Курс с++

Царьков Олег

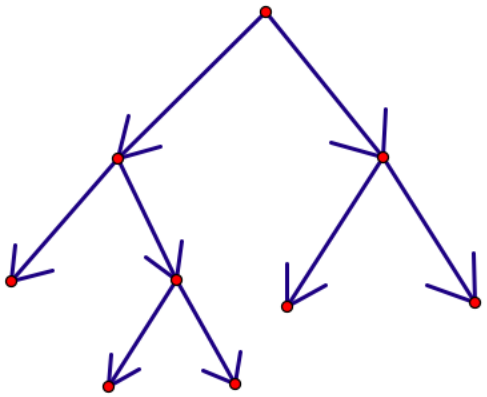
Описание тикета OTSN-12

Бинарное дерево.

Дерево — это структура данных, в которой есть несколько объектов — вершин, и у каждой вершины есть указатели на несколько других вершин. Вершины, на которые указывает данная вершина, называются ее сыновьями.

Список является частным случаям дерева, когда у каждой вершины ровно 1 сын.

Бинарное дерево — чатсный случай дерева, когда у каждой вершины два сына. Они называются соответственно левым и правым сыном. На картинке бинарное дерево можно изобразить следующим образом:



Обходы дерева.

Предположим, что у каждой вершины есть какое-то содержимое, например, номер вершины, или число, написанное на ней.

Для бинарных деревьев есть три типа обхода :

- $pre - order$
- $in - order$
- $post - order$

Все три определяются следующим образом:

Если у дерева одна вершина, то все три типа обходов просто печатают ее содержимое.

Предположим, что для всех деревьев размера $\leq n - 1$ определено, что такое $pre - order$, $in - order$ и $post - order$ обходы. Определим эти три понятия для дерева размера n.

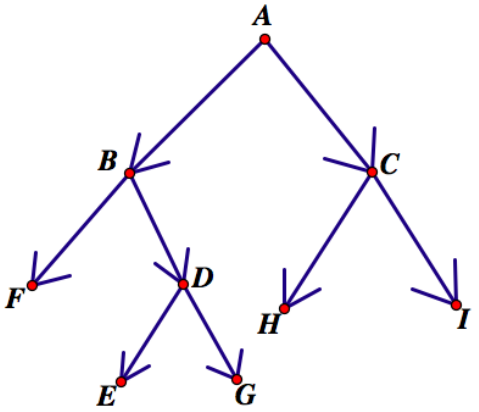
Рассмотрим верхнюю вершину дерева A, возьмем ее левое поддерево(левый сын и все его сыновья и сыновья его сыновей и так далее) α и правое поддерево β.

• $pre-order$ обход для такого дерева вначале печатает содержимое вершины A, потом делает то, что делает $pre-order$ обход для дерева α (мы уже знаем, что он делает, так как в дереве α имеются $\leq n-1$ вершины), потом делает то, что делает $pre-oder$ обход для дерева β.

• $in-order$ обход вначале делает то, что делает $in-order$ обход для дерева α, потом печатает содержимое A, потом делает то, что делает $in-order$ обход для дерева β.

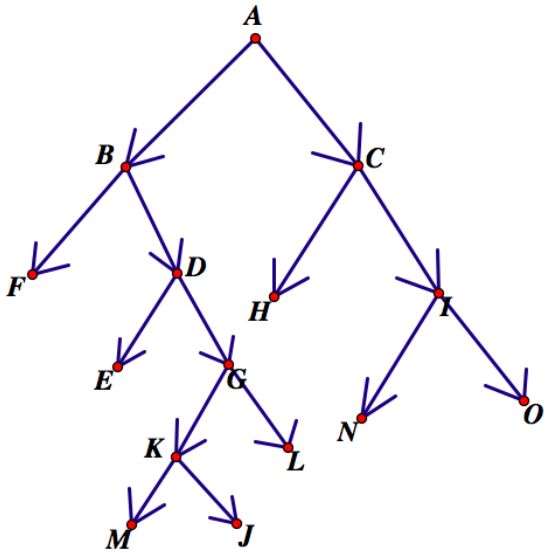
• $post-order$ обход вначале делает то, что делает $post-order$ обход для дерева α, потом делает то, что делает $post-order$ обход для дерева β, потом печатает содержимое вершины A.

