CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 9
Submit via Gradescope

Name: Zerun Tian

Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(25 points)*** *Exercise 17.3-3.(Hint: a reasonable potential function to use is $\phi(D_i) = kn_i \cdot \ln n_i$ where n_i is the number of elements in the binary heap, and k is a big enough constant. You can use this function and just show the change in potential for each of the two operations.)*

Consider an ordinary binary min-heap data structure with n elements supporting the instructions INSERT and EXTRACT-MIN in $O(\lg n)$ worst-case time. Give a potential function ϕ such that the amortized cost of INSERT is $O(\lg n)$ and the amortized cost of EXTRACT-MIN is $O(1)$, and show that it works.

Solution:

Using the given potential function, we find that $\phi(D_0) = 0$ when the heap is empty. Then, for all $i > 0$, n_i will never drop 0, so $\phi(D_i) \geq 0 = \phi(D_0)$. Therefore, the total amortized cost of a sequence of n operations is an upper bound on the total actual cost.

Let's first evaluate an expression $n \cdot \ln(n/(n-1))$ which will help simplifying derivations below.

$$\begin{aligned}
\lim_{n\to\infty} n \cdot \frac{n}{n-1} = \lim_{n\to\infty} \ln\left[(\frac{n}{n-1})^n\right] &= \lim_{n\to\infty} \ln\left[(\frac{n-1+1}{n-1})^n\right] \\
&= \lim_{n\to\infty} \ln\left[(1+\frac{1}{n-1})^n\right] \\
&= \lim_{n\to\infty} \ln\left[\left((1+\frac{1}{n-1})^{n-1}\right)^{\frac{n}{n-1}}\right] \\
&= \lim_{n\to\infty} \ln\left[(e)^{\frac{n}{n-1}}\right] \\
&= \lim_{n\to\infty} \frac{n}{n-1} \\
&= 1
\end{aligned}$$

Note that $\lim_{n\to\infty}(1+1/x)^x = e$.

Suppose the i-th operation is INSERT, its amortized cost is,

$$\begin{aligned}
\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\
&\leq k \cdot \ln n_i + kn_i \cdot \ln n_i - k(n_i - 1) \cdot \ln(n_i - 1) \\
&= k \cdot \ln n_i + kn_i \cdot \ln n_i - kn_i \cdot \ln(n_i - 1) + k \cdot \ln(n_i - 1) \\
&\leq 2k \cdot \ln n_i + kn_i \cdot (\ln n_i - \ln(n_i - 1)) \\
&= 2k \cdot \ln n_i + kn_i \cdot \ln \frac{n_i}{(n_i - 1)} \\
&\leq 2k \cdot \ln n_i + k \cdot c \text{ (based on the trick above)} \\
&= O(\lg n_i)
\end{aligned}$$

Note that $n_{i-1} = n_i - 1$ because we had one less element before the insertion.

Suppose the i-th operation is EXTRACT-MIN, its amortized cost is,

$$\begin{aligned}
\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\
&= k \cdot \ln n_i + kn_i \cdot \ln n_i - kn_{i-1} \cdot \ln n_{i-1} \\
&\leq k \cdot \ln n_{i-1} + k(n_{i-1} - 1) \cdot \ln(n_{i-1} - 1) - kn_{i-1} \cdot \ln n_{i-1} \\
&= k \cdot \ln n_{i-1} + kn_{i-1} \cdot \ln(n_{i-1} - 1) - k \cdot \ln(n_{i-1} - 1) - kn_{i-1} \cdot \ln n_{i-1} \\
&= k \cdot (\ln n_{i-1} - \ln(n_{i-1} - 1)) + kn_{i-1} \cdot (\ln(n_{i-1} - 1) - \ln n_{i-1}) \\
&= k \cdot \ln \frac{n_{i-1}}{n_{i-1} - 1} + kn_{i-1} \cdot \ln \frac{n_{i-1} - 1}{n_{i-1}} \\
&\leq k \cdot c + kn_{i-1} \cdot \ln(\frac{n_{i-1}}{n_{i-1} - 1})^{-1} \\
&\leq k \cdot c - k \cdot c' \text{ (based on the trick above)} \\
&= k \cdot c - k \cdot c' \\
&= O(1)
\end{aligned}$$

Note that $n_i = n_{i-1} - 1$ because we had one more element before extracting the min.

