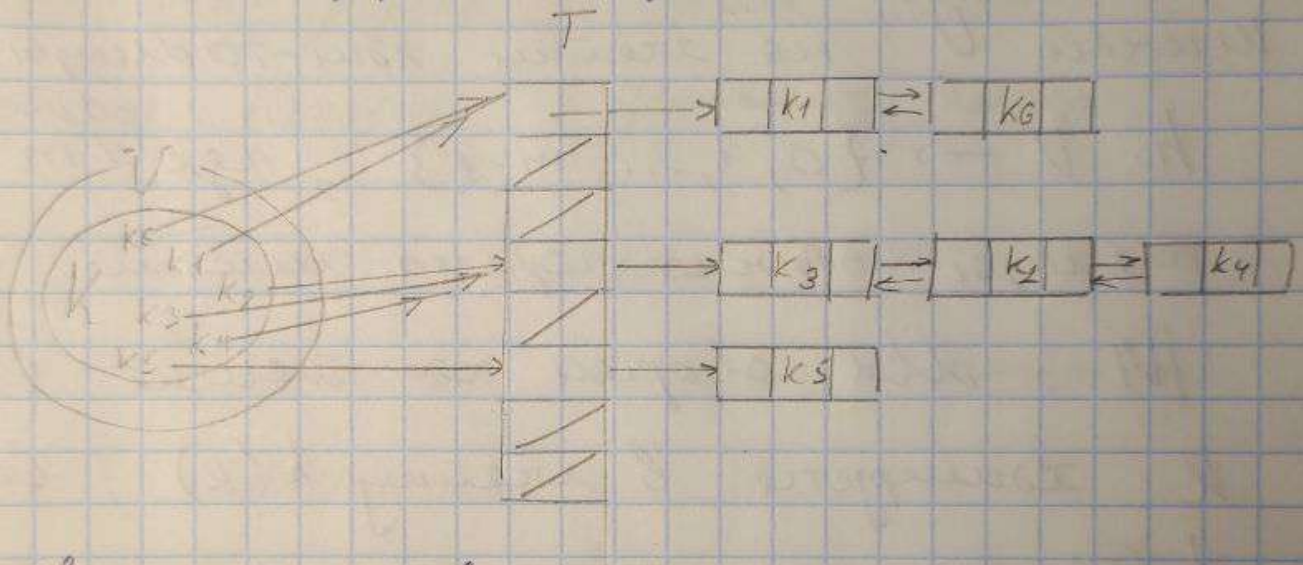
хэшироваться в одну и ту же ячейку, в связанный список. Ячейка j содержит ук-тель на заголовок списка всех элементов, хэш-значение ключа которых равно j. Если таких элементов нет, то ячейка содержит значение NIL.



В двусв. списке быстрее удаление $O(1)$, но доп. память!

Chained-Hash-Insert (T, x)

Вставка x в заголовок списка

T[h(x.key)]

Chained-Hash-Search (T, k)

T[h(k)]

Chain_Hash-Delete (T, x)

T[h(x.key)]

] У нас есть хеш-таблица T с m ячейками, в которых хранятся n элементов. Определим коэффициент заполнения α таблицы T как:

$\alpha = \frac{n}{m}$, т.е. ср. кол-во элементов, хранящихся в одной ячейке.

Худший случай: все n ключей хэшированы в одну ячейку, создав список длиной n, т.о., время поиска в худшем случае - $O(n)$ + время вычисления хэш-функции.

] Все элементы хэшируются по ячейкам равномерно и независимо. Назовем такое предположение простым равномерным хэшированием. Тогда время неудачного поиска в ср. случае составляет $O(1+\alpha)$.

Д-во: ключ k может быть помещен с равной вероятностью в любую из m ячеек. Мат. ожидание времени неуд. поиска ключа k равно времени поиска

№36 Понятие амортизационного анализа на примере операции Multipop ()

В ходе амортизационного анализа время, необходимое для выполнения посл-ти операций над структурой данных, усредняется по всем выполненным операциям. Этот анализ можно использовать, например, чтобы показать, что, даже если одна из операций посл-ти является дорогостоящей,

ная стоимость одной операции не превышает 3. Так как 3-эта конкатенация, то амортизированная стоимость = $O(l)$

до конца списка $T[h(k)]$, отличная от
~~длина~~ длина которой - $E[h_{h(k)}] = \alpha$. Т.о., при
неудачном поиске мат. ожидание равно
α, а общее время поиска, включая
время вычисл. хэш-функции $h(k)$, равно $O(1+\alpha)$.

