Декартово дерево

Эта статья про курево

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), также существует название курево (куча + дерево).

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 (x_0, y_0), то у всех элементов в левом поддереве $x < x_0$, у всех элементов в правом поддереве $x > x_0$, а также и в левом, и в правом поддереве имеем: $y < y_0$.

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

Содержание

Операции в декартовом дереве

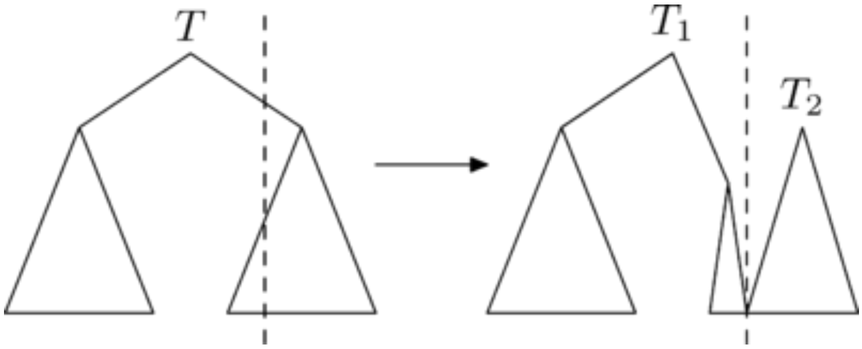
split

Операция split (*разрезать*) позволяет сделать следующее: разрезать исходное дерево T по ключу k. Возвращать она будет такую пару деревьев $\langle T_1, T_2 \rangle$, что в дереве T_1 ключи меньше k, а в дереве T_2 все остальные: $\mathrm{split}(T, k) \to \langle T_1, T_2 \rangle$.

Эта операция устроена следующим образом.

Рассмотрим случай, в котором требуется разрезать дерево по ключу, большему ключа корня. Посмотрим, как будут устроены результирующие деревья T_1 и T_2:

- T_1: левое поддерево T_1 совпадёт с левым поддеревом T. Для нахождения правого поддерева T_1, нужно разрезать правое поддерево T на T_1^R и T_2^R по ключу k и взять T_1^R.
- T_2 совпадёт с T_2^R.



Операция split

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

Псевдокод

```
⟨Treap, Treap⟩ split(t: Treap, k: int):
  if t == ∅
    return ⟨∅, ∅⟩
  else if k > t.x
    ⟨t1, t2⟩ = split(t.right, k)
    t.right = t1
    return ⟨t, t2⟩
  else
    ⟨t1, t2⟩ = split(t.left, k)
    t.left = t2
    return ⟨t1, t⟩
```

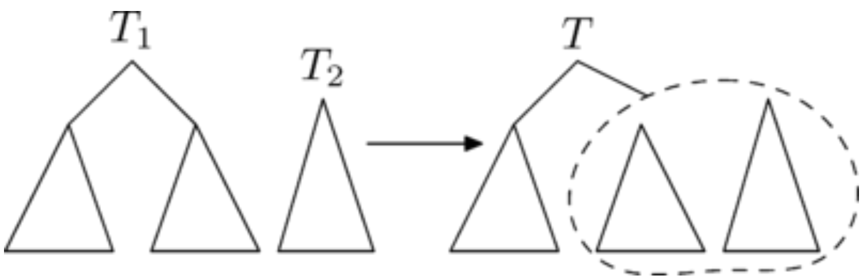
Время работы

Оценим 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



Операция merge

ключи во втором(*правом*). В результате получается дерево, в котором есть все ключи из первого и второго деревьев: $\mathrm{merge}(T_1, T_2) \to \{T\}$

Рассмотрим принцип работы этой операции. Пусть нужно слить деревья T_1 и T_2. Тогда, очевидно, у результирующего дерева T есть корень. Корнем станет вершина из T_1 или T_2 с наибольшим приоритетом y. Но вершина с самым большим y из всех вершин деревьев T_1 и T_2 может быть только либо корнем T_1, либо корнем T_2. Рассмотрим случай, в котором корень T_1 имеет больший y, чем корень T_2. Случай, в котором корень T_2 имеет больший y, чем корень T_1, симметричен этому.

