Problem 3615: Longest Palindromic Path in Graph

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an integer

n

and an

undirected

graph with

n

nodes labeled from 0 to

n - 1

and a 2D array

edges

, where

edges[i] = [u

i

, v

i

]

indicates an edge between nodes

u

i

and

v

i

.

You are also given a string

label

of length

n

, where

label[i]

is the character associated with node

i

.

You may start at any node and move to any adjacent node, visiting each node

at most

once.

Return the

maximum

possible length of a

palindrome

that can be formed by visiting a set of

unique

nodes along a valid path.

Example 1:

Input:

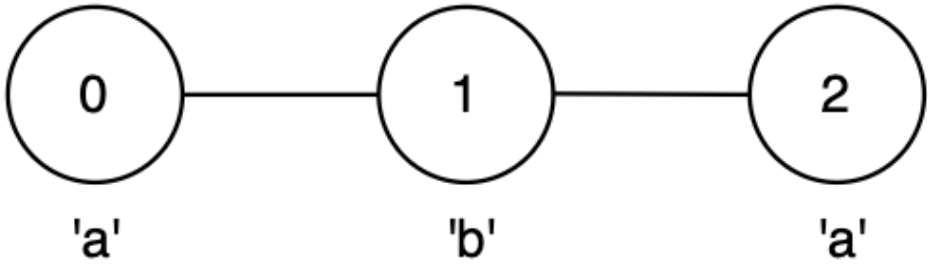
n = 3, edges = [[0,1],[1,2]], label = "aba"

Output:

3

Exp

lanation:



The longest palindromic path is from node 0 to node 2 via node 1, following the path

$0 \rightarrow 1 \rightarrow 2$

forming string

"aba"

.

This is a valid palindrome of length 3.

Example 2:

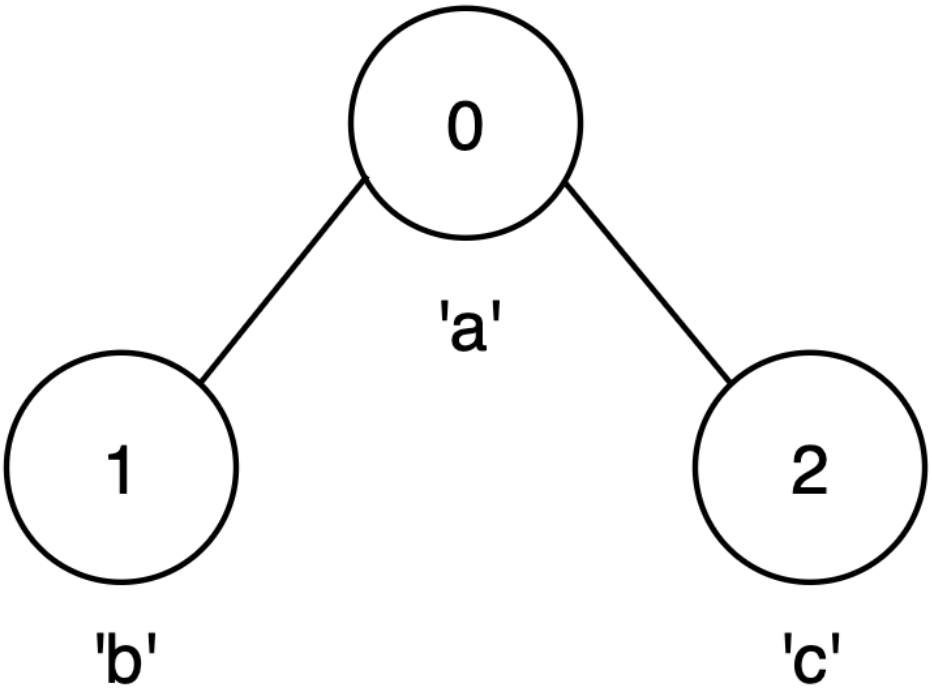
Input:

n = 3, edges = [[0,1],[0,2]], label = "abc"

Output:

1

Explanation:



No path with more than one node forms a palindrome.