The given potential function lets us derive that the amortized cost of INSERT is $O(\lg n)$ and the amortized cost of EXTRACT-MIN is $O(1)$.

2. ***(25 points)*** *Exercise 17.3-6.*

Show how to implement a queue with two ordinary stacks (Exercise 10.1-6) so that the amortized cost of each ENQUEUE and each DEQUEUE operation is $O(1)$.

Solution:

We have implemented such a queue in hw7-8. Here is the pseudocode for ENQUEUE and DEQUEUE.

```
1: function ENQUEUE(A, B, x)
2:     PUSH(A, x)                          ▷ call the PUSH api of stack
3: end function
```

ENQUEUE's implementation is trivial. Dequeuing an element involves first checking if the stack B has any elements. If it does, we don't move things around; otherwise, we transfer every element from A to B. Eventually, we pop the top element of B.

```
1: function DEQUEUE(A, B)
2:     if STACK-EMPTY(B) then
3:         while not STACK-EMPTY(A) do
4:             x = POP(A)
5:             PUSH(B, x)
6:         end while
7:     end if
8:     return POP(B)
9: end function
```

The actual cost of ENQUEUE is 1 and the cost of DEQUEUE depends on whether stack B is empty. In the best case, i.e. B is not empty, the cost is 1. In the worst case, i.e. B is empty, the cost is two times k, the number of elements in A, plus 1.

We define the following amortized costs: ENQUEUE operation costs 4; DEQUEUE operation costs 0.

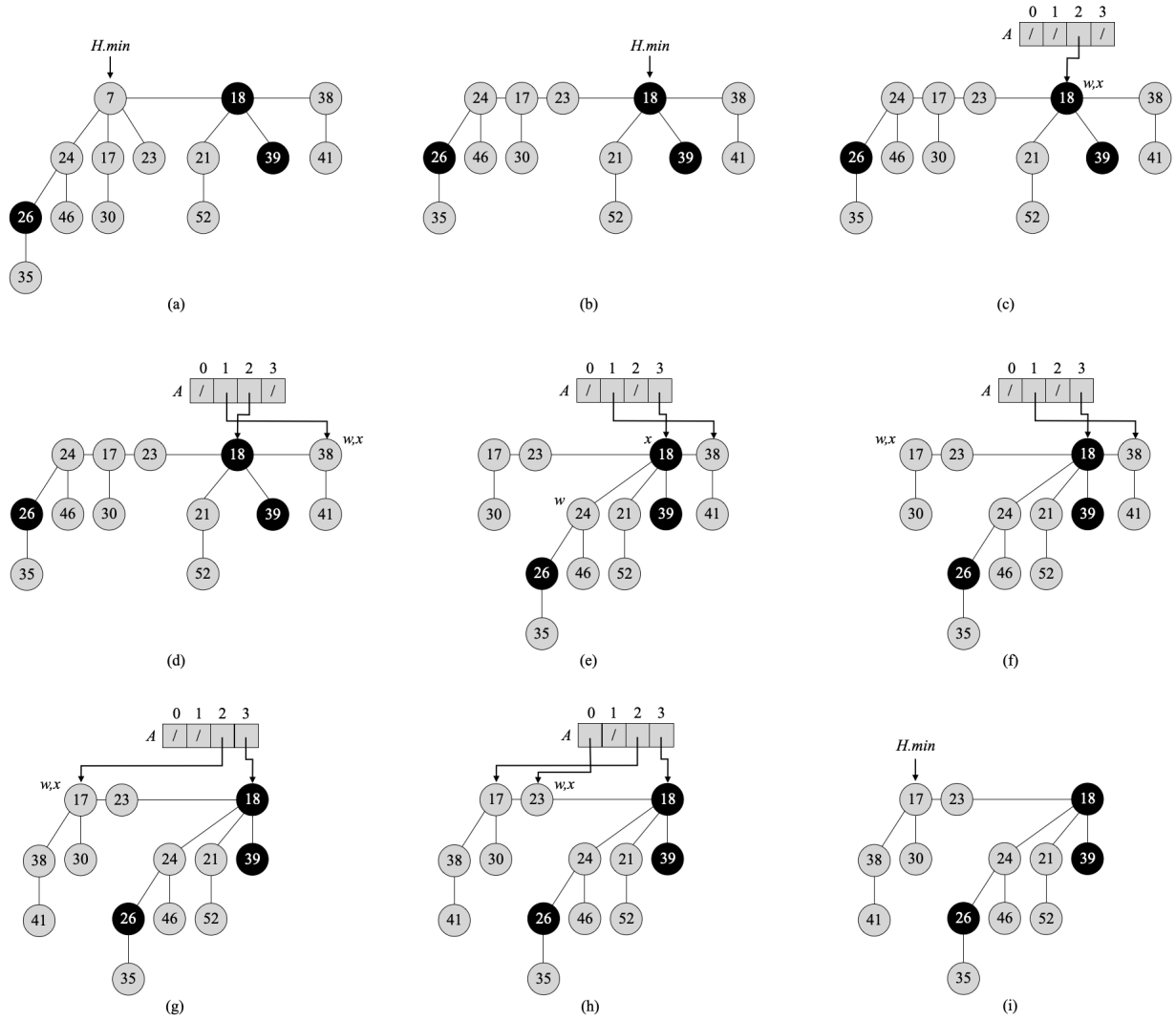
We can pay for any sequence of queue operations by charging the amortized costs. Here is how. When we enqueue, we push an element onto the stack A. We spend 1 for the actual cost and save 3 credits for later. Consequently, every element of A owns 3 credits that can be used by subsequent operations. For any element to be dequeued, it has to go through the process of being popped out from A, then being pushed onto B, and finally being popped from B. Each of the three phases costs exactly 1 credit, which has been prepaid when the element is enqueued. Thus, we have always charged enough up front to pay for DEQUEUE operations. This implies that the total amortized cost is an upper bound on the total actual cost.

