# Expriment 4

PB17111623 范睿

2019 年 12 月 12 日

## 1 基站放置

### 1.1 题目

现有N个村庄排成一排，在这些村庄里需要挑选若干村庄搭建基站，其中基站的覆盖半径为R。假设在村庄i处放置基站，村庄i的坐标为$X_i$，则处在$[X_i - R, X_i + R]$范围内的村庄都能被该基站所覆盖。在给出所有村庄坐标的情况下，求使所有村庄都能被基站覆盖所需要的最少的基站个数。

输入:
第一行包含两个整数N,R分别表示村庄个数和基站覆盖半径。接下来的N行每行一个整数$X_i$表示村庄i的坐标。

数据规模： $N \leq 100,000$
$R \leq 1,000$
$X_i \leq 1,000,000$

输出：
输出一个整数表示最少需要搭建的基站数目。

Sample Input:
5 10
1
100
10
30
20

Sample Output:
3

```
/*
在10, 30, 100处搭建基站
*/
```

## 1.2 思路

先将所有基站的位置从小到大排序。（用$\mathcal{O}(n \log n)$的算法）
每次选择最左边的未被覆盖的点，将直径的最左端与该点对齐，在此时比圆心小的距离圆心最近的村庄放置基站，直到所有村庄都被覆盖。

## 1.3 算法

**Algorithm 1**

**输入:** 村庄个数N，基站覆盖半径R，村庄位置X[1...N]
**输出:** 最少基站个数sum

```
RandomSort(X,N)
index ← 0
sum ← 0
while index < n do
    low ← X[index]
    mid ← low + R
    i ← index
    while i < N and X[i] ≤ mid do
        i++
    end while
    mid ← X[i − 1]
    high ← mid + R
    num++
    while index < N and X[index] ≤ high do
        index++
    end while
end while
return sum
```

## 1.4 复杂度分析

利用random sort，时间为$\mathcal{O}(n \log n)$，index从1到N，时间为$\mathcal{O}(n)$,所以总的时间复杂度为$\mathcal{O}(n \log n)$

# 2 任务调度

## 2.1 题目

现有N个任务需要在一台机器上进行调度，其中每个任务有相应的到达时间$r_i$和执行时间$p_i$任务必须在到达之后才能进行执行，该机器同时只能执行一个任务。当一个任务在机器上执行$p_i$时间后，该任务即视为完成。若该任务的完成时刻为$f_i$，则该任务的响应时间为$f_i - r_i + 1$.我们需要找到一个调度方案使所有任务的响应时间之和最短。
需要说明的是，在这个问题里我们假设已经将时间划分为若干不可再分的时隙，即一个任务最少的执行时间为1个单位时间，且任务只会在整数时刻到达。
此外，调度是可抢占的，这意味着在任意时隙开始时可以通过挂起正在执行的任务i转而执行另一项任务j，随后可以恢复任务i并执行其剩余的工作。

输入：
第一行一个整数N表示总共的任务个数。接下来的N行，每行两个整数$r_i, p_i$表示任务的到达时间和执行时间。

数据规模： $N \leq 100,000$
$r_i \leq 10,000,000$
$p_i \leq 1,000$

输出：
输出一个整数表示最小的所有任务的总响应时间。

Sample Input:

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

Sample Output:

```
7
/*
1+2+4=7
*/
```

## 2.2 思路

在每一个时刻，选择手上【剩余】时间最短的任务执行。

思路细节：
先将所有任务按照到达时间从小到大排序。从第1个任务开始执行。
若手上只有一个任务，当前执行的任务在下一个任务到来之前就可以执行完，就把当前任务执行完，然后跳到下一个任务的到达时间继续执行；
若手上有多个任务，正在被执行的任务在下一个任务到来之间就可以被执行完，先将它执行完，然后选择手上剩余执行时间最小的任务继续执行；
若手上正在被执行的任务在下一个任务到来之前不能被执行完（不管手上有几个任务），先将正在执行的任务执行到下一个任务的到来时间，然后将下一个任务的执行时间和手上所有的任务的剩余时间作比较，选择剩余时间最少的那个任务执行。

