

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №3**

**з дисципліни «Інженерія якості програмного забезпечення»**

**на тему: «Проектування структур даних ефективних за показниками обчислювальної складності»**

Виконала: ???

Прийняв: ???

Дніпро, 2022

**Тема**. Проектування структур даних ефективних за показниками обчислювальної складності.

**Мета роботи**. Вивчити комбінаторно-ймовірнісні методи визначення показників обчислювальної складності алгоритмів та отримати навички застосування цих методів при вирішенні задач розробки ефективних структур даних.

**Постановка задачі**

Розробити об’єктно-орієнтовану програму, що реалізує три структури даних, визначивши для них операції додавання, видалення, пошуку елементів.

Для тексту програми на заданій мові програмування виконати пошук усіх лексем класу, визначеного варіантом. Знайдені лексеми занести в розроблені структури, виключаючи повтор елементів.

Виконати ймовірнісно-комбінаторну оцінку функцій для роботи зі структурами, на основі якої зробити висновок про оптимальність структури з точки зору найменшої обчислювальної складності.

Мова програмування відповідно варіанту: visual basic.

Класи лексем відповідно варіанту: ідентифікатори.

Структури даних

1. Двозв’язний впорядкований список

2. Черга зі зв’язним представленням

3. Стек зі зв’язним представленням

**Інформація про структури даних**

**Двозв’язний впорядкований список**

Двозв’язний список — одна з найважливіших структур даних, в якій елементи лінійно впорядковані, але порядок визначається не номерами елементів, а вказівниками, які входять до складу елементів списку та вказують на наступний та попередній елементи. Список має «голову» — перший елемент та «хвіст» — останній елемент.

Проект структури даних на абстрактному рівні:

Структура даних – двозв’язний впорядкований список. Призначення – зберігання інформації про елементи списку, а також про вказівники на попередній та наступний елементи списку. Операції: додавання елементу до списку за значенням (виконується при умові впорядкування списку за зростанням), видалення елементу зі списку за значенням та пошуку елементу списку за значенням. Елементи списку впорядковані за зростанням.

**Черга зі зв’язним представленням**

Чергою називається такий послідовний список із змінній довжиною, в якому включення елементів виконується тільки з одного боку списку (хвіст черги), а виключення – з другого боку (голова черги).

Проект структури даних на абстрактному рівні:

Структура даних – черга зі зв’язним представленням. Призначення – зберігання інформації про елементи черги, а також про порядок додавання елементів у чергу. Операції: додавання елементу до черги за значенням (додавання виконується у кінець черги), видалення елементу з черги (видалення виконується з початку черги) та пошуку елементу черги за значенням.

**Стек зі зв’язним представленням**

Стек — різновид лінійного списку, структура даних, яка працює за принципом (дисципліною) «останнім прийшов — першим пішов». Всі операції (наприклад, видалення елемента) в стеку можна проводити тільки з одним елементом, який міститься на верхівці стека та був уведений в стек останнім.

Проект структури даних на абстрактному рівні:

Структура даних – стек зі зв’язним представленням. Призначення – зберігання інформації про елементи стеку, а також про порядок додавання елементів у стек. Операції: додавання елементу до стеку за значенням (додавання виконується у початок стеку), видалення елементу зі стеку (видалення виконується з початоку стеку) та пошуку елементу стеку за значенням.

**Критерії вибору мови програмування**

Для реалізації обраних структур та їх операцій була обрана мова програмування С++.

1) Зручна відладка коду;

2) Можливість роботи з вказівниками та адресами;

3) Велика кількість літератури про реалізацію структур даних;

4) Є досвід роботи з даною мовою програмування.

