Конспект лекции по теме "RMQ"

Тихомиров М.И.

3 января 2011 г.

1 Постановка задачи RMQ

Рассмотрим задачу: задан массив длины N, заполненный числами. Необходимо отвечать на запросы вида:

1) сделать i-ый элемент массива равным x;

2) найти минимальное число на отрезке $[l; r]$.

Эта задача называется задачей *RMQ (Range Minimum Query)*. Существует две модификации RMQ:

1) RMQ off-line: отсутствуют запросы типа 1, т.е. элементы массива не изменяются между запросами.

2) RMQ on-line: запросы типа 1 и 2 могут чередоваться.

Далее мы будем рассматривать только on-line вариант задачи.

2 Некоторые варианты решения

2.0 Решение "в лоб"

Самым простым решением будет изменение элементов массива непосредственно и линейный поиск минимума на отрезке. Сложность ответов на запросы в худшем случае — $O(1)$ и $O(N)$ соответственно, что в большинстве случае не удовлетворяет требованиям на время работы, т.к. количество запросов второго типа может быть очень большим.

2.1 Корневая оптимизация

Разобьем массив на отрезки длины $L = \lfloor\sqrt{N}\rfloor$ (оставшуюся часть рассмотрим как отдельный отрезок в разбиении), количество отрезков обозначим за $K(\leq \sqrt{N})$. Заведем дополнительный массив длины K, в i-ом элементе массива будем хранить минимум на i-ом отрезке в разбиении.

Предподсчет для организации такой структуры на начальном массиве будет занимать $O(N)$ операций.

Для выполнения запроса типа 1 необходимо обновить ровно один элемент дополнительного массива — соответствующий отрезку, содержащему изменяемый элемент. Для этого выполним линейный поиск минимума на этом отрезке, что потребует $O(L) = O(\sqrt{N})$ операций.

Пусть нам нужно найти минимум на отрезке $[l; r]$. Рассмотрим те отрезки разбиения, которые целиком находятся внутри $[l; r]$. Минимумы на этих отрезках нам известны — это элементы дополнительного массива. Таким образом, минимум M_1 среди тех элементов отрезка $[l; r]$, которые находятся в целых отрезках разбиения, можно вычислить за $O(K)$ перебором нужных элементов доп. массива.

Оставшиеся элементы находятся по краям отрезка. Поскольку каждый из краев лежит целиком в каком-то отрезков из разбиения, их суммарная длина не превосходит $2L$. Минимум M_2 среди этих элементов находится простым перебором этих элементов за $O(L)$ действий.

Ответ на запрос есть минимум из M_1 и M_2. Итоговая сложность — $O(K + L) = O(\sqrt{N})$. Можно доказать, что это разбиение оптимально, т.е. при таком подходе нельзя улучшить сложность запросов в худшем случае.

3 Дерево отрезков

3.1 Определение, высота, хранение в памяти

Рассмотрим подвешенное бинарное дерево, где в каждой вершине хранится минимум на некотором подотрезке отрезка $[1; N]$. Кроме того, выполнено следующее: для каждой вершины либо соответстующий ей отрезок состоит из одного элемента, и тогда у нее нет детей, либо отрезки, соответствующие ее детям, не пересекаются и в объединении дают отрезок для данной вершины. Корню сопоставлен отрезок $[1; N]$.

Назовем такое дерево *деревом отрезков* на отрезке [1; N].

Для балансировки дерева будем считать, что длины отрезков сыновей каждой вершины отличаются не более, чем на единицу. Если это верно, то высота дерева h равна $\lfloor log_2 N \rfloor + 1$, что можно доказать индукцией по N.

Для хранения дерева отрезков воспользуемся схемой, аналогичной схеме хранения кучи. Для этого заведем массив $tree[i]$ размера $2^h - 1$ для хранения . Будем хранить корень в вершине с номером 1, и скажем, что левый и правый сыновья вершины с номером i имеют номера $2i$ и $2i + 1$ соответственно. Кроме того, для удобства заведем два массива $left[i]$ и $right[i]$, в i-ых элементах которых будут хранится концы отрезка, соответствующего i-ой вершине.

Чтобы создать дерево по заданному массиву, требуется заполнить массивы $tree[i]$, $left[i]$ и $right[i]$. Это можно сделать при помощи следующей рекурсивной процедуры:

```
create(i, l, r) { // i - номер текущей вершины, l и r - концы текущего отрезка
  if (l == r) {
    tree[i] = arr[l]; //arr - исходный массив
  } else {
    left[i] = l; // заполняем left и right
    right[i] = r;
    m = (l + r) / 2;
    create(2*i, l, m);
    create(2*i + 1, m + 1, r);
    tree[i] = min(tree[2*i], tree[2*i + 1]);
  }
}
```

3.2 Операция изменения

Разобьем вершины дерева на слои по удаленности от корня: в первом слое находится корень, во втором — его дети, в третьем — дети детей, etc. Заметим, что в каждом слое отрезки не пересекаются, поэтому при изменении одного элемента массива на каждом слое дерева изменится не более одной вершины, а значит, всего нужно изменить не более h вершин дерева.

Поступим следующим образом: запустим рекурсивную процедуру по дереву. Если отрезок текущей вершины не содержит измененного элемента, выходим из процедуры. Иначе если длина отрезка равна одному, то меняем минимум явно (он равен новому значению элемента), в противном случае запустимся от детей вершины и после этого пересчитаем минимум в текущей вершине. На каждом слое произвойдет не более одного рекурсивного вызова, значит, время работы процедуры — $O(h) = O(\log N)$. Ниже приведен код этой процедуры (предполагается, что в массиве arr элемент уже изменен):

```
change(i, j) { // j - номер измененного элемента
  if (j < left[i] or j > right[i]) return;
  if (left[i] == right[i]) {
    tree[i] = arr[j];
  } else {
    change(2*i, j);
    change(2*i + 1, j);
    tree[i] = min(tree[2*i], tree[2*i + 1]);
  }
}
```

3.3 Операция нахождения минимума

Пусть требуется найти минимум на отрезке $[l; r]$. Поскольку в каждом слое отрезки не пересекаются, все отрезки разбиваются на три группы: те, которые не пересекаются с $[l; r]$; те, которые целиком лежат в $[l; r]$; и все остальные — те, которые содержат концы $[l; r]$.

Напишем рекурсивную процедуру, которая будет вычислять минимум на пересечении отрезка текущей вершины и $[l; r]$. В зависимости от типа вершины нужно совершить одно из трех действий: либо вернуть $+\infty$ (в случае, когда отрезки не пересекается), либо вернуть $tree[i]$ (в случае, если отрезок целиком лежит в $[l; r]$), либо рекурсивно запуститься для детей вершины и вернуть минимум (в остальных случаях). Поскольку "нетривиальных" отрезков на каждом слое будет не более двух, время работы процедуры составляет $O(\log n)$. Код процедуры:

```
countMin(i, l, r) { // l и r - концы отрезка, на котором ищем минимум
  if (left[i] > r or right[i] < l) then return INF; // первый случай
  if (l <= left[i] and right[i] <= r) then return tree[i]; //второй случай
  return min(countMin(2*i, l, r), countMin(2*i + 1, l, r));
}
```