Breadcrumbs

Soucieux de prendre l'avantage avant le concours de mangeurs de pain de l'année prochaine, vous avez envoyé votre espionne, Amelia, infiltrer une ville rivale. Après des mois d'observation attentive, Amelia a enfin découvert leurs techniques secrètes. La moindre erreur la démasquera immédiatement, elle doit donc être extrêmement prudente lorsqu'elle vous transmet des informations.

Toute communication directe est impossible. Elle transmettra un message en parcourant la ville et en semant des indices (miettes de pain). Vous devez reconstituer le message caché en observant son parcours.

La ville compte N lieux, numérotés de 1 à N, et M rues reliant chacune deux lieux différents. Vous pouvez emprunter les rues dans les deux sens, et il est possible de se déplacer entre tous les lieux grâce aux rues.

Le parcours d'Amelia est une séquence de lieux V_1, V_2, ..., V_L telle que pour tout $1 \leq i < L$, les lieux V_i et V_{i+1} sont reliés par une rue. Le parcours doit commencer **au lieu** 1 (donc $V_1 = 1$) et peut revenir plusieurs fois au même lieu ou à la même rue. À partir de ces informations, vous devez reconstituer un message binaire composé de **exactement 1 000 bits**.

Votre tâche consiste à implémenter les deux étapes de ce processus :

- **Encodage** : Étant donné la carte de la ville et une séquence de 1 000 bits, produisez un parcours de longueur quelconque commençant au lieu 1.
- **Décodage** : Étant donné la même carte et le parcours que vous avez produit, reconstituez les 1 000 bits.

Votre score est basé sur la longueur du parcours encodé : les parcours les plus courts obtiendront un meilleur score. Voir la section sur le score pour plus de détails.

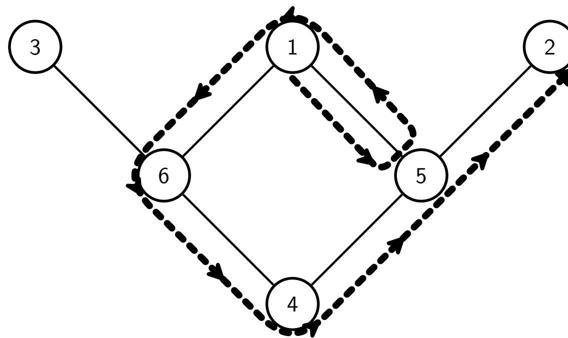

Figure 1: Exemple de ville et de parcours

La figure 1 représente une ville comportant N = 6 lieux et les rues comprises entre les points $(1, 5)$, $(1, 6)$, $(2, 5)$, $(3, 6)$, $(4, 5)$ et $(4, 6)$.

Le parcours illustré passe par $1 \to 5 \to 1 \to 6 \to 4 \to 5 \to 2$. Il s'agit d'un parcours valide de longueur 7, partant du lieu 1.

Ce parcours peut encoder un message binaire de longueur 1 000 (par exemple : $[1, 0, 1, 1, 0, 0, \ldots]$). Le décodeur, étant donné ce trajet et la ville, doit produire les mêmes 1 000 bits.

Détails d'implémentation

Dans cette tâche, **ne lisez pas l'entrée standard et n'écrivez pas sur la sortie standard.** N'interagissez avec aucun fichier. N'implémentez pas de fonction `main`. À la place, commencez votre programme en incluant le fichier d'en-tête `crumbs.h` (`#include "crumbs.h"`) et interagissez avec lui comme décrit ci-dessous.

