## 3. 228 CC - Can You Make a Palindrome

**[Version traduite]**
Une fonction reçoit une chaîne de caractères `s` et une série de requêtes. Pour chaque requête, il y aura un indice de début et de fin ainsi qu'un nombre de substitutions. Un palindrome est une chaîne de caractères qui se lit de la même manière de gauche à droite et de droite à gauche, comme "a", "mom" ou "abba". Pour chaque sous-chaîne représentée par l'indice de début et de fin de la chaîne `s`, déterminez s'il est possible de la réarranger en un palindrome après avoir effectué un nombre donné de substitutions.

Commencez par une chaîne de résultat vide. Après chaque requête, ajoutez un 1 ou un 0 pour indiquer si la sous-chaîne peut être convertie en palindrome. Un 1 représente oui et un 0 représente non. Retournez la chaîne de résultats après le traitement de toutes les requêtes.

**Exemple :**

**Exemple :**

s = cdecd

startIndex = [0, 0, 0, 0]

endIndex = [0, 1, 2, 3]

subs = [0, 1, 1, 0]

Requête 1 : s[0,0] = 'c', subs[0] = 0. La sous-chaîne 'c' est un palindrome après 0 substitutions, donc le résultat est '1'.

Requête 2 : s[0,1] = 'cd', subs[1] = 1. Changez 'c' en 'd' ou 'd' en 'c' en 1 substitution pour obtenir un palindrome 'dd' ou 'cc', donc le résultat est '11'.

Requête 3 : s[0,2] = 'cde', subs[2] = 1. Changez 'c' en 'e' ou 'e' en 'c' en 1 substitution pour obtenir un palindrome 'cdc' ou 'ede', donc le résultat est '111'.

Requête 4 : s[0,3] = 'cdec', subs[3] = 0. La chaîne ne peut pas être réarrangée en palindrome avec 0 substitutions, donc le résultat est '1110'.

**Exemple**

*s = 'xxdnssuqevu'*

*startIndex = [0]*

*endIndex = [10]*

*subs = [3]*

Il y a une requête qui examine l'ensemble de la chaîne et permet jusqu'à 3 substitutions. Une façon de créer un palindrome est de remplacer q et e par d et n, puis de réorganiser la chaîne.

21m gauche

TOUS

1

2

Il y a une requête qui examine l'ensemble de la chaîne et permet jusqu'à 3 substitutions. Une façon de créer un palindrome est de remplacer q et e par d et n, puis de réorganiser la chaîne. Après le remplacement, la chaîne s'écrit 'xxdnssudnvu'. Un palindrome qui peut être créé en réorganisant le résultat est 'xsudnvndusx'. Le résultat de la chaîne à retourner est '1'.

**Fonction Description :**
Complétez la fonction `palindromeChecker` dans l'éditeur ci-dessous.

`palindromeChecker` prend les paramètres suivants :

- `string s` : la chaîne de caractères initiale
- `int startIndex[q]` : un tableau d'entiers indiquant l'indice de début de chaque sous-chaîne à traiter
- `int endIndex[q]` : un tableau d'entiers indiquant l'indice de fin de chaque sous-chaîne à traiter
- `int subs[q]` : un tableau d'entiers indiquant le nombre de substitutions de caractères autorisées

Renvoie :

- `string` : une chaîne où chaque position représente les résultats d'une requête, 1 si un palindrome peut être formé, ou 0 sinon.

**Contraintes :**

- 1 ≤ longueur de la chaîne donnée ≤ 100000
- `s` ne contient que des lettres minuscules ascii[a-z]

21m gauche

TOUS

1

2

- 1 ≤ longueur de la chaîne donnée ≤ 100000
- `s` ne contient que des lettres minuscules, ascii[a-z]
- 1 ≤ q ≤ 100000
- 0 ≤ startIndex[i], endIndex[i] < longueur de `s`
- `startIndex[i]` ≤ `endIndex[i]`
- 0 ≤ subs[i] ≤ longueur de `s[startIndex[i]:endIndex[i]]`

**[Version originale]**

A function receives a string s and a series of queries. For each query, there will be a beginning and ending index and a number of substitutions. A palindrome is a string spelled the same way forward or backward, like *a, mom* or *abba*. For each substring represented by the beginning and ending index of the string s, determine if it is possible to rearrange it to a palindrome after performing up to the given number of substitutions.

Start with an empty result string. After each query, append a *1* or *0* to indicate whether the substring can be converted to a palindrome. A *1* represents *yes* and a *0* represents *no.* Return the results string after all queries have been processed.

