МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

федеральное государственное бюджетное образовательное учреждение высшего образования

**«Вологодский государственный университет»**

**Институт математики, естественных и компьютерных наук**

**Информатика и вычислительная техника**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К РАСЧЕТНО-ГРАФИЧЕСКОЙ РАБОТЕ**

Разработка визуализатора структуры данных «Декартово дерево».

Дисциплина: «Алгоритмы и структуры данных»

Направление подготовки: 09.03.01. Информатика и вычислительная техника

|  |  |
| --- | --- |
| Руководитель | Андрианов И.А. |
| Выполнили студенты | Пчелкина О.С. |
| Группа, курс | ВМ-31 |
| Дата сдачи | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Дата защиты | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись преподавателя)* |

Вологда

2022 г.

Оглавление

[1. Структура данных декартово дерево 4](#_Toc104421566)

[1.1. Операции в декартовом дереве 4](#_Toc104421567)

[1.2. Построение декартова дерева 6](#_Toc104421568)

[2. Разработка визуализатора 8](#_Toc104421569)

[2.1. Выбор средств разработки 8](#_Toc104421570)

[2.2. Проектирование интерфейса пользователя 8](#_Toc104421571)

[2.3. Проектирование классов визуализатора 10](#_Toc104421572)

[2.4. Тестирование разработанного визуализатора 12](#_Toc104421573)

[Заключение 17](#_Toc104421574)

[Список литературы и Интернет-ресурсов 18](#_Toc104421575)

**Введение**

Задача данной расчетно-графической работы – разработка визуализатора структуры данных декартово дерево. Визуализатор – это программа с графическим интерфейсом пользователя, наглядно демонстрирующая работу алгоритма как в пошаговом режиме, так и в автоматическом. При этом на каждом шаге даются пояснения выполненных действий.

# Структура данных декартово дерево

Декартово дерево или дерамида (англ. Treap) — это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу (отсюда и второе её название: treap (tree + heap) и дерамида (дерево + пирамида).

Более строго, это бинарное дерево, в узлах которого хранятся пары (x,y), где x — это ключ, а y — это приоритет. Также оно является двоичным деревом поиска по x и пирамидой по y. Предполагая, что все x и все y являются различными, получаем, что если некоторый элемент дерева содержит (x0,y0), то у всех элементов в левом поддереве x<x0, у всех элементов в правом поддереве x>x0, а также и в левом, и в правом поддереве имеем: y<y0.

Дерамиды были предложены Сиделем (Siedel) и Арагон (Aragon) в 1996 г.

## Операции в декартовом дереве

Split.

Операция split (разрезать) позволяет сделать следующее: разрезать исходное дерево T по ключу k. Возвращать она будет такую пару деревьев ⟨T1,T2⟩, что в дереве T1 ключи меньше k, а в дереве T2 все остальные: split(T,k)→⟨T1,T2⟩.

Эта операция устроена следующим образом.

Рассмотрим случай, в котором требуется разрезать дерево по ключу, большему ключа корня. Посмотрим, как будут устроены результирующие деревья T1 и T2:

T1: левое поддерево T1 совпадёт с левым поддеревом T. Для нахождения правого поддерева T1, нужно разрезать правое поддерево T на TR1 и TR2 по ключу k и взять TR1.

T2 совпадёт с TR2.

Случай, в котором требуется разрезать дерево по ключу, меньше либо равному ключа в корне, рассматривается симметрично.