实现细节：
利用数组和链表实现。
定义两个结构体work和worknode。
work中存放该任务的到达时间time，firstwork指向time时刻到来的第一个任务。 worknode中存放开始时间starttime，持续时间p，该时刻的下一个任务next。开一个大小为N的work数组W[1...N]，每个位置W[i].firstwork指向W[i].time到来的第一个任务，所以初始时W[i].time和W[i].firstwork->starttime相同。
index从1开始遍历。每次先比较一下W[index].time+W[index].firstwork->p和W[index+1].time，也就是比较看看是当前手头上的事情先执行完还是下一个任务先到：
如果手上的事情先执行完，就把W[index].time改成W[index].time+W[index].firstwork->p，也就是手上事情执行完的时刻，然后看看手上还有无任务，有的话就选择剩余时间最短的，也就是firstwork指着的任务（因为我将一个没执行完的任务加入链表的时候是保证剩余时间从小到大排列的）继续执行；如果下一个任务先到，就把firstwork指向的p减去W[index+1].time-W[index].time，表示我当前的任务先执行这么多，然后把W[index].firstwork后连的任务链表全部加入W[index+1].firstwork指向的任务链表（按照剩余时间排序）。将index+1，选择剩余时间最少的任务执行。

## 2.3 算法

## 2.4 复杂度分析

排序$\mathcal{O}(n\log n)$，初始化$\mathcal{O}(n)$，遍历每个任务$\mathcal{O}(n)$，总复杂度$\mathcal{O}(n\log n)$

**Algorithm 2**

**输入:** 总任务个数N，所有任务的到达时间R[1...N],所有任务的执行时间P[1...N]

**输出:** 最小总响应时间

```
1: RandomSort(R, P, N) //按照R的大小排序
2: make W[1...N] a new work array //work的定义与思路中写的一样
3: for i = 1 to N do
4:     W[i].time ← R[i]
5:     W[i].firstwork−>starttime ← R[i]
6:     W[i].firstwork−>p ← P[i]
7:     W[i].firstwork−>next ← NULL
8: end for
9: index ← 0
10: sum ← 0
11: while there is a work not finished do
12:     if W[index].time + W[index].firstwork−>p ≤ W[index].time then
13:         W[index].time ← W[index].time + W[index].firstwork−>p
14:         sum+ = W[index].time − W[index].firstwork−>starttime
15:         Delete(W[index].firstwork)
16:     else
17:         W[index].firstwork−>p− = W[index + 1].time − W[index].time
18:         Add all the works in the worklist W[index].firstwork into W[index+1].firstwork
    and sort by p
19:         index++
20:     end if
21: end while
22: return sum
```

# 3 数据缓存

## 3.1 题目

现有一个长为N的数据访问序列和一块大小为K的缓存，请设计算法使数据访问的MISS次数最少。
假设所有可能访问的数据在一个确定的范围内，这里用整数来表示存放数据的地址。
假设无论是否命中，这次访问的数据都需要存放在缓存内。

输入：
第一行两个整数N,K表示共有N个数据访问的请求，缓存大小为K。
接下来的N行每行一个整数i，表示访问内存地址为i的数据。

数据规模：
$N \leq 10,000$
$K \leq 1,000$
内存地址的范围为0 10,000

输出:
输出一个整数表示最少的MISS次数。

Sample Input:
5 2
1
2
3
1
2

Sample Output:
4
/*
1 MISS 存入
2 MISS 存入
3 MISS 存入并替换2
1 HIT
2 MISS
*/

### 3.2 思路

每次访问内存：
若hit，什么也不做
若miss：
若cache不满，将被访问的地址中的内容加入cache，miss数+1
若cache满，将cache中最晚被用到的数据覆盖，miss数+1

实现细节：
利用input[1...N]来表示所有对地址的访问
用nxt[1...N]表示比N大的，最小的对input[N]的访问
用InCache[0...10000]表示地址i是否在Cache中
利用c++中的优先队列实现将cache中nxt[i]最小的那个排在队列最前。
每次访问，若miss且cache不满，将input[i]加入cache，并将i加入队列；若miss且cache满，将优先队列弹出一个i，将input[i]踢出cache，将输入加入cache，且将其下标加入队列。

