Лекция 10.Хеш-таблицы и хеширование

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Алгоритмы и структуры данных

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель и план лекции

Цель лекции

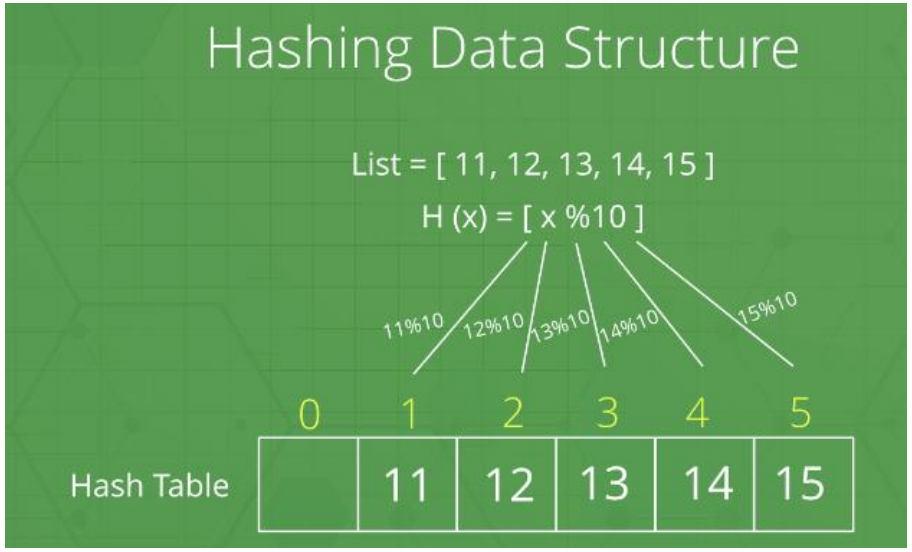
Изучить принципы хеширования, хеш-функции и хеш-таблицы, причины возникновения коллизий и методы их разрешения.

План лекции

1. Необходимость хеширования
2. Компоненты хеша: ключ, хеш-функция, хеш-таблица
3. Свойства хеш-функции
4. Коллизии и способы их разрешения
5. Метод цепочек, реализация на C++

Хеширование

- Хеширование-это процесс создания вывода постоянного размера из входов переменного размера с использованием математических формул, называемых хеш-функциями. Этот метод определяет индекс или место хранения элемента в структуре данных.



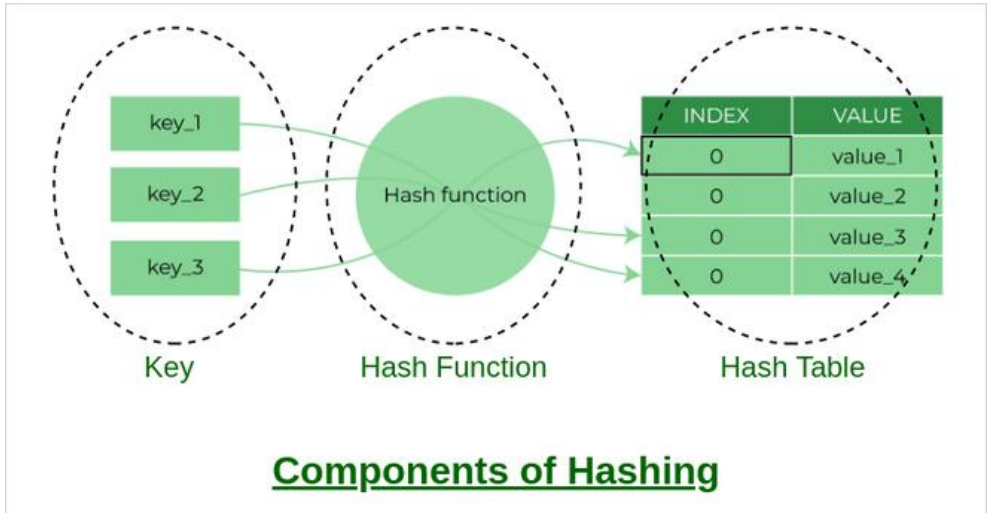
Необходимость хеша

- Объем данных в интернете растет в геометрической прогрессии с каждым днем, что затрудняет их эффективное хранение. В повседневном программировании этот объем данных может быть не таким большим, но его необходимо хранить, получать к нему доступ и легко и эффективно обрабатывать. Очень распространенная структура данных, используемая для этой цели,-это структура данных Array.
- Теперь возникает **вопрос**: если Array уже существует, какова потребность в новой структуре данных?

