题目：

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

Example 1:

Given `words` = `["bat", "tab", "cat"]`

Return `[[0, 1], [1, 0]]`

The palindromes are `["battab", "tabbat"]`

Example 2:

Given `words` = `["abcd", "dcba", "lls", "s", "sssll"]`

Return `[[0, 1], [1, 0], [3, 2], [2, 4]]`

The palindromes are `["dcbaabcd", "abcddcba", "slls", "llssssll"]`

Credits:

Special thanks to @dietpepsi for adding this problem and creating all test cases.

Subscribe to see which companies asked this question.

1.时间：O（N^2）;空间：O（1） ->超时

```
class Solution {
public:
    vector<vector<int>> palindromePairs(vector<string>& words) {
        if (words.empty()) return std::vector<std::vector<int>>();
        std::vector<std::vector<int>> result;
        for (int i = 0; i < words.size(); ++i){
            for (int k = 0; k < words.size(); ++k){
                if (k != i && isPalindrome(words[i]+words[k])){
                    result.push_back(std::vector < int > {i, k});
                }
            }
        }
        return result;
    }
private:
    inline bool isPalindrome(const std::string& str){
        const int size = str.size();
        for (int i = 0; i < size / 2; ++i){
            if (str[i] != str[str.size() - 1 - i]) return false;
        }
        return true;
    }
};
```

2.时间：O（N^2）;空间：O（N）　　->超时

```
class Solution {
public:
    vector<vector<int>> palindromePairs(vector<string>& words) {
        if (words.empty()) return std::vector<std::vector<int>>();
        std::vector<std::vector<int>> result;
        std::unordered_set<std::string> hashTable;
        for (int i = 0; i < words.size(); ++i){
            for (int k = 0; k < words.size(); ++k){
                if (i == k) continue;
                const std::string concatenation = words[i] + words[k];
                if      (hashTable.find(concatenation)      !=      hashTable.end()      ||
isPalindrome(concatenation)){
                    result.push_back(std::vector < int > {i, k});
                }
            }
        }
        return result;
    }
private:
    inline bool isPalindrome(const std::string& str){
        const int size = str.size();
        for (int i = 0; i < size / 2; ++i){
            if (str[i] != str[str.size() - 1 - i]) return false;
        }
        return true;
    }
};
```

3.时间：( ); 空间：O（N）

```
/*思路：

1.对于词典中 x，y

xright = x[:j]

xleft = x[j:]

即  x = xright | xleft

则 x 和 y 能形成回文的就 2 种情况：

                              1、xright | xleft | y

                              2、  y | xright | xleft

对 1：若 xright.reverse == y  且 xleft == xleft.reverse，那么 x+y 就是回文  对 2 同理


hashtable 将 string 当键值，这样查找  y 的时候会快很多.

2.当 x+y 和 y+x 都为全字符串的时候，会重复，设定规则，在全字符串时，仅 xright | xleft
| y

*/

class Solution {

public:

    vector<vector<int>> palindromePairs(vector<string>& words) {

        if (words.empty()) return std::vector<std::vector<int>>();

        std::unordered_map<std::string, int> hashTable;

        for (int i = 0; i < words.size(); ++i){

            hashTable[words[i]] = i;

        }
        std::vector<std::vector<int>> result;
        for (int i = 0; i < words.size(); ++i){
            for (int k = 0; k <= words[i].size(); ++k){ /* k <= words[i].size() : 数据例
子 : ["a", ""] */
                std::string left_str = words[i].substr(0, k);
                std::string right_str = words[i].substr(k);
                if (isPalindrome(left_str)){ /* x + y */
                    std::string str = right_str;
                    std::reverse(str.begin(), str.end());
                    if (hashTable.find(str) != hashTable.end() && hashTable[str] !=
i)      /* 不可以是同一个字符串 */
                        result.push_back(std::vector < int > {hashTable[str], i});
                }
                /* y + x */
                if (k != words[i].size() && isPalindrome(right_str)){    /*   k   !=
words[i].size()防止重复 */
                    std::string str = left_str;
                    std::reverse(str.begin(), str.end());
                    if (hashTable.find(str) != hashTable.end() && hashTable[str] !=
i)
                        result.push_back(std::vector < int > {i, hashTable[str]});
                }
            }
        }
        return result;
    }
private:
    inline bool isPalindrome(const std::string& str){
        const int size = str.size();
        for (int i = 0; i < size / 2; ++i){
            if (str[i] != str[str.size() - 1 - i]) return false;
        }
        return true;
    }
};
```