Overview

- Heaps
- Hash tables
Heaps
Heap

- **Not** a binary search tree; just a binary tree
- Always have the maximal (or minimal) element at the root
- Support removing the root element in $O(\log(N))$, and adding elements in $O(\log(N))$
Heap Property

• A binary tree has the heap property if:
  • It is empty
  • Its value is greater than or equal to both of its children, and the children have the heap property
Example

```
10

5  8

4  3  7  2
```
Advantage

- Heaps always have the highest priority element on top, so we always have easy access to it.
Additional Invariant

• In practice, heaps are always complete
• What does this mean?
Additional Invariant

- In practice, heaps are always complete
- What does this mean? - full except for the last row
Enqueue

• If the tree is complete, we can enqueue by putting the element on the end

• Not done yet - could violate heap property
Enqueue

• To restore the heap property, we can *bubble up* - ensure the heap property holds stepwise with parents, and swap if not
To restore the heap property, we can *bubble up* - ensure the heap property holds stepwise with parents, and swap if not.
Enqueue

- To restore the heap property, we can bubble up - ensure the heap property holds stepwise with parents, and swap if not
Enqueue

• To restore the heap property, we can *bubble up* - ensure the heap property holds stepwise with parents, and swap if not
Enqueue

- To restore the heap property, we can *bubble up* - ensure the heap property holds stepwise with parents, and swap if not
To restore the heap property, we can *bubble up* - ensure the heap property holds stepwise with parents, and swap if not.

9 < 10? - true; done
Dequeue

• After getting the element from the top of the tree, we must restore the heap
Dequeue

- After getting the element from the top of the tree, we must restore the heap.
Dequeue

- After getting the element from the top of the tree, we must restore the heap
Dequeue

- After getting the element from the top of the tree, we must restore the heap
- Idea: swap in the last node from the last level
Dequeue

• After getting the element from the top of the tree, we must restore the heap

• Idea: swap in the last node from the last level
Dequeue

- In order to restore the heap property, we must *bubble down* - swap with the greatest of the children recursively.
Dequeue

• In order to restore the heap property, we must bubble down - swap with the greatest of the children recursively
Dequeue

• In order to restore the heap property, we must *bubble down* - swap with the greatest of the children recursively

3 > 8? - false; need to bubble down
Dequeue

• In order to restore the heap property, we must *bubble down* - swap with the greatest of the children recursively
Dequeue

- In order to restore the heap property, we must *bubble down* - swap with the greatest of the children recursively.

```
5 > 8?
false; swap and bubble down on right
```
Dequeue

- In order to restore the heap property, we must *bubble down* - swap with the greatest of the children recursively
Dequeue

- In order to restore the heap property, we must *bubble down* - swap with the greatest of the children recursively

- base case - no children; done bubbling down
Time Complexity

- Because we force the construction to be complete, we get balanced trees
- Dequeue and enqueue are both $O(\log(N))$ as a result
Optimization

- Heaps can be concisely represented with arrays.

<table>
<thead>
<tr>
<th>As Array</th>
<th>10</th>
<th>5</th>
<th>8</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>As Tree</th>
<th>0</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>
Advantages of Arrays

• What sort of advantages does an array representation have?
Advantages of Arrays

• What sort of advantages does an array representation have?
  • Overall simpler
  • Less space consumed for the same data
  • Getting the last node at the last level is just getting the last valid element in the array
  • (Advanced) CPUs are much happier with arrays than trees (i.e., better performance)
Disadvantages of Arrays

• What sort of issues does the array representation have?
Disadvantages of Arrays

• What sort of issues does the array representation have?
  • Adding elements is more difficult; may entail reallocating the whole array
  • In practice, this is very minor compared to all the other advantages
Hash Tables
Motivation

• Maps are a very common data structure
  • Given a key, give me its corresponding value (**lookup**)
  • Add in a new value associated with some key (**add**)
  • E.g., an address book
Motivation

• We want the lookup and add operations to be as fast as possible

• How might we implement these?
Motivation

• We want the lookup and add operations to be as fast as possible

• How might we implement these?
  • Could use a binary search tree - $O(N)$
  • Force the tree to be balanced - $O(\log(N))$
Tree Style