**Example**

*s = cdecd*

*startIndex =[0, 0, 0, 0]*

Query 1: *s[0,0] = 'c', subs[0] = 0.* The substring 'c' is a palindrome after *0* substitutions, so *result = '1'.*

Query 2: *s[0,1] = 'cd', subs[1] = 1.* Change *'c'* to *'d'* or *'d'* to *'c'* in *1* substitution to make a palindrome 'dd' or 'cc', so *result = '11'.*

Query 3: *s[0,2] = 'cde', subs[2] = 1*. Change *'c'* to *'e'* or *'e'* to *'c'* in *1* substitution to make a palindrome 'cdc' or 'ede', so *result = '111'.*

Query 4: *s[0,3] = 'cdec', subs[3] = 0.* The string cannot be rearranged into a palindrome with *0* substitutions, so *result = '1110'.*

**Example**

*s = 'xxdnssuqevu'*

*startIndex = [0]*

*endIndex = [10]*

*subs = [3]*

There is one query that looks at the entire string and allows up to 3 substitutions. One way to create a palindrome is to replace *q* and *e* with *d* and *n*, then rearrange the string. After the replacement, the string *s' = 'xxdnssudnvu'*. One palindrome that can be created by rearranging the result is *'xsudnvndusx'.* The result string to return is *'1'.*

**Function Description**

Complete the function *palindromeChecker* in the editor below.

palindromeChecker has the following parameter(s):

palindromeChecker has the following parameter(s):

*string s:* the working string

*int startIndex [q]:* an array of integers that denotes the left index of each substring to process

*int endIndex[q]:* an array of integers that denotes the right index of each substring to process

*int subs[q]:* an array of integers that denotes the number of character substitutions that may occur

Returns:

*string*: a string where each position represents the results of a query, *1* if a palindrome can be formed, or *0* if not.

**Constraints**

- *1 ≤ length of the given string ≤ 100000*
- *s contains only lowercase letters , ascii[a-z]*
- *1 ≤ q ≤ 100000*
- *0 ≤ startIndex[i], endIndex[i] < length of s*
- *startIndex[i] ≤ endIndex[i]*
- *0 ≤ subs[i] ≤ length of s[startIndex[i]:endIndex[i]]*

**▶ Input Format for Custom Testing**

## ▼Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains a string *s*, the working string.
The next line contains an integer *q*, the size of the *startIndex* array.
The next *q* lines each contain an element *startIndex[i]* where $0 \le i < q$.
The next line contains an integer *q*, the size of the *endIndex* array.
The next *q* lines each contain an element *endIndex[i]* where $0 \le i < q$.
The next line contains an integer *q*, the size of the *subs* array.
The next *q* lines each contain an element *subs[i]* where $0 \le i < q$.

## ▼Sample Case 0

### Sample Input

```
STDIN       Function
-----       --------
bcba    →   s = "bcba"
3       →   startIndex[] size q = 3
1       →   startIndex = [ 1, 2, 1 ]
2
1
3       →   endIndex[] size q = 3
3       →   endIndex = [ 3, 3, 1 ]
```

```
3       →   subs[] size q = 3
2       →   subs = [ 2, 0, 0 ]
0
0
```

### Sample Output

```
101
```

### Explanation

*s = "bcba"*
*q = 3*
*startIndex = [1, 2, 1]*
*q = 3*
*endIndex = [3, 3, 1]*
*q = 3*
*subs = [2, 0, 0]*

*query 0, s[1,3], subs[0] = 2,* the substring is *"cba".* Up to two characters can be changed to make it "cbc" or "bbb", etc. A palindrome can be formed.
*query 1, s[2,3], subs[1] = 0,* the substring is *"ba"* With *subs = 0*, nothing can be changed, thus this is not a palindrome.
*query 2, s[1,1], subs[2] = 0,* the substring is *"c".* It is already a palindrome.

21m gauche

TOUS

1

2

The bit string is *101.*

▶ **Sample Case 1**

▶ **Sample Case 2**

Langage Java 8

Prêt pour la saisie semi-automatique

Environnement

```
14
15  class Result {
16
17      /*
18       * Complete the 'palindromeChecker' function below.
19       *
20       * The function is expected to return a STRING.
21       * The function accepts following parameters:
22       *  1. STRING s
23       *  2. INTEGER_ARRAY startIndex
24       *  3. INTEGER_ARRAY endIndex
25       *  4. INTEGER_ARRAY subs
26       */
27
```

21m gauche

TOUS

1

2

Environnement

```
27
28      /// Je vais moi même créer une fonction qui teste si une
    chaîne
29      /// est un palyndrome ou pas
30      static int estPalindrome(String s) {
31          int vrai = 0;
32          int tester = 0;
33
34          /// Je vais parcourir la chaîne pour voir si elle est un
35          /// palyndrome
36          if (s.toString().length() % 2 == 1) {
37              for (int i = 0, j = s.toString().length() - 1; i <= j; i
    ++, j--) {
38                  if (s.charAt(i) == s.charAt(j)) {
39                      tester++;
40                  }
41              }
```

