University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Schneider

Algorithms and Data Structures
Winter Term 2019/2020
Sample Solution Exercise Sheet 9

Remark: For this exercise, the material of the 13^{th} *video lecture is relevant.*

Exercise 1: "Reverse" Connected Components

(a) Let $G = (V, E)$ be a directed graph with n nodes and m edges given as *adjacency list*. Let $v \in V$ be a node. Give an algorithm with runtime $\mathcal{O}(n + m)$ that computes the set $U = \{u \in V : \exists \text{ Path from } u \text{ to } v\}$, i.e., all nodes u for which a path from u to v exists.

(b) Analyze the running time and argue the correctness of your algorithm.

Sample Solution

(a) We first compute the "reverse graph" $G' = (V, E')$ where $(y, x) \in E'$ if and only if $(x, y) \in E$. More specifically we compute the adjacency list L of G' from the adjacency list L' of G. We approach this by iterating through the respective lists of all nodes.

If the adjacency list $L[x]$ of node x has a pointer to node y, we add x to $L'[y]$. Finally, on the G' given by L' we conduct a BFS (or a DFS) starting from v and return all nodes that are reached (marked) in the process.

(b) Since a path from u to v in G becomes a path from v to u in G' and vice versa, we will find all nodes u that have a path to v (and not more).

Iterating through L to create L' takes $\mathcal{O}(n + m)$ time. BFS (or DFS) on G' also takes $\mathcal{O}(n + m)$ time. This is also asymptotically optimal since have to look at each node and edge at least once.

Exercise 2: Priority Queue with Decrease Key Operation

A heap data structure offers a simple implementation of the functionality of a priority queue. We already know that we can insert elements with keys (i.e. priorities) into a binary tree and then call `heapify` to make a valid heap out of it. We can also insert elements individually using the `insert` operation. Furthermore, we can get the element with the highest priority (that is, the one with the smallest key) with the `delete-min` operation.
For Dijkstras' algorithm, we also require an operation `decrease-key`(p, k) which gets a *pointer* p to directly access an element in the binary tree, and a key k to which the key of that element is lowered, provided that it is not already lower and subsequently restores the heap condition. Give pseudocode that implements `decrease-key`(p, k) in $\mathcal{O}(\log n)$ time if n is the number of elements in the heap.

Sample Solution

The following operation sifts up p as long as its parent has a bigger key. We are done after at most h loop iterations, where h is the height of the tree. Note that $h \in \mathcal{O}(\log n)$. Since a `swap-up`$(p)$ takes only constant time, the runtime of `decrease-key`(p, k) is $\mathcal{O}(\log n)$.

Algorithm 1 `decrease-key`(p, k)

```
if $p$.key $\leq k$ then return
else $p$.key $\leftarrow k$
$prt \leftarrow p$.parent
while $prt \neq \perp$ and $prt$.key $> k$ do
    swap-up($p$)
    $prt \leftarrow p$.parent
```

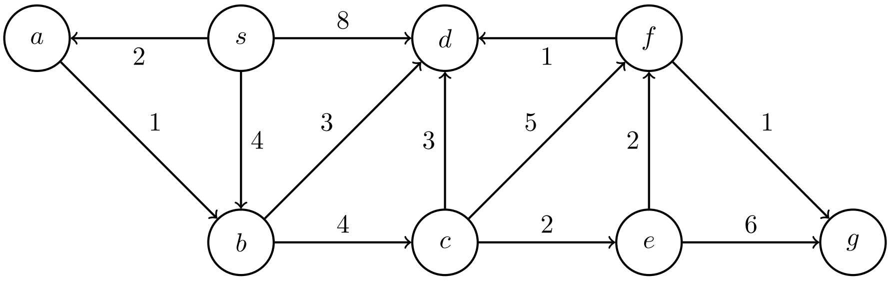
Operation `swap-up`(p) just switches places between a node p and its parent and reattaches all respective pointers. For the sake of completeness it is given below. Note that this becomes much easier if we assume that the heap is given in the form of an array instead.

