ПОИСК С ПРИМЕНЕНИЕМ ХЕШ-ТАБЛИЦЫ

1.1 Хеширование и его применение в алгоритмах поиска

Этот алгоритм так же относиться к поиску в динамических таблицах.

Хеш-таблица - это механизм, позволяющий обеспечит доступ к записи в таблице за фиксированное время, т.е. O(1) в лучшем случае и O(n) в худшем случае.

Вопрос: как можно достичь сложность O(1) ?

Ответ: только, если на прямую обратиться к нужному элементу, как в массиве.

Рассмотрим пример использования ключей в качестве индекса массива.

Пример 1. Пусть требуется создать систему для управления персоналом предприятия, численность сотрудников в котором 100 человек. Хранить надо анкетные данные этих сотрудников. Все сотрудники имеют ИНН. Поэтому ключом к данным в хранилище может быть ИНН.

ИНН состоит из 10 цифр. Для прямого доступа к данным по ключу можно создать массив из такого количества элементов, чтобы в качестве индекса массива можно было использовать ИНН. Это будет массив из 10^{10} элементов, а сотрудников всего 100, т.е. занятыми будут только 100 ячеек. Неэффективное представление данных в памяти.

Хороший вариант представления данных в рассматриваемом примере – это массив из 100 элементов. Но как обеспечить время доступа к элементу за время O(1)?

Хеширование предлагает создать отображение (установить соответствие), в рассматриваемой теме: множество ключей в множество целых чисел – множество индексов массива.

Отображение это функция, определенная на множестве элементов (области определения) одного типа и формирующая значение на множестве элементов (область значений) другого типа.

Т.е. отображение можно представить M(d)=r, где d – ‘значение из области определения M, а r значение из области значений.

Будем понимать под понятием *хеширование* - процесс преобразования ключа в индекс массива, называемого хеш – таблицей.

Хеш-таблица — это еще одна структура для организации доступа к данным в структурах. Она может хранить сами элементы данных, или ключи со ссылками на данные.

1.2 Хеш функция

Хеш функция – это алгоритм, осуществляющий процесс преобразования значения ключа в индекс хеш–таблицы.

Хеш–таблица – это массив определенного размера, в ячейках которого размещаются элементы данных. Место (индекс элемента в таблице) для размещения данных в таблице, вычисляет хеш-функция на основе значения ключа.

Пример 2. Заполнение хеш-таблицы записями с ключами из примера 1 и поиск значений в хеш таблице.

Пусть хеш-таблица имеет размер L=100 (по количеству сотрудников), хеш-функция h(key) должна вырабатывать индекс в пределах от 1 до 100, тогда алгоритм вычисления такого значения на основе ключа может быть совершенно простым: ***key % L.***

Пусть имеется список ИНН сотрудников:

1111112311, 1111112312, 1111112302.

Тогда хеш–функция h(key) для ключа 1111112311) вернет индекс – 11, а это значит, что запись о сотруднике с ИНН равным 1111112311 разместиться в хеш-таблице под индексом 11. Соответственно запись с ключом 1111112312 разместиться по индексу 12, а 1111112302 по индексу 2.

После того как хеш-таблица создана, можно выполнять поиск, удаление записей на основе заданных ключей.

На рисунке 10.9.1.1 представлена хеш-таблица рассматриваемого примера.

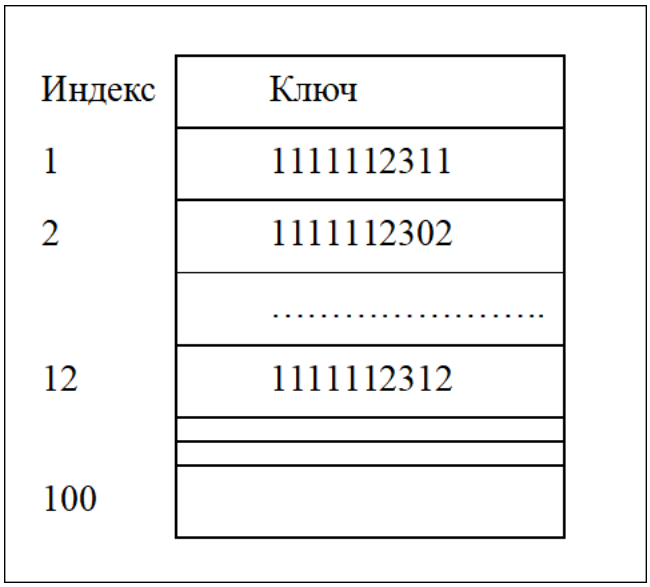

Рисунок 10.9.1.1. Пример хеш-таблицы с размещенными ключами

Например, мы хотим вывести сведения о сотруднике с ИНН равным 1111112312. Для этого ключ 1111112312 передадим той же хеш–функции, которая вычислит индекс равный 12, а следовательно, мы получаем доступ к значению с заданным ключом.