Line: 14 Col: 1

**Résultats de test** **Saisie personnalisée**

**Exécuter le code** **Exécuter des tests** **Soumettre**

21m gauche

TOUS

1

2

Environnement

```
            if (tester == s.toString().length() / 2 + 1) {
                vrai = 1;
            } else {
                vrai = 0;
            }

        } else {
            for (int i = 0, j = s.toString().length() - 1; i <= j; i
++, j--) {
                if (s.charAt(i) == s.charAt(j)) {
                    tester++;
                }
            }
            /// Si tester est ...
            if (tester == s.toString().length() / 2 - 1) {
                vrai = 1;
            } else {
                vrai = 0;
```

Line: 14 Col: 1

**Résultats de test** **Saisie personnalisée**

Exécuter le code | Exécuter des tests | Soumettre

20m gauche

TOUS

1

2

Environnement

```
59                  vrai = 0;
60              }
61          }
62
63          return vrai;
64      }
65
66      public static String palindromeChecker(String s, List<Integer>
    startIndex, List<Integer> endIndex, List<Integer> subs) {
67      // Write your code here
68          /// On commence
69          /// Je déclare la chaîne que je vais renvoyer
70          String chaineARenvoyer = "";
71          String chaineTesteuse = "";
72
73          /// Nous allons utiliser une boucle for
74          for (int i = 0; i < endIndex.size(); i++) {
75              chaineTesteuse = s.substring(startIndex.indexOf(i),
```

Line: 14 Col: 1

**Résultats de test** **Saisie personnalisée**

**Exécuter le code** **Exécuter des tests** **Soumettre**

20m gauche

TOUS

1

2

Environnement

```
73          /// Nous allons utiliser une boucle for
74          for (int i = 0; i < endIndex.size(); i++) {
75              chaineTesteuse = s.substring(startIndex.indexOf(i),
    endIndex.indexOf(i));
76
77              chaineTesteuse += s.substring(startIndex.indexOf(i),
    endIndex.indexOf(i));
78
79              if (estPalindrome(chaineTesteuse) == 1) {
80                  chaineARenvoyer += 1;
81              } else {
82                  chaineARenvoyer += 0;
83              }
84          }
85
86          return chaineARenvoyer;
87      }
88
```

Line: 14 Col: 1

**Résultats de test** **Saisie personnalisée**

Exécuter le code | Exécuter des tests | Soumettre

20m gauche

TOUS

1

2

Environnement

```
78
79              if (estPalindrome(chaineTesteuse) == 1) {
80                  chaineARenvoyer += 1;
81              } else {
82                  chaineARenvoyer += 0;
83              }
84          }
85
86          return chaineARenvoyer;
87      }
88
89
90
91      /// Maintenant, je sais s'il s'agit d'un palyndrome ou pas.
92
93  }
94
95  public class Solution {...
```

