CS 306 **Algorithms and Complexity**

Goodness

Iterative Improvement *Exploration*

Goal

The purpose of this exploration is for you to investigate and implement an important iterative improvement algorithm.

Requirements

Using the supplied article by Shawn Carlson as a springboard, do some good, collaborative research on *simulated annealing*, and then implement this iterative improvement algorithm. Apply your implementation to one specific search task: Validate the choices described by the following excerpts from the documentation and source code of `java.util.HashMap`.

```
 * This implementation provides constant-time performance for the basic
 * operations (get and put), assuming the hash function
 * disperses the elements properly among the buckets.

/**
  * Returns a hash value for the specified object.  In addition to
  * the object's own hashCode, this method applies a "supplemental
  * hash function," which defends against poor quality hash functions.
  * This is critical because HashMap uses power-of-two length
  * hash tables.
  *
  * The shift distances in this function were chosen as the result
  * of an automated search over the entire four-dimensional search space.
  */
 static int hash(Object x)
 {
     int h = x.hashCode();

     h += ~(h << 9);
     h ^=  (h >>> 14);
     h +=  (h << 4);
     h ^=  (h >>> 10);

     return h;
 }
```

A newer version of the `hash` function uses different shift distances, the same shift operator (>>>) for each shift, and a uniform combining of the shifted values:

```
   /**
     * Applies a supplemental hash function to a given hashCode, which defends
     * against poor quality hash functions. This is critical because HashMap
     * uses power-of-two length hash tables, that otherwise encounter collisions
     * for hashCodes that do not differ in lower bits.
     * Note: Null keys always map to hash 0, thus index 0.
     */
    static int hash(int h)
    {
       // This function ensures that hashCodes that differ only by
       // constant multiples at each bit position have a bounded
       // number of collisions (approximately 8 at default load factor).
       h ^= (h >>> 20) ^ (h >>> 12);
       return h ^ (h >>> 7) ^ (h >>> 4);
    }
```

Both versions use the same `indexFor` function:

```
    /**
     * Returns index for hash code h.
     */
    static int indexFor(int h, int length)
    {
       return h & (length-1);
    }
```

Use the supplied C++ sample code (`/home/cs306/samplecode/ch10/goodness.cpp`) as a guide and a starting point. This means that you will use C++ as your implementation language. Note that C++ does not have a '>>>' operator. Note too that Java does not support unsigned types!

Use the later version of the `hash` function from the above Java source code. Use the words found in the Linux Lab in the file `/usr/share/dict/words` as the elements to be hashed. Note that you can prehash these into `hashCodes` (which are numbers) instead of converting them each time you apply the `hash` function to test your objective function.

If conditions are right, you can build and submit your code in the Linux Lab via the command(s):

```
make it so
```

You must determine which test(s) to run.

Grading Criteria

Grading for this exploration, generally speaking, is based on application and engagement (whether or not or to what extent you figured out the hows and the whys), correctness and completeness, and elegance. Details are in the self-assessment.

Comments

- I have to say, I found this exploration a bit more of a challenge than I expected. It's such an open-ended project, with so many parameters, it was hard to know where to start. I think the benefit to this, however, is that we are free to explore without too many constraints.
- When I first glanced at this exploration, I thought it was going to be really complicated. I read the article and it didn't really make sense at first. It was interesting to see that the subject was really clear to me only after we were done programming it.
- Understanding the simulated annealing algorithm came with much difficulty and I am still not sure that we got it 100% correct.
- Overall this was very interesting. I really enjoyed working on this exploration. It was a lot of work, but I felt very rewarded for all of the work. It was also cool having to do a lot of fine tuning in C++ in order to make this usable for testing. I learned a ton even though it was learning via the fire hose.
- Knowing that time was an issue here, not space, [I] decided to implement a standard brute force search and one using dynamic programming for comparison purposes. [Both] produced the same results, but [..] the time trade-off in using dynamic programming was outstanding. I always try to think of good examples of whenever I'd use material I learn in this class, and this is a perfect one. I am interested in learning the concepts, but when I see them in action, it really solidifies the importance of using these principles.