МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 9

Тема:

Рекурсивные алгоритмы и их реализация

Выполнил студент Цемкало А. Р.

Фамилия И. О.

группа ИКБО-10-20

Москва 2021

СОДЕРЖАНИЕ

Ответы на вопросы 3
1.1. Условие задачи 4
1.2. Постановка задачи 4
1.3. Описание алгоритма – рекуррентная зависимость 4
1.4. Коды используемых функций 4
1.5. Ответы на задания по задаче 1: список требований к задаче 1 5
1.6. Код программы и скриншоты результатов тестирования 8
2.1. Условие задачи 9
2.2. Постановка задачи 9
2.3. Описание алгоритма – рекуррентная зависимость 9
2.4. Коды используемых функций 9
2.5. Ответы на задания по задаче 2: список требований к задаче 2 10
2.6. Код программы и скриншоты результатов тестирования 11
ВЫВОДЫ 13
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 14

Ответы на вопросы

1. Подпрограмму (функцию) называют рекурсивной, если хотя бы одна конструкция содержит имя этой подпрограммы.

2. Шаг рекурсии – это активизация очередного рекурсивного выполнения алгоритма при других исходных данных.

3. Глубина рекурсивных вызовов – наибольшее одновременное количество рекурсивных вызовов функции, определяющее максимальное количество слоев рекурсивного стека, в котором осуществляется хранение отложенных вычислений.

4. Условие выхода из рекурсии - определяет завершение рекурсии и формирование конкретного простейшего решения вычислительного процесса.

5. Виды рекурсии

Линейная рекурсия: каждый вызов порождает ровно один новый вызов.

Каскадная рекурсия: каждый вызов порождает несколько новых вызовов.

Повторная рекурсия: если в линейно-рекурсивном определении функции, во всех рекурсивных вызовах на ветвях различия случаев рекурсивный вызов является самым внешним.

Удаленная рекурсия: если в определении рекурсивной функции вызов функции в списке фактических параметров содержит вызов самой функции.

6. Различают виды рекурсии по организации обращения к подзадачам:

Косвенная рекурсия имеет место, если алгоритм А вызывает алгоритм В, и алгоритм В вновь вызывает алгоритм А.

Прямая рекурсия имеет место, если решение задачи сводится к разделению ее на меньшие подзадачи, выполняемые с помощью одного и того же алгоритма.

7. При каждом новом рекурсивном вызове функции в стеке создается новое множество локальных переменных и форменных параметров, их имена одинаковы, но они имеют различные значения.

Цель. Получить знания и практические навыки по разработке и реализации рекурсивных процессов.

Вариант 11.

Задание. Разработать и протестировать рекурсивные функции в соответствии с задачами варианта.

Отчёт по задаче 1

1.1. Условие задачи

Вычислить x1(x2+x3)(x4+x5+x6)....(x46+x47+...+x55).

1.2. Постановка задачи

Привести итерационный алгоритм решения задачи; реализовать алгоритм в виде функции и отладить его; определить теоретическую сложность алгоритма; описать рекуррентную зависимость в решении задачи; реализовать и отладить рекурсивную функцию решения задачи; определить глубину рекурсии, изменяя исходные данные; определить сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии; привести для одного из значений схему рекурсивных вызовов.

1.3. Описание алгоритма – рекуррентная зависимость

$$f(\,) = \begin{cases} s * f(j+1) \text{ если } j < 10 \\ s \text{ если } j = 10 \\ 1 \text{ иначе} \end{cases}$$

1.4. Коды используемых функций

```
long long it_count(int a[55]) {
	int i = 0, j = 0;
	long long result = 1, s;
	while (i < 55) {
		s = 0;
		j++;
		for (int k = 0; k < j; k++) {
			s += a[i++];
		}
		result *= s;
	}
	return result;
}
```

```
long long count(int i, int j, int a[55]) {
	long long s = 0;
	for (int k = 0; k < j; k++) {
		s += a[i++];
	}
	if (j < 10) {
		return s * count(i, j+1, a);
	}
	else if (j == 10) {
		return s;
	}
	else {
		return 1;
	}
}
```

1.5. Ответы на задания по задаче 1: список требований к задаче 1

1.5.1. Приведите итерационный алгоритм решения задачи.

