|  |  |
| --- | --- |
|  | **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**  **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Институт (факультет) | Институт компьютерных наук |
| Кафедра | Кафедра прикладной математики |

ЛАБОРАТОРНАЯ РАБОТА № 1

По дисциплине: «Технологии программирования и структуры данных»

На тему: «Программирование базовых алгоритмов сортировки».

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ПМ-24-1 |  |  |  | Сараев А.П. |
|  | группа |  | подпись, дата |  | фамилия, инициалы |
|  |  |  |  |  |  |
| Руководитель |  |  |  |  |  |
| доцент кафедры АСУ | |  |  |  | Мирсаитов Г.Р. |
| ученая степень, ученое звание | |  | подпись, дата |  | фамилия, инициалы |

Липецк 2024

**Цель работы**

Приобретение навыков реализации базовых алгоритмов сортировки массивов данных.

**Задание кафедры**

Осуществить программную реализацию сортировки информации заданного вида сбалансированным N-ленточным слиянием (в оперативной памяти), используя выбранные из табл. 1, в соответствии с вариантом, алгоритм внутренней сортировки и формат исходных данных.

**Задания к лабораторной работе №1**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Тип ключевого поля записи** | **Типы информационных полей записи** | **Метод внутренней сортировки** |
| 19 | int | char[], float[] | Метод Шелла |

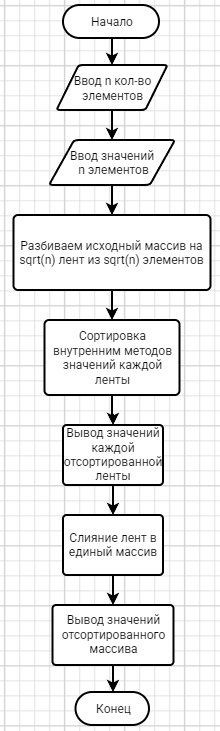
**Краткие теоретические сведения**

Сбалансированное N-ленточное слияние - это метод сортировки больших объемов данных, который использует множество вспомогательных файлов (лент) для сортировки данных. Процесс состоит из нескольких шагов:

1. Разделение исходных данных на лент из элементов.
2. Сортировка каждой ленты внутренним алгоритмом сортировки.
3. Последовательное слияние отсортированных лент в одну, пока все данные не будут объединены в одну отсортированную последовательность.

Сортировка Шелла. Метод предполагает разбиение исходной последовательности на группы элементов и сортировку элементов каждой группы методом вставок. Элементы, принадлежащие одной группе, находятся на определенном расстоянии друг от друга. При необходимости перестановки элементы перемещаются скачком на это расстояние, а не на одну позицию, как в методе вставок.

**Блок-схема**



**Текст программы**

#include <iostream>

#include <ctime>

#include <cmath>

using namespace std;

struct element {

int id;

char\* a;

float\* f;

};

struct lenta {

element\* elems;

int size = 0;

int cur\_ind = 0;

};

int gen\_id() {

return rand() % 300 - 100;

}

char\* gen\_chars() {

int cnt = rand() % 20 + 1;

char\* str = new char[cnt + 1];

str[cnt] = '\0';

for (int i = 0; i < cnt; i++) {

str[i] = char('!' + (rand() % 94));

}

return str;

}

float\* gen\_floats() {

float\* fl = new float[3];

for (int i = 0; i < 3; i++)

fl[i] = float(rand() % 200) / float(rand() % 200);

return fl;

}

void sort\_Shell(element els[], int n) {

for (int step = n / 2; step > 0; step /= 2) {

for (int i = step; i < n; i++) {

element cur = els[i];

int j;

for (j = i; j >= step && els[j - step].id > cur.id; j -= step)

els[j] = els[j - step];

els[j] = cur;

}

}

}

void merge\_lents(lenta\* lents, lenta& res, int sz, int n) {

for (int i = 0; i < n; i++) {

element mn;

mn.id = INT\_MAX;

int ind = -1;

for (int j = 0; j < sz; j++) {

if (lents[j].cur\_ind < lents[j].size && lents[j].elems[lents[j].cur\_ind].id < mn.id)

ind = j, mn = lents[j].elems[lents[j].cur\_ind];

}

res.elems[res.size] = mn;

res.size++;

lents[ind].cur\_ind++;

}

}

lenta balanced\_n\_lents(element els[], int n) {

lenta ans;

ans.elems = new element[n];

int sz = ceil(sqrt(n));

lenta\* lents = new lenta[sz];

for (int i = 0; i < sz; i++) {

lents[i].elems = new element[sz];

}

for (int i = 0; i < n; i++) {

lents[i % sz].elems[lents[i % sz].size] = els[i];

lents[i % sz].size += 1;

}

for (int i = 0; i < sz; i++) {

for (int j = 0; j < lents[i].size; j++) {

cout << lents[i].elems[j].id << " ";

}

cout << '\n';

}

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

for (int i = 0; i < sz; i++) {

sort\_Shell(lents[i].elems, lents[i].size);

}

for (int i = 0; i < sz; i++) {

for (int j = 0; j < lents[i].size; j++) {

cout << lents[i].elems[j].id << " ";

}

cout << '\n';

