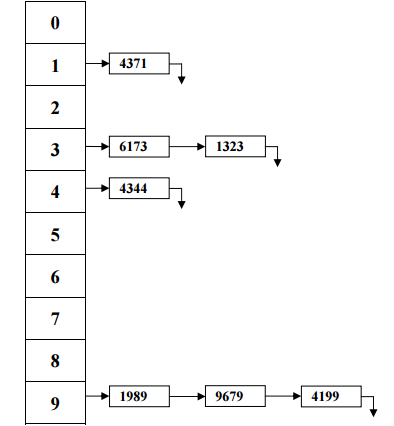
Guohua Jiang

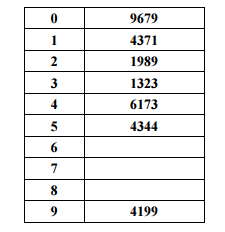
Homwork 3

5.1 Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989} and a hash function *h*(*x*) = *x* mod 10, show the resulting:

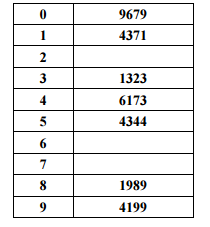
a. Separate chaining hash table.



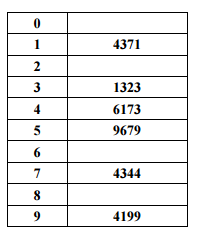
b. Hash table using linear probing.



c. Hash table using quadratic probing.



d. Hash table with second hash function *h*2(*x*) = 7 − (*x* mod 7).



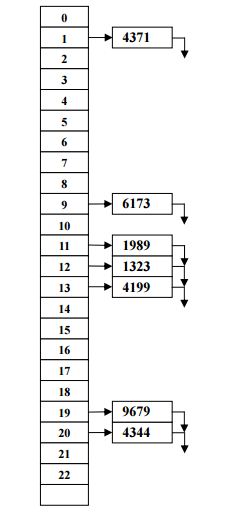
No place for 1989 since the next alternative 5, 1, 7 and 3 are all full.

5.2 Show the result of rehashing the hash tables in Exercise 5.1.

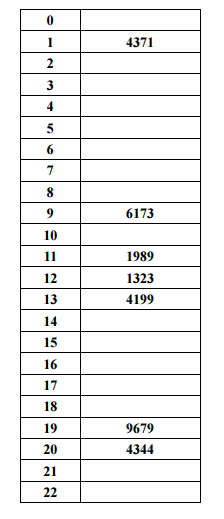
The new table size is 23 sine 23 is the next prime that larger than 20.

The new hash function is h(x) = x mod 23.

a. Separate chaining hash table.



b, c and d will have the same table because there is no collision,



5.10 What are the advantages and disadvantages of the various collision resolution strategies?

* Separate chaining:

Advantages:

1. Simple idea
2. Removals are clean

Disadvantages:

1. Need an extra data structure, which causes extra space

* Linear probing:

Advantages:

1. Insertion never fails if the table has at least one free field.

Disadvantages:

1. Compared to quadratic probing and double hashing, access becomes inefficient at a lower load factor. The reason for this is that resolution sequences for different table fields merge if they have a field in common.

* Quadratic probing:

Advantages:

1. Compared to linear probing access becomes inefficient at a higher

Disadvantages:

1. Insertion sometimes fails although the table still has free fields.

* Double hashing:

Advantages:

1. Compared to linear probing access becomes inefficient at a higher load factor.
2. Resolution sequences for different elements are different even if the first hash function hashes the elements to the same field.
3. If the hash functions are chosen appropriately, insertion never fails if the table has at least one free field.

5.19 Under certain assumptions, the expected cost of an insertion into a hash table with secondary clustering is given by 1*/*(1−*λ*)−*λ*−ln(1−*λ*). Unfortunately, this formula is not accurate for quadratic probing. However, assuming that it is, determine the following:

a. The expected cost of an unsuccessful search.

Expected cost for an unsuccessful search: 1 – *λ - ln(1-* *λ)*

b. The expected cost of a successful search.

