seanpgallivan
Posted on Jun 13, 2021

Solution: Palindrome Pairs

#algorithms #javascript #java #python

Leetcode Solutions (161 Part Series)

1	Solution: Next Permutation
2	Solution: Trim a Binary Search Tree
...	**157 more parts...**
160	Solution: Out of Boundary Paths
161	Solution: Redundant Connection

This is part of a series of Leetcode solution explanations (index). If you liked this solution or found it useful, ***please like*** *this post and/or* ***upvote*** *my solution post on Leetcode's*

forums.

Leetcode Problem #336 (*Hard*): Palindrome Pairs

Description:

(*Jump to*: *Solution Idea* || *Code*: *JavaScript* | *Python* | *Java* | *C++*)

> Given a list of **unique** `words`, return all the pairs of the **distinct** indices (`i, j`) in the given list, so that the concatenation of the two words `words[i] + words[j]` is a palindrome.

Examples:

> | **Example 1:** | |
> |---|---|
> | Input: | words = ["abcd","dcba","lls","s","sssll"] |
> | Output: | [[0,1],[1,0],[3,2],[2,4]] |
> | Explanation: | The palindromes are ["dcbaabcd","abcddcba","slls","llssssll"] |
>
> | **Example 2:** | |
> |---|---|
> | Input: | words = ["bat","tab","cat"] |
> | Output: | [[0,1],[1,0]] |
> | Explanation: | The palindromes are ["battab","tabbat"] |
>
> | **Example 3:** | |
> |---|---|
> | Input: | words = ["a",""] |

Output:	[[0,1],[1,0]]

Constraints:

- `1 <= words.length <= 5000`
- `0 <= words[i].length <= 300`
- `words[i]` consists of lower-case English letters.

Idea:

(*Jump to*: *Problem Description* || *Code*: *JavaScript* | *Python* | *Java* | *C++*)

A naive approach here would attempt every possible pairing of **words**, but that would be inefficient. Instead, we can figure out what possible words would pair with each word and specifically check for those.

To do this, we'll first have to store each word in a **map** structure (**wmap**), with the word as the key and the index as the value. This way, we can look up any possible matches with the current word as we iterate through **words**.

The next thing we'll want to do is define a helper function (**isPal**) to check if a word is a palindrome. Rather than having to pass it a substring of a word, we can define it to take a range of indexes to check, so that we're not constantly building new strings.

As we iterate through **words**, then, each word will possibly match another word in one of three ways:

- A blank string word will match on either side with any palindrome word. *(e.g. "" will match with "abc" and vice versa)*
- A full word will match on either side with its backwards version. *(e.g. "abc" will match with "cba", and vice versa)*
- A partial word will match its backwards version on the opposite side if the leftover portion of the word is a palindrome *(e.g. "abcddd" will match with "cba" because "abc" matches with "cba" and "ddd" is a palindrome)*

~~abc matches with "cba" and "dddd" is a palindrome)~~

The first check is easy to perform. If we find a blank string, we can iterate through the entire **words** list an extra time searching for palindromes to match. We just need to remember not to match the blank string with itself.

For the second check, since we'll eventually iterate to the matching full word, we should only add the one pair at this time, rather than both, as we'll be able to add the second ordering of the same pair when we get to the second word.

The third check is the most difficult. For this, we'll want to first reverse the current word to its backwards version (**bw**), since we'll be matching existing frontwards words in **wmap**. Then we should iterate through the indexes of the word itself, testing both sides of the dividing index (**j**) for being a palindrome.

If a palindrome is found, then we can attempt to lookup the *other* portion of the word in **wmap**. If a match is found, we can push that pair to our answer array (**ans**). At the end of the iteration of **words**, we can **return ans**.

- ***Time Complexity: O(N * M^2)*** *where* ***N*** *is the length of* ***words*** *and* ***M*** *is the average length of the words in* ***words***
- ***Space Complexity: O(N)*** *for* ***wmap***

Javascript Code:

(*Jump to*: *Problem Description* || *Solution Idea*)

```
var palindromePairs = function(words) {
    let wmap = new Map(), ans = []
    for (let i = 0; i < words.length; i++)
        wmap.set(words[i], i)
    for (let i = 0; i < words.length; i++) {
        if (words[i] === "") {
            for (let j = 0; j < words.length; j++)
                if (isPal(words[j]) && j !== i)
                    ans.push([i, j], [j, i])
            continue
        }
```

```
        let bw = words[i].split("").reverse().join("")
        let res = wmap.get(bw)
        if (res !== undefined && res !== i)
            ans.push([i, res])
        for (let j = 1; j < bw.length; j++) {
            if (isPal(bw, 0, j - 1)) {
                let res = wmap.get(bw.slice(j))
                if (res !== undefined)
                    ans.push([i, res])
            }
            if (isPal(bw, j)) {
                let res = wmap.get(bw.slice(0,j))
                if (res !== undefined)
                    ans.push([res, i])
            }
        }
    }
    return ans
};

const isPal = (word, i=0, j=word.length-1) => {
    while (i < j)
        if (word[i++] !== word[j--]) return false
    return true
}
```

Python Code:

(*Jump to*: *Problem Description* || *Solution Idea*)

```
class Solution:
    def palindromePairs(self, words: List[str]) -> List[List[int]]:
        wmap, ans = {}, []
        for i in range(len(words)):
            wmap[words[i]] = i
        for i in range(len(words)):
            if words[i] == "":
                for j in range(len(words)):
                    w = words[j]
                    if self.isPal(w, 0, len(w)-1) and j != i:
                        ans.append([i, j])
```

```
                    ans.append([j, i])
                continue
            bw = words[i][::-1]
            if bw in wmap:
                res = wmap[bw]
                if res != i: ans.append([i, res])
            for j in range(1, len(bw)):
                if self.isPal(bw, 0, j - 1) and bw[j:] in wmap:
                    ans.append([i, wmap[bw[j:]]])
                if self.isPal(bw, j, len(bw)-1) and bw[:j] in wmap:
                    ans.append([wmap[bw[:j]], i])
        return ans

    def isPal(self, word: str, i: int, j: int) -> bool:
        while i < j:
            if word[i] != word[j]: return False
            i += 1
            j -= 1
        return True
```

Java Code:

(*Jump to*: *Problem Description* || *Solution Idea*)

```
class Solution {
    public List<List<Integer>> palindromePairs(String[] words) {
        Map<String, Integer> wmap = new HashMap<>();
        List<List<Integer>> ans = new ArrayList<>();
        for (int i = 0; i < words.length; i++)
            wmap.put(words[i], i);
        for (int i = 0; i < words.length; i++) {
            if (words[i].equals("")) {
                for (int j = 0; j < words.length; j++) {
                    String w = words[j];
                    if (isPal(w, 0, w.length()-1) && j != i) {
                        ans.add(List.of(i, j));
                        ans.add(List.of(j, i));
                    }
                }
                continue;
            }
```

```
            }
            StringBuilder sb = new StringBuilder(words[i]);
            sb.reverse();
            String bw = sb.toString();
            if (wmap.containsKey(bw)) {
                int res = wmap.get(bw);
                if (res != i) ans.add(List.of(i, res));
            }
            for (int j = 1; j < bw.length(); j++) {
                if (isPal(bw, 0, j-1)) {
                    String s = bw.substring(j);
                    if (wmap.containsKey(s))
                        ans.add(List.of(i, wmap.get(s)));
                }
                if (isPal(bw, j, bw.length()-1)) {
                    String s = bw.substring(0,j);
                    if (wmap.containsKey(s))
                        ans.add(List.of(wmap.get(s), i));
                }
            }
        }
        return ans;
    }

    private boolean isPal(String word, int i, int j) {
        while (i < j)
            if (word.charAt(i++) != word.charAt(j--)) return false;
        return true;
    }
}
```

C++ Code:

(*Jump to*: *Problem Description* || *Solution Idea*)

```
class Solution {
public:
    vector<vector<int>> palindromePairs(vector<string>& words) {
        unordered_map<string, int> wmap;
        vector<vector<int>> ans;
        for (int i = 0; i < words.size(); i++)
```

```
            wmap[words[i]] = i;
        for (int i = 0; i < words.size(); i++) {
            if (words[i] == "") {
                for (int j = 0; j < words.size(); j++) {
                    string& w = words[j];
                    if (isPal(w, 0, w.size()-1) && j != i) {
                        ans.push_back(vector<int> {i, j});
                        ans.push_back(vector<int> {j, i});
                    }
                }
                continue;
            }
            string bw = words[i];
            reverse(bw.begin(), bw.end());
            if (wmap.find(bw) != wmap.end()) {
                int res = wmap[bw];
                if (res != i) ans.push_back(vector<int> {i, res});
            }
            for (int j = 1; j < bw.size(); j++) {
                if (isPal(bw, 0, j-1)) {
                    string s = bw.substr(j, bw.size()-j);
                    if (wmap.find(s) != wmap.end())
                        ans.push_back(vector<int> {i, wmap[s]});
                }
                if (isPal(bw, j, bw.size()-1)) {
                    string s = bw.substr(0, j);
                    if (wmap.find(s) != wmap.end())
                        ans.push_back(vector<int> {wmap[s], i});
                }
            }
        }
        return ans;
    }

private:
    bool isPal(string& word, int i, int j) {
        while (i < j)
            if (word[i++] != word[j--]) return false;
        return true;
    }
};
```

Leetcode Solutions (161 Part Series)

1	Solution: Next Permutation
2	Solution: Trim a Binary Search Tree
...	**157 more parts...**
160	Solution: Out of Boundary Paths
161	Solution: Redundant Connection

Discussion (0)

Code of Conduct • Report abuse

seanpgallivan

Fledgling software developer; the struggle is a Rational Approximation.

LOCATION
Seattle, WA, USA

EDUCATION
Flatiron School (Software Engineering)

WORK
Full Stack Software Engineer

JOINED
Dec 16, 2019

More from seanpgallivan

Solution: Redundant Connection

#algorithms #javascript #java #python

Solution: Out of Boundary Paths

#algorithms #javascript #java #python

Solution: Pascal's Triangle

#algorithms #javascript #java #python