Advanced Data Structures

Labwork 2: Disjoint-set structures. Binomial Heaps

October 14, 2020

Disjoint set structures

This homework is about using a disjoint-set data structure to compute a minimum-weight spanning tree of a weighted graph.

Weighted graphs

A **weighted graph** is a finite set of nodes connected by edges which have positive real numbers as weights. For example, the following is a weighted graph with 5 nodes and 6 edges: We will assume that

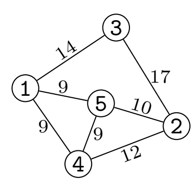


Figure 1: A weighed graph which is connected

- the nodes of a graph with n nodes are labeled with numbers from 1 to n.
- a text file which stores the representation of a weighted graph in the following way:
  - The first line contains the value of n (an integer)
  - The following lines contain 3 numbers separated by whitespace:
    $i \quad j \quad w$
    to indicate that the graph has an edge from node i to node j with weight w.

We assume that the edges are enumerated in increasing order of weight. For example, the weighted graph from Fig. **??** can be stored and read from a text file with the following content:

```
5
1 4 9
1 5 9
4 5 9
2 5 10
2 4 12
2 3 17
```

Minimum weight spanning trees

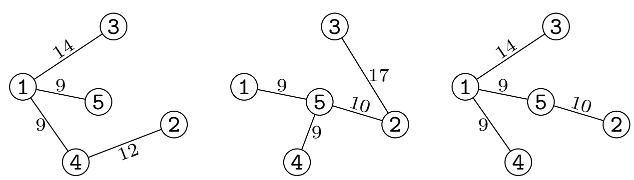
A graph is **connected** if there is a path between every two nodes in the graph. Fo example the weighted graph from Fig. **??** is connected.

A **spanning tree** of a weighted and connected graph G is a set T of edges of G such that

1. Every node of G is an endpoint of an edge in T
2. T has no loops.

The **weight** $w(T)$ of T is the sum of weights of edges in T.

For example, the following are spanning trees of the graph in Fig. **??**:



$w(T_1) = 44$ $\quad$ $w(T_2) = 45$ $\quad$ $w(T_3) = 42$

A **minimum weight spanning tree** (or **MWST**) of G is a spanning tree of G whose weight has minimum possible value. For example, T_3 is a MWST of the graph from Fig. **??**.

A MWST of a connected and weighted graph G with n nodes can be found with Kruskal algorithm:

Start with the initial partition $S = \{\{1\}, \{2\}, \ldots, \{n\}\}$, $T = \emptyset$ and $W = 0$
for each edge (i, j, w) of G, in increasing order of weights **do**
  if i, j are not in the same component of S
    add (i, j, w) to T
    Union(i, j)
    $W = W + w$
  end if
end for
return T, W

Labwork 1

Implement a program that reads from a text file `graph.txt` the representation of a connected weighted graph G and computes a MWST of G. The program will print the weight and the list of edges of the MWST.

Suggestion: implement a disjoint set-data structure using the explanations from Lecture 2, and use it to implement Kruskal algorithm.

Labwork 2

Consider a simply linked-list of nodes with the following structure (the nodes are linked via the `sibling` pointers)

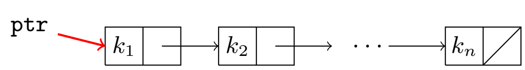
```
struct Node {
   int key;
   Node *sibling;
}
```

Write down a program that performs the following operations:

- It reads from the console a line of n integers separated by spaces

  $$k_1\ k_2\ \ldots\ k_n$$

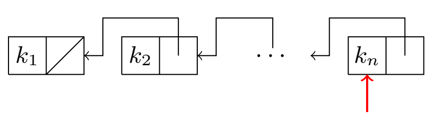
  and creates a pointer `ptr` to the linked list with nodes containing the keys $k_1, \ldots, k_n$, in this order:

- calls the function

  `Node* reverseList(Node *ptr);`

  that reverses te list `ptr` (by making the links to point in the opposite direction), and returns a pointer to its first element.

  (NOTE: You should implement `reverseList`)

- Displays the keys of the nodes in the inversed list,by traversing the nodes from head to tail.

Binomial heaps

Labwork 3

You can download from `Classroom` or from the webpage of this lecture

`http://staff.fmi.uvt.ro/~mircea.marin/lectures/ADS/binoheap.zip`

an incomplete implementation of binomial heaps. Complete the implementation with the implementation of the capability to extract the node with minimum key from a binomial heap. This amounts to implementing the following functions:

- `Node* reverseList(Node* l)`
  which should behave the same as the function implemented in the previous exercise.