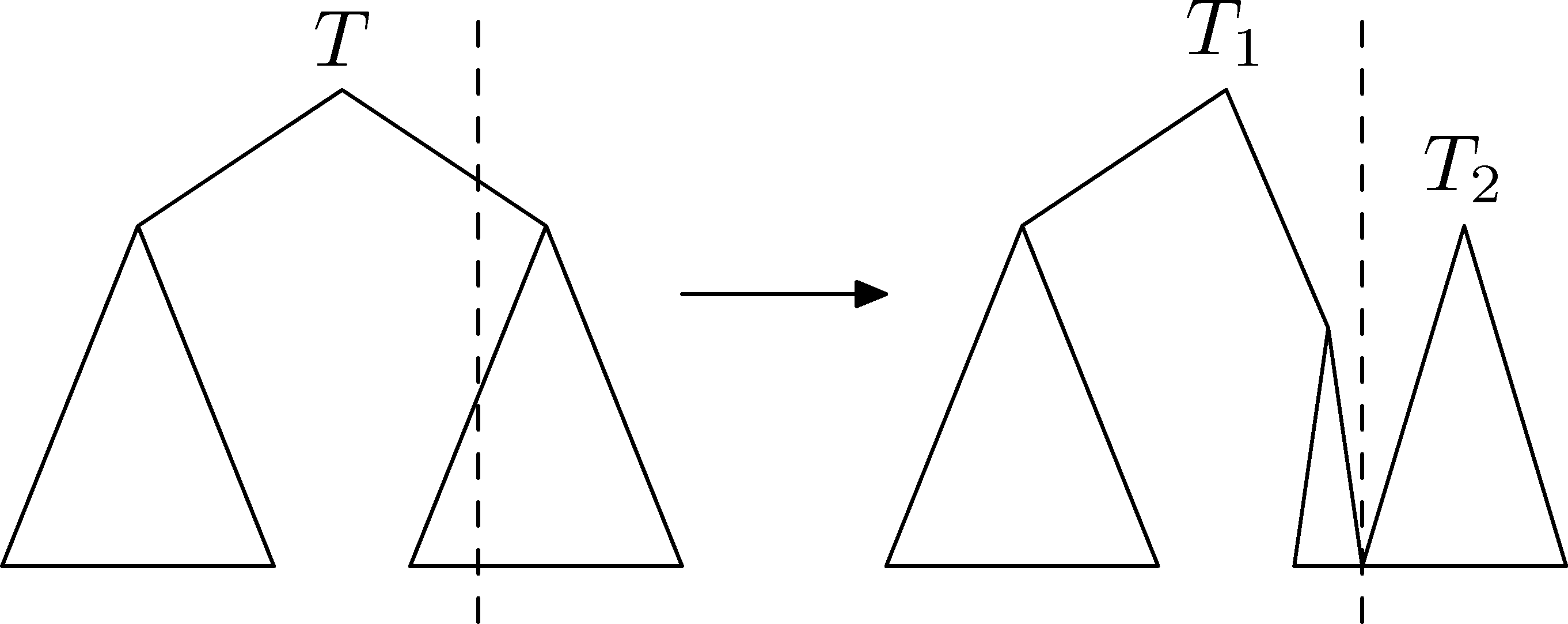


Рисунок 1 – Операция Split.

Время работы операции split. Во время выполнения вызывается одна операция split для дерева хотя бы на один меньшей высоты и делается ещё O(1) операций. Тогда итоговая трудоёмкость этой операции равна O(h), где h — высота дерева.

Merge.

Рассмотрим вторую операцию с декартовыми деревьями — merge (слить).

С помощью этой операции можно слить два декартовых дерева в одно. Причём, все ключи в первом(левом) дереве должны быть меньше, чем ключи во втором(правом). В результате получается дерево, в котором есть все ключи из первого и второго деревьев: merge(T1,T2)→{T}.

Рассмотрим принцип работы этой операции. Пусть нужно слить деревья T1 и T2. Тогда, очевидно, у результирующего дерева T есть корень. Корнем станет вершина из T1 или T2 с наибольшим приоритетом y. Но вершина с самым большим y из всех вершин деревьев T1 и T2 может быть только либо корнем T1, либо корнем T2. Рассмотрим случай, в котором корень T1 имеет больший y, чем корень T2. Случай, в котором корень T2 имеет больший y, чем корень T1, симметричен этому.

Если y корня T1 больше y корня T2, то он и будет являться корнем. Тогда левое поддерево T совпадёт с левым поддеревом T1. Справа же нужно подвесить объединение правого поддерева T1 и дерева T2.

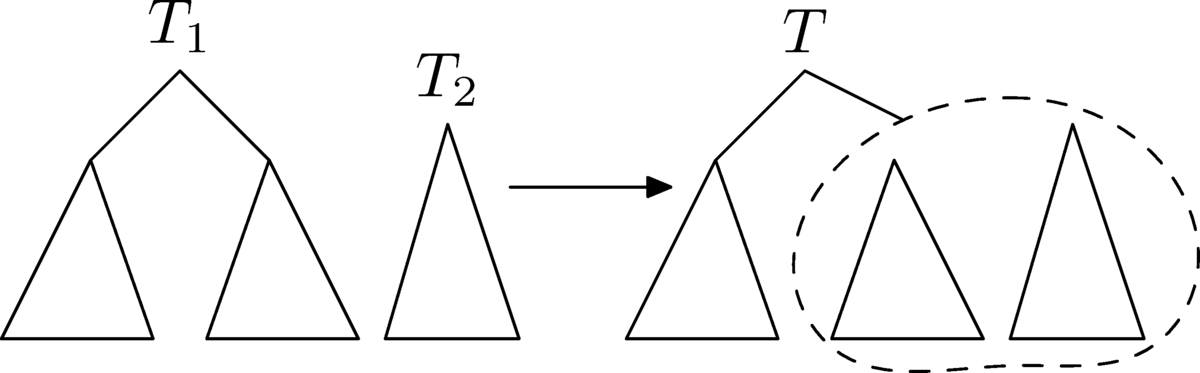


Рисунок 2 – Операция Merge.

Рассуждая аналогично операции split, приходим к выводу, что трудоёмкость операции merge равна O(h), где h — высота дерева.

Insert.

Операция insert(T,k) добавляет в дерево T элемент k, где k.x — ключ, а k.y — приоритет.

Представим что элемент k, это декартово дерево из одного элемента, и для того чтобы его добавить в наше декартово дерево T, очевидно, нам нужно их слить. Но T может содержать ключи как меньше, так и больше ключа k.x, поэтому сначала нужно разрезать T по ключу k.x.

Реализация №1

* Разобьём наше дерево по ключу, который мы хотим добавить, то есть split(T,k.x)→⟨T1,T2⟩.
* Сливаем первое дерево с новым элементом, то есть merge(T1,k)→T1.
* Сливаем получившиеся дерево со вторым, то есть merge(T1,T2)→T.

