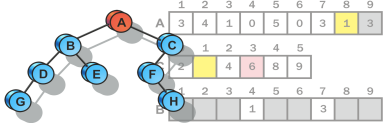
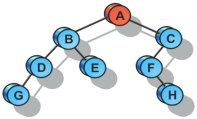
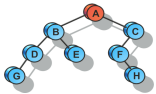


Хеширование



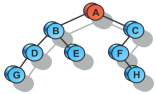
Хеш-таблицы с закрытой адресацией



Описание требуемой структуры данных

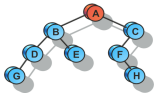
Множество

- Структура данных, поддерживающая операции:
  - Добавление элемента
  - Удаление элемента по его значению
  - Проверка на принадлежность элемента множеству



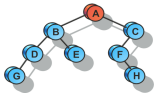
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$



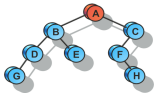
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$



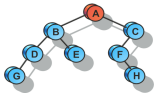
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$
- Ключи различны и принимают значения $0, 1, \ldots, m-1$



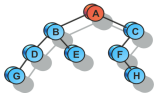
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$
- Ключи различны и принимают значения $0, 1, \ldots, m-1$
- Заведем массив $T[0 \ldots m-1]$



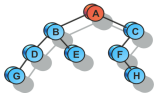
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$
- Ключи различны и принимают значения $0, 1, \ldots, m-1$
- Заведем массив $T[0 \ldots m-1]$
  - $T[i]$ — указатель на элемент с ключом i
  - Если элемента x нет в множестве, то $T[k[x]] = NIL$



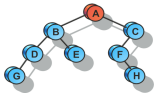
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$
- Ключи различны и принимают значения $0, 1, \ldots, m-1$
- Заведем массив $T[0 \ldots m-1]$
  - $T[i]$ — указатель на элемент с ключом i
  - Если элемента x нет в множестве, то $T[k[x]] = NIL$
- Поиск:
  - Direct_Address_Search(T, k):
    - return T[k]



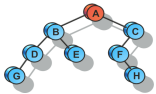
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$
- Ключи различны и принимают значения $0, 1, \ldots, m-1$
- Заведем массив $T[0 \ldots m-1]$
  - $T[i]$ — указатель на элемент с ключом i
  - Если элемента x нет в множестве, то $T[k[x]] = NIL$
- Поиск:
  - Direct_Address_Search(T, k):
    - return T[k]
- Вставка:
  - Direct_Address_Insert(T, x):
    - $T[k[x]] \leftarrow x$



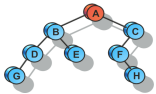
Прямая адресация

- Хранимые элементы U — небольшое множество, $|U| \leqslant m$
- Для каждого $x \in U$ определен ключ: $k[x]$
- Ключи различны и принимают значения $0, 1, \ldots, m-1$
- Заведем массив $T[0 \ldots m-1]$
  - $T[i]$ — указатель на элемент с ключом i
  - Если элемента x нет в множестве, то $T[k[x]] = NIL$
- Поиск:
  - Direct_Address_Search(T, k):
    - return T[k]
- Вставка:
  - Direct_Address_Insert(T, x):
    - $T[k[x]] \leftarrow x$
- Удаление:
  - Direct_Address_Delete(T, x):
    - $T[k[x]] \leftarrow NIL$



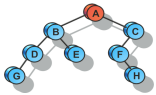
Проблемы прямой адресации

- Множество *U* обычно большое
  - Множество ключей нужно делать сильно меньше, чем *U*
    - По принципу Дирихле у нескольких элементов ключи совпадут
- Нужно знать метод вычисления ключа



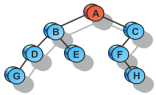
Хеширование

- Так как, теперь ключи не различны для разных элементов, назовем k — ***хеш-функцией***
  - $k[x]$ для простоты будет называться: хеш элемента x
- Случай, когда хеши двух элементов совпали, будем называть ***коллизией***



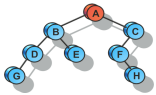
Разрешение коллизий с помощью цепочек

- Заведем массив списков $T[0 \dots m-1]$



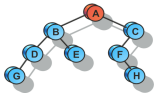
Разрешение коллизий с помощью цепочек

- Заведем массив списков $T[0 \dots m-1]$
- В списке $T[i]$ будут храниться элементы, у которых ключ равен i