- Итак, теперь мы ищем структуру данных, которая может хранить данные и искать их в обычное время, то есть во время O(1). Вот как появилась структура данных хеширования. Введя структуру данных Hash, теперь вы можете легко хранить данные в постоянное время, а также получать их в постоянное время.

Компоненты хеша

- **Ключ:**
  - ключ может быть любой строкой или целым числом, заданным в качестве входа в хеш-функцию – это хеш-функция, которая определяет индекс или место хранения элемента в структуре данных.
- **Хеш-функция:**
  - хеш-функция получает входной ключ и возвращает индекс элемента в массиве, называемом хеш-таблицей. Индекс известен как хеш-индекс .
- **Хеш-таблица:**
  - хеш-таблица-это структура данных, которая сопоставляет ключи со значениями с помощью специальной функции, называемой хеш-функцией. Хеш ассоциативно хранит данные в массиве, где каждое значение данных имеет свой уникальный индекс.



Как работает хеширование?

- Предположим, у нас есть набор строк {"ab", "cd", "efg"}, и мы хотим сохранить его в таблице.

- Наша основная цель здесь-быстро найти или обновить значения, хранящиеся в таблице во время O (1), и порядок строк в таблице нас не беспокоит. Таким образом, данный набор строк может действовать как ключ, а сама строка-как значение строки, но как сохранить значение, соответствующее ключу?

- Шаг 1: мы знаем, что хеш-функции (представляющие некоторую математическую формулу) используются для вычисления хеш-значения, которое действует как индекс структуры данных, в которой хранится значение.
- Шаг 2: Итак, давайте назначим
  - "a" = 1,
  - «b»=2, .. и т.д. Для всех букв алфавита.
- Шаг 3: таким образом, числовое значение получается путем добавления всех символов строки:
  - “ab” = 1 + 2 = 3,
  - “cd” = 3 + 4 = 7 ,
  - “efg” = 5 + 6 + 7 = 18

Как работает хеширование #2

Шаг 4: Теперь предположим, что у нас есть таблица размера 7 для хранения этих строк. Используемая здесь хеш-функция-это сумма символов в базовом модификаторе Table size . Мы можем вычислить положение строки в массиве, взяв sum (string) MOD 7 .

Шаг 5:

"ab" in 3 mod 7 = 3,

"cd" in 7 mod 7 = 0, and

"efg" in 18 mod 7 = 4.

0	1	2	3	4	5	6
cd			ab	efg		

Описанный выше метод позволяет вычислить положение заданной строки с помощью простой хеш-функции и быстро найти значение, хранящееся там. Таким образом, идея хеширования кажется идеальным способом хранения пар данных (ключ, значение) в таблице.

Что такое хеш-функция

- Хеш-функция выполняет сравнение между ключом и значением. Это делается с помощью математических формул, называемых хеш-функциями. Результат хеш-функции называется хеш-значением или хешем. Хеш-значение представляет начальную строку символов, но обычно меньше исходной.
- Например: думайте о массиве как о карте, где ключ-это индекс, а значение-это значение этого индекса. Итак, для массива a, если у нас есть индекс I, который рассматривается как ключ, мы можем найти значение, просто посмотрев на значение a[i].просто ищите A[i].

