PA - TEMA 1
- APROAPE FARA GIGEL -

Responsabili:
Cristian Pătrascu, Andrei Preda, Ioan Pop,
Razvan Paroiu, Andreea Dutulescu, Robert Truță, Theo Pruteanu

Deadline soft: **23.04.2023 23:55**
Deadline hard: **26.04.2023 23:55**

CUPRINS

1 Problema 1: Feribot 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: Nostory 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 Schelet . . . 5
2.6 Testare și punctare . . . 5
2.7 Exemple . . . 6
3 Problema 3: Sushi 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 Schelet . . . 8
3.6 Testare și punctare . . . 8
3.7 Exemple . . . 9

4 Problema 4: Semnale 10
4.1 Enunț . . . 10
4.2 Date de intrare . . . 10
4.3 Date de ieșire . . . 10
4.4 Restricții și precizări . . . 10
4.5 Testare și punctare . . . 10
4.6 Exemple . . . 11
5 Problema 5 (bonus): BadGPT 12
5.1 Enunț . . . 12
5.2 Date de intrare . . . 12
5.3 Date de ieșire . . . 12
5.4 Restricții și precizări . . . 12
5.5 Testare și punctare . . . 13
5.6 Exemple . . . 13
6 Punctare 14
6.1 Checker . . . 14
7 Folosire ChatGPT 15
8 Format arhivă 15
9 Links 16

1 PROBLEMA 1: FERIBOT

1.1 Enunț

Un convoi de **N** masini, trebuie să traverseze un râu (păstrându-se ordinea din convoi) folosind feribotul, iar în total sunt disponibile **K** feriboturi. Stim despre a i-a mașină că are greutatea G_i. Un feribot poate să susțină oricâte mașini, însă costul de traversare al acestuia este este egal cu suma greutăților tuturor mașinilor pe care le transportă. Notăm cu C_i costul pentru feribotul i.

Ținând cont că cele K feriboturi trebuie să treacă toate mașinile pe partea opusă a râului, fiecare făcând un singur drum, care este cea mai mică valoare posibilă a lui **C**, unde **C** este maximul dintre $C_1, C_2, \ldots, C_K$? (Cu alte cuvinte, niciun feribot să nu aibă costul de traversare mai mare decat **C**)

1.2 Date de intrare

Fișierul de intrare **feribot.in** conține pe prima linie numerele **N** si **K**.

A doua linie conține **N** numere intregi, al i-lea număr reprezentând gretuatea celei de-a i-a mașini

1.3 Date de ieșire

Fișierul de ieșire **feribot.out** va conține un singur număr, reprezentând greutatea minimă calculată.

1.4 Restricții și precizări

- $1 \leqslant \mathbf{K} \leqslant \mathbf{N} \leqslant 10^5$
- $1 \leqslant G_i \leqslant 10^{12}$

1.5 Testare și punctare

- Punctajul maxim este de **25** puncte.
- Timpul de execuție:
  - C/C++: **0.5 s**
  - Java: **1 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **feribot.c**, **feribot.cpp** sau **Feribot.java**.

1.6 Exemple

feribot.in	feribot.out	Explicație
10 5 3 4 2 1 6 7 1 2 2 3	8	Pe primul feribot intra primele 2 masini (3+4=7), pe al doilea urmatoarele doua (2+1=3), pe al treilea urmatoarea masina (6), pe al patrulea urmatoarele doua (7+1=8), iar pe ultimul ultimele trei masini (2+2+3=7).

2 PROBLEMA 2: NOSTORY

2.1 Enunț

Se dau $2 * N$ numere **distincte**, împărțite în două liste de lungime **N**: **A** și **B**. Avem voie să facem mutări de tipul: alegem două numere din cele două liste, și le interschimbăm. Putem alege ambele numere din aceeași listă.

După ce facem mutările, calculăm un scor astfel: pentru fiecare pereche de numere care se găsesc pe aceeași poziție în **A** și **B**, păstrăm maximul dintre ele; suma maximelor este scorul. Ne dorim să **maximizăm** acest scor.

Această problemă are două subpuncte:

1. putem face un număr nelimitat de mutări.
2. putem face cel mult **K** mutări.

2.2 Date de intrare

Pe prima linie a fișierului **nostory.in** se află un număr **T**, care reprezintă subpunctul pe care îl rezolvăm.

