Arrayser

Vous et votre ami jouez à votre jeu favori, appelé *Arrayser*. Comme tous les bons jeux, Arrayser nécessite un tableau A de N éléments, initialisés à $A[i] = 1$ pour chaque $0 \leq i \leq N-1$. Vous et votre ami jouez tour à tour, et vous commencez. À chaque tour :

1. Le joueur sélectionne un indice i avec $A[i] = 1$.
2. $A[i]$ est changé en 0. De plus,
   - Si $i \neq 0$, $A[i-1]$ est changé en 0.
   - Si $i \neq N-1$, $A[i+1]$ est changé en 0.

   Notez que $A[i-1]$ et $A[i+1]$ peuvent déjà avoir été changés en 0 lors d'un tour précédent.

Le jeu se termine lorsque $A[i] = 0$ pour chaque i.

Après avoir joué plusieurs parties, vous réalisez que votre ami joue complètement aléatoirement ! Plus précisément, il sélectionne toujours un indice i de manière uniformément aléatoire parmi tous ceux pour lesquels $A[i] = 1$.

Contrairement à votre ami, vous adorez jouer à Arrayser et voulez que chaque partie dure le plus longtemps possible. Dans ce problème, vous allez jouer $T = 5000$ parties, et votre objectif est de maximiser la moyenne [1] du nombre de tours que dure chaque partie.

Détails d'implémentation

Dans ce problème, **vous ne devez pas lire l'entrée standard, ni écrire sur la sortie standard.** Vous ne devez interagir avec aucun fichier. N'implémentez pas de fonction `main`. À la place, commencez votre programme par inclure le fichier d'en-tête `arrayser.h` (`#include "arrayser.h"`) et interagissez avec lui comme décrit ci-dessous.

Fonctions

Votre solution va jouer T parties pour chaque test. Vous devez implémenter `play_arrayser`, qui est appelée une fois par partie :

```
void play_arrayser(int N);
```

- `N` est la taille du tableau.
- Cette fonction doit appeler `do_turn` une fois pour chacun de vos tours.
- Cette fonction doit se terminer lorsque la partie se termine.
- Cette fonction sera appelée T fois pour chaque test. Nous recommandons de réinitialiser les variables globales à chaque fois qu'elle est appelée.

Depuis `play_arrayser`, vous pouvez faire des appels à `do_turn`:

```
int do_turn(int i);
```

- i doit être un indice $0 \leq i \leq N-1$ tel que $A[i] = 1$.
- Après avoir joué votre tour, votre ami jouera le sien. Cette fonction renvoie l'indice i que votre ami a choisi aléatoirement. Si votre tour était le dernier tour de la partie, cette fonction renvoie -1 à la place.

[1] La *moyenne* est la somme des valeurs divisée par le nombre de valeurs. Par exemple, la moyenne de $[3, 5, 4, 8]$ est $\frac{3+5+4+8}{4} = \frac{20}{4} = 5$.

Si vous ne respectez pas l'une des conditions ci-dessus, votre programme sera jugé incorrect, et vous recevrez 0% des points sur le test.

Sous-tâches et contraintes

Ce problème a une seule sous-tâche, qui a un seul test (notez que la limite de temps est élevée). Ce test consiste en $T = 5000$ parties, qui ont toutes $N = 300$. Soit S le nombre moyen de tours joués lors de ces parties (ce qui inclut vos tours et ceux de votre ami).

- Si $S > 144.2$, vous obtiendrez 100 points.
- Si $137 \leq S \leq 144.2$, vous obtiendrez entre 40 et 100 points selon une fonction affine. Autrement dit, vous obtiendrez $40 + 60 \times \frac{S-137}{144.2-137}$ points.
- Si $S < 137$, vous obtiendrez $40 \times \left(\frac{S}{137}\right)^{10}$ points.

Vous trouverez un tableau des points obtenus pour quelques valeurs de S ci-dessous.

