

PA - TEMA 2

- GIGEL, MEREU OCUPAT -

Responsabili:

Andrei Preda, Traian Enache, Theodor Pruteanu,
Robert Truță, Radu Nichita

Deadline soft: 26.05.2023 23:55
Deadline hard: 28.05.2023 23:55

CUPRINS

1	Problema 1: Supercomputer	3
1.1	Enunț	3
1.2	Date de intrare	3
1.3	Date de ieșire	3
1.4	Restricții și precizări	3
1.5	Testare și punctare	3
1.6	Exemple	4
2	Problema 2: Căi ferate	5
2.1	Enunț	5
2.2	Date de intrare	5
2.3	Date de ieșire	5
2.4	Restricții și precizări	5
2.5	Testare și punctare	5
2.6	Exemple	6
3	Problema 3: Teleportare	7
3.1	Enunț	7
3.2	Date de intrare	7
3.3	Date de ieșire	7
3.4	Restricții și precizări	7
3.5	Testare și punctare	8
3.6	Exemple	8
4	Problema 4 (bonus): Magazin	9
4.1	Enunț	9
4.2	Date de intrare	9

4.3	Date de ieșire	10
4.4	Restricții și precizări	10
4.5	Testare și punctare	10
4.6	Exemple	10
5	Punctare	11
5.1	Checker	11
6	Folosire ChatGPT	12
7	Format arhivă	12
8	Links	13

1 PROBLEMA 1: SUPERCOMPUTER

1.1 Enunț

Gigel a fost rugat de profesorul lui de informatică să îl ajute cu o problemă de organizare. Are nevoie să ruleze N task-uri pe un supercomputer, între task-uri existând dependențe de tipul: task-ul i trebuie terminat înainte de a începe task-ul j . De asemenea, task-urile au nevoie ca unul din **două seturi de date** să fie încărcate în memorie pentru a rula. Dacă două task-uri ce rulează consecutiv au nevoie de același set de date, setul poate rămâne în memorie între execuții, iar al doilea task îl va folosi direct, rapid. În schimb, dacă cele două task-uri au nevoie de **seturi diferite**, trebuie să realizăm un **context switch costisitor**, pe care ne dorim să îl evităm.

Pe Gigel și profesorul lui îi interesează numărul minim de context switch-uri costisitoare necesare pentru a rula toate cele N task-uri.

Așa cum v-ați obișnuit, Gigel nu se descurcă fără ajutorul vostru.

1.2 Date de intrare

Fișierul de intrare **supercomputer.in** conține pe prima linie N (numărul de task-uri) și M (numărul de dependențe).

Pe a doua linie se află N numere, a_1, a_2, \dots, a_n , unde a_k reprezintă setul de date cerut de task-ul k .

Pe următoarele M linii sunt câte două numere, u și v , cu semnificația că task-ul u trebuie terminat înainte de task-ul v .

1.3 Date de ieșire

Fișierul de ieșire **supercomputer.out** va conține numărul minim de context switch-uri costisitoare necesare pentru a rula toate cele N task-uri.

1.4 Restricții și precizări

- $1 \leq N \leq 10^5$
- $0 \leq M \leq 2 \times 10^5$
- $1 \leq a_i \leq 2$
- Va exista mereu cel puțin un mod de a executa toate task-urile.

1.5 Testare și punctare

- Punctajul maxim este de 30 puncte.
- Timpul de execuție:

- C/C++: **0.5 s**
- Java: **1.5 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită:
supercomputer.c, **supercomputer.cpp** sau **Supercomputer.java**.

1.6 Exemple

supercomputer.in	supercomputer.out	Explicație
5 6 1 2 1 2 1 1 2 1 3 2 4 3 4 2 5 3 5	2	<p>O ordine optimă poate fi: 1, 3, 2, 4, 5.</p> <p>Va trebui să schimbăm setul de date după ce terminăm task-urile 3 și 4.</p>

2 PROBLEMA 2: CĂI FERATE

2.1 Enunț

În cadrul unei firme de transport feroviar, s-a demarat un proiect pentru a extinde zona în care aceasta își desfășoară activitatea.

