Prof. Friedrich Eisenbrand
Martin Niemeier

Due Date: May 20, 2010
Discussions: May 06

Discrete Optimization

Spring 2010

Solutions 5

You can hand in written solutions for up to two of the exercises marked with (*) or (Δ) to obtain bonus points. The duedate for this is May 20, 2010, before the exercise session starts. Math students are restricted to exercises marked with (*). Non-math students can choose between (*) and (Δ) exercises.

Exercise 1
Let $T = (V, A)$ be a tree rooted in $r \in V$. Show that there are no two different paths from r to another node of the tree.

Solution
For a node $v \in T$, define the set of incoming arcs as

$$\delta^{in}(v) := \{(u, v) \, : \, u \in V, \, (u, v) \in A\}.$$

Note that by construction we have $\delta^{in}(v) \cap \delta^{in}(w) = \emptyset$ for all $v, w \in V$, $v \neq w$. Because T is a tree, there is a path starting in s and ending in v for each $v \in V$, $v \neq s$. We conclude that $|\delta^{in}(v)| \geq 1$.

Now assume that there is a node $t \in V$ such that there are two different $s-t$-paths P_1 and P_2 in T. Hence there is at least one arc $a' \in A$ such that $a' \in P_1$ but $a' \notin P_2$ (or vice versa: In case that $a' \in P_2$ and $a' \notin P_1$ we swap the roles of P_1 and P_2.

Now let v_k be the first node such that the paths are identical afterwards, i.e. the paths are of the form

$$P_1 = (s, a_1, v_1, \ldots, a_i, v_i, a_{i+1}, v_{i+1}, \ldots, v_{k-1}, a_k, v_k, a_{k+1}, v_{k+1}, \ldots, a_j, t)$$

and

$$P_2 = (s, a_1, v_1, \ldots, a_i, v_i, a'_{i+1}, v'_{i+1}, \ldots, v'_{l-1}, a'_l, v_k, a_{k+1}, v_{k+1}, \ldots, a_j, t),$$

where a_k and a'_l are different. (Note that possibly $s = v_i$ or $v_k = t$.) Moreover, $a_k \in \delta^{in}(v_k)$ and $a'_l \in \delta^{in}(v_k)$. We conclude that $|\delta^{in}(v_k)| \geq 2$. Hence we have

$$\sum_{v \in V, v \neq s} |\delta^{in}(v)| \geq |V|.$$

As seen previously, the $\delta^{in}(v)$ sets are disjoint. Hence $|A| \geq |V|$, in contradiction to the fact that T is a tree.

Exercise 2 ($*$)
Let $Q =< u_1,\dots,u_k >$ be the queue before an iteration of the **while** loop of the breadth-first-search algorithm. Show that $D[u_i]$ is monotonously increasing and that $D[u_1]+1 \geq D[u_k]$. Conclude that the sequence of assigned labels (over time) is a monotonously increasing sequence.

Solution
We prove that $D[u_i]$ is monotonously increasing and that $D[u_1]+1 \geq D[u_k]$ by induction on n, the iteration number of the while loop.

For $n = 0$, i.e. before the first iteration of the while loop, there is only the element s in the queue, thus the statement is trivially true.

Now assume that the statement holds before the n-th iteration of the while loop. We prove that it holds before the $n+1$-th iteration as well.

Let $< u_1,\dots,u_k >$ be the queue before the n-th iteration of the while loop. Hence $D[u_i]$ is monotonously increasing and $D[u_1]+1 \geq D[u_k]$. During the n-th iteration, the element u_1 will be removed and maybe some new elements are added. Thus the queue after the n-th iteration and before the $n+1$-th iteration looks as follows: $< u_2,\ u_3,\dots,u_k,u_{k+1},\dots,u_l >$. Observe that the algorithm will set $D[u_{k+1}] = D[u_{k+2}] = \cdots = D[u_l] = D[u_1]+1 \geq D[u_k]$, where the last $\geq$ relation is by induction hypothesis. Together that the fact that $D[u_2],\dots,D[u_k]$ is monotone, this shows that the whole sequence is monotone. Moreover $D[u_l] = D[u_1]+1 \leq D[u_2]+1$, since $D[u_1] \leq D[u_2]$ by induction hypothesis. This shows the second part of the claim.

Now let $u_1,\dots,u_k$ be the *full* sequence of the nodes in the order (over time) where they got assigned their labels. Assume that $D[u_1],\dots,D[u_k]$ is *not* monotonously increasing. Hence there are two nodes u_i and u_{i+1} such that $D[u_i] > D[u_{i+1}]$.