Hence, we showed that the amortized cost of every operation could be $O(1)$ in such design.

3. ***(25 points)*** *Exercise 19.2-1.*

Show the Fibonacci heap that results from calling FIB-HEAP-EXTRACT-MIN on the Fibonacci heap shown in Figure 19.4(m).

Solution:



In (a), we show the 19.4(m) setup. In (b), we remove the current min 7, meld its children into the root list, and move the min pointer to its right child. In (c), we start to run the CONSOLIDATE procedure. An array A is initialized to keep track of root nodes of different degrees for the purpose of merging. In (e), we are handling the node with value 24 whose degree is 2. Because the array A has tracked another root node with degree 2, a merge operation is performed, which results in the configuration (e). The next node to be processed is the node with value 17. It has degree 1 which is consistent with the degree of node 38 according to A. The resulting configuration is shown in (g). Last but not the least, the node with value 23 has degree 0 which is the only root node of that degree. Finally, we construct H with nodes linked in A and update the min pointer.

4. ***(50 points)*** *Implement binomial heaps as described in class and in the book. You should use links (pointers) to implement the structure as shown in the fig 1.*

Your implementation should include the operations: Make-heap, Insert, Minimum, Extract-Min, Union, Decrease-Key, Delete

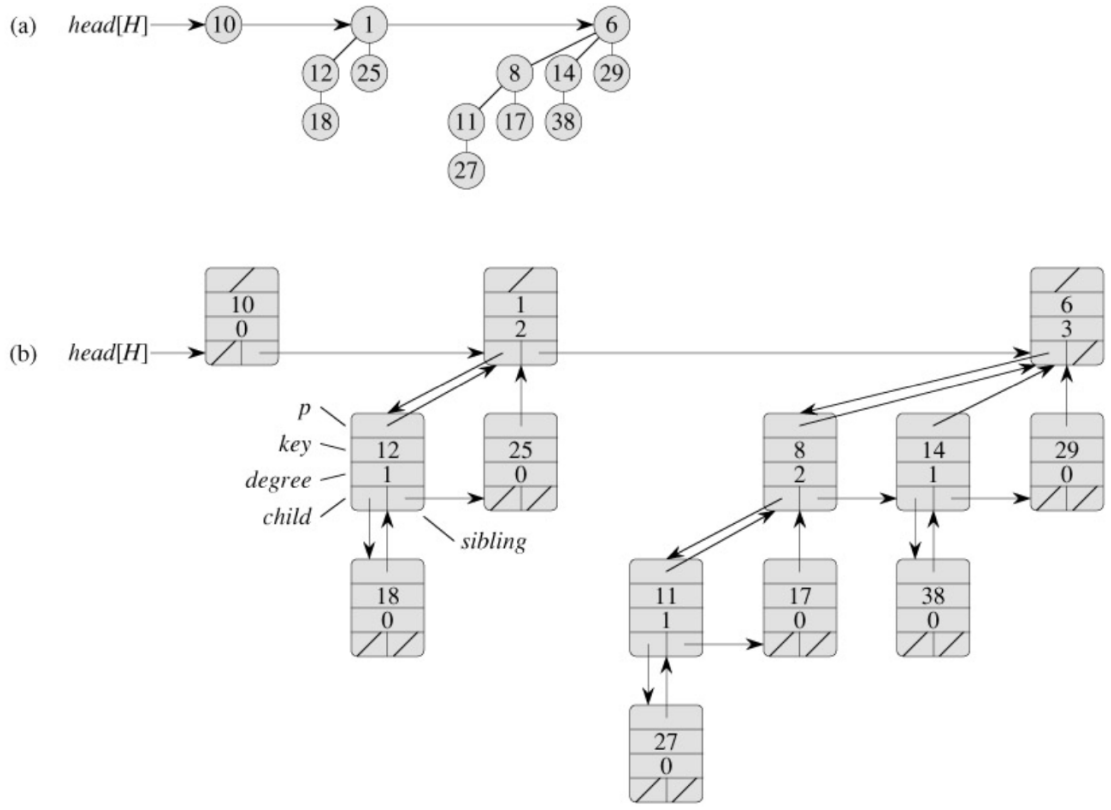


Figure 1: Binomial Heaps

Make sure to preserve the characteristics of binomial heaps at all times:

1. (1) each component should be a binomial tree with children-keys bigger than the parent-key;
2. (2) the binomial trees should be in order of size from left to right. Test your code several arrays set of random generated integers (keys).

Solution:

Code is listed in the next few pages.

```
import sys
from os import linesep
from print_tree import print_tree


class Node(object):
    def __init__(self, key):
        self.p = None
        self.key = key
        self.degree = 0
        self.child = None
        self.sibling = None

    def __str__(self):
        return f'{self.key}'

    def add_child(self, node):
        node.p = self
        node.sibling = self.child
        self.child = node
        self.degree += 1

    def search(self, key):
        if self.key == key:
            return self
        if self.child is None and self.sibling is None:
            return None
        res = None
        if self.sibling is not None:
            res = self.sibling.search(key)
        if res is None and self.child is not None:
            res = self.child.search(key)
        return res


class BinomialHeap(object):
    def __init__(self):
        self.root_list = None

    def add_root(self, node: Node):
        node.p = None
        node.sibling = self.root_list
        self.root_list = node

    def insert(self, node: Node):
        pass

    def print(self):
        root = self.root_list
        while root is not None:
            print_heap(root)
            root = root.sibling


class BinomialHeapOps(object):
    @staticmethod
    def _merge_trees(p: Node, q: Node):  # p has the smaller value
        p.add_child(q)

    @staticmethod
    def _merge_wo_consolidation(p: BinomialHeap, q: BinomialHeap):
        a = p.root_list
        b = q.root_list
```

```
        if a is None:
            return b
        if b is None:
            return a

        if a.degree <= b.degree:
            merged = a
            a = a.sibling
        else:
            merged = b
            b = b.sibling
        merged_p = merged

        while a is not None or b is not None:
            if a is None:
                merged_p.sibling = b
                break
            if b is None:
                merged_p.sibling = a
                break

            if a.degree <= b.degree:
                merged_p.sibling = a
                a = a.sibling
            else:
                merged_p.sibling = b
                b = b.sibling
            merged_p = merged_p.sibling

        return merged

    @staticmethod
    def make_heap():
        return BinomialHeap()

    @staticmethod
    def insert(heap: BinomialHeap, node: Node):
        other = BinomialHeapOps.make_heap()
        other.add_root(node)
        heap.root_list = BinomialHeapOps.union(heap, other).root_list

    @staticmethod
    def _find_minimum(heap: BinomialHeap):
        root = heap.root_list
        min_node = None
        min_prev = None
        min_val = sys.maxsize
        prev = None
        while root is not None:
            if root.key < min_val:
                min_val = root.key
                min_prev = prev
                min_node = root
            prev = root
            root = root.sibling
        return min_node, min_prev

    @staticmethod
    def minimum(heap: BinomialHeap):
        min_node, _ = BinomialHeapOps._find_minimum(heap)
        return min_node
```

```
    @staticmethod
    def extract_min(heap: BinomialHeap):
        min_node, min_prev = BinomialHeapOps._find_minimum(heap)
        if min_node is None:
            return None
        other = BinomialHeap()
        cur = min_node.child
        reversed_children = None
        while cur is not None:
            nxt = cur.sibling
            cur.sibling = reversed_children
            reversed_children = cur
            cur = nxt
        other.root_list = reversed_children

        if min_prev is None:
            heap.root_list = min_node.sibling
        else:
            min_prev.sibling = min_node.sibling

        heap.root_list = BinomialHeapOps.union(heap, other).root_list
        return min_node

    @staticmethod
    def union(p: BinomialHeap, q: BinomialHeap):
        heap = BinomialHeap()
        merged = BinomialHeapOps._merge_wo_consolidation(p, q)
        if merged is None:
            return heap
        cur = merged
        pre = None
        nxt = merged.sibling
        while nxt is not None:
            if cur.degree == nxt.degree and (nxt.sibling is None
               or nxt.key != nxt.sibling.key):
                if cur.key <= nxt.key:
                    cur.sibling = nxt.sibling  # advance the cur's sibling to next's
sibling
                    BinomialHeapOps._merge_trees(cur, nxt)
                    # no need to advance the cur pointer
                else:
                    if pre is None:
                        merged = nxt
                    else:
                        pre.sibling = nxt
                    BinomialHeapOps._merge_trees(nxt, cur)
                    cur = nxt
            else:
                pre = cur
                cur = nxt
            nxt = cur.sibling

        heap.root_list = merged
        return heap

    @staticmethod
    def decrease_key(node: Node, new_key):
        if new_key > node.key:
            raise ValueError(f'The new key {new_key} must be no
                             larger than the current key {node.key}.')
        node.key = new_key
        parent = node.p
        while parent is not None and parent.key > node.key:
            tmp = parent.key
```

```
            parent.key = node.key
            node.key = tmp
            node = parent
            parent = node.p

    @staticmethod
    def delete(heap: BinomialHeap, node: Node):
        BinomialHeapOps.decrease_key(node, -sys.maxsize)
        BinomialHeapOps.extract_min(heap)

    @staticmethod
    def search(heap: BinomialHeap, key):
        return heap.root_list.search(key) if heap.root_list is not None else None


class print_heap(print_tree):
    def get_children(self, node: Node):
        children = []
        child = node.child
        while child is not None:
            children.append(child)
            child = child.sibling
        return children

    def get_node_str(self, node: Node):
        return f'[{node.key} (d:{node.degree})]'


def build_example_heap():
    heap = BinomialHeap()

    root6 = Node(6)
    root6.add_child(Node(29))
    node14 = Node(14)
    node14.add_child(Node(38))
    root6.add_child(node14)
    node11 = Node(11)
    node11.add_child(Node(27))
    node8 = Node(8)
    node8.add_child(Node(17))
    node8.add_child(node11)
    root6.add_child(node8)
    heap.add_root(root6)

    root1 = Node(1)
    root1.add_child(Node(25))
    node12 = Node(12)
    node12.add_child(Node(18))
    root1.add_child(node12)
    heap.add_root(root1)

    root10 = Node(10)
    heap.add_root(root10)
    return heap


def build_example_heap2():
    heap = BinomialHeap()
    root5 = Node(5)
    heap.add_root(root5)
    return heap
```

```
def build_example_heap3():
    heap = BinomialHeap()
    root5 = Node(5)
    root5.add_child(Node(7))
    heap.add_root(root5)
    return heap


def build_example_heap4():
    heap = BinomialHeap()
    root5 = Node(5)
    root5.add_child(Node(7))
    node14 = Node(14)
    node14.add_child(Node(20))
    root5.add_child(node14)
    heap.add_root(root5)
    return heap


def parse_num():
    parsed_val = ''
    while type(parsed_val) is not float:
        value = input(': ')
        try:
            parsed_val = float(value.strip())
            return parsed_val
        except ValueError:
            print(f'> {value} is not a number, please retry')