1.3 Принцип однородного хеширования

Для успешного хеширования очень важно, чтобы хеш-функция создавала такие индексы, что размещаемые в таблице данные были равномерно (однородно) распределены по таблице. Т.е. созданный индекс не зависел от индексов с которым хешированы другие элементы.

Для применения хеширования необходимо:

- чтобы элементы размещаемые в хеш-таблице имели уникальный ключ
- подобрать хеш-функцию, обеспечивающую принцип однородности.

1.4 Коллизия - проблема хеш-таблицы -

Коллизия – это ситуация, когда для двух разных ключей хеш-функция создает одинаковый индекс.

Поэтому процесс хеширования включает 2 этапа:

- Вызывается хеш-функция, которая, оперируя с ключом, возвращает индекс в хеш-таблице.
- Обработка возникающей коллизии.

Пример 3. Появление коллизий.

Создадим хеш-таблицу для рассмотренного ранее примера для сотрудников (100 человек) из L=100 элементов. При этом будем использовать хеш - функцию h(key)=key % L.

Так как в одной ячейке массива может храниться только данное с одним ключом, тогда появляется задача: каким – то образом запись с ключом, получившим коллизию разместить в хеш-таблице. Этот процесс получил название - устранение коллизии.

На рисунке 10.9.4.1 представлен пример списка ключей, для которых хеш-функция h(key)=key % L определяла индексы. Для ключей 114101, 111312, 111305 хеш-функция сформировала дубликаты индексов, т.е возникли коллизии для этих ключей.

Ключ	Индекс	
111001	1	
111205	5	
111312	12	
114101	1	коллизия
111312	12	коллизия
111313	13	
111305	5	коллизия

Рисунок 10.9.4.1. Коллизии для ключей

При возникновении коллизии надо в таблице хранить значение записи, ключ которой был хеширован с коллизией. При этом необходимо обеспечить поиск значения такого ключа со сложностью O(1). Процесс размещения значения записи с таким ключом в таблице получил название устранение коллизии.

Для разрешения проблемы коллизий используют два основных метода:

- *Цепное (сцепленное) хеширование* (формирование цепочек из элементов, хешированных с одним индексом)
- *Хеширование с открытым адресом* (таблица большого объема в элементы которой записываются данные)

Разрешение коллизии – связанно в некоторой степени с хеш-функцией т.е. какая часть ключа выбрана для хеширования. Но какую бы функцию бы не выбрали коллизии будут иметь место.

Для разрешения коллизий, которые возникают *при добавлении данных* в хеш-таблицу, применяются два метода организации хеш-таблиц:

- метод цепного хеширования;
- метод открытого адреса.

1.5 Метод цепного хеширования

Это один из способов разрешения коллизий, когда элементы, ключи которых получили один индекс, хранятся в одном однонаправленном списке, а вершина этого списка хранится в хеш-таблице. Тогда хеш-таблица – это массив указателей на вершины списков.

Суть метода.

При добавлении нового ключа (элемента с данными) в таблицу, хеш-функция определяет индекс элемента таблицы, в список которого должен быть добавлен ключ. Если добавлять записи в список в порядке возрастания (убывания) значения ключа, то можно повысить эффективность операции поиска.

При поиске записи с ключом в хеш-таблице: хеш-функция определяет индекс элемента таблицы, в списке которого должен находиться запись с заданным ключом, затем осуществляется поиск уже в списке. Если запись найдена формируется результат поиска, это может быть указатель на узел, или данные записи, хранящейся в узле.

При удалении записи с ключом: осуществляется поиск записи с заданным ключом и в результате удобно вернуть указатель на найденный узел и удалить узел из однонаправленного списка.

1.5.1.1 Структура хеш-таблицы

Структура хеш-таблицы при цепном хешировании представлена на рисунке 10.9.5.1.1. Для знакомства со структурой хеш-таблицы для организации цепного хеширования рассмотрим следующий пример.

Пример 4. Создание хеш – таблицы, использующей динамические списки для устранения коллизий.

Создадим хеш-таблицу для ключей из примера 3. При возникновении коллизии данные с ключом помещаются в начало списка, который в таблице связан с элементом, для которого хеш-функция создала индекс.

На рисунке представлена таблица с реализацией коллизий на основе цепного хеширования.

