Agenda

- Binary Heap Operations
- Binary Heap Implementation
Binary Heaps
Binary Heap

Specification

- `BinaryHeap()` creates a new binary heap
- `insert(k)` adds a new item to the heap
- `find_min()` returns the item with the minimum key value, leaving item in the heap
- `del_min()` returns the item with the minimum key value, removing the item from the heap
Binary Heap

Specification (cont.)

- `is_empty()` returns true if the heap is empty, false otherwise
- `size()` returns the number of items in the heap
- `build_heap(list)` builds a new heap from a list of keys
- `decrease_key(k)` finds a key in the heap and updates its key value to a new lower value
Binary Heap
A Complete Binary Tree
Binary Heap

Complete Binary Tree with List Representation
Binary Heap

Create a New Binary Heap

class BinHeap:
    def __init__(self):
        self.heap_list = [0]
        self.current_size = 0
Binary Heap
Percolate the New Node up to its Proper Position
Binary Heap

Percolate the New Node up to its Proper Position
Binary Heap

Percolate the New Node up to its Proper Position

```
5
/   \
7     11
/ \    /   \
14 9 19 21
/   /   \
33 17 27 18
```

swap 2
def perc_up(self, i):
    while i // 2 > 0:
        if self.heap_list[i] < self.heap_list[i // 2]:
            tmp = self.heap_list[i // 2]
            self.heap_list[i // 2] = self.heap_list[i]
            self.heap_list[i] = tmp
        i = i // 2
Binary Heap

Adding a New Item to the Binary Heap

```python
def insert(self, k):
    self.heap_list.append(k)
    self.current_size = self.current_size + 1
    self.perc_up(self.current_size)
```
Binary Heap

Percolating the Root Node down the Tree

remove min
Binary Heap

Percolating the Root Node down the Tree

Diagram:

- 9 as root
- 27 as left child of 9
- 14 as left child of 27
- 33 as left child of 14
- 17 as right child of 14
- 18 as right child of 27
- 11 as right child of 9
- 19 as left child of 11
- 21 as right child of 11

Swap 1: 27 with 14
Binary Heap

Percolating the Root Node down the Tree
Binary Heap

Percolating the Root Node down the Tree

![Binary Heap Diagram]

- Root Node: 9
- Children: 14 and 11
  - 14: Children: 17 and 18
    - 17: Children: 33 and 27
    - 18: None
  - 11: Children: 19 and 21
    - 19: None
    - 21: None

Swap 3
def perc_down(self, i):
    while (i * 2) <= self.current_size:
        mc = self.min_child(i)
        if self.heap_list[i] > self.heap_list[mc]:
            tmp = self.heap_list[i]
            self.heap_list[i] = self.heap_list[mc]
            self.heap_list[mc] = tmp
        i = mc
def min_child(self, i):
    if i * 2 + 1 > self.current_size:
        return i * 2
    else:
        if self.heap_list[i*2] < self.heap_list[i*2+1]:
            return i * 2
        else:
            return i * 2 + 1
def del_min(self):
    retval = self.heap_list[1]
    self.heap_list[1] = self.heap_list[self.current_size]
    self.current_size = self.current_size - 1
    self.heap_list.pop()
    self.percDown(1)
    return retval
def build_heap(self, alist):
    i = len(alist) // 2
    self.current_size = len(alist)
    self.heap_list = [0] + alist[:]
    while (i > 0):
        self.perc_down(i)
        i = i - 1
Binary Heap

Building a New Heap from a List of Items

Initial Heap

i = 2

i = 1
Binary Heap

Building a Heap from the [9, 5, 6, 2, 3]

\[
i=2 \quad [0, 9, 5, 6, 2, 3]
i=1 \quad [0, 9, 2, 6, 5, 3]
i=0 \quad [0, 2, 3, 6, 5, 9]
\]
Binary Heap

Building a Heap from the [9, 5, 6, 2, 3]

bh = BinHeap()
bh.build_heap([9, 5, 6, 2, 3])

print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
print(bh.del_min())
Questions?