МИНОБРНАУКИ 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: Сортировки

Студентка гр. 9303 _____ Зарезина Е.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомление с алгоритмом нитевидной сортировки, реализация нитевидной сортировки на языке C++.

Задание.

17. Нитевидная сортировка

Выполнение работы.

Функция enterList() считывает введённую пользователем полседовательность для сортировки. Далее введённый список копируется и копия изначального списка сортируется std::list::sort для итоговой проверки разработанного алгоритма нитевидной сортировки.

В функции strandSort() реализован алгоритм нитевидной сортировки. Реализация основана на работе с тремя списками. Сначала необходимо пройти по исходному списку и переместить элементы, образующие упорядоченный подсписок во второй временный список. Затем производится слияние полученного упорядоченного подсписка с конечным списком. Алгоритм повторяется, пока в исходном списке не останется элементов.

Тестирование.

1) Ввод: 1 1 1 1 1

Вывод:

```
                    Step 0
                        list: 1 1 1 1 1
                        tempList:
                        resultList:

        push the first  element of "list" to the end of "tempList"
list: 1 1 1 1
tempList: 1
```

```
        iteration 0 in step 0:
list: 1 1 1
tempList: 1 1

        iteration 1 in step 0:
list: 1 1
tempList: 1 1 1

        iteration 2 in step 0:
list: 1
tempList: 1 1 1 1

        iteration 3 in step 0:
list:
tempList: 1 1 1 1 1
resultList: 1 1 1 1 1

                RESULT:
Strand sort: 1 1 1 1 1
std::list::sort: 1 1 1 1 1
```

2) Ввод: 5 1 4 2 0 9 3

Вывод:

```
                              Step 0
                        list: 5 1 4 2 0 9 3
                        tempList:
                        resultList:

        push the first  element of "list" to the end of "tempList"
list: 1 4 2 0 9 3
tempList: 5

        iteration 0 in step 0:
list: 1 4 2 0 9 3
tempList: 5
```

iteration 1 in step 0:
list: 1 4 2 0 9 3
tempList: 5

iteration 2 in step 0:
list: 1 4 2 0 9 3
tempList: 5

iteration 3 in step 0:
list: 1 4 2 0 9 3
tempList: 5

iteration 4 in step 0:
list: 1 4 2 0 3
tempList: 5 9

iteration 5 in step 0:
list: 1 4 2 0 3
tempList: 5 9
resultList: 5 9

Step 1
list: 1 4 2 0 3
tempList:
resultList: 5 9

push the first element of "list" to the end of "tempList"
list: 4 2 0 3
tempList: 1

iteration 0 in step 1:
list: 2 0 3
tempList: 1 4

iteration 1 in step 1:

list: 2 0 3

tempList: 1 4

iteration 2 in step 1:

list: 2 0 3

tempList: 1 4

iteration 3 in step 1:

list: 2 0 3

tempList: 1 4

resultList: 1 4 5 9

Step 2

list: 2 0 3

tempList:

resultList: 1 4 5 9

push the first element of "list" to the end of "tempList"

list: 0 3

tempList: 2

iteration 0 in step 2:

list: 0 3

tempList: 2

iteration 1 in step 2:

list: 0

tempList: 2 3

resultList: 1 2 3 4 5 9

```
                          Step 3
                        list: 0
                        tempList:
                        resultList: 1 2 3 4 5 9

        push the first  element of "list" to the end of "tempList"
list:
tempList: 0
resultList: 0 1 2 3 4 5 9


                RESULT:
Strand sort: 0 1 2 3 4 5 9

std::list::sort: 0 1 2 3 4 5 9
```

3) Ввод: 6 4 2 0 -2 -4 -6

Вывод:

```
                    Step 0
                        list: 6 4 2 0 -2 -4 -6
                        tempList:
                        resultList:

        push the first  element of "list" to the end of "tempList"
list: 4 2 0 -2 -4 -6
tempList: 6

        iteration 0 in step 0:
list: 4 2 0 -2 -4 -6
tempList: 6

        iteration 1 in step 0:
list: 4 2 0 -2 -4 -6
tempList: 6
```

```
        iteration 2 in step 0:
list: 4 2 0 -2 -4 -6
tempList: 6

        iteration 3 in step 0:
list: 4 2 0 -2 -4 -6
tempList: 6

        iteration 4 in step 0:
list: 4 2 0 -2 -4 -6
tempList: 6

        iteration 5 in step 0:
list: 4 2 0 -2 -4 -6
tempList: 6
resultList: 6

                  Step 1
                      list: 4 2 0 -2 -4 -6
                      tempList:
                      resultList: 6

        push the first  element of "list" to the end of "tempList"
list: 2 0 -2 -4 -6
tempList: 4

        iteration 0 in step 1:
list: 2 0 -2 -4 -6
tempList: 4

        iteration 1 in step 1:
list: 2 0 -2 -4 -6
tempList: 4

        iteration 2 in step 1:
```

