**Hash Tables**

**1. In the dynamic array implementation of the dictionary, the put operation removes any prior association with the given key before inserting a new association. An alternative would be to search the list of associations, and if one is found simply replace the value field. If no association is found, then insert a new association. Write the put method using both approaches. Which is easier to understand? Which is likely to be faster?**

With replacement:

void put(struct dynaArray \*d, KeyType k, ValueType v){

struct Association \*a;

if(dynaArrayContains(v,k))

dynaArrayRemoveKey(v,k);

a = malloc(sizeof(struct association));

assert(a != 0);

a->key = k;

a->value = v;

dynaArrayAdd(v,a);

}

With value update:

void put(struct dynaArray \*d, KeyType k, ValueType v){

int i;

for(i = 0; I < d->size; ++i){

if(d->data[i]->key == k){ //key was found

d->data[i]->value = v;

}

}

//key not in table yet

struct Association \*a;

a = malloc(sizeof(struct association));

a->value = v;

a->key = k;

dynaArrayAdd(v,a);

}

The first implementation is easier to understand and shorter. The second one will run faster though. The first has to find the key, then remove the key which which could be O(n) in worst case. The second implementation just needs to find the key a single time and replace the value.

**2. Revisit Chapter 12 to read about Amy's hash table (page. 3-4). When Alan wishes to join the circle of six friends, why can't Amy simply increase the size of the vector from six to seven? From chapter 12, this is a description of Amy's hash function assuming an initial table size of 6:**

**"Amy uses an interesting fact. If she selects the third letter of each name, treating the letter as a number from 0 to 25, and then Mods the number by 6, each name yields a different number"**

There would be a collision between Andy and Amy which both hash to #2.

**3. Amy's club has grown, and now includes the following members:**

|  |  |  |
| --- | --- | --- |
| Abel | Abigai | |
| Adam | Adria | Agnes |
| Albert | Alex | Alfred |
| Aman | Amy | Andy |
|  |  |  |

1. **Find what value would be computed by Amy's hash function for each member of the group.**

b) **What is the load factor of the table?**

**4. In searching for a good hash function over the set of integer values, one student thought he could use the following: int index = (int) Math.sin(value); What was wrong with this choice?**

Math.sin() will return a numbers from -1 to 1. It will only give -1 or 1 for exact values that are multiples of 3pi/2 or pi/2 which none of the ascii values will be. So this function will always hash to 0.

1. **Can you come up with a perfect hash function for the names of the week? And the names of the months? Assume a table size of 11 for days of the week and 17 for names of the months. In case you cannot find any perfect hash functions, we will accept solutions that produce a small number of collisions (< 3).**

No collisions for days of the week.

int stringHashDays(char \* str)

{

int i;

int r = 0;

for (i = 0; str[i] != '\0'; i++)

r += (i+3) \* str[i];

return r;

}

This gives two collisions for the months:

int stringHashMonth(char \* str)

{

int i;

int r = 0;

for (i = 0; str[i] != '\0'; i++)

r += str[i] << i;

return r;

}

**Graphs**

**6. Describe the following graph as both an adjacency matrix and an edge list:**

**Graph (image provided with the assignment)**

**1 {2,4}**

**2 {3}**

**3{5,6}**

**4 {5}**

**5 {}**

**6 {7}**

**7 {5}**

**8 {}**

**7. Construct a graph in which a depth first search will uncover a solution (a path from one vertex to another) in fewer steps than will a breadth first search. You may need to specify an order in which neighbor vertices are visited. Construct another graph in which a breadth-first search will uncover a solution in fewer steps.**

To find node #5 from #1. A description...

If BFS is set to compare node values it will take 4 iterations to find #5 (1, 2, 3, 4, 5). DFS will do it in two iterations (1, 2, 5).

To find node #4 from #1:

DFS will take 4 iterations (1, 2, 4, 3, 4) and BFS will take 3 iterations (1, 2, 3, 4).

**8. Complete Worksheet 41 (2 simulations). Show the content of the stack, queue, and the set of reachable nodes. (Submit it with the assignment)**

**9. Complete Worksheet 42 (1 simulation). Show the content of the priority queue and the cities visited at each step. (Submit it with the assignment)**

**10. Why is it important that Dijkstra’s algorithm stores intermediate results in a priority queue, rather than in an ordinary stack or queue?**

**11. How much space does an edge-list representation of a graph require?**

**12. For a graph with V vertices, how much space will an adjacency matrix require?**

1. **Suppose you have a graph representing a maze that is infinite in size, but there is a finite path from the start to the finish. Is a depth first search guaranteed to find the path? Is a breadth-first search? Explain your answer.**

DFS is not guaranteed to find a path. It may find a path but it could also find a infinite path in which case it will never stop. BFS is guaranteed to find the finish since it will find all of the possible paths from the starting point out level by level.