Семинар к лекции 11.

#вшпи #аисд #структуры_данных

Автор конспекта: Гридчин Михаил

Примеры

$\Omega = \{0, 1\}$, $2^\Omega = \{\{\varnothing\}, \{0\}, \{1\}, \{0, 1\}\}$
Зададим $\xi(x) = x$.
Посчитаем $\mathbb{E}\xi(x)$:

$$\mathbb{E}\xi(x) = \sum_{i=0}^{1} i \cdot \mathcal{P}(i), \quad \mathcal{P}(i) = \frac{1}{|\Omega|}$$

$$\boxed{\frac{1}{2}}$$

Зададим $\xi_2(x) = e^x - 2x$.

$$\mathbb{E}\xi_2(x) = \sum_{k=0}^{1} (e^k - 2k)\mathcal{P}(k) = 1 \cdot \frac{1}{2} + (e - 2) \cdot \frac{1}{2} = \boxed{\frac{e-1}{2}}$$

Def. O^* - в среднем.

Задача 1

Hash table:

- $\text{insert}(x)$ - $O^*(1)$
- $\text{erase}(x)$ - $O^*(1)$
- $\text{find}(x)$ - $O^*(1)$
- get random - $O^*(1)$ - вернуть случайный элемент из hash table

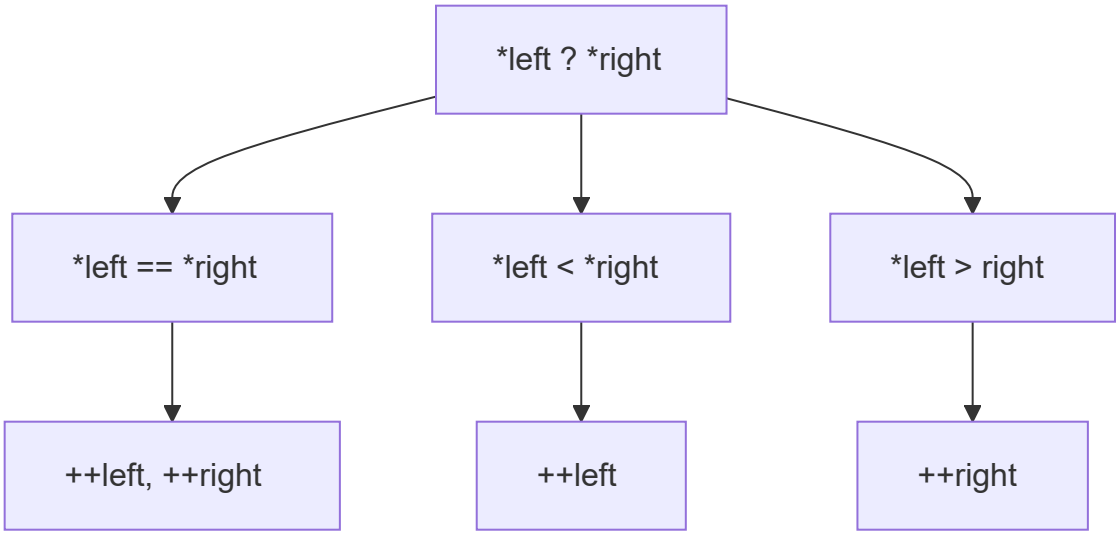
Решение: просто случайно выбирать bucket и внутри него выбирать случайный элемент нельзя - чем больше размер bucket, тем больше должна быть вероятность, что он будет выбран, чтобы был равновероятный выбор.
Давайте в дополнение к hash table заведём массив элементов, которые хранятся в hash table, который хранит указатели на элементы hash table в каком-то порядке. Пусть элементы hash table хранят итератор на элемент в динамическом массиве. Тогда удаление из динамического массива - это поменять местами с последним и удалить, поменять значение итератора для элемента, с которым поменяли. Добавление -

добавление в конец динамического массива за $O^*(1)$. get random - это выбор случайного элемента в динамическом массиве. Все остальные действия как в оригинальном hash table.

Задача 2

Дано: a, b - отсортированные массивы.
Найти: $a \cap b$.

Метод двух указателей.



Если a, b - то пройдёмся по a, добавим все элементы в multi hash set. Проходимся по массиву b. Если очередной элемент присутствует в multi hash set, то удаляем его из multihash set и выписываем в ответ, иначе переходим к следующему элементу.

Задача 3

Дано: a - массив, C - число.
Найти: $l, r : a_l + a_{l+1} + \ldots + a_r = C$, и при этом $(r - l) \to max$.

Для отсортированного массива умеем решать задачу через два указателя.

Решение: Пусть $\text{hash\_table}[i]$ - минимальный индекс ind:

$$\sum_{k=0}^{ind} a_k = i$$

Будем поддерживать префиксную сумму ps. Пусть сейчас мы находимся в позиции i: Добавим $\text{ps} = \text{ps} + a_i$. Если $\text{hash\_map}[ps - C] \neq \varnothing$, то обновляем:
$\text{ans} = \max(\text{ans}, i - \text{ hash\_map}[ps - C] + 1)$. Затем если $\text{hash\_table}[ps] = \varnothing$, то обновляем $\text{hash\_map}[ps] = i$.

