→ very large power

→ Disjoint intervals

→ Palindromic Substring Count

→ Consistent strings

→ Network Delay Time.

①

Given two Integers A, B. You have to calculate (A ^ (B!)) % (1e9 + 7).

$A^{B!} \% p$

$p \rightarrow 10^9+7$.

$1 \le A, B \le 5 \times 10^5$

$$a^{m-1} \% m = 1$$

$B! \rightarrow x * (m-1) + r$

$$A^{B!} \% m = \left(A^{x*(m-1) + B! \% (m-1)} \right) \% m$$

$$= \left(A^{x*(m-1)} \times A^{B! \% (m-1)} \right) \% m$$

$$= \left(\left(A^{x*(m-1)} \right) \% m \times \left(A^{B! \% (m-1)} \right) \% m \right) \% m$$

$\left(A^{x*(m-1)} \right) \% m = 1$

$$A^{B!} \% m = \left(A^{B! \% (m-1)} \right) \% m$$

code :->

```
long fact = 1, m = 10^9 + 7.
for( i = 1; i <= B; i++) {
    fact = (fact * i) % (m-1);
}
```

→ fastpower(A, fact, m);

T.C → O(B + log m)
S.C → O(log m)

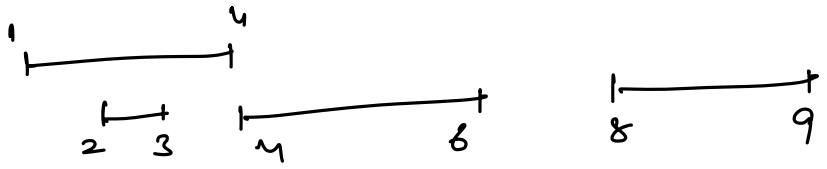
Disjoint intervals →

Given a set of N intervals denoted by 2D array A of size N x 2, the task is to find the length of maximal set of mutually disjoint intervals.

Two intervals [x, y] & [p, q] are said to be disjoint if they do not have any point in common.

Return a integer denoting the length of maximal set of mutually disjoint intervals.

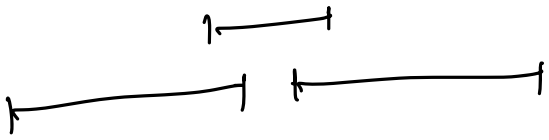
A → $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 4 & 6 \\ 8 & 9 \end{bmatrix}$



ans = 3

A → $\begin{bmatrix} 1 & 9 \\ 2 & 3 \\ 5 & 7 \end{bmatrix}$

ans = 2



→ Sort on starting time ✗

→ Sort on smallest duration ✗

→ Sort on end-time? ✓

```
public class Solution {
    public int solve(int[][] arr) {
        //arr[i][0] -> start time of the ith interval
        //arr[i][0] -> end time of the ith interval

        Arrays.sort(arr, new Comparator<int[]>(){
            public int compare(int[] a, int[] b){
                return a[1] - b[1];
            }
            });

        int ans = 1;
        int r = arr[0][1];

        for(int i = 1; i < arr.length; i++){
            if(arr[i][0] > r){
                ans++;
                r = arr[i][1];
            }
        }
        return ans;
    }
}
```

③ Palindromic Substring Count →

Given a string A consisting of lowercase English alphabets. Your task is to find how many substrings of A are palindrome.

The substrings with different start indexes or end indexes are counted as different substrings even if they consist of same characters.