În zona de exploatare există N gări. Firma are depozitul principal într-o gară S , de unde se dorește livrarea de mărfuri către toate celelalte gări, între care există în total M linii ferate pe care se poate circula într-un singur sens.

Pentru a putea livra mărfuri de la depozitul principal la o gară oarecare, trebuie să existe o rută care duce de la depozit la destinație, eventual trecând și prin alte gări (nu neapărat și invers).

Gigel, șef de lucrări în cadrul firmei, este responsabil pentru acest proiect. Ajutați-l să determine care este numărul minim de linii feroviare ce trebuie construite pentru a putea finaliza proiectul.

2.2 Date de intrare

Pe prima linie a fișierului **ferate.in** se află N (numărul de gări), M (numărul de linii ferate existente) și S (gara ce reprezintă depozitul principal).

Pe următoarele M linii se află câte 2 numere x și y , cu semnificația că există o linie ferată dinspre gara x spre gara y .

2.3 Date de ieșire

În fișierul **ferate.out** se va scrie numărul minim de linii ferate ce trebuie construite.

2.4 Restricții și precizări

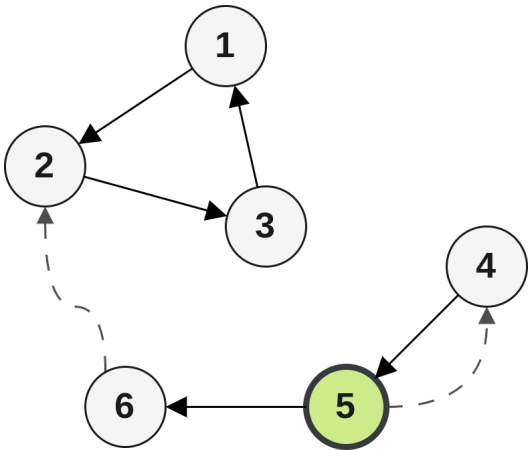
- $1 \leq N \leq 10^5$
- $0 \leq M \leq \min(2 \times 10^5, N \times (N - 1))$
- $1 \leq S, x, y \leq N$

2.5 Testare și punctare

- Punctajul maxim este de 35 puncte.
- Timpul de execuție:
 - C/C++: 0.5 s
 - Java: 2 s

- Sursa care conține funcția **main** trebuie obligatoriu denumită: **ferate.c**, **ferate.cpp** sau **Ferate.java**.

2.6 Exemple

ferate.in	ferate.out	Explicație
6 5 5 1 2 2 3 3 1 4 5 5 6	2	<div><pre>graph TD; 1((1)) --> 2((2)); 2 --> 3((3)); 3 --> 1; 5((5)) --> 6((6)); 4((4)) --> 5; 6 -.-> 2; 5 -.-> 4;</pre></div> <p>Se pot construi două linii ferate: spre exemplu, una din 5 către 4, și una din 6 către 2.</p>

3 PROBLEMA 3: TELEPORTARE

3.1 Enunț

Pentru că este în întârziere, Dorel își dorește să ajungă la curs cât mai repede. Harta facultății poate fi reprezentată ca un graf neorientat cu N noduri, în care costul fiecărei muchii reprezintă timpul în care Dorel o poate parcurge (măsurat în secunde).

Din fericire, Dorel a descoperit de curând că în facultate există câteva **tunele bidirecționale de teleportare**, pe care le poate parcurge **într-o singură secundă**. Din nefericire, fiecare astfel de tunel funcționează doar odată la fiecare P_i secunde, iar acest număr se numește *perioada tunelului*. De exemplu, Dorel poate intra într-un tunel de perioadă 3 doar la momentele 0, 3, 6, 9 etc.

În plus, pentru că devine agitat atunci când e în întârziere, Dorel nu poate sta pe loc, ci trebuie să se miște continuu (ceea ce înseamnă că nu poate alege să aștepte într-un nod).