S	100	110	120	130	136	138	140	142	144
Points	1.72	4.45	10.63	23.67	37.17	48.33	65	81.67	98.33

Expérimentation

Pour expérimenter avec votre code sur votre machine, commencez par télécharger les fichiers fournis `arrayser.cpp`, `arrayser.h` et `grader.cpp`.

Vous devez modifier `arrayser.cpp`, qui contient un exemple simple d'implémentation.

Compilez votre solution avec :

```
g++ -std=c++20 -O2 -Wall arrayser.cpp grader.cpp -o arrayser
```

Cela crée un exécutable `arrayser` que vous pouvez lancer avec `./arrayser`. Si vous avez des problèmes pour compiler, veuillez envoyer un message dans la section "Communication" du site web du concours.

L'évaluateur fourni (grader) lit depuis l'entrée standard au format suivant :

- L'unique ligne de l'entrée contient N et T.

Exemple d'entrée de l'évaluateur

L'entrée suivante va lancer $T = 10$ parties, chacune avec $N = 300$:

```
300 10
```

Graine aléatoire

L'évaluateur fourni définit une valeur `RANDOM_SEED` à la ligne 9. Si cette variable est modifiée, l'évaluateur fera des choix aléatoires différents. L'évaluateur utilisé pour le calcul de votre score utilisera une graine aléatoire différente de celle dans l'évaluateur fourni.

Sortie de débogage de l'évaluateur

L'évaluateur fourni définit une valeur `DEBUG_LEVEL` à la ligne 10, qui vaut `1` par défaut :

- Si `DEBUG_LEVEL = 0`, l'évaluateur affichera le nombre moyen de tours utilisés après que les T parties soient terminées.

- Si `DEBUG_LEVEL = 1`, l'évaluateur affichera le nombre de tours utilisés pour chaque partie. Il affichera aussi la moyenne une fois toutes les parties terminées.
- Si `DEBUG_LEVEL = 2`, l'évaluateur affichera après chaque tour les informations sur celui-ci. En particulier, l'évaluateur affichera l'indice choisi par votre code, l'indice choisi par l'évaluateur, ainsi que l'état actuel du tableau A. Il affichera aussi la moyenne une fois toutes les parties terminées.

Inondation

Le Massif Bitwise contient N montagnes, numérotées 1 à N de gauche à droite. La i-ème montagne est large de 1 kilomètre et est haute de H_i kilomètres. Chaque montagne possède un village.

Le gouvernement est en train de planifier Q scénarios. Dans le i-ème scénario, W_i averses massives vont avoir lieu sur la première montagne. Chaque averse délivre assez d'eau pour submerger un village sous 1 kilomètre d'eau. Une fois que l'eau a atteint la première montagne, elle va s'écouler de la façon suivante :

- Si l'eau peut couler vers le bas, alors elle s'écoule vers le bas.
- Sinon, si elle peut s'écouler vers la droite, alors elle s'écoule à droite.
- Sinon, si elle ne peut ni s'écouler vers le bas, ni vers la droite, alors elle ne s'écoule pas.

Si l'eau s'écoule au delà de la N-ème montagne, alors ses mouvements futurs seront ignorés.

