Hashing

Perfect Hash Functions

- All the hash functions we have considered up to now allow for multiple keys to hash to the same index
  - not only did we have to deal with this problem
  - but it cost us performance as well
- The reason for collisions is because we assumed we knew little about the values of keys

Perfect Hash Functions

- If we know something about the keys, it's possible to write a hash function that will never have collisions
  - this is called a *perfect hash function*
- For example,
  - if you had a class with exactly 100 students and each was given a 2 digit ID, your hash function could simply be to use the students ID number to index into the table

Perfect Hash Function

- The previous example was very simple
  - and not going to be very common
- A real world example
  - compilers need to check for reserved words
  - there are a limited number of reserved words
    - C has about 32; Java about 50
  - it is possible to examine each word and assign it a unique value
    - the performance penalty for this is small because n is small
    - if you were doing this for the entire dictionary, it would be much more time consuming

Cichelli's Algorithm

- *Cichelli's Algorithm* is a commonly used solution to the compiler problem
  - before Cichelli, a binary search was used
- Basic idea
  - assign a value to each letter appearing at the beginning and at the end of each key word
    - this is called a g-value
  - then use the following hash function
    - h(word) = (length(word) + g(firstletter) + g(lastletter)) % size

Cichelli’s Algorithm

- The real trick is to assign the g-values
  - guess the value of the first and last letter of the first word
  - compute the first word’s hash value and reserve it
  - guess the value of the first and last letter of the second word
    - if either letter has already been assigned a g-value, do not assign it a new one – use the assigned value
  - compute the second word’s hash value and reserve it
    - if it collides with the first’s hash value, make two new guesses
  - repeat this process until all words have a unique hash value

Psuedo-Code

```
// count the frequency that each letter appears as a first or last letter
// order the words by their frequency values – highest value first
    // frequency value = freq(first) + freq(last)
// pick a maxValue – usually the number of words divided by 2
boolean cichelli(Stack wordStack) {
  while(!wordStack.isEmpty()) {
    // pop the first word from wordStack
    if( // both first and last letter have been assigned g-values ) {
          if( // hash value for  word is valid ) {
            // assign hash value to word
            if( // recursive call to cichelli() returns true ) { return true; }
            else { detach the hash value for word }
          }
          // push word back on top of wordStack and return false
    }
```

Psuedo-Code (continued)

```
else if( // neither letter assigned g-value AND first != last letter ) {
  // for every value of m and n from 0 to maxValue {
      // assign first letter the g-value of m and second letter gets n
      if( // hash value for  word is valid ) {
        // assign hash value to word
        if( // recursive call to cichelli() returns true ) { return true; }
        else { detach the hash value for word }
      }
  }
  // reset g-value for letters so they are unassigned
  // push word back on top of wordStack and return false
}
```

Psuedo-Code

```
else {   // only one letter assigned g-value OR first = last letter
   // for every value of m from 0 to maxValue {
        // give unassigned letter the g-value of m
        if( // hash value for  word is valid ) {
           // assign hash value to word
           if( // recursive call to cichelli() returns true ) { return true; }
           else { detach the hash value for word }
        }
   }
   // reset g-value for letter so it is unassigned
   // push word back on top of wordStack and return false
 }
}   // end of  while(!wordStack.isEmpty())
return true;   // empty stack means we have a solution
}
```

Example

- Consider the following list of states
  - Alabama, Maine, Montana, Nevada, Idaho
- Step one, find frequencies (case insensitive)
  - a: 4;   m: 2;   n: 1;   e: 1;   i: 1;   o: 1
- Step two, order words based on frequency
  - Alabama-8, Montana-6, Maine-3, Nevada-3, Idaho-2
- Step three, pick a max value
  - maxValue = 4 / 2 = 2

Example

- Step 4, call cichelli()

```
Alabama:    a = 0,   h = 2     hash values -> { 2 }
Montana:    m = 0,   h = 2     hash values -> { 2 }
  Montana:  m = 1,   h = 3     hash values -> { 2, 3 }
Nevada:     n = 0,   h = 1     hash values -> { 1, 2, 3 }
Maine:      e = 0,   h = 1     hash values -> { 1, 2, 3 }
  Maine:    e = 1,   h = 2     hash values -> { 1, 2, 3 }
    Maine:  e = 2,   h = 3     hash values -> { 1, 2, 3 }
  Nevada:   n = 1,   h = 2     hash values -> { 2, 3 }
    Nevada: n = 2,   h = 3     hash values -> { 2, 3 }
    Montana: m=2,    h = 4     hash values -> { 2, 4 }
Nevada:     n = 0,   h = 1     hash values -> { 1, 2, 4 }
Maine:      e = 0,   h = 2     hash values -> { 1, 2, 4 }
  Maine:    e = 1,   h = 3     hash values -> { 1, 2, 3, 4 }
Idaho:      i=0, o=0, h=0      hash values -> { 0, 1, 2, 3, 4 }
```

Concerns

- Picking a maxValue is not always easy
  - what if the previous example had used 1?
    - no solution would have been found
  - if this happens, just pick a larger maxValue and try again
- Even with a large maxValue, may not always find a solution
  - if two words are the same length and have the same first and last letter, no solution
  - consider “brick” and “block”
    - no matter what, they will always hash to the same value