МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №3  
з дисципліни «Якість програмного забезпечення та тестування»**

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

Виконав: студент гр. ПЗ2012

Білоус Олег Вікторович

Перевірив: доцент КІТ

Куроп’ятник О. С.

Дніпро, 2023

Лабораторна робота №3

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

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

1. **Постановка задачі**

Варіант 2

1) виконати підготовку до роботи:

- ознайомитися з теоретичними відомостями;

- підготувати зразки даних (див. пояснення у пункті «Завдання»),

придатних для подальшого аналізу;

- вивчити базові методи визначення показників обчислювальної

складності алгоритмів;

2) виконати проектування альтернативних структур даних та операцій над

ними на абстрактному та логічному рівнях;

3) виконати проектування структур даних на рівні представлення: розробити програму, що формує альтернативні структури даних та забезпечує необхідні операції над ними;

4) розрахувати показники обчислювальної складності операцій обробки

структурованих даних;

5) обґрунтований вибір структур даних за допомогою комбінаторно-ймовірнісних методів;

6) оцінити якість виконання практичної частини роботи з представленням

результатів у висновках.

1. **Структури даних**

**Двозв’язний впорядкований список**

Операції:

* Додавання: У позицію, що визначається порядком.
* Видалення: За ключем з відповідної позиції.
* Пошук: Шляхом послідовного перебору елементів

**Черга зі зв’язним представленням**

Операції:

* Додавання: У кінець.
* Видалення: З початку
* Пошук: Шляхом послідовного перебору елементів. Якщо елемент не рівний шуканому, він переставляється у кінець черги.

**Стек зі зв’язним представленням**

Операції:

* Додавання: У кінець.
* Видалення: З кінця.
* Пошук: Шляхом послідовного перебору елементів. Якщо елемент не рівний шуканому, він вивантажується в допоміжну структуру. Після того, як елемент було знайдено або перебирання закінчилося, елементи допоміжної структури повертаються у стек в початковому порядку.

*Схематичні зображення*

Двозв’язний список

Черга

Стек

1. **Критерії відбору мови програмування**

Для реалізації цих структур була обрана мова C++. Основними плюсами c++ для роботи з даними структурами є ООП складова, що дозволяє розробляти ці структури у вигляді класів, а також тонка робота із комірками пам’яті, що ідеально підходить для реалізаціїї списків. А в нашому випадку основою для черги та стеку є список.

1. **Текст програми**

**Двозв’язний впорядкований список**

struct Node {

int data;

Node\* prev;

Node\* next;

Node(int val) : data(val), prev(nullptr), next(nullptr) {}

};

class OrderedList {

private:

Node\* head;

Node\* tail;

public:

OrderedList() : head(nullptr), tail(nullptr) {}

// Додавання елемента

void insert(int val) {

Node\* newNode = new Node(val);

if (head == nullptr) {

head = tail = newNode;

} else if (head->data >= val) {

newNode->next = head;

head->prev = newNode;

head = newNode;

} else if (tail->data <= val) {

newNode->prev = tail;

tail->next = newNode;

tail = newNode;

} else {

Node\* temp = head->next;

while (temp->data < val) {

temp = temp->next;

}

newNode->prev = temp->prev;

newNode->next = temp;

temp->prev->next = newNode;

temp->prev = newNode;

}

}

// Видалення елемента

void remove(int val) {

if (head == nullptr) {

std::cout << "Список порожній" << std::endl;

return;

}

if (head->data == val) {

Node\* temp = head;

head = head->next;

if (head != nullptr) {

head->prev = nullptr;

} else {

tail = nullptr;

}

delete temp;

} else if (tail->data == val) {

Node\* temp = tail;

tail = tail->prev;

tail->next = nullptr;

delete temp;

} else {

Node\* temp = head->next;

while (temp != nullptr && temp->data != val) {

temp = temp->next;

}

if (temp != nullptr) {

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

} else {

std::cout << "Елемент не знайдено" << std::endl;

}

}

}

// Пошук за значенням

bool search(int val) {

Node\* temp = head;

while (temp != nullptr) {

if (temp->data == val) {

return true;

}

temp = temp->next;

}

return false;

}

};

**Черга зі зв’язним представленням**

struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() : front(nullptr), rear(nullptr) {}

