



bool solve ( string  $s_1$ , string  $s_2$ , int  $i$ , int  $j$  )

```

  {
    if ((i < 0 and j < 0))
      return true;
    if (i < 0 and j ≥ 0)
      return false;
    if (j < 0 and i ≥ 0)
      return isAllStar(s1, i);
    if (dp[i][j] != -1)
      return dp[i][j];
    if (s1[i] == s2[j] or s1[i] == '?')
      op[i][j] = solve(s1, s2, i-1, j-1);
    else if (s1[i] == '_')
      op[i][j] = solve(s1, s2, i, j-1) or solve(s1, s2, i-1, j);
    else
      op[i][j] = false;
  }
  
```

### Tabulation

$\rightarrow$   $\text{dp}(n+1, \text{vector<b>}(n+1, \text{false}))$

Vector <vector<  
 $\xrightarrow{\text{b001}}$  dp(n+1, vector< $\xrightarrow{\text{b001}}$  (m, false));

$$\begin{array}{l} s_1 = n \\ \hline s_2 = m \end{array}$$

$$dp[0][0] = 1;$$

For ( j = 1; j <= m; j++ )

$$\leftarrow dp[0][j] = \underline{\text{false}};$$

For ( i = 1; i < n; i++ )

$$\leftarrow dp[i][0] = \boxed{\text{isAnSone}}(s_1, i);$$

for ( c = 1; c < n; c++ )

$\leftarrow$  For ( j = 1; j <= m; j++ )

$\leftarrow$  if (  $s_1[c-1] == s_2[j-1]$  or  $s_1[c-1] == '?'$  )

$$\leftarrow dp[i][j] = \boxed{dp[i-1][j-1];}$$

else if (  $s_1[c-1] == 'x'$  )

$$dp[i][j] = \boxed{dp[i-1][j]} \parallel \boxed{dp[i][j-1];}$$

else

$$\leftarrow dp[i][j] = 0$$

return dp[n][m];

$\begin{matrix} a \\ \downarrow \\ a \end{matrix}$

a	b	c	d
0	1	2	3
a	1	0	0
0	0	0	0
0	0	0	0

$\alpha$   $\beta$   $\gamma$

	1	0	0	0	0
	1	0	1	0	0
2	0	1	1	1	1
3	0	0	1	1	1
4	0	0	0	1	1

10 07 → Space optimize

$$a \times ? = ?$$

0 1 1 2 3 5 8 13 -  
20 DP

a b c d

vector<int> prev;

Vector  $c$  in  $\mathbb{R}^n$   $\rightarrow$   $c$

vector<int> test

Swap (Prev, cur)

卷之三

$$\tan i = \frac{v}{w} ;$$

$$\tan v = \frac{w}{z} ;$$

$$\tan r = \frac{z}{v} ;$$

Scramble String

$S_1 = \text{"great"}$

$S_2 = \text{"great")}$

$S = X + Y$

$X = Y$

$Y = X$

at erg

great

great)

at

erg

g:re

re

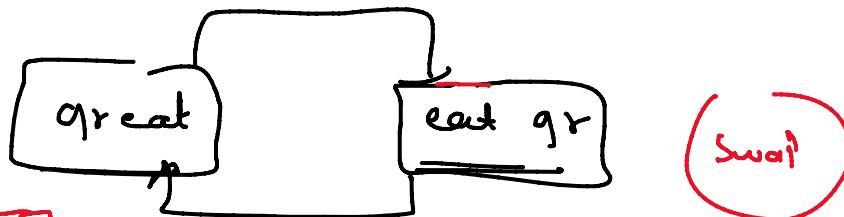
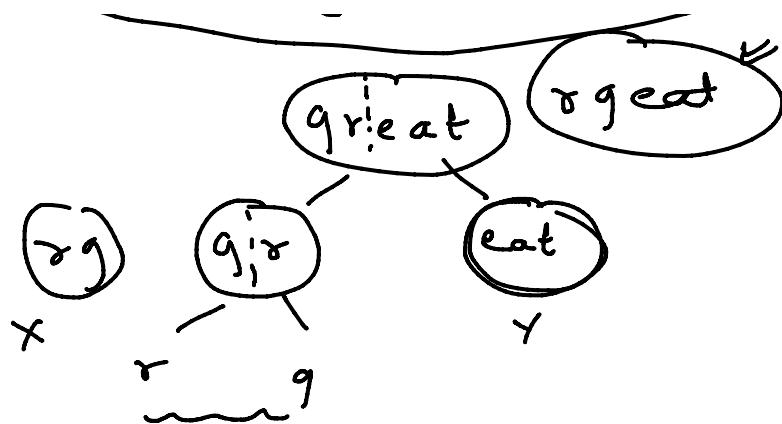
c

