Домашнее задание. Задача 5.2

И.В. Махотин

Описание алгоритма. Итак, у нас есть изначально n свободных мест на круговой парковке. А также последовательность из m произошедших событий.

Решать задачу будем с помощью дерева отрезков, для этого: заведем массив $parkingPlaces$ длины n для листов дерева, где в каждой ячейке будем хранить либо номер в массиве $+1$ (минимальное свободное место на отрезке длинны 1) в случае если место свободно, либо -1 в случае если место занято.

Далее уже до конца строим дерево отрезков $freePlaceTree$ такое, что в каждой вершине хранится свободное место с минимальным номером на соответствующем отрезке, в случае отсуствия такового будем хранить в вершине -1.

В вершинах, в том числе и в листовых, кроме указателей на правого и левого сына будем хранить еще и указатель на отца, а также определим тривиальные функции для вершин:

- isFree() – есть ли свободное место на отрезке.
- getMinFreePlace() – вернуть свободное место с наименьшим номером, либо вернуть -1, если свободных мест нет.
- setMinFreePlace(int newMinPlace) – обновить значение места с минимальным номером на отрезке вершины.

Под левым сыном будем понимать того, в чей подотрезок находится левее, то есть номера мест на отрезке левого сына меньше.

Пусть у нас будут две функции возвращающие целые значения:

- takeCarPlace(size_t placeNo)
- leaveCarPlace(size_t placeNo)

Будем вызывать соответсвующюю функцию в зависимости от события. События будем обрабатывать по ходу.

Опишим подробнее сами функции:

Функция takeCarPlace возвращает -1 если корень дерева занят , то есть на всей парковке нет свободного места. В противном случае мы идем к листу $ParkingPlaces[placeNo-1]$, если там свободно, занимаем это место с последующим обновлением значений в $freePlaceTree$, либо ищем во всех отрезках правее текущего листа в массиве $ParkingPlaces$ следующим образом:

1. Пока мы в правом сыне поднимаемся вверх, если уперлись в корень и в нем нет свободных мест, то возвращаем -1, иначе возвращаем значение корня, в соответствующем листе дерева выставляем -1 и обновляем дерево.

2. В противном случае(мы оказались в левом сыне), идем в правого брата (вверх-вправо) B. Если в правом брате нет мест или его вовсе не существует, то поднимаемся дальше. Если в нем есть место, то возвращаем $getMinFreePlace()$ от него, правим лист в массиве, обновляем $freePlaceTree$.

Функция leaveCarPlace возвращает -2, если $ParkingPlaces[placeNo-1]$ уже и так свободно. В другом случае устанавливаем значение $placeNo$ в листе соответствующем $ParkingPlaces[placeNo-1]$, то есть освобождаем место и идем вверх по дереву отрезков обновляя значения минимального свободного места на отрезке текущей вершины, пока не встретим в $getMinFreePlace()$ не отрицательное значение с местом меньшим чем $placeNo$.

Доказательство. Обновление дерева отрезков рассматривалось на лекциях тут все аналогично с точностью до поддерживаемого параметра.

Рассмотрим возможные случаи:

- Мы находимся не в корне, при этом на отрезке принадлежащем текущей вершине нет свободных мест больших $placeNo$.

  Если мы в правом сыне, то очевидно, следующий отрезок будет охватываться ближайшим «правым дядей» текущей вершины, так как «левые дяди» отвечают за отрезки с номерами парковочных мест меньшими чем $placeNo$, и отец «предка» и «дяди» также будет охватывать места меньшие чем $placeNo$.

  («дядей» здесь называется вершина, которая является братом предка, а не только отца, текущей вершины).

  Также понятно, что не будет зазора между отрезком найденного «правого дяди» и текущей вершиной, так как текущая вершина отвечает за самый правый подотрезок «брата правого дяди»(ближайшего левого предка текущей вершины), а между «правым дядей» и его братом нет зазора, так что какое либо свободное место в отрезке «отца правого дяди» мы не пропустим.

  Получается, что за следующий отрезок действительно отвечает «правый дядя».

- Если мы в левом сыне, то очевидно ближайший следующий отрезок будет охватываться нашим «правым братом», поэтому стоит проверить его и в случае отсутствия свободных мест подниматься вверх в поисках «правого дяди». Случай, когда правого брата не существует обрабатывается также – ищем следующего «правого дядю», так как отрезки отца и сына совпадут, а в сыне нет искомого свободного места.

- Мы находимся в корне, в этом случае на всем отрезке мы не нашли большего свободного места чем, $placeNo$, так как область поиска следующего большего свободного места должна быть не в этом поддереве, а либо в «братском»(мы в левом сыне), либо в одном из «дядь»(мы в правом сыне). У корня ни «братьев», ни «дядь», а следовательно на всем орезке нет следующего свободного места, большего чем $placeNo$. По условию задачи мы делаем круг и выбираем минимальное свободное место на всем отрезке парковочных мест, а оно либо находися в корне, либо его вообще нет (в корне -1).

- Если «правого дяди» нет, то мы упремся в корень и произойдет ситуация описанная выше.

Сложность. В итоге высота дерева $\lceil log_2(n) \rceil$. Значит сложность операций в худшем случае $O(log(n))$. Построение дерева можно сделать за $O(n)$, так как минимальные свободные места находятся тривиально. Итого сложность по времени $O(m \cdot \lceil log_2(n) \rceil + n)$.

Алгоритм требует $O(n)$ дополнительной памяти.