竞赛系列三阶课程 》



## priority\_queue

使用起来和queue非常类似，不同的地方：  
不是先进先出，而是最大（或者最小）先出  
和set比较呢？

```
//测试优先队列
#include <iostream>
#include <queue> //这个头文件包含了priority_queue
#include <functional>
using namespace std;
int main ()
{
//priority_queue<int > mypq;//缺省top最大的
priority_queue<int,vector<int>,greater<int> > mypq;//top是最小的
mypq.push(30);
mypq.push(30);//看出来没有，可以重复
mypq.push(100);
mypq.push(25);
cout << "Popping out elements...";
while (!mypq.empty())
{
    cout << ' ' << mypq.top();
    mypq.pop();
}
cout << '\n';// cout<<endl;
return 0;
}
```

# 本节目标

- 合适的数据结构能够大大提高我们解决问题的能力 and 提高程序执行效率。
- 接水问题
  - 有一类竞赛题目有很大的相似性，用贪心或者模拟的思路，考虑用计数数组或者维护间隔段的方式，找到一个队列来分析。具体我们通过例题来学习。后面可以体会：这里的队列不一定是先进先出，可能是最大或者最小先出，可以用优先队列，也可以就是数组排序，具体情况来选择。
  - 也可以把时间段离散化成发生变化的点。
  - 这类问题进一步衍生下去到高级算法就是线段树和扫描线的算法。