Pe a doua linie se află numerele **N**, sau **N** și **K**, în funcție de subpunct.

Pe liniile 3 și 4 se află două liste de câte **N** numere: **A** și **B**.

2.3 Date de ieșire

În fișierul **nostory.out** se va afișa scorul maxim pe care îl putem obține.

2.4 Restricții și precizări

- $\mathbf{T} \in \{1, 2\}$
- $1 \leqslant \mathbf{K} \leqslant \mathbf{N} \leqslant 10^5$
- $1 \leqslant A_i, B_i \leqslant 10^9$

2.5 Schelet

Problema are și un schelet de cod de la care vă puteți începe implementarea. Îl puteți găsiți în arhiva cu checkerul, în directorul **skel/nostory**.

2.6 Testare și punctare

- Punctajul maxim este de **25** puncte.
- Timpul de execuție:

  - C/C++: **0.5 s**
  - Java: **1 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **nostory.c**, **nostory.cpp** sau **Nostory.java**.

2.7 Exemple

nostory.in	nostory.out	Explicație
1 3 1 5 10 6 3 9	25	Avem la dispoziție un număr nelimitat de mutări. Putem interschimba, de exemplu, pe 3 cu 9, apoi pe 3 cu 10. Listele finale vor fi: 1 5 3 6 9 10 Scorul final va fi 6 + 9 + 10 = 25.
2 5 2 3 6 10 9 5 1 8 4 7 2	40	Avem la dispoziție două mutări. Putem interschimba pe 3 cu 8, și pe 7 cu 2. Listele finale vor fi: 8 6 10 9 5 1 3 4 2 7 Scorul final va fi 8 + 6 + 10 + 9 + 7 = 40.

3 PROBLEMA 3: SUSHI

3.1 Enunț

Mergem să mâncăm sushi împreună cu **(N-1)** prieteni. La restaurant sunt disponibile **M** platouri de sushi, fiecare platou cu un cost C_i. Atât tu, cât și prieteni tăi aveți anumite preferințe la mâncare, așa că fiecare persoană a dat o notă de la 1 la 10 fiecărei variantă de sushi (Prietenul numărul **i** a dat platoului numărul **j** nota G_{ij}). De asemenea aveți și un buget destul de fix, așa că ați hotărât ca fiecare din voi să contribuie maxim **X** lei pentru masă (deci în total $N * X$ lei pentru toată nota de plată).

Tu ești responsabil să faci comanda și ai vrea să alegi meniurile astfel încât să maximizezi suma notelor date de tine și prietenii tăi platourilor comandate. Dacă un platou este comandat de exemplu de 2 ori, bineînțeles, notele pentru acel platou vor fi adunate de 2 ori.

Vrei să încerci mai multe restricții și să vezi ce variantă îți convine mai mult la final, așa că ai următoarele cazuri (cerințe) cu restricții suplimentare în plus față de cerintă inițială:

1. Pe lângă cerințele de mai sus, ai vrea să comanzi maxim 1 platou de fiecare tip de sushi.
2. Vrei să vezi ce se întâmplă dacă comanzi maxim 2 platouri din fiecare tip de sushi.
3. Tot vrei să comanzi doar maxim 2 platouri din fiecare tip de sushi, dar ai vrea și să comanzi cel mult **N** platouri în total, pentru a nu comanda prea multă mâncare.

3.2 Date de intrare

Pe prima linie a fișierului **sushi.in** se află un număr **C**, corespunzător cerinței care trebuie rezolvată. Pe a doua linie se află numerele **N**, **M** și **X**. Pe a treia linie se află **M** numere, C_1, C_2, , C_m. Pe următoarele **N** linii se află câte **M** numere, pe linia **i** aflându-se numerele G_{i1}, G_{i2}, , G_{im}.

3.3 Date de ieșire

În fișierul **sushi.out** se va afișa suma maximă posibilă a notelor.

3.4 Restricții și precizări

- $2 \leqslant \mathbf{N} \leqslant 10$

- $1 \leqslant \mathbf{M} \leqslant 1000$
- $1 \leqslant \mathbf{X} \leqslant 100$
- $1 \leqslant P_i \leqslant 100$
- $1 \leqslant G_i \leqslant 10$

3.5 Schelet

Problema are și un schelet de cod de la care vă puteți începe implementarea. Îl puteți găsiți în arhiva cu checkerul, în directorul **skel/sushi**.