Line: 14 Col: 1

**Résultats de test** **Saisie personnalisée**

Exécuter le code | Exécuter des tests | Soumettre

20m gauche

TOUS

1

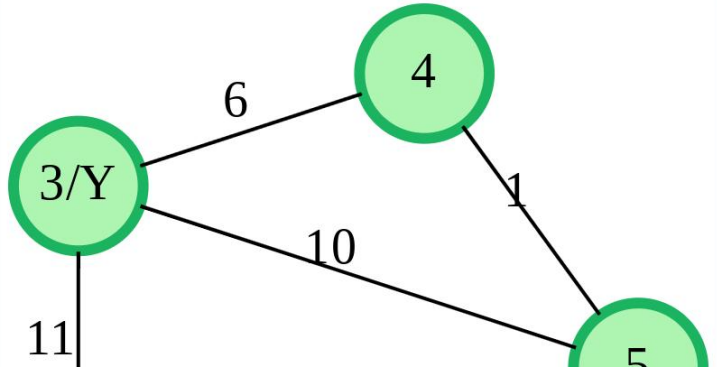
2

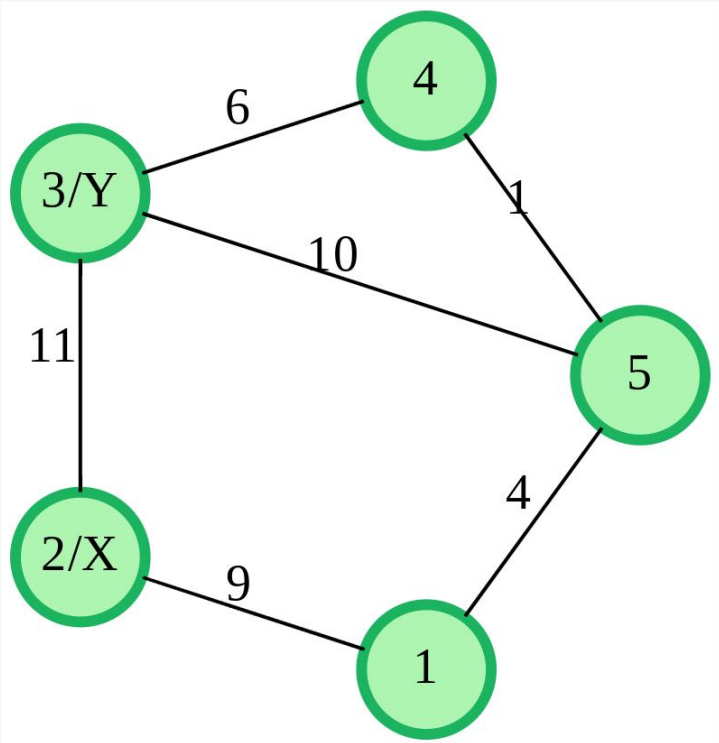
# 1. 228 CC - Two Junctions

***[Version traduite]***

Jack a deux courses à faire avant de pouvoir aller à l'école, et elles doivent être faites dans l'ordre. Le temps presse, donc il doit prendre le chemin le plus rapide. Déterminez le trajet le plus rapide depuis son point de départ, en passant par les deux lieux de courses, et en direction de son école. La ville sera représentée par un graphe non orienté de nœuds étiquetés de façon incrémentielle à partir de 1.

**Par exemple,** il y a g_nodes = 5, étiquetés de 1 à g_nodes inclus. Les nœuds représentent des intersections ou des points de repère le long d'un itinéraire. L'itinéraire de Jack commence toujours au nœud 1 et l'école est toujours au nœud g_nodes, dans ce cas, le nœud 5. Les nœuds de départ pour les routes, g_from = [1,2,3,4,5,3], les nœuds d'arrivée g_to = [2, 3, 4, 5, 1, 5] et les temps de trajet, donnés sous la forme de g_weight = [9, 11, 6, 1, 4, 10], sont alignés par indice. Dans cet exemple, il démarre au nœud 1, doit visiter x = nœud 2 puis y = nœud 3, dans cet ordre, avant d'aller à l'école au nœud 5. Ci dessous une représentation graphique du parcours






Il y a deux chemins qu'il peut prendre pour se rendre à l'école : 1 → 2 → 3 → 5 et 1 → 2 → 3 → 4 → 5. Le premier itinéraire prend 9 + 11 + 10 = 30 minutes de trajet tandis que le deuxième itinéraire ne prend que 9 + 11 + 6 + 1 = 27 minutes. La fonction devrait retourner 27 dans ce cas.

**Remarque : Jack peut visiter plusieurs fois un nœud.**

**Fonction**

Veuillez compléter la fonction `minCostPath` dans l'éditeur ci après.

La fonction `minCostPath` doit retourner le temps nécessaire pour parcourir le chemin le pl court du nœud 1 à travers x et ensuite y, arrivant enfin au nœud g_nodes.

La fonction `minCostPath` a les paramètres suivants :

g_from[g_from[0],...g_from[g_edges-1]]: un tableau d'entiers qui représentent les nœuds de départ des arêtes

g_to[g_to[0],...g_to[g_edges-1]]: un tableau d'entiers qui représentent les nœuds d'arrivée des arêtes

g_weight[g_weight[0],...g_weight[g_edges-1]]: un tableau d'entiers qui représentent le temps de trajet d'une arête, son poids