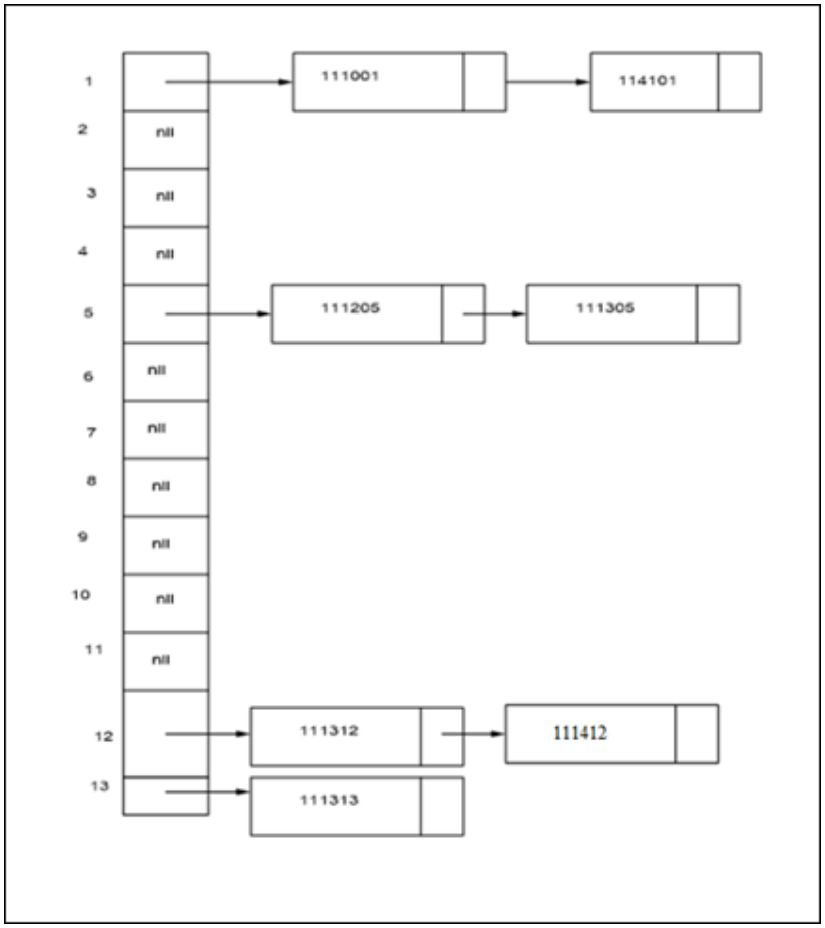

Рисунок 10.9.5.1.1. Организация хеш-таблицы при азрешение коллизий методом цепного хеширования

1.5.2 Задание для самостоятельной работы

Опишите пошагово алгоритм создания таблицы примера 4. Укажите ключи для которых возникла коллизия. Исходные данные: длина таблицы L=100, хеш-функция формирует индекс по алгоритму: key%L.

1.5.3 Проблемы таблицы цепного хеширования

1. Проблема однородного хеширования

Хеш функция должна удовлетворять принципу однородного хеширования. Т.е. желательно, чтобы списки были примерно одной длины, а не так, что в одном, например, 2 элемента, а в другом 100. При большом количестве элементов в списках теряется смысл хеширования – «доступ к любому элементу в таблице за фиксированное время», так как поиск приведет к линейному

перебору длинного списка. Можно создать списки упорядоченными и за счет этого еще оптимизировать время поиска. Нет функций, которые удовлетворяли бы принципу однородного хеширования для *всех* разрабатываемых приложений.

2. Проблема размера таблицы

Если создать таблицу сразу большого размера, то многие элементы могут просто не использоваться

Лучше создать таблицу из нескольких элементов и, по мере увеличения числа вставляемых элементов, ее увеличивать. Когда это может понадобиться? Если размер не изменять, то будут расти списки и время поиска опять будет расти пропорционально количеству размещенных в таблице записей с данными – O(n).

Поэтому вводится коэффициент нагрузки хеш-таблицы: это отношение количества размещенных в хеш таблице записей с данными к длине таблицы (размеру массива).

*Коэффициент нагрузки=**count/L,*** где count – количество записей в таблице, L – длина таблицы.

Если ***count/L>0,75***, то следует увеличить размер таблицы вдвое, это гарантирует, что размер списков будет небольшим, меньшим 1 (***count=L*0,75).***

Изменение размера таблицы влечет за собой повторное хеширование всех элементов из списков в новую таблицу большего размера, при этом алгоритм хеш- функции может остаться прежним, но размер таблицы увеличенным вдвое. Этот процесс называют *рехешированием.*

1.5.4 Анализ цепного хеширования

Докажем, что цепное хеширование позволяет выполнить операции вставки, удаления и поиска элемента за фиксированное время, т.е. время выполнения операции оценивается по нотации «большое О» как O(1).

Для доказательства, введем обозначения:

n - это количество элементов размещенных в таблице;

m - размер таблицы (количество списков).

В качестве критерия оценки времени выполнения операций будем использовать количество выполняемых итераций - сравнений.

Операция поиска.

Данная операция связана с тем, что задается ключ поиска, хеш-функция вычисляет индекс для хеш-таблицы и далее выполняется поиск в списке этого индекса.

Средний размер списка не должен превышать 0,75 (согласно правила создания хеш-таблицы). т.е. потребуется 0,75 итераций. Следовательно, среднее

время в лучшем и худшем случаях будет постоянным и может быть вычислено так:

$O_{ср}(n,m)_{наилучшем\ случае}=n/(2m)$ при удвоенном размере таблицы

$O_{ср}(n,m)_{наихудшем\ случае}=n/m$

Т.о. время выполнения операции поиска зависит не от n (количества размещенных элементов), а от отношения ***n/m***, а, следовательно, его можно аппроксимировать как ***O(1)***.

