第 3 章

整列と貪欲法

2017-10-11 Wed

概要

ある基準に従ってデータを並べ替えることを整列 (sort) という．ほとんどの言語の標準ライブラリでは，整列の方法が提供されているので，まずはそれの使い方を習得しよう．この資料では整列された結果で得られれば良いという立場をとるが，整列の手法そのものに興味がある場合は，アルゴリズムの教科書を参照されたい．
データの整列は，問題解決の道具として活躍する場合もある．それらには，最適化問題や配置問題など，整列とは関係がない見た目の問題も含まれる．

3.1 数値と文字列の整列 (sort)

C++ と Ruby では，標準関数として `sort` や `sort!`が用意されている．整列に要する計算量は列の長さを N として $O(N \log N)$ である．[*1] 通常は，N に対して $\log N$ は小さいため，整列の計算量が問題になることは少ない．

C++

```cpp
#include <algorithm>
#include <iostream>
using namespace std;
int A[5] = {3,5,1,2,4};
int main() {
    sort(A,A+5); // 半開区間 [l,r) で指定する．sort(&A[0], &A[5]) と同じ意味
    ... // cout に A を出力してみよう
}
```

配列を整列させる範囲を指定するには，`&A[0]` のように配列の要素を指すポインタを用いる．一次元配列の場合に単に `A` と書けば，先頭要素を指すポインタに自動的に変換される．すなわち，`sort(A,A+5)` は `sort(&A[0], &A[5])` と同じ意味であり，`A[0]` から `A[4]` まで (端点含む) が整列される．`sort(A,A+3);` などと，一部の区間のみを指定して整列することもできる．

[*1] ただし実装とデータ次第で $O(N^2)$ かかるものがあることに注意．

Ruby

```ruby
a = [3,5,1,2,4]
a.sort!
p a
```

例題 **Sort II** (AOJ)

与えられた n 個の数字を昇順に並び替えて出力するプログラムを作成せよ.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10029

回答例:

C++

```cpp
int N, A[1000000+10];
int main() {
    cin >> N;
    for (int i=0; i<N; ++i) cin >> A[i]; // A の入力
    ... // A をソートする
    for (int i=0; i<N; ++i)
        cout << (i?" ":"") << A[i]; // A の出力
    cout << endl;
}
```

■文字列の整列　標準ライブラリの sort 関数は，数値だけでなく文字列 string の整列を行うことも出来る．一般に，比較演算子<が定義されている要素の整列が可能である．

例題 **Finding Minimum String** (AOJ)

N 個の小文字のアルファベットのみからなる文字列の，辞書順で先頭の文字列を求める．問題文中に記述がないが N は 1000 を越えない．

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=10021

補足: 辞書式順序は，大文字小文字が混ざる場合は C++ の std::string の比較演算子の比較順序とは異なる．(が，今回は小文字のみなのでその心配はない)

C++

```cpp
#include <algorithm> // sort のため
#include <string> // string (文字列) のため
#include <iostream>
using namespace std;
string A[1000];
int N;
int main() {
    cin >> N;
    if (N > 1000) abort();
```

```
10      for (int i=0; i<N; ++i) cin >> A[i];
11      ... // 整数の時と同様に A を整列 (sort) する
12      cout << A[0] << endl;
13  }
```

Ruby

```
1  N = gets.to_i
2  A = []
3  (1..N).each {
4    A << gets.chomp
5  }
6  ... // 整数の時と同様に A を整列 (sort) する
7  puts A[0]
```

3.2 ペアと整列 (pair)

3.2.1 ペア

ペア (組み) の表現を導入する．⟨ 開始時刻, 終了時刻 ⟩ や ⟨ 身長, 体重 ⟩，⟨ 学籍番号, 点数 ⟩ など，現実世界の情報にはペアとして表現することが自然なものも多い．

整数のペアは以下のように構造体を用いて表現できる．

C++

```
1  struct pair {
2      int first;
3      int second;
4  };
```

同様の使い勝手のものが標準ライブラリに用意されていて pair<int,int>として利用可能であり，こちらの利用を勧める．int 以外のペアも作れるためである．ペアの 2 つの要素を同時に初期化する略記法として，make_pair が用意されている．

