1.1. Красно-чёрные деревья

Красно-чёрные деревья (Red-Black Trees) являются одним из видов самобалансирующихся двоичных деревьев поиска. Они были разработаны для обеспечения эффективного выполнения операций вставки, удаления и поиска элементов, что делает их широко используемыми в структурах данных, таких как множества и словари. Вот описание структуры и свойств красно-чёрных деревьев:

1. Структура:
    - Красно-чёрное дерево состоит из узлов, каждый из которых содержит ключ, значение, и ссылки на двух дочерних узлов (левого и правого) и родительский узел.
    - Каждый узел имеет ассоциированный с ним цвет: красный или черный.
    - Красно-чёрное дерево всегда удовлетворяет некоторым свойствам, которые обеспечивают балансировку дерева.
2. Свойства:
    - Каждый узел дерева либо красный, либо черный.
    - Корень дерева всегда черный.
    - Каждый лист (NIL-узел или нулевой узел) является черным.
    - Если узел красный, то оба его дочерних узла - черные.
    - Все пути от корня до листьев должны содержать одинаковое количество черных узлов. Это свойство называется "черной высотой" и гарантирует, что дерево всегда сбалансировано.
3. Балансировка:
    - Вставка и удаление элементов в красно-чёрных деревьях сопровождаются операциями перекрашивания и поворотов, которые обеспечивают сохранение свойств красно-чёрных деревьев.
    - При вставке элемента, дерево может временно нарушить свойства, но затем оно будет перебалансировано, чтобы исправить нарушения.
    - При удалении элемента также выполняются перекрашивания и повороты, чтобы сохранить баланс.

Красно-чёрные деревья предоставляют гарантированное ограничение высоты дерева, что обеспечивает эффек
тивные операции поиска, вставки и удаления элементов. Это делает их полезными структурами данных для мн
огих приложений, включая реализацию словарей, множеств, баз данных и многие другие алгоритмы и структу
ры данных.

1.2. Декортовы деревья

Декартовы деревья (Cartesian trees) - это структура данных в виде бинарных деревьев, которая была введена в математической литературе в 1980-х годах. Эта структура данных объединяет два важных аспекта: бинарное дерево и упорядоченный массив. Декартово дерево обладает следующими ключевыми свойствами:

1. Бинарное дерево: Каждый узел декартового дерева имеет не более двух дочерних узлов - левого и правого. Эти деревья являются бинарными по своей природе, что делает их удобными для решения различных задач.
2. Порядок по ключу: Каждый узел декартового дерева содержит ключ (число или другой сравниваемый элемент), и эти ключи упорядочены таким образом, что для любого узла X все ключи в левом поддереве меньше ключа X, а все ключи в правом поддереве больше ключа X. Это свойство обеспечивает возможность эффективного выполнения операций вставки, удаления и поиска элементов.
3. Приоритеты: Каждый узел декартового дерева также связан с приоритетом. Приоритеты узлов выбираются случайным образом и могут использоваться для балансировки дерева и гарантирования его высоты.

Основные операции над декартовыми деревьями включают в себя вставку элемента с ключом, удаление элемента с ключом, поиск элемента по ключу и обход дерева в различных порядках (например, инфиксный, префиксный, постфиксный обход). Декартовы деревья обладают хорошей производительностью для всех этих операций в среднем случае. Декартовы деревья могут быть использованы в различных приложениях, включая сортировку, поиск, построение выражений и другие задачи, где требуется эффективная работа с упорядоченными данными.