Реализация №2

* Сначала спускаемся по дереву (как в обычном бинарном дереве поиска по k.x), но останавливаемся на первом элементе, в котором значение приоритета оказалось меньше k.y.
* Теперь вызываем split(T,k.x)→⟨T1,T2⟩ от найденного элемента (от элемента вместе со всем его поддеревом)
* Полученные T1 и T2 записываем в качестве левого и правого сына добавляемого элемента.
* Полученное дерево ставим на место элемента, найденного в первом пункте.

В первой реализации два раза используется merge, а во второй реализации слияние вообще не используется.

## Построение декартова дерева

Пусть нам известно из каких пар (xi,yi) требуется построить декартово дерево, причём также известно, что x1<x2<…<xn.

Алгоритм за O(n logn)

Отсортируем все приоритеты по убыванию за O(nlogn) и выберем первый из них, пусть это будет yk. Сделаем (xk,yk) корнем дерева. Проделав то же самое с остальными вершинами получим левого и правого сына (xk,yk). В среднем высота Декартова дерева logn и на каждом уровне мы сделали O(n) операций. Значит такой алгоритм работает за O(nlogn).

Другой алгоритм за O(n logn)

Отсортируем пары (xi,yi) по убыванию xi и положим их в очередь. Сперва достанем из очереди первые 2 элемента и сольём их в дерево и положим в конец очереди, затем сделаем то же самое со следующими двумя и т.д. Таким образом, мы сольём сначала n деревьев размера 1, затем n2 деревьев размера 2 и так далее. При этом на уменьшение размера очереди в два раза мы будем тратить суммарно O(n) время на слияния, а всего таких уменьшений будет logn. Значит полное время работы алгоритма будет O(nlogn).

Алгоритм за O(n)

Будем строить дерево слева направо, то есть начиная с (x1,y1) по (xn,yn), при этом помнить последний добавленный элемент (xk,yk). Он будет самым правым, так как у него будет максимальный ключ, а по ключам декартово дерево представляет собой двоичное дерево поиска. При добавлении (xk+1,yk+1), пытаемся сделать его правым сыном (xk,yk), это следует сделать если yk>yk+1, иначе делаем шаг к предку последнего элемента и смотрим его значение y. Поднимаемся до тех пор, пока приоритет в рассматриваемом элементе меньше приоритета в добавляемом, после чего делаем (xk+1,yk+1) его правым сыном, а предыдущего правого сына делаем левым сыном (xk+1,yk+1).

Заметим, что каждую вершину мы посетим максимум дважды: при непосредственном добавлении и, поднимаясь вверх (ведь после этого вершина будет лежать в чьём-то левом поддереве, а мы поднимаемся только по правому). Из этого следует, что построение происходит за O(n).

# Разработка визуализатора

## Выбор средств разработки

Для разработки визуализатора был выбран язык программирования C#, так как это наиболее изученный мною языков. Данный язык программирования имеет необходимые качества для создания программы визуализации, а именно:

• Удобство создания GUI

• Функциональная полнота

• Кроссплатформенность

Само приложение будет написано на Windows Presentation Foundation. Windows Presentation Foundation (WPF) — аналог WinForms, система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе. NET Framework (начиная с версии 3.0), использующая язык XAML.

## Проектирование интерфейса пользователя

Визуализатор должен представлять следующие возможности:

1. генерация дерева;
2. задание дерева вручную;
3. пошаговое выполнение;
4. автоматическое выполнение;
5. отображение результата;
6. графическое представление дерева.