Expected cost for an successful search:

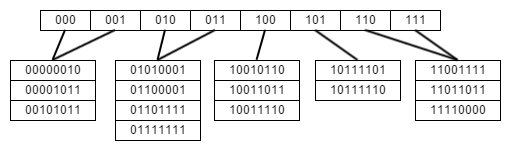
=

=

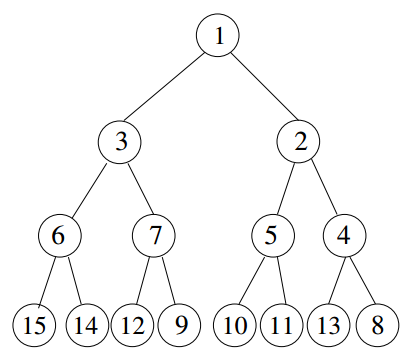
5.23 If a hopscotch table with parameter *MAX\_DIST* has load factor 0.5, what is the approximate probability that an insertion requires a rehash?

The probability that an insertion requires a rehash for a table with a load factor of 0.5 is almost 0.

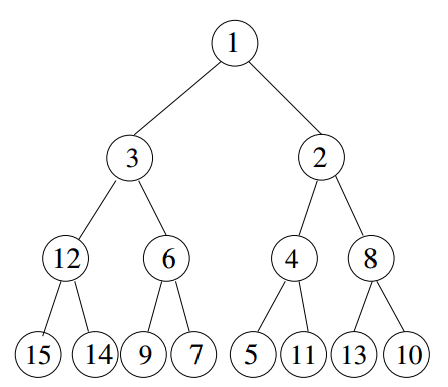
5.27 Show the result of inserting the keys 10111101, 00000010, 10011011, 10111110, 01111111, 01010001, 10010110, 00001011, 11001111, 10011110, 11011011, 00101011, 01100001, 11110000, 01101111 into an initially empty extendible hashing data structure with *M* = 4.



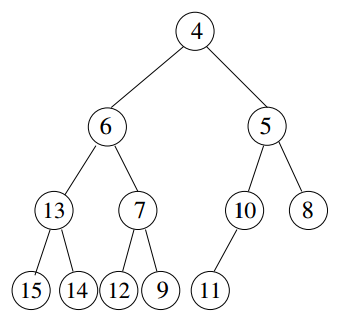
6.2 a. Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time, into an initially empty binary heap.



b. Show the result of using the linear-time algorithm to build a binary heap using the same input.



6.3 Show the result of performing three deleteMin operations in the heap of the previous exercise.



6.4 A complete binary tree of *N* elements uses array positions 1 to *N*. Suppose we try to use an array representation of a binary tree that is not complete. Determine how large the array must be for the following:

a. a binary tree that has two extra levels (that is, it is very slightly unbalanced)

A complete binary tree of N nodes has log2 N height. Now the tree has 2 extra levels. Thus the height of the tree is log2 N + 2.

A log2 N + 2 height tree can have, 2^(log2 N + 2 -1) -1 = 2N -1 nodes.

Thus the array needs be 2N – 1.

b. a binary tree that has a deepest node at depth 2 log*N*

The deepest node at depth 2 logN means that the height of the tree is also 2 log*N.*

Thus maximum number of nodes is 2^(2 log*N*) – 1 = 2N – 1

Thus the array needs to be 2 N – 1.

c. a binary tree that has a deepest node at depth 4.1 log*N*

Thus maximum number of nodes for height 4.1 log*N* is 2^(4.1 log*N*) – 1 = 2^(logN4.1) – 1 = N4.1 – 1.

Thus the array needs to be N4.1 – 1.

d. the worst-case binary tree

The worst-case, the binary tree can have height of N. In this case, the number of nodes for height of N can have 2N – 1.

Thus the array need to be 2N – 1.

6.15 Suppose we need to perform *M* percolateUps and *N* deleteMins on a *d*-heap that initially has *N* elements.

a. What is the total running time of all operations in terms of *M*, *N*, and *d*?

A percolateUp operation on a d-heap with N elements takes O(logd N).

A deleteMin operation on a d-heap with N elements takes O(d logd N).

Thus the total running time is O(M logd N + N d logd N).

b. If *d* = 2, what is the running time of all heap operations?

Substituting 2, we have O((M + N) log2 N).

c. If *d* = *Θ*(*N*), what is the total running time?

If d = θ(N) then d = cN, where c is a constant number.

Substituting d = cN, we have, M logcN N + NcN logcN N = O(M + N2 ).

d. What choice of *d* minimizes the total running time?

Let’s take a long at the function y = logd N. As d getting bigger, y gets smaller.

Thus we should choose the d as large as possible.