

题目描述

火车站附近的货物中转站负责将到站货物运往仓库，小明在中转站负责调度2K辆中转车（K辆干货中转车，K辆湿货中转车）。

货物由不同供货商从各地发来，各地的货物是依次进站，然后小明按照卸货顺序依次装货到中转车，一个供货商的货只能装到一辆车上，不能拆装，但是一辆车可以装多家供货商的货；

中转车的限载货物量由小明统一指定，在完成货物中转的前提下，请问中转车的统一限载货物数最小值为多少。

输入描述

- 第一行 length 表示供货商数量 $1 \leq \text{length} \leq 10^4$
- 第二行 goods 表示供货数数组 $1 \leq \text{goods}[i] \leq 10^4$
- 第三行 types表示对应货物类型，types[i]等于0或者1，其中0代表干货，1代表湿货
- 第四行 k表示单类中转车数量 $1 \leq k \leq \text{goods.length}$

输出描述

运行结果输出一个整数，表示中转车统一限载货物数

备注

中转车最多跑一趟仓库

用例

输入	4 3 2 6 3 0 1 1 0 2
输出	6
说明	货物1和货物4为干货，由2辆干货中转车中转，每辆车运输一个货物，限载为3 货物2和货物3为湿货，由2辆湿货中转车中转，每辆车运输一个货物，限载为6 这样中转车统一限载货物数可以设置为6（干货车和湿货车限载最大值），是最小的取值

输入	4 3 2 6 8 0 1 1 1 1
输出	16
说明	货物1为干货，由1辆干货中转车中转，限载为3 货物2、货物3、货物4为湿货，由1辆湿货中转车中转，限载为16 这样中转车统一限载货物数可以设置为16（干货车和湿货车限载最大值），是最小的取值

题目解析

根据用例意思，干货只能由干货车运输，湿货只能由湿货车运输，而干、湿货车都各有k辆。

如果 某类货数量 <= 某类车数量，那么该类车的最大限载就是该类货物中最重的。

如果 某类货数量 > 某类车数量，那么说明，无法为 每个供货商的货 都安排一辆车，必然由多个供货商的货，被安排到一辆车上。

此时，要实现：统一最小限载，则只能将比较轻的两个供货商的货物放到一辆车上，这样才能实现最小限载，比如有如下供货商干货：

10 20 30 40

但是只有三辆干货车，那么如何安放货物才能实现最小限载呢？

肯定是将最轻的两个货10、20放到一辆车上，此时相当于每辆干货车承载货物重量为：30、30、40，此时最小

题目解析

根据用例意思，干货只能由干货车运输，湿货只能由湿货车运输，而干、湿货车都各有k辆。

如果 某类货数量 \leq 某类车数量，那么该类车的最大限载就是该类货物中最重的。

如果 某类货数量 $>$ 某类车数量，那么说明，无法为 每个供货商的货 都安排一辆车，必然由多个供货商的货，被安排到一辆车上。

此时，要实现：统一最小限载，则只能将比较轻的两个供货商的货物放到一辆车上，这样才能实现最小限载，比如有如下供货商干货：

10 20 30 40

但是只有三辆干货车，那么如何安放货物才能实现最小限载呢？

肯定是将最轻的两个货10、20放到一辆车上，此时相当于每辆干货车承载货物重量为：30、30、40，此时最小限载为40

如果将10和30放在一辆车上，或者将10和40放在一辆车上，都可能造成最小限载超过40。

我的解题思路是：

定义一个干货数组dry，和一个湿货数组wet，然后将输入的goods根据types将干货存入dry，将湿货存入wet。

如果dry的数量 $\leq k$ ，则不需要将多个货物放到一辆车上

如果dry的数量 $> k$ ，那么需要将多个货物放到一辆车上，此时根据贪心思维，我们应该将最轻的连两个货物放到一辆车上。

那么如何找出最轻的两个货物呢？

我这里直接先将dry降序排序了，那么dry尾部两个货物就是最轻的，因此直接将dry尾部两个货物弹出，然后相加，得到一个组合货物重量。

此时需要将组合重量再次加入dry，但是为了保证dry的有序，这里我采用二分查找 $O(\log N)$ 后插入，这要比重新排序 $O(n)$ 性能更好。

然后再看dry的数量 是否 $> k$ ，如果还大于，则继续上面逻辑，直到dry数量 $\leq k$ 。

对于wet的处理同dry。

2023.02.04 根据网友指正，上面的解题逻辑没有考虑一个重要的点：

货物由不同供货商从各地发来，各地的货物是依次进站，然后**小明按照卸货顺序依次装货到中转车**，一个供货商的货只能装到一辆车上，不能拆装，但是一辆车可以装多家供货商的货；