The best option is any single node, giving a palindrome of length 1.

Example 3:

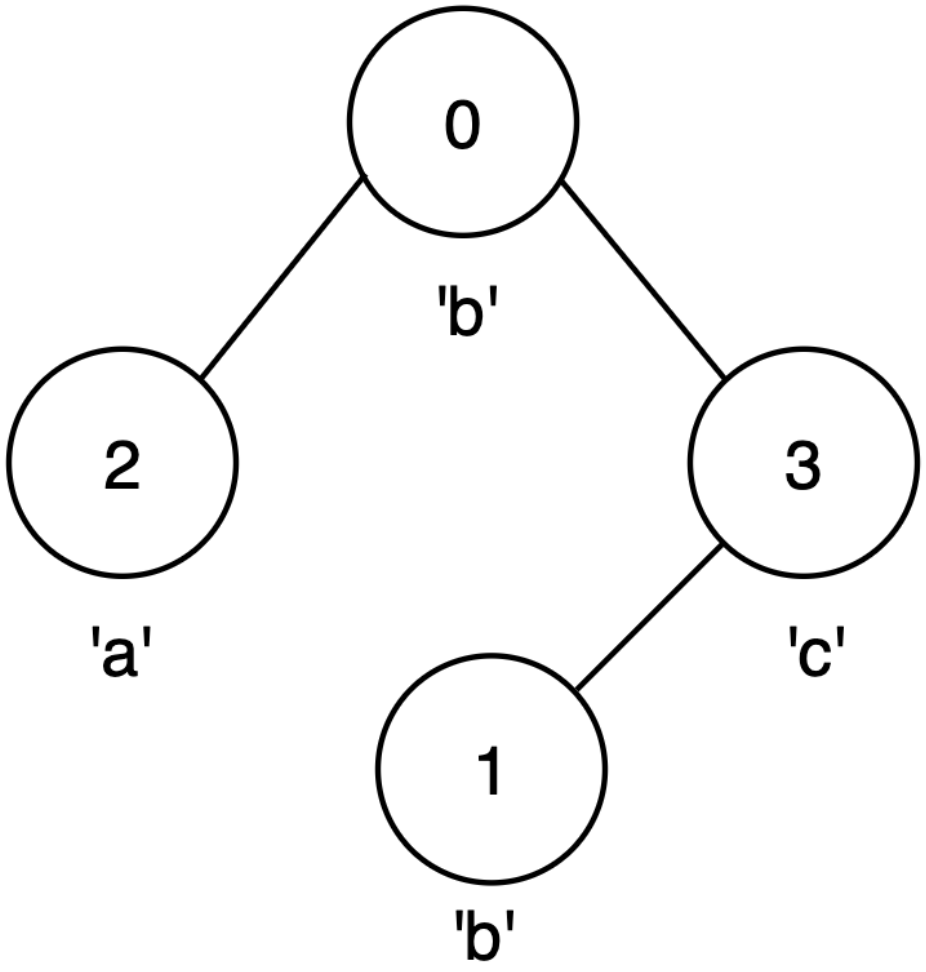
Input:

n = 4, edges = [[0,2],[0,3],[3,1]], label = "bbac"

Output:

3

Explanation:



The longest palindromic path is from node 0 to node 1, following the path

$0 \rightarrow 3 \rightarrow 1$

, forming string

"bcb"

.

This is a valid palindrome of length 3.

Constraints:

1 <= n <= 14

n - 1 <= edges.length <= n * (n - 1) / 2

edges[i] == [u

i

, v

i

]

0 <= u

i

, v

i

<= n - 1

u

i

!= v

i

label.length == n

label

consists of lowercase English letters.

There are no duplicate edges.

