Searching Tables

Table: sequence of (*key,info*) pairs

- (*key,info*) pair is a record
- *key* uniquely identifies *info*, so no duplicate records
- Sometimes the key is the whole record

Searching a table

- Given a key k and a table T = $(k_1,i_1),\ldots,(k_n,i_n)$, find the pair (k_j,i_j) in T such that $k=k_j$ (if it exists)
- Possible approaches:
  - Sequential search: simple, but only effective for small tables
  - Binary search: fast, but table must be sorted
  - Hashing

Hash Tables

Idea: use a function such that for every possible key k, f(k) = index of record with key k: O(1) time

Hash Function: maps keys to addresses

- Build the table using the hash function: if f(k)=1 then put record (k,i) in cell 1 of table (etc)
- Hash functions should ideally be:
  1. Easy to compute, and
  2. Ensure different keys are always mapped to different cells
- Perfect hash functions are not always possible

Hashing and Collisions

Collision: the effect of more than one key being mapped to the same cell

- Given $k_x \neq k_y$, we have $f(k_x) = f(k_y)$

Approaches to dealing with collisions:

1. Allow >1 record to be stored in each table index
   - Separate chaining: each index has a linked list of records
   - Buckets: each index is a fixed-size bucket of records
2. Open addressing: allow only 1 record at each index
   - When a collision occurs, use a *collision resolution policy* to find a new index for the item, e.g. linear probing etc.

Hash table
`findPos` – Linear Probing

```
int findPos(int k)
// search for index that should store
// record with key k
{
  current = hash(k, tableSize);
  while(array[current] != null &&
     array[currentPos].record.key != k)
  {
    current = (current+1) % tableSize;
  }
  return current;
}
```

Hash Table search

```
int find(int k)
// search for record with key k
{
 current = findPos(k);
 if(isActive(current))
  return current;
 else
  return -1;          // not found
}
```

Hash Table – insertion

```
void insert(record R)
// insert R if not already in table
{
  current = findPos(R.key);
  if(isActive(current))
  // already in hash table
    return;
  else
  {
    array[current].record = R;
    array[current].isActive = true;
  }
}
```

Hash table – deletion

```
void remove(int k)
// delete record with key k
{
  current = findPos(R.key);
  if(isActive(currentPos))
    array[current].isActive = false;
}
```