

2016/2017

**Programming, Algorithms and Data Structures (210CT)**

Coursework

**MODULE LEADER:** Dr. Diana Hintea

**Student Name: Alain Trantik (Erasmus student)**

**SID: 7455647**

**Github repository:** [**https://github.com/AlainTrantik/210CT.git**](https://github.com/AlainTrantik/210CT.git)

**I can confirm that all work submitted is my own: Yes**

This coursework is testing your ability to design algorithms, write pseudocode, describe their efficiency, implement various data structures and use the programming language of your choice for the implementation.

**Basic in C#:**

**1. Shuffle**

**Code:**

static int [] shuffle(int[] array)

{

Random rand = new Random(); (1 times)

for (int i = 0; i < array.Length; i++) (n times)

{

int r = rand.Next(0, array.Length);   
//random number between [0,tab.Length[

int tmp = array[i];

array[i] = array[r];   
//permutation of some elements randomly

array[r] = array;

}

return array; (1 times)

}

**Display:** static void Main(string[] args)

{

int[] array = {1,2,3,4,5,6,7,8,9};

shuffle(array);

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

{

Console.Write(array[i] + "|");

}

Console.ReadLine();

}

**2. Number of zeros**

**Code:**

static int Factorielle(int n) (n times)

{

if (n == 0)

return 1;

else

return n \* Factorielle(n - 1);

}

If a number is divisible n times by 5, it has n trailing 0s.

static int zeros(int n)

{

int primerFactorOfFive = 0;

while (n % 5 == 0) (n times)

{ n % 5 = 0 means n is divisible by 5

n = n / 5;

primerFactorOfFive++;

}

return primerFactorOfFive;

}

**Display**:

static void Main(string[] args)

{

Console.WriteLine(zeros(Factorielle(10))); }

**3. Perfect square**

**Pseudocode**:

PERFECT\_SQUARE(n)

for i <- n Downto 0

if(sqrt(i) % 1) = 0   
//if sqrt(i)%1=0 => sqrt(i) is a whole number => i is a perfect square

return i

return -1   
//error case: if there is not perfect square less or equal to n

**Code**:

static int perfectSquare(int n)

{

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

{

if (Math.Sqrt(i) % 1 == 0)

return i;

}

return -1;

}

**Display**

static void Main(string[] args)

{

Console.WriteLine(perfectSquare(30));

Console.ReadLine();

}

**4: Run time with Big O notation (week1)**

Basic Task 1: 5n +2 removing constans and multipliers gives O(n).

Basic Task2: 4n +2 removing constans and multipliers gives O(n).

**5: Matrix operations**

**Pseudocode** :

//Because mat1 and mat2 are two quadratic matrices of the same order   
//so we always have (mat1.GetLength(0)=mat1.GetLength(1)=mat2.GetLength(0)=mat2.GetLength(1))

//we add each elements which are in the same position

matAddition(mat1,mat2) O(n²)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

mat3[i,j] <- mat1[i,j] + mat2[i,j]

return mat3

//we substract each elements which are in the same position

matSubtraction(mat1,mat2) O(n²)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

mat3[i,j] <- mat1[i,j] - mat2[i,j]

return mat3

//Multiply the rows of mat1 by the colums of mat2

matMultiplication(mat1,mat2) O(n^3)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

for k <- 0 To length

mat3[i,j] <- mat3[i,j] + (mat1[i,k] \* mat2[k,j])

return mat3

//we multiply each element of the matrix with a (integer) O(n²)

matTimesInt(a,mat1)

length <- mat1.GetLength(0)

mat3 <- [length,length]

for i <- 0 To length

for j <- 0 To length

mat3[i,j] <- a \* mat1[i,j]

return mat3

**Main:**

static void Main(string[] args)

{

int[,] mat1 = { { 1, 2 }, { 3, 4 } }; //B

int[,] mat2 = { { 4, 5 }, { 6, 7 } }; //C

int[,] matAdd = matAddition(mat1, mat2); //(B+C)

int[,] matMul = matMultiplication(mat1, mat2);//(B\*C)

matAdd = matTimesInt(2, matAdd); //2\*(B+C)

int[,] finalMat = matSubtraction(matMul, matAdd);//(B\*C)-2\*(B+C)

Console.ReadLine();

}

Finally we just have to display finalMat.

