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

Algorithms Theory
Sample Solution Exercise Sheet 4

Due: Monday, 17th of December, 2018, 14:15 pm

Exercise 1: Priority Queues and Prim's Algorithm *(6+4 Points)*

(a) Assume we want to store $(key, data)$-pairs in a priority queue where the priorities (keys) are only from the set $\{1, \ldots, c\}$ and $c \in \mathbb{N}$ is constant.

Describe a priority queue that provides the operations `Insert`$(key, data)$, `Get-Min`, `Delete-Min`, and `Decrease-Key`$(pointer, newkey)$ all in constant time for the given scenario, and describe how these operations work on your data structure.

(b) State how fast Prim's algorithm to compute a minimum spanning tree is, under the assumption that edge weights are in the set $\{1, \ldots, c\}$ and $c \in \mathbb{N}$ is constant, using your implementation of a priority queue. Explain your answer.

Remark: *Assume you have a priority queue as in (a), even if you did not succeed in (a).*

Sample Solution

(a) We use an array $A[1, \ldots, c]$ of size c where each array entry contains a reference to a doubly linked list of $(key, data)$-pairs.

`Insert`$(key, data)$: When we insert a pair $(key, data)$ we simply append it to the list in $A[key]$ in $\mathcal{O}(1)$.

`Get-Min`: We iterate the Array starting from the beginning (in $\mathcal{O}(c) = \mathcal{O}(1)$), until we find a non-empty list at index i. We return the first pair $(i, data)$ from that list.

`Delete-Min`: We iterate the Array starting from the beginning (in $\mathcal{O}(c) = \mathcal{O}(1)$), until we find a non-empty list at index i. We remove the first pair $(i, data)$ from that list and return it.

`Decrease-Key`$(pointer, newkey)$: Since we have a pointer to the $(key, data)$-pair in question, we can remove and change its key in $\mathcal{O}(1)$. Afterwards we reinsert it into the correct list also in $\mathcal{O}(1)$

(b) Prim's Algorithm now runs in $\mathcal{O}(|E| + |V|)$ using our implementation of the priority queue.

The reason is that Prim's algorithm uses $\mathcal{O}(|E|)$ `Decrease-Key` operations and $\mathcal{O}(|V|)$ `Delete-Min`, `Get-Min` and `Insert` operations.

Exercise 2: Capacity Change in Flow Networks *(4+6 Points)*

We are given a maximum flow network $G = (V, E)$ with integer capacities together with a maximum flow Φ. Describe an algorithm with time complexity $\mathrm{O}(|V| + |E|)$ to compute a new maximum flow for each of the following cases:

(a) if the capacity of an arbitrary edge $(u, v) \in E$ increases by one unit.

(b) if the capacity of an arbitrary edge $(u, v) \in E$ decreases by one unit.

Sample Solution

(a) We just need to apply a single iteration of the Ford Fulkerson algorithm on the residual graph of the modified network to find a path along which the flow can be increased. If such a path is found in the residual graph, the path is augmented, and the maximum flow is increased by one unit. Otherwise, Φ is still the maximum flow. The residual graph search for finding an augmenting path takes time at most $O(|V| + |E|)$.

(b) If the previous maximum flow did not use the full capacity of (u, v), the capacity change does not influence the validity of Φ as the maximum flow. Otherwise, we modify the flow as follows. Since the flow over (u, v) cannot be more than (u, v)'s new capacity, we reduce the flow over (u, v) by one unit. Now the flow entering u is one unit more than the flow leaving it and the flow entering v is one unit less than the flow leaving it.

Primary solution: We search the residual graph for a path from u to v along which the flow can increase by one. This can be done in time $O(|V|+|E|)$. If there is such a path we augment the path, and Φ remains the maximum flow even after the capacity change. Otherwise, we must reduce the flow from both s to u and from v to t by one unit. We do so by finding two paths in the residual graph, one from u to s, and one from t to v, along each of which the flow can increase by one unit. There must be such paths, because we had a flow from s to t via (u, v). Augmenting these paths restores the condition $\Phi_{in}(u) = \Phi_{out}(u)$ and $\Phi_{in}(v) = \Phi_{out}(v)$. Then the value of the new max flow would be $|\Phi| - 1$. Here we perform a few searches in the residual graph, each takes time at most $O(|V| + |E|)$.