Fonctions

Vous devez implémenter les fonctions `init`, `encode` et `decode` :

```
void init(int N, int M, std::vector<int> A, std::vector<int> B);
```

- N représente le nombre de lieux et M le nombre de rues.
- Les vecteurs A et B ont chacun une taille de M. Pour $0 \leq j < M$, la rue j relie les lieux `A[j]` et `B[j]`. Les lieux sont numérotés de 1 à N. Il est possible de se déplacer entre n'importe quels lieux en empruntant les rues, et deux rues ne relient jamais la même paire de lieux.
- Cette fonction est appelée une fois pour chaque instance de votre programme. Consultez les sections Évaluation et Expérimentation pour plus de détails.

```
std::vector<int> encode(std::vector<int> bits);
```

- `bits` a une taille de 1 000 éléments et chaque élément vaut 0 ou 1.
- Vous devez retourner un chemin commençant à l'emplacement 1 : un `std::vector<int>` d'indices d'emplacement $V_1, \ldots, V_L$ avec $V_1 = 1$, où chaque paire consécutive (V_i, V_{i+1}) est reliée par une rue. La longueur du chemin est L, la longueur du vecteur. Notez que le vecteur doit commencer à l'indice 0, de sorte que l'indice 0 contient V_1 et l'indice $L - 1$ contient V_L.
- Cette fonction est appelée 50 fois par test.
- **Votre score pour un test dépend de la longueur maximale du chemin retourné. Consultez la section sur le score pour plus de détails.**

```
std::vector<int> decode(std::vector<int> walk);
```

- `walk` est une séquence de positions formant un parcours valide commençant à la position 1 (telle que produite par votre encodeur).
- Vous devez retourner un `std::vector<int>` de 1 000 entiers (chacun valant 0 ou 1) identique aux bits passés à `encode` pour produire ce parcours.
- Cette fonction est appelée 50 fois par test.

Si vous ne respectez pas l'une des conditions énumérées ci-dessus, votre programme sera jugé incorrect et vous obtiendrez 0% des points pour le cas de test.

Évaluation

L'évaluateur lancera deux instances distinctes de votre programme : un encodeur et un décodeur.

- Premièrement, l'évaluateur appellera la fonction `init` une fois sur l'encodeur.
- Deuxièmement, l'évaluateur sélectionnera 50 messages binaires et appellera la fonction `encode` 50 fois sur l'encodeur.
- Troisièmement, l'évaluateur appellera la fonction `init` une fois sur le décodeur. **Les mêmes paramètres que pour la première étape seront fournis à cette fonction.**
- Enfin, le juge appellera la fonction `decode` 50 fois sur le décodeur, une fois pour chaque parcours généré par l'encodeur, dans un ordre arbitraire.

L'encodeur et le décodeur ne peuvent communiquer d'aucune autre manière que par l'envoi d'un parcours. En particulier, toute information enregistrée dans des variables statiques ou globales de l'encodeur n'est pas accessible au décodeur.

Le temps d'exécution de votre solution sera la somme des temps d'exécution des deux instances.

Sous-tâches et contraintes

Pour toutes les sous-tâches :

- $3 \leq N \leq 1\,000$.
- $2 \leq M \leq 4\,000$.
- $1 \leq A[i] < B[i] \leq N$ pour tout i. Deux rues ne relient jamais deux endroits identiques.
- Il est possible de voyager entre deux lieux quelconques en empruntant un itinéraire de rues.

Il y a un test par sous-tâche. Les contraintes additionnelles pour les sous-tâches sont listées ci-dessous.

Sous-tâche	Points	Contraintes additionnelles	J (judge)	T (threshold)
1	8	$N = 3$ et la ville forme un graphe complet.[1]	1,001	1,001
2	8	$N = 5$ et la ville forme un graphe complet.	501	501
3	8	$N = 4$ et la ville forme un graphe complet.	632	1,001
4	8	$N = 10$, les lieux 1 à 5 forment un graphe complet, et il existe une rue $(i, i-5)$ pour tout $6 \leq i \leq 10$. Il n'y a pas d'autres rues.	481	501
5	8	$N = 3$ et la ville forme une ligne, avec des rues entre les emplacements i et $i+1$ pour tout i	1,999	2,001
6	8	$N = 5$ et la ville forme une ligne, avec des rues entre les emplacements i et $i+1$ pour tout i	1,263	1,391
7	8	$N = 9$ et la ville forme une grille de 3×3.[2]	668	831
8	8	$N = 25$ et la ville forme une grille de 5×5.[3]	560	711
9	12	$N = 10$, $M = 15$, et la carte de la ville est généré aléatoirement.[4]	591	821
10	12	$N = 100$, $M = 800$, et la carte de la ville est généré aléatoirement.	247	291
11	12	$N = 1\,000$, $M = 4\,000$, et la carte de la ville est généré aléatoirement.	315	391

Pour toutes les sous-tâches, le plan de la ville est disponible en téléchargement sur le site du concours. Il sera au même format que celui spécifié par l'évaluateur d'exemple.