- Существует множество хеш-функций, в которых используются цифровые или буквенно-цифровые ключи.
  - Типы хеш-функций:
  - Метод разделения.
  - Метод среднего квадрата.
  - Метод складывания.
  - Способ размножения

Свойство хеш-функции

- Хеш-функция, которая отображает каждый элемент в свой уникальный слот, называется идеальной хеш-функцией. Если мы знаем элементы, и набор никогда не меняется, мы можем создать идеальную хеш-функцию, но проблема в том, что нет систематического способа создать идеальную хеш-функцию для произвольного набора элементов. К счастью, мы получаем эффективность производительности, даже если хеш-функция не идеальна. Мы можем получить отличную хеш-функцию, увеличив размер хеш-таблицы для размещения всех возможных значений. В результате каждый элемент имеет уникальный слот. Хотя этот подход возможен для небольшого количества элементов, он непрактичен, когда количество элементов велико.
- Хорошая хеш-функция должна иметь следующие свойства:
  - Эффективно рассчитывается.
  - Ключи должны быть равномерно распределены (каждая позиция в таблице одинаково вероятна для каждой.
  - Коллизия (collision) должна быть уменьшена.
  - Коэффициент загрузки должен быть низким (количество элементов в таблице разделено на размер таблицы).

Преимущества хеширования

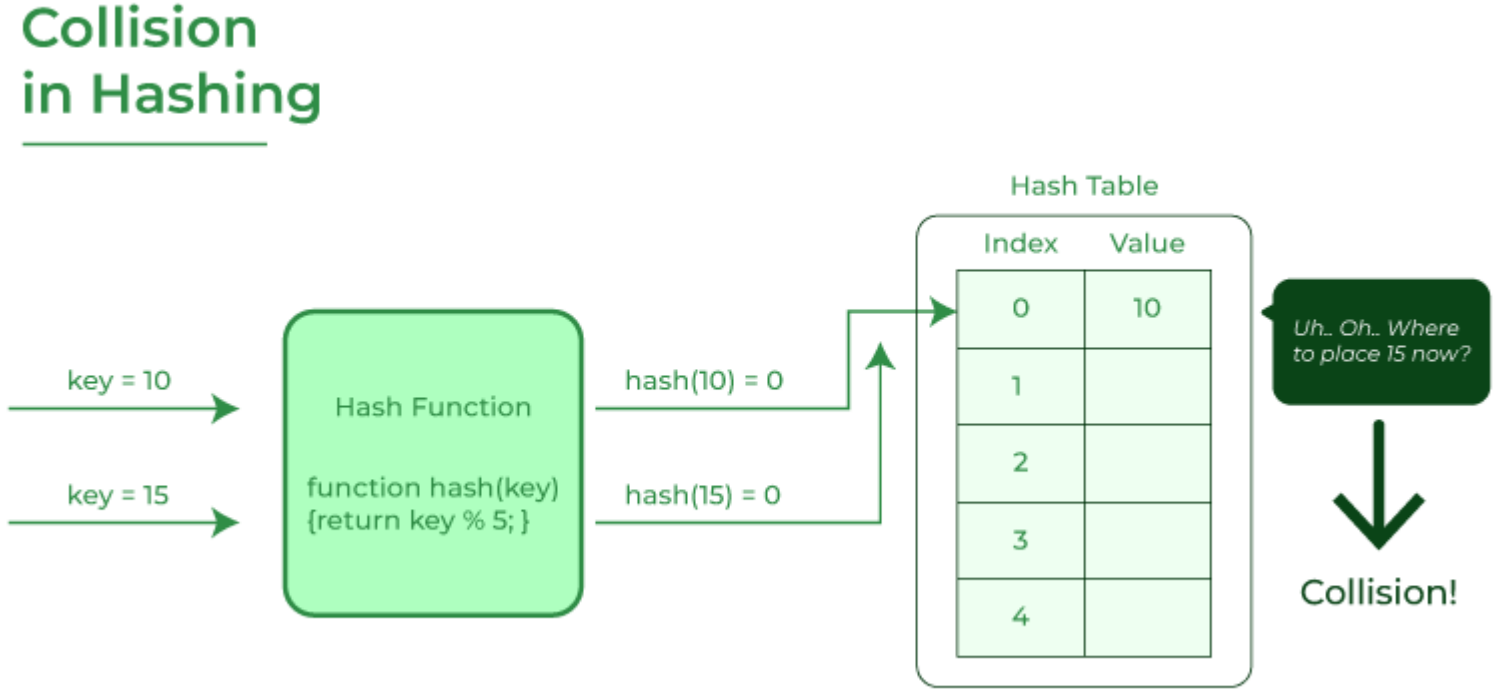
- Поддержка "ключ-значение". Хеширование ключ-значение идеально подходит для реализации структур данных.
- Быстрый поиск данных: хеширование обеспечивает быстрый доступ к элементам с постоянной сложностью.
- Эффективность: операции вставки, удаления и поиска очень эффективны.
- Сокращение использования памяти: для хеширования требуется меньше памяти, так как для хранения элементов выделяется выделенное место.
- Масштабирование: хеширование лучше всего работает с большими наборами данных, обеспечивая более стабильное время доступа.
- Безопасность и шифрование. Хеширование необходимо для безопасного хранения данных и проверки их целостности.

