# Chapter 16

## Gerti Gonxhi

1.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task1\_\_\_SumAndAveragesOfPositiveIntegers

{

class Program

{

static void Main(string[] args)

{

List<int> seq = new List<int>();

string str;

int sum = 0;

while ((str = Console.ReadLine()) != "")

{

int num = int.Parse(str);

seq.Add(num);

sum += num;

}

Console.WriteLine(sum);

Console.WriteLine((double)sum / seq.Count);

}

}

}

2.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task2\_\_\_StackDemo

{

class Program

{

static void Main(string[] args)

{

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

string[] numbers = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

Stack<int> stack = new Stack<int>();

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

{

stack.Push(int.Parse(numbers[i]));

}

while (stack.Count > 1)

{

Console.Write("{0} ",stack.Pop());

}

Console.WriteLine(stack.Pop());

}

}

}

3.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task3\_\_\_SortSeqOfIntegers

{

class Program

{

static void Main(string[] args)

{

List<int> seq = new List<int>();

string str;

while ((str = Console.ReadLine()) != "")

{

seq.Add(int.Parse(str));

}

int[] array = seq.ToArray();

Array.Sort(array);

seq = array.ToList<int>();

foreach (int number in seq)

{

Console.Write("{0} ", number);

}

Console.WriteLine();

}

}

}

4.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task4\_\_\_LongestSubseqOfEqualIntegers

{

class Program

{

static List<int> findLongestSeq(List<int> sequence)

{

List<int> currentSeq = new List<int>();

List<int> bestSeq = new List<int>();

bestSeq.Add(sequence[0]);

currentSeq.Add(sequence[0]);

for (int i = 1; i < sequence.Count; i++)

{

int current = sequence[i];

if (current == currentSeq[0])

{

currentSeq.Add(current);

}

else

{

if (currentSeq.Count > bestSeq.Count)

{

bestSeq = currentSeq;

}

currentSeq = new List<int>();

currentSeq.Add(current);

}

}

if (currentSeq.Count > bestSeq.Count)

{

bestSeq = currentSeq;

}

return bestSeq;

}

static void Main(string[] args)

{

string[] numbers = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

List<int> longestSeq = new List<int>();

for (int i = 0; i < numbers.Length; i++)

{

longestSeq.Add(int.Parse(numbers[i]));

}

longestSeq = findLongestSeq(longestSeq);

foreach (var item in longestSeq)

{

Console.Write("{0} ", item);

}

Console.WriteLine();

}

}

}

5.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task5\_\_\_RemovesNegativeNumbers

{

class Program

{

static void Main(string[] args)

{

string[] numbers = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

List<double> seq = new List<double>();

for (int i = 0; i < numbers.Length; i++)

{

seq.Add(double.Parse(numbers[i]));

}

List<double> positiveSeq = new List<double>();

foreach (var item in seq)

{

if (!(item < 0))

{

positiveSeq.Add(item);

}

}

foreach (var item in positiveSeq)

{

Console.Write("{0} ", item);

}

Console.WriteLine();

}

}

}

6.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task6\_\_\_RemoveIfAppearsOddNumberOfTimes

{

class Program

{

static void Main(string[] args)

{

List<int> seq = new List<int>();

string[] nums = Console.ReadLine().Split(new char[] { ',', ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (var item in nums)

{

seq.Add(int.Parse(item));

}

int[,] numbersTimes = new int[2, seq.Count];

int count = 0;

foreach (int num in seq)

{

bool didAppear = false;

for (int j = 0; j < count; j++)

{

if (num == numbersTimes[0, j])

{

numbersTimes[1, j] ^= 1;

didAppear = true;

}

}

if (!didAppear)

{

numbersTimes[0, count] = num;

numbersTimes[1, count++] ^= 1;

}

}

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

{

if (numbersTimes[1, i] == 1)

{

while (seq.IndexOf(numbersTimes[0, i]) != -1)

{

seq.Remove(numbersTimes[0, i]);

