Structuri de date

Tema 3
Grafuri

Deadline – 13.05.2018

Roxana Pavelescu
pavelescu.roxana13@gmail.com

Lavinia-Ștefania Sîrbu
sirbu.lavinia.stefania@gmail.com

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București
An universitar 2017-2018

[30p] Problema 1 – Terminarea programelor distribuite

Un program distribuit reprezintă un program împărțit în taskuri (procese separate) executate în sisteme diferite (cu resurse diferite). Procesele din cadrul unui program distribuit comunică între ele prin trimitere de mesaje. Procesele din cadrul unui program distribuit pot fi văzute ca un graf, în care practic nodurile grafului sunt reprezentate de procese, iar legăturile între noduri sunt reprezentate de legături între procese, două procese putând comunica între ele doar dacă sunt conectate (dacă nodurile din graf au muchie între ele). În general, programele distribuite au un proces **root**, care stabilește când un program distribuit începe și când se termină. El va avea rolul de a strânge practic informații despre celălalte procese din cadrul programului distribuit și va termina programul numai atunci când toate procesele din cadrul programului sunt pregătite pentru terminare. In programele distribuite, o problemă importantă o reprezintă terminarea programelor.

Un program paralel se termină dacă toate procesele sale se termină. Există însă situaţii în care procesele ce compun un program paralel conţin cicluri infinite. Definiţia anterioară nu este aplicabilă în astfel de situaţii, fiind necesară adoptarea unei variante mai cuprinzătoare. Astfel, considerăm că un program este terminat dacă fiecare proces este blocat sau terminat şi nu există nici o cerere de intrare/ieşire în curs de execuţie. Detecţia terminării se poate face simplu în cazul cînd toate procesele se execută pe acelaşi procesor: coada proceselor pregătite pentru execuţie este goală.

În cazul unui algoritm distribuit detectarea terminării este mai dificilă, deoarece nu există nici o cale de a "capta" starea instantanee a tuturor proceselor. Deci, chiar dacă, la un moment dat, un proces termină prelucrarea, el poate fi reactivat ulterior de un mesaj primit de la un alt proces al programului. Din acest motiv, stabilirea terminării unui program necesită comunicarea unor mesaje de control suplimentare între procesele sale.

În cadrul acestei probleme vom rezolva problema terminării programelor distribuite luând în considerare urmatoarele precizări/indicații/simplificări:

- Ca intrare vom avea un graf de procese specificat prin mulțimea de procese (fiecare proces are un id) și legăturile dintre aceste procese.
- Graful va fi considerat ca fiind **neorientat**.
- Fiecare legatură va avea o anumită **pondere** a cărei importanță va fi specificată mai jos.
- Există două tipuri de **mesaje de control**, **activate** și **ready**. Inițial, procesul **root** va porni executarea programului trimițând mesaje de tip activate copiilor săi, aceștia își vor executa treaba (acest pas este considerat ca fiind instant) și vor trimite mesaje de tip ready părinților.
- Un proces poate trimite mesajul de ready părintelui său doar dacă a primit mesaje de ready de la toti copiii săi.
- Programul se consideră ca fiind terminat când procesul root va primi mesajele de ready de la toți copiii săi.

Cazul I - graful de procese nu conține cicluri (arbore)
În acest caz, ignorăm ponderea asociată legaturilor.

Cazul II - graful de procese conține cicluri

- În acest caz, primul pas al rezolvării problemei este determinarea unui arbore de acoperire al grafului. Deoarece un graf poate avea mai mulți arbori de acoperire, dar nu vom putea acoperi în teste toate cazurile, vom alege construirea unui arbore **minim** de acoperire pe baza ponderilor asociate legaturilor dintre procese. Se garantează că pentru testele date va exista un singur arbore de acoperire ce are costul minim.

