**ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА**

**Факультет прикладної математики**

**Кафедра математичного забезпечення ЕОМ**

**Звіт**

з лабораторної роботи № 3

з курсу «Дискретні структури»

на тему: «Список з пропусками(Skip-List). Операції»

Виконав:

студент 3-го курсу

групи ПЗ-21у-1

Войцехов М.О.

м. Дніпро

2022 р.

**Постановка задачі**

Реалізувати структуру Skip-List. Визначена таким чином структура даних повинна мати властивості:

-розмірSkip-List(size, кількістьелементіввньому);

-кількість рівнів MaxLevel;

-вузол Node з полями {key, value, Level\_Node},де тип значення value-шаблонного типу (для перевірки організації та методів структури даних обирати цілочисельний тип).

Для виконання операцій з елементами заданої структури створити методи, які реалізують:

-додавання елементу;

-пошук елемента;

-видалення елементу;

-створення копії Skip-List;

-очищення Skip-List.

Надати можливість обирати для структури Skip-List номер рівня для нової вершини двома способами:

-для p виконується умова(кожен 2𝑖-ий вузол містить покажчик на 2𝑖вузлів вперед);

-p генерується випадково( з використанням генератора випадкових чисел). Надати можливість обирати для кількість рівнів MaxLevel:

-константа (наприклад, MaxLevel=3);

-не обмежена (𝑀𝑎𝑥𝐿𝑒𝑣𝑒𝑙→∞);

-обирається за умовою MaxLewel=function(p, N)(N-кількість елементів у списку).

**Програмний код**

#include <vector>

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <cstring>

const float P = 0.5;

using namespace std;

struct snode

{

int value;

snode\*\* forw;

snode(int level, int& value)

{

forw = new snode \* [level + 1];

memset(forw, 0, sizeof(snode\*) \* (level + 1));

this->value = value;

}

~snode()

{

delete[] forw;

}

};

struct skiplist

{

snode\* header;

int value;

int level;

int max\_level;

int mode;

skiplist()

{

max\_level = 20;

header = new snode(max\_level, value);

level = 0;

mode = 0;

}

~skiplist()

{

delete header;

}

void display();

bool contains(int&);

void insert\_element(int&,int);

void delete\_element(int&);

void copy(skiplist& target);

void clear();

int random\_level();

int getCount();

void updateLevels(int);

};

void skiplist::display()

{

if (header->forw[0] == NULL) {

cout << endl << "List empty";

}

else {

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

const snode\* x = header->forw[i];

cout << "level " << i << " : ";

while (x != NULL)

{

cout << x->value;

x = x->forw[i];

if (x != NULL)

cout << " -> ";

}

cout << endl;

}

}

}

bool skiplist::contains(int& s\_value)

{

snode\* x = header;

for (int i = level; i >= 0; i--)

{

while (x->forw[i] != NULL && x->forw[i]->value < s\_value)

{

x = x->forw[i];

}

}

x = x->forw[0];

return x != NULL && x->value == s\_value;

}

void skiplist::copy(skiplist& target) {

target.level = level;

snode\* x = header;

snode\*\* update = new snode \* [max\_level + 1];

memset(update, 0, sizeof(snode\*) \* (max\_level + 1));

for (int i = level; i >= 0; i--)

{

update[i] = x->forw[i];

}

snode\*\* newupdate = new snode \* [max\_level + 1];

memset(newupdate, 0, sizeof(snode\*) \* (max\_level + 1));

int counter = 0;

while (update[0] != NULL) {

int lvl = 0;

for (int i = 1; i <= level; i++) {

if (update[i] != NULL) {

if (update[i]->value == update[0]->value) lvl++;

}

}

snode\* newNode = new snode(lvl, update[0]->value);

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

if (newupdate[i] == NULL) {

newupdate[i] = newNode;

target.header->forw[i] = newupdate[i];

}

else {

newupdate[i]->forw[i] = newNode;

newupdate[i] = newupdate[i]->forw[i];

}

update[i] = update[i]->forw[i];

}

counter++;

}

}

void skiplist::clear() {

while (header->forw[0] != NULL) {

delete\_element(header->forw[0]->value);

}

}

int skiplist::getCount() {

int count = 0;

snode\* x = header->forw[0];

while (x != NULL) {

count++;

x = x->forw[0];

}

return count;

}

void skiplist::updateLevels(int max) {

if (max != max\_level) {

snode\* x = header->forw[0];

vector<int> values;

while (x != NULL) {

values.push\_back(x->value);

x = x->forw[0];

}

clear();

max\_level = max;

delete header;

header = new snode(max\_level, value);

for (int i = 0; i < values.size(); i++) {

insert\_element(values[i], i + 1);

}

}

}

int main()

{

skiplist ss;

skiplist copy;

int ans, n,b;

while (true)

{

cout << "1.Insert " << endl;

cout << "2.Delete " << endl;

cout << "3.Search " << endl;

cout << "4.Cout " << endl;

cout << "5.Copy " << endl;

cout << "6.Clear " << endl;

cout << "7.Change deep " << endl;

cout << "8.Exit " << endl;

cout << "Enter variant: ";

cin >> ans;

switch (ans)

{

case 1:

cout << "1 - random pos / 2 - algorithm: ";

cin >> b;

cout << "Enter element ";

cin >> n;

if (b == 1) {

ss.insert\_element(n,-1);

