Домашнее задание №6

Томинин Ярослав, 778 группа

17 марта 2018 г.

1.

Составим алгоритм действий, когда нам потребуется выполнить pop. В одной нашей очереди будут находиться все элементы по порядку их добавления. Перекинем n-1 элементов из этой очереди в другую, тогда мы с легкостью сможем извлечь последний элемент. То есть одна из наших очередей пустая до тех пор, пока нам не понадобиться извлечь элемент.

2.

Это дерево будет храниться следующим образом. Если мы нарисуем это дерево, то 1-корень, спускаемся на следующий уровень и нумеруем вершины по порядку слева направо и т.д. Поймем, что при добавлении новых элементов это дерево всегда будет оставаться почти полным. Поймем, что номер родителя i-ого элемента равняется (i-1) div 3 . Для этого докажем по индукции, что для k-ого уровня самый левый элемент будет равен $\underbrace{1...1}_{k}$, а самый правый элемент $\underbrace{1...1}_{k}0$. База: первый уровень. Переход. Если i-ый уровень оканчивается на $\underbrace{1...1}_{i}0$, то i+1 начинается на $\underbrace{1...1}_{i+1}$. Так как в предыдущем уровне было $\underbrace{1...1}_{i}0 - \underbrace{1...1}_{i}+1 = 1\underbrace{0...0}_{i}$. Значит i+1 будет оканчиваться на $\underbrace{1...1}_{i+1}+1\underbrace{0...0}_{i}0 = \underbrace{1...1}_{i+1}0$. Доказано. Теперь поймем,что если мы возьмем первые три элемента на i+1 уровне и применим к ним нашу операцию (k-1) div 3, то эти три элемента дадут $\underbrace{1...1}_{i}$. Так как наша операция удаляет последнюю цифру в троичной записи, то следующие 3 числа будут потомками второго и для них будет выполнена та же функция. Следовательно на каждом уровне мы можем найти родителя с помощью такой формулы: (k-1) div 3.

Вычислим номера детей. Для этого применим обратную операцию. Отсюда следует, что у родителя k элементы 3к+1, 3к+2, 3к+3 являются детьми.

3.

″Ответы при сравнении″-путь на дереве, который задается сравнениями с числами, стоящими в вершинах. Понятно, что если числа являются соседними по ключу(если их сортировать), то их пути будут совпадать вплоть до того, пока они не накнутся в вершину, соответствующюю значению одного из этих двух чисел.

Попробуем доказать это от обратного. Если у x нет правого ребенка и у x есть следующий за ним в порядке возрастания y, но при этом y не является самым нижним предком x, у которого левый дочерний узел так же является предком x. Поймем, что y не может стоять на уровне,

равном x или ниже, так как это означало бы, что ответы при сравнении числа y совпадали бы с ответами при сравнении x, это означатает, что y x был бы правый потомок. Но у нас другая задача. Следовательно y находится выше x. Теперь поймем, что x является потомком y, так как все ответы на вопросы при сравнении x совпадают с ответами на вопросы y вплоть до самого y, после этого x уходит на сравнение к левому потомку y. По нашему предположению y не является самым нижним предком x, у которого левый дочерний узел так же является предком x , следовательно существует такое d через которую можно провести путь y-левый потомок(y)-d-x. Причем, если d-предок x, у которого левый дочерний узел так же является предком x, то он не может быть меньше y(если бы он был меньше y, то правый потомок был бы тоже меньше y и меньше x, так как x и y подрядидущие). Это означает, что если x является потомком d, то d является его правым предком, а не левым. Следовательно y является самым нижним предком x, у которого левый дочерний узел так же является предком x. **4.**

″Ответы на вопросы″-путь на дереве, который задается сравнениями с числами, стоящими в вершинах. Понятно, что если числа являются соседними по ключу(если их сортировать), то их пути будут совпадать вплоть до того, пока они не накнутся в вершину, соответствующюю значению одного из этих двух чисел.

″Пути чисел″путь на дереве

С помощью предыдущего утверждения осознаем, что если пути чисел a и b расходятся в точке a, то y b не будет левого элемента, так как все элементы, меньшие b, будут меньше a и они будут уходить из a на левого потомка, а не на правого. Аналогично, если пути чисел c и b расходятся в точке c, то y b не будет правого потомка, так как все числа, большие b, будут больше c и будут при сравнении с c уходить на правого потомка а не на левого. Следовательно пути a и b расходятся в точке b , и пути b и c расходятся в точке b. Аналогично предыдушим рассуждениям у a не будет правого потомка, так как все элементы, большие b будут больше a, а y c не будет левого потомка, так как все элементы, меньшие c, будут меньше b.

5.

1)Чтобы понять, лежит ли один из n элементов в нашем множестве за один вопрос наша структура должна сожержать вопрос, из которого будет понятно лежит ли элемент в мн-ве A. Если в вопросе мы будем спрашивать больше, чем про один элемент, то ответ да и нет не даст нам понятия лежит ли наш элемент в мн-ве A(так как может быть 2^k различных вариантов и закодировать их с помошью 1 бита мы не сможем). Следовательно, если нам надо узнать лежит ли элемент в мн-ве A за

один вопрос, то этот вопрос должен спрашивать только про 1 элемент. Это означает, что под каждые n элементов мы должны хранить свой вопрос. Докажем достаточность этих вопросов. Она очевидна, так как под любой элемент у нас будет вопрос, который будет отвечать нам лежит элемент или не лежит в множестве A.
2) Докажем, что мы сможем хранить $s = \lceil \log n \rceil$ строк. Для этого создадим

6.

Допустим мы расставили наших клиентов в таком порядке так, чтобы суммарное время их ожидания было минимальным. Докажем, что они стоят в порядке возрастания времени. Так как общее время ожидания равно

$$\tau_o = \sum_{i=1}^{n} (n + 1 - i)\tau_i$$

То, если найдутся два человека i<j, для которых $\tau_i > \tau_j$. Мы поменяем их местами и τ_0 станет меньше, так как $\tau_i(n + 1 - j) + \tau_j(n + 1 - i) < \tau_j(n + 1 - j) + \tau_i(n + 1 - i)$ при i<j. Противоречие.
Теперь поймем, что наш алгоритм должен просто сортировать массив. А мы уже знаем, что самые эфективные алгоритмы сортиовки $O(n \log n)$

7.

Обозначим за a_k- переменную, которых будет бесконечное количество. Теперь при сосдании новой переменной будем вносить в стек сначала ее значение, а потом ее название. Тогда мы будем хранить все наши переменные в одном стеке, при этом мы будем использовать ограниченную оперативную память. При произвольной операции мы будем выгружать наши переменные из стека в оперативную память следуюшим образом: будем pop-ать и смотреть на название этой переменной, если это та переменная, которая используется в операции, то следующим pop-ом заносим ее значение в регистр. При этом во второй стек push-им все, что мы достали из первого стека. После того, как мы достали все необходимые переменные в оперативную память, проведем операцию и при необходимости, если мы создаем новую переменную, добавим ее значение в стек. Далее из всоторого стека перекинем все переменные обратно в 1 стек.