Program Promenade

連載

Silly Sort

湯淺 太一（京都大学大学院情報学研究科）
yuasa@kuis.kyoto-u.ac.jp

■問題

2002年の世界大会で出題されたSilly Sortという問題を取り上げる．整数の列が与えられたときに，2つの整数を入れ換えることを繰り返して，昇順に（小さいものから順に）並べ換えることを考える．ありふれたソーティングだが，ソートに要するコストが少し変わっている．ステップ数ではなく，各ステップで入れ換える2つの整数の和を求め，その合計をコストとする．たとえば，

[3,2,1]

をバブルソート風に

[3,2,1] →[2,3,1] →[2,1,3] →[1,2,3]

と並べ換えると，コストは

(3+2)+(3+1)+(1+2)=12

となる．与えられた整数列を並べ換えるための最小のコストを求めるプログラムを作れ，というのが問題である．ただし，それぞれの整数は1以上1000未満とし，整数は互いに異なるものとする．また，整数列の長さは1以上とする．

上の例の場合，最小コストが4であることはすぐに分かる．1と3はソート完了時の最終位置にはないので，必ず1回は移動する必要があり，どのような手順でソートしてもコストは4以上となる．そして，実際にコストが4である手順が存在する．1と3を入れ換えればよい．

問題文には，サンプル入力として次の4つの例が与えられており，括弧内の値が最小コストとされている．

例1: [3,2,1] (4)
例2: [8,1,2,4] (17)
例3: [1,8,9,7,6] (41)
例4: [8,4,5,3,2,7] (34)

次章に進む前に，読者自身で例2～例4の最小コスト手順を考えてみてほしい．

■試行錯誤

まず例2について考えよう．[8,1,2,4]をソートして[1,2,4,8]にする．各整数の最終位置は，初期位置のすぐ左隣である．ただし，先頭の整数は最後尾にまわる．そこで，すぐに思いつくのは，先頭の8を後ろの整数と順に入れ換えていく手順であろう．

$[8,1,2,4] \xrightarrow{9} [1,8,2,4] \xrightarrow{10} [1,2,8,4] \xrightarrow{12} [1,2,4,8]$

各ステップのコストを矢印の上に書いておく．コストの合計は31となり，これは大きすぎる．

[8,1,2,4]を**図-1** (a)に示すグラフで表現してみよう．ここで，○は整数列中の位置を表し，矢印は，始点の位置にある整数の最終位置が，終点の位置であることを表す．このグラフを見れば，なにもわざわざ大きな8を使ってソートしなくても，最小の1を使えばよさそうである．実際，

$[8,1,2,4] \xrightarrow{3} [8,2,1,4] \xrightarrow{5} [8,2,4,1] \xrightarrow{9} [1,2,4,8]$

となって，コストは最小とされている17になる．

では，どうして例2の最小コストが17なのだろう．図-1 (a)には，輪（循環構造）が1つある．上のステップを繰り返すと輪の本数が1ずつ増えていき（**図-1** (b), (c)），最終的に輪が4つになる（**図-1** (d)）．一般に，2つの整数を入れ換えると，グラフ中の輪は1増加する（同じ輪の整数を入れ換えるとき）か，1減少する（異なる輪の整数を入れ換えるとき）かのいずれかである（**図-2**参照）．例2の場合は，最初は1つだった輪が，最後は4つになるので，ソートするためには最低3回のステップが必要である．いま，[8,1,2,4]をk回($k \geq 3$)のステップでソートしたとすると，全部で$2k$個の整数を移動したことになる．最初は8,1,2,4のいずれ