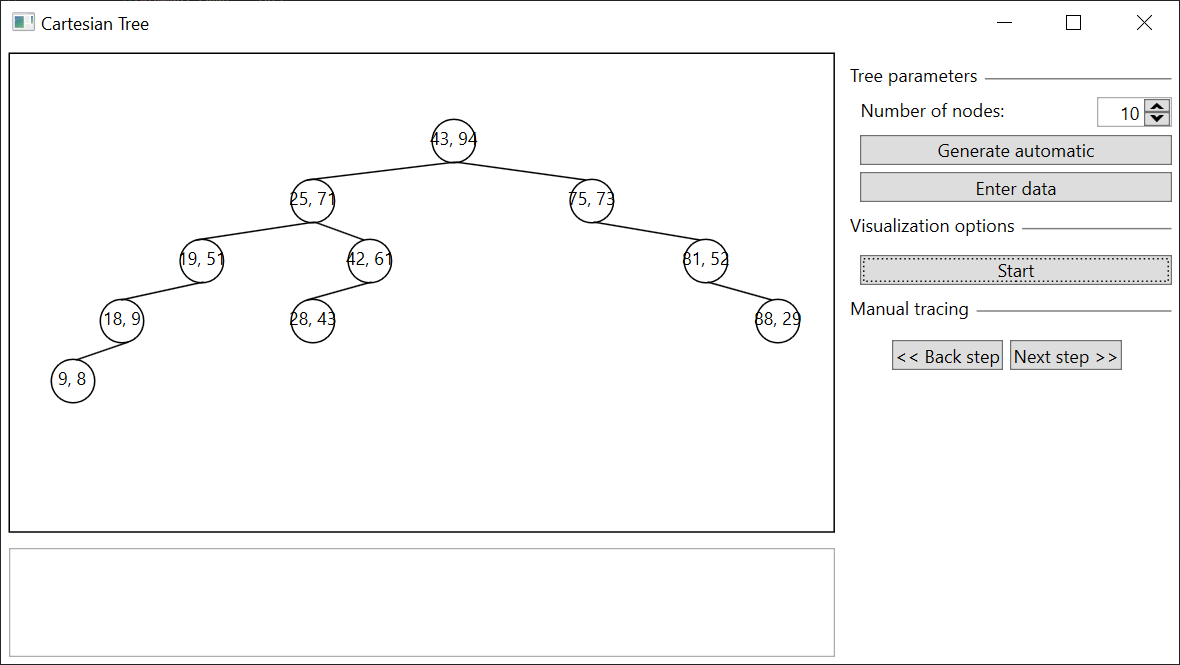


Рисунок 3 – Интерфейс пользователя

Интерфейс пользователя предоставляет возможность задания размера дерева, генерацию дерева из случайно заданных элементов, так же введенных пользователем элементов. При нажатии на кнопку «Enter data» открывается окна для ввода ключа и приоритета дерева (рис.4,5).

Рисунок – 4. Ввод ключа.

Рисунок – 5. Ввод приоритета.

## Проектирование классов визуализатора

При разработке визуализатора были разработаны следующие классы:

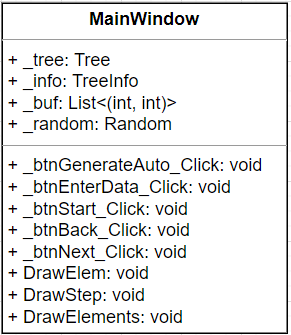
* класс главного окна MainWindow (рисунок 11);
* класс дерева Tree (рисунок 12);
* класс информации о дереве TreeInfo (рисунок 13);

Рисунок 6 – Класс MainWindow

Класс MainWindow содержит указатели дерево (\_tree) и класс – информацию о дереве (\_info). Также класс содержит обработчики нажатия кнопок пользовательского интерфейса:

* обработчик кнопки «Generate automatic» void \_btnGeneraetAuto\_Click();
* обработчик кнопки «Enter data» void \_btnEnterData\_Click
* обработчик кнопки «Start» void \_btnStart\_Click;
* обработчик кнопки «Next step >>» void \_btnBack\_Click ();
* обработчик кнопки «Back step <<» void \_btnNext\_Click

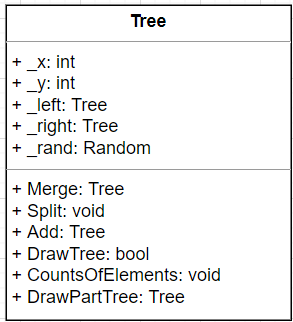


Рисунок 6 – Класс Tree

Класс Tree содержит ключ (\_x), приоритет(\_y), левого потомка(\_left), правого(\_right). Также класс содержит методы:

* метод слияния Tree Merge(Tree L, Tree R);
* метод разрезать дерево void Split(int x, out Tree L, out Tree R);
* метод добавления элемента в дерево Tree Add(int x);
* метод добавления элемента в дерево Tree Add(int x, int y);
* метод количества элементов void CountsOfElements(Tree root, ref List<(int, int)> elements, ref List<(int, int)> elemets\_prior, int lvl, ref List<List<(int, int)>> drawed\_elem)
* метод отрисовки дерева bool DrawTree(Tree root, ref Canvas canvas, int lvl, int width, int section\_top, int index, ref Line line);
* метод отрисовки части дерева bool DrawPartTree(Tree root, ref Canvas canvas, int lvl, int lvl\_border, int width, int section\_top, int index, ref Line line)

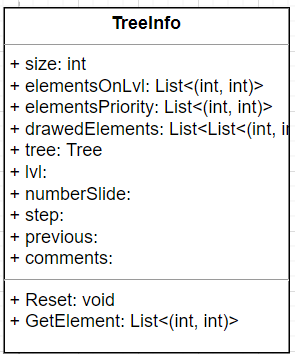


Рисунок 7 – Класс TreeInfo

Для визуализации был разработан класс TreeInfo в котором хранится вся информация о дереве для пошаговой визуализации.

### 2.4. Тестирование разработанного визуализатора

На рисунках 8-15 показана работа программы.