Если y корня T_1 больше y корня T_2, то он и будет являться корнем. Тогда левое поддерево T совпадёт с левым поддеревом T_1. Справа же нужно подвесить объединение правого поддерева T_1 и дерева T_2.

Псевдокод

```
Treap merge(t1: Treap, t2: Treap):
  if t2 == ∅
    return t1
  if t1 == ∅
    return t2
  else if t1.y > t2.y
    t1.right = merge(t1.right, t2)
    return t1
  else
    t2.left = merge(t1, t2.left)
    return t2
```

Время работы

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

insert

Операция $\mathrm{insert}(T, k)$ 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

1. Разобьём наше дерево по ключу, который мы хотим добавить, то есть $\mathrm{split}(T, k.x) \to \langle T_1, T_2 \rangle$.
2. Сливаем первое дерево с новым элементом, то есть $\mathrm{merge}(T_1, k) \to T_1$.
3. Сливаем получившиеся дерево со вторым, то есть $\mathrm{merge}(T_1, T_2) \to T$.

- Реализация №2

1. Сначала спускаемся по дереву (как в обычном бинарном дереве поиска по $k.x$), но останавливаемся на первом элементе, в котором значение приоритета оказалось меньше $k.y$.
2. Теперь вызываем $\mathrm{split}(T, k.x) \to \langle T_1, T_2 \rangle$ от найденного элемента (от элемента вместе со всем его поддеревом)
3. Полученные T_1 и T_2 записываем в качестве левого и правого сына добавляемого элемента.
4. 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.

remove

Операция $\mathrm{remove}(T, x)$ удаляет из дерева T элемент с ключом x.

- Реализация №1

1. Разобьём наше дерево по ключу, который мы хотим удалить, то есть $\mathrm{split}(T, k.x) \to \langle T_1, T_2 \rangle$.
2. Теперь отделяем от первого дерева элемент x, то есть самого левого ребёнка дерева T_2.
3. Сливаем первое дерево со вторым, то есть $\mathrm{merge}(T_1, T_2) \to T$.

- Реализация №2

1. Спускаемся по дереву (как в обычном бинарном дереве поиска по x), и ищем удаляемый элемент.
2. Найдя элемент, вызываем merge его левого и правого сыновей
3. Результат процедуры merge ставим на место удаляемого элемента.

В первой реализации один раз используется split, а во второй реализации разрезание вообще не используется.

Построение декартова дерева

Пусть нам известно из каких пар (x_i, y_i) требуется построить декартово дерево, причём также известно, что $x_1 < x_2 < \ldots < x_n$.

Алгоритм за $O(n \log n)$

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

Другой алгоритм за $O(n \log n)$

Отсортируем парочки (x_i, y_i) по убыванию x_i 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 $\dfrac{n}{2}$ 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 $\log n$. Значит полное время работы алгоритма будет $O(n \log n)$.

Алгоритм за $O(n)$

Будем строить дерево слева направо, то есть начиная с (x_1, y_1) по (x_n, y_n), при этом помнить последний добавленный элемент (x_k, y_k). Он будет самым правым, так как у него будет максимальный ключ, а по ключам декартово дерево представляет собой двоичное дерево поиска. При добавлении (x_{k+1}, y_{k+1}), пытаемся сделать его правым сыном (x_k, y_k), это следует сделать если $y_k > y_{k+1}$, 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 (x_{k+1}, y_{k+1}) его правым сыном, а предыдущего правого сына делаем левым сыном (x_{k+1}, y_{k+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 $O(n)$.

Случайные приоритеты

Мы уже выяснили, что сложность операций с декартовым деревом линейно зависит от его высоты. В действительности высота декартова дерева может быть линейной относительно его размеров. Например, высота декартова дерева, построенного по набору ключей $(1, 1), \ldots, (n, n)$, будет равна n. Во избежание таких случаев, полезным оказывается выбирать приоритеты в ключах случайно.

Высота в декартовом дереве с случайными приоритетами

Теорема:

В декартовом дереве из n узлов, приоритеты y которого являются случайными величинами с равномерным распределением, средняя глубина вершины $O(\log n)$.

Доказательство:

▷

Будем считать, что все выбранные приоритеты y попарно различны.

