21. Open addressing, linear probing, quadratic probing, double hashing, cuckoo hashing, complexities

ОТКРЫТАЯ АДРЕСАЦИЯ

В отличии от закрытой адресации, **открытая адресация** не позволяет ставить различные элементы в одной и той же ячейке.

Этого можно достичь многими способами.

1. Самый тривиальный.
   При коллизии сделать перехеширование. Что плохого в этом? Сложность O(∞) (если хеш функция плохая).

Linear probing, quadratic probing.

Алгоритм поиска элемента e.

1. К индексу приравниваем index=m(h(e))
2. Если индекс указывает на свободную не занятую ячейку, завершить алгоритм: элемент не найден.
3. Иначе сравниваем элемент e с элементом в index. Если равны — завершить алгоритм: элемент найден. Иначе index=m(index+1) (m-map функция). Вернуться к шагу 2.

Для удаления элемента каждой ячейке присваиваются **три состояния**:

1. Свободна. (элемент в эту ячейку никогда не добавлялся).
2. Занята. (в данной ячейке присутствует элемент)
3. Зарезервирована. (элемент раньше был, а сейчас нет.

Удаление:
Если элемент уже найден, удалить элемент и присвоить к состоянию ячейки Зарезервирована.

Rehashing происходит в зависимости от $\alpha=n/m$, также как и при закрытой адресации.
но ОБЯЗАТЕЛЬНО $\alpha<1$. (чтобы избежать бесконечного цикла)

Оценка сложностей.

Поиск элемента.	в худшем случае: O(n). Вероятностно амортизировано $O(1/(1-\alpha))$.
Добавление элемента.	в худшем случае: O(n). Вероятностно амортизировано $O(1/(1-\alpha))$.
Удаление элемента. (если элемент найден)	O(1).

Алгоритм добавления элемента с хешом h в таблицу

1. index = m(h), например m(h)= h%m
2. если индекс не занят, добавляем элемент и завершаем алгоритм.
3. в ином случае index = m(index+1), например (index+1)%m и возвращаемся к шагу 2.

ВОПРОС: что лучше линейная порция или закрытая адресация, те что лучше $O(1/(1-\alpha))$ или $O(\alpha)$?

Когда $1/(1-\alpha) < \alpha \rightarrow 1 < \alpha - \alpha^2 \Rightarrow \alpha^2 - \alpha + 1 < 0 \Rightarrow D = 1-4 = 3$ **=> ОТВЕТ:** никогда

Подытожым: поиск и добавление элемента с хешом h происходит путем пробации $m(h+1)$, где $i = 0,1,...$

При квадратной пробации $m(h+i^2)$. Поиск и добавление также амортизировано $O(1/(1-\alpha))$.

Пример:

Условия:

- Размер хеш-таблицы (m) = 7.
- Хеш-функция:h(k)=k%m
- Вставляемые элементы: 10,20,15,17,24.

1. Линейная пробация

При линейной пробации, если ячейка занята, проверяем следующую ячейку (index+1)%m.

Шаги:

- **Вставка** 10
  10:
  - h(10)=10mod7=3.
  - Ячейка 3 пуста. Вставляем 10 в ячейку 3.
- **Вставка** 20
  20:
  - h(20)=20mod7=6.
  - Ячейка 6 пуста. Вставляем 20 в ячейку 6.
- **Вставка** 15
  15:
  - h(15)=15%7=1.
  - Ячейка 1 пуста. Вставляем 15 в ячейку 1.
- **Вставка** 17
  17:
  - h(17)=17%7=3.
  - Ячейка 3 уже занята (10).
  - Пробуем следующую ячейку: (3+1)%7=4.
  - Ячейка 4 пуста. Вставляем 17 в ячейку 4.
- **Вставка** 24
  24:
  - h(24)=24mod7=3, занята (10).
  - Пробуем следующую ячейку: (3+1)%7=4, занята (17).
  - Пробуем следующую ячейку: (4+1)%7=5.
  - Ячейка 5 пуста. Вставляем 24 в ячейку 5.

Результат таблицы (линейная пробация):

Индекс	0	1	2	3	4	5	6
Значение		15		10	17	24	20

2. Квадратичная пробация

При квадратичной пробации, если ячейка занята, используем формулу: $index=(h(k)+i^2)\%m$

где i — номер попытки (1, 2, 3...).