// Додавання елемента в кінець черги

void enqueue(int val) {

Node\* newNode = new Node(val);

if (front == nullptr) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

// Видалення елемента з початку черги

void dequeue() {

if (front == nullptr) {

std::cout << "Черга порожня" << std::endl;

return;

}

Node\* temp = front;

front = front->next;

delete temp;

}

// Пошук елемента та перестановка в кінець черги

void searchAndMoveToEnd(int val) {

if (front == nullptr || front->next == nullptr) {

return;

}

Node\* prev = nullptr;

Node\* current = front;

while (current != nullptr && current->data != val) {

prev = current;

current = current->next;

}

if (current != nullptr) {

if (current == front) {

front = front->next;

rear->next = current;

rear = current;

rear->next = nullptr;

} else if (current != rear) {

prev->next = current->next;

rear->next = current;

rear = current;

rear->next = nullptr;

}

}

}

};

**Стек зі зв’язним представленням**

struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

class Stack {

private:

Node\* top;

public:

Stack() : top(nullptr) {}

// Додавання елемента в стек

void push(int val) {

Node\* newNode = new Node(val);

newNode->next = top;

top = newNode;

}

// Видалення елемента зі стеку

void pop() {

if (top == nullptr) {

std::cout << "Стек порожній" << std::endl;

return;

}

Node\* temp = top;

top = top->next;

delete temp;

}

void searchAndUnload(int val) {

Node\* current = top;

Node\* prev = nullptr;

while (current != nullptr) {

if (current->data == val) {

// Знайдено шуканий елемент

break;

}

// Якщо елемент не є шуканим, вивантажуємо його зі стеку

if (prev == nullptr) {

top = current->next;

delete current;

current = top;

} else {

prev->next = current->next;

delete current;

current = prev->next;

}

}

// Повертаємо елементи назад у стек

while (current != nullptr && current->next != nullptr) {

current = current->next;

}

if (current != nullptr) {

current->next = top;

top = top->next;

current->next->next = nullptr;

}

}

// Виведення стеку для перевірки результату

void displayStack() {

Node\* temp = top;

std::cout << "Елементи у стеці: ";

while (temp != nullptr) {

std::cout << temp->data << " ";

temp = temp->next;

}

std::cout << std::endl;

}

};

1. **Р****озрахунок показників обчислювальної складності**

**Двозв’язний впорядкований список**

Допустимо, що у списку наразі знаходиться n елементів

1. Додавання

*Найкращий випадок:*

Опис: коли елемент додається в початок або кінець списку.

*Середній випадок:*

Опис: коли елемент потрібно вставити у середину списку, не на початок або кінець.

*Найгірший випадок:*

Опис: коли елемент додається у відсортований список після N ітерацій пошуку місця для вставки.

1. Пошук

*Найкращий випадок:*

Опис: коли шуканий елемент знаходиться в початку списку.

*Середній випадок:*

Опис: коли шуканий елемент знаходиться в середині списку.

*Найгірший випадок:*

Опис: коли шуканий елемент відсутній у списку або коли його розташування потрібно знайти в останньому елементі.

1. Видалення

*Найкращий випадок:*

Опис: коли елемент видаляється з початку або кінця списку.

*Середній випадок:*

Опис: коли елемент знаходиться у середині списку.

*Найгірший випадок:*

Опис: оли потрібно пройти всі N елементів для видалення останнього елемента.

**Черга зі зв’язним представленням**

Допустимо, що у черзі наразі знаходиться n елементів

1. Додавання

*Найкращий випадок:*

Опис: коли черга порожня і елемент додається безпосередньо в кінець або у черзі є один елемент і виконується додавання.

*Середній випадок:*

Опис: коли черга має деяку кількість елементів і елемент додається в кінець.

*Найгірший випадок:*

Опис: коли черга має деяку кількість елементів і елемент додається в кінець.

1. Пошук

*Найкращий випадок:*

Опис: коли елемент, який потрібно знайти, перебуває в кінці.

*Середній випадок:*

Опис: коли елемент, який потрібно знайти, перебуває у середині черги.

*Найгірший випадок:*

Опис: коли елемент, який потрібно знайти перебуває у початку черги.

1. Видалення

*Найкращий випадок:*

Опис: коли черга порожня або видаляється перший елемент.