Операция удаления.

Данная операция состоит из двух алгоритмов:

- Поиск элемента с заданным ключом.
- Непосредственно удаление элемента из списка.

Для поиска элемента требуется время, определяемое как O(1). Для удаления элемента из списка требуется также время, определяемое как O(1). Эти две операции будут выполняться последовательно, согласно правилу суммы, время выполнения операции удаления так же определяется как ***O(1).***

Операция вставки нового элемента в хеш-таблицу.

Данная операция включает алгоритмы:

- Выполнение хеш-функции
- Вставка нового элемента в список.

Оба алгоритма выполняются за время O(1). Тогда и весь алгоритм так же за O(1). Но это справедливо для таблицы с однородным распределением элементов, которое поддерживается правилом рехеширования. Время рехеширования будет зависеть от n, но рехеширование выполняется один раз на 2m вставок, а, следовательно, будет постоянным т.е. O(1). При выполнении операций вставки и только в случае рехеширования время будет большим.

1.6 Метод хеширования с открытым адресом

Другой метод разрешения коллизий – это организация *таблицы с открытым адресом.*

Открытый адрес – это свободная ячейка таблицы, а *закрытый адрес* – это занятая ячейка.

Суть метода

Значение вставляется непосредственно в таблицу в открытую ячейку (массив из записей с ключами).

Таблица – это массив, элементы которого могут содержать только ключ и ссылку на данные, связанные с ключом и хранящиеся в другой структуре данных программы, или данные вместе с ключом. Связанных списков нет.

При выполнении операции вставки записи в таблицу:

1. Хеш-функция определяет индекс элемента в таблице в соответствии со значением ключа записи;
2. Выполняется проверка ячейки: открыта или закрыта
   2.1. если ячейка открыта, то запись вставляется в эту ячейку таблицы;
   2.2. если ячейка закрыта, то выполняется процесс разрешения коллизии.

При выполнении операции поиска записи в хеш-таблице:

1. Хеш-функция определяет индекс элемента в таблице в соответствии со значением ключа записи;
2. Выполняется проверка ячейки на содержание заданного ключа поиска
   2.1. если ячейка содержит запись с ключом поиска, то формируется результат удачного поиска;
   2.2. если ячейка не содержит запись с ключом поиска, то продолжается процесс поиска заданного ключа, при этом используется алгоритм, применяемый при добавлении записи в таблицу в случае коллизии. Поиск будет продолжаться при безуспешном поиске до первой открытой ячейки,

Выполнение операции удаления записи из хеш-таблицы с открытым адресом имеет свои трудности и связано это с тем, что при удалении нельзя делать ячейку открытой, так как другие ключи при разрешении коллизий были вставлены после удаляемой и поиск свободной выполнялся при закрытой ячейке.

Более подробно все тонкости операций рассмотрим далее на примерах.

1.6.1 Структура хеш-таблицы

Рассмотрим пример на создание хеш-таблиц с открытым адресом.

Пример 5. Пусть множество значений, в котором необходимо выполнять операции поиска представлен двоичным файлом. Записи файла имеют уникальный ключ – целое число. Для доступа к элементам файла создадим хеш-таблицу с открытым адресом, элемент хеш-таблицы будет хранить ключ и указатель на запись с этим ключом в файле. Тогда хеш-таблицу можно определить в программе так:

```
Typedef int Tkey;          //тип ключа (для общего случая)
struct Trec{               //структура элемента таблицы
        Tkey Key;          //ключ
        void * ptr;        //указатель на запись в файле
};
struct TTable{             //определение структуры хранения таблицы
        int M;         //длина таблицы
        int N;         //количество закрытых адресов таблицы
```

```
        Trec *Keys; //непосредственно таблица из элементов
    };
    TTable HeshTable;       //переменная – представляет таблицу
```

Пример 6. Пусть множество значений, в которых необходимо выполнять операции поиска таково, что можно все значения разместить в оперативной памяти. Например, сведения о студентах какого-то факультета. Все значения множества имеют уникальный ключ, например, номер зачетной книжки. Тогда хеш-таблицу можно определить так:

```
struct Titem  {
        Tkey Key;           //ключ
        char Fam[40];
        char Adress[50];
        Tdate BirthData;
        char NumGroup[10]
};
struct TTable{
        int M;      //длина таблицы
        int N;      //количество закрытых адресов таблицы
        Titem *values; //непосредственно таблица из элементов;
};
  TTable HeshTable;
```

Пример 7. Создать хеш-таблицу с открытым адресом и разместить в ней данные с ключами. Данными будут записи содержащие: номер зачетной книжки и фамилия студента.

```
struct Titem  {
        Tkey Key;           //ключ
        char Fam[40];
};
struct TTable{
        int M;      //длина таблицы
        int N;      //количество закрытых адресов таблицы
        Titem *values; //непосредственно таблица из элементов;
};
```

Новый элемент вставляется в ячейку таблицы, индекс которой создала хеш-функция. Если ячейка свободна (открытый адрес), то в нее вставляется значение

с ключом. В случае появления коллизии, осуществляется поиск свободной ячейки, *путем линейного опробования, следующих ячеек (индекс увеличивается на единицу),* пока не будет найдена свободная ячейка. Эффективность данного подхода падает, когда коэффициент нагрузки приближается к единице.