Algorithm 2 `swap-up`(p)

```
$prt \leftarrow p$.parent
$gprt \leftarrow prt$.parent
$tmp \leftarrow$ copy of node $prt$                ▷ For convenience create a new copy of node $prt$
$tmp$.left $\leftarrow p$.left               ▷ new node $tmp$ gets $p$ as parent and adopts $p$'s children
$tmp$.right $\leftarrow p$.right
$tmp$.parent $\leftarrow p$
if $p = prt$.left then                   ▷ $p$ adopts $tmp$ and $tmp$'s other child (that was not $p$)
    $p$.left $\leftarrow tmp$
    $p$.right $\leftarrow prt$.right
else
    $p$.right $\leftarrow tmp$
    $p$.left $\leftarrow prt$.left
$p$.parent $\leftarrow gprt$                                  ▷ $p$'s parent is now $gprt$
if $prt = gprt$.left then $gprt$.left $\leftarrow p$      ▷ $gprt$ adopts $p$ as child in place of $prt$
else $gprt$.right $\leftarrow p$
delete $prt$                                  ▷ delete old, detached node $prt$
```

Exercise 3: Dijkstras' Algorithm

Execute Dijkstras' Algorithm on the following weighted, directed graph, starting at node s. Into the table further below, write the distances from each node to s that the algorithm stores in the priority queue after each iteration.



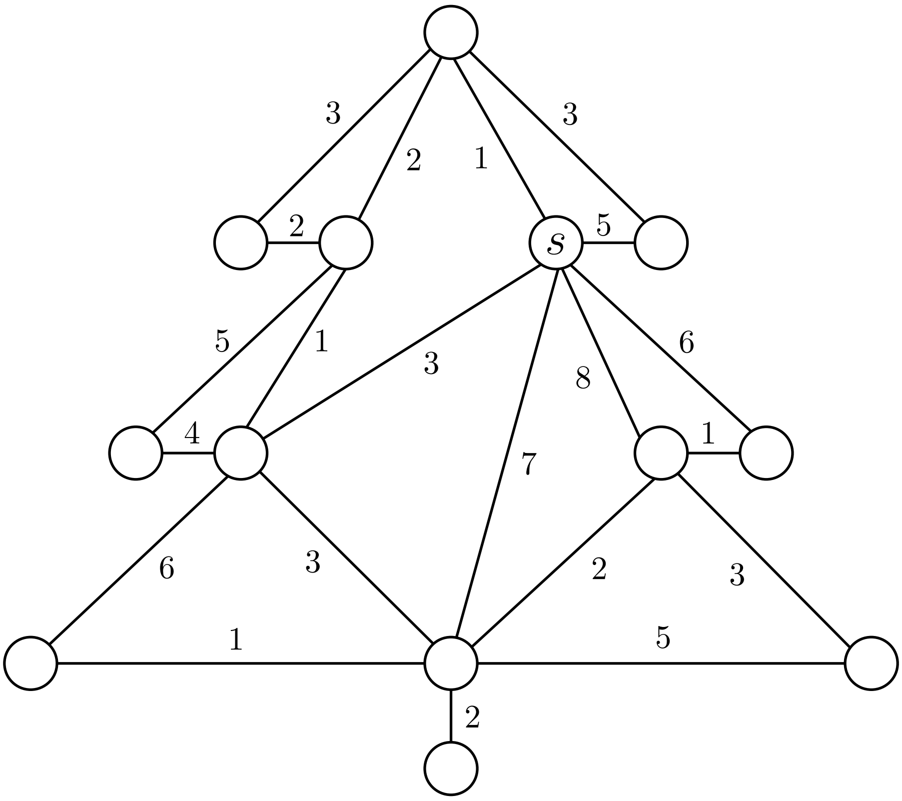
Initialization	s	a	b	c	d	e	f	g
$\delta(s,\cdot) =$	0	∞	∞	∞	∞	∞	∞	∞
1. Step ($u = s$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
2. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
3. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
4. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
5. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
6. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
7. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								
8. Step ($\boldsymbol{u =}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$								