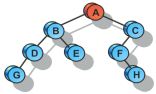
Разрешение коллизий с помощью цепочек

- Заведем массив списков $T[0 \dots m-1]$
- В списке $T[i]$ будут храниться элементы, у которых ключ равен i
- Поиск:
  - Chained_Hash_Search(T, x):
    - найти элемент x в списке $T[k[x]]$
  - Время работы в худшем случае: $O(L)$, где L — длина списка



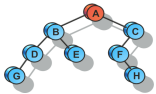
Разрешение коллизий с помощью цепочек

- Заведем массив списков $T[0 \dots m-1]$
- В списке $T[i]$ будут храниться элементы, у которых ключ равен i
- Поиск:
  - Chained_Hash_Search(T, x):
    - найти элемент x в списке $T[k[x]]$
  - Время работы в худшем случае: $O(L)$, где L — длина списка
- Вставка:
  - Chained_Hash_Insert(T, x):
    - добавить x в голову списка $T[k[x]]$
  - Время работы в худшем случае: $O(1)$



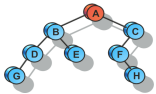
Разрешение коллизий с помощью цепочек

- Заведем массив списков $T[0 \dots m-1]$
- В списке $T[i]$ будут храниться элементы, у которых ключ равен i
- Поиск:
  - Chained_Hash_Search(T, x):
    - найти элемент x в списке $T[k[x]]$
  - Время работы в худшем случае: $O(L)$, где L — длина списка
- Вставка:
  - Chained_Hash_Insert(T, x):
    - добавить x в голову списка $T[k[x]]$
  - Время работы в худшем случае: $O(1)$
- Удаление:
  - Chained_Hash_Delete(T, x):
    - удалить x из списка $T[k[x]]$
  - Время работы в худшем случае: $O(L)$, где L — длина списка



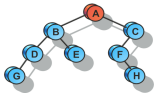
Анализ хеширования с цепочками

- Пусть в множестве лежит *n* элементов



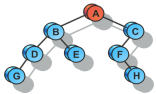
Анализ хеширования с цепочками

- Пусть в множестве лежит n элементов
- Число $\alpha = \frac{n}{m}$ назовем коэффициентом заполнения хеш-таблицы



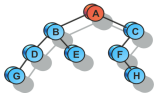
Анализ хеширования с цепочками

- Пусть в множестве лежит *n* элементов
- Число $\alpha = \frac{n}{m}$ назовем коэффициентом заполнения хеш-таблицы
- Пусть хеш-функция распределяет элементы *U* равномерно по таблице



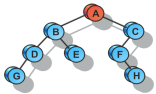
Анализ хеширования с цепочками

- Пусть в множестве лежит *n* элементов
- Число $\alpha = \frac{n}{m}$ назовем коэффициентом заполнения хеш-таблицы
- Пусть хеш-функция распределяет элементы *U* равномерно по таблице
- Тогда поиск и удаление элемента из хеш-таблицы будут выполняться за время $O(1 + \alpha)$ в среднем



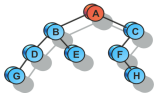
Анализ хеширования с цепочками

- Пусть в множестве лежит *n* элементов
- Число $\alpha = \frac{n}{m}$ назовем коэффициентом заполнения хеш-таблицы
- Пусть хеш-функция распределяет элементы *U* равномерно по таблице
- Тогда поиск и удаление элемента из хеш-таблицы будут выполняться за время $O(1 + \alpha)$ в среднем
  - Так как математическое ожидание *L* равно числу α



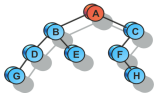
Анализ хеширования с цепочками

- Пусть в множестве лежит *n* элементов
- Число $\alpha = \frac{n}{m}$ назовем коэффициентом заполнения хеш-таблицы
- Пусть хеш-функция распределяет элементы *U* равномерно по таблице
- Тогда поиск и удаление элемента из хеш-таблицы будут выполняться за время $O(1 + \alpha)$ в среднем
  - Так как математическое ожидание *L* равно числу α
- Если выбрать *m*, число ячеек в таблице, как число порядка *n*, число добавленных элементов в множество, то α — константа



