Imperial College London

Department of Computing

CO-202 Algorithms II, Autumn 2018

Tutorial sheet 7

Notational Conventions. We denote weighted undirected graphs G by 3-tuples (V, E, w), where V is the set of vertices and $E \subseteq \binom{V}{2}$ is the set of edges. We denote the weight of an edge $e = \{v, u\}$ by $w(e) = w(\{v, u\}) = w(v, u)$. We denote directed graphs in the same way. A notable difference is that directed edges are 2-tuples (u, v); not sets $\{u, v\}$. Unless otherwise specified, all of graphs in this tutorial will be *simple*, i.e., they will not contain self-loops or multiple edges.

Exercise 1

Consider a country that consists of 5 towns $V = \{1, 2, 3, 4, 5\}$. The cost of constructing a road between towns i and j is w_{ij}. Given the costs

$$W = (w_{ij})_{i,j \in V} = \begin{pmatrix} 0 & 3 & 5 & 11 & 9 \\ 3 & 0 & 3 & 9 & 8 \\ 5 & 3 & 0 & \infty & 10 \\ 11 & 9 & \infty & 0 & 7 \\ 9 & 8 & 10 & 7 & 0 \end{pmatrix}$$

find the minimum-cost road network connecting the towns to each other.

Model Answer

The solution is the minimum spanning tree $T = (V, \{\{1, 2\}, \{2, 3\}, \{2, 5\}, \{4, 5\}\})$ of cost equal to 21.

Exercise 2

Let $G = (V, E)$ be a connected graph and let $S \subseteq E$ be a subset of its edges. Give an $O(|V| + |E|)$-time algorithm that decides whether there exists a spanning tree $T = (V, E_T)$ with $S \subseteq E_T$.

Model Answer

Use BFS to check whether there exists a cycle in the graph induced by the edges in S (like we did in Tutorial 6). This graph is $G_S = (V_S, S)$, where

$$V_S = \{u \in V \mid \exists\, e \in S : \text{the edge } e \text{ is incident on } u\}\,.$$

We can construct G_S in time $O(|V| + |S|) = O(|V| + |E|)$ and BFS runs in time $O(|V_S| + |S|)$, which is upper-bounded by $O(|V| + |E|)$.

We now need to prove that there is a spanning tree that contains all edges in S if and only if there are no cycles in G_S.

If there is a spanning tree T that contains all edges in S, then by the acyclicity property of the spanning tree there are no cycles in $G_S \subseteq T$.

Suppose now that there are no cycles in G_S. We will prove that there is a spanning tree that contains all edges in S. In the case where the original graph G is acyclic then we are done, as it is already a spanning tree (since G is connected). Suppose now that G contains cycles. As we know that there are no cycles induced by S, we can infer that any cycle C will contain at least one edge e out of S; if we remove e then we destroy C. If we apply this observation repeatedly, then we can destroy all cycles in G by removing only edges out of S. In the end, we have a spanning tree that contains all edges in S.

Exercise 3

Consider a weighted connected graph $G\,(V, E, w)$, with $w : E \to \mathbb{R}_{>0}$, and one of its minimum spanning trees (MSTs) $T = (V, E')$. Let $e \in E$ be an edge in G and suppose that the weight $w(e)$ of e changes to $\widehat{w}(e)$. Devise an algorithm that computes an updated MST $\widehat{T}$ in $O(|V| + |E|)$-time.

Model Answer

There are four cases here, depending on whether $e \in T$, or not, and whether $\widehat{w}(e) > w(e)$, or $\widehat{w}(e) < w(e)$. The minimum spanning tree T may only change to some tree $\widehat{T}$ if (a) $e \in T$ and $\widehat{w}(e) > w(e)$, or (b) $e \notin T$ and $\widehat{w}(e) < w(e)$.