1.3. Сравнение красно-чёрных и декратовых деревьев

Красно-чёрные деревья и декартовы деревья - это два различных типа бинарных деревьев, используемых в информатике и структурах данных. Давайте рассмотрим их основные характеристики и проведем сравнение:

1. Красно-чёрные деревья (Red-Black Trees):
    - Красно-чёрные деревья являются одним из видов самобалансирующихся бинарных деревьев поиска. Они используются для хранения данных и обеспечивают эффективные операции вставки, удаления и поиска.
    - В красно-чёрных деревьях каждый узел имеет ассоциированный с ним цвет - красный или черный. Это правило цвета служит для обеспечения сбалансированности дерева.
    - Основные свойства красно-чёрных деревьев:

        - Каждый узел является либо красным, либо черным.
        - Корень и листья (NIL-узлы) считаются черными.
        - Если узел красный, то его дети должны быть черными.
        - Все простые пути от узла к его потомкам должны иметь одинаковое количество черных узлов (черное высота).
    - Вставка и удаление элементов в красно-чёрных деревьях может приводить к перекрашиванию узлов и вращениям, чтобы сохранить баланс.
2. Декартовы деревья (Cartesian Trees):
    - Декартовы деревья - это тип бинарных деревьев, который используется для представления последовательности элементов с приоритетами. Они позволяют эффективно выполнять операции вставки, удаления и поиска с учетом приоритетов.
    - В декартовых деревьях каждый узел содержит два значения: значение элемента и его приоритет. Приоритеты упорядочивают элементы в дереве.
    - Основные свойства декартовых деревьев:
        - Каждый узел содержит элемент и приоритет.
        - Декартово дерево является бинарным деревом поиска по элементам и бинарным кучей по приоритетам.
        - При вставке нового элемента с приоритетом, дерево может перестраиваться для сохранения свойств.
        - Удаление элемента выполняется с помощью поиска по значению элемента, а затем пересортировки дерева.

Сравнение:

- Красно-чёрные деревья и декартовые деревья оба служат для организации данных и обеспечения эффективных операций вставки, удаления и поиска.
- Красно-чёрные деревья используют цвета узлов для соблюдения баланса, в то время как декартовые деревья используют приоритеты для сортировки элементов.
- Декартовые деревья могут использоваться для реализации структуры данных "декартово дерево", которая позволяет выполнять операции очереди с приоритетами.
- Вставка и удаление элементов в красно-чёрных деревьях может потребовать перекрашивания узлов и вращений, в то время как в декартовых деревьях перестройка выполняется на основе приоритетов.
- Выбор между красно-чёрными и декартовыми деревьями зависит от конкретных требований задачи и операций, которые требуется выполнять.

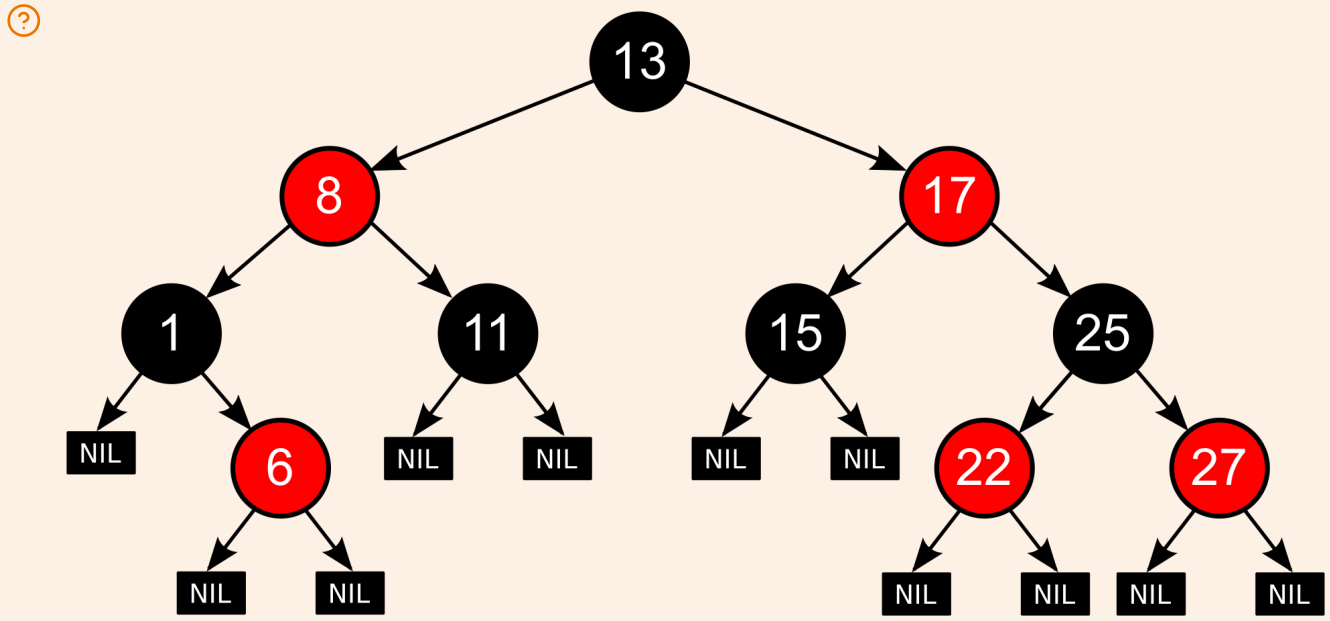
Оба типа деревьев имеют свои уникальные преимущества и ограничения и могут использоваться в различных контекстах в зависимости от задачи, которую необходимо решить.

