

PA - TEMA 1 - GREEDY & DP

Responsabili:

Ștefania Budulan, Irina Niculescu, Iustin Sîrbu,
Isabella Mincă, Andrei Dinu

Deadline soft: **11.04.2018, 23:55**

Deadline hard: **18.04.2018, 23:55**

- 27.03.2018 22:00
 - [Clarificare cerință problema 2](#) (pag. 5)
 - [Modificare restricție problema 2](#) (pag. 5)
- 31.03.2018 23:00
 - Timpi de execuție pentru fiecare problemă

CUPRINS

1	Problema 1: Frați	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	4
1.6	Exemple	4
1.6.1	Exemplu 1	4
1.6.2	Exemplu 2	4
2	Problema 2: Urși	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
2.6.1	Exemplu 1	6
2.6.2	Exemplu 2	6
2.6.3	Exemplu 3	6

3	Problema 3: Planificare	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
3.6.1	Exemplu 1	8
3.6.2	Exemplu 2	8
4	Bonus: Numărătoarea	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
4.6.1	Exemplu 1	10
4.6.2	Exemplu 2	10
4.6.3	Exemplu 3	11
5	Punctare	12
5.1	Checker	12
6	Format arhivă	13
7	Links	14

1 PROBLEMA 1: FRAȚI

1.1 Enunț

În orașul Game City se organizează o serie de concursuri sportive. Doi frați, Sam și Jon sunt pasionați de sport și vor să participe la toate concursurile, însă la același concurs nu pot participa mai mulți membri din aceeași familie.

La fiecare concurs, câștigătorul primește ca premiu fie un număr de jocuri video (jocuri_i), fie un (alt) număr de benzi desenate (benzi_i).

Jon adoră jocurile video și detestă benzile desenate, iar Sam tocmai invers și de aceea fiecare ar vrea ca la finalul concursurilor să aibă **mai multe obiecte preferate** decât fratele său și **diferența** dintre numărul de obiecte preferate și numărul de obiecte detestate să fie **cât mai mare**, iar, dacă **nu poate** avea mai multe obiecte preferate decât fratele său, diferența dintre numărul de obiecte detestate și numărul de obiecte preferate să fie **cât mai mică**.

În cazul în care există mai multe posibilități ca la final să fie aceeași diferență, fiecare va dori să aibă **cât mai multe obiecte preferate**.

Înainte de începerea înscrierilor, cei doi stabilesc la ce concursuri să meargă fiecare astfel: fiecare alege un concurs până când nu mai rămâne niciun concurs de ales, iar Jon face primul alegerea, pentru că este mai fratele mai mare. De asemenea, fiind foarte încrezători în aptitudinile lor, ei fac alegerile în ipoteza că **vor câștiga concursurile la care participă**.

1.2 Date de intrare

Pe prima linie a fișierului **frati.in** se află un număr întreg N , reprezentând numărul de concursuri.

Pe următoarele N linii se află câte o pereche de numere întregi, jocuri_i și benzi_i , reprezentând numărul de jocuri, respectiv de benzi desenate pe care câștigătorul concursului cu numărul i le primește.

1.3 Date de ieșire

În fișierul **frati.out** se va scrie numărul de obiecte preferate pe care le-ar avea Jon, respectiv Sam, dacă ambii ar câștiga fiecare dintre concursurile la care decid să participe.

1.4 Restricții și precizări

- $1 \leq N \leq 10^6$
- $0 \leq \text{jocuri}_i, \text{benzi}_i \leq 10^3$

1.5 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: **frati.c**, **frati.cpp** sau **Frati.java**.

1.6 Exemple

1.6.1 Exemplu 1

Exemplu 1		
frati.in	frati.out	Explicație
6 10 4 2 8 7 9 3 5 12 10 6 4	28 22	<p>Alegerile vor fi următoarele: Jon alege în ordine concursurile 4, 0, 5, iar Sam alege în ordine concursurile 2, 1, 3 (numerotând concursurile de la 0 la $N - 1$ în ordinea apariției în fișier).</p> <p>Astfel, numărul total de jocuri video pe care le-ar primi Jon ar fi $12 + 10 + 6 = 28$, iar numărul total de benzi desenate pe care le-ar primi Sam ar fi $9 + 8 + 5 = 22$.</p>

