CS F211

Data Structures and Algorithms

Practice Problems

Recursion and Time Complexity

Problem 1

Write a recursive function that finds the length of the longest common subsequence of two strings. A subsequence of a string is a string that can be obtained by deleting several characters from the original string. For example, "hlo", "hello", "elo", and "hll" are all valid subsequences of "hello".

Along with the two strings, the function also takes two integers i and j, which represent the starting indices of the two strings from which the length of the subsequence must be found. For example, calling the function with the parameters ("concise", "conscious", 2, 3) requires that the function find the length of the longest common subsequence of "ncise" (suffix of "concise" starting from index 2) and "scious" (suffix of "conscious" starting from index 3).

You must use this definition of the function to write a recurrence relation and handle base cases.

```cpp
#include <iostream>
using namespace std;

int lcs(string s1, string s2, int i, int j) {
    /* ??? */
}

int main() {
    string s1 = "concise", s2 = "conscious";
    cout << "Length of LCS: " << lcs(s1, s2, 0, 0) << endl;
    return 0;
}
```

Next task: Alter the code to find the longest common subsequence of two linked lists of integers instead of strings. The function must only take the heads of the two lists as input.

Problem 2

The Sudoku class is a container for a 9x9 grid that represents a Sudoku puzzle. Your job is to complete its friend function solveSudoku(), which solves a Sudoku using recursion and backtracking. The function takes 3 parameters: a Sudoku object, and two integers that represent the cell that the solver function must guess for. If the cell is unoccupied (which is represented by 0), the function must iterate through all the possible values that the cell can take and then check if the guess leads to a possible answer by recursively calling itself for the next pair of row and column. You must also complete the implementation of a helper method called isSafe(), which checks whether placing a number in a particular cell leads to any row, cell, or subgrid clashes. The solveSudoku() function, if it finds a solution, returns **true**, and the solution it finds is saved by altering the original Sudoku object itself.

```cpp
#include <iostream>
using namespace std;

class Sudoku {
private:
    int grid[9][9];

public:
    Sudoku(int inputGrid[9][9]) {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) grid[i][j] =
                inputGrid[i][j];
        }
    }

    void display() const {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) cout << grid[i][j] <<
                " \n"[j == 8];
        }
    }

    friend bool solveSudoku(Sudoku &sudoku, int row, int col)
        ;

private:
    bool isSafe(int row, int col, int num) const {
        /* ??? */
    }
};

bool solveSudoku(Sudoku &sudoku, int row, int col) {
    /* ??? */
}

int main() {
    int puzzle[9][9] = {
        {5, 3, 0, 0, 7, 0, 0, 0, 0},
        {6, 0, 0, 1, 9, 5, 0, 0, 0},
        {0, 9, 8, 0, 0, 0, 0, 6, 0},
        {8, 0, 0, 0, 6, 0, 0, 0, 3},
        {4, 0, 0, 8, 0, 3, 0, 0, 1},
        {7, 0, 0, 0, 2, 0, 0, 0, 6},
        {0, 6, 0, 0, 0, 0, 2, 8, 0},
        {0, 0, 0, 4, 1, 9, 0, 0, 5},
        {0, 0, 0, 0, 8, 0, 0, 7, 9}
    };

    Sudoku sudoku(puzzle);
    cout << "Original Sudoku:\n";
    sudoku.display();

    if (solveSudoku(sudoku, 0, 0)) {
        cout << "\nSolved Sudoku:\n";
        sudoku.display();
    } else {
        cout << "\nNo solution exists.\n";
    }

    return 0;
}
```

Next task: Alter the code to handle the general case of any $n^2 * n^2$ Sudoku.

Problem 3

Below are two implementations of a function that finds the nth Fibonacci number.

```
int fib(int n) {
    if (n == 1) return 0;
    else if (n == 2) return 1;
    else return fib(n - 2) + fib(n - 1);
}
```

```
int fib(int n) {
    if (n == 0) return 0;
    else if (n == 1) return 1;
    int x = 0, y = 1;
    for (int i = 2; i < n; i++) {
        int r = x + y;
        x = y;
        y = r;
    }
    return y;
}
```

What are the time complexities of the two algorithms? Which is more efficient?

Problem 4

Arrange the following functions in increasing order of their time complexities.

1. $f_1 = 3n + \log(n)$
2. $f_2 = 2^n + 3^n$
3. $f_3 = n^{1.5}$
4. $f_4 = n\log(n)$
5. $f_5 = \log(n!)$
6. $f_6 = n^2 + 2n + 1$

Problem 5

What is the time complexity of the function `b()`?

```
int a(int n) {
    for (int i = 0; i < n; i++) cout << "Hello!\n";
    a(n / 2);
}

int b(int n) {
    for (int i = 1; i <= n; i += i) a(i);
}
```