1408. String Matching in an Array

Given an array of string `words`, return *all strings in* `words` *that is a* ***substring*** *of another word*. You can return the answer in **any order**.

A **substring** is a contiguous sequence of characters within a string

Example 1:

```
Input: words = ["mass","as","hero","superhero"]
Output: ["as","hero"]
Explanation: "as" is substring of "mass" and "hero" is substring of "superhero".
["hero","as"] is also a valid answer.
```

Example 2:

```
Input: words = ["leetcode","et","code"]
Output: ["et","code"]
Explanation: "et", "code" are substring of "leetcode".
```

Example 3:

```
Input: words = ["blue","green","bu"]
Output: []
Explanation: No string of words is substring of another string.
```

Constraints:

- `1 <= words.length <= 100`
- `1 <= words[i].length <= 30`
- `words[i]` contains only lowercase English letters.
- All the strings of `words` are **unique**.

1408. String Matching in an Array

```
/*
   Brute force: STL find
   Time complexity: O(n^2.m^2)
   Space complexity: O(1)
   n: size of the words's list
   m: size of a word in the list of words
*/
class Solution {
   public:
      std::vector<std::string> stringMatching(std::vector<std::string>& words){
         int n=words.size();
         std::vector<std::string> ans;

         // For each words[i] as a needle:
         for(int i=0;i<n;++i){
            for(int j=0;j<n;++j){
               // take words[j] as a haystack
               // if same string ignore it
               if(i==j) continue;

               // If the the needle words[i] math in the haystack words[j]
               // add the needle words[i] to the answer
               if(words[j].find(words[i])!=string::npos){
                  ans.push_back(words[i]);
                  break; // ans must contains unique words
               }
            }
         }
         return ans;
      }
};
```

1408. String Matching in an Array

```
/*
   Brute force: customized function
   Time complexity: O(n^2.m^2)
   Space complexity: O(1)
   n: size of the words's list
   m: size of a word in the list of words
*/
class Solution {
   public:
      // Function to check if a needle match is a haystack using a naive approach
      // Time complexity: O(n.m)
      // Space compelxity: O(1)
      bool is_match(std::string& haystack,std::string& needle){
         int n=haystack.size();
         int m=needle.size();
         for(int i=0;i<n;++i){
            bool match=true;
            int j=0;
            while(j<m && i+j<n && haystack[i+j]==needle[j]) j++;
            if(j==m) return true;
         }
         return false;
      }
```

```
  std::vector<std::string> stringMatching(std::vector<std::string>& words){
        int n=words.size();
        std::vector<std::string> ans;

        // For each words[i] as a needle:
        for(int i=0;i<n;++i){
           for(int j=0;j<n;++j){
              // take words[j] as a haystack
              // if same string ignore it
              if(i==j) continue;

              // If the the needle words[i] math in the haystack words[j]
              // add the needle words[i] to the answer
              if(is_match(words[j],words[i])){
                 ans.push_back(words[i]);
                 break; // ans must contains unique words
              }
           }
        }

        return ans;
     }
};
```

1408. String Matching in an Array

```
/*
   KMP
   Time complexity: O(n^2.m)
   Space complexity: O(m)
   n: size of the words's list
   m: size of a word in the list of words
*/
class Solution {
   public:
```

```
 // Function to check if a needle match is a haystack using KMP algorithm
// Time complexity: O(m+n)
// Space complexity: O(m)
bool is_match(std::string& haystack,std::string& needle){
    if(needle=="") return false;

    // Longest prefix suffix
    int n=haystack.size();
    int m=needle.size();
    std::vector<int> lps(m,0);
    int prev_lps_index=0,i=1;
    while(i<m){
        if(needle[prev_lps_index]==needle[i]){
            lps[i]=prev_lps_index+1;
            prev_lps_index++;
            i++;
        }
        else if(prev_lps_index==0){
            lps[i]=0;
            i++;
        }
        else prev_lps_index=lps[prev_lps_index-1];
    }


    i=0; // Pointer for haystack
    int j=0; // Pointer for needle
    while(i<n){
        if(haystack[i]==needle[j]){
            i++;
            j++;
        }
        else if(j==0) i++;
        else j=lps[j-1];

        if(j==m) return true;
    }
    return false;
}
```

```
    std::vector<std::string> stringMatching(std::vector<std::string>& words){
        int n=words.size();
        std::vector<std::string> ans;
        // For each words[i] as a needle:
        for(int i=0;i<n;++i){
            for(int j=0;j<n;++j){
                // take words[j] as a haystack
                // if same string ignore it
                if(i==j) continue;

                // If the the needle words[i] math in the haystack words[j]
                // add the needle words[i] to the answer
                if(is_match(words[j],words[i])){
                    ans.push_back(words[i]);
                    break; // ans must contains unique words
                }
            }
        }

        return ans;
    }
};
```

1408. String Matching in an Array

```
/*
   Rabin Karp (single works fo this problem)
   Time complexity: O(n^2 * m)
   Space complexity: O(1)
   n: size of the words's list
   m: size of a word in the list of words
*/

typedef long long ll;
class Solution {
   private:
      const ll MOD1=1e9+7;
      const ll MOD2=1e9+33;
      const int radix1=26;
      const int radix2=27;
   public:
      ll power(ll a,int b,ll mod){
         ll res=1;
         while (b>0){
            if (b&1) res=(res*a)%mod;
            a=(a*a)%mod;
            b/=2;
         }
         return res%mod;
      }

      ll hash(std::string& s,int m,int radix,bool is_mod1){
         ll mod=is_mod1?MOD1:MOD2;
         ll h=0,factor=1;
         for(int i=m-1;i>=0;--i){
            h+=((s[i]-'a')*factor)%mod;
            factor=(factor*radix)%mod;
         }
         return h%mod;
      }
```

```
bool is_match(std::string& haystack,std::string& needle){
        int n=haystack.size();
        int m=needle.size();

        if(needle=="" || n<m) return false;

        ll max_weight1=power(radix1,m,MOD1);
        //ll max_weight2=power(radix2,m,MOD2);

        ll needle_hash1=hash(needle,m,radix1,true);
        //ll needle_hash2=hash(needle,m,radix2,false);

        ll win_hash1=hash(haystack,m,radix1,true);
        //ll win_hash2=hash(haystack,m,radix2,false);

        for(int i=0;i<=n-m;++i){
            if(win_hash1==needle_hash1) return true;

            win_hash1=((((win_hash1*radix1)%MOD1-((haystack[i]-'a')*max_weight1)%MOD1+haystack[i+m]-'a')%MOD1)+MOD1)%MOD1;
          //win_hash2=((((win_hash2*radix2)%MOD2-((haystack[i]-'a')*max_weight2)%MOD2+haystack[i+m]-'a')%MOD2)+MOD2)%MOD2;
        }

        return false;

    }
```

```
std::vector<std::string> stringMatching(std::vector<std::string>& words){
        int n=words.size();
        std::vector<std::string> ans;
        for(int i=0;i<n;++i){
            for(int j=0;j<n;++j){
                if(i==j) continue;
                if(is_match(words[j],words[i])){
                    ans.push_back(words[i]);
                    break;
                }
            }
        }

        return ans;
    }
};
```