x: un entier, le premier nœud qui doit être sur le chemin

y: un entier, le deuxième nœud qui doit être sur le chemin

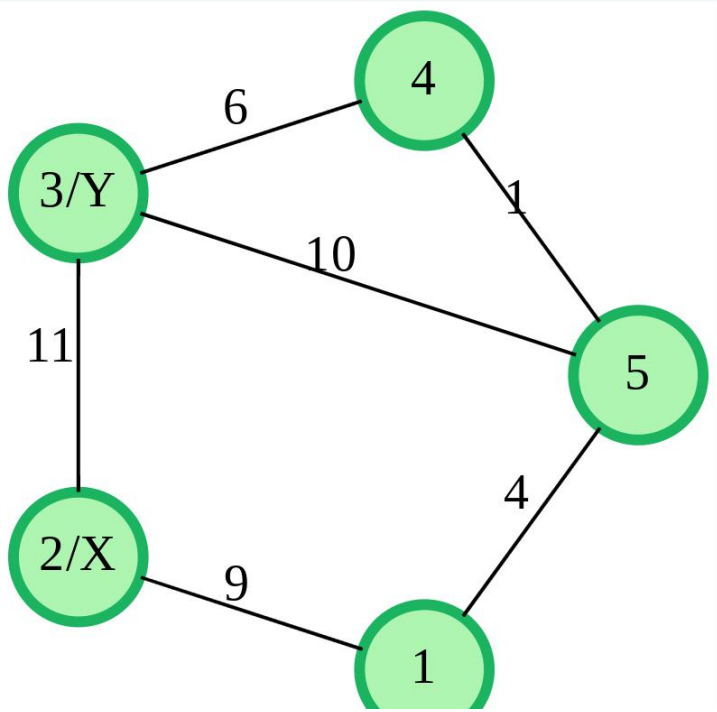
**Contraintes :**

- $4 \leq g\_nodes \leq 105$
- $4 \leq g\_edges \leq \min(105, (g\_nodes \times (g\_nodes - 1)) / 2)$
- $1 \leq g\_weight[i] \leq 103$
- $2 \leq x \leq g\_nodes-1$
- $2 \leq y \leq g\_nodes-1$

***[Version originale]***

Jack has two errands he needs to run before he can go to school, and they need to be done in order. Time is getting tight, so he needs to take the quickest route. Determine the fastest route from where he starts, through the two errand locations, and on to his school. The city will be represented as an undirected graph of nodes labeled incrementally from *1*.

**For example**, there are *g_nodes = 5*, labeled *1* to *g_nodes*, inclusive. Nodes represent junctions or milestones along a route. Jack's route always starts at node *1* and the school is always at node *g_nodes,* in this case, node *5.* The starting nodes for roads, *g_from = [1,2,3,4,5,3],* ending nodes *g_to = [2, 3, 4, 5, 1, 5]* and times to travel, given as *g_weight = [9, 11, 6, 1, 4, 10]*, are aligned by index . In this example, he starts at node *1*, has to visit *x = node 2* then *y = node 3*, in that order, before going to the school at node *5*. A graphical representation follows.



There are two paths he can take to get to school: *1 → 2 → 3 → 5* and *1 → 2 → 3 → 4 → 5*. The first route takes *9 + 11 + 10 = 30* travel time while the second route only takes *9 + 11 + 6 + 1 = 27*. The function should return *27* in this case.

**Note: Jack can visit any node multiple times.**

**Function Description**

Complete the function *minCostPath* in the editor below. The function must return the time it takes to travel the shortest path from node *1* through *x* and then *y*, arriving finally at node *g_nodes*.

minCostPath has the following parameter(s):

*g_from[g_from[0],...g_from[g_edges-1]]:* an array of integers that represent edge start nodes

*g_to[g_to[0],...g_to[g_edges-1]]:* an array of integers that represent edge end nodes

*g_weight[g_weight[0],...g_weight[g_edges-1]]:* an array of integers that represent time to travel an edge, its weight

*x*: an integer, the first node that must be on the path

*y*: an integer, the second node that must be on the path

**Constraints**

- $4 \leq g\_nodes \leq 10^5$
- $4 \leq g\_edges \leq min(10^5, (g\_nodes \times (g\_nodes - 1)) / 2)$
- $1 \leq g\_weight[i] \leq 10^3$

- $1 \le g\_weight[i] \le 10^3$
- $2 \le x \le g\_nodes-1$
- $2 \le y \le g\_nodes-1$

