Hash Tables Algorithms

Hash Tables

- A data structure that offers very fast insertion & searching.
- Hash tables are faster than trees.
- Hash tables are relatively easy to program.

Applications

Hash Tables

- **Dictionary**
- **Database Indexes**
- **Sets**
- **Object representation**
- **Compiler**

Hash Function

Hash Tables

- One important concept is how a range of key values is transformed into a rang of array index values.
- In a hash table this is accomplished with a hash functions.

Hash Tables

- Let's say we want to store a 50,000 - word English-language dictionary in main memory.
- But what is the relationship of these index numbers to the words?
- **Adding the Digits!!!**
  - Zzzzzzzzz:
  - 26 + 26 + 26 + 26 + 26 + 26 + 26 + 26 + 26 + 26 = 260

Hash Tables

- **Multiplying by powers:**
- Say we want to convert the word Data to a number.
- $4 * 27^3 + 1 * 27^2 + 20 * 27^1 + 1 * 27^0 = 60,337$
- This process does indeed generate a unique number for every potential word.
- We just calculated a 4-letter word. What happens with larger words? Unfortunately, the range of numbers becomes rather large. The largest 10-letter word, zzzzzzzzzz , translates into:
- $26 * 27^9 + 26 * 27^8 \ldots\ldots\ldots\ldots + 26 * 27^0$
- Just by itself, 27^9 is more than 7,000,000,000,000, so you can see that the sum will be huge.

Hash Tables

- **Hashing:**
- hashvalue = LargeNumber % SmallIrange;
- arrayIndex = hugeNumber % arraySize;
- This is an example of hash function.
- **Collisions**

Hash Tables

```
public void insert(int item)
  {
     int key = item;
     int hashVal = hashFunc(key);
     while(hashArray[hashVal] != 0 &&
  hashArray[hashVal] != -1)
     {
        ++hashVal;
        hashVal %= arraySize;
     }
     hashArray[hashVal] = item;
  }
```

```
public int
hashFunc(int key)
   {
      return key %
arraySize;
   }
```

Hash Tables

```
public int delete(int key)
   {
      int hashVal = hashFunc(key);
      while(hashArray[hashVal] != 0)
      {
         if(hashArray[hashVal] == key)
         {
            int temp = hashArray[hashVal];
            hashArray[hashVal] = nonItem;
            return temp;
         }
         ++hashVal;
         hashVal %= arraySize;
      }
      return 0;
   }
```

Hash Tables

```
public int find(int key)  {
        int hashVal = hashFunc(key);
        while(hashArray[hashVal] != 0) {
            if(hashArray[hashVal] == key)
                return hashArray[hashVal];
            ++hashVal;
            hashVal %= arraySize;
        }
        return 0;
    }
}
```

Hash Tables

```
class HashTableApp {
 public static void main(String[] args) {
      HashTable theHashTable = new HashTable(100);
      theHashTable.insert(45);
       theHashTable.insert(26);
       theHashTable.insert(85);
       theHashTable.insert(123);
       theHashTable.insert(26);
       theHashTable.insert(4805);
       theHashTable.insert(45);
       }
   }
```