Часть I
Введение

В данном конспекте рассказывается о такой структуре данных как дерево отрезков(segment tree). Дерево отрезков часто применяется в геометрических и многих других алгоритмах. Оно позволяет для любой аддитивной по отрезкам функции(минимум, максимум, сумма и т.д.) вычислять ее на произвольном отрезке с конечным числом точек за $O(\log N)$, где N — общее число точек в структуре.

Часть II
Постановка задачи и простейшие методы решения

1 Постановка задачи

Предположим, что дана следующая задача: дан массив a из N чисел. Требуется обрабатывать следующие запросы:

- $set(i, x)$ — установить значение i-го элемента, равное x.
- $get(l, r)$ — вычислить сумму с l-го элемента по r-й, $l \leq r$.

2 Простейшие методы решения

Рассмотрим, как можно решать эту задачу, не используя различные структуры данных.

- Наиболее простым в идее и реализации является линейный подсчет суммы чиел в массиве при запросе get за время $O(N)$ и изменение за $O(1)$.
- Также можно предподсчитать суммы для всех возможных отрезков в матрицу sum и запрос get обрабатывать за $O(1)$, непосредственно обращаясь к нужному элементу матрицы. Однако при изменении числа изменяется сумма на порядка N^2 отрезках. Соответственно, для корректной работы необходимо их все обновить, что требует $O(N^2)$ времени.
- Значительно более полезным на практике является метод «частичных сумм». Заключается он в следующем: для каждой позиции i,

$1 \le i \le N$, вычисляется $s(i)$ — сумма элементов от 1 до i-го включительно. Тогда запрос get можно по-прежнему обрабатывать за $O(1)$: $get(l, r) = s(r) - s(l-1)$, при этом $s(0)$ считается равной 0. Однако запрос изменения числа по-прежнему обрабатывается долго: каждое число может участвовать в $O(N)$ частичных суммах, поэтому запрос изменения числа требует $O(N)$ времени на выполнение.

Часть III
Дерево отрезков

3 Описание структуры

Основная идея дерева отрезков такова: предположим, что длина массива является степенью двойки (при необходимости дополним массив нулями). Для каждой степени двойки K, не превышащюей N разобьем весь массив на последовательные отрезки длины K и для каждого предподсчитаем сумму чисел на нем. Запрос $get(l, r)$ тогда можно обрабатывать следующим образом: разбить отрезок с l по r на $O(\log(r-l))$ непересекающихся отрезков, длины которых являются степенями двойки и сумма значений на которых уже посчитана и в силу аддитивности суммы по отрезку сложить. Запрос $set(i, x)$ нужно будет обрабатывать следующим образом: необходимо увеличить значение всех $\log(N)$ отрезков, в которых лежит i-й элемент, на $x - v$, где v — текущее значение i-го элемента.

4 Обработка запросов, хранение и построение дерева отрезков

4.1 Хранение

Рассмотрим, как можно хранить дерево отрезков. Сначала рассмотрим, каким образом можно хранить дерево отрезков в виде подвешенного бинарного дерева: каждому отрезку из описанных ранее разбиений сопоставим отдельную вершину дерева. Вершина u будет являться родителем вершины v тогда и только тогда, когда отрезок, соответствующий u — наименьший из всех отрезков, которые покрывают отрезок, соответствующий v. Тогда корню дерева будет соответствовать отрезок, покрывающий весь массив, а листьям дерева будут соответствовать отрезки, покрывающие отдельные элементы массива. При этом т.к. длины всех отрезков — степени двойки, то у каждой вершины будет ровно два сына (если эта вершина не является листом).

Теперь покажем, как можно хранить дерево отрезков в массиве. В описанном выше бинарном подвешенном дереве для каждой вершины вычислим расстояние от нее до корня и назовем уровнем вершины. Для каждой вершины левым сыном будем считать того из сыновей, которому соответствует отрезок массива, расположенный левее. Пронумеруем вершины следующим образом: корень будет иметь номер 1, его левый сын — номер 2, правый сын — номер 3 и т.д.: слева направо номеровать вершины по мере увеличения уровня. Теперь значение вершины с номером i будем хранить в i-м элементе массива. Заметим тогда, что значение, соотвествующее левому сыну вершины с номером i, будет находиться в $2i$-м элементе массива, правому сыну — в $2i+1$-м, предку — в $i/2$-м.

