Московский Авиационный институт
(Национальный исследовательский университет)

Факультет №8
«Компьютерные науки и прикладная математика»

Кафедра 806
«Вычислительная математика и программирование»

Курсовая работа

по теме

«Сортировка и поиск»

Студент:	Концебалов О.С.
Группа:	М8О-109Б-22
Преподаватель:	Сысоев М. А.
Подпись:	
Оценка:	

Москва, 2023

Постановка задачи

Составить и отладить программу на языке C++ с использованием функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Метод сортировки

Турнирная сортировка

Структура таблицы

Тип ключа: комбинированный (строка + целое)
Длина ключа в байтах: 8
Хранение данных и ключей: вместе
Число элементов таблицы: 8-12

Основная часть

Описание структуры

Структура содержит произвольный набор символов - запись. Доступ осуществляется по индексу. Есть массив ключей вещественного типа, индексы ключа в массиве и строки, ему соответствующей, совпадают.

Метод решения

После запуска программы открывается указанный файл на чтение. В таблицу вводятся строки и ключи. Осуществляется вывод несортированной таблицы. После запускается функция сортировки. Если таблица уже отсортирована, то ничего не меняется. Если таблица указана наоборот, осуществляется реверс строк и ключей. Если таблица никак не отсортирована, работает алгоритм сортировки. Дальше отсортированная таблица выводится, после чего можно вывести отдельные строки по ключу. Здесь используется бинарный поиск. Если указан несуществующий ключ, работа программы прекращается.

Функциональное назначение

Программа предназначена для демонстрации использования метода сортировки, бинарного поиска, структуры таблицы.

Турнирная сортировка

Основная идея Tournament sort заключается в использовании относительно небольшой (по сравнению с основным массивом) вспомогательной кучи, которая выполняет роль приоритетной очереди. В этой куче сравниваются элементы на нижних уровнях, в результате чего меньшие элементы (в данном случае дерево у нас MIN-HEAP) поднимаются наверх, а в корень всплывает текущий минимум из той порции элементов массива, которые попали в эту кучу. Минимум переносится в дополнительный массив «победителей», в результате чего в куче происходит сравнение/перемещение оставшихся элементов — и вот уже в корне дерева новый минимум. Отметим, что при такой системе очередной минимум по значению больше чем предыдущий — тогда легко собирается новый упорядоченный массив «победителей», где новые минимумы просто добавляются в конец дополнительного массива.

Перенос минимумов в дополнительный массив приводит к тому, что в куче появляются вакантные места для следующих элементов основного массива — в результате чего в порядке очереди обрабатываются все элементы.

Главной тонкостью является то, что кроме «победителей» есть ещё и «проигравшие». Если в массив «победителей» уже перенесены какие-то элементы, то если необработанные элементы меньше, чем эти «победители» то просеивать их через турнирное дерево на этом этапе уже нет смысла — вставка этих элементов в массив «победителей» будет слишком дорогой (в конец их уже не поставишь, а чтобы поставить в начало — придётся сдвинуть уже вставленные раннее минимумы). Такие элементы, не успевшие вписаться в массив «победителей», отправляются в массив «проигравших» — они пройдут через турнирное дерево в одной из следующих итерации, когда все действия повторяются для оставшихся неудачников.

Сложность O(**n** log / **n**)

Текст программы

Line.hpp

```
#ifndef LINE_HPP
#define LINE_HPP

#include <string>

template <typename T>
```

```
class Table;

template <typename T>
class Line{
    friend class Table<T>;
private:
    T key;
    std::string data;

public:
    Line() = default;
    Line(const T& key, const std::string& data);
    ~Line() = default;

    T get_key();
    std::string get_data() const;

    bool operator<(const Line<T>& other) const;
};

#include "Line.cpp"

#endif
```

Line.cpp

```
#include "Line.hpp"
#include <string>

template <typename T>
Line<T>::Line(const T& key, const std::string& data): key(key),
data(data){}

template <typename T>
T Line<T>::get_key(){
    return key;
}

template <typename T>
std::string Line<T>::get_data() const{
    return data;
```

```
}

template <typename T>
bool Line<T>::operator<(const Line<T>& other) const{
    return key < other.key;
}
```

Table.hpp

```
#ifndef TABLE_HPP
#define TABLE_HPP

#include "MyVector.hpp"
#include "MyVector.cpp"
#include "Line.hpp"
#include <iostream>

template <typename T>
class Table{
private:
    myVector<Line<T>> lines;
    size_t winner(const size_t pos_a, const size_t pos_b, const
size_t n, myVector<size_t>& win_match, myVector<T>& lines);

public:
    Table() = default;
    ~Table() = default;
    myVector<Line<T>>& get_lines() const;

    void push_back(const T& key, const std::string& data);
    Line<T> search(const T& key) const;
    void sort(const T& inf);
    void print() const;

};

#include "Table.cpp"

#endif
```