Рассмотрим дерево



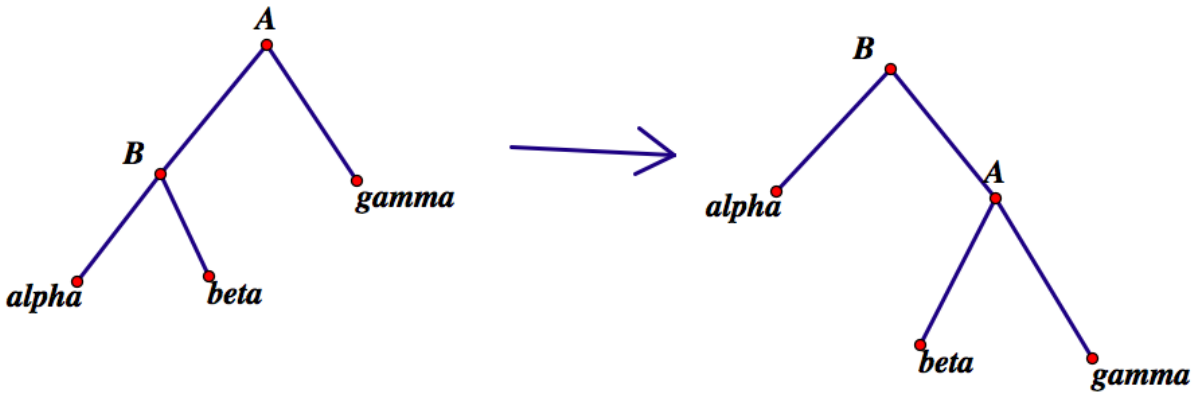
Его $pre-order$ обход таков: $ABFDEGCHI$
Его $in-order$ обход таков: $FBEDGAHCI$
Его $post-order$ обход таков: $FEGDBHICA$

Задание 1. Выписать $pre-order$, $in-order$ и $post-order$ обходы для дерева

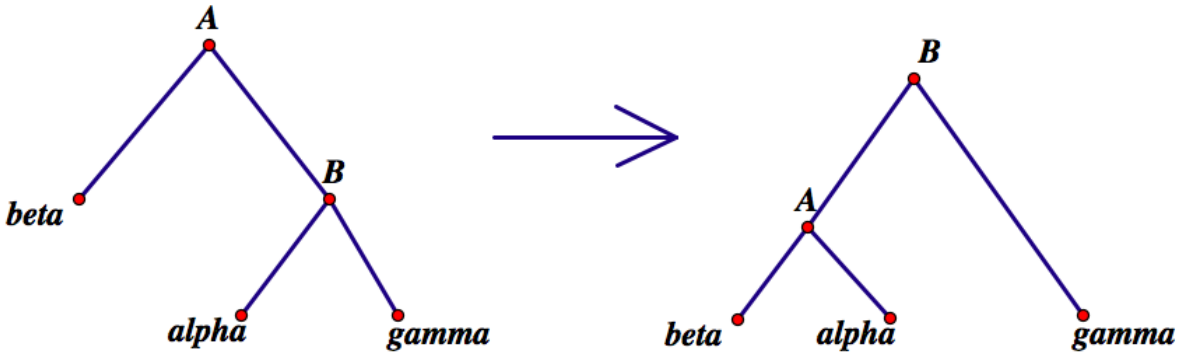


Повороты дерева.

Пусть A, B — вершины, α, β, γ — поддеревья
Правый поворот:



Левый поворот:



Эти операции изменяют только взаимное расположение вершин A, B и α, β, γ, взаимное расположение остальных вершин не меняется.

При поворотах дерева не изменяются его обходы.

Задание 2. Доказать, что при поворотах дерева его $pre - order$, $in - order$ и $post - order$ обходы не изменяются.

Splay-дерево. Splay-дерево — это обычное дерево, у которого есть операция *Splay*. Она заключается в том, чтобы с помощью повоторов дерева поднять какую-либо его вершину наверх.

Заметим, что это очень легко. Повороты, приведенные выше, поднимают вершину B выше на 1 уровень. Таким образом, применив несколько поворотов подряд, можно поднять вершину B на самый верх, сделав ее корнем.

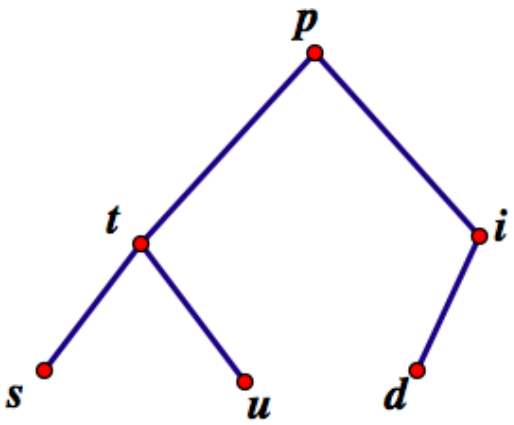
Rope. Веревка — это структура данных, представляющая из себя строку(слово из букв). Его можно печатать на экране, а так же разбивать слово на два слова, или сливать два слова в одно за $log(n)$, где n – длина слова.