2.1. Описание архитектуры и структуры красно-чёрных деревьев

Красно-чёрное дерево — двоичное дерево поиска, в котором каждый узел имеет атрибут *цвета*. При этом:

1. Узел может быть либо красным, либо чёрным и имеет двух потомков;
2. Корень — как правило чёрный. Это правило слабо влияет на работоспособность модели, так как цвет корня всегда можно изменить с красного на чёрный;
3. Все листья, не содержащие данных — чёрные.
4. Оба потомка каждого красного узла — чёрные.
5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов.

Благодаря этим ограничениям путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего, и дерево примерно сбалансировано. Операции вставки, удаления и поиска требуют в худшем случае времени, пропорционального длине дерева, что позволяет красно-чёрным деревьям в худшем случае быть более эффективными, чем обычные двоичные деревья поиска.



Красно-чёрное дерево

Чтобы понять, как это работает, достаточно рассмотреть эффект свойств 4 и 5 вместе. Пусть для красно-чёрного дерева T число чёрных узлов от корня до листа равно B. Тогда кратчайший возможный путь до любого листа содержит B узлов и все они чёрные. Более длинный возможный путь может быть построен путём включения красных узлов. Однако, благодаря п.4 в дереве не может быть двух красных узлов подряд, а согласно пп. 2 и 3, путь

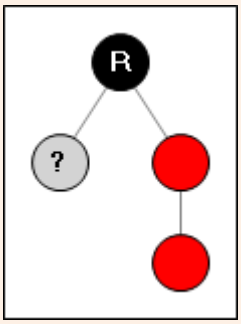
начинается и кончается чёрным узлом. Поэтому самый длинный возможный путь состоит из 2B-1 узлов, попеременно красных и чёрных.

Если разрешить нелистовому узлу иметь меньше двух потомков, а листовым — содержать данные, дерево сохраняет основные свойства, но алгоритмы работы с ним усложнятся. Поэтому в статье рассматриваются только «фиктивные листовые узлы», которые не содержат данных и просто служат для указания, где дерево заканчивается. Эти узлы могут быть опущены в некоторых иллюстрациях. Из п.5, также следует, что потомками красного узла могут быть либо два чёрных промежуточных узла, либо два чёрных листа, а с учётом п.3 и 4 — что если у чёрного узла один из потомков — листовой узел, то вторым должен быть либо тоже листовой, либо вышеописанная конструкция из одного красного и двух листовых.

2.2. Принцип работы красно-черных деревьев

Вставка
Находим в дереве место для вставки, вставляем узел красного цвета. Поднимаемся вверх по дереву и находим ситуации, которые условно можно обозначить так:

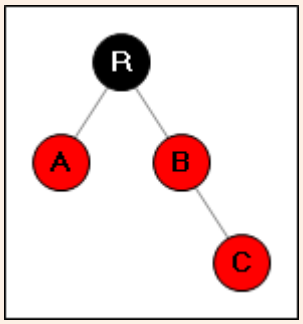
Рис. 2.2.1.