Fisierul de **input** va avea următorul format:

- Linia 1: NrNoduri NrMuchii IdRadacinaArbore
- Linia 2 (muchia1): NodStart1 NodSfarsit1 Pondere1
- ….
- Linia NrMuchii + 1 (muchiaM): NodStartM NodSfarsitM PondereM

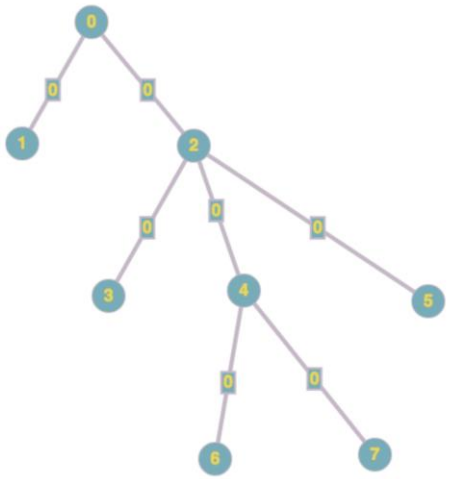
Fiecare proces va avea un id intre 0 si NrNoduri-1.
Pentru cazul I, ponderile vor fi 0 si vor fi ignorate in rezolvarea problemei.

Fisierul de **output** va avea urmatorul format:

- Lista mesaje pas1
- …
- Lista mesaje pasX

O listă de messaje este reprezentată de o listă de perechi de forma X>Y, perechi separate printr-un singur spatiu. X>Y arată faptul că nodul X îi trimite un mesaj nodului Y. Pentru clarificări urmăriți fișierele de input și output folosite în cazul următoarelor exemple.

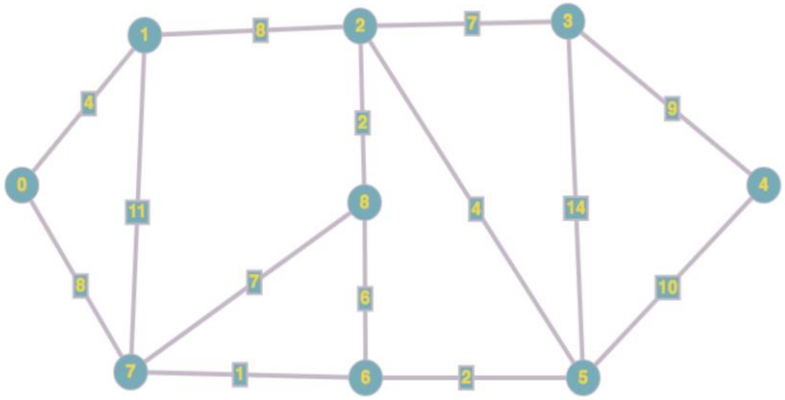
Exemplu caz I



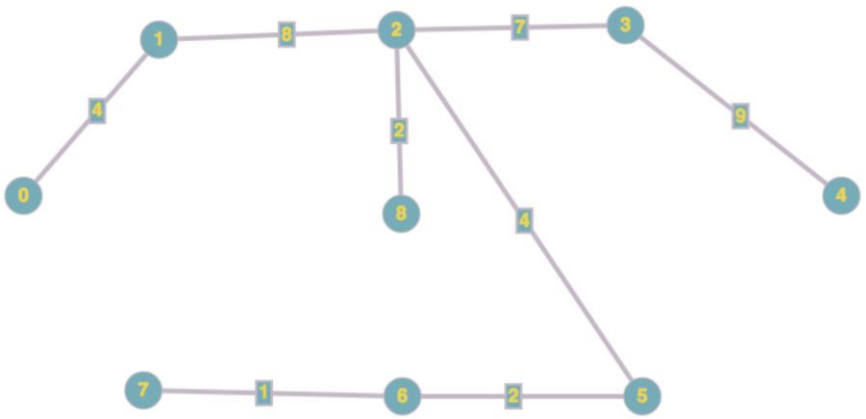
Nodul radacină este considerat nodul cu id 0. Folosind culoarea verde se va marca linia ce va aparea în fișierul de output în urma fiecărui pas. Urmariți fișierele **input11.txt și output11.txt** din arhiva temei.

Pas1: Nodul 0 trimite mesaje de **activate** către 1 și 2. 0>1 0>2
Pas2: Nodul 1 nu are copii, deci trimite mesaj de **ready** către 0. Nodul 2 trimite mesaje de **activate** către 3, 4 și 5. 1>0 2>3 2>4 2>5
Pas3: Nodurile 3 si 5 trimit mesajul de **ready** către 2. Nodul 4 trimite mesaje de **activate** către 6 și 7. 3>2 5>2 4>6 4>7
Pas4: Nodurile 6 și 7 trimit mesajul de **ready** către 4. 6>4 7>4
Pas5: Nodul 4 trimite mesajul de **ready** către 2. 4>2
Pas6: Nodul 2 trimite mesajul de **ready** către 0. 0 a primit mesajele de ready de la toți copiii săi, deci programul se termina. 2>0

