Sphere Online Judge (SPOJ) Algorithms

What is the algorithmic approach to solve this problem: SPOJ.com - Problem LCS0 ?

This question previously had details. They are now in a comment.

Answer Request Follow 37 Comment 1 Downvote

3 Answers

Swapnil Joshi, Computer Science Undergraduate

Answered Jul 6, 2014

This question is the real shit and believe me when I say so:-

Here is what I did ,

1)Used normal DP ,O(mn) complexity O(mn) space got tle instead of run time error(don't know why) for the 9th case.

2)Used Hirschbirg algorithm O(mn) complexity ,O(min(m,n)) space ,still got tle for 9th case.

```
#include <cstdlib>
#include <algorithm>
#include <iostream>
#include <cstring>
#include <string>
#include <cstdio>
using namespace std;

int* compute_help_table(const string & A,const string & B);
string lcs(const string & A, const string & B);
string simple_solution(const string & A, const string & B);

int main(void) {
    string A,B;
    cin>>A>>B;

    cout << lcs(A, B).size() << endl;

    return 0;
}

string lcs(const string &A, const string &B) {
    int m = A.size();
    int n = B.size();

    if (m == 0 || n == 0) {
        return "";
    }
    else if(m == 1) {
        return simple_solution(A, B);
    }
    else if(n == 1) {
```


Related Questions

How do I solve the problem "Remove The String" (PSTRING) on SPOJ?

Are SPOJ problems worth solving?

How do I solve the nested doll problem on SPOJ?

What changes can be made to Floyd-Warshall Algorithm to solve this problem NAJKRACI of SPOJ?

What is the importance of solving classical SPOJ problems?

What is the algorithmic approach to solving the problem "Paradox" on SPOJ?

How does one solve ANARC07G SPOJ.com - Problem ANARC07G on SPOJ?

How do you solve SPOJ.com - Problem FSEATS on SPOJ?

What is the correct approach to solve SPOJ.com - Problem ACMAKER?

How do I solve SPOJ ANT problem?

+ Ask New Question

More Related Questions

Question Stats

36 Public Followers

12,387 Views

Last Asked Sep 22, 2015

Edits

```
        int i = m / 2;

        string Asubstr = A.substr(i, m - i);
        //reverse(Asubstr.begin(), Asubstr.end());
        string Brev = B;
        reverse(Brev.begin(), Brev.end());

        int* L1 = compute_help_table(A.substr(0, i), B);
        int* L2 = compute_help_table(Asubstr, Brev);

        int k;
        int M = -1;
        for(int j = 0; j <= n; j++) {
            if(M < L1[j] + L2[n-j]) {
                M = L1[j] + L2[n-j];
                k = j;
            }
        }

        delete [] L1;
        delete [] L2;

        return lcs(A.substr(0, i), B.substr(0, k)) + lcs(A.substr(i, m -
    }
}

int* compute_help_table(const string &A, const string &B) {
    int m = A.size();
    int n = B.size();

    int* first = new int[n+1];
    int* second = new int[n+1];

    for(int i = 0; i <= n; i++) {
        second[i] = 0;
    }

    for(int i = 0; i < m; i++) {
        for(int k = 0; k <= n; k++) {
            first[k] = second[k];
        }

        for(int j = 0; j < n; j++) {
            if(j == 0) {
                if (A[i] == B[j])
                    second[1] = 1;
            }
            else {
                if(A[i] == B[j]) {
                    second[j+1] = first[j] + 1;
                }
                else {
                    second[j+1] = max(second[j], first[j+1]);
                }
            }
        }
    }

    delete [] first;
    return second;
}

string simple_solution(const string & A, const string & B) {
    int i = 0;
    for(; i < B.size(); i++) {
        if(Page on Page on Page on Page on b.at(i) == Page on Page on Pa
```

```
166 }
```

3)Used an NlogN algorithm.
http://www.cs.ucf.edu/courses/cap5510/fall2009/SeqAlign/LCS.efficient.pdf

4)Used the combination of 2 and 3,still got TLE,

The only things left to try are bit vector algorithm and BK tree ,one of them will surely work so incase you are trying to implement one of the above 4,I saved your time.
UPDATE:Incase you are wondering about the NlogN algorithm it has a high constraint value and eventually turns out to be equal to N^2 in worst cases, however a bit vector algorithm has time complexity of O(mn)/32 or O(mn)/64 in worst case and O(n) in best case!!
BK tree claims to solve in O(log n)!! ,however I am still not sure if it is actually that good.The algorithm is complex enough to go above my head.

Mark Gordon, ICPC 2011 Gold Medalist, TC Target, USACO coach, 'msg555' on TopCoder/Codeforces

Answered Jul 12, 2014

This problem is pure evil. At the end of the day, I was eventually able to force a $O(NM/W)$ algorithm through (where W is the word size (e.g. 32))

This method is based on row by row LCS. We have R(i, j) represent the longest common subsequence of A[1...i] and B[1...j]. In particular our base case has R(0, j) = R(i, 0) = 0. Then we can write

$R(i,j) = 1 + R(i-1,j-1)$ when $A_i = B_j$
$R(i,j) = max(R(i-1,j), R(i,j-1))$ otherwise

This leads immediately to a rather simple dynamic programming algorithm, however even the most optimized version that I could write using this recurrence still was too slow.

To make it faster notice that for $j > 0$ we have $0 \leq R(i,j) - R(i,j-1) \leq 1$. In particular this means that we can instead express the R table using only a single bit for each entry. Let $X(i,j) = R(i,j) - R(i,j-1)$. Then we can write the following recurrence to calculate X using another bitmatrix, Y.

$Y(i,j) = Y(i,j-1)$ otherwise

$X(i,j) = 0$ if $Y(i,j-1)$
$X(i,j) = 1$ if $A_i = B_j$ or $X(i-1,j)$ (and not the previous condition)
$X(i,j) = 0$ otherwise

Where these recurrences come from is going to have to be an exercise for the reader. However, once we have them we're in luck, these values can be readily calculated with bit arithmetic by representing X(i, j) as a bitmask, Y(i, j) as a bitmask, and $B_i = c$ as a bitmask.

All of the calculations are pretty obvious except for how to express the "otherwise" case of Y(i, j). To make this case work out we need to make arithmetic subtraction do the carry work for us. To do so you can construct a bitmask of all the bits that are "forced" by the first two conditions and another bitmask of all the bits that are forced on by the first two conditions. Subtract the first mask from the second mask shifted one to the left (you also need to bring in the high bit from the last word of Y) and we've done it.

This works because

1) We subtract 1 from the start of each section that should be on.
2) The borrowing ripples through until we get to the next forced bit.
3) The borrowing caused by different bits never overlaps

Anyway, here's how I expressed this bit arithmetic in C in my solution. Let xprev be the bitmask value of X in the previous row, C be the set of matching indexes, and ylst be the highest bit of the previous Y word.

```
1 u32 s = C | xprev; // calculate forced bits of y
2 u32 t = C & ~xprev; // calculate forced on bits of y
3 u32 q = s - ((t << 1) | (ylst ? 1 : 0)); // calculate non forced bit val
4 u32 y = (q & ~s) | t; // put it all together
5 u32 x = (C | xprev) & ~((y << 1) | (ylst ? 1 : 0)); // use y to calculat
```

8.7k Views · View Upvoters · Answer requested by Swapnil Joshi

Shubham Kashyap, studied at School of Engineering, Cochin University of Science and Technology
Answered Jun 15, 2016

At first I thought this is due to slow I/O spent 3–4 hours trying to optimize my I/O. Then I read the comments and in there someone has mentioned that O(m*n) or O(n^2) will fail. After that I started googling about different

```
1  O( ceiling(|a|/w)*|b| )
```

This gives a speedup over simple O(|a|*|b|) algorithms. The time does not depend on properties of 'a' and 'b'. For random strings and moderate alphabets, the bit-string LCS algorithm will be faster than the special case algorithms. Here is a C program of the above but I didn't understand the implementation.

Happy coding