

Scramble String

→ (a) (b)

Scramble String

T/P

boolean

Problem statement

- How to identify
- How to approach
- How to breakdown
- Base Cond
- Code

Given two strings S_1 and S_2 of equal lengths the task is to determine if S_2 is a scrambled form of S_1 .

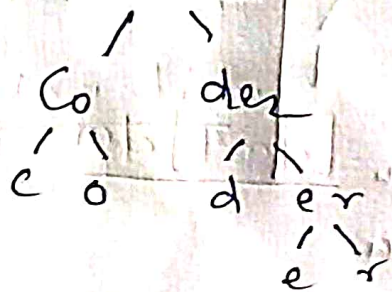
Scrambled string :-

Given string str, we can represent it as a binary tree by partitioning it to two non-empty substrings recursively :

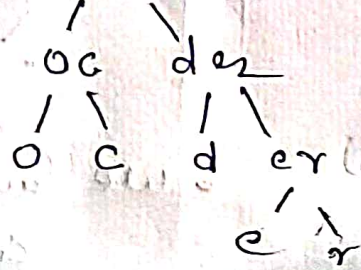
Scramble string is not same as an Anagram

Example

str1: Codes



str2: Ocder



Thus 'ocder' is a scrambled string of 'coder'.
 Similarly if we continue to swap the children

of nodes 'der' and 'er' it produces a

scrambled string 'dcored'.

O c r e d

O c r e d

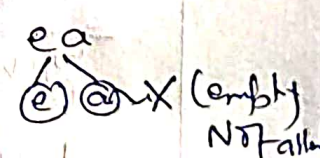
O c r e d

O c r e d

Thus 'dcored' is a scrambled string of 'coder'.

[S1: coder, S2: dcored, Output: true]

