МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Демонстрация вставки в красно-черное дерево

Студент гр. 9382 _____ Русинов Д.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Русинов Д.А.

Группа 9382

Тема работы: Демонстрация вставки в красно-черное дерево

Исходные данные:
На вход программе подаются элементы, которые необходимо вставить в дерево.

Содержание пояснительной записки:
«Содержание», «Введение», «Основные теоретические положения», «Описание алгоритма», «Описание функций и структур данных», «Описание работы программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи курсовой работы: 22.11.2020

Дата защиты курсовой работы: 22.11.2020

Студент _____ Русинов Д.А.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

В курсовой работе происходит вставка элемента в красно-черное дерево. Программа демонстрирует процесс вставки при помощи вывода на экран состояния элементов на каждом шаге, раскрашивая в другой цвет рассматриваемые элементы. Результатом будет изображение красно-черного дерева после вставки.

SUMMARY

In the course work, an element is inserted into the red-black tree. The program demonstrates the insertion process by displaying the status of elements at each step, coloring the elements in question in a different color. The result will be a red-black tree image after pasting.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы являлось изучение красно-черного дерева. Для этого потребовалось изучить его структуру, алгоритм построения, алгоритм вставки в него, а также придумать визуализацию работы алгоритма. Красно-черное дерево является двоичным деревом поиска, следовательно необходимо было также изучить, что такое двоичное дерево поиска. Результатом является программа, которая считывает элемент и вставляет его в красно-черное дерево, визуализируя работу алгоритма.

1. ЗАДАНИЕ

Вариант 27. Красно-чёрные деревья – вставка. Демонстрация

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

2.1 Двоичное дерево поиска.

Двоичное дерево поиска — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

1) Оба поддерева — левое и правое — являются двоичными деревьями поиска.
2) У всех узлов левого поддерева произвольного узла X значения ключей данных меньше либо равны, нежели значение ключа данных самого узла X.
3) У всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

2.2 Красно-черное дерево.

Красно-чёрное дерево — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

Свойства красно-черного дерева:

1) Узел может быть либо красным, либо черным и имеет двух потомков.
2) Корень – как правило черный.
3) Все листья, не содержащие данных – черные.
4) Оба потомка красного узла – черные.
5) Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число черных узлов.

3. ОПИСАНИЕ АЛГОРИТМА

3.1 Выполнение вставки в дерево.

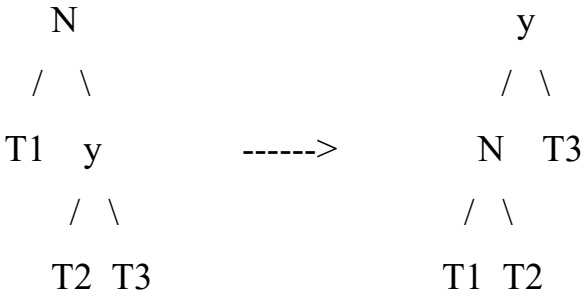
Новый узел в красно-черное дерево добавляется на место одного из листьев и окрашивается в красный цвет.

Если значение вставляемого элемента меньше значения текущей ветки, то следующей веткой будет левая.

Если значение вставляемого элемента больше либо равно значению текущей ветки, то следующей веткой будет правая.

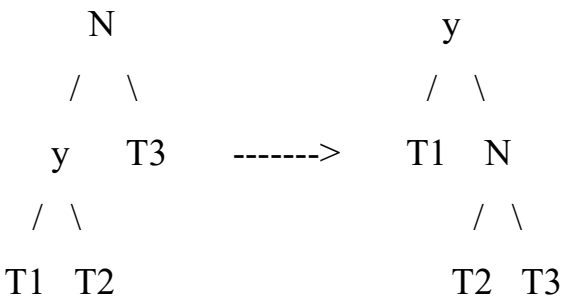
Если следующая ветка пустая, то туда вставляется заданный элемент.

3.2 Выполнение левого поворота дерева N.



В правую ветку N подставляется левая ветка y, в левую ветку y подставляется N.

3.3 Выполнение правого поворота дерева N.



В левую ветку N подставляется правая ветка y, в правую ветку y подставляется N.

3.4 Балансировка дерева после вставки.

Буквой N будем обозначать текущий узел (окрашенный красным). Сначала это новый узел, который вставляется, но эта процедура может применяться рекурсивно к другим узлам. P будем обозначать предка N, через G обозначим дедушку N, а U будем обозначать дядю (узел, имеющий общего родителя с узлом P). В некоторых случаях роли узлов могут меняться, но, в любом случае, каждое обозначение будет представлять тот же узел, что и в начале.

1) Случай 1

Текущий узел N в корне дерева. В этом случае, он перекрашивается в чёрный цвет, чтобы оставить верным Свойство 2 (Корень — чёрный). Так как это действие добавляет один чёрный узел в каждый путь, Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается.

2) Случай 2

Предок P текущего узла чёрный, то есть Свойство 4 (Оба потомка каждого красного узла — чёрные) не нарушается. В этом случае дерево остаётся корректным. Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается, потому что текущий узел N имеет двух чёрных листовых потомков, но так как N является красным, путь до каждого из этих потомков содержит такое же число чёрных узлов, что и путь до чёрного листа, который был заменен текущим узлом, так что свойство остается верным.

3) Случай 3

Если и родитель P, и дядя U — красные, то они оба могут быть перекрашены в чёрный, и дедушка G станет красным (для сохранения свойства 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов)). Теперь у текущего красного узла N чёрный родитель. Так как любой путь через родителя или дядю должен проходить через дедушку,

число чёрных узлов в этих путях не изменится. Однако, дедушка G теперь может нарушить свойства 2 (Корень — чёрный) или 4 (Оба потомка каждого красного узла — чёрные) (свойство 4 может быть нарушено, так как родитель G может быть красным). Чтобы это исправить, вся процедура рекурсивно выполняется на G из случая 1.

4) Случай 4

Родитель P является красным, но дядя U — чёрный. Также, текущий узел N — правый потомок P, а P в свою очередь — левый потомок своего предка G. В этом случае может быть произведен поворот дерева, который меняет роли текущего узла N и его предка P. Тогда, для бывшего родительского узла P в обновленной структуре используем случай 5, потому что Свойство 4 (Оба потомка любого красного узла — чёрные) все ещё нарушено. Вращение приводит к тому, что некоторые пути (в поддереве, обозначенном «1» на схеме) проходят через узел N, чего не было до этого. Это также приводит к тому, что некоторые пути (в поддереве, обозначенном «3») не проходят через узел P. Однако, оба эти узла являются красными, так что Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) не нарушается при вращении. Однако Свойство 4 всё ещё нарушается, но теперь задача сводится к Случаю 5.

5) Случай 5

Родитель P является красным, но дядя U — чёрный, текущий узел N — левый потомок P и P — левый потомок G. В этом случае выполняется поворот дерева на G. В результате получается дерево, в котором бывший родитель P теперь является родителем и текущего узла N и бывшего дедушки G. Известно, что G — чёрный, так как его бывший потомок P не мог бы в противном случае быть красным (без нарушения Свойства 4). Тогда цвета P и G меняются и в результате дерево удовлетворяет Свойству 4 (Оба потомка любого красного узла — чёрные). Свойство 5 (Все пути от любого данного узла до листовых узлов содержат одинаковое число чёрных узлов) также остается верным, так как все пути, которые проходят через любой из этих трех узлов, ранее

проходили через G, поэтому теперь они все проходят через P. В каждом случае, из этих трёх узлов только один окрашен в чёрный.

3.5 Визуализация дерева.

Каждый узел печатается с цветом, соответствующий этому узлу. Узлы, рассматриваемые в данный момент алгоритмом, выделяются отличным цветом от основных цветов.

4. ОПИСАНИЕ ФУНКЦИЙ И СТРУКТУР ДАННЫХ

1) Структура узла красно-черного дерева Node.

Была создана структура узла красно-черного дерева. Данная структура содержит в себе следующие поля:

Elem data – данные, содержащиеся в узле. Elem – тип элемента.

Node* left – левая ветка красно-черного дерева.

Node* right – правая ветка красно-черного дерева.

Node* parent – указатель на родителя узла.

bool color – цвет узла. Если true, то узел черный, иначе красный.

2) Класс красно-черного дерева RBTree.

Был создан класс красно-черного дерева. Данный класс содержит в себе поле с указателем на корень: Node<Elem>* root. Elem – тип элемента. Класс включает в себя следующие методы.

3) Описание конструктора RBTree().

При помощи него можно создавать экземпляры данного класса. Конструктор задает в качестве корня nullptr. Данный конструктор является публичным.

4) Описание метода Node<Elem>* RBTree::getGrandparent(Node<Elem>* n).

При помощи данного метода можно получить дедушку переданного узла. Данный метод на вход получает узел n, дедушку которого нужно вернуть. Если дедушки нет, то вернется nullptr. Данный метод является приватным.

5) Описание метода Node<Elem>* RBTree::getUncle(Node<Elem>* n).

При помощи данного метода можно получить дядю переданного узла. Данный метод на вход получает узел n, дядю которого нужно вернуть. Если дяди нет, то вернется nullptr. Данный метод является приватным.

6) Описание метода void RBTree::restoreRoot(Node<Elem>* n).

При балансировке красно-черного дерева после вставки элемента мог изменить корень в результате операций поворота. В этом случае необходимо восстановить корень. Данный метод ищет в дереве новый корень и устанавливает его в качестве root. Данный метод является приватным.

7) Описание метода void RBTree::leftRotate(Node<Elem>* n).

Данный метод выполняет левый поворот переданного узла n в дереве. Операция описывается следующим образом: в правую ветку заданного узла подставляется левая ветка правого сына n, а затем n подставляется в левую ветку правого сына n. Данный метод является приватным.

8) Описание метода void RBTree::rightRotate(Node<Elem>* n).

Данный метод выполняет правый поворот переданного узла n в дереве. Операция описывается следующим образом: в левую ветку заданного узла подставляется правая ветка левого сына n, а затем n подставляется в правую ветку левого сына n. Данный метод является приватным.

9) Описание метода void RBTree::insertCase1(Node<Elem>* n).

Данный метод является началом балансировки красно-черного дерева после вставки элемента. В нем проверяется, есть ли родитель у вставленного элемента n. Если его нет, значит n является корнем дерева, в таком случае балансировка окончена, и по свойству красно-черного дерева данный узел перекрашивается в черный цвет. Если родитель есть, значит необходимо выполнить метод insertCase2 над этим узлом. Данный метод является приватным. Метод вызывается после вставки элемента в дерево или в методе insertCase3.

10) Описание метода void RBTree::insertCase2(Node<Elem>* n).

Данный метод является одним из этапов балансировки красно-черного дерева после вставки в него элемента n. Вызов этого метода происходит только

из метода insertCase1. В данном методе выполняется проверка на цвет родителя узла n. Если цвет родителя является черным, то никакие свойства красно-черного дерева не нарушаются, в этом случае балансировка окончена. В ином случае необходимо вызвать метод insertCase3 над этим узлом. Данный метод является приватным.

11) Описание метода void RBTree::insertCase3(Node<Elem>* n).

Данный метод является одним из этапов балансировки красно-черного дерева после вставки в него элемента n. Вызов этого метода происходит только из метода insertCase2. В данном методе выполняется проверка на наличие красного дяди у узла n. Если красный дядя есть, то родитель n становится черным, дядя n становится черным, а дедушка n становится красным, затем происходит операция балансировки уже для дедушки, поскольку родитель дедушки тоже мог быть красным, вызывается метод insertCase1 для дедушки. В ином случае необходимо вызвать метод insertCase4 над этим узлом n. Данный метод является приватным.

12) Описание метода void RBTree::insertCase4(Node<Elem>* n).

Данный метод является одним из этапов балансировки красно-черного дерева после вставки в него элемента n. Вызов этого метода происходит только из метода insertCase3. В данном методе проверяются два случая:

А) Является ли n правым сыном, а отец n левым сыном. В этом случае выполняется левый поворот отца n. После левого поворота отца, n становится левым сыном, и его отец становится левым сыном.

Б) Является ли n левым сыном, а отец n правым сыном. В этом случае выполняется правый поворот отца n. После правого поворота отца, n становится правым сыном, и его отец становится правым сыном.

Далее происходит переход в метод insertCase5. Данный метод является приватным.

13) Описание метода void RBTree::insertCase5(Node<Elem>* n).

Данный метод является одним из этапов балансировки красно-черного дерева после вставки в него элемента n. Вызов этого метода происходит только

из метода insertCase4. В данном методе отец n перекрашивается в черный цвет, а дедушка n перекрашивается в красный цвет. В этом методе проверяются затем два случая:

А) Является ли n левым сыном и его отец левым сыном. В данном случае выполняется правый поворот дедушки n.

Б) Является ли n правым сыном и его отец правым сыном. В данном случае выполняется левый поворот дедушки n

После выполнения операции поворота балансировка окончена. Данный метод является приватным.

14) Описание деструктора ~RBTree().

Деструктор вызывает деструктор поля root. Деструктор является публичным.

15) Описание метода Node<Elem>* RBTree::getRoot().

Данный метод возвращает корень дерева. Метод является публичным.

16) Описание метода void RBTree::insert(Elem stuff).

Данный метод позволяет выполнить вставку элемента в красно-черное дерево. На вход метод принимает элемент, где Elem – тип элемента. Вставка выполняется следующим образом:

А) Если значение вставляемого элемента больше либо равно текущему рассматриваемому узлу, то следующим рассматриваемым узлом становится узел в правой ветке текущего узла.

Б) Если значение вставляемого элемента меньше текущего рассматриваемого узла, то следующим рассматриваемым узлом становится узел в левой ветке текущего узла.

В) Если текущий рассматриваемый узел пустой, то в этот узел вставляется переданный элемент stuff, этот элемент имеет красный цвет.

После вставки элемента выполняется балансировка дерева. Вызывается метод insertCase1, в него передается только что вставленный узел. После

балансировки вызывается метод restoreRoot, в него передается вставленный узел.

17) Описание функции void printTree(Node<int>* tree, int level, std::vector<Node<int>*> specialNodes).

Функция используется для печати дерева. Данная функция получает на вход узел tree, с которого необходимо начать печать дерева, уровень рекурсии и вектор “специальных узлов”. Печать выполняется рекурсивно по алгоритму ПКЛ. Узлы печатаются в соответствии с их цветом. “Специальные узлы” окрашиваются в белый цвет.

18) Описание функции void waitNextStep().

Данная функция используется для ожидания продолжения выполнения алгоритма пользователем. Чтобы продолжить выполнение алгоритма, пользователю необходимо ввести любой символ. После ввода символа очищается текущий вывод и продолжается работа алгоритма.

19) Описание функции char userInput(RBTree<int>* tree).

Данная функция используется для ввода пользователем элемента, который необходимо вставить в красно-черное дерево. На вход подается дерево, в которое нужно вставлять элемент. Возвращается символ, который говорит, нужно ли продолжать вставку или нет.

5. ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

Программа приветствует пользователя. Для начала работы пользователю необходимо ввести предложенный символ. Затем пользователю предлагается ввести элемент, который нужно вставить в красно-черное дерево. После ввода пользователем элемента, начинается выполнение алгоритма вставки. Алгоритм на различных этапах останавливается, чтобы пользователь мог изучить, каким образом происходит вставка. Каждая остановка сопровождается пояснениями и визуализацией дерева. Чтобы продолжить исполнение алгоритма, пользователь должен ввести символ. После окончания работы алгоритма пользователь может выбрать, продолжать ввод элементов или нет.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы было изучено красно-черное дерево, бинарное дерево поиска. Была изучена структура красно-черного дерева, алгоритм вставки в него, а также визуализирована работа алгоритма. Была написана программа, которая считывает элемент, вводимый пользователем. Вставляет считанный элемент и визуализирует его вставку.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

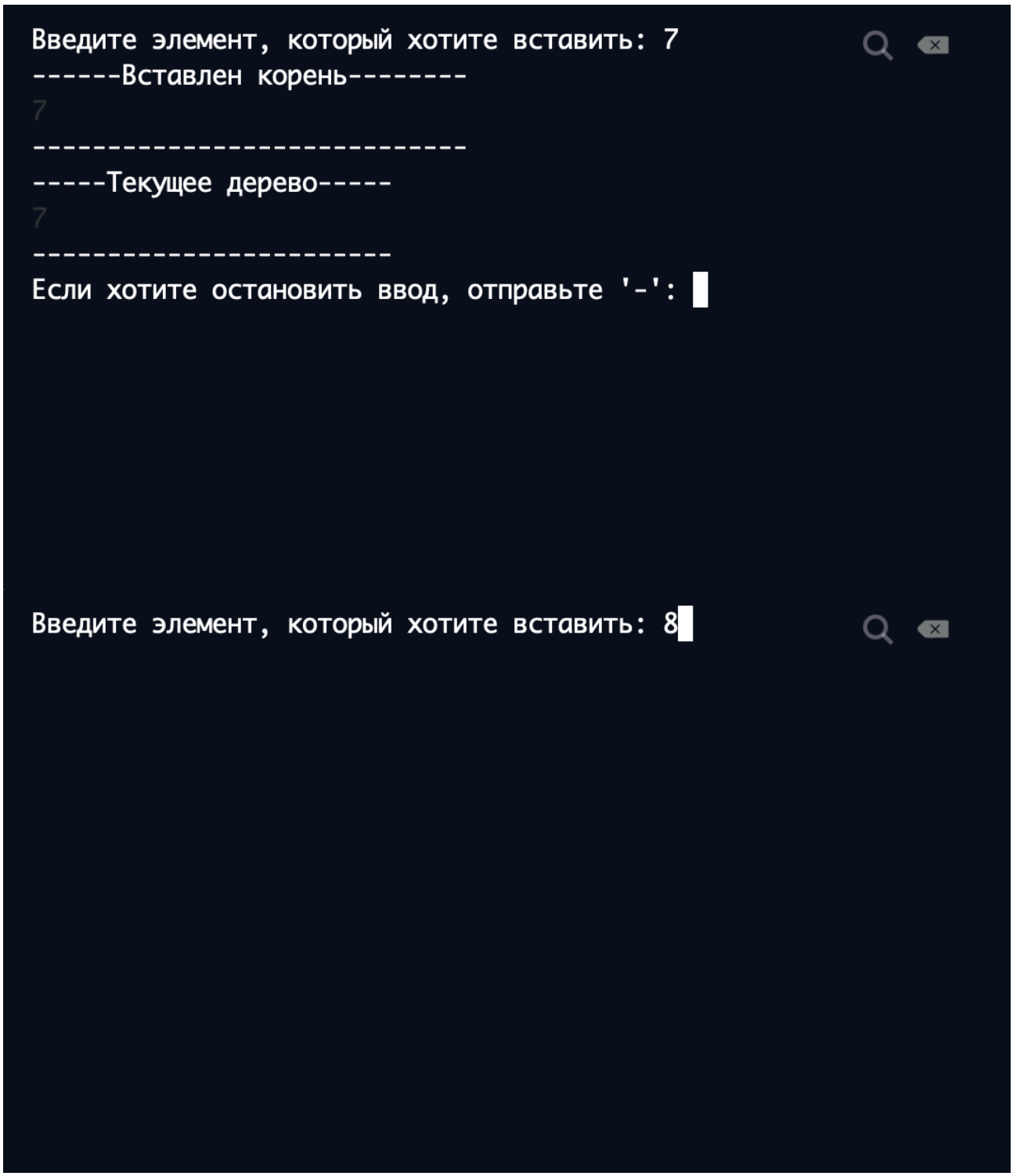
1. Wikipedia. URL: https://ru.wikipedia.org/wiki/Красно-чёрное_дерево (дата обращения: 15.11.2020)
2. Habr. URL: https://habr.com/ru/post/330644/ (дата обращения: 15.11.2020)
3. Habr. URL: https://habr.com/ru/company/otus/blog/472040/ (дата обращения: 15.11.2020)
4. Neerc.ifmo.ru. URL: https://neerc.ifmo.ru/wiki/index.php?title=Красно-черное_дерево (дата обращения: 15.11.2020)
5. Algolist.ru. URL: http://algolist.ru/ds/rbtree.php (дата обращения: 15.11.2020)

ПРИЛОЖЕНИЕ А
ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

1. Проверка на корректных данных

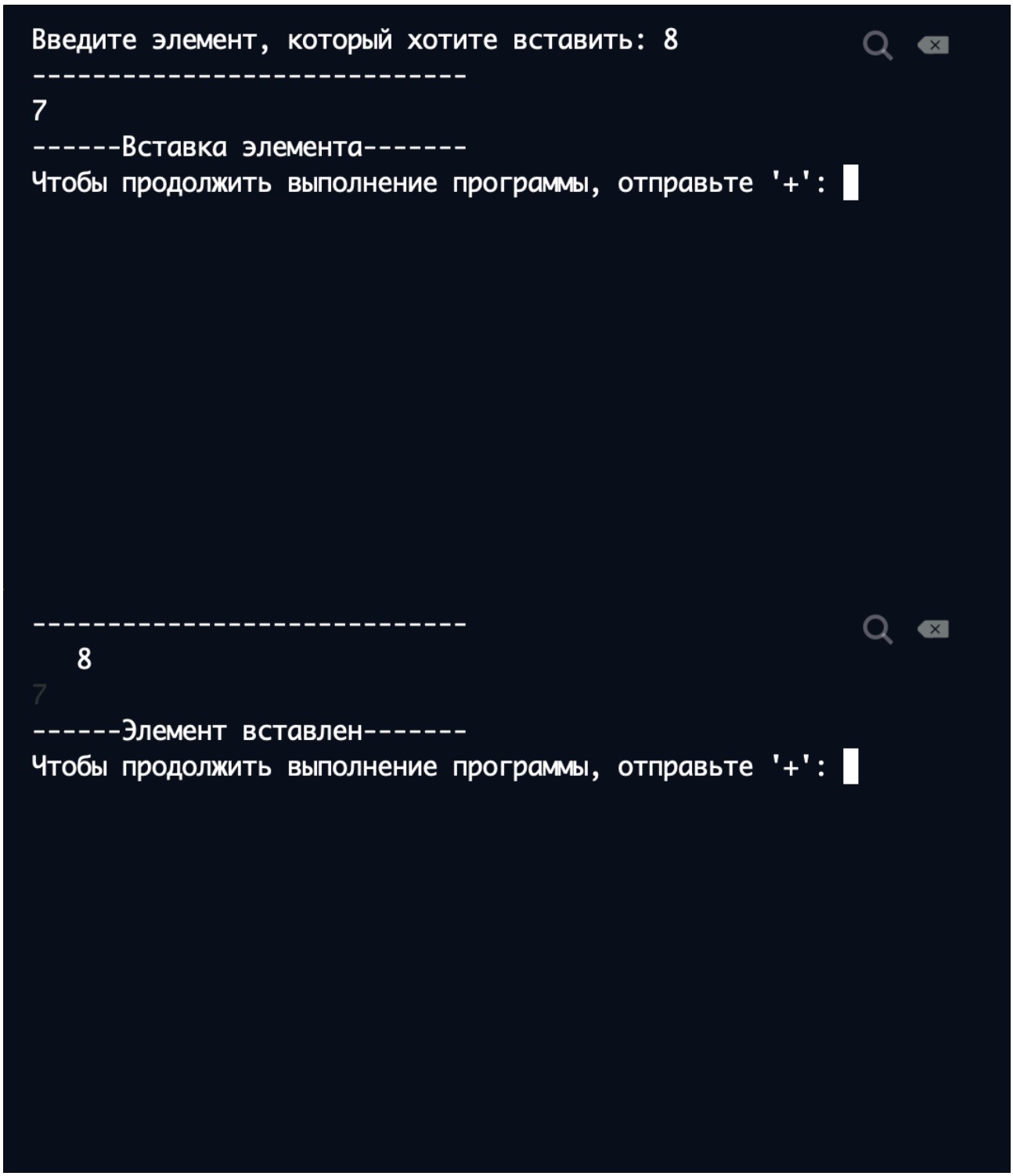
```
clang++-7 -pthread -std=c++17 -o main main.cpp
./main
 ------------------------------------------------
| Здравствуйте, вы попали в курсовую работу студента |
|             Русинова Д.А. группы 9382            |
 ------------------------------------------------
|    Демонстрация вставки в красно-черное дерево    |
 ------------------------------------------------
|   Красным цветом выделяются красные элементы КЧД  |
|    Черным цветом выделяются черные элементы КЧД   |
|  Белым цветом выделяются рассматриваемые элементы  |
|             в ходе выполнения алгоритма           |
 ------------------------------------------------
Для начала работы, введите символ '+': +

Введите элемент, который хотите вставить: 7
```


