Qixiang (David) Zhang

Nai-Jack Li

**LCS - longest common subsequence**

**Problem Statement**

Let *bstring(n, x)* be the length-*n* character string consisting of 0's and 1's which, when interpreted as a number expressed in binary, has value *x*.

Implement and analyze the time complexity of an algorithm **CountLCS** that, given n in the range , and two integers x and y that are each in the range , determines the number of distinct strings that are LCS's of *bstring(n, x)* and *string(n, y)* and displays them.

**Algorithm Description**

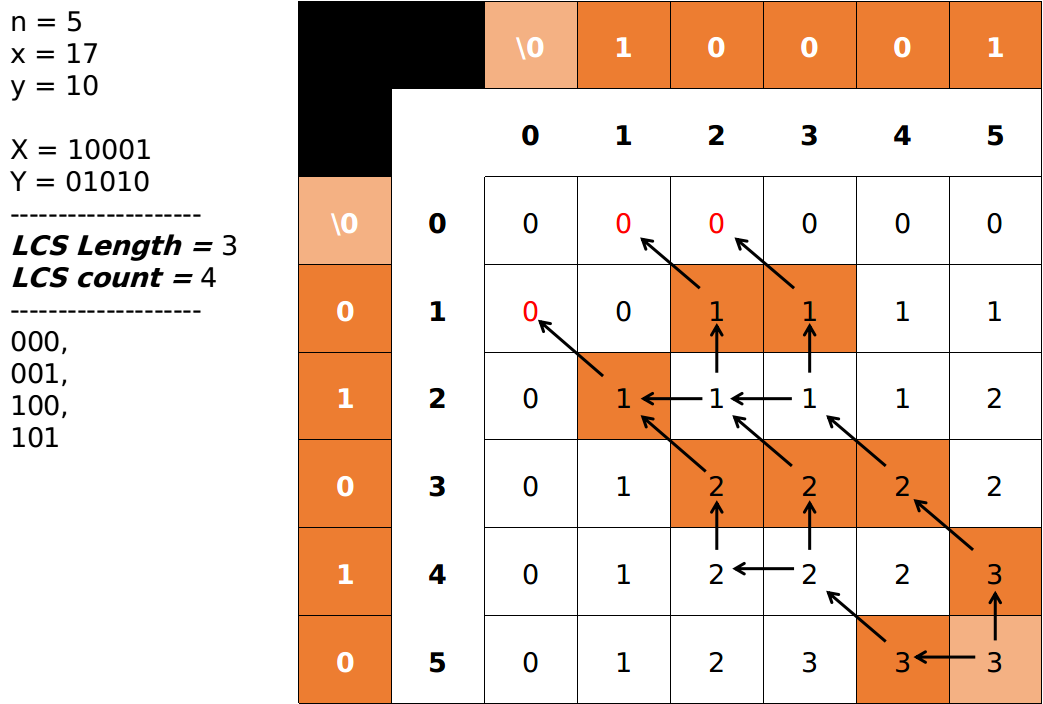
Our algorithm (***CountLCS***) for this project consists of 2 major parts: count the size of LCS (***LCS***) and print out the longest common subsequences (***findLCS***).

In the first part - ***LCS***, the idea is to compare each character one by one from each of X and Y strings. The algorithm builds a table with size of n by n. The binary string X would be the vertical axis while the binary string Y would be the horizontal axis. Start from the top left corner, fill the first row and the first column with 0’s. Then, start from the first character of X and Y strings, if they are identical, then we put a number that equals the cell’s upper left neighbor cell’s value plus 1. Otherwise, the number should be the larger of the above cell and the left cell. In this case, if the first characters of X and Y are a match, then the number should be . Applying the same method, we can complete the table. The size of LCS will be the value of the cell at the lower right corner.

In the second part - ***findLCS***, the idea is to backtrack the values in the table. Start from the last characters of both X and Y. If matched, then record the matching character as a string and move both pointers 1 space back. If not matched, look at the value on the left and the value above. If the value on the left is larger, move the pointer on string X one space to the left and recursively call the ***findLCS*** function and vice versa. If the value on the left and above are same, then do the process in both directions. The base case is when either the pointers reach to the front of the strings. In the end, all the stored strings in the set need to be reversed since they are recorded from the end of the string to front.

Notice that the procedure of ***findLCS*** is just like building up a tree. The nodes are the cells in the table and we add one or two child nodes depends on the condition mentioned above. Therefore, we can optimize the procedure by memorization. The idea is if each node has recorded the matching characters of the child nodes, then we can construct the LCS string by combining the matching characters of below nodes and above nodes when revisiting nodes. To do that, we record the matching characters from child nodes when a recursion function is called back.

Below is the chart for the case n = 5, x = 17 and y = 10. The cells filled with color are the nodes that the characters of X and Y are matched.



**Analysis of the asymptotic worst-case time complexity (θ-notation) in terms of *n***

Subroutine ***CountLCS*** consists of 2 major parts - ***LCS*** and ***findLCS***.

Subroutine ***LCS*** is double for-loop and both are from 0 to n, therefore, the time complexity should be .

Subroutine ***findLCS*** builds up a tree with branching factor of 2 and height of . The tree takes . Therefore, the recurrence function is when . When This can be solved by induction:… substitute out for times. Then,. When ,.

To sum up, the time complexity for the algorithm ***CountLCS*** is .

**Instructions on how to compile your program on the ICS Linux system**

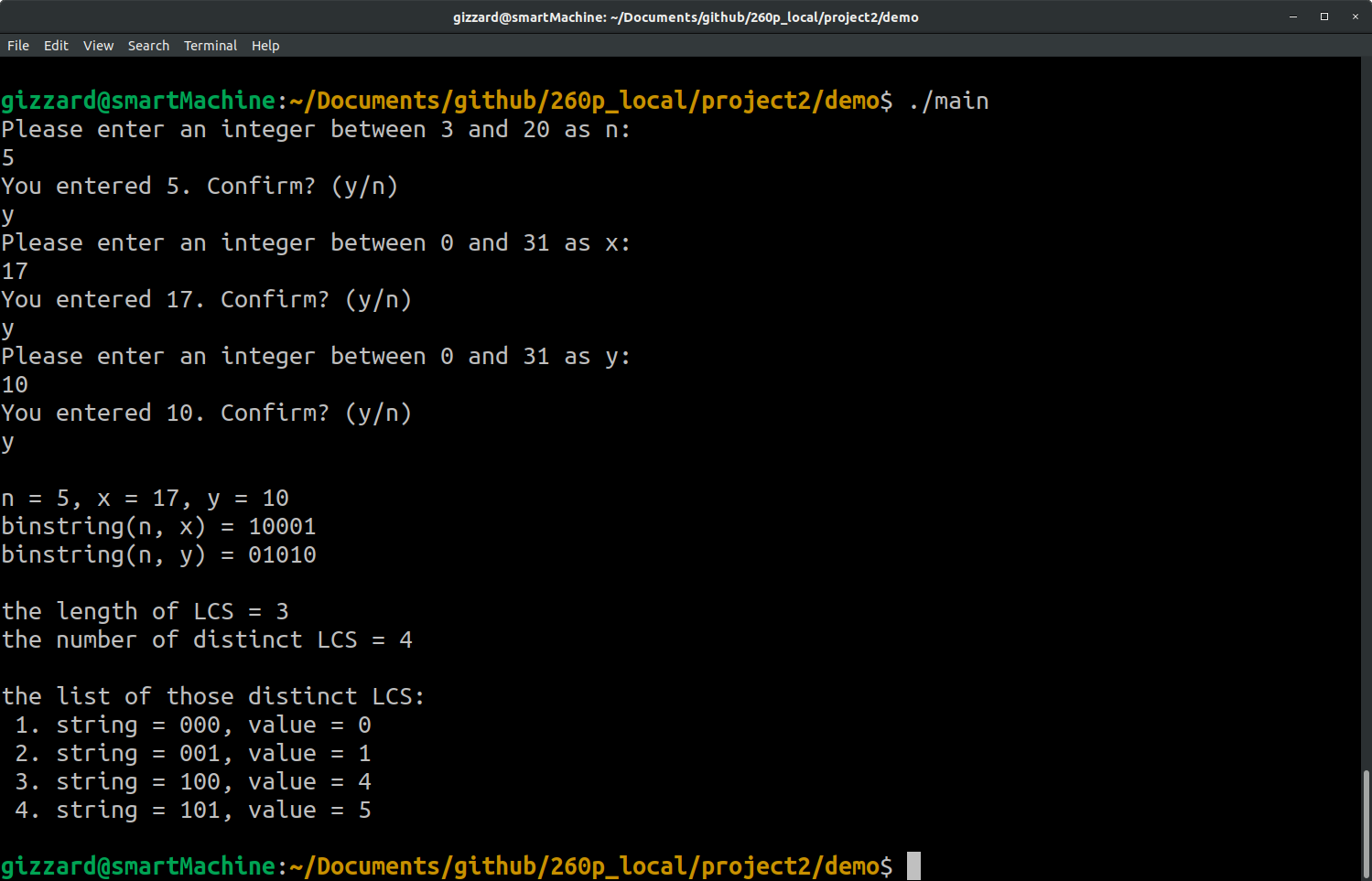
1. Open terminal and go to the directory contains main.cpp
2. Type in `g++ -std=c++11 -o main main.cpp` to **compile** the program



1. Type in `./main` to **run** the program



**Sample input and output produced by the execution of your program**



As the terminal shown above, we sample input . For the outputs, the *bstring(n, x)* outputs and *bstring(n, y)* outputs. The length of LCS is 3. The determined number of distinct LCS is . The list of those LCS is:

**BONUS: identify integers x and y that yield the largest possible number of distinct LCS's when n = 14, when n = 15, and when n = 16**

We first try to use brute force method to find out the largest possible number of distinct LCS's when n = 14, but there are (2^14 - 1)^2 = 268402689 possibles combinations for x and y. Without any optimization, this kind of computation would make most of the laptops run out of memory.

We discovered some pattern when using our algorithm to run the small cases such as when n = 5, 6, 7, 8, 9, 10. We found that one of X and Y strings are always 101010… or 010101... . 1’s and 0’s are alternating. Let’s put 10 as a subgroup in those alternating strings, at least one of the group is reversed based on the output results. Therefore, we narrowed down the search by making Y string the same alternating pattern with one of the subgroup reversed. In addition, we found that the other string to make the LCS is always larger or equal to . Summing up, we narrowed down the search a lot from these patterns. **Below are our results:**

**n=14:**

**largest # of distinct LCS's = 70**

x = 8988

y = 5461

x = 3633

y = 10922

x = 9100

y = 5461

x = 3185

y = 10922

x = 9102

y = 5461

x = 7281

y = 10922

x = 9116

y = 5461

x = 3697

y = 10922

x = 9830

y = 5461

x = 6553

y = 10922

x = 10012

y = 5461

x = 3641

y = 10922

x = 12684

y = 5461

x = 3171

y = 10922

x = 12686

y = 5461

x = 7267

y = 10922

x = 12700

y = 5461

x = 3683

y = 10922

x = 12742

y = 5461

x = 6371

y = 10922

x = 12748

y = 5461

x = 3299

y = 10922

x = 12750

y = 5461

x = 7395

y = 10922

x = 13084

y = 5461

x = 3635

y = 10922

x = 13196

y = 5461

x = 3187

y = 10922

x = 13198

y = 5461

x = 7283

y = 10922

x = 13212

y = 5461

x = 3699

y = 10922

**n=15:**

**largest # of distinct LCS's = 96**

x = 17969

y = 10922

x = 17969

y = 10922

x = 17971

y = 10922

x = 26161

y = 10922

x = 17977

y = 10922

x = 20017

y = 10922

x = 18019

y = 10922

x = 25393

y = 10922

x = 18033

y = 10922

x = 18225

y = 10922

x = 18035

y = 10922

x = 26417

y = 10922

x = 18201

y = 10922

x = 19569

y = 10922

x = 18225

y = 10922

x = 18033

y = 10922

x = 18227

y = 10922

x = 26225

y = 10922

x = 18233

y = 10922

x = 20081

y = 10922

x = 19555

y = 10922

x = 25369

y = 10922

x = 19569

y = 10922

x = 18201

y = 10922

x = 19571

y = 10922

x = 26393

y = 10922

x = 19683

y = 10922

x = 25497

y = 10922

x = 20017

y = 10922

x = 17977

y = 10922

x = 20019

y = 10922

x = 26169

y = 10922

x = 20025

y = 10922

x = 20025

y = 10922

x = 20067

y = 10922

x = 25401

y = 10922

x = 20081

y = 10922

x = 18233

y = 10922

x = 20083

y = 10922

x = 26425

y = 10922

x = 25369

y = 10922

x = 19555

y = 10922

x = 25393

y = 10922

x = 18019

y = 10922

x = 25395

y = 10922

x = 26211

y = 10922

x = 25401

y = 10922

x = 20067

y = 10922

x = 25497

y = 10922

x = 19683

y = 10922

x = 26161

y = 10922

x = 17971

y = 10922

x = 26163

y = 10922

x = 26163

y = 10922

x = 26169

y = 10922

x = 20019

y = 10922

x = 26211

y = 10922

x = 25395

y = 10922

x = 26225

y = 10922

x = 18227

y = 10922

x = 26227

y = 10922

x = 26419

y = 10922

x = 26393

y = 10922

x = 19571

y = 10922

x = 26417

y = 10922

x = 18035

y = 10922

x = 26419

y = 10922

x = 26227

y = 10922

x = 26425

y = 10922

x = 20083

y = 10922

**n=16:**

**largest # of distinct LCS's = 141**

x = 50972

y = 21845

x = 14563

y = 43690