Конечный результат вставки

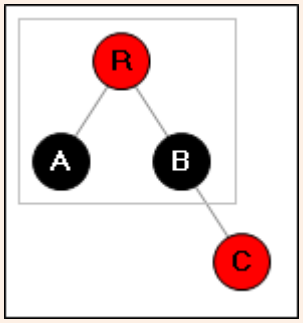
Не важно что и куда вставлено, надо просто найти в дереве нужные шаблоны.
Ситуация 1:

Рис. 2.2.2.



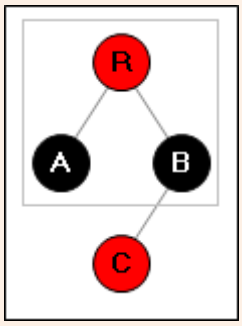
Маркируем узлы красно-чёрного дерева

Рис. 2.2.3.



Берём предполагаемый шаблон и перекрашиваем 3 вершины

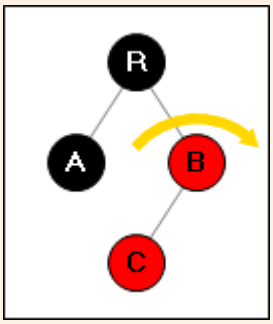
Рис. 2.2.4.



Получаем нужный шаблон

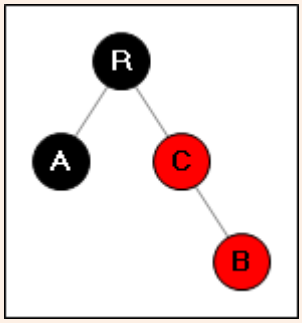
Ситуация 2:

Рис. 2.2.5.



Выполняем маленький поворот без перекраски вершин

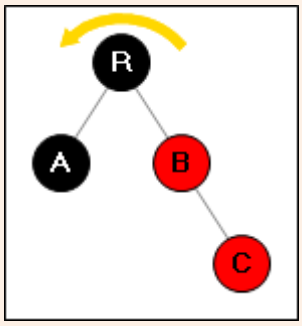
Рис. 2.2.6.



Получаем искомый шаблон

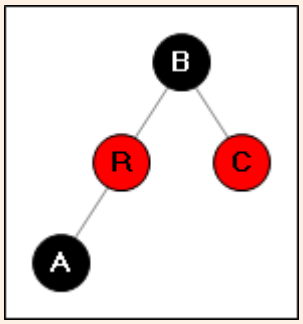
Ситуация 3:

Рис. 2.2.7.



Перекрашиваем две вершины (R,B) и выполняем поворот

Рис. 2.2.7.



Получаем искомый шаблон

Удаление

Почти такие-же по смыслу действия. Находим вершину которую надо удалить (назовём её A). Если вершину нельзя удалить сразу, находим вершину которую можно вставить на место удаляемой (назовём её B). Замена при этом окрашивается в цвет удалённой вершины (B красится в цвет A). Далее мы смотрим цвет той вершины, которую мы убрали физически (если была замена, то смотрим цвет вершины B). Теперь рассматриваем ситуации возникающие на дереве (крестиком пометили вершину, которая была убрана):

Ситуация 0:

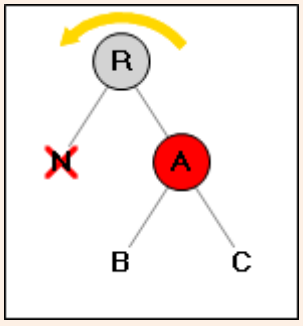
Если была убрана красная вершина, перекрашиваем её в чёрную.

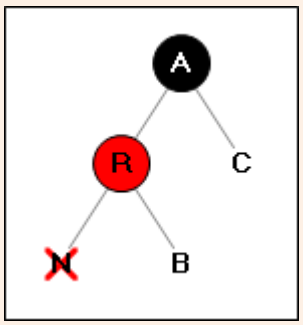
Прекращаем работу.