Введите элемент, который хотите вставить: 7
------Вставлен корень--------
7

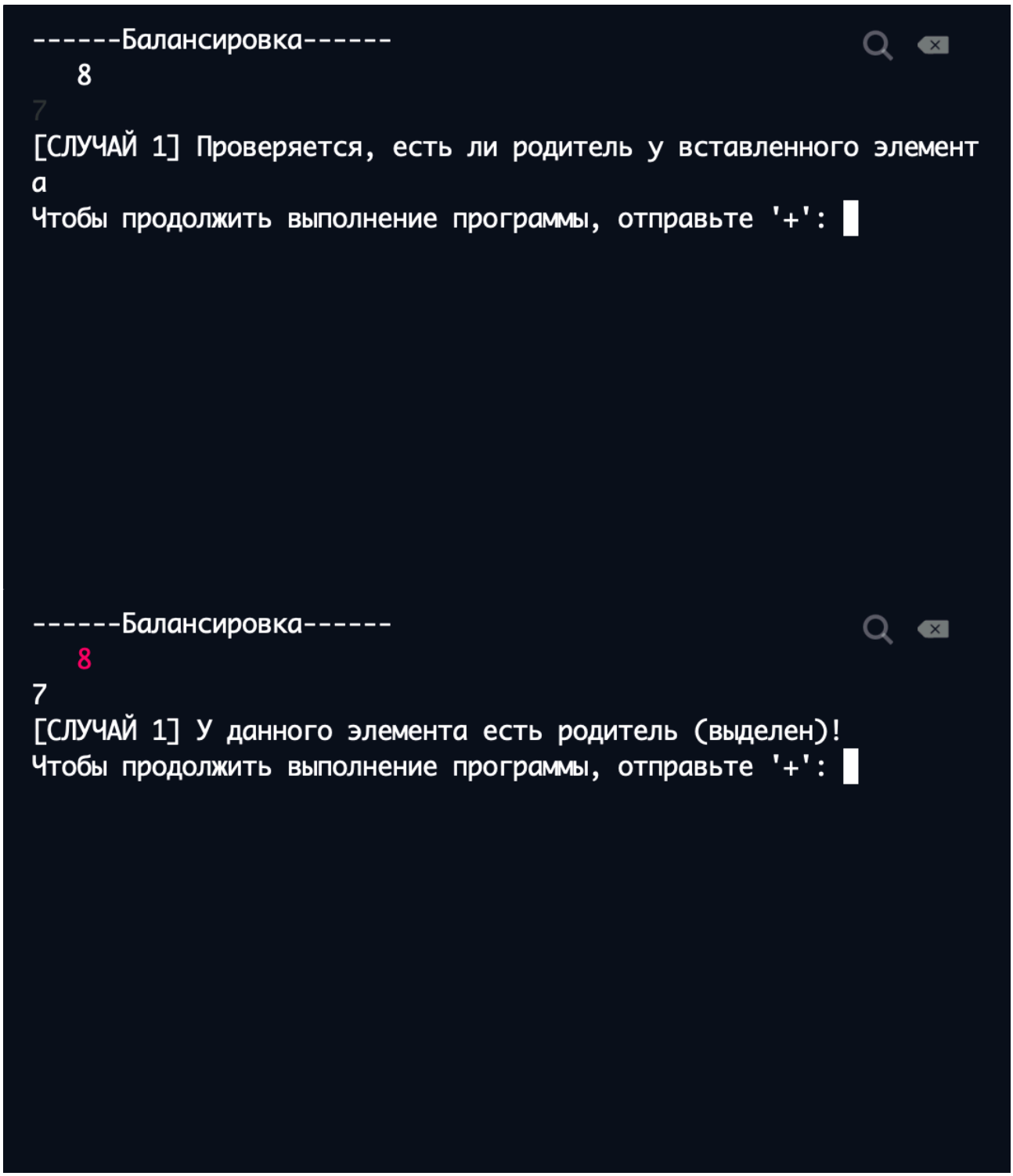
-----Текущее дерево-----
7

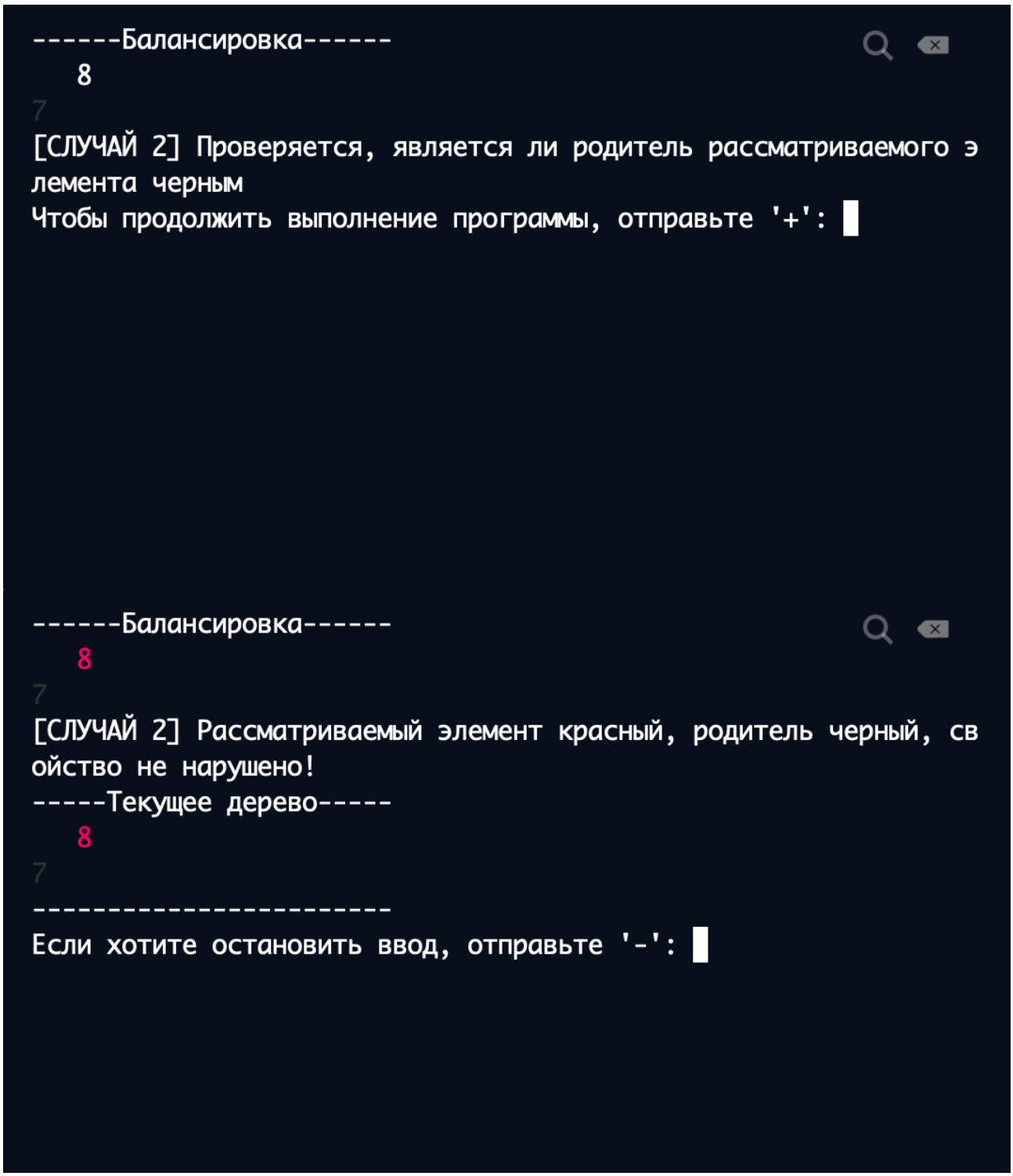
Если хотите остановить ввод, отправьте '-':
Введите элемент, который хотите вставить: 8


Введите элемент, который хотите вставить: 8

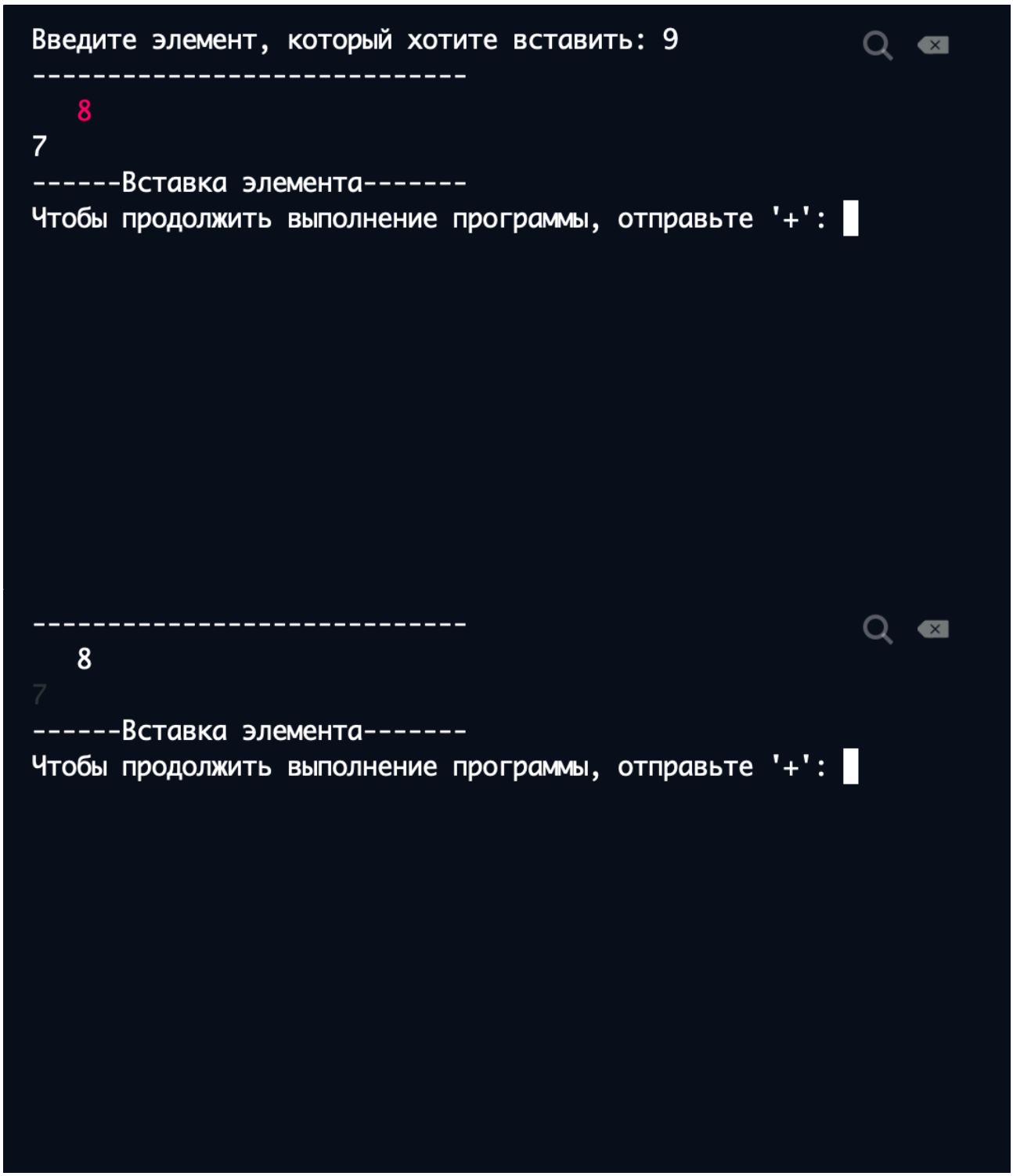
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

8
7
------Элемент вставлен-------
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемент
а
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':

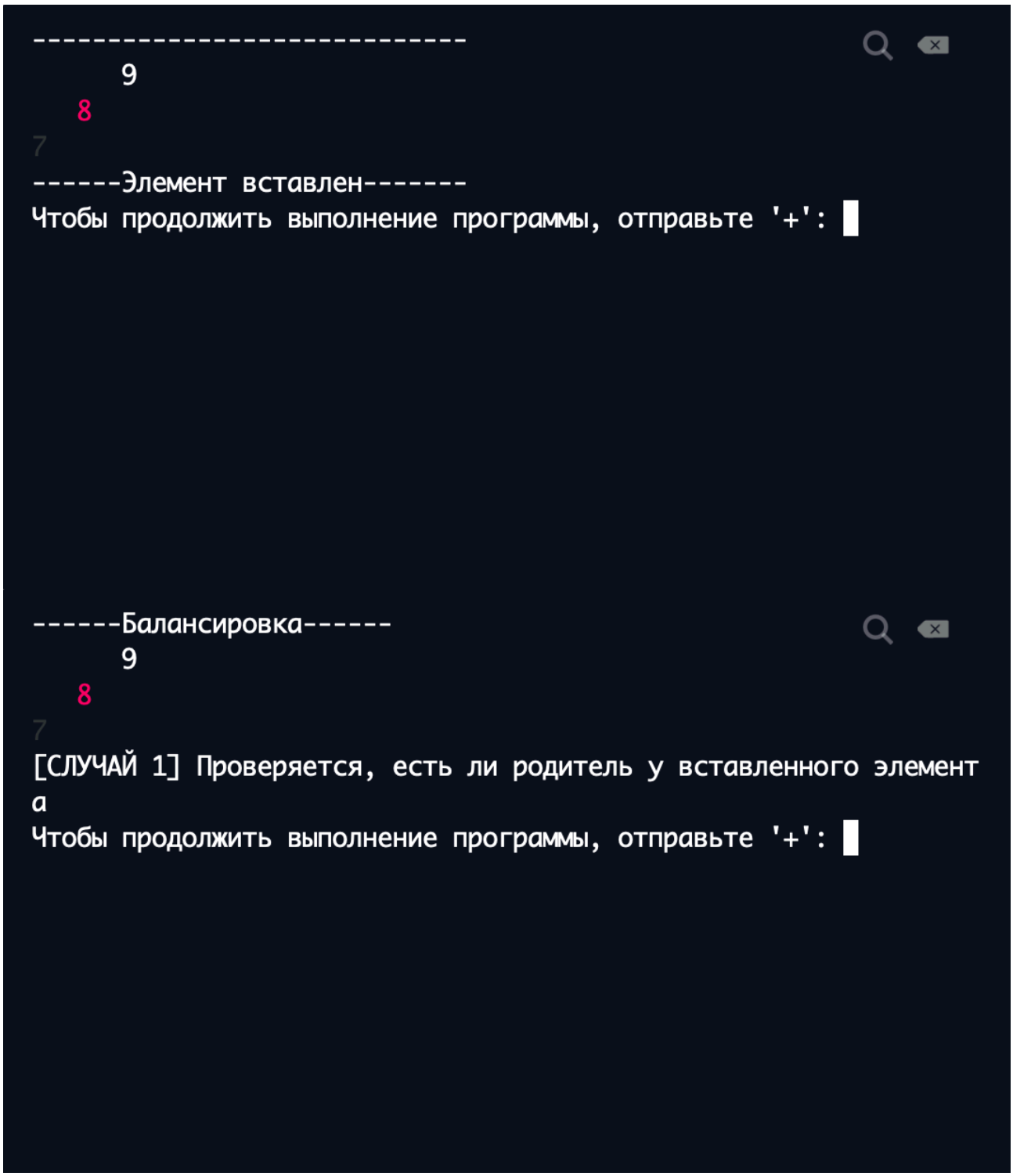

------Балансировка------
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель черный, св
ойство не нарушено!
-----Текущее дерево-----
8
7

Если хотите остановить ввод, отправьте '-':

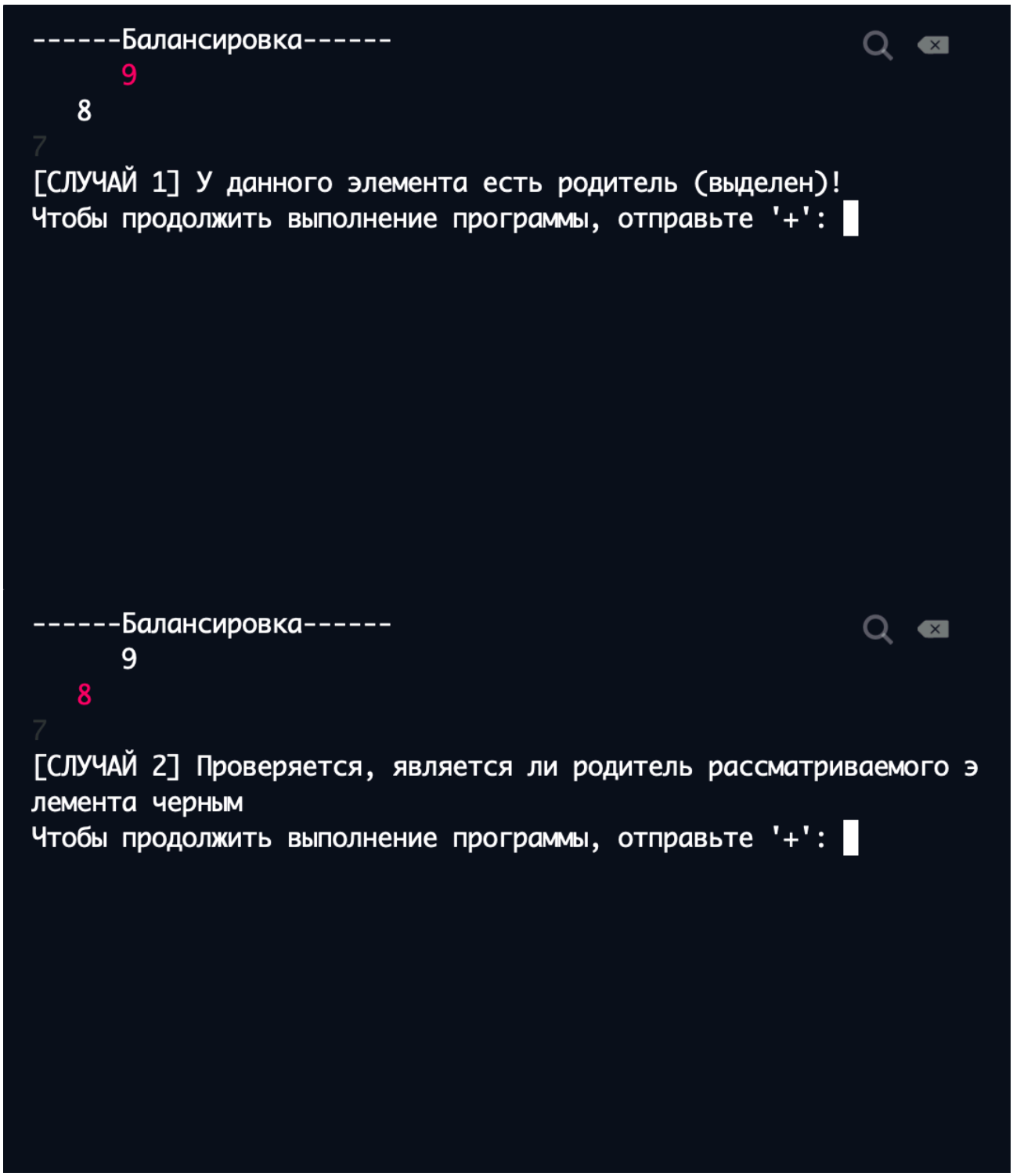

Введите элемент, который хотите вставить: 9

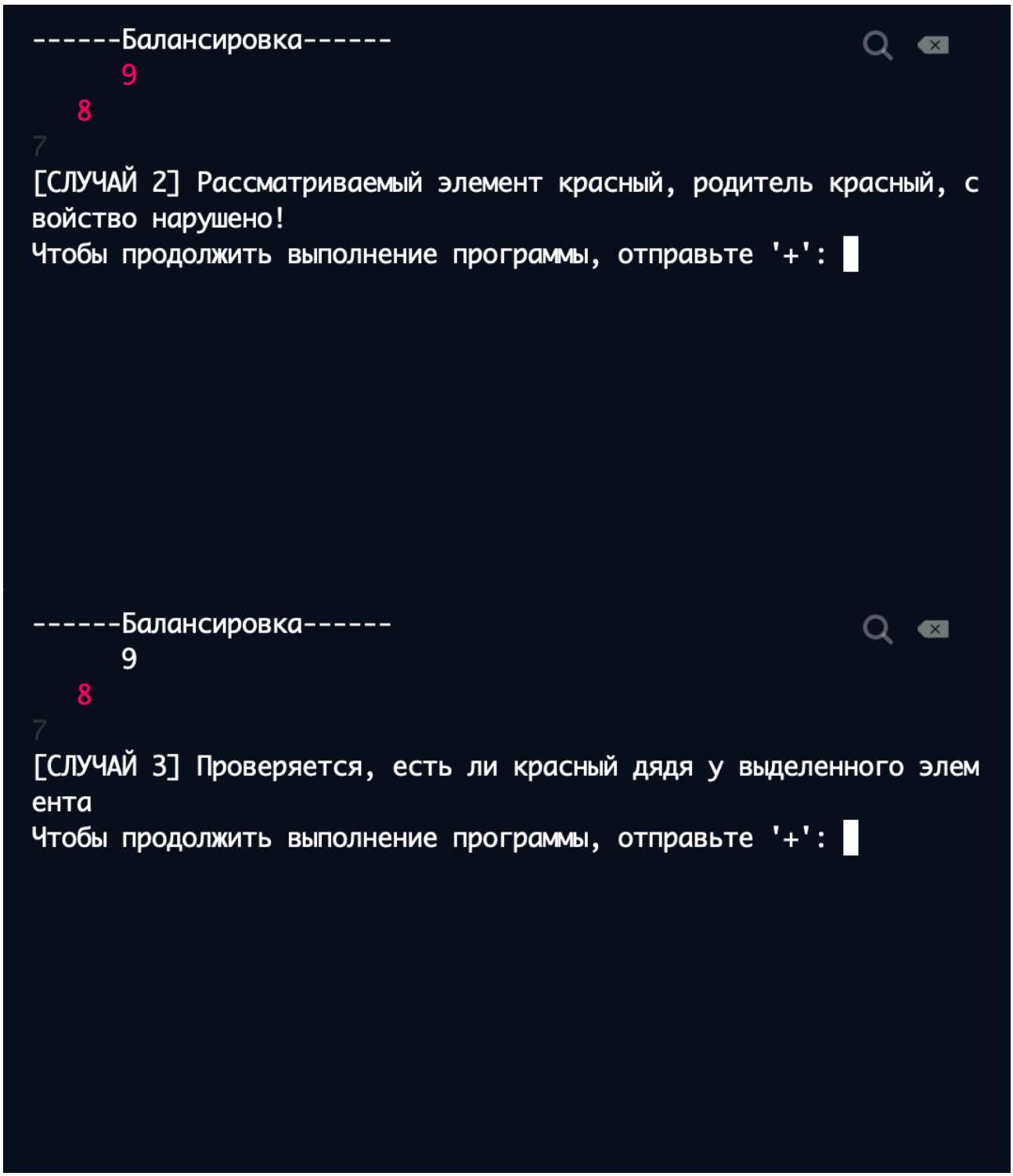
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