Алгоритм операции вставки нового значения в таблицу будет состоять из блоков:

- получение индекса с помощью хеш-функции (обозначим индекс ind)
- если этот адрес открыт, то разместить в нем значение;
- если адрес закрыт (коллизия), то надо подобрать *первый открытый адрес.*

Самый простой способ подбора – это *линейный подбор со смещением равным единице.*

Суть линейного подбора: проверить ячейку таблицы, следующую за ячейкой с индексом ind, т.е. ind=ind+1, если и этот адрес закрыт, то снова выполнить подбор со смещением на единицу и так до тех пор, пока не будет найден открытый адрес.

Пусть в хеш-таблицу надо вставить следующие данные:

ключ	фамилия
111001	Иванов
111002	Петров
111007	Сидоров
112001	Волков
212001	Лисицын
303002	Жаворонков
304002	Медведев
305002	Рыбин
303010	Акулов

Создадим хеш-таблицу с открытым адресом из 100 элементов. Хеш-функция определяет индекс по формуле key%100/

Первые три значения вставляются соответственно в ячейки с индексами 1, 2, 7. В результате хеш-таблица будет содержать следующие данные.

индекс	ключ	фамилия
1	111001	Иванов

2	111002	Петров
3		
4		
5		
6		
7	111007	Сидоров
8		
9		
10		
11		
…….	……...	…….
100		

При вставке значения с ключом 112001 возникает коллизия, тогда подбираем линейно со смещением единица открытый адрес, это ячейка с индексом ind=2, но это закрытый адрес, тогда вновь применяем линейный подбор (ind=ind+1), это ячейка с индексом 3 открытая, и в ней размещается запись.

индекс	ключ	фамилия
1	111001	Иванов
2	111002	Петров
3	112001	Волков
4		
5		
6		
7	111007	Сидоров
8		
9		
10		
11		
…….	……...	…….
100		

Далее надо вставить запись с ключом 212001. Ячейки с индексами 1 и 2 и 3 закрыты, но открыта ячейка с индексом 4, производим в нее вставку записи.

индекс	ключ	фамилия
1	111001	Иванов
2	111002	Петров

3	112001	Волков
4	212001	Лисицын
5		
6		
7	111007	Сидоров
8		
9		
10		
11		
…….	…….	…….
100		

Запись с ключом 303002 не может быть размещена в ячейке с индексом 2, так как адрес закрыт, тогда подбирается открытая ячейка и это ячейка с индексом 5.

индекс	ключ	фамилия
1	111001	Иванов
2	111002	Петров
3	112001	Волков
4	212001	Лисицын
5	303002	Жаворонков
6		
7	111007	Сидоров
8		
9		
10		
11		
…….	…….	…….
100		

Задание для самостоятельной работы

Завершите заполнение хеш – таблицы примера 7. Отметьте строки таблицы, в которых разрешена коллизия.

1.6.2 Операция вставки записи в хеш-таблицу с открытым адресом

При выполнении операции вставки, необходимо найти открытый адрес в таблице для вставляемой записи в соответствии со значением ключа.

Как проверить, открыт ли адрес в таблице?

Само самой напрашивается операция - контроль ключа.

Но в одном случае ключ — это числовое значение, в другом - строковое, или ключ имеет какой-то другой тип.

Удобно добавить в таблицу еще одно поле – признак свободной ячейки. Например, это поле может быть логического типа, тогда значение true, будет указывать на то, что ячейка свободна (адрес открыт), а значение false на то, что адрес закрыт.

Для рассматриваемого примера хеш-таблица будет иметь вид:

Индекс	Ключ	Фамилия	Признак открытого адреса
1	111001	Иванов	false
2	111002	Петров	false
3	112001	Волков	false
4	212001	Лисицын	false
5	303002	Жаворонков	false
6	304002	Медведев	false
7	111007	Сидоров	false
8	305002	Рыбин	false
9			true
10	303010	Акулов	false
11			true
……..	……...	…….	true
100			true

Рисунок 10.9.6.2.1. Таблица со вставленными записями

В связи с этим внесем изменения в структуру элемента хеш-таблицы, представленную в примере 7.

```
Struct Titem{
        Key Tkey;          //ключ
        char Fam[40];
        bool Pins;    //признак свободной для вставки ячейки
}
```

Тогда реализация алгоритма операции вставки на языке C++ может быть такой:

```
Имена объектов алгоритма
v – значение вставляемой записи
T – хеш-таблица
```

```
void ins(Titem v, TTable &T){
 int     ind= hesh(v.key,T.m);
 while (!T.values[ind].Pins) do
        ind++;
 if (ind<T.m){
        T.values[ind]=v;
        T.n=T.n+1;
        T.Pins=false;
 }}
```