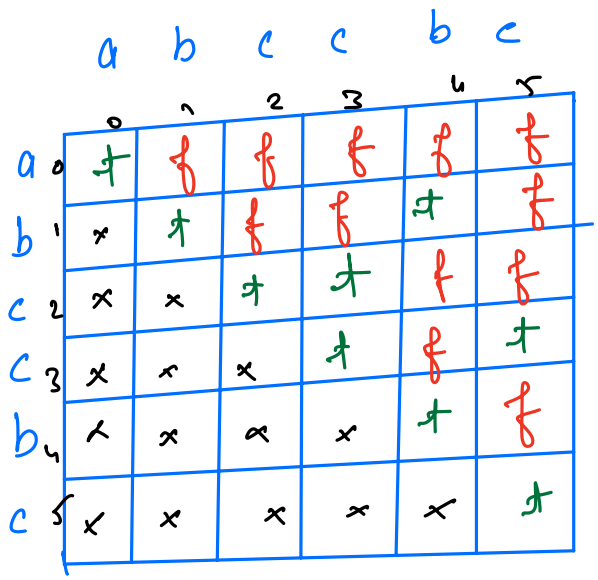
Return the count of palindromic substrings.

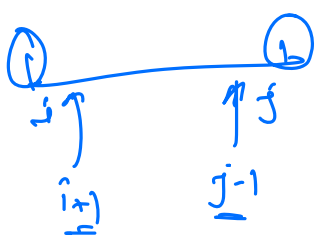
Str → abccbc

$1 \le n1 \le 10^3$

B.f. idea. →

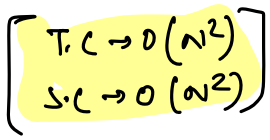
Consider all the substrings.





ans = 9.

```
int count = 0;
        boolean[][] dp = new boolean[str.length()][str.length()];
        for(int gap = 0; gap < str.length(); gap++){
            for(int si = 0, ei = gap; ei < str.length(); si++,ei++){
                if(gap == 0){
                    dp[si][ei] = true;
                }else if(gap == 1){
                    dp[si][ei] = str.charAt(si) == str.charAt(ei) ? true : false;
                }else{
                    if(str.charAt(si) == str.charAt(ei)){
                        dp[si][ei] = dp[si + 1][ei - 1];
                    }else{
                        dp[si][ei] = false;
                    }
                }
                if(dp[si][ei] == true){
                    count++;
                }
            }
        }
        return count;
```



(4)

You are given a string A consisting of distinct characters and an array of strings B. A string is consistent if all characters in the string appear in the string A.

Return the number of consistent strings in the array B.

A = "ab"

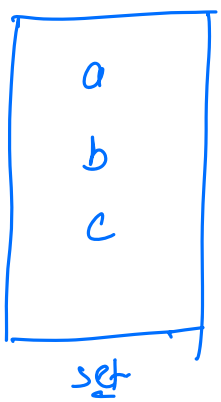
B → ["al", "abb", "baaba a"]

ans = 2.

A = "abc"

B → ["dab", "ba", "abbca"]

ans = 2.



dab. ✗

ba ✓

abbca ✓

$1 \le N \le 10^5$

$1 \le$ len of str in array ≤ 10

```
public class Solution {
    public int solve(String A, String[] B) {
        HashSet<Character> set = new HashSet<>();
        for(int i = 0; i < A.length(); i++){
            set.add(A.charAt(i));
        }

        int count = 0;

        for(int i = 0 ; i < B.length; i++){
            String str = B[i];
            boolean isConsistent = true;
            for(int j = 0; j < str.length(); j++){
                char c = str.charAt(j);
                if(set.contains(c) == false){
                    isConsistent = false;
                    break;
                }
            }
            if(isConsistent == true){
                count++;
            }
        }

        return count;
    }
}
```

