CS261 Assignment 7

Eric Rouse

1. The simplest version is easier to understand, that would be the first version. It does the same thing every time. The second version is more efficient and faster, but harder to understand. Here is the version that puts any new value into the map (dictionary) at the pre-existing key.

```
//if word is not null, iterate through all the words in the file.
while(word != NULL){
  insertMap(&hashTable, word, newValue);
  //iterate
  word = getWord(file);
}
```

   Here is the version that puts any new value into the map (dictionary) at the already existing key.

```
//if word is not null, iterate through all the words in the file.
while(word != NULL){
  int * value;
  //if the word is found, increase the count by 1
  if (containsKey(&hashTable, word) == 1){
        count = atMap(&hashTable, word);
        *value += newValue;
  }
  //otherwise insert a new word with a count of 1
  else insertMap(&hashTable, word, newValue);
  //iterate
  word = getWord(file);
}
```

2. She would have to rehash all the other names in her group. And that still doesn't guarantee that Alan would get his own group, or that some other collision would not occure. In fact, he would hash to 0 in a table of 7, and he would be alone there. But Amy and Andy would both has to 3.
3.
   a. Abel: 4, Abigail: 2, Abraham: 0, Ada: 0, Adam: 0, Adrian: 5, Adrienne: 5, Agnes: 1, Albert: 1, Alex: 4, Alfred: 5, Alice: 2, Amanda: 0, Amy: 0, Andrew: 4, Andy: 3, Angela: 0, Anita: 2, Anne: 1, Antonia: 1, Arnold: 1, Arthur: 1, Audrey: 3
   b. For table size of 6, load factor is 24/6 or 4.
4. Math.sin(value) returns negative values, as sine fluctuates between -1 and 1. Thus, half of his output is invalid as negative values do not correspond to proper hashes. It would be better to use 1 + (int) Math.sin(value). The output is also cyclical between 0 and pi*value. So collisions would occur pretty frequently.
5. Both can be solved by assigning some value to the first letter of the thing and adding the length.

```
#include <assert.h>
#include <stdio.h>
```

```c
#include <stdlib.h>
#include <time.h>

int weekdayHash(char * str)
{

    int length = 0;
    char first = str[0];
    int r = 0;
    while (str[length] != '\0'){
       length++;
    }

    if (first == 'm') r = -1;
    else if (first == 't') r = -4;
    else if (first == 'w') r = -3;
    else if (first == 'f') r = -5;
    else if (first == 's') r = -6;

    r+=length;
    //printf("%s, %d \n", str, r);
    return r%10;
}

int monthHash(char * str)
{

    int length = 0;
    char first = str[0];
    char last;
    int r = 0;
    while (str[length] != '\0'){
       length++;
    }

    if (first == 'j') r = -1;
    else if (first == 'f') r = -4;
    else if (first == 'm') r = 6;
    else if (first == 'a') r = 3;
    else if (first == 's') r = -10;
    else if (first == 'o') r = -6;
    else if (first == 'n') r = -4;
    else if (first == 'd') r = -6;

    last = str[length-1];
    if (last == 'e') r+=11;
    if (last == 't') r+=1;
    r += length;
    //printf("%s, %d \n", str, r);
    return r%15;
}

int main (int argc, const char * argv[]) {
      //day test
      char mon[] = "monday";
```

```
    char tue[] = "tuesday";
    char wed[] = "wednesday";
    char thu[] = "thursday";
    char fri[] = "friday";
    char sat[] = "saturday";
    char sun[] = "sunday";

    //month test
    char jan[] = "january";
    char feb[] = "february";
    char mar[] = "march";
    char apr[] = "april";
    char may[] = "may";
    char jun[] = "june";
    char jul[] = "july";
    char aug[] = "august";
    char sep[] = "septemeber";
    char oct[] = "october";
    char nov[] = "novemeber";
    char dec[] = "december";

    printf("%d, %d, %d, %d, %d, %d, %d\n",
                weekdayHash(mon), weekdayHash(tue), weekdayHash(wed),
                weekdayHash(thu), weekdayHash(fri), weekdayHash(sat),
                weekdayHash(sun));

    printf("%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d\n",
                        monthHash(jan), monthHash(feb), monthHash(mar),
                        monthHash(apr), monthHash(may), monthHash(jun),
                        monthHash(jul), monthHash(aug), monthHash(sep),
                        monthHash(oct), monthHash(nov), monthHash(dec));
}
```

6. Adjacency Matrix:

N	1	2	3	4	5	6	7	8
1	1	1	0	1	0	0	0	0
2	0	1	3	0	0	0	0	0
3	0	0	1	0	1	1	1	1
4	0	0	0	1	1	0	1	0
5	0	0	0	0	1	0	1	0
6	0	0	0	0	0	1	1	1
7	0	0	0	0	1	0	1	0
8	0	0	0	0	0	0	0	1

Node List:

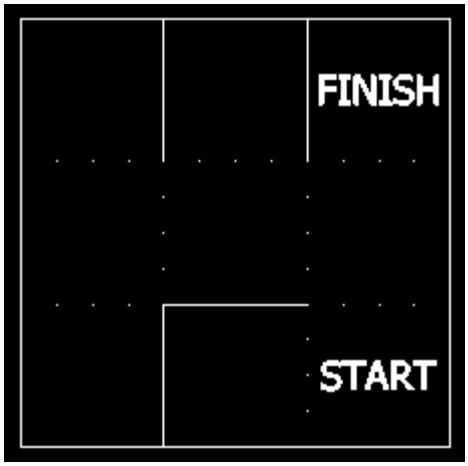
1:{2,4}

2:{3}

3:{5,6}

4:{5}
5:{}
6:{7,8}
7:{5}
8:{}

7. The following will solve in will solve in DFS before BFS. Up is the first choice, left 2nd, right 3rd and down last.



8. From WS 41:
   Depth first search (stack version)
   r: 1,6,11,16,21,22,23,17,12,13,18,8,3,4,9,14,15,20,19,24,25
   Stack: 2, 12, 5,10

   Breadth first search (queue version)
   r: 1,2,6,3,7,11,4,8,12,16,5,9,13,17,21,10,14,18,22,15,19,23,20,24,25
   Queue: empty at completion
9. From WS 42:

	Pensacola:0
Pensacola: 0	Phoenix: 5
Phoenix: 5	Pueblo: 8, Peoria: 9
Pueblo: 8	Peoria: 9, Pierre: 11
Peoria: 9	Pierre: 11, Pueblo: 12, Pittsburgh: 14
Pierre: 11	Pueblo: 12, Pendleton: 13, Pittsburgh: 14
-	Pendleton: 13, Pittsburgh: 14
Pendleton: 13	Pittsburgh: 14, Phoenix: 17, Pueblo: 21
Pittsburgh: 14	Phoenix: 17, Pensacola: 18, Pueblo: 21
-	Pensacola: 18, Pueblo: 21
-	Pueblo: 21
-	{}

10. The priority queue automatically finds the shortest path by putting the shortest distance to a given node at the top of the queue. Because it is possible to reach a single node multiple ways.
11. O(e), assuming e is edges and v is vertexes and e > v.
12. $O(v^2)$
13. BFS will always find the answer as it propogates forward as a wave. Thus it works multiple path possibilities at once. It doesn't get lost in an infinite corridor like DFS. DFS relies on working a single path to failure. If that path is infinite in length, it will not find the end, it won't fail and it won't try a different path. It will be lost.