Проблема с хешированием

- Если мы посмотрим на приведенный выше пример, то хеш-функция, которую мы используем, представляет собой сумму букв, но если мы внимательно изучим хеш-функцию, мы можем легко представить проблему: одно и то же значение хеша для разных строк начинает генерироваться хеш-функцией.Например: {"ab", "ba"} имеет одно и то же хеш-значение, а строка {"cd", "be"} также создает одно и то же хеш-значение и т. д.б.это называется столкновением и вызывает проблемы при поиске. , ввод, удаление и обновление значения.

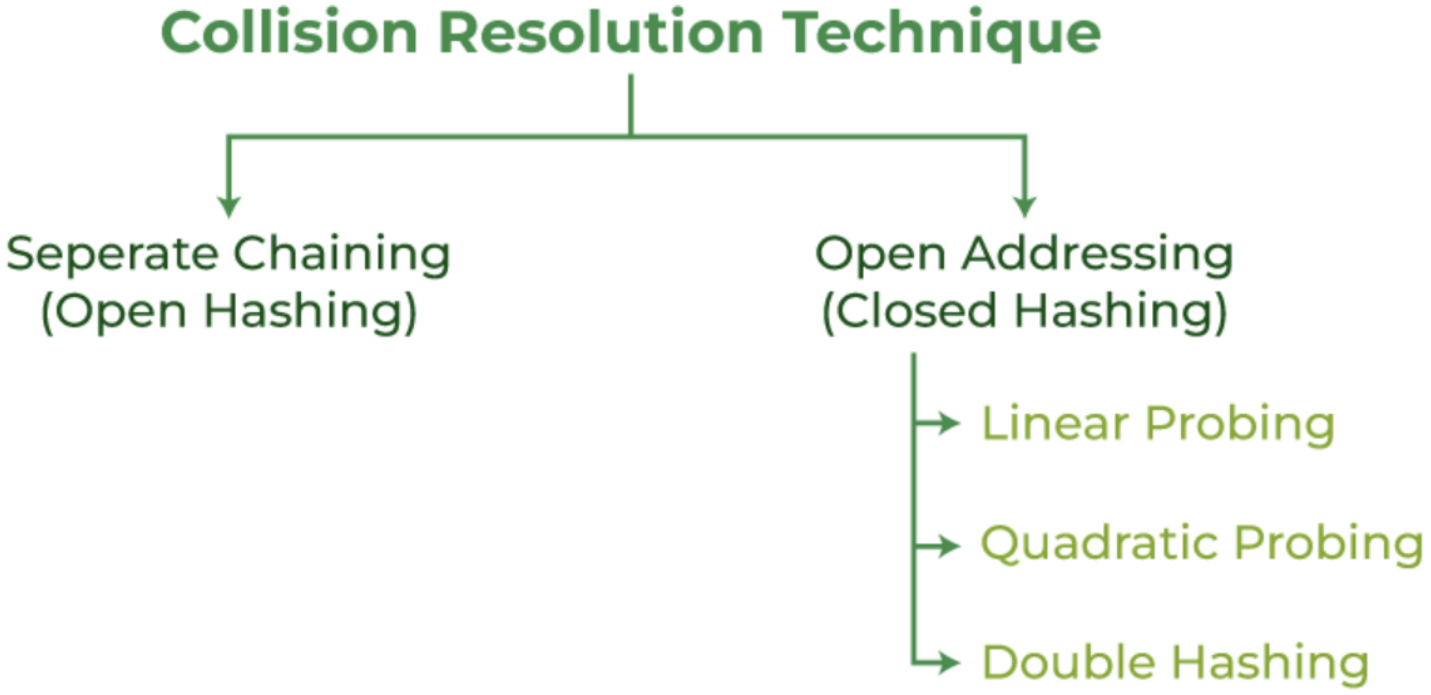
Коллизия (Collision)

