浅谈邻项交换排序的应用以及需要注意的问题

武汉外国语学校 游宇凡

2018 年 11 月 18 日

摘要

邻项交换排序是一种常见的贪心算法，通过比较两个相邻元素交换前后的优劣对整个序列进行排序，从而使得这个序列成为题目所求的最优解。然而，邻项交换排序的应用有一些需要注意的地方，稍有不慎便会成为一个错误的算法。

目录

第一部分 算法简介

1 例题引入

1.1 题目来源

NOIP2012提高组D1T2 国王游戏

1.2 题目描述

恰逢 H 国国庆，国王邀请 n 位大臣来玩一个有奖游戏。首先，他让每个大臣在左、右手上面分别写下一个整数，国王自己也在左、右手上各写一个整数。然后，让这 n 位大臣排成一排，国王站在队伍的最前面。排好队后，所有的大臣都会获得国王奖赏的若干金币，每位大臣获得的金币数分别是：排在该大臣前面的所有人的左手上的数的乘积除以他自己右手上的数，然后向下取整得到的结果。

国王不希望某一个大臣获得特别多的奖赏，所以他想请你帮他重新安排一下队伍的顺序，使得获得奖赏最多的大臣，所获奖赏尽可能的少。注意，国王的位置始终在队伍的最前面。

1.3 输入格式

第一行包含一个整数 n ，表示大臣的人数。

第二行包含两个整数 a 和 b ，之间用一个空格隔开，分别表示国王左手和右手上的整数。

接下来 n 行，每行包含两个整数 a 和 b ，之间用一个空格隔开，分别表示每个大臣左手和右手上的整数。

1.4 输出格式

一个整数，表示重新排列后的队伍中获奖赏最多的大臣所获得的金币数。

1.5 数据范围

对于 20% 的数据，有 $1 \leq n \leq 10, 0 < a, b < 8$；
对于 40% 的数据，有 $1 \leq n \leq 20, 0 < a, b < 8$；
对于 60% 的数据，有 $1 \leq n \leq 100$；
对于 60% 的数据，保证答案不超过 10^9；
对于 100% 的数据，有 $1 \leq n \leq 1000, 0 < a, b < 10000$ 。

2 使用邻项交换排序解决问题

选取相邻的两个大臣 i 和 j （j 此时在 i 后一个），分别用 a_i, b_i, a_j, b_j 表示这两位大臣左手上和右手上的数字，设这两位大臣前面的所有大臣左手上的数乘积为 k 。

此时，若调整这两名大臣的顺序，对前面和后面的大臣都不会造成影响，因此我们只要使得这两位大臣中获得较多奖赏的那位获得的奖赏尽量少即可。

当 i 在前 j 在后时，这个值为 $\max(\frac{k}{b_i}, \frac{ka_i}{b_j})$ 。

当 j 在前 i 在后时，这个值为 $\max(\frac{k}{b_j}, \frac{ka_j}{b_i})$ 。

因此，若 $\max(\frac{k}{b_i}, \frac{ka_i}{b_j}) > \max(\frac{k}{b_j}, \frac{ka_j}{b_i})$ ，就需要交换 i 和 j 。

实际上，由于 $\frac{ka_i}{b_j} \geq \frac{k}{b_j}, \frac{ka_j}{b_i} \geq \frac{k}{b_i}$ ，只需比较 $\frac{ka_i}{b_j}$ 和 $\frac{ka_j}{b_i}$ ，也就只需比较 $\frac{a_i}{b_j}$ 和 $\frac{a_j}{b_i}$ ，即：若 $a_i b_i > a_j b_j$，就要交换 i 和 j 。