Ситуация 1:

Удалённая вершина имеет красного брата.

Рис. 2.2.8.

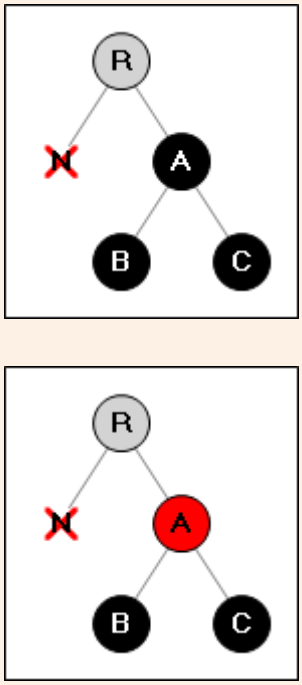




Перекрашиваем две вершины (A,R). Делаем поворот. Выполняем повторую проверку ситуаций для вершины (C), если для неё была применена ситуация 2, красим (C) в чёрный цвет. Прекращаем работу.

Ситуация 2:
Удалённая вершина имеет чёрного брата с чёрными потомками.

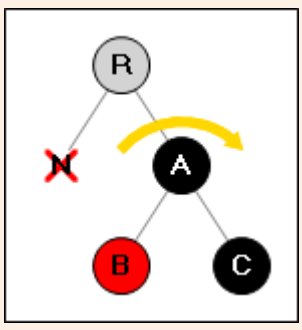
Рис. 2.2.9.

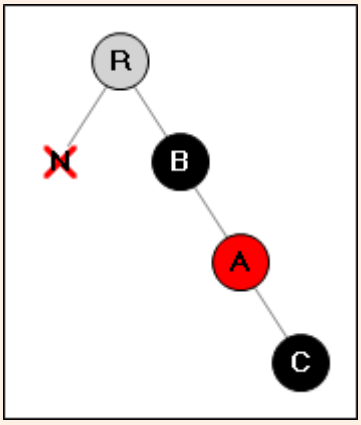


Перекрашиваем вершину (A) в чёрный цвет.
Продолжаем работу, над вершиной выше по дереву.

Ситуация 3:
Удалённая вершина имеет чёрного брата, у которого красный левый и чёрный правый потомок.

Рис. 2.2.10.

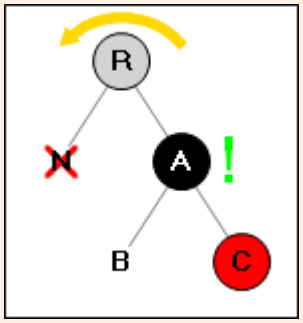


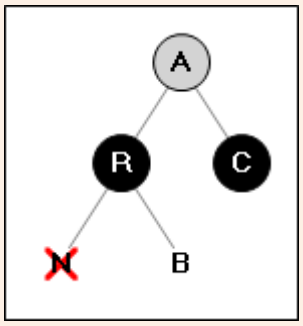


Перекрашиваем вершины (A,B). Поворачиваем вершину (A). Сразу же, переходим в проверке ситуации 4.

Ситуация 4:
Удалённая вершина имеет чёрного брата, у которого красный правый потомок (цвет левого потомка брата тут не важен).

Рис. 2.2.11.





Перекрашиваем вершину (A) в цвет вершины (R). Перекрашиваем вершины (R,C) в черный цвет. Делаем поворот. Прекращаем работу.

