Problem 1938: Maximum Genetic Difference Query

Problem Information

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

Problem Description

There is a rooted tree consisting of

n

nodes numbered

0

to

n - 1

. Each node's number denotes its

unique genetic value

(i.e. the genetic value of node

x

is

x

). The

genetic difference

between two genetic values is defined as the

bitwise-

XOR

of their values. You are given the integer array

parents

, where

parents[i]

is the parent for node

i

. If node

x

is the

root

of the tree, then

parents[x] == -1

.

You are also given the array

queries

where

queries[i] = [node

i

, val

i

]

. For each query

i

, find the

maximum genetic difference

between

val

i

and

p

i

, where

p

i

is the genetic value of any node that is on the path between

node

i

and the root (including

node

i

and the root). More formally, you want to maximize

val

i

XOR p

i

.

Return

an array

ans

where

ans[i]

is the answer to the

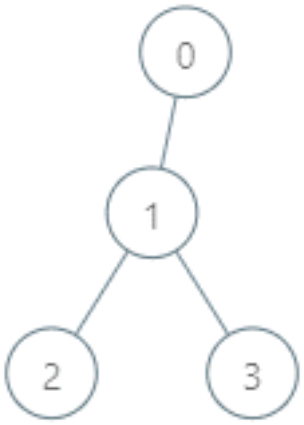
i

th

query

.

Example 1:



Input:

parents = [-1,0,1,1], queries = [[0,2],[3,2],[2,5]]

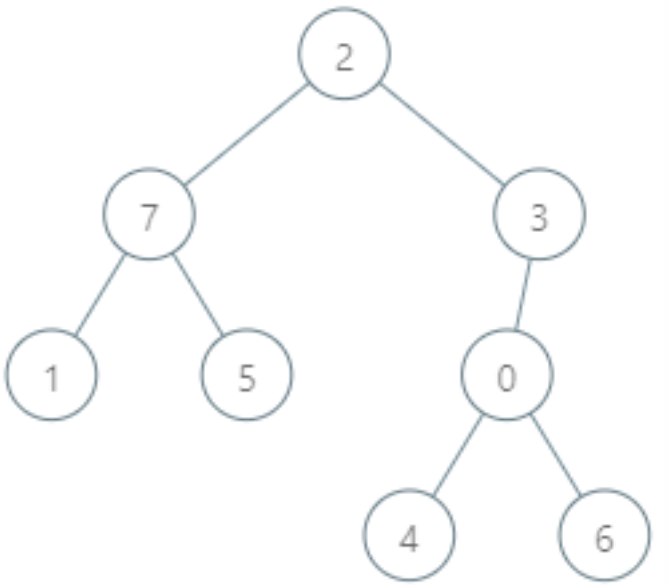
Output:

[2,3,7]

Explanation:

The queries are processed as follows: - [0,2]: The node with the maximum genetic difference is 0, with a difference of 2 XOR 0 = 2. - [3,2]: The node with the maximum genetic difference is 1, with a difference of 2 XOR 1 = 3. - [2,5]: The node with the maximum genetic difference is 2, with a difference of 5 XOR 2 = 7.

Example 2:



Input:

parents = [3,7,-1,2,0,7,0,2], queries = [[4,6],[1,15],[0,5]]

Output:

[6,14,7]

Explanation:

The queries are processed as follows: - [4,6]: The node with the maximum genetic difference is 0, with a difference of 6 XOR 0 = 6. - [1,15]: The node with the maximum genetic difference is 1, with a difference of 15 XOR 1 = 14. - [0,5]: The node with the maximum genetic difference is 2, with a difference of 5 XOR 2 = 7.