3.6 Testare și punctare

- Punctajul maxim este de **30** puncte.
- Timpul de execuție:
  - C/C++: **1.5 s**
  - Java: **2.5 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **sushi.c**, **sushi** sau **Sushi.java**.

3.7 Exemple

sushi.in	sushi.out	Explicație
1 4 2 6 6 10 7 9 5 10 6 10 9 8	64	Comandăm câte un platou din ambele variante de sushi, cu preț $6+10 = 16 \leqslant N*X = 24$. Suma notelor este: $(7+5+6+9)+(9+10+10+8) = 64$
2 4 2 6 6 10 7 9 5 10 6 10 9 8	91	Comandăm 2 platouri din prima variantă și unul din a 2-a, cu preț $2*6+10 = 22 \leqslant N*X = 24$. Suma notelor este: $2*(7+5+6+9)+(9+10+10+8) = 91$
3 3 4 20 8 10 12 14 6 5 4 7 5 5 5 8 7 6 5 7	62	Comandăm 2 platouri din a 4-a variantă și 1 din prima variantă, cu preț $2*14+8 = 36 \leqslant N*X = 60$. Suma notelor este: $2*(7+8+7)+(6+5+7) = 62$. Nu putem comanda niciun alt platou pentru că am trece de limita de N = 3 platouri.

4 PROBLEMA 4: SEMNALE

4.1 Enunț

Gigel, student la ACS, a aflat de curând că orice semnal poate fi transmis ca o secvență consecutivă de biți. Curios din fire acesta se întreabă câte semnale diferite poate transmite folosind **X** biți de 0 și **Y** biți de 1. Deoarece o secvență consecutivă de biți de 1 poate fi interpretată drept începutul unui semnal nou, Gigel își propune sa analizeze doar 2 tipuri:

- Tipul 1: semnalul nu poate contine 2 sau mai mulți biți de 1 consecutivi
- Tipul 2: semnalul nu poate contine 3 sau mai mulți biți de 1 consecutivi

Ajutați-l pe Gigel sa calculeze numărul de semnale diferite pe care le poate transmite. Deoarece rezultatul poate fi foarte mare se va afișa în schimb restul împărțirii sale la 1000000007 ($10^9 + 7$).

4.2 Date de intrare

Fișierul de intrare **semnale.in** va conține pe prima linie cifra 1 sau 2 reprezentând tipul semnalului pe care vrea să-l analizeze la momentul curent. Cea de-a doua linie va conține 2 numere **X** și **Y** separate printr-un spațiu reprezentând numărul de biți de 0 respectiv numărul de biți de 1.

4.3 Date de ieșire

Fișierul de ieșire **semnale.out** va conține pe prima linie numărul de semnale diferite modulo 1000000007 ($10^9 + 7$).

4.4 Restricții și precizări

- $1 \leqslant \mathbf{X}, \mathbf{Y} \leqslant 2000$

4.5 Testare și punctare

- Punctajul maxim este de **30** puncte.
- Pentru **tipul 1** de semnale se acordă **15** puncte. Pentru **tipul 2** de semnale se acordă **15** puncte.
- Timpul de execuție:
  - C/C++: **0.5 s**
  - Java: **1 s**

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

4.6 Exemple

semnale.in	semnale.out	Explicație
1 4 2	10	Cele 10 semnale sunt: 000101, 001001, 010001, 100001, 001010, 010010, 100010, 010100, 100100, 101000.
2 2 3	7	Cele 7 semnale sunt: 01011, 10011, 01101, 11001, 10110, 11010, 10101.

5 PROBLEMA 5 (BONUS): BADGPT

5.1 Enunț

Gigel a decis să folosească noul tool EnunțGPT pentru a genera enunțuri la temele de la Politehnică. Acest tool nu reușește să genereze neaparat enunțuri interesate, dar folosind o nouă extensie (creată chiar de el) GPT2PDF preia enunțul și îl transformă în format PDF, numai bun pentru a putea fi publicat.