In case (a), we remove $e = \{u, v\}$, thus breaking T into two trees T_1 and T_2. We can compute T_1 and T_2 in time $O(|V| + |E'|) = O(n)$, by running, say, BFS on u and v. In time $O(|E|)$ we can compute the minimum weight edge e' that connects T_1 to T_2, by making use of the `predecessor` function implicitly computed while running BFS. Here, for some vertex a, `predecessor`(a) yields either u or v, depending on whether a belongs to T_1 or T_2, respectively. Note that we can indeed compute `predecessor` implicitly while running BFS by modifying the original `predecessor` function to set `predecessor`$(b) \leftarrow$ `predecessor`(a), when visiting a vertex b from a vertex a, instead of the default assignment `predecessor`$(b) \leftarrow a$. Using `predecessor` we can check in $O(1)$ time whether an edge $\{a, b\} \in E$ connects T_1 to T_2, or not, as in the case where e connects T_1

to T_2 we have $\texttt{predecessor}(a) \neq \texttt{predecessor}(b)$ and $\texttt{predecessor}(a) = \texttt{predecessor}(b)$, otherwise. Finally, if $w(e') < \widehat{w}(e)$, we update T by adding e' and removing e; else, we keep T as it is. *Many thanks to student Harry (Halite) for bringing into our attention this elegant solution!*

In case (b), we add e to T, and compute in time $O(|V| + |E'|) = O(n)$, using, say, BFS, the cycle $C = (V_C, E_C)$ that emerges. In time $O(n)$ we can check if there is an edge $e' \in E_C$ such that $w(e') > \widehat{w}(e)$. If yes, then we add e to T and remove e' from T; else, we keep T as it is.

Exercise 4

Consider a country $G = (V, E, w)$ that consists of towns V, roads E between towns, and road lengths given by $w : E \to \mathbb{R}_{>0}$. For any $i, j \in V$, with $i \neq j$, let the *distance* $\texttt{dist}(i, j)$ denote the length of the shortest path from i to j. The government wants to decrease the distance between the capital town s and the most populous town t, namely $\texttt{dist}(s, t)$. To this end, they have put forward a set $E' \subseteq \binom{V}{2}$, with $E \cap E' = \emptyset$, of potential new roads to construct. As constructing roads is expensive, they want to construct only one road $e' \in E'$ that minimizes the distance $\texttt{dist}(s, t)$ over the choices of roads from E'.

1. Devise an $O(|E'| \cdot (|E| + |V| \log |V|))$-time algorithm that computes the optimal e' from E'.

2. Devise an $O(|E'| + |E| + |V| \log |V|)$-time algorithm that computes the optimal e' from E'.

Model Answer

1. In time $O(|E'| \cdot (|E| + |V| \log |V|))$ one can test every $e' \in E'$ for being optimal. To do that, create $G' = (V, E \cup e', w)$ and run Dijkstra (in time $O(|E| + |V| \log |V|)$) on input (G, s) to compute $\texttt{dist}(s, t)$. Finally, output the edge $e' \in E'$ that yields the minimum $\texttt{dist}(s, t)$.

2. The main idea behind the second solution is that we want to minimize $\texttt{dist}(s, u) + w(e') + \texttt{dist}(v, t)$ where $e' = \{u, v\} \in E'$. Observe that running once Dijkstra on (V, E, s) computes the distances $\texttt{dist}(s, u)$ for all $u \in V$. A similar observation allows us to compute the distances $\texttt{dist}(v, t)$ for all $v \in V$; namely by running Dijkstra on $\left(V, \widehat{E}, t\right)$, where $\widehat{E} = \{(u, v) \mid (v, u) \in E\}$. Finally, we compute in time $O(|E'| + |V|)$ the edge $e' = (u, v) \in E'$ that minimizes $\texttt{dist}(s, u) + w(e') + \texttt{dist}(v, t)$. The running time is $O(|E'| + |E| + |V| \log |V|)$, as the running time of Dijkstra is $O(|E| + |V| \log |V|)$.