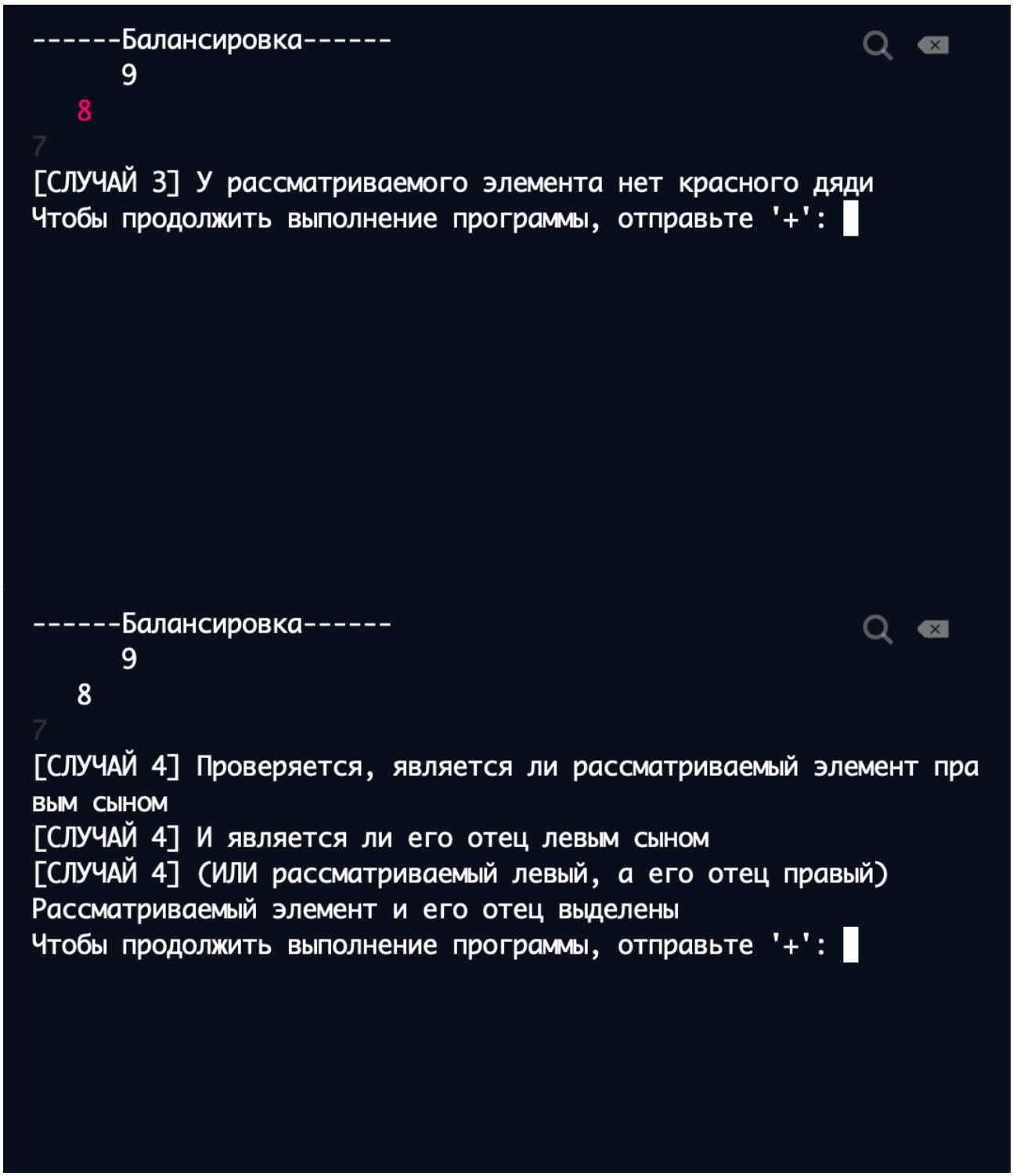
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

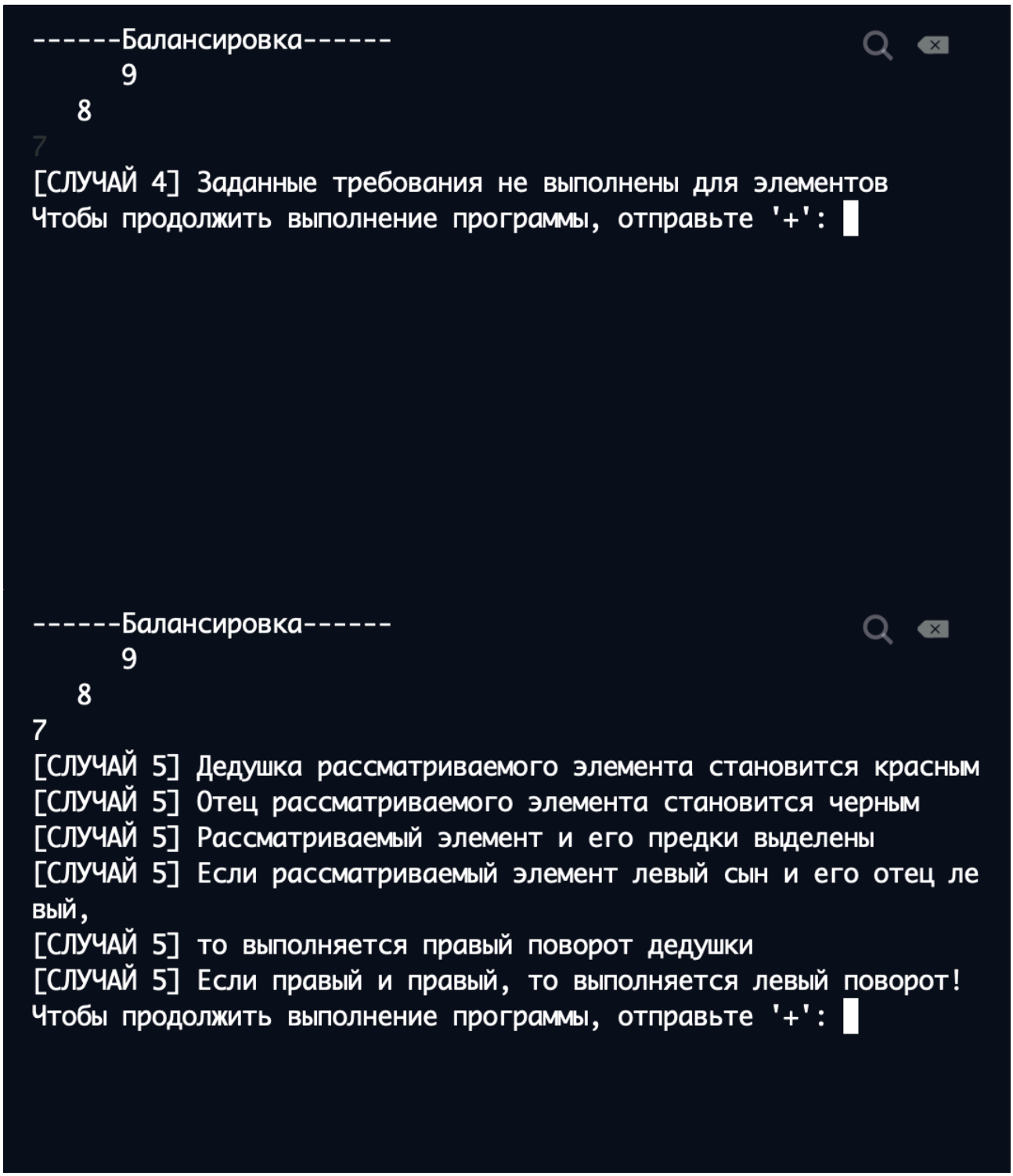


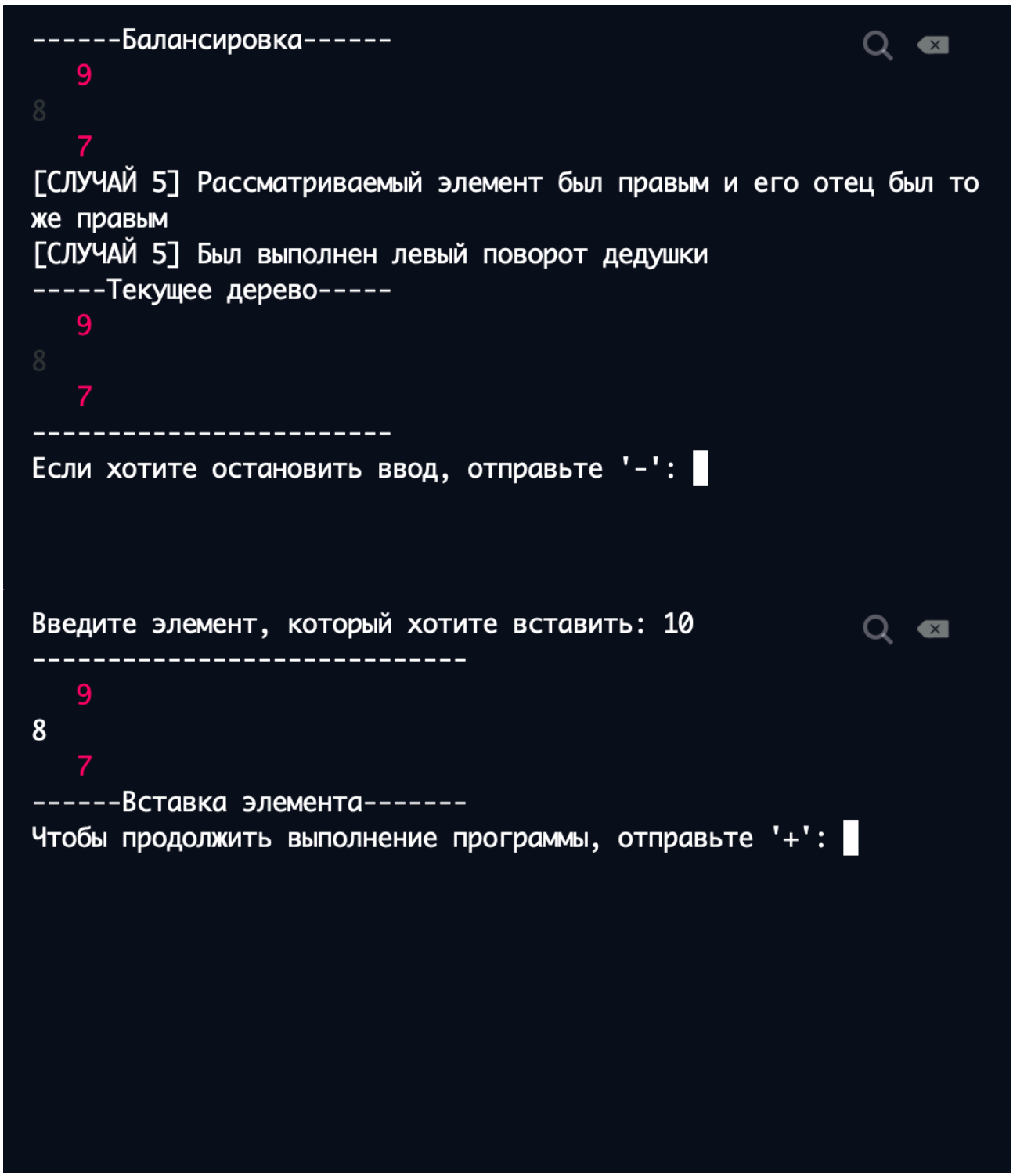
9
8
7
------Элемент вставлен-------
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
9
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемент
а
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
9
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
9
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
9
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель красный, с
войство нарушено!
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
9
8
7
[СЛУЧАЙ 3] Проверяется, есть ли красный дядя у выделенного элем
ента
Чтобы продолжить выполнение программы, отправьте '+':

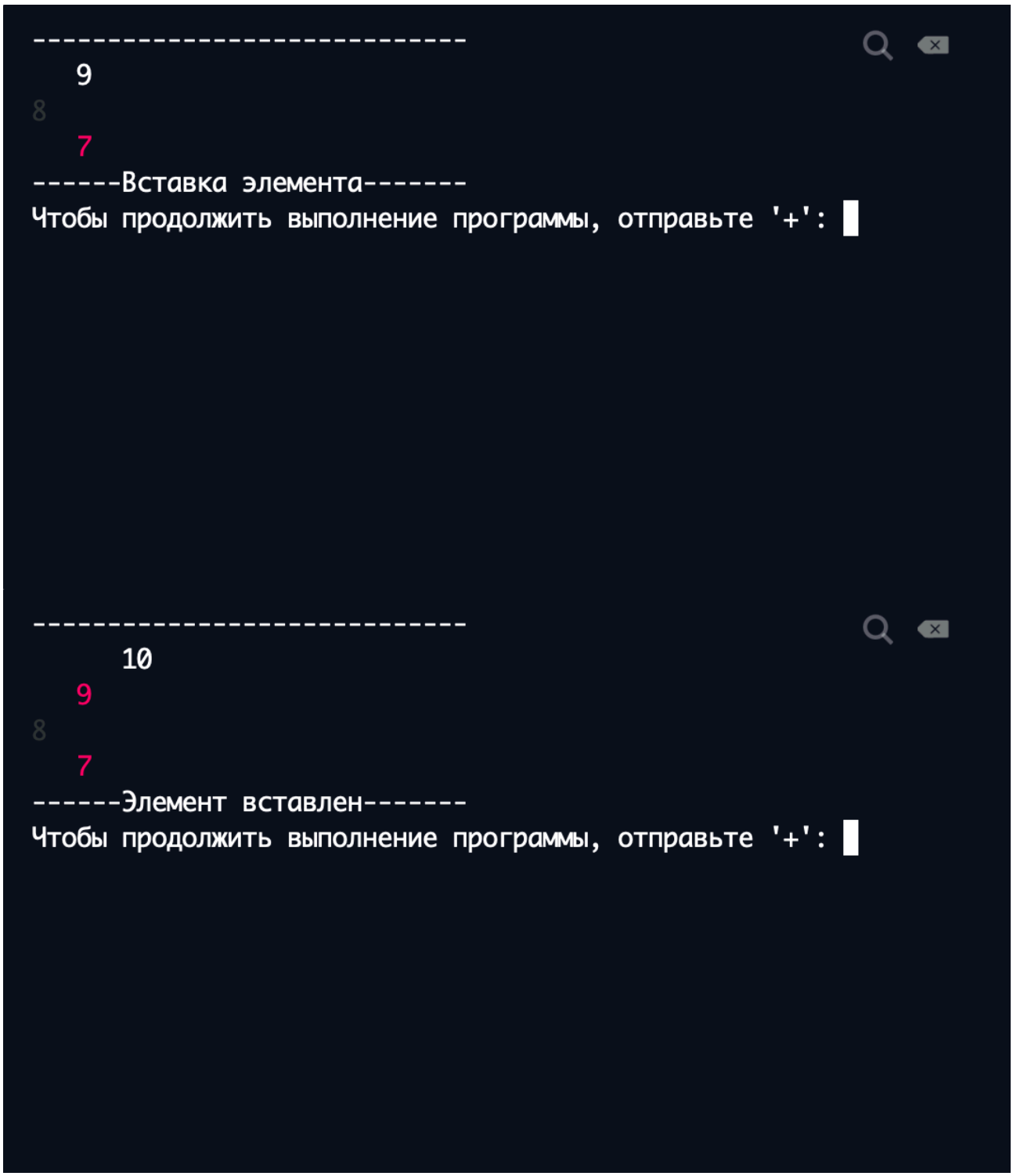

------Балансировка------
9
8
7
[СЛУЧАЙ 3] У рассматриваемого элемента нет красного дяди
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
9
8
7
[СЛУЧАЙ 4] Проверяется, является ли рассматриваемый элемент пра
вым сыном
[СЛУЧАЙ 4] И является ли его отец левым сыном
[СЛУЧАЙ 4] (ИЛИ рассматриваемый левый, а его отец правый)
Рассматриваемый элемент и его отец выделены
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
9
8
7
[СЛУЧАЙ 4] Заданные требования не выполнены для элементов
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
9
8
7
[СЛУЧАЙ 5] Дедушка рассматриваемого элемента становится красным
[СЛУЧАЙ 5] Отец рассматриваемого элемента становится черным
[СЛУЧАЙ 5] Рассматриваемый элемент и его предки выделены
[СЛУЧАЙ 5] Если рассматриваемый элемент левый сын и его отец ле
вый,
[СЛУЧАЙ 5] то выполняется правый поворот дедушки
[СЛУЧАЙ 5] Если правый и правый, то выполняется левый поворот!
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
9
8
7
[СЛУЧАЙ 5] Рассматриваемый элемент был правым и его отец был то же правым
[СЛУЧАЙ 5] Был выполнен левый поворот дедушки
-----Текущее дерево-----
9
8
7

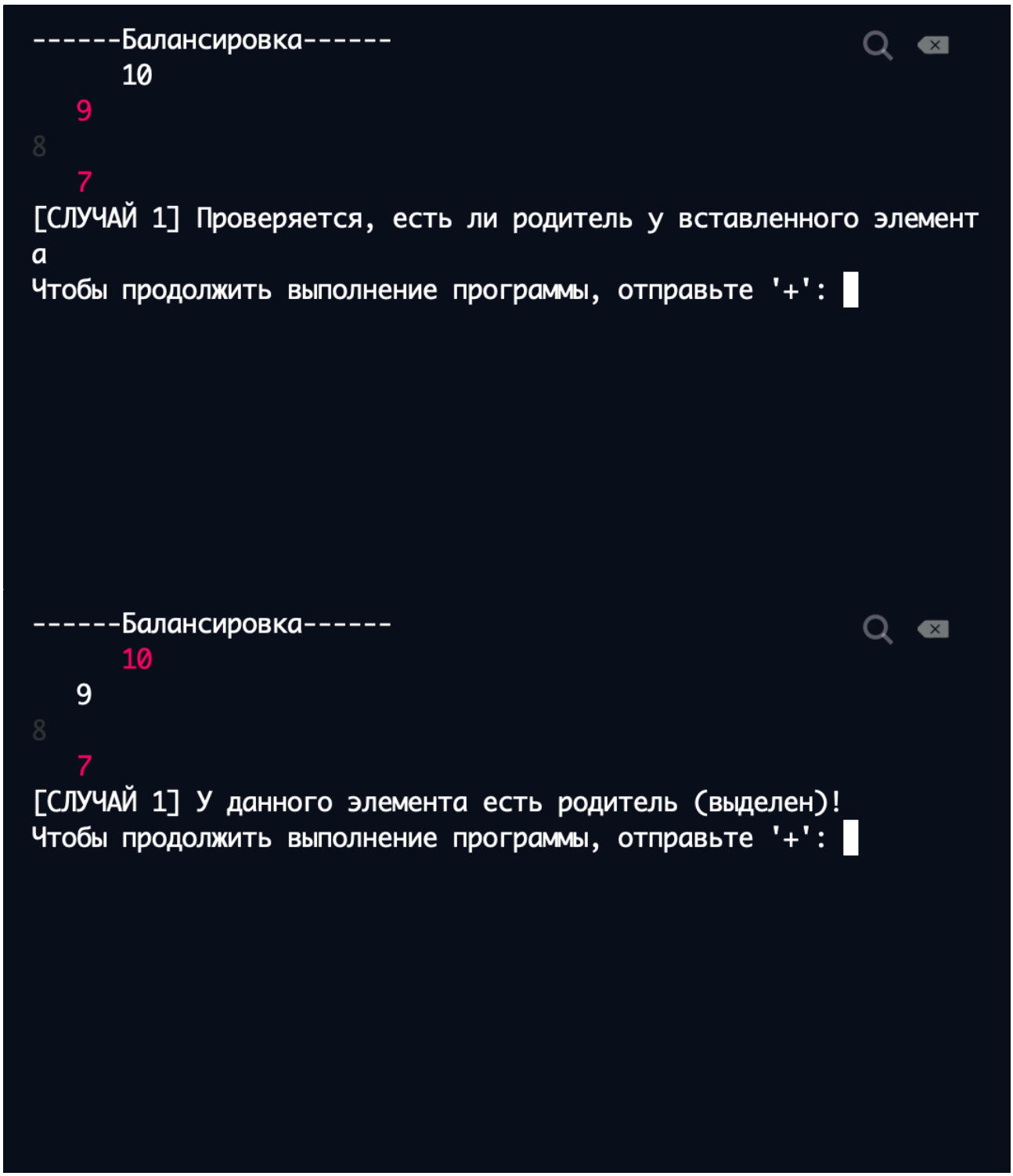
Если хотите остановить ввод, отправьте '-':
Введите элемент, который хотите вставить: 10

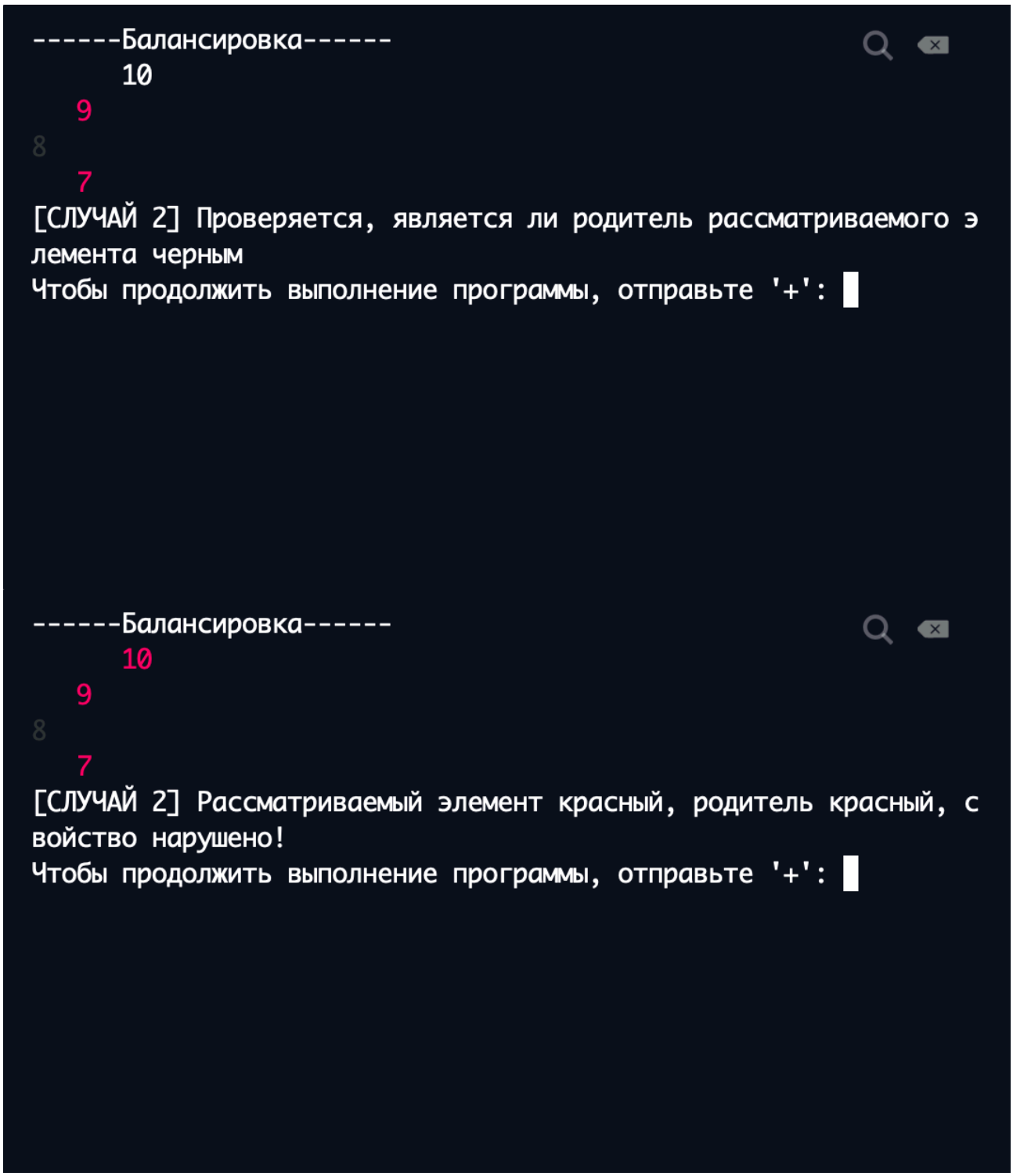
9
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