- Процесс хеширования дает меньшее число для большого ключа, поэтому два ключа могут давать одинаковое значение. Ситуация, когда вновь введенный ключ сравнивается с непустым ключом, и его необходимо обработать с помощью некоторой технологии обработки столкновений.



решения Collision

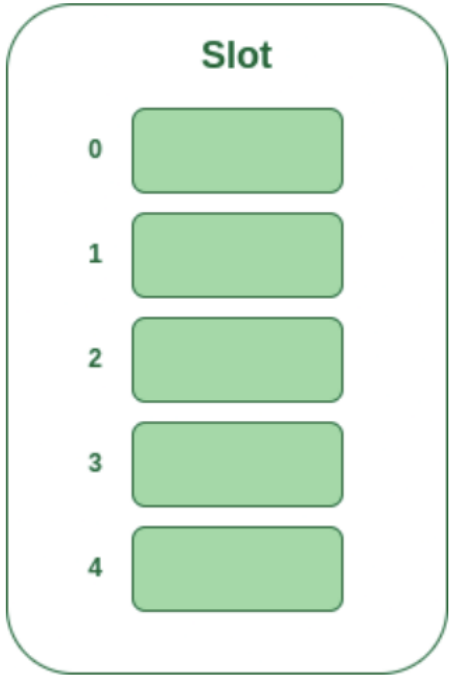
- 1. Separate chaining (отдельная цепочка)
- 2. Open addressing (Closed hashing, открытая адресация)



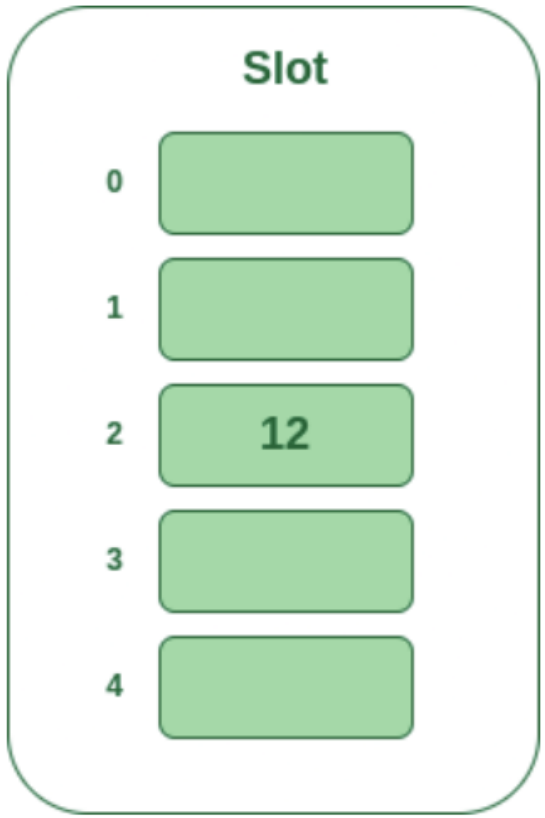
Separate chaining

- Каждая ячейка хеш-таблицы должна отображать связанный список записей с одинаковым значением хеш-функции. Схема проста, но требует дополнительной памяти вне расписания.
- Пример: мы дали хеш-функцию и должны ввести некоторые элементы в хеш-таблицу, используя метод отдельной цепи для метода разрешения столкновений.