Exercise 5

You possess a bank-note of $p \in \mathbb{N}$ pounds and you want to change it to coins. The available coins are coins of $c_1, c_2, \ldots, c_n \in \mathbb{N}$ pounds, where $1 \leq c_1 < c_2 < \cdots < c_n$. Devise an $O(n \cdot p + p \cdot \log p)$-time algorithm that outputs the minimum number k of coins needed to represent the amount p. *Note that repeating/reusing coins is acceptable.* For example, if $p = 10$, $c_1 = 1$, and $c_2 = 2$, then the answer is $k = 5$, as $5 \cdot c_2 = 10 = p$. However, if $p = 3$, then the answer is $k = 2$, as $c_1 + c_2 = 3 = p$. *Hint: Formulate the problem as a shortest path problem.*

Model Answer

We create a directed graph $G = (V, E)$, with

$$\begin{aligned} V &= \{0, 1, \ldots, p\}, \\ E_i &= \{(0, c_i + 0), (1, c_i + 1), (2, c_i + 2), \ldots, (p - c_i, c_i + (p - c_i))\} \\ &= \{(0, c_i), (1, c_i + 1), (2, c_i + 2), \ldots, (p - c_i, p)\}, \end{aligned}$$

for all $1 \leq i \leq n$, and

$$E = \bigcup_{i=1}^{n} E_i.$$

See Figure 1 for an example graph G. Then, we run Dijkstra's algorithm on G to compute the length of the shortest path from 0 to p. The running time is $O(|E| + |V| \cdot \log |V|) = O(n \cdot p + p \cdot \log p)$.

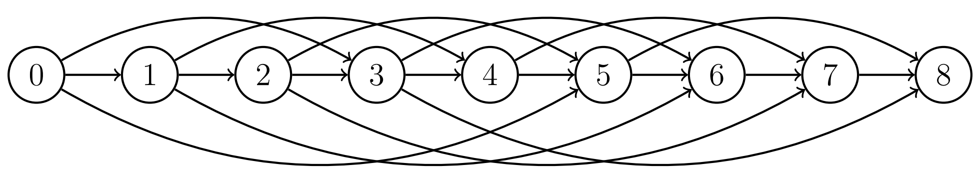


Figure 1: Example instance with $p = 8$, $c_1 = 1$, $c_2 = 3$, and $c_3 = 5$. Note that $p = 8 = 3 + 5 = c_2 + c_3$. Note also that there are two paths from 0 to 8 of length 2: Namely $(0, 3, 8)$ and $(0, 5, 8)$.

We now need to prove that there exists a path from 0 to p of length $k \geq 1$ if and only if there is a way to express the amount p as a sum of k coins from $\{c_1, c_2, \ldots, c_n\}$ *where repetitions of coins are allowed*, i.e., $p = \sum_{j=1}^{k} c_{\ell_j}$ with $1 \leq \ell_j \leq n$, for all $1 \leq j \leq k$.

Suppose that there is a path from 0 to p in G of length $k \geq 1$. We will prove by induction on k that there is a way of expressing the amount p as a sum of k coins from $\{c_1, c_2, \ldots, c_n\}$. For $k = 1$ we get that there is an edge $(0, p) \in E$, which implies that $p = 0 + c_\ell$, for some $1 \leq \ell \leq n$, by the construction of G. For the induction hypothesis, assume that, for some k, the existence of a length-k path that connects 0 to p implies that there is a way to express p as a sum of k coins from $\{c_1, c_2, \ldots, c_n\}$. We will show that it holds for $k+1$ as well. Consider a path $(0, i_2, \ldots, i_k, p)$ of length $k+1$. By the induction hypothesis we know that $i_k = \sum_{j=1}^{k} c_{\ell_j}$ with $1 \leq \ell_j \leq n$, for all $1 \leq j \leq k$. As the edge (i_k, p) is a part of the path $(0, i_2, \ldots, i_k, p)$ we get that $p = i_k + c_{\ell'}$, for some $1 \leq \ell' \leq n$, by the construction of G. Thus, $p = \sum_{j=1}^{k} c_{\ell_j} + c_{\ell'}$, which is what we want to show.