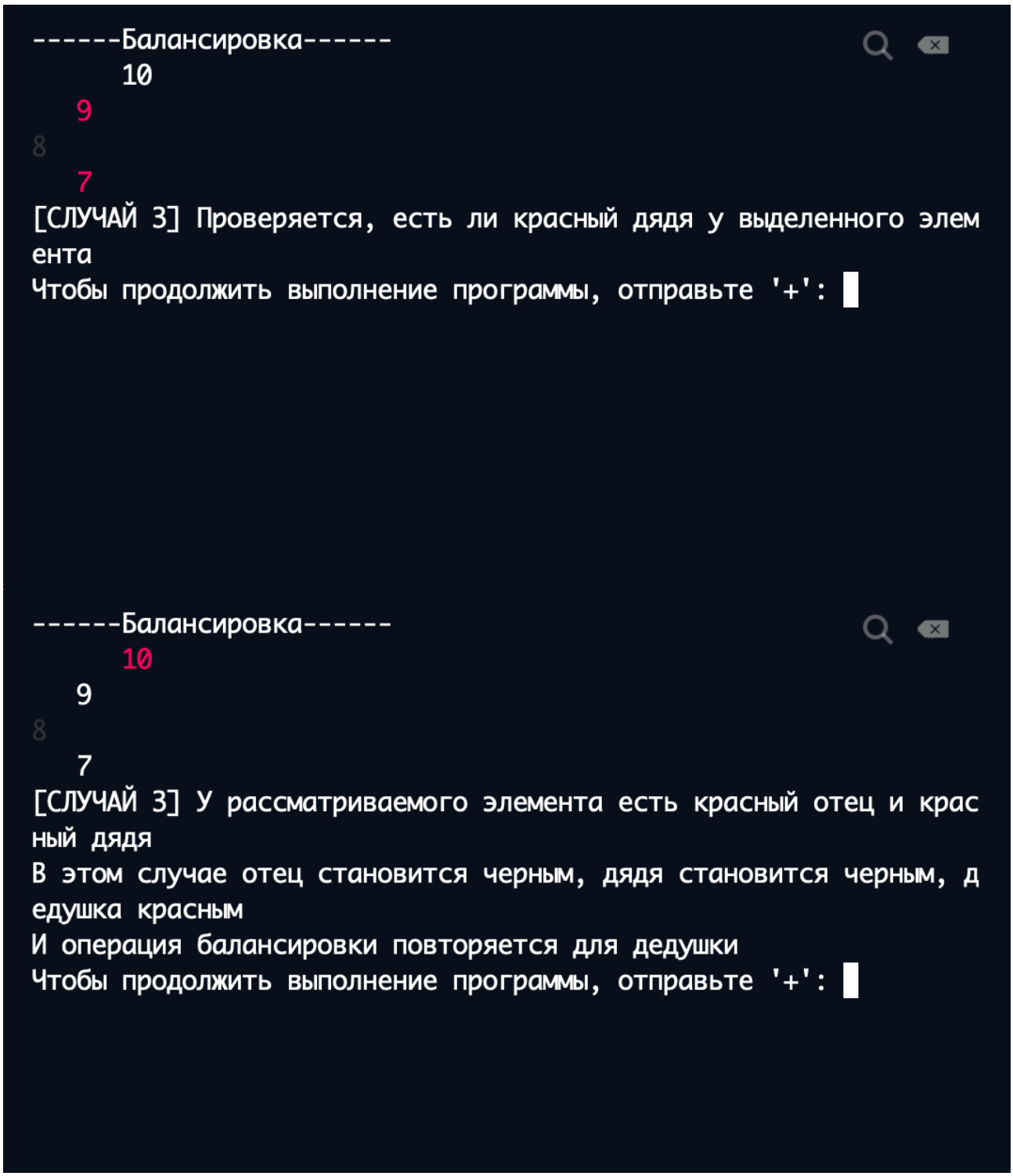


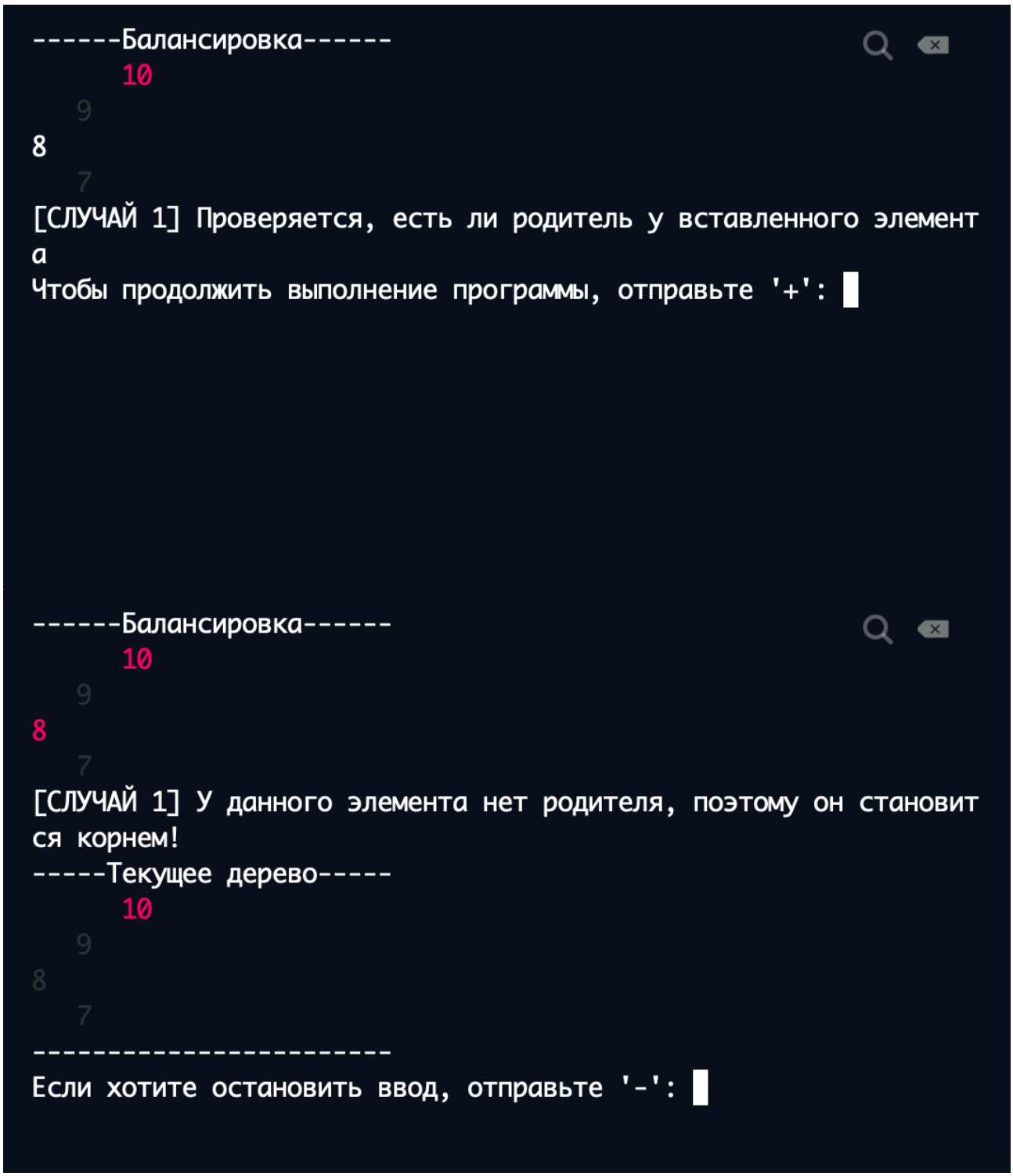
9
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

10
9
8
7
------Элемент вставлен-------
Чтобы продолжить выполнение программы, отправьте '+':

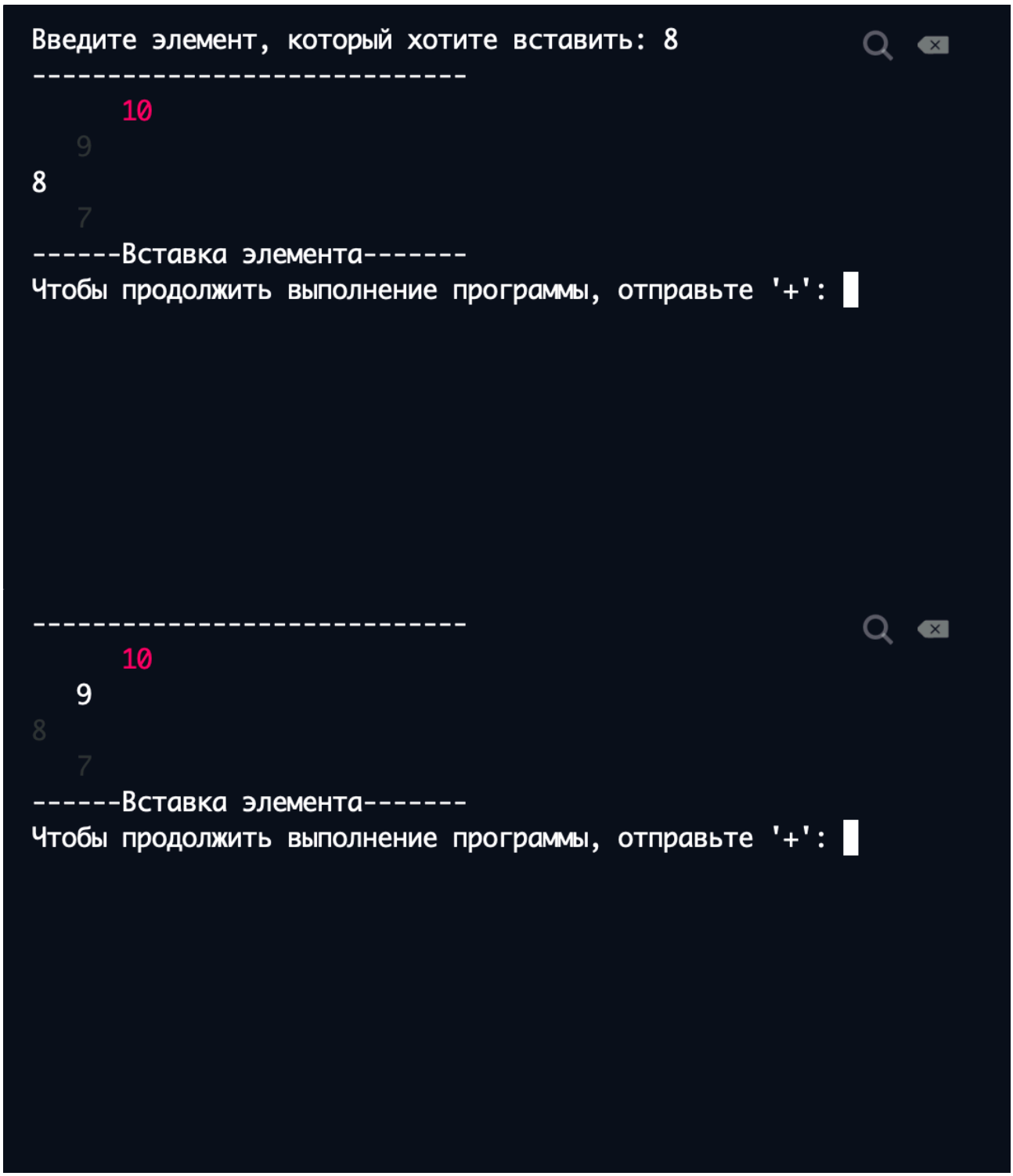

------Балансировка------
10
9
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемента
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель красный, с
войство нарушено!
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
8
7
[СЛУЧАЙ 3] Проверяется, есть ли красный дядя у выделенного элем
ента
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
8
7
[СЛУЧАЙ 3] У рассматриваемого элемента есть красный отец и крас
ный дядя
В этом случае отец становится черным, дядя становится черным, д
едушка красным
И операция балансировки повторяется для дедушки
Чтобы продолжить выполнение программы, отправьте '+':

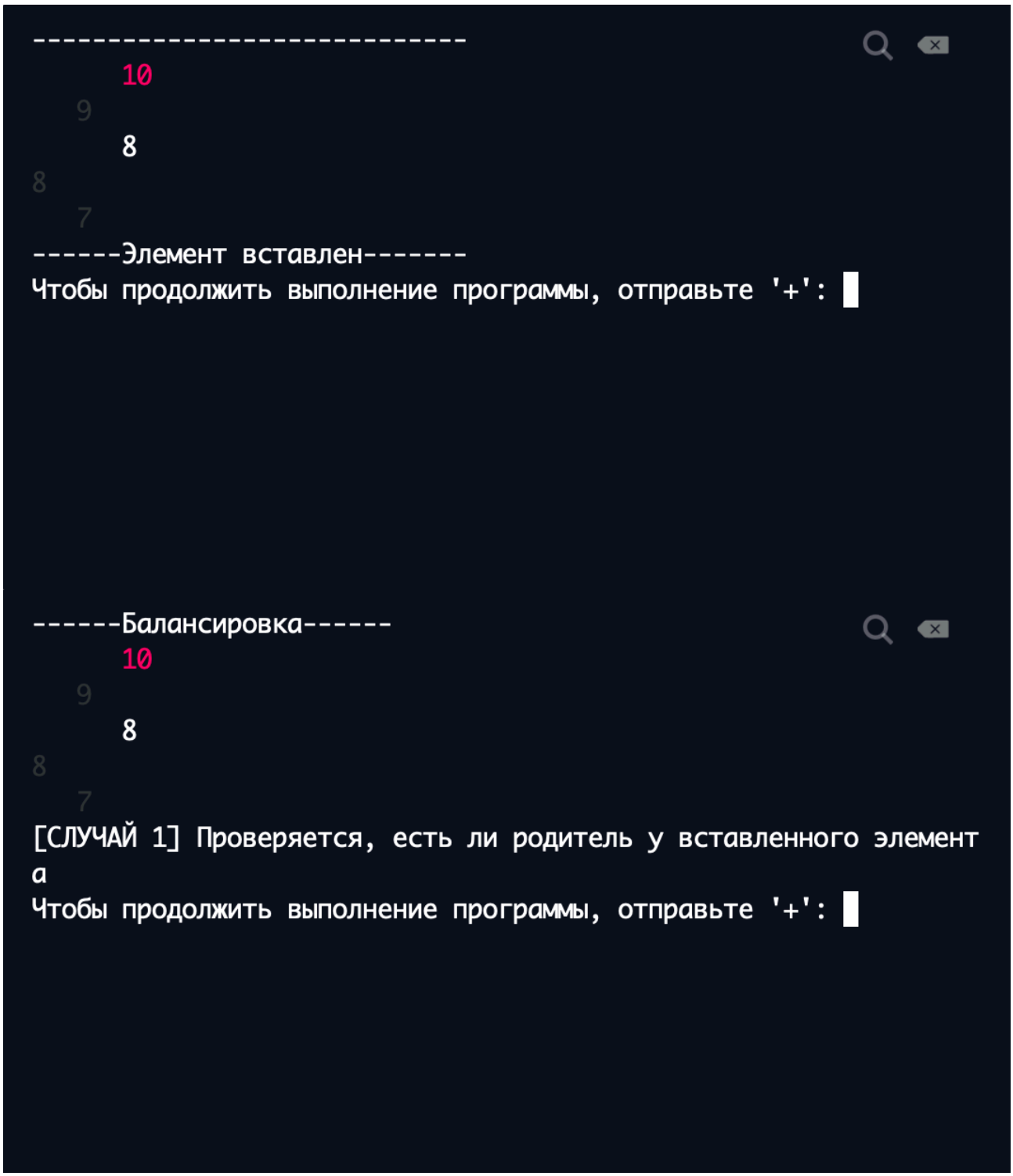

------Балансировка------
10
9
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемента
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
8
7
[СЛУЧАЙ 1] У данного элемента нет родителя, поэтому он становится корнем!
-----Текущее дерево-----
10
9
8
7

Если хотите остановить ввод, отправьте '-':

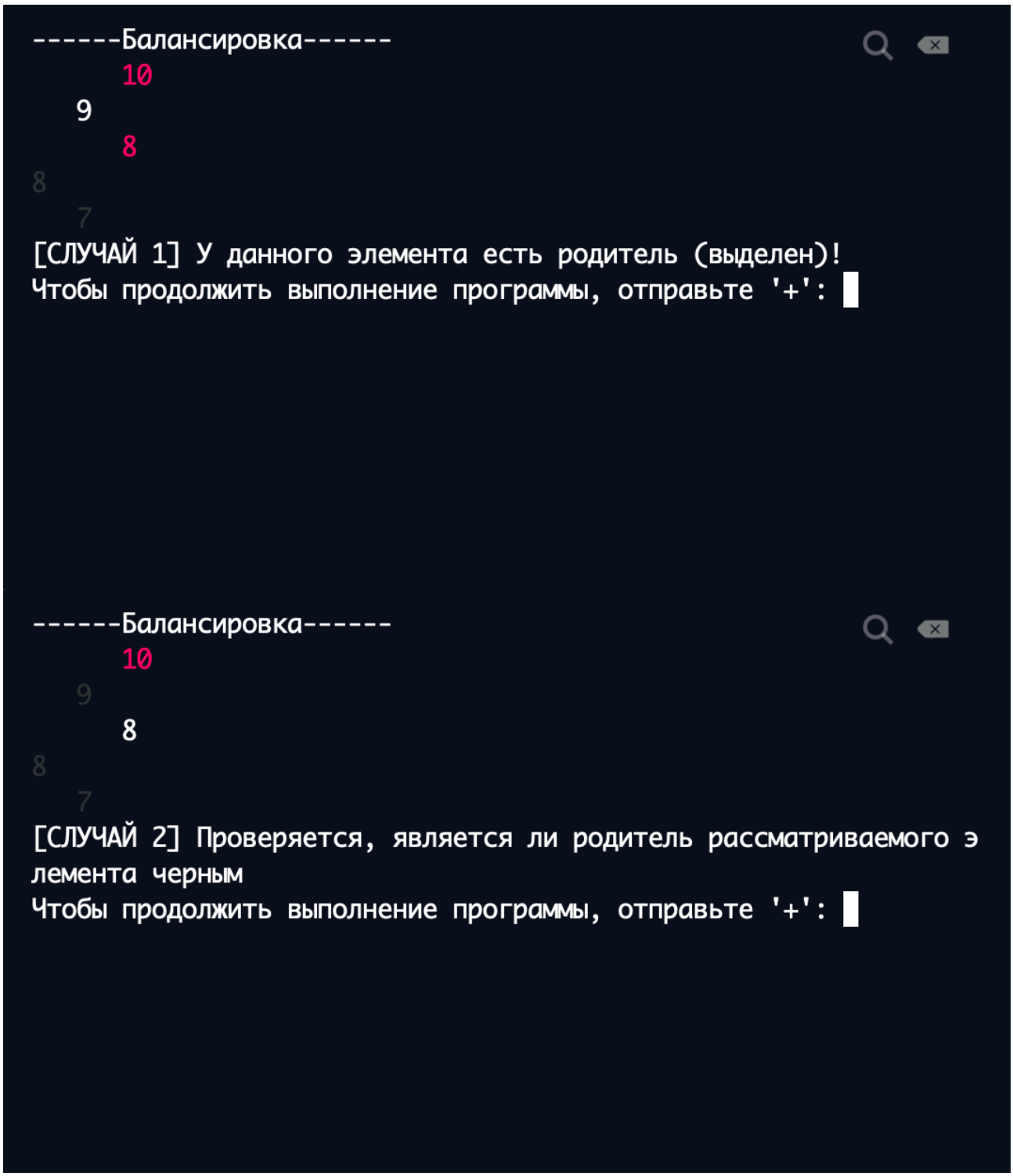

Введите элемент, который хотите вставить: 8

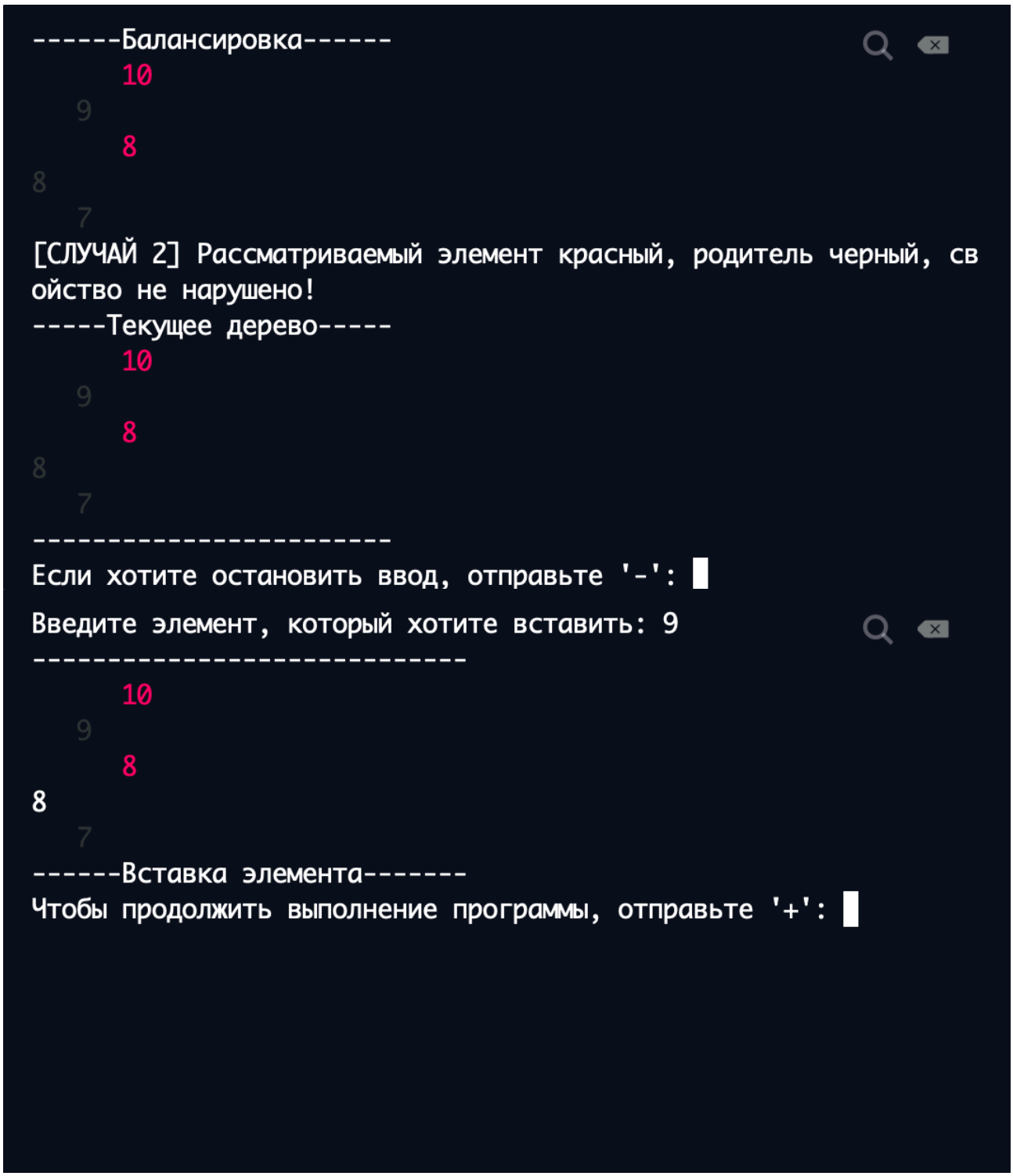
10
9
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