- `Node* findMinRoot(Node* l)`
  should return a pointer to the node with minimum key from the linked list of nodes pointed to by `l`. If `l` is the null pointer, the function should return the null pointer.

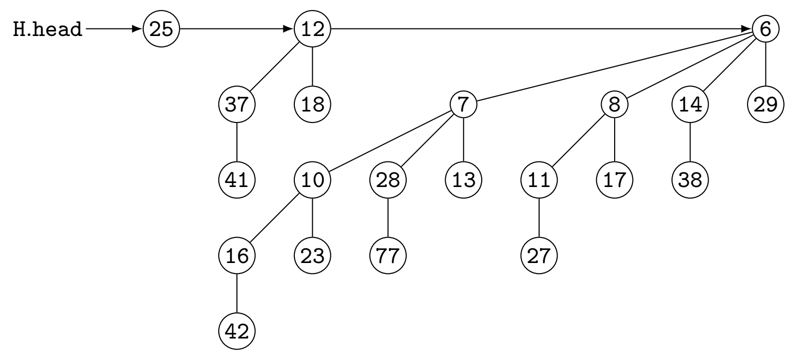
Exercises

1. Suppose that x is a node in a binomial tree within a binomial heap, and assume that $x \to$ `sibling` $\neq$ `NIL`.

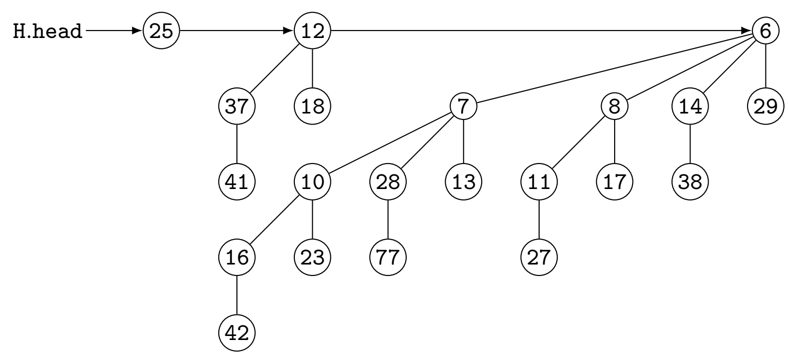
   (a) If x is not a root, how does $x \to$ `sibling` $\to$ `degree` compare to $x \to$ `degree`?

   (b) If x is a root, how does $x \to$ `sibling` $\to$ `degree` compare to $x \to$ `degree`?

2. Show the binomial heap that results when node with key `24` is inserted into the binomial heap shown below:



3. Show the binomial heap that results when the node with key 28 is deleted from the binomial heap shown below:

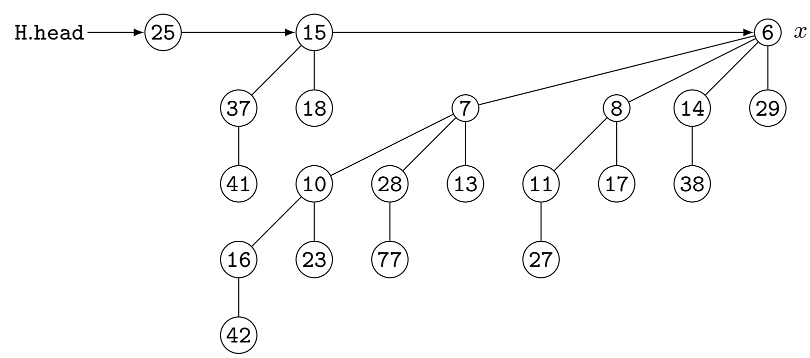


4. Suppose H is a binomial heap implemented as described in the lecture notes. Write the pseudocode for the operation

   `increaseKey(H,` x, k`)`

   which takes as inputs a pointer to a node x in H with $x \rightarrow \texttt{key} < k$ and increases the key of x to new value k.

   (a) Draw the binomial heap that results after increasing the key of node x in the heap depicted below to new value 12.



   (b) What is the worst runtime complexity of this operation?

   (c) Indicate a binomial heap H with 16 nodes, a node x of H, and a value k such that the operation

   `increaseKey(H,` x, k`)`

   takes the longest possible time.