All Subsets I (Medium)

Given a set of characters represented by a String, return a list containing all subsets of the characters.

Assumptions

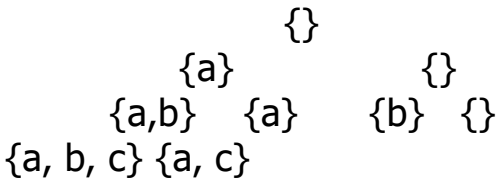
- There are no duplicate characters in the original set.

Examples

- Set = "abc", all the subsets are [“”, “a”, “ab”, “abc”, “ac”, “b”, “bc”, “c”]
- Set = "", all the subsets are [""]
- Set = null, all the subsets are []

```
void FindSubSet(const string& s, int index, string& result, vector<string>&
results){
    if(index == s.length()){
        if(result.size() == 0)
            results.push_back("");
        else
            results.push_back(result);
        return;
    }
    result.push_back(s[index]);
    FindSubSet(s, index+1, result, results);
    result.pop_back();
    FindSubSet(s, index+1, result, results);
}
```

Time Complexity: O(2^1 + … + 2^n) = O(n * 2^n), where n is the length of s.
//The solution is right but the time complexity is O(2^n). It is the same with the result size.

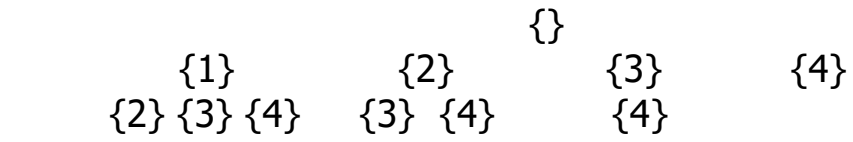


Combinations (Medium)

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

E.g. Input: n = 4, k = 2
Output: [
[2,4],
[3,4],
[2,3],
[1,2],
[1,3],
[1,4]
]

```
void combinations(int n, int k, int index, vector<int>& result, vector<
vector<int> >& results){
    if(result.size() == k){
        results.push_back(result);
        return;
    }
    for(int i = index; i < n; i++){
        result.push_back(i+1);
        combinations(n, k, i+1, result, results);
        result.pop_back();
    }
}
```



//Ok, great!
//Time complexity: O(C(n,k))

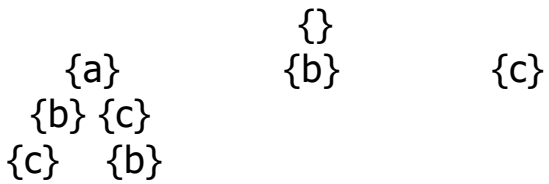
All Permutations I (Medium)

Given a string with no duplicate characters, return a list with all permutations of the characters.

Assume that input string is not null.

Examples
Set = “abc”, all permutations are [“abc”, “acb”, “bac”, “bca”, “cab”, “cba”]
Set = "", all permutations are [“”]

```
void permutations(const string& s, int level, vector<int>& visited, string&
result, vector<string>& results){
    if(level == s.size())
        results.push_back(result);
    for(int i = 0; i < s.length(); i++){
        if(visited[i] == 1)
            continue;
        else
            visited[i] = 1;
        result.push_back(s[i]);
        permutations(s, level+1, visited, result, results);
        result.pop_back();
        visited[i] = 0;
    }
}
```



//Ok, no problem! There is a function in C++ STL named next_permutation and you can study it!

All Subsets II (Hard)

Given a set of characters represented by a String, return a list containing all subsets of the characters. Notice that each subset returned will be sorted to remove the sequence.

Assumptions

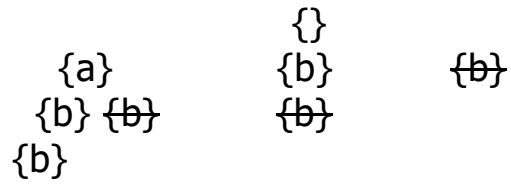
- There could be duplicate characters in the original set.

Examples

- Set = "abc", all the subsets are ["", "a", "ab", "abc", "ac", "b", "bc", "c"]
- Set = "abb", all the subsets are ["", "a", "ab", "abb", "b", "bb"]
- Set = "abab", all the subsets are ["", "a", "aa","aab", "aabb", "ab","abb","b", "bb"]
-
- Set = "", all the subsets are [""]
- Set = null, all the subsets are []

```
// sort string first
void stringSort(string& s){
    sort(s.begin(), s.end());
}
void FindSubSet2(const string& s, int index, string& result, vector<string>&
results){
    results.push_back(result);
    for(int i = index; i < s.size(); i++){
        // add a condition here
        if(i != index && s[i] == s[i-1])
            continue;
        result.push_back(s[i]);
        FindSubSet2(s, i+1, result, results);
        result.pop_back();
    }
}
```

e.g. “abb”



//Ok, pretty good!

