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 că 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, $\mathbf{jocuri_i}$ și $\mathbf{benzi_i}$, 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

- $1 \leqslant N \leqslant 10^6$
- $0 \leqslant jocuri_i, benzi_i \leqslant 10^3$

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	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). 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.

1.6.2 *Exemplu 2*

Exemplu 2		
frati.in	frati.out	Explicație
4 4 12 9 15 41 26 15 10	50 22	Jon alege în ordine concursurile 2, 1, iar Sam alege în ordine concursurile 3, 0. 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. 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, însa și fratele său ar fi câștigat tot atât la următoarea tură, ceea ce ar fi fost mai dezavantajos pentru el.

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**.

Pierderea 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, $\mathbf{durate_i}$, reprezentând durata probei i în minute ($0 \leqslant j \leqslant P-1$).

3.3 Date de ieșire

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

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

3.4 Restricții și precizări

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

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	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: $(7-3)^3 + (7-2-2-2)^3 + 0^3 = 4^3 + 1^3 = 65$ 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.

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	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 si 7; se obține un cost total de 568: $(12-6)^3 + (12-5-4-2)^3 + (12-5)^3 + (12-4-4-2)^3 + 0^3 = 6^3 + 1^3 + 7^3 + 2^3 = 568$

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 si 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 gasit soluție.

4.4 Restricții și precizări

- $1 \leqslant s \leqslant 150$
- $1 \leqslant n \leqslant 150$
- $0 \leqslant i \leqslant 2 * 10^9$
- 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.