1.6.3 Операция поиска записи с в хеш-таблице с открытым адресом

Эта операция имеет следующий алгоритм:

1. Получение ключа поиска К.
2. Определение индекса элемента в таблице по ключу К посредством хеш – функции, которая использовалась при вставке значений в таблицу.
3. Проверка, содержит ли элемент, ключ поиска:
   3.1. если этот элемент содержит введенный ключ, то поиск завершается и возвращаются данные записи;
   3.2. если элемент таблицы по этому индексу не содержит заданный ключ и индекс не вышел за границу таблицы, то осуществляется алгоритм подбора нового индекса по тому же алгоритму, который использовался при выполнении операции вставки в случае коллизии (подбор со смещением 1), пока не будет найден элемент с таким ключом, или не будет найдена открытая ячейка. Если найдена открытая ячейка, то это означает, что такого ключа в таблице нет.

Например, найти запись с ключом 304002 или с ключом 507002. С ключом 304002 будет найдена, а с ключом 507002 не найден и процесс поиска будет остановлен, как только будет найдена в пути поиска первая открытая ячейка - это ячейка с индексом 9.

1.6.4 Операция удаления записи из хеш-таблицы с открытым адресом

При выполнении операции удаления сначала осуществляется поиск записи с заданным ключом в таблице, а затем, в случае удачного поиска, и выполняется удаление записи из таблицы, т.е. ячейка должна стать открытой.

Но давайте попробуем удалить из таблицы, представленной на рисунке 10.9.6.2.1, значение с ключом 304002.

Тогда в поле *Признак открытого адреса* надо поставить значение true. В таблице 22 представлено содержание хеш-таблицы после выполнения операции

Таблица 22. Содержание хеш-таблица с открыты адресом после удаления ключа 304002

индекс	ключ	фамилия	Признак открытого адреса
1	111001	Иванов	false
2	111002	Петров	false
3	112001	Волков	false
4	212001	Лисицын	false
5	303002	Жаворонков	false
6	304002	Медведев	true
7	111007	Сидоров	false
8	305002	Рыбин	false
9			
10	303010	Акулов	false
11			true
…….	……..	…….	true
100			true

Выполненная операция удаления, в таком виде как мы ее только что рассмотрели, породила проблему поиска и вот какую.

Например, нам необходимо найти запись с ключом 305002.

Процесс поиска: сначала хеш–функция создала индекс 2, затем последовательно просматриваем все элементы хешированного пути: (элементы с индексами) 3, 4, 5 (они закрыты) и переходим к элементу с индексом 6, в поле *Признак открытого адреса* которого находиться значение true – т.е. ячейка свободна и процесс поиска завершается, но элемент не найден, хотя запись с таким ключом присутствует в таблице.

Чтобы устранить эту проблему операции удаления, удобно в таблицу ввести еще одно поле - *признак удаленной записи*. Это поле, так же, как и поле *Признак открытого адреса,* может быть логического типа и содержать true, если запись удалена и false, если не удалена.

В таком случае таблица примет вид

индекс	ключ	фамилия	Признак открытого адреса	Признак удаления
1	111001	Иванов	false	false
2	111002	Петров	false	false

3	112001	Волков	false	false
4	212001	Лисицын	false	false
5	303002	Жаворонков	false	false
6	304002	Медведев	true	true
7	111007	Сидоров	false	false
8	305002	Рыбин	false	false
9			true	false
10	303010	Акулов	false	false
11			true	false
… ……	……..	…….	true	false
100			true	false

Рис.2. Таблица с добавленными признаками удаления и свободной ячейки

В связи с этими изменениями таблицы, изменится алгоритм поиска записи по ключу.

При поиске записи с заданным ключом, в случае свободной ячейки надо проверять и значение поля *Признак удаления.*

В связи с этим, внесем изменения в структуру элемента таблицы.

```
Struct Titem{
        Key Tkey;          //ключ
        char Fam[40];
        bool Pins;    //признак свободной для вставки ячейки
bool Pdel;   //признак удаления
}
```

Пример 8. Алгоритм поиска записи с ключом в таблице с открытым адресом, имеющей поля с признаками свободной и удаленной ячеек на языке С++.

```
Int  Find_value(TTable T, int key){
  int i;
   i= hesh(key,T.m);
 while(! T.values[i].Pdel && ! T.values[i].Pins and (T.values[i].key!=key){
 i++
   if (T.values[i].Pdel and T.values[i].Pins) return -1
   else
      return i;
}
```

1.6.5 Проблемы хеширования с открытым адресом

1) Рехеширование

По мере вставки элементов в таблицу может потребоваться ее расширение. Когда это сделать?