Constraints:

2 <= parents.length <= 10

5

0 <= parents[i] <= parents.length - 1

for every node

i

that is

not

the root.

parents[root] == -1

1 <= queries.length <= 3 * 10

4

0 <= node

i

<= parents.length - 1

0 <= val

i

<= 2 * 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> maxGeneticDifference(vector<int>& parents, vector<vector<int>>&
queries) {

}
};
```

Java:

```
class Solution {
public int[] maxGeneticDifference(int[] parents, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def maxGeneticDifference(self, parents: List[int], queries: List[List[int]])
-> List[int]:
```

Python:

```
class Solution(object):
def maxGeneticDifference(self, parents, queries):
"""
:type parents: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} parents
* @param {number[][]} queries
* @return {number[]}
*/
var maxGeneticDifference = function(parents, queries) {

};
```

TypeScript:

```
function maxGeneticDifference(parents: number[], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public int[] MaxGeneticDifference(int[] parents, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxGeneticDifference(int* parents, int parentsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func maxGeneticDifference(parents []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun maxGeneticDifference(parents: IntArray, queries: Array<IntArray>):
IntArray {

}
}
```

Swift:

```
class Solution {
func maxGeneticDifference(_ parents: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn max_genetic_difference(parents: Vec<i32>, queries: Vec<Vec<i32>>) ->
```

```
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} parents
# @param {Integer[][]} queries
# @return {Integer[]}
def max_genetic_difference(parents, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $parents
* @param Integer[][] $queries
* @return Integer[]
*/
function maxGeneticDifference($parents, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> maxGeneticDifference(List<int> parents, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def maxGeneticDifference(parents: Array[Int], queries: Array[Array[Int]]):
Array[Int] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec max_genetic_difference(parents :: [integer], queries :: [[integer]]) ::
[integer]
def max_genetic_difference(parents, queries) do

end
end
```

Erlang:

```
-spec max_genetic_difference(Parents :: [integer()], Queries ::
[[integer()]]) -> [integer()].
max_genetic_difference(Parents, Queries) ->
.
```

Racket:

```
(define/contract (max-genetic-difference parents queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Genetic Difference Query
* Difficulty: Hard
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
class Solution {
public:
vector<int> maxGeneticDifference(vector<int>& parents, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Genetic Difference Query
* Difficulty: Hard
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] maxGeneticDifference(int[] parents, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Genetic Difference Query
Difficulty: Hard
Tags: array, tree, hash, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def maxGeneticDifference(self, parents: List[int], queries: List[List[int]])
-> List[int]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxGeneticDifference(self, parents, queries):
"""
:type parents: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Genetic Difference Query
* Difficulty: Hard
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} parents
* @param {number[][]} queries
* @return {number[]}
*/
var maxGeneticDifference = function(parents, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Genetic Difference Query
* Difficulty: Hard
* Tags: array, tree, hash, search
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function maxGeneticDifference(parents: number[], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Genetic Difference Query
* Difficulty: Hard
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] MaxGeneticDifference(int[] parents, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Genetic Difference Query
* Difficulty: Hard
* Tags: array, tree, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maxGeneticDifference(int* parents, int parentsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximum Genetic Difference Query
// Difficulty: Hard
// Tags: array, tree, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxGeneticDifference(parents []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maxGeneticDifference(parents: IntArray, queries: Array<IntArray>):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maxGeneticDifference(_ parents: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Genetic Difference Query
// Difficulty: Hard
// Tags: array, tree, hash, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn max_genetic_difference(parents: Vec<i32>, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} parents
# @param {Integer[][]} queries
# @return {Integer[]}
def max_genetic_difference(parents, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $parents
* @param Integer[][] $queries
* @return Integer[]
*/
function maxGeneticDifference($parents, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maxGeneticDifference(List<int> parents, List<List<int>> queries) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maxGeneticDifference(parents: Array[Int], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_genetic_difference(parents :: [integer], queries :: [[integer]]) ::
[integer]
def max_genetic_difference(parents, queries) do

end
end
```

Erlang Solution:

```
-spec max_genetic_difference(Parents :: [integer()], Queries ::
[[integer()]]) -> [integer()].
max_genetic_difference(Parents, Queries) ->
.
```

Racket Solution:

```
(define/contract (max-genetic-difference parents queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```