АКМИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»
Тема: Алгоритмы сортировки

Студент гр. 9303 _____ Халилов Ш.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Реализовать алгоритм пасьянсная сортировка

Задание.

Пасьянсная сортировка.

Основные теоретические положения.

Терпеливая сортировка (Пасьянсная сортировка) (англ. patience sorting) — алгоритм сортировки с худшей сложностью O(n log n). Позволяет также вычислить длину наибольшей возрастающей подпоследовательности данного массива. Алгоритм назван по одному из названий карточной игры "Солитёр" — "Patience".

Имеем массив source[0..n−1], элементы которого нужно отсортировать по возрастанию. Разложим элементы массива по стопкам: для того чтобы положить элемент в стопку, требуется выполнение условия — новый элемент меньше элемента, лежащего на вершине стопки; либо создадим новую стопку справа и сделаем наш элемент её вершиной. Используем жадную стратегию: каждый элемент кладётся в самую левую стопку из возможных, если же таковой нет, справа от существующих стопок создаётся новая. Для получения отсортированного массива сначала построим массив стопок, затем выполним n шагов (здесь и далее нумерация шагов начинается с единицы): на i-м шаге выберем из всех вершин стопок наименьшую, извлечём её и запишем в массив ans[0..n−1] на i-ю позицию.

Мы формируем новую стопку, когда встречаем элемент больший, чем вершины всех стопок, расположенных слева. В то же время стопки слева были созданы ранее, то есть элементы в них идут в исходной последовательности раньше текущего. Каждая стопка представляет собой убывающую последовательность, то есть длина НВП в пределах стопки равна единице, поэтому появление новой стопки можно понимать как увеличение длины наибольшей возрастающей подпоследовательности на единицу (изначально

длина НВП равна единице). Поэтому длина наибольшей возрастающей подпоследовательности равна количеству стопок.

Для получения наибольшей возрастающей подпоследовательности при формировании стопок проведём следующие операции: каждый раз, положив элемент на вершину стопки, будем создавать указатель на возможный предыдущий элемент (вершину ближайшей слева стопки). В конце для получения наибольшей возрастающей подпоследовательности нужно выполнить p шагов, начав с вершины самой правой стопки: на i-м шаге записать в lis[0..p−1] на (p−i)-ю позицию текущий элемент, перейти к предыдущему элементу по указателю, p — количество стопок.

Сложность - Создадим список стеков для хранения стопок. При раскладывании элементов по стопкам для поиска самой левой подходящей стопки используем бинарный поиск. Соответственно, поиск самой левой стопки занимает O(log p), где p — количество стопок (стеков). Таким образом, временная сложность раскладывания по стопкам не превышает O(nlogn).

Для получения отстортированного массива используем бинарную кучу. На каждом шаге алгоритма необходимо извлечь из кучи стек с минимальной вершиной за O(log p), где p — количество стеков в куче. Снять вершину выбранного стека и вернуть его в кучу за O(log p). Получение отсортированного массива займёт O(nlog n) времени. Получение наибольшей возрастающей подпоследовательности выполняется за O(n) по описанному выше алгоритму. Таким образом, алгоритм сортировки требует O(nlogn) времени в худшем случае и O(n) дополнительной памяти при любом раскладе.Рекурсивный - это объект, который содержит сам себя или определяется с помощью самого себя. Сила рекурсии заключается в том, что она позволяет вам определять бесконечное количество объектов с помощью конечного оператора. Точно так же бесконечные вычисления можно описать с помощью конечной рекурсивной программы. Рекурсивные алгоритмы лучше всего использовать, когда решаемая проблема, вычисляемая функция или обрабатываемая структура данных определяются с помощью рекурсии. Если процедура (функция) P содержит

явный вызов самой себя, она называется непосредственно рекурсивной. Если P содержит вызов процедуры (функции) Q, которая содержит (прямо или косвенно) вызов P, то P называется косвенно рекурсивным. Многие известные функции можно определить рекурсивно. Например, факториал, который присутствует почти во всех учебниках программирования, а также наибольший общий делитель, числа Фибоначчи, степенная функция и т. Д.

Шабло́ны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию). В C++ возможно создание шаблонов функций и классов. Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).

Выполнение работы.

Для реализации алгоритм Терпеливая сортировка (Пасьянсная сортировка) были реализованы следующие функции:

template <class T>

class stack – Класс для хранения значений времени

stack() - конструктор класса он создает буфер размером 50 для хранения значения

T* Buffer – указатель на память для буфера

int size – размер буфера

int counter – количество элементов в стеке

void addSize() - метод расширения буферов в случае пополнения стека

void pop() - метод удаления верхнего элемента

T top() - метод для получения верхнего элемента из стека

void push - метод для добавления элемента в стек

bool isEmpty() - метод проверки наличия элемента в стеке

void show(ofstream &fout) - это метод проверки состояния стека

template <class T>

class listStack

listStack() - конструктор класса он создает буфер размером 50 для хранения стека

stack <T> *st – указатель на память для буфера по типу стеков

int count – количество стеков в буфере

int size - размер буфера

void addSize() - метод расширения буфера, если количество стеков достигает размера буфера

void push(T x) - метод добавления элемента в стек, если верхние элементы в стеках меньше указанного элемента, то создается новый стек и добавляется элемент в новый стек

void show(ofstream &fout) - метод ввода состояния стеков

T get() - метод получения верхнего элемента из стеков, метод проверяет верхние элементы в стеках и возвращает наименьший из них

int * patienceSorting – функция получает массив и его размер. В нем listStak инициализируется и начинает класть в стек до тех пор, пока в массиве не останется ни одного элемента, а там класс сам определяет, где класть элемент. Затем он начнет получать элемент из стека и записывать его в массив, этот массив сортируется, потому что элементы из стейка будут отсортированы

Пример работи.

