R-2.19 Draw the 11-item hash table resulting from hashing the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, using the hash function h(i) = (2i + 5) mod 11 and assuming collisions are handled by chaining.

|  |  |  |  |
| --- | --- | --- | --- |
| **index** | **key** | **H(K)** | **Key** |
| 0 | 12 | 7 | Ø |
| 1 | 44 | 5 | 20 |
| 2 | 13 | 9 | Ø |
| 3 | 88 | 5 | Ø |
| 4 | 23 | 7 | 16,5 |
| 5 | 94 | 6 | 44,88,11 |
| 6 | 11 | 5 | 94,83 |
| 7 | 39 | 6 | 12,13 |
| 8 | 20 | 1 | Ø |
| 9 | 16 | 4 | 13 |
| 10 | 5 | 4 | Ø |

R-2.20 What is the result of the previous exercise, assuming collisions are handled by linear probing?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **index** | **key** | **H(K)** | **Probes** | **Key** |
| 0 | 12 | 7 | 7 | 11 |
| 1 | 44 | 5 | 5 | 39 |
| 2 | 13 | 9 | 9 | 20 |
| 3 | 88 | 5 | 5🡪6 | 5 |
| 4 | 23 | 7 | 7🡪8 | 16 |
| 5 | 94 | 6 | 6🡪10 | 44 |
| 6 | 11 | 5 | 5🡪0 | 88 |
| 7 | 39 | 6 | 6🡪1 | 12 |
| 8 | 20 | 1 | 1🡪2 | 23 |
| 9 | 16 | 4 | 4 | 13 |
| 10 | 5 | 4 | 4🡪3 | 94 |

R-2.21 Show the result of Exercise R-2.19, assuming collisions are handled by quadratic probing, up to the point where the method fails because no empty slot is found.

Quadratic probing formula: A [(i + j^2) mod N]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **index** | **Key** | **H(K)** | **Probes** | **Key** |
| 0 | 12 | 7 | 7 | 5 |
| 1 | 44 | 5 | 5 | 20 |
| 2 | 13 | 9 | 9 | 16 |
| 3 | 88 | 5 | 5->6 | 11 |
| 4 | 23 | 7 | 7->8 | 39 |
| 5 | 94 | 6 | 6->10 | 44 |
| 6 | 11 | 5 | 5->3 | 88 |
| 7 | 39 | 6 | 6->1 | 12 |
| 8 | 20 | 1 | 1->2 | 23 |
| 9 | 16 | 4 | 4->2 | 13 |
| 10 | 5 | 4 | 4->0 | 94 |

R-2.22 What is the result of Exercise R-2.19 assuming collisions are handled by double hashing using a secondary hash function h’(k) = 7 – (k mod 7) ?

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **h(i)** | **H’(k)** | **Probes** |
| 12 | 7 |  | 7 |
| 44 | 5 |  | 5 |
| 13 | 9 |  | 9 |
| 88 | 5 | 3 | 3 |
| 23 | 7 | 5 | 1 |
| 94 | 6 |  | 6 |
| 11 | 5 | 3 | 8 |
| 39 | 6 | 3 | 4 |
| 20 | 1 | 1 | 2 |
| 16 | 4 | 5 | 0 |
| 5 | 4 | 2 | 10 |

Give the pseudo-code description for performing a removal from a hash table that uses linear probing to resolve collisions. Why is it necessary to use a special marker to represent deleted elements?

|  |
| --- |
| Algorithm removalLinearProbing**(**key**)**      Input**:** key to remove from      Output**:** remove and **return** the element        key**,** element**)** **<-** findElement**(**key**)**      If key **!=** NO\_SUCH\_KEY then          key **<-** AVAILABLE  **return** element  **return** NO\_KEY\_FOUND |
| Linear probing handles collision by putting the item in the next empty or available block. So, It is necessary to use a special marker to represent the deleted elements. If we remove it then we will not find the value that might be put in the next block after that because the search will end when it find an empty and not available block. |

C-4.10 Suppose we are given an n-element sequence S such that each element in S   
represents a different vote in an election, where each vote is given as an integer   
representing the ID of the chosen candidate. Without making any assumptions about   
who is running or even how many candidates there are, design an efficient algorithm to   
see who wins the election S represents, assuming the candidate with the most votes   
wins. Handle the possibility of multiple winners and do this using a Dictionary.   
Today specify your solution using pseudo code (tomorrow we will implement in   
JavaScript after discussing today’s pseudo code solution).

Algorithm findElectionWinner(S)

    Input: n-element sequence S where each element represents a different vote

    Output: ID of winning candidate

    mergeSort(S, C)

    winCandidateId <- S.first()

    maxVote <- 0

    prevId <- S.first()

    noOfVote <- 0

    while !S.isEmpty() do

        curId <- S.remove(S.first())

        if curId != prevId then

            if maxVote < noOfVote then

                   maxVote <- noOfVote

                   winCandidateId <- curId

    noOfVote <- 0

        else

                prevId <- curId

                noOfVote <- noOfVote + 1

     return winCandidateId