A. Le roi s'échappe !

1 s., 256 MB

Alice et Bob jouent aux échecs sur un énorme échiquier de dimensions $n \times n$. Alice n'a plus qu'une seule pièce — une reine, située en (a_x, a_y), tandis que Bob n'a que son roi se tenant en (b_x, b_y). Alice pense que sa reine dominant l'échiquier, la victoire lui est assurée.

Mais Bob a élaboré un plan astucieux pour s'emparer de la victoire pour lui-même — il doit faire avancer son roi jusqu'en (c_x, c_y) pour revendiquer la victoire. Alors qu'Alice est distraite par son sentiment de supériorité, **elle ne déplace plus aucune pièce, et c'est seulement Bob qui effectue des coups.**

Bob gagnera s'il peut déplacer son roi de (b_x, b_y) à (c_x, c_y) **sans jamais se mettre en échec**. Rappelez-vous qu'un roi peut se déplacer vers l'un des 8 carrés adjacents. Un roi est en échec s'il se trouve sur la même rangée (c'est-à-dire ligne), colonne (c'est-à-dire colonne), ou diagonale que la reine ennemie.

Déterminez si Bob peut gagner ou non.

Input

La première ligne contient un entier n ($3 \le n \le 1000$) — les dimensions de l'échiquier.

La deuxième ligne contient deux entiers a_x et a_y ($1 \le a_x, a_y \le n$) — les coordonnées de la reine d'Alice.

La troisième ligne contient deux entiers b_x et b_y ($1 \le b_x, b_y \le n$) — les coordonnées du roi de Bob.

La quatrième ligne contient deux entiers c_x et c_y ($1 \le c_x, c_y \le n$) — les coordonnées de l'endroit où Bob veut se rendre.

Il est garanti que le roi de Bob n'est actuellement pas en échec et que l'emplacement cible n'est pas non plus en échec.

De plus, le roi n'est pas situé sur la même case que la reine (c'est-à-dire $a_x \neq b_x$ ou $a_y \neq b_y$), et la cible ne coïncide ni avec la position de la reine (c'est-à-dire $c_x \neq a_x$ ou $c_y \neq a_y$) ni avec la position du roi (c'est-à-dire $c_x \neq b_x$ ou $c_y \neq b_y$).

Output

Affichez "YES" (sans guillemets) si Bob peut se rendre de (b_x, b_y) à (c_x, c_y) sans jamais se mettre en échec, sinon affichez "NO".

Vous pouvez imprimer chaque lettre en majuscule ou en minuscule.

