

FA Laborator

Raluca Laura Portase, Eugen-Richard Ardelean

November 16, 2025

Contents

1 Ghid de laborator	6
1.1 Despre laborator	6
1.2 Formatul laboratorului	6
1.2.1 Reguli	6
1.2.2 Notare	6
1.2.3 Predare teme	7
1.2.4 Evaluarea complexității algoritmilor	7
1.2.5 Termene de predare	8
1.2.6 Tentativa de fraudare	8
1.3 Transferul între cadre didactice	8
1.4 Bibliografie sugerată	9
1.5 Profiler	9
2 Sesiune introductiva	10
2.1 Microsoft Visual Studio	10
2.1.1 Instalare	10
2.1.2 Utilizare	10
2.2 JetBrains CLion	13
2.2.1 Instalare	13
2.2.2 Utilizare	14
2.3 C/C++	16
2.3.1 Citire/scriere fișiere	16
2.3.2 Generare cazuri de testare	16
2.3.3 Generare grafice	16
2.3.4 Microsoft Office Excel	17
2.3.5 Exercițiu	19
3 Tema Nr. 1: Analiza și Compararea Metodelor Directe de Sortare	20
3.1 Implementare	20
3.2 Cerințe minimale pentru notare	20
3.3 Cerințe	21
3.3.1 Implementarea metodelor de sortare (5.5p)	21

3.3.2	Analiză comparativă algoritmilor pentru caz mediu statistic (1.5p – 0.5 per algoritm)	21
3.3.3	Analiză comparativă în caz favorabil și defavorabil (3p - câte 0.5p pentru fiecare caz a fiecărui algoritm)	21
3.3.4	Bonus: Insertion sort prin inserție binară (0.5p)	21
3.4	Informații adiționale	22
4	Tema Nr. 2: Analiza și Compararea a două metode de construire a structurii de date Heap: “De jos în sus” (Bottom-up) vs. “De sus în jos” (Top-down)	23
4.1	Implementare	23
4.2	Cerințe minimale pentru notare	23
4.3	Cerințe	24
4.3.1	Analiza comparativă a <i>unuia</i> din algoritmii de sortare din L1 (la alegere) în versiune <i>iterativă</i> vs <i>recursivă</i> . Analiza se va efectua atât din <i>perspectiva numărului de operații, cât și a timpului de rulare</i> (2p)	24
4.3.2	Implementarea metodei bottom-up de construire a unui heap (2p)	24
4.3.3	Implementarea metodei top-down de construire a unui heap (2p)	24
4.3.4	Analiza comparativă a celor două metode de construire în cazul mediu statistic (2p)	24
4.3.5	Analiza comparativă a metodelor de construcție în cazul defavorabil (1p)	25
4.3.6	Implementarea și exemplificarea corectitudinii algoritmului heapsort (1p)	25
4.4	Informații adiționale	25
5	Tema Nr. 3: Analiza și compararea metodelor avansate de sortare – HeapSort și QuickSort / QuickSelect	26
5.1	Implementare	26
5.2	Cerințe minimale pentru notare	26
5.3	Cerințe	27
5.3.1	QuickSort: implementare (2p)	27
5.3.2	QuickSort: evaluare în caz mediu statistic, favorabil și defavorabil (3p)	27
5.3.3	QuickSort și HeapSort: analiză comparativă a cazului mediu statistic (2p)	27
5.3.4	Implementarea hibridizării quicksort-ului (1p)	27
5.3.5	Determinare a unui prag optim în hibridizare din punct de vedere a numărului de operații și a timpului efectiv de execuție + motivatie (1p)	27
5.3.6	Analiză comparativă (intre quicksort și quicksort hibridizat) din punct de vedere a numărului de operații și a timpului efectiv de execuție (1p)	28

5.3.7	Bonus: QuickSelect - Randomized-Select (0.5p)	28
5.4	Informații adiționale	28
6	Tema Nr. 4: Interclasarea eficientă a k liste ordonate	29
6.1	Implementare	29
6.2	Cerințe minimale pentru notare	29
6.3	Cerințe	29
6.3.1	Demo pentru generarea a k liste aleatoare sortate de dimensiuni diferite (având în total n elemente, unde n și k sunt date) și interclasarea a 2 liste (5p)	29
6.3.2	Adaptare operațiilor de <i>min-heap</i> pe structura nouă și interclasarea a k liste (3p)	30
6.3.3	Evaluarea algoritmului în cazul mediu statistic (2p)	30
7	Tema Nr. 5: Căutarea în tabele de dispersie	31
7.1	Implementare	31
7.1.1	Hashing (se referă la tabela de dispersie (hash table))	31
7.2	Cerințe minimale pentru notare	32
7.3	Cerințe	32
7.3.1	Implementarea operației de inserare și căutare folosind structura de date cerută + <i>demonstrare</i> (dimensiune 10) (5p)	32
7.3.2	Evaluarea operației de căutare pentru un singur factor de umplere 95% (2p)	32
7.3.3	Completarea evaluării pentru toți factorii de umplere (2p)	33
7.3.4	Implementare operației de ștergere într-o tabelă de dispersie și evaluarea operației de căutare după ștergerea unor elemente (1p)	33
8	Tema Nr. 6: Arbo里 Multicăi	35
8.1	Implementare	35
8.2	Cerințe minimale pentru notare	35
8.3	Cerințe	36
8.3.1	Implementare a parcurgerii iterative și recursive a unui arbore binar în $O(n)$ și cu <i>memorie aditională constantă</i> (3p)	36
8.3.2	Implementarea corectă la pretty-print la $R1$ (2p)	36
8.3.3	Implementarea corectă la $T1$ (din $R1$ în $R2$) și pretty-print la $R2$ (1p) + $T1$ în timp liniar (1p)	36
8.3.4	Implementarea corectă la $T2$ (din $R2$ în $R3$) și pretty-print la $R3$ (2p) + $T2$ în timp liniar (1p)	36
9	Tema Nr. 7: Statistici dinamice de ordine	38
9.1	Implementare	38
9.2	Cerințe minimale pentru notare	38
9.3	Cerințe	39
9.3.1	BUILD_TREE: implementare corectă și eficientă (5p)	39

9.3.2 OS_SELECT: implementare corectă și eficientă (1p)	39
9.3.3 OS_DELETE: implementare corectă și eficientă (2p)	39
9.3.4 Evaluarea operațiilor de management - BUILD, SELECT, DELETE (2p)	39
9.3.5 Bonus: Implementarea utilizând AVL / arbori roșu și ne- gru (1p)	40
10 Tema Nr. 8: Multimi disjuncte	41
10.1 Implementare	41
10.2 Cerințe minimale pentru notare	41
10.3 Cerințe	42
10.3.1 Implementare corectă a MAKE_SET, UNION și FIND_SET (5p)	42
10.3.2 Implementarea corectă și eficientă a algoritmului lui Kruskal (2p)	42
10.3.3 Evaluarea operațiilor pe multimi disjuncte (MAKE, UNION, FIND) folosind algoritmului lui Kruskal (3p)	42
11 Tema Nr. 9: Căutarea în lățime (BFS)	44
11.1 Introducere	44
11.2 Cerințe minimale pentru notare	44
11.3 Structura proiectului	44
11.3.1 Inițializarea proiectului pe Windows, cu Visual Studio . .	45
11.3.2 Inițializarea proiectului pe Linux și Mac	45
11.4 Funcționare	45
11.4.1 Exemplu: comanda <code>neighb</code>	46
11.4.2 Exemplu: comenziile <code>bfs</code> și <code>bfs_step</code>	46
11.4.3 Exemplu: comanda <code>bfs_tree</code>	47
11.4.4 Exemplu: comanda <code>path</code>	47
11.4.5 Structuri de date folosite	48
11.5 Cerințe	49
11.5.1 Determinarea vecinilor unei celule (2p)	49
11.5.2 Implementarea algoritmului BFS (3p)	49
11.5.3 Afisarea arborelui BFS (2p)	50
11.5.4 Evaluarea performanței algoritmului BFS (3p)	50
11.5.5 Bonus: Determinarea celui mai scurt drum (0.5p)	51
11.5.6 Bonus: Unde poate ajunge un cal pe tablă? (0.5p)	51
12 Tema Nr. 9: Tutorial	52
12.1 Configurare Proiect Visual Studio	52
12.2 Eroare ‘unsafe’ în Visual Studio	55
12.3 Eroare ‘Assertion’ în Visual Studio	60
12.4 Eroare undefined în CLion	61
12.5 CLion Visual Studio – Opțiunea 1 (mai lentă)	62
12.6 CLion MinGW – Opțiunea 2 (mai rapidă, necesită consolă externă)	65
12.6.1 Eroare Clear în CLion	65

12.6.2 CLion nu afișează grila	67
12.7 Comanda de rulare pe Mac	68
13 Tema Nr. 10: Căutare în adâncime (DFS)	69
13.1 Implementare	69
13.2 Cerințe minimale pentru notare	69
13.3 Cerințe	69
13.3.1 DFS (5p)	69
13.3.2 Sortare topologică (1p)	70
13.3.3 Tarjan (2p)	70
13.3.4 Analiza performanței pentru DFS (2p)	70
14 Rezolvarea erorilor	72
14.1 Profiler + VS Code error	72
References	74

1 Ghid de laborator

1.1 Despre laborator

În acest semestru, urmează să implementați o serie de algoritmi și să analizați corectitudinea și eficiența proprietăilor implementări. Implementările vor avea ca punct de pornire pseudo-codul de la curs și seminar. Le puteți scrie în C sau C++ sau un alt limbaj de programare pe care îl cunoașteți atâtă timp cât implementați toate structurile de date de care aveți nevoie.

1.2 Formatul laboratorului

1.2.1 Reguli

- Prezența este obligatorie
- O absență poate fi recuperată (tema corespunzătoare poate fi predată în următoarea săptămână)
- Absența a 2-a poate fi recuperată în laboratorul special de la sfârșitul semestrului (contra cost) rezolvând teme suplimentare (NU se rezolva tema corespunzătoare sesiunii în care s-a înregistrat absența)
- Dacă ai mai mult de 2 absențe **nu poți participa la examen în sesiunea normală**
- **IN MOD EXCEPTIONAL**, poți participa la laborator în aceeași săptămână cu o altă grupă dacă mă anunță din timp pe email, și primești acordul meu (a cadrului didactic de laborator de la care lipsești); tema se prezintă tot cadrului didactic la sesiunile căruia ești înscris (tema se încarcă la timp, se prezintă data urmatoare când participi cu semigrupa ta)

1.2.2 Notare

- Nota de la laborator valorează **30%** din nota finală
- Nota de la laborator este compusă din două părți: Nota pe teme - 2/3 din nota de laborator (altfel spus **20%** din nota finală) și Colocviu - 1/3 din nota de laborator (altfel spus **10%** din nota finală)
- Teme
 - Fiecare temă are o pondere egală la calcului notei finale pe teme - media aritmetică
 - Fiecare temă are mai multe praguri de notare accesibile în documentul de cerințe
- Colocviu

- Colecțiul constă într-un test de laborator de tip closed book în ultima săptămână a semestrului (W14)
- Colecțiul va fi susținut de fiecare student pe calculatoarele disponibile în sala de laborator (nu pe laptopuri / calculatoare personale)
- Pentru a promova laboratorul și a fi acceptați în examen, trebuie să îndepliniți **ambele cerințe**:
 - Nota teme ≥ 5
 - Nota coloctrui ≥ 5

1.2.3 Predare teme

- La discuția de evaluare se vor prezenta: **codul sursă, graficele și un exemplu de rulare**
- **Codul sursă** (ex: program.cpp) și **graficele** trebuie încărcate pe Moodle, într-o arhivă (ex: program.zip), **înainte de sesiunea de laborator**
- **Nu evaluăm** teme cu cod neindentat
- **Nu evaluăm** teme pentru care studentul nu poate explica algoritmul (algoritmii) utilizati
- Fiecare fișier sursă trebuie să conțină la început un comentariu cu următorul format:

```
/*
* @author Ionescu Popescu
* @group 30221
*
* Specificațiile problemei, ex: Comparați metodele de sortare X, Y
*
* Interpretarea personală despre complexitate (timp și spațiu), despre cazurile
de testare (favorabil,
* mediu-statistic și nefavorabil) ex: Metoda X are complexitatea Y în cazul
Z pentru ca ..
*/
```

1.2.4 Evaluarea complexității algoritmilor

- Pentru cazul mediu-statistic, repetați măsurările de cel puțin 5 ori
- Măsurăți numărul de operații efectuate de algoritm (atribuiri și comparații pe datele de intrare sau pe variabile auxiliare ce conțin date de intrare)
- Variați dimensiunea datelor de intrare în concordanță cu specificația fiecărei teme

- Aplicați aceleasi date de intrare pe fiecare algoritm în cazul evaluărilor comparative (cazul mediu statistic)
- Generați grafice pentru evaluare (fie în **Excel** sau folosind **Profiler-ul**)
- Analizați graficele și adăugați observațiile personale în secțiunea de început

1.2.5 Termene de predare

Temele pot fi predate:

- În cadrul laboratorului în care sunt discutate. La finalul laboratorului trebuie să încărcați pe **moodle** o versiune a temei curente (cu cat ați apucat să lucrați la ea). Lipsa unei submisii la finalul orei (sau a unei submisii cu prea puțin cod relevant) se penalizează cu pana la 2 puncte din nota pe acea tema.
 - **Extensia_1** (E1): la începutul următorului laborator
 - **Extensia_2** (E2): anumite teme se pot preda la începutul celui de-al doilea laborator de după cel în care a fost prezentata tema (cu o **penalizare** de -2)
 - Din cel de-al treilea laborator, tema nu mai poate fi predata (valorează 0)
- Găsiți o planificare a temelor și a extensiilor corespunzătoare pe Moodle.
Temele trebuie încărcate pe Moodle înainte de începutul laboratorului în care sunt predate.

1.2.6 Tentativa de fraudare

Pentru prima tentativă de fraudare descoperită (copiatul codului altor persoane sau folosirea de cod generat prin unelte AI), tema respectivă se puntează cu 0 puncte. *O tentativă ulterioară de fraudare duce la recontractarea materiei anul următor.*

1.3 Transferul între cadre didactice

Dacă doriți să participați la orele de laborator cu un alt cadru didactic trebuie să respectați următoarea regulă:

- Studentul S1 din grupa G1 poate să se mute în grupa G2 doar dacă găsește un student S2 din grupa G2 ce e dispus să participe la laborator cu grupa G1.

Pentru a formaliza ”transferul” trebuie să trimiteți un email în care să precizați cu cine faceți transferul:

- Un email de la S1 către cele două cadre didactice implicate
- Un email de la S2 către cele două cadre didactice implicate

Termenul limită pentru ”transfer” e sfârșitul **săptămânii 2** de scoala.

1.4 Bibliografie sugerată

- Cormen, T. H. et al (2009). Introduction to algorithms. MIT press
- J. Kleinberg, E. Tardos (2005). Algorithm Design. Addison Wesley
- Tutoriale C/C++
 - <http://www.cprogramming.com/begin.html>
 - <http://www.learn-c.org>
 - Accelerated C++: Practical Programming by Example
- Ghiduri de stil
 - <http://users.ece.cmu.edu/~eno/coding/CCodingStandard.html>
 - http://www.cs.swarthmore.edu/~newhall/unixhelp/c_codestyle.html
 - <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

1.5 Profiler

Biblioteca care se va utiliza pentru generarea graficelor, fiecare student va trebui sa parcurgă exemplul și tutorialul dat.

Cea mai recentă versiune se găseste aici:

<https://github.com/cypyryoprisa/utcn-fa-profiler>

Profiler Tutorial:

- Part 1
- Part 2
- Part 3

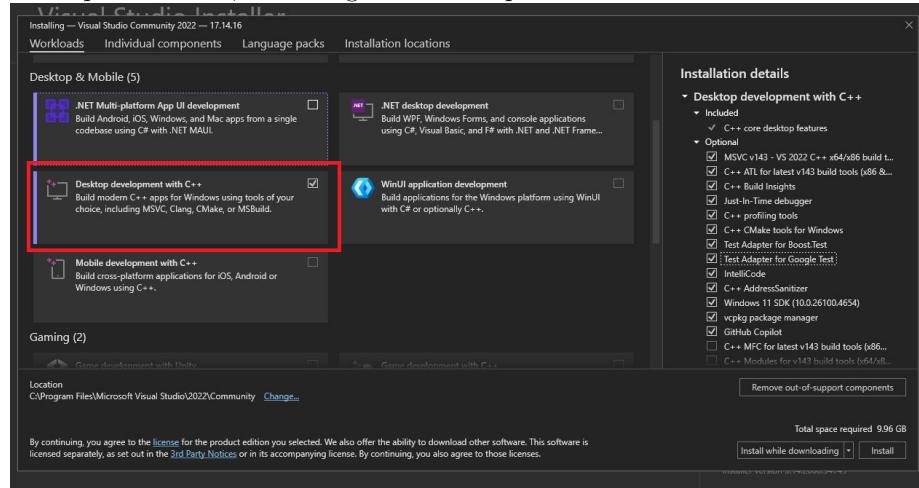
2 Sesiune introductiva

Pentru început asigură-te că ai citit Ghidul de laborator de pe Moodle. În acest laborator vei învăța cum să scrii un program C/C++ în Microsoft Visual Studio sau JetBrains CLion. De asemenea vei învăța cum să-ți generezi datele pentru evaluarea algoritmilor și cum să generezi grafice în Microsoft Office Excel.