C++

```
1  #include <utility> // pair のため
2  #include <iostream>
3  using namespace std;
4  int main() {
5    pair<int,int> a(2,4); // 整数のペア
6    cout << a.first << '␣' << a.second << endl; // 2 4 と表示
7
8    a.first = 3;
9    a.second = 5;
10   cout << a.first << '␣' << a.second << endl; // 3 5 と表示
11
12   a = make_pair(10, -30);
13   cout << a.first << '␣' << a.second << endl; // 10 -30 と表示
14
15   pair<double,char> b; // 小数と文字のペア
```

```
16    b.first = 0.5;
17    b.second = 'X';
18    cout << b.first << '␣' << b.second << endl; // 0.5 X と表示
19  }
```

ruby の場合は配列を手軽に使えるので，取り立てて pair を区別せずに要素数 2 の配列を使う．

Ruby

```
1  a = [3,5]
2  p a # [3,5] と表示
3  b = [0.5,"X"]
4  p b # [0.5,"X"] と表示
```

3.2.2 ペアの配列と整列

続いて，ペアの配列を扱う．たとえば，一人の身長と体重をペアで表現する時に，複数の人の身長と体重のデータはペアの配列と対応づけることができる．前回，整数の配列を整列 (sort) したように，ペアの配列も整列することができる．標準では，第一要素が異なれば第一要素で順序が決まり，第一要素が同じ時には第二要素が比較される．

C++

```
1  #include <utility> // pair のため
2  #include <algorithm> // sort のため
3  #include <iostream>
4  using namespace std;
5  int main() {
6    pair<int,int> a[3]; // 整数のペア
7    a[0] = make_pair(170,60);
8    a[1] = make_pair(180,90);
9    a[2] = make_pair(170,65);
10
11   for (int i=0; i<3; ++i) // a[0] から a[2] まで表示
12     cout << a[i].first << '␣' << a[i].second << endl;
13   // 170 60
14   // 180 90
15   // 170 65 と表示されるはず
16
17   sort(a, a+3); // a[0] から a[2] まで整列
18
19   for (int i=0; i<3; ++i) // a[0] から a[2] まで表示
20     cout << a[i].first << '␣' << a[i].second << endl;
21   // 170 60
22   // 170 65
23   // 180 90 と表示されるはず
24 }
```

Ruby

```
1  a = [];
2  a << [3,5]
```

```
3    a << [2,9]
4    a << [3,6]
5
6    p a # [[3, 5], [2, 9], [3, 6]] と表示
7    p a[0] # [3, 5] と表示
8
9    a.sort!
10
11   p.a # [[2, 9], [3, 5], [3, 6]] と表示
```

3.3 貪欲法 (greedy algorithms)

整列の重要な応用として貪欲法を紹介する．

3.3.1 整数の整列と貪欲な選択

問題 カントリーロード (UTPC 2008)

カントリーロードと呼ばれるまっすぐな道に沿って， 家がまばらに建っている． 指定された数までの発電機と電線を使って全ての家に給電したい．家に電気が供給されるにはどれかの発電機に電線を介してつながっていなければならず， 電線には長さに比例するコストが発生する． できるだけ電線の長さの総計が短くなるような発電機 および電線の配置を求める．

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2104

考察課題 なぜ正しい回答を得られるかの説明をまとめよ

ヒント: 発電機が一つなら，端から端まで全ての家を電線でつなぐ必要がある．発電機が 2 つなら，端から端まで全ての家を電線でつないだ状態から，家と家の間の一箇所に電線を引かずに節約することができる．つまり，一番長い一箇所を節約するのが得．発電機が 3 つなら，端から端まで全ての家を電線でつないだ状態から，家と家の間で一番長い部分と二番目に長い部分には電線を引かずに節約することができる．...

方針の確認

プログラムを書く前に Sample Input を手で解いてみよう

回答例

C++

```
1  int T, N, K, X[100000+10], A[100000+10];
2  int main() {
3      cin >> T; // データセットの数を読み込む
4      for (int t=0; t<T; ++t) {
5          ... // 家と発電機の数を読み込む
6          for (int i=0; i<N; ++i) cin >> X[i]; // 家の位置を入力
7          for (int i=0; i+1<N; ++i) A[i] = X[i+1]-X[i]; // 家と家の間
```

```
8            ... // 配列 A を整列する
9            ... // 配列 A の先頭 max(0,N-1-(K-1)) 個の和を出力する
10       }
11   }
```

よくあるバグ

発電機の個数が家の数を越えて増えても節約できる区間は増えない

このような解法が貪欲法 (greedy) と呼ばれる意味は，節約する区間の「組み合わせ」を考えることなく，(長さ順に並べた) 単独の区間を順に一つづつ見て閾値を越えるまで節約することを貪欲に決める点である．

3.3.2 ペアの整列と選択

問題 **Princess's Marriage** (模擬国内予選 2008)

