Ленивый кот

Решение.

Задача, в частности, приятна тем, что для неё имеется много различных (и эффективных) методов решения. Все они упираются в выбор подходящей структуры для хранения и поиска данных, но эти структуры могут быть весьма разнообразны.

Я изложу решение, основанное на хранениия данных в бинарном дереве. Очень советую почитать учебные материалы, посвящённые очень красивой (и очень близкой, фактически такой же) структуре данных – дереву отрезков. Учебные материалы содержат два небольших фрагмента из книги

Препарата Ф., Шаймос М. Вычислительная геометрия: Введение, 478 стр. М.: Мир, 1989,

и статью

"Segment trees and interval trees" by Antoine Vigneron,

размещённую по адресу

http://w3.jouy.inra.fr/unites/miaj/public/vigneron/cs4235/l5cs4235.pdf

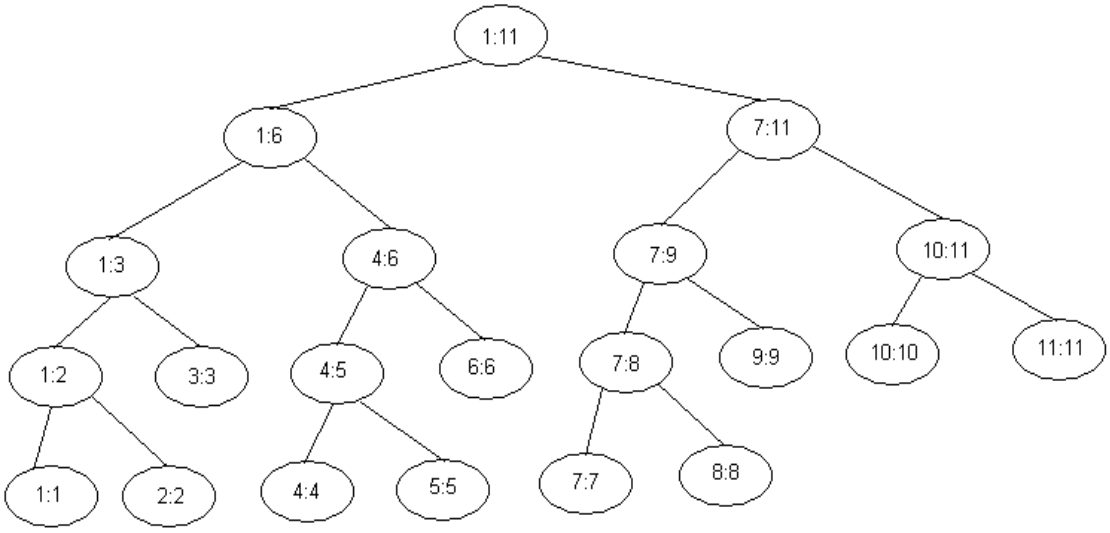
Оба материала вращаются вокруг классического применения дерева отрезков – задачи об объединении ортогональных прямоугольников, задачи самой по себе весьма поучительной, но и имеющей непосредственные выходы к нашей задаче.

Вот и вернёмся к задаче про кота. Заведём бинарное дерево, в котором каждый узел отвечает за некоторый отрезок ряда лампочек, скажем узел v хранит информацию о лампочках от L(v) до R(v). Корень дерева хранит информацию обо всех лампах: L(root)=1, R(root)=N, где N – количество ламп (ну, или длина отрезка, говоря более формально).

Если L(v)<R(v) – вершина отвечает за отрезок из 2-х или более ламп, - то к вершине v пристраиваем два поддерева: левое, отвечающее за отрезок ряда от L(v) до (L(v)+R(v)) div 2, и правое, отвечающее за отрезок ряда от (L(v)+R(v)) div 2 до R(v).

Если же L(v)=R(v) – вершина отвечает за единственную лампу, - то вершина v является листом дерева.

На рисунке приведён пример такого дерева для N=11.



Отметим, что это дерево зависит только от N и ни от чего больше. В процессе решения задачи само дерево не изменяется, в этом смысле построенное дерево - это статическая струкутра. Будем изменять только некую дополнительную информацию, хранящуюся в вершинах дерева.

Перечислю несколько ценных свойств такого дерева:

1. Каждая нелистовая вершина имеет ровно двух потомков.
2. Дерево имеет ровно N листьев. Из этих двух очевидных утверждений сразу следует, что
3. дерево имеет ровно 2N-1 вершин.
4. Если две вершины расположены на одном уровне (на одинаковом расстоянии от корня), то длины отрезков, за которые отвечают эти вершины отличаются не более, чем на 1
5. Отсюда сразу следует, что все листья располагаются максимум на двух уровнях, и высота дерева (максимальное расстояние от корня до листа) не превосходит $\lceil \log_2 N \rceil$.
6. Любой отрезок может быть представлен в виде объединения не более, чем $2\lceil \log_2 N \rceil$ попарно непересекающихся отрезков, каждый из которых соответствует некоторой вершине дерева.

