搜索-双向广搜

一、AcWing 190. 字串变换

【题目描述】

已知有两个字串A, B及一组字串变换的规则（至多6个规则）：

$A_1 \to B_1$

$A_2 \to B_2$

$\dots$

规则的含义为：在A中的子串A_1可以变换为B_1、A_2可以变换为$B_2 \dots$

例如：$A =$ `abcd`，$B =$ `xyz`。

变换规则为：

`abc` $\to$ `xu`、`ud` $\to$ `y`、`y` $\to$ `yz`。

则此时，A可以经过一系列的变换变为B，其变换的过程为：

`abcd` $\to$ `xud` $\to$ `xy` $\to$ `xyz`

共进行了三次变换，使得A变换为B。

【输入格式】

输入格式如下：

第一行是两个给定的字符串A和B。

接下来若干行，每行描述一组字串变换的规则。

所有字符串长度的上限为20。

【输出格式】

若在10步（包含10步）以内能将A变换为B ，则输出最少的变换步数；否则输出`NO ANSWER!`。

【输入样例】

```
abcd xyz
abc xu
ud y
y yz
```

【输出样例】

```
3
```

【分析】

假设每次决策数量是K，那么如果直接BFS，最坏情况下的搜索空间是K^{10}，非常大，所以会TLE或者MLE。

如果采用双向BFS，则可以把搜索空间降到$2K^5$。在实际测试中只需$20ms$左右，剪枝效果很好。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#include <queue>
#include <unordered_map>
using namespace std;

const int N = 10;
string A, B;
string a[N], b[N];
unordered_map<string, int> disA, disB;
int n;

int bfs()
{
    if (A == B) return 0;
    disA[A] = 0, disB[B] = 0;
    queue<string> QA, QB;
    QA.push(A), QB.push(B);
    int step = 0;//两端扩展的总步数
    while (QA.size() && QB.size())
    {
```

```
        if (QA.size() <= QB.size())//如果QA中的状态数更少那么从QA扩展
        {
            int dis = disA[QA.front()];//将QA中步数相同的状态全部扩展一步
            while (QA.size() && disA[QA.front()] == dis)
            {
                auto t = QA.front();
                QA.pop();
                for (int i = 0; i < n; i++)//遍历每种变换规则
                    for (int j = 0; j < t.size(); j++)
                        if (t.substr(j, a[i].size()) == a[i])
                        {
                            string str = t.substr(0, j) + b[i] + t.substr(j + a[i].size());
                            if (disB.count(str)) return disA[t] + disB[str] + 1;
                            if (!disA.count(str)) QA.push(str), disA[str] = disA[t] + 1;
                        }
            }
        }
        else//反之从QB扩展，以下写法同上
        {
            int dis = disB[QB.front()];
            while (QB.size() && disB[QB.front()] == dis)
            {
                auto t = QB.front();
                QB.pop();
                for (int i = 0; i < n; i++)
                    for (int j = 0; j < t.size(); j++)
                        if (t.substr(j, b[i].size()) == b[i])
                        {
                            string str = t.substr(0, j) + a[i] + t.substr(j + b[i].size());
                            if (disA.count(str)) return disB[t] + disA[str] + 1;
                            if (!disB.count(str)) QB.push(str), disB[str] = disB[t] + 1;
                        }
            }
        }
        if (++step == 10) return -1;//总共扩展了10次还没相遇返回-1
    }

    return -1;//有一端已经没有可扩展的状态则返回-1
```

```cpp
}

int main()
{
    cin >> A >> B;
    while (cin >> a[n] >> b[n]) n++;
    int res = bfs();
    if (~res) cout << res << endl;
    else puts("NO ANSWER!");
    return 0;
}
```