Программа реализована благодаря двум циклам. Вложенный считает произведение, внешний – сумму этих произведений.

1.5.2. Реализуйте алгоритм в виде функции и отладьте его.

```
long long it_count(int a[55]) {
	int i = 0, j = 0;
	long long result = 1, s;
	while (i < 55) {
		s = 0;
		j++;
		for (int k = 0; k < j; k++) {
			s += a[i++];
		}
		result *= s;
	}
	return result;
}
```

1.5.3. Определите теоретическую сложность алгоритма.

Теоретическая сложность $O(n^2)$, так как используются два цикла, один из которых вложенный.

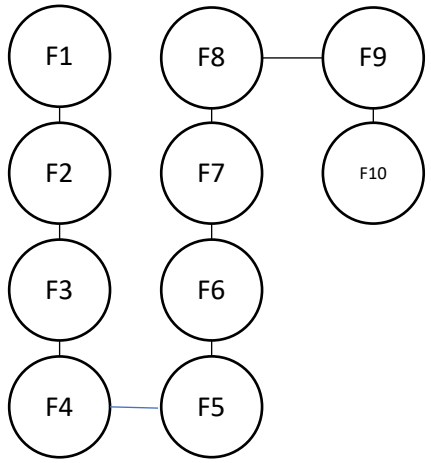
1.5.4. Опишите рекуррентную зависимость в решении задачи.

$$f() = \begin{cases} s * f(j+1) \text{ если } j < 10 \\ s \text{ если } j = 10 \\ 1 \text{ иначе} \end{cases}$$

1.5.5. Реализуйте и отладьте рекурсивную функцию решения задачи

```
long long count(int i, int j, int a[55]) {
	long long s = 0;
	for (int k = 0; k < j; k++) s += a[i++];
	if (j < 10) return s * count(i, j+1, a);
	else if (j == 10) return s;
	else return 1;
}
```

1.5.6. Определите глубину рекурсии, изменяя исходные данные



Характеристиками рассматриваемого метода оценки алгоритма будут следующие величины.

D = 10	**D = n**
R(D) = 10	R(D) = fn
$R_V(D) = 9$	$R_V(D) = fn - 1$
$R_L(D) = 1$	$R_L(D) = 1$
$H_R(D) = 9$	$H_R(D) = n - 1$

R(x) – общее число вершин дерева рекурсии,

$R_V(x)$ – объем рекурсии без листьев (внутренние вершины),

$R_L(x)$ – количество листьев дерева рекурсии,

$H_R(x)$ – глубина рекурсии определяется как количество незавершенных входов в рекурсию, т.е. в дереве рекурсии — это количество внутренних вершин (т.е. без листьев).

1.5.7. Определите сложность рекурсивного алгоритма, используя метод подстановки и дерево рекурсии

T(1)
|
T(2)
|
T(3)
|
T(4)
|
T(5)
|
…
|
T(n)

Общее время можно вычислить через сумму арифметической прогрессии: $S_n = \frac{a_1+a_n}{2}n = \frac{1+n}{2}n = \frac{n+n^2}{2}$.

$T(n)=O(n^2)$

1.5.8. Приведите для одного из значений схему рекурсивных вызовов

P(j) ≡ if j = конечное значение

return завершение рекурсии

else

return P(j+1) шаг в рекурсию

1.5.9. Разработайте программу, демонстрирующую выполнение обеих функций, и покажите результаты тестирования

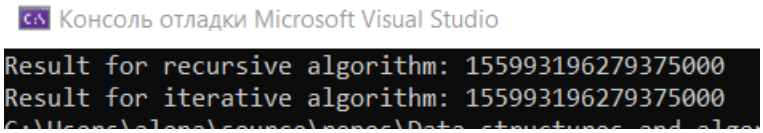
```
#include <iostream>
using namespace std;

long long it_count(int a[55]) {
	int i = 0, j = 0;
	long long result = 1, s;
	while (i < 55) {
		s = 0;
		j++;
		for (int k = 0; k < j; k++) {
			s += a[i++];
		}
		result *= s;
	}
	return result;
}

long long count(int i, int j, int a[55]) {
	long long s = 0;
	for (int k = 0; k < j; k++) s += a[i++];
	if (j < 10) return s * count(i, j+1, a);
	else if (j == 10) return s;
	else return 1;
}

int main() {
	int a[55];
	for (int i = 0; i < 55; i++) {
		a[i] = i + 1;
	}
	cout << "Result for recursive algorithm: " << count(0, 1, a) << endl;
	cout << "Result for iterative algorithm: " << it_count(a);
	return 0;
}
```