2.1 Microsoft Visual Studio

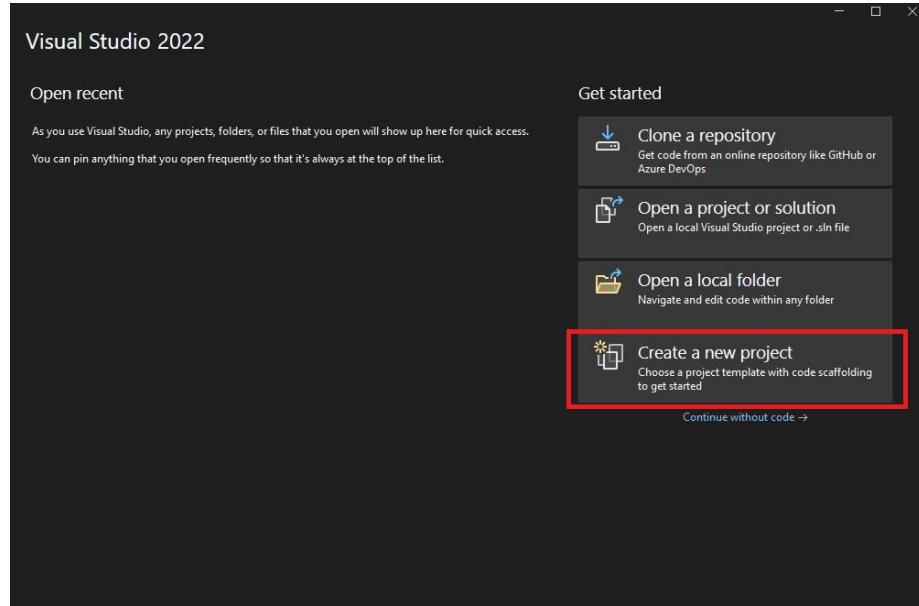
2.1.1 Instalare

Ediție gratuită: <https://visualstudio.microsoft.com/vs/community/>
În timpul instalării, se va alege următorul pachet:

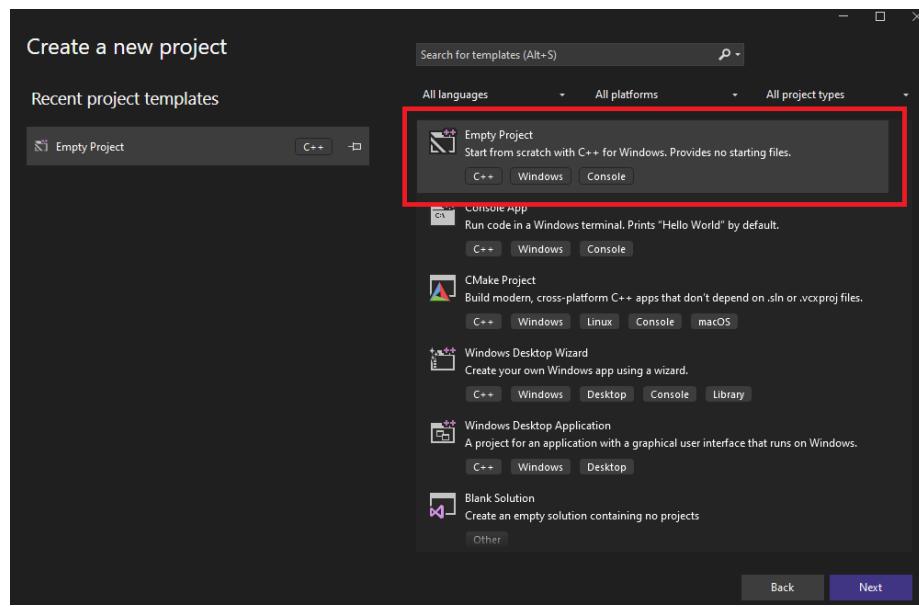


2.1.2 Utilizare

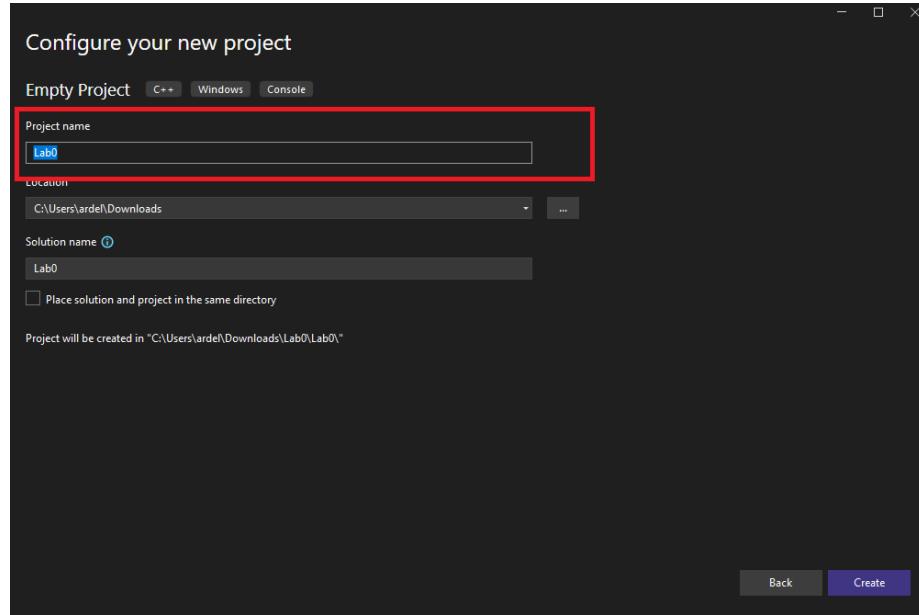
1. Create project:



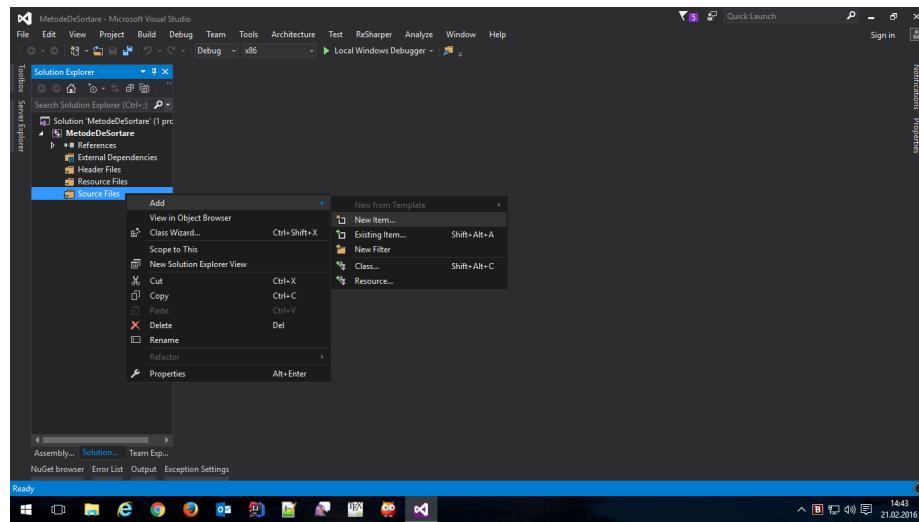
2. Select “Empty project”

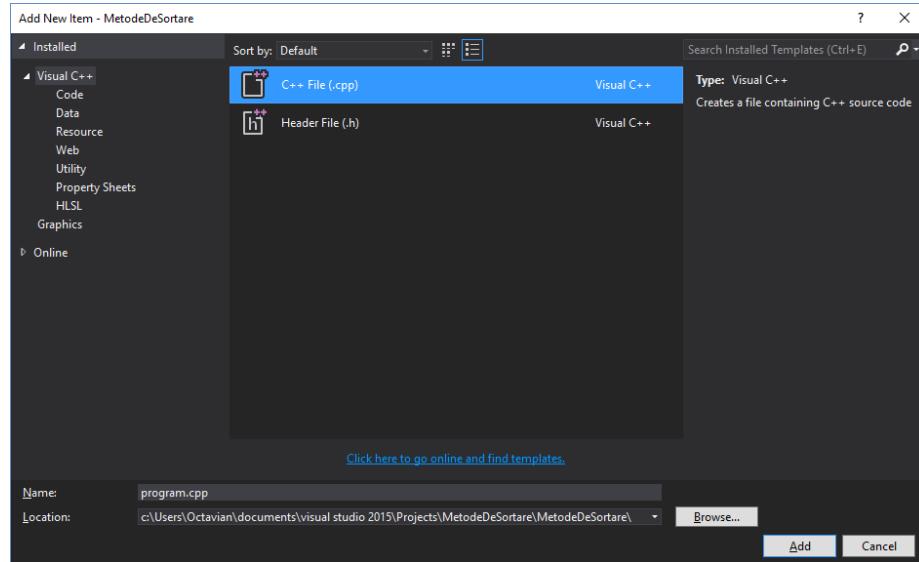


3. Give the project a name.

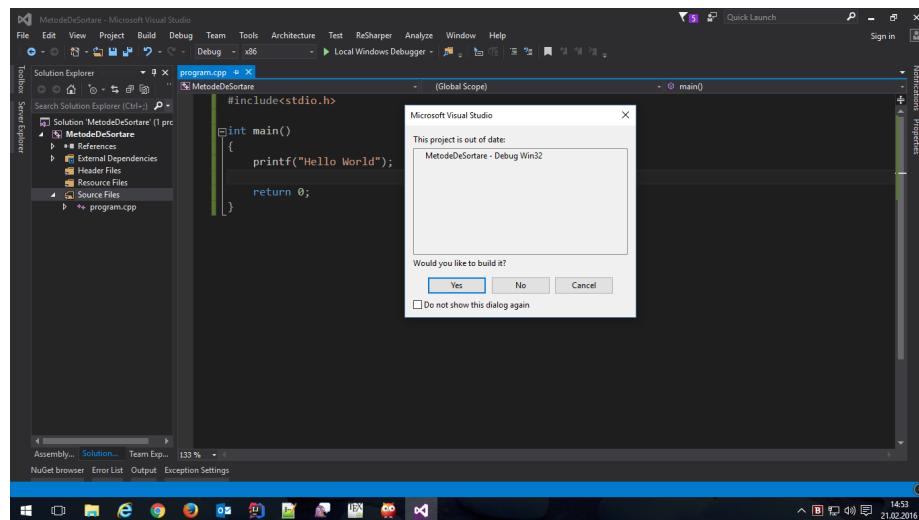


4. Creează un fișier `*.cpp` [select from 'Solution Explorer' menu - might be left/right].





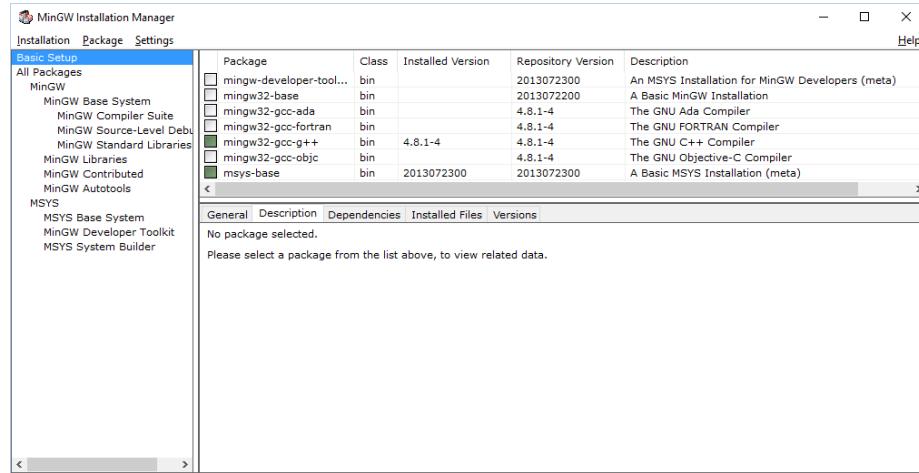
5. Compilare și execuție program (în mod DEBUG)



2.2 JetBrains CLion

2.2.1 Instalare

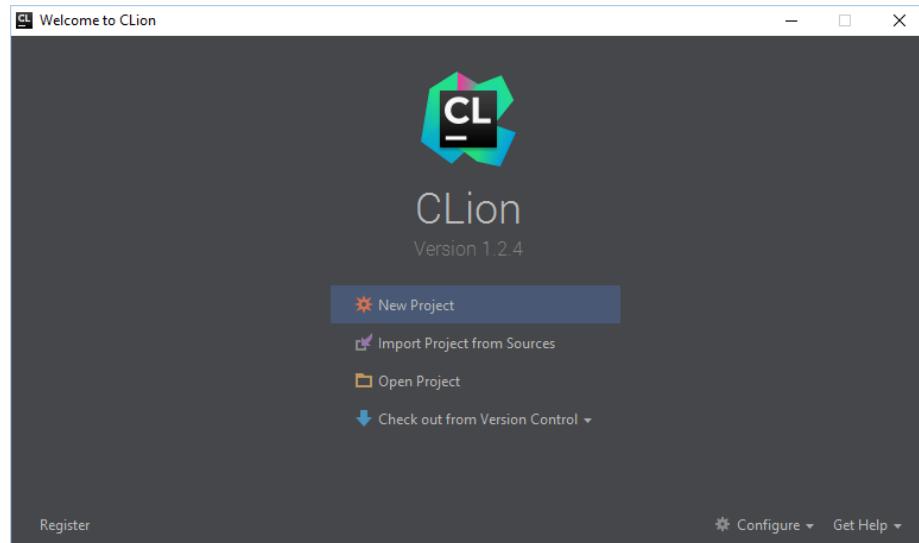
Înregistrează-te cu adresa [@student.utcluj.ro](https://www.jetbrains.com/student/) pe <https://www.jetbrains.com/student/>
Descarcă și instalează MinGW: <https://sourceforge.net/projects/mingw/>



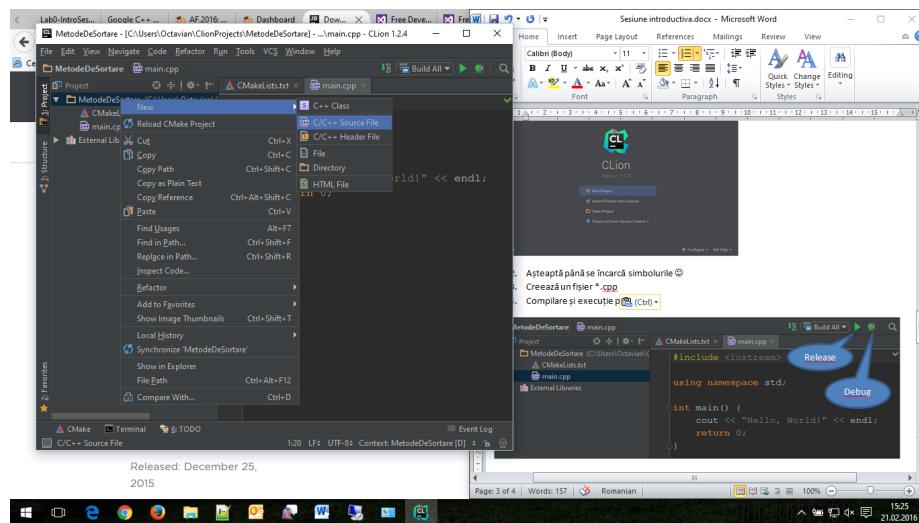
Descarcă și instalează CLion: <https://www.jetbrains.com/clion/download/#section=windows>