这意味着，我们不难改变输入的顺序，要按照输入的顺序去装车。并且干货只能装到干货车，湿货只能装到湿货车。比如用例1中：

```
4
3 2 6 3
0 1 1 0
2
```

第1个货物是：干货，因此装到干货车。

第2个货物是：湿货，由于湿货不难装到干货车，因此第一辆车只装一个货物就必须走。第二个货物需要装到一辆湿货车上。

第3个货物是：湿货，前一步的湿货车还没走，此时我们有两种选择：

- 将第3个货物继续装到前面的湿货车上，即和第2个货物装一起，但是会导致最低限载被提高
- 将第3个货物装到下一辆湿货车上

第4个货物是：干货，因此只能重新要一辆干货车，继续装载。

说到这里，大家可能已经感觉出来，连续的0（干货）可以用一辆车运，也可以分成多辆车运。而一旦连续0（干货）被中断，即出现了1（湿货），则前面的干货车必须走，这个湿货需要申请一辆湿货车来运。

根据上面逻辑，我们可以将连续干货 和 连续湿货 对应的区间段统计出来：

连续干货：[3]、[3]

连续湿货：[2, 6]

- 如果连续干货的区间段数 刚好等于 k，则刚好每段连续干货，都可以分配一辆干货车。
- 如果连续干货的区间段数 $< k$ ，则出了必要的给每段分配一辆干货车外，还有多余的干货车可以继续分配：此时，我们应该将多段连续干货，按照**和大小降序**，优先给**和最大**的连续干货段多分配一辆车，即相当于**将一个区间一分为二**，这样**最低限载就降低了**，并且多出的干货车数量减1。如果多余的干货车数量不等于0，则继续按上面逻辑，分配一辆车给**和最大**的连续干货段。直到多余的干货车用完。
- 如果连续干货的区间段数 $> k$ ，此时是无法完成运输任务的，应该视为异常情况，但是本题没有说明异常处理，因此可以认为此场景不存在。

上面逻辑中，每次将区间一份为二后，都要重新按照 区间和大小降序，这里可以用 **优先队列**。

JavaScript算法源码

```
1 /* JavaScript Node ACM模式 控制台输入获取 */
2 const readline = require("readline");
```

伏城之外 已关注

2 6 13 专栏目录 已订阅

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 4) {
14     const n = lines[0] - 0;
15     const goods = lines[1].split(" ").map(Number);
16     const types = lines[2].split(" ").map(Number);
17     const k = lines[3] - 0;
18
19     console.log(getResult(n, goods, types, k));
20     lines.length = 0;
21   }
22 });
23
24 function getResult(n, goods, types, k) {
25   const dries = new PriorityQueue((a, b) => b[0] - a[0]);
26   const wets = new PriorityQueue((a, b) => b[0] - a[0]);
27
28   let isDry = true;
29   let tmp = [];
30   let sum = 0;
31
32   for (let i = 0; i < n; i++) {
33     if (types[i] === 0) {
34       if (isDry) {
35         tmp.push(goods[i]);
36         sum += goods[i];
37       } else {
38         tmp.unshift(sum);
39         wets.offer(tmp);
40
41         tmp = [];
42         sum = 0;
43
44         tmp.push(goods[i]);
45         sum += goods[i];
46
47         isDry = true;
48       }
49     } else {
50       if (isDry) {
51         tmp.unshift(sum);
52         dries.offer(tmp);
53
54         tmp = [];
55         sum = 0;
56
57         tmp.push(goods[i]);
58         sum += goods[i];
59
60         isDry = false;
61       } else {
62         tmp.push(goods[i]);
63         sum += goods[i];
64       }
65     }
66   }
67
68   tmp.unshift(sum);
69   if (isDry) {
70     dries.offer(tmp);
71   } else {
72     wets.offer(tmp);
73   }
74
75   return Math.max(divide(dries, k), divide(wets, k));
76 }
77
78 function divide(goods, k) {
79   while (goods.size() < k) {
80     const top = goods.poll();
81
82     if (!top) break;
83
84     const dry_sum = top.shift();
85
86     if (top.length <= 1) {
87       return dry_sum;
88     }
89
90     let splitIdx = 0;
91     let splitHalf = 0;
92     let half = 0;
```