Table.cpp

```cpp
#include "Table.hpp"
#include <iomanip>

template <typename T>
myVector<Line<T>>& Table<T>::get_lines() const{
    return lines;
}

template <typename T>
void Table<T>::push_back(const T& key, const std::string&
data){
    lines.push_back(Line<T>(key, data));
}

template <typename T>
Line<T> Table<T>::search(const T& key) const{
    size_t left = 0;
    size_t right = lines.size();

    while (left <= right){
        size_t mid = left + (right - left) / mid;
        if (lines[mid].get_key() == key){
            return lines[mid];
        } else if (lines[mid].get_key() < key){
            left = mid + 1;
        } else{
            right = mid - 1;
        }
    }

    return Line<T>(key, "\0");
}

template <typename T>
void Table<T>::sort(const T& inf){
    size_t n;
    myVector<T> t_keys;
    myVector<std::string> t_lines;
    myVector<size_t> win_match;
```

```
    n = lines.size();
    if (n <= 1) {
        return;
    }
    t_keys.resize(n + 1);
    t_lines.resize(n + 1);
    win_match.resize(2 * n + 2);

    for (size_t i = 0; i < n; ++i) {
        t_keys[i] = lines[i].key;
        t_lines[i] = lines[i].data;
        win_match[n + i] = i;
    }
    for (int64_t i = 2 * n - 1; i > 1; i -= 2) {
        int64_t winner_place = i / 2;
        win_match[winner_place] = winner(i, i - 1, n,
win_match, t_keys);
    }


    lines[0].key = t_keys[win_match[win_match[1]]];
    lines[0].data = t_lines[win_match[win_match[1]]];
    t_keys[win_match[win_match[1]]] = inf;

    for (size_t iteration = 1; iteration < n; ++iteration) {
        int64_t i = win_match[1];
        while (i > 1) {
            int64_t winner_place = i / 2;
            int64_t challenger;
            if (i % 2 == 0 && i < 2 * n - 1) {
                challenger = i + 1;
            }
            else {
                challenger = i - 1;
            }
            win_match[winner_place] = winner(i, challenger, n,
win_match, t_keys);
            i = winner_place;
        }
        lines[iteration].key = t_keys[win_match[win_match[1]]];
```

```
        lines[iteration].data =
t_lines[win_match[win_match[1]]];
        t_keys[win_match[win_match[1]]] = inf;
    }
}

template <typename T>
size_t Table<T>::winner(const size_t pos_a, const size_t pos_b,
const size_t n, myVector<size_t>& win_match, myVector<T>&
lines){
    size_t u = pos_a >= n ? pos_a : win_match[pos_a];
    size_t v = pos_b >= n ? pos_b : win_match[pos_b];

    if (lines[win_match[u]] <= lines[win_match[v]]) return u;

    return v;
}

template <typename T>
void Table<T>::print() const{
    std::cout << "  | " << "  key   " << " | " <<
"      data      |\n";
    std::cout << "--+"  << "---------" << "-+-" << "----------
-----+\n";
    for (size_t i = 0; i != lines.size(); ++i){
        std::cout << i + 1 << " | " << std::setw(8) <<
std::left << lines[i].get_key() << " | " <<
                    std::setw(15) << std::left <<
lines[i].get_data() << " |\n";
        std::cout << "--+"  << "---------" << "-+-" << "-------
---------+\n";
    }
}
```

Benchmark.cpp

```
#include "Table.hpp"
#include <algorithm>
#include <chrono>
#include <iostream>
#include <string>
```

```cpp
#include <vector>

void benchmark(){
    std::cout << "Comparing my \"Sort\" and \"sort\" from
STL\n\n";

    Table<std::string> test1;
    std::vector<std::string> test2;

    for (size_t i = 0; i != 1000000; ++i){
        test1.push_back("fka1fn" + std::to_string(i),
"fmkdgsnrnr4312");
        test2.push_back("ekfaff" + std::to_string(i));
    }

    std::chrono::steady_clock::time_point start_time =
std::chrono::steady_clock::now();
    test1.sort("zzzzzzzzzzzzzzzzz");
    std::chrono::steady_clock::time_point end_time =
std::chrono::steady_clock::now();
    std::cout << "Time of work my Sort: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";

    start_time = std::chrono::steady_clock::now();
    std::sort(test2.begin(), test2.end());
    end_time = std::chrono::steady_clock::now();
    std::cout << "Time of work original sort: " <<
        std::chrono::duration_cast<std::chrono::milliseconds>(e
nd_time - start_time).count() << " milliseconds\n";
}
```

Run.cpp

```cpp
#include "Table.hpp"
#include "benchmark.cpp"
#include <iostream>
#include <string>

int main(){
    Table<std::string> t;
```

```
    t.push_back("zzzz0012", "fdrkr");
    t.push_back("ttttt45g", "glpdsg");
    t.push_back("qqqqrer4", "gldg");
    t.push_back("bgag3431", "gfggf");
    t.push_back("aaaare45", "fdlafdl");

    t.sort("zzzzzzzzzzzzzzzzzzzzz");
    t.print();

    benchmark();
}
```

Тесты производительности

Количество записей: 1000000

Comparing my "Sort" and "sort" from STL

Time of work my Sort: 1349 milliseconds
Time of work original sort: 2507 milliseconds

Как мы видим из результата сравнения, турнирная сортировка отработала в 2 раза !!! быстрее чем std::sort. Я предполагаю, что это связано с тем, что турнирная сортировка лучше справляется с большими объемами данных, чем std::sort. Также std::sort основан на нескольких способах сортировки и в худшем случае его сложность может быть O(n^2) или O(n log n), а моя турнирная сортировка затачивалась именно под один конкретный тип данных – таблицу с определенными ключами. Возможно, повлияло это.

Заключение

В задании номер 9 КП я познакомился с различными методами сортировок, а именно с Турнирной сортировкой. Реализовать ее было непросто, но довольно интересно и полезно. Так же еще я создал свою таблицу с данными, которые хранятся там по ключу и отсортировал ее по ключу. Довольно неплохо задание с прикладным смыслом.

Список литературы

1. https://habr.com/ru/companies/edison/articles/508646/
2. Методические указания к выполнению курсовых работ. Зайцев В. Е.
3. http://all-ht.ru/inf/prog/c/func/fgets.html
4. https://www.cyberforum.ru/cpp-beginners/thread401868.html