Всего для хранения дерева отрезков потребуется $O(N)$ памяти: в описанном выше дереве будет одна вершина с расстоянием 0 до корня, две — с расстоянием 1, 2^k — с расстоянием k до корня. Т.к. высота дерева отрезков — $\log N$, то $k \leq \log(N)$ и $1+2+4+..+2^{\log(N)} = 2*2^{\log(N)}-1 = 2*N-1$.

4.2 Обработка запросов

Рассмотрим обработку запросов более подробно. Начнем с запроса $set(i,x)$. Заметим, что если для некторой вершины v известно, что значения ее сыновей являются истинными, то в силу аддитивности суммы по отрезку истинное значение в v можно вычислить как их сумму за $O(1)$. Тогда покажем, как реализовать запрос $set(i,x)$. Рассмотрим наименьший отрезок, в котором находится i-й элемент. Этот отрезок будет являться листом дерева отрезков и ему будет соответствовать отрезок только из i-го элемента. Новым значением суммы на этом отрезке будет x. Теперь перейдем к предку $i/2$ текущей вершины. Значение в одном из его сыновей не менялось и, соответственно, является верным, значение в другом сыне было пересчитано на предыдущем шаге и тоже является истинным. Тогда можно пересчитать значение в $i/2$ и перейти к ее предку. Действуя так, пока предок существует ($v \neq 0$, v — номер текущей вершины), мы обновим все дерево. Т.к. каждый раз при переходе к предку расстояние до корня уменьшается на единицу (а номер вершины делится на 2), то таких переходов будет $O(\log N)$.

Рассмотрим теперь запрос $get(l,r)$. Для его реализации (и всего дерева отрезков) существует два различных метода: «сверху» и «снизу». Описанная выше реализация set является реализацией «снизу», хотя может применяться и в методе «сверху».

При реализации запроса get «снизу» поддерживаетя следующий инвариант: на i-м шаге l и r указывают на крайние левую и правую вершины, находящиеся на уровне $\log N - i$, которым соответствуют отрезки, покрытые еще не обработанной частью исходного запроса. Переход на уровень выше производится следующим образом: заметим, что если вершина l является левым сыном, то ее предок является самой левой вершиной свего уровня, отрезок которой еще полностью не обработан, поэтому перейти на уровень выше можно непосредственно в вершину $l/2$. Если же l — правый сын, то ее отрезок ее предка уже частично обработан либо не лежит в запросе, поэтому переходить в него нельзя, но требуется обновить ответ, прибавив к нему значение в l и перейти в вершину $l+1$, которая является левым сыном своего предка. Аналогично для вершины r — если r правый сын, то можно перейти в его предка, иначе — добавить в ответ значение в вершине r, перейти в $r-1$, которая является правым сыном своего предка, и перейти в предка $r-1$. Действовать так нужно, пока обрабатываемый отрезок не пуст, т.е. $l \leq r$. При этом т.к. каждый раз происходит переход на уровень вверх, а уровней всего $\log N$, то запрос будет обрабатываться за $O(\log N)$.

При реализации запроса $get(l, r)$ сверху необходимо хранить в вершине дополнительную информацию — границы отрезка, которы соответствует этой вершине. Обработка запроса в этом случае происходит следующим рекурсивным методом: если v — текущая вершина, v_l и v_r — границы отрезка, который ей соответствует, то тогда:

- Если $l \leq v_l$ и $v_r \leq r$, то к ответу нужно добавить значение в v.
- Если $v_r \leq l$ или $r \leq v_l$, то отрезок вершины v не лежит в запросе и обрабатывать v не нужно.
- Если ни одно из перечисленного не верно, то результатом запроса для вершины v является сумма запросов сыновей v — $2v$ и $2v+1$.

Заметим, что при таком методе на каждом уровне может быть посещено не более четырех вершин, что также дает оценку $O(\log N)$ на время работы. Аналогично можно обрабатывать запрос set.

4.3 Построение

Наконец, покажем, как построить над массивом дерево отрезков. За $O(N \log N$ это можно сделать следующим образом: построить дерево отрезков на массиве из нулей (тогда в каждой вершине будет храниться 0) и для каждого элемент массива произвести $set(i, a[i])$, где $a[i]$ — исходный массив.