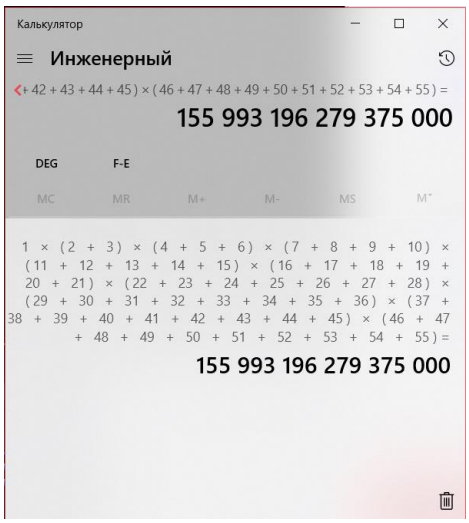
Для i = 55 и j = 10 результат 155993196279375000

Для i = 45 и j = 9 результат 308897418375000

1.6. Код программы и скриншоты результатов тестирования

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

long long it_count(int a[55]) {
	int i = 0, j = 0;
	long long result = 1, s;
	while (i < 55) {
		s = 0;
		j++;
		for (int k = 0; k < j; k++) {
			s += a[i++];
		}
		result *= s;
	}
	return result;
}

long long count(int i, int j, int a[55]) {
	long long s = 0;
	for (int k = 0; k < j; k++) s += a[i++];
	if (j < 10) return s * count(i, j+1, a);
	else if (j == 10) return s;
	else return 1;
}

int main() {
	int a[55];
	for (int i = 0; i < 55; i++) {
		a[i] = i + 1;
	}
	cout << "Result for recursive algorithm: " << count(0, 1, a) << endl;
	cout << "Result for iterative algorithm: " << it_count(a);
	return 0;
	}
```





Отчёт по задаче 2

2.1. Условие задачи

Удаление двунаправленного списка.

2.2. Постановка задачи

Разработать рекурсивную функцию для обработки списковой структуры согласно варианту. Информационная часть узла – простого типа – целого; определить глубину рекурсии; определить теоретическую сложность алгоритма; разработать программу, демонстрирующую работу функций и показать результаты тестов.

2.3. Описание алгоритма – рекуррентная зависимость

$$f(p) = \begin{cases} \text{прекращение работы, если список пустой} \\ \text{прекращение работы, если удалён пустой элемент} \\ p(p \to next) \end{cases}$$

2.4. Коды используемых функций

```
struct Node {
	int value;
	Node* next;
	Node* previous;
	Node(int _value) : next(nullptr), previous(nullptr), value(_value) {}
};

struct list {
	Node* first;
	Node* last;
	list() : first(nullptr), last(nullptr) {}

	bool list_is_empty() {
		return first == nullptr && last == nullptr;
	}

	Node* add(int _value) {
		Node* p = new Node(_value);
		if (list_is_empty()) {
			first = p;
			last = p;
			return p;
		}
		last->next = p;
		p->previous = last;
		last = p;
		return p;
	}

	void print(string direction = "left_to_right") {
		if (list_is_empty()) return;
```

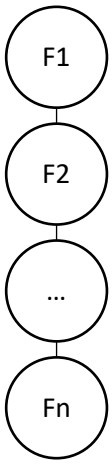
```
            if (direction == "left_to_right") {
                Node* p = first;
                while (p) {
                    cout << p->value << " ";
                    p = p->next;
                }
            }
            else if (direction == "right_to_left") {
                Node* p = last;
                while (p) {
                    cout << p->value << " ";
                    p = p->previous;
                }
            }
        }
};
```

2.5. Ответы на задания по задаче 2: список требований к задаче 2

2.5.1. Приведите рекурсивную функцию для обработки списковой структуры согласно варианту. Информационная часть узла – простого типа – целого.