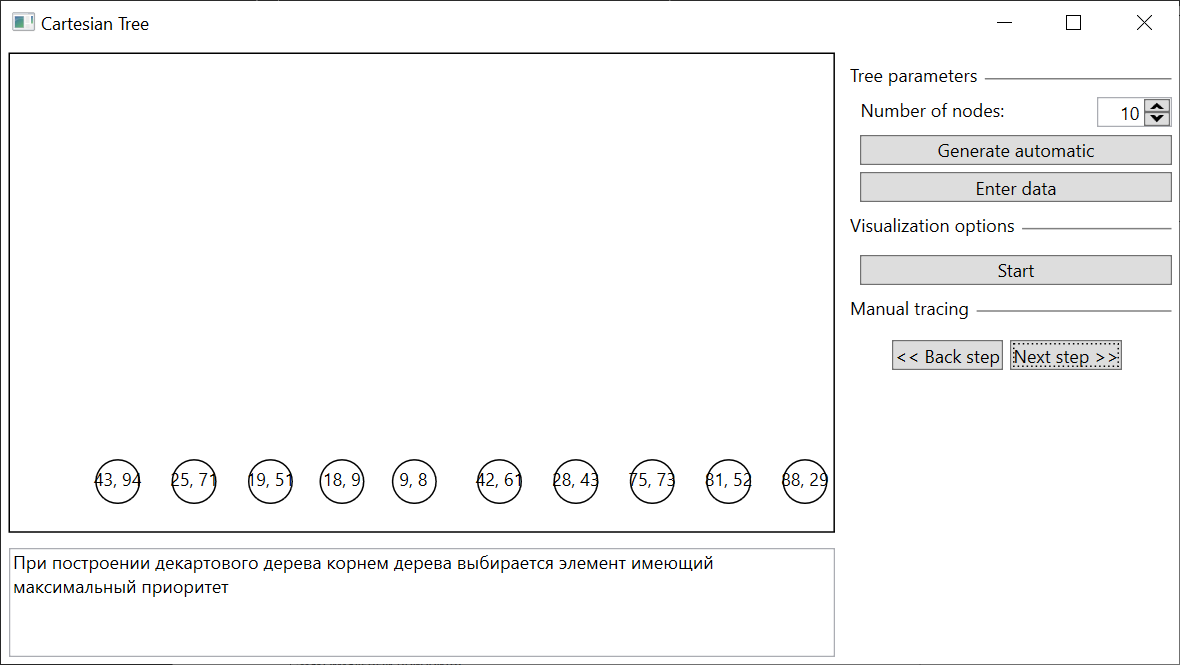


Рисунок – 8

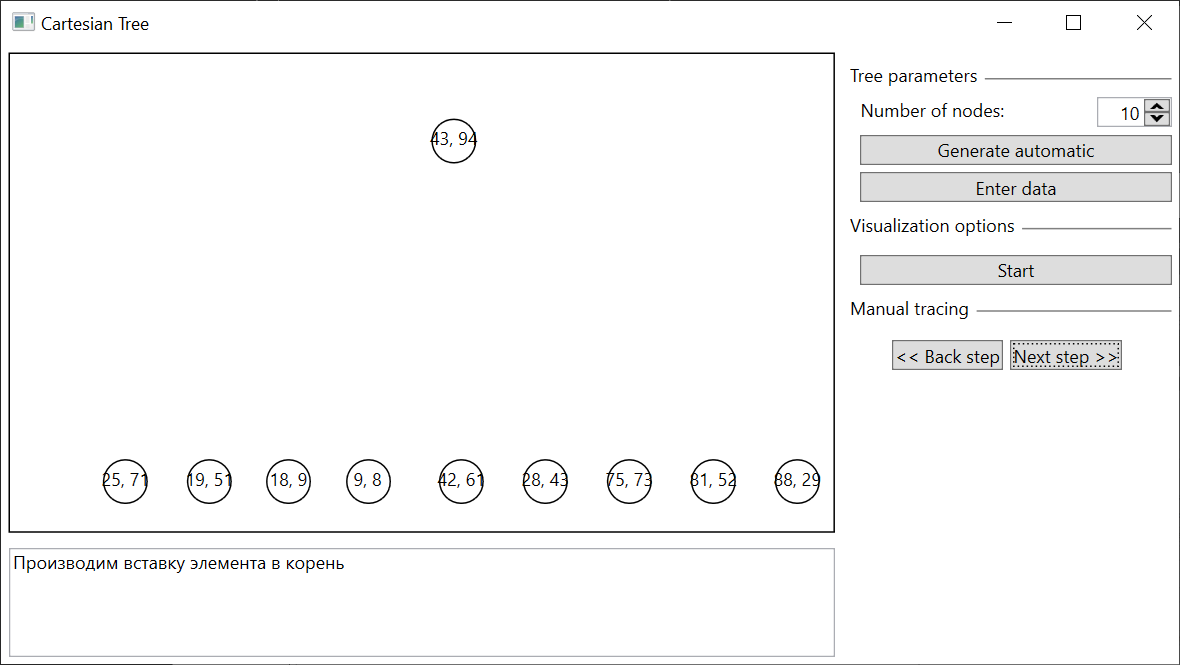
