# Lecture: DP-5

Agenda:

- LCS
- LPS
- Edit distance
- Wildcard pattern matching

Qu1 LCS [Longest common subsequence]

s1: a b b c d g f

s2: b a c h e g f

ans = 4

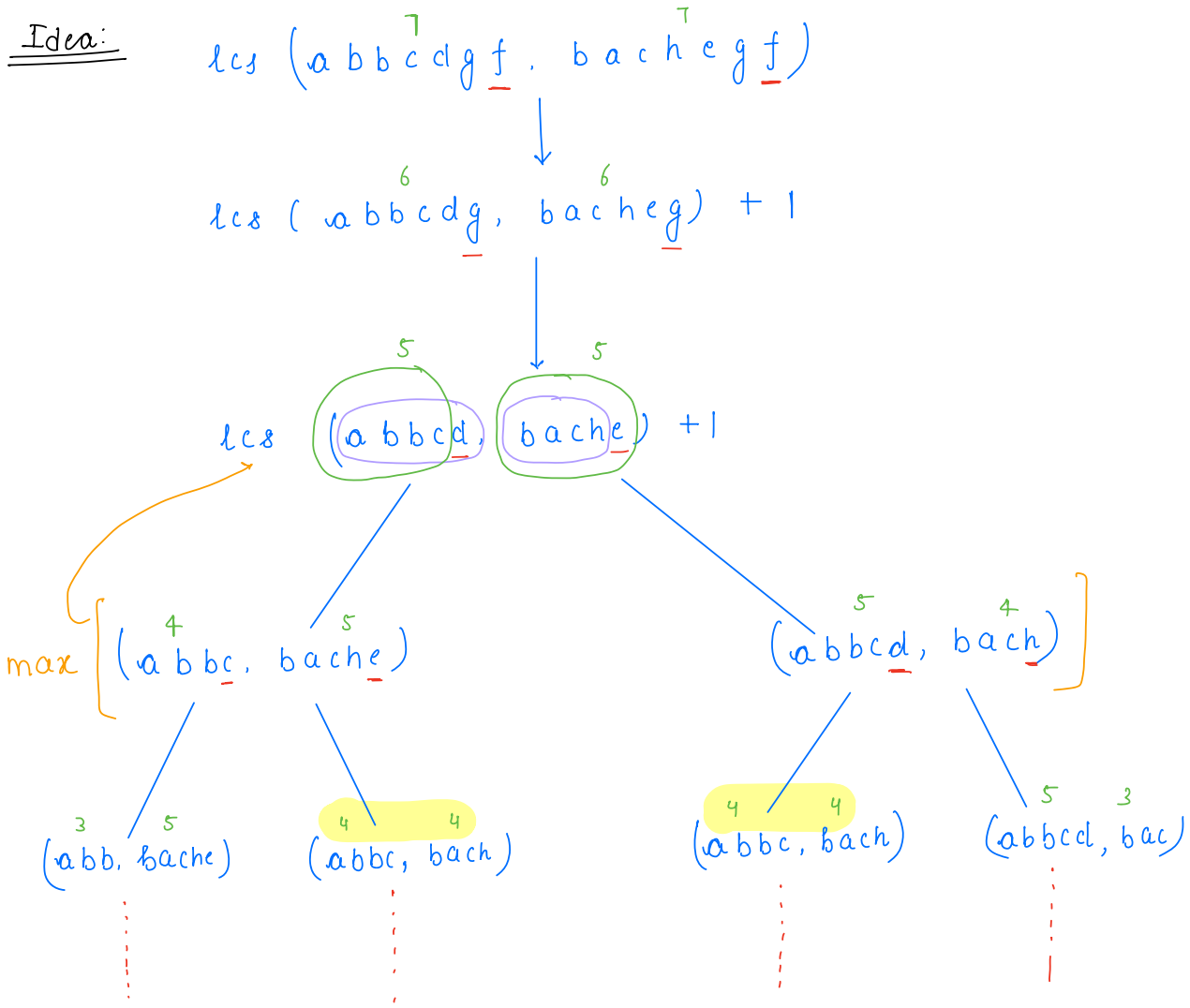
ans: a c g f , b c g f

s1: k l a g r i p

s2: l g i g k m

ans = 3

Idea:




changing factors:

len of s1

len of s2

⟶ dp[ ][ ] ⟶ Overlapping subproblems

n+1 m+1 optimal substructure

## Recursive code:

```
int lcs( s1, s2, n, m) {
        if( n == 0  ||  m == 0) {
                return 0;
        }
        if( s1.charAt(n-1) == s2.charAt(m-1)) {
                return lcs( s1, s2, n-1, m-1) + 1;
        }
        op1 = lcs( s1, s2, n, m-1.);
        op2 = lcs( s1, s2, n-1, m);
        return max( op1, op2);
}
```

s1.size()  s2.size()

Memoised code:

s1.size()  s2.size()

```
int lcs( s1, s2, n, m, dp[][] ) {
    if( n==0 || m==0) {
        return 0;
    }
    if(dp[n][m] != -1) {
        return dp[n][m];
    }
    if( s1.charAt(n-1) == s2.charAt(m-1)) {
        dp[n][m] = lcs( s1, s2, n-1, m-1) + 1;
    } else {
        op1 = lcs( s1, s2, n, m-1.);
        op2 = lcs( s1, s2, n-1, m);
        dp[n][m]=max( op1, op2);
    }
    return dp[n][m];
}
```

dp[i][j] = lcs of s1 ( first i characters ) and
s2 ( " j " )

n+1 m+1

Dry run:

s1 = m a i c a (0 1 2 3 4)

s2 = i a i y a s (0 1 2 3 4 5)

dp[6][7]

| | | i | a | i | y | a | s |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| m | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 2 | 0 | 0 | 1 | 1 | 1 | 1 |
| i | 3 | 0 | 1 | 1 | 2 | 2 | 2 |
| c | 4 | 0 | 1 | 1 | 2 | 2 | 2 |
| a | 5 | 0 | 1 | 2 | 2 | 2 | 3 |

dp[2][1]

s1.charAt(1) != s2.charAt(0)

max ( [1][1] , [2][0] )

dp[2][2]

s1.charAt(1) != s2.charAt(1)

dp[1][1] + 1 = 0 + 1 = 1

dp[3][3]

s1[2] == s2[2]

dp[2][2] + 1

## Tabulative

```
int lcs(s1, s2) {
    n = s1.length();
    m = s2.length();
    dp[n+1][m+1];

    for(i=0; i<=n; i++) {
        for(j=0; j<=m; j++) {
            if(i==0 || j==0) {
                dp[i][j] = 0;
            } else {
                if(s1.charAt(i-1) == s2.charAt(j-1)) {
                    dp[i][j] = 1 + dp[i-1][j-1];
                } else {
                    op1 = dp[i-1][j];
                    op2 = dp[i][j-1];
                    dp[i][j] = max(op1, op2);
                }
            }
        }
    }
    return dp[n][m];
}
```

TC: $O(n*m)$

SC: $O(n*m)$ [ optimise the space ??? ]

Qu21 L P S [ Length of longest palindromic subsequence ]

Ex: e d g a b c a h d [ 5 ]

ans = d a b a d

d a c a d

Solution

s1: e d g a b c a h d

s2: d h a c b a g d e ⟹ rev ( s1 )

lcs ( s1, s2 ) = 5

lps ( s )

↓

lcs ( s, reverse ( s ) )

Break: 8:08 - 8:19 AM

Qu: **Edit distance**

Given 2 strings s1 and s2. find **min no. of operations** to **convert s1 to s2.** You can perform these op^r.

1.) **insert**

2.) **replace**

3) ~~**delete**~~

Example: s1 = sunday

s2 = saturday.

Way 1:

replace (u, a) = sanday

replace (n, t) = satday

insert (u) = saturday

(r) = saturday

} 4

Way 2:

s1 = sunday

s2 = saturday.

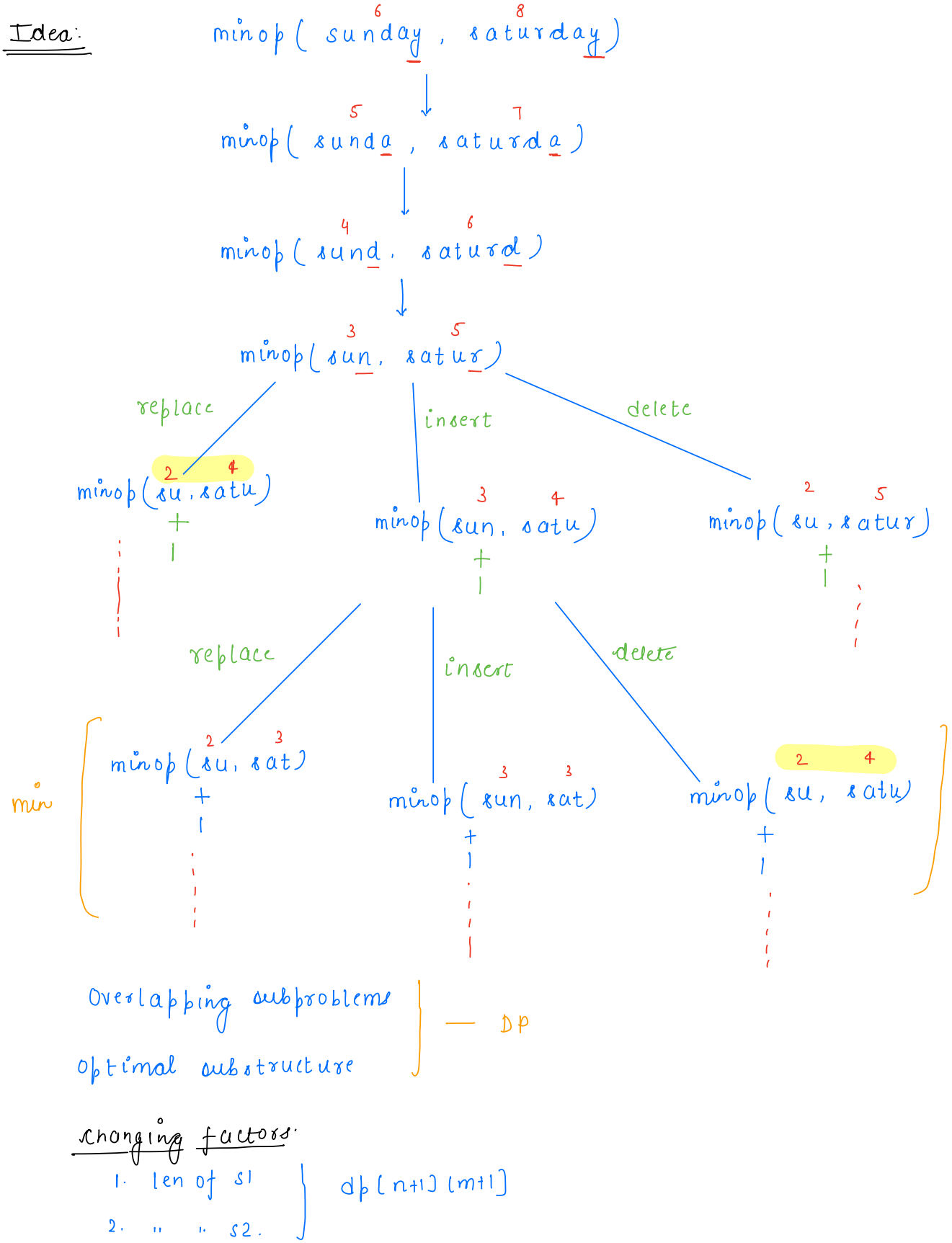
insert (a) = saunday

(t) = satunday

replace (n, r) = saturday

} 3

Idea:




Overlapping subproblems
Optimal substructure
— DP

changing factors:
1. len of s1
2. " " s2.
} dp[n+1][m+1]

Code:

```
int minop( s1, s2, n, m, dp[][] )
    if(n==0 && m==0) {
        return 0;
    }
    if(n==0) {                    s1 = ""
        return m;                 s2 = jawan
    }
    if(m==0) {
        return n;                 s1 = jawan
    }                             s2 = ""
    if( dp[n][m] != -1) {
        return dp[n][m];
    }
    if( s1.charAt(n-1) == s2.charAt(m-1)) {
        dp[n][m] = minop( s1, s2, n-1, m-1);
    } else {
        replace = minop( s1, s2, n-1, m-1);
        insert = minop( s1, s2, n, m-1);
        delete = minop( s1, s2, n-1, m);
        dp[n][m]=min(replace, insert, delete) +1;
    }
}   return dp[n][m];
```

TC: $O(n*m)$

SC: $O(n*m)$ + stack space [neglected]

**Qu:** Wildcard pattern matching [ LC - Hard ]

Given a string s and pattern p. check if string matches the pattern.

* = matches any no. of character [ 0, 1, 2 ... ]

? = matches only one character.

**Example:** isMatch(ab, a * b) ✓

isMatch( a, a*b) ✗

" (aab, a*b) ✓

(axyzpqb, a*b) ✓

(aacb, a?b) ✗

(abc, a?b*c) ✗

(adbc, a?b*c) ✓

## Approach:

**Case 1:** s = a b c d e f

p = a * d ? f

ismatch(abcdef, a * d ? f)

↓

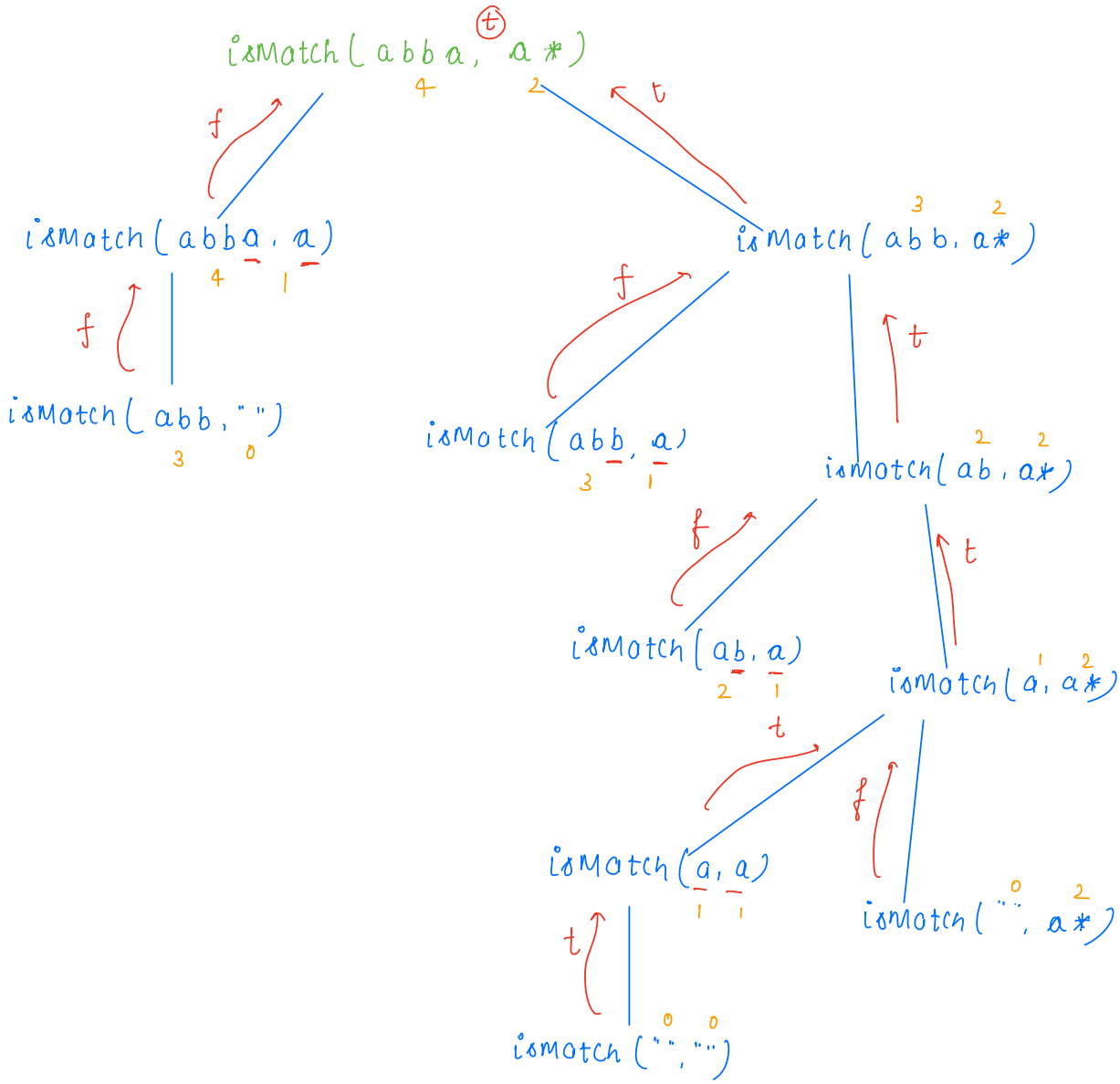
ismatch(abcde, a * d ?)