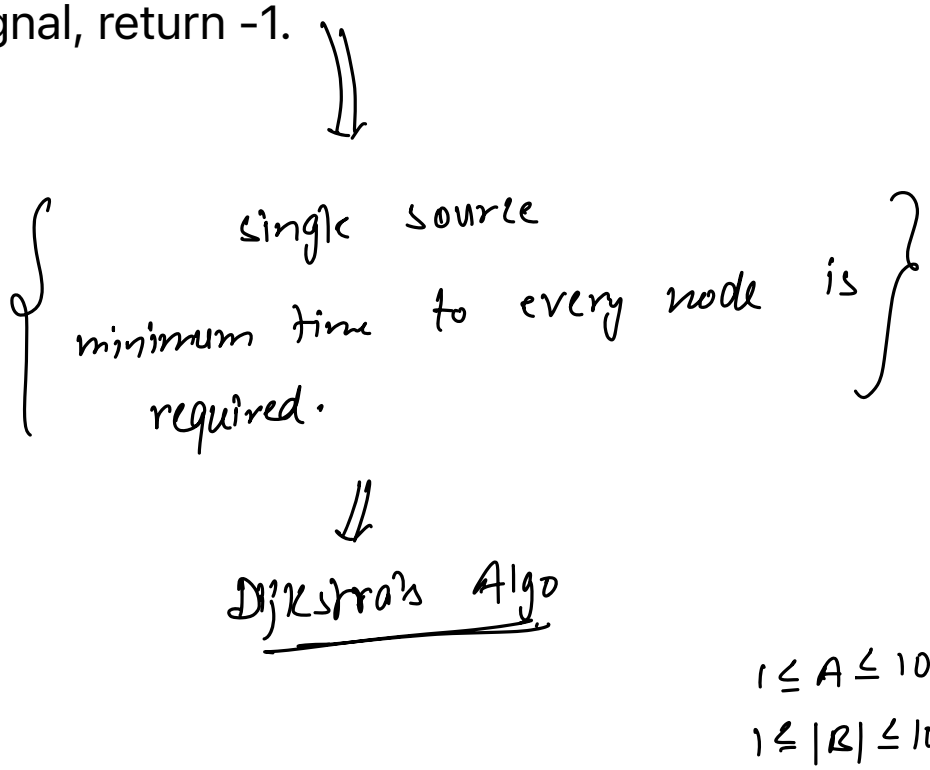
T.C → O(N x maxlen(B[i]))

S.C → O(1)

Network Delay Time

You are given a network of A nodes, labeled from 1 to A. You are also given B, a list of travel times as directed edges B[i] = (ui, vi, wi), where ui is the source node, vi is the target node, and wi is the time it takes for a signal to travel from source to target.

We will send a signal from a given node C. Return the minimum time it takes for all the A nodes to receive the signal. If it is impossible for all the A nodes to receive the signal, return -1.



$1 \le A \le 10^5$

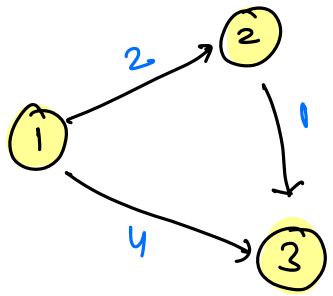
$1 \le |B| \le 10^5$

Eg. →

A = 3

src = 1.

$$B \rightarrow \begin{bmatrix} 1 & 3 & 4 \\ 1 & 2 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$



dist →

0	2	3
∞	∞	∞
1	2	3


```
Pair {
    int v;
    int wsf;
}
```

```
public int solve(int A, int[][] B, int C) {
    int[] dist = new int[A + 1];
    for(int i = 0 ; i < dist.length; i++){
        dist[i] = Integer.MAX_VALUE;
    }

    PriorityQueue<Pair> pq = new PriorityQueue<>();
    pq.add(new Pair(C, 0));

    while(pq.size() != 0){
        Pair rp = pq.remove();
        if(dist[rp.v] != Integer.MAX_VALUE){
            continue;
        }
        dist[rp.v] = rp.wsf;
        for(int[] arr : B){
            if(arr[0] == rp.v && dist[arr[1]] == Integer.MAX_VALUE){
                pq.add(new Pair(arr[1], rp.wsf + arr[2]));
            }
        }
    }

    int ans = 0;
    for(int i = 1; i < dist.length; i++){
        ans = Math.max(dist[i], ans);
    }

    return ans == Integer.MAX_VALUE ? -1 : ans;
}
```

T.C → O(E log E)
S.C → O(E+N)

R1

31 May → 2 June.

12:00 A.M

R2

9 Jun → 18 Jun

R3

18 Jun → 2 Aug.

Target ⇒ 30th June