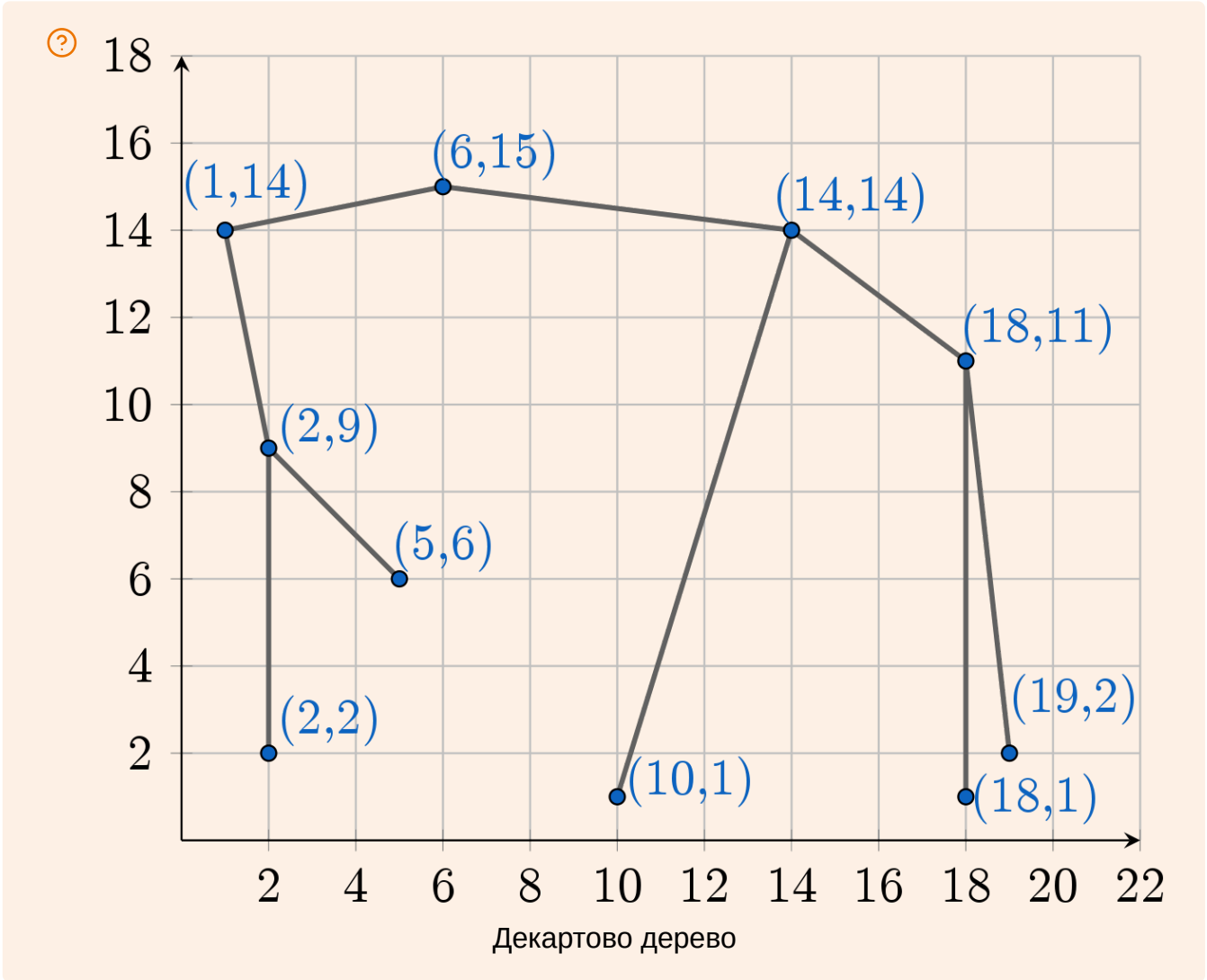
2.3. Описание архитектуры и структуры декартовых деревьев

Дека́ртово де́рево, **дуча**, **дерамида** — это структура данных, сочетающая в себе двоичное дерево и двоичную кучу. Хранит пары *(x, y)*, где для ***ключа*** *x* служит бинарным деревом поиска, а для ***приоритета*** *y* — двоичной кучей.