All Permutations II(Hard)

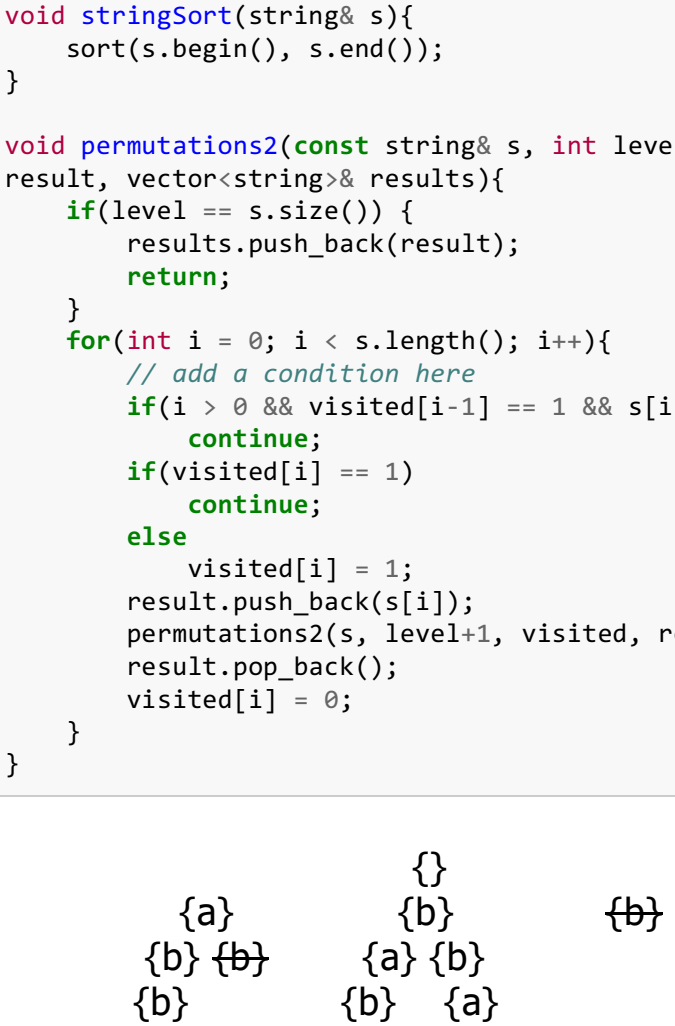
Given a string with possible duplicate characters, return a list with all permutations of the characters.

Examples

- Set = “abc”, all permutations are [“abc”, “acb”, “bac”, “bca”, “cab”, “cba”]
- Set = "aba", all permutations are ["aab", "aba", "baa"]
- Set = "", all permutations are [""]
- Set = null, all permutations are []

```
// sort string first
void stringSort(string& s){
    sort(s.begin(), s.end());
}

void permutations2(const string& s, int level, vector<int>& visited, string&
result, vector<string>& results){
    if(level == s.size()) {
        results.push_back(result);
        return;
    }
    for(int i = 0; i < s.length(); i++){
        // add a condition here
        if(i > 0 && visited[i-1] == 1 && s[i-1] == s[i])
            continue;
        if(visited[i] == 1)
            continue;
        else
            visited[i] = 1;
        result.push_back(s[i]);
        permutations2(s, level+1, visited, result, results);
        result.pop_back();
        visited[i] = 0;
    }
}
```



//Ok, no problem!

*I have some trouble when computing time complexity of DFS in Homework 7.

//可以这样考虑，对于内部没有循环的 DFS，其结果取决于 result 的 size 的大小，那么每个位置就是选择或者不选择，N 个位置，每个位置 2 种方法，乘法原理，就是 O(2^N)。
//对于内部有循环的 DFS，每个位置两种选择的，直接乘上 N 即可，就是(N*2^N)。
对于没有两种选择的，而是第一个位置选了 N，第二个位置只有 N-1 可选，依次 N-2....1，那么显然乘法原理直接相乘，就是 O(N!)。
//均是根据数学来进行计算，因为排列组合本质是数学问题

//Any other question still, please submit next homework! Thanks!