Score

Si votre solution produit un décodage incorrect (bits erronés) ou enfreint l'une des conditions de la section Détails d'implémentation, votre score sera de 0.

Sinon, soit W la longueur maximale du parcours que votre encodeur produit pour tous les appels de ce test, et J la longueur maximale du parcours que la solution des juges peut produire pour ce même test, pour tous les messages d'encodage possibles. De plus, chaque sous-tâche possède un seuil T (voir le tableau ci-dessus). Votre score en pourcentage pour ce test est :

- **100%** si $W \leq J$,
- $\mathbf{25 + \frac{T-W}{T-J} \times 75\%}$ si $J < W \leq T$,
- **10%** si $T < W \leq 10000$,

[1] Dans un graphe complet, il existe une rue entre chaque paire de points

[2] Si nous réorganisons la ville en 3 lignes et 3 colonnes où les emplacements sont numérotés de 1 à N de gauche à droite et de haut en bas, il existe une rue entre deux emplacements si et seulement si ils sont adjacents horizontalement ou verticalement.

[3] Définie de la même manière que la sous-tâche précédente.

[4] Toutes les villes qui satisfont aux contraintes données ont une probabilité égale d'être générées.

- **0%** sinon.

Il y a un test par sous-tâche. Votre score pour cette sous-tâche correspond au score de ce test multiplié par le nombre de points attribués à cette sous-tâche.

Expérimentation

Pour tester votre code sur votre ordinateur, commencez par télécharger les fichiers fournis : `crumbs.cpp`, `crumbs.h` et `grader.cpp`.

Modifiez ensuite `crumbs.cpp`, qui contient un exemple d'implémentation basique.

Compilez votre solution avec :

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

Cela créera un fichier exécutable `crumbs` que vous pourrez lancer avec `./crumbs`.

En cas de problème de compilation, veuillez envoyer un message dans la section Communication de la plateforme.

L'évaluateur d'exemple lit l'entrée standard au format suivant :

- La première ligne contient N et M.
- Les M lignes suivantes décrivent les rues. La i-ième ligne contient deux entiers A et B, représentant une rue entre les points A et B.

L'évaluateur d'exemple commencera par appeler `init`, puis sélectionnera 50 messages différents. Pour chaque message, il exécutera `encode` afin de déterminer un chemin, puis appellera immédiatement `decode`. Il renverra une erreur si le message renvoyé est différent de l'original. Si tous les messages sont validés, il renverra la longueur du chemin le plus long obtenu par `encode`.

L'évaluateur d'exemple choisit les bits aléatoirement. **L'évaluateur officiel peut procéder différemment**. Notez également que l'évaluateur d'exemple exécute `encode` et `decode` à partir de la même instance de votre programme.

Entrée de l'évaluateur & exemple d'interaction

L'évaluateur d'exemple reçoit l'entrée suivante :

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

Ceci correspond au diagramme de la première page. Une interaction possible est la suivante :

L'évaluateur appelle d'abord `init(6, 6, [1, 1, 2, 3, 4, 4], [5, 6, 5, 6, 5, 6])`. Cela correspond à l'entrée fournie.

L'évaluateur choisit ensuite un message composé de 1 000 bits.

- L'évaluateur appelle `encode([1, 0, 1, 1, 0, 0, ...])` avec ces 1 000 bits.
- L'étudiant renvoie `[1, 5, 1, 6, 4, 5, 2]`, un chemin de longueur 7 commençant au lieu 1.
- L'évaluateur appelle `decode([1, 5, 1, 6, 4, 5, 2])` avec ce chemin.
- L'étudiant renvoie `[1, 0, 1, 1, 0, 0, ...]`, soit 1 000 bits. C'est correct.

