Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

ЗВІТ

про виконання лабораторної роботи № 4

з дисципліни

“Дискретні структури”

Тема:Спискові структури.Однонаправлена черга та черга з пріоритетом.

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| [ст.вик. Москаленко Н. В.](http://rozklad.kpi.ua/Schedules/ViewSchedule.aspx?v=98f24a67-7c5b-4cd0-b696-d43e4af682ea) |  | студент 1-го курсу  гр. ІП-81 ФІОТ  Педоренко Микита Русланович |

Київ 2018

Зміст

[1 МЕТА РОБОТИ 3](#_Toc2507274)

[2 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc2507275)

[3 БЛОК СХЕМА АЛГОРИТМУ 5](#_Toc2507276)

[4 ТЕКСТ ПРОГРАМНОГО КОДУ 6](#_Toc2507277)

[5 РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ 7](#_Toc2507278)

[ВИСНОВКИ 8](#_Toc2507279)

[Додаток 9](#_Toc2507280)

1 МЕТА РОБОТИ

*Мета роботи* – опанувати теоритичні положення динамічної структури односторонньої черги та черги з пріоритетом. Реалізація черг на алгоритмічній мові програмування.

1. ПОСТАНОВКА ЗАДАЧІ

Реалізувати на алгоритмічній мові програмування АВЛ-дерево пошуку з операціями вставки, пошуку та видалення елементів, а також префіксним, постфіксним та інфіксним обходами.

**Завдання**:  
Знайти ширину піддерева k-ї вершини.

3 БЛОК СХЕМА АЛГОРИТМУ



4 ТЕКСТ ПРОГРАМНОГО КОДУ

using System;

namespace Lab2

{

class Program

{

static bool IsEquals(DynamicQueue<int> q1, DynamicQueue<int> q2)

{

if (q1 == null || q2 == null || (q1.Size() != q2.Size()))

return false;

bool flag = true;

for (int i = 0; i < q1.Size(); i++)

{

int value1, value2;

value1 = q1.PopBack();

value2 = q2.PopBack();

if (value1 != value2)

flag = false;

q1.PushFront(value1);

q2.PushFront(value2);

}

return flag;

}

static void Main(string[] args)

{

Console.Write("Enter number of queues(>=2): ");

int numberOfQueues = int.Parse(Console.ReadLine());

DynamicQueue<int> q1 = GenerateQueue(1);

string similars = "";

for (int i = 1; i < numberOfQueues; i++)

{

if (IsEquals(q1, GenerateQueue(i + 1)))

similars += ((i + 1) + " ");

}

Console.Clear();

if(similars.Length == 0)

{

Console.WriteLine("No similar queues");

}

else

{

Console.WriteLine("Indexes of similar queues: " + similars);

}

Console.ReadLine();

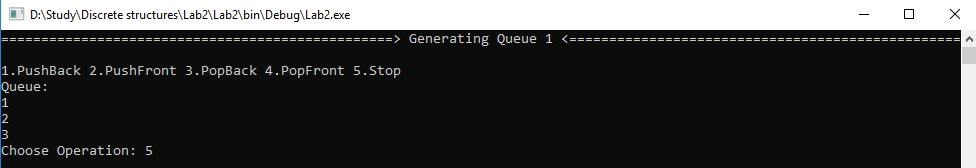
}

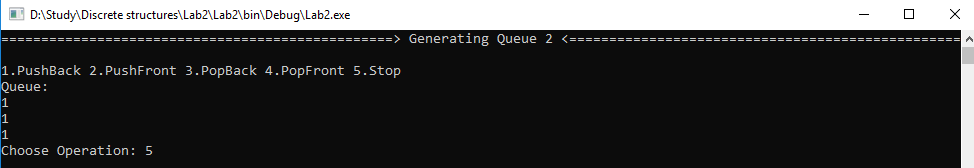
}

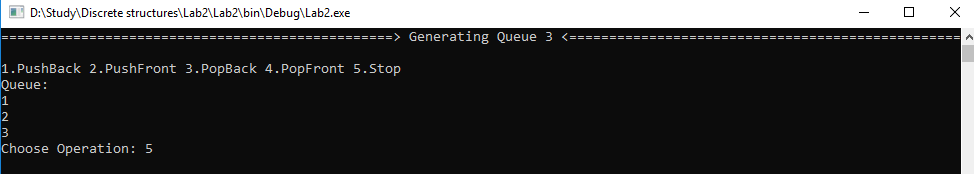
}

5 РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ











ВИСНОВКИ

Під час виконання даної лабораторної роботи я опанував теоретичні положення динамачної структури двостороння черга. Реалізував двосторонню чергу за допомогую динамічного масиву та спискової структури.