Вставлять новые значения на место удаленных значений в таблицу с открытым адресом нельзя, тогда количество занятых ячеек равно n+количество удаленных после последнего рехеширования таблицы, где n количество значений не удаленных ячеек. Тогда рехеширование надо выполнять когда ***n+кол.удаленных>m*0,75.*** При рехешировании в хешированную таблицу не вставляются удаленные записи.

2) Первичная кластеризация

Кластер - это последовательность непустых позиций в таблице. При добавлении нового элемента в кластер он не только становится длиннее, но и растет быстрее. Так как ключи, хешированные с новым индексом, будут следовать по тому же пути, что и те, что уже вставлены в него. Например, в приведенной на рис.2 таблице кластер составляют ячейки с индексами с 1 по 8. При добавлении в таблицу ключа 777002, подбор свободного адреса требует просмотра ячеек с индексами со второго по 9. Пути подбора становятся все длиннее.

Первичная кластеризация – это явление, которое происходит, когда обработчик коллизий создает условие роста кластера. Обработчик коллизий со смещением 1 способствует первичной кластеризации. Первичная кластеризация порождает длинные пути.

Как устранить появление кластеров? Выбрать другое значение смещения, например, 20. Но тогда ключи, хешированные с индексом ind, будут перекрывать путь, занимаемый ключами: ind+20, ind+40, ind+60 и т.д. Но такое смещение порождает еще большую проблему, чем первичная кластеризация. И вот какую.

Пусть таблица имеет размер m=100. Смещение равно 20. Если ключ хеширован с индексом 33, то для данного кластера будут доступны ключи 33, 53, 73, 93, 13, а далее вновь ключи будут хешированны с индексами 33, 53, 73, 93, 13, т.е. больше ничего вставит нельзя

Причина появления такой последовательности в том, что смещение и размер таблицы имеют общий множитель, а хеш-функция определена по делению на размер таблицы.

*Данной проблемы можно избежать, если **размер таблицы** определить **простым числом.***

Причина появления первичной кластеризации в том, что смещение остается постоянным значением.

1.6.6 Двойное хеширование

Для устранения первичной кластеризации надо сделать смещение, зависимым от ключа, т.е. получать его тоже с помощью хеш-функции.

Например, индекс вычислять по формуле key % m, а смещение по формуле m / m.

Рассмотрим пример использования двойного хеширования. Пусть имеется последовательность ключей, рассчитаем для них индекс и смещение и посмотрим, появляется ли проблема первичной кластеризации. Значения ключей индекс и смещения, рассчитанные для ключей, приведены в таблице 23. Размер таблицы m=19.

Таблица 23. Ключи и их параметры для алгоритма двойного хеширования

key	Key % m	Key / m
33	14	1
72	15	3
71	14	3
112	17	5
56	18	2
109	14	5

Этот пример показывает, что ключи из различных коллизий и даже из одной и той же не следуют одному пути.

Рассмотрим процесс вставки значений в таблицу и как реализуются коллизии.

1) Вставляется запись с ключом key=33 в ячейку таблицы с индексом 14.
2) Вставляется запись с ключом key=72 в ячейку таблицы с индексом 15.
3) Вставляется запись с ключом key=71 в ячейку таблицы с индексом 14, но эта ячейка занята, тогда надо реализовать коллизию. При этом индекс рассчитывается по формуле (14+3) % m, что равно 17. Т.е. 71 вставляется в ячейку с индексом 17.
4) Ключ 112 хеширован с индексом 17, но ячейка занята, следовательно, рассчитываем новый индекс, он равен (17+5) % 19=3. Ячейка 3 свободна в нее и вставляем 112.
5) Ключ 56 хеширован с индексом 18, позиция свободна.
6) Ключ 109. хеширован с индексом 14, позиция занята, (14+5)%19=0
7) В данном примере применялись две хеш-функции и этот способ реализации коллизий получил название – *двойное хеширование*.

Если смещение будет числом кратным размеру таблицы, то может появиться новая проблема. Например, пусть в таблицу размером 19 вставляется запись с ключом 736, тогда:

индекс (736 % 19) = 14

Так как ячейка с индексом 14 занята, то возникает коллизия, для ее реализации рассчитаем смещение (736/19)=38. Находим индекс следующей ячейки (14+38)%19 = 14 т.е. получаем вновь 14. Чтобы выйти из этого замкнутого круга можно применить смещение равным 1 в тех случаях, когда смещение кратно размеру таблицы.

1.7 Методы разработки хеш–функций

В этом разделе мы рассмотрим некоторые вопросы, связанные с разработкой качественных хеш-функций, и познакомимся с двумя схемами их построения: хеширование делением и хеширование умножением, эвристичны по своей природе.

Существуют и другие схемы, например — универсальное хеширование — использует рандомизацию для обеспечения доказуемо высокой производительности.

1.7.1 Метод деления

Построение хеш-функции методом деления состоит в отображении ключа к в одну из n ячеек путем получения остатка от деления к на m, т.е. хеш-функция имеет вид $h(k) = к \bmod m$.

Например, если хеш-таблица имеет размер m = 12, а значение ключа к = 100, то h(k) = 4.