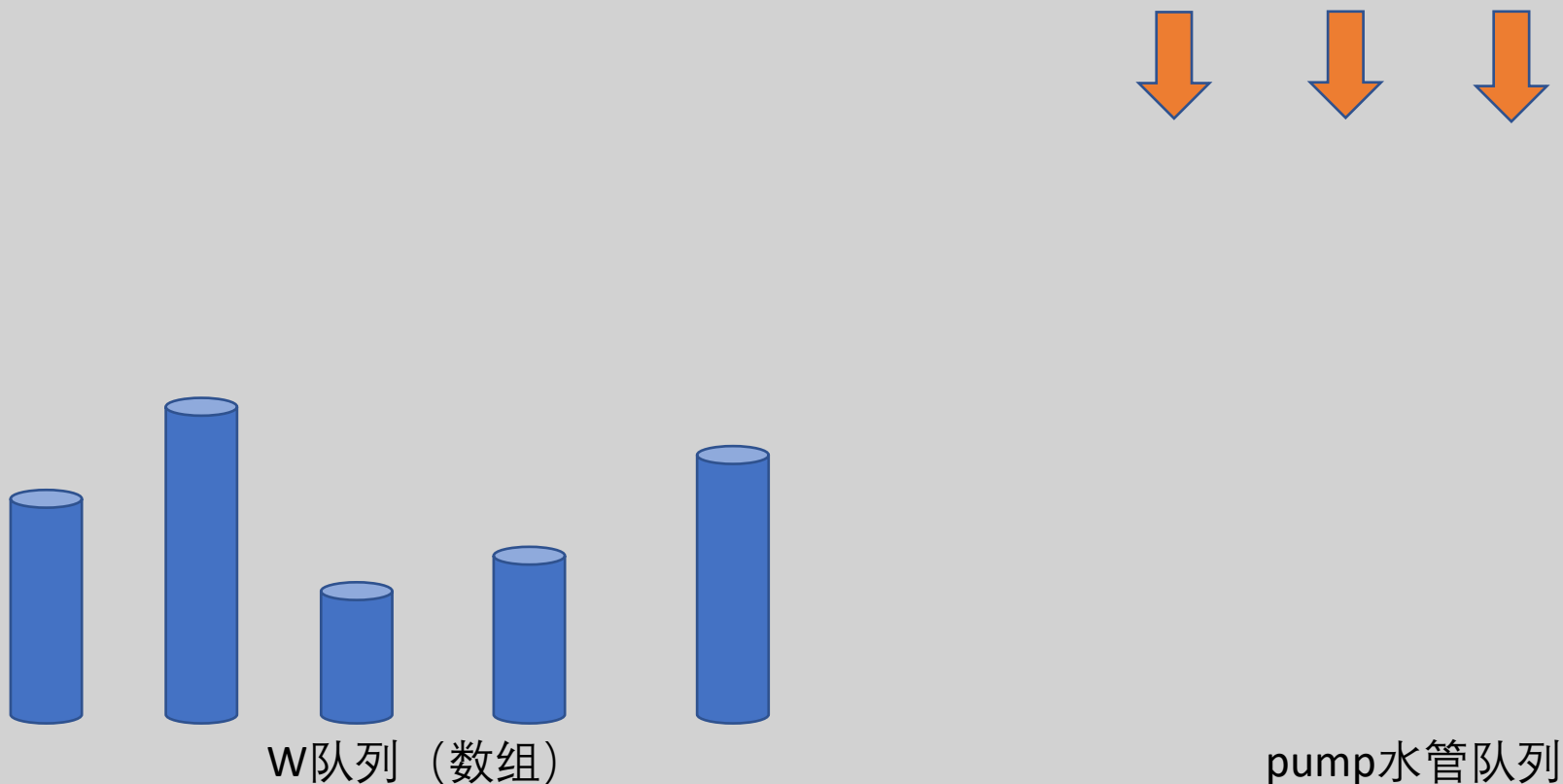
# NOIP 2010 接水问题

- 学校里有一个水房，水房里一共装有  $m$  个龙头可供同学们打开水，每个龙头每秒钟的供水量相等，均为 1。现在有  $n$  名同学准备接水，他们的初始接水顺序已经确定。将这些同学按接水顺序从 1 到  $n$  编号， $i$  号同学的接水量为  $w_i$ 。接水开始时，1 到  $m$  号同学各占一个水龙头，并同时打开水龙头接水。当其中某名同学  $j$  完成其接水量要求  $w_j$  后，下一名排队等候接水的同学  $k$  马上接替  $j$  同学的位置开始接水。这个换人的过程是瞬间完成的，且没有任何水的浪费。即  $j$  同学第  $x$  秒结束时完成接水，则  $k$  同学第  $x+1$  秒立刻开始接水。若当前接水人数  $n'$  不足  $m$ ，则只有  $n'$  个龙头供水，其它  $m-n'$  个龙头关闭。现在给出  $n$  名同学的接水量，按照上述接水规则，问所有同学都接完水需要多少秒。
- $1 \leq n \leq 10000$ ,  $1 \leq m \leq 100$  且  $m \leq n$ ;  $1 \leq w_i \leq 100$ 。



方法一：排队的同学依次安排在最先结束接水的水管接水。每一个排队同学自己去检查，根据现在水管在接水的同学的接水量，找到最快结束的一个去接水。

- 这一类的问题主要是要找到一个队列，核心思路都是找到一个队列来排队，然后不停的按照贪心策略安排下一个，修改队列。下图圆柱高度代表接水需要的时间。这个题目有一个特殊性，就是同学们是按照排队顺序接水的，也就是水管空出来来后依次选择下一个同学，所以对于W队列（排队接水同学）来讲，其实是一个数组就可以了，不一定用queue。下面模拟3个水管，5个同学接水。
- 最后的答案就是pump队列里面时间最多的一个（也就是最后完成任务的水管用的总时间=最后一个同学接完水的时间）。



# 第一个方法：队列

- 现在按照人来排队，就是先把1-m个人派给1-m个水管，每一个水管都有一个当前结束时间，然后对于后面的n-m个人，每一个人找管子队列里面结束时间最小的一个，把这个人排进去：修改当前水管的最后结束时间。再找下一个人。最后，全部排完了之后，找水管中结束时间最后的一个就是答案。
- 时间复杂度：  $O((n-m) * m)$ ;
- 这里用数组实现

```
int w[10001];
int pump[101];
for (int i=1;i<=n;i++) fin>>w[i];
for (int i=1;i<=m;i++) {
    pump[i]=w[i]; //m个pump排上前m个接水的 表示当前结束时间
}
for (int i=m+1;i<=n;i++)
{
    int minw=10000000;
    int minindex;
    for (int j=1;j<=m;j++){
        //打擂台找到当前最小的pump，也就是最先空下来的一个
        if (pump[j]<minw)
        {
            minw=pump[j];
            minindex=j;
        }
    }
    pump[minindex]+=w[i];
    //修改这个最先空的水管的结束时间，把当前排队的第一个同学需要的接水时间加上就是这个水管最新的结束时间
}
int maxw=0;
for (int i=1;i<=n;i++) if (pump[i]>maxw) maxw=pump[i]; //打擂台找所有水管结束时间最大的一个就是答案
```



# 优化：优先队列-priority\_queue

- pump数组其实是一个队列，但是这个队列特殊，不是先进先出，而是每一次取出最小的。找当前最先结束的水管，也就是在pump数组找最小的一个，比较耗费时间需要遍历m次。特殊的队列：是否可以把m个元素放在优先队列来维护？优先队列是自动排序的，而且时间复杂度是 $\log m$ 这样每一次可以取到当前最小并跟新。最终时间函数可以成为 $O((n-m)*\log m)$ ;
- priority\_queue
- 一起来改写

```
int w[10001];
int pump[101];
for (int i=1;i<=n;i++) fin>>w[i];
for (int i=1;i<=m;i++) {
    pump[i]=w[i]; //m个pump排上前m个接水的 表示当前结束时间
}
for (int i=m+1;i<=n;i++)
{
    int minw=10000000;
    int minindex;
    for (int j=1;j<=m;j++){
        //打擂台法找到当前最小的pump，也就是最先空下来的一个
        if (pump[j]<minw)
        {
            minw=pump[j];
            minindex=j;
        }
    }
    pump[minindex]+=w[i];

    //修改这个最先空的水管的结束时间，把当前排队的第一个同学需要的接水时间加上去就是这个水管最新的结束时间

}
int maxw=0;
for (int i=1;i<=n;i++) if (pump[i]>maxw) maxw=pump[i]; //打擂台找所有水管结束时间最大的一个就是答案
```

# 对比一下

```
int w[10001];
int pump[101];
for (int i=1;i<=n;i++) fin>>w[i];
for (int i=1;i<=m;i++) {
    pump[i]=w[i];
}

for (int i=m+1;i<=n;i++)
{
    int minw=10000000;
    int minindex;
    for (int j=1;j<=m;j++){
        //打擂台法找到当前最小的pump, 也就是最先空下来的一个
        if (pump[j]<minw)
        {
            minw=pump[j];
            minindex=j;
        }
    }
    pump[minindex]+=w[i];
}

int maxw=0;
for (int i=1;i<=n;i++) if (pump[i]>maxw) maxw=pump[i];
```