- We could get $O(\log(N))$ performance
- Still some issues - what?
Tree Style

- We could get $O(\log(N))$ performance
- Still some issues - what?
  - Need to perform $O(\log(N))$ comparisons, and comparisons may not be cheap
  - Performance-wise, $O(1)$ would be better
Doing Better

• What data structure is needed for $O(1)$ lookups?
Doing Better

- What data structure is needed for $O(1)$ lookups?
- Arrays
Using Arrays

• Not obvious how we might utilize arrays for this

• First, a simplifying assumption: all keys are integers \( \geq 0 \)

• How can we take advantage of this?
Using Arrays

• Not obvious how we might utilize arrays for this

• First, a simplifying assumption: all keys are integers \( \geq 0 \)

• How can we take advantage of this?
  • Use keys as indices!
Example

- The following example uses integers $\geq 0$ for keys and characters for values.
Example

Initial array contents: all -1 (indicator that the space is unused)

Array

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Indices
Example

\texttt{insert(3, 'g')}
Example

```
insert(3, 'g')
```

<table>
<thead>
<tr>
<th>-l</th>
<th>-l</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Array

Indices
Example

```
insert(1, 'f')
```

<table>
<thead>
<tr>
<th>-l</th>
<th>-l</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
</table>

0   1   2   3   4   Array

Indices
Example

```
insert(1, 'f')
```

Array

```
-1 f -1 g -1
```

Indices

```
0 1 2 3 4
```
Example

\[\text{insert}(10, \ 'k' \ )\]

<table>
<thead>
<tr>
<th>-l</th>
<th>f</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Array

Indices
Example

No index 10! What do we do?

insert(10, 'k')

Array

-1  f  -1  g  -1

0  1  2  3  4  Indices
Fixing Index Out of Bounds

- We might have a key whose index is out of bounds for the array
- How can we fix this?
Fixing Index Out of Bounds

• We might have a key whose index is out of bounds for the array

• How can we fix this?

• Resizing is suboptimal - may have key 100,000

• Modular arithmetic - insert at \( \text{key} \mod \text{arraySize} \), which guarantees it will be in bounds
Example

insert(10, 'k')

No index 10!
What do we do?
Example

insert(10, 'k')

10 % 5 == 0

-1  f  -1  g  -1

Array

0  1  2  3  4
Indices
Example

$\text{insert}(10, 'k')$

$10 \% 5 == 0$

Array

<table>
<thead>
<tr>
<th>k</th>
<th>f</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
</table>

0 1 2 3 4  Indices
Example

```plaintext
insert(11, 'o')
```

<table>
<thead>
<tr>
<th>k</th>
<th>f</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>Indices</td>
</tr>
</tbody>
</table>

Tuesday, August 19, 14
Example

insert(11, 'o')

11 % 5 == 1

<table>
<thead>
<tr>
<th>k</th>
<th>f</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Array

Indices
Example

Problem - we already have something at 1. Additionally, f was inserted with a different key (1). Both now belong at this position.

```
insert(11, 'o')
```

```
11 % 5 == 1
```

<table>
<thead>
<tr>
<th>k</th>
<th>f</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Array

Indices
Collision Problem

• We have multiple entries that belong in the same slot, even though they have different keys

• Downside of using modular arithmetic

• How might we fix this?
Collision Problem

• We have multiple entries that belong in the same slot, even though they have different keys

• Downside of using modular arithmetic

• How might we fix this?

• Store a linked list at this position of key/value pairs
Example

Problem - we already have something at 1. Additionally, f was inserted with a different key (1). Both now belong at this position.

```
insert(11, 'o')
```

```
11 % 5 == 1
```

<table>
<thead>
<tr>
<th>k</th>
<th>f</th>
<th>-l</th>
<th>g</th>
<th>-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Array

Indices
Example

\texttt{insert(11, 'o')}\n
\[11 \% 5 == 1\]

Array

Indices

\begin{tabular}{cccccc}
0 & 1 & 2 & 3 & 4 \\
\end{tabular}

\begin{tabular}{c}
\texttt{k,10} \\
\texttt{f,1} \\
\texttt{g,3} \\
\end{tabular}
Example

insert(11, 'o')

11 % 5 == 1

Array

Indices

0 1 2 3 4

Array

Indices

0 1 2 3 4

Array

Indices

0 1 2 3 4

Array

Indices

0 1 2 3 4

Array

Indices
Example

lookup(11)

Array

Indices

0 1 2 3 4

lookup(11)
Example

lookup(11)

11 % 5 == 1

Array

Indices

0 1 2 3 4

X X X

k,10 f,11 g,3

o,11

Tuesday, August 19, 14
Example

lookup(11)

11 % 5 == 1

```
0 1 2 3 4
| X | X |
```

```
Array
Indices
```

```
11 % 5 == 1
```

```
X
```

```
X
```

```
k,10
```

```
of,1
```

```
go,3
```

```
o,11
```

```
X
```

```
X
```

```
X
```
Example

lookup(11)

11 \% 5 == 1

1 == 11?

