336. Palindrome Pairs

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:

```
Input: ["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: ["bat","tab","cat"]Output: [[0,1],[1,0]]
Explanation: The palindromes are ["battab","tabbat"]
```

Intution:

#Naive Approach

check concatenation of every two words and see if it forms a pallindrome

```
for(int i=0;i<n;i++)
    for(int j=0;j<n;j++){
        if(i != j)
        {
        if(isPallindrome(words[i] + words[j]){
        cout<<"i : "<<i<<" | j : "<<j;
        }
    }
}
```

Naive approach has time complexity O(n*n* k) //k is the length of longest word

#Trie - type approach

1. Make a map containing *m[reversed words] = index of the word*
2. Edge case : if we have a null string ("") , then check for pallindromes in words array because combination of "" + pallindrome words will give a pallindrome , also pallindrome words + ""
3. Main case : traverse all words(for every word) :

- split it in left + right = word
- check : if left is present in map and right is a pallindrome then this pair will form a pallindrome
- also check : if right is present in map and left is a pallindrome then this pair will form a pallindrome

Time complexity : O(n*k*k) //since k is generally small, but can be very large : n*n*k >>> n*k*k

#TALK IS CHEAP, SHOW ME THE CODE

```
bool isPall(string a)
```

```
{
    int n = a.length();
    for(int i=0;i<n/2;i++)
    { if(a[i] != a[n-i-1]) return false;}
    return true;
}
vector<vector<int>> palindromePairs(vector<string>& words) {
 map<string,int> m; //map stores {reversed strings,indices}
 vector<vector<int>> ans;

 for(int i=0;i<words.size();i++){
     string key = words[i];
     reverse(key.begin(),key.end());
     m[key] = i;
 }

 //Edge case : if "" exists find all pairs {"",pall} & {pall,""}
 if(m.find("") != m.end())
     for(int i=0;i<words.size();i++){
       if(words[i] == "")   continue;
      if(isPall(words[i]))  ans.push_back({m[""],i});
     }

//Main case : if left + candidate + right is a pallindrome
//or            right + candidate + left is a pallindrome

 for(int i=0;i<words.size();i++){
     for(int j=0;j<words[i].length();j++){
        string left = words[i].substr(0,j);
        string right = words[i].substr(j,words[i].length() - j);

      if(m.find(left) != m.end() && isPall(right) && m[left]!=i){
          ans.push_back({i,m[left]});
      }

      if(m.find(right) != m.end() && isPall(left) && m[right]!=i){
      ans.push_back({m[right],i});
      }
     }
 }
    return ans;
}
```