**So** **the run time of the algorithm above is O(n^3)**.

**6: Reverse a string**

**Pseudocode:**

reverseString(sentence)

tabReversed <- [sentence.Length]

reversed <- ""

j <- 0

for(i <-0 To sentence.Length)

if(sentence[i]!=' ')

tabReversed[j] <- tabReversed[j] + sentence[i]

else j <- j + 1

for(i <- j Downto 0)

reversed <- reversed + tabReversed[i] + " "

return reversed

**Code:**

static string reverseString(string sentence)

{

string[] tabReversed = new string[sentence.Length];

string reversed = "";

int j = 0;

for (int i = 0; i < sentence.Length; i++) (n times)

{

if (sentence[i] != ' ')   
//Fill each cell of tabReversed with each word of the sentence

tabReversed[j] += sentence[i];

else j++;

}

for (int i = j; i >=0; i--) (n times)

{

reversed+=tabReversed[i] + " "; //Display of the array upside down

}

return reversed;

}

**Display (in the main)** : Console.WriteLine(reverseString("This is awesome"));

**Run time:** 4 + 6n removing constans and multipliers gives O(n).

**7: Prime number ?**

**Pseudocode:**

n is prime if it’s only divisible by 2 and itself

prime(n,i) //i starts at 2

if(n = i) //base case: all numbers between 1 and n don’t divide n

return true

if(n % i = 0) //to be prime n must not be divisible by 2,…,n-1

return false

return prime(n, i+1)

**Code:**

static bool prime(int n, int i) //i starts at 2

{

if (n == i) //Base case (n times)

return true;

else if (n % i == 0) (n times)

return false;

return prime(n, i+1); (n times)

}

**Display (in the main) :**

Console.WriteLine(prime(11,2)); =>true

**Run time:**

2 + 3n removing constans and multipliers gives O(n).

**8: Remove vocals from a sentence/word**

**Pseudocode:**

I check the word letter by letter, if it’s a vocal i remove it, if it’s not I check the next letter.

removeVocals(s,i)

if(i>s.Length) //base case (all the letters have been checked)

return s

if(s[i]='a'||'e'||'i'||'o'||'u'||'y')

remove letter i from s

return removeVocals(s,i)

else

return removeVocals(s,i+1)

**Code:**

static string removeVocals(string s,int i)

{

if (i >= s.Length) //Base case

return s;

if (s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u' || s[i] == 'y')

{

s = s.Remove(i, 1); //Remove a letter (index, length)

return removeVocals(s,i);

}

else

{

return removeVocals(s,i+1);

}

}

**Display (in the main) :**

Console.WriteLine(removeVocals(“beautiful”,0);

**9: Binary search with an interval**

**Pseudocode:**

binarySearch(A, a, b, first, last)

if(last<first)

return FALSE

i = first + (last - first) / 2

if(A[i] >= a && A[i] <=b)

return TRUE

if(a[i] < a)

return binarySearch(A, a, b, i+1, last)

else

return binarySearch(A, a, b, first, i-1)

**Code:**

static bool binarySearch(int[] A, int a ,int b, int first, int last)

{

if (last < first)

return false;

int i = first + (last - first) / 2;

if (A[i] >= a && A[i]<=b)

return true;

if (A[i] < a)

return binarySearch(A, a, b, i + 1, last);

else

return binarySearch(A, a, b, first, i - 1);

}

**Display (in the main) :**

static void Main(string[] args)

{

int[] array = { 1, 2, 3, 8, 10, 15, 21 };

Console.WriteLine(binarySearch(array, 4, 8, 0, array.Length - 1));

Console.ReadLine();

}

**Run time :**It’s a binary search so **O(log n)**

**10: Longer sub-sequence of ascending numbers**

**Code :**

static int[] max\_sub\_sequence(int[] sequence)

{

int curr\_length = 0; //current length of the sub-sequence

int max\_length = 0; //maximum length of the sub-sequence

int end = 0; //end index of the longer sub-sequence

int curr\_end = 0; //end index of the current sub-sequence

for (int i = 0; i < sequence.Length - 1; i++)

{

if (sequence[i] < sequence[i + 1])   
//Couting the length of the current sub-sequence of ascending numbers and saving the index of the last number

{

temp\_end = i + 1;

length++;

}

if (!(sequence[i] < sequence[i + 1]) || i == sequence.Length - 2)   
//I put (if i=sequence.Length-2) in order to save the variables if the last number of the sub-sequence is also the last number of the sequence.

{

if (length > max\_length)   
//when the sub-sequence is broken (sequence[i]>sequence[i+1]), I save its length and the index of its last number if the length is higher than the maximum length stored.

{

max\_length = length;

end = temp\_end;

}

length = 0;   
//I reinitialize the length at 0 so it cans count the length of the second sub-sequence...