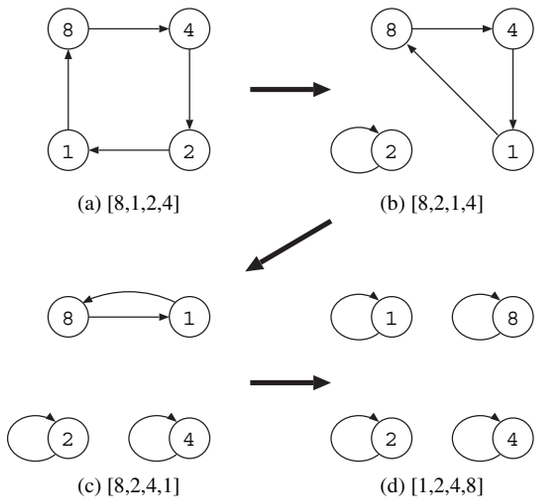

図 -1　整数列のグラフ表現

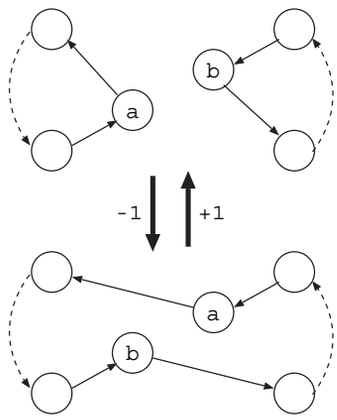

図 -2　輪の本数の増減（a と b を入れ換えると，それぞれのもとの位置から出ていた矢印は，終点は変わらないが始点が新しい位置に変わる）

も最終位置にはないので，それぞれ最低 1 回は移動する．したがって，コストは

$8+1+2+4+x_1+x_2+\cdots+x_{2k-4}$

と表せる．ここで，各 x_i は 8,1,2,4 のいずれかである．このコストが最小になるのは，k=3 かつ各 x_i が最小の 1 のときである．したがって，コストの最小値は，

$8+1+2+4+1+1=17$

となる．

一般に，長さ n の整数列

$[a_1, a_2,..., a_n]$

が与えられ，そのグラフに輪が 1 つしかないときは，コストの最小値は

$$\sum_{i=1}^{n} a_i+(n-2)\times \min_{1\le i\le n} a_i \qquad (1)$$

となる．$n \ge 2$ のとき，この式 (1) は，上の例 2 の場合とまったく同様に証明できる．n=1 のとき，整数列は最初からソート済みなので，コストは 0 である．このとき，式 (1) の値も 0 となり，式 (1) は n=1 のときにも適用できる．

以下では，

$\langle a_1, a_2,..., a_n\rangle$

によって，整数 $a_1, a_2,..., a_n$ から構成される輪を表すことにする．各 a_i の最終位置が，1 つ後ろの a_{i+1} の位置である．また，輪 R に対して，それを構成する整数の和を $\sigma(R)$，個数を $\#(R)$，最小値を $\min(R)$ と表記することにする．たとえば，図 -1 (a) の輪 R は ⟨8,4,2,1⟩ や ⟨1,8,4,2⟩ と表せ，$\sigma(R)$=15，$\#(R)$=4，$\min(R)$=1 である．また，これらの記法を使えば，式 (1) は，

$\sigma(R) + (\#(R) - 2) \times \min(R)$

と表せる．

与えられた整数列のグラフに複数の輪が存在するときは，それぞれの輪に対して式 (1) を適用して合計を求めればよさそうである．実際，例 1 のグラフには，2 つの輪 ⟨3,1⟩ と ⟨2⟩ があり，式 (1) を使えばコストはそれぞれ 4 と 0 で，その和 4 が最小コストである．しかし例 3 については，そうはいかない．輪 ⟨8,7,9,6⟩ のコストを計算すると

$(8+7+9+6)+(4-2)\times 6=42$

となり，最小コストとされる 41 を上回る．どうやら，最初から最終位置にある 1 が曲者（くせもの）である．

コストが 42 になる手順（これを T_1 とする）は次のとおりである．

$[1,8,9,7,6] \xrightarrow{15} [1,8,6,7,9] \xrightarrow{13} [1,8,7,6,9] \xrightarrow{14} [1,6,7,8,9]$

⟨8,7,9,6⟩ の最小値 6 を使ってソートするので，6 が 3 回移動している．この 6 の代わりに，1 を使ってソートすることを考える．まず 1 と 6 を入れ換える．

$[1,8,9,7,6] \xrightarrow{7} [6,8,9,7,1]$

その後，手順 T_1 の各ステップで 6 の代わりに 1 を使ってみる．

$[6,8,9,7,1] \xrightarrow{10} [6,8,1,7,9] \xrightarrow{8} [6,8,7,1,9] \xrightarrow{9} [6,1,7,8,9]$

最後に，1 をもとの位置に戻せば，ソートが完了する．

$[6,1,7,8,9] \xrightarrow{7} [1,6,7,8,9]$

手順 T_1 と比べると，最初に 1 を借りてくるコストと

最後に 1 を戻すコストが増えるが，手順 T_1 の各ステップで 6 の代わりに 1 を使った分だけコストが減少する．その結果，総コストは最小とされる 41 になる．