Для сравнения представим себе строку как массив из символов. Тогда операция разбиения на две строки занимала бы у нас количество шагов, равное по длине строке. Нужно было бы стереть хвост строки и создать новую строку, записав в нее поэлементно этот хвост.

Rope делает эти операции всего лишь за $log(n)$, а не за n. Это существенное ускорение. Как такое сделать?

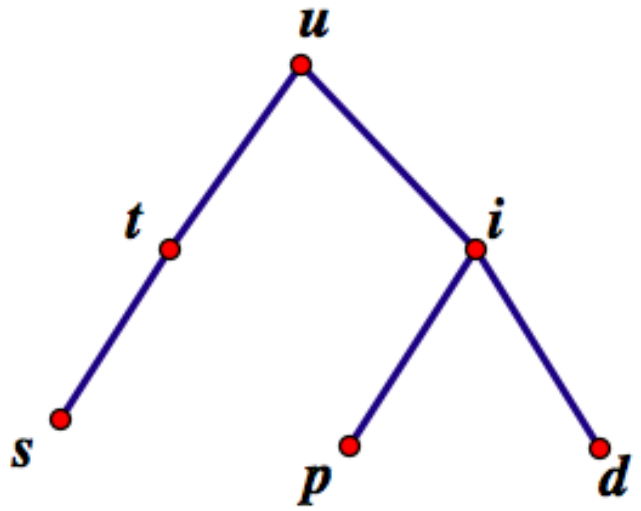
Rope представляет из себя *Splay*-дерево, в вершинах которого написаны буквы. Строка, которую мы будем хранить, получается в результате $in-order$ обхода дерева.

Например, если мы захотим хранить слово *stupid*, мы можем это делает с помощью дерева

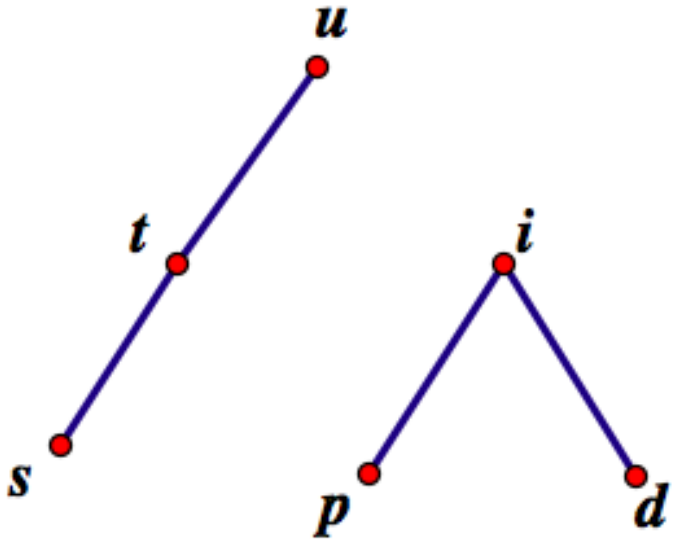


Чтобы разбить слово на два *stu* и *pid*, нужно сначала найти вершину, соответствующую третьей букве u(для этого будем хранить в каждой вершине количество вершин в ее левом и правом поддереве; смотрим на вершину p, ее левое поддерево состоит из ≥ 3 вершин, значит на поиски третьей вершины надо отправится в левое поддерево; у вершины t левое поддерево имеет < 3 вершин, значит за поиском надо уйти в правое поддерево, как раз таким образом мы и приходим к вершине u).

Далее, нужно операцией *Splay* отправить вершину u наверх, получив такое дерево:



Остается очевидным образом разбить его на два дерева:



Первое дерево соответствует слову stu, второе — слову pid, как и требовалось.

Слить два слова можно также просто:

Пусть имеются два слова stu и pid. Рассмотрим дерево первого и сделаем так, чтобы его последняя буква u была бы вершиной. Поскольку она является последней в $in-order$ обходе и вершиной дерева, то у нее нет правого поддерева. Значит, можно приделать второе дерево в качестве ее правого поддерева, получив требуемое.

Задание 3. Имеется некоторая строчка и запросы вида $begin, end, shift$ — три числа

Запрос применяется к строчке следующим образом: часть строки между $begin$ и end циклически сдвигается на $shift$.

Нужно написать программу, принимающую на вход строку и все запросы, применяющую все запросы к данной строке и печатающую на экран получившуюся в результате строку.