L'évaluateur d'exemple effectue 49 appels supplémentaires aux fonctions `encode` et `decode` avec des messages différents.

L'évaluateur d'exemple affiche la longueur du plus long chemin généré par `encode`.

Hedge Maze III

Jacques vous a piégé dans un labyrinthe — un labyrinthe de haies !

Vous savez que le labyrinthe comporte N salles, numérotées de 1 à N, et qu'il existe N-1 passages reliant chacun deux salles différentes. En suivant les passages dans un sens ou dans l'autre, vous pouvez voyager entre n'importe quelles salles. Cependant, Jacques ne vous a pas indiqué l'emplacement de ces passages ; votre objectif est de tous les découvrir.

Pour vous aider à vous échapper, Jacques a accepté de répondre à vos *questions*. Pour chaque question, vous indiquez deux pièces distinctes, A et B. Jacques vous indiquera la pièce ayant l'**indice le plus bas** située sur l'unique chemin reliant la pièce A à la pièce B, **excluant** les pièces A et B elles-mêmes. Si les pièces A et B sont directement reliées par un couloir (de sorte qu'aucune autre pièce ne se trouve entre elles), Jacques vous renverra -1.

Vous devez reconstituer la structure complète du labyrinthe à l'aide d'un nombre limité de requêtes. Consultez la section sur le score pour plus de détails.

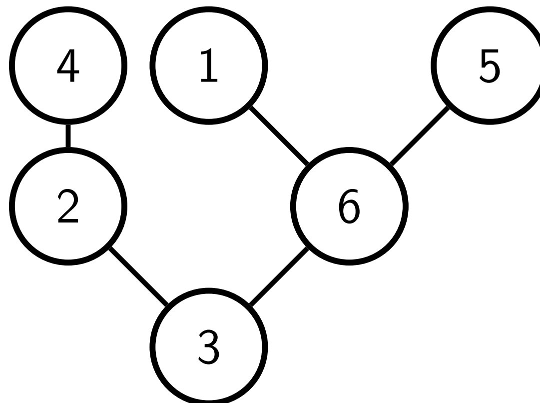

Figure 1: Un exemple de labyrinthe

La figure 1 représente un labyrinthe comportant $N = 6$ salles et des passages reliant $(1, 6)$, $(5, 6)$, $(2, 3)$, $(3, 6)$ et $(2, 4)$. Par exemple :

- Si vous posez une question avec $A = 1$ et $B = 5$, le chemin de la chambre 1 à la chambre 5 passe par la chambre 6. La seule chambre entre elles est la chambre 6, donc Jacques répond 6.
- Si vous posez une question avec $A = 1$ et $B = 2$, le chemin passe par les chambres 6 et 3. L'indice le plus bas entre elles est 3, donc Jacques répond 3.
- Si vous posez une question avec $A = 2$ et $B = 4$, les chambres 2 et 4 sont directement reliées. Jacques répond -1.

Détails d'implémentation

Dans cette tâche, **ne lisez pas l'entrée standard et n'écrivez pas sur la sortie standard.** N'interagissez avec aucun fichier. N'implémentez pas de fonction `main`. À la place, commencez votre programme en incluant le fichier d'en-tête `maze.h` (`#include "maze.h"`) et interagissez avec lui comme décrit ci-dessous.

Fonctions

Vous devez implémenter la fonction `map_maze`, qui est appelée une fois dans chaque fichier test :

```
void map_maze(int N);
```

- `N` représente le nombre de pièces du labyrinthe.
- Cette fonction peut d'appeler `query`.
- Vous devez appeler `report_passageway` exactement $N - 1$ fois : une fois pour chaque passage du labyrinthe.