input

```
8
4 4
1 3
3 1
```

output

```
YES
```

input

```
8
4 4
2 3
1 6
```

output

```
NO
```

input

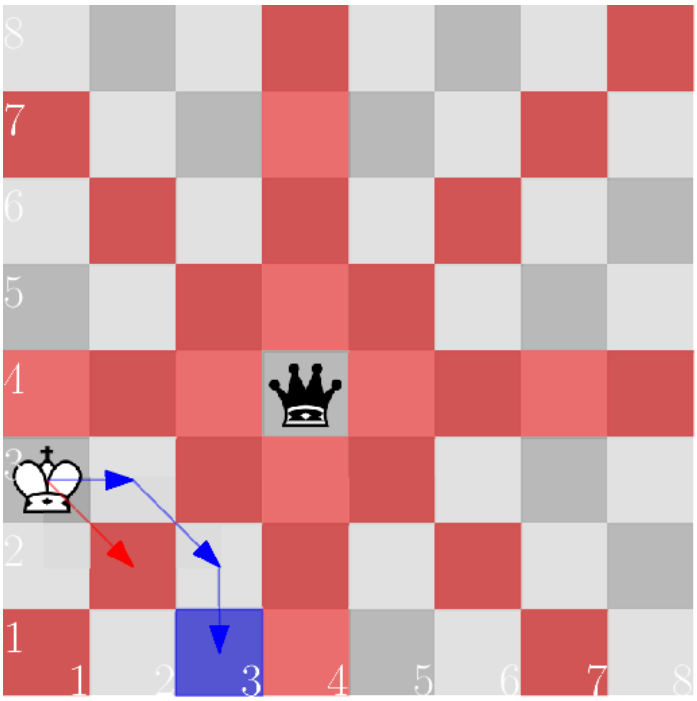
```
8
3 5
1 2
6 1
```

output

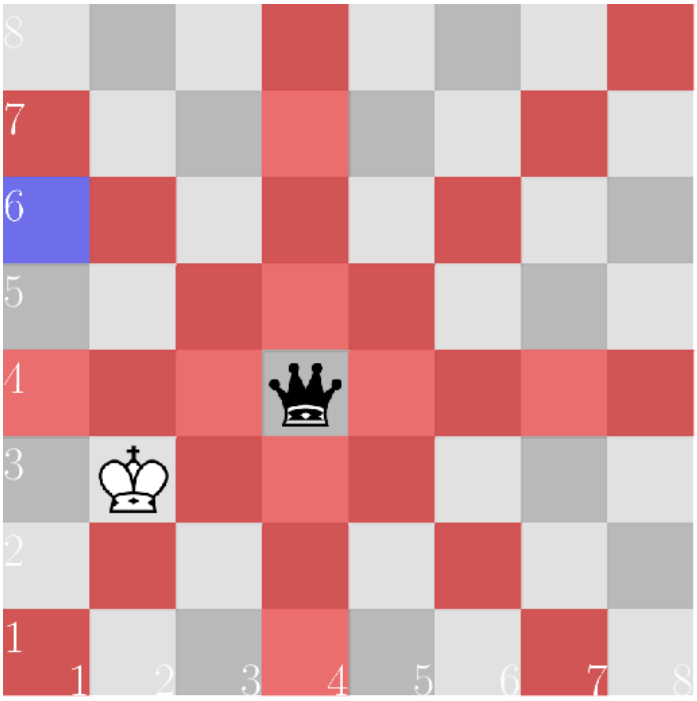
```
NO
```

Dans les diagrammes ci-dessous, les cases contrôlées par la reine noire sont marquées en rouge, et la case cible est marquée en bleu.

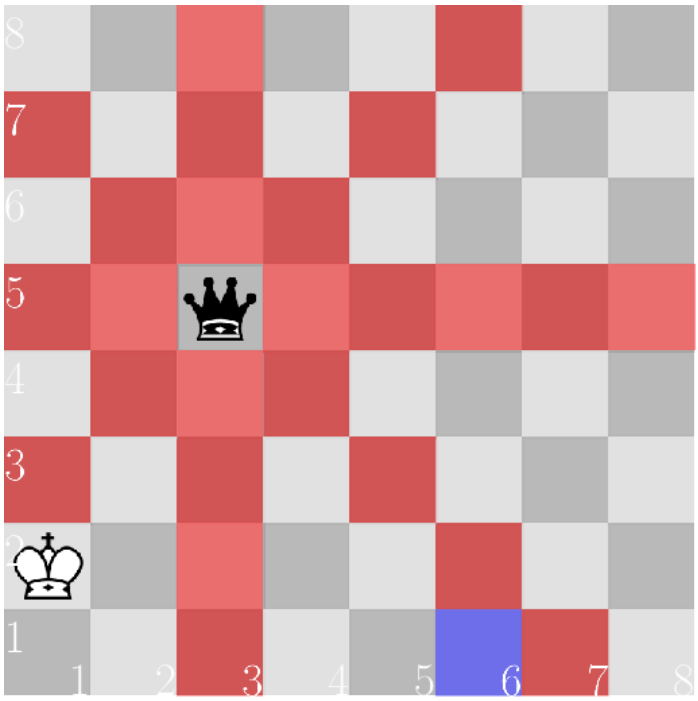
Dans le premier cas, le roi peut se déplacer, par exemple, via les cases $(2, 3)$ et $(3, 2)$. Notez que la route directe par $(2, 2)$ passe par un échec.



Dans le deuxième cas, la reine surveille la quatrième rangée, et le roi n'a aucun moyen de la traverser.



Dans le troisième cas, la reine surveille la troisième colonne.



B. Pommier

4 s., 512 MB

Timofey possède un pommier dans son jardin; c'est un arbre raciné de n sommets avec la racine au sommet 1 (les sommets sont numérotés de 1 à n). Un arbre est un graphe connecté sans boucles ni arêtes multiples.

