**Assignment 4**
**50 Points**

**Problem 1.** Consider hash tables with collision resolved by chaining, implemented as vector-of-lists, as in `fsu::HashTable<K,D,H>`. Show that the standard traversal has run-time $\Theta(b + n)$, where $b$ is the number of buckets and $n$ is the size of the table. Use the context and notation established on the next page. (Hint: use aggregate analysis.)

**Problem 2.** Consider the `Partition` data structure implementing the Union/Find disjoint sets algorithms. Let $T$ be any tree in the forest, and denote the rank of $T$ by $d$ and the number of elements of the set represented by $T$ by $k$. Show that $d \leq \log_2 k$. (Hint: Use mathematical induction on $d$. For the inductive step, examine the tree of rank $d$ with the fewest number of nodes.)

**Problem 3.** Consider the family of rectangular mazes described in Disjoint Sets Appendix: Maze Technology.

(a) Devise an algorithm that translates a 2-D maze of square cells into a graph whose characteristics reflect all properties of the maze. For example, a path in the graph would correspond to a path in the maze. (We'll refer to this translation as an *isomorphism*.)
(b) Describe in more general terms how the isomorphism would generalize to 2-D mazes of cells of other shapes, such as hexagonal, or variable shape as long as the shapes are polygons. (E.g., any tile floor would do.)
(c) Based on the technology for 2-D mazes of square cells, invent maze technology for describing 3-D mazes of cubical cells. How would the isomorphism generalize to this case?

```
// standard traversal of HashTable t:
for (HashTable::Iterator i = t.Begin(); i != t.End(); ++i)

// HashTable and HashTableIterator context:
class HashTable
{
public:
  typedef HashTableIterator Iterator;
  Iterator Begin();
  Iterator End();   // known to be constant time
  ...
private:
  Vector<List> v;   // vector of lists (bucket vector)
};

class HashTableIterator
{
public:
  typedef HashTableIterator Iterator;
  Iterator& operator++();
  ...
private:
  unsigned     vi;  // vector index
  ListIterator li;  // bucket iterator
};

// algorithms used in traversal:
HashTableIterator HashTable::Begin()
{
  Iterator i;
  i.vi = 0;                    // start at 0th bucket
  while (v[i.vi].Empty())      // while bucket is empty
    ++i.vi;                      // go to next bucket
  i.li = v[i.vi].Begin();      // start at beginning of this bucket
  return i;                    // NOTE: Begin() == End() for an empty bucket
}

HashTableIterator& HashTableIterator::operator++()
{
  ++li;                        // go to next item in bucket
  if (li == v[vi].End())       // if at end of bucket
  {
    do
      ++vi;                    // go to next bucket
    while (v[vi].Empty());     // until bucket is not empty
    li = v[vi].Begin();        // start at beginning of this bucket
  }
  return *this;                // NOTE: Begin() == End() for an empty bucket
}
```