Depuis `map_maze`, vous pouvez appeler les fonctions suivantes :

```
int query(int A, int B);
```

- `A` et `B` doivent être deux pièces distinctes, chacune comprise entre 1 et N inclus.
- Cette fonction renvoie la pièce ayant l'indice le plus bas sur le chemin de `A` à `B`, extrémités `A` et `B` exclues.
- Si `A` et `B` sont directement reliées par un passage, cette fonction renvoie -1.
- **Votre score dépend du nombre d'appels à cette fonction. Consultez la section sur le score pour plus de détails.**

```
void report_passageway(int A, int B);
```

- Vous devez appeler cette fonction pour chaque passage entre les pièces A et B.
- Vous devez appeler cette fonction exactement $N - 1$ fois : une fois pour chaque passage du labyrinthe.
- Chaque passage est bidirectionnel : vous pouvez appeler (A, B) ou (B, A), indifféremment.

Si vous ne respectez pas l'une des conditions énumérées ci-dessus, votre programme sera jugé incorrect et vous obtiendrez 0% des points pour le cas de test.

L'évaluateur n'est pas adaptatif. Cela signifie que les réponses à toutes les questions sont basées sur des données d'entrée fixes.

Sous-tâches et contraintes

Pour toutes les sous-tâches :

- $2 \leq N \leq 1\,000$.
- Le labyrinthe comporte $N - 1$ passages.
- Il est possible de se déplacer entre n'importe quelles pièces en empruntant les passages.

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

Sous-tâche	Points	Contraintes supplémentaires
1	25	Pour chaque chambre i, il y a moins de 10 chambres entre la chambre i et la chambre N.
2	20	Le labyrinthe forme une ligne : c'est-à-dire que chaque pièce est reliée à au maximum 2 autres pièces.

Sous-tâche	Points	Contraintes supplémentaires
3	15	Pour tout $i < N$, la chambre i est reliée à au plus 2 autres chambres. La chambre N peut être reliée à un nombre quelconque d'autres chambres.
4	40	Pas de contraintes additionnelles.

Score

Si votre solution ne parvient pas à trouver tous les passages, ou si elle enfreint l'une des conditions de la section Détails d'implémentation, votre score sera de 0.

Sinon, soit Q le nombre d'appels que votre solution effectue à `query` :

- Si $Q \leq 15\,000$, vous recevrez 100% des points de ce test.
- Si $Q \leq 100\,000$, vous recevrez 20% des points de ce test.
- Si $Q \leq 1\,000\,000$, vous recevrez 10% des points de ce test.
- Sinon, vous ne recevrez 0% des points de ce test.

Votre score pour une sous-tâche sera le score **minimum** de tous les tests de la sous-tâche, multiplié par le nombre de points que vous pouvez obtenir dans la sous-tâche.

Expérimentation

Pour tester votre code sur votre ordinateur, commencez par télécharger les fichiers fournis : `maze.cpp`, `maze.h` et `grader.cpp`.

Modifiez ensuite `maze.cpp`, qui contient un exemple d'implémentation basique.

Compilez votre solution avec :

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

Cela créera un fichier exécutable `maze` que vous pourrez lancer avec `./maze`.

En cas de problème de compilation, veuillez envoyer un message dans la section Communication de la plateforme.

L'évaluateur d'exemple lit l'entrée standard au format suivant :

- La première ligne contient N.
- Les $N-1$ lignes suivantes décrivent les passages. La $i$ème ligne contient deux entiers a et b, représentant un passage entre les pièces a et b.

Notez que l'évaluateur d'exemple vérifie si les passages fournis constituent une entrée valide.

Entrée de l'évaluateur & exemple d'interaction

L'évaluateur d'exemple reçoit l'entrée suivante :

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

Cela correspond au schéma de la première page.

Une interaction possible est décrite ci-dessous :

Évaluateur	Étudiant	Description
`map_maze(6)`		L'évaluateur appelle votre programme.
	`query(1, 5)`	Vous posez une question sur le chemin allant de la chambre 1 à la chambre 5.
renvoie `6`		Le chemin passe par la chambre 6, donc Jacques répond avec 6.
	`query(1, 2)`	Vous posez une question sur le chemin allant de la chambre 1 à la chambre 2.
renvoie `3`		Le chemin passe par les salles 6 et 3. L'indice le plus petit est 3.
	`query(2, 4)`	Vous posez une question sur le chemin allant de la chambre 2 à la chambre 4.
renvoie `-1`		Les chambres 2 et 4 sont directement reliées, donc Jacques répond -1.
	`report_passageway(1, 6)`	Vous signalez un passage entre les chambres 1 et 6. C'est exact.
	...	(Vous continuez jusqu'à ce que les 5 passages soient signalés.)