}

}

int[] sub\_sequence = new int[max\_length + 1];   
//I create an array to store the longer sub-sequence found

for (int i = 0; i <sub\_sequence.Length; i++)

{

sub\_sequence[i] = sequence[end - max\_length + i];   
//the first index of the sub-sequence is lastIndex - length.

}

for (int i = 0; i < sub\_sequence.Length;i++ )

{

Console.Write(sub\_sequence[i] + "|");   
//I display the sub-sequence before return it back.

}

return sub\_sequence;

}

**Main :**

static void Main(string[] args)

{

int[] sequence = {1,2,3,1,2,3,4,5,1};

max\_sub\_sequence(sequence);

Console.ReadLine();

}

**11: Node delete function from a double linked list in C++**

class Node

{

public:

int value; //Any type

Node\* next;

Node\* prev;

Node(int val){

std::cout << "Node constructor!" << std::endl;

this->value = val;

this->next = (Node\*)0;

this->prev = (Node\*)0;

}

~Node(){

std::cout << "Node destructor" << std::endl;

std::cout << "I had the value " << this->value << std::endl;

}

};

class List

{

public:

Node\* head;

Node\* tail;

List(){

std::cout << "List Constructor!" << std::endl;

this->head = 0;

this->tail = 0;

}

~List(){ //Destruction of each node

std::cout << "List destructor!" << std::endl;

Node\* temp;

while (head != 0) {

temp = head->next;

delete head;

head = temp;

}

std::cout << "List destructed!" << std::endl;

}

bool remove(Node\* n) { //remove a node in consideration of its position

if (n != 0 && this->head != n && this->tail != n) {

std::cout << "Remove node" << std::endl;

n->next->prev = n->prev;

n->prev->next = n->next;

return true;

}

if (this->head == n) {

std::cout << "Remove head" << std::endl;

this->head = n->next;

n->next->prev = 0;

return true;

}

else if (this->tail == n) {

std::cout << "Remove tail" << std::endl;

this->tail = n->prev;

n->prev->next = 0;

return true;

}

return false; }

**12: In order traversal of Binary Search Tree in C++**

void in\_order\_iterative(BinTreeNode\* tree){

stack<BinTreeNode\*> S;

BinTreeNode\* current = tree;

S.push(current);

current = current->left;

while (!S.empty() || current){

if (current != NULL) {

S.push(current);

current = current->left;

}

if (current == NULL){

BinTreeNode\* popped=S.top();

S.pop();

std::cout << popped->value << std::endl;

current = popped->right;

}

}

}

**13: Unweighted graph data structure**

**Pseudocode:**

// I chose the adjacency list approach so I created a new class Vertex (Node) with a label and a list of vertices representing the edges and a class Graph containing a list of vertices.

**class Vertex**

label <- 0

edges <- []

addEdge(node)

if(node!=null)

egdes.Add.node

return TRUE

return FALSE

**class Graph**

vertices <- []

addNode(v)

if(v!=null)

vertices.Add(v)

return TRUE

return FALSE

**Main**:

one <- new Vertex

two <- new Vertex

one.label <- 1

two.label <- 2

one.addEdge(two)

two.addEdge(one)

myGraph <- new Graph

myGraph.addNode(one)

myGraph.addNode(two)

**Code :**

class Vertex

{

public int label;

public List<Vertex> edges = new List<Vertex>();

public Vertex()

{

}

public Vertex(int data)//Constructor to Create/Add a new node

{

this.label = data;

}

//Function to add an edge, I add the node in the adjacency list of both nodes with the other one

public bool addEdge(Vertex node)

{

if (node != null)

{

edges.Add(node);

return true;

}

return false;

}

}

class Graph

{

public List<Vertex> vertices = new List<Vertex>();

public Graph()

{

}

//To add a node a simply add it the list of nodes

public bool addNode(Vertex v)

{

if (v != null)

{

this.vertices.Add(v);

return true;

}

return false;

}

//To add several nodes in one time using an array of vertices

public void addNodes(Vertex[] v)

{

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

{

addNode(v[i]);

}

}

}

static void Main(string[] args)

{

Vertex one=new Vertex(1); //I create 10 nodes

Vertex two = new Vertex(2);

Vertex three = new Vertex(3);

Vertex four = new Vertex(4);

Vertex five = new Vertex(5);

Vertex six = new Vertex(6);

Vertex seven = new Vertex(7);

Vertex eight = new Vertex(8);

