Chained Hash Table Expected Time

when key is not present:
expected work \( \approx 1 + \) expected number of nodes examined
\[ = 1 + \frac{n}{m} = 1 + \alpha \]

when key is present:
expected work \( \approx \) sum over keys of \( P(\text{search for that key}) \ast (1 + \) expected nodes examined searching for key)\]
\[ = \sum_{k=1}^{n} \frac{1}{m} \left( 1 + \left( 1 + \frac{k-1}{m} \right) \right) = \Theta(1+\alpha) \]

Open Addressing Expected Time

when key is not present, uniform hashing
\[ 1 + \frac{1}{1 - \alpha} \]
## Map/Set Time Complexity

<table>
<thead>
<tr>
<th></th>
<th>unsorted array</th>
<th>hash table</th>
<th>hash table</th>
<th>sorted array</th>
<th>balanced BST</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>open addressing</td>
<td>chaining</td>
<td></td>
<td></td>
</tr>
<tr>
<td>contains_key/contains</td>
<td>O(n) worst case</td>
<td>O(1) avg</td>
<td>O(1) avg</td>
<td>O(log n) w.c.</td>
<td>O(log n) w.c.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O(n) w.c.</td>
<td>O(n) w.c.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>put/add</td>
<td>O(n) w.c.</td>
<td>O(1) avg</td>
<td>O(1) avg</td>
<td>O(n) w.c.</td>
<td>O(n) w.c.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O(n) w.c.</td>
<td>O(n) w.c.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>O(n) w.c.</td>
<td>O(1) avg</td>
<td>O(1) avg</td>
<td>O(log n) w.c.</td>
<td>O(log n) w.c.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O(n) w.c.</td>
<td>O(n) w.c.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for_each</td>
<td>O(n)</td>
<td>O(m)</td>
<td>O(m+n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>

n = number of (key, value) pairs
m = capacity of hash table
\( \alpha = \) load factor = \( n / m \)
Binary Search Trees

struct tree-node {
    char *keys;
    int value;
    struct tree-node *left;
    struct tree-node *right;
};

Binary Search Tree

struct tree-map {
    struct tree-node *root;
}
bool smap_contains_key(smap *m, const char *key)
{
    smap_node *curr = m->root;
    while (curr != NULL && strcmp(key, curr->key) != 0)
    {
        if (strcmp(key, curr->key) < 0)
        {
            curr = curr->left;
            \[ O(h) \]
        } else
        {
            curr = curr->right;
        }
        // height = length of longest path
    }
    root->leaf = \[ O(\log n) \] if tree is balanced
    return (curr != NULL);
}
Unshapely Trees

rule ball chain hamster cheese dunk donut