

Magic City

Le maire de Tachkent veut rénover le célèbre parc d'attraction Magic City. On vous donne un entier strictement positif K et vous devez concevoir un parc qui respecte les contraintes suivantes :

- Vous devez choisir un entier N , le nombre d'attractions. Les attractions sont numérotées de 0 à $N - 1$.
- Vous devez ajouter des chemins bidirectionnels entre des attractions **distinctes**. Il est autorisé que plusieurs chemins relient la même paire d'attractions. **Il n'est pas nécessaire de pouvoir se déplacer entre toutes les paires d'attractions en utilisant ces chemins.**
- Pour chaque i tel que $0 \leq i < N$, vous devez assigner à l'attraction i un type $T[i]$. Il y a $2K$ types numérotés de 0 à $2K - 1$. Chaque type doit être assigné à au moins une attraction. Deux attractions différentes peuvent avoir le même type.

Les études de marché ont révélé que :

1. Chaque visiteur préfère visiter trois attractions sans visiter deux attractions du même type consécutivement.
2. Les visiteurs n'apprécient pas les attractions qui sont aux extrémités d'un nombre trop élevé de chemins.

Afin de satisfaire **tous** les potentiels visiteurs, le maire impose les deux conditions suivantes pour votre parc :

On dit qu'un triplet ordonné de **types** (t_1, t_2, t_3) pour $0 \leq t_1, t_2, t_3 < 2K$ est **intéressant** si $t_1 \neq t_2$ et $t_2 \neq t_3$. Notez que t_1 peut être égal à t_3 . Il y a donc $2K \cdot (2K - 1)^2$ triplets intéressants.

Condition 1: Pour chaque triplet intéressant (t_1, t_2, t_3) , il doit exister trois attractions a_1, a_2, a_3 (avec $0 \leq a_1, a_2, a_3 < N$) telles que :

- Les types de a_1, a_2, a_3 sont respectivement égaux à t_1, t_2, t_3 . Plus formellement, $T[a_1] = t_1, T[a_2] = t_2, T[a_3] = t_3$
- Il existe un chemin entre a_1 et a_2 .
- Il existe un chemin entre a_2 et a_3 .

Il n'est pas nécessaire qu'il y ait un chemin entre a_1 et a_3 . Notez que quand $t_1 = t_3$, les attractions a_1 et a_3 peuvent être la même.

Condition 2: Chaque attraction peut être l'extrémité d'**au plus** K chemins.

Ce problème est constitué de 50 sous-tâches **output-only** avec un **score partiel**. Chaque sous-tâche correspond à une valeur spécifique de K , et vous devez trouver un parc qui satisfie les conditions ci-dessus pour cette valeur de K . Votre score dépend du nombre d'attractions dans votre parc : avoir moins d'attractions permet d'obtenir un meilleur score.

Détails d'implémentation

Il y a deux méthodes pour soumettre votre solution, et vous pouvez utiliser soit l'une soit l'autre pour chaque sous-tâche :

- **Appel de fonction**
- **Fichier de sortie**

Pour soumettre votre solution via un **appel de fonction**, vous devez implémenter la fonction suivante :

```
std::pair<std::vector<int>, std::vector<std::pair<int, int>>>  
construct(int K)
```

- K : la moitié du nombre de types d'attractions, et le nombre maximum de chemins dont une attraction peut être l'extrémité.
- Cette fonction est appelée exactement une fois pour chaque sous-tâche.

Cette procédure doit renvoyer une paire (T, E) décrivant le parc d'attractions que vous avez conçu. Soit M le nombre de chemins de votre parc.

- T : une liste de longueur N décrivant les types des attractions.
- E : une liste de longueur M décrivant les chemins. Pour tout $0 \leq j < M$, $E[j] = (U[j], V[j])$ représente un chemin bidirectionnel entre les deux attractions distinctes $U[j]$ et $V[j]$.

Pour soumettre votre solution via un **fichier de sortie**, vous devez créer et soumettre un fichier texte au format suivant :

```
N M  
T[0] T[1] ... T[N-1]  
U[0] V[0]  
U[1] V[1]  
...  
U[M-1] V[M-1]
```

Notez que votre solution doit satisfaire les contraintes suivantes pour être considérée **valide** :

- $N \leq 2000$

- $0 \leq T[i] < 2K$ pour tout $0 \leq i < N$, et chaque type doit être assigné à au moins une attraction.
- $0 \leq U[j], V[j] < N$ et $U[j] \neq V[j]$ pour tout $0 \leq j < M$.
- Les conditions 1 et 2 doivent être respectées.

Contraintes

- $1 \leq K \leq 50$

Score

Il y a 50 sous-tâches, une pour chaque entier K de 1 à 50. Pour la sous-tâche i ($1 \leq i \leq 50$), la valeur de K est i .

Chaque sous-tâche a un score S et un objectif pour votre nombre d'attractions P spécifié dans la table ci-dessous.

Sous-tâches	S	P
1	1	2
2	8	12
3	9	24
4	9	40
5	9	50
6 - 10	4	$12 \cdot K$
11 - 12	3	$12 \cdot K$
13 - 50	1	$12 \cdot K$

Pour chaque sous-tâche, si votre solution ne décrit pas un parc d'attraction valide, le score de votre solution sera 0 (affiché en tant que `Output isn't correct` dans CMS).

Sinon, votre score pour cette sous-tâche sera calculé à partir de N et des paramètres S et P de la façon suivante :

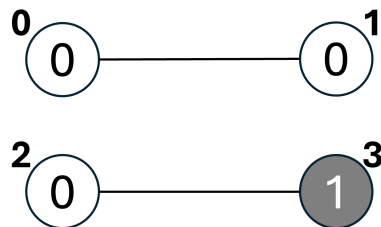
Condition	Score
$N \leq P$	S
$P < N \leq 2P$	$(0.4 + 0.3 \cdot \frac{2P-N}{P}) \cdot S$
$2P < N \leq 2000$	$(0.1 + 0.3 \cdot \frac{2P}{N}) \cdot S$

Exemple

Considérez l'appel suivant :

```
construct(1)
```

Dans cet exemple, $K = 1$ et il y a donc $2K = 2$ types d'attractions. La figure ci-dessous montre une solution valide pour $N = 4$ attractions et $M = 2$ chemins. Les attractions 0, 1, 2 sont de type 0 et l'attraction 3 est de type 1.



Il y a deux triplets intéressants :

- Pour le triplet de types $(0, 1, 0)$, on peut choisir $(a_1, a_2, a_3) = (2, 3, 2)$.
- Pour le triplet de types $(1, 0, 1)$, on peut choisir $(a_1, a_2, a_3) = (3, 2, 3)$.

La condition 1 est donc satisfaite. La fonction peut renvoyer la paire $([0, 0, 0, 1], [(0, 1), (2, 3)])$. Notez que la solution pour cet exemple n'est potentiellement pas optimale pour $K = 1$.

Évaluateur d'exemple

Format d'entrée :

```
K
```

Format de sortie :

```
N M
T[0] T[1] ... T[N-1]
U[0] V[0]
U[1] V[1]
...
U[M-1] V[M-1]
```

Notez que le format de sortie de l'évaluateur d'exemple est le même que le format requis pour les fichiers de sortie.