205. Isomorphic Strings

Tags	
Property	@August 16, 2022

Question

原文：

Given two strings `s` and `t`, *determine if they are isomorphic*.

Two strings `s` and `t` are isomorphic if the characters in `s` can be replaced to get `t`.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

我的理解：

給定兩個string，s和t，s中的所有字母轉換為某個相對應的字母就可以將s轉換成t，則return true，不可以則return false，例如：s=egg t=add ⇒true，s=foo t=bar ⇒false。不會有兩個字母轉化為同樣字母，但字母可以轉化成他們本身

翻譯：

给出两个字符串s和t，确定它们是否同构。

如果两个字符串s和t可以被替换成t，那么这两个字符串就是同构的。

一个字符的所有出现都必须用另一个字符替换，同时保持字符的顺序。没有两个字符可以映射到同一个字符，但一个字符可以映射到它自己。

自評翻譯正確性：99%

- Word Memory：
  - isomorphic 同構
  - occurrences 出現

Code//純自己寫的ver 測資通過43/43 （但因Memory Limit Exceeded 未通過 下有通過的版本）

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        int i,p,temp;
        int n=s.length();//檢查長度是否相同
        int k=t.length();
        if(n!=k){
            return false;
        }
        string a;
          vector<char>pass;
```

```
            for(i=0;i<s.size();i++){//先把 st 合併成 a
                a=a+s[i]+t[i];
            }
            pass.push_back(a[0]);//讓 pass 裡有東西可以識別
            pass.push_back(a[1]);
            for(temp=0;temp<a.size()-1;temp=temp+2){//依序檢查 a 用 pass 來檢查
                for(p=0;p<pass.size();p=p+2){
                    if(((a[temp]==pass[p])&&(a[temp+1]!=pass[p+1]))|((a[temp]!=pass[p])&&(a[temp+1]==pass[p+1]))){
                        return 0;//出現跟之前組合不同的情況 return false
                    }
                    if((a[temp]==pass[p])&&(a[temp+1]==pass[p+1])){//有找到相同組合 break 繼續下一位
                        break;
                    }
                    if(p==pass.size()-2){//出現新組合放進 pass
                        pass.push_back(a[temp]);
                        pass.push_back(a[temp+1]);
                    }
                }
            }
            return 1;
        }
};
```

> 思路：先把 st 合再一起產生 a，後面的檢查與搜尋就字符兩兩一組一起看，然後把 pass 當作字典，如果出現根字典紀錄不相符的組合就判斷是 false，如果字典已存在的組合完全相同就檢查下一組，如果掃完字典也沒有看到與自身相同或不相符的情況就這個組合加入字典尾端。（我覺得想法很讚，可惜記憶體爆炸ＱＱ）

Code（Accepted 版本 有用到 unordered map 該函式概念是利用hash table）

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        unordered_map<char,int>m1;
        unordered_map<char,int>m2;
        int i;
        for(i=0;i<s.size();i++){
            m1[s[i]]=i;
            m2[t[i]]=i;
        }
        for(i=0;i<s.size();i++){
            if(m1[s[i]]!=m2[t[i]]){
                return false;
            }
        }
        return 1;
    }
};
```

> 思路：宣告unordered map時可以宣告兩個型別，這邊前面的型別可以用來被索引，所以在索引的時候就不一定要用正整數，在這題直接用字符當作索引建立m1、m2，同時映射的值再擺放出現的的位置，如果在下方比對時兩個字符最後出現的位置不同，則表示這個組合與之前出現的組合不同，故返回 false，如果檢查完之後都相同則返回true （unordered map詳細點的解釋在C++函式）

優良code參考

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        int a[128] = {0}, b[128]={0};
        for(int i=0; i<s.size() ; i++){
          if(a[s[i]] != b[t[i]] ) return false;
          a[s[i]] = b[t[i]] = i+1;
        }
        return true;
    }
};
```

> 思路：為s跟t各開一個128格的全為0陣列ab，因為一個char最多就是紀錄128種不同的字符，ab陣列用來記錄特定字符上次出現的位置，例如 第一組進來的 s t配對為 ‘q’ ’d’，那因為是第一次出現就把 ‘q’ ’d’目前的位址i輸入在a、b這128格裡各自對應的地方，a⇒’q’（char可以轉成數字ascii，故能在128格中找到對應位置）、b⇒’d’，下次出現s⇒’q’ t⇒'d'就可以通過上次出現的位址一樣來識別正確，並且更新出現的位址，且等到下一次出現s⇒’q’或t⇒'d'搭配上另外一個不是原組合的字符，它們上次紀錄的i值就會不同，以此來判定不是原搭配 return false

優良code參考

```

```

> 思路：