2.2.2 Utilizare

1. Creare proiect: *New Project*



2. Așteaptă până se încarcă simbolurile
3. Creează un fisier **.cpp*



4. Compilare și execuție program

```
#include <iostream>

using namespace std;

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

Debug

Release

2.3 C/C++

2.3.1 Citire/scriere fișiere

Exercițiu - pași:

- Declară un sir v de lungime MAX_SIZE (o constantă definită de tine)
- Citește n de la tastatură
- Deschide fișierul *input.txt*, citește n numere din el și salvează-le în v
- Salvează cele n numere în fișierul *output.txt* în ordine **inversă**

2.3.2 Generare cazuri de testare

Pentru a testa algoritmii care o să-i implementezi, va trebui să folosești o serie de date de intrare: siruri ordonate crescător, siruri ordonate descrescător, siruri aleatoare etc. Generarea sirurilor crescătoare/descrescătoare ar trebui să fie simplă. Pentru generarea sirurilor aleatoare poți folosi următoarele:

- Biblioteca *Profiler* de pe Moodle (sau <https://github.com/cypyopriza/utcn-fa-profiler>)
- metodele *rand()*, *srand()*, citește:
 - <http://www.cplusplus.com/reference/cstdlib/rand/>
 - <http://www.cplusplus.com/reference/cstdlib/srand/>
 - http://www.cplusplus.com/reference/cstdlib/RAND_MAX/

Exercițiu - pași:

- Citește n , min și max de la tastatură
- Generează un sir aleatoriu de n elemente cu valori cuprinse între min și max
- Sirul trebuie să fie diferit la fiecare rulare a programului
- Adaugă sirul în fișierul *output.txt*

2.3.3 Generare grafice

Pentru generarea graficelor poți folosi:

- Biblioteca *Profiler* de pe Moodle (sau <https://github.com/cypyopriza/utcn-fa-profiler>)
- Microsoft Office Excel

2.3.4 Microsoft Office Excel

Va trebui să creezi un fișier cu extensia *.csv* (comma-separated values). Fișierul ar trebui să aibă o structură de felul următor:

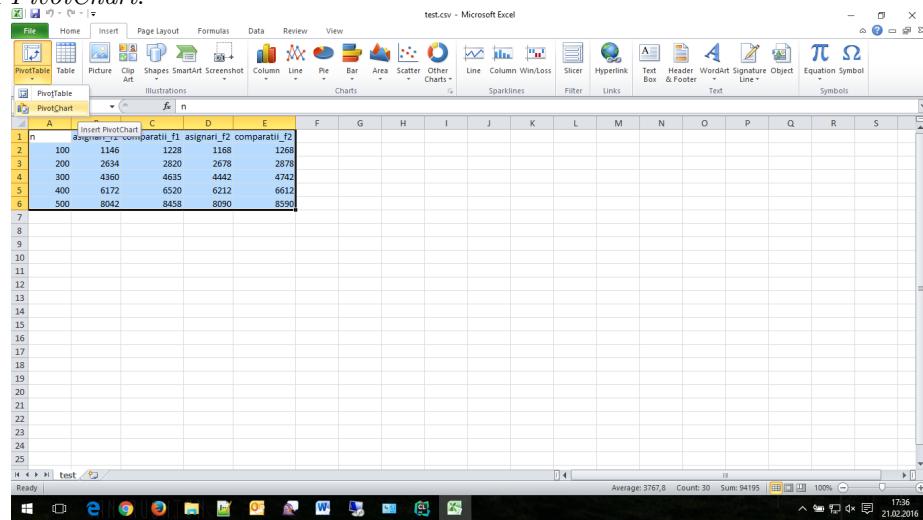
```
n,asignari_f1,comparatii_f1,asignari_m2,comparatii_m2
100,1146,1228,1168,1268
200,2634,2820,2678,2878
300,4360,4635,4442,4742
400,6172,6520,6212,6612
500,8042,8458,8090,8590
```

Legendă:

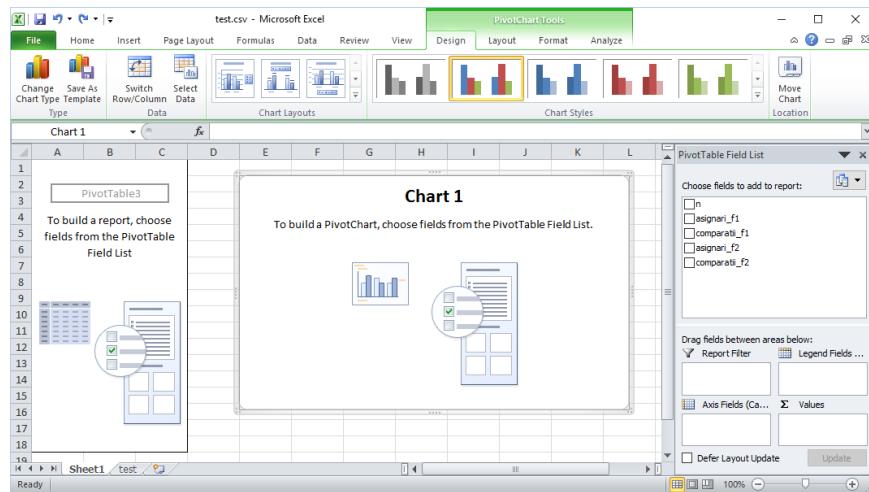
- n=dimensiunea problemei (ex: lungimea sirului de intrare)
- asignari_f1=numărul de asignări pentru cazul favorabil și metoda 1
- comparatii_f2=numărul de comparații pentru cazul favorabil și metoda 2

Atenție: Dacă deschizi fișierul CSV în Excel și valorile apar pe o singura coloană înseamnă că trebuie să folosești alt caracter de separare (ex: folosește punct și virgulă ”,”).

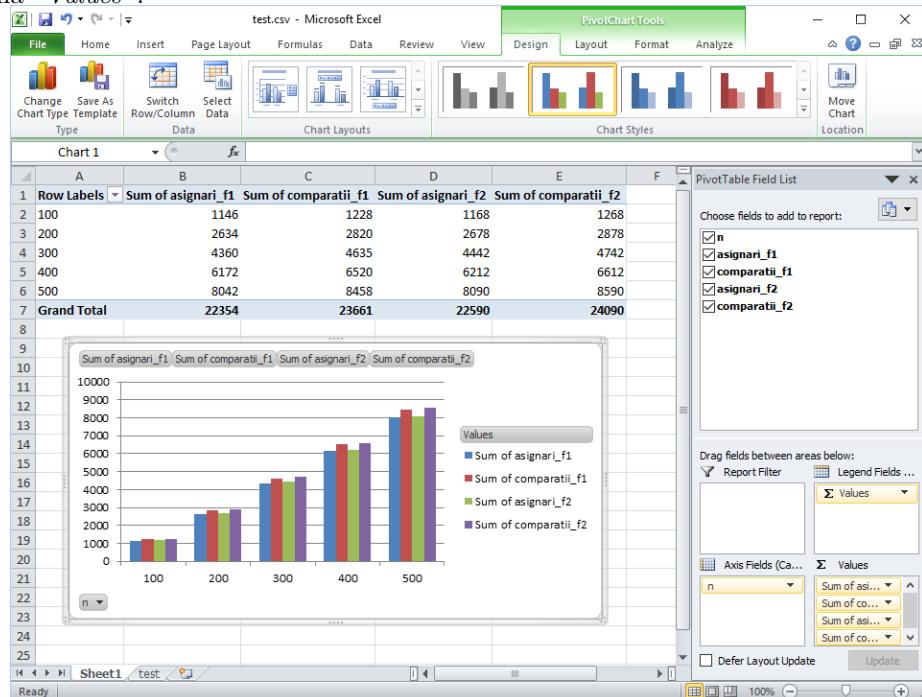
După ce ai deschis fișierul CSV în Excel, selectează toate valorile și creează un *PivotChart*.



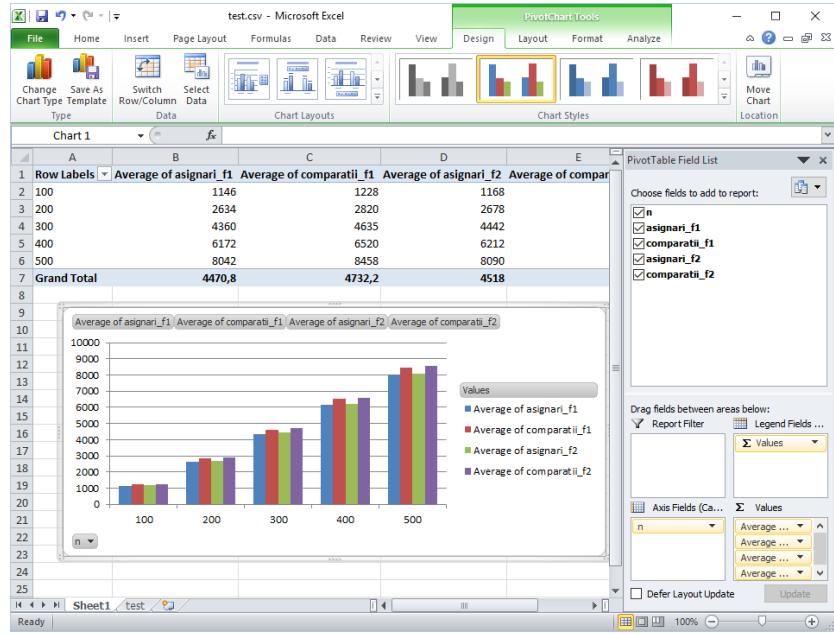
După ce apeși pe *Ok*”, fereastra ar trebui să arate ca în poza de mai jos.



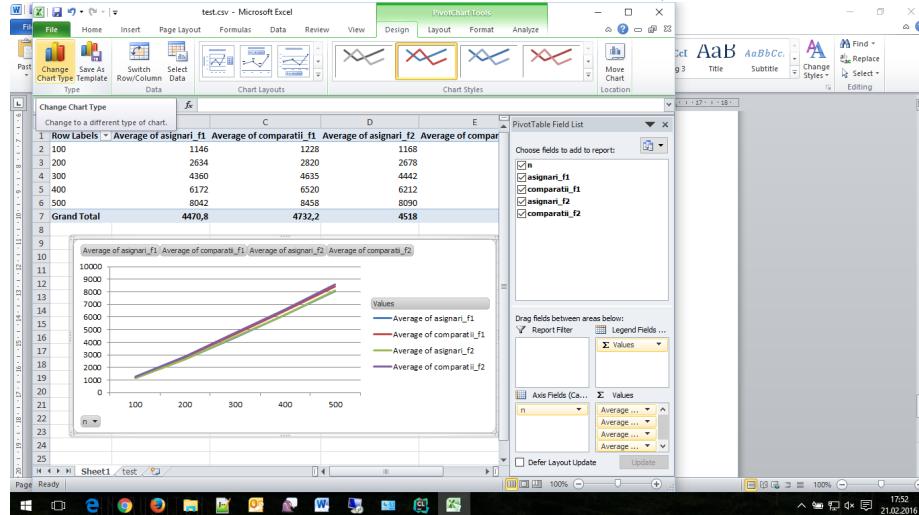
În panoul din stânga trage ”n” în zona ”Axis Fields” și celelalte coloane în zona ”Values”.



Schimbă funcția de agregare ”sum” în ”average”: click pe săgeata neagră de la fiecare rând din zona ”Values”, apoi ”Value Field Settings” și alege ”Average”. Dacă le-ai schimbat corect, fereastra ar trebui să arate ca în poza de mai jos.



Ultimul pas este să schimbi tipul graficului într-un grafic de tip linie de la "Change Chart Type". Rezultatul final ar trebui să arate aşa:



2.3.5 Exercițiu

Scrie un program care pentru fiecare n din intervalul $\{100, 200, \dots, 10.000\}$ calculează și adaugă într-un fișier următoarele valori:

$$n, 100*\log(n), 10*n, n*\log(n), 0.1*n^2, 0.01*n^3$$

Folosește valorile din fișier ca să generezi un grafic în funcție de n .

3 Tema Nr. 1: Analiza și Compararea Metodelor Directe de Sortare

Timp alocat: 2 ore

3.1 Implementare

Se cere implementarea **corectă** și **eficientă** a 3 metode directe de sortare (*Sortarea Bulelor*, *Sortarea prin Inserție* – folosind inserție liniară sau binară, și *Sortarea prin Selecție*)

- Intrare: un sir de numere $\langle a_1, a_2, \dots, a_n \rangle$
- Ieșire: o permutare ordonată a sirului de la intrare $\langle a'_1 \leq a'_2 \leq \dots \leq a'_n \rangle$

Toate informațiile necesare și pseudo-codul se găsesc în notițele de la **Seminarul nr. 1** (Sortarea prin Insertie este prezentată și în **cartea[1]** – **secțiunea 2.1**). Să verificați că ati implementat varianta eficientă pentru fiecare din algoritmii de sortare (dacă mai multe versiuni au fost prezentate)

3.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumea voastră la întrebările puse de către profesor.*

3.3 Cerințe

3.3.1 Implementarea metodelor de sortare (5.5p)

- Bubble sort (1.5p)
- Insertion sort (2p)
- Selection sort (2p)

Demo: Corectitudinea algoritmului va trebui exemplificată pe date de intrare de dimensiuni mici.

3.3.2 Analiză comparativă algoritmilor pentru caz mediu statistic (1.5p – 0.5 per algoritm)

Se cere compararea celor 3 algoritmi în cazurile: **favorabil (best)**, **mediu statistic (average)** și **defavorabil (worst)**. Pentru cazul **mediu** va trebui să repetați măsurările de **m** ori ($m=5$ este suficient) și să raportați media rezultatelor; de asemenea, pentru cazul **mediu**, asigurați-vă că folosiți **aceleași** date de intrare pentru cele 3 metode de sortare (astfel încât compararea lor să fie corectă); identificați și generați date de intrare pentru cazurile: **favorabil** și **defavorabil**, pentru toate cele 3 metode de sortare.

Pașii de analiză ai metodelor de sortare pentru fiecare din cele 3 cazuri (**favorabil, defavorabil, mediu**):

- variați dimensiunea sirului de la intrare (n) între [100...10.000], cu un increment de maxim 500 (sugerăm 100).
- pentru fiecare dimensiune, generați datele de intrare adecvate pentru metoda de sortare; rulați metoda de sortare numărând operațiile (numărul de atribuiră, numărul de comparații și suma lor).

3.3.3 Analiză comparativă în caz favorabil și defavorabil (3p - câte 0.5p pentru fiecare caz a fiecărui algoritm)

Pentru fiecare caz de analiză (**favorabil, defavorabil și mediu**), generați grafice care comparație cele 3 metode de sortare; folosiți grafice diferite pentru numărul de atribuiră, comparații și suma lor. Dacă o curbă nu poate fi vizualizată corect din cauza că celelalte curbe au o rată mai mare de creștere (ex: o funcție liniară pare constantă atunci când este plasată în același grafic cu o funcție pătratică), atunci plasați noua curbă și pe un alt grafic. Denumiți adecvat graficele și curbele.

3.3.4 Bonus: Insertion sort prin inserție binară (0.5p)

Demo: Corectitudinea algoritmului va trebui exemplificată pe date de intrare de dimensiuni mici.

Va trebui să comparați insertion sort prin inserție binară cu toti ceilalți algoritmi (bubble, insertion, selection) pentru toate cazurile (favorabil, mediu statistic, defavorabil).

3.4 Informatii aditionale

! Doar atribuirile (=) și comparațiile (<, ==, >, !=) care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

4 Tema Nr. 2: Analiza și Compararea a două metode de construire a structurii de date Heap: “De jos în sus” (Bottom-up) vs. “De sus în jos” (Top-down)

Timp de lucru: 2 ore

4.1 Implementare

Se cere implementarea **corectă** și **eficientă** a două metode de construire a structurii de date Heap i.e., “de jos în sus” (*bottom-up*) și “de sus în jos” (*top-down*). De asemenea, se cere implementarea algoritmului *heapsort*.

Informații utile și pseudo-cod găsiți în notițele de curs sau în bibliografie [1]:

- “*De jos în sus*”: secțiunea 6.3 (Building a heap)
- *Heapsort*: secțiunea 6.4 (The heapsort algorithm)
- “*De sus în jos*”: secțiunea 6.5 (Priority queues) și problema 6-1 (Building a heap using insertion)

4.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo*: Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumea voastră la întrebările puse de către profesor.*

4.3 Cerințe

4.3.1 Analiza comparativă a uneia din algoritmii de sortare din L1 (la alegere) în versiune iterativă vs recursivă. Analiza se va efectua atât din perspectiva numărului de operații, cât și a timpului de rulare(2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

Pentru analiza comparativă a versiunii iterative vs recursive, alegeți oricare din cei 3 algoritmi din laboratorul 1 (bubble sort, insertion sau selection). Folosiți varianta iterativă pe care ati implementat-o și predat-o în cadrul laboratorului (corectată, dacă este nevoie, în funcție de feedback-ul pe care l-ați primit) și implementați același algoritm de sortare în mod recursiv.

Trebuie să măsurați atât efortul total, cât și timpul de rulare al celor două versiuni (iterativ și recursiv) => două grafice, fiecare comparând cele două versiuni de algoritm.

4.3.2 Implementarea metodei bottom-up de construire a unui heap (2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

4.3.3 Implementarea metodei top-down de construire a unui heap (2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

4.3.4 Analiza comparativă a celor două metode de construire în cazul mediu statistic (2p)

Se cere compararea celor două metode de construcție a structurii heap în cazul mediu statistic. Pentru cazul mediu statistic, va trebui să repetați măsurătorile de m ori ($m=5$) și să raportați valoarea lor medie; de asemenea, pentru cazul mediu statistic, asigurați-vă că folosiți aceleasi date de intrare pentru cele două metode.

Pașii de analiză:

- variați dimensiunea sirului de intrare (n) între [100...10000], cu un increment de maxim 500 (sugerăm 100).
- pentru fiecare dimensiune (n), generați date de intrare adecvate metodei de construcție; rulați metoda numărând operațiile elementare (atribuiri și comparații - pot fi numărate împreună pentru această temă).

Generați un grafic ce compară cele două metode de construcție în cazul mediu statistic pe baza numărului de operații obținut la pasul anterior. Dacă o curbă nu poate fi vizualizată corect din cauza că celelalte curbe au o rată

mai mare de creștere, atunci plasați noua curbă și pe un alt grafic. Denumiți adecvat graficele și curbele.

4.3.5 Analiza comparativă a metodelor de construcție în cazul defavorabil (1p)

4.3.6 Implementarea și exemplificarea corectitudinii algoritmului heapsort (1p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

4.4 Informatii aditionale

! Doar atribuirile (=) și comparațiile (<, ==, >, !=) care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

Pentru a măsura timpul de execuție puteți folosi Profiler similar cu exemplul de mai jos.

```
profiler.startTimer("your_function", current_size);
for(int test=0; test<nr_tests; ++test) {
    your_function(array, current_size);
}
profiler.stopTimer("your_function", current_size);
```

Numărul de teste (*nr.tests* din exemplul de mai sus) trebuie ales în funcție de procesor și modul de compilare. Sugerăm valori mai mari, precum 100 sau 1000.

În momentul în care măsurăți timpul de execuție, asigurați-vă că opriți orice alte proceze care nu sunt necesare.

! *Observație.* Pentru a evalua cât mai corect timpul, Profiler nu va face numărătoarea operațiilor. Astfel, evaluarea de timp și de operații trebuie făcute separat.

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

5 Tema Nr. 3: Analiza și compararea metodelor avansate de sortare – HeapSort și QuickSort / QuickSelect

Timp alocat: 2 ore

5.1 Implementare

Se cere implementarea **corectă** și **eficientă** a Sortării Rapide (*Quicksort*), Sortării Rapide Hibridizate (*Hybrid Quicksort*) și *Quick-Select* (*Randomized-Select*). Se cere și analizarea comparative a complexității Sortării folosind Heapsuri (*Heapsort*, implementat în Tema Nr. 2) și Sortarea Rapidă (*Quicksort*).

Informații utile și pseudo-cod găsiți în notițele de curs sau în carte[1]:

- *Heapsort*: capitolul 6 (Heapsort)
- *Quicksort*: capitolul 7 (Quicksort)
- *Hibridizare quicksort utilizând insertion sort iterativ* - în quicksort, pentru dimensiuni de sir < prag, se utilizează insertion sort (folosiți implementarea insertion sort din Tema Nr. 1).
- *QuickSelect/Randomized Select*: capitolul 9

5.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo*: Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul correct dat de dumea voastră la întrebările puse de către profesor.*

5.3 Cerințe

5.3.1 QuickSort: implementare (2p)

Demo: Corectitudinea algoritmului va trebui exemplificată pe date de intrare de dimensiuni mici.

5.3.2 QuickSort: evaluare în caz mediu statistic, favorabil și defavorabil (3p)

Pașii de analiză:

- variați dimensiunea sirului de intrare (n) între [100...10000], cu un increment de maxim 500 (sugerăm 100).
- pentru fiecare dimensiune (n), generați date de intrare adecvate metodei de construcție; rulați metoda numărând operațiile elementare (atribuirile și comparațiile pot fi numărate împreună pentru această temă).

5.3.3 QuickSort și HeapSort: analiză comparativă a cazului mediu statistic (2p)

Se cere compararea celor două metode de sortare în cazul **mediu statistic**. Pentru cazul **mediu statistic** va trebui să repetați măsurările de m ori ($m=5$) și să raportați valoarea lor medie; de asemenea, pentru cazul **mediu statistic**, asigurați-vă că folosiți **aceleasi** date de intrare pentru cele două metode.

Generați un grafic ce compară cele două metode de construcție în cazul **mediu statistic** pe baza numărului de operații obținut la pasul anterior.

Dacă o curbă nu poate fi vizualizată corect din cauza că celelalte curbe au o rată mai mare de creștere, atunci plasați noua curbă pe un alt grafic. Denumiți adekvat graficele și curbele.

5.3.4 Implementarea hibridizării quicksort-ului (1p)

Demo: Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

5.3.5 Determinare a unui prag optim în hibridizare din punct de vedere a numărului de operații și a timpului efectiv de execuție + motivație (1p)

Determinarea optimului din perspectiva pragului utilizat se realizează prin variarea valorii de prag pentru care se aplică insertion sort.

Comparați rezultatele obținute din perspectiva performanței (timpului de execuție și a numărului de operații) pentru a determina o valoare optimă a pragului. Puteti folosi 10,000 ca dimensiune fixă a vectorului ce urmează să fie sortat și variați pragul între [5,50] cu un increment de 1 până la 5.

Numărul de teste care trebuie să fie repete (nr_tests din exemplul de mai sus) trebuie ales în funcție de procesor și modul de compilare. Sugerăm valori mai mari, precum 100 sau 1000.

5.3.6 Analiză comparativă (între *quicksort* și *quicksort hibridizat*) din punct de vedere a numărului de operații și a timpului efectiv de execuție (1p)

Pentru hibridizare quicksort, trebuie să utilizați versiunea iterativă de insertion sort din prima temă în cazul în care dimensiunea vectorului este mică (sugerați utilizarea insertion sort dacă vectorul are sub 30 de elemente - dar este obligatoriu de găsit valoarea potrivită prin analiză). Comparați *timpul de rulare și numărul de operații* (asignări + comparații) pentru quicksort implementat în tema 3 cu cel hibridizat.

5.3.7 Bonus: QuickSelect - Randomized-Select (0.5p)

Demo: Corectitudinea algoritmului va trebui exemplificată pe date de intrare de dimensiuni mici.

Pentru QuickSelect (Randomized-Select) nu trebuie facuta analiza complexității, doar corectitudinea trebuie exemplificată.

5.4 Informatii aditionale

! Doar atribuirile (=) și comparațiile (<, ==, >, !=) care se fac pe datele de intrare și pe datele auxiliare corespunzătoare se iau în considerare.

Pentru a măsura timpul de execuție puteți folosi Profiler similar cu exemplul de mai jos.