Let n be the iteration where u_{i+1} was put to the queue. Clearly u_i is not in the queue after iteration n, since otherwise the queue after iteration n would not have monotonously increasing labels in contradiction to what we have shown above. Hence u_i was dropped from the queue in iteration n, which implies that $D[u_{i+1}] = D[u_i]+1$, in contradiction to the assumption that $D[u_i] > D[u_{i+1}]$.

Exercise 3
There are n types of animals, and you want to assign them to two stables. Unfortunately, some animals would eat other animals when left unattended. Therefore you need to assign the animals carefully. There are m relations of the form "u eats v", where u and v are animals.

Find an $O(n+m)$ algorithm that decides whether there is an assignment of animals to the two stables such that no animal eats another one of the same stable, and outputs a feasible assignment.

Hint: Breadth-first-search might be useful.

Solution

Consider the following directed graph $D=(V,A)$, where each node in V corresponds to an animal, and we have an arc (u,v) for each relation "u eats v". Observe an assignment of the animals to the two stables is feasible if and only if the corresponding partition of the nodes into sets V_1 and V_2 we have that $(u,v)\notin A$ for all $u,v\in V_1$ and $(u,v)\notin A$ for all $u,v\in V_2$.

In other words, there is a feasible assignment if and only if the underlying undirected graph $G=(V,E)$, where $E:=\{\{u,v\} : (u,v)\in A\}$ is bipartite.

Consider the following algorithm:

```
1:  function FULLBFS($G=(V,E)$)
2:      $D[v] \leftarrow \infty \ \forall v\in V$.
3:      for all $v\in V$ do
4:          if $D[v]=\infty$ then
5:              $D[v]\leftarrow 0$
6:              BFS($G$, $v$, $D$)
7:          end if
8:      end for
9:      return $D$
10: end function
```

We run this algorithm on the graph G. Let D be the output. We define

$$V_1 := \{v\in V : D[v] \text{ is odd}\}$$

and

$$V_2 := \{v\in V : D[v] \text{ is even}\}.$$

We claim that the assignment given by V_1 and V_2 is a feasible solution for our problem if and only if there exists a feasible solution. With the observation from above, it is sufficient to show that V_1 and V_2 are feasible if and only if G is bipartite.

Trivially if V_1 and V_2 are feasible, then G is bipartite, since V_1 and V_2 give a bipartite partition of the nodes.

Now assume that V_1 and V_2 are infeasible, i.e. there is an edge $e\in E$ such that $e\subseteq V_1$ (or $e\subseteq V_2$. In the latter case we swap the roles of V_1 and V_2). We need to show that G is not bipartite. As seen in the lecture, it is sufficient to show that G contains an odd cycle.

Let $e=(u,w)$. Hence w is reachable from u and vice versa. Hence, u and w got their D-label assigned in the same run of BFS in Line 6. Thus there is a node v such that there is a v,u-path P of length $D[u]$ and a v,w-path P' of length $D[w]$. Since u and w both belong to V_1, we have $D[u]\equiv D[w] \bmod 2$, and therefore $D[u]+D[v]$ is *even*. Let $P\Delta P'$ be the set of edges that are contained in either P or P' (but not in both of them). Note that since $D[u]+D[v]$ is *even*, the number of arcs in this set even as well. Moreover e is contained in neither of the paths (otherwise the algorithm would not have assigned u and w to the same V_i).

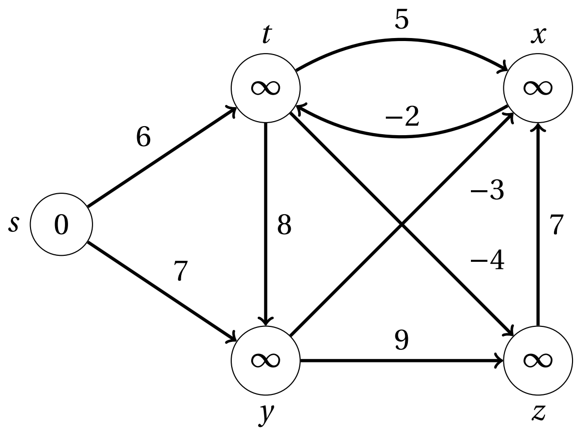
We conclude that $(P\Delta P') \cup \{e\}$ is an odd cycle in G.

Exercise 4

Consider the directed graph below with distance labels.

