

**LAB6.h**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <random>

#include <string>

#include <numeric>

extern int c;

extern double d;

int get\_rand();

using namespace std;

int hash\_div(int key, int table\_size);

int hash\_mul(int key, int table\_size);

//диапазон сумм ключей для аддитивного метода 288 <= hash <= 732 (445 значений)

struct Key {

char value[7];

Key() {

for (int i = 0; i < 7; i++) value[i] = 0;

}

};

int to\_int(Key& key);

struct Keys {

Key\* key;

int length;

Keys(int length) {

this->length = length;

key = new Key[length];

for (int i = 0; i < length; i++) key[i] = Key();

}

~Keys() {

delete[] key;

}

};

class Table {

public:

int specific\_column\_size = 0;

string legendr1 = "";

string legendr2 = "";

bool avg\_step\_only = false;

int step\_scale = 10;

int display\_h = 10;

int display\_w\_limit = 190;

int actual\_size = 0;

int length;

Key\* inclusions;

int\* put\_path;

int put\_linear\_div(Key& k, int table\_size) {

actual\_size++;

int index0 = hash\_div(to\_int(k), table\_size);

if (inclusions[index0].value[0] == '\0') {

inclusions[index0] = k;

put\_path[actual\_size] = 0;

return 0;

}

for (int i = 1; i <= table\_size; i++) {

int index = (index0 + c \* i) % table\_size;

if (inclusions[index].value[0] == '\0') {

strcpy(inclusions[index].value, k.value);

put\_path[actual\_size] = i;

return i;

}

}

actual\_size--;

throw exception("Failed to insert LD");

}

int put\_linear\_mul(Key& k, int table\_size) {

actual\_size++;

int index0 = hash\_mul(to\_int(k), table\_size);

if (inclusions[index0].value[0] == '\0') {

inclusions[index0] = k;

return 0;

}

for (int i = 1; i <= table\_size; i++) {

int index = (index0 + c \* i) % table\_size;

if (inclusions[index].value[0] == '\0') {

strcpy(inclusions[index].value, k.value);

put\_path[actual\_size] = i;

//cout << i << ' ' << index0 << ' ' << actual\_size << endl;

return i;

}

}

actual\_size--;

throw exception("Failed to insert LM");

}

int put\_squared\_div(Key& k, int table\_size) {

actual\_size++;

int index0 = hash\_div(to\_int(k), table\_size);

if (inclusions[index0].value[0] == '\0') {

inclusions[index0] = k;

return 0;

}

int next\_step;

for (int i = 1; (next\_step = index0 + c\*i + d\*i\*i) > 0; i++) {

int index = next\_step % table\_size;

if (inclusions[index].value[0] == '\0') {

strcpy(inclusions[index].value, k.value);

put\_path[actual\_size] = i;

return i;

}

}

actual\_size--;

throw exception("Failed to insert SD");

}

int put\_squared\_mul(Key& k, int table\_size) {

actual\_size++;

int index0 = hash\_mul(to\_int(k), table\_size);

if (inclusions[index0].value[0] == '\0') {

inclusions[index0] = k;

return 0;

}

int next\_step;

for (int i = 1; (next\_step = index0 + c \* i + d \* i \* i) > 0; i++) {

int index = next\_step % table\_size;

if (inclusions[index].value[0] == '\0') {

strcpy(inclusions[index].value, k.value);

put\_path[actual\_size] = i;

return i;

}

}

actual\_size--;

throw exception("Failed to insert SM");

}

void clear() {

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

inclusions[i] = Key();

inclusions[i].value[0] = '\0';

put\_path[i] = 0;

}

put\_path[length] = 0;

actual\_size = 0;

}

Table(int length) {

this->length = length;

inclusions = new Key[length];

put\_path = new int[length+1];

clear();

}

~Table(){

delete[] inclusions;

}

int calculate\_avg\_max() {

int step = 1;

float acc = 0;

int avg\_steps = 0;

int avg\_max = 0;

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

acc += put\_path[i];

avg\_steps++;

if (!avg\_step\_only && i < step \* (length / (float)display\_w\_limit) || avg\_steps < step\_scale) continue;

if (acc != 0) {

acc = acc / avg\_steps;

avg\_max = max((float)avg\_max, ceil(acc));

}

avg\_steps = 0;

acc = 0;

step++;

}

return avg\_max;