}

else

{

ss.insert\_element(n, 0);

}

break;

case 2:

cout << "Enter the element ";

cin >> n;

if (!ss.contains(n))

{

cout << "Element not found" << endl;

break;

}

ss.delete\_element(n);

if (!ss.contains(n))

cout << "Element Deleted" << endl;

break;

case 3:

cout << "Enter the element: ";

cin >> n;

if (ss.contains(n))

cout << "Element " << n << " is in the list" << endl;

else

cout << "Element not found" << endl;

case 4:

cout << "The List is: " << endl;

ss.display();

break;

case 5:

ss.copy(copy);

cout << "The List is: " << endl;

copy.display();

copy.clear();

break;

case 6:

ss.clear();

cout << "List cleared" << endl;

break;

case 7:

cout << "Select max level type 1 - constant/ 2 - max(20)/ 3- function: ";

cin >> n;

switch (n) {

case 1:

cout << "Enter the element to be inserted: ";

cin >> n;

ss.updateLevels(n);

break;

case 2:

ss.updateLevels(20);

break;

case 3:

int count = ss.getCount();

n = 1;

for (int i = 2; i < 20; i++) {

if (count > pow(2, i)) n++;

}

ss.updateLevels(n+1);

break;

}

break;

case 8:

exit(1);

break;

default:

cout << "Wrong Choice" << endl;

}

}

return 0;

}

float frand()

{

return (float)rand() / RAND\_MAX;

}

int skiplist::random\_level()

{

static bool first = true;

if (first)

{

srand((unsigned)time(NULL));

first = false;

}

float random = frand();

int lvl = (int)(log(random) / log(1. - P));

cout << endl << "lvl = " << lvl << " frand = " << random << "log = " << (1. -P) << endl;

return lvl < max\_level ? lvl : max\_level;

}

void skiplist::insert\_element(int& value, int first)

{

if (first == 0) {

mode = 0;

snode\* x = header->forw[0];

vector<int> values;

if (!contains(value)) {

bool inserted = false;

while (x != NULL) {

if (x->value > value && inserted == false) {

values.push\_back(value);

inserted = true;

}

values.push\_back(x->value);

x = x->forw[0];

}

if(inserted == false) values.push\_back(value);

clear();

for (int i = 0; i < values.size(); i++) {

insert\_element(values[i], i + 1);

}

}

else {

cout << endl << "Element exist";

}

}

else

{

mode = -1;

snode\* x = header;

snode\*\* update = new snode \*[max\_level + 1];

memset(update, 0, sizeof(snode\*) \* (max\_level + 1));

for (int i = level; i >= 0; i--)

{

while (x->forw[i] != NULL && x->forw[i]->value < value)

{

x = x->forw[i];

}

update[i] = x;

}

x = x->forw[0];

if (x == NULL || x->value != value)

{

int lvl = 0;

if(first == -1) lvl = random\_level();

else {

for (int i = 1; i < max\_level; i++) {

if (first % (int)pow(2, i) == 0) lvl = i;

}

}

if (lvl > level)

{

for (int i = level + 1; i <= lvl; i++)

{

update[i] = header;

}

level = lvl;

}

x = new snode(lvl, value);

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

{

x->forw[i] = update[i]->forw[i];

update[i]->forw[i] = x;

}

}

}

}

void skiplist::delete\_element(int& value)

{

snode\* x = header;

snode\*\* update = new snode \* [max\_level + 1];

memset(update, 0, sizeof(snode\*) \* (max\_level + 1));

for (int i = level; i >= 0; i--)

{

while (x->forw[i] != NULL && x->forw[i]->value < value)

{

x = x->forw[i];

}

update[i] = x;

}

x = x->forw[0];

if (x->value == value)

{

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

{

if (update[i]->forw[i] != x)

break;

update[i]->forw[i] = x->forw[i];

}

delete x;

while (level > 0 && header->forw[level] == NULL)

{

level--;

}

}

}

**Отримані результати**

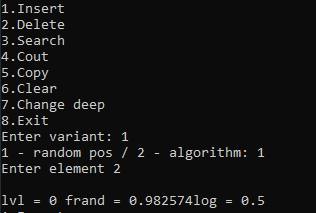


Рисунок 1 ­­– Додавання

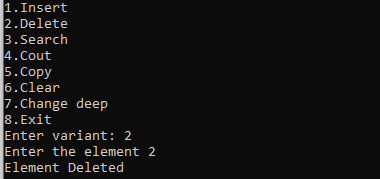


Рисунок 2 ­­– Видалення

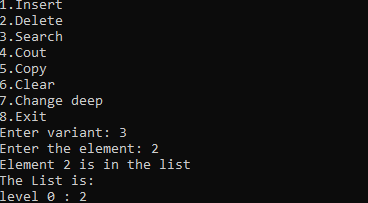


Рисунок 3 ­­– Пошук

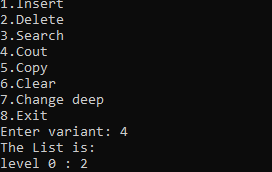


Рисунок 4 ­­– Вивід

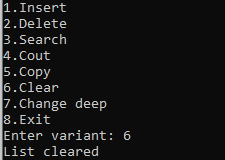


Рисунок 5 ­­– Видалення