PERFECTION

Un tableau d'entiers positifs est dit *parfait* si, pour chaque entier positif K, la valeur K apparaît soit **0 fois**, soit **exactement** K **fois**. Un tableau vide est également considéré comme parfait. Par exemple, le tableau $[2, 4, 2, 4, 4, 4]$ est parfait car 2 apparaît exactement 2 fois, 4 apparaît exactement 4 fois, et toute autre valeur apparaît 0 fois.

On vous donne un tableau de N entiers positifs $V_1, \ldots, V_N$ et vous pouvez effectuer les opérations suivantes, un nombre quelconque de fois, dans n'importe quel ordre :

- **Ajouter** un élément (un entier positif) pour A pièces.
- **Supprimer** un élément pour D pièces.
- **Modifier** un élément (changer sa valeur en un entier strictement positif) pour M pièces.

Quel est le nombre minimum de pièces requis pour rendre le tableau parfait ?

Sous-tâches et contraintes

Pour toutes les sous-tâches :

- $1 \leq N \leq 5\,000$.
- $1 \leq V_i \leq 5\,000$ pour tout i.
- $1 \leq A, D, M \leq 1\,000\,000\,000$.

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

Sous-tâche	Points	Contraintes additionnelles
1	10	$N = 1$.
2	10	$V_i = V_j$ pour tout i, j.
3	15	$M > A + D$.
4	25	$M = 1,\ A = 1\,000\,000\,000,\ D = 1\,000\,000\,000$.
5	20	$N \leq 300,\ V_i \leq 300$ pour tout i.
6	20	Pas de contraintes additionnelles.

Entrée

- La première ligne contient quatre entiers : N, A, D et M.
- La deuxième ligne contient N entiers : $V_1, \ldots, V_N$.

Sortie

Afficher un seul entier, le nombre minimal de pièces nécessaires pour rendre le tableau parfait.

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

Entrée d'exemple 1

```
6 1 2 3
2 4 2 4 4 4
```

Sortie d'exemple 1

```
0
```

Entrée d'exemple 2

```
1 5 3 4
2
```

Sortie d'exemple 2

```
3
```

Entrée d'exemple 3

```
3 1 5 2
3 1 1
```

Sortie d'exemple 3

```
3
```

Entrée d'exemple 4

```
14 10 6 5
3 3 3 3 3 4 4 5 5 5 5 5 5 14
```

Sortie d'exemple 4

```
20
```

Entrée d'exemple 5

```
4 1 4 3
2 2 2 2
```

Sortie d'exemple 5

```
7
```

Explications

Dans le premier exemple, le tableau $[2, 4, 2, 4, 4, 4]$ est déjà parfait, aucune opération n'est donc nécessaire. La réponse est 0.

Dans le second exemple, on peut supprimer l'élément 2 pour 3 pièces. Le tableau est alors vide, ce qui est parfait. Il n'existe pas d'option moins coûteuse, la réponse est donc 3.

Dans le troisième exemple, on peut transformer un 1 en un 3 pour 2 pièces et ajouter un autre 3 pour 1 pièce. On obtient ainsi $[3, 3, 1, 3]$, qui est un tableau parfait. Il n'existe pas de solution moins chère, donc la réponse est 3.

Dans le quatrième exemple, on peut transformer deux 3 en deux 2 pour 10 pièces, un 5 en un 4 pour 5 pièce, et le 14 en un 4 pour 5 pièces. On obtient ainsi $[2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 4]$, qui est un tableau parfait. Il n'existe pas de solution moins chère, donc la réponse est 20.

Dans le cinquième exemple, on peut transformer un 2 en un 1 pour 3 pièces et supprimer un 2 pour 4 pièces. On obtient ainsi $[2, 2, 1]$, qui est un tableau parfait. Il n'existe pas de solution moins chère, donc la réponse est 7.