Vertex nine = new Vertex(9);

one.addEdge(two); //Node 1 is linked to Node 2

two.addEdge(one); //Node 2 is linked to Node 1 and four ...

two.addEdge(four);

four.addEdge(two);

four.addEdge(three);

four.addEdge(five);

three.addEdge(four);

three.addEdge(five);

five.addEdge(three);

five.addEdge(four);

five.addEdge(six);

six.addEdge(five);

six.addEdge(seven);

six.addEdge(eight);

seven.addEdge(six);

seven.addEdge(eight);

eight.addEdge(six);

eight.addEdge(seven);

eight.addEdge(nine);

nine.addEdge(eight);

Graph myGraph = new Graph();

Vertex[] v = { one, two, three, four, five, six, seven, eight, nine}; //I add all the nodes to the graph

myGraph.addNodes(v);

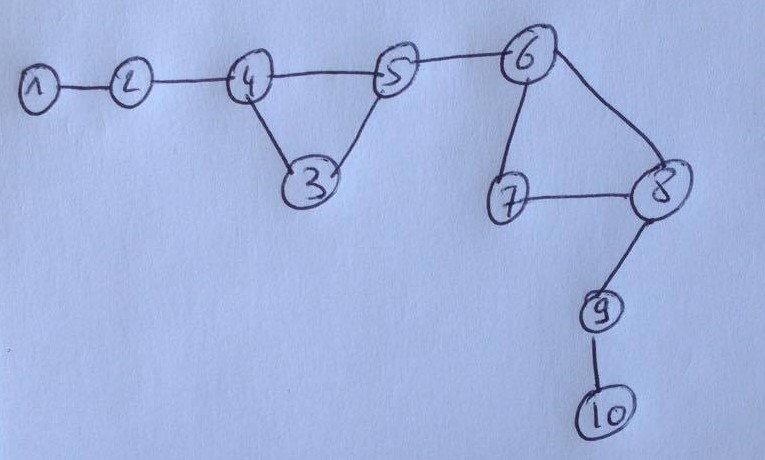
Vertex ten = new Vertex(10);

nine.addEdge(ten); //I create a new node and link it with node 9 by adding it in the edges list of node 9 and I don't forget to add node 10 to the egdes list of node 9   
 ten.addEdge(nine);

myGraph.addNode(ten); //I add it to the graph

}

**MyGraph**:



**14: BFS and DFS traversals**

static List<Vertex> DFS(Graph G, Vertex v)

{

Console.WriteLine("DFS traversal");

Stack<Vertex> S = new Stack<Vertex>();

List<Vertex> visited = new List<Vertex>();

S.Push(v);

while (S.Count != 0)

{

Vertex u = S.Pop();

if(!visited.Contains(u))

{

visited.Add(u);

foreach (Vertex e in u.edges)

S.Push(e);

}

}

saveToText(visited,"DFS");

return visited;

}

static List<Vertex> BFS(Graph G, Vertex v)

{

Console.WriteLine("BFS traversal");

Queue<Vertex> Q = new Queue<Vertex>();

List<Vertex> visited = new List<Vertex>();

Q.Enqueue(v);

while (Q.Count != 0)

{

Vertex u = Q.Dequeue();

if (!visited.Contains(u))

{

visited.Add(u);

foreach (Vertex e in u.edges)

Q.Enqueue(e);

}

}

saveToText(visited,"BFS");

return visited;

}

static void displayList(List<Vertex> l)

{

foreach (Vertex v in l)

Console.WriteLine(v.label);

}

static void saveToText(List<Vertex> vs,string fileName)   
//Function to save a List of vertices in a text file

{

string path = @"c:\Users\Alain\Downloads\"+fileName+".txt";   
//location of the file

List<int> values = new List<int>(); //I create an list of integers and fill it with the label of each vertex in the list of vertices

foreach(Vertex v in vs)

{

values.Add(v.label);

}

//The File.WriteAllLines C# function needs a string array in parameter so I convert my list of integers in a string array

string [] array = values.Select(x => x.ToString()).ToArray();

// This text is added only once to the file.

if (!File.Exists(path))

{

// Create a file to write to.

File.WriteAllLines(path, array);

}

}

static void Main(string[] args)

{

displayList(DFS(myGraph, one));   
//Display myGraph by a DFS beginning with node one

displayList(BFS(myGraph, one));  
//Display myGraph by a DFS beginning with node one

}

**DFS**: 1-2-4-5-6-8-9-10-7-3

**BFS**: 1-2-4-3-5-6-7-8-9-10

**14: Dijkstra’s algorithm for a weighted graph**

**Not done.**