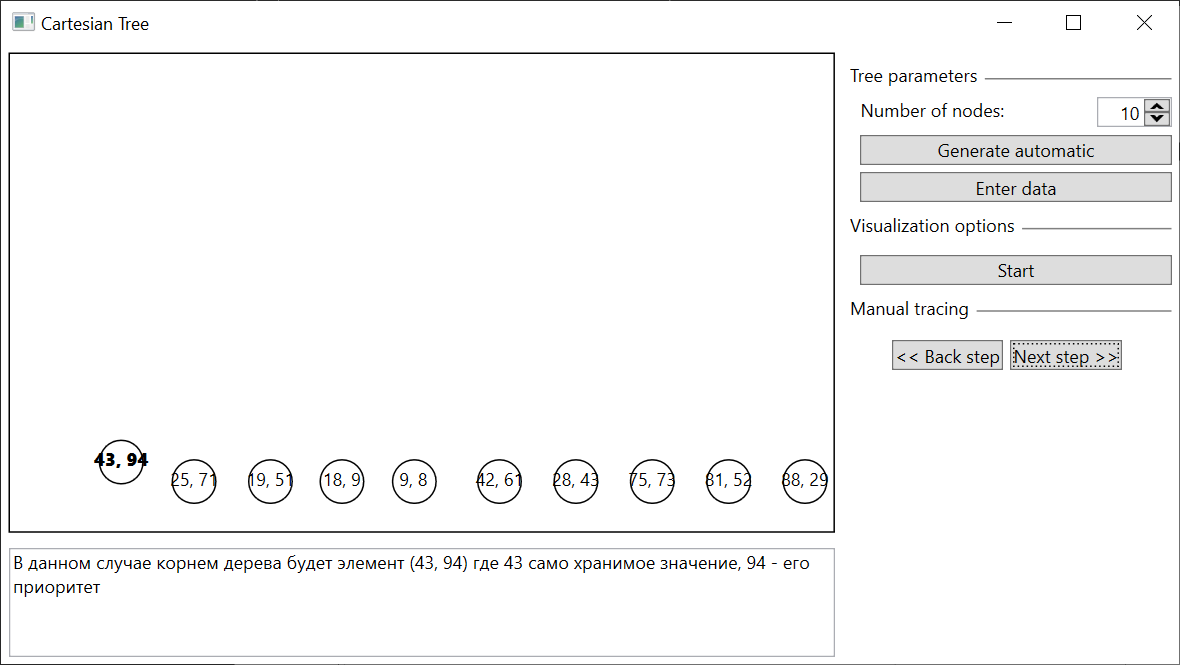
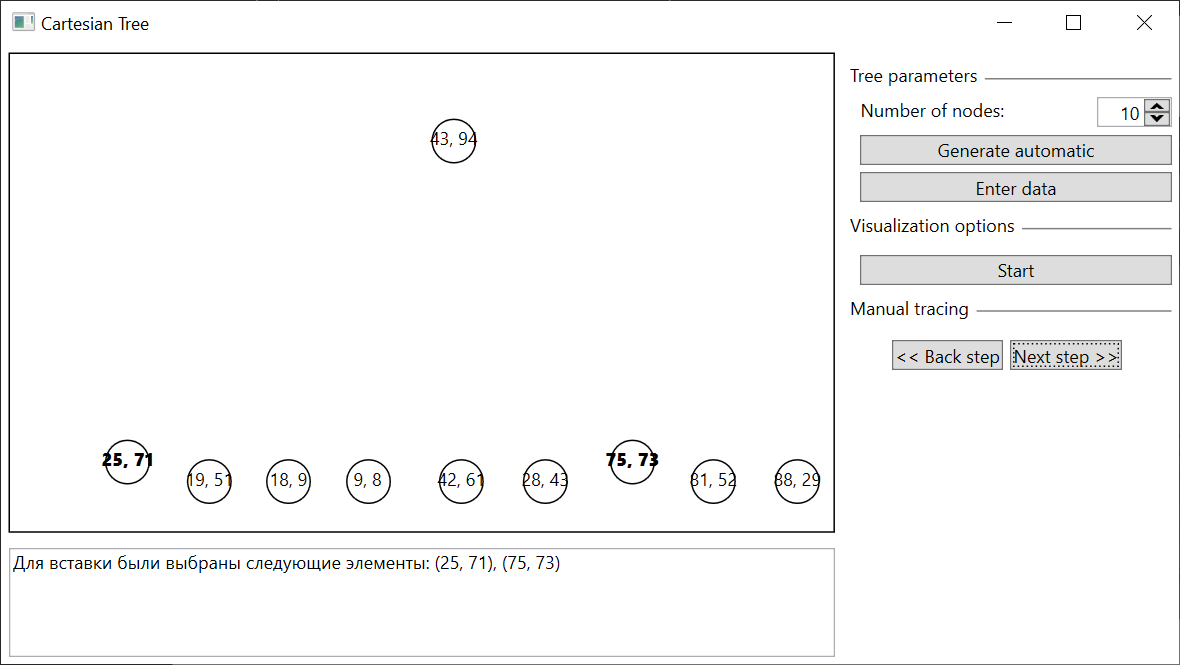
Рисунок – 9.

Рисунок – 10.

Рисунок – 11.

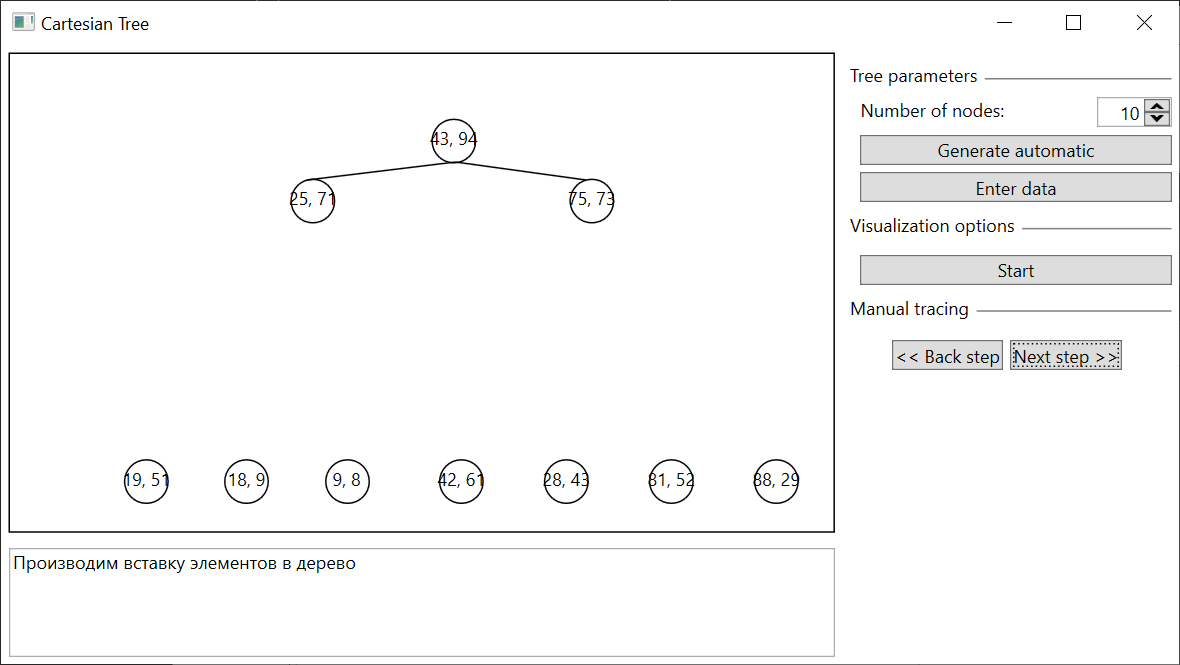


Рисунок – 12.

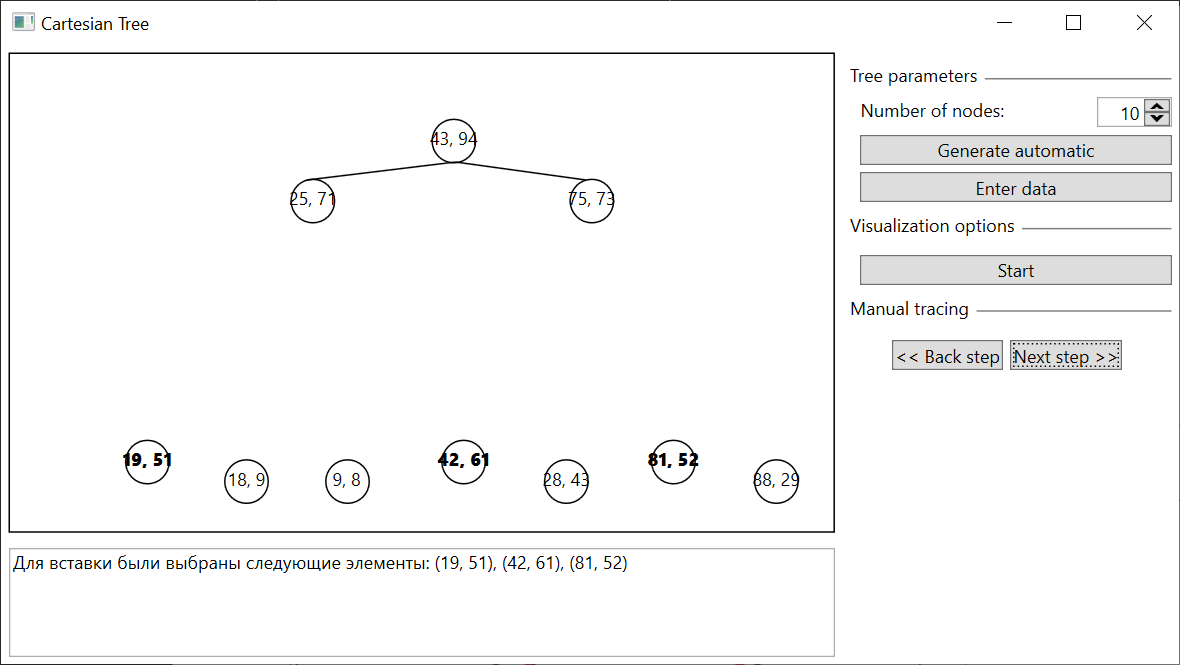


Рисунок – 13.