Sample Solution

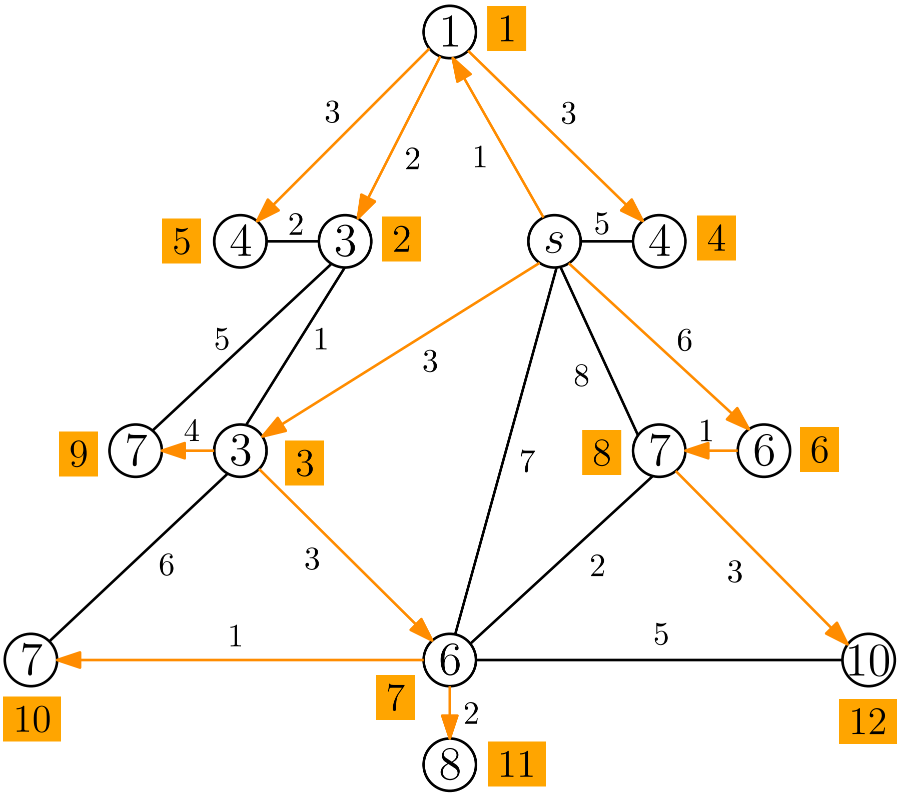
Initialisation	s	a	b	c	d	e	f	g
$\delta(s,\cdot) =$	0	∞	∞	∞	∞	∞	∞	∞
1. Step ($u = s$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	4	∞	8	∞	∞	∞
2. Step ($\boldsymbol{u = a}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	∞	8	∞	∞	∞
3. Step ($\boldsymbol{u = b}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	∞	∞	∞
4. Step ($\boldsymbol{u = d}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	∞	∞	∞
5. Step ($\boldsymbol{u = c}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	12	∞
6. Step ($\boldsymbol{u = e}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	11	15
7. Step ($\boldsymbol{u = f}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	11	12
8. Step ($\boldsymbol{u = g}$)	s	a	b	c	d	e	f	g
$\boldsymbol{\delta(s,\cdot) =}$	0	2	3	7	6	9	11	12

Exercise 4: More of Dijkstras' Algorithm

In the following graph execute Dijkstras' Algorithm starting from node s. Write the distance of each node to s into the respective node. Mark the order in which nodes are settled by the algorithm and mark all edges belonging to the *shortest path tree*.



Sample Solution



Enjoy the holidays and have a happy new year!