Exemplu caz II



Nodul radăcină este nodul cu id 0. Urmariți fișierele **input12.txt și output12.txt** din arhiva temei. Pentru a putea aplica logica folosită în exemplul 1, mai întâi trebuie să se determine arborele minim de acoperire ce va arata ca în imaginea următoare:



[60p] Problema 2

In regatul DTSTR, după o luptă grea cu căpcăunul VALGRIND, regele PTR a reușit să ajungă la o înțelegere. Căpcăunul nu îi va mai ataca pe slujitorii regatului daca va primi în fiecare zi mâncarea pe care și-o dorește, mâncare adusă de peste mari și tări de negustorii care vin săptămânal în cel mai mare port al orașului, portul GDB.

Din cauza poftei incredibile a bestiei, regatul a rămas rapid fără resurse și are nevoie de ajutor din partea locuitorilor. Împăratul a calculat cat costă mâncarea căpcăunului pentru următorul an și vrea să strângă o anumită sumă de MMRLKS, moneda de schimb a negustorilor. Neștiind cate persoane sunt in regat, vrea mai întâi ca servitorii lui să afle câte familii sunt în regat, ca apoi să calculeze contribuția necesară a fiecărei persoane. El vrea să fie drept cu supușii, așa că fiecare om va plati aceeași taxă.

Servitorii nu au putut să afle numărul de persoane, dar i-au adus liste cu toate legăturile între persoane pe care le-au putut afla. Supărat, regele a făcut o listă completă cu toate informațiile de la servitori și a început să calculeze câte familii are regatul și componența acestora.

[20p] Cerința 1 – Determinați numărul de pesoane din regat, taxa pentru fiecare persoană, numărul de familii și numărul de MMRLKS pe care trebuie să îl plătească.

Fișierul de **input** va conține o linie cu numărul de MMRLKS de care are nevoie regatul și apoi linii cu numele a doua persoane între care se știe că există o legătură, deci fac parte din aceeași familie. Pot exista legături duplicat în acest fișier.
Fișierul de **output** va conține o linie cu numărul de persoane, o linie cu taxa pentru fiecare locuitor, o linie cu numărul de familii și apoi o linie cu sumele pe care trebuie să le plătească fiecare familie (sortate crescător).

Exemplu cerința 1 – Urmăriți fișierele **input21.txt și output21.txt** din arhiva temei.

După ce a stabilit câte persoane sunt și a anunțat care este taxa, regele le-a dat termen de o săptămână în care pot aduce banii la castelul din centrul regatului. Fiecare familie își desemnează un reprezentant care poate aduce banii in două moduri:

- Capul familiei merge pe drumul direct dintre casa acelei familii și castel. Acest drum are un anumit cost.
- Capul familiei alege un drum indirect care va trece pe langa alte familii care ii vor cere fiecare o taxă de trecere sau una de transport.

Dacă nu se alege drumul direct, reprezentantul familiei nu poate ajunge la castel și trebuie să plătească taxa de transport uneia dintre familii (nu i se va mai reține și taxa de trecere deoarece va preda banii acelei familii și se va întoarce acasă) și drumul acesteia către castel. Normal, fiecare om vrea ca drumul să fie cât mai puțin costisitor, așa că se gândește ce drum ar fi cel mai bun și îl urmează pe acela. Drumul de întoarcere este gratis.

[10p] Cerința 2 – Considerând că fiecare familie are taxa de transport 0%, dar că aplică taxa de trecere chiar dacă îi sunt predați banii de cel care trece pe pamânturile familiei, calculați drumul cel mai ieftin pentru fiecare familie. De exemplu, dacă există un drum direct familie1 –> castel si un drum indirect familie1 –> familie2 –> familie3 –> castel, banii consumați pe drum sunt fie cei pentru drumul familie1 –> castel, fie o suma drumul familie1 –> familie2 + taxă trecere pe pământul familiei 2 + drumul familie2 --> familie3 + taxă de trecere pe pământul familiei 3 + drumul familie3 --> castel.

[30p] Cerința 3 – Aflați cel mai ieftin drum pentru fiecare familie. De exemplu, dacă există un drum direct familie1 –> castel si un drum indirect familie1 –> familie2 –> familie3 –> castel, banii consumați pe drum sunt fie cei pentru drumul familie1 –> castel, fie o suma drumul familie1 –> familie2 + taxă trecere pe pământul familiei 2 + drumul familie2 --> familie3 + taxă de transport familie3 * banii pe care trebuie ca familia1 să-l ducă la castel + drumul familie3 --> castel.