10
9
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':



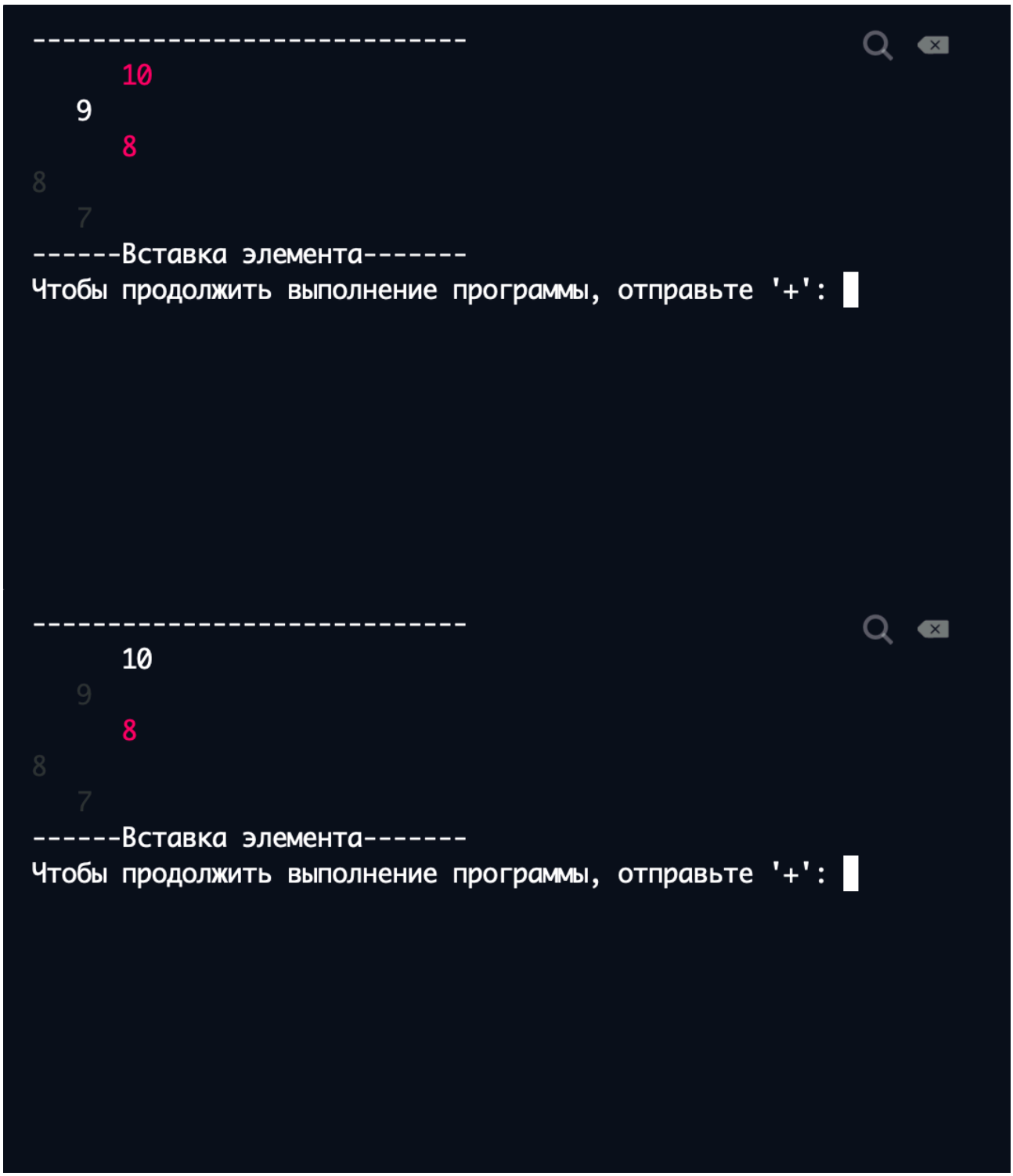
10
9
8
8
7
------Элемент вставлен-------
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
8
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемент
а
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
8
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
8
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
8
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель черный, св
ойство не нарушено!
-----Текущее дерево-----
10
9
8
8
7

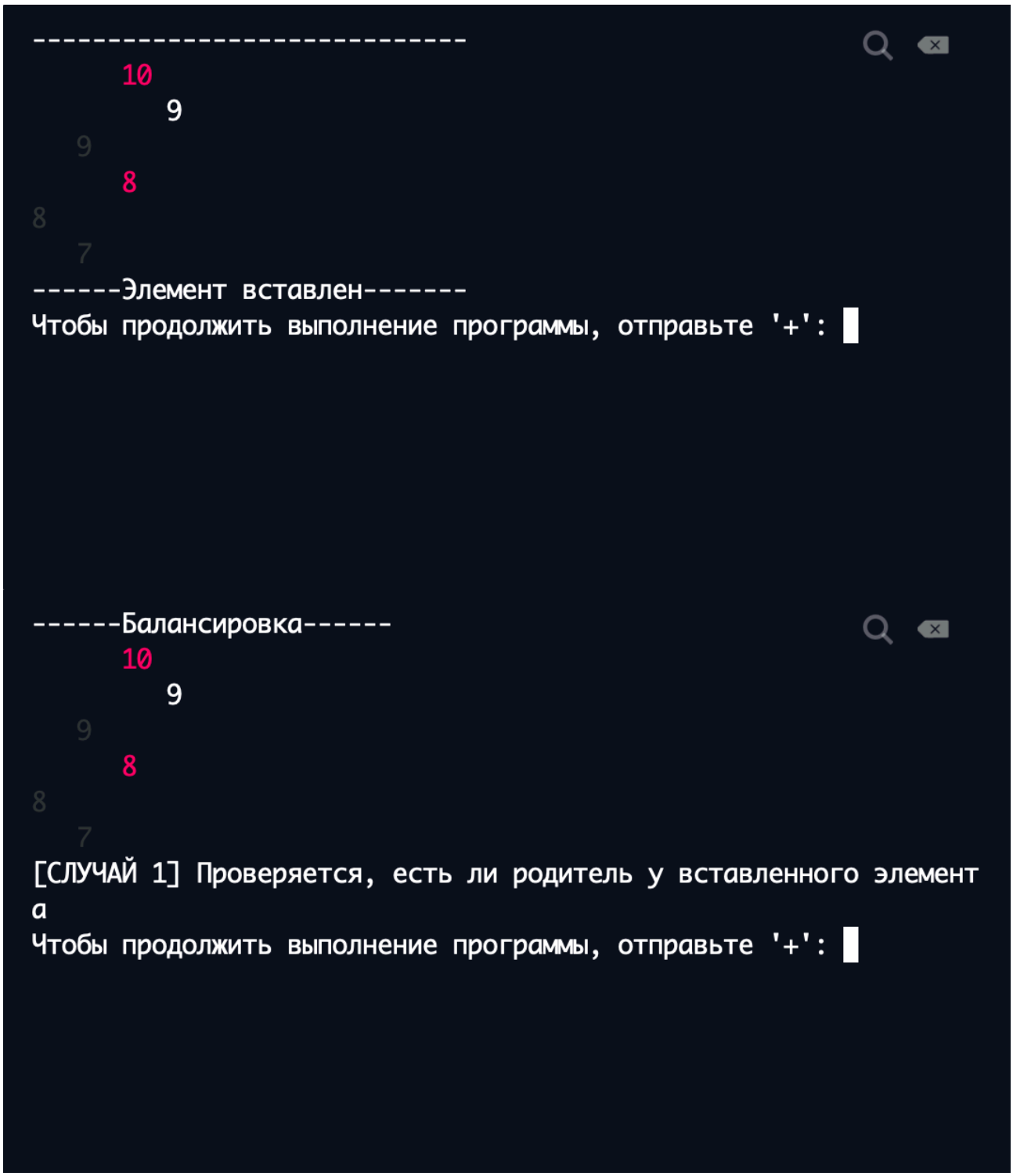
Если хотите остановить ввод, отправьте '-':
Введите элемент, который хотите вставить: 9

10
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

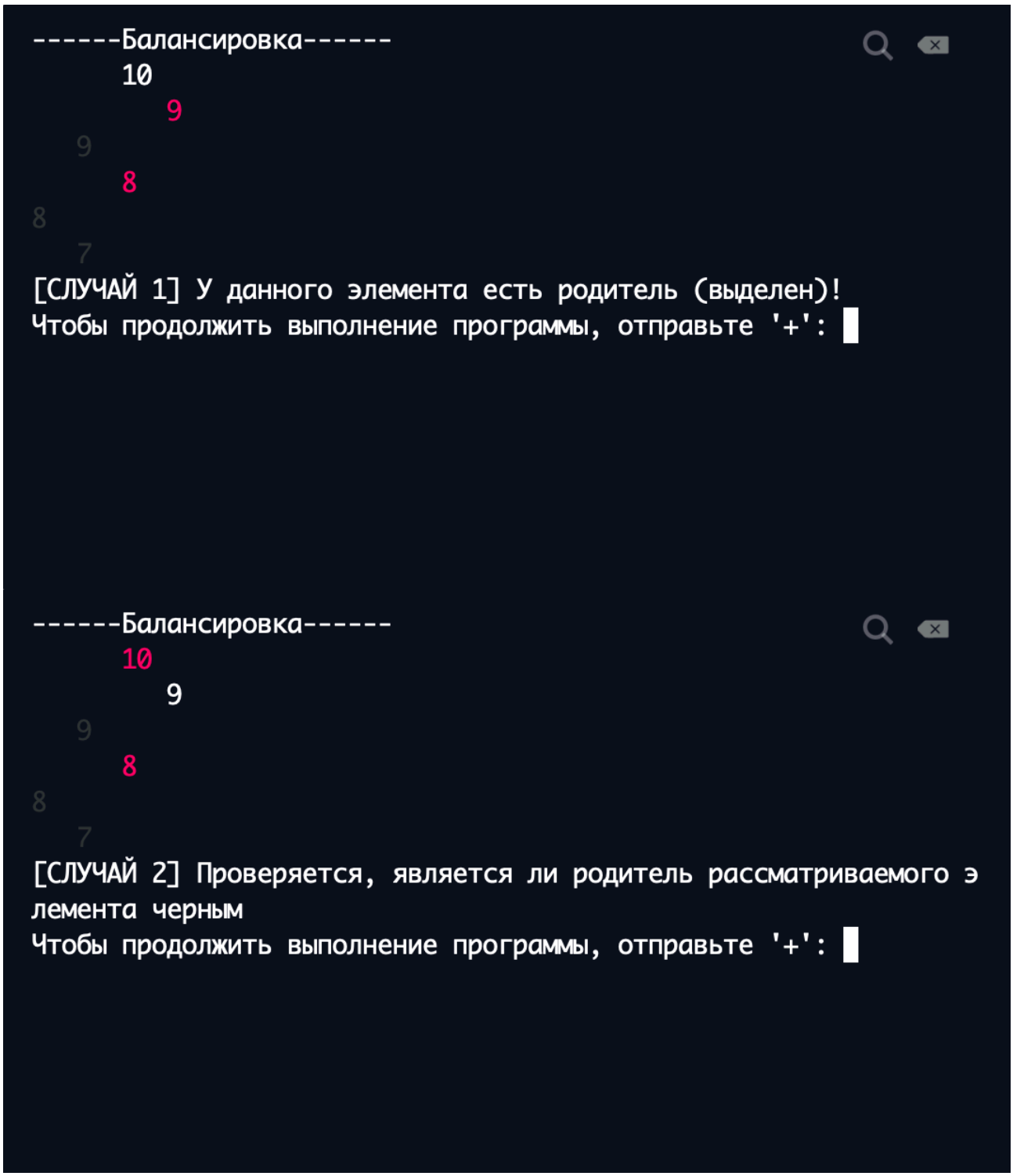


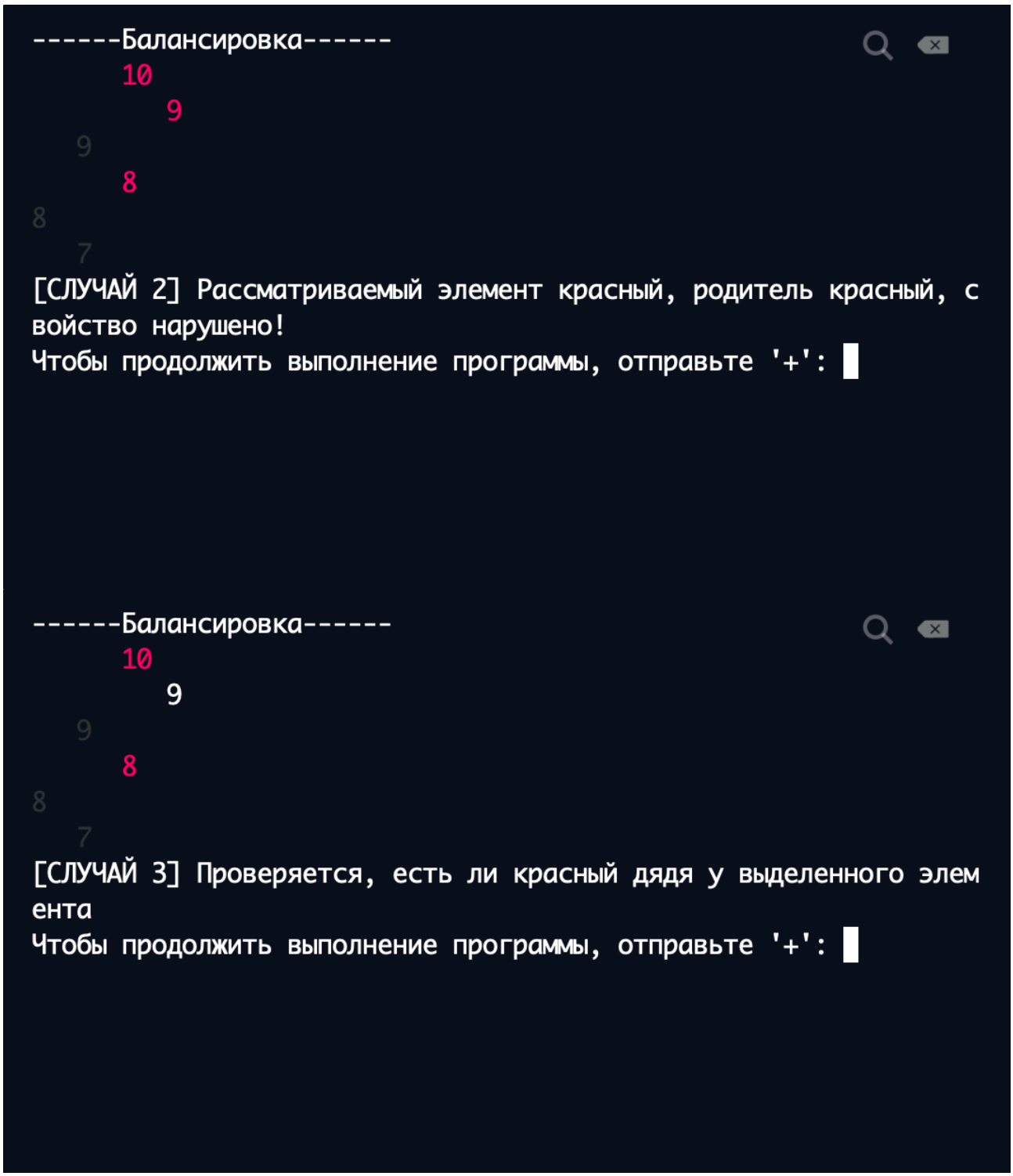
10
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

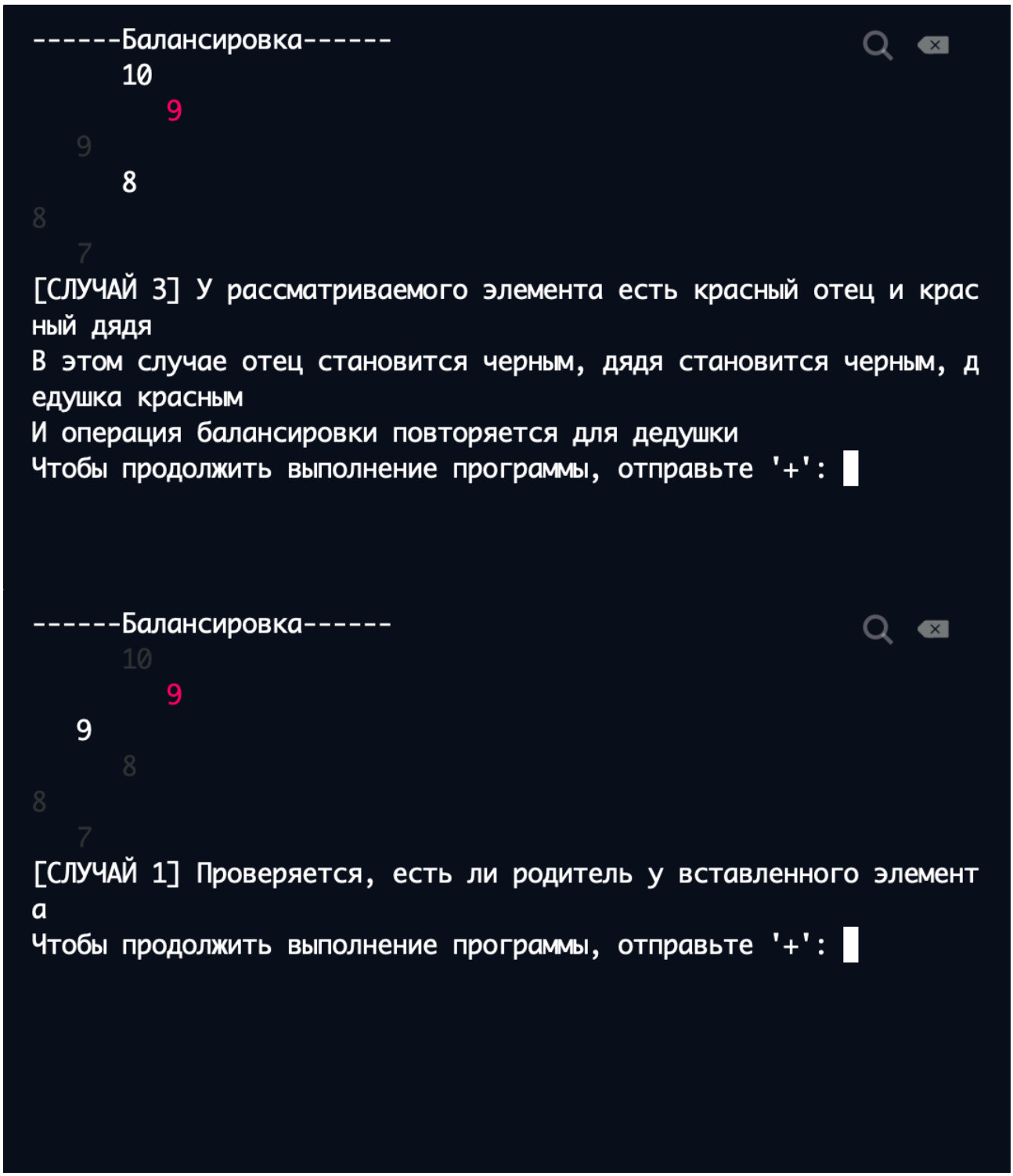
10
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