For the other direction, suppose that there is a way to express p as a sum of $k \geq 1$ coins from $\{c_1, c_2, \ldots, c_n\}$. Suppose that the k coins used are $\{c_{\ell_1}, c_{\ell_2}, \ldots, c_{\ell_k}\}$ with $1 \leq \ell_j \leq n$, for all $1 \leq j \leq k$. Then, the path

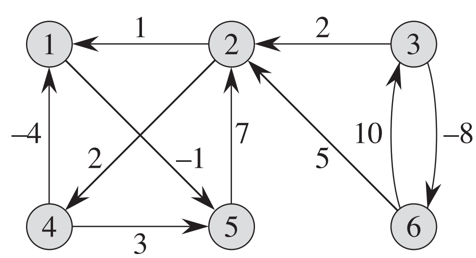
$$\left(0, c_{\ell_1}, c_{\ell_1} + c_{\ell_2}, \ldots, \sum_{j=1}^{k-2} c_{\ell_j} + c_{\ell_{k-1}}, \sum_{j=1}^{k-1} c_{\ell_j} + c_{\ell_k} = p\right)$$

belongs to G and connects 0 to p.

Exercise 6

(All-pairs shortest paths and the Floyd-Warshall algorithm)

1. Run the Floyd-Warshall algorithm on the weighted, directed graph depicted below. Show the matrix $D^{(k)}$ for each iteration of the algorithm.



2. The *transitive closure* of an unweighted directed graph $G = (V, E)$, denoted by $G^* = (V^*, E^*)$, is defined as the unweighted directed graph satisfying $V^* := V$ and

$$E^* := \{(i, j) : \text{ there exists a path between } i \text{ and } j \text{ in } G\}.$$

Using the Floyd-Warshall algorithm, design a procedure that on input a graph G computes its transitive closure G^* in time $O(|V|^3)$.

3. A cycle $(v_1, v_2, \ldots, v_k)$ is said to be a *negative weight cycle* if

$$\sum_{i=1}^{k} w(v_i, v_{i+1}) < 0,$$

where we set $v_{k+1} := v_1$. Show how to detect whether there exists a negative weight cycle in a graph in time $O(|V|^3)$ using the Floyd-Warshall algorithm.

Hint: Look at the entries $D_{ii}^{(k)}$ of the matrices produced while running the Floyd-Warshall algorithm.

Model Answer

1. The matrices obtained when running the Floyd-Warshall algorithm are the following:

$$D^{(0)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(1)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

$$D^{(3)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^{(5)} = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^{(6)} = \begin{bmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}.$$

2. Fix a graph $G = (V, E)$, and let $n := |V|$. Consider the weighted graph G' with weight function w satisfying $w(u, v) = 1$ whenever $(u, v) \in E$, and $w(u, v) = 0$ otherwise. Suppose we run the Floyd-Warshall algorithm on G' and obtain the matrix $D^{(n)}$. Then, there is a path from i to j in G if and only if $D^{(n)}_{ij} < n$. This can be checked for all pairs of vertices $i, j \in V$ in time $O(|V|^2)$. As a result, we can compute the transitive closure G^* of G in time $O(|V|^3 + |V|^2) = O(|V|^3)$, as desired.

3. Fix a weighted directed graph $G = (V, E)$, and let $n := |V|$. The existence of a negative-weight cycle can be checked by running the Floyd-Warshall algorithm on G and checking whether $D^{(n)}_{ii} < 0$ for some $i \in V$. In fact, we have $D^{(n)}_{ii} < 0$ if and only if there is some path in G starting and ending in i, i.e., a cycle, with negative total weight.