}

}

}

foreach (var item in seq)

{

Console.Write("{0} ", item);

}

}

}

}

7.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task7\_\_\_FindNumberOfTimesANumberAppears

{

class Program

{

static void Main(string[] args)

{

List<int> seq = new List<int>();

string[] nums = Console.ReadLine().Split(new char[] { ',', ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (var item in nums)

{

seq.Add(int.Parse(item));

}

int[] numberOfTimes = new int[1001];

foreach (int item in seq)

{

numberOfTimes[item]++;

}

for (int i = 0; i < numberOfTimes.Length; i++)

{

if (numberOfTimes[i] != 0)

{

Console.WriteLine("{0} -> {1} times", i, numberOfTimes[i]);

}

}

}

}

}

8.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task8\_\_\_Majorant

{

class Program

{

public class NumberOfTimes

{

private int number;

private int times;

public NumberOfTimes()

{

this.number = 0;

this.times = 0;

}

public NumberOfTimes(int num, int Times)

{

this.number = num;

this.times = Times;

}

public int Number

{

get { return this.number; }

set { this.number = value; }

}

public int Times

{

get { return this.times; }

set { this.times = value; }

}

}

static void Main(string[] args)

{

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

List<int> seq = new List<int>();

string[] nums = Console.ReadLine().Split(new char[] { ',', ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (var item in nums)

{

seq.Add(int.Parse(item));

}

List<NumberOfTimes> nt = new List<NumberOfTimes>();

foreach (var num in seq)

{

bool flag = true;

for (int i = 0; i < nt.Count; i++)

{

if (num == nt[i].Number)

{

nt[i].Times++;

flag = false;

}

}

if (flag)

{

nt.Add(new NumberOfTimes(num, 1));

}

}

foreach (var item in nt)

{

if (item.Times >= n / 2 + 1)

{

Console.WriteLine(item.Number);

break;

}

}

}

}

}

9.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task9\_\_\_SequenceOfOperations

{

class Program

{

static void Main(string[] args)

{

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

Queue<int> seq = new Queue<int>();

seq.Enqueue(n);

for (int i = 1; i < 51; i++)

{

int temp = seq.Peek();

if (i % 3 == 1)

{

seq.Enqueue(temp + 1);

}

else if (i % 3 == 2)

{

seq.Enqueue(temp \* 2 + 1);

}

else

{

seq.Enqueue(temp + 2);

Console.Write("{0} ", seq.Dequeue());

}

}

while (seq.Count != 0)

{

Console.Write("{0} ", seq.Dequeue());

}

Console.WriteLine();

}

}

}

10.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task10\_\_\_ShortestSeqOfOperations

{

class Program

{

static void FindMinSeq(int n, int m)

{

int count = 0;

if (n < 0)

{

int current = n;

while (current < 0)

{

count++;

if (m - current == 2)

{

current = m;

break;

}

if (m - current == 1)

{

current = m;

break;

}

current += 2;

}

n = current;

}

int temp = m;

if (n == 0 && m > 1)

{

count++;

n = 2;

}

if (m >= 0)

{

while (temp - n > n)

{

if (temp - n == 2)

{

count++;

temp = n;

break;

}

if (temp % 2 == 1)

{

count++;

temp = temp - 1;

}

if (temp == n)

{

break;

}

count++;

temp /= 2;

}

}

while (temp != n)

{

count++;

if (temp - n >= 2)

{

temp -= 2;

continue;

}

if (temp - n == 1)

{

temp -= 1;

}

}

Console.WriteLine("Count = {0}", count);

}

static void Main(string[] args)

{

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

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

FindMinSeq(n, m);

}

}

}

11.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task11\_\_\_DoubleLinkedList

{

public class DoubleLinkedList<T> : IEnumerable<T>

{

private DoubleLinkedListNode<T> head;

private DoubleLinkedListNode<T> tail;

private int count;

public DoubleLinkedList()