Cet arbre est très inhabituel — il pousse avec sa racine vers le haut. Cependant, c'est assez normal pour les arbres des programmeurs.

Le pommier est assez jeune, donc seules deux pommes y pousseront. Les pommes pousseront dans certains sommets (ces sommets peuvent être les mêmes). Après la croissance des pommes, Timofey commence à secouer l'arbre à pommes jusqu'à ce que les pommes tombent. Chaque fois que Timofey secoue l'arbre à pommes, ce qui suit se produit pour chacune des pommes :

- Si un sommet u a un enfant, la pomme se déplace vers lui (s'il y a plusieurs tels sommets, la pomme peut se déplacer vers l'un quelconque).
- Sinon, la pomme tombe de l'arbre.

On peut montrer qu'après un temps fini, les deux pommes tomberont de l'arbre.

Timofey a q hypothèses dans lesquelles les pommes peuvent pousser sur certains sommets. Il suppose que les pommes peuvent pousser dans les sommets x et y, et veut connaître le nombre de paires de sommets (a, b) à partir desquels les pommes peuvent tomber de l'arbre, où a — le sommet à partir duquel une pomme du sommet x tombera, b — le sommet à partir duquel une pomme du sommet y tombera. Aidez-le à le faire.

Input

La première ligne contient l'entier t ($1 \le t \le 10^4$) — le nombre de cas de test.

La première ligne de chaque cas de test contient l'entier n ($2 \le n \le 2 \cdot 10^5$) — le nombre de sommets dans l'arbre.

Ensuite, il y a $n - 1$ lignes décrivant l'arbre. À la ligne i, il y a deux entiers u_i et v_i ($1 \le u_i, v_i \le n, u_i \ne v_i$) — arête dans l'arbre.

La ligne suivante contient un seul entier q ($1 \le q \le 2 \cdot 10^5$) — le nombre d'hypothèses de Timofey.

Chacune des q lignes suivantes contient deux entiers x_i et y_i ($1 \le x_i, y_i \le n$) — les sommets supposés où les pommes pousseront pour l'hypothèse i.

Il est garanti que la somme de n ne dépasse pas $2 \cdot 10^5$. De même, il est garanti que la somme de q ne dépasse pas $2 \cdot 10^5$.

Output

Pour chaque hypothèse de Timofey, affichez le nombre de paires ordonnées de sommets à partir desquels les pommes peuvent tomber de l'arbre si l'hypothèse est vraie sur une ligne séparée.

input

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

output

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

input

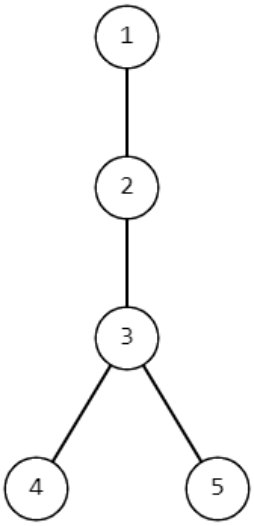
```
2
5
5 1
1 2
2 3
4 3
2
5 5
5 1
5
3 2
5 3
2 1
4 2
3
4 3
2 1
4 2
```

output

```
1
2
1
4
2
```

Dans le premier exemple :

- Pour la première hypothèse, il existe deux paires possibles de sommets à partir desquels les pommes peuvent tomber de l'arbre : $(4, 4), (5, 4)$.
- Pour la deuxième hypothèse, il existe également deux paires : $(5, 4), (5, 5)$.
- Pour la troisième hypothèse, il n'y a qu'une seule paire : $(4, 4)$.
- Pour la quatrième hypothèse, il existe 4 paires : $(4, 4), (4, 5), (5, 4), (5, 5)$.



Arbre du premier exemple.

Pour le deuxième exemple, il y a 4 paires possibles de sommets à partir desquels les pommes peuvent tomber : $(2, 3), (2, 2), (3, 2), (3, 3)$. Pour la deuxième hypothèse, il n'y a qu'une seule paire possible : $(2, 3)$. Pour la troisième hypothèse, il existe deux paires : $(3, 2), (3, 3)$.