**Текст розробленої програми**

Файл List.h:

#pragma once

#include<string>

class List

{

private:

struct Node

{

std::string field; // поле данных

Node\* next=nullptr; // указатель на следующий элемент

Node\* prev=nullptr; // указатель на предыдущий элемент

};

Node\* head = nullptr;

void add(std::string value, Node\* node);

public:

List() {};

~List() {};

void add(std::string value);

Node\* find\_node(std::string value);

Node\* search(std::string value);

void del(std::string value);

void show\_list();

};

Файл List.cpp:

#include "List.h"

#include <iostream>

List::Node\* List::search(std::string value)

{

Node\* iterator = head;

while (iterator != nullptr)

{

if (iterator->field == value)

break;

iterator = iterator->next;

}

return iterator;

}

List::Node\* List::find\_node(std::string value)

{

Node\* iterator = head;

if (iterator==nullptr || iterator->field > value)

return nullptr;

while (iterator->next != nullptr)

{

if (iterator->next->field > value)

break;

iterator = iterator->next;

}

return iterator;

}

void List::add(std::string value, Node\* node=nullptr)

{

Node\* tmp = new Node;

tmp->field = value;

if (node == nullptr || head==nullptr)

{

tmp->next = head;

if (tmp->next != nullptr)

tmp->next->prev = tmp;

head = tmp;

return;

}

tmp->next = node->next;

tmp->prev = node;

node->next = tmp;

}

void List::add(std::string value)

{

add(value, find\_node(value));

}

void List::show\_list()

{

Node\* info;

info = head;

while (info)

{

std::cout << info->field << " ";

info = info->next;

}

std::cout << "\n";

}

void List::del(std::string value)

{

Node\* tmp = search(value);

if (tmp == nullptr)

return;

if (tmp == head)

head = head->next;

if(tmp->next!=nullptr)

tmp->next->prev = tmp->prev;

if (tmp->prev != nullptr)

tmp->prev->next = tmp->next;

delete tmp;

}

Файл Queue.h:

#pragma once

#include <iostream>

#include <string>

struct Node

{

std::string data;

Node\* next = nullptr;

};

class Queue

{

private:

Node\* Head = nullptr;

Node\* Tail = nullptr;

public:

Queue() {};

~Queue();

void push(std::string val);

std::string pop();

Node\* find(std::string val);

void remove(std::string a);

void show()

{

Node\* temp = Head; //Временный указатель на начало списка

while (temp != nullptr) //Пока в списке что-то встречается

{

std::cout << temp->data << " "; //Выводим значения из списка на экран

temp = temp->next; //Сдвигаем указатель на начало на адрес следующего элемента

}

std::cout << std::endl;

}

};

Файл Queue.cpp:

#include "Queue.h"

Queue::~Queue()

{

Node\* temp = Head;

while (temp != nullptr)

{

temp = Head->next;

delete Head;

Head = temp;

}

}

void Queue::push(std::string val)//добавление

{

Node\* temp = new Node;

temp->data = val;

if (Head != nullptr)

{

Tail->next = temp;

Tail = temp;

}

else

{

Head = Tail = temp;

}

}

std::string Queue::pop()//удаление

{

std::string data = "";

if (Head != nullptr)

{

Node\* temp = Head;

data = Head->data;

Head = Head->next;

delete temp;

}

else

{

throw "Try to pop from an empty queue";

}

return data;

}

void Queue::remove(std::string a)//удаление

{

Node\* temp = Head;

if (Head->data == a)

{

pop();

return;

}

while (temp != nullptr && temp->next!=nullptr)

{

if (temp->next->data == a)

{

Node\* temp\_next = temp->next;

temp->next = temp\_next->next;

delete temp\_next;

return;

}

temp = temp->next;

}

}

Node\* Queue::find(std::string val)//поиск

{

Node\* temp = Head;

while (temp != nullptr)

{

if (temp->data == val)

return temp;

else

temp = temp->next;

}

return nullptr;

}

Файл stack.h:

#pragma once

#include<string>

class Stack

{

private:

struct Node

{

std::string item;

Node \* next;

};

Node\* top=nullptr; // указатель на вершину стека

public:

Stack() {};

~Stack() {};

void add(std::string value);

bool search(std::string value);

void del();

void show\_stack();

};

Файл stack.cpp:

#include"stack.h"

#include<iostream>

#define stas std

void Stack::add(std::string value)

{

Node\* p=new Node;

p->item = value;

p->next = top; // p указывает на 1-й элемент

top = p;

}

bool Stack::search(std::string value)

{

Node\* iterator = top;

while (iterator != nullptr)

{

if (iterator->item == value)

break;

iterator = iterator->next;

}

return iterator!=nullptr;

}

void Stack::del()