1.6.2 Exemplu 2

Exemplu 2		
frati.in	frati.out	Explicație
4 4 12 9 15 41 26 15 10	50 22	<p>Jon alege în ordine concursurile 2, 1, iar Sam alege în ordine concursurile 3, 0.</p> <p>Astfel, numărul total de jocuri video pe care le-ar primi Jon ar fi $41 + 9 = 50$, iar numărul total de benzi desenate pe care le-ar primi Sam ar fi $10 + 12 = 22$.</p> <p>Sam a ales concursul 3, deși acesta îi aducea cele mai puține puncte. El știa, totuși, că e mai avantajos să nu îi lase lui Jon acel concurs (el a luat 10 puncte, urmând ca Jon să nu poată câștiga în continuare mai mult de 9 puncte, dar, dacă ar fi ales concursul 1, el ar fi primit 15 puncte, înșă și fratele său ar fi câștigat tot atât la următoarea tură, ceea ce ar fi fost mai dezavantajos pentru el.</p>

2 PROBLEMA 2: URȘI

2.1 Enunț

Martin și prietenii săi urși polari și-au creat un canal prin care să comunice online în scris. Urșilor le place să arate că sunt fericiți prin utilizarea fețelor zâmbitoare.

O față zâmbitoare este formată din 0 **sau mai multe** caractere **underscore** între **două** caractere **^**.

Cea mai scurtă versiune a unei fețe zâmbitoare este "^ ^". Șirurile de caractere "^ _ ^" și "^ _____ ^" reprezintă, de asemenea, fețe zâmbitoare valide.

Având o stare generală bună, Martin le-a trimis prietenilor săi o serie de fețe zâmbitoare, însă, din cauza unei probleme a canalului de comunicație, fețele zâmbitoare s-au intercalat rezultând un mesaj eronat.

Se dă un șir format numai din caracterele "^" și "_" reprezentând mesajul eronat primit de prietenii lui Martin. Se cere să se găsească în **câte moduri diferite** poate fi împărțit mesajul eronat în una sau mai multe fețe zâmbitoare. Fiecare față zâmbitoare extrasă trebuie să fie un **subșir** al mesajului și fiecare caracter al mesajului **trebuie** să apară în **exact o față zâmbitoare**.

2.2 Date de intrare

Datele de intrare se vor citi din fișierul **ursi.in**.

Acesta va conține, pe prima linie, șirul de caractere ce reprezintă mesajul primit de urși.

2.3 Date de ieșire

În fișierul **ursi.out** se va afișa pe prima linie numărul de posibilități de împărțire a mesajului în fețe zâmbitoare, conform cerinței. Numărul poate fi destul de mare; așadar, se va afișa **modulo** ($10^9 + 7$).

2.4 Restricții și precizări

Lungimea mesajului este cuprinsă între 1 și 2000 de caractere, inclusiv.

2.5 Testare și punctare

- Punctajul maxim este de **55** puncte.
- Timpul de execuție:
 - C/C++: **0.2 s**

– Java: **0.6 s**

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

2.6 Exemple

2.6.1 Exemplu 1

Exemplu 1		
ursi.in	ursi.out	Explicație
^ _ ^ _ ^	3	<p>Șirul se poate împărți fie în 2 fețe zâmbitoare de tipul ^ _ ^, fie într-una de tipul ^ _ _ și una de tipul ^ ^.</p> <p>Există 3 modalități de a împărți șirul "^ _ ^ _ ^" în 2 fețe zâmbitoare (indentare pentru claritate) :</p> <pre> ^ _ ^ ^ _ ^ sau ^ _ ^ ^ _ ^ sau ^ _ _ ^ ^^ </pre>

2.6.2 Exemplu 2

Exemplu 2		
ursi.in	ursi.out	Explicație
^ _ ^ _ ^	0	Fiind un număr impar de caractere ^, șirul de caractere nu poate fi împărțit în fețe zâmbitoare.

2.6.3 Exemplu 3

Exemplu 3		
ursi.in	ursi.out	Explicație
^^^ _ ^^^	27	<p>Singurul underscore trebuie să facă parte dintr-o față zâmbitoare. Aceasta față zâmbitoare trebuie să conțină un ^ dintre primele 3 și un ^ dintre ultimele 3.</p> <p>Rezultă, astfel, 9 combinații posibile.</p> <p>Pentru fiecare dintre aceste 9 combinații, restul de 4 caractere ^ se pot combina în 3 moduri, în 2 fețe zâmbitoare de tipul ^ ^.</p>

3 PROBLEMA 3: PLANIFICARE

3.1 Enunț

Clark este organizatorul concursurilor sportive din orașul Game City.