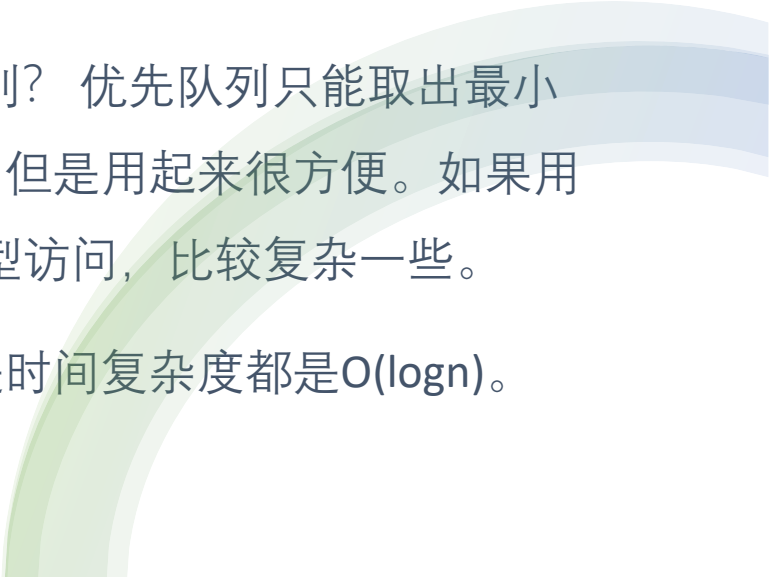
```
int w[10001];
priority_queue<int,vector<int>,greater<int> > pump;//top是最小的
for (int i=1;i<=n;i++) fin>>w[i];
for (int i=1;i<=m;i++)
{
    pump.push(w[i]);
    //pump[i]=w[i];//m个pupm排上前m个接水的
}

for (int i=m+1;i<=n;i++)
{
    int next=pump.top();
    pump.pop();
    pump.push(next+w[i]);
}

int maxw=0;
for (int i=1;i<=n;i++) if (pump[i]>maxw) maxw=pump[i];
```

