University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithms and Datastructures
Winter Term 2022
Sample Solution Exercise Sheet 10

Due: Wednesday, July 3nd, 2pm

Exercise 1: Dijkstra's Algorithm *(10 Points)*

In the lecture we saw that the runtime of Dijkstra using Fibonacci heaps is $O(m + n \log n)$. Is this the actual runtime of the algorithm? Maybe our analysis is just not good enough! We will show that the analysis is indeed tight.

(a) Argue that any algorithm solving SSSP (Single Source Shortest Paths) must spend at least $\Omega(m)$ time. An intuitive explanation is sufficient. *(1 Point)*

(b) Proof that the Dijkstra algorithm determines shortest paths in a sorted order. For a source node v and any other two nodes $u \neq w$ the distance $d(v, u)$ will be marked before $d(v, w)$ if $d(v, u) < d(v, w)$. Give a formal proof. *(4 Points)*

(c) Proof that Dijkstras Algorithm needs $\Omega(n \log n)$ time if the algorithm is implemented using a comparison based heap. The idea is the following: reduce the problem of sorting n numbers to the SSSP problem. (Given n numbers in an array A create an instance of SSSP.) Give a precise description and a formal proof. *(5 Points)*

Sample Solution

(a) Suppose the algorithm ran in $T(m) \in o(m)$ time, then there exists an m_0 such that $T(m_0) \leq \frac{1}{2} m_0$[1]. As a result the algorithm will only see half of all edges. Now it is impossible for the algorithm to determine wether the shortest paths could be improved using the unknown edges.

(b) Dijkstra marks a node u once the correct shortest path to u is computed. This happens exactly when u is removed from the priority queue. So lets suppose v is the source node and $u \neq w$ are two other nodes such that $d(v, u) < d(v, w)$. We will prove that u gets marked first.
Since $d(v, u) < d(v, w)$ it follows that $d(v, p_1) \leq \ldots \leq d(v, p_k) < d(v, w)$ for the shortest path $(v = p_0, p_1, \ldots, p_k = u)$[2]
Seeing as p_1 is a direct neighbor of v, it will be inserted into the priority queue in the first step. Whenever a p_i gets marked p_{i+1} will immediately be added to the priority queue before another node gets removed from the priority queue. As a result: As long as $p_k = u$ is not marked itself, one of $\{p_1, \ldots, p_k = v\}$ is in the priority queue. Since $d(v, p_1) \leq \ldots \leq d(v, p_k) < d(v, w)$ each of these p_i would be chosen over w and as a result the entire path gets handled before w is considered. This immediately implies that $d(v, u)$ is determined before $d(v, w)$.

(c) Suppose Dijkstras algorithm runs in time $T_{Dij}(n)$ we will show that we can sort n numbers in time $O(n) + T_{Dij}(n + 1)$. Given n numbers in an Array A, we generate an instance of SSSP on a star graph. The middle node v will be the source node and to it we attach n nodes $v_0, \ldots, v_{n-1}$.

[1] This is a direct consequence of the $o(m)$ definition
[2] By optimality of subpaths from the lecture.

