LCS in short

Yann POMIE - DO3
13/12/2021

Summary

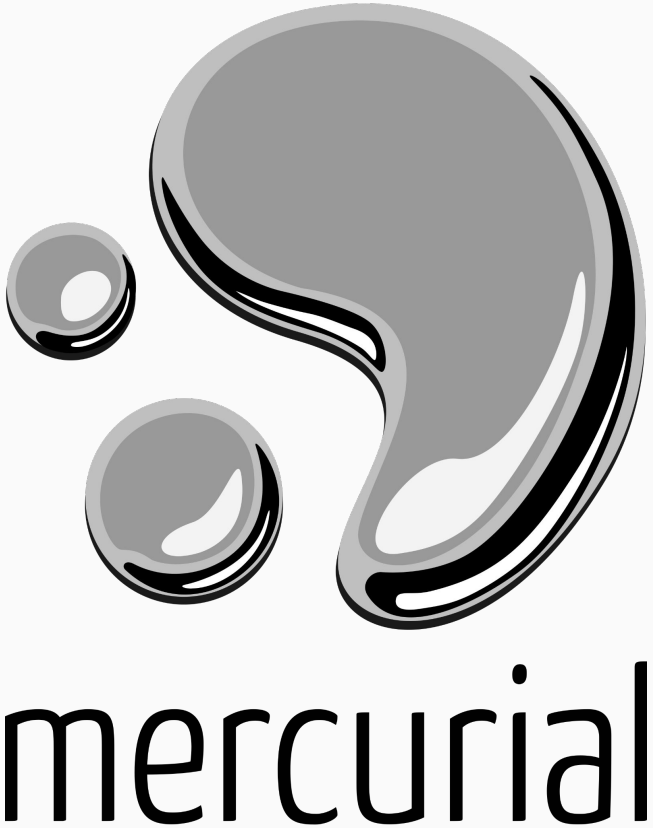
1. Context
2. How it works
    a. Length’s computing
    b. Backtracking
3. Conclusion

1. Context

```
yann@yann-Latitude-5420 ~ cat a
I
am
a
file
with
some
cool
lines
```

```
yann@yann-Latitude-5420 ~ cat b
I
am
a
file
with
some
different
and
new
lines
```

```
yann@yann-Latitude-5420 ~ diff a b
7c7,9
< cool
---
> different
> and
> new
```


mercurial

2. How it works

The LCS (Longest Common Subsequence) Algorithm

- Finds common sequences of characters
- A subsequences of ABBA = {A, AB, ABA, AA, …}
- subsequences ∉LCS(X, Y) => diff
- ex : `LCS(“rabbit”, “boitier”) = bit`

a. Length's computing

```
#define str std::string
#define mat std::vector<std::vector<unsigned>>
```

```
mat LCSLength(str seq1, str seq2) {
    //constructing matrix
    mat C(seq1.size() + 1, std::vector<unsigned>(seq2.size() +1 , 0));

    //for each letters of seq1...
    for (unsigned y = 1; y < C.size(); ++y)
        //we look at all the letters of seq2
        for (unsigned x = 1; x < C[0].size(); ++x)
            //if both letters are the same the current common subsequence grows
            if (seq1[y - 1] == seq2 [x - 1])C[y][x] = C[y-1][x-1] + 1;
            //else it gets the max length between top
            else C[y][x] = (C[y][x - 1] > C[y -1][x]) ? C[y][x - 1] : C[y -1][x];

    return C;

}
```

	Ø	A	G	C	A	T
Ø	0	0	0	0	0	0
G	0	↑←0	↖1	←1	←1	←1
A	0	↖1	↑←1	↑←1	↖2	←2
C	0	↑1	↑←1	↖2	↑←2	↑←2

ex 3rd line : lcslength(A,∅) = 0, lcslength(A,A) = 1 ….

https://en.wikipedia.org/wiki/Longest_common_subsequence_problem#Traceback_approach

b. Backtracking

		0	1	2	3	4	5	6	7
		Ø	M	Z	J	A	W	X	U
0	Ø	**0**	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	1	1
2	M	0	**1**	1	1	1	1	1	1
3	J	0	1	1	**2**	2	2	2	2
4	Y	0	1	1	2	2	2	2	2
5	A	0	1	1	2	**3**	3	3	3
6	U	0	1	1	2	3	3	3	**4**
7	Z	0	1	2	2	3	3	3	4

```
str backtrackRec(const mat C, const str row, const str column, unsigned y, unsigned x) {
    //stop condition : if we are on the upper edge of the matrix
    if (x == 0 || y == 0) return "";
    //if the letters are the same we go one case to the left and we concatenate the letter
    if (row[y - 1] == column[x - 1])
        return backtrackRec(C, row, column, y -1, x - 1) + row[x -1];
    //if the left value is higher than the upper one go left
    if (C[y][x - 1] > C[y - 1][x])
        return backtrackRec(C, row, column, y, x - 1);
    //go up by default
    return backtrackRec(C, row, column, y - 1, x);
}

str backtrack(const mat C, const str row, const str column) {
    //the matrix will form a path that will go fromm the upper left to the down left
    //we will go through this path from down to up and from right to left
    return backtrackRec(C, row, column, C.size() - 1, C[0].size() - 1);
}
```

https://en.wikipedia.org/wiki/Longest_common_subsequence_problem#Reading_out_a_LCS

```
function backtrackAll(C[0..m,0..n], X[1..m], Y[1..n], i, j)
    if i = 0 or j = 0
        return {""}
    if X[i] = Y[j]
        return {Z + X[i] for all Z in backtrackAll(C, X, Y, i-1, j-1)}
    R := {}
    if C[i,j-1] ≥ C[i-1,j]
        R := backtrackAll(C, X, Y, i, j-1)
    if C[i-1,j] ≥ C[i,j-1]
        R := R ∪ backtrackAll(C, X, Y, i-1, j)
    return R
```

3. Conclusion

Thank you for listening!

my code : https://github.com/WoodenMaiden/LCSpresentation