一般に，輪 $R=\langle a_1, a_2,..., a_n\rangle$ を，輪の外にある整数 x を使って上のようにソートする場合，コストは次のように表せる．

$$\sigma(R) + (\#(R)-2) \times \min(R)$$
$$-(\#(R)-1) \times (\min(R)-x)+2(\min(R)+x)$$
$$= \sigma(R) + \min(R) + (\#(R)+1) \times x$$

x が小さいほどこの値は小さくなるので，与えられた整数列中の最小値 s を使うことにすると，コストは

$$\sigma(R) + \min(R) + (\#(R)+1) \times s \qquad (2)$$

となる．これと式 (1) を比較して，もし式 (2) の値の方が小さければ，最小値 s を借りてきてソートし，そうでなければ，輪の中だけで整数を入れ換えてソートすればよさそうである．R が最初から最小値 s を含む場合は，式 (1) の値は必ず式 (2) より小さくなるので，輪の中だけでソートすればよさそうである．

最後の例 4 には ⟨8,7,2⟩ と ⟨4,5,3⟩ の 2 つの輪がある．それぞれについて，上のようにソートした場合のコストを計算してみよう．前者は最小値 s=2 を含んでいるので，式 (1) からコストは 19 になる．後者に対しては，式 (1) の値は 15，式 (2) の値は 23 となるので，小さい方の 15 を採用とする．合計 34 となり，これは最小とされるコストと一致する．

■プログラム

前章の検討結果をまとめると，次のようになる．

(1) 与えられた整数列にある輪を $R_1, R_2,..., R_l$ とする．また，整数列中の最小値を s とする．
(2) 各 R_i に対して，式 (1) と式 (2) の値を計算し，その小さい方を cost(R_i) とする．
(3) cost(R_i) (i=1,2,...,l) の和を求め，それを，与えられた整数列をソートするための最小コストとして採用する．

「最小コストである」と書かずに，「最小コストとして採用する」と書いたのは，このように求めたコストが最小であることを厳密に証明するのが難しいからである．少なくとも，プログラミングコンテストの制限時間内には証明できそうにない．次のことは直観的に理解できるのであろう．

- 整数列の最小値 s を輪 R が最初から含んでいる場合は，R の外部から整数を借りてくるより，R 内の入れ換えだけでソートした方がよい．
- 輪 R の外部から整数を 1 つ借りてきてソートするなら，最小値 s を借りるのがよい．その際に，s と入れ換える R の要素としては，R の最小値 min(R) がベストである．
- R の外部から複数の整数を借りてくると，s だけを借りてきた場合よりもコストが大きくなってしまう．

これらのことから，実際のコンテストの場では，上のように求めるコストが最小であると信じて（あるいは祈って），プログラムを書いて提出することになるのであろう．そして,「正解！」と返事が返ってきて「これでよかったんだ」と喜ぶことになる．

この Silly Sort の問題は，ここまでの考察が重要で，プログラミングは比較的容易である．**図 -3** に，問題を解く C 関数 `sillysort` の定義をあげる．この関数は，与えられた整数列 `seq` と，その長さ `m` を引数として受け取り，整数列をソートするための最小コストを値として返す．関数本体は，初期化を行う部分 (`initialize`) と，実際にコストを計算する部分 (`compute`) からなる．

初期化部分では，まず整数列に対するグラフを配列 `nodes` に構築する．グラフのノード（節）は，構造体

```
typedef struct {
  int value;
  int place;
} node;
```

で表現する．各 `nodes[`i`]` ($0 \le i < m$) には，`seq[`i`]` の値 x を `value` フィールドに格納し，x の最終位置（0 ～ m−1 のいずれかの値）を `place` フィールドに格納したノードを置く．各整数の最終位置を求めるために，最初の for ループで `temp` という配列を用意する．各 `temp[`i`]` ($0 \le i < m$) には，整数 `seq[`i`]` とその初期位置 i を設定しておいて，関数 `sort_by_value` を使って整数が昇順になるようにソートする（`sort_by_value` の定義は省略する）．そして 2 つ目の for ループで，ソート済みの `temp` を使って `nodes` を初期化する．最後に，整数列の最小値を `s` に設定しておく．