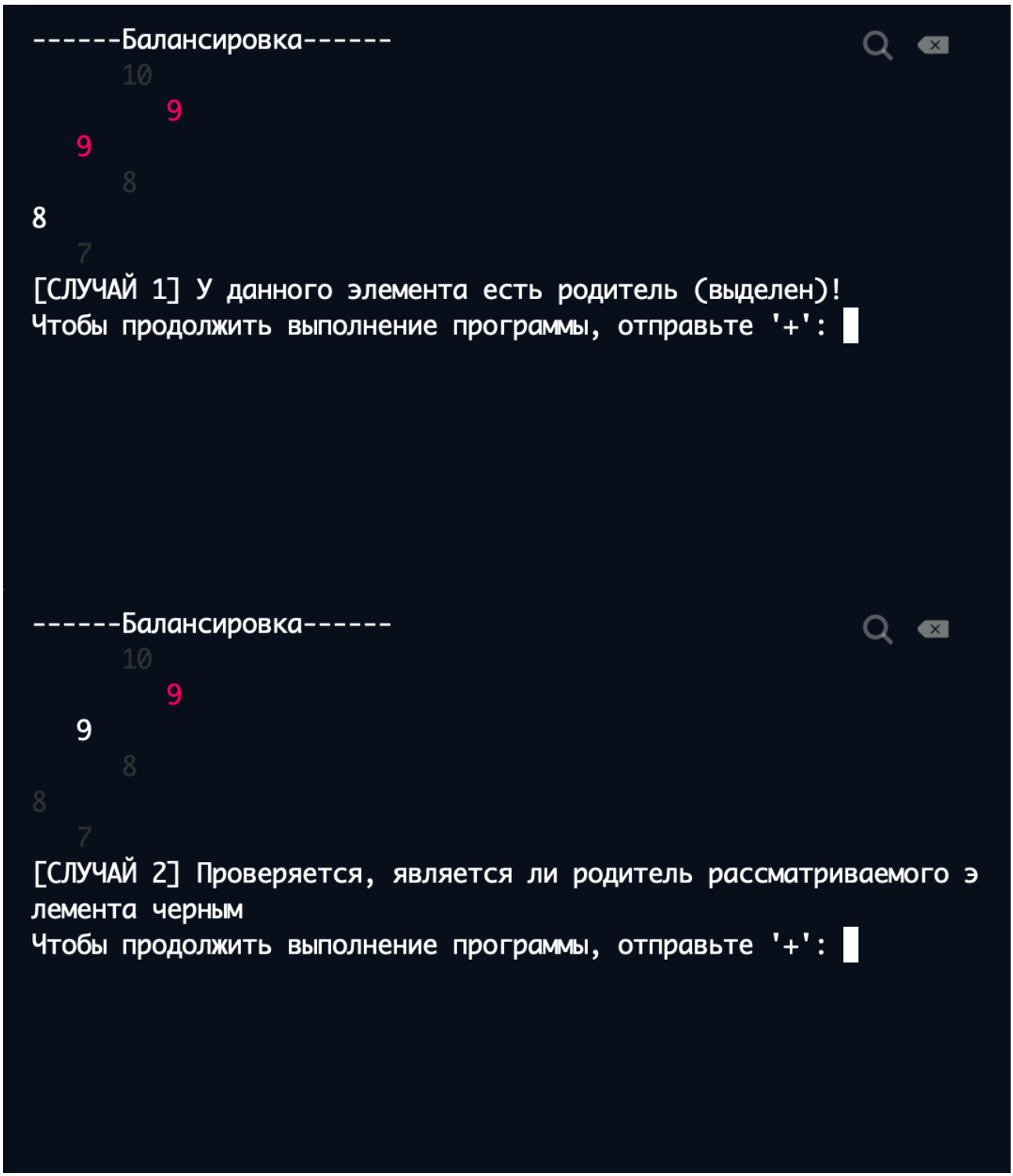


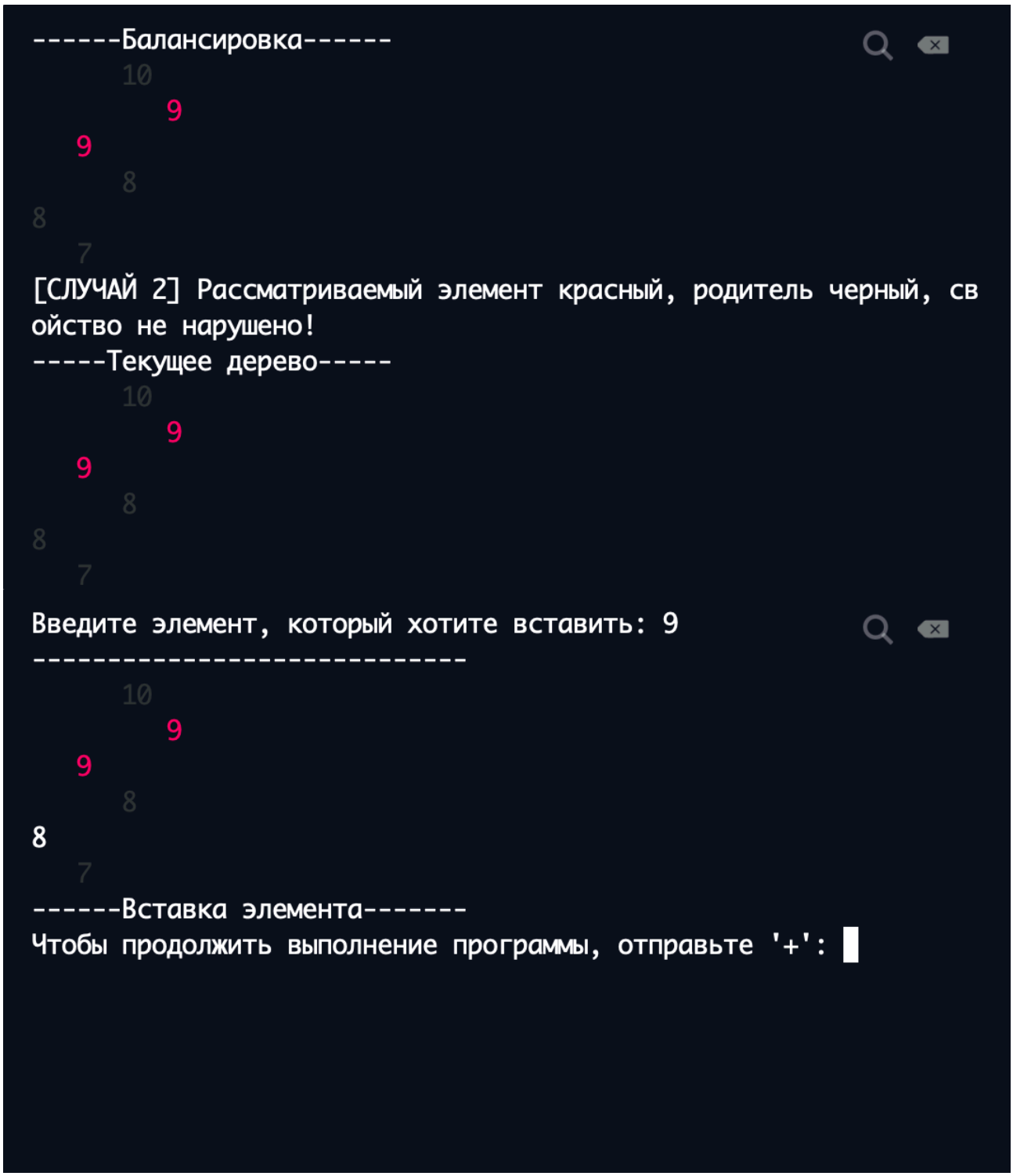
10
9
9
8
8
7
------Элемент вставлен-------
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемент
а
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':

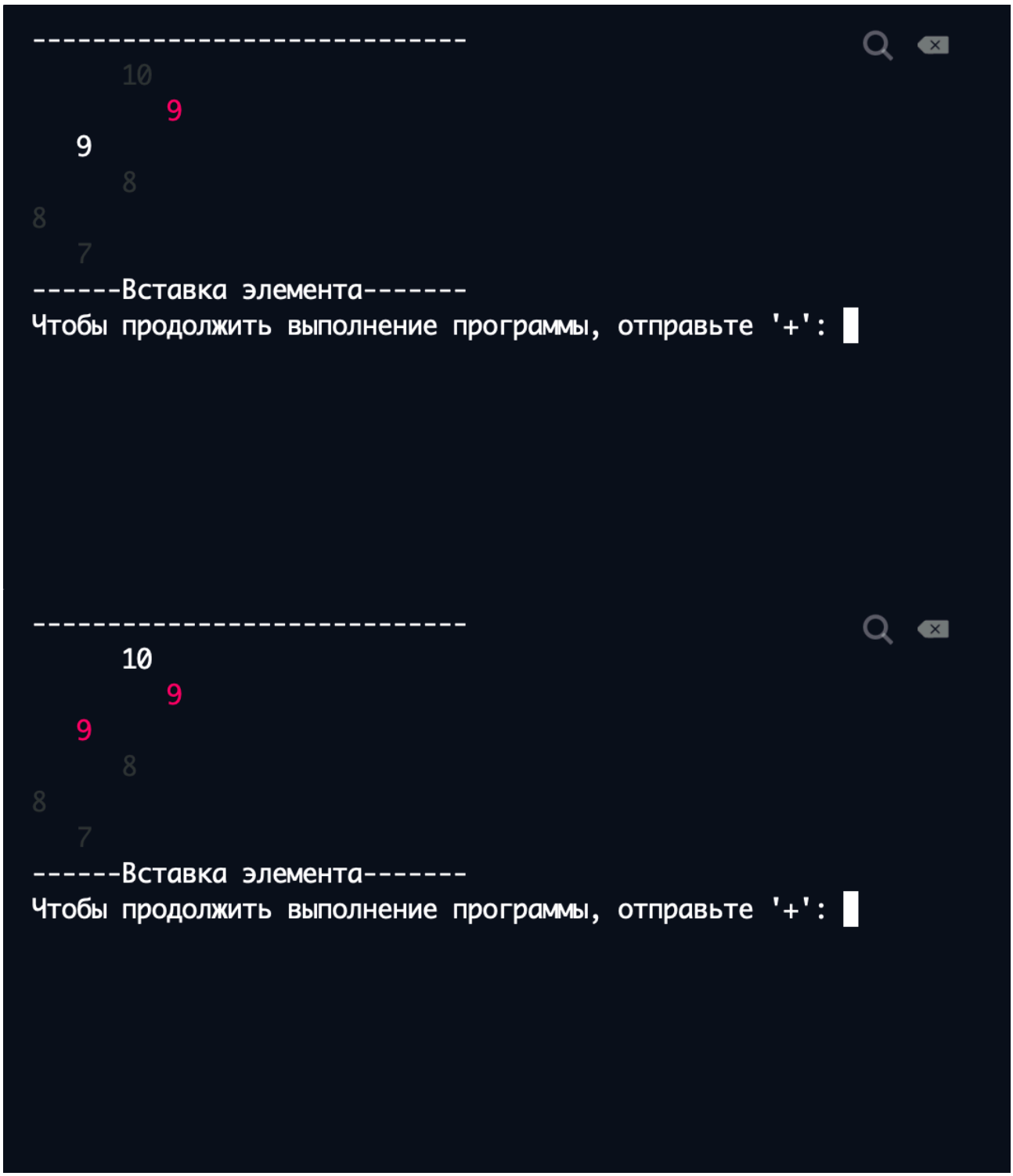

------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель красный, с
войство нарушено!
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 3] Проверяется, есть ли красный дядя у выделенного элем
ента
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 3] У рассматриваемого элемента есть красный отец и крас
ный дядя
В этом случае отец становится черным, дядя становится черным, д
едушка красным
И операция балансировки повторяется для дедушки
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемент
а
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':

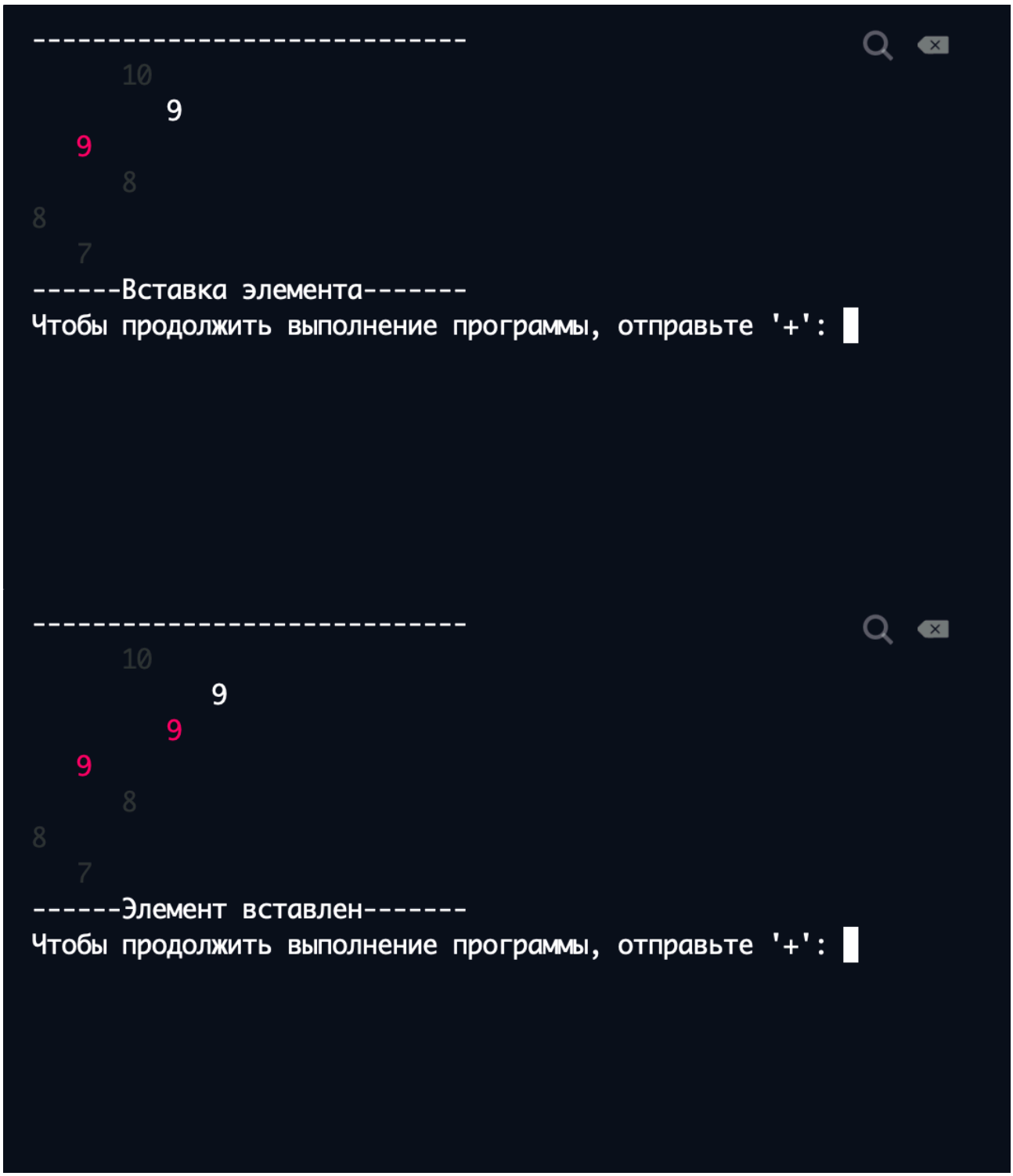

------Балансировка------
10
9
9
8
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель черный, св
ойство не нарушено!
-----Текущее дерево-----
10
9
9
8
8
7
Введите элемент, который хотите вставить: 9

10
9
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':



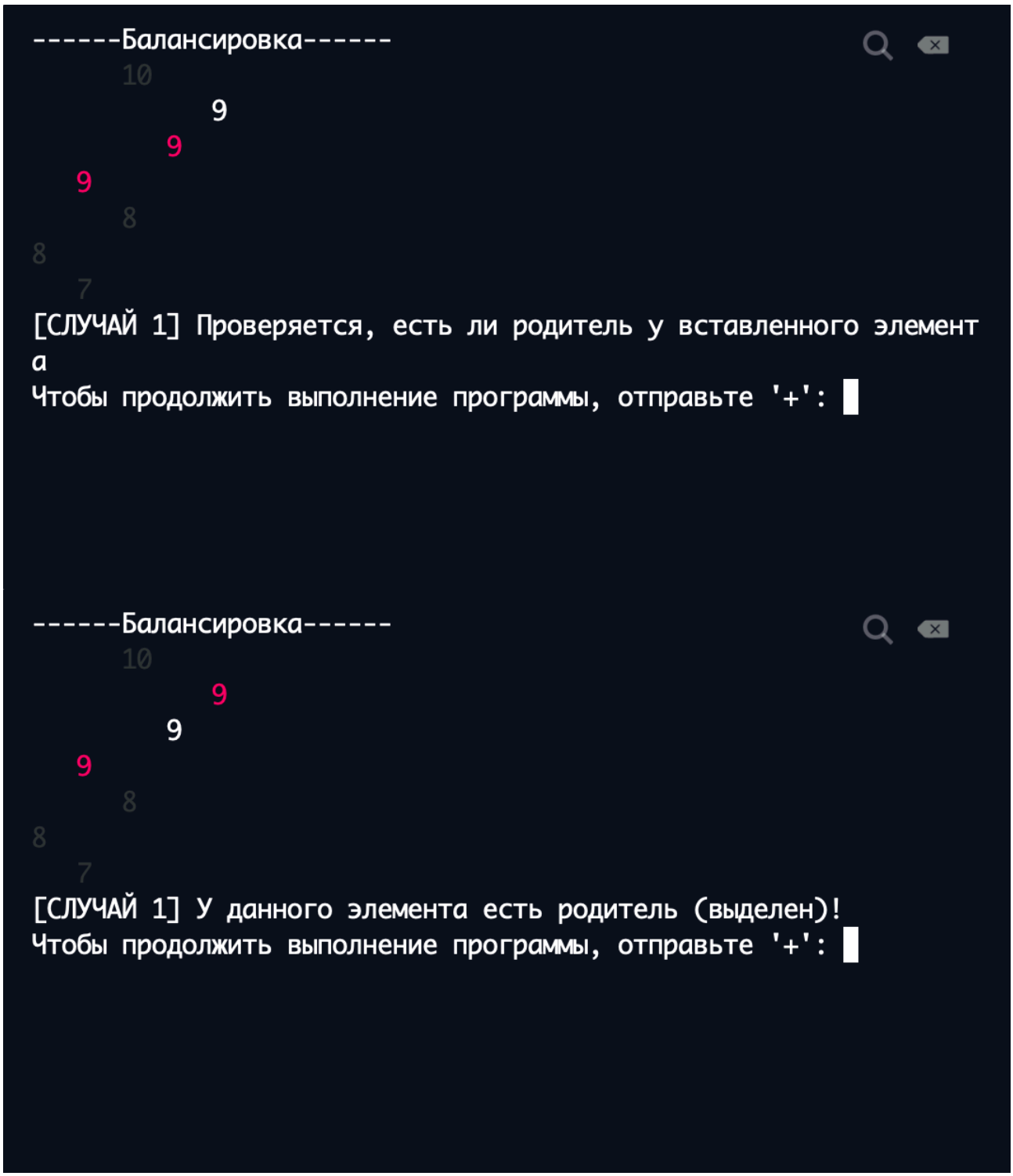
10
9
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

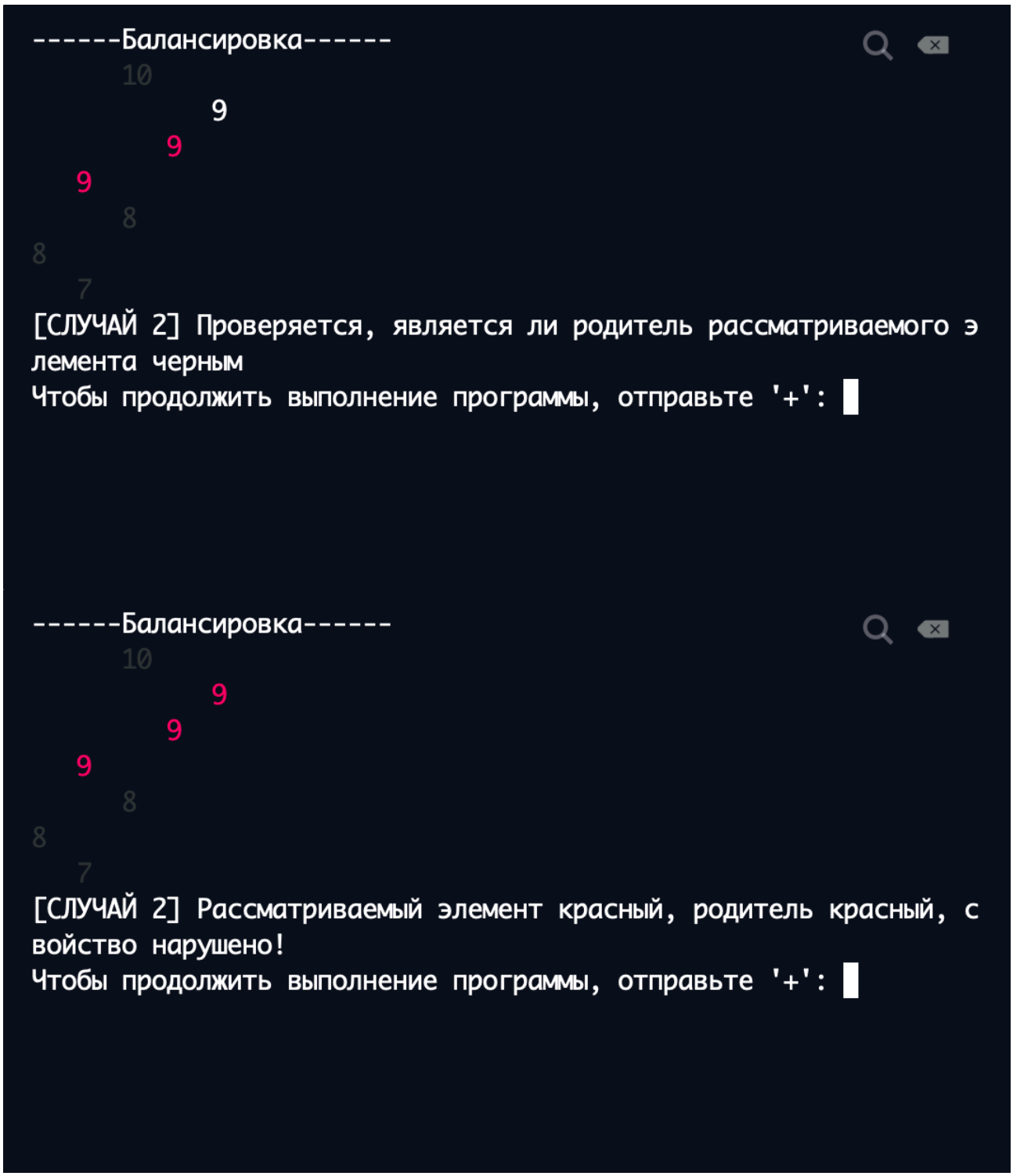
10
9
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