Для начала введём несколько обозначений:

- x_k — вершина с k-ым по величине ключом;

- индикаторная величина $A_{i,j} = \begin{cases} 1, & x_i \text{ is ancestor of } x_j \\ 0, & \text{otherwise} \end{cases}$
- $d(v)$ — глубина вершины v;

В силу обозначений глубину вершины можно записать как количество предков:

$$d(x_k) = \sum_{i=1}^{n} A_{i,k}.$$

Теперь можно выразить математическое ожидание глубины конкретной вершины:

$E(d(x_k)) = \sum_{i=1}^{n} Pr[A_{i,k} = 1]$ — здесь мы использовали линейность математического ожидания, и то что $E(X) = Pr[X = 1]$ для индикаторной величины X ($Pr[A]$ — вероятность события A).

Для подсчёта средней глубины вершин нам нужно сосчитать вероятность того, что вершина x_i является предком вершины x_k , то есть $Pr[A_{i,k} = 1]$.

Введём новое обозначение:

- $X_{i,k}$ — множество ключей $\{x_i, \ldots, x_k\}$ или $\{x_k, \ldots, x_i\}$, в зависимости от $i < k$ или $i > k$. $X_{i,k}$ и $X_{k,i}$ обозначают одно и тоже, их мощность равна $|k - i| + 1$.

Лемма:
Для любых $i \neq k$, x_i является предком x_k тогда и только тогда, когда x_i имеет наибольший приоритет среди $X_{i,k}$.
Доказательство:
$\triangleright$

Если x_i является корнем, то оно является предком x_k и по определению имеет максимальный приоритет среди всех вершин, следовательно, и среди $X_{i,k}$.

С другой стороны, если x_k — корень, то x_i — не предок x_k , и x_k имеет максимальный приоритет в декартовом дереве; следовательно, x_i не имеет наибольший приоритет среди $X_{i,k}$.

Теперь предположим, что какая-то другая вершина x_m — корень. Тогда, если x_i и x_k лежат в разных поддеревьях, то $i < m < k$ или $i > m > k$, следовательно, x_m содержится в $X_{i,k}$. В этом случае x_i — не предок x_k , и наибольший приоритет среди $X_{i,k}$ имеет вершина с номером m.

Наконец, если x_i и x_k лежат в одном поддереве, то доказательство применяется по индукции: пустое декартово дерево есть тривиальная база, а рассматриваемое поддерево является меньшим декартовым деревом.
$\triangleleft$

Так как распределение приоритетов равномерное, каждая вершина среди $X_{i,k}$ может иметь максимальный приоритет, мы немедленно приходим к следующему равенству:

$$Pr[A_{i,j} = 1] = \begin{cases} \dfrac{1}{k-i+1}, & k > i \\ 0, & k = i \\ \dfrac{1}{i-k+1}, & k < i \end{cases}$$

Подставив последнее в нашу формулу с математическим ожиданием получим:

$$E(d(x_k)) = \sum_{i=1}^{n} Pr[A_{i,k} = 1] = \sum_{i=1}^{k-1} \frac{1}{k-i+1} + \sum_{i=k+1}^{n} \frac{1}{i-k+1} \leqslant$$

$\leqslant \ln(k) + \ln(n-k) + 2$ (здесь мы использовали неравенство $\sum_{i=1}^{n} \frac{1}{i} \leqslant \ln(n) + 1$)

$\log(n)$ отличается от $\ln(n)$ в константу раз, поэтому $\log(n) = O(\ln(n))$.

В итоге мы получили что $E(d(x_k)) = O(\log(n))$.
$\triangleleft$

Таким образом, среднее время работы операций split и merge будет $O(\log(n))$.

См. также

- Декартово дерево по неявному ключу

Источники информации

- Декартово дерево — Википедия (http://ru.wikipedia.org/wiki/Декартово_дерево)
- Treaps и T-Treaps (http://rain.ifmo.ru/cat/data/theory/trees/treaps-2006/article.pdf)

Источник — «http://neerc.ifmo.ru/wiki/index.php?title=Декартово_дерево&oldid=68427»

- Эта страница последний раз была отредактирована 16 января 2019 в 20:09.