コストを計算するために，`nodes` を先頭から走査する．新しい輪 R（を構成するノード）に出会ったら，cost(R) を計算して `cost` に加える．コストの計算が

```
int sillysort(int m, int seq[]) {
  int cost = 0, s, i;
  node nodes[1000], temp[1000];

  /* initialize */
  for (i = 0; i < m; i++) {
    temp[i].value = seq[i];
    temp[i].place = i;
  }
  sort_by_value(m, temp);
  for (i = 0; i < m; i++) {
    nodes[i].value = seq[i];
    nodes[temp[i].place].place = i;
  }
  s = temp[0].value;

  /* compute */
  for (i = 0; i < m; i++) {
    int j = nodes[i].place;
    if (j >= 0 && j != i) {
      int n = 1, amin, sum;
      amin = sum = nodes[i].value;
      while (j != i) {
        int next = nodes[j].place;
        if (nodes[j].value < amin)
          amin = nodes[j].value;
        sum += nodes[j].value;
        n++;
        nodes[j].place = -1;
        j = next;
      }
      cost += min(sum + (n-2)*amin,
                  sum + amin + (n+1)*s);
    }
  }
  return cost;
}
```

図-3 Silly Sortを解くC関数

終わった輪に対しては，それを構成するノードを以後の nodes の走査で無視したいので，place の値を −1（負の整数なら何でもよいが）にしておく．これによって，nodes[i] が新しい輪の一部であるかどうかは，

nodes[i].place ≥ 0

かどうかで判定できる．ただし，

nodes[i].place = i

の場合は，nodes[i].value は最初から最終位置にあるのでスキップする．

cost(R) を計算するために，R のノードを nodes[i] から順にたどって，sum (=σ(R))，n (=#(R))，amin(=min(R)) を求める．そして，式 (1) と式 (2) の値を計算して，その小さい方，つまり cost(R) を cost に加える．

nodes の走査が終わると，与えられた数列をソートするための最小コストが cost に格納されているので，その値を返す．

図 -3 の定義では，グラフとの対応がつきやすいように，配列 nodes を初期化したが，nodes の代わりにソート済みの配列 temp を直接使ってそれぞれの輪のコストを求めてもよい．その場合，輪をたどる方向は，グラフの矢印と逆になるが，これは本質的でない．temp を直接使うことによって，次のメリットが生じる．

- 大きさ 1000 の整数配列 nodes を割り当てる必要がなくなる．
- nodes を初期化するための 2 つ目の for ループが不要になる．
- amin を更新するための if 文が不要になる．

ただし，図 -3 のプログラムは，メモリ使用量も実行時間もたいしたことはないので，頑張って最適化する必要はなさそうである．

■証明

整数列 A が与えられたときに，これをソートする任意の手順 T に対して，そのコストが前章で最小コストとして採用した

$$\sum_{i=1}^{l} \mathrm{cost}(R_i)$$

以上であることを証明する．ここで，A に含まれる輪を $R_1, R_2, \ldots, R_l$ とする．

以下の証明では，整数列中の位置が重要な役割を果たす．そこで，整数列 A の位置のうち，R_i (i=1,2,…,l) のノードに相当するもの全体（位置集合と呼ぶことにする）を，P_i と表記することにする．図 -3 のプログラムのように整数列を配列で表す場合は，P_i はインデックスの集合と考えればよいだろう．たとえば A が例 4 の整数列

[8,4,5,3,2,7]

で，R_1=⟨8,7,2⟩，R_2=⟨4,5,3⟩ とするとき，位置集合は，P_1={0,5,4}，P_2={1,2,3} となる．

手順 T によって，位置集合 $P_1, P_2, \ldots, P_l$ をいくつか

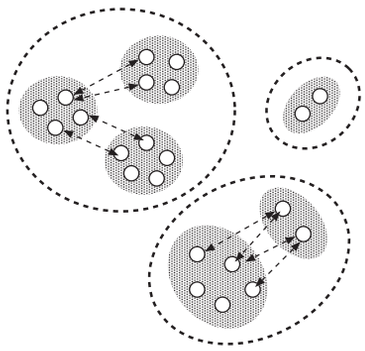

図 -4　手順 T による位置集合のグループ分け（○は整数列中の位置，灰色の円は位置集合，破線の矢印は異なる位置集合間の整数の入れ換え，破線の円が最終グループ）

のグループに分ける．手順 T を適用して A をソートする過程で，あるステップにおいて位置集合 P 内（のどこか）にあった整数と別の位置集合 P' 内にあった整数とを入れ換えたとき，P が属するグループと P' が属するグループを合併する．このように合併を繰り返して（合併するだけで分裂はしない），最終的に A のソートが完了した時点のグループを，最終グループと呼ぶことにする（**図 -4** 参照）．輪 R_i を構成する整数 x は，最初は P_i 内にある．手順 T でソートする過程で，x は P_i と同じ最終グループに属する他の位置集合（のどこか）へ移動した可能性はあるが，P_i が属さない最終グループへは移動していないはずである．また，T の各ステップで入れ換える 2 つの整数は，必ず同じ最終グループ内にある．