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N=1010;

struct Node
{
        int a,b;
        bool operator<(Node& y)
        {
                return a*b<y.a*y.b;
        }
} dc[N];

long long n,ans,k;

int main()
{
```

```
        cin>>n;
        for (int i=0;i<=n;++i)
        {
                cin>>dc[i].a>>dc[i].b;
        }
        sort(dc+1,dc+n+1);
        k=dc[0].a;
        for (int i=1;i<=n;++i)
        {
                ans=max(ans,k/dc[i].b);
                k*=dc[i].a;
        }
        cout<<ans;
        return 0;
}
```

使用以上代码即可得到60分，而AC此题需要使用高精度乘除法，不在本文讨论范围内。

第二部分　使用邻项交换排序需要注意的问题

3　另一道例题

3.1　题目来源

洛谷P2123 皇后游戏

3.2　题目描述

皇后有 n 位大臣，每位大臣的左右手上面分别写上了一个正整数。恰逢国庆节来临，皇后决定为 n 位大臣颁发奖金，其中第 i 位大臣所获得的奖金数目为第 i−1 位大臣所获得奖金数目与前 i 位大臣左手上的数的和的较大值再加上第 i 位大臣右手上的数。

形式化地讲：我们设第 i 位大臣左手上的正整数为 a_i ，右手上的正整数为 b_i ，

则第 i 位大臣获得的奖金数目为 c_i 可以表达为：

$$c_i = \begin{cases} a_1 + b_1 & i = 1 \\ \max(c_{i-1}, \sum_{j=1}^{i} a_j) + b_i & 2 \le i \le n \end{cases}$$

当然，吝啬的皇后并不希望太多的奖金被发给大臣，所以她想请你来重新安排一下队伍的顺序，使得获得奖金最多的大臣，所获奖金数目尽可能的少。

注意：重新安排队伍并不意味着一定要打乱顺序，我们允许不改变任何一位大臣的位置。

3.3 输入格式

第一行包含一个正整数 T ，表示测试数据的组数。

接下来 T 个部分，每个部分的第一行包含一个正整数 n ，表示大臣的数目。

每个部分接下来 n 行中，每行两个正整数，分别为 a_i 和 b_i ，含义如上文所述。

3.4 输出格式

共 T 行，每行包含一个整数，表示获得奖金最多的大臣所获得的奖金数目。

3.5 数据范围

对于全部测试数据满足：$T \leq 10, 1 \leq n \leq 20000, 1 \leq a_i, b_i \leq 10^9$ 。

4 尝试用邻项交换排序解决问题

还是选取相邻的两个大臣 i 和 j （j 此时在 i 后一个），交换 i 和 j 对前面的大臣无影响，对后面的大臣的影响在于排在后面的那个大臣获得的奖金，需要使之尽量小。

设这两个大臣前面的所有大臣左手上的数之和为 sum ，这两个大臣的再往前一个大臣得到的奖金是 pre 。

当 i 在前 j 在后时，这个值为 $\max(\max(\text{pre}, \text{sum}+a_i)+b_i, \text{sum}+a_i+a_j)+b_j$ 。

当 j 在前 i 在后时，这个值为 $\max(\max(\text{pre}, \text{sum}+a_j)+b_j, \text{sum}+a_j+a_i)+b_i$ 。

由于 $\max(x, y)+z = \max(x+z, y+z)$ ，需要比较的就是 $\max(\text{pre}+b_i+b_j, \text{sum}+a_i+b_i+b_j, \text{sum}+a_i+a_j+b_j)$ 和 $\max(\text{pre}+b_j+b_i, \text{sum}+a_j+b_j+b_i, \text{sum}+a_j+a_i+b_i)$ ，其中 $\text{pre}+b_i+b_j = \text{pre}+b_j+b_i$ ，需要比较的就是 $\max(\text{sum}+a_i+b_i+b_j, \text{sum}+a_i+a_j+b_j)$ 和 $\max(\text{sum}+a_j+b_j+b_i, \text{sum}+a_j+a_i+b_i)$ ，再用 $\text{sum}+a_i+b_i+a_j+b_j$ 减去两边并变号，即：当 $\min(a_i, b_j) > \min(a_j, b_i)$ 时，需要交换 i 和 j。

于是，可以写出以下代码，并在洛谷上AC此题：

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N=20010;

struct Node
{
    int a,b;
    bool operator<(Node& y)
    {
        return min(a,y.b)<min(b,y.a);
    }
```

```
} dc[N];

long long ans,sum,t,n;

int main()
{
    cin>>t;
    while (t--)
    {
        cin>>n;
        for (int i=1;i<=n;++i)
        {
            cin>>dc[i].a>>dc[i].b;
        }
        sort(dc+1,dc+n+1);
        ans=sum=0;
        for (int i=1;i<=n;++i)
        {
            sum+=dc[i].a;
            ans=max(ans,sum)+dc[i].b;
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

5 hack数据

事实上，上面的做法是错误的，无法通过下面这组数据：