Все эти утверждения доказать несложно, и я не буду здесь приводить их доказательства. Чуть менее очевидно разве что свойство 6. Очень советую внимательно разберите процедуры ВСТАВИТЬ и УДАЛИТЬ (и рисунок 1.2, конечно) из текста в файле `prep1.pdf` (это первый фрагмент из книги *Препарата, Шаймос*).

Для решения данной задачи будем в каждой вершине v построенного дерева хранить две дополнительные величины: first – номер первой (самой левой) включённой лампы на отрезке от L(v) до R(v), и last – номер последней (самой правой) включённой лампы на отрезке от L(v) до R(v). Если на отрезке нет включённых ламп, то полагаем first = +∞, а last = -∞. В программной реализации можно представить +∞ числом N+1, а -∞ - числом 0.

Первым делом создаём дерево. Из свойства 3 сразу следует, что для хранения дерева достаточно завести массив из 2N-1 вершин, и что создание дерева требует O(N) операций. В приведённой программе дерево хранится в массиве `tree`, и создаёт его процедура `CreateTree`.

Переключение лампочки номер P требует изменить характеристики first и last во всех вершинах, отвечающих за отрезки, содержащие позицию P. Количество таких вершин не превосходит $\lceil \log_2 N \rceil$ (свойство 5). Процедура переключения выполняется следующим образом:

если вершина v – лист, то устанавливаем last и first равными P или $\pm\infty$ в зависимости от того, включаем мы лампочку или выключаем; отметим, что информацию о состоянии лампочки перед переключением можно отдельно не хранить – её легко узнать из значения first (или last) вершины v;

если же вершина v – не лист, то first равен меньшему из значений first сыновей вершины v, а last - большему из значений last сыновей вершины v (вот где у нас срабатывают $\pm\infty$, и вот почему можно представлять +∞ числом N+1, а -∞ - числом 0).

Обработка каждой вершины требует константного времени, всего изменяются характеристики $\lceil \log_2 N \rceil$ вершин, т.е. обработка каждого запроса на переключение лампочки требует O(logN) операций.

В приведённой программе переключает лампочки процедура `Switch`.

Осталось узнать, куда переместится Кот, когда переключают лампу, от которой он греется.

Если Кот двигается вправо (в сторону увеличения номеров ламп), то мы, во-первых, выключаем лампу номер Cat (Cat – это позиция Кота), а, во-вторых, ищем первую

включённую лампу на отрезке от лампы номер Cat до лампы номер N. Если такой лампы нет (т.е. она расположена в +∞), то поворачиваем Кота влево, и ищем последнюю (самую правую) включённую лампу на отрезке от 1 до Cat.

Если Кот направлен влево, то поступаем совершенно аналогично.

Осталось научиться находить первую включенную лампу на отрезке от Cat до N, и последнюю включённую лампу на отрезке от 1 до Cat. А это совсем несложно в свете вышесказанного: если мы разобьём отрезок S на несколько непересекающихся (впрочем это условие не обязательно) отрезков, то номер первой включённой лампы на S равен наименьшему из номеров включённых ламп на каждом из отрезков разбиения. Напомню свойство 6: любой отрезок может быть представлен в виде объединения не более, чем $2\lceil \log_2 N \rceil$ попарно непересекающихся отрезков, каждый из которых соответствует некоторой вершине дерева. Вот на эти вот отрезки и будем разбивать отрезок S. Фактически операция рабиения это и есть операция ВСТАВИТЬ из текста в файле `prep1.pdf`.

Количество отрезков разбиения не превосходит $2\lceil \log_2 N \rceil$, следовательно сложность вычисления первой включённой лампы на любом отрезке S равна O(logN). Всё сказанное по поводу поиска первой включённой лампы почти дословно переносится на поиск последней включённой лампы на отрезке S.

В приведённой программе эти поиски реализованы функциями `Last` и `First`.

Итак, и запрос на переключение лампы, и запрос на опеределение нового местоположения Кота имееют сложность O(logN), следовательно общая сложность решения задачи есть $O(M \cdot \log N)$, где M – количество запросов, при затратах памяти O(N).

В заключение я всё-таки не могу отказать себе в удовольствии процитировать Михаила Рыбака. Он очень коротко описывает своё решение так: "… придумал изящное решение с двумя кучами (сет жутко медленный). В одной куче - лампочки, которые правее, в другой - которые левее (только лампочки, фигурировавшие в событиях). Причем можно не удалять тогда, когда лампочка гаснет, а вместо этого, когда кот пересаживается, проверять, горящая ли лампочка наверху кучи, и если нет, ее удалять и опять смотреть" (*куча – это, конечно же, бинарная куча. С.М)*. Как красиво! Мне очень понравилось. Спасибо, Михаил.