    return parsed_val


def interact():
    heap = BinomialHeapOps.make_heap()
    while True:
        print("> Enter 'i', 'd', 'm', 'e', 'k', or 'q' to select one of
              the operations")
        print('- [i]insert, [d]delete, [m]minimum, [e]extract-min, [k] decrease-key,
              [q]quit')
        option = input(': ').lower()
        if option == 'i':
            print('> Enter a number you want to insert')
            key = parse_num()
            node = Node(key)
            BinomialHeapOps.insert(heap, node)
            print('Here is the resulting heap: ')
            heap.print()
        elif option == 'd':
            print("> Enter the key of the node you want to delete")
            key = parse_num()
            node = BinomialHeapOps.search(heap, key)
            if node is not None:
                BinomialHeapOps.delete(heap, node)
                print('| Here is the resulting heap: ')
                heap.print()
            else:
                print(f'| Warning: there is no node with key {key}')
        elif option == 'm':
            print(f'| The current minimum is {BinomialHeapOps.minimum(heap).key}')
        elif option == 'e':
            node = BinomialHeapOps.extract_min(heap)
            if node is not None:
                print(f'| Extracted the min node of key {node.key}')
            print('| Here is the resulting heap: ')
```

```
            heap.print()
        elif option == 'k':
            print('> Enter the key of the node for which you want to decrease key')
            key = parse_num()
            node = BinomialHeapOps.search(heap, key)
            if node is not None:
                print(f'> Enter a new key (must be <= {node.key})')
                new_key = parse_num()
                BinomialHeapOps.decrease_key(node, new_key)
                print('| Here is the resulting heap: ')
                heap.print()
            else:
                print(f'| Warning: there is no node with key {key}')
        elif option == 'q':
            return
        else:
            print(f'{option} is invalid, please retry{linesep}')
            continue
        print('')


def main():
    print('Demo')
    # CLI interactive demo
    interact()

    # Demo for union
    # - trivial 1
    heap1 = build_example_heap()
    heap2 = build_example_heap2()
    print('Merge A >>>>>>>>>>>>>>>>>>>>')
    heap1.print()
    print('with B >>>>>>>>>>>>>>>>>>>>')
    heap2.print()
    print('yields >>>>>>>>>>>>>>>>>>>>')
    heap = BinomialHeapOps.union(heap1, heap2)
    heap.print()

    # - trivial 2
    heap1 = build_example_heap()
    heap3 = build_example_heap3()
    print('Merge A >>>>>>>>>>>>>>>>>>>>')
    heap1.print()
    print('with B >>>>>>>>>>>>>>>>>>>>')
    heap3.print()
    print('yields >>>>>>>>>>>>>>>>>>>>')
    heap = BinomialHeapOps.union(heap1, heap3)
    heap.print()

    # - complex
    heap1 = build_example_heap()
    heap4 = build_example_heap4()
    print('Merge A >>>>>>>>>>>>>>>>>>>>')
    heap1.print()
    print('with B >>>>>>>>>>>>>>>>>>>>')
    heap4.print()
    print('yields >>>>>>>>>>>>>>>>>>>>')
    heap = BinomialHeapOps.union(heap1, heap4)
    heap.print()


if __name__ == '__main__':
    main()
```

5. ***(Extra Credit)*** *Find a way to nicely draw the binomial heap created from input, like in the figure.*

Solution:

6. ***(Extra Credit)*** *Write code to implement Fibonacci Heaps, with discussed operations:ExtractMin, Union, Consolidate, DecreaseKey, Delete.*

Solution:

7. ***(Extra Credit)*** *Figure out what are the marked nodes on Fibonacci Heaps. In particular explain how the potential function works for FIB-HEAP-EXTRACT-MEAN and FIB-HEAP-DECREASE-KEY operations.*

Solution: