Min-Max Heap. Бинарная куча. Тестирование и сравнение.

Никонов Михаил

Санкт-Петербургский академический университет
mishanickonov@gmail.com

Содержание

Вступление

В данной работе приведены описание и реализация обычной бинарной кучи и ее модификации - Min-Max Heap, их сравнение и некоторая статистика.

1 Бинарная куча

Куча - простая структура данных, предназначенная для быстрого поиска минимума (максимума) множества, с возможностью добавления и удаления из него элементов. Куча представляет из себя дерево, в котором все вершины не меньше своих предков.

Бинарная (двоичная) куча - это куча, для которой выполнены дополнительные условия:

- У каждой вершины не более двух потомков.
- На каждом, кроме последнего, слое `i`, ровно 2^i вершин. (слои нумеруются с нуля)
- Последний слой заполнен слева направо.

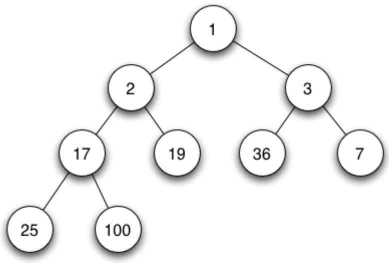

Рис. 1: Пример бинарной кучи

1.1 Реализация: вспомогательные функции

Двоичную кучу удобно писать на массиве так, что для каждой вершины `node`: `Heap[node / 2]` - ее предок, а `Heap[node * 2]`, `Heap[node * 2 + 1]` - сыновья. Также для упрощения написания большинства операций, удобно завести вспомагательные функции: `SiftDown` - восстанавливает свойство кучи после уменьшения значения элемента, и `SiftUp` - после увеличения.

- `SiftDown`: если элемент `id` меньше своих детей, то свойство кучи не нарушено. Иначе меняем его с наименьшим из его сыновей, после чего рекурсивно запускаемся от новой позиции `id`. Так как дерево двоичное и `SiftDown` в худшем случае работает за глубину дерева, то асимптотика: $O(\log n)$. Пример реализации на `C++`:

```
void SiftDown(int id)
{
    int left = id * 2,
        right = id * 2 + 1;
    if (left > HeapSize) return;

    if (right > HeapSize){
        if (Heap[id] > Heap[left]) swap(Heap[id], Heap[left]);
    } else {
        int nxt = (Heap[left] < Heap[right] ? left : right);
        if (Heap[id] > Heap[nxt]) {
            swap(Heap[id], Heap[nxt]);
            SiftDown(nxt);
        }
    }
}
```

- `SiftUp`: если элемент `id` не меньше своего предка, то свойство кучи не нарушено. Иначе, поменяем местами его и предка, после чего рекурсивно запустимся от новой позиции `id`. Асимптотика: $O(\log n)$. Пример реализации на `C++`:

```
void SiftUp(int id)
{
    if (id == 1) return;

    int par = id / 2;
    if (Heap[par] > Heap[id]) {
        swap(Heap[par], Heap[id]);
        SiftUp(par);
    }
}
```

1.2 Реализация: основные функции

Благодаря определенным функциям `SiftUp` и `SiftDown` практически все нужные нам методы для работы с кучей реализуются очень просто:

1. `Поиск минимума`:
   - Результат - корень кучи.
   - Константное время работы.
2. Добавление элемента:
   - Добавляем новый элемент в конец массива.
   - Запускаем `SiftUp` от нового элемента, чтоб восстановить свойство кучи.
3. Удаление элемента:
   - Меняем его местами с последним элементом в массиве.
   - Уменьшаем размер кучи.
   - Восстанавливаем свойство кучи используя `SiftDown`
4. Удаление минимума: удаляем первый элемент массива.
5. Построение кучи из неупорядоченного массива за $O(n)$ (см. далее).

1.3 Построение кучи из неупорядоченного массива за `O(n)`

Метод построения кучи за $O(n)$ также актуален и для `Min-max heap`. Алгоритм: для каждой вершины, имеющей хотя бы одного потомка, запустим `SiftDown`. После завершения будет получена корректная куча, т.к. при запуске от очередного элемента, все его поддеревья - корректные кучи.

Докажем, что построение кучи таким способом работает за $O(n)$. Для этого нужно посчитать такую сумму (время работы `SiftDown` для каждой вершины - ее высота):
$\sum\limits_{h=1}^{\log n} \frac{n}{2^h} h = n \sum\limits_{h=1}^{\log n} \frac{h}{2^h} = nO(\sum\limits_{h=1}^{\infty} \frac{h}{2^h}) = O(n)$

1.4 Мини-итог

Сейчас только скажем, что все операции, кроме нахождения минимума ($O(1)$), работают за $O(\log n)$. Двоичная куча также занимает линейную, относительно количества элементов, память. Все это, в совокупности с простотой реализации, делает бинарную кучу довольно полезной структурой данных, применяемой во многих задачах. В частности, например, двоичная куча может быть использована, как очередь с приоритетами.