護衛を上手に雇って，道中に襲われる人数の期待値を最小化する．護衛は 1 単位距離あたり 1 の金額で，予算がある限り，自由に雇える．

入力は，区間の数 N, 予算 M に続いて，距離と 1 単位距離あたりの襲撃回数の期待値のペア ⟨D,P⟩ が N 個.

http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2019

考察課題 問題の入力と計算量のオーダに関して説明せよ

考え方: せっかく護衛を雇うなら，もっとも危険な区間を守ってもらうのが良い．一番危険な道を選び，予算がある限りそこに護衛を雇う．予算の残額で道の区間全てをカバーできない場合は，安全になる区間と，危険なままの区間が生ずる．予算が残っている限り，2 番目に危険な道，3 番目に危険な道の順に同様に繰り返す．残った危険な区間について，期待値の和が答えとなる．この計算課程では，⟨ 危険度, 長さ ⟩ のペアで道を表現し，危険な順に整列しておくと便利である. (ここで危険度は，距離 1 移動する間に襲われる回数の期待値を表す)

方針の確認

プログラムを書く前に Sample Input を手で解いてみよう

回答例 (入力と整列):

C++

```
1  int N, M;
2  pair<int,int> PD[10010];
3  int main() {
4    while (cin >> N >> M && N) {
5      int d, p;
6      for (int i=0; i<N; ++i) {
7        cin >> d >> p;
8        PD[i] = make_pair(p, d);
```

```
9         // PD[i].first は道 i の危険度
10        // PD[i].second は道 i の長さ
11      }
12      ... // PD を大きい順に整列しよう
13      // 整列がうまく行ったか，PD を表示してみよう
14      // うまく整列できたら，次は答えを計算しよう
15    }
16  }
```

Ruby

```
1  while line = gets
2    pN, pM = line.split("␣").map{|s| s.to_i}
3    break if pN == 0
4    pd = [] # <p,d>をしまう配列
5    (1..pN).each { # pN 回何かする
6      d, p = gets.split("␣").map{|s| s.to_i}
7      pd << [p, d]
8    }
9    # pd を大きい順に並べてみよう
10   # pd を出力してみよう
11   # 並べ替えがうまく行っていたら答えを計算してみよう
12 end
```

回答例 (答えの計算):

C++

```
1     int S = 0;
2     for (int i=0; i<N; ++i)
3       S += 道[i]の危険度 * 道[i]の長さ;
4     // 予算 0 の時の答えが，現在の S の値
5     for (int i=0; i<N; ++i) {
6       if (M <= 0) break;
7       int guarded = Mと道[i]の長さの小さい方; // 雇う区間
8       S -= 道[i]の危険度 * guarded;
9       M -= guarded;
10    }
11    S が答え
```

- 予算が 0 の場合は，答えとして必要な “刺客に襲われる回数の期待値” である S は，各道 i について $S = \sum_i P_i \cdot D_i$ となる．
- 予算が M の場合，S から護衛を雇えた区間だけ期待値を減らす．例えば道 j の区間全部で護衛を雇うなら $P_j \cdot D_j$ だけ S を減らすことができる．もし残り予算 m が道の長さ D_j より小さく道の一部だけしか護衛を雇えないなら S から減らせるのは $P_j \cdot m$ だけである．

問題	**Radar Installation**	(Beijing 2002)

全ての島が見えるようにレーダーを配置する際に、レーダーの数を最小化したい
http://poj.org/problem?id=1328
注意: y=0 の島が存在する模様
考察課題 なぜ正しい回答を得られるかの説明を述べよ

POJ のアカウントは、AOJ とは別に作る必要がある。
ヒント:

- 島が観測できるレーダーの区間を列挙したとする．まだレーダにカバーされていない島を一番沢山カバーできる場所にレーダを配置し，それを繰り返すという戦略を考える．この戦略が最適な配置より多くのレーダを必要とするような島の並びの例を示せ
- 左から順にレーダを配置するとする．(まだレーダにカバーされていないなかで) 見えはじめるのが最も左にある島を選び，その島が見える区間の左端にレーダを配置するとする．この戦略が最適な配置より多くのレーダを必要とするような島の並びの例を示せ
- 左から順にレーダを配置するとする．(まだレーダにカバーされていないなかで) 見えはじめるのが最も左にある島を選び，その島が見える区間の右端にレーダを配置するとする．この戦略が最適な配置より多くのレーダを必要とするような島の並びの例を示せ

厳密に説明を行う場合は、レーダを配置する正しい戦略 A を示し，他のどのような配置も戦略 A による配置よりレーダ数が小さくならないことを背理法で証明する。

よくあるバグ

遠すぎる島など，例外を考慮する必要がある．複数のテストケースを扱うので，遠すぎる島があっても入力は全ての島を読み込むこと (さもないと次のテストケースで先頭がずれる).

