Хеш-таблицы

**Поиск**

Достоинства отсортированных массивов:

* быстрый поиск в ОП (),
* возможность поиска в диапазоне, на  или , простота получения порядковых статистик.

Недостатки отсортированных массивов:

* неприспособленность к динамическим изменениям,
* поиск, недостаточно быстрый для ВУ.

=>

Во многих случаях необходимы динамические структуры данных для быстрого поиска информационных объектов по ключам. Как правило, такие структуры хранят ключевые значения и ссылки (указатели) на соответствующие объекты (реже – сами объекты).

**Хеширование**

Основная идея – получать адрес искомого объекта на основе функционального преобразования его ключевого значения:

по ключу  вычисляется значение , определяющее номер записи в некоторой таблице объектов (ссылок).

Требования к функции :

1.  => 

2. простота вычисления (быстрое вычисление адреса)

Для сохранения  объектов\ссылок необходимо выделить таблицу с  записями. Если диапазон возможных значений ключа , то, как правило, .

=>

в общем случае возможно образование конфликтов (коллизий): , но .

Общий подход:

* не исключать коллизии, а предусмотреть их обработку,
* выбирать  таким образом, чтобы она по возможности равномерно распределяла  потенциальных значений ключа по  адресам.

 - хеш-функция (hash=смесь).

Хеш-таблица – информационная структура – таблица записей, номера\адреса которых вычисляются на основе .

Пример хорошей функции:, где  - простое (при таком  значения  достаточно равномерно распределяются на , т.е. хеш-таблица используется эффективно).

Использование хеш-функции в криптографии.**Обработка коллизий – метод цепочек**

**Идея**: объединять элементы, образующие коллизии, в списки.

Хеш-таблица представляет собой массив из  списков. Номер списка, в который включается элемент, определяется значением хеш-функции.

Пусть, например, , тогда элементы  и  попадут в список с номером *i*.











Выбор . Добавление, поиск и удаление элементов. Успешный и безуспешный поиск.

Равномерное размещение ключей по  спискам:

трудоемкость построения таблицы , поиска .

Наихудший случай –  списков (коллизий):

трудоемкость построения таблицы , поиска .

**Обработка коллизий – метод открытой адресации**

**Идея**:

* хеш-таблица длины  содержит только ключи,
* элементы, образующие коллизии, объединяются в списки,
* если список образуют элементы, для которых , то они займут **незанятые** ячейки таблицы с номерами , , где , а остальные шаги  не хранятся, а вычисляются.

Пример: в таблицу последовательно включаются  (все значения разные), , 

**Поиск элемента со значением  в хеш-таблице A** (возвращает номер элемента в таблице или -1):

























**k = i = ord(p) % q;** // **k = i = hash\_func(p);**

**j = 0;** // j – номер шага

**while (A[k] != p && A[k] != )** // не **p** и не пустой

**{**

**j++; s = текущий\_шаг(j);**

**k = (i + s) % q;**

**}**

**if (A[k] == p) return k;** // поиск успешный

**else return -1;** // поиск безуспешный

Добавление и удаление элементов

Методы вычисления текущего шага  должны исключать зацикливание при поиске элементов.

**Вычисление шага – линейные пробы**:

* выбирается ,  и  взаимно простые,
* шаги , , а соответствующие адреса ячеек таблицы .

Недостатки, приводящие к росту трудоемкости поиска:

* первичное скучивание – группировка элементов одного списка возле начального элемента (особенно при ),
* вторичное скучивание – группировка элементов разных списков.

Желательно, чтобы размещение элементов в таблице было по возможности равномерным и выглядело, как случайное.

**Вычисление шага – квадратичные пробы**:

, соответствующие адреса .

Поиск приведет к повторному выбору элемента на шаге , если для  и некоторого предыдущего шага  выполнится

.

Тогда , т.е. .

Это может выполниться только при числе шагов (длине списка) .

**Вычисление шага – двойное хеширование**:

**Идея** заключается в том, что при вычислении шага используется вторая хеш-функция, аргументом которой будет не само поисковое значение *p*, а значение хеш-функции от него *f*(*p*):



В один список входят элементы с разными ключами , но одинаковыми  (и, следовательно, фиксированным шагом), но для разных списков будут использоваться различные шаги. Это позволяет более равномерно распределить элементы в хеш-таблице.

В качестве второй хеш-функции можно взять:

*h*( *f* ( *p*)) = *f*( *p*) mod *r* + *s*, где , *s* > 0, *r* и *s* - константы.

Пусть в таблице хранится  элементов, и  – это заполненность таблицы. Рассмотрим асимптотический случай .

Будем полагать, что при поиске на каждом шаге события «элемент таблицы занят» и «элемент таблицы свободен» независимы, и их вероятности равны, соответственно,  и .

Вероятности событий «получение свободного элемента за…»:

1 шаг – ,

2 шага – ,

 шаг – .

В силу независимости данных событий:

 (условие нормировки).

**Средняя длина безуспешного поиска** (или трудоемкость размещения нового элемента) при заполненности :



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0.1 | 0.25 | 0.5 | 0.75 | 0.9 | 0.99 |
|  |  |  | 2 | 4 | 10 | 100 |

Пусть элементы таблицы не удаляются. Тогда при успешном поиске элемента проводятся те же самые действия, что и при размещении данного элемента в таблице.

Трудоемкость успешного поиска элемента равна числу шагов при размещении данного элемента, а это число зависело от текущей заполненности таблицы.

Пусть  - текущая заполненность таблицы. При размещении  элементов  изменяется от  до .

**Средняя длина успешного поиска**:

 

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0.1 | 0.25 | 0.5 | 0.75 | 0.9 | 0.99 |
|  |  |  |  |  |  |  |

**Вывод**: задавать ; если при добавлении новых эл-тов (увеличении ) достигается указанная граница, следует выделить новую таблицу с  записями и последовательно разместить в ней все элементы из старой.

**Применение хеширования для данных на ВУ**

Для размещения  элементов (ключей) выделяется  блоков по  записей т.о., что .

Данные читаются и записываются блоками.

Блок  () содержит от 0 до  элементов , для которых , и еще 2 значения – текущее число элементов и  или номер блока в области переполнения, куда попадает ()-й элемент с таким же значением хеш-функции:

При увеличении  (и соответствующем уменьшении ) происходит более равномерное распределение записей по блокам.

4

6

2

m

5

1

0

q-1

Д.Кнут показал: при  и заполненности таблицы  среднее число чтений блоков составляет 1.04.

Недостатки хеширования:

* недостаточная приспособленность к динамическим изменениям (число элементов таблицы должно быть известно заранее, удаление элементов повышает трудоемкость поиска),
* высокая трудоемкость в наихудшем,
* поиск только по совпадению ключей.