```
profiler.startTimer("your_function", current_size);
for(int test=0; test<nr_tests; ++test) {
    your_function(array, current_size);
}
profiler.stopTimer("your_function", current_size);
```

Numărul de teste (*nr_tests* din exemplul de mai sus) trebuie ales în funcție de procesor și modul de compilare. Sugerați valori mai mari, precum 100 sau 1000.

În momentul în care măsurăți timpul de execuție, asigurați-vă că opriți orice alte procese care nu sunt necesare.

! *Observație.* Pentru a evalua cât mai corect timpul, Profiler nu va face numărătoarea operațiilor. Astfel, evaluarea de timp și de operații trebuie făcute separat.

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

6 Tema Nr. 4: Interclasarea eficientă a k liste ordonate

Timp alocat: 2 ore

6.1 Implementare

Se cere implementarea **corectă și eficientă** a unei metode de complexitate $O(n \log k)$ pentru **interclasarea a k liste sortate**. Unde n este numărul total de elemente (Sugestie: folosiți un heap, vezi notițele de la *Seminarul al 2-lea*).

Cerințe de implementare:

- Folosiți liste înlățuite pentru a reprezenta cele k secvențe sortate și secvența de ieșire

Intrare: k șiruri de numere $\langle a_1^i, a_2^i, \dots, a_{m_i}^i \rangle$, $\sum_{i=1}^k m_i = n$

Ieșire: o permutare a reuniunii șirurilor de la intrare $a'_1 \leq a'_2 \leq \dots \leq a'_n$

6.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumea voastră la întrebările puse de către profesor.*

6.3 Cerințe

6.3.1 Demo pentru generarea a k liste aleatoare sortate de dimensiuni diferite (având în total n elemente, unde n și k sunt date) și interclasarea a 2 liste (5p)

Demo: Corectitudinea algoritmului (*generare și interclasare*) va trebui demonstrată pe date de intrare de dimensiuni mici. Generați liste de dimensiuni $k=4$,

$n=20$ și interclasăți două dintre acestea.

6.3.2 Adaptare operațiilor de *min-heap* pe structura nouă și interclasarea a k liste (3p)

Demo: Corectitudinea algoritmului de (*interclasare*) va trebui demonstrată pe date de intrare de dimensiuni mici (ex: $k=4$, $n=20$). În cazul în care ați ajuns la acest punct, demostrația de interclasare de la primul demo nu este necesară, dar generarea este.

6.3.3 Evaluarea algoritmului în cazul mediu statistic (2p)

Se cere analiza algoritmului în cazul **mediu statistic**. Pentru cazul **mediu statistic** va trebui să repetați măsurările de câteva ori. Din moment ce k și n pot varia, se va face o analiză în felul următor:

- Se alege, pe rând, 3 valori constante pentru k ($k1=5$, $k2=10$, $k3=100$); generează k siruri **aleatoare** sortate pentru fiecare valoare a lui k astfel încât numărul elementelor din toate sirurile să varieze între **100 și 10000** cu un increment maxim de 400 (sugerăm 100); rulați algoritmul pentru toate valorile lui n (pentru fiecare valoare a lui k); generați un grafic ce reprezintă **suma atribuirilor și a comparațiilor** făcute de acest algoritm pentru fiecare valoare a lui k (în total sunt 3 curbe).
- Se alege $n=10.000$; valoarea lui k va varia între **10 și 500** cu un increment de 10; generați k siruri **aleatoare** sortate pentru fiecare valoare a lui k astfel încât numărul elementelor din toate sirurile să fie 10000; testați algoritmul de interclasare pentru fiecare valoare a lui k și generați un grafic care reprezintă **suma atribuirilor și a comparațiilor**.

7 Tema Nr. 5: Căutarea în tabele de dispersie

Adresare deschisă prin verificare pătratică

Timp alocat: 2 ore

7.1 Implementare

Se cere implementarea **corectă și eficientă** a operațiilor de *inserare* și *căutare* într-o tabelă de dispersie ce folosește *adresarea deschisă cu verificare pătratică*.

Informații utile și pseudo-cod găsiți în notițele de curs sau în carte ([1]), în secțiunea 11.4 *Open addressing*.

Noțiunile închis/deschis (closed/open) specifică dacă se obligă folosirea unei poziții sau structuri de date.

7.1.1 Hashing (se referă la tabela de dispersie (hash table))

- Open Hashing: pe o anumită poziție se pot stoca mai multe elemente (ex: chaining)
- Closed Hashing: se poate stoca doar un singur element pe o anumită poziție (ex. linear/quadratic probing)

Addressing (se referă la poziția finală a unui element față de poziția inițială)

- Open Addressing (Adresare Deschisă): adresa finală (poziția finală) nu este complet determinată de către codul hash. Poziția depinde și de elementele care sunt deja în tabelă. (ex: linear/quadratic probing - verificare liniară/pătratică)
- Closed Addressing (Adresare Închisă): adresa finală este întotdeauna determinată de codul hash (poziția inițială calculată) și nu există probing (ex: chaining)

Pentru această temă tabela de dispersie va avea o structură similară cu cea de mai jos:

```
typedef struct {
    int id;
    char name[30];
} Entry;
```

unde *adresa finală* în tabelă va fi calculată aplicând funcția de hashing pe câmpul *id* din structură. Câmpul *name* va fi folosit doar pentru demonstrarea corectitudinii operațiilor de căutare și stergere și nu este necesară pentru evaluarea performanței (câmpul *name* va fi afișat în consolă dacă operația de căutare găsește *id*-ul respectiv, altfel afișați “negăsit”).

7.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minime (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu preluăm teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumeavosă la întrebările puse de către profesor.*

7.3 Cerințe

7.3.1 Implementarea operației de inserare și căutare folosind structura de date cerută + demo (dimensiune 10) (5p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (ex. 10).

7.3.2 Evaluarea operației de căutare pentru un singur factor de umplere 95% (2p)

Se cere evaluarea operației de *căutare* în tabele de dispersie cu adresare deschisă și verificare pătratică, în cazul **mediu statistic** (nu uitați să repetați măsurătorile de 5 ori). Pentru a obține evaluarea, trebuie să:

1. Alegeți N , dimensiunea tabelei, un număr prim în jur de 10000 (e.g. 9973, sau 10007);
2. Pentru fiecare din următoarele valori pentru factorul de umplere $\alpha = 0.95$:
 - a. Inserați în tabela n elemente aleator, astfel încât să ajungeți la valoare lui α ($\alpha = n/N$)
 - b. Căutați aleator, în fiecare caz, m elemente ($m = 3000$), astfel încât aproximativ jumătate din elemente să fie *găsite*, iar restul să *nu fie găsite* (în tabelă). *Asigurați-vă că elementele găsite sunt generate*

uniform, i.e. să căutați elemente care au fost introduse la moment diferite, cu probabilitate egală (există mai multe moduri în care se poate garanta acest lucru)

- c. Numărați operațiile efectuate de procedura de căutare (i.e. numărul de celule accesate)
- d. Atenție la valorile pe care le căutați, să fie într-o ordine aleatoare din cele introduse. *Dacă sunt primele 1500 adăugate, implicit efortul mediu găsite va fi 1.*

3. Generați un tabel de forma:

Table 1: Effort measurements at various filling factors

Filling factor	Avg. Effort (found)	Max Effort (found)	Avg. Effort (not-found)	Max Effort (not-found)
0.95

$$\text{Efort mediu} = \text{efort_total} / \text{nr_elemente}$$

$$\text{Efort maxim} = \text{număr maxim de accese efectuat de o operație de căutare}$$

7.3.3 Completarea evaluării pentru toți factorii de umplere (2p)

Respectând cerințele de la punctul 2 cu $\alpha \in \{0.8, 0.85, 0.9, 0.95, 0.99\}$, generați un tabel de forma:

Table 2: Effort measurements at various filling factors

Filling factor	Avg. Effort (found)	Max Effort (found)	Avg. Effort (not-found)	Max Effort (not-found)
0.80				
0.85				
...

7.3.4 Implementare operației de ștergere într-o tabelă de dispersie și evaluarea operației de căutare după ștergerea unor elemente (1p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (ex. 10).

Pentru evaluarea operației de căutare după stergere, umpleți tabela de dispersie până la factor de umplere 0.99. Stergeți elemente din tabela până ajungeți la factor umplere 0.8, după care căutați m elemente aleator ($m \sim 3000$), astfel încât aproximativ jumătate din elemente să fie *găsite*, iar restul să nu fie găsite (în tabelă). Măsurăți efortul necesar pentru *căutare* și adăugați în tabelul generat anterior.

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

8 Tema Nr. 6: Arbori Multicăi

Transformări între diferite reprezentări

Timp alocat: 2 ore

8.1 Implementare

1. Se cere implementarea **corectă** și **eficientă** a traversării *iterative* și *recursivă* a unui arbore binar. Puteți găsi orice informații necesare și pseudocod în notele de curs și seminar.
2. În plus, se cere implementarea **corectă** și **eficientă** a unor algoritmi de complexitate *liniară* pentru transformarea arborilor multicăi între următoarele reprezentări:
 1. **R1:** *reprezentarea părinte*: pentru fiecare index, valoare din vector reprezentă indexul părintele, ex: $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$
 2. **R2:** *reprezentare arbore multicăi*: fiecare nod conține cheia și un vector de noduri copil
 3. **R3:** *reprezentare binara*: fiecare nod conține cheia și doi pointeri: unul către primul copil și al doilea către fratele din dreapta (ex: următorul frate).

Așadar, trebuie să definiți transformarea **T1** din *reprezentarea părinte* (**R1**) în *reprezentarea arbore multicăi* (**R2**), iar apoi transformarea **T2** din *reprezentarea arbore multicăi* (**R2**) în *reprezentarea binară* (**R3**). Pentru toate reprezentările (**R1**, **R2**, **R3**) trebuie să implementați afișarea prietenoasă (pretty print, **PP**) (vezi pagina 2).

Definiți structurile de date. Puteți folosi structuri intermediare (ex: memorie adițională).

8.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.

- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerintei, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumeavostră la întrebările puse de către profesor.*

8.3 Cerințe

8.3.1 Implementare a parcurgerii iterative și recursive a unui arbore binar în $O(n)$ și cu memorie aditională constantă (3p)

Demo: Corectitudinea algoritmilor va trebui exemplificată pe date de intrare de dimensiuni mici.

8.3.2 Implementarea corectă la pretty-print la $R1$ (2p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe exemplul $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$.

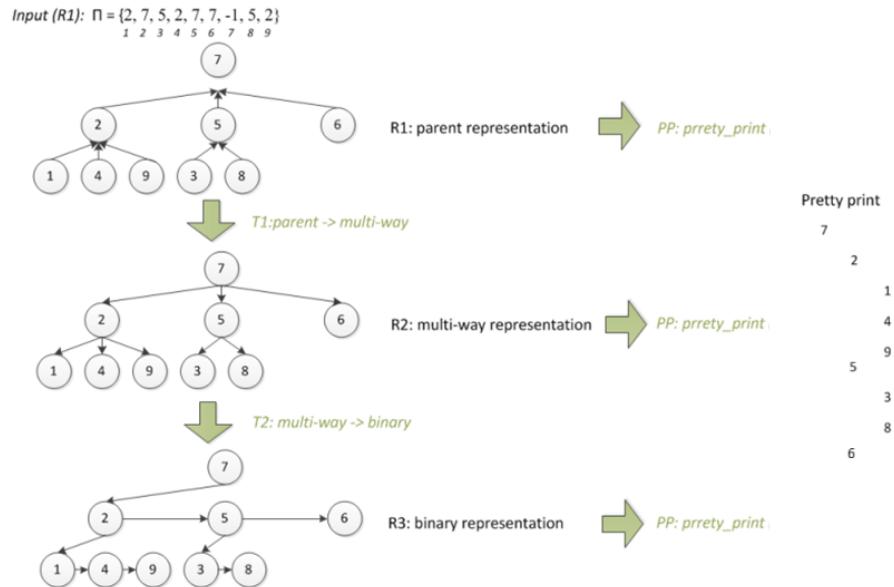
8.3.3 Implementarea corectă la $T1$ (din $R1$ în $R2$) și pretty-print la $R2$ (1p) + $T1$ în timp liniar (1p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe exemplul $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$.

8.3.4 Implementarea corectă la $T2$ (din $R2$ în $R3$) și pretty-print la $R3$ (2p) + $T2$ în timp liniar (1p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe exemplul $\Pi = \{2, 7, 5, 2, 7, 7, -1, 5, 2\}$.

Folosiți afișarea prietenoasă pentru cele trei reprezentări. *Fiecare reprezentare ($R1, R2, R3$) necesită o afișare prietenoasă cu o implementare diferită dar aceeași afișare.*



Analizați eficiența în timp și spațiu a celor două transformări. Ați atins $O(n)$? Ați folosit memorie adițională?

9 Tema Nr. 7: Statistici dinamice de ordine

Timp alocat: 2 ore

9.1 Implementare

Se cere implementarea **corectă și eficientă** a operațiilor de management ale unui **arbore de statistică de ordine** (*capitolul 14 din [1]*).

Se cere să folosiți un *arbore binar de căutare perfect echilibrat*. Fiecare nod din arbore trebuie extins cu un câmp *size* (dimensiunea sub-arborelui ce are nodul ca rădăcină).

Operațiile de management ale unui **arbore de statistică de ordine**:

- BUILD_TREE(n)
 - construiește un arbore binar de căutare **echilibrat** cu cheile $1, 2, \dots, n$
(hint: divide et impera)
 - nu uitați să inițializați câmpul *size*
- OS-SELECT(tree, i)
 - selectează elementul cu a i -a cea mai mică cheie
 - pseudocodul poate fi găsit la *Capitolul 14.1 din [1]*
- OS-DELETE(tree, i)
 - puteți folosi ștergerea dintr-un arbore binar de căutare, fără a crește înălțimea arborelui (De ce nu trebuie să re-balansați arborele?)
 - nu uitați să păstrați câmpul *size* consistent o dată cu ștergerile din arbore
 - există mai multe abordări prin care puteți modifica câmpul *size* fără a crește complexitatea algoritmului (găsiți cea mai bună soluție)

Seamănă OS-SELECT cu ceva ce ați studiat în acest semestru?

9.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.

- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerintei, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumea vostră la întrebările puse de către profesor.*

9.3 Cerințe

9.3.1 BUILD_TREE: implementare corectă și eficientă (5p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

- afișați (cu pretty print) arborele construit inițial

9.3.2 OS_SELECT: implementare corectă și eficientă (1p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

- executați OS-SELECT pentru câțiva (cel puțin 3) indecsi selectați aleator.

9.3.3 OS_DELETE: implementare corectă și eficientă (2p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

- executați secvența OS-SELECT urmat de OS-DELETE pentru câțiva (cel puțin 3) indecsi selectați aleator (3) și afișați arborele după fiecare execuție.

9.3.4 Evaluarea operațiilor de management - BUILD, SELECT, DELETE (2p)

! Înainte de a începe să lucrați pe partea de evaluare, asigurați-vă că aveți un **algoritm corect!**

După ce sunteți siguri că algoritmul funcționează corect:

- variați n de la 100 la 10000 cu un pas de 100;
- pentru fiecare n (nu uitați să repetați de 5 ori)
 - construiți (BUILD) arborele cu elemente de la 1 la n
 - repetați de n ori secvența OS-SELECT urmat OS-DELETE folosind un index selectat aleator dintre elementele rămasse în arbore

- Evaluati numărul de operații necesare pentru fiecare operație de management (BUILD, SELECT, DELETE – *reprezentați rezultatele sub forma unui grafic cu trei serii*). Evaluati complexitatea operațiilor de management ca și suma atribuirilor și a comparațiilor pentru fiecare valoare a lui n .

9.3.5 Bonus: Implementarea utilizând AVL / arbori roșu și negru (1p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici (11)

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

10 Tema Nr. 8: Multimi disjuncte

Timp alocat: 2 ore

10.1 Implementare

Se cere implementarea **corectă și eficientă** a operațiilor de bază pe **multimi disjuncte** (*capitolul 21.1* din [1]) și a algoritmului lui **Kruskal** (găsirea arborelui de acoperire minimă, *capitolul 23.2* din [1]) folosind multimi disjuncte.

Se cere să folosiți o pădure de arbori pentru reprezentarea multimilor disjuncte. Fiecare arbore trebuie extins cu un câmp *rank* (înălțimea arborelui).

Operațiile de bază pe **multimi disjuncte** sunt:

- **MAKE_SET (x)**
 - creează o mulțime nouă ce conține elementul *x*
- **UNION (x, y)**
 - realizează reuniunea dintre mulțimea care îl conține pe *x* și mulțimea care îl conține pe *y*
 - euristică *union by rank* ține cont de înălțimea celor doi arbori pentru a realiza reuniunea dintre mulțimi
 - pseudocodul poate fi găsit la *capitolul 21.3*[1]
- **FIND_SET (x)**
 - cauță mulțime în care se află *x*
 - euristică *path compression* leagă toate elementele de pe ramura cu *x* la rădăcina arborelui

10.2 Cerințe minime pentru notare

Lipsa oricărei cerințe minime (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.

- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumeavostră la întrebările puse de către profesor.*

10.3 Cerințe

10.3.1 Implementare corectă a MAKE_SET, UNION și FIND_SET (5p)

Demo: Corectitudinea algoritmilor va trebui demonstrată pe date de intrare de dimensiuni mici.

- creați (MAKE) 10 multimi + afișare conținuturilor seturilor
- execuțați secvența UNION și FIND_SET pentru 5 elemente + afișare conținuturilor seturilor

10.3.2 Implementarea corectă și eficientă a algoritmului lui Kruskal (2p)

Demo: Corectitudinea algoritmului va trebui demonstrată pe date de intrare de dimensiuni mici.

- creați un graf cu 5 noduri și 9 muchii + **afișare muchii**
- aplicarea algoritmului lui Kruskal + **afișarea muchiilor alese**

10.3.3 Evaluarea operațiilor pe multimi disjuncte (MAKE, UNION, FIND) folosind algoritmului lui Kruskal (3p)

! Înainte de a începe să lucrăți pe partea de evaluare, asigurați-vă că aveți un **algoritm corect!**

O dată ce sunteți siguri că algoritmul funcționează corect:

- variați n de la 100 la 10000 cu un pas de 100
- pentru fiecare n
 - construiți un graf **conex, neorientat și aleatoriu** cu ponderi pe muchii (**n** noduri, n^*4 muchii)
 - determinați arborele de acoperire minima folosind algoritmul lui Kruskal
 - * evaluați efortul computațional al fiecărei operații de bază (MAKE, UNION, FIND – *reprezentăți rezultatele sub forma unui grafic cu trei serii*) pe multimi disjuncte ca suma comparațiilor și atribuțiilor efectuate; astfel, ar trebui să existe **3 serii în grafic**, câte una pentru fiecare operație.

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

11 Tema Nr. 9: Căutarea în lățime (BFS)

11.1 Introducere

În acest laborator se va implementa algoritmul BFS (căutarea în lățime), conform secțiunii 22.2 din [1].

11.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului `main.cpp`, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în `main`).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul correct dat de dumea voastră la întrebările puse de către profesor.*

11.3 Structura proiectului

La acest laborator veți porni de la 3 fișiere:

- `main.cpp` - sursa principală, responsabilă cu interfața de vizualizare
- `bfs.h` - definiții pentru tipuri de structuri și funcții
- `bfs.cpp` - implementarea algoritmilor
- `grid.txt` - labirintul care va fi afișat și traversat
- `Profiler.h` - biblioteca pentru numărarea operațiilor și pentru grafice

!!! Pentru rezolvarea cerințelor trebuie să faceți modificări doar în `bfs.cpp`. Pentru a vizualiza mai ușor funcționarea algoritmului, `main.cpp` va afișa o interfață de tip text, în care se va vedea o un labirint ce trebuie traversat (celulele negre sunt libere iar cele albe sunt perete).

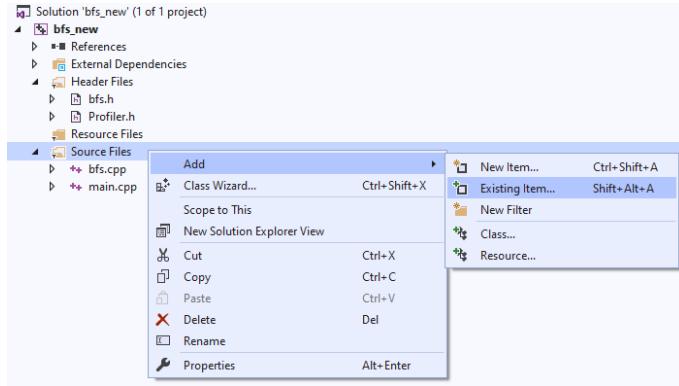


Figure 1: Fereastra “*Solution Explorer*” în Visual Studio

11.3.1 Inițializarea proiectului pe Windows, cu Visual Studio

Creați un proiect nou în Visual Studio, de tipul “*Empty Project*” și copiați fișierele de mai sus în folderul proiectului.

Adăugați cele două fișiere .h la secțiunea “*Header Files*” și cele două fișiere .cpp la secțiunea “*Source Files*”, efectuând click dreapta → “*Add*” → “*Existing Item*”, ca în Figura 1.

11.3.2 Inițializarea proiectului pe Linux și Mac

Puteți edita fișierele proiectului cu orice editor doriti. Proiectul conține și un fișier **Makefile**, deci este suficient să rulați comanda **make** pentru compilarea acestuia. Executabilul .exe rezultat se va numi **main**, și se poate rula în terminal, executând ./main.

11.4 Funcționare

La pornirea programului se va afișa labirintul, în mod similar cu Figura 2.

În partea de jos, utilizatorul poate tăsi una din comenziile de mai jos:

- **exit**
terminarea programului
- **clear**
curățarea informațiilor anterioare din grilă
- **neighb <row> <col>**
se vor afișa vecinii celulei de pe linia <row> și coloana <col>.
- **bfs <row> <col>**
se va efectua o parcurgere BFS, pornind de la celula de la linia <row> și coloana <col>.

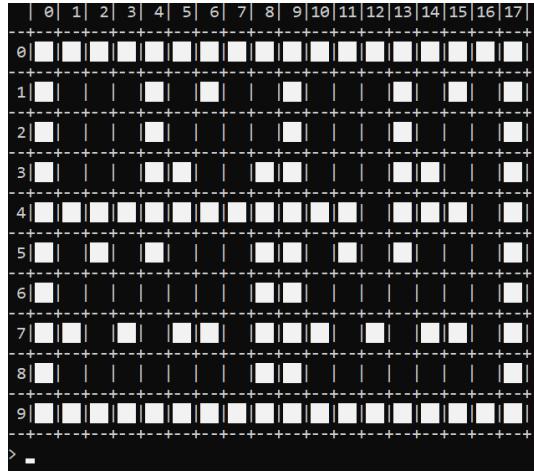


Figure 2: Interfața programului

- **bfs_step <row> <col>**
la fel ca la **bfs**, dar rezultatul se va afișa pas cu pas, în funcție de distanță față de sursă
- **bfs_tree <row> <col>**
la fel ca la **bfs**, dar se va afișa și arborele BFS sub grilă
- **path <row1> <col1> <row2> <col2>**
se va afișa cel mai scurt drum între celulele (<row1> <col1>) și (<row2> <col2>)
- **perf**
se vor genera graficele pentru evaluarea performanței algoritmului

11.4.1 Exemplu: comanda `neighb`

Dacă se rulează:

`neighb 2 3` ar trebui să apară imaginea din Figura 3.

Celula de start se va colora cu verde, iar vecinii acesteia cu albastru.

În momentul de față funcția `get_neighbours()` nu este implementată, deci nu se va afișa rezultatul dorit. Puteți verifica dacă ați implementat corect această funcție rulând comanda pentru diverse celule. Fiecare celulă va avea maxim 4 vecini (sus, jos, stânga, dreapta), și nu trebuie afișate celule din afara grid-ului sau celule care conțin pereți.

11.4.2 Exemplu: comenziile `bfs` și `bfs_step`

Dacă se rulează:

`bfs 6 3` ar trebui să apară imaginea din Figura 4.

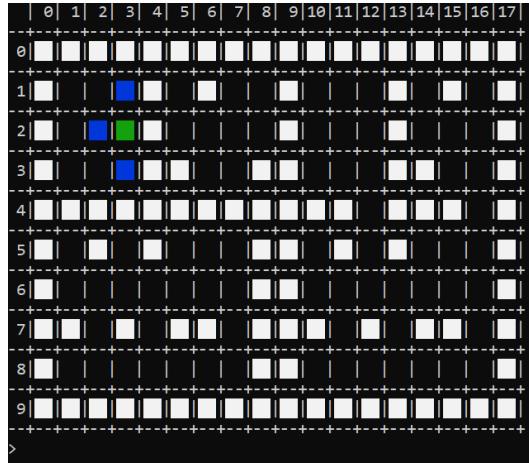


Figure 3: Rezultatul comenzi `neighb 2 3`

Celula de start se va colora cu verde, iar celulele parcurse se vor colora cu albastru. Pe fiecare celulă albastră va apărea o săgeată care va indica în ce direcție se află părintele din arborele BFS.

În momentul de față funcția `bfs()` nu este implementată, deci nu se va afișa rezultatul dorit. Puteți verifica dacă ați implementat corect această funcție rulând comanda pentru diverse celule.

11.4.3 Exemplu: comanda `bfs_tree`

Dacă se rulează:

`bfs 2 6` ar trebui să apară imaginea din Figura 5.

Rădăcina arborelui este nodul de start, respectiv (2, 6). Copii acestui nod, sunt nodurile în care se poate ajunge direct din rădăcină: (2, 5), (2, 7) și (3, 6) (ordinea acestora poate să difere în altă implementare).

11.4.4 Exemplu: comanda `path`

Dacă se rulează:

`path 5 10 3 15` ar trebui să apară imaginea din Figura 6.

Celula de start se va colora cu verde, cea de final cu roșu, iar celulele care fac parte din drumul cel mai scurt se vor colora cu albastru. Pe fiecare celulă albastră va apărea o săgeată care va indica direcția de mers.

În momentul de față funcțiile `shortest_path()` și `bfs()` nu sunt implementate, deci nu se va afișa rezultatul dorit. Puteți verifica dacă ați implementat corect aceste funcții rulând comanda pentru diverse perechi de celule.

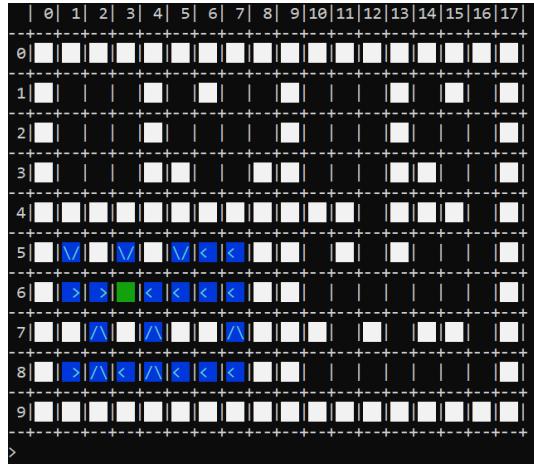


Figure 4: Rezultatul comenzi `bfs 6 3`

11.4.5 Structuri de date folosite

În fișierul `bfs.h` sunt definite câteva structuri de date utile în cadrul framework-ului.

Structura `Grid` modelează o grilă, formată din `rows` linii și `cols` coloane, elementele acesteia fiind în matricea `mat`. O celulă liberă va avea valoarea 0, iar una ce conține un perete va avea valoarea 1.

Structura `Point` modelează un punct sau o celulă din grilă, câmpurile `row` și `col` reprezentând linia și coloana la care se află.

Structura `Node` modelează un nod din graf și conține următoarele câmpuri:

- `position` de tip `Point` reprezintă celula din grilă corespunzătoare nodului.
- `adjSize` - numărul de vecini ai nodului respectiv
- `adj` - vectorul de vecini, de dimensiune `adjSize`
- `color` - culoarea nodului; la început toate nodurile au culoarea `COLOR_WHITE`, adică valoarea 0
- `dist` - distanța față de nodul de start, în parcurgerea BFS
- `parent` - pointer la nodul părinte, în arborele BFS

Structura `Graph` modelează un graf și conține numărul de noduri `nrNodes` și vectorul `v` cu pointeri spre acestea.

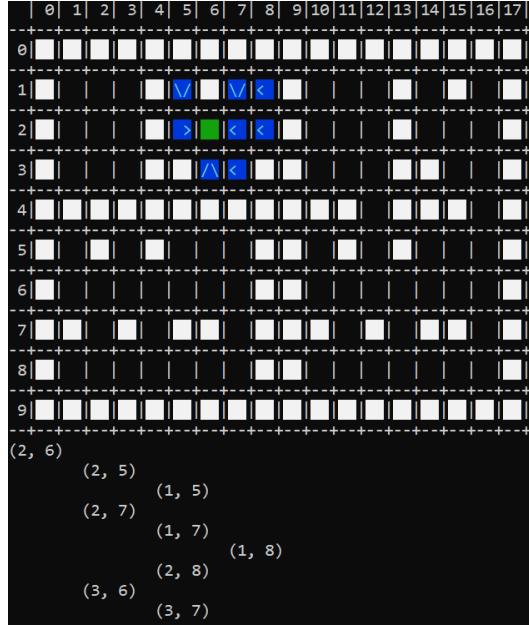


Figure 5: Rezultatul comenzi `bfs.tree 2 6`

11.5 Cerinte

11.5.1 Determinarea vecinilor unei celule (2p)

În `bfs.cpp`, trebuie completată funcția `get_neighbors()` care primește ca parametri un pointer la structura de tip `Grid`, un punct `p` de tip `Point` și un vector de puncte `neighb` care se va completa cu vecinii punctului `p`. Funcția va returna numărul de vecini completăți în vectorul `neighb`.

Un punct din grilă va avea maxim 4 vecini (sus, jos, stânga, dreapta). Nu toți vecinii sunt valizi: unii vecini pot ajunge în afara grilei (coordonate negative, sau peste dimensiuni) sau pot fi în interiorul unui zid. Din acest motiv, după ce calculați poziția unui vecin, ar trebui să verificați că aceasta cade în interiorul grilei, apoi că aceasta este liberă (valoarea din matrice la poziția respectivă e 0).

Vecinii valizi se vor completa în vectorul `neighb`. Se garantează că la apelul funcției din framework, acesta va avea cel puțin 4 elemente, deci nu se poate depăși capacitatea acestuia. Deoarece numărul de vecini completăți poate fi mai mic de 4, trebuie să returnăm numărul acestora.

11.5.2 Implementarea algoritmului BFS (3p)

În `bfs.cpp`, trebuie completată funcția `bfs()` care primește ca parametri un pointer la structura de tip `Graph` și nodul de start `s` de tip `Node*`. Funcția va aplica algoritmul BFS conform secțiunii 22.2 din [1].

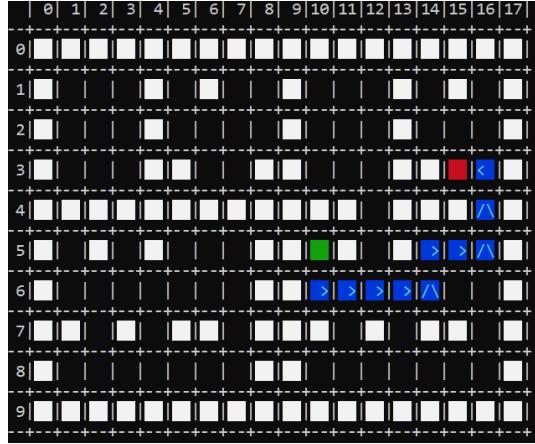


Figure 6: Rezultatul comenzi path 5 10 3 15

Nodurile din graf au la început culoarea `COLOR_WHITE`, iar câmpurile `dist` și `parent` sunt inițializate cu 0, respectiv `NULL`. După parcurgere, toate nodurile la care se poate ajunge din nodul de start trebuie să aibă culoarea `COLOR_BLACK`, distanța `dist` setată pe numărul de pași de la nodul de start până la nodul respectiv, iar pointerul `parent` trebuie să indice părintele în arborele BFS.

11.5.3 Afișarea arborelui BFS (2p)

În `bfs.cpp`, trebuie completată funcția `print_bfs_tree()` care primește ca parametru un pointer la structura de tip `Graph` pe care s-a rulat deja algoritmul BFS, deci culorile nodurilor și părintii sunt deja setate.

În funcția `data`, este deja implementată construcția unui vector de părinți `p`, în care nodurile colorate cu negru în parcurgerea BFS vor fi numerotate de la 0 la n . Deasemenea se construiește vectorul `repr` care conține coordonatele fiecărui nod.

Pentru afișarea acestui arbore se poate adapta codul din laboratorul de arbori multi-căi.

11.5.4 Evaluarea performanței algoritmului BFS (3p)

Funcția `performance()` realizează numărarea operațiilor, variind pe rând numărul de muchii, respectiv numărul de vârfuri al grafului. Pentru fiecare valoare, trebuie să implementați construcția unui graf aleator, conex, care să aibă numărul respectiv de vârfuri și de muchii.

În interiorul funcției `bfs()` va trebui să implementați numărarea propriu-zisă a operațiilor, folosind parametrul optional `op`. Deoarece acest parametru este optional, uneori funcția `bfs()` va fi apelată din framework cu valoarea acestuia setată pe `NULL`. Din acest motiv, atunci când numărați o operație, verificați tot timpul că `op` e un pointer valid, ca în exemplul de mai jos:

```
if(op != NULL) op->count();
```

11.5.5 Bonus: Determinarea celui mai scurt drum (0.5p)

În `bfs.cpp`, trebuie completată funcția `shortest_path()` care primește ca parametri un pointer la structura de tip `Graph`, nodurile de început și sfârșit `start` și `end` de tip `Node*`, respectiv vectorul `path`, ca parametru de ieșire în care se vor completa nodurile de pe traseu, în ordine. Funcția va returna numărul de noduri completeate în vectorul `path`.

Pentru determinarea celui mai scurt drum între două noduri, se recomandă folosirea algoritmului BFS, implementat anterior, apoi reconstrucția drumului mergând din părinte în părinte în arborele BFS.

Vectorul `path`, în care se completează traseul, va avea o lungime de minim 1000 de elemente, la apelarea funcției. Returnați numărul de elemente care au fost completeate în el, sau -1 în cazul în care nu se poate ajunge la nodul `end` pornind de la nodul `start`.

11.5.6 Bonus: Unde poate ajunge un cal pe tablă? (0.5p)

Folosind framework-ul dat, arătați că un cal poate ajunge pe orice poziție a unei table de șah goale, pornind din colțul din stânga-sus. Dați exemple de tablă care să conțină poziții libere la care nu se poate ajunge.

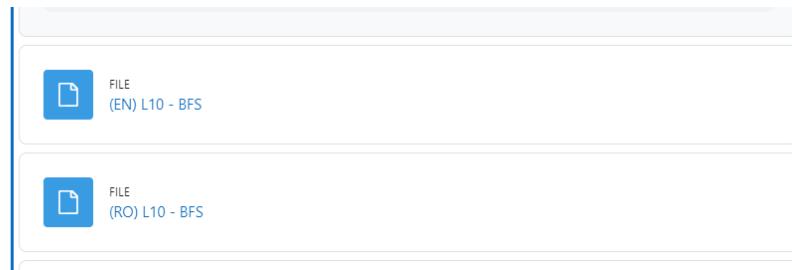
References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

12 Tema Nr. 9: Tutorial

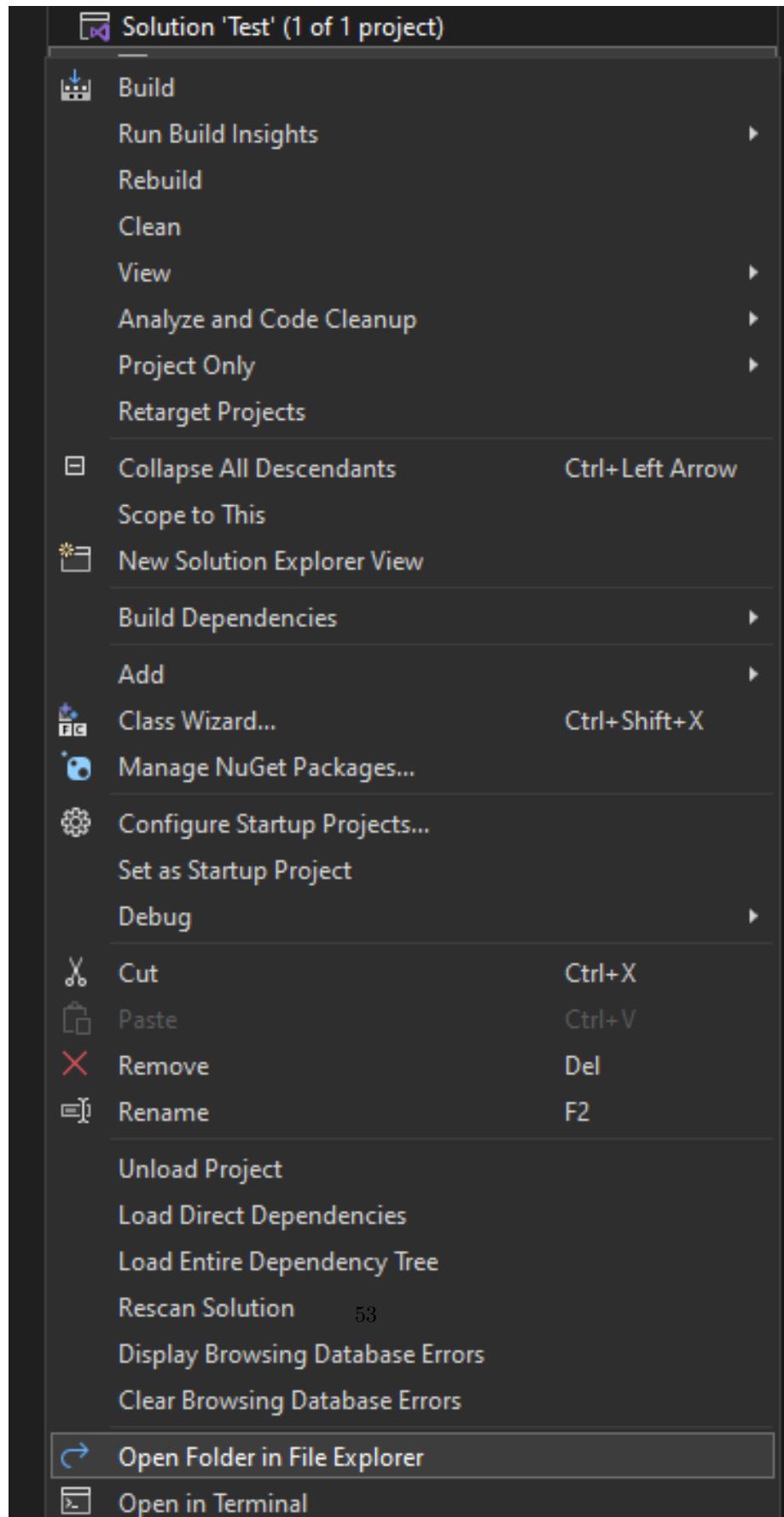
12.1 Configurare Proiect Visual Studio

1. Accesați Moodle și selectați una dintre următoarele:

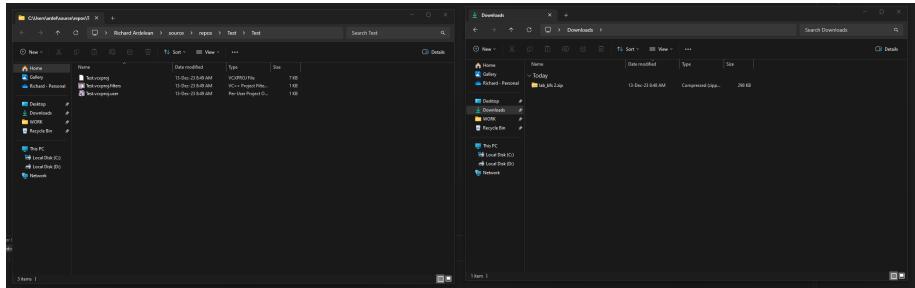


Aceasta va rezulta în descărcarea unui fișier '.zip'.

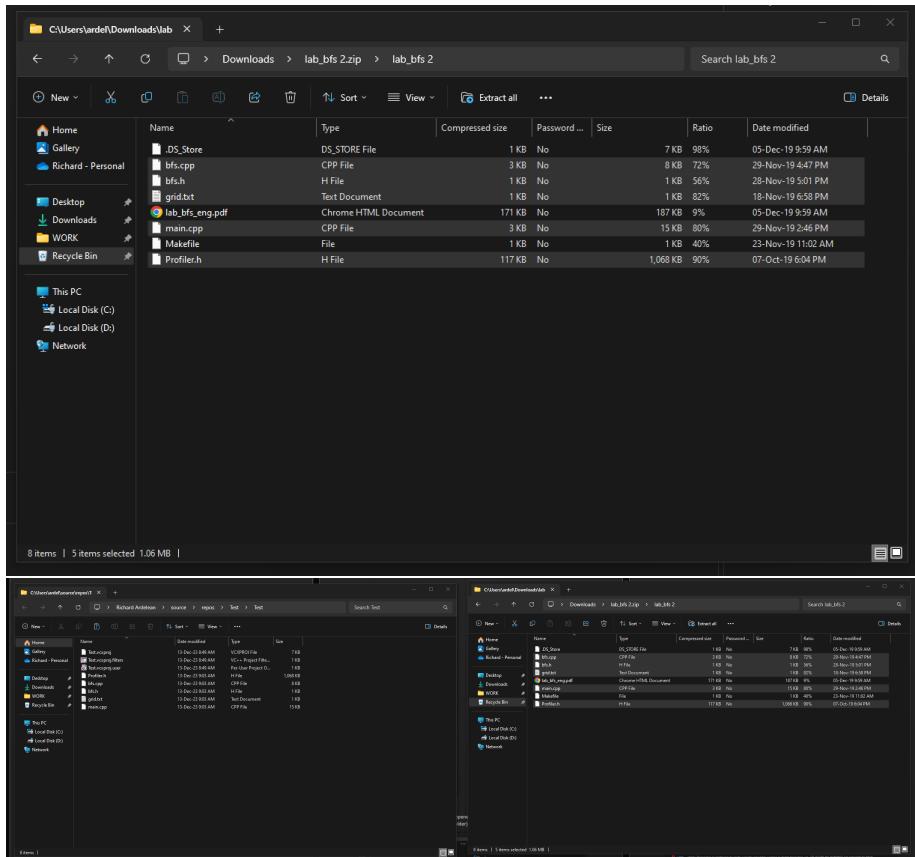
2. Creați un Proiect C++ Gol și făcând clic dreapta pe proiect (nu pe soluție) veți putea selecta ‘Deschideți folderul în File Explorer’.



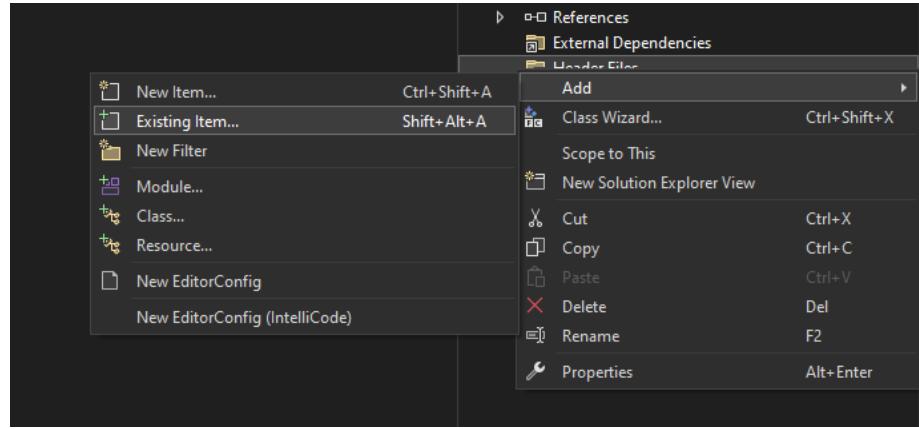
3. În acest moment, se va deschide o fereastră ‘File Explorer’. Deschideți o altă fereastră ‘File Explorer’ la locația descăr cărilor dvs. (cel mai probabil folderul Downloads).



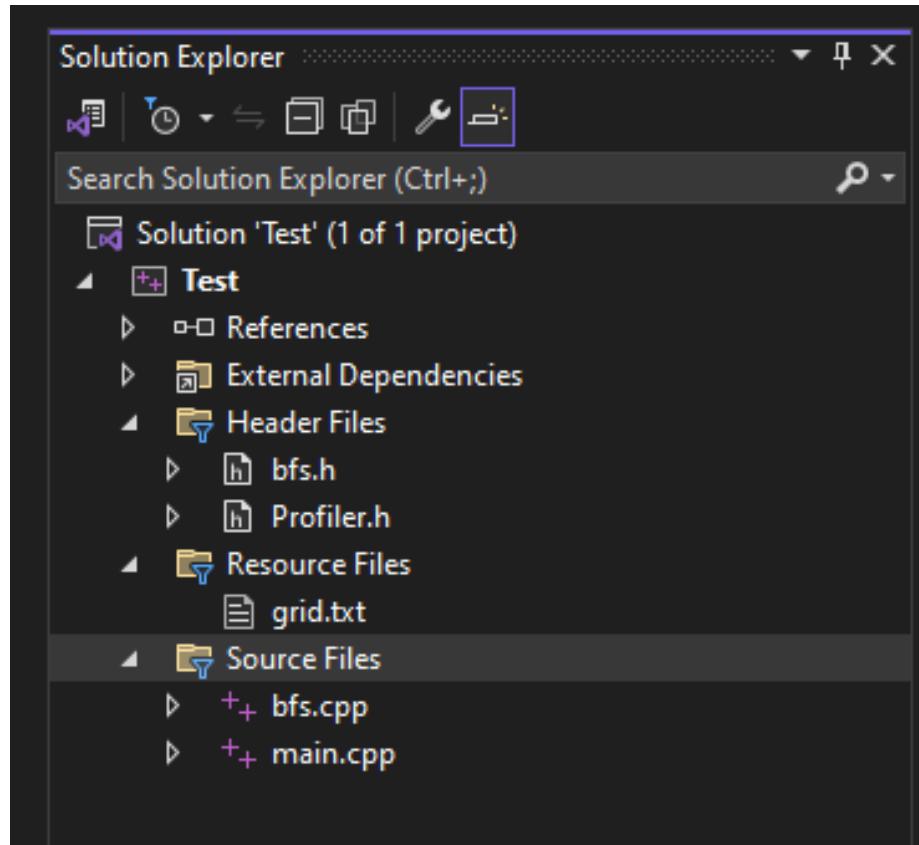
4. Deschideți fisierul ‘.zip’ și copiați, aşa cum se arată mai jos, fișierele din arhivă în folderul proiectului.



5. Selectând folderele din Solution Explorer al Visual Studio ‘Fișiere Header / Resurse / Fișiere Sursă‘ utilizați următoarele opțiuni:

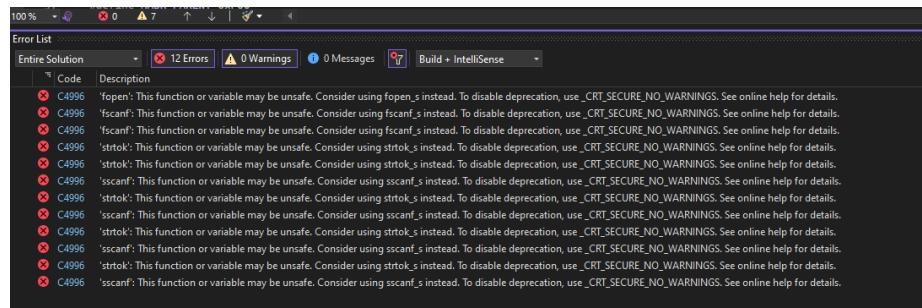


6. Astfel, se realizează următoarea structură:



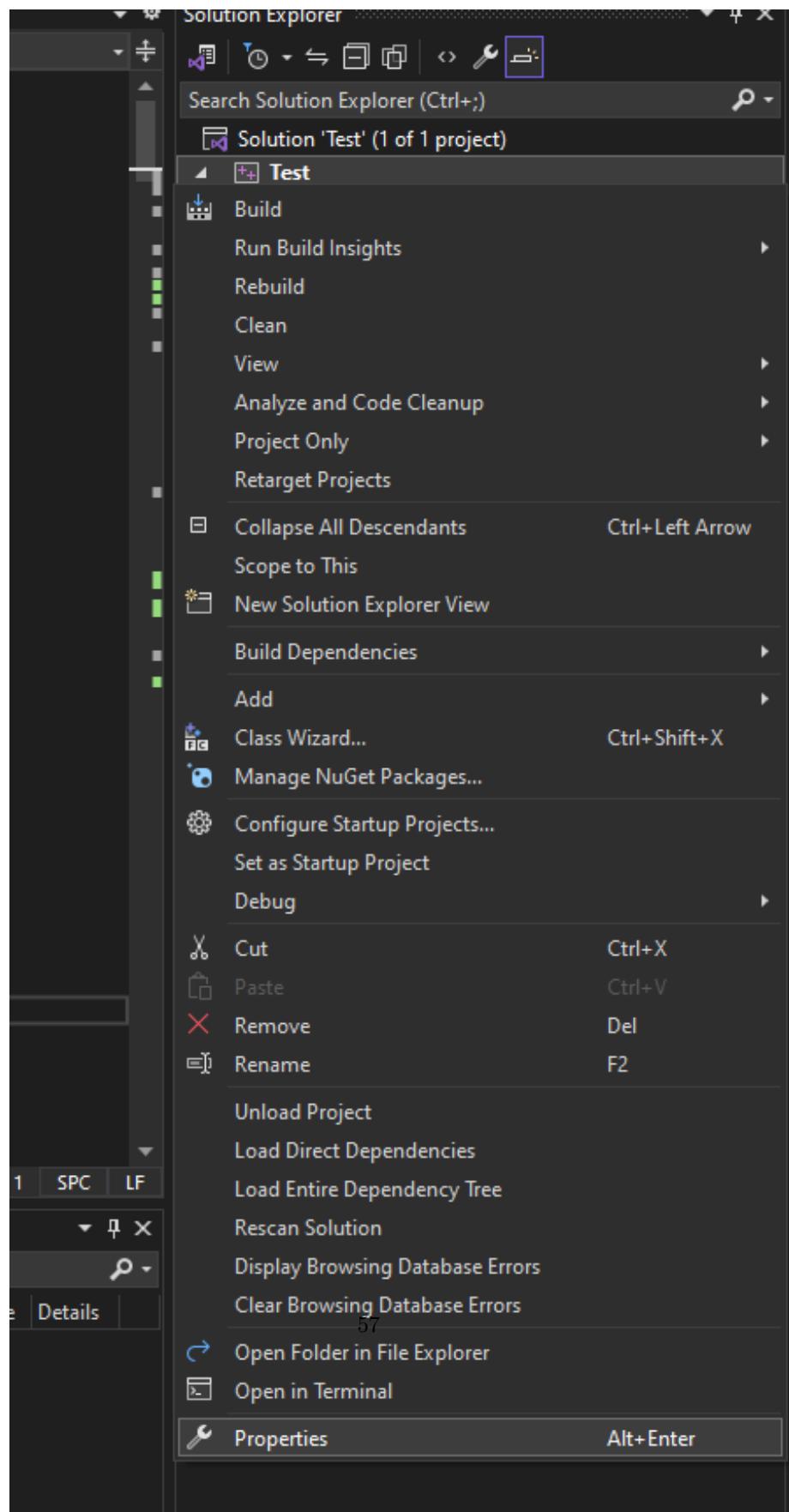
12.2 Eroare ‘unsafe’ în Visual Studio

Exemplu:



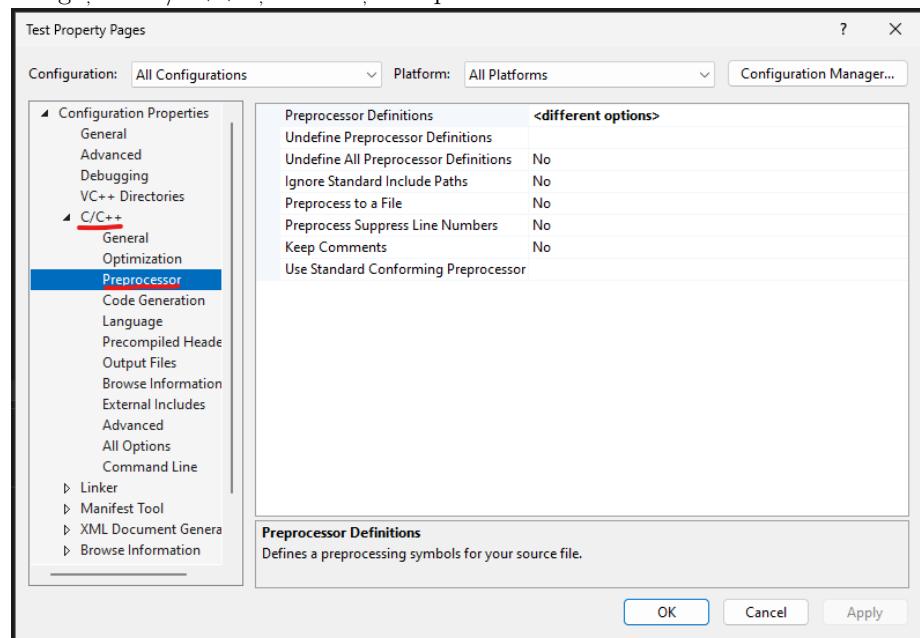
Soluție:

Faceți clic dreapta pe proiect (nu pe Soluție, care cel mai probabil are același nume) și selectați ‘Proprietăți’.

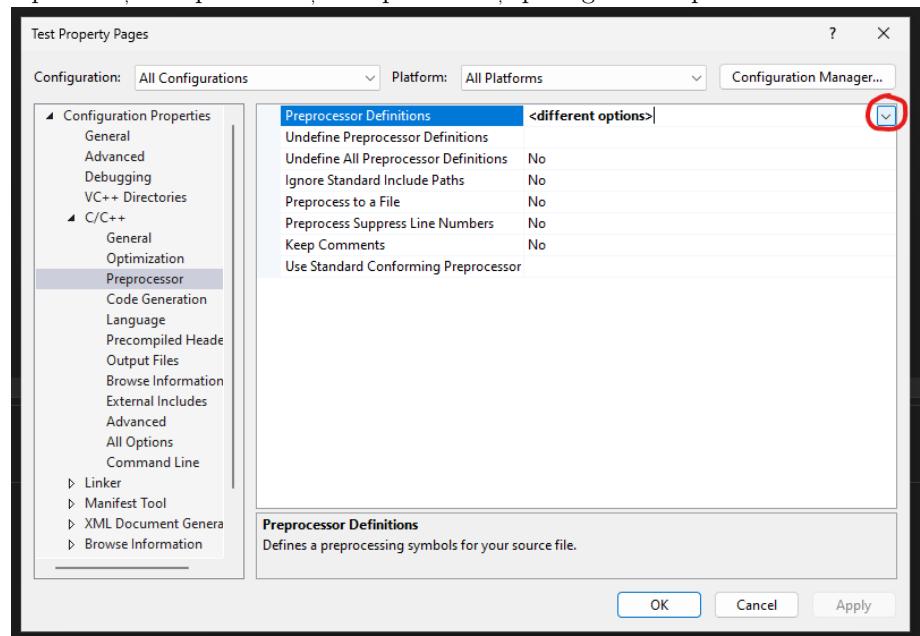


Actualizați ‘Configurație’ și ‘Platformă’ în partea superioară a ferestrei la ‘Toate configurațiile’ și ‘Toate platformele’, respectiv.

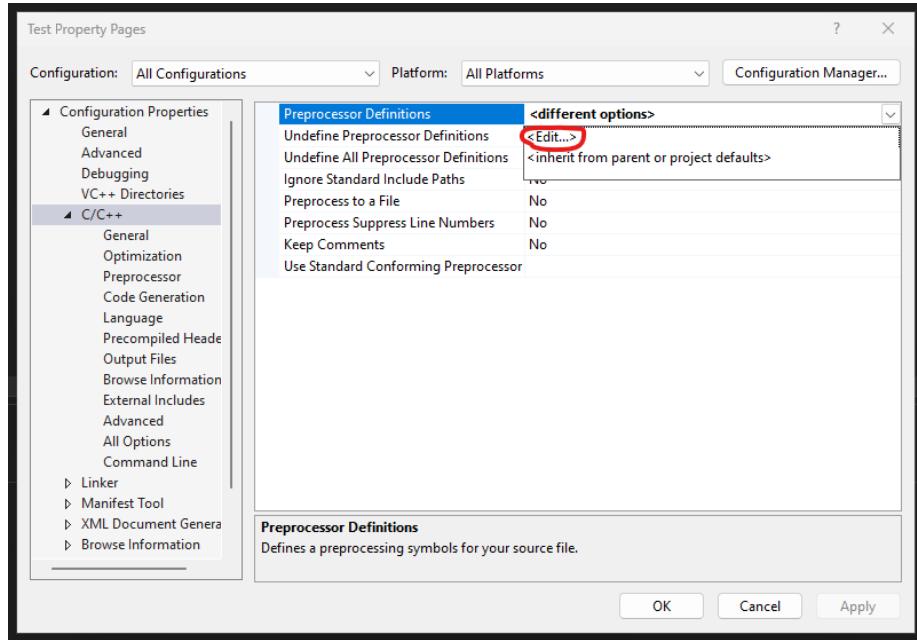
Mergeți la ‘C/C++’ și selectați ‘Preprocesor’.



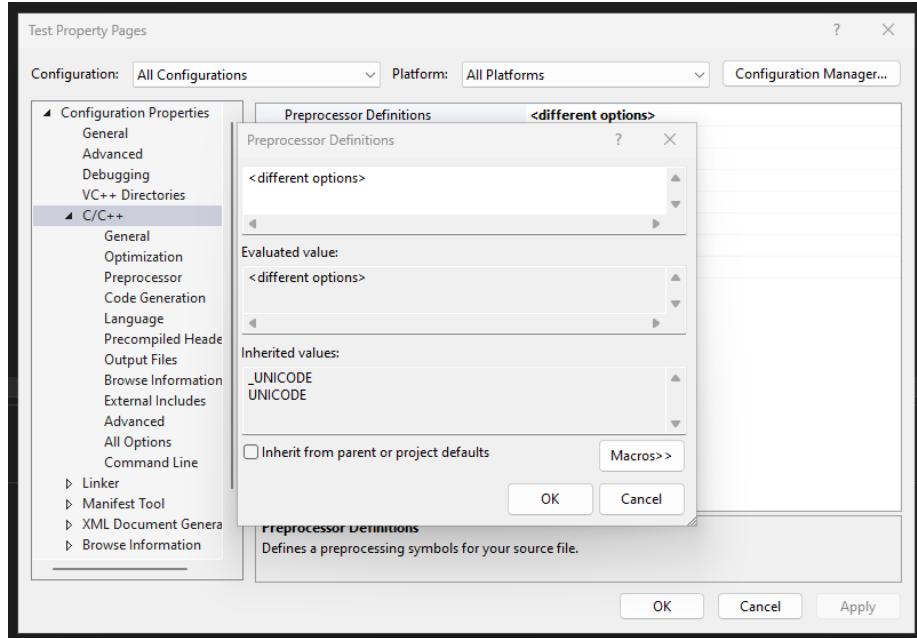
Apoi faceți clic pe ‘Definiții Preprocesor’ și pe săgeata dropdown.



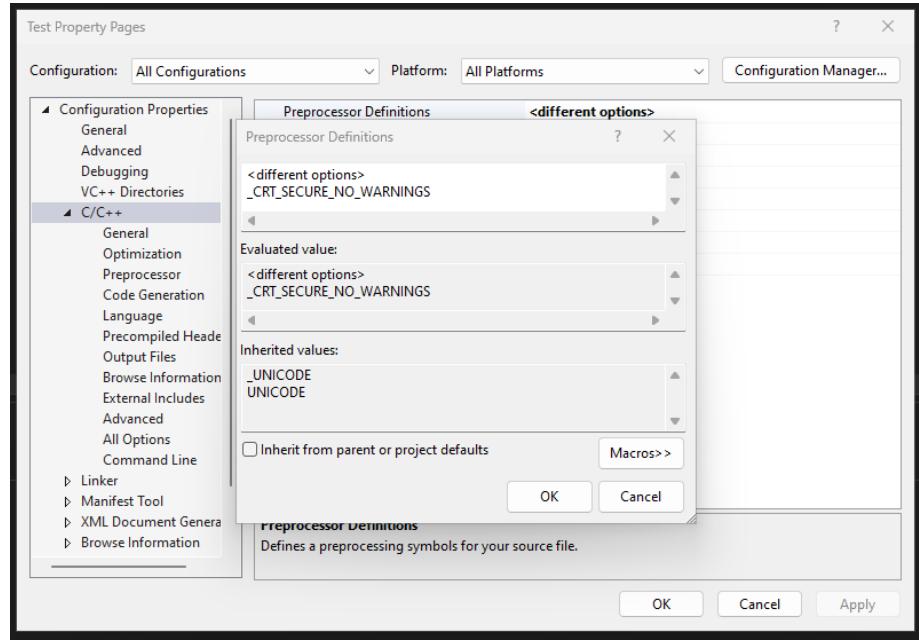
Selectați ‘Editare’



Si se va deschide următoarea fereastră nouă:



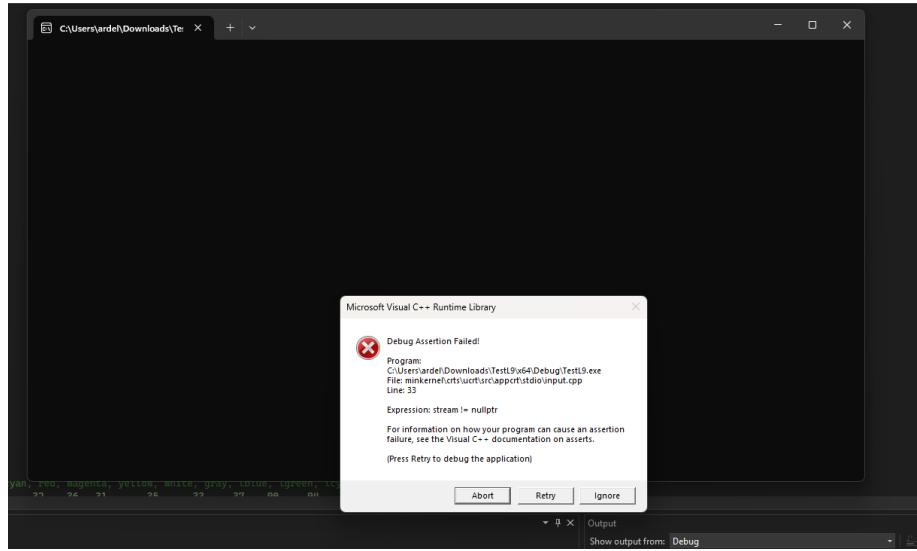
Introduceți ‘_CRT_SECURE_NO_WARNINGS’ sub ‘<opțiuni diferite>’, așa cum se arată mai jos.



Faceți clic pe ‘OK’ la toate ferestrele deschise până când reveniți la fereastra principală Visual Studio a proiectului și veți putea rula proiectul.

12.3 Eroare ‘Assertion’ în Visual Studio

Aceasta indică faptul că nu ați urmat tutorialul. Întoarceți-vă la prima pagină și asigurați-vă că ați mutat fișierele din folderul ‘Downloads’ în folderul ‘Project’. De asemenea, este posibil să fie nevoie să *ștergeți* toate fișierele din IDE-ul Visual Studio și să le *adăugati din nou* manual.



12.4 Eroare undefined în CLion

Exemplu:

Soluție:

Deschideți fișierul ‘CMakeLists.txt’ și modificați după cum urmează:

The screenshot shows the Clion IDE interface. The left pane is the 'Project' view, displaying a file tree with the following structure:

- Test (C:\Users\larde\CLionProjects\Test)
- cmake-build-debug
- bfs.cpp
- bfs.h
- CMakeLists.txt
- grid.txt
- main.cpp
- Profiler.h

The right pane is the 'Editor' view, showing the contents of the CMakeLists.txt file:cmake_minimum_required(VERSION 3.27)
project(Test)

set(CMAKE_CXX_STANDARD 17)

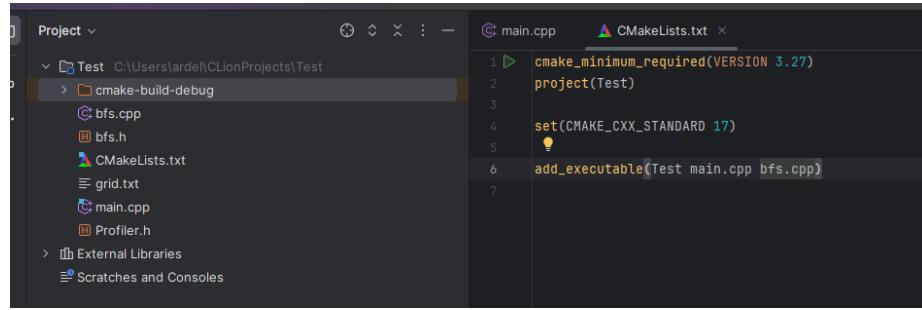
add_executable(Test main.cpp)

adăugați