2 Min-Max heap

`Min-Max Heap` является модификацией обычной двоичной кучи, для которой, помимо уже изложенных, выполняется следующее правило: на нечетных уровнях находятся элементы, меньшие всех своих потомков, а на четных - большие. Как мы скоро увидим, такая куча дает нам значительно больше возможностей, нежели обычная.

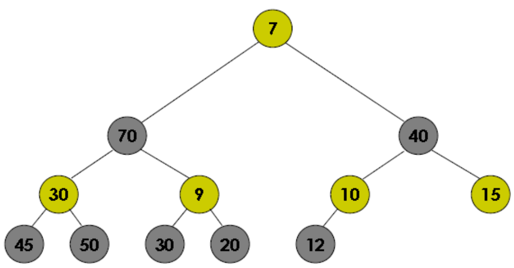

Рис. 2: Пример min-max кучи

2.1 Реализация: вспомогательные функции

Как и обычную двоичную кучу, мы будем ее реализовывать на массиве. Также нам понадобятся операции, аналогичные `SiftDown` и `SiftUp` для восстановления свойств кучи. Асимптотика этих операция не изменилась.

- `TrickleDown`(сохранено оригинальное название): принцип работы аналогичен `SiftDown` у бинарной кучи, однако, из-за условия с поддержкой максимумов, придется разобрать несколько случаев. При обработке очередного элемента важны не только его 'дети', но и 'внуки'. Ниже пример реализации на `C++`:

```cpp
void TrickleDown(int pos)
{
    if (lvl[pos] % 2) TrickleDownMin(pos);
    else TrickleDownMax(pos);
}

void TrickleDownMin(int pos)
{
    node nxt = get_nxt(MIN, pos); // getting min child/gr-child
    if (!nxt.exists) return;

    if (lvl[nxt.id] - lvl[pos] == 2) { // is grandchild
        if (Heap[nxt.id] < Heap[pos]) {
            swap(Heap[nxt.id], Heap[pos]);
            if (Heap[nxt.id] > Heap[nxt.id / 2])
                swap(Heap[nxt.id], Heap[nxt.id / 2]);
            TrickleDownMin(nxt.id);
        }
    } else {
        if (Heap[nxt.id] < Heap[pos])
            swap(Heap[pos], Heap[nxt.id]);
    }
}

void TrickleDownMax(int pos)
{
    node nxt = get_nxt(MAX, pos); // getting max child/gr-child
    if (!nxt.exists) return;

    if (lvl[nxt.id] - lvl[pos] == 2) { // is grandchild
        if (Heap[nxt.id] > Heap[pos]) {
            swap(Heap[nxt.id], Heap[pos]);
            if (Heap[nxt.id] < Heap[nxt.id / 2])
                swap(Heap[nxt.id], Heap[nxt.id / 2]);
            TrickleDownMax(nxt.id);
        }
    } else {
        if (Heap[nxt.id] > Heap[pos])
            swap(Heap[pos], Heap[nxt.id]);
    }
}
```

- `BubbleUp`: принцип работы аналогичен `SiftUp`. Пример реализации на `C++`:

```cpp
void BubbleUp(int pos)
{
    if (pos == 1) return;
    int par = pos / 2;

    if (lvl[pos] % 2) {              // min level
        if (Heap[pos] > Heap[par]) {
            swap(Heap[pos], Heap[par]);
            BubbleUpMax(par);
        } else {
            BubbleUpMin(pos);
        }
    } else {                         // max level
        if (Heap[pos] < Heap[par]) {
            swap(Heap[pos], Heap[par]);
            BubbleUpMin(par);
        } else {
            BubbleUpMax(pos);
        }
```

```
        }
    }

    void BubbleUpMin(int pos)
    {
        if (lvl[pos] < 3) return; // no grandparent
        int par = (pos / 2) / 2; // par -> grandpar
        if (Heap[par] > Heap[pos]) {
            swap(Heap[par], Heap[pos]);
            BubbleUpMin(par);
        }
    }

    void BubbleUpMax(int pos)
    {
        if (lvl[pos] < 3) return; // no grandparent
        int par = (pos / 2) / 2; // par -> grandpar
        if (Heap[par] < Heap[pos]) {
            swap(Heap[par], Heap[pos]);
            BubbleUpMax(par);
        }
    }
```

2.2 Реализация: основные функции

Все операции, реализованные на двоичной куче можно также реализовать и на `Min-Max Heap`, вызывая вместо `SiftDown` - `TrickleDown` и вместо `SiftUp` - `BubbleUp`. Так как асимптотика вспомогательных функций одинакова, все методы сохраняют свое время работы. Более точная оценка будет дана позже.

Из того, что не было изложено для бинарной кучи: нахождение максимума:

- Максимум - одна из первых трех вершин кучи, значит нужно просто вернуть наибольшую из них.
- Время работы константно.

