# Design and Analysis of Algorithms

## Solution for Theory Assignment for Hashing

1. Modify the pseudo code for Dijkstra’s algorithm in the slides by saving for each node u, the last edge of the shortest special path from s to u, and write pseudo code for printing the shortest path from s to t. [10%]
2. Apply Prim's algorithm for finding a minimum spanning tree. Start with node a. Show the selected node in the first column of the table, show the current values of the D vector and the last edge of the path after each node is selected in the rest of the columns. [10%]



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Node added to solution tree | *D*(*a*) | *D*(*b*) | *D*(*c*) | *D*(*d*) |
| Initial *D* values |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

1. Assume a hash table where collisions are resolved by chaining with *m* slots and *n* stored keys.
   1. What is the worst-case time complexity for searching for a key? When does it occur?

**Answer:** The worst case time complexity is O(n). This occurs when O(n) keys are stored in one slot and the search is for a key that hashes to that slot.

* 1. What is the expected time for searching under the assumption of simple uniform hashing?

**Answer:** Under the assumption of simple uniform hashing the expected time is Θ(α + 1) where α is the load factor.

1. What is the expected time for searching under the assumption of simple uniform hashing when *n* = O(*m*)?

**Answer:** When *n* = O(*m*) the expected time is Θ(1) since α = O(*m*) / *m* = O(1).

1. What is a universal collection of hash functions? What is gained by randomly selecting a hash function from a universal set of hash functions?

**Answer:** A collection H of hash function is universal when the number of functions in H for which a pair of distinct keys collide is at most |H|/m. This means that the probability of a collision is at most (|H|/m)/|H|=1/m. So the expected search time is Theta(1+alpha). When n=O(m) where m is the number of chains the expected time for find, insert and delete becomes O(1).

When the number of possible keys is at least nm, for any single hash function there are bad subsets of the keys that collide and are stored in the same chain resulting in O(n) search time. If we choose the hash functions randomly the probability that our set of n keys is a bad set for the randomly selected hash function is very small.

1. Given the collection H={h1, h2, h3, h4, h5, h6} of hash functions, with *m* = 3. The hash functions map the keys A, B, C according to the following table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | h1 | h2 | h3 | h4 | h5 | h6 |
| A | 0 | 1 | 2 | 0 | 1 | 2 |
| B | 1 | 2 | 0 | 1 | 1 | 1 |
| C | 1 | 2 | 1 | 0 | 2 | 1 |

* 1. Is the collection H universal? Justify your answer.

**Answer:** The collection is not a universal collection since in a universal collection there are at most |H|/m =6/3 = 2 hash functions for which two distinct keys collide. But there are 3 hash functions for which keys B and C collide.

* 1. What is the probability of B and C colliding? **Answer:** 3/6=1/2.

1. What is linear probing? Discuss advantages and disadvantages

**Answer:** The probe sequence = (h(k)+i)mod m for i = 0, …, m-1. Advantage: simplicity and efficiency. Disadvantage: primary cluster can cause the operations to have O(m) time complexity

1. Let *m* = 13. *h*’(*k*) = *k* mod *m*. *h*2(*k*) = 1 + (*k* mod(*m* - 1)). Assume open addressing and use double hashing *h*1(*k*) = (*h*’(*k*)+ *i* *h*2(*k*)) mod *m* for the probe sequence. Show the table after the following keys are inserted: 13, 26, 55, 43, 70 and all the computations.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 |  |  | 26 | 43 | 70 |  |  |  |  |  | 55 |  |

**Answer:**

Key = 13. Probe 0. *h*’(13) = 13 mod 13= 0. *h*1(13) = (0+ 0 \* *h*2(*k*)) mod 13 =0

Key = 26. Probe 0. *h*’(26) = 26 mod 13= 0. *h*1(26) = (0+ 0 \* *h*2(*k*)) mod 13 =0. There is a collision so try the next probe with *i* = 1. *h*2(26) = 1 + (26 mod 12) = 3. *h*1(26) = (0+ 1 \* 3) mod 13 = 3.

Key = 55. Probe 0. *h*’(55) = 55 mod 13= 3. *h*1(55) = (3+ 0 \* *h*2(*k*)) mod 13 = 3. There is a collision so try the next probe with *i* = 1. *h*2(55) = 1 + (55 mod 12) = 8 and. *h*1(55) = (3+ 1 \* 8) mod 13 = 11.

Key = 43. Probe 0. *h*’(43) = 43 mod 13= 4. *h*1(13) = (4+ 0 \* *h*2(*k*)) mod 13 = 4

Key = 70. Probe 0. *h*’(70) = 70 mod 13= 5. *h*1(13) = (5+ 0 \* *h*2(*k*)) mod 13 = 5

1. What is the advantage of having perfect hashing?

**Answer:** Perfect hashing guarantees worst case performance of O(1) for insert delete and search.

1. Assume a hash table with collision resolution by open addressing with *m* slots and *n* keys.
   1. What is the load factor of the hash table?

**Answer:** The load factor is α = *n*/*m*.

* 1. What is the range of values the load factor may have?

**Answer:** In this case the maximum number of keys that can be inserted is m. So 0 ≤α ≤ 1.

1. Let m be the number of slots, n the expected number of keys that will be stored in a hash table with chaining, and h the hash function. Show that if |U|>=mn, there is a subset of U of size n consisting of keys that all hash to the same slot, so that the worst case search time for hashing with chaining is Theta(n). (Hint: consider the size of the chains if all U keys were hashed to the table.)

**Answer:** If all |U| keys are hashed and there are only m slots at least one slot will have n or more keys. So the worst case search time for hashing with chaining is Theta(n)

1. Assume a linked list where the key stored in each node is a long character string. Assume that you are doing a search and the key you are searching for is also quite long. Your algorithm will compare the search key sequentially to each key in the list until it either finds it, or the whole list has been searched. This may take a very long time.

Now assume that each node in the list contains in addition to the key k, also its hash value h(k). How can h(k) be used to make searching for a key k more efficient? (Assume the hash function h is known).

**Answer:** When stored objects are large having a hash value for each object is very beneficial for equality comparison. First the hash values of the two objects are compared. If they are not the same we know that the objects are different. The only time that we have to do the full comparison of the objects is when we have a collision.

For this problem if there are n keys and the size of each key is m and a hash function is not used the worst case computation time is O(nm) . But when hash values are used the hash value h(k) will be compared to each h(k’) in the linked list. If they are not equal the search continues. Only when h(k)=h(k’) the large keys need to be compared taking O(m) time. The overall worst case time in this case is = 1 to compute h(k)+n to compare with the n hash values)+( O(mq) to compare the small subset q of the keys for which h(k)=h(k’).