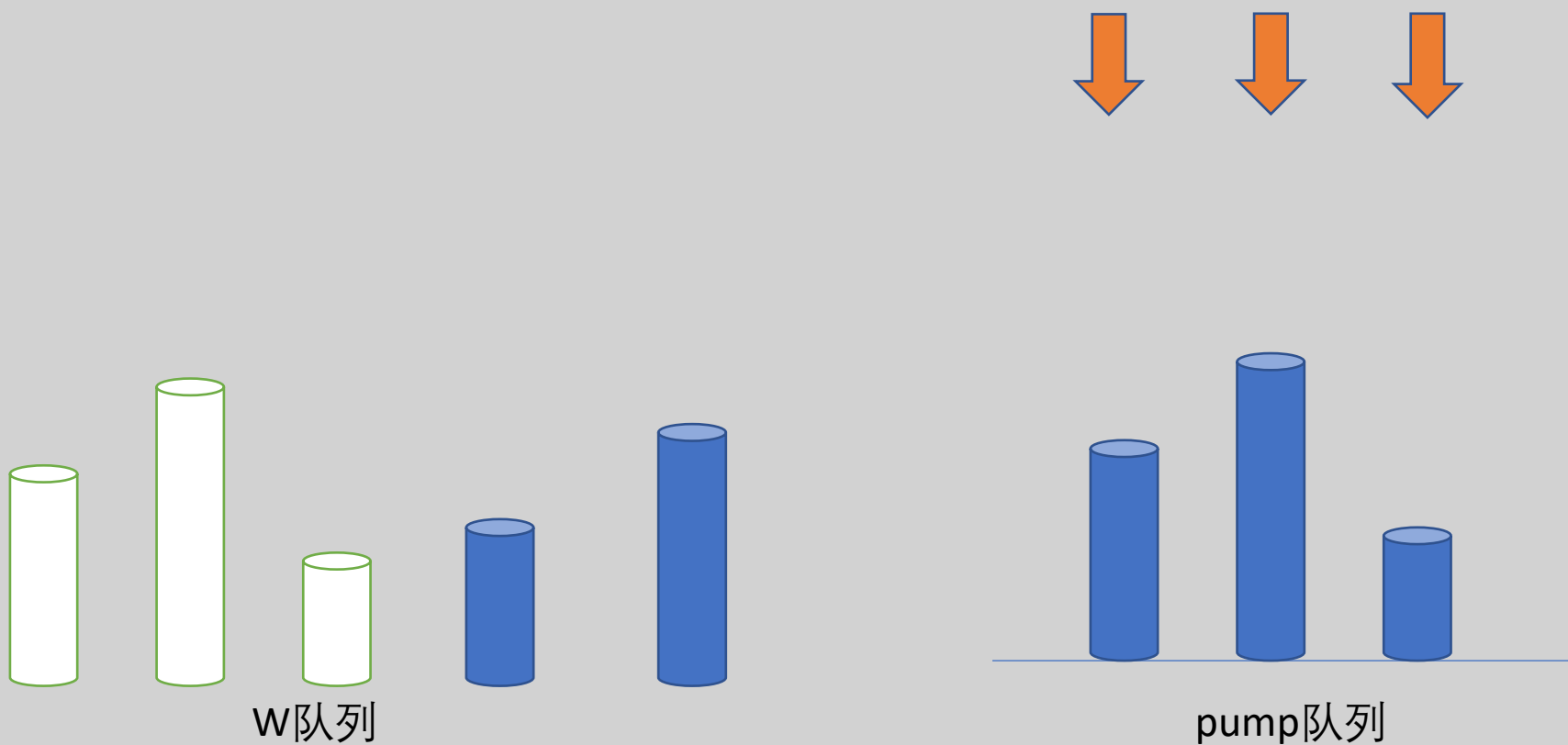


思考：这里可以用  
set么？

- 不能！有重复数据，比如多个同学接水时间一样，那么对当前水管接水的结束时间可能一样，用数组或者优先队列其实只要找到最小的一个（多个最小的取任意一个就可以）。
  - 但是，可以用multiset，这个东西和set唯一的区别就是容许重复。
  - 那么multiset和优先队列的区别？优先队列只能取出最小或者最大的一个，不能遍历，但是用起来很方便。如果用multiset就需要使用iterator类型访问，比较复杂一些。
  - 二者底层实现虽然不同。但是时间复杂度都是 $O(\log n)$ 。
- 

第二个方法时间扫描线，有一个管理员每1s就去看看所有的水管有没有空出来的，有就安排排队的第一个被来接水。

- 下图移动的直线代表时间的增加，每增加1s，当前有同学接水的水管的接水结束时间-1，减为0的时候就是接完了，换人继续



## 第二个方法时间扫描线

```
int n,m; fin>>n>>m;
int w[10001]; int pump[101];
for (int i=1;i<=n;i++) fin>>w[i];
for (int i=1;i<=m;i++){
    pump[i]=w[i];//m个pupm排上前m个接水的
}
int t=0; //当前用的时间
int curn=m+1;//当前排队的第一个同学的编号
while(true){
    if (curn==n+1) break;
    t++;//时间加1
    for (int j=1;j<=m;j++){//每一个水管结束时间减1
        pump[j]--;
        if (pump[j]==0&& curn<=n){//如果为0, 换上排队的同学
            pump[j]+=w[curn];
            curn++;
        }
    }
}
//处理现在还在接水的人, 把剩余时间的最大一个加到t里面
int leftt=0;
for (int i=1;i<=m;i++){
    if (pump[i]>=leftt) leftt=pump[i];//m个pupm排上前m个接水的
}
fout<<t+leftt<<endl;
```

- 模拟时间，每过1s检查一次：还是有个水管队列，m个水管，记录每一个水管离空出来还有多少时间（当前接水结束时间），但是模拟的维度不是看水管，而是看时间。按照时间从1到开始1秒一秒的加上去，时间每增加一秒，m个水管的结束时间都-1，什么时候哪个水管为0了就是这个水管空出来了，给下一个同学。一直到n个同学都接上水，并且最后全部接完水（所有水管的接完水时间都是0）。
- 时间复杂度是 $O(n*w)$
- w就是最大可能的接水时间

# 方法对比

这道题给定的数据范围

$1 \leq n \leq 10000, 1 \leq m \leq 100$  且  $m \leq n$

$1 \leq w_i \leq 100$

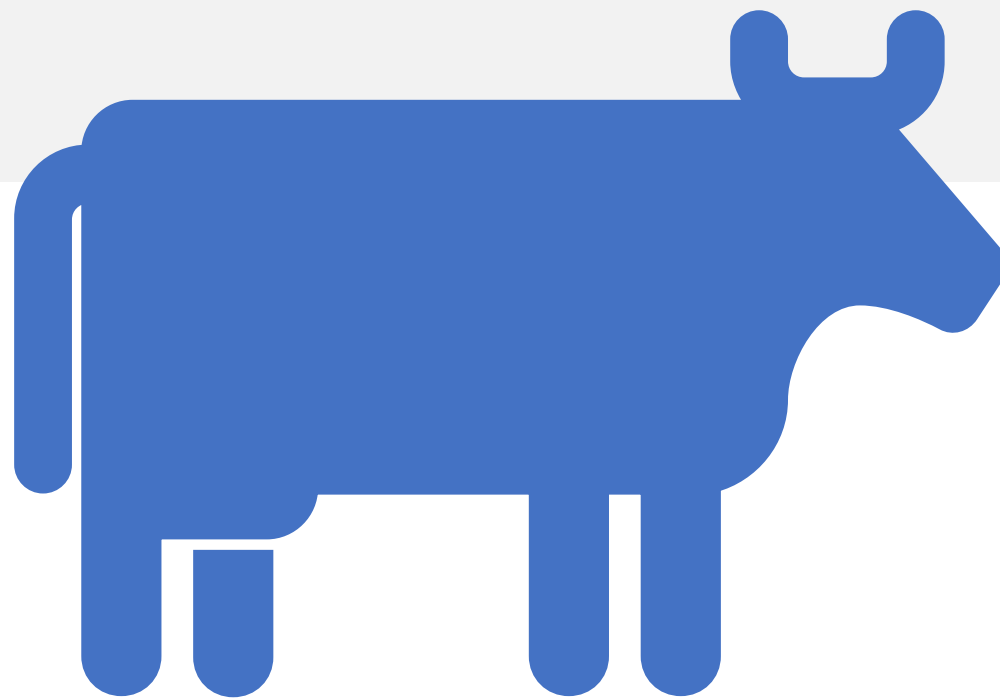
第一种方法模拟：每一个排队同学自己去检查，根据现在水管在接水的同学的接水量，找到最快结束的一个去接水。按照接水同学来看水管，时间复杂度是  $O((n-m) * m)$ ，大约10的6次方；