Усп. поиск тоже $O(1+\alpha)$

№40. Добавление, удаление, поиск элементов в таблице.

Прямая адресация.

Прямая адресация представляет собой простую технологию, которая хорошо работает

при усреднении по всей посл-сти средняя стоимость операций будет небольшой. При выполнении амортизационного анализа гарантируется средняя производительность операций в наихудшем случае.

Рассмотрим на примере стека:

Push (S, x) - O(1)

Pop (S) - O(1). Вызов Pop с пустым стеком генерирует ошибку.

Теперь добавим к стеку операцию Multipop (S, k), снимающую k объектов с вершины стека S.

Пусть s - кол-во объектов в стеке.

Лучший случай - O(min(s, k))

Худший случай: ~~O(s)~~ O(n), n - ~~s-n~~ вместимость стека.

Пусть нужно выполнить n операций со стеком.

n_1 - кол-во Push

n_2 - кол-во Pop

В некоторых случаях элементы дин. мн-ва могут храниться непосредственно в таблице с прямой адресацией, что приводит к значительным затратам.

Дин. массив - массив, который может изменять свой размер во время выполнения программы. Находится в динамич. памяти.

№41. Понятие хэш-таблицы. Разрешение коллизий с помощью цепочек.

Хэш-таблица представляет собой эффективную структуру данных для реализации словарей. Хотя на поиск элемента в хэш-таблице может в наихудшем случае потребоваться столько же времени, сколько на связном списке, а именно - $O(n)$, на практике хэширование эффективно: ~~эффективно~~ среднее время поиска элемента в хеш-таблице составляет $O(1)$.

n_3 - Multipop

$n_1 \cdot Push + n_2 \cdot Pop + n_3 \cdot Multipop$

Число $Pop \leq n_1$ (нельзя удалить больше, чем добавить)

$n_1 \leq n$

$\Rightarrow O(n)$ - суммарное время

$\frac{O(n)}{n} = O(1)$ - средняя стоимость операции

$\Rightarrow$ все три операции стека характеризуются амортизированной стоимостью, равной $O(1)$

№37 Динамическая таблица (вектор). Амортизированная сложность решения.

Динамич. таблица - это структура данных которая может менять свой размер во время выполнения программы.

Предположим, что место для хранения таблицы выделяется в виде массива ячеек. Таблица заполнена, если заполнены все её ячейки. Если нужно вставить новый элемент, то мы выделим ей

ря небольших совокупностей ключей.

Предполагается, что требуется динамическое мн-во, каждый элемент которого имеет ключ из совокупности $U = \{0, 1, \ldots, m-1\}$, где m не слишком велико. Кроме того, предполагается, что никакие два элемента не имеют одинаковых ключей.

Для представления динамич. мн-ва мы используем массив, или таблицу с прямой адресацией, который обозначим как $T[0 \ldots m-1]$, каждая ячейка которого соответствует ключу из совокупности ключей U.

Реализация элементарных операций тривиальна.

```
Search(T, k)                    Время - O(1)
    return T[k]
Insert(T, x)
    T[x.key] = x
Delete(T, x)
    T[x.key] = NIL
```

В случае прямой адресации элемент с ключом k хранится в ячейке k. При хешировании этот элемент хранится в ячейке h(k), т.е. мы используем хеш-функцию h для вычисления ячейки для данного ключа k. Ф-я h отображает совокупность ключей U на ячейки хеш-таблицы T[0..m-1]:

$h: U \to \{0, 1, \ldots, m-1\}$, где m (размер) хеш-таблицы обычно гораздо меньше значения $|U|$. Мы говорим, что элемент с ключом k хешируется в ячейку h(k); вычисление h(k) наз-ся значением ключа k.

Цель хеш-функции состоит в том, чтобы уменьшить рабочий диапазон индексов (размера) массива, и вместо $|U|$-значений мы можем обойтись массивом всего лишь размера m.

При разрешении коллизий (два ключа могут записаться в одну ячейку) с помощью цепочек мы помещаем все элементы,