Today

- Hash table wrap up
- Binary search tree intro

Assignments

- HW8 out (due Thurs)
- No quiz – postponed to next week
“Separate Chaining”

- Each array index has its own linked list of elements (the “chain”)

- The list grows and shrinks with collisions and removals
Benefits of Separate Chaining

Ideally, insert, remove, search are $O(1)$

However, collisions **increase** the cost

- Cost depends on the **load factor** ... how full the table is
  
  
  \[
  \text{load\_factor} = \frac{\text{collection\_size}}{\text{table\_capacity}}
  \]

- As table fills, chance of collision increases

- And hashing efficiency decreases (e.g., using open addressing)

Cost of separate chaining

- Insertion is still $O(1)$ (e.g., insert at front of linked list chains)

- Removal and search require navigating linked list chain ...
  
  \[
  \text{cost to search chains depends on number of collisions}
  \]
  
  \[
  \text{assuming a “good” hash function (distributes keys)}
  \]
  
  \[
  \text{load\_factor < 100\% implies at most length 1 chains}
  \]
  
  \[
  \text{load\_factor > 100\% implies average chain length over 1}
  \]

- Worst case search and remove cost is $O(n)$ since ...

  \[
  \text{avg\_chain\_len} = \text{load\_factor} = \frac{n}{\text{table\_capacity}}
  \]

- But in practice (like with quick sort), hash tables provide efficient search
Resizing and Rehashing

For HW 8 we are going to implement a “resizable” Hash Table

Our implementation is inspired by Java’s HashMap data structure

• Like with HashMap:
  – the load factor threshold is 75%
  – the table array is doubled when the load factor goes over 75%
  – the table array is never decreased in size
  – the initial size of the table array is 16
  – separate chaining is used

When the load factor goes over 75% the array is resized and rehashed

1. a new table array is created with twice the size (the new_table_capacity)
2. (re-)hash elements in current table into new table:
   • int new_index = hash_fun(key) % new_table_capacity;
   • note that “table capacity” is the size of the array
   • iterate through elements in old table, and insert into new table
3. the old table (and linked list chains) are de-allocated / deleted
4. the new table replaces the old one
Binary Search Trees

Basic Idea:

- a tree-like structure (as opposed to list like)
- mixes linked lists and binary search

A tree is a “Non-Linear” Data Structure

A list is a “linear” data structure

A tree is a “non-linear” data structure
In general, a tree forms a **heirarchy** with

- zero or more “**nodes**”
- a distinguished “**root**” node (no parents)

Nodes are arranged in “parent-child” relationships

- each node has zero or more “**children**” (child nodes)
- each node has at most one “**parent**” (parent node)
- a node *without* children is a “**leaf**” node
- a node *with* children is an “**internal**” node

A collection of trees is called a “**forest**”