{

if (top == nullptr)

return;

Node\* p2; // дополнительный указатель

std::string item;

item = top->item;

p2 = top;

top = top->next;

// Освободить память, выделенную под 1-й элемент

delete p2;

}

void Stack::show\_stack()

{

if (top == nullptr)

std::cout << "stack is empty." << std::endl;

else

{

Node\* p; // дополнительный указатель

p = top;

while (p != nullptr)

{

stas::cout << p->item << " ";

p = p->next;

}

std::cout << stas::endl;

}

}

Файл Source.cpp:

#include<iostream>

#include"List.h"

#include"stack.h"

#include"Queue.h"

int main()

{

Stack s1;

s1.add("Getmanoid");

s1.add("a");

s1.add("i");

s1.add("j");

s1.add("m");

s1.add("n");

s1.add("s");

s1.add("st");

s1.show\_stack();

std::cout << s1.search("a") << std::endl;

std::cout << s1.search("j") << std::endl;

std::cout << s1.search("n") << std::endl;

std::cout << s1.search("s") << std::endl;

s1.del();

s1.show\_stack();

Queue q1;

q1.push("Getmanoid");

q1.push("a");

q1.push("i");

q1.push("j");

q1.push("m");

q1.push("n");

q1.push("s");

q1.push("st");

q1.show();

std::cout << "find: " << q1.find("a")->data << std::endl;

std::cout << "find: " << q1.find("j")->data << std::endl;

std::cout << "find: " << q1.find("n")->data << std::endl;

std::cout << "find: " << q1.find("s")->data << std::endl;

q1.pop();

q1.show();

List l1;

l1.add("Getmanoid");

l1.add("a");

l1.add("i");

l1.add("j");

l1.add("m");

l1.add("n");

l1.add("s");

l1.add("st");

l1.show\_list();

std::cout << l1.search("a") << std::endl;

std::cout << l1.search("j") << std::endl;

std::cout << l1.search("n") << std::endl;

std::cout << l1.search("s") << std::endl;

l1.del("a");

l1.show\_list();

return 0;

}

**Розрахунок показників обчислювальної складності**

Будемо вважати, що на поточний момент у структурах даних міститься n елементів.

**Двозв’язний впорядкований список**

Пошук:

while (iterator != nullptr)

{

if (iterator->field == value)

break;

iterator = iterator->next;

}

1) У найкращому випадку цикл виконається 1 раз і має 1 виконання, це стається за умови знаходження шуканого елементу у голові списку.

2) У середньому випадку буде n/2+1 виконань, це стається за умови знаходження шуканого елементу у середині списку.

3) У найгіршому випадку буде n виконань, це стається за умови, якщо не вдалося знайти шуканий елемент при проходженні по всьому списку.

Додавання елементу:

while (iterator->next != nullptr)

{

if (iterator->next->field > value)

break;

iterator = iterator->next;

}

1) У найкращому випадку цикл не виконається оскільки він знаходиться у альтернативній гілці оператору розгалуження, що має істинну умову, це стається за умови додавання елементу, найменшого елементу у список.

2) У середньому випадку буде n/2-1 виконань, це стається за умови додавання елементу у середину списку.

3) У найгіршому випадку буде n виконань, це стається за умови додавання елементу у кінець списку.

Видалення елементу:

while (iterator != nullptr)

{

if (iterator->field == value)

break;

iterator = iterator->next;

}

1) У найкращому випадку цикл виконається 1 раз і має 1 виконання, це стається за умови знаходження видаляємого елементу у голові списку.

2) У середньому випадку буде n/2+1 виконань, це стається за умови знаходження видаляємого елементу у середині списку.

3) У найгіршому випадку буде n виконань, це стається за умови, якщо не вдалося знайти видаляємого елемент при проходженні по всьому списку.

**Підрахунок кількості виконаних операторів при обробці двозв’язного впорядкованого списку**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції та оператори | Кількість виконаних операторів | | |
| Найкращий  випадок | Середній  випадок | Найгірший  випадок |
| Пошук: | | | |
| Node\* iterator = head;  while (iterator != nullptr)  {  if (iterator->field == value)  break;  iterator = iterator->next;  }  return iterator; | 1  1  1  1  0  1 | 1  n/2+1  n/2  1  n/2-1  1 | 1  n  n-1  0  n-1  1 |
| **Разом** | 5 | 3\*n/2+3 | 3\*n |
| Додавання елементу: | | | |
| Node\* iterator = head;  if (iterator==nullptr || iterator->field > value)  return nullptr;  while (iterator->next != nullptr)  {  if (iterator->next->field > value)  break;  iterator = iterator->next;  }  return iterator;  Node\* tmp = new Node;  tmp->field = value;  if (node == nullptr || head==nullptr)  {  tmp->next = head;  if (tmp->next != nullptr)  tmp->next->prev = tmp;  head = tmp;  return;  }    tmp->next = node->next;  tmp->prev = node;  node->next = tmp; | 1  1  1  0  0  0  0  0  1  1  1  1  1  1  1  1  0  0  0 | 1  1  0  n/2-1  n/2-2  1  n/2-3  1  1  1  1  0  0  0  0  0  1  1  1 | 1  1  0  n  n-1  1  n-2  1  1  1  1  0  0  0  0  0  1  1  1 |
| **Разом** | 11 | 3\*n/2+4 | 3\*n+7 |
| Видалення елементу: | | | |
| Node\* tmp = search(value);  if (tmp == nullptr)  return;  if (tmp == head)  head = head->next;  if(tmp->next!=nullptr)  tmp->next->prev = tmp->prev;  if (tmp->prev != nullptr)  tmp->prev->next = tmp->next;  delete tmp; | 5  1  0  1  1  1  0  1  0  1 | 3\*n/2  1  0  1  0  1  0  1  0  1 | 3\*n  1  1  0  0  0  0  0  0  0 |
| **Разом** | 11 | 3\*n/2+5 | 3\*n+2 |

**Черга зі зв’язним представленням**

Пошук:

while (temp != nullptr)

{

if (temp->data == val)

return temp;

else

temp = temp->next;

}

1) У найкращому випадку цикл виконається 1 раз і має 1 виконання, це стається за умови знаходження шуканого елементу у голові черги.

2) У середньому випадку буде n/2+1 виконань, це стається за умови знаходження шуканого елементу у середині черги.

3) У найгіршому випадку буде n виконань, це стається за умови, якщо не вдалося знайти шуканий елемент при проходженні по всій черзі.

Додавання елементу:

Node\* temp = new Node;

temp->data = val;

if (Head != nullptr)

{

Tail->next = temp;

Tail = temp;

}

else

{

Head = Tail = temp;

}

У даному випадку не існує найкращого середнього та найгіршого випадку. Додавання завжди виконується у кінець черги.

Видалення елементу:

int data = 0;

if (Head != nullptr)

{

Node\* temp = Head;

data = Head->data;

Head = Head->next;

delete temp;

}

else

{

throw "Try to pop from an empty queue";

}

return data;

У кращому випадку буде 3 виконання, це стається за умови, якщо черга порожня. У даному випадку не існує середнього та найгіршого випадку. Видалення завжди виконується з початку черги.

**Підрахунок кількості виконаних операторів при обробці черги зі зв’язним представленням**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції та оператори | Кількість виконаних операторів | | |
| Найкращий  випадок | Середній  випадок | Найгірший  випадок |
| Пошук: | | | |
| Node\* temp = Head;  while (temp != nullptr)  {  if (temp->data == val)  return temp;  else  temp = temp->next;  }  return nullptr; | 1  1  1  1  0  0  0 | 1  n/2+1  n/2  1  n/2-1  n/2-1  0 | 1  n  n-1  0  n-2  n-2  1 |
| **Разом** | 4 | 2\*n+1 | 4\*n-3 |
| Додавання елементу: | | | |
| Node\* temp = new Node;  temp->data = val;  if (Head != nullptr)  {  Tail->next = temp;  Tail = temp;  }  else  {  Head = Tail = temp;  } | 1  1  1  1  1  0  0 | 1  1  1  1  1  0  0 | 1  1  1  1  1  0  0 |
| **Разом** | 5 | 5 | 5 |
| Видалення елементу: | | | |
| int data = 0;  if (Head != nullptr)  {  Node\* temp = Head;  data = Head->data;  Head = Head->next;  delete temp;  }  else  {  throw "Try to pop from an empty queue";}  return data; | 1  1  0  0  0  0  1  1  1 | 1  1  1  1  1  1  0  0  1 | 1  1  1  1  1  1  0  0  1 |
| **Разом** | 5 | 7 | 7 |