{

head = null;

tail = null;

count = 0;

}

public DoubleLinkedList(DoubleLinkedList<T> list)

{

this.tail.Next = list.head;

list.head.Previous = this.tail;

count += list.Count;

this.tail = list.tail;

}

//Add an element to the list

public void Add(T element)

{

if (count == 0)

{

this.head = new DoubleLinkedListNode<T>(element);

this.tail = this.head;

}

else

{

this.tail = new DoubleLinkedListNode<T>(element, this.tail);

}

count++;

}

//Remove an element if exist, if not returns false;

public bool Remove(T element)

{

int index = Find(element);

if (index == -1)

{

return false;

}

DoubleLinkedListNode<T> node = ReturnsNodeByIndex(index);

node.RemoveEl();

if (index == 0)

{

this.head = node.Next;

}

if (index == count - 1)

{

this.tail = node.Previous;

}

count--;

return true;

}

//Returns the index of an element, returns -1 if not found

public int Find(T element)

{

DoubleLinkedListNode<T> forwardNode = this.head;

int forward = 0;

while (forwardNode != null)

{

if (forwardNode.Element.Equals(element))

{

return forward;

}

forward++;

forwardNode = forwardNode.Next;

}

return -1;

}

private DoubleLinkedListNode<T> ReturnsNodeByIndex(int index)

{

if (index > count - index)

{

int i = count - 1;

DoubleLinkedListNode<T> node = this.tail;

while (true)

{

if (i == index)

{

return node;

}

node = node.Previous;

i--;

}

}

else

{

int i = 0;

DoubleLinkedListNode<T> node = this.head;

while (true)

{

if (i == index)

{

return node;

}

node = node.Next;

i++;

}

}

}

public void Insert(T element, int index)

{

if (index < 0 || index > count)

{

throw new ArgumentOutOfRangeException("The index is out of bounds");

}

if (index == 0)

{

DoubleLinkedListNode<T> node = new DoubleLinkedListNode<T>(element);

node.Next = this.head;

this.head = node;

count++;

}

else if (index == count)

{

this.tail = new DoubleLinkedListNode<T>(element, this.tail);

count++;

}

else

{

DoubleLinkedListNode<T> node = ReturnsNodeByIndex(index - 1);

DoubleLinkedListNode<T> newNode = new DoubleLinkedListNode<T>(element);

newNode.Next = node.Next;

newNode.Previous = node;

node.Next = newNode;

count++;

}

}

public T[] ToArray()

{

T[] elements = new T[count];

DoubleLinkedListNode<T> h = this.head;

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

{

elements[i] = h.Element;

h = h.Next;

}

return elements;

}

public T this[int index]

{

get

{

if (index < 0 || index > count - 1)

{

throw new ArgumentOutOfRangeException("Index out of bounds");

}

return ReturnsNodeByIndex(index).Element;

}

}

public class DoubleLinkedListNode<T>

{

private T element;

DoubleLinkedListNode<T> next;

DoubleLinkedListNode<T> previous;

public DoubleLinkedListNode()

{

element = default(T);

next = null;

previous = null;

}

public DoubleLinkedListNode(T element)

{

this.element = element;

next = null;

previous = null;

}

public DoubleLinkedListNode(T element, DoubleLinkedListNode<T> previous)

{

this.element = element;

this.next = null;

this.previous = previous;

previous.next = this;

}

public void RemoveEl()

{

if (this.previous != null && this.next != null)

{

this.previous.next = this.next;

this.next.previous = this.previous;

}

else if (this.previous != null)

{

this.previous.next = null;

}

else

{

this.next.previous = null;

}

}

public T Element

{

get { return element; }

set { element = value; }

}

public DoubleLinkedListNode<T> Next

{

get { return this.next; }

set { next = value; }

}

public DoubleLinkedListNode<T> Previous

{

get { return this.previous; }

set { previous = value; }

}

}

public int Count

{

get { return count; }

}

IEnumerator<T> IEnumerable<T>.GetEnumerator()

{

DoubleLinkedList<T> list = this;

if (list != null)

{

DoubleLinkedListNode<T> el = list.head;

while (el != null)

{

yield return el.Element;

el = el.Next;

}

}

}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return ((IEnumerable<T>)this).GetEnumerator();

}

}

class Program

{

static void Main(string[] args)

{

DoubleLinkedList<int> linkedList = new DoubleLinkedList<int>();

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

string[] nums = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

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

{

linkedList.Add(int.Parse(nums[i]));

}

string[] positionAndElements = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

int elementToBeInserted = int.Parse(positionAndElements[1]);

int positionToBeInserted = int.Parse(positionAndElements[0]);

int elementToBeRemoved = int.Parse(positionAndElements[2]);

int elementToBeFound = int.Parse(positionAndElements[3]);

linkedList.Insert(elementToBeInserted, positionToBeInserted);

linkedList.Remove(elementToBeRemoved);

foreach (int num in linkedList)

{

Console.Write("{0} ", num);

}

Console.WriteLine();

Console.WriteLine(linkedList.Find(elementToBeFound));

}

}

}

12.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task12\_\_\_DynamicStack

{

public class DynamicStack<T>

{

private int count;

private int capacity;

private int top;

private T[] stack;

public DynamicStack()

{

count = 0;

capacity = 4;

stack = new T[capacity];

top = -1;

}

public void Push(T element)

{

if (count < capacity)

{

stack[count++] = element;

top++;

}

else

{

capacity \*= 2;

T[] tempStack = new T[capacity];

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

{

tempStack[i] = stack[i];

}

stack = tempStack;

stack[count++] = element;

top++;

}

}

public T Peek()

{

if (count > 0)

{

return stack[top];

}

else

{

throw new System.InvalidOperationException("The Stack is empty");

}

}

public T Pop()

{

if (count > 0)

{

top--;

count--;

return stack[top + 1];

}

else

{

throw new System.InvalidOperationException("The Stack is empty");

}

}

public int Count

{

get { return this.count; }

}

public int Capacity

{

get { return this.capacity; }

}

}

class Program

{

static void Main(string[] args)

{

DynamicStack<int> stack = new DynamicStack<int>();

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

string[] nums = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

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

{

stack.Push(int.Parse(nums[i]));

}

Console.WriteLine("Count = {0}", stack.Count);

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

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

{

Console.WriteLine(stack.Pop());

}

Console.WriteLine("Count = {0}", stack.Count);

Console.WriteLine(stack.Peek());

Console.WriteLine("Count = {0}", stack.Count);