Шаги:

- **Вставка** 10
  10:
  - h(10)=10%7=3.
  - Ячейка 3 пуста. Вставляем 10 в ячейку 3.
- **Вставка** 20
  20:
  - h(20)=20%7=6.
  - Ячейка 6 пуста. Вставляем 20 в ячейку 6.
- **Вставка** 15
  15:
  - h(15)=15%7=1.
  - Ячейка 1 пуста. Вставляем 15 в ячейку 1.
- **Вставка** 17
  17:
  - h(17)=17%7=3.
  - Ячейка 3 занята (10).
  - Пробуем следующую ячейку: (3+12)%7=4.
  - Ячейка 4 пуста. Вставляем 17 в ячейку 4.
- **Вставка** 24
  24:
  - h(24)=24%7=3.
  - Ячейка 3 занята (10)
  - Пробуем следующую ячейку: $(3+1^2)\%7=4$ (занята 17).
  - Пробуем следующую ячейку: $(3+2^2)\%7=0$.
  - Ячейка 0 пуста. Вставляем 24 в ячейку 0.

Результат таблицы (квадратичная пробация):

Индекс	0	1	2	3	4	5	6
Значение	24	15		10	17		20

Итог:

- **Линейная пробация** распределяет элементы последовательно, что может привести к кластеризации.
- **Квадратичная пробация** лучше распределяет элементы, уменьшая вероятность кластеризации, но может возникнуть проблема, если таблица заполнена слишком сильно ($\alpha>0.5$).

Если m-простое число то пробация $(h(x) + i^2)$ %m , i = 0,1… не создает цикла меньше чем [m/2] .
Если мы хотим, чтобы икс не создавался вовсе нужно, чтобы $\alpha<0.5$.
Полиномиальное пробирование: $m(h(x) + c1\cdot i + c2\cdot i^2+..+ck\cdot i^k)$

Double hashing, cuckoo hashing.

·Double Hashing

Пусть задана Хеш функция h (x). Обозначим: $m_1(h) = h\%m$, $m_2(h) = h / m$.

Принцип пробации: $(m_1(h) + m_2(h)\cdot i)\%m$, i = 0,1…

Могут возникнуть проблемы если $m_2(h)$ и m не взаимно простые.

Допустим m = 8, h = 70 **=>** $m_1(h) = h\%m = 6$ $m_2(h) = h / m = 8$ **=>** $6+8\cdot i$, i =0,1 .., m-1.

Те постоянна получаем 6.

Решается эта проблема выбором $m_2(h)$ и m взаимно простыми.

·$m = 2^r$, $m_1(h) = h\%m$, $m_2(h) = (h / m)\cdot 2 + 1$.

Можно было взять наоборот **$m_1(h) = (h / m)\cdot 2 + 1$**, **$m_2(h) = h\%m$.**

·Cuckoo Hashing

- **Описание**: Используются две (или более) map-функции и две хеш-таблицы. Если при вставке возникает коллизия, элемент, уже находящийся в ячейке, "выталкивается" в другую таблицу, используя вторую map-функции.
- **Алгоритм**:
  1. Вычисляется первая map-функции $m_1(k)$
  2. Если ячейка занята, вытесняем элемент и пробуем вставить его в другое место, используя $m_2(k)$.
  3. Процесс повторяется, пока элемент не будет вставлен или не будет обнаружен цикл (тогда требуется перестройка таблицы).
- **Преимущества**:
  1. Постоянное время для поиска O(1).
  2. Подходит для приложений, где поиск имеет высокий приоритет.
- **Недостатки**:
  1. Сложность реализации.
  2. Возможность зацикливания требует перестройки таблицы.
- **Сложности**:
  1. Поиск и удаление O(1) (всегда).
  2. Вставка: O(1) (в среднем), перестройка может занять O(n).

Сравнение сложностей

Метод	Поиск	Вставка	Удаление
Линейное пробирование	в худшем случае: O(n). Вероятностно амортизировано O(1/(1-α)).	в худшем случае: O(n). Вероятностно амортизировано O(1/(1-α)).	O(1), если эл. найден
Квадратичное пробирование	в худшем случае: O(n). Вероятностно амортизировано O(1/(1-α)).	в худшем случае: O(n). Вероятностно амортизировано O(1/(1-α)).	O(1), если эл. найден
Двойное хеширование	в худшем случае: O(n) в среднем O(1).	в худшем случае: O(n) в среднем O(1).	O(1), если эл. найден
Кукушечное хеширование	O(1)	O(1), в худшем случае: O(∞).	O(1)

Итог

- **Линейное пробирование**: Простое, но страдает от кластеризации.
- **Квадратичное пробирование**: Уменьшает кластеризацию, но сложнее в реализации.
- **Двойное хеширование**: Более эффективное распределение, но требует двух хеш-функций.
- **Кукушечное хеширование**: Быстрое O(1) для поиска, но может потребовать перестройки таблицы.