*Середній випадок:*

Опис: коли черга має деяку кількість елементів і видаляється перший елемент.

*Найгірший випадок:*

Опис: коли черга має деяку кількість елементів і видаляється перший елемент.

**Стек зі зв’язним представленням**

Допустимо, що у стеку наразі знаходиться n елементів

1. Додавання

*Найкращий випадок:*

Опис: коли елемент додається на вершину порожнього стеку

*Середній випадок:*

Опис: коли елемент додається на вершину стеку з кількома елементами.

*Найгірший випадок:*

Опис: коли елемент додається на вершину стеку з кількома елементами.

1. Пошук

*Найкращий випадок:*

Опис: коли шуканий елемент знаходиться на вершині.

*Середній випадок:*

Опис: коли елемент потрібно знайти у середній частині стеку.

*Найгірший випадок:*

Опис: коли шуканий елемент знаходиться на дні стеку або його немає у стеці.

1. Видалення

*Найкращий випадок:*

Опис: коли видаляється вершина порожнього стеку.

*Середній випадок:*

Опис: коли видаляється вершина стеку з кількома елементами.

*Найгірший випадок:*

Опис: коли видаляється вершина стеку з кількома елементами.

Таблиця 1

Підрахунок кількості виконаних операторів при обробці двозв’язного впорядкованого списку

|  |  |  |  |
| --- | --- | --- | --- |
| Операції та оператори | Кількість виконаних операторів | | |
| Найкращий випадок | Середній випадок | Найгірший випадок |
| Додавання | | | |
| Node\* newNode = new Node(val);  if (head == nullptr) {  head = tail = newNode;  } else if (head->data >= val) {  newNode->next = head;  head->prev = newNode;  head = newNode;  } else if (tail->data <= val) {  newNode->prev = tail;  tail->next = newNode;  tail = newNode;  } else {  Node\* temp = head->next;  while (temp->data < val) {  temp = temp->next;  }  newNode->prev = temp->prev;  newNode->next = temp;  temp->prev->next = newNode;  temp->prev = newNode;  } | 1  1  0  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0 | 1  1  0  1  0  0  0  1  0  0  0  1  1  n/2 + 1  n/2  1  1  1  1 | 1  1  0  1  0  0  0  1  0  0  0  1  1  n + 1  n  1  1  1  1 |
| **Разом** | 6 | 11 + n | 11 + 2n |
| Видалення | | | |
| if (head == nullptr) {  std::cout << "Список порожній" << std::endl;  return;  }  if (head->data == val) {  Node\* temp = head;  head = head->next;  if (head != nullptr) {  head->prev = nullptr;  } else {  tail = nullptr;  }  delete temp;  } else if (tail->data == val) {  Node\* temp = tail;  tail = tail->prev;  tail->next = nullptr;  delete temp;  } else {  Node\* temp = head->next;  while (temp != nullptr && temp->data != val) {  temp = temp->next;  }  if (temp != nullptr) {  temp->prev->next = temp->next;  temp->next->prev = temp->prev;  delete temp;  } else {  std::cout << "Елемент не знайдено" << std::endl;  }  } | 1  0  0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 | 1  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  1  1  n/2 + 1  n/2  1  1  1  1  0  0 | 1  0  0  1  0  0  0  0  0  0  0  1  0  0  0  0  1  1  n + 1  n  1  1  1  1  0  0 |
| **Разом** | 6 | 10 + n | 10 + 2n |
| Пошук | | | |
| Node\* temp = head;  while (temp != nullptr) {  if (temp->data == val) {  return true;  }  temp = temp->next;  }  return false;  } | 1  1  1  1  0  0 | 1  n/2  n/2  1  n/2 - 1  0 | 1  n  n  1  n -1  0 |
| **Разом** | 4 | 1 + 1.5n | 1 + 3n |