Fișierul de **input** va conține pe prima linie costul drumurilor directe pentru fiecare familie, pe următoarele NrFamilii linii informații despre taxa de transport în proncente și taxa de trecere a fiecărei familii și pe următoarele X linii, informații despre drumurile dintre familii (graful va fi considerat ca fiind **orientat**).

Fișierul de **output** va conține costul drumurilor minime pentru fiecare familie.

Observații!

Pentru rezolvarea cerințelor 2 si 3 este nercesară rezolvarea cerinței 1. Dacă la cerința 1 am găsit 3 familii, acestea vor fi notate cu numere de la 1 la 3, inclusiv, în cazul reprezentării drumurilor. Nodul cu indexul 0 este folosit pentru reprezentarea castelului. De asemenea, costul drumurilor directe este ordonat să coincidă cu ordinea în care au fost afișate sumele de la cerința 1. De exemplu, pentru un output [1000, 2000, 3000] al cerinței 1 și un input [200, 100, 300] al cerinței 2, știm următoarele informații:

- Familia 1 trebuie să plătească 1000 MMRLKS și drumul direct costă 200 MMRLKS.
- Familia 2 trebuie să plătească 2000 MMRLKS și drumul direct costă 100 MMRLKS.
- Familia 3 trebuie să plătească 3000 MMRLKS și drumul direct costă 300 MMRLKS.

Fișierele de input arată la fel pentru ambele cerințe, dar **in cazul cerinței 2 se va ignora taxa de transport.**

[20p] Bonus

[10p] Afișați calea (secvența de noduri) pe care o parcurge fiecare cap de familie și câți bani plătește în fiecare punct conform cerinței 2.

[10p] Afișati calea (secvența de noduri) pe care o parcurge fiecare cap de familie și câți bani plătește în fiecare punct conform cerinței 3.

Pentru formatul fișierului de output dorit urmăriți următoarele două exemple.

În următoarele exemple vom folosi următoarele culori pentru fiecare sumă de bani plătită de o familie pe drumul către castel:

- Verde pentru suma de bani necesară pentru un drum între familii.
- Roșu pentru taxa de trecere.
- Albastru pentru banii rezultatiți din aplicarea taxei de transport.
- Negru pentru un drum direct familie --> castel.

Exemplu cerința 2 și bonus

Pentru această cerință considerăm **input22_cerinta1.txt** și **output22_cerinta1.txt** ca fiind fișierele folosite în cerința 1, cerință ce trebuie rezolvată pentru a se putea trece la cerința 2. Găsiți aceste fișiere, alături de **input22.txt și output22.txt,** în arhiva temei, dar pentru explicația acestei cerințe puteți urmări Figura 1.

input22.txt

```
10 30 25
10 3
20 2
10 5
1 2 3
2 3 10
3 1 1
3 2 10
```

Pentru Familia1, cel mai scurt drum este cel direct F1 → C si are costul 10.

Pentru Familia2, cel mai scurt drum este F2 → F3 → F1 → C cu un cost de 10 + 5 + 1 + 3 + 10 = 29.

Pentru Familia3, cel mai scurt drum este F3 → F1 → C cu un cost de 1 + 3 + 10 = 14