### 3.3 算法

*另外还要考虑的是，如果是hit，还需要更新一下队列，因为hit值的下标已经改变，队列中的排序是按照上一次加入时的nxt值，这一点在算法中没有写出来。

### 3.4 复杂度分析

初始化previous $\mathcal{O}(n)$，初始化nxt $\mathcal{O}(n)$ ，初始化InCache $\mathcal{O}(1)$，计算sum $\mathcal{O}(n)$，所以总的时间复杂度为$\mathcal{O}(n)$。

## 4 晾衣服

### 4.1 题目

小明在一个很长的晾衣架上挂了N件等待被晾干的衣服，相邻的两件衣服之间若距离过近会影响衣服的晾干时间。例如，我们可以用相邻两件衣服的距离的最小值来衡量全部衣服晾干的速度。
现在小明选择将其中的M件衣服撤走，以使得剩余的衣服可以更快晾干。
给定N件衣服在晾衣架上的位置，请设计算法使得撤走MM件衣服后相邻衣服距离的最小值最大。

输入:
第一行两个整数N,M表示共有N件衣服，其中可以撤走M件。接下来的N行，每行一个整数$X_i$表示衣服的坐标。

**Algorithm 3**

**输入:** cache大小K，访问次数N，内存访问input[1...N]

**输出:** 最少MISS数sum

```
let nxt[1...N] be a new array
let previous[1...N] be a new array
let InCache[0...10000] be a new array
sum ← 0
for i = 0 to 10000 do
    InCache[i] ← 0
end for
for i = 1 to N do
    previous[i] ← Inf
end for
for i = 1 to N do
    if previous[input[i]] is not Inf then
        nxt[previous[input[i]]] ← i
    end if
    previous[input[i]] ← i
end for
for i = 1 to N do
    if InCache[input[i]] is 0 then
        sum++
        if cache is not full then
            InCache[input[i]] ← 1
            queue.push(i)
        else
            x ← queue.pop()
            InCache[x] ← 0
            InCache[i] ← 1
            queue.push(i)
        end if
    end if
end for
return sum
```

数据规模：
$M < N \leq 200,000$
$X_i \leq 20,000,000$
输出:
输出一个整数表示撤走M件衣服后最大的相邻衣服距离的最小值。

Sample Input:
5 2
4
2
25
7
29

Sample Output:
5
/*
撤走4，25
最小距离为7-2=5
*/

## 4.2 思路

先将所有衣服的位置从小到大排序，然后将每个衣服的位置减去第一个衣服的位置，预处理成每件衣服到第一件衣服的距离。
先记录距离第一讲衣服最远的衣服的距离为length。
从(0,length)开始向下二分查找，先看撤掉m件衣服可不可以达到最小距离为length/2；可以的话在二分查找(length/2,length)，否则二分查找(0,length/2)...直到二分查找的上下界相差小于一，此时最大的相邻衣服最小值为下界值。

## 4.3 算法

## 4.4 复杂度分析

排序$\mathcal{O}(n \log n)$，初始化pos$\mathcal{O}(n)$，二分查找$\mathcal{O}(n \log length)$，所以总的时间复杂度为$\mathcal{O}(n \log(n * length))$

**Algorithm 4**

**输入:** 衣服件数N，可撤去衣服件数M，衣服位置pos[1...N]

**输出:** 最大相邻衣服最小值

```
1: RandomSort(pos, 0, N)
2: start ← pos[0]
3: for i = 0 to N do
4:     pos[i]-=start
5: end for
6: length ← pos[N − 1]
7: low ← 0
8: high ← length + 1
9: while high − low > 1 do
10:    if OK((high-low)/2, pos) then
11:        low ← (high − low)/2
12:    else
13:        high ← (high − low)/2
14:    end if
15: end while
16: return low
```

**Algorithm 5** OK

**输入:** 距离dis，位置pos[1...N]

**输出:** 撤走M件衣服最大的最小间距是否可以达到dis

```
1: last ← 0
2: for i = 0 to N-M-1 do
3:     current ← last + 1
4:     while current < n and pos[current] − pos[last] < mid do
5:         current++
6:     end while
7:     if current is N then
8:         return 0
9:     end if
10:    last ← current
11: end for
12: return 1
```