C. Transformer A en B

1 s., 256 MB

Vasily possède un nombre a qu'il souhaite transformer en un nombre b. À cette fin, il peut effectuer deux types d'opérations :

- multiplier le nombre actuel par 2 (c'est-à-dire remplacer le nombre x par $2 \cdot x$) ;
- ajouter le chiffre 1 à droite du nombre actuel (c'est-à-dire remplacer le nombre x par $10 \cdot x + 1$).

Vous devez aider Vasily à transformer le nombre a en le nombre b en n'utilisant que les opérations décrites ci-dessus, ou déterminer que cela est impossible.

Notez que dans cette tâche, il n'est pas nécessaire de minimiser le nombre d'opérations. Il suffit de trouver n'importe quelle façon de transformer a en b.

Input

La première ligne contient deux entiers positifs a et b ($1 \le a < b \le 10^9$) — le nombre que Vasily possède et le nombre qu'il souhaite obtenir.

Output

S'il n'est pas possible d'obtenir b à partir de a, imprimez "NO" (sans guillemets).

Sinon, imprimez trois lignes. Sur la première ligne, imprimez "YES" (sans guillemets). La deuxième ligne doit contenir un seul entier k — la longueur de la séquence de transformations. Sur la troisième ligne, imprimez la séquence de transformations $x_1, x_2, \ldots, x_k$, où :

- x_1 doit être égal à a,
- x_k doit être égal à b,
- x_i doit être obtenu à partir de x_{i-1} en utilisant l'une quelconque des deux opérations décrites ($1 < i \le k$).

S'il existe plusieurs réponses, imprimez l'une d'entre elles.

input
2 162
output
YES 5 2 4 8 81 162

input
4 42
output
NO

input
100 40021
output
YES 5 100 200 2001 4002 40021

D. Grands Graphes

2 s., 256 MB

Le fermier John possède une ferme composée de n pâturages reliés par des routes unidirectionnelles. Chaque route a un poids, représentant le temps nécessaire pour aller du début à la fin de la route. Les routes pourraient avoir un poids négatif, où les vaches vont si vite qu'elles retournent dans le temps ! Cependant, le fermier John garantit qu'il est impossible que les vaches restent coincées dans une boucle temporelle, où elles peuvent voyager indéfiniment dans le temps en traversant une séquence de routes. De plus, chaque paire de pâturages est reliée par au plus une route dans chaque direction.

Malheureusement, le fermier John a perdu la carte de la ferme. Tout ce dont il se souvient est un tableau d, où d_i est la plus petite quantité de temps qu'il a fallu aux vaches pour atteindre le i-ème pâturage depuis le pâturage 1 en utilisant une séquence de routes. Le coût de sa ferme est la somme des poids de chacune des routes, et le fermier John a besoin de connaître le coût **minimal** d'une ferme conforme à sa mémoire.

Input

La première ligne contient un entier t ($1 \le t \le 10^4$) — le nombre de cas de test. Ensuite, t cas suivent.

La première ligne de chaque cas de test contient un seul entier n ($1 \le n \le 10^5$) — le nombre de pâturages.

La deuxième ligne de chaque cas de test contient n entiers séparés par des espaces $d_1, d_2, \ldots, d_n$ ($0 \le d_i \le 10^9$) — le tableau d. Il est garanti que $d_1 = 0$.

Il est garanti que la somme de n sur tous les cas de test ne dépasse pas 10^5.

Output

Pour chaque cas de test, imprimez le coût minimum possible d'une ferme conforme à la mémoire du fermier John.

input
3 3 0 2 3 2 0 1000000000 1 0
output
-3 0 0

Dans le premier cas de test, vous pouvez ajouter des routes

- du pâturage 1 au pâturage 2 avec un temps de 2,
- du pâturage 2 au pâturage 3 avec un temps de 1,
- du pâturage 3 au pâturage 1 avec un temps de -3,
- du pâturage 3 au pâturage 2 avec un temps de -1,
- du pâturage 2 au pâturage 1 avec un temps de -2.

Le coût total est $2 + 1 + -3 + -1 + -2 = -3$.
Dans le deuxième cas de test, vous pouvez ajouter une route du pâturage 1 au pâturage 2 avec un coût de 1000000000 et une route du pâturage 2 au pâturage 1 avec un coût de -1000000000. Le coût total est $1000000000 + -1000000000 = 0$.

Dans le troisième cas de test, vous ne pouvez pas ajouter de routes. Le coût total est 0.