Problema lui Gigel este că extensia lui transformă enunțul în format PDF nu prin copierea textului, ci prin crearea unei imagini pe baza textului original. Acest tool însa nu reușește să proceseze corect literele **m** și **w**. Litera **m** va apărea în enunț ca **nn**, iar litera **w** va apărea ca **uu** (EnunțGPT preferă engleza). De exemplu, dacă în enunțul din PDF va apărea secvența de litere **anna**, atunci secvența originală putea să fie **anna** sau **ama**.

Gigel este curios să afle dacă extensia lui este bună sau nu. El vrea să știe, pornind de la șirul de caractere din PDF, câte șiruri distincte puteau sta la baza enuntului. Speriat de eventualul număr mare de posibilități, rezultatul se va afișa modulo $10^9 + 7$ (așa poate rezultatul devine 1 și Gigel va fi satisfăcut)

5.2 Date de intrare

Fișierul de intrare **badgpt.in** va conține pe prima linie un singur șir de caractere, comprimat sub forma $l_1n_1l_2n_2$ etc., unde secvența l_in_i sugerează ca litera l_i va apărea de n_i ori la rând.

Se garantează faptul că nu va exista aceeași literă pe două poziții consecuive. De exemplu, nu putem avea șirul codificat u3u4. Acesta va fi u7.

5.3 Date de ieșire

Fișierul de ieșire **badgpt.out** va conține pe prima linie numărul de șiruri distincte care puteau sta la baza enunțului, modulo $10^9 + 7$.

5.4 Restricții și precizări

- $1 \leqslant \mathbf{L}, n_i \leqslant 10^{18}$, unde **L** este lungimea șirului necomprimat.
- $1 \leqslant \mathbf{G} \leqslant 10^5$, unde **G** este numărul de grupuri comprimate. Un grup comprimat reprezintă o pereche l_in_i ce semnifică faptul că litera l_i va apărea de n_i ori la rând.
- Pentru teste in valoare de 10 puncte, $1 \leqslant \mathbf{L} \leqslant 2 * 10^6$, unde L este lungimea șirului necomprimat.

5.5 Testare și punctare

- Punctajul maxim este de **25** puncte.
- Pentru teste în valoare de 10 puncte, $1 \leqslant \mathbf{L} \leqslant 2 * 10^6$.
- Timpul de execuție:
  - C/C++: **2 s**
  - Java: **3 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **badgpt.c**, **badgpt.cpp** sau **Badgpt.java**.

5.6 Exemple

badgpt.in	badgpt.out	Explicație
a1c3n2	2	Codificarea corespunde șirului **acccnn**. Există 2 șiruri inițiale posibile: **acccnn** și **acccm**.

6 PUNCTARE

- Punctajul temei este de **125** puncte, distribuit astfel:
  - Problema 1: **25p**
  - Problema 2: **25p**
  - Problema 3: **30p**
  - 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 pană 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 **25p** rezolvând problema **Gigel regele comerțului**. 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 o 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.

6.1 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 | 5>
./check.sh <feribot | nostory | sushi | semnale | badgpt >
./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 in Java se recomanda folosirea **BufferedReader**.

7 FOLOSIRE CHATGPT

- Folosirea ChatGPT, Copilot sau a oricărui model de limbaj sau tool (denumit in continuare **LLM**) ce vă poate ajuta la rezolvarea temei, cu idei sau cod, este puternic descurajata, dar nu interzisă.
- În cazul in care folosiți LLM-uri, trebuie sa specificați acest lucru în README, precum si modul in 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 in README, acest lucru se va considera **încercare de copiere** si se va sancționa conform regulamentului.

8 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_Tema1.zip** (ex: 399CX_PuiuGigel_Tema1.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 4
- **run-p5**, 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:

- **feribot.c**, **feribot.cpp** sau **Feribot.java** - pentru problema 1
- **nostory.c**, **nostory.cpp** sau **Nostory.java** - pentru problema 2
- **sushi.c**, **sushi.cpp** sau **Sushi.java** - pentru problema 3
- **semnale.c**, **semnale.cpp** sau **Semnale.java** - pentru problema 4
- **badgpt.c**, **badgpt.cpp** sau **Badgpt.java** - pentru problema 5

- **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).

9 LINKS

- Regulament general PA
- Google C++ Style Guide
- Google Java Style Guide
- Debugging și Structuri de Date