}

}

}

13.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Task13\_\_\_Deque  {  class Program  {  public class Deque<T>  {  private DequeNode<T> first;  private DequeNode<T> last;  private int count;  private int leftCount;  private int rightCount;  public Deque()  {  first = null;  last = null;  count = 0;  leftCount = 0;  rightCount = 0;  }  public void AddRight(T element)  {  if (count == 0)  {  first = new DequeNode<T>(element, true, null, null);  last = first;  count++;  return;  }  last = new DequeNode<T>(element, true, null, last);  count++;  rightCount++;  }  public void AddLeft(T element)  {  if (count == 0)  {  first = new DequeNode<T>(element, false, null, null);  last = first;  count++;  leftCount++;  return;  }  first = new DequeNode<T>(element, false, first, null);  count++;  leftCount++;  }  public T GetLeft()  {  if (first.IsRight)  {  throw new InvalidOperationException("Invalid operation over this element");  }  DequeNode<T> node = first;  first = first.Next;  count--;  leftCount--;  node.Delete();  return node.Element;  }  public T GetRight()  {  if (!last.IsRight)  {  throw new InvalidOperationException("Invalid operation over this element");  }  DequeNode<T> node = last;  last = last.Prev;  count--;  rightCount--;  node.Delete();  return node.Element;  }  public void Clear()  {  first = null;  last = null;  count = 0;  rightCount = 0;  leftCount = 0;  }  public int LeftCount  {  get  {  return leftCount;  }  }  public int RightCount  {  get  {  return rightCount;  }  }  public int Count  {  get  {  return count;  }  }  //NODE  public class DequeNode<T>  {  private T element;  private DequeNode<T> next;  private DequeNode<T> prev;  private bool isRight;  internal DequeNode<T> Next  {  get  {  return next;  }  }  internal DequeNode<T> Prev  {  get  {  return prev;  }  }  public DequeNode()  {  this.element = default(T);  next = null;  prev = null;  this.isRight = false;  }  public DequeNode(T element, bool isRight, DequeNode<T> next, DequeNode<T> prev)  {  this.element = element;  this.next = next;  this.prev = prev;  this.isRight = isRight;  if (prev != null)  {  prev.next = this;  }  if (next != null)  {  next.prev = this;  }  }  public T Element  {  get  {  return element;  }  }  public bool IsRight  {  get  {  return isRight;  }  }  public void Delete()  {  if (this.prev != null)  {  this.prev.next = null;  }  if (this.next != null)  {  this.next.prev = null;  }  this.next = null;  this.prev = null;  }  }  }  static void Main(string[] args)  {  Deque<int> deck = new Deque<int>();    int n = int.Parse(Console.ReadLine());  string[] nums = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);  for (int i = 0; i < n; i++)  {  deck.AddLeft(int.Parse(nums[i]));  }  int m = int.Parse(Console.ReadLine());  nums = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);  for (int i = 0; i < m; i++)  {  deck.AddRight(int.Parse(nums[i]));  }  Console.WriteLine("Left: {0} Right: {1} Total: {2}", deck.LeftCount, deck.RightCount, deck.Count);  string[] line = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);  switch (line[0])  {  case "L":  for (int i = 0; i < int.Parse(line[1]); i++)  {  Console.WriteLine(deck.GetLeft());  }  break;  case "R":  for (int i = 0; i < int.Parse(line[1]); i++)  {  Console.WriteLine(deck.GetRight());  }  break;  default:  Console.WriteLine("Wrong command");  break;  }  Console.WriteLine("Left: {0} Right: {1} Total: {2}", deck.LeftCount, deck.RightCount, deck.Count);  }  }  } |

14.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task14\_\_\_CyclicQueue

{

public class CyclicQueue<T>

{

private T[] queue;

private int head;

private int tail;

private int count;

public int Count

{

get

{

return count;

}

}

private int capacity;

private const int INITIAL\_CAPACITY = 16;

public CyclicQueue()

{

capacity = INITIAL\_CAPACITY;

head = 0;

tail = 0;

count = 0;

queue = new T[capacity];

}

public void Enqueue(T element)

{

if (count == capacity)

{

resizeQueue();

}

if (tail == capacity)

{

tail = 0;

}

count++;

queue[tail++] = element;

}

public T Dequeue()

{

if (count == 0)

{

throw new System.InvalidOperationException("Queue empty!");

}

if (head == capacity - 1)

{

head = 0;

count--;

return queue[capacity - 1];

}

count--;

return queue[head++];

}

public T Peek()

{

if (count == 0)

{

throw new System.InvalidOperationException("Queue empty!");

}

return queue[head];

}

public void resizeQueue()

{

T[] newQueue = new T[capacity + INITIAL\_CAPACITY];

if (head < tail)

{

int pos = 0;

for (int i = head; i < tail; i++)

{

newQueue[pos++] = queue[i];

}

head = 0;

tail = pos;

}

else

{

int pos = 0;

for (int i = head; i < capacity; i++)

{

newQueue[pos++] = queue[i];

}

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

{

newQueue[pos++] = queue[i];

}

head = 0;

tail = pos;

}

queue = newQueue;

capacity += INITIAL\_CAPACITY;

}

}

class Program

{

static void Main(string[] args)

