Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Лабораторная работа
«Вероятностное моделирование метрических характеристик программ»

Выполнил:
Студент группы ИП-113
Шпилев Д. И.
Работу проверил:
старший преподаватель кафедры ПМиК
Агалаков А.А.

Новосибирск 2024 г.

Содержание

1. Задание

1. Написать подпрограммы на двух языках программирования для решения следующих задач:

ЗАДАЧА
1. Отыскать минимальный элемент одномерного массива целых, его значение и значение его индекса.
2. Сортировка одномерного массива в порядке возрастания методом пузырька.
3. Бинарный поиск элемента в упорядоченном одномерном массиве.
4. Отыскать минимальный элемент двумерного массива целых, его значение и значение его индексов.
5. Осуществить перестановку значений элементов одномерного массива в обратном порядке.
6. Осуществлять циклический сдвиг элементов одномерного массива на заданное число позиций влево.
7. Заменить все вхождения целочисленного значения в целочисленный массив.

2. Для каждой подпрограммы вычислить следующие метрические характеристики:

- η^*_2 – число единых по смыслу входных и выходных параметров, представленных в сжатой без избыточной форме;
- η_1 - число отдельных операторов;
- η_2 - число отдельных операндов;
- η - длина словаря реализации;
- N_1 - общее число вхождений всех операторов в реализацию;
- N_2 - общее число вхождений всех операндов в реализацию;
- N - длина реализации;
- $N^{\wedge}$ - предсказанная длина реализации по соотношению Холстеда;
- V^* - потенциальный объем реализации:

$$V^* = (2+\eta^*_2) * \log_2(2+\eta^*_2).$$

- V - объем реализации:

$$V = N * \log_2 \eta.$$

- L - уровень программы через потенциальный объем:

$$L = V^* / V.$$

- $L^{\wedge}$ - уровень программы по реализации:

$$L^{\wedge} = (2/\eta_1) * (\eta_2 / N_2).$$

- I - интеллектуальное содержание программы:

$$I = (2/\eta_1) * (\eta_2 / N_2) * (N_1 + N_2) * \log_2(\eta_1 + \eta_2).$$

- $T^{\wedge}_1$ - прогнозируемое время написания программы, выраженное через потенциальный объем:

$$T = \frac{V^2}{S * V^*}.$$

- $T^{\wedge}_2$ - прогнозируемое время написания программы, выраженное через длину реализации, найденную по Холстеду (т.е. в предположении, что программа совершенна):

$$T = \frac{\eta_1 \times N_2 \times (\eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2) \times \log_2 \eta}{2 \times S \times \eta_2}.$$

- $T^{\wedge}_3$ - прогнозируемое время написания программы, выраженное через метрические характеристики реализации:

$$T = \frac{\eta_1 \times N_2 \times N \times \log_2 \eta}{2 \times S \times \eta_2}.$$

3. По всем реализациям алгоритмов определить средние значения уровней языков программирования λ:

$$\lambda_1 = L^{\wedge 2} \times V,$$

$$\lambda_2 = \frac{V^{*2}}{V}.$$

2 Код программы

```cpp
#include <iostream>
#include <vector>
#include <limits>

std::pair<int, int> findMin(const std::vector<int>& vec){
    if (vec.empty()) return {std::numeric_limits<int>::max(), -1};
    int mmin = vec[0];
    int index = 0;
    for(int i = 1; i < vec.size(); ++i){
        if (vec[i] < mmin){
            mmin = vec[i];
            index = i;
        }
    }
    return {mmin, index};
}

void bubbleSort(std::vector<int>& vec){
    for(int i = 0; i < vec.size() - 1; ++i){
        for(int j = vec.size() - 1; j > i; --j){
            if (vec[j] < vec[j-1]) {
                int temp = vec[j];
                vec[j] = vec[j-1];
                vec[j-1] = temp;
            }
        }
    }
}

int binarySearch(const std::vector<int>& vec, int target){
    int l = 0;
    int r = vec.size()-1;
    while(l <= r){
        int mid = (l+r)/2;
        if (vec[mid] == target) return mid;
        if (vec[mid] < target) l = mid + 1;
        else r = mid - 1;
    }
    return -1;
}

std::tuple<int, int, int> findMin(const std::vector<std::vector<int>>& vec){
    if (vec.empty() || vec[0].empty()) return {std::numeric_limits<int>::max(), -
1, -1};
    int mmin = vec[0][0];
    int indexi = 0;
    int indexj = 0;
    for(int i = 0; i < vec.size(); ++i){
        for(int j = 0; j < vec[i].size(); ++j){
```

```
            if (vec[i][j] < mmin){
                mmin = vec[i][j];
                indexi = i;
                indexj = j;
            }
        }
    }
    return {mmin, indexi, indexj};
}

void reverseVector(std::vector<int>& vec) {
    for(int i = 0, j = vec.size()-1; i <= j; ++i, --j){
        int temp = vec[i];
        vec[i] = vec[j];
        vec[j] = temp;
    }
}

void cyclicShift(std::vector<int>& vec, int positions) {
    positions %= vec.size();

    if (positions == 0) return;

    std::vector<int> temp(vec.size());

    for (int i = 0; i < vec.size(); ++i) {
        temp[i] = vec[(i + positions) % vec.size()];
    }

    vec = temp;
}

void replaceValue(std::vector<int>& arr, int oldValue, int newValue) {
    for (auto& value : arr) {
        if (value == oldValue) {
            value = newValue;
        }
    }
}
```

3. Результаты

η^*_2	η_1	η_2	η	N_1	N_2	N	$N^\wedge$	V^*	V	L	$L^\wedge$	I	$T^\wedge$	$T^\wedge$	$T^\wedge$	λ_1	λ_2
2	8	8	16	17	23	40	48	8	160	0.05	0.0869565	13.913	320	220.8	184	1.20983	0.4
1	8	7	15	24	25	49	43.6515	4.754889	191.438	0.0248378	0.07	13.4006	770.752	243.631	273.482	0.938044	0.118101
3	12	9	21	21	25	46	71.5489	11.6096	202.047	0.0574602	0.06	12.1228	351.629	523.776	336.744	0.727368	0.667092
2	10	10	20	29	41	70	66.4386	8	302.535	0.0264432	0.0487805	14.7578	1144.09	588.643	620.197	0.719893	0.211546
1	7	6	13	14	19	33	35.1613	4.754889	122.115	0.0389379	0.0902256	11.0179	313.613	144.208	135.344	0.994092	0.185146
2	11	6	17	14	19	33	53.5635	8	134.886	0.0593092	0.0574163	7.74467	227.429	381.319	234.927	0.44467	0.474474
3	6	4	10	6	6	12	23.5098	11.6096	39.8631	0.291238	0.222222	8.85847	13.6875	35.144	17.9384	1.96855	3.38116

4. Вывод

Лабораторная работа позволила глубже понять процесс разработки программных модулей на разных языках программирования, а также оценить их сложность и прогнозируемое время разработки с помощью метрик Холстеда. Методы оценки сложности программного кода являются полезными инструментами для оценки эффективности программного обеспечения, а также для планирования времени разработки и оптимизации кода.