El primește o listă cu duratele a P probe în minute ($durate_i$). Între fiecare 2 probe există o pauză de B minute, iar un concurs durează fix D minute.

Clark trebuie să distribuie probele în concursuri astfel încât fiecare probă să facă parte dintr-un **singur concurs**, probele să fie planificate **în ordinea de pe listă**, iar **pierderile totale să fie cât mai mici**.

Pierdere asociată unui concurs se definește ca fiind cubul timpului scurs, în minute, de la finalul ultimei probe din concurs până la începutul concursului următor. Între 2 concursuri nu există pauză. Pierderile totale reprezintă suma pierderilor asociate concursurilor.

3.2 Date de intrare

Datele vor fi citite din fișierul **planificare.in**.

Pe prima linie se vor afla numărul total de probe P ce trebuie planificate, durata D a unui concurs, în minute, și durata B a pauzei între 2 probe, în minute.

Pe următoarele P linii se află câte un număr natural, $durate_i$, reprezentând durata probei i în minute ($0 \leq j \leq P - 1$).

3.3 Date de ieșire

Ieșirea va fi afișată în fișierul **planificare.out**.

Acesta va conține 2 numere naturale separate prin spațiu: pierderile totale L și numărul N de concursuri rezultate.

3.4 Restricții și precizări

- $1 \leq P \leq 1000$
- $1 \leq D \leq 1000$
- $1 \leq durate_i \leq D$
- $1 \leq B \leq D$

3.5 Testare și punctare

- Punctajul maxim este de **27.5** puncte.
- Timpul de execuție:
 - C/C++: **0.015 s**
 - Java: **0.5 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **planificare.c**, **planificare.cpp** sau **Planificare.java**.

3.6 Exemple

3.6.1 Exemplu 1

Exemplu 1		
planificare.in	planificare.out	Explicație
4 7 2 3 2 2 5	65 3	<p>Distribuind probele astfel: primul concurs: proba 0, al doilea concurs: probele 1 și 2, al treilea concurs: proba 3, se obține un cost total de 65:</p> $(7 - 3)^3 + (7 - 2 - 2 - 2)^3 + 0^3 = 4^3 + 1^3 = 65$ <p>Proba 1 ar fi încăput, alături de proba 0, în primul concurs, dar apoi proba 2 ar fi rămas singură și genera pierderi prea mari.</p>

3.6.2 Exemplu 2

Exemplu 2		
planificare.in	planificare.out	Explicație
8 12 4 6 5 2 5 4 2 3 4	568 5	<p>Distribuind probele astfel: primul concurs: proba 0, al doilea concurs: probele 1 și 2, al treilea concurs: proba 3, al patrulea concurs: probele 4 și 5, și ultimul concurs: probele 6 și 7; se obține un cost total de 568:</p> $(12 - 6)^3 + (12 - 5 - 4 - 2)^3 + (12 - 5)^3 + (12 - 4 - 4 - 2)^3 + 0^3 = 6^3 + 1^3 + 7^3 + 2^3 = 568$

4 BONUS: NUMĂRĂTOAREA

4.1 Enunț

Într-o zi de primăvară, după ce au rezolvat toate problemele propuse, Paula și Pavel au găsit un nou joc la care să se gândească. Fiind date două numere, **s (suma)** și **n (numărul de poziții)**, ei găsesc că există mai multe moduri în care pot exprima **s**, ca sumă de **n** numere întregi, strict pozitive.

Există, însă, câteva restricții. Două sume sunt considerate diferite doar dacă au cel puțin un termen diferit. De exemplu, pentru $s = 7$, $n = 3$; $7 = 4 + 2 + 1$ și $7 = 1 + 4 + 2$ nu sunt considerate distincte.

În plus, doresc să aranjeze descrescător termenii fiecărei sume. Așadar, în loc să scrie $7 = 2 + 4 + 1$, vor prefera $7 = 4 + 2 + 1$.

După ce au găsit toate variantele diferite prin care pot exprima suma **s** cu **n** termeni, Paula și Pavel le așează într-o **ordine descrescătoare**, ținând cont de fiecare termen în parte. Astfel, dacă primul termen din fiecare sumă este același, se vor uita la cel de-al doilea etc. Suma care are un termen (primul găsit, în ordinea de la stânga la dreapta) distinct mai mare, se va afla mereu înaintea celorlalte. Pentru exemplul anterior, obținem ordinea:

$$7 = 5 + 1 + 1$$

$$7 = 4 + 2 + 1$$

$$7 = 3 + 3 + 1$$

$$7 = 3 + 2 + 2$$

Având această ordine în minte, cei doi prieteni vă roagă să îi ajutați să găsească șirul de la un anumit **indice i**. Prima exprimare (variantă) a sumei începe de la **indicele 0**.

