データ構造とアルゴリズム: ソート

2022/05/11 糸川倫太朗

バブルソートとクイックソート

- バブルソート
  - バブルソートとは、与えられたデータ列を大小などの順序どおりになるように並べ変えるソート手法です。アルゴリズムの最も基本的な手法で、端から隣接する要素を比較・交換していくことでソートします。
- クイックソート
  - クイックソートとは、大きい値のグループと小さい値のグループに分けていき、それぞれのグループで同じようにグループを分けていくことを繰り返してソートする手法です。

> 出典: 各ソートアルゴリズムのスピード検証

計算量

バブルソートでは、必ずn(n-1)/2回の比較が行われます。したがって、計算回数のオーダーはO(n^2)です。

クイックソートの計算回数は、平均でO(n log n)ですが、最悪のケースではO(n^2)です。

演算の結果

- Bubble Sort

データの個数	比較回数	交換回数
10	45	27
100	4950	2571
1000	499500	247964
10000	49995000	24883769

- Quick Sort

データの個数	比較回数	交換回数
10	42	21
100	735	199
1000	12589	1999
10000	173324	19999

演算の考察

データが増えるに連れて比較交換ともにクイックソートのほうが効率が良かった。計算量に従っていた

自作したバブルソート、クイックソートのコード

cpp : bubble-sort.cpp

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <stdexcept>
#include <algorithm>

using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::vector;

template<typename T>
void printVector(const vector<T> &vec) {
  for ([[maybe_unused]] auto &i : vec) {
    cout << i << "; ";
  }
  cout << endl;
}

template<typename T>
void bubbleSort(vector<T> &vec) {
  for (size_t i = 0; i < vec.size() - 1; ++i) {
    for (size_t j = 0; j < vec.size() - i - 1; ++j) {
      if (vec.at(j) > vec.at(j + 1)) {
        std::swap(vec.at(j), vec.at(j + 1));
      }
    }
  }
}

int main(int argc, char* argv[]) {
  try {
    if (argc != 2) {
      throw std::out_of_range("out of range");
    }
    std::ifstream in(argv[1]);
    vector<int> vec;
    string value;
    if (in.fail()) {
      throw std::logic_error("file couldn't open");
    }
    while (!in.eof()) {
      in >> value;
      vec.push_back(std::stoi(value));
    }
    printVector(vec);
    bubbleSort(vec);
    printVector(vec);
```

```
    return 0;
  }
  catch (std::exception& e) {
    cout << e.what() <<endl;
    return -1;
  }
}
```

cpp : quick-sort.cpp

```cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <stdexcept>

using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::vector;

template<typename T>
void printVector(const vector<T> &vec) {
  for ([[maybe_unused]] auto &i: vec) {
    cout << i << "; ";
  }
  cout << endl;
}

template<typename T>
T partitionVec(vector<T> &vec, size_t start, size_t end) {
  T pivot = vec.at(start);
  auto lh = start + 1;
  auto rh = end;
  while (true) {
    while (lh < rh && vec.at(rh) >= pivot) {
      rh--;
    }
    while (lh < rh && vec.at(lh) < pivot) {
      lh++;
    }
    if (lh == rh) {
      break;
    }
    T tmp = vec.at(lh);
    vec.at(lh) = vec.at(rh);
    vec.at(rh) = tmp;
  }
  if (vec.at(lh) >= pivot) {
    return start;
  }
  vec.at(start) = vec.at(lh);
  vec.at(lh) = pivot;
  return lh;
}
```

```
template<typename T>
void sort(vector<T> &vec, size_t start, size_t end) {
  if (start >= end) {
    return;
  }
  auto boundary = partitionVec(vec, start, end);
  sort(vec, start, boundary);
  sort(vec, boundary + 1, end);
}

template<typename T>
void quickSort(vector<T> &vec) {
  sort(vec, 0, vec.size() - 1);
}

int main(int argc, char *argv[]) {
  try {
    if (argc != 2) {
      throw std::out_of_range("out of range");
    }
    std::ifstream in(argv[1]);
    vector<int> vec;
    string value;
    if (in.fail()) {
      throw std::logic_error("file couldn't open");
    }
    while (!in.eof()) {
      in >> value;
      vec.push_back(std::stoi(value));
    }
    printVector(vec);
    quickSort(vec);
    printVector(vec);
  }
  catch (std::exception &e) {
    cout << e.what() << endl;
    return -1;
  }
}
```