第二种方法模拟：有一个管理员每1s就去看看所有的水管有没有空出来的，有就安排排队的第一个被来接水。按照时间来看水管，时间复杂度是  $O(n * w)$ ，大约也是10的6次方。

所以对于这道题时间复杂度二者几乎是一样的，但是如果  $w$ （最大接水时间）很大呢？或者  $m$ （水管数目）很大呢？

# USACO 2018 December Contest, Bronze The Bucket List

- 100个奶牛，给出每一个奶牛开始和结束挤奶的时间，这个时间内需要用到10个以内的桶，大家都在挤奶的时候桶不能混用，问一共最少需要多少个桶？类似求最大的并发桶数目。
- 奶牛不超过100个，时间间隔不超过1000，桶不超过10。所以数量级是可以控制的，模拟的时候看时间或者奶牛都可以。
- 找要素，决定因素是什么？桶，时间。这就是2个不同的分析角度。



# 理解

输入

4

1 10 1

2 7 1

3 8 1

11 15 1

第1个奶牛在1到10时间内挤奶，用1个桶。

输出

3

- 第1个奶牛占用1个桶，第2个开始的时候第1个没有结束，所以还要用1个，第3个开始的时候1，2都没有结束，所以再用1个，第4个开始的时候前面都结束了，可以用前面的桶，不需要新的，所以最大需要3个桶



方法一：双循环遍历区间数组（结构数组，数组每一个元素表示一个奶牛，是一个结构体，包含开始时间和结束时间）。

把奶牛挤奶按照挤奶开始时间排序（上一个例题不需要因为是按照同学顺序来接水，而不是开始或者等待时间），第一个奶牛占用若干个桶，然后依次判断后面的奶牛，每一次看看他前面**所有**的奶牛是不是已经结束，如果结束就可以用空出来的桶，没有的话就用新的桶。这里维护两个数字，一个是当前总的空余水桶，一个就是当前用的水桶（答案就是打擂台取最大）。数据结构选择结构数组（数组每一个元素是一个包含开始和结束时间以及需要奶桶数目的结构体）就可以了，因为数组操作简单，可以排序，而且排序后遍历某个元素前面的所有元素也很简单。 $O(n^2)$

编号	开始时间	结束时间	本次需要桶	空余桶	当前在用
1	1	10	1	0	1
2	2	7	1	0（1号没有结束，没有空的）	2
3	3	8	1	0（1，2都没有结束,没有空的）	3
4	11	15	1	2（1，2，3都结束了，空出3个桶，用1个，空2个）	1
5	12	18	2	0（4没有结束，但是有2个空的可用）	2

## 第二种方法： 双循环按照时间扫描

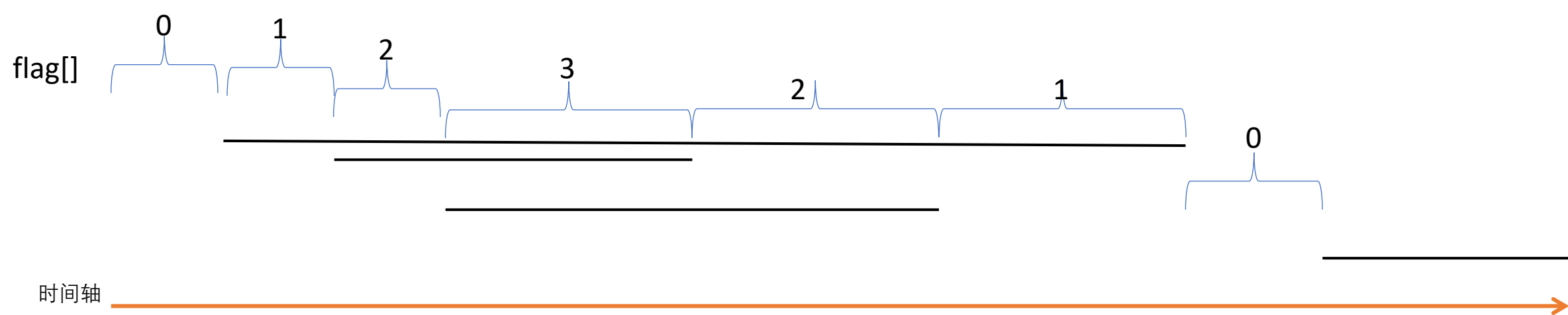
- 按照时间来看。时间 $t++$ ，每过1s就判断一次：判断每一个奶牛当前是否在挤奶（看 $t$ 如果落在一个奶牛的开始和结束时间，那么这个奶牛在 $t$ 时间挤奶， $t$ 时间需要的桶的数目就要加上这个奶牛挤奶要的桶的数目）。也就是要计算每一个时间所有正在挤奶的奶牛，然后所有时间里面找最大的并发挤奶的桶数目。
- 每个时间扫描所有奶牛挤奶区间。数据结构和第一种方法一样，不过就是双循环的维度不同。
- 时间复杂度 $O(t*n)$   $t$ 为挤奶时间的最大值。