Code Snippets

C++:

```
class Solution {
public:
int maxLen(int n, vector<vector<int>>& edges, string label) {

}
};
```

Java:

```
class Solution {
public int maxLen(int n, int[][] edges, String label) {

}
}
```

Python3:

```
class Solution:
def maxLen(self, n: int, edges: List[List[int]], label: str) -> int:
```

Python:

```
class Solution(object):
def maxLen(self, n, edges, label):
"""
:type n: int
:type edges: List[List[int]]
:type label: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {string} label
* @return {number}
*/
var maxLen = function(n, edges, label) {
```

```
};
```

TypeScript:

```
function maxLen(n: number, edges: number[][], label: string): number {

};
```

C#:

```
public class Solution {
public int MaxLen(int n, int[][] edges, string label) {

}
}
```

C:

```
int maxLen(int n, int** edges, int edgesSize, int* edgesColSize, char* label)
{

}
```

Go:

```
func maxLen(n int, edges [][]int, label string) int {

}
```

Kotlin:

```
class Solution {
fun maxLen(n: Int, edges: Array<IntArray>, label: String): Int {

}
}
```

Swift:

```
class Solution {
func maxLen(_ n: Int, _ edges: [[Int]], _ label: String) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn max_len(n: i32, edges: Vec<Vec<i32>>, label: String) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {String} label
# @return {Integer}
def max_len(n, edges, label)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param String $label
* @return Integer
*/
function maxLen($n, $edges, $label) {

}
}
```

Dart:

```
class Solution {
int maxLen(int n, List<List<int>> edges, String label) {

}
```

```
}
```

Scala:

```
object Solution {
def maxLen(n: Int, edges: Array[Array[Int]], label: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_len(n :: integer, edges :: [[integer]], label :: String.t) ::
integer
def max_len(n, edges, label) do

end
end
```

Erlang:

```
-spec max_len(N :: integer(), Edges :: [[integer()]], Label ::
unicode:unicode_binary()) -> integer().
max_len(N, Edges, Label) ->
.
```

Racket:

```
(define/contract (max-len n edges label)
(-> exact-integer? (listof (listof exact-integer?)) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Palindromic Path in Graph
* Difficulty: Hard
```

```
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxLen(int n, vector<vector<int>>& edges, string label) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Palindromic Path in Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxLen(int n, int[][] edges, String label) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Palindromic Path in Graph
Difficulty: Hard
Tags: array, string, graph, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxLen(self, n: int, edges: List[List[int]], label: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxLen(self, n, edges, label):
"""
:type n: int
:type edges: List[List[int]]
:type label: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Palindromic Path in Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number[][]} edges
* @param {string} label
* @return {number}
*/
var maxLen = function(n, edges, label) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Palindromic Path in Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxLen(n: number, edges: number[][], label: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Palindromic Path in Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxLen(int n, int[][] edges, string label) {

}
}
```

C Solution:

```
/*
* Problem: Longest Palindromic Path in Graph
* Difficulty: Hard
* Tags: array, string, graph, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxLen(int n, int** edges, int edgesSize, int* edgesColSize, char* label)
{

}
```

Go Solution:

```
// Problem: Longest Palindromic Path in Graph
// Difficulty: Hard
// Tags: array, string, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxLen(n int, edges [][]int, label string) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxLen(n: Int, edges: Array<IntArray>, label: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxLen(_ n: Int, _ edges: [[Int]], _ label: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Palindromic Path in Graph
// Difficulty: Hard
// Tags: array, string, graph, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_len(n: i32, edges: Vec<Vec<i32>>, label: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {String} label
# @return {Integer}
def max_len(n, edges, label)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param String $label
* @return Integer
*/
function maxLen($n, $edges, $label) {

}
}
```

Dart Solution:

```
class Solution {
int maxLen(int n, List<List<int>> edges, String label) {

}
}
```

Scala Solution:

```
object Solution {
def maxLen(n: Int, edges: Array[Array[Int]], label: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_len(n :: integer, edges :: [[integer]], label :: String.t) ::
integer
def max_len(n, edges, label) do

end
end
```

Erlang Solution:

```
-spec max_len(N :: integer(), Edges :: [[integer()]], Label ::
unicode:unicode_binary()) -> integer().
max_len(N, Edges, Label) ->
.
```

Racket Solution:

```
(define/contract (max-len n edges label)
(-> exact-integer? (listof (listof exact-integer?)) string? exact-integer?)
)
```