## ▶ Input Format For Custom Testing

## ▼ Sample Case 0

### Sample Input 0

```
4 5
1 2 6
1 4 9
2 4 10
2 3 6
3 4 11
2
3
```

### Sample Output 0

```
23
```

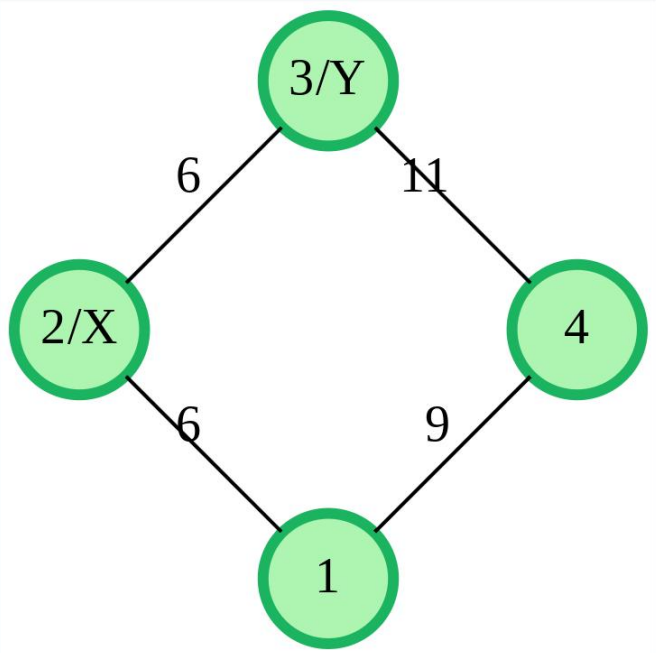
20m gauche

TOUS

1

2

## Explanation 0




For the given graph Jack has to go from junction *1* to junction *g_nodes* and his path should include junctions *2* and *3* in that order. The path that he selects is:
1 → 2 → 3 → 4, thus the total cost of this path is 6 + 6 + 11 = 23.

Langage Java 8 Prêt pour la saisie semi-automatique

Environnement

```
1  > import java.io.*; ...
14
15   class Result {
```

include junctions 2 and 3 in that order. The path that he selects is:
1 → 2 → 3 → 4, thus the total cost of this path is 6 + 6 + 11 = 23.

19m gauche

TOUS

1

2

Langage Java 8

Prêt pour la saisie semi-automatique

Environnement

```
27      /*
28       * For the weighted graph, <name>:
29       *
30       * 1. The number of nodes is <name>Nodes.
31       * 2. The number of edges is <name>Edges.
32       * 3. An edge exists between <name>From[i] and <name>To[i]. The
     weight of the edge is <name>Weight[i].
33       *
34       */
35
36      public static int minCostPath(int gNodes, List<Integer> gFrom,
     List<Integer> gTo, List<Integer> gWeight, int x, int y) {
37
38      }
39
40  }
41
42  public class Solution {...
```

Line: 14 C

19m gauche

TOUS

1

2

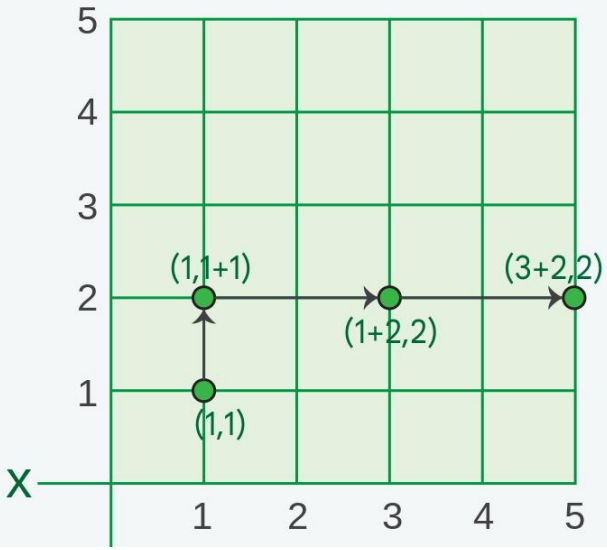
# 2. 228 CC - Reaching Points

***[Version traduite]***

Un robot est situé à un emplacement de coordonnées (x, y). Il doit être déplacé vers un autre emplacement avec une autre paire de coordonnées. Bien que le robot puisse se déplacer un nombre illimité de fois, il ne peut effectuer que les deux types de déplacements suivants :

1. De l'emplacement (x, y) à l'emplacement (x + y, y).
2. De l'emplacement (x, y) à l'emplacement (x, x + y).

**Par exemple,** si le robot démarre à (1, 1), il pourrait effectuer la séquence de mouvements suivante : (1, 1) → (1, 2) → (3, 2) → (5, 2). Notez que le mouvement sera toujours vers le haut ou vers la droite.



19m gauche

TOUS

1

2

Étant donné les coordonnées de départ et d'arrivée, déterminez si le robot peut atteindre les coordonnées d'arrivée selon les règles de déplacement.

**Fonction**

La fonction `canReach` doit retourner la chaîne "Yes" si le robot peut atteindre son objectif, sinon retourner "No".

La fonction `canReach` a les paramètres suivants :

- x1: valeur entière, coordonnée x de départ
- y1: valeur entière, coordonnée y de départ
- x2: valeur entière, coordonnée x cible
- y2: valeur entière, coordonnée y cible