```
void delete_list(Node* p) {
    if (list_is_empty()) return;
    first = p->next;
    if (p->next != NULL) {
        p->next->previous = p->previous;
    }
    else {
        last = p->previous;
        return;
    }
    delete_list(p->next);
}
L.delete_list(L.first);
```

2.5.2. Определите глубину рекурсии.



Характеристиками рассматриваемого метода оценки алгоритма будут следующие величины.

D = 10	**D = n**
R(D) = 10	R(D) = fn
$R_V(D) = 9$	$R_V(D) = fn - 1$
$R_L(D) = 1$	$R_L(D) = 1$
$H_R(D) = 9$	$H_R(D) = n - 1$

R(x) – общее число вершин дерева рекурсии,

$R_V(x)$ – объем рекурсии без листьев (внутренние вершины),

$R_L(x)$ – количество листьев дерева рекурсии,

$H_R(x)$ – глубина рекурсии определяется как количество незавершенных входов в рекурсию, т.е. в дереве рекурсии — это количество внутренних вершин (т.е. без листьев).

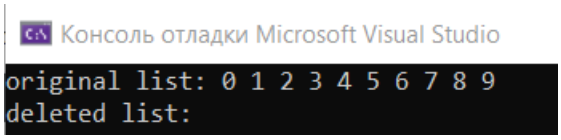
2.5.3. Определите теоретическую сложность алгоритма.

Теоретическая сложность O(n), так как в каждой функции сама функция вызывается не более одного раза (= один цикл).

2.6. Код программы и скриншоты результатов тестирования

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

struct Node {
	int value;
	Node* next;
	Node* previous;
	Node(int _value) : next(nullptr), previous(nullptr), value(_value) {}
};

struct list {
	Node* first;
	Node* last;
	list() : first(nullptr), last(nullptr) {}

	bool list_is_empty() {
		return first == nullptr && last == nullptr;
	}

	Node* add(int _value) {
		Node* p = new Node(_value);
		if (list_is_empty()) {
			first = p;
			last = p;
			return p;
		}
		last->next = p;
		p->previous = last;
		last = p;
		return p;
	}

	void print(string direction = "left_to_right") {
		if (list_is_empty()) return;
		if (direction == "left_to_right") {
			Node* p = first;
			while (p) {
				cout << p->value << " ";
				p = p->next;
			}
		}
		else if (direction == "right_to_left") {
			Node* p = last;
			while (p) {
				cout << p->value << " ";
				p = p->previous;
```

```
                        }
                }
        }

        void delete_list(Node* p) {
                if (list_is_empty()) return;
                first = p->next;
                if (p->next != NULL) {
                        p->next->previous = p->previous;
                }
                else {
                        last = p->previous;
                        return;
                }
                delete_list(p->next);
        }
};

int main() {
        list L;
        for (int i = 0; i < 10; i++) {
                L.add(i);
        }
        cout << "original list: ";
        L.print();
        L.delete_list(L.first);
        cout << endl << "deleted list: ";
        L.print();
        return 0;
}
```



ВЫВОДЫ

В ходе выполнения задания получены знания и практические навыки по разработке и реализации рекурсивных процессов. Разработана рекурсивная функция вычисления x1*(x2+x3)*(x4+x5+x6)*…*(x46+x47+...+x55); Разработана рекурсивная функция удаления двунаправленного списка; Получены навыки в определении теоретической ложности алгоритма, определении глубины рекурсии, определении сложности рекурсивного алгоритма, используя метод подстановки и дерево рекурсии.

Тестирования всех операций пройдены успешно.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Процедурное программирование Языки программирования – Сайт lizochekk! [Электронный ресурс]: URL: https://lizochekk.jimdofree.com/программирование/
2. Документация по Microsoft C/C++ | Microsoft Docs – [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160
3. Все публикации подряд / Хабр – [Электронный ресурс] URL: https://habr.com/ru/
4. Курс: Структуры и алгоритмы обработки данных (Часть 1) Скворцова Л.А. [II.20-21] – [Электронный ресурс] URL: https://online-edu.mirea.ru/course/view.php?id=6600