# 判断子序列

> 给定字符串s和t,判断s是否为t的子串。

分析：

1. 可以采用双指针法，增加unordered_map来判断字符串是否已经存在。

```
bool isSubsequence(string s, string t) {
  int i =0,j=0;
  while(i < s.size() && j < t.size()) {
    if(s[i] == t[j]) {
      i++;
    }
    j++;
  }

  return i == s.size();
}
```

进阶

> 如果给定一组字符串s，和一个字符串t，如何判断s中的每个字符串是否是t的子串。

依照上述思路求解：

```
std::vector<bool> isSubsequence(std::vector<string> s, string t){
  std::vector<bool> sub;
  for(auto item : s) {
    sbd.push_back(isSubsequence(ite, s));
  }

  return sub;
}


bool isSubsequence(string s, string t){
  int i = 0, j = 0;
  while(i < s.size() && j < t.size()){
    if(s[i] == t[j]) {
      i++;
    }
    j++;
  }
```

```
  return i == s.size();
}
```

二分查找优化： 算法思路：

1. 构造一个map存储了字符串t中所有字符的索引位置；

```
#include <algorithm>
#include <string>
#include <unordered_map>
#include <vector>

bool isSubsequence(std::string s, std::string t) {
  int m = s.size(), n = t.size();

  std::unordered_map<int, std::vector<int>> map;
  for (int i = 0; i < n; i++) {
    map[t[i]].push_back(i);
  }

  int j = 0;  // t索引位置
  for (int i = 0; i < m; i++) {
    char c = s[i];
    // 字符c不存在
    if (map.count(c) == 0) {
      return false;
    }

    std::vector<int>::iterator pos =
        std::lower_bound(map[c].begin(), map[c].end(), j);
    if (pos == map[c].end()) {
      return false;
    }

    j = *pos;
  }

  return true;
}
```