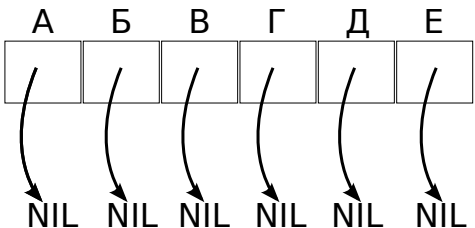
Анализ хеширования с цепочками

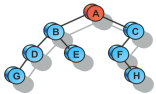
- Пусть в множестве лежит *n* элементов
- Число $\alpha = \frac{n}{m}$ назовем коэффициентом заполнения хеш-таблицы
- Пусть хеш-функция распределяет элементы *U* равномерно по таблице
- Тогда поиск и удаление элемента из хеш-таблицы будут выполняться за время $O(1 + \alpha)$ в среднем
  - Так как математическое ожидание *L* равно числу α
- Если выбрать *m*, число ячеек в таблице, как число порядка *n*, число добавленных элементов в множество, то α — константа
  - Операции вставки, удаления и поиска будут работать за $O(1)$



Пример

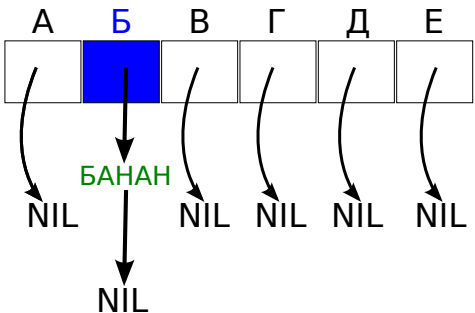
- Хеш-функцией будет первая буква слова

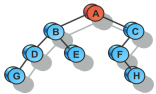




Пример

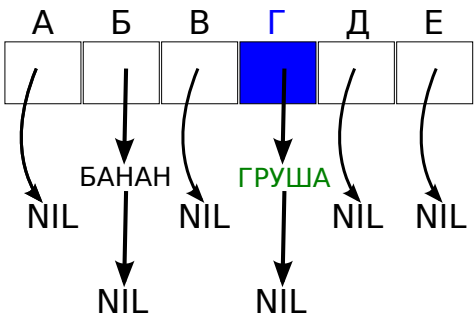
- Добавим слово БАНАН

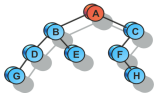




Пример

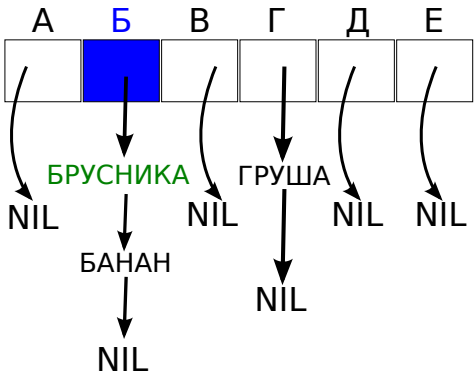
- Добавим слово ГРУША

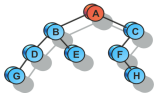




Пример

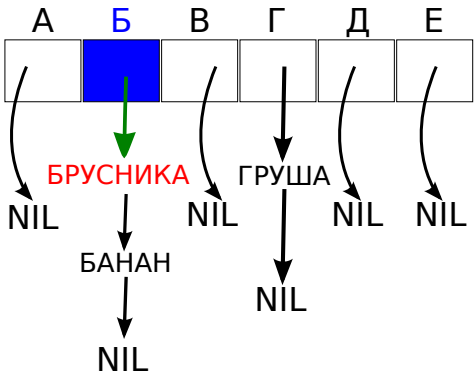
- Добавим слово БРУСНИКА

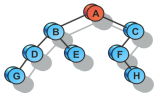




Пример

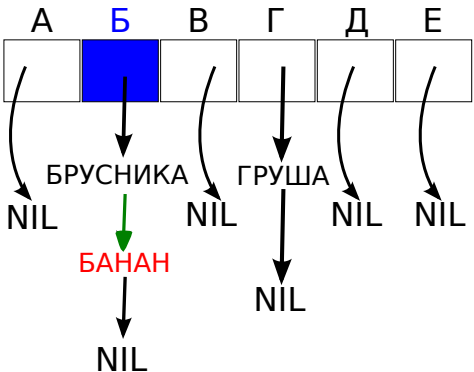
- Поиск слова БУДКА

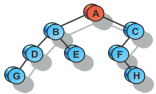




Пример

- Поиск слова БУДКА





Пример

- Поиск слова БУДКА