```

70     drys.offer(tmp);
71   } else {
72     wets.offer(tmp);
73   }
74
75   return Math.max(divide(drys, k), divide(wets, k));
76 }
77
78 function divide(goods, k) {
79   while (goods.size() < k) {
80     const top = goods.poll();
81
82     if (!top) break;
83
84     const dry_sum = top.shift();
85
86     if (top.length <= 1) {
87       return dry_sum;
88     }
89
90     let splitIdx = 0;
91     let splitHalf = 0;
92     let half = 0;
93     let min = dry_sum;
94
95     for (let i = 1; i < top.length; i++) {
96       const val = top[i - 1];
97       half += val;
98
99       const max = Math.max(half, dry_sum - half);
100      if (max < min) {
101        min = max;
102        splitIdx = i;
103        splitHalf = half;
104      }
105    }
106
107    const halfDry = [splitHalf, ...top.slice(0, splitIdx)];
108    goods.offer(halfDry);
109
110    const otherhalfDry = [dry_sum - splitHalf, ...top.slice(splitIdx)];
111    goods.offer(otherhalfDry);
112  }
113
114  return goods.peek()[0];
115 }
116
117 // 基于堆实现优先队列
118 class PriorityQueue {
119   constructor(cpr) {
120     this.queue = [];
121     this.cpr = cpr;
122   }
123
124   swap(a, b) {
125     const tmp = this.queue[a];
126     this.queue[a] = this.queue[b];
127     this.queue[b] = tmp;
128   }
129
130   // 上浮
131   swim() {
132     let c = this.queue.length - 1;
133
134     while (c >= 1) {
135       const f = Math.floor((c - 1) / 2);
136
137       if (this.cpr(this.queue[c], this.queue[f]) < 0) {
138         this.swap(c, f);
139         c = f;
140       } else {
141         break;
142       }
143     }
144   }
145
146   // 入队
147   offer(val) {
148     this.queue.push(val);
149     this.swim();
150   }
151
152   // 下沉
153   sink() {
154     let f = 0;
155
156     while (true) {
157       let c1 = 2 * f + 1;
158       let c2 = c1 + 1;
159
160       let c;
161       let val1 = this.queue[c1];
162       let val2 = this.queue[c2];
163       if (val1 && val2) {
164         c = this.cpr(val1, val2) < 0 ? c1 : c2;

```

```

132 let c = this.queue.length - 1;
133
134 while (c >= 1) {
135     const f = Math.floor((c - 1) / 2);
136
137     if (this.cpr(this.queue[c], this.queue[f]) < 0) {
138         this.swap(c, f);
139         c = f;
140     } else {
141         break;
142     }
143 }
144 }
145
146 // 入队
147 offer(val) {
148     this.queue.push(val);
149     this.swim();
150 }
151
152 // 下沉
153 sink() {
154     let f = 0;
155
156     while (true) {
157         let c1 = 2 * f + 1;
158         let c2 = c1 + 1;
159
160         let c;
161         let val1 = this.queue[c1];
162         let val2 = this.queue[c2];
163         if (val1 && val2) {
164             c = this.cpr(val1, val2) < 0 ? c1 : c2;
165         } else if (val1 && !val2) {
166             c = c1;
167         } else if (!val1 && val2) {
168             c = c2;
169         } else {
170             break;
171         }
172
173         if (this.cpr(this.queue[c], this.queue[f]) < 0) {
174             this.swap(c, f);
175             f = c;
176         } else {
177             break;
178         }
179     }
180 }
181
182 // 出队
183 poll() {
184     this.swap(0, this.queue.length - 1);
185     const res = this.queue.pop();
186     this.sink();
187     return res;
188 }
189
190 peek() {
191     return this.queue[0];
192 }
193
194 size() {
195     return this.queue.length;
196 }
197 }

```

Java算法源码

```

1 import java.util.LinkedList;
2 import java.util.PriorityQueue;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int n = sc.nextInt();
10
11         int[] goods = new int[n];
12         for (int i = 0; i < n; i++) {
13             goods[i] = sc.nextInt();
14         }
15
16         int[] types = new int[n];
17         for (int i = 0; i < n; i++) {
18             types[i] = sc.nextInt();
19         }
20
21         int k = sc.nextInt();
22
23         System.out.println(getResult(n, goods, types, k));
24     }

```

伏城之外 已关注

2 6 13

专栏目录

已订阅