at

a

t

great



$s' =$

$s'' =$

isScramble((s<sub>1</sub>, s<sub>2</sub>)) and isScramble((s<sub>3</sub>, s<sub>4</sub>))

$\Rightarrow \neq = g r e a t$

$$x + y = (x) + y$$

$\Rightarrow \neq \neq e a t | g r$

$$y + x = (y) + x = x + y$$

map < pair<string, string>, bool> m;

return [ isScramble(s<sub>1</sub>, s<sub>2</sub>) and isScramble(s<sub>3</sub>, s<sub>4</sub>) ] or

= isScramble(s<sub>1</sub>, s<sub>5</sub>) and isScramble(s<sub>2</sub>, s<sub>6</sub>)

map =

map < string, bool> m;

bool solve(string s<sub>1</sub>, string s<sub>2</sub>)

{ if (s<sub>1</sub>.size() == 1)

a

abc

abc

map.count((s<sub>1</sub>, s<sub>2</sub>))

S   if   ( s1.. size() == 1 )  
S   return   s1 == s2  
n

ma::count(s1, s2)

String  $\underline{s} = s_1 + \dots + s_2$

```

if (s1 == s2)
    return true;
else if (ma.count(str3)) return ma[str3];
int n = s1.size();

```



for (i=1 ; i < n ; i++)  
    *i*=2  
    *S<sub>i</sub>*

31

if  $\{ \underbrace{\text{solve}(s_1 \cdot \text{substr}(0, i))}_{\text{and}}, \underbrace{s_2 \cdot \text{substr}(0, i)}_{\text{and}} \}$   
 $\text{solve}(s_1 \cdot \text{substr}(i), \underbrace{s_2 \cdot \text{substr}(i)}_{\text{or}})$   
 solve  $\{ \underbrace{s_1 \cdot \text{substr}(0, i)}_{\text{and}}, \underbrace{s_2 \cdot \text{substr}(n-i)}_{\text{and}} \}$  and solve  
 $\{ \underbrace{s_1 \cdot \text{substr}(i)}_{\text{and}}, \underbrace{s_2 \cdot \text{substr}(0, n-i)}_{\text{and}} \}$

reben, trull;

$$S_1 \in a \triangleright c d$$

$\Rightarrow$   $S_1 = \text{sub}(S_0(1))$

$$ma(s+3) =$$

mai & S1, s2 3) = true / false

return false

marks

$\alpha \Gamma S \tau_3$

A diagram illustrating a stack-based parser for the sentence "great eat qz". The input words are shown in boxes at the bottom: "great" (S1) and "eat qz" (S2). Above them, a stack contains two frames. The top frame (F1) has "great" (S3) at the top and "eat" (S4) below it. The bottom frame (F2) has "eat" (S5) at the top and "qz" (S6) below it. Red arrows indicate the movement of words from the input to the stack: "great" moves to F1, "eat" moves to S4, and "qz" moves to S6. A red box encloses the word "great" in the top frame.

agreat

a-  
eat  
-er

a-  
eat  
-er

Accepted, *[Signature]*