Таблиця 2

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

|  |  |  |  |
| --- | --- | --- | --- |
| Операції та оператори | Кількість виконаних операторів | | |
| Найкращий випадок | Середній випадок | Найгірший випадок |
| Додавання | | | |
| Node\* newNode = new Node(val);  if (front == nullptr) {  front = rear = newNode;  } else {  rear->next = newNode;  rear = newNode;  } | 1  1  1  0  0  0 | 1  1  0  1  1  1 | 1  1  0  1  1  1 |
| **Разом** | 3 | 5 | 5 |
| Видалення | | | |
| if (front == nullptr) {  std::cout << "Черга порожня" << std::endl;  return;  }  Node\* temp = front;  front = front->next;  delete temp; | 1  1  1  0  0  0 | 1  0  0  1  1  1 | 1  0  0  1  1  1 |
| **Разом** | 3 | 4 | 4 |
| Пошук | | | |
| if (front == nullptr || front->next == nullptr) {  return;  }  Node\* prev = nullptr;  Node\* current = front;  while (current != nullptr && current->data != val) {  prev = current;  current = current->next;  }  if (current != nullptr) {  if (current == front) {  front = front->next;  rear->next = current;  rear = current;  rear->next = nullptr;  } else if (current != rear) {  prev->next = current->next;  rear->next = current;  rear = current;  rear->next = nullptr;  }  } | 1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 | 1  0  1  1  n/2 + 1  n/2  n/2  1  1  0  0  0  0  1  1  1  1  1 | 1  0  1  1  n + 1  n  n  1  1  0  0  0  0  1  1  1  1  1 |
| **Разом** | 2 | 11 + 1.5n | 11 + 3n |

Таблиця 3

Підрахунок кількості виконаних операторів при обробці стека зі зв’язним представленням

|  |  |  |  |
| --- | --- | --- | --- |
| Операції та оператори | Кількість виконаних операторів | | |
| Найкращий випадок | Середній випадок | Найгірший випадок |
| Додавання | | | |
|  |  |  |  |
| Node\* newNode = new Node(val);  newNode->next = top;  top = newNode; | 1  1  1 | 1  1  1 | 1  1  1 |
| **Разом** | 3 | 3 | 3 |
| Видалення | | | |
| if (top == nullptr) {  std::cout << "Стек порожній" << std::endl;  return;  }  Node\* temp = top;  top = top->next;  delete temp; | 1  1  1  0  0  0 | 1  0  0  1  1  1 | 1  0  0  1  1  1 |
| **Разом** | 3 | 4 | 4 |
| Пошук | | | |
| Node\* current = top;  Node\* prev = nullptr;  while (current != nullptr) {  if (current->data == val) {  // Знайдено шуканий елемент  break;  }  // Якщо елемент не є шуканим, вивантажуємо його зі стеку  if (prev == nullptr) {  top = current->next;  delete current;  current = top;  } else {  prev->next = current->next;  delete current;  current = prev->next;  }  }  // Повертаємо елементи назад у стек  while (current != nullptr && current->next != nullptr) {  current = current->next;  }  if (current != nullptr) {  current->next = top;  top = top->next;  current->next->next = nullptr;  } | 1  1  1  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0 | 1  1  n/2  n/2  1  n/2 -1  1  1  1  n/2 -1  n/2 -1  n/2 -1  n/2 -1  n/2 + 1  n/2  1  0  0  0 | 1  1  n  n  1  n -1  1  1  1  n -1  n -1  n-1  n-1  n+1  n  1  0  0  0 |
| **Разом** | 5 | 3 + 4.5n | 3 + 9n |

Таблиця 4

Зразок програми для аналізу та підрахунку кількості використань операторів

обробки структур даних

|  |  |  |
| --- | --- | --- |
| Текст програми | Кількість використань | |
| Операція  додавання | Операція  пошуку |
| struct Node {  int data;  Node\* next;  Node(int val) : data(val), next(nullptr) {}  };  class Queue {  private:  Node\* front;  Node\* rear;  public:  Queue() : front(nullptr), rear(nullptr) {}  // Додавання елемента в кінець черги  void enqueue(int val) {  Node\* newNode = new Node(val);  if (front == nullptr) {  front = rear = newNode;  } else {  rear->next = newNode;  rear = newNode;  }  }  // Видалення елемента з початку черги  void dequeue() {  if (front == nullptr) {  std::cout << "Черга порожня" << std::endl;  return;  }  Node\* temp = front;  front = front->next;  delete temp;  }  // Пошук елемента та перестановка в кінець черги  void searchAndMoveToEnd(int val) {  if (front == nullptr || front->next == nullptr) {  return;  }  Node\* prev = nullptr;  Node\* current = front;  while (current != nullptr && current->data != val) {  prev = current;  current = current->next;  }  if (current != nullptr) {  if (current == front) {  front = front->next;  rear->next = current;  rear = current;  rear->next = nullptr;  } else if (current != rear) {  prev->next = current->next;  rear->next = current;  rear = current;  rear->next = nullptr;  }  }  }  }; | 1  1  1  1  1  1  1  1  1  1  1  1  1  1 | 1  1  2  1  1  1  2  2  1  2  1  1  2  1  1  2  3  1  1  2  2  1  1  1  1 |
| **Разом** | 14 | 35 |

