Задание 1.

```cpp
#include <iostream>
#include <vector>
#include <tuple>
#include <deque>
enum ActionType { // варианты операций
    INSERT,
    DELETE,
    EDIT
};
class Action { // объект, который будет хранить информацию об операции
public:
    Action() {}
    Action(ActionType t, int p, char v) : type(t), position(p), value(v) {}
    void print() { // распечатка
        switch (type) {
            case INSERT:
            std::cout << "INSERT";
                break;
            case DELETE:
            std::cout << "DELETE";
                break;
            case EDIT:
            std::cout << "EDIT";
                break;
        }
        std::cout << " " << value << " at position " << position << std::endl;
    }
    ActionType type; // тип
    int position; // позиция изменения
    char value; // измененное значение
};
std::tuple<int, std::vector<Action> > Levenstain(std::string S1, std::string S2)
{
    S1 = " " + S1;
    S2 = " " + S2;
    auto minVal = [](int a, int b, int c) {return std::min(a, std::min(b, c));};
// минимум из трех
    int M = S1.size();
    int N = S2.size();
    std::vector< std::vector<int> > D(M+1, std::vector<int>(N+1)); // матрица
    D[0][0] = 0;
    for(int i = 0; i <= M; i++) D[i][0] = i;
    for(int j = 0; j <= N; j++) D[0][j] = j;
    for(int i = 1; i <= M; i++) {
        for(int j = 1; j <= N; j++) {
            if(S1[i] == S2[j]){
                D[i][j] = D[i-1][j-1];
            }
            else {
                D[i][j] = minVal(
                        D[i-1][j],
                        D[i][j-1],
                        D[i-1][j-1]
                ) + 1;
            }
        }
    }
    std::deque<Action> actions;
```

```
    int i = M;
    int j = N;
    while(i > 0 or j > 0) {
        int minimum = minVal(D[i-1][j], D[i][j-1], D[i-1][j-1]);
        if(minimum == D[i-1][j]) { // определяем по минимуму операцию и
дальнейший план дейатвий
            actions.push_front(Action(DELETE, i, S1[i]));
            i--;
        } else if(minimum == D[i][j-1]) {
            actions.push_front(Action(INSERT, j, S2[j]));
            j--;
        } else {
            if(D[i-1][j-1] != D[i][j]) {
                actions.push_front(Action(EDIT, i, S2[j]));
            }
            j--;
            i--;
        }
    }
    return std::make_tuple(D[M][N], std::vector<Action>(actions.begin(),
actions.end()));
}
int main() {
    std::vector<Action> acts;
    int lev;
    std::tie(lev, acts) = Levenstain("Hello", "Lehho");
    for(auto e : acts) e.print();
    std::cout << std::endl;
    std::tie(lev, acts) = Levenstain("POLYNOMIAL", "EXPONENTIAL");
    for(auto e : acts) e.print();
    return 0;
}
```

Задание 2.

Также как и в первом задании будем восстанавливать количество операций. Однако теперь добавим еще переход в D[i-2][j-2], который бы означал применение транспозиции.

Далее остается только сверится с заданными в задании условиями.

```
int main() {
    std::string password, attempt;
    std::cin >> password >> attempt;
    std::string S1 = "  " + attempt, S2 = "  " + password;
    auto minVal3 = [](int a, int b, int c) {return std::min(a, std::min(b,
c));}; // минимум из трех
    auto minVal4 = [](int a, int b, int c, int d) {return std::min(std::min(a,
b), std::min(c, d));}; // из четырех
    int M = S1.size();
    int N = S2.size();
    std::vector< std::vector<int> > D(M+1, std::vector<int>(N+1)); // матрица
    D[0][0] = 0;
    for(int i = 0; i <= M; i++) D[i][0] = i;
    for(int j = 0; j <= N; j++) D[0][j] = j;
    for(int i = 1; i <= M; i++) {
        for(int j = 1; j <= N; j++) {
            if(i > 1 and j > 1 and S1[i-1] == S2[j] and S1[i] == S2[j-1]) { //
возможна транспозиция
```

```
                D[i][j] = minVal4(
                        D[i-1][j] + 1,
                        D[i][j-1] + 1,
                        D[i-1][j-1] + (int)(S1[i] != S2[j]), // добавим 1
операцию, если не совпали и нужно заменить
                        D[i-2][j-2] + 1
                );
            } else { // классический левенштейн
                if(S1[i] == S2[j]){
                    D[i][j] = D[i-1][j-1];
                }
                else {
                    D[i][j] = minVal3(
                            D[i-1][j],
                            D[i][j-1],
                            D[i-1][j-1]
                    ) + 1;
                }
            }
        }
    }
    int i = M;
    int j = N;
    int deleted = 0, inserted = 0, edited = 0, transposed = 0;
    while(i > 1 or j > 1) {
        int minimum = minVal4(D[i-1][j], D[i][j-1], D[i-1][j-1], D[i-2][j-2]);
        if(minimum == D[i-1][j]) { // определяем по минимуму операцию и
дальнейший план дейатвий
            deleted++;
            i--;
        } else if(minimum == D[i][j-1]) {
            inserted++;
            j--;
        } else if(minimum == D[i-1][j-1]){
            if(D[i-1][j-1] != D[i][j]) {
                edited++;
            }
            j--;
            i--;
        } else {
            transposed++;
            i-=2;
            j-=2;
        }
    }
    if(attempt.size() < 7) {
        if(deleted + inserted + edited <= 1 and transposed <= 1) {
            std::cout << "ACCESS ALLOWED" << std::endl;
        } else {
            std::cout << "ACCESS DENIED" << std::endl;
        }
    } else {
        if(deleted + inserted + edited <= 2 and transposed <= 3) {
            std::cout << "ACCESS ALLOWED" << std::endl;
        } else {
            std::cout << "ACCESS DENIED" << std::endl;
        }
    }
    return 0;
```

```
}
```