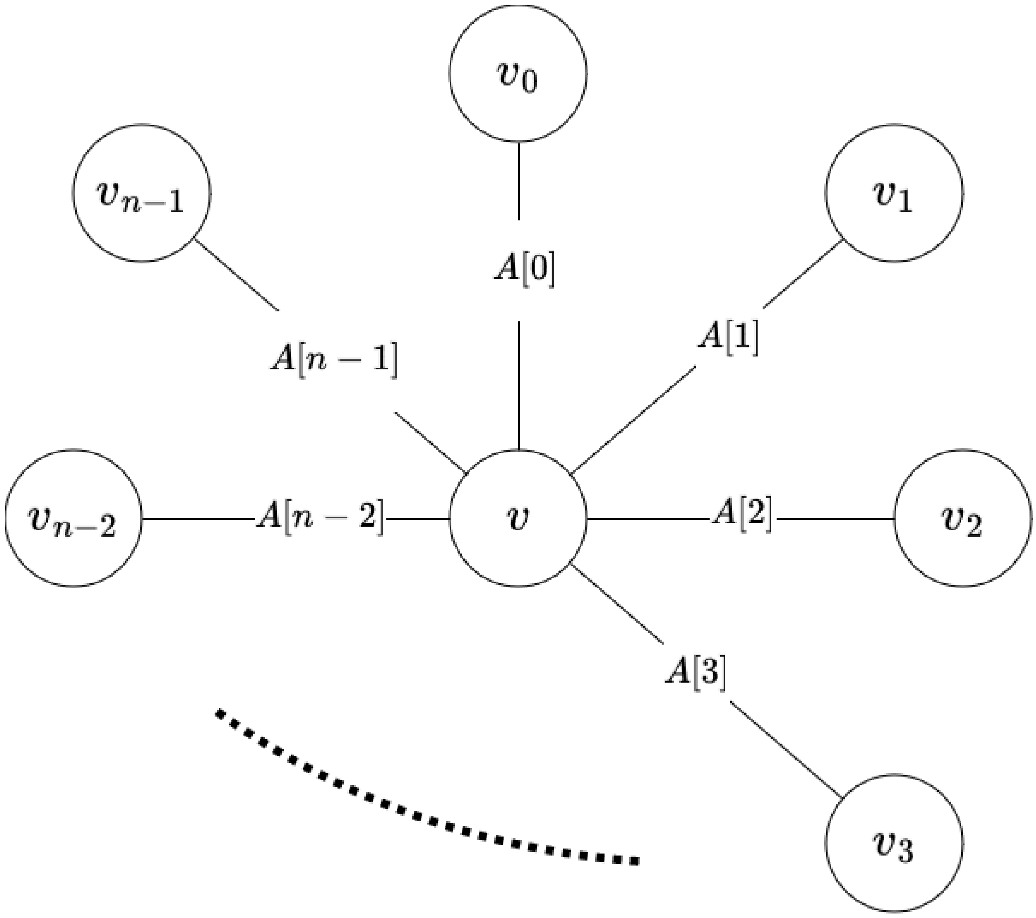

Figure 1: Die Konstruktion um das Sortieren von n Zahlen als SSSP Instanz zu lösen.

The weight of the edge $\{v, v_i\}$ will be exactly $A[i]$. Refer to Figure 1 for a visualisation. Clearly the shortest path from v to any v_i is to use the edge $\{v, v_i\}$ so $d(v, v_i) = A[i]$. Now we simply use the order in which the nodes are marked as the sorted order of the n numbers $A[i]$, by exercise b) this is a correct sorted order.
The SSSP Instance has $n + 1$ nodes and can be created in $O(n)$ time since it contains only n edges. Writing back the sorted values into A requires at most $O(n)$ extra time, so in total we have $O(n) + T_{Dij}(n + 1)$ time spent.
Now suppose that $T_{Dij}(n) \in o(n \log n)$ then the above construction gives a comaprison based sorting algorithm in $o(n \log n)$ contradicting the comparison based sorting lowerbound of $\Omega(n \log n)$. Therefore $T_{Dij}(n) \in \Omega(n \log n)$ as claimed.

Exercise 2: Currency Exchange *(10 Points)*

Consider n currencies $w_1, \dots, w_n$. The exchange rates are given in an $n \times n$-matrix A with entries a_{ij} ($i, j \in \{1, \dots, n\}$). Entry a_{ij} is the exchange rate from w_i to w_j, i.e., for one unit of w_i one gets a_{ij} units of w_j.
Given a currency w_{i_0}, we want to find out whether there is a sequence $i_0, i_1, \dots, i_k$ such that we make profit if we exchange one unit of w_{i_0} to w_{i_1}, then to w_{i_2} etc. until w_{i_k} and then back to w_{i_0}.

(a) Translate this problem to a graph problem. That is, define a graph and a property which the graph fulfills if and only if there is a sequence of currencies as described above. *(4 Points)*

(b) Give an algorithm that decides in $\mathcal{O}(n^3)$ time steps whether there is a sequence of currencies as described above. Explain the correctness and runtime. *(6 Points)*

Hint: It is $a \cdot b > 1 \iff -\log a - \log b < 0$

Sample Solution

(a) We define a weighted graph $G = (V, E, w)$ with $V = \{1, \dots, n\}$, $E = V^2$ (i.e., the graph is directed and complete) and $w(i,j) = a_{ij}$ (i.e., A is the adjacency matrix). A sequence of currencies as described exists if and only if there is a cycle $(i_0, i_1, \dots, i_k, i_0)$ such that

$$\prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) > 1 \ . \tag{1}$$

(b) In the adjacency matrix, we replace a_{ij} by $-\log a_{ij}$. That is, we define a graph $G = (V, E, w')$ with V and E as before and $w'(i,j) = -\log w(i,j)$. We run Bellman-Ford on G' with source i_0. This algorithm checks if G' contains a negative cycle, i.e., nodes $i_0, \dots, i_k$ with

$$\begin{aligned}
& \sum_{j=0}^{k-1} w'(i_j, i_{j+1}) + w'(i_k, i_0) < 0 \\
\iff \quad & \sum_{j=0}^{k-1} - \log w(i_j, i_{j+1}) - \log w(i_k, i_0) < 0 \\
\iff \quad & \sum_{j=0}^{k-1} \log w(i_j, i_{j+1}) + \log w(i_k, i_0) > 0 \\
\iff \quad & \log \left(\prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) \right) > 0 \\
\iff \quad & \prod_{j=0}^{k-1} w(i_j, i_{j+1}) \cdot w(i_k, i_0) > 1 \ .
\end{aligned}$$

So the algorithm checks property (1) from part (a). The runtime of Bellman-Ford is $\mathcal{O}(|V| \cdot |E|)$. With $|V| = n$ and $|E| = n^2$ we obtain a runtime of $\mathcal{O}(n^3)$.