Поскольку для вычисления хеш-функции требуется только одна операция деления, хеширование методом деления достаточно быстрое.

При использовании данного метода мы обычно стараемся избегать некоторых значений m.

Например, m не должно быть степенью 2, поскольку если $m = 2^P$, то h(k) представляет собой просто p младших битов числа к. Если только заранее не известно, что все наборы младших p битов ключей равновероятны, лучше строить хеш-функцию таким образом, чтобы ее результат зависел от всех битов ключа.

Можно показать неудачность выбора $m = 2^P — 1$, когда ключи представляют собой строки символов, интерпретируемые как числа в системе счисления с основанием 2^P, поскольку перестановка символов ключа не приводит к изменению его хеш-значения.

Зачастую хорошие результаты можно получить, выбирая в качестве ***значения m простое число,*** достаточно далекое от степени двойки.

Предположим, например, что мы хотим создать хеш-таблицу с разрешением коллизий методом цепочек для хранения порядка n = 2000 символьных строк, размер символов в которых равен 8 бит.

Нас может устроить проверка в среднем трех элементов при неудачном поиске, так что мы выбираем размер таблицы равным m — 701. Число 701 выбрано как простое число, близкое к величине 2000/3 и не являющееся степенью 2.

Рассматривая каждый ключ ***к*** как целое число, мы получаем искомую хеш-функцию h(k) = k mod 701.

1.7.2 Метод умножения

1. Построение хеш-функции методом умножения выполняется в два этапа. Сначала мы умножаем ключ **к** на константу 0<A<1 и выделяем дробную часть полученного произведения.
2. Затем мы умножаем полученное значение на m и применяем к нему функцию “пол”.

Тогда хеш-функция имеет вид

h(k) = m (к*A mod 1) ,

где выражение “к*A mod 1” означает получение дробной части произведения **к***A*, т.е. величину *к***A* — [*к***A*].

Преимущество метода умножения заключается в том, что значение m перестает быть критичным. Обычно величина m из соображений удобства реализации функции выбирается равной степени 2 ($m = 2^P$ для некоторого натурального p), поскольку такая функция легко реализуема на большинстве компьютеров.

Пример применения метода к построению алгоритма хеш-функции.

Пусть у нас имеется компьютер с размером слова w бит и ***к*** помещается в одно слово. Ограничим возможные значения константы A дробями вида $s/2^{w,}$ где s — целое число из диапазона $0<s<2^W$. Тогда мы сначала умножаем **к** на w-битовое целое число s , где $s= A \cdot 2^W$.

Результат представляет собой 2w-битовое число вида: $r_1\ 2^w + r_o$, где r_1 — старшее слово произведения, а r_o — младшее.

Старшие p бит числа r_o представляют собой искомое p-битовое хеш-значение. Значение p передается хеш-функции.

Примечание. Хотя описанный метод работает с любыми значениями константы A, одни значения дают лучшие результаты по сравнению с другими. Оптимальный выбор зависит от характеристик хешируемых данных.

Кнут (Knuth) предложил использовать дающее неплохие результаты значение

$$A \approx \frac{\sqrt{5}-1}{2} = 0.6180339887..$$

Пример использования метода на данных

Предположим, что $к$ = 123456, p = 14, m = 2^{14} = 16384 и w = 32.

Принимая предложения Кнута, выберем A в виде дроби s/2^{32}, ближайшей к значению $\frac{\sqrt{5}-1}{2}$, так что A = 2654435769/2^{32}. Тогда к*s = 327706022297664 = (76300 • 2^{32}) +17612864, и, таким образом, r1 = 76300 и r0 = 17612864.

Старшие 14 бит числа r0 дают хеш-значение h(k) = 67.

1.7.3 Примеры хеш - функций

1. Хеш-функция использует метод деления на количество элементов таблицы

Пусть L – длина (количество элементов) таблицы. Тогда Хеш-функция может представлять собой [key mod L]

```
 h(int key, int L)
{
        return key % L;
}
```

Этот метод приводит к коллизиям, только для ключей с одинаковыми последними L цифрами.

2. Хеш-функция использует метод симметричных ключей

Используется для символьных ключей. Суть алгоритма вычисления индекса: к коду первого символа ключа прибавляется код последнего символа ключа, результат - остаток от деления на длину таблицы.

```
int h(string key, int L)
 {
        return ((int)key[0])+(int)key[key.size()]) % L
 }
```

Этот метод приводит к коллизиям, только для ключей с одинаковыми последним и первым символами.

3. Хеш-функция использует метод суммирующих ключей.

Используется для символьных ключей. Суть метода: суммируются коды всех символов, входящих в состав ключа и полученная сумма делится на длину таблицы. Коллизии возникают при одинаковых суммах.

4. Хеш-функция использует метод средних квадратов.

В этом методе ключ – строка преобразуется в целое число, используя коды символов, в целое число результат возводиться в квадрат, из полученного результата извлекается средняя последовательность битов. Если код 32 бита, то средняя часть 11-27.