Пример работи находится в файле ResultPatienceSorting.txt

Выводы.

В процессе работы был изучен интересный алгоритм сортировки пасьянсов, который является самым быстрым способом сортировки вручную. но сам по себе алгоритм проигрывает в скорости работы алгоритму быстрой сортировки.

но некоторые учения говорят, что при правильной реализации алгоритм может дать очень хорошую производительность.

Приложения А

```
#include <iostream>
#include <fstream>
#include <unistd.h>
using namespace std;

template <class T>
class stack
{
private:
    T* Buffer;
    int size;
    int counter;
    void addSize(){
        size += 100;
        T *tmp = new int[size];
        for (int i = 0; i < counter; i++)
        {
            tmp[i] = Buffer[i];
        }
        delete Buffer;
        Buffer = tmp;
    }
public:
    stack(){
        size = 100;
        counter = 0;
        Buffer = new int[size];
```

```
    }
    void pop(){
        --counter;
        if(counter < 0 )
            counter = 0;

    }
    T top(){
        return Buffer[counter-1];
    }
    void push(int x){
        if( counter >= size-1 )
            addSize();

        Buffer[counter] = x;
        counter++;
    }
    bool isEmpty(){
        return counter <= 0;
    }
    void show(ofstream &fout){
        fout << " ";
        for (int i = 0; i < counter; i++)
        {
            fout << " [ "<< Buffer[i] << " ] ";
        }
        fout << "\n";
    }
};
```

```
template <class T>
class listStack
{
private:
    stack <T> *st;
    int count, size;
    void addSize(){
        size += 50;
        stack <T>*tmp = new stack <T>[size];
        for (int i = 0; i < count; i++)
        {
            tmp[i] = st[i];
        }
        delete st;
        st = tmp;
    }
public:
stack <T> *st;
    int count, size;
    void addSize()
    listStack(){
        size = 50;
        st = new stack<T>[size];
        count = 0;
    }
    void push(T x) {
        for (int i = 0; i < count; i++)
        {
            if( x < st[i].top() ){
                st[i].push(x);
                return;
```

```
        }
    }
    st[count].push(x);
    count++;
}
void show(ofstream &fout){
    fout << "\n";
    for (int i = 0; i < count; i++)
    {
        if(st[i].isEmpty())
            continue;
        fout << "stak[" << i << "] = ";
        st[i].show(fout);
    }
    fout << "\n";
}
T get(){
    int tmp_num;
    int tmp;
    for (int i = 0; i < count; i++)
    {
        if(st[i].isEmpty())
            continue;
        tmp = st[i].top();
        tmp_num = i;
        break;
    }

    for (int i = 0; i < count; i++)
    {
        if(st[i].isEmpty())
```

```
                continue;

            if( tmp > st[i].top() )
            {
                tmp = st[i].top();
                tmp_num = i;
            }
        }
        st[tmp_num].pop();
        return tmp;
    }
};

int * patienceSorting(int *mas, int size, ofstream &fout){

    listStack <int>ls;
    for (int i = 0; i < size; i++)
    {

        ls.push(mas[i]);
        fout << "[ ";
        for (int j = i; j < size-1; j++)
        {
            fout << mas[j] << ", ";
        }
        fout << mas[i] << " ]\n";

        ls.show(fout);
    }

    for (int i = 0; i < size; i++)
```

```
    {

        mas[i] = ls.get();
        fout << "[ ";
        for (int j = 0; j < i-1; j++){

            fout << mas[j] << ", ";
        }
        fout << mas[i-1] << " ]\n";
        ls.show(fout);

    }
    return mas;
}

int main(int argc, char const *argv[])
{

    ifstream fin;
    ofstream fout;
    if( argc > 1 ){

        fin.open(argv[1]);
        if( argc > 2 ) {
            fout.open(argv[2], ios_base::app);
        }
        else {
            fout.open("ResultPatienceSorting.txt");
        }
        int n, m;
        fin >> n;
```

```
        for (int i = 0; i < n; i++)
        {
            fin >> m;
            int *arr = new int[m+1];
            for (int j = 0; j < m; j++)
            {
                fin >> arr[j];
            }
            fout << "\n\n[   in   ]\n[  Array ]\n";
            int *res = patienceSorting(arr, m, fout);
            fout << "[   Result   ]\n[";
            for (int j = 0; j < m-1; j++)
            {
                fout << res[j] << ", ";
            }
            fout << res[m-1] << " ]";
            // delete res;
            delete arr;
        }
        fin.close();
        fout.close();
    }
    return 0;
}
```