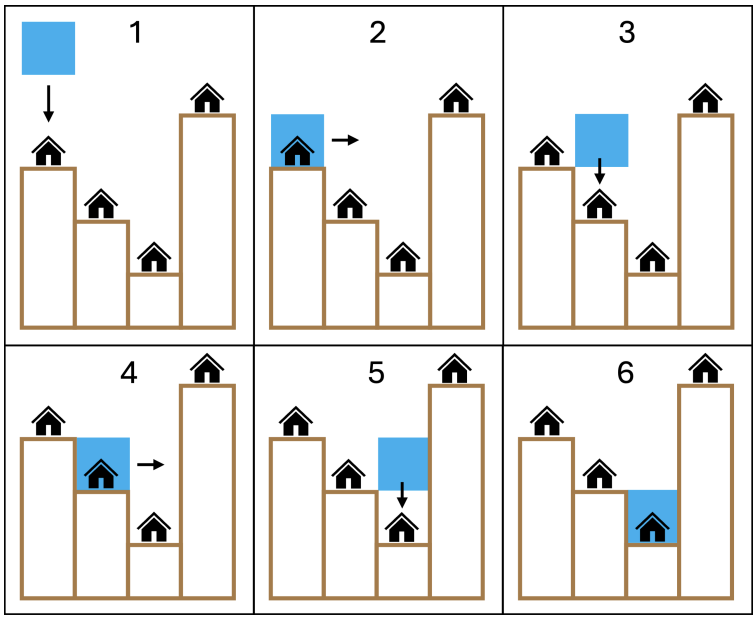

Figure 1: Un exemple d'averse. La pluie arrive initialement à la première montagne et arrive éventuellement au dessus de la 3-ème montagne. Chaque image représente une étape du processus.

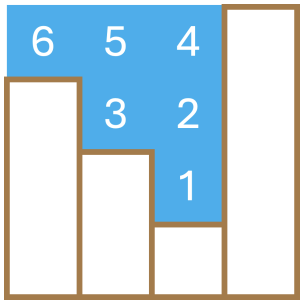

Figure 2: Un exemple de scénario avec $W_i = 6$ averses. Les positions finales des 6 averses sont numérotées dans l'image.

On dit qu'un village est *inondé* si de l'eau traverse à un moment ce village, **même si l'eau ne s'arrête pas dans ce village**.

Dans chaque scénario, le gouvernement a un budget de K dollars. Pour un dollar, le gouvernement peut augmenter la hauteur d'une montagne de 1 kilomètre. En dépensant correctement ces K dollars, le gouvernement espère minimiser le nombre de villages qui sont inondés. On vous a demandé de calculer le nombre de villages minimum qui seront inondés si le gouvernement dépense optimalement les K dollars avant que les averses commencent.

Les Q scénarios sont indépendants, et les hauteurs retournent à leurs valeurs initiales avant que le scénario suivant ne commence. Cependant, le budget K est le même dans tous les scénarios.

Sous-tâches et contraintes

Pour toutes les sous-tâches :

- $2 \leq N \leq 200\,000$.
- $1 \leq Q \leq 200\,000$.
- $0 \leq K \leq 1\,000\,000\,000$.
- $1 \leq H_i, W_i \leq 1\,000\,000\,000$ pour tout i.

Les contraintes additionnelles pour les sous-tâches sont listées ci-dessous.

Sous-tâche	Points	Contraintes additionnelles
1	17	$N, Q \leq 50$ et $K = 0$.
2	20	$N, Q \leq 50$.
3	18	$N \leq 2000$.
4	21	$K = 0$.
5	24	Pas de contraintes additionnelles.

Entrée

- La première ligne de l'entrée contient un entier N.
- La deuxième ligne contient N entiers $H_1, H_2, \ldots, H_N$.
- La troisième ligne contient les entiers Q et K.
- La quatrième ligne contient Q entiers $W_1, W_2, \ldots, W_Q$.

Sortie

Affichez Q entiers sur une seule ligne : Les résultats pour les Q scénarios.

Exemple d'entrée 1

```
4
3 2 1 4
4 0
1 6 7 1000
```

Exemple de sortie 1

```
3 3 4 4
```

Exemple d'entrée 2

```
4
3 2 1 4
3 2
1 7 1000
```

Exemple de sortie 2

```
1 3 4
```

Exemple d'entrée 3

```
7
2 1 1 3 1 2 1
4 1
10 9 8 1
```

Exemple de sortie 3

```
7 5 3 2
```

Explication

Le premier exemple est affiché dans la figure 1 et la figure 2 plus tôt dans le sujet. Le gouvernement a un budget de $K = 0$ dollars et ne peut pas élever de montagnes :

- Quand $W_1 = 1$ et $W_2 = 6$, les trois premiers villages sont inondés.
- Quand $W_3 = 7$ et $W_4 = 1000$, les quatre villages sont inondés.