```
1 import java.util.LinkedList;
2 import java.util.PriorityQueue;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int n = sc.nextInt();
10
11         int[] goods = new int[n];
12         for (int i = 0; i < n; i++) {
13             goods[i] = sc.nextInt();
14         }
15
16         int[] types = new int[n];
17         for (int i = 0; i < n; i++) {
18             types[i] = sc.nextInt();
19         }
20
21         int k = sc.nextInt();
22
23         System.out.println(getResult(n, goods, types, k));
24     }
25
26     public static int getResult(int n, int[] goods, int[] types, int k) {
27         PriorityQueue<LinkedList<Integer>> dries = new PriorityQueue<((a, b) -> b.get(0) - a.get(0));
28         PriorityQueue<LinkedList<Integer>> wets = new PriorityQueue<((a, b) -> b.get(0) - a.get(0));
29
30         boolean isDry = true;
31         LinkedList<Integer> tmp = new LinkedList<>();
32         int sum = 0;
33
34         for (int i = 0; i < n; i++) {
35             if (types[i] == 0) {
36                 if (isDry) {
37                     tmp.add(goods[i]);
38                     sum += goods[i];
39                 } else {
40                     tmp.addFirst(sum);
41                     wets.offer(tmp);
42
43                     tmp = new LinkedList<>();
44                     sum = 0;
45
46                     tmp.add(goods[i]);
47                     sum += goods[i];
48
49                     isDry = true;
50                 }
51             } else {
52                 if (isDry) {
53                     tmp.addFirst(sum);
54                     dries.offer(tmp);
55
56                     tmp = new LinkedList<>();
57                     sum = 0;
58
59                     tmp.add(goods[i]);
60                     sum += goods[i];
61
62                     isDry = false;
63                 } else {
64                     tmp.add(goods[i]);
65                     sum += goods[i];
66                 }
67             }
68         }
69
70         tmp.addFirst(sum);
71         if (isDry) {
72             dries.offer(tmp);
73         } else {
74             wets.offer(tmp);
75         }
76
77         int dry_min = divide(dries, k);
78         int wet_min = divide(wets, k);
79
80         return Math.max(dry_min, wet_min);
81     }
82
83     public static int divide(PriorityQueue<LinkedList<Integer>> goods, int k) {
84         while (goods.size() < k) {
85             LinkedList<Integer> top = goods.poll();
86
87             if (top == null) break;
88
89             int dry_sum = top.removeFirst();
90
91             if (top.size() == 1) {
```



```

57         sum = 0;
58
59         tmp.add(goods[i]);
60         sum += goods[i];
61
62         isDry = false;
63     } else {
64         tmp.add(goods[i]);
65         sum += goods[i];
66     }
67 }
68 }
69
70 tmp.addFirst(sum);
71 if (isDry) {
72     drys.offer(tmp);
73 } else {
74     wets.offer(tmp);
75 }
76
77 int dry_min = divide(drys, k);
78 int wet_min = divide(wets, k);
79
80 return Math.max(dry_min, wet_min);
81 }
82
83 public static int divide(PriorityQueue<LinkedList<Integer>> goods, int k) {
84     while (goods.size() < k) {
85         LinkedList<Integer> top = goods.poll();
86
87         if (top == null) break;
88
89         int dry_sum = top.removeFirst();
90
91         if (top.size() == 1) {
92             return dry_sum;
93         }
94
95         int splitIdx = 0;
96         int splitHalf = 0;
97         int half = 0;
98         int min = dry_sum;
99
100        for (int i = 1; i < top.size(); i++) {
101            int val = top.get(i - 1);
102            half += val;
103
104            int max = Math.max(half, dry_sum - half);
105            if (max < min) {
106                min = max;
107                splitIdx = i;
108                splitHalf = half;
109            }
110        }
111
112        LinkedList<Integer> halfList = new LinkedList<>();
113        halfList.add(splitHalf);
114        halfList.addAll(top.subList(0, splitIdx));
115        goods.offer(halfList);
116
117        LinkedList<Integer> otherHalfList = new LinkedList<>();
118        otherHalfList.add(dry_sum - splitHalf);
119        otherHalfList.addAll(top.subList(splitIdx, top.size()));
120        goods.offer(otherHalfList);
121    }
122
123    return goods.peek().get(0);
124 }
125 }

```

Python算法源码