Fiind date **s**, **n** și **i**, în fișierul de intrare, va trebui să scrieți, în fișierul de ieșire, un șir de forma

$$"< suma>=< termen_1>+< termen_2>+...+< termen_n>",$$

care s-ar afla pe poziția **i** în exprimarea sumelor ordonate descrescător după termeni. Dacă indicele este prea mare pentru a găsi o sumă validă, atunci rezultatul este șirul "-" (semnul minus).

4.2 Date de intrare

Datele vor fi citite din fișierul **numaratoare.in**.

Pe fiecare dintre cele 3 linii vor fi citite, în ordine, valorile **s** (suma), **n** (numărul de poziții) și **i** (indicele exprimării căutate).

4.3 Date de ieșire

Outputul va fi scris în fișierul **numaratoare.out**.

Rezultatul va fi scris pe o singură linie, de forma șirului ce exprimă suma de indice i , sau șirul "-", dacă nu s-a găsit soluție.

4.4 Restricții și precizări

- $1 \leq s \leq 150$
- $1 \leq n \leq 150$
- $0 \leq i \leq 2 * 10^9$
- Presupunem că numărul total de variante în care putem exprima suma nu va depăși, nici el, $2 * 10^9$.

4.5 Testare și punctare

- Punctajul maxim este de **25** puncte.
- Timpul de execuție:
 - C/C++: **0.4 s**
 - Java: **0.4 s**
- Sursa care conține funcția **main** trebuie obligatoriu denumită: **numaratoare.c**, **numaratoare.cpp** sau **Numaratoare.java**.

4.6 Exemple

4.6.1 Exemplu 1

Exemplu 1		
numaratoare.in	numaratoare.out	Explicație
7 3 2	7=3+3+1	Din exemplul desfășurat de mai sus, putem selecta cea de-a treia exprimare a sumei (de indice 2).

4.6.2 Exemplu 2

Exemplu 2		
numaratoare.in	numaratoare.out	Explicație
7 3 6	-	În exemplul de mai sus există doar 4 exprimări valide ale valorii 7, ca sumă de 3 termeni, deci indicele 6 este prea mare.

4.6.3 *Exemplu 3*

Exemplu 3		
numaratoare.in	numaratoare.out	Explicație
9 9 0	$9=1+1+1+1+1+1+1+1+1$	Există o singură exprimare a lui 9 ca sumă de 9 termeni strict pozitivi. Din fericire, ni se cere suma de indice 0 - singurul indice valid, de altfel.

5 PUNCTARE

- Punctajul temei este de 125 puncte, distribuit astfel:
 - Problema 1: 30p
 - Problema 2: 55p
 - Problema 3: 27.5p
 - 5 puncte vor fi acordate pentru coding style
 - 7.5 puncte vor fi acordate pentru comentarii și README

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

Se poate obține un bonus de 25p rezolvând problema 4: **Numărătoarea**. Acordarea bonusului **NU** este condiționată de rezolvarea celorlalte probleme. În total, se pot obține 150 de puncte (**NU** se trunchiază).

Pentru detalii, vă rugăm să citiți **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.
- **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 corectare 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.

6 FORMAT ARHIVĂ

- Temele pot fi testate automat pe vmchecker. Acesta suportă temele rezolvate în C/C++ și Java. Dacă doriți să realizați tema în alt limbaj, trebuie să-i trimiteți un email lui Traian Rebedea (traian.rebedea@cs.pub.ro), în care să îi cereți explicit acest lucru.
- Arhiva cu rezolvarea temei trebuie să fie **.zip**, având un nume de forma **Grupa_NumePrenume_Tema1.zip** (ex: 399CX_PuiuGigel_Tema1.zip sau 399CX_BucurGigel_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 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:
 - **frati.c, frati.cpp** sau **Frati.java** - pentru problema 1
 - **ursi.c, ursi.cpp** sau **Ursi.java** - pentru problema 2
 - **planificare.c, planificare.cpp** sau **Planificare.java** - pentru problema 3
 - **numaratoare.c, numaratoare.cpp** sau **Numaratoare.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).

7 LINKS

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