1. Binary tree: only 2 child allowed

2. No child should be empty or null.  (empty NOT allowed)

great

gr eat

g r e at

g r e at

g r e at

g r e at

g r e at

great

gr eat

g r e at

g r e at

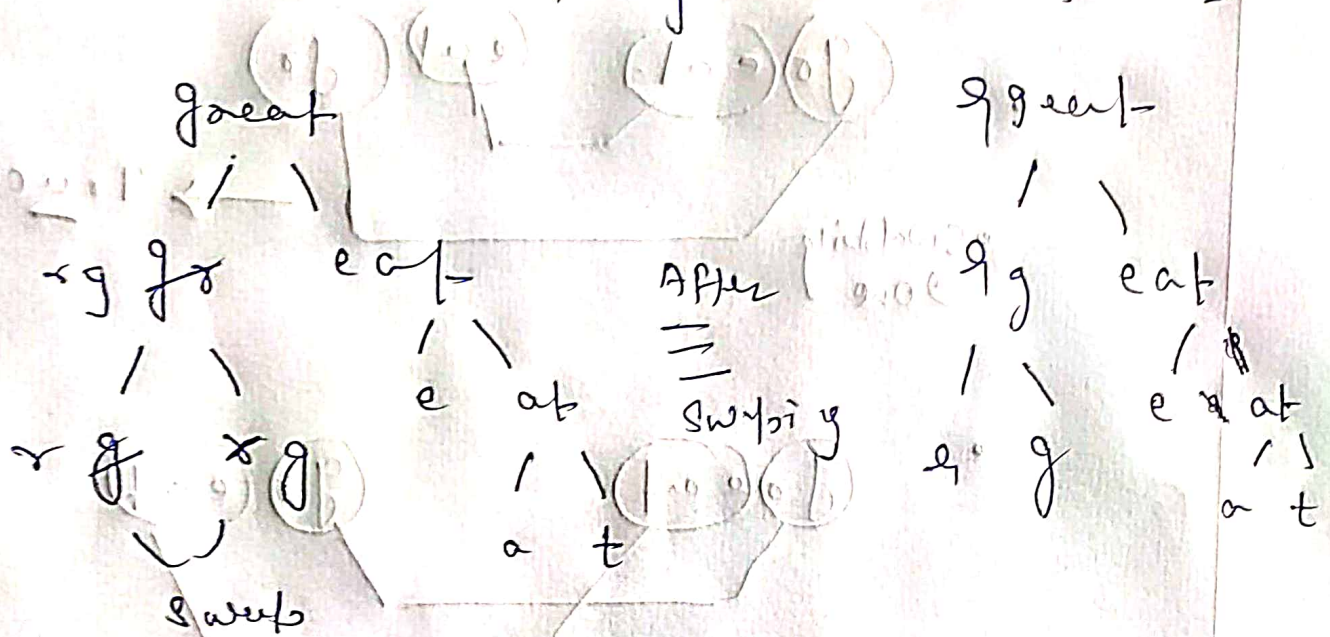
g r e at

g r e at

g r e at

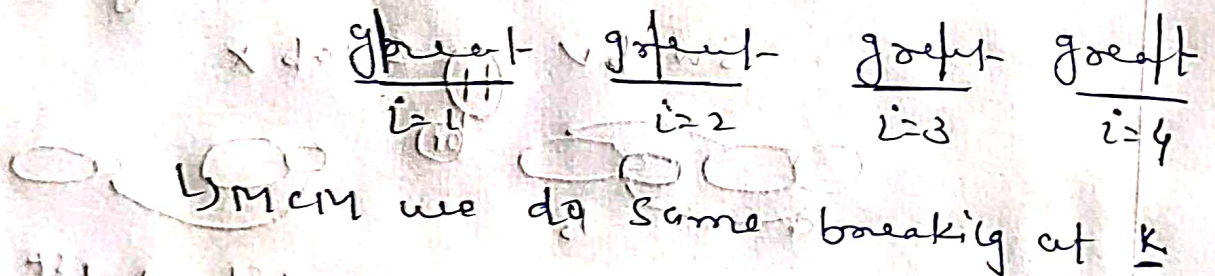
• we can break on any pattern

- we can do swapping in non leaf node



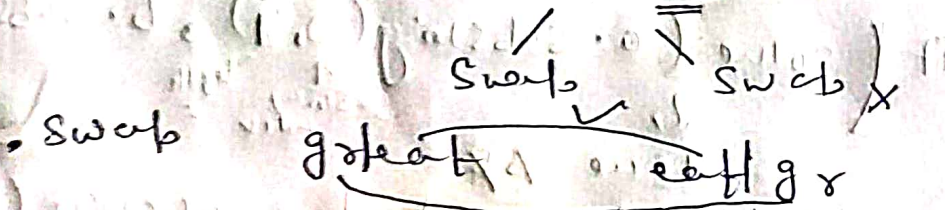
- zero or more swapping can be done.
So [great] \rightarrow scrambled string

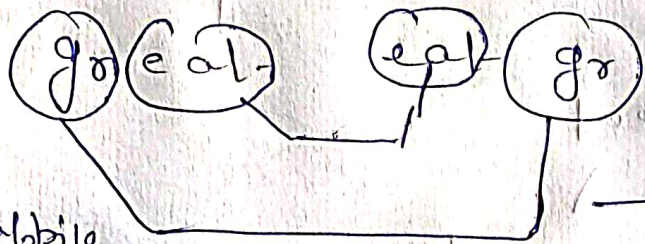
- we can break at $i=1$ to $i=n-1$.



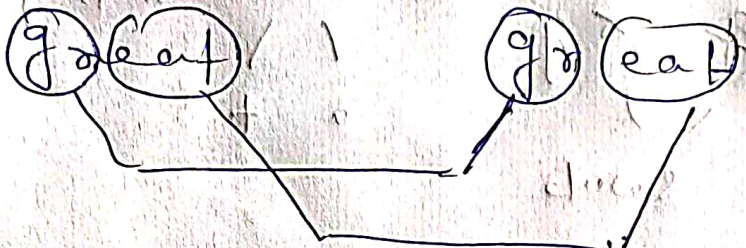
- a: great b: great \Rightarrow 0 swap, True
- a: great b: eat gr \Rightarrow 1 swap, True

- Scramble string can be swapped or it can not be swapped.





