COL106: Data Structures and Algorithms

Ragesh Jaiswal, IIT Delhi

Data Structures: Hashing

Data Structures
Hashing

- We have seen data structures for storing and accessing *entries* (key-value pairs) such that the running time for each of the operation is:
  - Search: $O(\log n)$
  - Insert: $O(\log n)$
  - Delete: $O(\log n)$
- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity that all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?

- Question: Can you design a data structure with the following running time?
    - Search: $O(1)$
    - Insert: $O(1)$
    - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?
    - Wastage of space.

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?
  - Wastage of space.
- Question: How do we fix this issue?

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?
  - Wastage of space.
- Question: How do we fix this issue?
  - Use array $A[0...N-1]$ and store an entry with key k at $A[h(k)]$, where $h : K \rightarrow \{0, ..., N-1\}$, where K denote the space of keys.

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?
  - Wastage of space.
- Question: How do we fix this issue?
  - Use array $A[0...N-1]$ and store an entry with key k at $A[h(k)]$, where $h : K \rightarrow \{0, ..., N-1\}$, where K denote the space of keys.
- Question: What is the new issue raised by the above idea?

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?
  - Wastage of space.
- Question: How do we fix this issue?
  - Use array $A[0...N-1]$ and store an entry with key k at $A[h(k)]$, where $h : K \rightarrow \{0, ..., N-1\}$, where K denote the space of keys.
- Question: What is the new issue raised by the above idea?
  - Collision: There may exist keys $i \neq j$ such that $h(i) = h(j)$.

Data Structures
Hashing

- Question: Can you design a data structure with the following running time?
  - Search: $O(1)$
  - Insert: $O(1)$
  - Delete: $O(1)$
- Suppose for the sake of simplicity, all keys are positive integers.
- Main idea: Use an array indexed by the keys and store the entry with key i at $A[i]$.
- Question: What is the main issue with this idea?
  - Wastage of space.
- Question: How do we fix this issue?
  - Use array $A[0...N-1]$ and store an entry with key k at $A[h(k)]$, where $h : K \rightarrow \{0, ..., N-1\}$, where K denote the space of keys.
- Question: What is the new issue raised by the above idea?
  - Collision: There may exist keys $i \neq j$ such that $h(i) = h(j)$.
- Question: How do we avoid collisions (as much as possible)?
- Question: How do we resolve collisions?

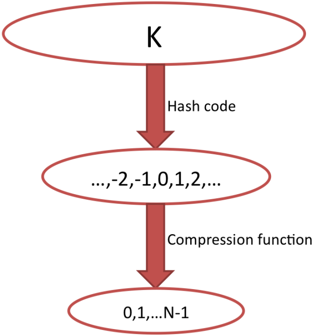
- Main idea: Use array $A[0...N-1]$ and store an entry with key k at $A[h(k)]$, where $h : K \rightarrow \{0, ..., N-1\}$, where K denote the space of keys.
- Question 1: How do we avoid collisions (as much as possible)?
- Question 2: How do we resolve collisions?

- Question 1: How do we avoid collisions (as much as possible)?
- The nature of keys of entries may be varied depending on the context (*it may not always be positive integer as we assumed*):
  - In case of school records, the key may be the identification number of students.
  - In case of file system, it may be the file identifier.
  - In case of photograph storage, it may be the photos itself.

- Question 1: How do we avoid collisions (as much as possible)?
- The nature of keys of entries may be varied depending on the context (*it may not always be positive integer as we assumed*):
  - In case of school records, the key may be the identification number of students.
  - In case of file system, it may be the file identifier.
  - In case of photograph storage, it may be the photos itself.
- Let K denote the space of keys. K depends on the context.
- It would be be good idea to first map the keys to integers. That is a function $f : K \rightarrow \mathbb{Z}$.
- Such a mapping from keys to integers is known as a hash code.
- We will then use a mapping from the set of integers to the set $\{0, ..., N-1\}$. That is $g : \mathbb{Z} \rightarrow \{0, ..., N-1\}$.
- Such a mapping is called a compression function.

- Question 1: How do we avoid collisions (as much as possible)?
- The nature of keys of entries may be varied depending on the context (*it may not always be positive integer as we assumed*):
  - In case of school records, the key may be the identification number of students.
  - In case of file system, it may be the file identifier.
  - In case of photograph storage, it may be the photos itself.
- Let K denote the space of keys. K depends on the context.
- It would be be good idea to first map the keys to integers. That is a function $f : K \rightarrow \mathbb{Z}$.
- Such a mapping from keys to integers is known as a hash code.
- We will then use a mapping from the set of integers to the set $\{0, ..., N-1\}$. That is $g : \mathbb{Z} \rightarrow \{0, ..., N-1\}$.
- Such a mapping is called a compression function.
- Given hash code f and compression function g, the hash function $h : K \rightarrow \{0, ..., N-1\}$ is given by $h(k) = g(f(k))$.

- Question 1: How do we avoid collisions (as much as possible)?
- Given hash code f and compression function g, the hash function $h : K \rightarrow \{0, ..., N-1\}$ is given by $h(k) = g(f(k))$.



- The hash code f should be such that it avoids collisions (Note that this depends on the context).