Exercise 7

(All-pairs shortest paths and Johnson's algorithm)

1. Use Johnson's algorithm to find the shortest paths between all pairs of vertices in the graph depicted in Exercise 6. Show the values of h and w' computed by the algorithm.

2. Let $G = (V, E)$ be a weighted, directed graph such that $w(u, v) \geq 0$ for all edges $(u, v) \in E$. How is the reweighted weight function w' related to w in this case?

3. When first learning about Johnson's algorithm, it is natural to wonder if there is a simpler way of reweighting the graph. For example, consider the following natural reweighting technique:

- Let $w^* := \min_{(u,v)\in E} w(u,v)$ be the minimum weight over all edges in a graph $G = (V, E)$. Then, we set $w'(u,v) := w(u,v) - w^*$ for all $(u,v) \in E$.

By the definition of w^*, we have $w'(u,v) \geq 0$ for all $(u,v) \in E$, so this reweighting could potentially be correct.

Give an example of a graph G for which this reweighting technique does not work. In other words, running Johnson's algorithm with the alternative (and simpler) reweighting technique does not recover the all-pairs shortest paths of G.

4. A cycle $(v_1, v_2, \ldots, v_k)$ in a graph $G = (V, E)$ is said to be a 0-*weight cycle* if

$$\sum_{i=1}^{k} w(v_i, v_{i+1}) = 0,$$

where we set $v_{k+1} := v_1$. Show that if a graph G contains a 0-weight cycle $(v_1, v_2, \ldots, v_k)$, then $w'(v_i, v_{i+1}) = 0$ for all edges (v_i, v_{i+1}) in the cycle.

Model Answer

1. By computing h and w' according to the reweighting function used by Johnson's algorithm, we obtain

$$\begin{aligned}
h(1) &= -5 \\
h(2) &= -3 \\
h(3) &= 0 \\
h(4) &= -1 \\
h(5) &= -6 \\
h(6) &= -8,
\end{aligned}$$

which leads to a new weight function w' satisfying

$$\begin{aligned}
w'(1,5) &= 0 \\
w'(2,1) &= 3 \\
w'(2,4) &= 0 \\
w'(3,2) &= 5 \\
w'(3,6) &= 0 \\
w'(4,1) &= 0 \\
w'(4,5) &= 8 \\
w'(5,2) &= 4 \\
w'(6,2) &= 0 \\
w'(6,3) &= 2.
\end{aligned}$$

2. Since $w(u,v) \geq 0$ for all $(u,v) \in E$, it follows that $h(u) = 0$ for all $u \in V$. Hence, we have $w'(u,v) = w(u,v)$ for all $(u,v) \in E$. This makes sense because in such a case we can simply run Dijkstra's algorithm between all pairs of vertices.

3. Simple example from the slides works here.

4. Let P_{i+1} be the shortest path from the node q to v_{i+1}. Then, we have that $h(v_{i+1})$ equals the length of P_{i+1}. Since we can follow the path P_{i+1} from q to v_{i+1} and then follow the cycle through $v_{i+1}, v_{i+2}, \ldots, v_k, v_1, \ldots, v_i$ to arrive at v_i, it follows that

$$h(v_i) \leq h(v_{i+1}) + \sum_{j \neq i} w(v_j, v_{j+1}).$$

Since we are in the presence of a 0-cycle, we have

$$\sum_{j \neq i} w(v_j, v_{j+1}) = -w(v_i, v_{i+1}).$$

Then,

$$\begin{aligned} w'(v_i, v_{i+1}) &= w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) \\ &\leq w(v_i, v_{i+1}) + h(v_{i+1}) - w(v_i, v_{i+1}) - h(v_{i+1}) \\ &= 0. \end{aligned}$$

Since we know $w'(v_i, v_{i+1}) \geq 0$, we conclude that $w'(v_i, v_{i+1}) = 0$, as desired.