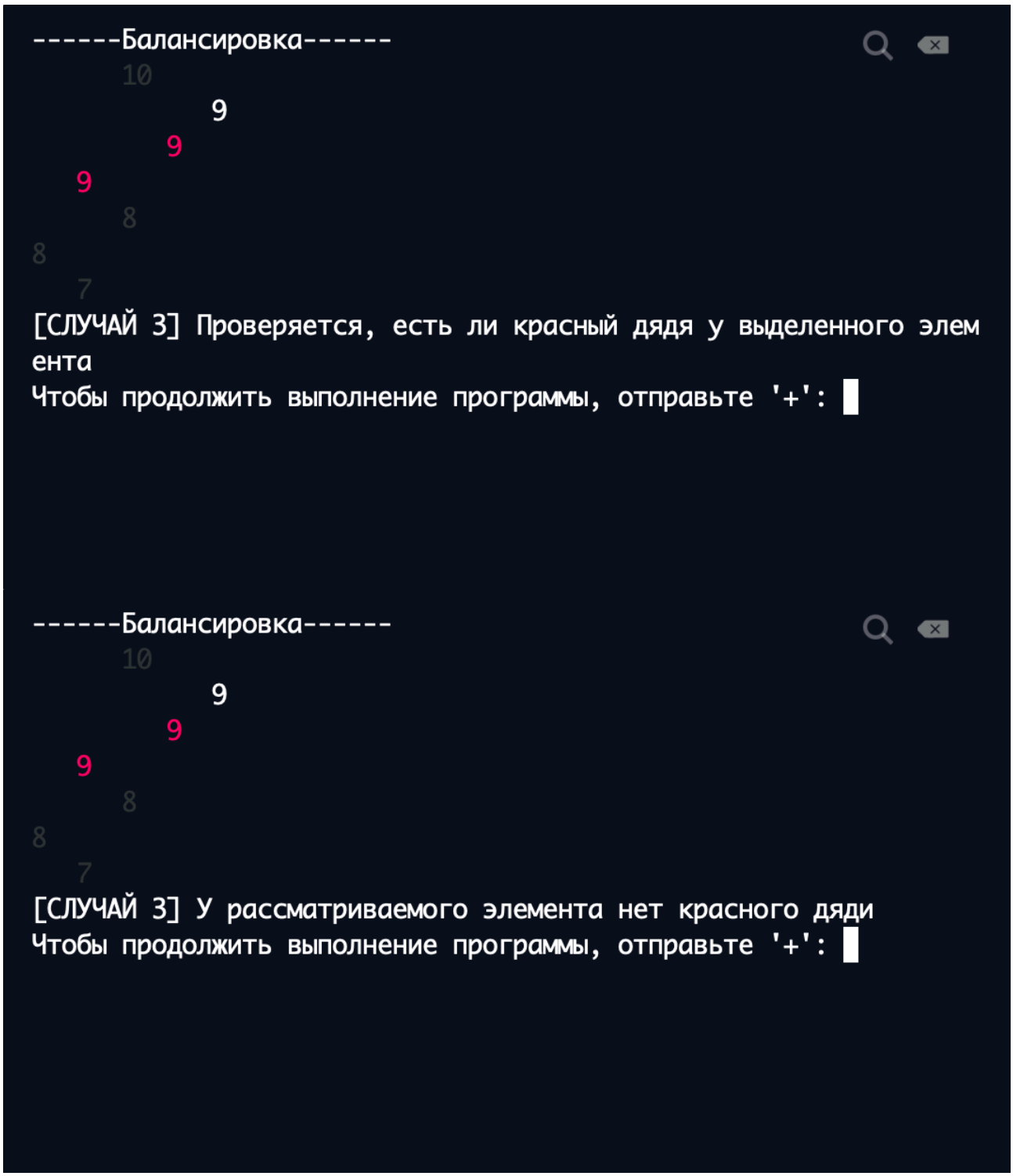


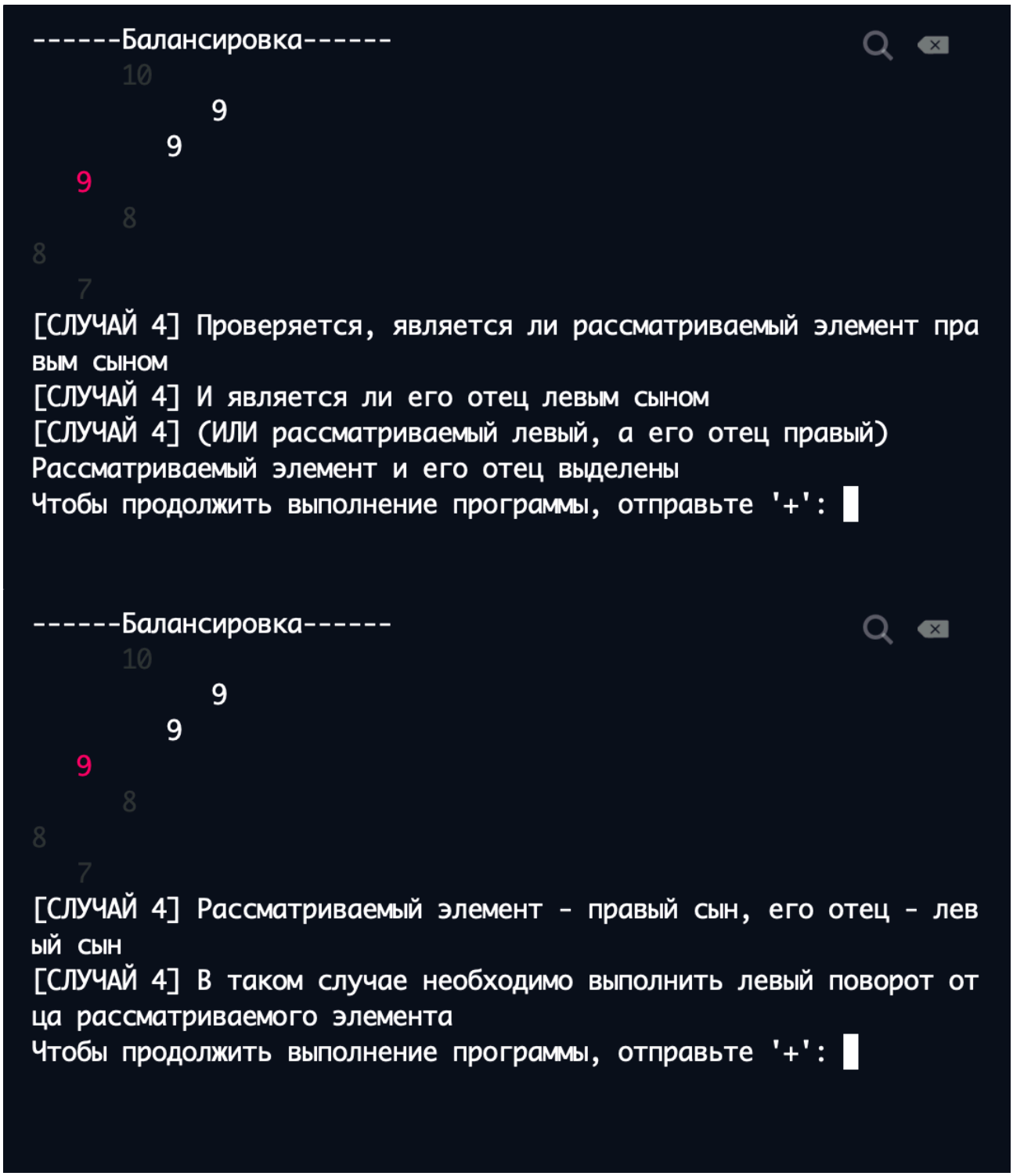
10
9
9
8
8
7
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+':

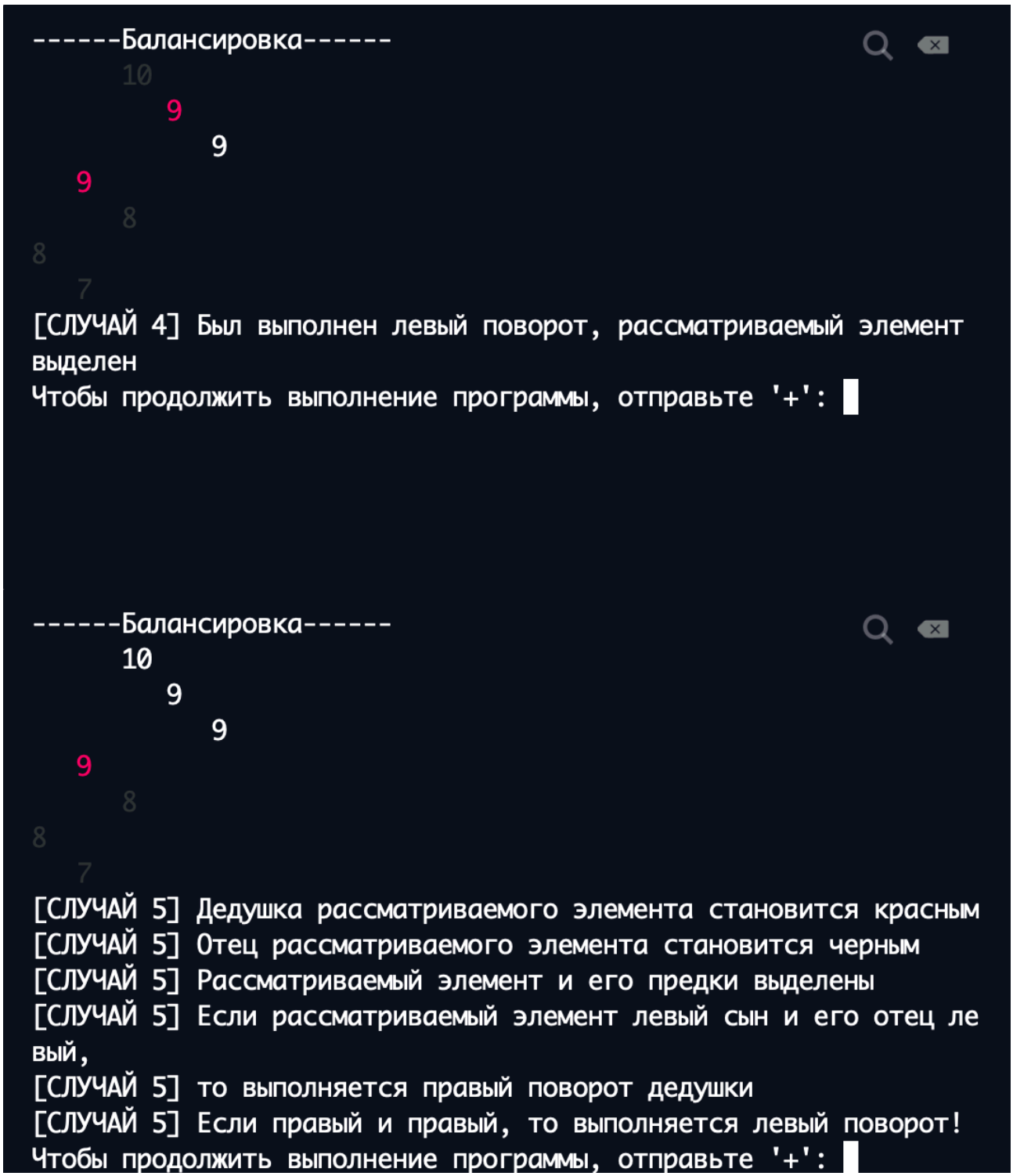
10
9
9
9
8
8
7
------Элемент вставлен-------
Чтобы продолжить выполнение программы, отправьте '+':

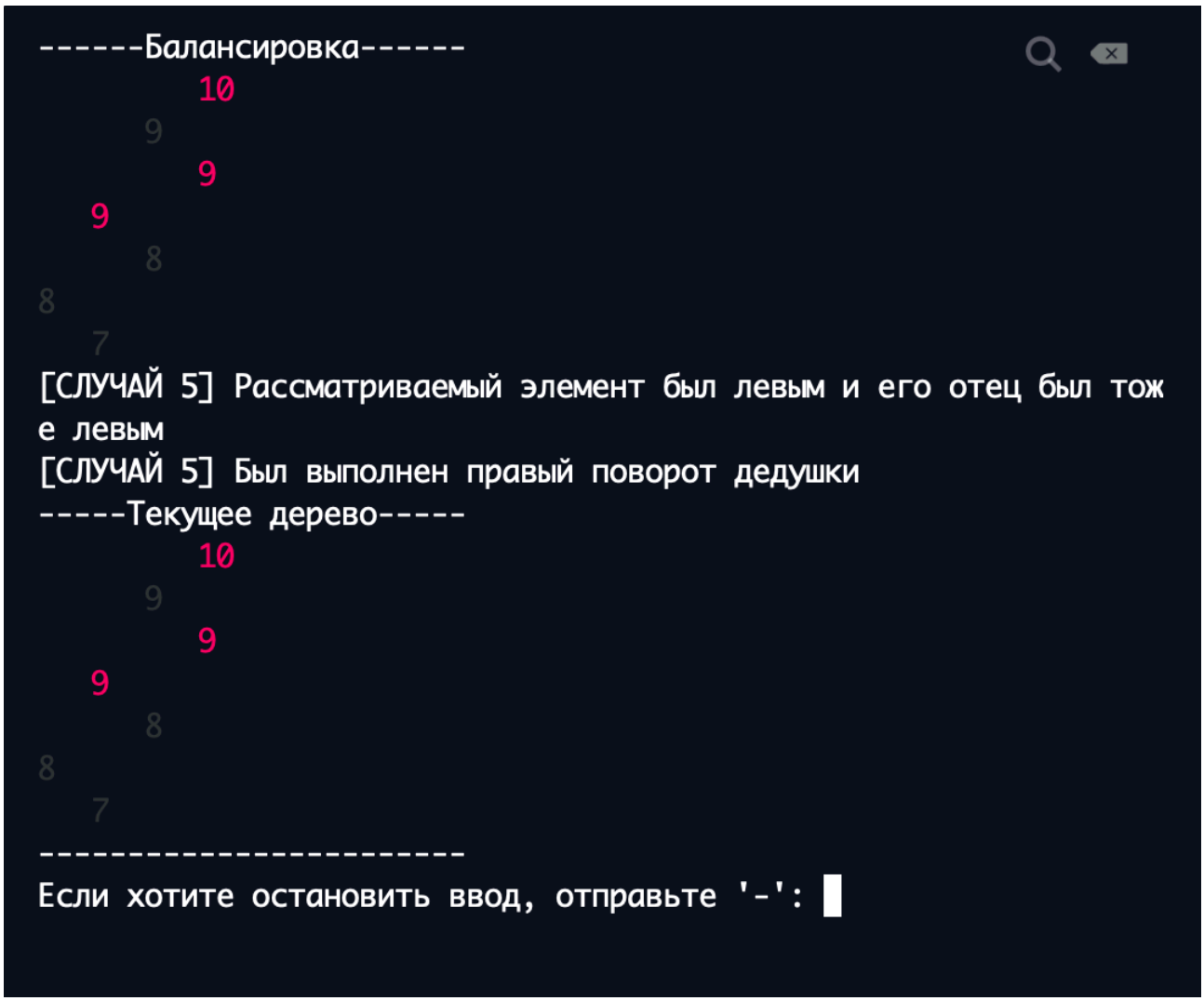

------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного элемента
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 1] У данного элемента есть родитель (выделен)!
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 2] Проверяется, является ли родитель рассматриваемого э
лемента черным
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель красный, с
войство нарушено!
Чтобы продолжить выполнение программы, отправьте '+':

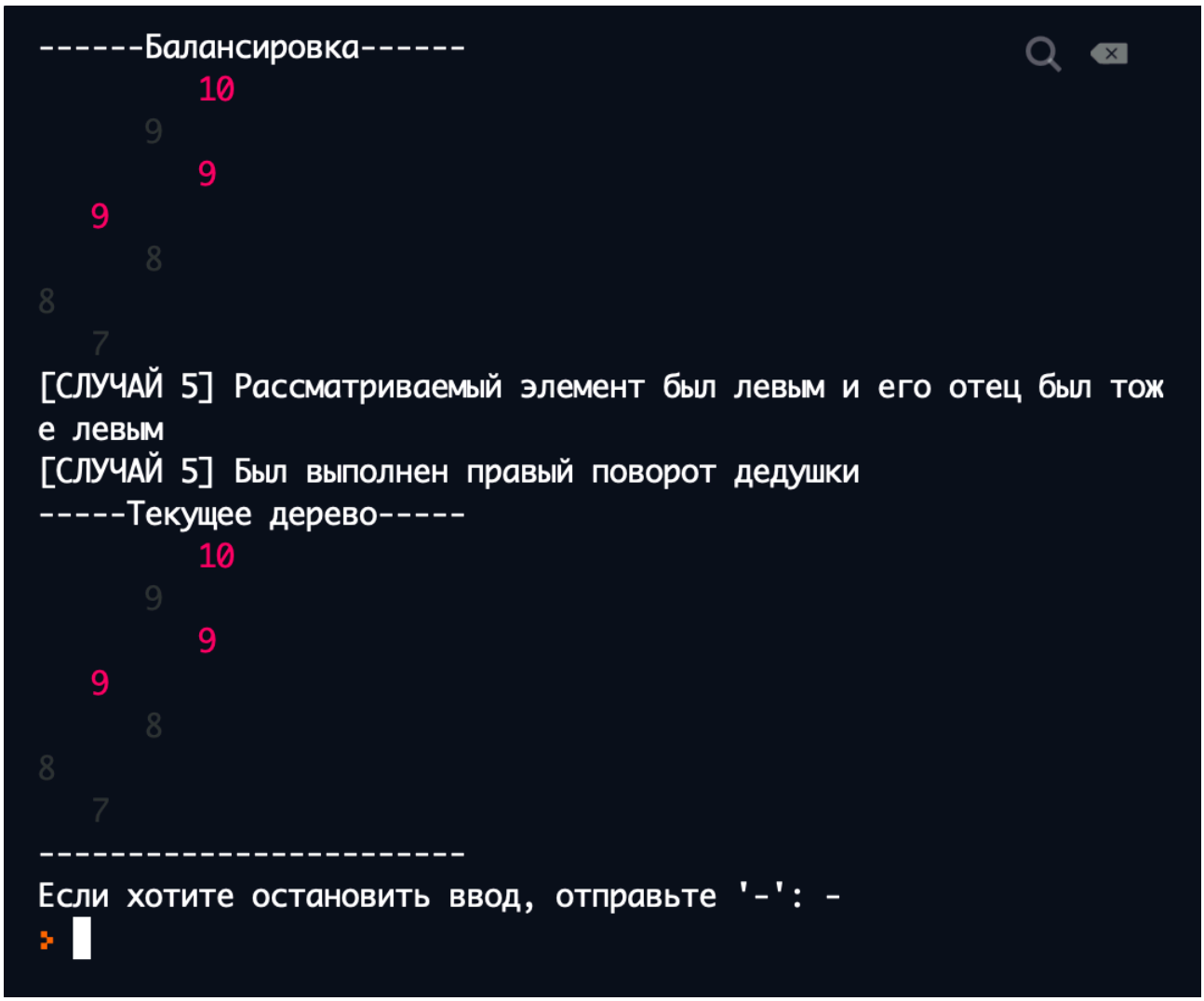

------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 3] Проверяется, есть ли красный дядя у выделенного элем
ента
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 3] У рассматриваемого элемента нет красного дяди
Чтобы продолжить выполнение программы, отправьте '+':


------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 4] Проверяется, является ли рассматриваемый элемент правым сыном
[СЛУЧАЙ 4] И является ли его отец левым сыном
[СЛУЧАЙ 4] (ИЛИ рассматриваемый левый, а его отец правый)
Рассматриваемый элемент и его отец выделены
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 4] Рассматриваемый элемент - правый сын, его отец - левый сын
[СЛУЧАЙ 4] В таком случае необходимо выполнить левый поворот от ца рассматриваемого элемента
Чтобы продолжить выполнение программы, отправьте '+':

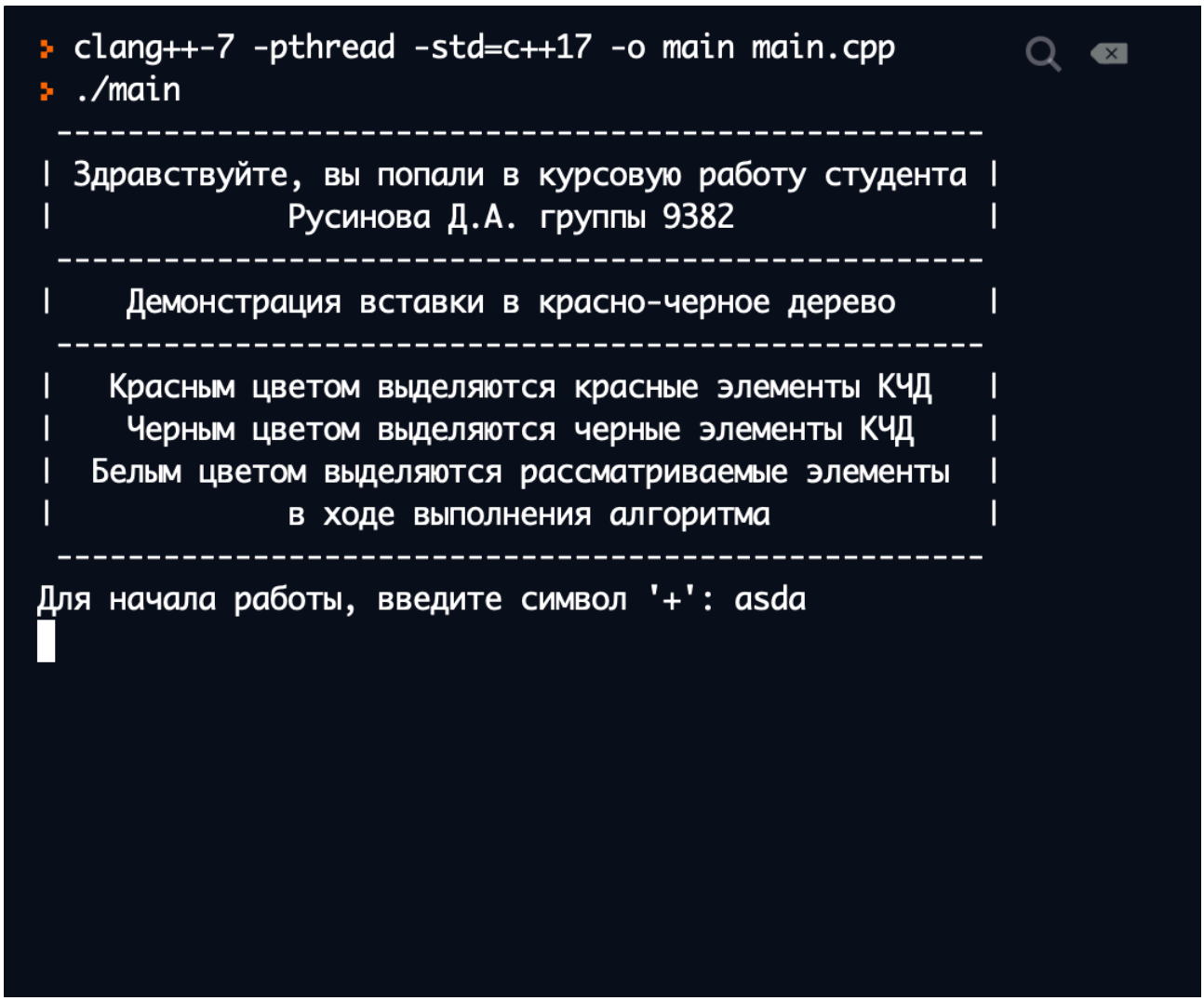

------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 4] Был выполнен левый поворот, рассматриваемый элемент выделен
Чтобы продолжить выполнение программы, отправьте '+':
------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 5] Дедушка рассматриваемого элемента становится красным
[СЛУЧАЙ 5] Отец рассматриваемого элемента становится черным
[СЛУЧАЙ 5] Рассматриваемый элемент и его предки выделены
[СЛУЧАЙ 5] Если рассматриваемый элемент левый сын и его отец левый,
[СЛУЧАЙ 5] то выполняется правый поворот дедушки
[СЛУЧАЙ 5] Если правый и правый, то выполняется левый поворот!
Чтобы продолжить выполнение программы, отправьте '+':

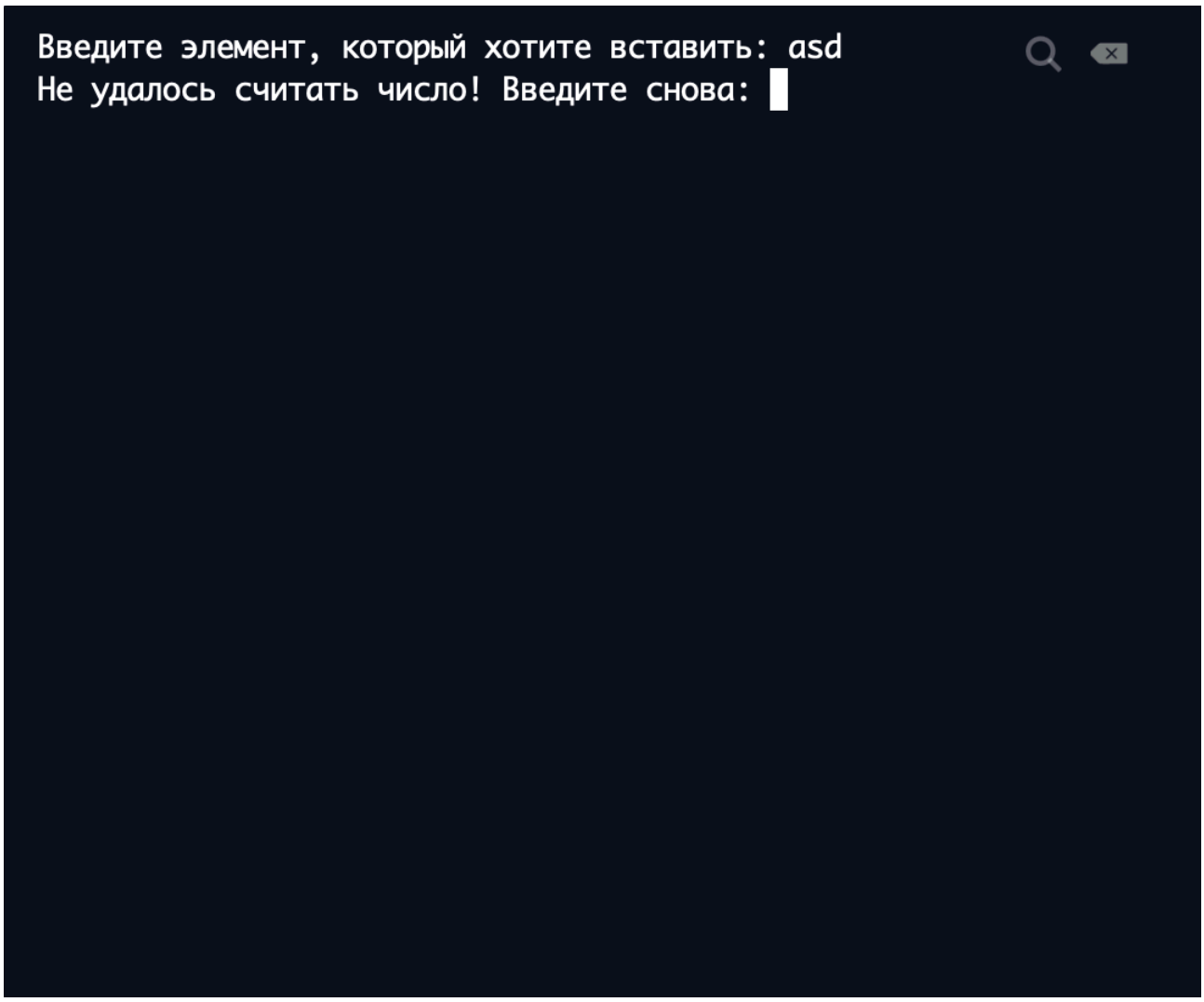

------Балансировка------
10
9
9
9
8
8
7
[СЛУЧАЙ 5] Рассматриваемый элемент был левым и его отец был тож
е левым
[СЛУЧАЙ 5] Был выполнен правый поворот дедушки
-----Текущее дерево-----
10
9
9
9
8
8
7

