Báo cáo thực nghiệm các thuật toán sắp xếp

Người thực hiện: Vũ Ngọc Quốc Khánh

Giới thiệu

Báo cáo về thời gian thực thi của các thuật toán sắp xếp như Quicksort, Heapsort, Mergesort và hàm `std::sort()` trong thư viện chuẩn của C++.

Chuẩn bị

- Bộ dữ liệu kiểm thử gồm 10 dãy số thực, trong đó có 1 dãy số tăng dần, 1 dãy số giảm dần và 8 dãy số ngẫu nhiên
- Code cài đặt Quicksort
- Code cài đặt Heapsort
- Code cài đặt Mergesort
- Code sử dụng `std::sort` trong thư viện chuẩn của C++.
- Code thực hiện thuật toán và đo thời gian.

Tất cả các code và tệp tin liên quan đều được để trong **Github này**

Bộ dữ liệu kiểm tra

Được sinh ngẫu nhiên bằng code sử dụng thư viện Testlib của Mike Mirzayanov - Người sáng lập Codeforces

Code sinh:

```
#include "testlib.h"
#include <bits/stdc++.h>
int main(int argc, char *argv[]){
    registerGen(argc, argv, 1);

    //Increasing number
    std::ofstream out1("../tests\\1-increasing.txt");
    for(int i = 0; i < 1000000; ++i){
```

```
        double value = i + rnd.next(0.0000, 0.9999);
        out1 << std::fixed << std::setprecision(4) << value << ' ';
    }
    out1.close();

    //Decreasing number
    std::ofstream out2("../tests\\2-decreasing.txt");
    for(int i = 1000000; i > 0; --i){
        double value = i + rnd.next(0.0000, 0.9999);
        out2 << std::fixed << std::setprecision(4) << value << ' ';
    }
    out2.close();

    //Random number
    for(int tc = 3; tc <= 10; ++tc){
        std::string file_name = "../tests\\" + std::to_string(tc) + "-random.txt";
        std::ofstream out(file_name);
        for(int i = 0; i < 1000000; ++i){
            double value = 1.0 * rnd.next(0.0000, 999999.0000)
                 + (1.0 * rnd.next(0.0000, 0.9999) * rnd.next(0.0000, 0.9999));
            out << std::fixed << std::setprecision(4) << value << ' ';
        }
        out.close();
    }
}
```

Phân tích các thuật toán sắp xếp

Quicksort

Mô tả

Đây là thuật toán sắp xếp theo hướng tiếp cận Chia để trị với ý tưởng như sau:

1. Chọn 1 phần tử trong mảng cần sắp xếp làm chốt `pivot`.
2. Chia mảng thành 2 phần với điểm chính giữa là `pivot`, các phần tử nhỏ hơn `pivot` sẽ nằm ở một phần và ngược lại, các phần tử lớn hơn `pivot` sẽ nằm ở phần còn lại.
3. Với mỗi phần đã chia, ta thực hiện lại bước 1.

Code cài đặt

```
std::mt19937 rd(std::chrono::steady_clock::now().time_since_epoch().count());

//hàm sinh số ngẫu nhiên
int Rand(int l, int h) {
    return l + (rd() >> 1) * 1LL * (rd() >> 1) % (h - l + 1);
}

template <typename T>
void sort(T arr[] , int left, int right){
    int pivot = arr[Rand(left, right)]; //ngẫu nhiên chọn pivot
    int l = left;
    int r = right;
    do{
        while((l <= right) && (arr[l] < pivot)) //bỏ qua các phần tử nhỏ hơn pivot và nằm bên trái
            ++l;
        while((r >= left) && (arr[r] > pivot)) //bỏ qua các phần tử lớn hơn pivot và nằm bên phải
            --r;
        if(l > r) //nếu như chạy quá pivot, thoát
            break;
        std::swap(arr[l], arr[r]); //thay đổi vị trí 2 phần tử nằm khác bên của pivot
        ++l, --r; //tiếp tục
    } while(l <= r);
    //chia thành các nửa để sắp xếp
    if(left < r)
```

```
        sort(arr, left, r);
    if(l < right)
        sort(arr, l, right);
}
```