}

operator string() {

string result = "";

result += " ";

for (int i = 0; i < min(length / step\_scale, display\_w\_limit) && !avg\_step\_only || avg\_step\_only && i < length / step\_scale; i++) result += '-';

result += '\n';

int mcol;

for (int row = 0; row < display\_h; row++) {

result += '|';

int avg\_max = specific\_column\_size;

if (!specific\_column\_size) {

avg\_max = calculate\_avg\_max();

}

int step = 1;

int acc = 0;

int empty\_count = 0;

int avg\_steps = 0;

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

empty\_count += (put\_path[i] == 0);

acc += put\_path[i];

avg\_steps++;

if (!avg\_step\_only && i < step \* (length / (float)display\_w\_limit) || avg\_steps < step\_scale) continue;

if (acc == 0) acc = -1;

else acc = acc/avg\_steps;

avg\_steps = 0;

result +=

(acc >= (display\_h - row) \* (avg\_max / (float)display\_h)) ? (empty\_count == 0) ? (char)178 : (empty\_count == 1) ? (char)177 : (char) 176 :

(acc && acc >= (display\_h - row - 1) \* (avg\_max / (float)display\_h)) ? (empty\_count == 0) ? (char)177 : (empty\_count == 1) ? (char)176 : '\_'

: ' ';

acc = 0;

empty\_count = 0;

step++;

}

result += "|";

int col = (int)round((display\_h - row) \* (avg\_max / (float)display\_h));

if(row == 0) result += "\033[107m\033[30m\033[4m";

result += to\_string(col);

if (row == 0) result += "\033[49m\033[0m\033[24m";

if (row == 0) {

mcol = col;

result += " " + legendr1;

}

else if (row == 1) {

for (int i = to\_string(col).size(); i < to\_string(mcol).size(); i++) result += " ";

result += " " + legendr2;

}

else if (row == 2 && specific\_column\_size) result += " [M] MAX: " + to\_string(calculate\_avg\_max());

result += "\n";

}

result += " ";

for (int i = 0; i < min(length / step\_scale, display\_w\_limit) && !avg\_step\_only || avg\_step\_only && i < length / step\_scale; i++) result += '-';

result += '\n';

return result;

}

};

**hash.cpp**

#include "LAB6.h"

int get\_rand() {

char c[4] = { rand() % 256, rand() % 256, rand() % 256, rand() % 128 };

int result = (unsigned)\*((int\*)c);

return result;

}

int hash\_div(int key, int table\_size) {

return key % table\_size;

}

int hash\_mul(int key, int table\_size) {

double rand\_value = (sqrt(5) - 1) \* 0.5;//0,61803398875...

//while (abs(table\_size / rand\_value - floor(table\_size / rand\_value) - 0.5) > 0.3) rand\_value = 1-sqrt(rand\_value)\*rand\_value;

double m = rand\_value \* key;

return table\_size \* (m - floor(m));

}

char xor\_array[6] = { rand() % 256, rand() % 256, rand() % 256, rand() % 256, rand() % 256, rand() % 256};

int to\_int(Key& key) {

int hash = 0;

for (int i = 0; i < 6; i++) {

hash += 128 + ((key.value[i] ^ xor\_array[i % 6]));

}

return hash;

//return get\_rand();

}

**AISD\_LAB6.cpp**

#include "LAB6.h"

#include <Windows.h>

#include <stdio.h>

#include <conio.h>

int c;

double d;

void ScrollUp()

{

CONSOLE\_SCREEN\_BUFFER\_INFO csbiInfo;

SMALL\_RECT srctWindow;

if (!GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbiInfo))

printf("GetConsoleScreenBufferInfo (%d)\n", GetLastError());

srctWindow = csbiInfo.srWindow;

if (true) {

int offs = srctWindow.Top;

srctWindow.Top = -offs; // move top up

srctWindow.Bottom -= offs; // move bottom up

if (!SetConsoleWindowInfo(

GetStdHandle(STD\_OUTPUT\_HANDLE), // screen buffer handle

TRUE, // absolute coordinates

&srctWindow)) // specifies new location

;//printf("SetConsoleWindowInfo (%d)\n", GetLastError());

}

else

;// printf("\nCannot scroll; the window is too close to the top.\n");

}

void fill(Keys& keys) {

for (int i = 0; i < keys.length; i++) {

Key& key = keys.key[i];

for (int j = 0; j < 6; j++) {

char& c = key.value[j];

int r = rand()%62;

c = r;

c +=

(0 <= r && r <= 9) ? '0' - 0 :