编号	开始时间	结束时间	本次需要桶
1	1	10	1
2	2	7	1
3	3	8	1
4	11	15	1

时间 $t$	该时间正在挤奶的奶牛编号	需要桶
1	1	1
2	1, 2	2
3	1, 2, 3	3
4	1, 2, 3	3
5	1, 2, 3	3
6	1, 2, 3	3
7	1, 3	2
8	1	1
9	1	1
10		0
11	4	1
12	4	1
13	4	1
14	4	1
15	4	1

第三个方法：扫描挤奶区间。把时间区间的信息变成了时间点的信息。

每个区间更新时间包含的时间内的需要奶桶数目flag[]数组表示



黑色横线表示奶牛的挤奶时间间隔，方便起见，我们假设每个奶牛挤奶都需要一个奶桶。

当某一个时间段3个奶牛都挤奶的时候显然需要的桶最多。

把所有的1-1000时间做个计数，看看每一秒的并发桶需求，然后找最大。（这个和方法2类似）

扫描所有奶牛，比如第一个奶牛是1到10这个时间段，那么flag[1]到flag[9]都+1，奶牛2是2到7，那么flag[2]到flag[6]都加1...这样flag[]数组就表示了每一个时间点需要的桶的数目。

方法三图解：

就是算出每一个时间点所需要的奶桶数量

最后找里面最大的



编号	开始时间	结束时间	本次需要桶	flag数组设置(初始为0)
1	1	10	1	flag[1]到flag[9]的值 ++
2	2	7	1	flag[2]到flag[6]的值 ++
3	3	8	1	flag[3]到flag[7]的值 ++
4	11	15	1	flag[11]到flag[14]的值 ++
5	12	18	2	flag[12]到flag[17]的值 +2

```
int flag[1001]={0};//时间计数flag[i]表示第i分钟使用多少奶桶
int n;
int s,t,b;
fin>>n;
for (int i=1;i<=n;i++) {
    fin>>s>>t>>b;
    for (int j=s;j<t;j++) flag[j]+=b;
    //某个奶牛挤奶的开始和结束时间中每一个时间点
    //加上需要的奶桶数目
}
int r=0;
for (int i=0;i<=1000;i++) if(flag[i]>r) r=flag[i]; //找最大
fout<<r<<endl;
```

用的计数，就是把所有的1-1000时间做个计数，看看每一秒的并发桶需求，然后找最大。

这里不需要对所有奶牛的挤奶区间排队。

时间复杂度和方法2类似， $O(t*n)$   $t$ 为挤奶时间的最大值。本题给出条件 $t$ 应该不大于1000

## 第三个方法



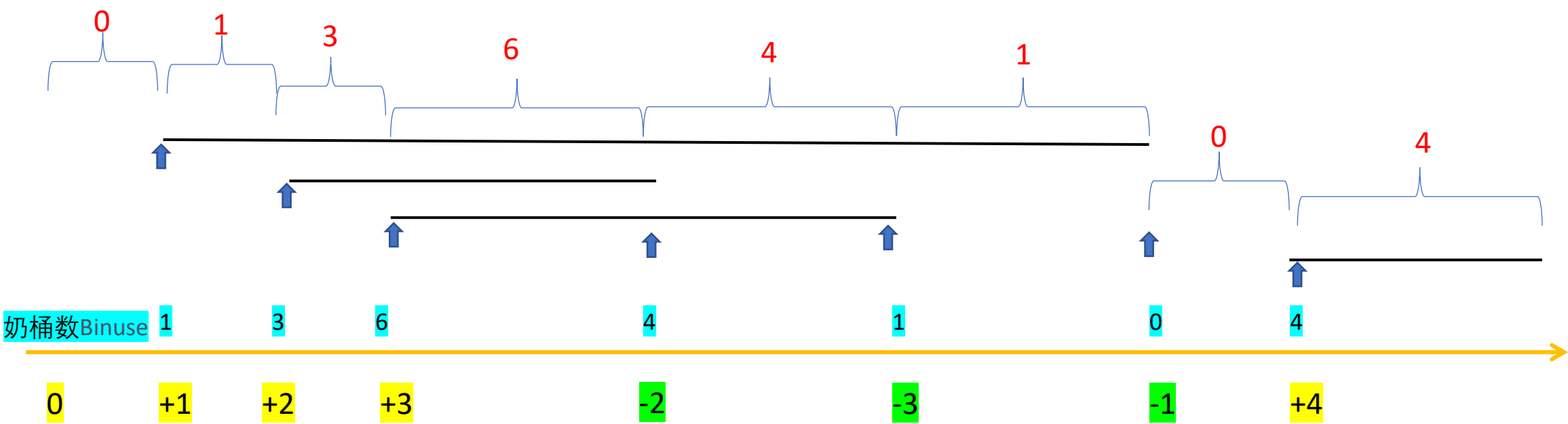
## 第四个方法

- 第三个方法还可以改进，因为这里flag数组当时间不超过1000的情况下是ok的，但是如果奶牛挤奶的时间跨度非常大，这个时间太大，就不能用flag数组记录每一个时间点这样计数了。而是按照时间顺序记录所有发生变化的点，然后遍历这个发生变化的点+或者-。注意是遍历变化点不是遍历时间。