{

CyclicQueue<int> cq = new CyclicQueue<int>();

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

string[] nums = Console.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

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

{

cq.Enqueue(int.Parse(nums[i]));

}

Console.WriteLine(cq.Peek());

Console.WriteLine("Count= {0}",cq.Count);

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

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

{

Console.WriteLine(cq.Dequeue());

}

int count= cq.Count;

Console.WriteLine("Count= {0}",cq.Count);

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

{

Console.WriteLine(cq.Dequeue());

}

}

}

}

15.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Collections;

namespace Task15\_\_\_SortLinkedList

{

class Program

{

public static LinkedList<int> MergeSort(LinkedList<int> seq)

{

int count = seq.Count;

if (count <= 1)

{

return seq;

}

LinkedList<int> left = new LinkedList<int>();

LinkedList<int> right = new LinkedList<int>();

int leftInd = 0;

int rightInd = count;

int middleInd = count / 2;

while (seq.Count != middleInd)

{

left.AddLast(seq.First.Value);

seq.RemoveFirst();

}

while (seq.Count != 0)

{

right.AddFirst(seq.First.Value);

seq.RemoveFirst();

}

left = MergeSort(left);

right = MergeSort(right);

return Merge(left, right);

}

static LinkedList<int> Merge(LinkedList<int> left, LinkedList<int> right)

{

LinkedList<int> merged = new LinkedList<int>();

while (left.Count != 0 && right.Count != 0)

{

if (left.First.Value.CompareTo(right.First.Value) > 0)

{

merged.AddLast(right.First.Value);

right.RemoveFirst();

}

else

{

merged.AddLast(left.First.Value);

left.RemoveFirst();

}

}

if (left != null)

{

while (left.Count != 0)

{

merged.AddLast(left.First.Value);

left.RemoveFirst();

}

}

else

{

while (right.Count != 0)

{

merged.AddLast(right.First.Value);

left.RemoveFirst();

}

}

return merged;

}

static void Main(string[] args)

{

LinkedList<int> seq = new LinkedList<int>();

StreamReader reader = new StreamReader("in9.txt");

string[] str;

using (reader)

{

str = reader.ReadLine().Split(new char[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries);

}

foreach (var item in str)

{

seq.AddLast(int.Parse(item));

}

seq = MergeSort(seq);

foreach (var item in seq)

{

Console.Write("{0} ", item);

}

Console.WriteLine();

}

}

}

16.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

namespace Task16\_\_\_TraverseDirectoriesBFSRealization

{

class Program

{

static string[] FindSubdirs(string mainDir)

{

try

{

return Directory.GetDirectories(mainDir);

}

catch (UnauthorizedAccessException)

{

Console.WriteLine("Unauthorized Access ");

}

catch (DirectoryNotFoundException)

{

Console.WriteLine("Directory not found");

}

return new string[0];

}

static void TraverseDirs(string mainDir)

{

Queue<string> dirs = new Queue<string>();

dirs.Enqueue(mainDir);

while (dirs.Count != 0)

{

string[] subDirs = FindSubdirs(dirs.Dequeue());

foreach (string directory in subDirs)

{

dirs.Enqueue(directory);

}

foreach (var dir in subDirs)

{

Console.WriteLine(dir);

}

}

}

static void Main(string[] args)

{

string[] logicalDrives = Directory.GetLogicalDrives();

foreach (var drive in logicalDrives)

{

TraverseDirs(drive);

}

}

}

}

17.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

namespace Task17\_\_\_TraverseDirectoriesDFSRealization

{

class Program

{

static string[] FindSubdirs(string mainDir)

{

try

{

return Directory.GetDirectories(mainDir);

}

catch (UnauthorizedAccessException)

{

Console.WriteLine("Unauthorized Access ");

}

catch (DirectoryNotFoundException)

{

Console.WriteLine("Directory not found");

}

return new string[0];

}

static void TraverseDirs(string mainDir)