- Question 1: How do we avoid collisions (as much as possible)?
- The hash code f should be such that it avoids collisions (Note that this depends on the context).
- Some examples of hash codes:
  - Bit representation as integer: Any key will have a bit representation $(x_{n-1}, ..., x_0)$. Use the integer value of this bit representation as the hash code. That is:

$$f(x_{n-1}, ..., x_0) = \sum_{i=0}^{n-1} x_i \cdot 2^i$$

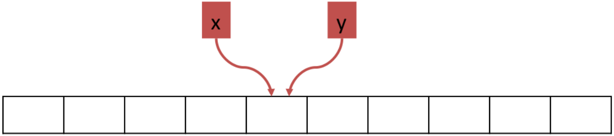
  - Sum of ASCII codes: Given that the keys are sequence of strings sum the ASCII values of each of the characters. Can you point some issues with this code?
  - Polynomial code: For a constant $a \neq 0, 1$ use:

$$f(x_{n-1}, ..., x_0) = \sum_{i=0}^{n-1} x_i \cdot a^i$$

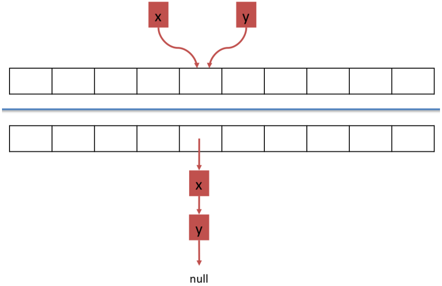
- Question 1: How do we avoid collisions (as much as possible)?
  - Using carefully chosen hash functions.
- The hash code f should be such that it avoids collisions (Note that this depends on the context).
- Some examples of hash codes:
  - Bit representation as integer
  - Sum of ASCII codes
  - Polynomial code
- Some examples of compression functions:
  - Division method: $g(i) = i \; mod \; N$
  - MAD method: $g(i) = [(ai + b) mod \; p] mod \; N$ for some carefully chosen constants a, b, p.

- Question 1: How do we avoid collisions (as much as possible)?
    - Using carefully chosen hash functions.
- Even though we carefully chose the hash function, collisions may still happen since the cardinality of the key space K is usually much larger than N.

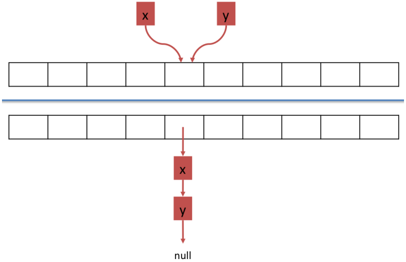
- Question 2: How do we resolve collisions?
- Suppose we are using an array $A[0, ..., N-1]$ and using a hash function h.
- Suppose we need to insert two keys x, y such that $h(x) = h(y) = i$. As per our scheme both these keys should go to array location $A[i]$. Can you think of a way to resolve this?



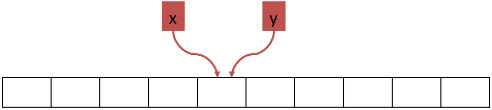
- Question 2: How do we resolve collisions?
- Suppose we are using an array $A[0, ..., N-1]$ and using a hash function h.
- Suppose we need to insert two keys x, y such that $h(x) = h(y) = i$. As per our scheme both these keys should go to array location $A[i]$. Can you think of a way to resolve this?
    - Create a link list of all entries that map to the same array location.
    - This is called separate chaining.



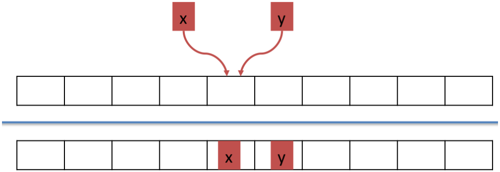
- Question 2: How do we resolve collisions?
- Suppose we are using an array $A[0, ..., N-1]$ and using a hash function h.
- Suppose we need to insert two keys x, y such that $h(x) = h(y) = i$. As per our scheme both these keys should go to array location $A[i]$. Can you think of a way to resolve this?
  - Create a link list of all entries that map to the same array location.
  - This is called separate chaining.
  - One disadvantage of this scheme is that an auxiliary data structure of required.



- Question 2: How do we resolve collisions?
    - Separate chaining.
- Suppose we are using an array $A[0, ..., N-1]$ and using a hash function h. Furthermore, we would like to use only A for storage and access.
- Suppose we need to insert two keys x, y such that $h(x) = h(y) = i$. As per our scheme both these keys should go to array location $A[i]$. Can you think of a way to resolve this?



- Question 2: How do we resolve collisions?
    - Separate chaining.
- Suppose we are using an array $A[0, ..., N-1]$ and using a hash function h. Furthermore, we would like to use only A for storage and access.
- Suppose we need to insert two keys x, y such that $h(x) = h(y) = i$. As per our scheme both these keys should go to array location $A[i]$. Can you think of a way to resolve this?
    - Insert the elements into the next available array slot.
    - This idea is known as open addressing.



- Question 2: How do we resolve collisions?
  - Separate chaining.
  - Open addressing:
    - Linear probing: The sequence of locations probed for key k are given by $A[(h(k) + i) mod\ N]$ for $i = 0, 1, ...$
    - Quadratic probing: The sequence of locations probed for key k are given by $A[(h(k) + f(i)) mod\ N]$ for $i = 0, 1, ...$, where f is a quadratic function such as $f(i) = i^2$.

- Main idea: Use array $A[0...N-1]$ and store an entry with key k at $A[h(k)]$, where $h : K \rightarrow \{0, ..., N-1\}$, where K denote the space of keys.
- Question 1: How do we avoid collisions (as much as possible)?
  - Use a good hash function.
- Question 2: How do we resolve collisions?
  - Separate chaining
  - Open addressing
- Given that the number of entries in the hash table is at most n, the load factor λ is defined as $\lambda = n/N$. N is chosen so as to have the load factor $\lambda < 1$.
- Note that if the hash function uniformly distributes the entries into the table, then there will be $\lceil \lambda \rceil$ entries that map to each of the table locations.
- Under such favourable circumstances, the running time for all operations will be $O(1)$ given that λ is a constant.

End