True



No swapping

Scramble

at any i

Swab ✓

Swab x



a → b

1st → 1st

a → b

a → b

Case 1: goeat

eat

gr solve(a, b)

Swapping ✓

swab

if (solve(a.substring(0, i), b.substring(n-i, n)) || solve(a.substring(i, n-i), b.substring(0, n-i)))

return True

Cond 1

if (solve(a.substr(0, i), b.substr(0, i)) &&
solve(a.substr(i, n-i), b.substr(i, n-i)))

Case 1

swaplog

if (solve(smaller a, smaller b) == true &&
solve(smaller b, smaller a) == true)

Case II

if (a[i] == b[i])

if (a[i+1] == b[i+1])

i++

No swapping

Cond 2

2nd Cond

if (solve(smaller a, smaller b) == true &&
solve(smaller b, smaller a) == true)

No swapping

if (solve(a.substr(0, i-1), b.substr(0, i-1)) == true &&
solve(a.substr(i, n-i), b.substr(i, n-i)) == true)

if (Cond I || Cond II)
return "Scalable String"

Base Condition

Ans: $\text{grout} = (1.0) \left(\frac{1.0 \times 10^{-3} \text{ m}^3}{\text{m}^2} \right) \times 1000 \left(\frac{\text{kg}}{\text{m}^3} \right) = 1.0 \text{ kg}$

10: 89 seat } think of smallest valid input
11: 100 } (100, 100, 100, 100)

$$\neg (a.\text{length} \neq b.\text{length})$$

return false

$$T(\emptyset, \emptyset) = 1$$

Refin ~~false~~ true.

77 $(a \cdot \text{Compare}(b) == 0) \models \text{Both strings equal}$

Return time: 110 swab

11 Both string is equal

if (a.length() < 2) // check for either of
(return false;) the string (either a
or b)

11 Same length strings with
12 Compared

if $\text{main}() \rightarrow (1, 0) \rightarrow (2, 0) \rightarrow (0, 0)$

{ string a
string b

- G. length — 10 length \rightarrow different \rightarrow false
- Both Empty \rightarrow true

Solve (a, b):


```
bool solve (String a, String b)
```

```
{
```

```
if (a.compare(b) == 0) // Both string is equal  
    return true;
```

```
if (a.length() < 1) // either of string is  
    return false; // not equal length
```

```
int n = a.length(); // take length of any  
// one string
```

```
bool flag = false;
```

```
for (int i = 1; i < n - 1; i++)
```

```
{ if (Cond 1 || Cond 2)
```

```
    flag = true;
```

```
    break;
```

```
}  
return flag;
```

```
}
```

```
int main()
```

```
{ String a, String b
```

```
• a.length() - b.length() → different → false
```

```
• Both Empty → true
```

```
{ solve(a, b);
```

Base
Cond

Code

Scramble String Top-Down (Memoized)

• we already seen the recursive form of Scramble String.

• The function is called again and again so

the function (which is already called will

be stored in table and so can

avoid overlapping and get solution more

effectively.

• either we can store in table and we can
store in map.

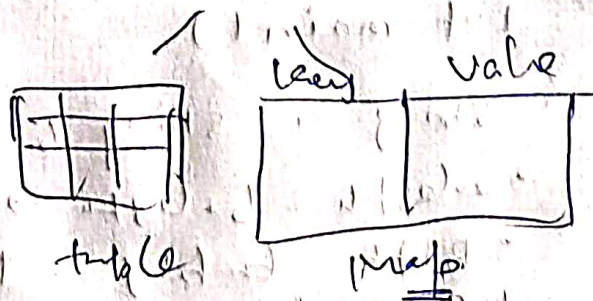
• Check on map for value then call the function

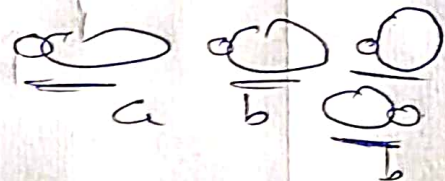
• after calling function if value is not there

on map then store the value in map.

unordered_map < string, bool > mp;

Recursive Call + Cache



- Here we will use changing variable.
- Here a and b is changing variable it is breaking at a point. 
- before recursive call check the value in map.

Code

```
bool solve (String a , String b)
{
    if (a.compare(b) == 0)
        return True
    if (a.length() <= 1)
        return false;
}
```

Base case

```
String key = a;
key.push-back(b);
key.append(b);
if (mp.find(key) != mp.end())
    return mp[key]
```

```
int n = a.length();
bool flag = false
for (int i = 1; i < n; i++)
```



```

{
    if (Cond 1 || Cond 2)

```

```

    {
        flag = True;

```

```

        break;
    }

```

```

    {
        return mp[key] = flag; // store in map
    }
    before returning
    to avoid overlapping

```

Sum-up,

⇒ To avoid overlapping of recursion we use Top down approach.

key	value

- Store value of question call in map and check before calling another function.