**Contraintes :**

- $1 \leq x1, y1, x2, y2 \leq 1000$

***[Version originale]***

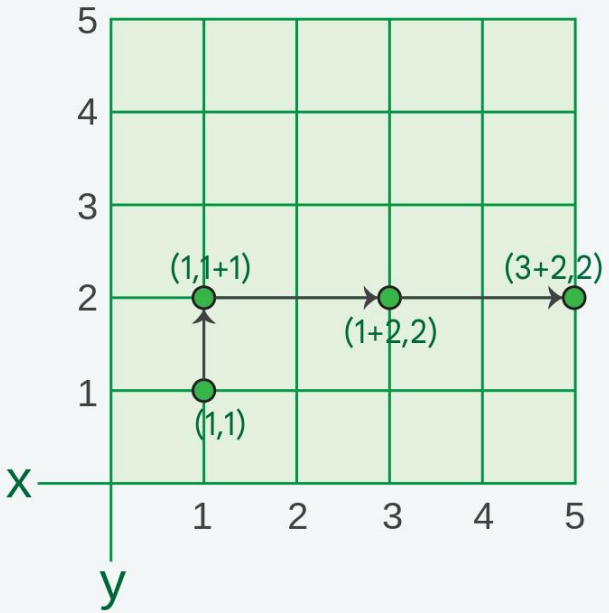
There is a bot located at a pair of integer coordinates, *(x, y)*. It must be moved to a location with another set of coordinates. Though the bot can move any number of times, it can only make t following *two types* of moves:

1. From location *(x, y)* to location *(x + y, y)*.

1. From location *(x, y)* to location *(x + y, y)*.

2. From location *(x, y)* to location *(x, x + y)*.

For example, if the bot starts at *(1, 1)*, it might make the following sequence of moves: *(1, 1) → (1, 2) → (3, 2) → (5, 2)*. Note that movement will always be either up or to the right.




Given starting and target ending coordinates, determine whether the bot can reach the ending coordinates given the rules of movement.

**Function Description**

**Function Description**

Complete the function *canReach* in the editor below. The function must return the string *Yes* if the bot can reach its goal, otherwise return *No*.

canReach has the following parameter(s):

- *x1:* integer value, starting x coordinate
- *y1:* integer value, starting y coordinate
- *x2:* integer value, target x coordinate
- *y2:* integer value, target y coordinate