Le deuxième exemple de cas de test a les mêmes montagnes que le premier cas de test, mais le budget est $K = 2$:

- Quand $W_1 = 1$, le gouvernement peut élever la deuxième montagne de 2 kilomètres, pour qu'elle ait une hauteur de 4 kilomètres. Maintenant, seul le premier village est inondé.
- Quand $W_2 = 7$, le gouvernement peut élever la première et la quatrième montagne de 1 kilomètre chacune. Dans ce cas, seuls les trois premiers villages seront inondés.
- Quand $W_3 = 1000$, les quatre villages seront inondés quelle que soit la manière dont le budget sera dépensé.

Le troisième exemple de cas de test est affiché dans la figure 3 ci-dessous. Le gouvernement a un budget de $K = 1$:

- Quand $W_1 = 10$, les sept villages seront inondés quelle que soit la manière dont le budget sera dépensé.
- Quand $W_2 = 9$, le gouvernement peut élever la quatrième montagne d'un kilomètre, de façon à ce que seuls les cinq premiers villages soient inondés.
- Quand $W_3 = 8$, le gouvernement peut élever la quatrième montagne d'un kilomètre, de façon à ce que seuls les trois premiers villages soient inondés.
- Quand $W_4 = 1$, le gouvernement peut élever la troisième montagne d'un kilomètre, de façon à ce que seuls les deux premiers villages soient inondés.

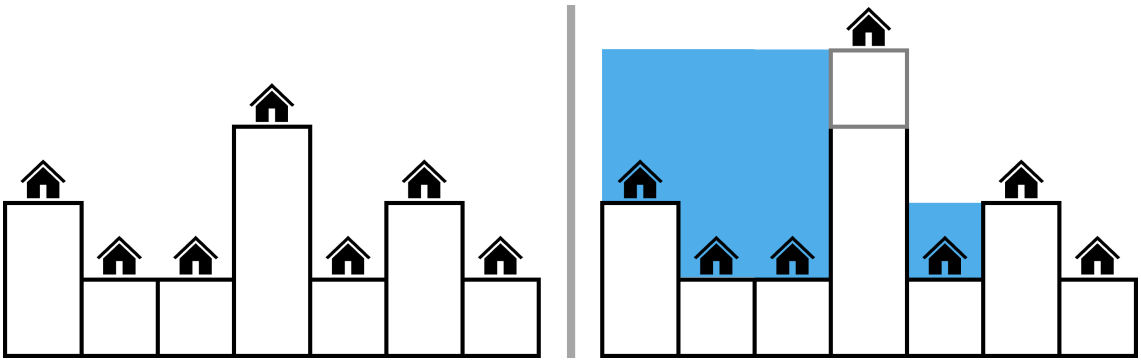

Figure 3: Exemple de cas de test 3. Dans l'image de gauche sont affichées les hauteurs des montagnes. Sur la droite de l'image est affiché le deuxième scénario avec $W_2 = 9$, et où la quatrième montagne a été élevée d'un kilomètre. Les cinq premiers villages sont inondés.

Tree Dash

Vous avez trouvé un *arbre enraciné*. Cet arbre est constitué de N sommets, numérotés de 1 à N. Un de ces sommets, numéroté R, est la *racine* de l'arbre.

Chaque sommet i autre que la racine a un parent P_i, où $P_i \neq i$. Si vous partez d'un sommet et vous vous déplacez à son parent, puis au parent de son parent, ainsi de suite, vous finirez par atteindre la racine. Les sommets rencontrés sur ce chemin sont appelés les *ancêtres* de i.

Le sommet i est un *enfant* du sommet P_i. Les enfants d'un sommet i, les enfants de ses enfants, ainsi de suite, sont appelés les *descendants* de i.