Однако построить дерево отрезков можно быстрее — за $O(N)$, используя один линейный проход по нему. Сначала заполним самый нижний уровень. В нем хранятся суммы на отрезках с i по i, т.е. элементы исходного массива. Заметим, что если k-й уровень заполнен, то можно заполнить $k-1$, для каждого его элемента посчитав значение в нем как сумму значений в сыновьях. Таким образом, если сначала заполнить нижний уровень элементами массива, а затем пройти по всем уровням снизу вверх, действуя указанным методом, то значения всех вершин дерева отрезков будут посчитаны корректно за линейное время.

Часть IV
Реализация

5 Пример реализации дерева отрезков на Java

Пример реализации дерева отрезков на сумме без групповых операций. В конструкторе $SegTree$ параметр $numberOfValues$ — количество элементов в массиве, над которым строится дерево отрезков. Элементы массива читаются из входного потока методом $nextLong$. В массиве $tree$ хранится само дерево отрезков, $size$ — количество листьев в дереве отрезков, inf — нейтральный элемент, для сложения равный нулю.

Реализация снизу:

```
public class SegTree {
    private long[] tree;
    private int size;
    private final long inf = 0;

    private void update(int vertex) {
      this.tree[vertex] = calc(this.tree[vertex * 2],
          this.tree[vertex * 2 + 1]);
    }

    private long calc(long a, long b) {
      return a + b;
    }

    public SegTree(int numberOfValues) throws IOException {
      this.size = 1;
      while (this.size < numberOfValues) {
        this.size *= 2;
      }
      this.tree = new long[this.size * 2];
      Arrays.fill(tree, inf);
      for (int i = 0; i < numberOfValues; i++)
        tree[i + size] = nextLong();
      for (int i = size - 1; i > 0; i--)
        update(i);
```

```
    }

    public void set(int pos, long newValue) {
        pos += size;
        this.tree[pos] = newValue;
        while (pos > 1) {
            pos /= 2;
            update(pos);
        }
    }

    public long get(int left, int right) {
        left += this.size;
        right += this.size;
        long answer = this.inf;
        while (left <= right) {
            if (left % 2 == 1) {
                answer = calc(answer, this.tree[left]);
                left++;
            }
            if (right % 2 == 0) {
                answer = calc(answer, this.tree[right]);
                right--;
            }
            left /= 2;
            right /= 2;
        }
        return answer;
    }
}
```

Реализация сверху:

```java
public class SegTree {
  private Vertex[] tree;
  private int size;
  private final long inf = 0;

  private class Vertex {
    private long left;
    private long right;
    private long value;

    public Vertex(long left, long right, long value) {
      this.left = left;
      this.right = right;
      this.value = value;
    }
  }

  public SegTree(int numberOfElements) throws IOException {
    size = 1;
    while (size < numberOfElements)
      size *= 2;
    tree = new Vertex[size * 2];
    for (int i = 0; i < numberOfElements; i++) {
      tree[i + size] = new Vertex(i, i, nextLong());
    }
    for (int i = numberOfElements; i < size; i++)
      tree[i + size] = new Vertex(i, i, inf);
    for (int i = size - 1; i > 0; i--) {
      tree[i] = new Vertex(tree[i * 2].left, tree[i * 2 + 1].right,
          tree[i * 2].value + tree[i * 2 + 1].value);
    }
  }

  public long get(int vertex, int left, int right) {
    if (tree[vertex].left > right || tree[vertex].right < left)
      return inf;
    if (tree[vertex].left >= left && tree[vertex].right <= right)
      return tree[vertex].value;
    long answer = get(vertex * 2, left, right)
        + get(vertex * 2 + 1, left, right);
    return answer;
  }
```

```java
  public void set(int vertex, int left, int right, long value) {
    if (tree[vertex].left > right || tree[vertex].right < left)
      return;
    if (tree[vertex].left >= left && tree[vertex].right <= right) {
      tree[vertex].value = value;
      return;
    }
    set(vertex * 2, left, right, value);
    set(vertex * 2 + 1, left, right, value);
    tree[vertex].value = tree[vertex * 2].value
        + tree[vertex * 2 + 1].value;
  }

}
```