list: 2 0 -2 -4 -6
tempList: 4

iteration 3 in step 1:
list: 2 0 -2 -4 -6
tempList: 4

iteration 4 in step 1:
list: 2 0 -2 -4 -6
tempList: 4
resultList: 4 6

Step 2
list: 2 0 -2 -4 -6
tempList:
resultList: 4 6

push the first element of "list" to the end of "tempList"
list: 0 -2 -4 -6
tempList: 2

iteration 0 in step 2:
list: 0 -2 -4 -6
tempList: 2

iteration 1 in step 2:
list: 0 -2 -4 -6
tempList: 2

iteration 2 in step 2:
list: 0 -2 -4 -6
tempList: 2

iteration 3 in step 2:

list: 0 -2 -4 -6
tempList: 2
resultList: 2 4 6

Step 3

list: 0 -2 -4 -6
tempList:
resultList: 2 4 6

push the first element of "list" to the end of "tempList"
list: -2 -4 -6
tempList: 0

iteration 0 in step 3:
list: -2 -4 -6
tempList: 0

iteration 1 in step 3:
list: -2 -4 -6
tempList: 0

iteration 2 in step 3:
list: -2 -4 -6
tempList: 0
resultList: 0 2 4 6

Step 4

list: -2 -4 -6
tempList:
resultList: 0 2 4 6

push the first element of "list" to the end of "tempList"
list: -4 -6
tempList: -2

iteration 0 in step 4:
list: -4 -6
tempList: -2

iteration 1 in step 4:
list: -4 -6
tempList: -2
resultList: -2 0 2 4 6

Step 5
list: -4 -6
tempList:
resultList: -2 0 2 4 6

push the first element of "list" to the end of "tempList"
list: -6
tempList: -4

iteration 0 in step 5:
list: -6
tempList: -4
resultList: -4 -2 0 2 4 6

Step 6
list: -6
tempList:
resultList: -4 -2 0 2 4 6

push the first element of "list" to the end of "tempList"
list:
tempList: -6
resultList: -6 -4 -2 0 2 4 6

```
                RESULT:
Strand sort: -6 -4 -2 0 2 4 6

std::list::sort: -6 -4 -2 0 2 4 6
```

Выводы.

Был реализован алгоритм нитевидной сортировки. Лучше всего применять эту сортировку для двусвязных списков, т.к. алгоритм основан на вставке и удалении элементов. В худшем случае (например, когда последовательность отсортирована по убыванию) сложность алгоритма O(n^2) . Если исходная последовательность почти упорядочена, то сложность алгоритма O(n). В отчёте представлены результаты тестирования, разработанный код программы представлен в Приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