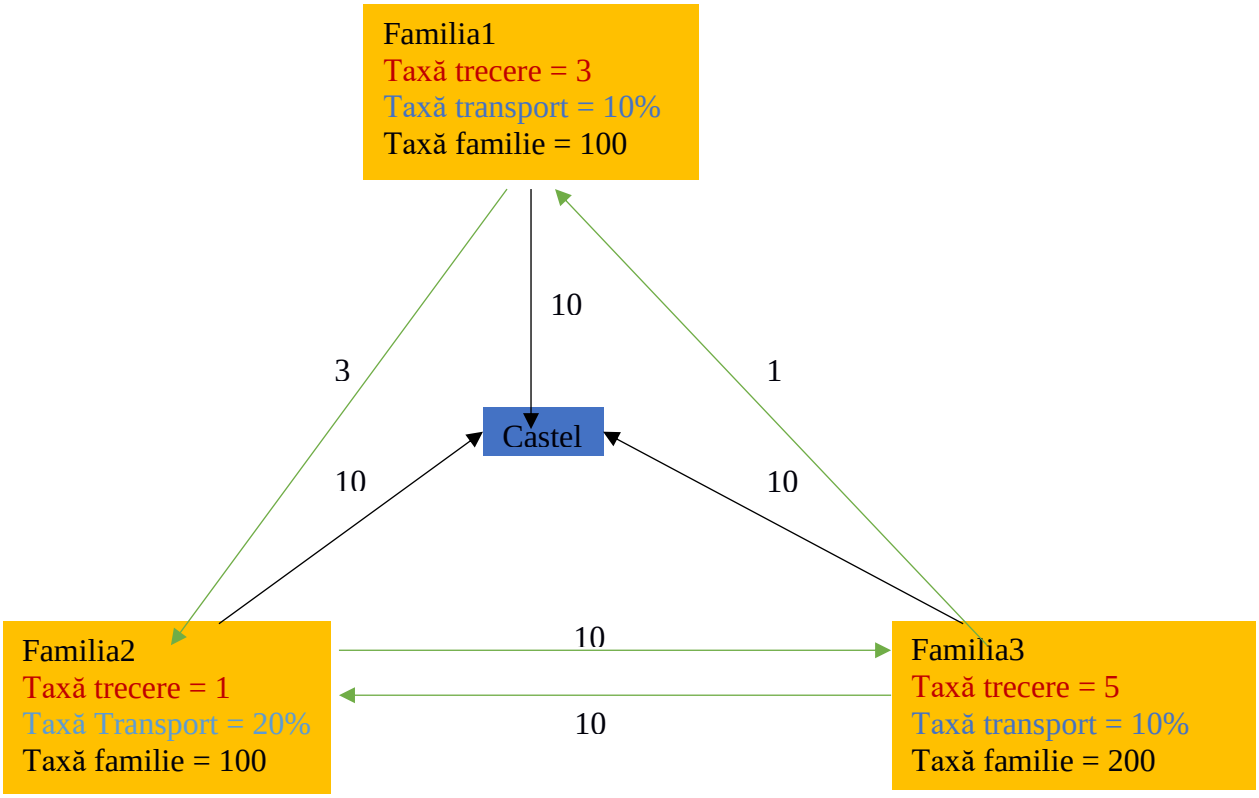


Figura 1

output22.txt
10 29 14

output22_bonus.txt
1 0
1 3 10
2 3 1 0
10 5 1 3 10
3 1 0
1 3 10

Exemplu cerința 3 și bonus

Pentru această cerință considerăm **input23_cerinta1.txt** și **output23_cerinta1.txt** ca fiind fișierele folosite în cerința 1, cerința ce trebuie rezolvată pentru a se putea trece la cerința 3. Găsiți aceste fișiere, alături de **input23.txt și output23.txt,** în arhiva temei, dar pentru explicația acestei cerințe puteți urmări Figura 2.

input23.txt
100 50 300 100
5 2
10 0
10 0
20 7
1 2 30
4 1 30
3 4 30
3 1 80
3 2 100

Familia1 trebuie să plătească 100 MMRLKS și are următoarele posibilități:

- F1 → C cu un cost de 100.
- F1 → F2 → C cu un cost de 30 + 10% * 100 + 50 = 90.
- Drumul cel mai scurt este F1 → F2 → C.

Familia2 are o singură posibilitate F2 → C cu un cost de 50.

Familia3 trebuie să plătească 300 MMRLKS și are următoarele posibilități:

- F3 → C cu un cost de 300.
- F3 → F4 → C cu un cost de 30 + 20% * 300 + 100 = 190.
- F3 → F4 → F1 → C cu un cost de 30 + 7 + 30 + 5% * 300 + 100 = 192.
- F3 → F4 → F1 → F2 →C cu un cost de 30 + 7 + 30 + 2 + 30 + 10% * 300 + 50 = 179.
- Drumul cel mai scurt este F3 → F4 → F1 → F2 →C.

Familia4 trebuie să plătească 400 MMRLKS și are următoarele posibilități:

- F4 → C cu un cost de 100.
- F4 → F1 → C cu un cost de 30 + 5% * 400 + 100 = 150.
- F4 → F1 → F2 → C cu un cost de 30 + 2 + 30 + 10% * 400 + 50 = 152.
- Drumul cel mai scurt este F4 → C.