# Додаток А. Код програми.

DynamicQueue.cs:

using System;

namespace Lab2

{

class DynamicQueue<T>

{

private class Node

{

public T data;

public Node previous;

public Node next;

}

private Node back = null, front = null;

public void PushBack(T value)

{

Node temp = new Node();

temp.data = value;

temp.previous = null;

if(back == null)

{

back = front = temp;

temp.next = null;

}

else

{

temp.next = back;

back.previous = temp;

back = temp;

}

}

public void PushFront(T value)

{

Node temp = new Node();

temp.data = value;

temp.next = null;

if(front == null)

{

front = back = temp;

temp.previous = null;

}

else

{

temp.previous = front;

front.next = temp;

front = temp;

}

}

public T PopBack()

{

T value;

if(back == null)

throw new Exception("Queue is empty");

if(back.next == null)

{

value = back.data;

back = front = null;

return value;

}

value = back.data;

back = back.next;

back.previous = null;

return value;

}

public T PopFront()

{

if(front == null)

throw new Exception("Queue is empty");

T value;

if(front.previous == null)

{

value = front.data;

back = front = null;

return value;

}

value = front.data;

front = front.previous;

front.next = null;

return value;

}

public void Clear()

{

back = null;

front = null;

}

public void Print()

{

Node cursor = back;

while(cursor != null)

{

Console.WriteLine(cursor.data);

cursor = cursor.next;

}

}

public int Size()

{

int size = 0;

Node cursor = back;

while (cursor != null)

{

size++;

cursor = cursor.next;

}

return size;

}

}

}

StaticQueue.cs:

namespace Lab2

{

using System;

internal class StaticQueue<T>

{

private T[] data;

private readonly int maxSize;

private int size = 0, back, front;

public int Size

{

get { return size; }

}

public StaticQueue(int maxSize = 10)

{

this.maxSize = maxSize;

data = new T[maxSize];

back = front = maxSize - 1;

}

public void PushBack(T value)

{

if (size >= maxSize)

{

throw new OverflowException("Queue overflow");

}

data[back] = value;

back = (back + maxSize - 1) % maxSize;

size++;

}

public void PushFront(T value)

{

if (size >= maxSize)

{

throw new OverflowException("Queue overflow");

}

front = (front + 1) % maxSize;

data[front] = value;

size++;

}

public T PopBack()

{

if (size > 0)

{

back = (back + 1) % maxSize;

size--;

return data[back];

}

throw new Exception("Queue is empty");

}

public T PopFront()

{

if (size > 0)

{

T value = data[front];

front = (front + maxSize - 1) % maxSize;

size--;

return value;

}

throw new Exception("Queue is empty");

}

public void Print()

{

int i = back;

for (int k = 0; k < size; k++)

{

i = (i + 1) % maxSize;

Console.WriteLine(data[i]);

}

}

public bool IsEmpty()

{

if (size > 0) return false;

return true;

}

public void Clear()

{

back = front = maxSize - 1;

size = 0;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab2

{

class Program

{

static bool IsEquals(DynamicQueue<int> q1, DynamicQueue<int> q2)

{

if (q1 == null || q2 == null || (q1.Size() != q2.Size()))

return false;

bool flag = true;

for (int i = 0; i < q1.Size(); i++)

{

int value1, value2;

value1 = q1.PopBack();

value2 = q2.PopBack();

if (value1 != value2)

flag = false;

q1.PushFront(value1);

q2.PushFront(value2);

}

return flag;

}

static DynamicQueue<int> GenerateQueue(int number)

{

DynamicQueue<int> queue = new DynamicQueue<int>();

bool stop = false;

while (!stop)

{

Console.Clear();

Console.WriteLine("=================================================> Generating Queue {0} <=================================================", number);

Console.WriteLine("1.PushBack 2.PushFront 3.PopBack 4.PopFront 5.Stop");

Console.WriteLine("Queue:");

queue.Print();

Console.Write("Choose Operation: ");

switch (Console.ReadLine())

{

case "1":

{

Console.Write("PushBack: ");

queue.PushBack(int.Parse(Console.ReadLine()));

break;

};

case "2":

{

Console.Write("PushFront: ");

queue.PushFront(int.Parse(Console.ReadLine()));

break;

};

case "3":

{

queue.PopBack();

break;

}

case "4":

{

queue.PopFront();

break;

}

case "5":

{

stop = true;

break;

}

case "":

break;

default:

break;

}

}

return queue;

}

static void Main(string[] args)

{

Console.Write("Enter number of queues(>=2): ");

int numberOfQueues = int.Parse(Console.ReadLine());

DynamicQueue<int> q1 = GenerateQueue(1);

string similars = "";

for (int i = 1; i < numberOfQueues; i++)

{

if (IsEquals(q1, GenerateQueue(i + 1)))

similars += ((i + 1) + " ");

}

Console.Clear();

if(similars.Length == 0)

{

Console.WriteLine("No similar queues");

}

else

{

Console.WriteLine("Indexes of similar queues: " + similars);

}

Console.ReadLine();

}

}

}