```
2
4
1 1
1 1
3 5
2 7
4
1 1
3 5
1 1
```

```
2 7
```

这两组数据只有大臣给出的顺序不同，但上面的代码输出为：

```
16
17
```

输出中间结果，可以发现，排列后的最终结果分别为：

```
1 1
1 1
2 7
3 5
```

和

```
1 1
3 5
1 1
2 7
```

这两种排列方式都满足 $\forall i \in [1,n), \min(a_i, b_{i+1}) \leq \min(a_{i+1}, b_i)$，但第二种方式并不是最优解。具体原因将在下文分析。

6 严格弱序

6.1 严格弱序简介

要知道为什么这种做法是错误的，首先需要了解严格弱序（strict weak ordering）。

对于一个比较运算符（用“$<$”表示此运算符，用“$!<$”表示不满足此运算符），若满足以下四个条件，则称其是满足严格弱序的：

1. $x !< x$ （非自反性）
2. 若 $x < y$，则 $y !< x$ （非对称性）
3. 若 $x < y, y < z$，则 $x < z$ （传递性）
4. 若 $x !< y, y !< x, y !< z, z !< y$，则 $x !< z, z !< x$ （不可比性的传递性）

而 C++ 标准库要求用于排序的运算符必须满足严格弱序：[1]

The sorting criterion must define strict weak ordering, which is defined by the following four properties:

1. It has to be antisymmetric.

   This means that for operator $<$: If $x < y$ is true, then $y < x$ is false.

   This means that for a predicate op(): If op(x,y) is true, then op(y,x) is false.

2. It has to be transitive.

   This means that for operator <: If $x < y$ is true and $y < z$ is true, then $x < z$ is true.

   This means that for a predicate op(): If op(x,y) is true and op(y,z) is true, then op(x,z) is true.

3. It has to be irreflexive.

   This means that for operator <: $x < x$ is always false.

   This means that for a predicate op(): op(x,x) is always false.

4. It has to have transitivity of equivalence, which means roughly: If a is equivalent to b and b is equivalent to c, then a is equivalent to c.

   This means that for operator <: If $!(a < b)\&\&!(b < a)$ is true and $!(b < c)\&\&!(c < b)$ is true then $!(a < c)\&\&!(c < a)$ is true.

   This means that for a predicate op(): If op(a,b), op(b,a), op(b,c), and op(c,b) all yield false, then op(a,c) and op(c,a) yield false.

上述做法的判断条件满足传递性，但不满足不可比性的传递性。

6.2 满足传递性的证明

命题：$\forall \begin{cases} \min(a_i, b_j) < \min(a_j, b_i) \\ \min(a_j, b_k) < \min(a_k, b_j) \end{cases}$ ，有 $\min(a_i, b_k) < \min(a_k, b_i)$。

将上式拆解成逻辑式，即证：

$$\forall \begin{cases} (a_i < a_j \vee b_j < a_j) \wedge (a_i < b_i \vee b_j < b_i) \\ (a_j < a_k \vee b_k < a_k) \wedge (a_j < b_j \vee b_k < b_j) \end{cases} \text{，有 } (a_i < a_k \vee b_k < a_k) \wedge (a_i < b_i \vee b_k < b_i)。$$

$$\text{假设原命题不成立，即 } \exists \begin{cases} (a_i < a_j \vee b_j < a_j) \wedge (a_i < b_i \vee b_j < b_i) & (1) \\ (a_j < a_k \vee b_k < a_k) \wedge (a_j < b_j \vee b_k < b_j) & (2) \\ (a_i \geq a_k \wedge b_k \geq a_k) \vee (a_i \geq b_i \wedge b_k \geq b_i) & (3) \end{cases}$$

分别讨论 (3) 式成立的两种情况：

若 $a_i \geq a_k \wedge b_k \geq a_k$，由 (2) 式得 $a_j < a_k$，进而推出 $a_j < a_i$，再由 (1) 式得 $b_j < a_j$，再由 (2) 式得到 $b_k < b_j$，所以 $b_k < b_j < a_j < a_k$，与 $b_k \geq a_k$ 矛盾，不成立。

若 $a_i \geq b_i \wedge b_k \geq b_i$，与上面类似，由 (1) 式得 $b_j < b_i$，进而推出 $b_j < b_k$，再由 (2) 式得到 $a_j < b_j$，再由 (1) 式得到 $a_i < a_j$，所以 $a_i < a_j < b_j < b_i$，与 $a_i \geq b_i$ 矛盾，不成立。

综上所述，假设不成立。

所以，$P_{i,j} = \min(a_i, b_j) < \min(a_j, b_i)$ 具有传递性。

6.3 不具有不可比性的传递性的证明

命题：$\forall \begin{cases} \min(a_i, b_j) = \min(a_j, b_i) \\ \min(a_j, b_k) = \min(a_k, b_j) \end{cases}$，有 $\min(a_i, b_k) = \min(a_k, b_i)$。

很明显，当 $a_j = b_j$ 且都很小时存在反例，如：

$$a_i = 3, b_i = 5, a_j = 1, b_j = 1, a_k = 2, b_k = 7$$

$\begin{cases} \min(3,1) = \min(1,5) \\ \min(1,7) = \min(2,1) \end{cases}$，但 $\min(3,7) \neq \min(2,5)$。

这样的反例还有很多，所以，$P_{i,j} = \min(a_i, b_j) < \min(a_j, b_i)$ 不具有不可比性的传递性。

6.4 为何会错

简单地说，$P_{i,j} = \min(a_i, b_j) < \min(a_j, b_i)$ 不满足严格弱序，不能作为 std::sort 的比较函数。

究其原因，“不具有不可比性的传递性”意味着：将序列中若干个不可比的相邻元素对（在这种比较方式下即 $\min(a_i, b_{i+1}) = \min(a_{i+1}, b_i)$ 的 i 和 i+1）互换后，可能会出现前面的元素“大于”后面的元素（在这种比较方式下即出现 $\min(a_i, b_{i+1}) > \min(a_{i+1}, b_i)$），从而使得原先的排列方式不是最优的。

7 正确解法

7.1 更加完善的贪心解法

比较相邻两项时，若 $\min(a_i, b_j) = \min(a_j, b_i)$，从全局来看，由于 a 的前缀和对答案有一定的影响，把 a 更小的放前面是更优的。从而可以得到这样一个解法：

```
#include <iostream>
#include <algorithm>

using namespace std;

const int N=20010;

struct Node
{
    int a,b;
    bool operator<(Node& y)
    {
        return min(a,y.b)==min(b,y.a)?a<y.a:min(a,y.b)<min(b,y.a);
    }
} dc[N];
```

```
long long ans,sum,t,n;

int main()
{
    cin>>t;
    while (t--)
    {
        cin>>n;
        for (int i=1;i<=n;++i)
        {
            cin>>dc[i].a>>dc[i].b;
        }
        sort(dc+1,dc+n+1);
        ans=sum=0;
        for (int i=1;i<=n;++i)
        {
            sum+=dc[i].a;
            ans=max(ans,sum)+dc[i].b;
        }
        cout<<ans<<endl;
    }
    return 0;
}
```

这个解法的正确性将在下文证明。

7.2 一个解法是否正确的判断方式

由上文的讨论可以得知，一个排序方式 $P_{i,j}$ 要是正解，需要满足两个条件：

1. 满足严格弱序。
2. $\forall P_{i,j} = true$，$\min(a_i, b_j) \leq \min(a_j, b_i)$ 。

即，可以作为比较函数，且排序完成后任意交换相邻元素均不会使答案更优。
从而可以写出下面的正解判断器：（为了减少一行显示不下的情况已删去所有缩进）