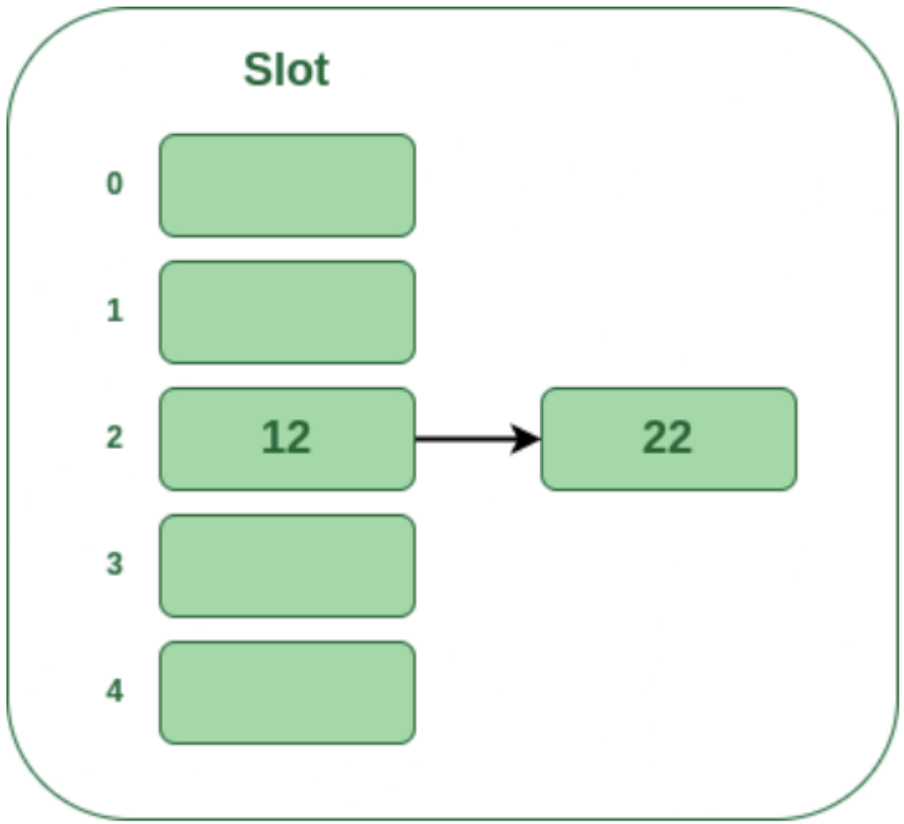
```
Хэш-функция = ключ % 5,
элементы = 12, 15, 22, 25 и 37.
```