Задача 4

LRU - last recently used - hash table на C последних элементах. Это бывает полезно, чтобы не выделять очень много памяти, а обращаться быстро к часто запрашиваемым элементам.

- $\text{insert}(x)$ - если больше C, удаляем самый давний.
- $\text{erase}(x)$ - $O^*(1)$
- $\text{find}(x)$ - $O^*(1)$

Решение: Вспомним *задачу 1*. Будем хранить не динамический массив, а двусвязный список. Очевидно, теперь вместо итераторов будут указатели. Добавление - добавление в конец и удаление (если размер больше C) из начала списка. Удаление - просто удаление из двусвязного списка по заранее известному указателю. Запрос поиска - это нахождение нужного элемента в списке и перенос элемента в конец (теперь он самый "новый"). Все операции внутри списка за $O(1)$, таким образом, решение работает за $O^*(1)$.

Задача 5

Дано: n точек (x, y) на плоскости, $x, y \in \mathbb{Z}$.
Найти: $\#(A, B, C) : \Delta ABC$ - равнобедренный - $O^*(n^2)$.

Решение: заметим, что если координаты целые, то равносторонних треугольников не будет. Заведём для каждой точки (x, y) выполнено:
$\text{hash\_map}[\text{dist}_k] = \#\text{points} : \forall \text{point} \in \text{points} \Rightarrow \text{dist}(\text{point}, (x, y)) = \text{dist}_k$. Теперь количество равнобедренных треугольников с фиксированной вершиной P - это

$$\sum_{k=0}^{|\text{hash\_map}|} \frac{\text{hash\_map}[\text{dist}_k] \cdot (\text{hash\_map}[\text{dist}_k] - 1)}{2}$$

Задача 6

Дано: n точек (x, y) на плоскости, $x, y \in \mathbb{Z}$.
Найти: $\max k : \exists i_1, \ldots, i_k : P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ коллинеарны - $O^*(n^2)$.

Решение: фиксируем каждую точку P_i и переносим систему координат в точку P_i. Теперь приведём все координаты точек из старой системы координат в новую. Заметим, что точки лежат на одной прямой с центром координат тогда и только тогда, когда тангенс угла у этих точек совпадает. Тангенс угла - это пара (up, down) - сокращённая дробь. Занесём все эти пары чисел в $\text{hash\_map}[(\text{up}, \text{down})] = \#\text{points}$.

Замечание: Как хешировать пару чисел? Предположим, числа 32-битные. Тогда возьмём число $a \cdot 2^{32} + b$. Заметим, что разные пары чисел переходят в разные. Получается, что

нужно хешировать 64-битные числа.

Задача 7

Реализуйте шаблонный hash table, в который подаётся тип значений и функтор хеширования этого типа

Решение:

```
#include <cstddef>
#include <iostream>
#include <vector>

// Hash - это функтор ( имеет operator() )
template <typename T, class Hash = std::hash<T>>
class HashTable {
 public:
    typename std::vector<T>::iterator Find(const T& object) {
       size_t position = PosHash(object);
       for (auto iter = table_[position].begin(); iter !=
table_[position].end();
            ++iter) {
          if (*iter == object) {
             return iter;
          }
           }
       return table_[position].end();
    }

    void Insert(const T& object) {
       size_t position = PosHash(object);
       auto iter = Find(object);
       if (iter != table_[position].end()) {
          return;
       }
       table_[position].insert(iter, object);
       ++size_;
    }

    void Erase(const T& object) {
       size_t position = PosHash(object);
       auto iter = Find(object);
       if (iter == table_[position].end()) {
          return;
       }
       table_[position].erase(iter);
```

```
        --size_;
    }

    size_t Size() { return size_; }

    bool Contains(const T& object) {
        size_t position = PosHash(object);
        return Find(object) != table_[position].end();
    }

    explicit HashTable(size_t size) { table_.resize(size); }

 private:
    // узнать по объекту, в какой bucket он отправится
    size_t PosHash(const T& object) { return Hash{}(object); }

    std::vector<std::vector<T>> table_;
    size_t size_ = 0;
};

int main() {
    HashTable<int> h(10);
    h.Insert(5);
    h.Insert(3);
    h.Insert(9);
    std::cout << h.Size() << '\n'; // 3
    h.Insert(4);
    h.Insert(8);
    std::cout << h.Size() << '\n'; // 5
    h.Insert(1);
    h.Insert(2);
    h.Insert(2);
    h.Insert(2);
    std::cout << h.Size() << '\n';        // 7
    std::cout << h.Contains(3) << '\n';   // 1
    std::cout << h.Contains(1) << '\n';   // 1
    std::cout << h.Contains(100) << '\n'; // 0
    std::cout << h.Contains(4) << '\n';   // 1
}
```