Ajutați-l pe Dorel să ajungă cât mai repede din camera lui (nodul 1), în sala de curs (nodul N).

3.2 Date de intrare

Pe prima linie a fișierului **teleportare.in** se află numărul N de noduri, numărul M de muchii normale și numărul K de muchii de teleportare.

Pe următoarele M linii se află triplete de numere întregi pozitive $X Y T$, reprezentând muchii obișnuite între nodurile X și Y , ce sunt parcurse în T secunde.

Pe următoarele K linii se află triplete de numere întregi pozitive $X Y P$, reprezentând muchii de teleportare între X și Y , cu perioada P (muchia există doar la timpurile $0, P, 2P, 3P \dots$).

3.3 Date de ieșire

În fișierul **teleportare.out** se va scrie durata minimă a drumului din nodul 1 în nodul N .

3.4 Restricții și precizări

- $2 \leq N \leq 10^4$
- $1 \leq M + K \leq \min\left(2 \times 10^5, \frac{N \times (N-1)}{2}\right)$
- $1 \leq T_i \leq 10^9$
- $1 \leq P_i \leq 8$
- Va exista cel puțin un drum accesibil de la sursă la destinație.
- Dorel pornește la drum la momentul 0 (zero).

3.5 Testare și punctare

- Punctajul maxim este de **40** puncte.
- Timpul de execuție:
 - C/C++: **2 s**
 - Java: **2.5 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **teleportare.c**, **teleportare.cpp** sau **Teleportare.java**.

3.6 Exemple

teleportare.in	teleportare.out	Explicație
4 3 1 1 2 2 2 3 1 3 4 1 2 4 2	3	<p>Drumul optim este: 1 -> 2 -> 4.</p> <p>Plecăm din nodul 1 și ajungem în nodul 2 la momentul 2. De aici ne putem teleporta până în nodul 4, unde vom ajunge la momentul 3.</p>
4 3 1 1 2 1 2 3 1 3 4 2 2 4 5	4	<p>Drumul optim este: 1 -> 2 -> 3 -> 4.</p> <p>Nu ne putem teleporta din 2 în 4 la momentul 1, pentru că tunelul este închis atunci. Dacă ne mai plimbăm până se redeschide tunelul, drumul va dura mai mult de 4 secunde.</p>

4 PROBLEMA 4 (BONUS): MAGAZIN

4.1 Enunț

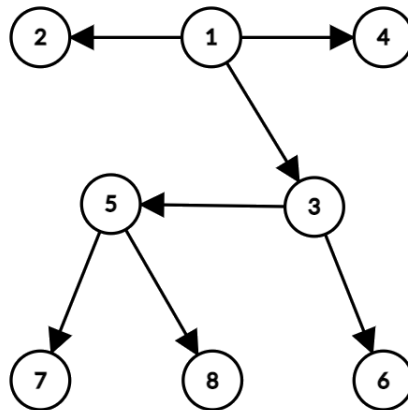
Un lanț de magazine deține N depozite în toată țara, numerotate de la 1 la N . Pentru a facilita livrarea de produse, fiecare depozit va primi colete doar de la **un singur depozit** și poate trimite colete spre zero sau mai multe depozite. O excepție de la această regulă este depozitul numărul 1, care nu va primi colete de la niciun alt depozit.

O expediție de colete se desfășoară în următorul mod: Fie X depozitul care vrea să expedieze, și x_1, x_2, \dots, x_k depozitele spre care acesta poate trimite colete în mod direct, **ordonate crescător** după număr.

X va expedia inițial spre x_1 . x_1 va păstra o parte din colete, și va expedia la rândul său celelalte colete spre alte depozite, respectând același procedeu. După ce x_1 a terminat de expedit, va trimite o confirmare spre X , în urma căreia X va continua cu următorul depozit din lista sa, x_2 . Procedeu se repetă până la ultimul depozit din listă.

Sarcina voastră este să răspundeți la Q întrebări de forma: „Ce depozit va primi coletele plecate de la depozitul D , după E expedieri consecutive?”

