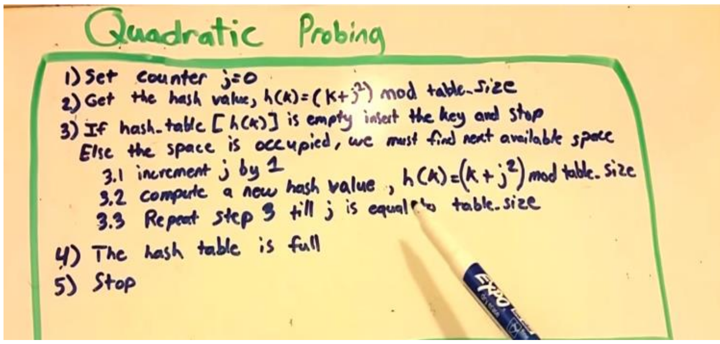
Hash Table Methods

Quadratic Probing



Double Hashing

Exam 2 Practice | CSE 331 | Spring 2018

m=11

(5) (2 points) You are inserting an element x with $h_1(x) = 42$ and $h_2(x) = 7$ into a hash table with 11 cells that uses double hashing. What is the index of your *third* attempt to insert x?

(a) Index 0
(b) Index 1
(c) Index 9
(d) Index 10
(e) A different index

set j count =0 +1 for each attempt reset after insert

$h' = [h_1(x) + i \cdot h_2(x)] \bmod m$

$\Rightarrow [42 + 2(7)] \bmod m$

$\rightarrow [56] \bmod 11 \Rightarrow 1$

(6) (2 points) You are inserting an element x with $h_1(x) = 42$ and $h_2(x) = 7$ into a hash table with 11 cells that uses quadratic probing. What is the index of your *third* attempt to insert x?

(a) Index 0
(b) Index 1
(c) Index 9
(d) Index 10
(e) A different index

don't care about h_2

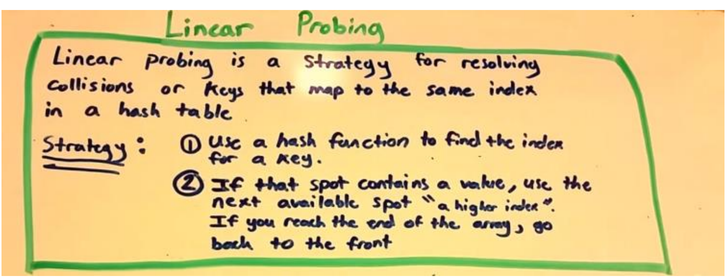
$h = [h_1(x) + i^2] \bmod m$

$42 + 4 \bmod 11$

$46 \bmod 11 = \boxed{2}$

(7) (2 points) What does MysteryFunction1 do?

Linear Probing



Path Finding

Dijkstra's Algorithm vs. Floyd's

- Dijkstra's algorithm is O($m$log$n$) = O(n^2log$n$) in the worst case.
- Floyd's algorithm is O(n^3), but computes all shortest paths.
- Dijkstra's algorithm can compute all shortest paths by starting it n times, once for each node. Thus, to compute all paths, Dijkstra's algorithm is O(nmlog$n$) = O(n^3log$n$) in the worst case.
- Which is better depends on the application.
  - Dijkstra's algorithm is better if only a path from a single node to another or to all others is needed.
  - For all paths, Dijkstra's algorithm is better for sparse graphs, and Floyd's algorithm is better for dense graphs.

- Two main methods
  - Depth-first search (DFS)
    - Go as far as possible, then backtrack
    - Uses stacks
  - Breadth-first search (BFS)
    - Check neighborhood first and then spread out
    - Uses queues

Heaps

Heap(Min) Insert

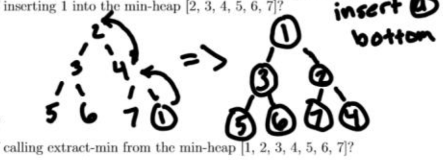
(e) None of the above

(14) (2 points) The result of inserting 1 into the min-heap [2, 3, 4, 5, 6, 7]?

Parent & child

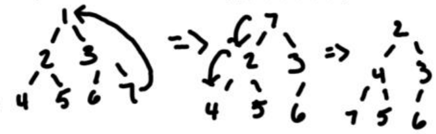
(a) [1, 2, 3, 4, 5, 6, 7]
(b) [2, 3, 4, 5, 6, 7, 1]
(c) [1, 2, 4, 5, 6, 7, 3]
(d) [1, 3, 2, 5, 6, 7, 4]
(e) None of the above

insert 1 bottom



(15) (2 points) The result of calling extract-min from the min-heap [1, 2, 3, 4, 5, 6, 7]?