以下では，ある最終グループ G について考える．簡単のために，G に属する位置集合を $P_1, P_2, \ldots, P_m$ とする．T のステップのうちで G 内の整数を入れ換えるものだけを考え，あたかも輪 $R_1, R_2, \ldots, R_m$ の全体をソートする手順であるかのように見なしたものを T_G とする．T_G のコスト，つまり $R_1, R_2, \ldots, R_m$ の全体をソートするコストの下限を，式 (1) のときと類似の方法で求めることにする．m=1，つまり G がただ 1 つの位置集合 P_1 からなる場合，その最小コストが式 (1) で与えられることは「試行錯誤」の議論で明らかなので，以下では $m \geq 2$ と仮定する．

まず，T_G のステップ数 k の下限を求める．グループ G は，最初は m 個のグループ（それぞれの P_i が 1 つのグループ）だったものが，次々と合併していって 1 つのグループになったのであった．グループ X

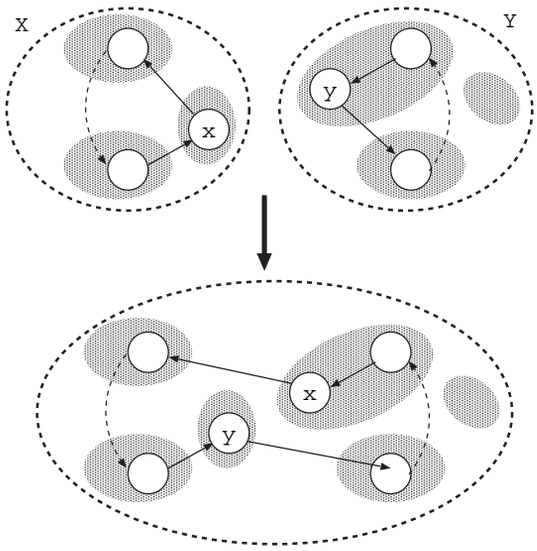

図 -5　グループの合併と輪の本数（破線の円は，過渡的なグループ）

内の x と，グループ Y 内の y を入れ換えて，X と Y を合併したときを考える（**図 -5** 参照）．合併前は，x が属する輪（のすべての構成要素）は X 内にあり，y が属する輪は Y 内にあるので，この 2 つは異なる輪である．したがって，x と y を入れ換えることによって G 内の輪の本数は 1 減少する．合併は全部で $m-1$ 回起きるので，その回数だけ輪の数は減少する．一方，最初の輪の数は m 個であり，ソート完了時には輪の数が $\sum_{i=1}^{m} \#(R_i)$ 個になる．T_G の 1 回のステップでは，輪の数は高々 1 しか増加しない（同じ輪の 2 つの整数を入れ換えたとき）ので，ステップ数 k は，

$$\sum_{i=1}^{m} \#(R_i) - m + 2(m-1) = \sum_{i=1}^{m} \#(R_i) + m - 2 \qquad (3)$$

以上である．

G は $m-1$ 回の合併によって作られたのだが，合併の際に入れ換えた整数の組を

$$(x_1, y_1), (x_2, y_2), \ldots, (x_{m-1}, y_{m-1})$$

としよう．これら $2(m-1)$ 個の整数の中には，各 R_i (i=1,2,..., m) の要素が少なくとも 1 つ含まれている．位置集合 P_i 1 つだけからなるグループが他と合併する際に，P_i から入れ換えによって出て行く整数は，必ず R_i の要素だからである．そのような整数を a_i とすると，$(m-1)$ 回の合併に要したコストは，