Đánh giá

- **Ưu điểm**
  - Trong đa số trường hợp, thuật toán quicksort có độ phức tạp là $\mathcal{O}(NlogN)$, khá nhanh và cài đặt đơn giản.
- **Nhược điểm**
  - Trong trường hợp tệ nhất, thuật toán quicksort có độ phức tạp là $\mathcal{O}(N^2)$

Mergesort

Mô tả

Đây là thuật toán sắp xếp theo hướng tiếp cận Chia để trị với ý tưởng như sau:

1. Giả sử ta có 2 mảng con A và B đã được sắp xếp, ta tạo thêm một mảng C để "trộn" 2 mảng A và B sao cho sau khi "trộn" thì mảng C cũng được sắp xếp.
2. Nếu mảng A hoặc B chưa được sắp xếp thì ta chia mảng chưa sắp xếp thành 2 phần và quay lại bước 1.

Code cài đặt

```
template <typename T>
void sort(T arr[], int left, int right){
    std::vector<T> MidArr(right - left + 1); //khởi tạo mảng trung gian để "trộn"
    if(right - left + 1 == 1) //mảng chỉ có 1 phần tử, không cần phải sắp xếp
        return;
    int mid = (left + right) / 2; //chọn phần giữa để chia mảng thành 2 phần

    //gọi hàm sắp xếp 2 nửa
    sort(arr, left, mid);
    sort(arr, mid + 1, right);
```

```
    //"trộn" 2 nửa đã sắp xếp
    int i = left, j = mid + 1;
    int cur = 0;// chỉ số của mảng trung gian
    while(i <= mid || j <= right){
        if(i > mid) //bên trái không còn phần tử
            MidArr[cur++] = arr[j++];
        else if(j > right) //bên phải không còn phần tử
            MidArr[cur++] = arr[i++];
        else if(arr[i] < arr[j]) //phần tử bên trái nhỏ hơn, ta cho vào mảng trước
            MidArr[cur++] = arr[i++];
        else //phần tử bên trái nhỏ hơn, ta cho vào mảng trước
            MidArr[cur++] = arr[j++];
    }

    //đưa các giá trị ở mảng trung gian về mảng chính
    for(int i = 0; i < cur; ++i)
        arr[left + i] = MidArr[i];
}
```

Đánh giá

- **Ưu điểm**
  - Thuật toán chạy ổn định với độ phức tạp là $\mathcal{O}(N \log N)$
- **Nhược điểm**
  - Ta phải sử dụng thêm một mảng trung gian để trộn nên tốn nhiều bộ nhớ nếu dữ liệu lớn

Heapsort

Mô tả

Là một thuật toán sắp xếp được thiết kế trên cấu trúc dữ liệu Heap

Heapsort sẽ thực hiện 2 công việc khác nhau:

- **Tạo heap**
    - Thực hiện việc tạo ra cấu trúc max-heap từ một mảng đã có sẵn
    - Công việc tạo heap sẽ được thực hiện như sau:
    - 1. Ta có đỉnh cha i và 2 đỉnh con $l = i \times 2 + 1, r = i \times 2 + 2$.
    - 2. Kiểm tra trong 3 đỉnh có giá trị lớn nhất.
    - 3. Nếu đỉnh có giá trị lớn nhất không phải đỉnh cha, ta thực hiện đổi chỗ 2 đỉnh với nhau và thực hiện việc tạo heap với cây con tại vị trí đỉnh vừa đổi chỗ
- **Sắp xếp**
    - Sau khi đã có được một cấu trúc max-heap như mong muốn, ta thực hiện việc đổi thứ tự các đỉnh nhỏ nhất và tạo heap lại với độ lớn nhỏ hơn ở các đỉnh chưa được lấy ra.

Code cài đặt

Trong C++, đã có sẵn cấu trúc dữ liệu cho Heapsort trong thư viện chuẩn như `priority_queue`, ta chỉ cần sử dụng.

```
template <typename T>
void sort(T arr[], int N){ //ta cần độ dài của mảng
    std::priority_queue< T, std::vector<T>, std::greater<T> > pq;
    //khai báo cấu trúc là giá trị nhỏ hơn sẽ nằm ở phía bên trái
    for(int i = 0; i < N; ++i)
        pq.push(arr[i]); //cho các phần tử của mảng vào priority queue
    for(int i = 0; i < N; ++i){
        arr[i] = pq.top(); //trả các phần tử đã sắp xếp trong priority queue ngược lại mảng
        pq.pop(); //xoá phần tử đã trả khỏi priority queue
    }
}
```

