Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
ТОМСКИЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра комплексной информационной безопасности электронно-вычислительных систем (КИБЭВС)

БИНАРНЫЕ ДЕРЕВЬЯ ПОИСКА

Отчёт по практической работе №5 по дисциплине «Технологии и методы программирования»

Студент гр.728-2
_____ Полонский Е. В.
7 апреля 2020 г.

Руководитель
Аспирант кафедры КИБЭВС
_____ Перминов П. В.
_____

Томск 2020

1 Введение

Целью данной работы является закрепление теоретических знаний об бинарных деревьях поиска, реализация бинарного дерева поиска на практике.

Задание: Написать на языке программирования C программу, позволяющую работать с бинарными деревьями поиска.

2 Ход работы

Для реализации дерева необходимы две структуры:

– tree – непосредственно само дерево, с полями root – корень и count – количеством элементов в дереве;

– node – элемент списка, содержит в себе 4 поля, data – значение элемента, left – указатель на левого потомка и right – указатель на правого потомка и parent – указатель на родителя.

Также для удобства поиска элемента по значению была создана структура find_return с 2 полями: bool exist – true если элемент существует и node *root – указатель на элемент или предшествующий ему элемент, при условии, что exist = false.

Также необходимо реализовать несколько функций и процедур:

– void init(tree *t) – инициализирует дерево;

– void clean(tree *t) – удаляет все элементы из дерева;

– find_return __find_node(node *n, int value) – рекурсивно ищет элемент со значение value, возвращает true, node, если элемент найден или false, prevnode если такого элемента нет, prevnode – это предполагаемый родитель искомого элемента, если бы он существовал;

– node *find(tree* t, int value) – находит и возвращает указатель на элемент со значением value, если элемент не найден возвращает NULL;

– int insert(tree* t, int value) – вставляет элемент в дерево, возвращает 0 если вставка успешна, 1 если элемент существует и 2 если не удалось выделить память для нового элемента;

– void __remove(tree *t, node *n) – удаляет элемент на который указывает n из дерева t;

– int remove_min(tree *t, node* n) – удаляет минимальный элемент из поддерева с корнем n, возвращает значение удаленного элемента;

– int remove_node(tree* t, int value) – удаляет элемент со значением value из дерева, возвращает 0 если удаление успешно, 1 если элемента со значением value нет;

– int rotate_right(tree *t, node* n) – выполняет правое вращение дерева кор-

нем которого является n, возвращает 0 если операция выполнена, 1 если вращение невозможно;

– int rotate_left(tree *t, node* n) – выполняет левое вращение дерева корнем которого является n, возвращает 0 если операция выполнена, 1 если вращение невозможно;

– void print_tree(tree* t) – выводит дерево в stdout;

– void print(node* n) – выводит дерево в stdout поддерево, корнем которого является n.

Для реализации некоторых функций необходима работа с кучей, а именно выделение чанка памяти при добавлении нового элемента в список (процедура – calloc()). И удаление чанка, соответственно при удалении элемента списка (процедура free()).

Для вывода дерева понадобилось работать со строками в C, для этого были использованы такие процедуры как:

– strcpy(str1, str2) – копирует str2 на адрес str1;

– snprintf (char *s, size_t n, const char *format, ...) – аналог printf, только помещает отформатированную строку по адресу s.

Для вывода дерева было необходимо реализовать очередь, в качестве очереди был использован двусвязный список, реализованный в практике №4.

Весь исходный код программы можно посмотреть на гите.

Исходные коды программы были скомпилированы в исполняемый файл при помощи команды:

gcc pr5/01_bst.c pr5/src/list.c

Результат работы программы представлен на рисунке 2.1.

Также программа была проверена на возможные утечки памяти, с помощью утилиты valgrind(Рисунок 2.2).