De exemplu, pentru următoarea ierarhie:



Ordinea depozitelor în care sunt primite coletele expediate de depozitul 1 este: 2, 3, 5, 7, 8, 6, 4. Astfel, pentru o întrebare de forma „Ce depozit va primi coletele plecate de la depozitul 1 după 5 expedieri consecutive?”, răspunsul este 8.

4.2 Date de intrare

Fișierul de intrare **magazin.in** conține pe prima linie 2 numere: N și Q , reprezentând numărul de depozite, respectiv numărul de întrebări.

Pe a doua linie se află $N - 1$ numere. Al k -lea număr reprezintă depozitul de la care va primi colete depozitul $k + 1$.

Următoarele Q linii conțin câte o întrebare codificată printr-o pereche $D E$, unde D reprezintă depozitul de la care începe expedierea, iar E reprezintă numărul de expediții consecutive.

4.3 Date de ieșire

Fișierul de ieșire **magazin.out** va conține Q linii cu câte un număr fiecare, reprezentând răspunsul la cele Q întrebări. Pentru situația în care **nu se pot efectua** E expediții consecutive **se va afișa -1**.

4.4 Restricții și precizări

- $2 \leq N \leq 10^5$
- $1 \leq Q \leq 5 \times 10^5$
- $1 \leq D, E \leq N$

4.5 Testare și punctare

- Punctajul maxim este de **30 puncte**.
- Timpul de execuție:
 - C/C++: **0.5 s**
 - Java: **2 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **magazin.c**, **magazin.cpp** sau **Magazin.java**.

4.6 Exemple

magazin.in	magazin.out	Explicație
8 5 1 1 1 3 3 5 5 1 5 1 8 3 1 3 4 4 1	8 -1 5 6 -1	Vezi ierarhia de mai sus.

5 PUNCTARE

- Punctajul temei este de **150** puncte, distribuit astfel:
 - Problema 1: **30p**
 - Problema 2: **35p**
 - Problema 3: **40p**
 - Problema 4: **30p**
 - 10 puncte vor fi acordate pentru comentarii și README.
 - 5 puncte vor fi acordate automat de checker pentru coding style. Totuși, la corectarea manuala se pot aplica **depunctari de până la 20 de puncte** pentru **coding style neadecvat**.

Punctajul pe README, comentarii și coding style este condiționat de obținerea unui punctaj strict pozitiv pe cel puțin un test.

Se poate obține un **bonus** de **30p** rezolvând problema **Magazin**. Acordarea bonusului **NU** este condiționată de rezolvarea celorlalte probleme. În total se pot obține 150 de puncte (**NU** se trunchiază).

Pentru detalii puteți să vă uitați și peste **regulile generale** de trimitere a temelor.

- O temă care **NU** compilează va fi punctată cu 0.
- O temă care **NU** trece niciun test pe vmchecker va fi punctată cu 0.
- Vor exista mai multe teste pentru fiecare problemă în parte. Punctele pe teste sunt independente, punctajul pe un anumit test nefiind condiționat de alte teste.
- Fiecare problemă va avea o limită de timp pe test (precizată mai jos și pe pagina cu enunțul). Dacă execuția programului pe un test al acelei probleme va dura mai mult decât limita de timp, veți primi automat 0 puncte pe testul respectiv și execuția va fi întreruptă.
- În fișierul README va trebui **să descrieți soluția** pe care ați ales-o pentru fiecare problemă, **să precizați complexitatea** pentru fiecare și alte lucruri pe care le considerați utile de menționat.