Alternative solution: We repair the condition $\Phi_{in}(u) = \Phi_{out}(u)$ and $\Phi_{in}(v) = \Phi_{out}(v)$ by reducing the flow from both s to u and from v to t by one unit. We do so by finding two paths in the graph G_b which consists only of the backward edges of the residual graph (c.f. lecture), along each of which the flow can increase by one unit. There must be such paths, because we had a flow from s to t via (u, v). Here we perform BFS searches in the residual graph, each takes time at most $O(|V| + |E|)$. However, it might be possible that there is an augmenting path that circumvents (u, v). If it exists, we find it in time $O(|V| + |E|)$ and augment the flow accordingly.

Exercise 3: Fibonacci Heaps *(10 Points)*

Show that for any positive integer n, there exists a sequence of Fibonacci Heap operations that can construct a Fibonacci Heap consisting of just one tree that is a linear chain of n nodes. Provide the pseudocode of a recursive procedure to construct such a Fibonacci Heap, and show its correctness.

Sample Solution

To construct a Fibonacci Heap which is a chain of n nodes, we call `Chain-Construction(`$n-1, n$`)`.

Algorithm 1 `Chain-Construction`(m, n)

```
if m = 1 then
    Initialize-Heap(F)                          ▷ assume F is now globally known
    Insert(F, n, null)                          ▷ inserting a node with key n and data null into F.
    Insert(F, n − 1, null)
    Insert(F, n − 2, null)
    Delete-Min(F)
    return
Chain-Construction(m − 1, n)
min ← Get-Min(F)
Insert(F, n + 1, null)
Insert(F, min.key − 1, null)
Insert(F, min.key − 2, null)
Delete-Min(F)
Decrease-Key(n + 1, min.key − 3)
Delete-Min(F)
return
```

Exercise 4: *Edge Minimal* Minimum Cut[1] *(4+6 Points)*

Consider an undirected, weighted graph $G = (V, E)$ with integral edge weights. Among all cuts of G with minimum weight you want to find a cut $(S, V \setminus S)$ with the smallest number of edges (i.e. edges with exactly one endpoint in S).

(a) Modify the weights of G to create a new graph G' in which any minimum cut in G' is a minimum cut with the smallest number of edges in G.

(b) Prove that G' has the property claimed in part (a).

Sample Solution

(a) Let $G = (V, E, w)$ be a weighted undirected graph with integer edge weights $w(e) \geq 0$ for $e \in E$. We define $G' = (V, E, w')$ with edge weights $w'(e) := |E| \cdot w(e) + 1$.

(b) Let $(S, V \setminus S)$ be a min-cut in G' and let F be the set of edges crossing the cut. Then the weight of the cut in G' is $\sum_{e \in F} w'(e) = \sum_{e \in F} \big[|E| \cdot w(e) + 1\big] = |F| + |E| \sum_{e \in F} w(e)$ and the weight of the corresponding cut in G is $w(S) := \sum_{e \in F} w(e)$.

We prove the claim by contradiction. At first assume that $(S, V \setminus S)$ is not a min-cut in G, i.e., there is a cut $(S', V \setminus S')$ in G with weight smaller than $w(S)$. Let F' be the edges crossing this cut. Then the cut in G' has weight

$$
\begin{aligned}
w'(S') = \sum_{e \in F'} w'(e) &= |F'| + |E| \sum_{e \in F'} w(e) \\
&\overset{*}{\leq} |F'| + |E|\big(-1 + \sum_{e \in F} w(e)\big) \\
&\leq |E| + |E|\big(-1 + \sum_{e \in F} w(e)\big) \\
&= |E| \sum_{e \in F} w(e) \;<\; |E| \sum_{e \in F} w(e) + |F| = w'(S).
\end{aligned}
$$

[1] This exercise will be considered as a bonus exercise, which earns points but does not count towards the threshold of exam admittance.

This is a contradiction to $(S, V\backslash S)$ being a minimum cut of G'. At $*$ we used that all capacities are integers and thus the weight of $(S', V\backslash S)$ is at least one smaller than the weight of the cut $(S, V\backslash S)$.

Now we know that $(S, V\backslash S)$ is a minimum cut of G. Now, assume that there is a minimum cut $(S', V\backslash S')$ of G that uses fewer edges than the cut S. Again, let F' be the edges crossing this cut. Then we can perform a similar calculation as above and we obtain that the cut S' in G' has weight

$$\begin{aligned} w'(S') &= \sum_{e \in F'} w'(e) = |F'| + |E| \sum_{e \in F'} w(e) \\ &\leq |F'| + |E| \sum_{e \in F} w(e) \\ &< |F| + |E| \sum_{e \in F} w(e) = w'(S). \end{aligned}$$

This is a contradiction to $(S, V \setminus S)$ being a minimum cut of G', which proves the claim.