**Стек зі зв’язним представленням**

Пошук:

while (iterator != nullptr)

{

if (iterator->item == value)

break;

iterator = iterator->next;

}

1) У найкращому випадку цикл виконається 1 раз і має 1 виконання, це стається за умови знаходження шуканого елементу у початку стеку.

2) У середньому випадку буде n/2+1 виконань, це стається за умови знаходження шуканого елементу у середині стеку.

3) У найгіршому випадку буде n виконань, це стається за умови, якщо не вдалося знайти шуканий елемент при проходженні по всьому стеку.

Додавання елементу:

Node\* p=new Node;

p->item = value;

p->next = top; // p указывает на 1-й элемент

top = p;

У даному випадку не існує найкращого середнього та найгіршого випадку. Додавання завжди виконується у початок стеку.

Видалення елементу:

if (top == nullptr)

return;

Node\* p2;

int item;

item = top->item;

p2 = top;

top = top->next;

delete p2;

У кращому випадку буде 2 виконання, це стається за умови, якщо стек порожній. У даному випадку не існує середнього та найгіршого випадку. Видалення завжди виконується з початку стеку.

**Підрахунок кількості виконаних операторів при обробці стеку зі зв’язним представленням**

|  |  |  |  |
| --- | --- | --- | --- |
| Операції та оператори | Кількість виконаних операторів | | |
| Найкращий  випадок | Середній  випадок | Найгірший  випадок |
| Пошук: | | | |
| Node\* iterator = top;  while (iterator != nullptr)  {  if (iterator->item == value)  break;  iterator = iterator->next;  }  return iterator!=nullptr; | 1  1  1  1  0  1 | 1  n/2+1  n/2  1  n/2-1  1 | 1  n  n-1  0  n-1  1 |
| **Разом** | 5 | 3\*n/2+3 | 3\*n |
| Додавання елементу: | | | |
| Node\* p=new Node;  p->item = value;  p->next = top;  top = p; | 1  1  1  1 | 1  1  1  1 | 1  1  1  1 |
| **Разом** | 4 | 4 | 4 |
| Видалення елементу: | | | |
| if (top == nullptr)  return;  Node\* p2;  int item;  item = top->item;  p2 = top;  top = top->next;  delete p2; | 1  1  0  0  0  0  0  0 | 1  0  1  1  1  1  1  1 | 1  0  1  1  1  1  1  1 |
| **Разом** | 2 | 7 | 7 |

**Зразок програми для аналізу та підрахунку кількості використань операторів обробки структур даних**

|  |  |  |  |
| --- | --- | --- | --- |
| Текст програми | Кількість використань | | |
| Операція  додавання | Операція  пошуку | Операція видалення |
| Sub Getmanoid()  Dim a() As Integer, i As Integer, j As Integer, m As Integer, n As Integer, s As Integer, st As String  Randomize  m = Val(InputBox("Введите число строк матрицы", , 3))  n = Val(InputBox("Введите число столбцов матрицы", , 15))  Sheets(1).UsedRange.ClearContents  ReDim a(1 To m, 1 To n)  For i = 1 To m  For j = 1 To n  a(i, j) = Int(Rnd \* 10)  Cells(6 + i, 1 + j).Value = a(i, j)  Next j  Next i  For j = 1 To n  i = 2  'For i = 2 To m  Do While a(i - 1, j) > a(i, j)  i = i + 1  If i > m Then  s = s + 1  st = st & j & "; "  Exit Do  End If  Loop  'Next  Next  MsgBox ("Количество столбцов, элементы которых упорядочены по убыванию = " & s & IIf(st = "", "", vbCrLf & "Их номера: " & st))  End Sub | 1  4  3 | 1  1  3  2  2  3  5  1  1  2  1  2  6  2  2  2  3  1  2 | 1  1  1  2  1  1  1 |
| **Разом** | 8 | 42 | 8 |