問題	**Dinner⋆**	(夏合宿 2004)

自炊か外食かを考えながら、N 日間で得られる幸福度の合計を最大化する。
http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2642
考察課題 なぜ正しい回答を得られるかの説明をまとめよ

3.4 今週の課題

問題を解いて，AOJ で Accepted を得たことを確認して，各問題指定の考察課題もしくは指定がない問題については感想 (学んだことや課題にかけた時間など) とともに，ITC-LMS に提出する．

コメントの記述は，`#if 0` と`#endif` で囲むことが簡便である．

C++
```cpp
// minmaxsum.cc
#if 0
  所要時間は..であった. 方針は....
  ...
#endif
int main() {
}
```

扱う問題は以下の箇条書きのどちらかで合格 (言語指定なし):

- 「カントリーロード」と (「Princess's Marriage」または「Rader Installation」) を解いて AC を得て考察とともに提出. (Rader Installation の方を勧める)
- 「Dinner」で AC を得たうえで考え方の説明とともに提出する (どの問題も解いたことがある場合は,「似た」方針と難易度の問題を一つ探して, 回答・解説し, 提出する)

提出先: ITC-LMS
提出の注意:

- 問題毎にテキストファイルを一つ作り, 提出する (zip は避ける; PDF, Microsoft Word, rtf などは避けること). グラフを作成した場合は, グラフのみ別ファイルで提出する.
- 提出物は履修者全体に公開される. この授業では, 他者のソースコードを読んで勉強することを推奨するが, 課題を解く上で参考にした場合は, 何をどのように参加したかを明記すること.
- 締切: 授業終了時, 次回授業の前日, 学期終了時のそれぞれで最善のものを提出する

3.A 列に対する操作

3.A.1 反転 reverse

C++
```cpp
#include <algorithm>
using namespace std;
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5);
   reverse(A,A+5); // 与えられた範囲を逆順に並び替え
   ... // cout に A を出力してみよう
}
```

Ruby
```ruby
a = [3,5,1,2,4]
a.sort!
a.reverse! # a を逆順に並び替え
p a
```

3.A.2 回転 rotate

C++ で rotate(a,b,c) は範囲 [a,b) と範囲 [b,c) を入れ替える．

C++

```cpp
#include <algorithm>
int A[7] = {0,1,2,3,4,5,6};
int main() {
  rotate(A,A+3,A+7);
  // vector の場合は rotate(A.begin(),A.begin()+3,A.begin()+7);
  // A を出力してみよう　　➔　　3 4 5 6 0 1 2
}
```

Ruby は，a[p,q] で配列 a の位置 p から q 文字を参照したり，置き換えるという強力な機能を持つ．これを用いると回転も簡単に実現可能である

Ruby

```ruby
a = [0,1,2,3,4,5,6]
a[0,7] = a[3,4]+a[0,3]
p a #　　➔　　[3, 4, 5, 6, 0, 1, 2]
```

3.A.3 大きい順に整列

標準の sort 関数は小さい順に並べ替える．大きい順に並べ替えるにはどのようにすれば良いだろうか?

1. 小さい順に並べ替えた後に，並び順を逆転させる (当面これで十分)

C++

```cpp
int A[5] = {3,5,1,2,4};
int main() {
   sort(A,A+5);
   reverse(A,A+5); // 与えられた範囲を逆順に並び替え
   ...  // cout に A を出力してみよう
}
```

Ruby

```ruby
a = [3,5,1,2,4]
a.sort!
a.reverse! # a を逆順に並び替え
p a
```

2. 各要素を負にした配列を整列する

C++

```cpp
int A[5] = {3,5,1,2,4}, B[5];
int main() {
   for (int i=0; i<5; ++i) B[i] = -A[i];
   sort(B,B+5);
   ...  // cout に-B[i] を出力してみよう
}
```

3. 比較関数を渡す

C++11

```
1 int A[5] = {3,5,1,2,4};
2 int main() {
3     sort(A,A+5,[](int p, int q){ return p > q; });
4     ... // cout に A[i] を出力してみよう
5 }
```

文法の説明は省略するが `[](int p, int q){ return p > q; }` の部分が 2 つの整数の並び順を判定する匿名関数である.