{

Stack<string> dirs = new Stack<string>();

dirs.Push(mainDir);

Console.WriteLine(mainDir);

while (dirs.Count != 0)

{

string dir = dirs.Pop();

string[] subDirs = FindSubdirs(dir);

foreach (var subDir in subDirs)

{

Console.WriteLine(subDir);

dirs.Push(subDir);

}

}

}

static void Main(string[] args)

{

string[] logicalDrives = Directory.GetLogicalDrives();

DateTime start = DateTime.Now;

foreach (var drive in logicalDrives)

{

TraverseDirs(drive);

}

}

}

}

18.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Task18\_\_\_Labyrinth

{

//Coordinates

public class Coords

{

private int row;

private int col;

private int distance;

public Coords(int Row, int Col, int Distance)

{

row = Row;

col = Col;

distance = Distance;

}

public int Row

{

get { return this.row; }

}

public int Col

{

get { return this.col; }

}

public int Distance

{

get { return this.distance; }

}

}

class Program

{

//Labyrinth

static string[,] lab;

//Check if the coords are in the labyrinth

static bool IsInRange(int x, int y)

{

if (x < 0 || x >= lab.GetLength(0) || y < 0 || y >= lab.GetLength(1))

{

return false;

}

return true;

}

static Coords FindStartPoint()

{

for (int i = 0; i < lab.GetLength(0); i++)

{

for (int j = 0; j < lab.GetLength(1); j++)

{

if (lab[i, j] == "\*")

{

return new Coords(i, j, 0);

}

}

}

return new Coords(-1, -1, 0);

}

static void FindDistance(Coords startPoint)

{

Queue<Coords> fields = new Queue<Coords>();

fields.Enqueue(startPoint);

while (fields.Count != 0)

{

Coords currentField = fields.Dequeue();

int row = currentField.Row;

int col = currentField.Col;

int dis = currentField.Distance;

//Goes Down

if (IsInRange(row + 1, col) && lab[row + 1, col] == "0")

{

lab[row + 1, col] = (dis + 1).ToString();

fields.Enqueue(new Coords(row + 1, col, dis + 1));

}

//Goes Up

if (IsInRange(row - 1, col) && lab[row - 1, col] == "0")

{

lab[row - 1, col] = (dis + 1).ToString();

fields.Enqueue(new Coords(row - 1, col, dis + 1));

}

//Goes Right

if (IsInRange(row, col + 1) && lab[row, col + 1] == "0")

{

lab[row, col + 1] = (dis + 1).ToString();

fields.Enqueue(new Coords(row, col + 1, dis + 1));

}

//Goes Left

if (IsInRange(row, col - 1) && lab[row, col - 1] == "0")

{

lab[row, col - 1] = (dis + 1).ToString();

fields.Enqueue(new Coords(row, col - 1, dis + 1));

}

}

FindUnreachableFields();

}

private static void FindUnreachableFields()

{

for (int i = 0; i < lab.GetLength(0); i++)

{

for (int j = 0; j < lab.GetLength(1); j++)

{

if (lab[i, j] == "0")

{

lab[i, j] = "u";

}

}

}

}

static void PrintLabyrinth()

{

for (int i = 0; i < lab.GetLength(0); i++)

{

for (int j = 0; j < lab.GetLength(1); j++)

{

Console.Write("{0,3}", lab[i, j]);

}

Console.WriteLine();

}

}

static void Main(string[] args)

{

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

if (n==0)

{

return;

}

lab = new string[n,n];

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

{

string[] line = Console.ReadLine().Split(new char[] { ' ', '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries);

for (int j = 0; j < n; j++)

{

lab[i, j] = line[j].Trim();

}

}

Coords startPoint = FindStartPoint();

if (!IsInRange(startPoint.Row, startPoint.Col))

{

Console.WriteLine("There is no starting point in the lab!");

return;

}

//PrintLabyrinth();

FindDistance(startPoint);

PrintLabyrinth();

}

}

}