(10 <= r && r <= 35) ? 'A' - 10 :

(36 <= r && r <= 61) ? 'a' - 36 :

0;

}

}

}

int pick\_cd(int table\_size) {

int a, b;

int M = table\_size;

do {

c = 1+table\_size/8;

c += (get\_rand() % table\_size) / 5;

a = c;

b = M;

while (b != 0) {

int t = a % b;

a = b;

b = t;

}

} while (a > 1 || b > 1);

d = sqrt((c - 1) \* (sqrt(5) - 1) \* 0.5);

return c;

}

int main() {

cout << "\033[1m";

HWND console = GetConsoleWindow();

RECT r;

GetWindowRect(console, &r);

ShowWindow(console, SW\_MAXIMIZE);

CONSOLE\_FONT\_INFOEX cfi = { sizeof(CONSOLE\_FONT\_INFOEX) };

HANDLE hcsb = CreateFileA("CONOUT$", GENERIC\_WRITE | GENERIC\_READ, FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

GetCurrentConsoleFontEx(hcsb, FALSE, &cfi);

cfi.dwFontSize.X \*= 0.95;

cfi.dwFontSize.Y \*= 0.95;

SetCurrentConsoleFontEx(hcsb, FALSE, &cfi);

CloseHandle(hcsb);

int table\_size, key\_size;

key\_size = 2000;

table\_size = 2000;

bool full\_auto = false;

bool unify\_max = false;

bool manual\_c = true;

if (!full\_auto) {

cout << "SKIP manual \"c\" and \"d\" assignment? (0 / 1) >> ";

cin >> manual\_c;

}

manual\_c = !manual\_c;

if (manual\_c) {

cout << "c(int) d(int) >> ";

cin >> c >> d;

}

while (true) {

if (!full\_auto) {

cout << "key\_size(int) table\_size(int) >> ";

cin >> key\_size >> table\_size;

system("cls");

}

if (!manual\_c) pick\_cd(table\_size);

Keys keys = Keys(key\_size);

fill(keys);

//system("pause");

Table tableLD = Table(table\_size);

Table tableSD = Table(table\_size);

Table tableLM = Table(table\_size);

Table tableSM = Table(table\_size);

Table res\_tableLD = Table(table\_size);

Table res\_tableSD = Table(table\_size);

Table res\_tableLM = Table(table\_size);

Table res\_tableSM = Table(table\_size);

//tableLD.avg\_step\_only = true;

//tableLM.avg\_step\_only = true;

//system("cls");

cout << key\_size << " keys on " << table\_size << " addresses (" << (int)(100\*((float)key\_size/table\_size)) << "% c=" << c << " d=" << (int)d << "." << (int)(d\*1000) << "):\n";

for (int i = 0; i < key\_size; i++) {

try {

tableLD.put\_linear\_div(keys.key[i], table\_size);

tableSD.put\_squared\_div(keys.key[i], table\_size);

tableLM.put\_linear\_mul(keys.key[i], table\_size);

tableSM.put\_squared\_mul(keys.key[i], table\_size);

}

catch (exception e) {

cout << '[' << e.what() << ']' << endl;

i--;

cout << "Keys will be generated again" << endl;

system("pause");

fill(keys);

continue;

return 0;

}

}

tableLD.legendr1 = "Linear";

tableLD.legendr2 = "Divide";

tableSD.legendr1 = "Square";

tableSD.legendr2 = "Divide";

tableLM.legendr1 = "Linear";

tableLM.legendr2 = "Multiply";

tableSM.legendr1 = "Square";

tableSM.legendr2 = "Multiply";

if (unify\_max) {

int mx = max(tableLD.calculate\_avg\_max(), tableSD.calculate\_avg\_max());

mx = max(mx, tableLM.calculate\_avg\_max());

mx = max(mx, tableSM.calculate\_avg\_max());

tableLD.specific\_column\_size = mx;

tableSD.specific\_column\_size = mx;

tableLM.specific\_column\_size = mx;

tableSM.specific\_column\_size = mx;

}

cout << string(tableLD);

cout << string(tableSD);

cout << string(tableLM);

cout << string(tableSM);

//ScrollUp();

if(full\_auto) system("pause");

//table\_size += 250;

//key\_size += 250;

}

return 0;

}