**Розрахунки**

Під час обробки наведеного тексту операція додавання та видалення у структури даних буде виконана 8 разів, а операція пошуку – 42. Тоді розрахуємо ймовірність використання операцій:

Pдодавання = 8/(8+42+8) = 0,14

Pпошуку = 42/(8+42+8) = 0,72

Pвидалення = 8/(8+42+8) = 0,14

Після операцій обробки кількість елементів (n) у структурах даних, які містять лексеми розібраної програми, буде дорівнювати 8. Використаємо отримані значення та визначимо обчислювальну складність операцій зі структурами даних:

**Двозв’язний впорядкований список:**

Sнайкращий = 0,14\*11+0,72\*5 + 0,14\*11 = 1,54 + 3,6 + 1,54= 6,68

Sсередній = 0,14\*(3\*n/2+4)+ 0,72\*(3\*n/2+3) + 0,14\*(3\*n/2+5) = =0,14\*(3\*8/2+4)+ 0,72\*(3\*8/2+3) + 0,14\*(3\*8/2+5) = 2,24 + 10,8 + 2,38 = 15,42

Sнайгірший = 0,14\*(3\*n+7)+ 0,72\*(3\*n) + 0,14\*(3\*n+2) =

=0,14\*(3\*8+7)+ 0,72\*(3\*8) + 0,14\*(3\*8+2) = 4,34 + 17,28 + 3,64 = 25,26

**Черга зі зв’язним представленням:**

Sнайкращий = 0,14\*5+0,72\*4 + 0,14\*5 = 0,7 + 2,88 + 0,7 = 4,28

Sсередній = 0,14\*5+0,72\*(2\*n+1) + 0,14\*7 = 0,14\*5+0,72\*(2\*8+1) + 0,14\*7 =

= 0,7 + 12,24 + 0,98 = 13,92

Sнайгірший = 0,14\*5+0,72\*(4\*n-3) + 0,14\*7 = 0,14\*5+0,72\*(4\*8-3) + 0,14\*7 =

= 0,7 + 20,88 + 0,98 = 22,56

**Стек зі зв’язним представленням:**

Sнайкращий = 0,14\*4+0,72\*5 + 0,14\*2= 0,56 + 3,6 + 0,56 = 4,72

Sсередній = 0,14\*4+0,72\*(3\*n/2+3) + 0,14\*7 =

= 0,14\*4+0,72\*(3\*8/2+3) + 0,14\*7 = 0,56 + 10,8 + 0,98 = 12,34

Sнайгірший = 0,14\*4+0,72\*(3\*n) + 0,14\*7 = 0,14\*4+0,72\*(3\*8) + 0,14\*7 =

= 0,56 + 17,28 + 0,98= 18,82

**Результати розрахунку обчислювальної складності**

|  |  |  |  |
| --- | --- | --- | --- |
| Структура даних | Обчислювальна складність | | |
| Найкращий  випадок | Середній  випадок | Найгірший  випадок |
| Двозв’язний впорядкований список | 6,68 | 15,42 | 25,26 |
| Черга зі зв’язним представленням | 4,28 | 13,92 | 22,56 |
| Стек зі зв’язним представленням | 4,72 | 12,34 | 18,82 |

**Висновок**: виконані дослідження дають змогу обрати кращу структуру за критерієм обчислювальної складності.

У даному випадку найкращою структурою для представлення таблиці ідентифікаторів є стек зі зв’язним представленням. Він має найменші показники обчислювальної складності у середньому та найгіршому випадках.

Черга зі зв’язним представленням має менші показники обчислювальної складності у середньому та найгіршому випадках, ніж двозв’язний впорядкований список, і менші показники обчислювальної складності найкращому випадку, ніж стек зі зв’язним представленням. Тому чергу зі зв’язним представленням краще використати для представлення таблиці зарезервованих слів, ніж двозв’язний впорядкований список, але гірше, ніж стек зі зв’язним представленням.

Двозв’язний впорядкований список має найбільші показники обчислювальної складності у всіх випадках, тому двозв’язний впорядкований список є найгіршою структурою для представлення таблиці зарезервованих слів.