方法四：离散，不扫每个奶牛，也不扫描时间，而是直接扫每一个发生变化的“点”

4条黑色线段表示4个挤奶的区间，假设按照开始时间的顺序，他们需要的奶桶数目分别是1，2，3，4。



离散，把所有奶牛的挤奶的两个时间点分开作为2个元素，每一个元素都记录时间和需要的奶桶数目，需要的奶桶数目如果是开始时间，就用正的，否则用负的，其实就是记录下来上面所有的黄色和绿色的数字，按时间排序。维护一个当前需要的奶桶数目的变量（Binuse），然把，遍历黄色和绿色的点，加上这些数字。最终答案就是Binuse的最大值。

最快：  
时间复杂度只有  
 $O(N \cdot \log N)$  排序消耗

tpoints数组就是包含这 $2 \cdot N$ 个拆开的变化点的数组

```
int N;
struct timpoint{
    int time;
    int buckets_needed;
};
timpoint tpoints[201];

bool mycompare(timpoint a,timpoint b)
{
    if (a.time<b.time) return true;
    else return false;
}
```

```
int solve(){
    int Bneeded=0,Binuse=0;
    for (int i=0; i<2*N; i++) {
        //循环2*N个开始或者结束的时间点， 分别+或者-需要的奶桶数目，
        //打擂台找到其中最大的一个就是答案
        Binuse+=tpoints[i].buckets_needed;
        Bneeded = max(Bneeded, Binuse);
    }
    return Bneeded;
}

int main(){
    ifstream fin ("blist.in");
    fin >> N;
    int s,t ,b;
    for (int i=0; i<N; i++) {
        //读入数据， 并把挤奶的起止时间拆开， 把N个点变成2*N个
        fin>>s>>t>>b;
        tpoints[2*i].time=s; tpoints[2*i]. buckets_needed =b;
        tpoints[2*i+1].time=t; tpoints[2*i+1]. buckets_needed =-b;
    }
    sort(tpoints,tpoints+2*N,mycompare); //排序很重要
    ofstream fout ("blist.out");
    fout << solve() << "\n";
    return 0;
}
```



# 方法四解析

编号	开始时间	结束时间	本次需要桶
1	1	10	1
2	2	7	2
3	3	8	3
4	11	15	4



编号	时间	本次需要桶
0	1	+1
1	2	+2
2	3	+3
3	7	-2
4	8	-3
5	10	-1
6	11	+4
7	15	-4

Binuse+=  
tpoints[i].buckets\_needed;



当前需要桶数目 Binuse
1
3
6
4
1
0
4
0

# 学到了什么？

- 一旦涉及到了时间，先后顺序，那么第一个要想一想需不需要sort？
- 这一类问题其实原理上方法都差不多，模拟和贪心。要么就是遍历时间间隔（人，水管，牛），要么就是遍历时间，要么就是找变化的时间点。
- 注意这里面离散化的方法：把每一个区间离散化成2个发生变化的点，然后我们去处理每一个点（区分开始和结束做不同处理）。
- 同样的问题很多解决办法，关键是维度。
- 数据结构上来讲数组是我们第一选择，简单。只有当效率不够，或者数据存在明显的先进先出，或者每一次选择最大/最小，这些明显的特点就可以用队列或者优先队列来优化。
- 如果数据量大约是10的5次方，基本上肯定要用 $O(n)$ 或者 $O(n \cdot \log n)$ ，是在想不出来就用 $O(n^2)$ 的算法，至少可以得一些分数。

# pair

pair就是包含两个元素的一个数据类型，如果一个struct里面只有两个类型，就可以用pair取代。非常类似结构体。

```
pair<int,int> ap; //定义一个变量ap，类型是pair
```

```
ap.first=4; ap.second=5; //通过.first和.second依次访问两个元素
```

```
cout<<ap.second; //输出ap的第二个元素
```

```
ap=make_pair(4,5); // make_pair构建一个pair类型，可以赋值
```

右侧程序读取n对整数，  
放到vector数组，排  
序，然后输出

```
#include <bits/stdc++.h>

using namespace std;

bool mycompare(pair<int,int> A,pair<int,int> B)
{//排序顺序是两个数的和小的放在前面，和一样，按照第一个数排序
    if (A.first+A.second<B.first+B.second) return true;
    return A.first<B.first;
    return false;
}

int main ()
{
    vector<pair<int,int> > myvector;
    int n;
    cin>>n;
    for (int i=0;i<n;i++)
    {
        int a,b;
        cin>>a>>b;
        myvector.push_back(make_pair(a,b));
    }
    sort(myvector.begin(),myvector.end(),mycompare);
    //如果不加自定义函数mycompare，缺省按照pair的第一个元素从小到大排序
    for (int i=0;i<myvector.size();i++)
        cout<<myvector[i].first<<" "<<myvector[i].second<<endl;

    return 0;
}
```