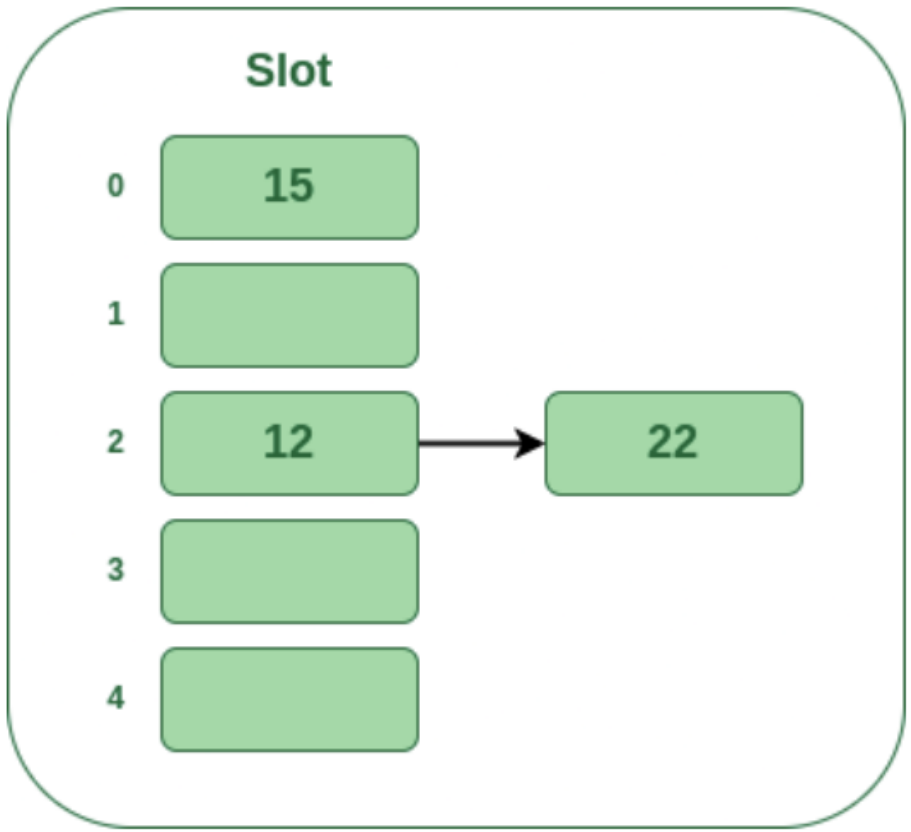
Хеш-таблица



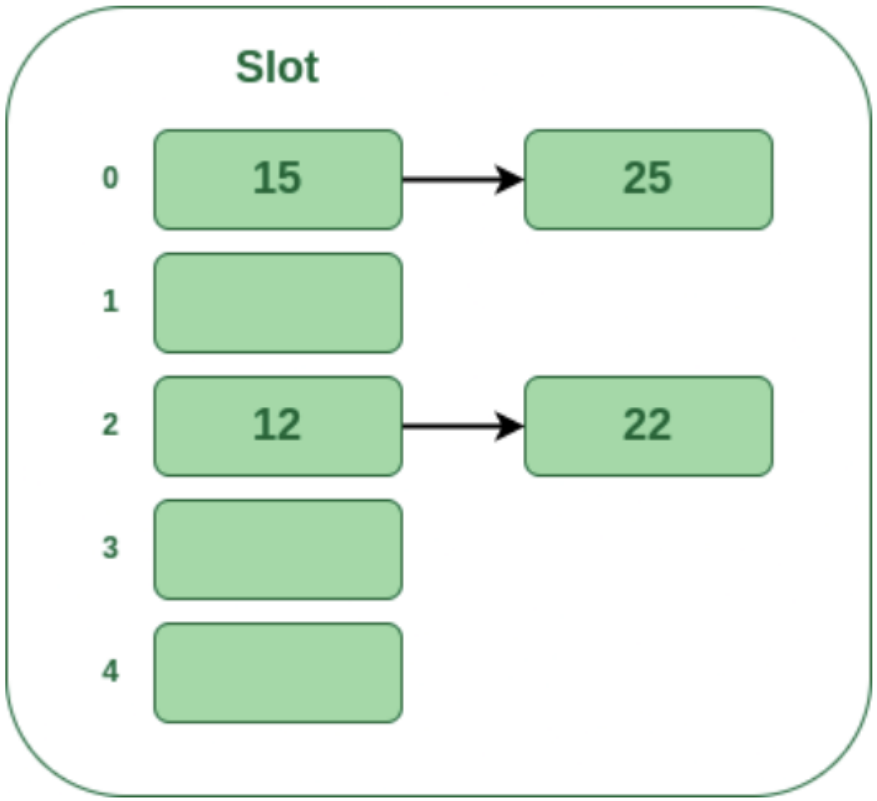
Вставка 12



Вставка 22



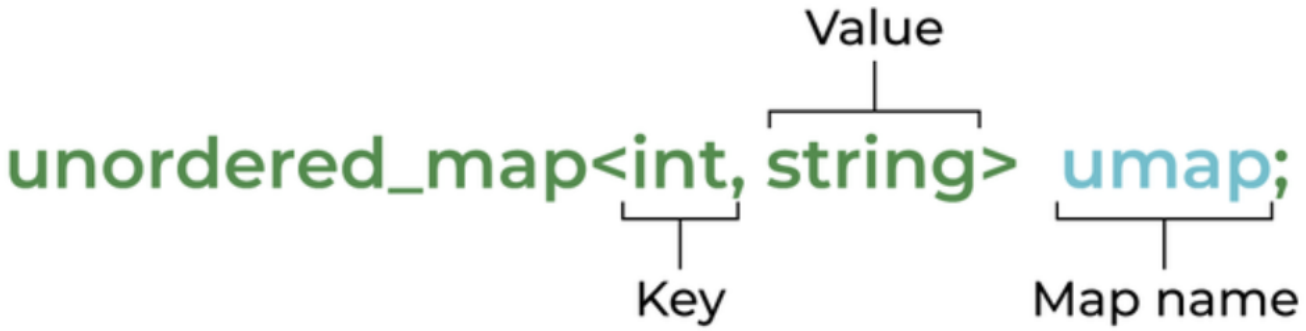
Вставка 15



Вставка 25

C++ реализация

- Внутренний unordered_map реализован с помощью Hash Table, ключ, предоставленный для отображения, хешируется в индексы хеш-таблицы, поэтому производительность структуры данных зависит от хеш-функции, но в среднем стоимость поиска, вставки и удаления хеш-таблица O(1).
- **Unordered Map ==> Time Complexity O(1)**
- В худшем случае его временная сложность может варьироваться от O (1) до (n), особенно для больших простых чисел. В этом случае рекомендуется вместо этого использовать карту (превышающую лимит времени), чтобы избежать ошибки TLE.



unordered_map syntax

code Example

```
#include <iostream>
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
// Declaring umap to be of
// <string, int> type key
// will be of STRING type
// and mapped VALUE will
// be of int type
unordered_map<string, int> umap;

// inserting values by using [] operator
umap[“Comp"] = 10;
umap["Practice"] = 20;
umap["Contribute"] = 30;

// Traversing an unordered map
for (auto x : umap)
            cout << x.first << " " << x.second << endl;
}
```

Заключение

- Использование Хеш-структуры данных
  - Хэш используется в базе данных для индексации.
  - Хеш используется в дисковых структурах данных.
  - Некоторые языки программирования, такие как Python, используют хэш JavaScript для реализации объектов.
- Приложения для структуры хеш-данных в реальном времени
  - Хеш используется для сравнения кеша для быстрого доступа к данным.
  - Вы можете использовать хэш для проверки пароля.
  - Хеш используется в криптографии как дайджест сообщений.
  - Алгоритм Рабина-карпа, соответствующий шаблону в строке.
  - Расчет количества различных подстрок строки.
- Преимущества хеш-структуры данных