```
add_executable(Test main.cpp)
```

in

```
add_executable(Test main.cpp bfs.cpp)
```

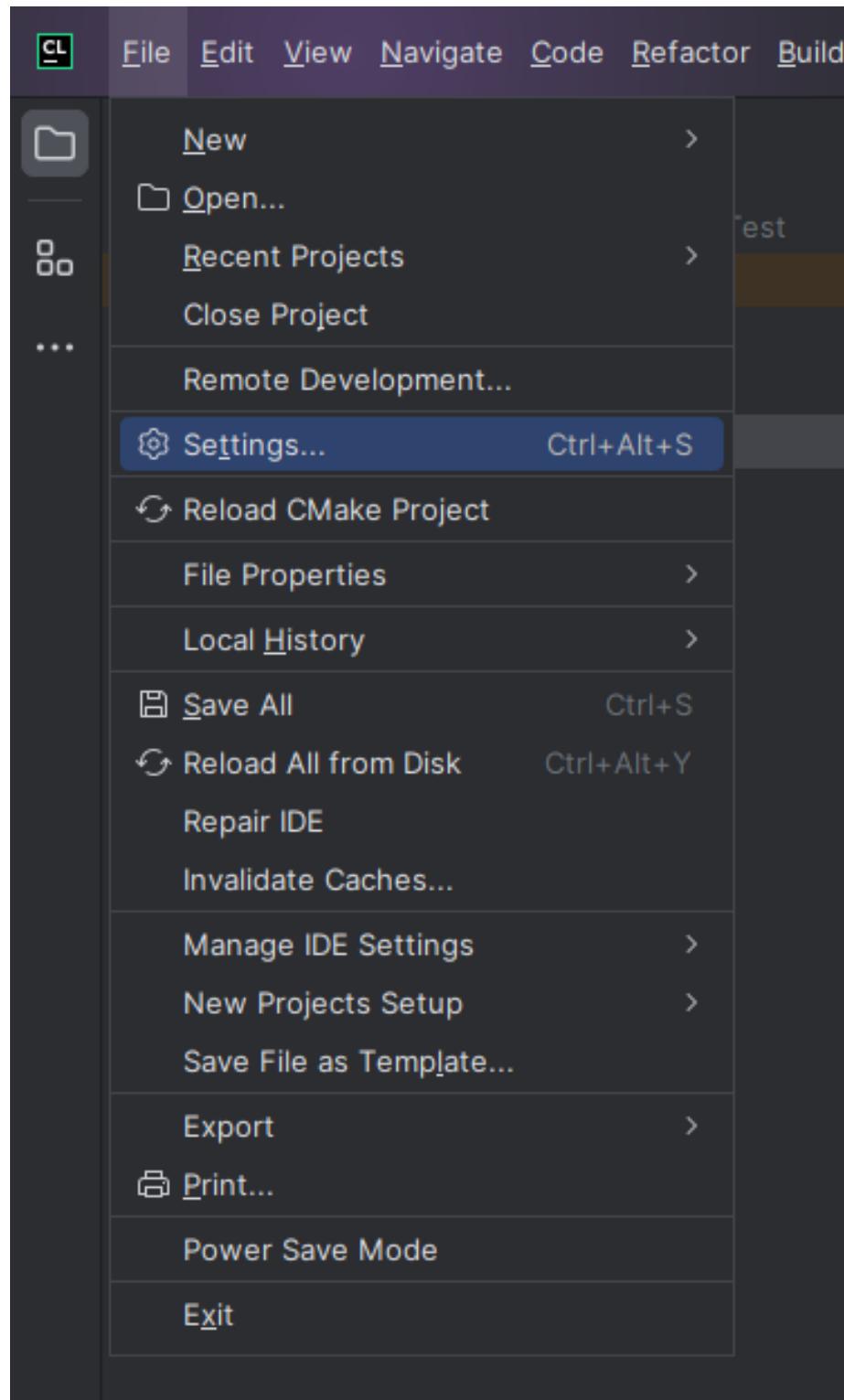


The screenshot shows the CLion IDE interface. On the left is the Project tool window, displaying a CMake project named 'Test' located at 'C:\Users\ardel\CLionProjects\Test'. Inside the project are several files: 'bfs.cpp', 'bfs.h', 'CMakeLists.txt', 'grid.txt', 'main.cpp', and 'Profiler.h'. On the right is the main code editor window, which is currently open to 'CMakeLists.txt'. The code content is as follows:

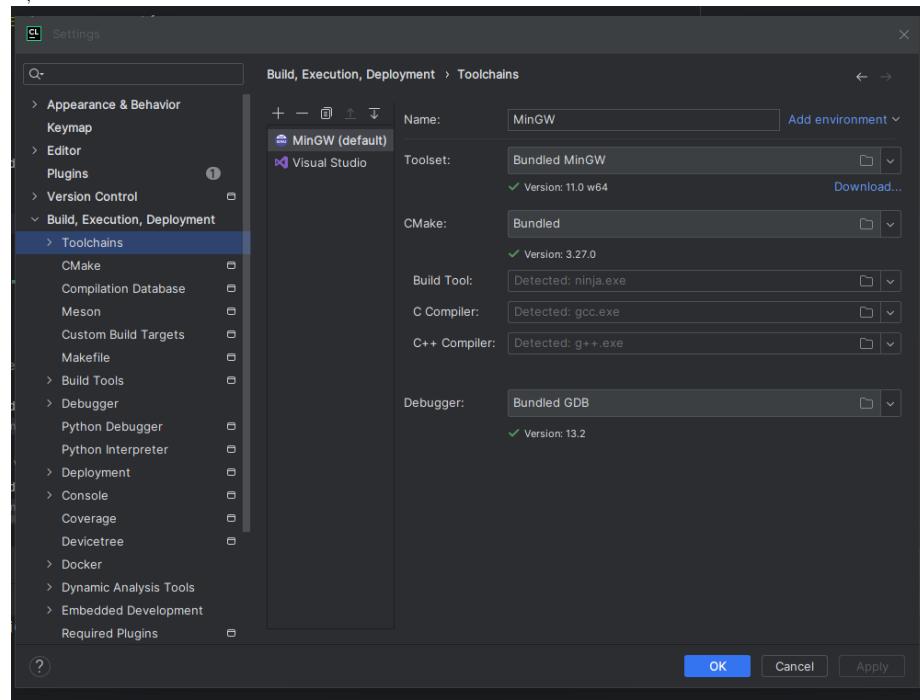
```
1 cmake_minimum_required(VERSION 3.27)
2 project(Test)
3
4 set(CMAKE_CXX_STANDARD 17)
5
6 add_executable(Test main.cpp bfs.cpp)
```

12.5 CLion Visual Studio – Opțiunea 1 (mai lentă)

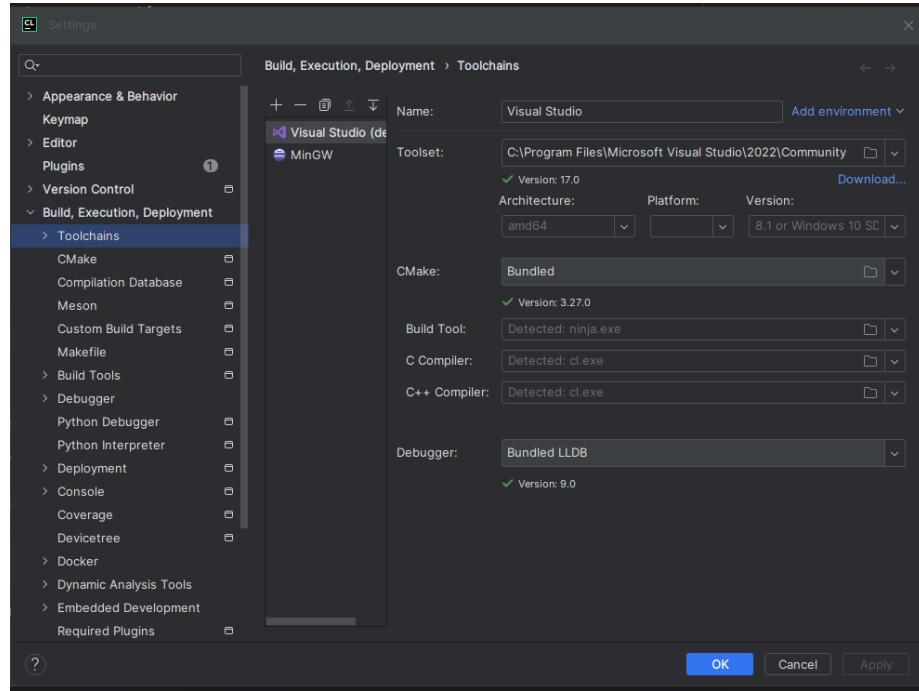
Mergeți la ‘Fișier’ -> ‘Setări’:



În fereastra nou deschisă, mergeți la ‘Compilare, Execuție, Implementare’ -> ‘Lanțuri de instrumente’



Folosind săgețiile, setați Visual Studio ca implicit:



12.6 CLion MinGW – Optiunea 2 (mai rapidă, necesită consolă externă)

12.6.1 Eroare Clear în CLion

Exemplu:

A screenshot of the CLion Test console window. The tab bar shows 'Run' and 'Test' (which is active). The console output shows the path 'C:\Users\ardel\CLionProjects\Test\cmake-build-debug\Test.exe' followed by the error message "'clear' is not recognized as an internal or external command, operable program or batch file." The console has standard scroll and search controls on the left.

Soluție:

Accesați fișierul main.cpp și derulați la liniile 113-117 în funcția displayGrid.