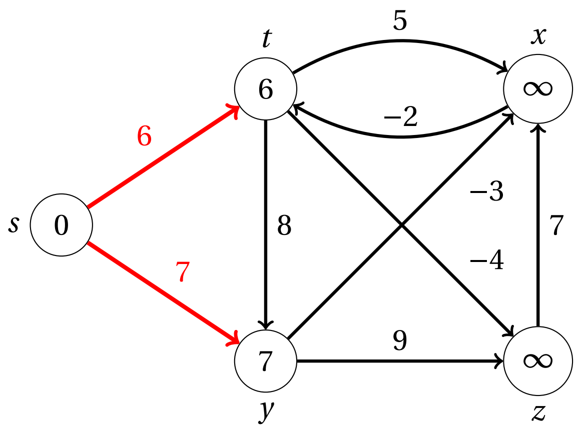
Perform the Bellman-Ford algorithm on this graph, using s as source vertex. For each $k = 1, \ldots, 5$: Draw the graph with the f_k values as node labels. For each node, mark an arc for which the minimum of step (ii) is achieved (if any).

The node labels for $k = 0$ are already given in the graph.

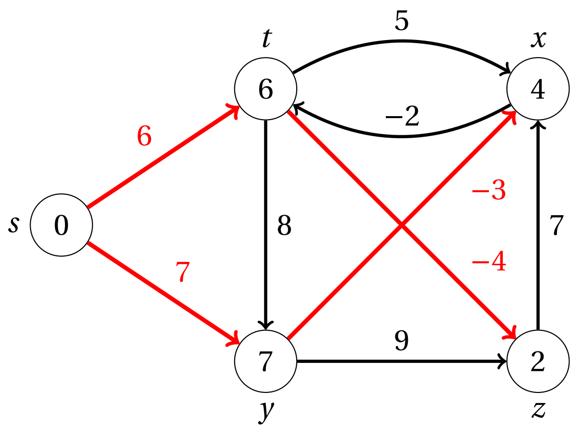


Solution

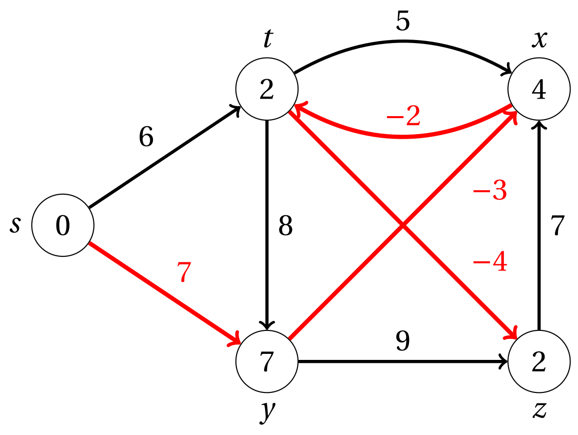
After 1. iteration:



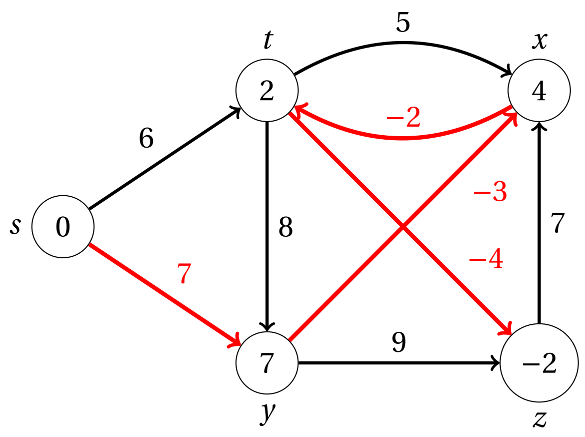
After 2. iteration:



After 3. iteration:



After 4. iteration:



Exercise 5 ($*$)
As discussed in the lecture, to compute shortest paths we require that our input graphs do not have cycles of negative length. In this exercise we will show how to detect negative cycles with the Bellman-Ford algorithm.

Let $D = (V, A)$ be a digraph, and $c\colon\ A \to \mathbb{R}$ a length function. Let $n := |V|$.

1. Let $f_k(v)$ be defined for all $k = 0, \dots, n$ and $v \in V$ as in the lecture. Show that D contains a cycle of negative length *reachable from the source node* s if and only if $f_n(v) \neq f_{n-1}(v)$ for some $v \in V$.

2. Consider the following modification of the Bellman-Ford algorithm: We maintain a *predecessor* vector $\Pi \in V^V$, i.e. for $v \in V$ we say that $\Pi(v) \in V$ is the *predecessor* of v.

   Initially we set $\Pi(v) := \text{INVALID}$ for all $v \in V$. Now in step (ii) of the algorithm, when calculating $f_{k+1}(v)$ and $f_{k+1}(v) = f_k(u) + c(u, v)$ for some $u \in V$, we set $\Pi(v) := u$.

   Assume that D has a negative cycle reachable from s. Show how to use this modified Bellman-Ford algorithm to find a cycle of negative length.

