Data Structure and Advanced Programming

Homework #12
Due: 2020/6/23 08:00am (CST)

NOTE: Please upload your answers in either English or Chinese as a PDF to NTU COOL before the due date and time.

1. (20%) Imagine an application program that behaves like an English dictionary. The user types a word and the program provides the word's definition. Thus, the dictionary needs only a retrieval operation. Which implementation of the ADT dictionary would be most efficient as an English dictionary? Please consider a sorted array-based implementation, a sorted linked-based implementation, an implementation that uses a binary search tree, and an implementation that uses hashing, respectively.

2. (40%) The success of a hash-table implementation of the ADT dictionary is related to the choice of a good hash function. A good hash function is one that is easy to compute and will evenly distribute the possible data. **Comment on the appropriateness of the following hash functions. What patterns would hash to the same location?**

   a. The hash table has size 2,048. The search keys are English words.
   The hash function is

   $h(key)$ = (Sum of positions in alphabet of *key*'s letters) mod 2048

   b. The hash table has size 2,048. The keys are strings that begin with a letter.
   The hash function is

   $h(key)$ = (position in alphabet of first letters *key*) mod 2048

   Thus, "BUT" maps to 2. **How appropriate is this hash function if the strings are random? What if the strings are English words?**

   c. The hash table is 10,000 entries long. The search keys are integers in the range 0 through 9999. The hash function is

   $h(key)$ = (*key* * random) truncated to an integer

   where random represents a sophisticated random-number generator that returns a real value between 0 and 1.

d. The hash table is 10,000 entries long (HASH_TABLE_SIZE is 10000). The search keys are integers in the range 0 through 9999. The hash function is given by the following C++ function:

```
int hashIndex(int x) {
    for (int i = 1; i <= 1000000; i++)
    x = (x * x) % HASH_TABLE_SIZE;
    return x;
} // end hashIndex
```

3. (40%) For graph in the right figure, draw its adjacency matrix, DFS and BFS spanning trees rooted at a, and the minimum spanning tree rooted at a.