Операцій видалення було стільки ж, скільки і додавання

Р+ = 14 / 63 = 0,22

P- = 14 / 63 = 0,22

P? = 53/83 = 0,56

Список

S+ = 6 \* 0.22 + 6 \* 0.22 + 4 \* 0.56 = 4.88

Sс = 0.22 \* (11 + n) + 0.22 \* (10 + n)+ 0.56 \* (1 + 1.5n) = 0.22 \* 25 + 0.22 \* 24 + 0.56 \* 22 = 23.1

S- = 0.22 \* (11 + 2n) + 0.22 \* (10 + 2n)+ 0.56 \* (1 + 3n) = 0.22 \* 39 + 0.22 \* 38 + 0.56 \* 43 = 41.02

Черга

S+ = 3 \* 0.22 + 3 \* 0.22 + 2 \* 0.56 = 2.44

Sс = 0.22 \* 5 + 0.22 \* 4+ 0.56 \* (11 + 1.5n) = 0.22 \* 5 + 0.22 \* 4 + 0.56 \* 32 = 19.9

S- = 0.22 \* 5 + 0.22 \* 4+ 0.56 \* (11 + 3n) = 0.22 \* 5 + 0.22 \* 4 + 0.56 \* 53 = 31.66

Стек

S+ = 3 \* 0.22 + 3 \* 0.22 + 5 \* 0.56 = 4.12

Sс = 0.22 \* 3 + 0.22 \* 4+ 0.56 \* (3 + 4.5n) = 0.22 \* 3 + 0.22 \* 4 + 0.56 \* 66 = 38.5

S- = 0.22 \* 3 + 0.22 \* 4+ 0.56 \* (3 + 9n) = 0.22 \* 3 + 0.22 \* 4 + 0.56 \* 129 = 73.78

Таблиця 5

Результати розрахунку обчислювальної складності

|  |  |  |  |
| --- | --- | --- | --- |
| Структура даних | Обчислювальна складність | | |
| Найкращий випадок | Середній випадок | Найгірший випадок |
| Впорядкований двозв’язний список | 4.88 | 23.1 | 41.02 |
| Черга зі зв’язним представленням | 2.44 | 19.9 | 31.66 |
| Стек зі зв’язним представленням | 4.12 | 38.5 | 73.78 |

Найкращою структурою для виконання поставленої задачі видалася черга зі зв’язним представленням. Вона показує кращий результат в усіх випадках.

Список на другому місці, він програє черзі у найкращому випадку у два рази, у середньому майже не відстає а у найгіршому досить великий розрив. А от стек дуже погано себе показує у найгіршому випадку, також і у середньому. Це робить його найгіршим вибором.

**Висновок**

У ході лабораторної роботи №3 було вивчено та застосовано комбінаторно-ймовірнісні методи для аналізу показників обчислювальної складності різних структур даних: двозв'язного впорядкованого списку, черги зі зв'язним представленням, та стека зі зв'язним представленням. Було проведено детальне проектування цих структур, розроблено програми для їх реалізації на мові C++, та оцінено їх обчислювальну складність за різними критеріями.

Аналіз показав, що кожна структура має свої переваги та недоліки в залежності від контексту використання. Наприклад, черга зі зв'язним представленням виявилася найбільш ефективною за всіма показниками обчислювальної складності, що робить її відмінним вибором для різних задач обробки даних. Водночас, стек зі зв'язним представленням показав себе менш ефективним, особливо у середньому та найгіршому випадках.

Такий аналіз дозволяє більш свідомо підходити до вибору структур даних під конкретні задачі, враховуючи специфіку даних та операцій, які будуть з ними проводитися. Розуміння обчислювальної складності є ключовим для створення ефективних та оптимізованих програмних рішень.