Свойства:

Декартово дерево не является самобалансирующимся в обычном смысле, и применяют его по следующим причинам:

- Проще реализуется по сравнению, например, с настоящими самобалансирующимися деревьями вроде красно-чёрного.
- Хорошо ведёт себя «в среднем», если ключи *y* раздать случайно.
- Типичная для сортирующего дерева операция «разделить по ключу *x* на „меньше *x_0“* и *„не меньше _x_0“»* работает за *_O*(*h*), где *h* — высота дерева. На красно-чёрных деревьях придётся восстанавливать балансировку и окраску узлов.



Декартово дерево

Простейший алгоритм:
Простейший для понимания алгоритм построения декартового дерева по множеству данных пар (x, y) выглядит следующим образом. Упорядочим все пары по ключу x и пронумеруем получившуюся последовательность ключей y:

y(1), y(2), y(3), …, y(n).

Найдём минимальный ключ y. Пусть это будет y(k). Он будет корнем дерева. Ключ y(k) делит последовательность ключей y на две:

y(1), …, y(k−1); y(k+1), …, y(n).

В каждой из них найдём минимальный y — это будут дети узла y(k) — левый и правый. С получившимися 4 кусочками (возможно меньше) поступим аналогичным образом. Предложенный алгоритм построения декартового дерева основан на рекурсии: находим в последовательности минимальный y и назначаем его корнем. Найденный y разбивает последовательность на две части, для каждой из частей запускаем алгоритм построения декартового дерева.

Схематически это можно записать так:

```
T( y(1), ..., y(n) ) =  root: y(k)
                           left_tree: T( y(1), ..., y(k-1) )
                           right_tree: T( y(k+1), ..., y(n)) )
                        where  y(k) = min( y(1), ..., y(n) )
```

Из данного алгоритма следует, что множество пар (x, y) однозначно определяет структуру декартового дерева. Заметим для сравнения, что множество ключей, которые хранятся в двоичном дереве поиска, не определяют однозначно структуру дерева. То же самое касается двоичной кучи — какова будет структура двоичной кучи (как ключи распределятся по узлам), зависит не только от самого множества ключей, но и от последовательности их добавления. В декартовом дереве такой неоднозначности нет.

Линейный алгоритм:
Другой алгоритм построения дерева также основан на рекурсии. Только теперь мы последовательно будем добавлять элементы y и перестраивать дерево. Дерево T(y(1), …, y(k+1)) будет строиться из дерева T(y(1), …, y(k)) и следующего элемента y(k+1).

T(y(1), ..., y(k+1)) = F (T(y(1), ..., y(k)), y(k+1))

На каждом шаге будем помнить ссылку на последний добавленный узел. Он будет самым правым. Действительно, мы упорядочили ключи y по прикреплённому к ним ключу x. Так как декартово дерево — это дерево поиска, то после проекции на горизонтальную прямую

ключи x должны возрастать слева направо. Самый правый узел всегда имеет максимально возможное значение ключа x.

Функция F, которая отображает декартово дерево T(y(1), …, y(k)) предыдущего шага и очередное y(k+1) в новое дерево T(y(1), …, y(k+1)), выглядит следующим образом. Вертикаль для узла y(k+1) определена. Нам необходимо определиться с его горизонталью. Для начала мы проверяем, можно ли новый узел y(k+1) сделать правым ребёнком узла y(k) — это следует сделать, если y(k+1) > y(k). Иначе мы делаем шаг по склону от узла y(k) вверх и смотрим на значение y, которое там хранится. Поднимаемся вверх по склону, пока не найдём узел, в котором значение y меньше, чем y(k+1), после чего делаем y(k+1) его правым ребёнком, а его предыдущего правого ребёнка делаем левым ребёнком узла y(k+1).

Это алгоритм амортизационно(в сумме за все шаги) работает за линейное время (по числу добавляемых узлов). Действительно, как только мы «перешагнули» через какой-либо узел, поднимаясь вверх по склону, то мы его уже никогда не встретим при добавлении следующих узлов. Таким образом, суммарное число шагов вверх по склону не может быть больше общего числа узлов.