(a) [2, 3, 4, 5, 6, 7]
(b) [7, 2, 3, 4, 5, 6]
(c) [2, 7, 3, 4, 5, 6]
(d) [2, 4, 3, 5, 6, 7]
(e) None of the above



* Compare left & Right child to determine which side of tree to traverse [2 4 3 7 5 6]

Min heap (smaller values)

Max heap (larger values)

Child = 2n+1 & 2n+2

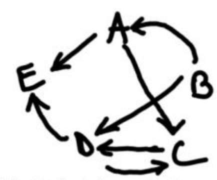
n = index   L   R

Directed Graphs

(17) (10 points) Directed Graphs

v_a	v_b, v_c, v_e
v_b	v_a, v_d
v_c	v_d
v_d	v_c, v_e
v_e	

(a) (6 points) Draw the directed graph associated with the adjacency list shown above.



(b) (2 points) What is the indegree of v_e? 2

(c) (2 points) What is the out-degree of v_b? 2

Search Methods

Degree

- The *degree* of a vertex v is the number of edges connected to v
- In a digraph
  - The out-degree is the number of arcs leaving v
  - The in-degree is the number of arcs entering v

(11) (2 points) Which search algorithm should you use to find a path through a maze?

(a) Depth-first search
(b) Breadth-first search
(c) Djikstra's Algorithm
(d) Floyd-Warshall Algorithm
(e) None of the above

(12) (2 points) How many edges are there in a clique graph with n vertices? (Hint: In a clique, every vertex shares and edge with every other vertex.)

(a) n
(b) $2n$
(c) n^2
(d) $\frac{n(n+1)}{2}$
(e) None of the above

$\frac{(n-1)}{2}$

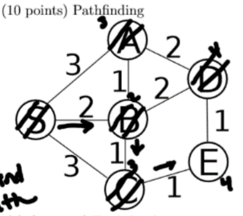
n vertices each has n-1 edges, each edge connects 2 vertices that share it

Path Finding examples /Max & Min & Median Heap Returns

Exam 2 Practice   CSE 331   Spring 2018

(18) (10 points) Pathfinding

only update weight values if we find shorter path



	w
A	3
B	2
C	3
D	4
E	4
S	0

(a) (7 points) Find the the minimum-weight path from s to each vertex in the gr[aph] above as well as its total weight. Show your work.

(b) (3 points) Construct the shortest path from s to e.

S → C → E

```
def getMax ():
    # raise exception if list empty.
    error if list.size = 0

    # If maximum unknown, calculate it on demand.
    if maxcount = 0:
        maxval = list[0]
        for each val in list:
            if val = maxval:
                maxcount += 1
            elsif val > maxval:
                maxval = val
                maxcount = 1

    # Now it is known, just return it.
    return maxval
```

Problems 1-15: Multiple-choice. Circle the letter of the best response.

(1) (2 points) Below is a Bloom filter with hashes $h_1(x) = x \mod 11$ and $h_2(x) = (3x) \mod 11$. Which of the following can be elements of the associated set?

0	1	2	3	4	5	6	7	8	9	10
		X	X		X			X		

* both elements must fit in a marked slot

(a) 14
(b) 27
(c) 31
(d) 41
(e) 42

h_1	3	5	9	8
h_2	9	4		2

(2) (2 points) What is wrong with the following hash function?

```
def hash_point(pt):
    return pt.x + pt.y
```

(a) The function is non-deterministic.
(b) It is not possible to generate all possible hash values.
(c) The hash values are not uniformly distributed.
(d) It is not possible to convert a hash function back to the original point.
(e) Nothing. This is a good hash function.

(3) (2 points) You are inserting an element x with $h(x) = 42$ into a hash table with 11 cells that uses linear probing. Unfortunatly, the table contains a single element at the spot where you want to insert x. Where do you insert x?

(a) Index 0
(b) Index 1
(c) Index 9
(d) Index 10
(e) A different index

42 mod 11 = 9. linear probing so we go to the next unfilled slot

(4) (2 points) You are inserting an element x with $h(x) = 42$ into a hash table with 11 cells that uses separate chaining. Unfortunatly, the table contains a single element at the spot where you want to insert x. Where do you insert x?

(a) Index 0
(b) Index 1
(c) Index 9
(d) Index 10
(e) A different index

With chaining we chain the two elements together at that index

```
def find_median(seq):
    """
    Finds the median (middle) item of the given sequence.
    Ties are broken arbitrarily.
    :param seq: an iterable sequence
    :return: the median element
    """
    if not seq:
        raise IndexError
    else:
        min_heap = Heap(lambda a, b: a <= b)
        max_heap = Heap(lambda a, b: a >= b)

        item =((len(seq))//2)
        min_heap.extend(seq[:item])
        max_heap.extend(seq[item:])

        minpeek =  min_heap.peek()
        maxpeek  = max_heap.peek()
        while minpeek < maxpeek:
            min_heap.extract()
            max_heap.extract()
            min_heap.insert(maxpeek)
            max_heap.insert(minpeek)
            minpeek = min_heap.peek()
            maxpeek = max_heap.peek()
        return max_heap.peek()
```