$$\sum_{i=1}^{m} a_i + \sum_{i=1}^{m-2} b_i \qquad (4)$$

と表せる．ここで，各 b_i (i=1, 2,..., $m-2$) は，$R_1, R_2, \ldots, R_m$ のいずれかの要素である．b_i は，$a_1, a_2, \ldots, a_m$ のいずれかと一致するかもしれないし，それら以外かもしれない．$b_1, b_2, \ldots, b_{m-2}$ のうちで，$a_1, a_2, \ldots, a_m$ のいずれとも一致しないものを $c_1, c_2, \ldots, c_u$ ($0 \le u \le m-2$) とする．$a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_u$ のそれぞれは，合併によって，最初にあった位置集合とは異なる位置集合に移動する．だから，ソートが完了する前に，もとの位置集合に戻らなければならない．戻るためのコストは，少なくとも

$$\sum_{i=1}^{m} a_i + \sum_{i=1}^{u} c_i \qquad (5)$$

である．$R_1, R_2, \ldots, R_m$ を構成する整数のうち，$a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_u$ 以外のものは，最終位置へ移動するために少なくとも 1 回は移動する（最初から最終位置にあった整数は，それ 1 つでいずれかの輪 R_i を構成していたはずなので，$a_1, a_2, \ldots, a_m$ のいずれかと一致する）．それらのコストの総和は

$$\sum_{i=1}^{m} \sigma(R_i) - \sum_{i=1}^{m} a_i - \sum_{i=1}^{u} c_i \qquad (6)$$

となる．

ここまでのコスト計算には，合計

$$(m+m-2)+(m+u)+\left(\sum_{i=1}^{m} \#(R_i) - m - u\right)$$
$$= \sum_{i=1}^{m} \#(R_i) + 2(m-1)$$

個の整数移動がカウントされている．T_G のステップ数を k としたので，合計 $2k$ 個の整数移動が起きているはずである．その差を v とすると，

$$v = 2k - \left(\sum_{i=1}^{m} \#(R_i) + 2(m-1)\right)$$

であり，k を式 (3) で置き換えると，

$$v \ge \sum_{i=1}^{m} \#(R_i) - 2$$

となる．移動がまだカウントされていない整数を $d_1, d_2, \ldots, d_v$ とすると，T_G のコストは，

$$\text{式 (4)} + \text{式 (5)} + \text{式 (6)} + \sum_{i=1}^{v} d_i$$
$$= \sum_{i=1}^{m} a_i + \sum_{i=1}^{m-2} b_i + \sum_{i=1}^{m} \sigma(R_i) + \sum_{i=1}^{v} d_i$$
$$\ge \sum_{i=1}^{m} (\sigma(R_i) + \min(R_i) + (\#(R_i)+1) \times a_0) - 4a_0$$

となる．ここで a_0 は，G 内の最小の整数である．一般性を失うことなく，a_0 が R_1 の構成要素であったとすると，$\min(R_1)=a_0$ だから，上の最後の式は

$$\sigma(R_1) + (\#(R_1) - 2) \times \min(R_1)$$
$$+ \sigma(R_2) + \min(R_2) + (\#(R_2) + 1) \times a_0$$
$$\ldots$$
$$+ \sigma(R_m) + \min(R_m) + (\#(R_m) + 1) \times a_0 \qquad (7)$$

となる．これが T_G のコストの下限であり，これは

$$\sum_{i=1}^{m} \mathrm{cost}(R_i) \qquad (8)$$

以上である．このことは，すべての最終グループ G（位置集合 1 つだけからなるグループでも）に対して成り立つので，T の総コストも，前章で採用した最小コスト以上になることがいえる．

上の証明では，与えられた数列 A の位置集合を手順 T がどのようにグループ分けするかに焦点をあてて，グループごとにソートに要するコストの下限（式 (7)）を求め，それらの合計が，前章で採用した最小コスト以上になることを示した．手順が異なれば，一般にグループ分けも異なる．グループ分けが異なれば，下限の合計も異なる．その合計が，与えられた数列 A をソートするための最小コストとなるグループ分けは，次のものである．A に含まれるそれぞれの輪 R に対して，R が A の最小値 s を含まず，かつ式 (1) の値が式 (2) よりも小さい場合は，R（の位置集合）だけで 1 つのグループ（最終グループ）とする．そして，その他の輪（の位置集合）すべてを 1 つのグループとする．この最後のグループについては，式 (7) における a_0 は s と一致するので，式 (7) の値は式 (8) と一致する．

（平成 16 年 5 月 10 日受付）