```

1 import queue
2
3 # 输入获取
4 n = int(input())
5 goods = list(map(int, input().split()))
6 types = list(map(int, input().split()))
7 k = int(input())
8
9 class Obj:
10     def __init__(self, sumV, goods):
11         self.sumV = sumV
12         self.goods = goods
13
14     def __lt__(self, other):
15         return self.sumV > other.sumV
16
17 # 算法入口
18 def getResult(n, goods, types, k):
19     drys = queue.PriorityQueue()
20     wets = queue.PriorityQueue()
21

```

```

1 import queue
2
3 # 输入获取
4 n = int(input())
5 goods = list(map(int, input().split()))
6 types = list(map(int, input().split()))
7 k = int(input())
8
9 class Obj:
10     def __init__(self, sumV, goods):
11         self.sumV = sumV
12         self.goods = goods
13
14     def __lt__(self, other):
15         return self.sumV > other.sumV
16
17 # 算法入口
18 def getResult(n, goods, types, k):
19     drys = queue.PriorityQueue()
20     wets = queue.PriorityQueue()
21
22     isDry = True
23     tmp = []
24     sumV = 0
25
26     for i in range(n):
27         if types[i] == 0:
28             if isDry:
29                 tmp.append(goods[i])
30                 sumV += goods[i]
31             else:
32                 wets.put(Obj(sumV, tmp))
33
34                 tmp = []
35                 sumV = 0
36
37                 tmp.append(goods[i])
38                 sumV += goods[i]
39
40                 isDry = True
41         else:
42             if isDry:
43                 drys.put(Obj(sumV, tmp))
44
45                 tmp = []
46                 sumV = 0
47
48                 tmp.append(goods[i])
49                 sumV += goods[i]
50
51                 isDry = False
52             else:
53                 tmp.append(goods[i])
54                 sumV += goods[i]
55
56     if isDry:
57         drys.put(Obj(sumV, tmp))
58     else:
59         wets.put(Obj(sumV, tmp))
60
61     return max(divide(drys, k), divide(wets, k))
62
63 def divide(goods, k):
64     while goods.qsize() < k:
65         top = goods.get()
66
67         if top is None:
68             break
69
70         dry_sum = top.sumV
71
72         if len(top.goods) <= 1:
73             return dry_sum
74
75         splitIdx = 0
76         splitHalf = 0
77         half = 0
78         minV = dry_sum
79
80         for i in range(1, len(top.goods)):
81             val = top.goods[i - 1]
82             half += val
83
84             maxV = max(half, dry_sum - half)
85             if maxV < minV:
86                 minV = maxV
87                 splitIdx = i
88                 splitHalf = half
89
90         halfDry = Obj(splitHalf, top.goods[:splitIdx])
91         goods.put(halfDry)
92

```

复制


```

43         drys.put(Obj(sumV, tmp))
44
45         tmp = []
46         sumV = 0
47
48         tmp.append(goods[i])
49         sumV += goods[i]
50
51         isDry = False
52     else:
53         tmp.append(goods[i])
54         sumV += goods[i]
55
56     if isDry:
57         drys.put(Obj(sumV, tmp))
58     else:
59         wets.put(Obj(sumV, tmp))
60
61     return max(divide(drys, k), divide(wets, k))
62
63
64 def divide(goods, k):
65     while goods.qsize() < k:
66         top = goods.get()
67
68         if top is None:
69             break
70
71         dry_sum = top.sumV
72
73         if len(top.goods) <= 1:
74             return dry_sum
75
76         splitIdx = 0
77         splitHalf = 0
78         half = 0
79         minV = dry_sum
80
81         for i in range(1, len(top.goods)):
82             val = top.goods[i - 1]
83             half += val
84
85             maxV = max(half, dry_sum - half)
86             if maxV < minV:
87                 minV = maxV
88                 splitIdx = i
89                 splitHalf = half
90
91         halfDry = Obj(splitHalf, top.goods[:splitIdx])
92         goods.put(halfDry)
93
94         otherHalfDry = Obj(dry_sum - splitHalf, top.goods[splitIdx:])
95         goods.put(otherHalfDry)
96
97     return goods.queue[0].sumV
98
99
100 # 算法调用
101 print(getResult(n, goods, types, k))

```