Đánh giá

- **Ưu điểm**
    - Không tốn bộ nhớ nhiều như Mergesort

    - Độ phức tạp là $\mathcal{O}(N \log N)$
- **Nhược điểm**
    - Không được ổn định

`std::sort` của thư viện chuẩn C++

Mô tả

Đây là một thuật toán Introsort sắp xếp được kết hợp bởi 3 thuật toán sắp xếp khác nhau là Quicksort, Heapsort và Selectionsort. Các thuật toán như Heapsort và Selectionsort được sử dụng để loại bỏ các trường hợp tệ nhất của quicksort.

Cài đặt

Đã có sẵn hàm `std::sort` trong thư viện `algorithm` của C++:

```
std::sort(a, a + n);
```

Thực nghiệm

Code thực hiện thuật toán sắp xếp và đo thời gian:

```
#include <bits/stdc++.h>
#include <chrono>
// using namespace std;
using std::chrono::high_resolution_clock;
using std::chrono::duration_cast;
using std::chrono::duration;

namespace quicksort{
    #include "QuickSort.h"
}

namespace mergesort{
    #include "MergeSort.h"
```

```cpp
}

namespace heapsort{
    #include "HeapSort.h"
}


template <typename T>
void benchmark(T arr[], int N, const std::string& name){
    auto t1 = high_resolution_clock::now(); //thời gian trước khi thực hiện thuật toán
    if(name == "Quicksort")
        quicksort::sort(arr, 0, N - 1);
    else if(name == "Mergesort")
        mergesort::sort(arr, 0, N - 1);
    else if(name == "Heapsort")
        heapsort::sort(arr, N);
    else
        std::sort(arr, arr + N);
    auto t2 = high_resolution_clock::now(); //thời gian sau khi thực hiện thuật toán

    duration<double, std::milli> ms_double = t2 - t1;
    std::cout << name << "\t\t: " << ms_double.count() << "ms\n";
}

template <typename T>
void init(T arr[], T ans[],int N){
    for(int i = 0; i < N; ++i) {
        ans[i] = arr[i];
    }
}
```

```cpp
template <typename T>
void startbench(T arr[], int N){
    T *array = new T[N + 7];
    init<T>(arr, array, N);
    std::cout << "quicsort\n";
    benchmark<T>(array, N, "Quicksort");
    init<T>(arr, array, N);
    benchmark<T>(array, N, "Mergesort");
    init<T>(arr, array, N);
    benchmark<T>(array, N, "Heapsort");
    init<T>(arr, array, N);
    benchmark<T>(array, N, "std::sort()");
    delete(array);
}

double arr[1'000'000];

void input(const std::string& file){
    std::ifstream in("../tests\\" + file);
    for(int i = 0; i < 1000000; ++i){
        in >> arr[i];
    }
}

signed main(){

    //test 1: increasing
    input("1-increasing.txt");
    std::cout << "test 1:\n";
    startbench<double>(arr, 1'000'000);
```

```
    //test 2: decreasing
    input("2-decreasing.txt");
    std::cout << "test 2:\n";
    startbench<double>(arr, 1'000'000);

    //test 3 - 10: random
    for(int i = 3; i <= 10; ++i){
        std::string suffix = "-random.txt";
        std::string file = std::to_string(i) + suffix;
        input(file);
        std::cout << "test " << i << ":\n";
        startbench<double>(arr, 1'000'000);
    }
    return 0;
}
```

Kết quả thực nghiệm

Bảng dữ liệu (Đơn vị ms)

Bộ test	Quicksort	Mergesort	Heapsort	std::sort()
1	68.2218	170.504	101.369	21.4166
2	99.0501	181.226	77.8732	24.3394
3	78.4028	197.191	82.5971	23.7718
4	87.6631	221.236	83.8544	23.7792
5	74.9384	247.369	75.5021	20.3559
6	70.8858	208.926	84.3462	23.4191
7	71.6707	203.803	78.5044	25.3392
8	77.259	197.443	73.8054	24.593
9	89.3164	249.554	89.2753	15.5982
10	66.18	147.516	64.5286	15.0978

Biểu đồ dữ liệu