2.3 Пример использования: нахождение медианы за `O(1)`

Необходимо построить структуру данных, поддерживающую все операции двоичной кучи, а также поиск максимума и медианы за константное время и их удаление за логарифмическое время. Эту задачу можно решить используя четыре бинарные кучи, однако такое решение довольно громоздко, по сравнению с предложенным далее.

Решим эту задачу используя `Min-max Heap`:

- Будем поддерживать медиану и две `Min-max Heap`. Первая куча содержит все элементы меньшие или равные медиане, вторая - большие или равные. Пусть `R` - медиана, `S1` - 'меньшее' поддерево, `S2` - 'большее' поддерево.
- Будем поддерживать следующие размеры куч: $|S1| = \lceil \frac{n-1}{2} \rceil$ и $|S2| = \lfloor \frac{n-1}{2} \rfloor$. Таким образом размеры поддеревьев отличаются не более, чем на единицу и суммарно использовано ровно n памяти.
- Запрос на добавление: добавим элемент в нужную (относительно медианы) кучу. Если после этого баланс куч испортился, то новой медианой становится либо максимум из `S1`, либо минимум из `S2`. Старая медиана добавляется в соответствующую ей кучу.
- Запрос на удаление: если удаляется медиана, то ее место занимает либо максимум из `S1`, либо минимум из `S2`. При удалении остальных элементов, мог испортиться баланс куч, в случае чего следует проделать аналогичную, описанной выше, операцию.
- Нахождение минимума, максимума и медианы тривиально и имеет константное время работы.

Реализацию `Min-max heap` с поддержкой медианы можно найти в приложении к этому документу и в моем репозитории на `GitHub` [3].

3 Сравнительная статистика

В целом различие во времени работы между бинарной и `min-max` кучей символическое (предполагаем использование двух двоичный куч: по минимуму и максимуму).

Таблица 1: Асимптотики основных функций

Heap	FindMin	FindMax	Insert	DeleteMin	DeleteMax	Decrease key	Build (`O(n)`)
Binary heap	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	ok
Min-max heap	$O(1)$	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	ok

Обе кучи используют $O(n)$ памяти.

Проанализируем константы основных операций, предполагая использование двух бинарных куч:

Таблица 2: Приблизительный анализ констант

Heap	Insert	Delete	Decrease key	Build	memory
Binary heap	$2\log(n+1)$	$4\log(n)$	$4\log(n)$	$4n$	$2n$
Min-max heap	$0.5\log(n+1)$	$6\log(n)$	$6\log(n)$	$12n$	n

Во время `TrickleDown` необходимо каждый раз перебирать всех детей и внуков, что увеличивает константу в 3 раза относительно `SiftDown`. С другой стороны для работы с максимум требуется поддерживать 2 двоичных кучи, из-за чего она проигрывает по памяти.
Так как при вставке элемента в `min-max heap` рассматриваются переходы вверх через одного предка, константа при этой операции $\frac{1}{2}$, а в бинарной куче - 2, так как нужно повторить операцию для дубликата.

4 Тестирование реализаций Min-Max heap и двоичной кучи

Таблица 3: Пройденные тесты

Heap	WAtest	TL1test	TL2test	TL3test	TL4test
Binary heap	*ok*	6.689s	6.746s	15.320s	8.233s
Min-max heap	*ok*	14.406s	13.598s	26.298s	6.956s

Здесь представленна иформация о тестах и результатах, полученных использованием `Min-Max heap` и двоичной кучи [3].

- **WAtest**: Сначала $n = 10^6$ раз добавим в кучу случайное число, потом n раз выполним: добавить случайное число, удалить минимум, удалить максимум. Всего $m = 4 * 10^6$ запросов. Этот тест был использован для проверки корректности реализаций куч.
- **TL1test**: Сначала $n = 10^6$ раз добавим в кучу случайное число, потом n раз удалим минимум. Всего $m = 2*10^6$ запросов. Времена работы на этом (и последующих) тесте представленны в таблице выше.
- **TL2test**: Сначала $n = 10^6$ раз добавим в кучу случайное число, потом n раз удалим масимум. Всего $m = 2 * 10^6$ запросов.
- **TL3test**: Сначала $n = 10^6$ раз добавим в кучу случайное число, потом n раз выполним: добавить случайное число, удалить минимум, удалить максимум. Всего $m = 4 * 10^6$ запросов.
- **TL4test**: $n = 10^6$ раз добавим в кучу случайное число.

5 Заключение

Теоретически, `Min-Max heap` в определенных задачах должен иметь небольшое преимущество, по сравнению с двоичной кучей, но на практике такого результата довольно трудно достичь. Так что я бы скорее отнес `Min-Max heap` к хорошей модификации своего родителя, с которой удобно, приятно, и, зачастую, сильно проще работать.

Ссылки на источники

[1] M. D. ATKINSON, J.-R. SACK, N. SANTORO, and T. STROTHOTTE Min-Max Heaps and Generalized Priority Queues

[2] Wikipedia: Min-max heap

[3] Репозиторий, в котором вы можете найти реализацию `Min-max heap`: link