5.1 Checker

- Arhiva se va trimite pe **vmchecker**, unde tema se va testa folosind un set de teste private.
- Pentru testarea locală, aveți disponibil un set de teste publice (de aceeași dificultate) pe pagina cu **resurse** a temei.
- Checkerul se poate rula fără niciun parametru, caz în care va verifica toate problemele. De asemenea, se mai poate rula cu un parametru pentru a rula o anumită problemă:


```
./check.sh < 1 | 2 | 3 | 4 >
./check.sh < supercomputer | ferate | teleportare | magazin >
./check.sh cs
```

- **Punctajul pe teste** este cel de pe vmchecker și se acordă rulând tema doar cu testele private.
- Checkerul verifică doar existența unui README cu denumire corectă și conținut nenul. **Punctajul final pe README și comentarii** se acordă la corectarea manuală a temei.
- La corectarea manuală se poate depuncta pentru **erori de coding style** care nu sunt semnalate de checker.
- Corectorii își rezervă dreptul de a scădea puncte pentru orice problemă găsită în implementare, dacă vor considera acest lucru necesar.
- Pentru citirea în Java se recomandă folosirea **BufferedReader**.

6 FOLOSIRE CHATGPT

- Folosirea ChatGPT, Copilot sau a oricărui model de limbaj sau tool (denumit în continuare **LLM**) ce vă poate ajuta la rezolvarea temei, cu idei sau cod, este puternic descurajată, dar nu interzisă.
- În cazul în care folosiți LLM-uri, trebuie să specificați acest lucru în README, precum și modul în care acestea au fost folosite (ex: ce prompt-uri ați folosit), pentru fiecare problemă pentru care au fost folosite tool-uri.
- Pentru fiecare problemă rezolvată folosind un LLM, pentru care a fost specificat în README acest lucru, se va aplica o penalizare de 33% din punctajul acelei probleme.
- În cazul în care o problemă este rezolvată folosind un LLM, dar nu este specificat acest lucru în README, acest lucru se va considera **încercare de copiere** și se va sancționa conform **regulamentului**.

7 FORMAT ARHIVĂ

- Temele pot fi testate automat pe vmchecker. Acesta suportă temele rezolvate în C/C++ și Java.
- Arhiva cu rezolvarea temei trebuie să fie **.zip**, având un nume de forma **Grupa_NumePrenume_Tema2.zip** (ex: 399CX_PuiuGigel_Tema2.zip) și va conține:
 - Fișierul/fișierele sursă
 - Fișierul **Makefile**
 - Fișierul **README** (fără extensie)
- Fișierul pentru make trebuie denumit obligatoriu **Makefile** și trebuie să conțină următoarele reguli:
 - **build**, care va compila sursele și va obține executabilele

- **run-p1**, care va rula executabilul pentru problema 1
 - **run-p2**, care va rula executabilul pentru problema 2
 - **run-p3**, care va rula executabilul pentru problema 3
 - **run-p4**, care va rula executabilul pentru problema bonus (**doar dacă** ați implementat și bonusul)
 - **clean**, care va șterge executabilele generate
- **ATENȚIE!** Funcția **main** din rezolvarea unei probleme se va găsi într-o sursă ce trebuie obligatoriu denumită astfel:
 - **supercomputer.c**, **supercomputer.cpp** sau **Supercomputer.java** - pentru problema 1
 - **ferate.c**, **ferate.cpp** sau **Ferate.java** - pentru problema 2
 - **teleportare.c**, **teleportare.cpp** sau **Teleportare.java** - pentru problema 3
 - **magazin.c**, **magazin.cpp** sau **Magazin.java** - pentru problema 4
 - **ATENȚIE!** Tema va fi compilată și testată **DOAR pe Linux**.
 - **ATENȚIE!** Numele regulilor și a surselor trebuie să fie exact cele de mai sus. Absența sau denumirea diferită a acestora va avea drept consecință obținerea a 0 puncte pe testele asociate problemei rezolvate de regula respectivă.
 - **ATENȚIE!** Pentru cei ce folosesc C/C++ **NU** este permisă compilarea cu opțiuni de optimizare a codului (O1, O2, etc.).
 - **ATENȚIE!** Orice nerespectare a restricțiilor duce la pierderea punctajului (după regulile de mai sus).

8 LINKS

- [Regulament general PA](#)
- [Google C++ Style Guide](#)
- [Google Java Style Guide](#)
- [Debugging și Structuri de Date](#)