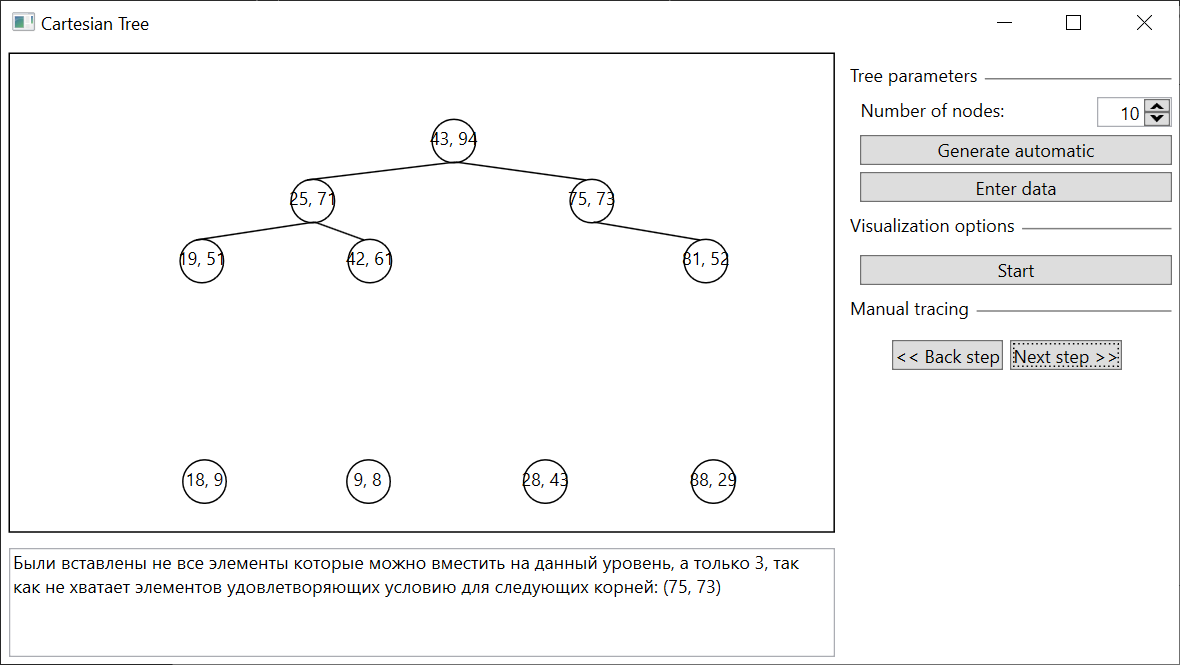


Рисунок - 14

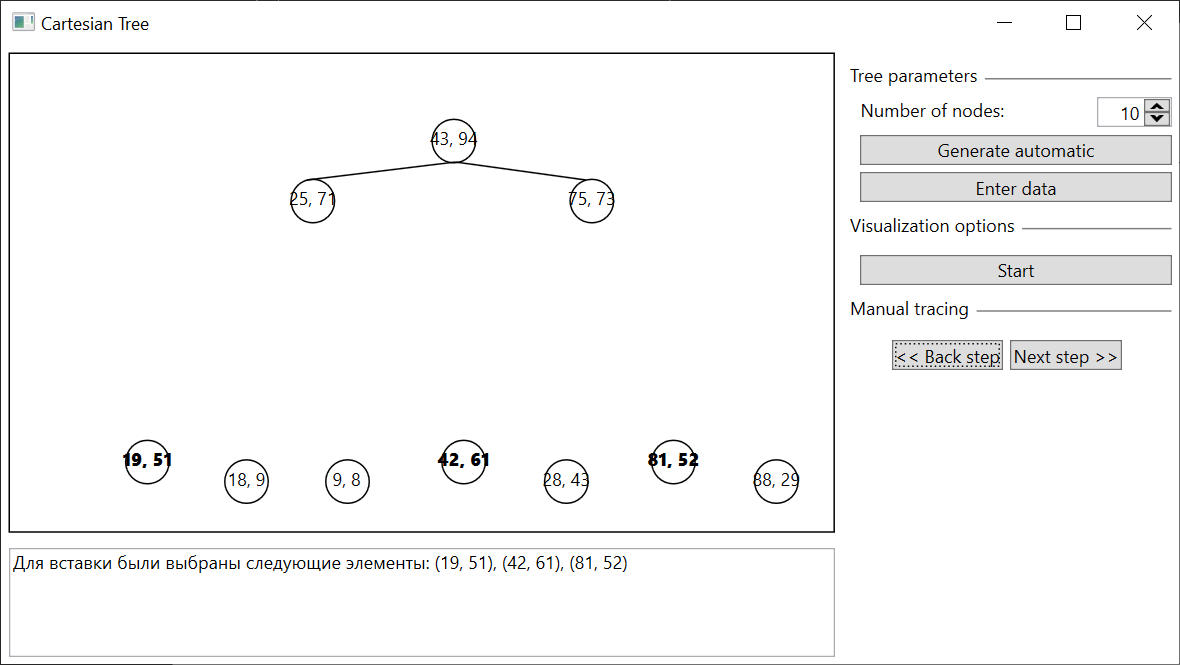


Рисунок – 15. Шаг назад.

# Заключение

В ходе выполнения расчётно-графической работы был разработан визуализатор алгоритма «Декартово дерево». Знание работы определенного алгоритма поможет при необходимости реализовать его и эффективно выполнить поставленную задачу.

Визуализатор разработанный в ходе расчётно-графической работы также выполняет все поставленные в задаче функции.

# Список литературы и Интернет-ресурсов

1. http://e-maxx.ru/algo/treap

2. https://neerc.ifmo.ru/wiki/index.php?title=Декартово\_дерево

3. https://habr.com/ru/post/101818/

4. https://ru.wikipedia.org/wiki/Декартово\_дерево