Solution

1. Let v be a node such that $f_n(v) \neq f_{n-1}(v)$. By construction we have thus $f_n(v) < f_{n-1}(v)$. As seen in the lecture $f_n(v)$ is the length of a shortest $s-v$-walk using at most n arcs. Hence, since $f_n(v) < f_{n-1}(v)$, this shortest $s-v$ path uses exactly n arcs, and hence contains a cycle. Let C be this cycle, and let k be the number of arcs in the cycle. Let $c(C)$ be the length of the cycle. Skipping this cycle yields a walk using $n-k$ arcs of length $f_n(v) - c(C)$. Hence $f_{n-k}(v) \leq f_n(v) - c(C)$. Thus

$$f_n(v) < f_{n-1}(v) \leq f_{n-k}(v) \leq f_n(v) - c(C)$$

   and we conclude that $c(C) < 0$.

   Now assume that $f_{n-1}(v) = f_n(v)$ for all $v \in V$. Let C be a cycle reachable from s, and let $v_1, \dots, v_k =: v_0$ be the nodes of the cycle such that $(v_{i-1}, v_i) \in C$ for all $i = 1, \dots, k$. As C is reachable, we have $f_{n-1}(v) < \infty$ for all $v \in C$.

   For each $i = 1, \dots, k$ we have

$$f_n(v_i) \leq \min\{f_{n-1}(v_i), \min_{(u,v_i)\in A} f_{n-1}(u) + c(u, v_i)\} \leq f_{n-1}(v_{i-1}) + c(v_{i-1,v_i}).$$

   Summing up these inequalities for the cycle, we get

$$\sum_{i=1}^{k} f_n(v_i) \leq \sum_{i=1}^{k} f_{n-1}(v_i) + c(C).$$

   As $f_{n-1}(v) = f_n(v)$ for all $v \in V$, we conclude $c(C) \geq 0$. Thus there is no cycle of negative length in G.

2. As proved in the lecture $f_i(v)$ is the length of a shortest $s-v$-walk using at most i arcs. Completely analogous one can show that $\Pi(v)$ is the (last) predecessor of v on a shortest $s-v$-walk using at most n arcs and we omit the details here.

   Not let v be a node such that $f_n(v) \neq f_{n-1}(v)$. Hence the shortest $s-v$-walk uses exactly n arcs, and thus contains a cycle. Hence we can use the Π vectors to traverse the path in reverse order, until we eventually run into the cycle. Note that the node v is not necessarily part of that cycle.

Exercise 6 (Δ)
Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of currency into more than one unit of the same currency. For example, suppose that 1 CHF buys 46.4 Indian rupees, 1 Indian rupee buys 2.5 Japanese yen, and 1 Japanese yen buys 0.0091 CHF. Then, by converting currencies, a trader can start with 1 CHF and buy $46.4 \times 2.5 \times 0.0091 = 1.0556$ CHF, thus turning a profit of 5.56 percent.

Suppose that we are given n currencies $c_1, c_2, \dots, c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i bys $R[i,j]$ units of curency c_j. Give an efficient algorithm to determine whether there exists a sequence of currencies $c_{i_1}, c_{i_2}, \dots, c_{i_k}$ such that

$$R[i_1,i_2] \cdot R[i_2,i_3] \cdots R[i_{k-1},i_k] \cdot R[i_k,i_1] > 1,$$

and computes such a sequence if it exists.

Hint: Exercise 5 might help.

Solution

Consider a sequence of currencies $c_{i_1}, c_{i_2}, \dots, c_{i_k}$. Due to monotonicity of the log function, the condition for a profitable cycle can be rewritten to:

$$\log(R[i_1,i_2]) + log(R[i_2,i_3]) + \cdots + log(R[i_{k-1},i_k]) + \log(R[i_k,i_1]) > 0,$$

or equivalently

$$-\log(R[i_1,i_2]) - log(R[i_2,i_3]) - \cdots - log(R[i_{k-1},i_k]) - \log(R[i_k,i_1]) < 0.$$

This motivates to consider a directed graph $D = (V, A)$ with arc lengths $c\colon A \to \mathbb{R}$ defined as follows:

The nodes of the graph correspond to the n currencies, i.e. $V = \{c_1, \dots, c_n\}$. Now, for each entry $R[i,j]$ in our exchange rate table, we have an arc $a = (c_i, c_j)$ with arc length $c(a) := -\log(R[i,j])$.

By construction and the observation from above, there is a profitable sequence of currencies if and only if D has a cycle of negative length. Deciding whether D contains a negative cycle and computing one can be done with the modified Bellman-Ford algorithm from exercise 5.