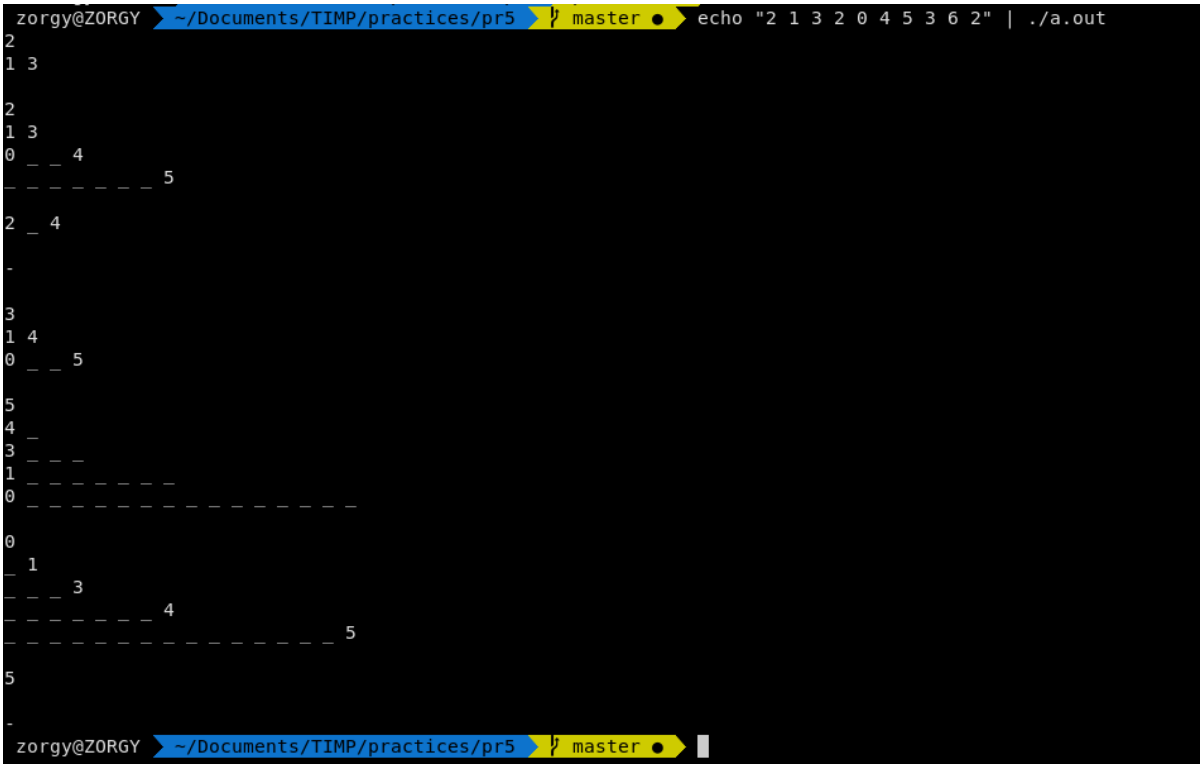

Рисунок 2.1 – Результат работы программы

Рисунок 2.2 – Проверка программы на утечки памяти

Далее исходные коды программ были запушены на гитлаб командой git push. Все пайплайны были пройдены успешно (Рисунок 2.3).

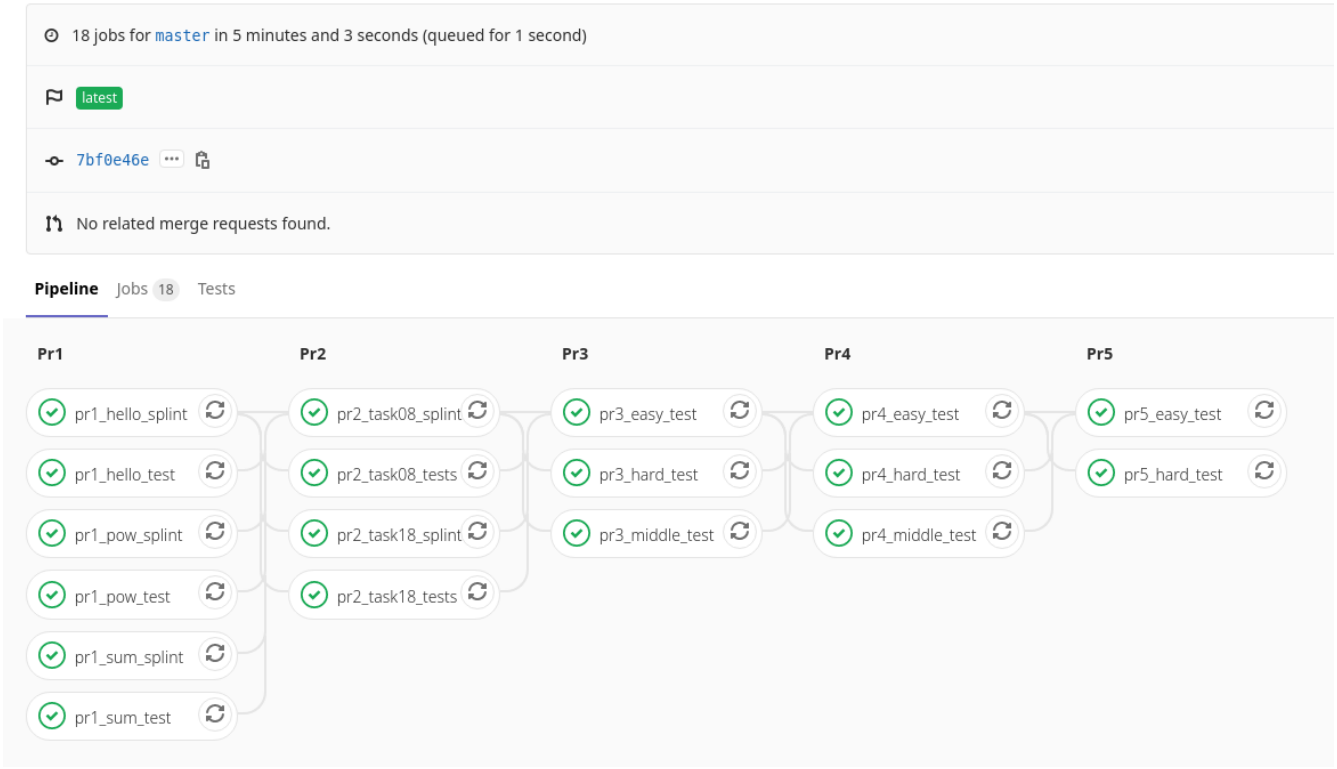

Рисунок 2.3 – Пройденные пайплайны

3 Заключение

В результате выполнения практической работы были закреплены теоретические знания о бинарных деревьях поиска, написана программа, реализующая это дерево.