}

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

/\*for (int i = 0; i < sz; i++)

merge\_lents(lents[i], ans);\*/

merge\_lents(lents, ans, sz, n);

for (int i = 0; i < n; i++) {

cout << ans.elems[i].id << " ";

}

cout << '\n';

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

return ans;

}

int main() {

srand(time(NULL));

int n;

cout << "Input integer number of elements (from 1 to 100): ";

while (cin >> n) {

if (n < 1 || n > 100) {

cout << "Invalid input.\n";

cout << "Input integer number of elements (from 1 to 100): ";

}

else

break;

}

lenta els;

els.elems = new element[n];

for (int i = 0; i < n; i++) {

els.elems[i].id = gen\_id();

els.elems[i].a = gen\_chars();

els.elems[i].f = gen\_floats();

}

for (int i = 0; i < n; i++) {

cout << els.elems[i].id << " ";

cout << els.elems[i].a << " - ";

for (int j = 0; j < 3; j++) {

cout << els.elems[i].f[j] << " ";

}

cout << '\n';

}

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

cout << "\_-----------------------------\_\n";

els = balanced\_n\_lents(els.elems, n);

for (int i = 0; i < n; i++) {

cout << els.elems[i].id << " ";

cout << els.elems[i].a << " - ";

for (int j = 0; j < 3; j++) {

cout << els.elems[i].f[j] << " ";

}

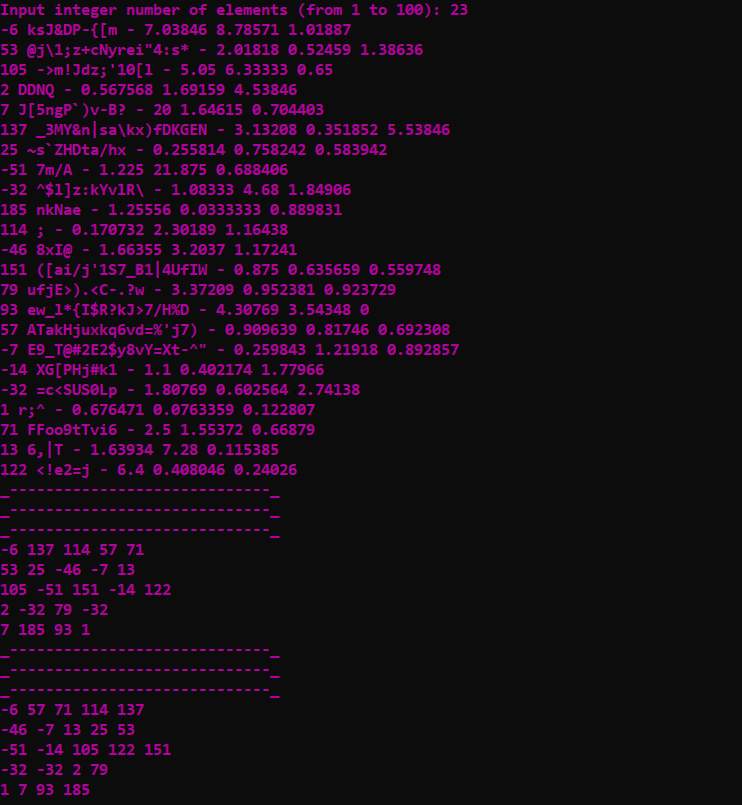
cout << '\n';

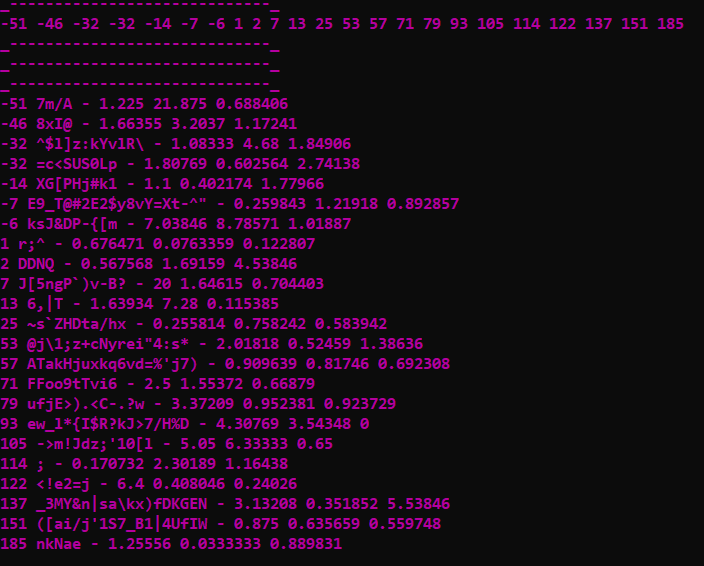
}

return 0;

}

**Контрольный пример**





**Вывод**

1. Сбалансированное N-ленточное слияние - эффективный метод сортировки больших объемов данных, особенно при ограниченных ресурсах памяти.
2. Метод Шелла может быть эффективно использован для сортировки лент внутри алгоритма сбалансированного слияния, который минимален по памяти и обеспечивает высокую скорость сортировки.
3. Программа успешно справилась с сортировкой данных и демонстрирует хорошую производительность на различных входных данных.