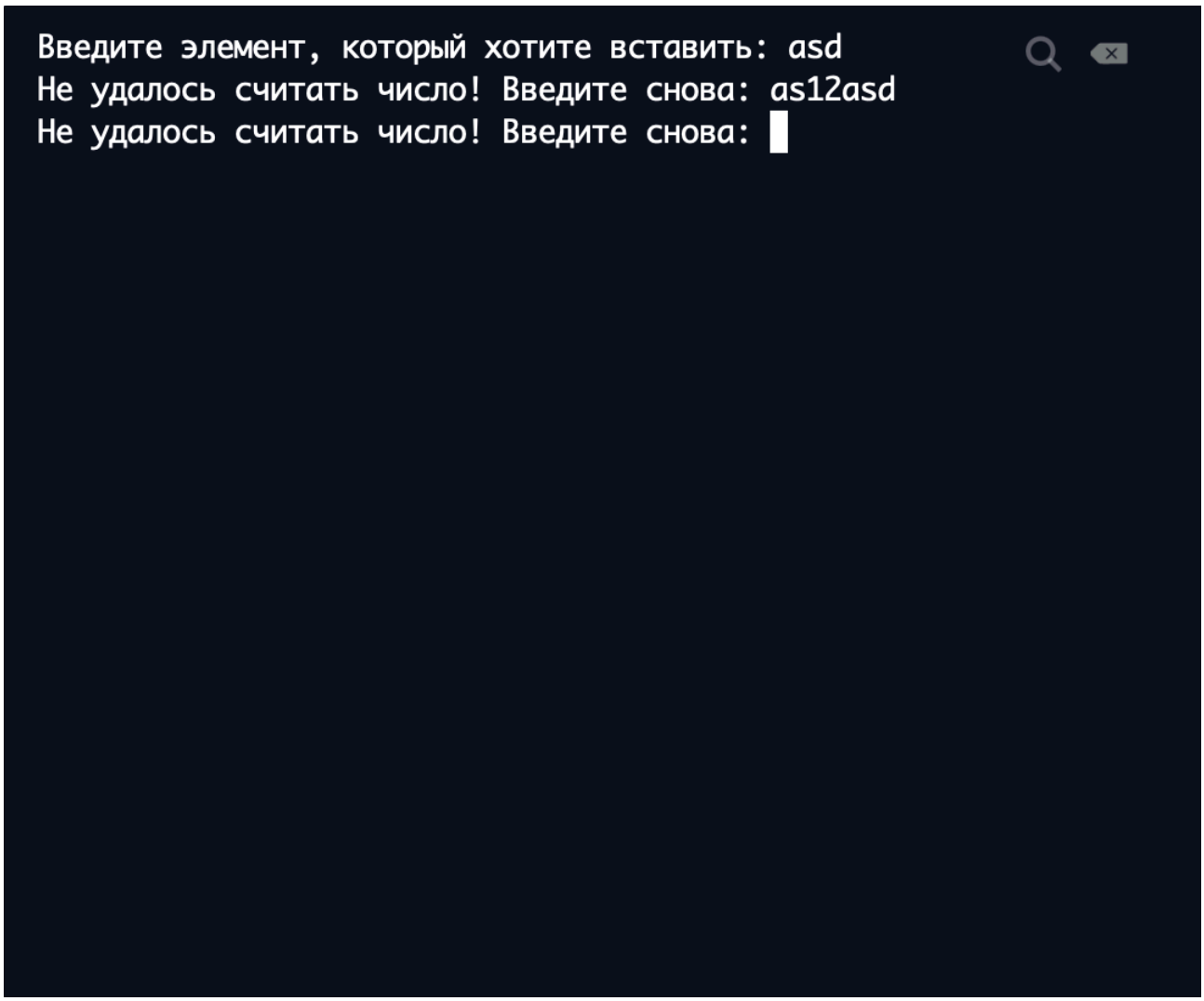
Если хотите остановить ввод, отправьте '-':

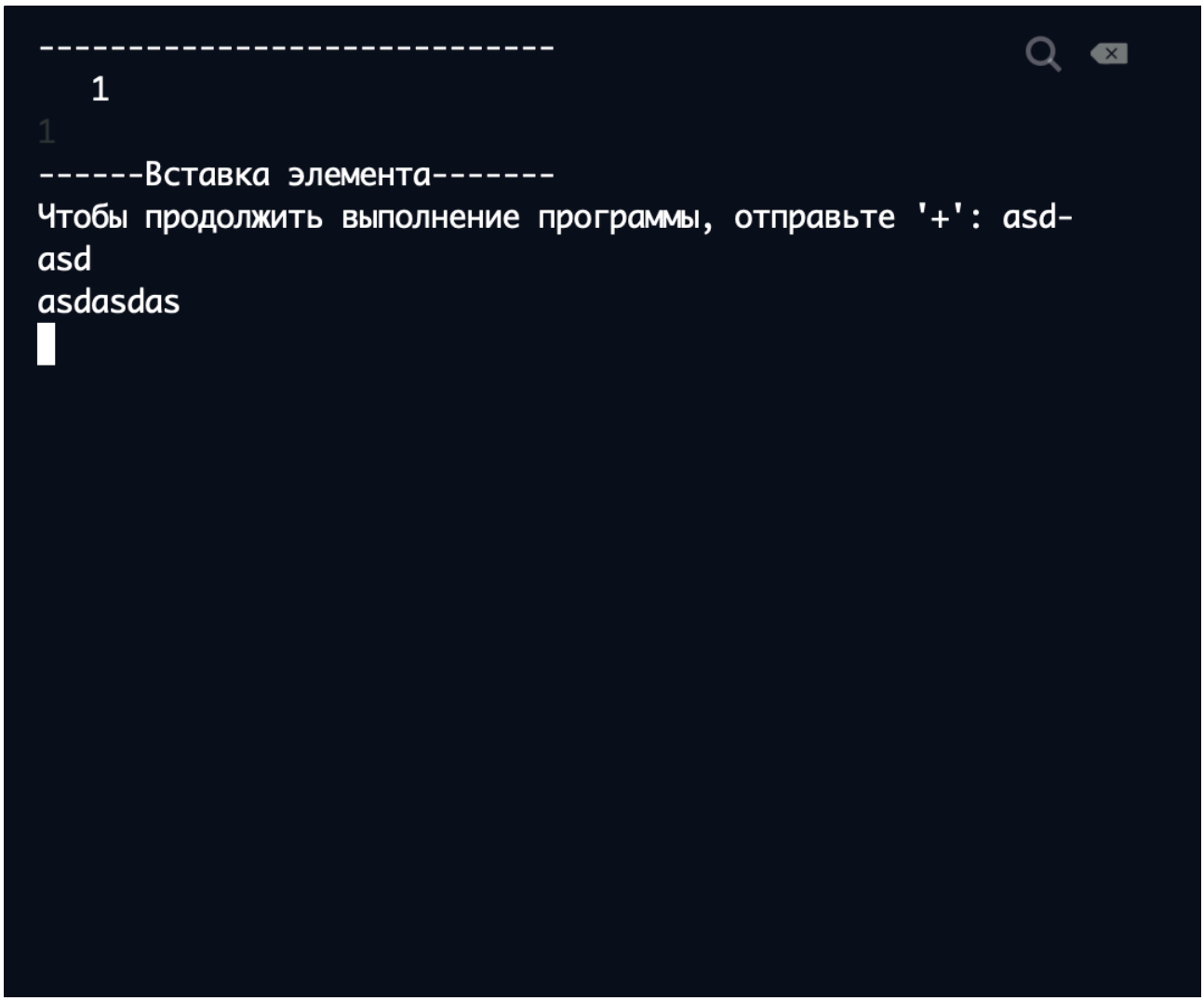


2. Проверка на некорректных данных


clang++-7 -pthread -std=c++17 -o main main.cpp
./main
Здравствуйте, вы попали в курсовую работу студента
Русинова Д.А. группы 9382
Демонстрация вставки в красно-черное дерево
Красным цветом выделяются красные элементы КЧД
Черным цветом выделяются черные элементы КЧД
Белым цветом выделяются рассматриваемые элементы
в ходе выполнения алгоритма
Для начала работы, введите символ '+': asda


Введите элемент, который хотите вставить: asd
Не удалось считать число! Введите снова:


Введите элемент, который хотите вставить: asd
Не удалось считать число! Введите снова: as12asd
Не удалось считать число! Введите снова:



1
1
------Вставка элемента-------
Чтобы продолжить выполнение программы, отправьте '+': asd-
asd
asdasdas

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "iostream"
#include "vector"
#include "algorithm"

#define BLACK true
#define RED false


template<typename Elem>
struct Node {
    Elem data{};
    Node* left = nullptr;
    Node* right = nullptr;
    Node* parent = nullptr;
    bool color{};
};

void printTree(Node<int>* tree, int level, std::vector<Node<int>*> specialNodes)
// печать дерева
{
    if(tree)
    {
        std::vector<Node<int>*>::iterator it;
        it = std::find(specialNodes.begin(), specialNodes.end(), tree);
        printTree(tree->right, level + 1, specialNodes);

        for (int i = 0; i < level; ++i) std::cout << "   ";
        if (it != specialNodes.end()) std::cout << "\x1b[38m" << tree->data <<
"\x1b[0m" << std::endl;
        else if (tree->color == BLACK) std::cout << "\x1b[30m" << tree->data <<
"\x1b[0m" << std::endl;
        else std::cout << "\x1b[31m" << tree->data << "\x1b[0m" << std::endl;
        printTree(tree->left, level + 1, specialNodes);
    }
}

void waitNextStep() {
    char s = 0;
    std::cout << "Чтобы продолжить выполнение программы, отправьте '+': ";
    while (s != '+') std::cin >> s;
    system("clear");
}

template<typename Elem>
class RBTree {
    Node<Elem>* root;

    Node<Elem>* getGrandparent(Node<Elem>* n) { // дедушка
        if ((n != nullptr) && (n->parent != nullptr)) return n->parent->parent;
        return nullptr;
    }

    Node<Elem>* getUncle(Node<Elem>* n) { // дядя
        Node<Elem> *g = getGrandparent(n);
        if (g == nullptr) return nullptr;
        if (n->parent == g->left) return g->right;
        return g->left;
```

```
    }

    void restoreRoot(Node<Elem>* n) { // восстановление корня
        while (n->parent) n = n->parent;
        root = n;
    }

    /*
     * Функция левого поворота
     *        n                     y
     *       / \                   / \
     *      T1  y     --->        n  T3
     *         / \                / \
     *        T2 T3              T1 T2
     *
     * В правую ветку N подставляется левая ветка Y
     * В левую ветку Y подставляется N
     *
     */

    void leftRotate(Node<Elem>* n) {
        Node<Elem>* y = n->right;

        y->parent = n->parent; /* при этом, возможно, y становится корнем дерева
*/
        if (n->parent != nullptr) {
            if (n->parent->left == n)
                n->parent->left = y;
            else
                n->parent->right = y;
        }

        n->right = y->left;
        if (y->left != nullptr)
            y->left->parent = n;

        n->parent = y;
        y->left = n;
        restoreRoot(n);
    }

    /*
     * Функция правого поворота
     *          n                         y
     *         / \                       / \
     *        y  T3       --->          T1  n
     *       / \                           / \
     *      T1 T2                         T2 T3
     *
     * В левую ветку N подставляется правая ветка Y
     * В правую ветку Y подставляется N
     *
     */

    void rightRotate(Node<Elem>* n) {
        Node<Elem>* y = n->left;

        y->parent = n->parent; /* при этом, возможно, pivot становится корнем
дерева */
        if (n->parent != nullptr) {
            if (n->parent->left==n)
                n->parent->left = y;
            else
```

```
                n->parent->right = y;
        }

        n->left = y->right;
        if (y->right != nullptr)
            y->right->parent = n;

        n->parent = y;
        y->right = n;
        restoreRoot(n);
    }

    // случай, когда нет корня
    void insertCase1(Node<Elem>* n) {
        waitNextStep();
        std::cout << "------Балансировка------" << std::endl;
        std::vector<Node<Elem>*> specialNodes {};
        specialNodes.push_back(n);
        printTree(root, 0, specialNodes);
        std::cout << "[СЛУЧАЙ 1] Проверяется, есть ли родитель у вставленного
элемента" << std::endl;

        waitNextStep();
        if (n->parent == nullptr) {
            specialNodes.clear();
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 1] У данного элемента нет родителя, поэтому он
становится корнем!" << std::endl;
            n->color = BLACK;
        } else {
            specialNodes.clear();
            specialNodes.push_back(n->parent);
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 1] У данного элемента есть родитель
(выделен)!" << std::endl;
            insertCase2(n);
        }
    }

    // случай, когда отец черный
    void insertCase2(Node<Elem>* n) {
        waitNextStep();
        std::cout << "------Балансировка------" << std::endl;
        std::vector<Node<Elem>*> specialNodes {};
        specialNodes.push_back(n);
        printTree(root, 0, specialNodes);
        std::cout << "[СЛУЧАЙ 2] Проверяется, является ли родитель
рассматриваемого элемента черным" << std::endl;

        waitNextStep();
        specialNodes.clear();
        if (n->parent->color == BLACK) {
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель
черный, свойство не нарушено!" << std::endl;
            return;
        } else {
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 2] Рассматриваемый элемент красный, родитель
```

```
красный, свойство нарушено!" << std::endl;
            insertCase3(n);
        }
    }

    // случай, когда отец красный и есть красный дядя
    void insertCase3(Node<Elem>* n) {
        waitNextStep();
        std::cout << "------Балансировка------" << std::endl;
        std::vector<Node<Elem>*> specialNodes {};
        specialNodes.push_back(n);
        printTree(root, 0, specialNodes);
        std::cout << "[СЛУЧАЙ 3] Проверяется, есть ли красный дядя у выделенного
элемента" << std::endl;

        Node<Elem>* u = getUncle(n), *g;
        waitNextStep();
        specialNodes.clear();
        if ((u != nullptr) && (u->color == RED)) {
            // && (n->parent->color == RED) Второе условие проверяется в
insertCase2, то есть родитель уже является красным.

            std::cout << "------Балансировка------" << std::endl;
            specialNodes.push_back(n->parent);
            specialNodes.push_back(u);
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 3] У рассматриваемого элемента есть красный
отец и красный дядя" << std::endl;
            std::cout << "В этом случае отец становится черным, дядя становится
черным, дедушка красным" << std::endl;
            std::cout << "И операция балансировки повторяется для дедушки" <<
std::endl;

            n->parent->color = BLACK;
            u->color = BLACK;
            g = getGrandparent(n);
            g->color = RED;
            insertCase1(g);
        } else {
            std::cout << "------Балансировка------" << std::endl;
            specialNodes.push_back(n);
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 3] У рассматриваемого элемента нет красного
дяди" << std::endl;
            insertCase4(n);
        }
    }

    // случай, когда нет красного дяди
    void insertCase4(Node<Elem>* n) {
        Node<Elem>* g = getGrandparent(n);

        waitNextStep();
        std::cout << "------Балансировка------" << std::endl;
        std::vector<Node<Elem>*> specialNodes {};

        specialNodes.push_back(n);
        specialNodes.push_back(n->parent);

        printTree(root, 0, specialNodes);
        std::cout << "[СЛУЧАЙ 4] Проверяется, является ли рассматриваемый
элемент правым сыном" << std::endl;
        std::cout << "[СЛУЧАЙ 4] И является ли его отец левым сыном" <<
```

```
std::endl;
        std::cout << "[СЛУЧАЙ 4] (ИЛИ рассматриваемый левый, а его отец правый)"
<< std::endl;
        std::cout << "Рассматриваемый элемент и его отец выделены" << std::endl;

        waitNextStep();
        // n правый сын и отец левый сын
        if ((n == n->parent->right) && (n->parent == g->left)) {
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 4] Рассматриваемый элемент - правый сын, его
отец - левый сын" << std::endl;
            std::cout << "[СЛУЧАЙ 4] В таком случае необходимо выполнить левый
поворот отца рассматриваемого элемента" << std::endl;
            leftRotate(n->parent);

            waitNextStep();
            specialNodes.clear();
            n = n->left;
            specialNodes.push_back(n);
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 4] Был выполнен левый поворот, рассматриваемый
элемент выделен" << std::endl;

            // n левый сын и отец правый сын
        } else if ((n == n->parent->left) && (n->parent == g->right)) {
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 4] Рассматриваемый элемент - левый сын, его
отец - правый сын" << std::endl;
            std::cout << "[СЛУЧАЙ 4] В таком случае необходимо выполнить правый
поворот отца рассматриваемого элемента" << std::endl;
            rightRotate(n->parent);

            waitNextStep();
            specialNodes.clear();
            n = n->right;
            specialNodes.push_back(n);
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 4] Был выполнен правый поворот,
рассматриваемый элемент выделен" << std::endl;
        } else {
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 4] Заданные требования не выполнены для
элементов" << std::endl;
        }
        insertCase5(n);
    }

    // продолжение случая 4
    void insertCase5(Node<Elem>* n)
    {
        Node<Elem>* g = getGrandparent(n);

        waitNextStep();
        std::cout << "------Балансировка------" << std::endl;
        std::vector<Node<Elem>*> specialNodes {};

        specialNodes.push_back(n);
        specialNodes.push_back(n->parent);
```

```
        specialNodes.push_back(g);

        printTree(root, 0, specialNodes);

        std::cout << "[СЛУЧАЙ 5] Дедушка рассматриваемого элемента становится
красным" << std::endl;
        std::cout << "[СЛУЧАЙ 5] Отец рассматриваемого элемента становится
черным" << std::endl;
        std::cout << "[СЛУЧАЙ 5] Рассматриваемый элемент и его предки выделены"
<< std::endl;
        std::cout << "[СЛУЧАЙ 5] Если рассматриваемый элемент левый сын и его
отец левый," << std::endl;
        std::cout << "[СЛУЧАЙ 5] то выполняется правый поворот дедушки" <<
std::endl;
        std::cout << "[СЛУЧАЙ 5] Если правый и правый, то выполняется левый
поворот!" << std::endl;

        n->parent->color = BLACK;
        g->color = RED;
        // n левый сын и отец левый сын
        waitNextStep();
        specialNodes.clear();
        if ((n == n->parent->left) && (n->parent == g->left)) {
            rightRotate(g);
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 5] Рассматриваемый элемент был левым и его
отец был тоже левым" << std::endl;
            std::cout << "[СЛУЧАЙ 5] Был выполнен правый поворот дедушки" <<
std::endl;
        } else { // n правый сын и отец правый сын
            leftRotate(g);
            std::cout << "------Балансировка------" << std::endl;
            printTree(root, 0, specialNodes);
            std::cout << "[СЛУЧАЙ 5] Рассматриваемый элемент был правым и его
отец был тоже правым" << std::endl;
            std::cout << "[СЛУЧАЙ 5] Был выполнен левый поворот дедушки" <<
std::endl;
        }
    }


public:
    RBTree() : root(nullptr) {}
    ~RBTree() { delete root; }

    Node<Elem>* getRoot() { return root; }

    void insert(Elem stuff) { // Вставка элемента
        auto newNode = new Node<Elem>();
        newNode->data = stuff;
        newNode->color = RED;

        auto linker = root;
        std::vector<Node<int>*> specialNodes = {};


        // если node < linker, то идем в левую ветку
        // если node >= linker, то идем в правую ветку
        // когда нет след ветки, то вставляем туда элемент
        while (linker) {
            std::cout << "-----------------------------" << std::endl;
            specialNodes.push_back(linker);
```

```
            printTree(getRoot(), 0, specialNodes);
            std::cout << "------Вставка элемента-------" << std::endl;


            if (newNode->data < linker->data) {
                if (!linker->left) {
                    linker->left = newNode;
                    newNode->parent = linker;

                    specialNodes.clear();
                    waitNextStep();
                    std::cout << "-----------------------------" << std::endl;
                    specialNodes.push_back(newNode);
                    printTree(getRoot(), 0, specialNodes);
                    std::cout << "------Элемент вставлен-------" << std::endl;

                    break;
                } else linker = linker->left;
            } else {
                if (!linker->right) {
                    linker->right = newNode;
                    newNode->parent = linker;

                    specialNodes.clear();
                    waitNextStep();
                    std::cout << "-----------------------------" << std::endl;
                    specialNodes.push_back(newNode);
                    printTree(getRoot(), 0, specialNodes);
                    std::cout << "------Элемент вставлен-------" << std::endl;

                    break;
                } else linker = linker->right;
            }
            specialNodes.clear();
            waitNextStep();
        }

        if (!newNode->parent) {
            newNode->color = BLACK;
            std::cout << "------Вставлен корень--------" << std::endl;
            specialNodes.push_back(root);
            printTree(newNode, 0, specialNodes);
            std::cout << "-----------------------------" << std::endl;
            specialNodes.clear();
        } else insertCase1(newNode);
        restoreRoot(newNode);
    }
};


char userInput(RBTree<int>* tree) {
    system("clear");
    int stuff = 1;
    std::cout << "Введите элемент, который хотите вставить: ";

    std::cin >> stuff;

    while (std::cin.fail()) {
        std::cout << "Не удалось считать число! Введите снова: ";
        std::cin.clear();
        std::cin.ignore(10, '\n');
        std::cin >> stuff;
    }
```

```
    tree->insert(stuff);
    std::vector<Node<int>*> specialNodes;
    std::cout << "-----Текущее дерево-----" << std::endl;
    printTree(tree->getRoot(), 0, specialNodes);
    std::cout << "------------------------" << std::endl;

    char flag;
    std::cout << "Если хотите остановить ввод, отправьте '-': ";
    std::cin >> flag;
    return flag;
}


int main()
{
    std::cout << " ------------------------------------------------------ " <<
std::endl;
    std::cout << "| Здравствуйте, вы попали в курсовую работу студента |" <<
std::endl;
    std::cout << "|             Русинова Д.А. группы 9382              |" <<
std::endl;
    std::cout << " ------------------------------------------------------ " <<
std::endl;
    std::cout << "|    Демонстрация вставки в красно-черное дерево     |" <<
std::endl;
    std::cout << " ------------------------------------------------------ " <<
std::endl;
    std::cout << "|   Красным цветом выделяются красные элементы КЧД   |" <<
std::endl;
    std::cout << "|    Черным цветом выделяются черные элементы КЧД    |" <<
std::endl;
    std::cout << "|  Белым цветом выделяются рассматриваемые элементы  |" <<
std::endl;
    std::cout << "|             в ходе выполнения алгоритма            |" <<
std::endl;
    std::cout << " ------------------------------------------------------ " <<
std::endl;

    char start = 0;
    std::cout << "Для начала работы, введите символ '+': ";
    while (start != '+') std::cin >> start;

    auto* tree = new RBTree<int>;
    char flag = userInput(tree);
    while (flag != '-') flag = userInput(tree);

    return 0;
}
```