LCS – Longest Common Subsequence

Full search & greedy search

Problem: Given two sequences $X = \langle x_1, x_2, \ldots, x_n \rangle$ and $Y = \langle y_1, y_2, \ldots y_m \rangle$, find the **longest subsequence** $Z = \langle z_1, z_2, \ldots, z_k \rangle$ that is common to X and Y.

A subsequence is a subset of elements from the sequence with **strictly increasing** order (**not necessarily contiguous**).

Example: **X = a,d,c,a,e,b,a,d** **Y = c,l,a,c,b,s,d**

Z = a,c,b,d

Example: **X = a,d,c,a,e,b,a,d** **Y = c,l,a,c,b,s,d**

Z = c,a,b,d

תת-סדרה משותפת ארוכה ביותר LCS

נניח ש $X = <x_1,x_2,\ldots,x_n>$ היא סדרה של **n** אלמנטים

ו $Y = <y_1,y_2,\ldots y_m>$ היא סדרה של **m** אלמנטים .

לשתי סדרות X, Y נאמר ש Z היא תת-סדרה משותפת ל X ול Y אם Z היא תת סדרה של X וגם של Y .

תת-סדרה משותפת ארוכה ביותר - סדרה Z עם מספר אלמנטים המקסימלי.

For example, if X = A,B,C,B,D,A,B and Y = B,D,C,A,B,A, then some common subsequences are:

- A
- B
- C
- D
- A,A
- B,B
- B,C,A
- B,C,B,A **This is one of the longest common subsequences.**
- B,D,A,B **This is one of the longest common subsequences.**

X = "ABCBDAB" and Y = "BDCABA"

Full search חיפוש שלם

Check every subsequence of x[1 . . m] to see if it is also a subsequence of y[1 . . n].

Analysis

Checking = O(n) time per subsequence.

2^m subsequences of x (each bit-vector of length m determines a distinct subsequence of x).

Worst-case running time = O(n*2^m) = exponential time.

X = "ABCBDAB" (128 elements 2^7)

B, A, AB, D, DB, DA, DAB, B, BB, BA, BAB, BD, BDB, BDA, **BDAB**, C, CB, CA, CAB, CD, CDB, CDA, CDAB, CB, CBB, CBA, CBAB, CBD, CBDB, CBDA, CBDAB, B, BB, BA, BAB, BD, BDB, BDA, BDAB, BB, BBB, BBA, BBAB, BBD, BBDB, BBDA, BBDAB, BC, BCB, BCA, BCAB, BCD, BCDB, BCDA, BCDAB, BCB, BCBB, **BCBA**, BCBAB, BCBD, BCBDB, BCBDA, BCBDAB, A, AB, AA, AAB, AD, ADB, ADA, ADAB, AB, ABB, ABA, ABAB, ABD, ABDB, ABDA, ABDAB, AC, ACB, ACA, ACAB, ACD, ACDB, ACDA, ACDAB, ACB, ACBB, ACBA, ACBAB, ACBD, ACBDB, ACBDA, ACBDAB, AB, ABB, ABA, ABAB, ABD, ABDB, ABDA, ABDAB, ABB, ABBB, ABBA, ABBAB, ABBD, ABBDB, ABBDA, ABBDAB, ABC, ABCB, ABCA, ABCAB, ABCD, ABCDB, ABCDA, ABCDAB, ABCB, ABCBB, ABCBA, ABCBAB, ABCBD, ABCBDB, ABCBDA, ABCBDAB, ,

Y = "BDCABA" (64 elements 2^6)

A, B, BA, A, AA, AB, ABA, C, CA, CB, CBA, CA, CAA, CAB, CABA, D, DA, DB, DBA, DA, DAA, DAB, DABA, DC, DCA, DCB, DCBA, DCA, DCAA, DCAB, DCABA, B, BA, BB, BBA, BA, BAA, BAB, BABA, BC, BCA, BCB, **BCBA**, BCA, BCAA, BCAB, BCABA, BD, BDA, BDB, BDBA, BDA, BDAA, **BDAB**, BDABA, BDC, BDCA, BDCB, BDCBA, BDCA, BDCAA, BDCAB, BDCABA, ,

LCS = (BCBA, BCBA)

==========

X = "ABCBDAB" and Y = "BDCABA"

Algorithm on the whole (האלגוריתם בגדול):

Step 1 :

Find all subsequences of X.

Step 2 :

For every subsequence of X check its occurrence in Y.

Step 1

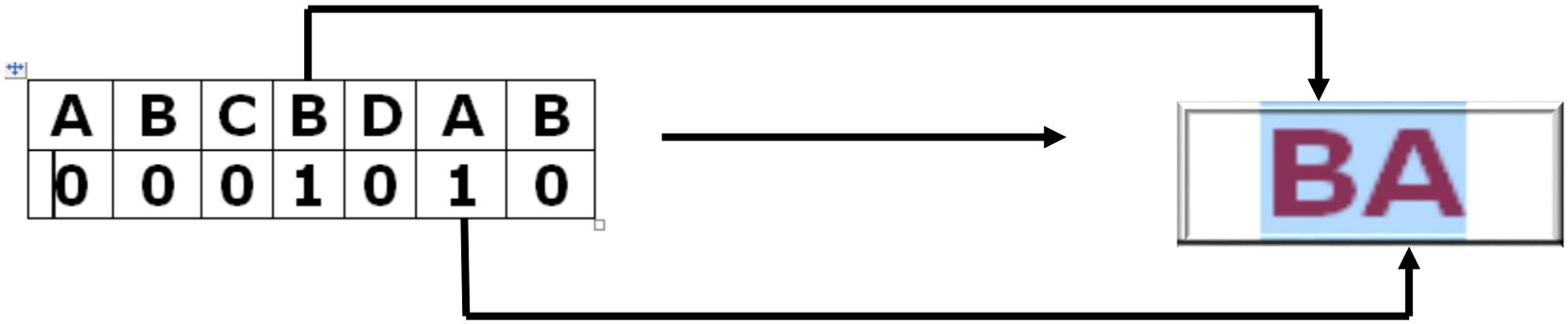
Find all subsequences of X

X = "ABCBDAB"

A	B	C	B	D	A	B	subsequence
0	0	0	0	0	0	0	''
0	0	0	0	0	0	1	B
0	0	0	0	0	1	0	A
0	0	0	0	0	1	1	AB
0	0	0	0	1	0	0	D
0	0	0	0	1	0	1	DB
0	0	0	0	1	1	0	DA
0	0	0	0	1	1	1	DAB
0	0	0	1	0	0	0	B
0	0	0	1	0	0	1	BB
0	0	0	1	0	1	0	BA
0	0	0	1	0	1	1	BAB

Accordance : **0 – ‘’**
1 – ‘letter’

Example :



Write method `int[] plus1(int[] arr){}`

Input arr[] = [0,0,0,0,0,0,0].
Output arr[] = [0,0,0,0,0,0,1].

Input arr[] = [0,0,0,0,0,0,1].
Output arr[] = [0,0,0,0,0,1,0].

……

Input arr[] = [0,1,0,1,0,1,0].
Output arr[] = [0,1,0,1,0,1,1].

……

Input arr[] = [1,1,1,1,1,1,0].
Output arr[] = [1,1,1,1,1,1,1].

arr[] =	[0,0,0,0,0,0,**0**]	+ **1** =	[0,0,0,0,0,0,**1**].	1
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,0,0,**1**]	+ **1** =	[0,0,0,0,0,**1,0**].	2
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,0,1,**0**]	+ **1** =	[0,0,0,0,0,1,**1**].	3
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,0,1,**1**]	+ **1** =	[0,0,0,0,**1,0,0**].	4
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,1,0,**0**]	+ **1** =	[0,0,0,0,1,0,**1**].	5
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,1,0,**1**]	+ **1** =	[0,0,0,0,1,**1,0**].	6
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,1,1,**0**]	+ **1** =	[0,0,0,0,1,1,**1**].	7
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,0,1,1,**1**]	+ **1** =	[0,0,0,**1,0,0,0**].	8
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,1,0,0,**0**]	+ **1** =	[0,0,0,1,0,0,**1**].	9
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	
arr[] =	[0,0,0,1,0,0,**1**]	+ **1** =	[0,0,0,1,0,**1,0**].	10
index	0 1 2 3 4 5 6		0 1 2 3 4 5 6	

algorithm :

arr[i]= 0 → arr[i]= 1 ; arr[i-1] Does not change ;stop
arr[i]= 1 → arr[i]= 0 ; **if** arr[i-1]= 0 arr[i-1]= 1; arr[i-2] Does not change ;stop
arr[i]= 1 → arr[i]= 0 ; **if** arr[i-1]= 1 arr[i-1]= 0; **if** arr[i-2]= 0 arr[i-2]= 1;stop
if arr[i-2]= 1 .....

Write method **public static** String[] *allCombinations*(String X){}

int **count** is count elements of X is 2^(length of X).

String[] **result** - array of String length of result is 2^(length of X).

int[] **bin** - array of 1 & 0 (length of bin is length of X).

```
loop  from 0 to count    (index i)
begin loop 1
        bin  = plus1(bin)
        String resultElement
        loop   from 0 to length of X   (index j)
        begin loop 2
            if (bin[j]==1)
                resultElement = resultElement + X. charAt(j)
        end loop 2
        result[i] =  resultElement
end loop 1
```

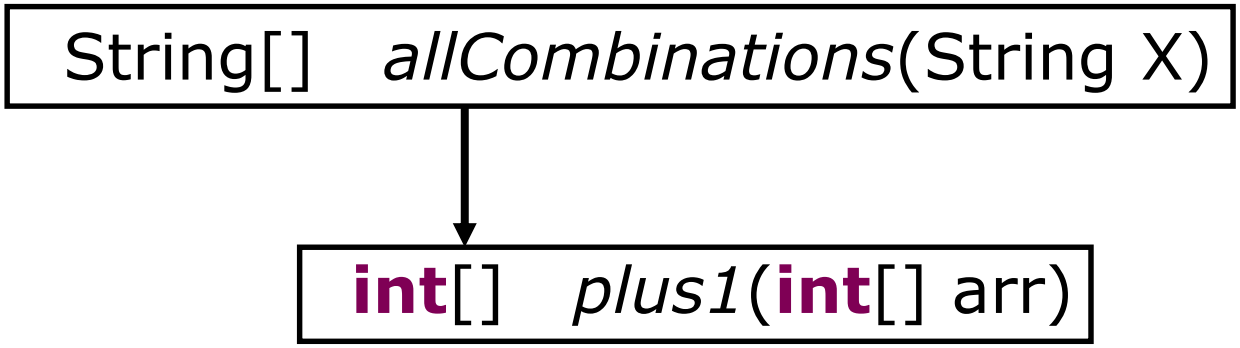
output : String[] **result**

Conclusions :

write 2 methods –

int[] *plus1*(**int**[] arr)

String[] *allCombinations*(String X)



Step 2

For every subsequence of X check its occurrence in Y.

find element (the first occurrence)

str

	A	**B**	**C**	**B**	**D**	**A**	**B**	**C**	**B**	**D**	**A**	**B**
index	0	1	2	3	4	5	6	7	8	9	10	11

Input: int **begin** String **str** char **element**

begin = 3 element = 'A'

	A	**B**	**C**	**B**	**D**	**A**	**B**	**C**	**B**	**D**	**A**	**B**
index	0	1	2	3	4	5	6	7	8	9	10	11

Output : index = 5

Write method

```
public static int findElement(int beg, String str, char elem){}
```

input : parameters - int beg, String str, **char** elem

```
index = -1
boolean flag = true
loop  from  beg to  str.length && flag        (index i)
begin loop
    if (str.charAt(i)==elem)
        index = i
        flag = false
end loop
```

output : index

Write method

public static boolean findSubstr(String small, String big){}

**input : parameters - ** String small, String big

```
boolean result = true
int index = 0
loop  from 0 to length of small && result    (index i)
 begin loop
     index = findElement(index, big, small.charAt(i))
     if (index==-1)
         result = false
 end loop
```

output : result

Write method

public static String fullSearch(String X, String Y){}

**input : parameters - ** String X, String Y

```
String result = ""
String sSmall   if (length of X < length of Y) sSmall = Y  else sSmall = X
String sLong    if (length of X < length of Y) sLong = X  else sLong = Y

String[] t = allCombinations(sSmall)

loop  from  0 to  length of t          (index i)
begin loop
        if (findSubstr(t[i], sLong))                    ??????
            if (length of t[i] > length of result))     ??????
                result = t[i]
end loop
output : result
```

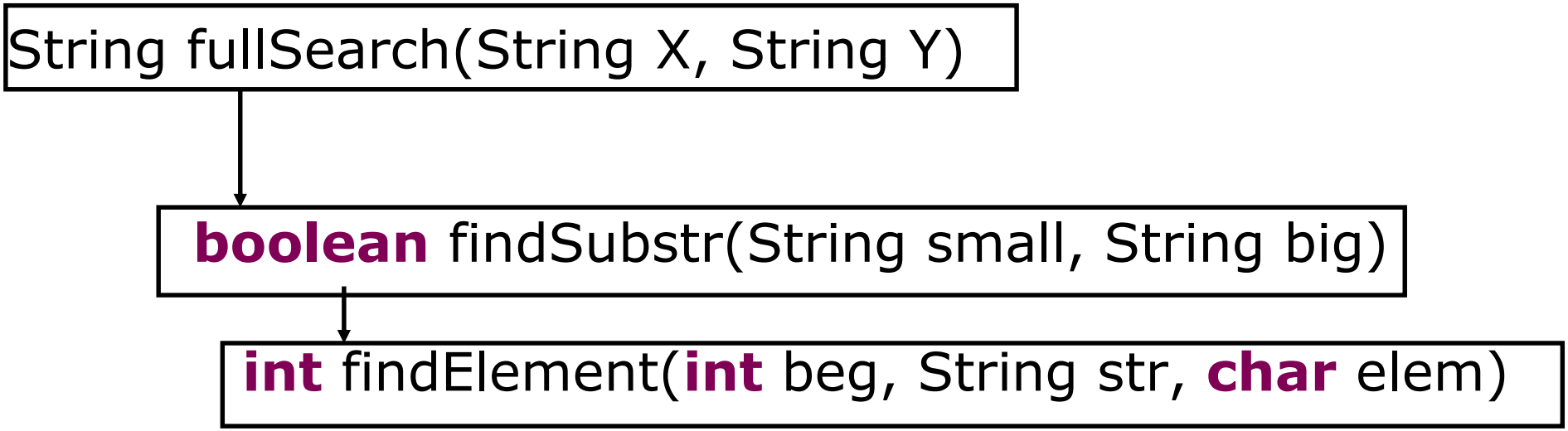
Conclusions :

write 3 methods –

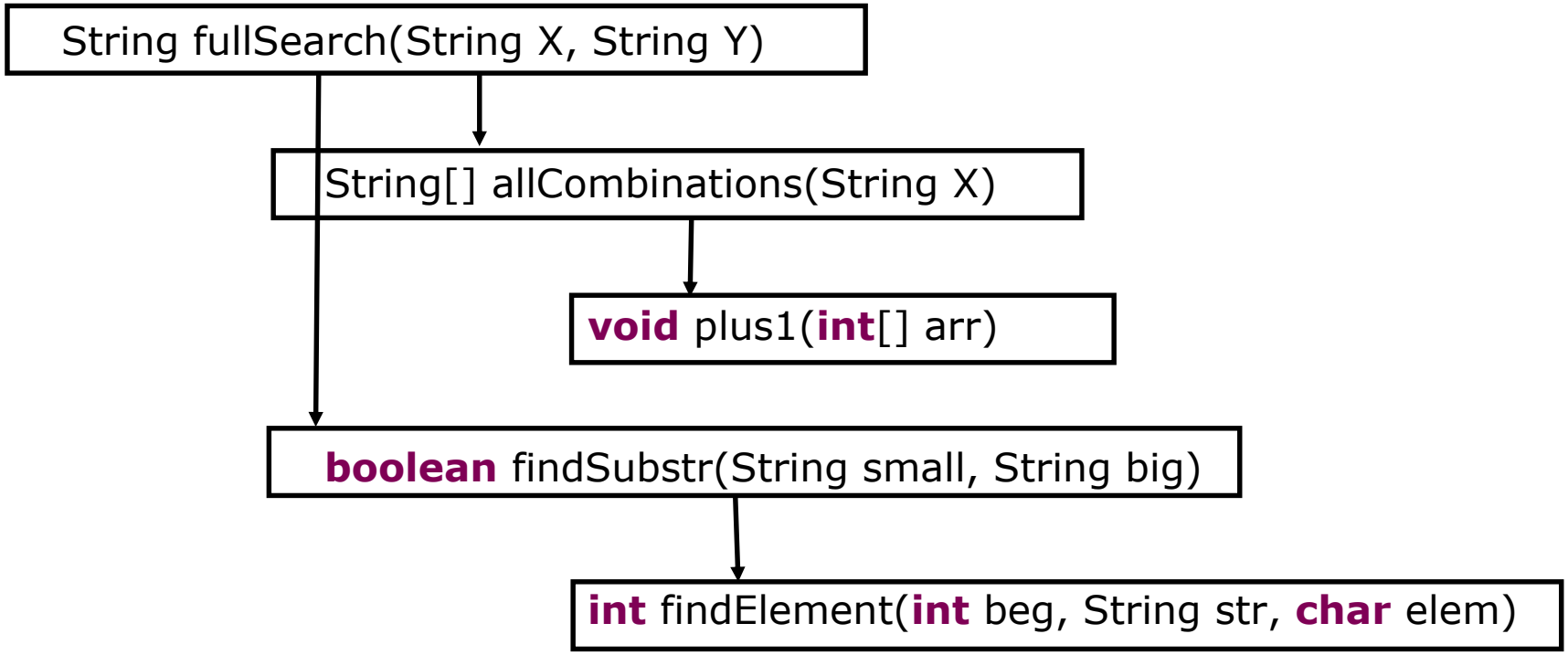
int *findElement*(**int** beg, String str, **char** elem)

boolean *findSubstr*(String small, String big)

String *fullSearch*(String X, String Y)



Complete algorithm



Greedy search חיפוש חמדני

X = "ABCBDAB"

0 1 2 3 4 5 6 index1

Y = "BDCABA"

0 1 2 3 4 5 index2

X Y

0

ABCBDAB → BDC***A***BA "A"

1

A***B***CBDAB → BDC***AB***A "AB"

3

ABCBDAB → BDC***ABA*** "ABA"

Y	X			
0				
BDCABA	→	A***B***CBDAB	"B"
1			
B***D***CABA	→	A***B***CB***D***AB	"BD"
3			
BD***C***ABA	→	A***B***CB***D***AB	"BD"
4			
BDC***A***BA	→	A***B***CB***DA***B	"BDA"
5			
BDCA***B***A	→	A***B***CB***DAB***	"BDAB"
6			
BDCAB***A***	→	A***B***CB***DAB***	"BDAB"

X,Y

ABA

****

Y,X

BDAB

1. Algorithm 1

Write method String greedy(String X, String Y)

input : parameters - String X, String Y

```
String result = “”
int ind     = 0
int index = 0
int beg     = 0

loop ( ind < length of X )
begin loop
      index = findElement(beg, Y, X.charAt(ind))
      if (index != -1)
            result = result + X.charAt(ind)
            beg = index + 1
end loop
```

output : result

Write method

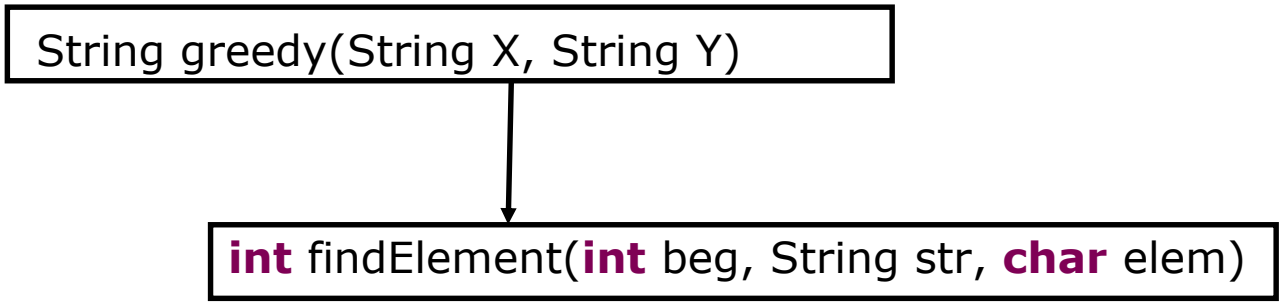
```
public static int findElement(int beg, String str, char elem){}
```

algorithm :

input : parameters - int beg, String str, **char** elem

```
boolean flag = true
loop  from  beg to  str.length && flag          (index i)
begin loop
     if (str.charAt(i)==elem)
          index = i
          flag = false
end loop
```

output : index


String greedy(String X, String Y)
int findElement(int beg, String str, char elem)

2. Algorithm 2

Write method String greedy2(String X, String Y)

```
String result = “”
int ind     = 0
int index = 0
int beg     = 0
int letters[] = new int[26]

loop from 0 to length of X          (index i)

begin loop
     int place = (int)(X.charAt(i)-'A')
     letters[place]++
end loop

loop ( ind < length of X )
begin loop
     char x = X.charAt(ind)
          int place = (int)(x - 'A')
          if (letters[place] > 0
```

```
            index = findElement(beg, Y, x)
            if (index != -1)
                result = result + X.charAt(ind)
                beg = index + 1
                letters[place]--
            ind = ind + 1
end loop
```

output : result