```

98     return "/\\*";
99 }else if((x & MASK_PARENT) == MASK_DOWN){
100    return "\\/";
101 }else if((x & MASK_PARENT) == MASK_LEFT){
102    return "< ";
103 }else if((x & MASK_PARENT) == MASK_RIGHT){
104    return "> ";
105 }else{
106    return " ";
107 }
108 }
109 void displayGrid(const Grid *grid, int lastCommand)
110 {
111    int i, j;
112 #ifdef _MSC_VER
113    system("cls");
114 #else
115    system( Command: "clear");
116 #endif

```

Modificați după cum urmează:

În ramura else de la linia 116

```
system("clear");
```

înlocuiți cu

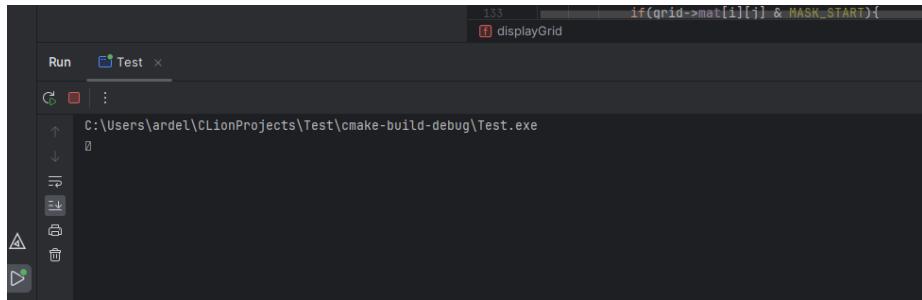
```
system("cls");
```

```

98     return "/\\*";
99 }else if((x & MASK_PARENT) == MASK_DOWN){
100    return "\\/";
101 }else if((x & MASK_PARENT) == MASK_LEFT){
102    return "< ";
103 }else if((x & MASK_PARENT) == MASK_RIGHT){
104    return "> ";
105 }else{
106    return " ";
107 }
108 }
109 void displayGrid(const Grid *grid, int lastCommand)
110 {
111    int i, j;
112 #ifdef _MSC_VER
113    system("cls");
114 #else
115    system( Command: "cls");
116 #endif
117    for(i=0; i<grid->rows; ++i)

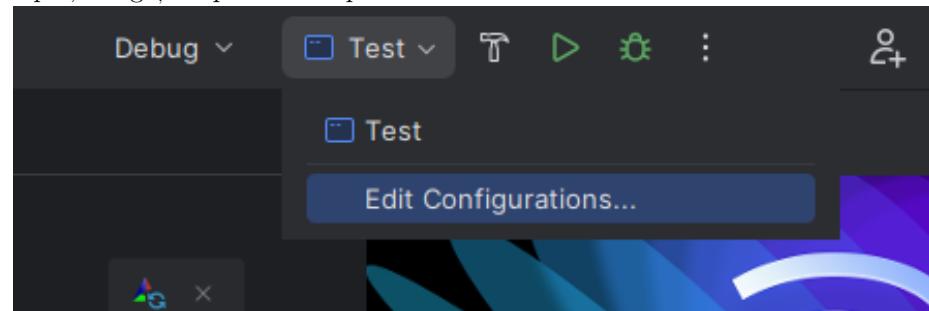
```

12.6.2 CLion nu afișează grila



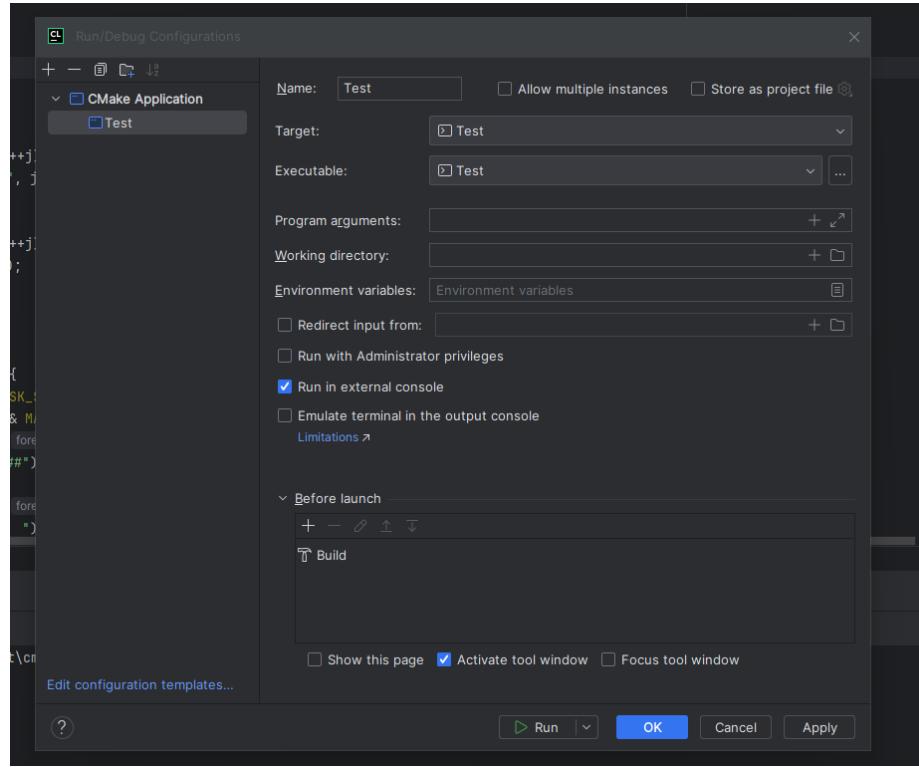
Copiați fișierul ‘grid.txt‘ în folderul ‘cmake-build-debug‘.

Apoi, mergeți în partea dreaptă sus a ecranului:



Selectați ‘Edit Configurations‘ și bifați următoarele opțiuni:

- Rulați în consolă externă



12.7 Comanda de rulare pe Mac

```
g++ main.cpp bfs.cpp -std=c++11 && ./a.out
```

13 Tema Nr. 10: Căutare în adâncime (DFS)

Tema Nr. 10: Căutare în adâncime (DFS)

Timp Alocat: 2 ore

13.1 Implementare

Se cere implementarea corectă și eficientă a algoritmului de căutare în adâncime (Depth-First Search - DFS) (*Capitolul 22.3 din [1]*). Pentru reprezentarea grafurilor, va trebui să folosești liste de adiacență. De asemenea, va trebui să:

- Implementați sortarea topologică (*Capitolul 22.4 din [1]*)
- Implementați algoritmul Tarjan pentru componente tare conexe

13.2 Cerințe minimale pentru notare

Lipsa oricărei cerințe minimale (chiar și parțială) poate rezulta într-o notă mai mică prin penalizări sau refuzul de a prelua tema, rezultând în nota 0.

- *Demo:* Pregătiți un exemplu pentru exemplificarea corectitudinii fiecărui algoritm implementat. Corectitudinea fiecărui algoritm se demonstrează printr-un exemplu simplu (maxim 10 valori).
- Graficele create trebuie să fie ușor de evaluat, adică grupate și adunate prin funcțiile Profiler după cerințele temei. Tema nu va fi evaluată dacă conține o multitudine de grafice negrupate. De exemplu, analiza comparativă implică gruparea într-un singur grafic a algoritmilor comparați.
- Interpretați graficul/graficele și notați observațiile personale în antetul fișierului *main.cpp*, într-un comentariu bloc informativ.
- Nu preluăm teme care nu sunt indentate și care nu sunt organizate în funcții (de exemplu, nu prelăum teme unde tot codul este pus în main).
- *Punctajele din barem sunt corespondente unei rezolvări corecte și complete a cerinței, calitatea interpretărilor din comentariul bloc și răspunsul corect dat de dumea voastră la întrebările puse de către profesor.*

13.3 Cerințe

13.3.1 DFS (5p)

Demo: Demonstrați corectitudinea algoritmului pe un graf de dimensiune mică:

- afișați graful inițial (liste de adiacență)
- afișați arborele rezultat în urma DFS

13.3.2 Sortare topologică (1p)

Demo: Demonstrați corectitudinea algoritmului pe un graf de dimensiune mică:

- afișați graful inițial (liste de adiacență)
- afișați listă de noduri sortate topologic (dacă are / dacă nu are de ce nu are?)

13.3.3 Tarjan (2p)

Demo: Demonstrați corectitudinea algoritmului pe un graf de dimensiune mică:

- afișați graful inițial (liste de adiacență)
- afișați componentele puternic conexe ale grafului

Prezentăm algoritmul lui Tarjan pentru identificarea componentelor tare conexe (SCC) într-un graf orientat. Pentru context și mai multe detalii, consultați articolul de pe Wikipedia: Algoritmul lui Tarjan pentru componentele tare conexe. Pseudocodul poate fi găsit și aici.

13.3.4 Analiza performanței pentru DFS (2p)

Cum timpul de execuție al algoritmului DFS variază în funcție de numărul de vârfuri ($|V|$) și de numărul de muchii ($|E|$) aveți de făcut următoarele analize:

1. Fixați $|V|=100$ și variați $|E|$ între 1000 și 4500 cu un pas de 100. Generați pentru fiecare caz un graf aleator și asigurați-vă că nu generați aceeași muchie de 2 ori. Executați DFS pentru fiecare graf generat și numără operațiile efectuate. Apoi construiește graficul cu variația numărului de operații în funcție de $|E|$;
2. Fixați $|E|=4500$ și variați $|V|$ între 100 și 200 cu un pas de 10. Repetă procedura de mai sus și construiește graficul cu variația numărului de operații în funcție de $|V|$.

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.

Algorithm 1 Tarjan's strongly connected components

```
1: procedure TARJAN( $G = (V, E)$ )
2:   Input: graph  $G = (V, E)$ 
3:   Output: set of strongly connected components (sets of vertices)
4:    $index \leftarrow 0$ 
5:    $S \leftarrow$  empty stack
6:   for all  $v \in V$  do
7:     if  $v.index$  is undefined then
8:       STRONGCONNECT( $v$ )
9:     end if
10:    end for
11:   end procedure

12: function STRONGCONNECT( $v$ )
13:   // Set the depth index for  $v$  to the smallest unused index
14:    $v.index \leftarrow index$ 
15:    $v.lowlink \leftarrow index$ 
16:    $index \leftarrow index + 1$ 
17:    $S.push(v)$ 
18:    $v.onStack \leftarrow true$ 
19:   // Consider successors of  $v$ 
20:   for all  $(v, w) \in E$  do
21:     if  $w.index$  is undefined then
22:       // Successor  $w$  has not yet been visited; recurse on it
23:       STRONGCONNECT( $w$ )
24:        $v.lowlink \leftarrow \min(v.lowlink, w.lowlink)$ 
25:     else if  $w.onStack$  then
26:       // Successor  $w$  is in stack  $S$  and hence in the current SCC
27:       // If  $w$  is not on stack, then  $(v, w)$  is an edge pointing to
28:       // an SCC already
29:       // found and must be ignored
30:        $v.lowlink \leftarrow \min(v.lowlink, w.index)$ 
31:     end if
32:   end for
33:   // If  $v$  is a root node, pop the stack and generate an SCC
34:   if  $v.lowlink = v.index$  then
35:     start a new strongly connected component
36:     repeat
37:        $w \leftarrow S.pop()$ 
38:        $w.onStack \leftarrow false$ 
39:       add  $w$  to current strongly connected component
40:     until  $w = v$ 
41:     output the current strongly connected component
42:   end if
43: end function
```

14 Rezolvarea erorilor

14.1 Profiler + VS Code error

În fișierul `Profiler.h` ai următoarele linii începând cu linia 11:

```
// OS detection
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
    #define PROFILER_WINDOWS
#elif __APPLE__
    #define PROFILER OSX
#elif __linux__
    #define PROFILER_LINUX
#endif

#ifndef PROFILER_WINDOWS
    #include <Windows.h>
    #include <Shellapi.h>
#else
    #include <unistd.h>
#endif
```

Trebuie să le modifici în felul următor:

```
// OS detection
#if defined(__MINGW32__) || defined(__CYGWIN__)
    #define PROFILER_VSCODE
#elif defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
    #define PROFILER_WINDOWS
#elif defined(__APPLE__)
    #define PROFILER OSX
#elif defined(__linux__)
    #define PROFILER_LINUX
#endif

#if defined(PROFILER_WINDOWS) || defined(PROFILER_VSCODE)
    #include <Windows.h>
    #include <Shellapi.h>
#else
    #include <unistd.h>
#endif
```

Mai jos sunt capturile de ecran (în jur de 80% din lățimea textului), încadrate ca figure:

```

(DWORD, PCVOID, DWORD, DWORD, LPWSTR, DWORD, va_list *);

In file included from c:\users\ardel\mingw\include\windows.h:48:0,
                 from C:\Users\ardel\Downloads\Test\Profiler.h:21,
                 from C:\Users\ardel\Downloads\Test\example1.cpp:1:
c:\users\ardel\mingw\include\winuser.h:4351:50: error: 'va_list' has not been declared
WINUSERAPI int WINAPI wsprintfA (LPSTR, LPCSTR, va_list arglist);
                                         ^~~~~~

c:\users\ardel\mingw\include\winuser.h:4352:52: error: 'va_list' has not been declared
WINUSERAPI int WINAPI wsprintfW (LPWSTR, LPCSTR, va_list arglist);
                                         ^~~~~~

In file included from C:\Users\ardel\Downloads\Test\example1.cpp:1:0:
C:\Users\ardel\Downloads\Test\Profiler.h: In member function 'int Profiler::showReport()':
C:\Users\ardel\Downloads\Test\Profiler.h:11263:35: error: 'localtime_s' was not declared in this scope
    localtime_s(&now, &crtTime);
                                         ^
C:\Users\ardel\Downloads\Test\Profiler.h:11282:9: error: '_snprintf_s' was not declared in this scope
    );
           ^
C:\Users\ardel\Downloads\Test\Profiler.h:11284:40: error: 'fopen_s' was not declared in this scope
    fopen_s(&fout, reportName, "wb");
                                         ^
Build finished with error(s).

```

Figure 7: Eroarea initială în VS Code Profiler

```

5 #ifndef _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES_1
6 #endif
7
8 #define _CRT_SECURE_NO_DEPRECATED
9 #define _CRT_SECURE_NO_WARNINGS
10
11 //OS detection
12 #if defined(_MINGW_) || defined(__CYGWIN__)
13 # define PROFILER_LINUX
14 #elif defined(WIN32) || defined(_WIN32) || defined(_WIN32_) || defined(_NT_)
15 # define PROFILER_WINDOWS
16 #elif __APPLE__
17 # define PROFILER OSX
18 #elif __linux
19 # define PROFILER_LINUX
20 #endif
21
22 #ifdef PROFILER_WINDOWS
23 # include <windows.h>
24 # include <Shellapi.h>
25 #else
26 # include <unistd.h>
27 #endif
28
29 #include <stdio.h>
30 #include <string.h>
31 #include <stdlib.h>

```

Figure 8: Captura de ecran originală (before fix)

```

...  C Profiler.h  x  C example1.cpp
C Profiler.h ...
C Profiler.h ...

1 #ifndef _PROFILER_H
2 #define _PROFILER_H
3
4 #ifndef _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES_
5 #define _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES_1
6 #endif
7
8 #define _CRT_SECURE_NO_DEPRECATED
9 #define _CRT_SECURE_NO_WARNINGS
10
11 //OS detection
12 #if defined(WIN32) || defined(_WIN32) || defined(_WIN32_) || defined(_NT_)
13 # define PROFILER_WINDOWS
14 #elif __APPLE__
15 # define PROFILER OSX
16 #elif __linux
17 # define PROFILER_LINUX
18 #endif
19
20 #ifdef PROFILER_WINDOWS
21 # include <windows.h>
22 # include <Shellapi.h>
23 #else
24 # include <unistd.h>
25 #endif
26
27 #include <stdio.h>
28 #include <string.h>
29 #include <stdlib.h>
30 #include <time.h>
31
32 #include <typeinfo>
33 #include <vector>
34 #include <vector>
35 #include <algorithm>
36 #include <functional>
37 #include <strings>

```

Figure 9: Captura de ecran după actualizare (after fix)

References

- [1] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. The MIT Press, 2001. ISBN: 0262032937.