**Case 2:** ismatch(abcde, a * d ?)

↓

ismatch(abcd, a * d)

**Case 3:** ismatch(abba, a *)


isMatch(abba, a*)
4 2
t
f
isMatch(abba, a)
4 1
isMatch(abb, a*)
3 2
f
t
f
isMatch(abb, a)
3 1
isMatch(ab, a*)
2 2
f
t
isMatch(ab, a)
2 1
isMatch(a, a*)
1 2
t
f
isMatch(a, a)
1 1
isMatch("", a*)
0 2
t
isMatch("", "")
0 0

Recursive code:

```
boolean isMatch( s , p, n. m)
        if( n==0   && m==0) {
            return true;
        }
        if( m==0) {
            return false;
        }
        if ( n==0 ) {
            for(i=0; i<m; i++){
                if( p.charAt(i) != '*'){
                    return false;
                }
            }
            return true;
        }

        if( s.charAt(n-1) == p.charAt(m-1)  ||
            p.charAt(m-1) == '?' )  {
            return isMatch( s, p, n-1. m-1);
        } else if( p.charAt(m-1) == '*') {
            op1 = isMatch( s. p, n. m-1);
            op2 = isMatch( s. p. n-1. m) ;
            return op1 || op2;
        } else {
            return false;
        }
}
```

s = " "
p = **
(t)

s = " "
p = * a *
(f)

* → 0 characters

* → 1 characters

## Memoised code:

n+1 m+1

~~boolean~~ int isMatch( s , p, n. m. dp[][] )

int[][] : 0 - t, 1 - t, 2 - not visited

```
if ( n==0  &&  m==0) {
    return ~~true;~~ 1
}
if ( m==0 ) {
    return ~~false;~~ 0
}
if ( n==0 ) {
    for(i=0; i<m; i++){
        if ( p.charAt(i) != '*'){
            return ~~false;~~ 0
        }
    }
    return ~~true;~~ 1
}
if ( dp[n][m] != 2 ) {
    return dp[n][m];
}
if ( s.charAt(n-1) == p.charAt(m-1)  ||
     p.charAt(m-1) == '?' ) {
    dp[n][m] = isMatch( s, p, n-1, m-1);
} else if ( p.charAt(m-1) == '*') {
    op1 = isMatch( s, p, n, m-1);   * → 0 characters
    op2 = isMatch( s, p, n-1, m);   * → 1 characters
    return op1 (or) op2;
} else {
    return 0;
}
}
```

s = "" p = **  (t)

s = "" p = *a*  (f)

```
boolean getBoolvalue(int n){
    if(n==0){
        return false;
    }
    return true;
}

boolean solve(s, p) {
    n = s.length();
    m = p.length();
    dp[n+1][m+1];
    int ans = isMatch(s, p, n, m, dp);
    return getBoolval(ans);
}
```

Thanks :)