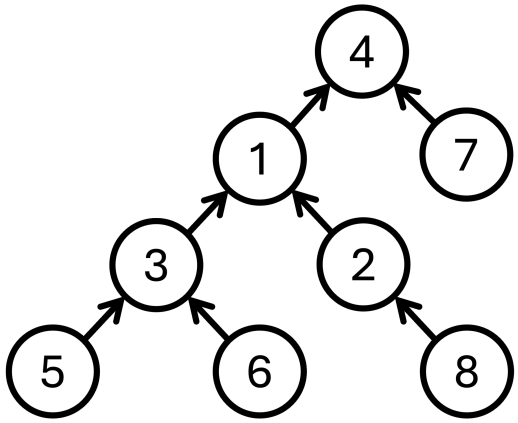

Figure 1: Un exemple d'arbre avec $N = 8$. La racine est $R = 4$ et chaque sommet (sauf la racine) a une flèche vers son parent. Le sommet 1 a un ancêtre (4) et cinq descendants (2, 3, 5, 6, 8), tandis que le sommet 8 a trois ancêtres (1, 2, 4) et aucun descendant. La racine n'a pas d'ancêtre et a $N - 1$ descendants.

Si vous êtes au sommet i dans l'arbre, vous pouvez aller à l'un des sommets suivants :

- L'ancêtre de i avec le plus petit numéro (si i a au moins un ancêtre).
- L'ancêtre de i avec le plus grand numéro (si i a au moins un ancêtre).
- Le descendant de i avec le plus petit numéro (si i a au moins un descendant).
- Le descendant de i avec le plus grand numéro (si i a au moins un descendant).

Un sommet u peut atteindre un autre sommet v s'il est possible d'aller de u à v, en passant potentiellement par des sommets intermédiaires.

Chaque sommet a un *poids* W_i. Calculez la somme des $W_u \times W_v$ parmi toutes les paires ordonnées de sommets (u, v) telles que $u \neq v$ et le sommet u peut atteindre v.

Sous-tâches et Contraintes

Dans toutes les sous-tâches :

- $2 \leq N \leq 500\,000$.
- $1 \leq R \leq N$.
- $1 \leq P_i \leq N$ pour tout $i \neq R$.
- $1 \leq W_i \leq 1000$ pour tout i.

- Si vous partez d'un sommet et vous vous déplacez à son parent, puis au parent de son parent, ainsi de suite, vous finirez par atteindre la racine.

Les contraintes additionnelles pour les sous-tâches sont listées ci-dessous.

Sous-tâche	Points	Contraintes additionnelles
1	20	$N \leq 100$ et l'arbre est une *ligne* (voir $*$ ci-dessous).
2	15	$N \leq 3000$.
3	25	L'arbre est une ligne (voir $*$ ci-dessous).
4	20	La racine est 1 et elle a un seul enfant. L'unique enfant de la racine est N. Autrement dit, $R = 1$, $P_N = 1$, et $P_i \neq 1$ pour tout $2 \leq i \leq N - 1$.
5	20	Pas de contraintes additionnelles.

$*$: une *ligne* est un arbre où chaque sommet a au plus 1 enfant. L'exemple 2 est une ligne.

Entrée

- La première ligne de l'entrée contient les deux entiers N et R.
- La deuxième ligne contient N entiers $P_1, P_2, \ldots, P_N$. Vu que R n'a pas de parent, P_R est remplacé par un 0.
- La troisième ligne contient N entiers $W_1, W_2, \ldots, W_N$.

Sortie

Affichez un seul entier, la somme des $W_u \times W_v$ parmi toutes les paires $u \neq v$ telles que le sommet u peut atteindre v.

Remarque: Votre solution peut avoir besoin de grands entiers. Vous aurez peut-être besoin d'utiliser des entiers 64-bits ('long long' en C++) pour votre solution.

Exemple d'entrée 1

```
8 4
4 1 1 0 3 3 4 2
1 1 1 1 1 1 1 1
```

Exemple de sortie 1

```
30
```

Exemple d'entrée 2

```
6 5
6 5 4 2 0 3
1 2 3 4 5 6
```

Exemple de sortie 2

```
228
```

Exemple d'entrée 3

```
12 1
0 3 12 3 6 12 8 2 8 12 5 1
70 92 86 72 22 79 20 98 42 56 76 34
```

Exemple de sortie 3

```
228776
```

Explication

L'exemple 1 est illustré dans la figure 1 sur la première page :

1. Le sommet 1 peut atteindre les sommets $2, 4, 8$.
2. Le sommet 2 peut atteindre les sommets $1, 4, 8$.
3. Le sommet 3 peut atteindre les sommets $1, 2, 4, 5, 6, 8$.
4. Le sommet 4 peut atteindre les sommets $1, 2, 8$.
5. Le sommet 5 peut atteindre les sommets $1, 2, 4, 8$.
6. Le sommet 6 peut atteindre les sommets $1, 2, 4, 8$.
7. Le sommet 7 peut atteindre les sommets $1, 2, 4, 8$.
8. Le sommet 8 peut atteindre les sommets $1, 2, 4$.