```
#include <cstdio>
#include <algorithm>

using namespace std;
```

```
bool cmp(int i,int j);

int a[10],b[10];

int main()
{
for (a[0]=1;a[0]<=6;++a[0])
{
for (b[0]=1;b[0]<=6;++b[0])
{
if (cmp(0,0))
{
printf("No irreflexivity:%d %d\n",a[0],b[0]);
}
for (a[1]=1;a[1]<=6;++a[1])
{
for (b[1]=1;b[1]<=6;++b[1])
{
if (cmp(0,1)&&min(a[0],b[1])>min(a[1],b[0]))
{
printf("Not the best:%d %d %d %d\n",a[0],b[0],a[1],b[1]);
}
for (a[2]=1;a[2]<=6;++a[2])
{
for (b[2]=1;b[2]<=6;++b[2])
{
if (cmp(0,1)&&cmp(1,2)&&!cmp(0,2))
{
printf("No transitivity:%d %d %d %d %d %d\n",a[0],b[0],a[1],b[1],a[2],b[2]);
}
if (!cmp(0,1)&&!cmp(1,0)&&!cmp(1,2)&&!cmp(2,1)&&(cmp(0,2)||cmp(2,0)))
{
printf("No transitivity of incomparability:%d %d %d %d %d %d\n",a[0],b[0],a
    [1],b[1],a[2],b[2]);
}
}
}
}
}
```

```
}
}

return 0;
}

bool cmp(int i,int j)
{
return min(a[i],b[j])==min(a[j],b[i])?a[i]<a[j]:min(a[i],b[j])<min(a[j],b[i
    ]);
}
```

运行程序，没有任何输出，说明上文所述的排序方式是一个正解。

用其它排序方式替换cmp，若没有任何输出即可作为本题的正确排序方式。

下面是几种排序方式的例子：

```
bool cmp(int i,int j)
{
        return min(a[i],b[j])==min(a[j],b[i])?b[i]>b[j]:min(a[i],b[j])<min(a
            [j],b[i]);
}
```

输出为空，是正解。

```
bool cmp(int i,int j)
{
        return min(a[i],b[j])<min(a[j],b[i]);
}
```

共输出1694行，前10行如下：

```
No transitivity of incomparability:1 2 1 1 2 1
No transitivity of incomparability:1 2 1 1 2 2
No transitivity of incomparability:1 2 1 1 2 3
No transitivity of incomparability:1 2 1 1 2 4
No transitivity of incomparability:1 2 1 1 2 5
No transitivity of incomparability:1 2 1 1 2 6
No transitivity of incomparability:1 2 1 1 3 1
No transitivity of incomparability:1 2 1 1 3 2
No transitivity of incomparability:1 2 1 1 3 3
No transitivity of incomparability:1 2 1 1 3 4
```

```
bool cmp(int i,int j)
{
        return min(a[i],b[j])==min(a[j],b[i])?a[i]>a[j]:min(a[i],b[j])<min(a
            [j],b[i]);
}
```

共输出280行，前10行如下：

```
No transitivity:1 2 2 1 1 1
No transitivity:1 2 2 2 1 1
No transitivity:1 2 2 3 1 1
No transitivity:1 2 2 4 1 1
No transitivity:1 2 2 5 1 1
No transitivity:1 2 2 6 1 1
No transitivity:1 2 3 1 1 1
No transitivity:1 2 3 2 1 1
No transitivity:1 2 3 3 1 1
No transitivity:1 2 3 4 1 1
```

```
bool cmp(int i,int j)
{
        return min(a[i],b[j])<=min(a[j],b[i]);
}
```

共输出883行，前10行如下：

```
No irreflexivity:1 1
No irreflexivity:1 2
No irreflexivity:1 3
No irreflexivity:1 4
No irreflexivity:1 5
No irreflexivity:1 6
No irreflexivity:2 1
No transitivity:2 1 1 1 1 2
No transitivity:2 1 1 1 1 3
No transitivity:2 1 1 1 1 4
```

第三部分　总结

在可以通过比较相邻两项得出交换或不交换一定不会更差时，可以通过邻项交换排序的方式来得到

最优解。

邻项交换排序的比较函数需要满足严格弱序，并且排序完成后任意交换相邻元素都不会更优。

使用这种算法时，一定要注意以上两点，才能得到真正正确的算法。

参考文献

[1] Nicolai M. Josuttis. The c++ standard library: a tutorial and reference, 2nd edition. pages 314–315, 2012.