Tuesday, August 19, 14
Example

lookup(11)

11 % 5 == 1

11 % 5 == 1

1 == 11?
false; continue

Tuesday, August 19, 14
Example

lookup(11)

11 % 5 == 1

11 == 11?
Example

lookup(11)

11 % 5 == 1

11 == 11?
true;
found

11 % 5 == 1

11 == 11?
thue;
found
Example

lookup(8)

Array

Indices

0 1 2 3 4

k, 10 f, 1 g, 3 o, 11
Example

\[ \text{lookup}(8) \]

\[ 8 \mod 5 == 3 \]

Array

Indices

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \]

\[ k,10 \quad f,1 \quad g,3 \]

\[ o,11 \]
Example

lookup(8)

8 \% 5 == 3

8 \% 5 == 3?
Example

lookup(8)

8 % 5 == 3

8 == 3?
false;
continue

Tuesday, August 19, 14
Example

lookup(8)

8 \% 5 == 3

End of list; key not contained
Lifting Restriction

- To make progress, we had assumed that keys were positive integers
- How might we extend this to arbitrary keys?
Lifting Restriction

• To make progress, we had assumed that keys were positive integers

• How might we extend this to arbitrary keys?

• Idea: an alternative numeric representation for everything which behaves as a key
Hash Codes

- A way of getting a numeric representation for some non-numeric data
- We can determine which slot a key goes into based on its hash code

```c
int stringHashCode(char* str) {
    int retval = 0;
    for(int x = 0;
        x < strlen(str);
        x++) retval += str[x];
    return retval;
}
```
On Performance

• What time complexity do lookups and additions have?
On Performance

- What time complexity do lookups and additions have?
  - $O(N)$! Worse than the $O(\log(N))$ we were trying to beat!
- Why is this happening?
On Performance

• What time complexity do lookups and additions have?
  • \(O(N)!\) Worse than the \(O(\log(N))\) we were trying to beat!

• Why is this happening?
  • Worst case, all keys end up in the same slot (bucket), and this degrades into a linked list
Degradation

• What circumstances make it more likely that a hash table turns into a linked list?
Degradation

• What circumstances make it more likely that a hash table turns into a linked list?
  • Small array - more keys compete for fewer slots (buckets)
  • Hash function claims the majority of the keys are in the same bucket, e.g. return 0;
Small Array

- How can we address the issue with the array being small?
Small Array

• How can we address the issue with the array being small?

• Initial huge allocation: wastes space

• Dynamically reallocate and redistribute when we get too large: complex and resizing is expensive (common in practice)
Hash Function

- How can we address the issue with the hash function putting everything into the same bucket?
Hash Function

• How can we address the issue with the hash function putting everything into the same bucket?

• Build a better hash function

\[ \text{str}[0] \times 31^{(\text{len}-1)} + \text{str}[1] \times 31^{(\text{len}-2)} + \ldots + \text{str}[\text{len}-1] \]
Time Complexity

• What are the time complexities after adjusting for the small array issue and improving the hash function?
Time Complexity

• What are the time complexities after adjusting for the small array issue and improving the hash function?

• Still $O(N)$! We didn’t change anything in the worst case!
Best-Case Time Complexity

• What is a best-case scenario? What sort of time complexity do we have in this best-case scenario?
Best-Case Time Complexity

- What is a best-case scenario? What sort of time complexity do we have in this best-case scenario?
- Each bucket contains at most one entry
- Constant time - $O(1)$
In Practice

• With a relatively good hash function, in practice, hash tables perform in constant time, despite the $O(N)$ worst-case complexity

• Worst-case complexity only gives you part of the picture

• A little experiment with ~300,000 entries showed that most 95% of buckets had between 0-2 entries, and had at most 7