L'exemple 2 est une ligne illustrée dans la figure 2 :

1. Le sommet 1 peut atteindre les sommets $2, 5, 6$.
2. Le sommet 2 peut atteindre les sommets $1, 5, 6$.
3. Le sommet 3 peut atteindre les sommets $1, 2, 5, 6$.
4. Le sommet 4 peut atteindre les sommets $1, 2, 5, 6$.
5. Le sommet 5 peut atteindre les sommets $1, 2, 6$.
6. Le sommet 6 peut atteindre les sommets $1, 2, 5$.

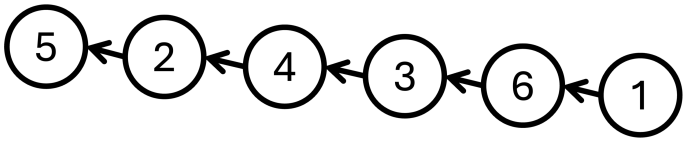

Figure 2: Exemple 2. Chaque sommet vérifie $W_i = i$.

L'exemple 3 est un arbre satisfaisant les contraintes de la sous-tâche 4, illustré dans la figure 3 :

1. Le sommet 1 peut atteindre les sommets $2, 7, 9, 11, 12$.
2. Le sommet 2 peut atteindre les sommets $1, 7, 9, 11, 12$.
3. Le sommet 3 peut atteindre les sommets $1, 2, 7, 9, 11, 12$.
4. Le sommet 4 peut atteindre les sommets $1, 2, 7, 9, 11, 12$.
5. Le sommet 5 peut atteindre les sommets $1, 2, 7, 9, 11, 12$.
6. Le sommet 6 peut atteindre les sommets $1, 2, 5, 7, 9, 11, 12$.
7. Le sommet 7 peut atteindre les sommets $1, 2, 9, 11, 12$.
8. Le sommet 8 peut atteindre les sommets $1, 2, 7, 9, 11, 12$.
9. Le sommet 9 peut atteindre les sommets $1, 2, 7, 11, 12$.
10. Le sommet 10 peut atteindre les sommets $1, 2, 7, 9, 11, 12$.
11. Le sommet 11 peut atteindre les sommets $1, 2, 7, 9, 12$.
12. Le sommet 12 peut atteindre les sommets $1, 2, 7, 9, 11$.

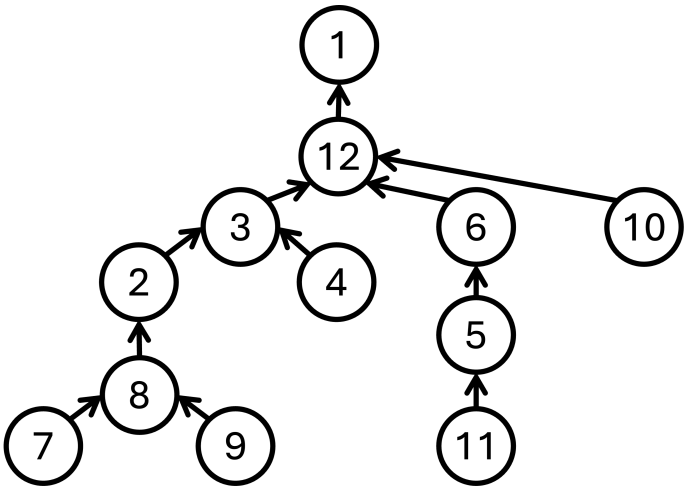

Figure 3: Exemple 3.