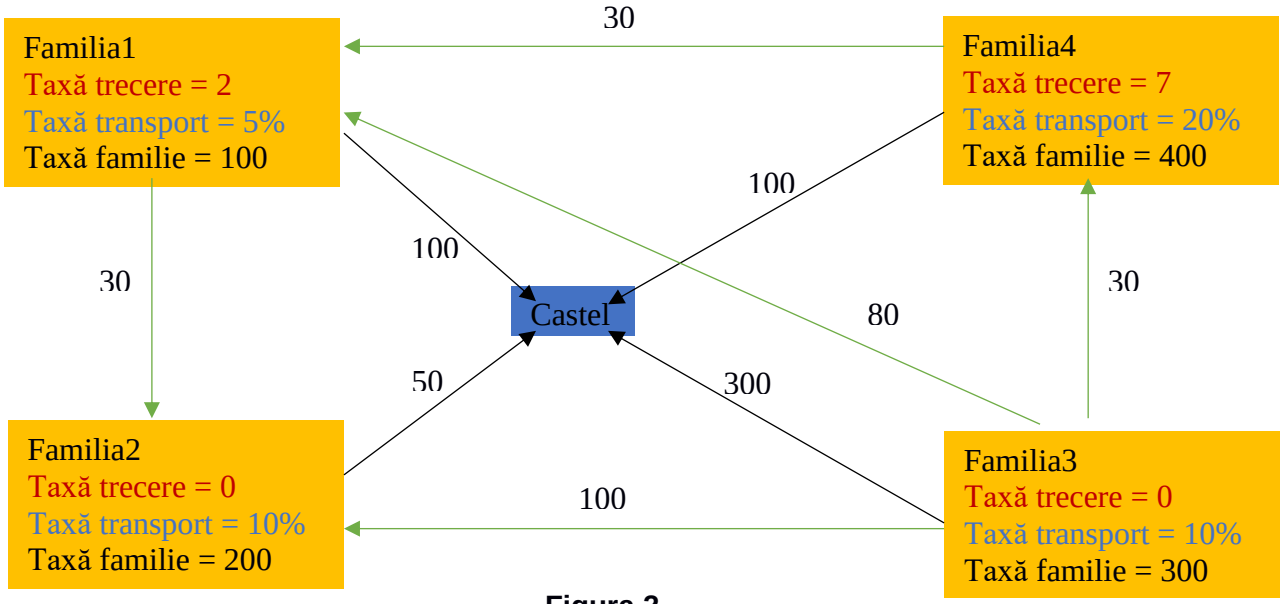


Figura 2

output23.txt
90 50 179 100

output23_bonus.txt
1 2 0
30 10 50
2 0
50
3 4 1 2 0
30 7 30 2 30 30 50
4 0
100

Restricții și precizări:

- Urmăriți cu atenție formatul fișierelor de input/output din exemple pentru a evita greșelile în care rezolvarea este bună, dar afișarea nu, caz în care testele vor pica.
- Temele trebuie să fie încărcate pe vmchecker. NU se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului.
- O rezolvare constă într-o arhivă de tip zip care conţine toate fişierele sursă alaturi de un Makefile, ce va fi folosit pentru compilare, şi un fisier README, în care se vor preciza detaliile implementării si algoritmii folositi.
- Makefile-ul trebuie să aibă obligatoriu regulile pentru build şi clean. Regula build trebuie să aibă ca efect compilarea surselor şi crearea binarelor problema1 si problema2.
- Problemele vor primi ca argumente in linia de comandă numele fișierelor de input și output.
  ./problema1 input.txt output.txt
  ./problema2 input1.txt output2.txt input_2_3.txt output2.txt output3.txt output2_bonus.txt output3_bonus.txt
- Argumentele sunt aceleași chiar dacă se rezolvă parțial problema 2, dar se ignoră numele fișierelor. De exemplu, daca se rezolvă doar cerințele 1 și 3, se ignoră output2.txt, output2_bonus.txt și output3_bonus.txt.
- Tema va fi implementată folosind limbajul **C.**
- **Tema este individuală! Toate soluţiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.**

Punctaj (Total = 120p)

Problemă	Puncte
Problema 1	30p
Problema 2 – Cerința 1	20p
Problema 2 – Cerința 2	10p
Problema 2 – Cerința 3	30p
Coding style, Makefile, Readme	10p
Bonus – Cerința 1	10p
Bonus – Cerința 2	10p