NODETREE.H

```
#INCLUDE <IOSTREAM>
#INCLUDE <LIST>
#INCLUDE <ITERATOR>
#INCLUDE <TYPEINFO>
#INCLUDE <STRING>

USING NAMESPACE STD;
STATIC INT STEP = 0;


TEMPLATE <TYPENAME T>
LIST<T> ENTERLIST(){
    T NUM;
    LIST<T> LIST;
    COUT<<"ENTER YOUR SEQUENCE:"<<ENDL;
    STRING N;
    DO{
        CIN >> N;
        IF (TYPEID(T) == TYPEID(INT)){
            NUM = STOI(N);
        } ELSE IF (TYPEID(T) == TYPEID(FLOAT)){
            NUM = STOF(N);
        } ELSE IF (TYPEID(T) == TYPEID(DOUBLE)){
            NUM = STOD(N);
        } ELSE IF (TYPEID(T) == TYPEID(LONG INT)){
            NUM = STOL(N);
        } ELSE {
                        COUT<<"WRONG  TYPE  OF  SEQUENCE
ELEMENTS!"<<ENDL;
            EXIT(0);
        }

        LIST.PUSH_BACK(NUM);
    }WHILE(CIN.PEEK() != '\N');
    RETURN LIST;
}



TEMPLATE <TYPENAME T>
```

```
LIST<T> STRANDSORT(LIST<T>& LIST){
    IF(LIST.SIZE()<2){
        RETURN LIST;
    }
    STD::LIST<T> TEMPLIST;
    STD::LIST<T> RESULTLIST;

    WHILE(!LIST.EMPTY()){
            COUT<<"\X1B[1;31M\T\T\T\T\TSTEP  "<<STEP<<"\
X1B[0M"<<ENDL;
        COUT<<"\X1B[1;32M\T\T\TLIST: \X1B[0M";
            FOR(TYPENAME  STD::LIST<T>::ITERATOR  ITER  =
LIST.BEGIN(); ITER != LIST.END(); ITER++){
            COUT<<*ITER<<' ';
        }
        COUT<<ENDL;
        COUT<<"\X1B[1;33M\T\T\TTEMPLIST: \X1B[0M";
            FOR(TYPENAME  STD::LIST<T>::ITERATOR  ITER  =
TEMPLIST.BEGIN(); ITER != TEMPLIST.END(); ITER++){
            COUT<<*ITER<<' ';
        }
        COUT<<ENDL;
        COUT<<"\X1B[1;31M\T\T\TRESULTLIST: \X1B[0M";
            FOR(TYPENAME  STD::LIST<T>::ITERATOR  ITER  =
RESULTLIST.BEGIN(); ITER != RESULTLIST.END(); ITER++){
            COUT<<*ITER<<' ';
        }
        COUT<<ENDL;

         COUT<<"\N\TPUSH THE FIRST  ELEMENT OF \"LIST\"
TO THE END OF \"TEMPLIST\""<<ENDL;
             TEMPLIST.PUSH_BACK(LIST.FRONT());  //  FIRST
ELEMENT OF "LIST" TO THE END OF "TEMPLIST"
          LIST.POP_FRONT();                    // DELETE
FIRST ELEMENT IN "LIST"
        COUT<<"\X1B[1;32MLIST: \X1B[0M";
            FOR(TYPENAME  STD::LIST<T>::ITERATOR  ITER  =
LIST.BEGIN(); ITER != LIST.END(); ITER++){
            COUT<<*ITER<<' ';
        }
        COUT<<ENDL;
        COUT<<"\X1B[1;33MTEMPLIST: \X1B[0M";
            FOR(TYPENAME  STD::LIST<T>::ITERATOR  ITER  =
TEMPLIST.BEGIN(); ITER != TEMPLIST.END(); ITER++){
            COUT<<*ITER<<' ';
```

```
        }
        COUT<<ENDL;

        INT I =0;
        TYPENAME LIST<T>::ITERATOR IT;
        FOR(IT = LIST.BEGIN(); IT!=LIST.END();){
                  COUT<<"\N\TITERATION  "<<I<<"  IN  STEP
"<<STEP<<":"<<ENDL;
                IF(TEMPLIST.BACK() <= *IT){   // IF LAST
ELEMENT IN "TEMPLIST" LESS THAN CURRENT IN "LIST"
                   TEMPLIST.PUSH_BACK(*IT); // CURRENT
ELEMNT IN "LIST" TO THE END OF "TEMPLIST"
                  IT = LIST.ERASE(IT);      // CURRENT
ELEMENT IS NEXT AFTER DELETED ONE
            } ELSE {
                IT++;
            }
            COUT<<"\X1B[1;32MLIST: \X1B[0M";
             FOR(TYPENAME STD::LIST<T>::ITERATOR ITER =
LIST.BEGIN(); ITER != LIST.END(); ITER++){
                COUT<<*ITER<<' ';
            }
            COUT<<ENDL;
            COUT<<"\X1B[1;33MTEMPLIST: \X1B[0M";
             FOR(TYPENAME STD::LIST<T>::ITERATOR ITER =
TEMPLIST.BEGIN(); ITER != TEMPLIST.END(); ITER++){
                COUT<<*ITER<<' ';
            }
            COUT<<ENDL;

            I++;
        }

         RESULTLIST.MERGE(TEMPLIST);       // MERGE TWO
SORTED LISTS
        COUT<<"\X1B[1;31MRESULTLIST: \X1B[0M";
            FOR(TYPENAME  STD::LIST<T>::ITERATOR  ITER  =
RESULTLIST.BEGIN(); ITER != RESULTLIST.END(); ITER++){
            COUT<<*ITER<<' ';
        }
        COUT<<ENDL;
        COUT<<"\N\N";
        STEP++;

    }
```

```
    RETURN RESULTLIST;
}



INT MAIN()
{
    STD::LIST<LONG INT> LIST = ENTERLIST<LONG INT>();

    STD::LIST<LONG INT> LISTCOPY;
     FOR(TYPENAME STD::LIST<LONG INT>::ITERATOR ITER =
LIST.BEGIN(); ITER != LIST.END(); ITER++){
        LISTCOPY.PUSH_BACK(*ITER);
    }

     STD::LIST<LONG INT> SORTEDLIST = STRANDSORT<LONG
INT>(LIST);
    LISTCOPY.SORT();

    COUT<<"\N\T\TRESULT:"<<ENDL;
    COUT<<"\X1B[1MSTRAND SORT: \X1B[0M";
     FOR(TYPENAME STD::LIST<LONG INT>::ITERATOR ITER =
SORTEDLIST.BEGIN(); ITER != SORTEDLIST.END(); ITER++){
        COUT<<*ITER<<' ';
    }
    COUT<<ENDL;
    COUT<<"\X1B[1M\NSTD::LIST::SORT: \X1B[0M";
     FOR(TYPENAME STD::LIST<LONG INT>::ITERATOR ITER =
LISTCOPY.BEGIN(); ITER != LISTCOPY.END(); ITER++){
        COUT<<*ITER<<' ';
    }
    COUT<<ENDL;


    RETURN 0;
}
```