Hashing (III): Open Addressing

CSCD 300 – Data Structures

Eastern Washington University

Goal

In this lecture, we will discuss another mechanism called open addressing to solve the hashing collisions.

Outline

Open addressing

In open addressing, if an element is hashed to a table location which is already occupied by another element, the hash function will then try another table location [1] until it finds a table location where there is no element, unless the whole hash table is already full.

What does that mean ?

- Each table location can have no more than one element.
- The number of elements $\leq$ the hash table size.
- The sequence of the table positions probed by the hash function should be a permutation of all the hash table positions.
- We expect each permutation used in the probing procedure has equal probability — uniform hashing.
- It is space-saving, because open addressing saves space from avoiding pointers which are used in the hashing with chains.

[1] The hash function used in open addressing is not just a fixed function, as we will show later.

Hashing functions for open addressing

Unfortunately, uniform hashing for open addressing is difficult to implement, so next we are to introduce linear probing, which is the simplest type of hash function for open addressing, to compute the probe sequence for open addressing. Note that linear probing is not uniform hashing [2].

In the rest of the lecture:

- U denotes the universe where element id's are drawn from.
- m denotes the hash table size.

[2] If you want to know why, I will be happy to discuss it off the lecture

Linear probing

For $i = 0, 1, \ldots, m-1$,

$$h(k, i) = (h'(k) + i) \mod m$$

where $h' : U \to \{0, 1, \ldots, m-1\}$ is a normal hash function.

Over $i = 0, 1, \ldots, m-1$, function $h()$ is to try and find the "first" empty array location where the given key k (and its associated value) will be saved.

Operations

Let $\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$ denote the permutation that probing procedure will follow for a given key k using a hash function h.

```
Open_addr_search(T,k)
   for i = 0 ... m-1
      j = h(k,i)
      if T[j] == k
         return j
      if T[j] == NIL
         return NIL
   return NIL
```

```
//assume we have checked that k does not exist
Open_addr_insert(T,k)
   for i = 0 ... m-1
      j = h(k,i)
      if T[j] == NIL or T[j] == "DELETED"
         T[j] = k
         return j
   return "table overflow"
```

```
Open_addr_delete(T,k)
   p = Open_addr_search(T,k)
   if p != NIL
      T[p] = "DELETED"
```

Questions and comments

- Why DON'T we set NIL to $T[p]$ in the delete operation ?
- Because of this "DELETE" assignment, search performance of open addressing can be worse than hashing with chaining, if there are many delete operations on the hash table. (why ?)