  - Хеш обеспечивает лучшую синхронизацию, чем другие структуры данных.
  - Хеш-таблицы более эффективны, чем деревья поиска или другие структуры данных.
  - Хэш обеспечивает среднее стабильное время для операций поиска, вставки и удаления.
- Недостатки хеш-структуры данных
  - Хэш неэффективен во многих конфликтах.
  - Для большого набора возможных ключей столкновение хешей невозможно избежать.
  - Хеш не допускает нулевых значений.

Выводы

1. Хеширование преобразует ключи в индексы таблицы; в среднем поиск, вставка и удаление выполняются за O(1).
2. Хорошая хеш-функция быстро вычисляется и равномерно распределяет ключи.
3. Коллизии неизбежны; их разрешают методом цепочек (separate chaining) или открытой адресацией.
4. При большом числе коллизий эффективность хеш-таблицы снижается.
5. Хеширование применяют в индексах баз данных, кешах, проверке паролей, криптографии и алгоритме Рабина–Карпа.

Контрольные вопросы

1. Зачем нужны хеш-таблицы, если существуют массивы?
2. Что такое ключ, хеш-функция и хеш-таблица?
3. Каковы свойства хорошей хеш-функции?
4. Что такое коллизия и почему её нельзя полностью исключить?
5. Опишите метод цепочек (separate chaining).
6. Чем открытая адресация отличается от метода цепочек?

Список литературы

Основная литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 3-е изд. — М.: Вильямс, 2013.
2. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд. — СПб.: БХВ-Петербург, 2011.
3. Седжвик Р., Уэйн К. Алгоритмы на Java. 4-е изд. — М.: Вильямс, 2013.

Дополнительная литература

4. Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск. 2-е изд. — М.: Вильямс, 2007.
5. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. 4th ed. — MIT Press, 2022.
6. Бхаргава А. Грокаем алгоритмы. — СПб.: Питер, 2017.

Электронные ресурсы

7. VisuAlgo — визуализация алгоритмов и структур данных: visualgo.net
8. Справочник по C++ и STL: en.cppreference.com