回过头看刚才拆点的这个程序，改造成vector和pair其实更简单

```
int N;
struct timpoint{
    int time;
    int buckets_needed;
};
timpoint tpoints[201];

bool mycompare(timpoint a,timpoint b)
{
    if (a.time<b.time) return true;
    else return false;
}
```

```
int solve(){
    int Bneeded=0,Binuse=0;
    for (int i=0; i<2*N; i++) {
        //循环2*N个开始或者结束的时间点， 分别+或者-需要的奶桶数目，
        //打擂台找到其中最大的一个就是答案
        Binuse+=tpoints[i].buckets_needed;
        Bneeded = max(Bneeded, Binuse);
    }
    return Bneeded;
}

int main(){
    ifstream fin ("blist.in");
    fin >> N;
    int s,t ,b;
    for (int i=0; i<N; i++) {
        //读入数据， 并把挤奶的起止时间拆开， 把N个点变成2*N个
        fin>>s>>t>>b;
        tpoints[2*i].time=s; tpoints[2*i]. buckets_needed =b;
        tpoints[2*i+1].time=t; tpoints[2*i+1]. buckets_needed =-b;
    }
    sort(tpoints,tpoints+2*N,mycompare);//排序很重要
    ofstream fout ("blist.out");
    fout << solve() << "\n";
    return 0;
}
```

回过头看刚才拆点的这个程序，改造成vector和pair其实更简单

```
int N;
/*struct timpoint{
    int time;
    int buckets_needed;
};
timpoint tpoints[201];*/
vector <pair<int,int> > tpoints;

//pair中的first放time， second放
buckets_needed
```

```
int solve(){
    int Bneeded=0,Binuse=0;
    for (int i=0; i<2*N; i++) { //循环2*N个开始或者结束的时间点， 分别+或者-需要的奶桶数目， 打擂台找到其中最大的一个就是答案
        //Binuse+=tpoints[i].buckets_needed;
        Binuse+=tpoints[i].second;

        Bneeded = max(Bneeded, Binuse);
    }
    return Bneeded;
}

int main(){
    ifstream fin ("blist.in");
    fin >> N;
    int s,t ,b;
    for (int i=0; i<N; i++) { //读入数据， 并把挤奶的起止时间拆开， 把N个点变成2*N个
        fin>>s>>t>>b;
        /*tpoints[2*i].time=s; tpoints[2*i]. buckets_needed =b;
        tpoints[2*i+1].time=t; tpoints[2*i+1]. buckets_needed =-b; */
        tpoints.push_back(make_pair(s,b)); //插入开始点
        tpoints.push_back(make_pair(t,-b)); //插入结束点
    }
    //sort(tpoints,tpoints+2*N,mycompare);
    sort(tpoints.begin(),tpoints.end()); //缺省按照pair第一个元素排序， 所以不需要自定义函数
    ofstream fout ("blist.out");
    fout << solve() << endl;
    return 0;
}
```

# 下节课预习

回忆一下DFS搜索，复习一下前面DFS搜索的例题

# 作业1：吃饭

customer.cpp

建议可以尝试

使用pair而不是

struct

给出若干客人到达饭馆和离开饭馆的时间。求饭馆里最多某一个时间有多少客人。

输入：

第一行一个整数 $n$ ，表示客人的数目。

后面 $n$ 行，每一行2个整数 $a$ 和 $b$ ，按照顺序表示一个客人到达和离开的时间。所有到达和离开的时间不同。

输出：

一个整数。饭馆里最多某一个时间的客人数目。

数据限制：  $1 \leq n \leq 2 \cdot 10^5$   $1 \leq a < b \leq 10^9$

示例输入：

3

5 8

2 4

3 9

示例输出：

2

示例说明：一共3个客人，第3到4的时候客人2和客人3都在饭馆，在时间5的时候，客人1来了，这时候客人2已经离开，所以还是有2个客人同时在饭馆。



# 挑战作业2

USACO training Section 1.3 PROB Milking Cows

<https://www.luogu.com.cn/problem/P1204>

给出若干农名挤奶的开始时间和结束时间，求最长至少有一人在挤奶的时间段以及最长的无人挤奶的时间段。

# 作业2提示

- 和课堂挤奶不同的区别在于现在记录的不是最大并发奶桶，而是最长连续挤奶区间，就是说过程中维护的东西不一样了。
- 可以按照开始时间排序，然后奶牛一个一个来看，如果挤奶时间在刚才的挤奶区间内就扩大区间，不在就开一个新的继续算，保留最大的。
- 也可以考虑离散的方法，把开始和结束时间拆开成2个点，然后排序，再遍历这 $2*n$ 个点，设置一个变量，遇到开始点就+1，遇到结束点就-1。当这个变量从0变成1的时候，就是没有挤奶的一个时间段的结束，从1变成0的时候就是有奶牛挤奶的时间段的结束。对这两个时间段用打擂台法找分别取最大。



由易到难，思维体系训练  
实战结合，创新协作培养  
兴趣导向，未来职业引领

<https://www.35tang.com>



扫码关注公众号

<https://www.三五堂.com>



添加辅导老师