Персистентные структуры( PDS).

1. Что такое персистентность

***///***

Рассмотрим те структуры, которые мы выбрали для глубокого исследования. Наш выбор не был случаен - он опирался на широту использования и практичность объектов основного исследования.

ПДО = ДО + частичная персистентность -> обновление ветви

ДО = структура данных, поддерживающая для заданной операции вычислять значение на отрезке в массиве с динамически изменяющимися значениями за логарифмическое время, затрачивающую O(n) памяти. Операция должна быть такой, что если известно ее значение на интервалах [L, M] и [M + 1, R], то возможно за константу вычислить значение на отрезке [L, R].

Частичная персистентность -> обновление ветви = ***///***

ПСтек( Система контроля версий) = стек + полная персистентность -> двоичные подъемы

Стек = структура данных, поддерживающая добавление объектов в стек и удаление элементов из стека за O(1). По определению стек позволяет иметь доступ только к последнему элементу за константу, но в сегодня мы покажем, что ухудшив асимптотику операции добавления до логарифмической, мы сможем добиться за тот же самый O(logn) обращаться к произвольному элементу стека. Эта идея будет использовать механику двоичных подъемов и будет применима в различных олимпиадных задачах( к примеру с помощью нее можно решить IOI2012 задача S), но не нашла серьезного практического применения, поэтому мы упомянем о ней вскользь.

2) motivation problem и практическая ценность персистентного дерева отрезков

Нахождение К(К = количество различных объектов) на отрезке в массиве( поддерживаются добавления в конец массива) за O(logn) на запрос и добавлением за O(logn), затрачивая O(nlogn) памяти.

Применение этой задачи:

Имеется веб-ресурс с большой посещаемостью. Когда пользователь входит на сайт, его IP-адрес передается в нашу программу. Так же из системы приходят запросы найти количество пользователей( количество различных IP) за заданный отрезок времени.

Ответы на запросы могут быть использованы для изучения статистики посещаемости отдельных страниц, исследования вкусов пользователей и улучшения продажи продукции/ размещения рекламы в зависимости от типа ресурса.

Цель использования персистентных структур: добиться наилучшего баланса время/ память в сравнении со следующими вариантами решения:

3) Варианты решения без PDS:

!!!Ввести обозначения!!! ***///***

1- Решение «в лоб». O(n) память, O(1) на добавление элемента и O(n) на запрос.

Решение: Добавляем все элементы в динамический массив, каждый за амортизированную O(1). При поступлении запроса завести хэш-таблицу( множество) добавляя в него все элементы из запроса, ответом будет итоговый размер множества.

2- Динамическое программирование. O(n^2) память, O(n) на добавление элемента и O(1) на запрос.

Решение: При добавлении i-ого элемента в массив просчитаем K(L, i) для всех L за O(n) предыдущим способом с помощью хэш-множества, теперь для запроса l, r нужно будет лишь найти в массиве K(l, r).

3- SegTree + Treap. Дерево отрезков с сохранением всего подотрезка в виде декартового дерева. O(nlogn) память, O(log^2n) на добавление элемента и O(log^2n) на запрос.

Решение: Скажем что Next[i] = j, если a[i] = a[j] U j > i U j - минимальное возможное. Если такого j нет, Next[i] = oo. Если к нам пришел запрос K(l, r), то очевидно, что рассматривая для каждого числа только его самое правое вхождение, которое попадает в интервал [l, r], то: во-первых все K чисел будут рассмотрены и учтены, а во вторых для каждого рассматриваемого i - Next[i] > r. Таким образом задача сводится к нахождению количества чисел в массиве Next на интервале [l, r], которые превосходят r. Эта задача может быть решена за вышесказанную сложность с помощью дерева отрезков с сохранением всего подотрезка в виде декартового дерева. Мы реализовали для сравнения с основным объектом исследования этот алгоритм, но довольно очевидно, что алгоритм менее эффективный, в то же время более трудный для реализации и понимания.

4- O(n^2) память, O(n + logn) на добавление и O(logn) на запрос. (Дерево отрезков на сумму + итоговая идея). Этот метод использует такую же идею как и алгоритм с ПДО, но не использует персистентность. Мы покажем, что он из себя представляет вместе с итоговой идеей.

4) Итоговая идея. Для начала зашифруем наши IP-адреса так, чтобы в будущем работать только с числами [0, n). Можно сделать это разными способами, но самым простым( можно увидеть, что это не ухудшит нашу асимптотику) будет использование любого ассоциативного массива, работающего за O(logn), к примеру STL:map в C++, или более быстрых методов, которые мы так же рассмотрим( префиксное дерево «Trie», хэш-множество C++ unordered\_map).

Будем использовать персистентное дерево отрезков. Версия R будет рассматривать префикс массива [0, R]. ***///***

Заметим, что версия R будет отличаться от версии R - 1 на 1 изменение. ***//***/