**Constraints**

- $1 \leq x1, y1, x2, y2 \leq 1000$

▶ Input Format for Custom Testing

▼Sample Case 0

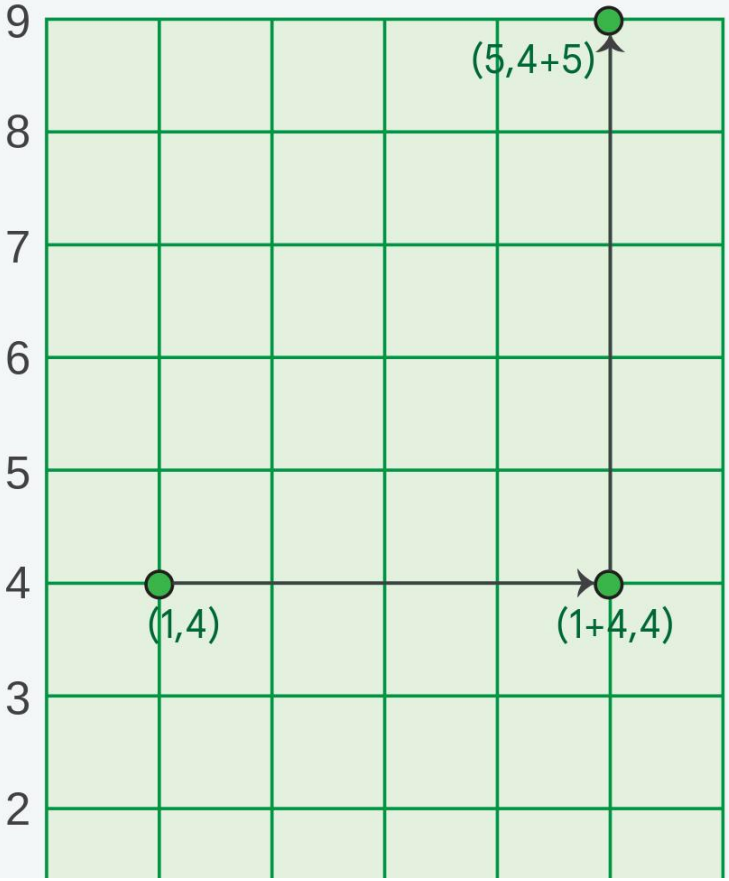
**Sample Input 0**

```
1
4
5
9
```

## Sample Output 0

```
Yes
```

## Explanation 0

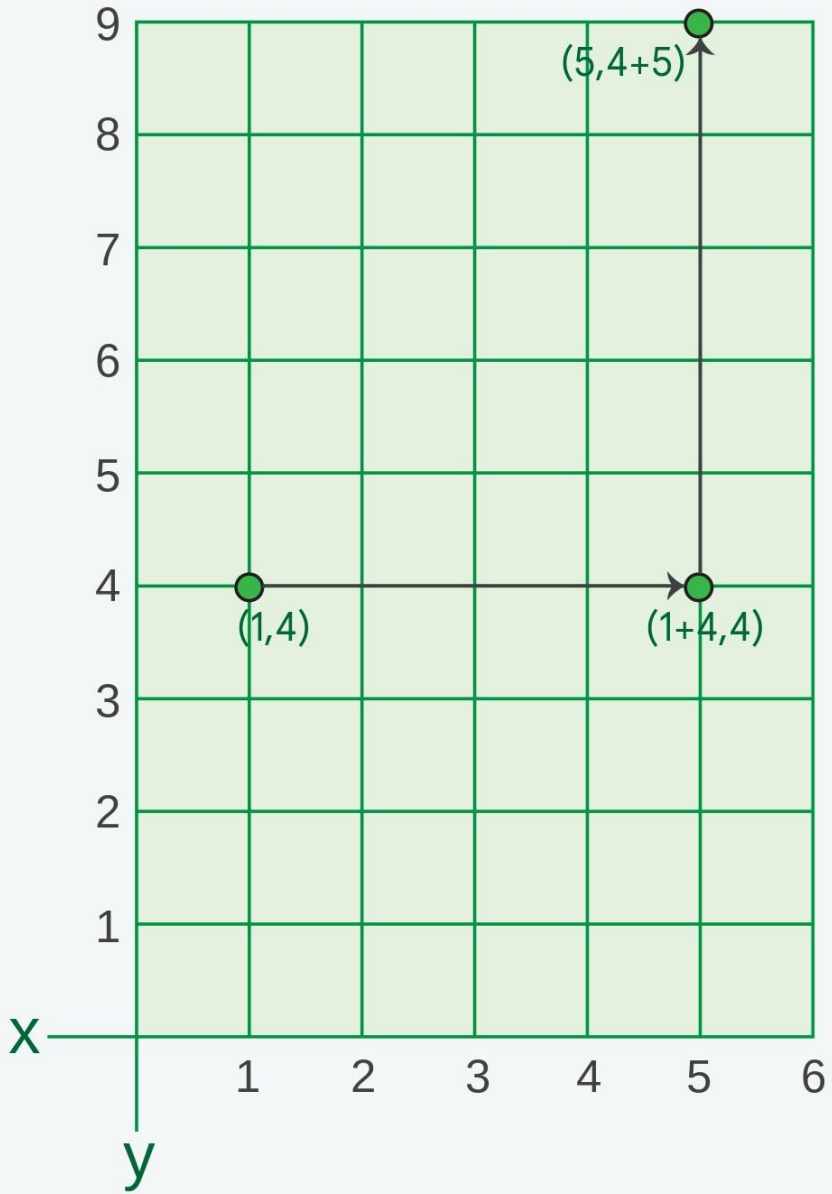


19m gauche

TOUS

1

2




*start = (1, 4), end = (5, 9)*

The bot starts at *(1, 4)* and makes a move of type *1*, meaning that it moves to *(1 + 4, 1) = (5, 4)*.

19m gauche

TOUS

1

2

*start = (1, 4), end = (5, 9)*
The bot starts at *(1, 4)* and makes a move of type *1*, meaning that it moves to *(1 + 4, 1) = (5, 4)*.
Then it makes a move of type *2* from *(5, 4)* to *(5, 5 + 4) = (5, 9)*.

▶ Sample Case 1

Langage Java 8 ✓ Prêt pour la saisie semi-automatique

Environnement

```
1  > import java.io.*; ...
14
15 class Result {
16
17     /*
18      * Complete the 'canReach' function below.
19      *
20      * The function is expected to return a STRING.
21      * The function accepts following parameters:
22      *  1. INTEGER x1
23      *  2. INTEGER y1
24      *  3. INTEGER x2
25      *  4. INTEGER y2
26      */
27
28     public static String canReach(int x1, int y1, int x2, int y2) {
```

19m gauche

TOUS

1

2

*start = (1, 4), end = (5, 9)*
The bot starts at *(1, 4)* and makes a move of type *1*, meaning that it moves to *(1 + 4, 1) = (5, 4)*.
Then it makes a move of type *2* from *(5, 4)* to *(5, 5 + 4) = (5, 9)*.

▶ Sample Case 1

Langage Java 8 ✓ Prêt pour la saisie semi-automatique

Environnement

```
18      * Complete the 'canReach' function below.
19      *
20      * The function is expected to return a STRING.
21      * The function accepts following parameters:
22      *  1. INTEGER x1
23      *  2. INTEGER y1
24      *  3. INTEGER x2
25      *  4. INTEGER y2
26      */
27
28     public static String canReach(int x1, int y1, int x2, int y2) {
29     // Write your code here
30
31     }
32
33 }
```