École Polytechnique Fédérale de Lausanne
Prof. Eisenbrand

May 3, 2018

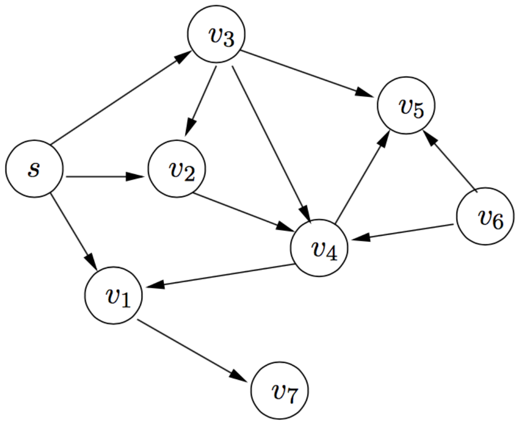
Discrete Optimization (Spring 2018)

Assignment 10

Problem 3 can be **submitted** until May 11, 12:00 noon, into the box in front of MA C1 563. You are allowed to submit your solutions in groups of at most three students.

Problem 1

In an unweighted graph (i.e., where all edges are of the unit weight) the shortest path from s to all other vertices can be computed by using the breadth-first search (BFS) algorithm. Apply it to the graph bellow.



Consider each iteration of the BFS (i.e. each time when a new vertex is taken from the head of the queue). Note which vertex has been currently processed, the distance labels and the snapshot of the queue at the end of the iteration.

Solution:

Distance labels will be denoted as a list of $[D[s], D[v_1], \dots, D[v_7]]$.

iteration	vertex	processed neighbors	distance labels								queue
			s	v_1	v_2	v_3	v_4	v_5	v_6	v_7	
			0	∞	∞	∞	∞	∞	∞	∞	$[s]$
1	s	v_1, v_2, v_3	0	1	1	1	∞	∞	∞	∞	$[v_1, v_2, v_3]$
2	v_1	v_7	0	1	1	1	∞	∞	∞	2	$[v_2, v_3, v_7]$
3	v_2	v_4	0	1	1	1	2	∞	∞	2	$[v_3, v_7, v_4]$
4	v_3	v_5	0	1	1	1	2	2	∞	2	$[v_7, v_4, v_5]$
5	v_7		0	1	1	1	2	2	∞	2	$[v_4, v_5]$
6	v_4		0	1	1	1	2	2	∞	2	$[v_5]$
7	v_5		0	1	1	1	2	2	∞	2	$[]$

Problem 2

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: Observe that the problem is equivalent to checking if the underlying (undirected) graph $G = (V, E)$ is bipartite, i.e., can V be partitioned into sets A and B so that there is no edge $\{u, v\} \in E$ such that $\{u, v\} \subseteq A$ or $\{u, v\} \subseteq B$? Modify the BFS algorithm seen in class so that it can be run even if G has several connected components (i.e. there is no path between each two nodes in G) and if G is undirected. Use this modified BFS to check if G is bipartite. Recall that a graph is bipartite if and only if it does not contain an odd cycle.

Solution:
Consider the underlying undirected graph $G = (V, E)$, where each node in V corresponds to an animal, and there is an edge $\{u, v\} \in E$ if "u eats v" or "v eats u". Observe that there exists a feasible assignment of the animals to the two stables is if and only if there is a partition of V into sets V_1 and V_2 such that for each $\{u, v\} \in E$ either $u \in V_1, v \in V_2$ or $v \in V_1, u \in V_2$, i.e, if and only if G is a bipartite graph.
We will extend the BFS algorithm to obtain a routine for checking if a given graph is bipartite, and prove the correctness of this routine. In the case of the positive answer, we will show how to construct a bipartition.
Consider the following algorithm:

```
function FULLBFS($G = (V, E)$)
    $D[v] \leftarrow \infty \ \ \forall v \in V$.
    for all $v \in V$ do
        if $D[v] = \infty$ then
            $D[v] \leftarrow 0$
            BFS($G$, $v$, $D$)
        end if
    end for
    return $D$
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 say that the assignment given by $V_1 \cup V_2$ is a feasible solution to our problem if and only if there is no edge $e \in E$ such that $e \subseteq V_1$ or $e \subseteq V_2$. With the observation from above, it is sufficient to show that $V_1 \cup V_2$ is feasible if and only if G is bipartite.
Trivially if $V_1 \cup V_2$ is feasible, then G is bipartite, since V_1 and V_2 give a bipartition of the nodes.
Now assume that $V_1 \cup V_2$ is 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. For that matter, 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 $D[u] + D[w]$ is *even.*
Consider the shortest path three (seen in class) of the BFS algorithm run on the vertex v. Both P and P' start in v and there is a unique vertex v' after which they split (it might be that $v = v'$). Let $P_{v'-u}$ be the portion of P from v' to u and analogously $P'_{v'-w}$ a portion of P' from v' to w. The number of edges in the union of $P_{v'-u}$ and $P'_{v'-w}$ is $D[u] - D[v'] + D[w] - D[v']$, so it is even.

Moreover e is contained in neither P nor P' (otherwise the algorithm would not have assigned u and w to the same V_i).
We conclude that the union of $P_{v'-u}$, $P'_{v'-w}$ and e is an odd cycle in G.

Problem 3 ($\star$)

Consider a directed graph $D = (V, A)$ with n vertices and m arcs. A toplogical sort of the vertices is a total ordering on V such that there is no arc $(u, v) \in D$ such that $u > v$, i.e, such that u is placed after v in the ordering.
Formulate an $O(m + n)$ algorithm that finds a topological sort of the vertices or decides that there is a directed cycle in G.

Solution:
Consider the following algorithm. Iterate through every vertex $v \in V$ and find a vertex that has zero indegree. If no such vertex exists we know that the graph has a directed cycle: construct a chain of $n + 1$ vertices starting at an arbitrary vertex and following the incoming arcs. Since there are only n vertices, by pigeon hole principle, we know that there is one vertex that appeared twice. The part of the chain between two of the vertice's occurences constitutes a directed cycle.
Once, we have found a vertex v with zero indegree, put this vertex first in the ordering and delete it from the graph. Then, recurse on the reduced graph. Clearly, this yields a feasible topological ordering.
Let us consider the running time of this algorithm. Checking if a given vertex has zero indegree can be done in constant time. Hence, finding a vertex with zero indegree can be done in $O(n)$ time. Deleting a vertex from the graph can be done in constant time. Since one vertex is deleted from the graph in each iteration, the number of iterations is n. Hence, the total running time is $O(n^2)$. But, we can speed up this process to arrive at an $O(n + m)$ algorithm by speeding up the time it takes to find a vertex with indegree zero.
Let $S \subseteq V$ be the set of vertices with indegree zero. We can find all $v \in S$ by scanning once through the set of vertices in $O(n)$ time. Then, in each iteration, one element v is taken from S and becomes the next element of the ordering. The only vertices that change their indegree when deleting v are the neighbors of v. Hence, when deleting v we check all of v's neighbors if they have zero indegree and if so, put them in S. The number of vertices that we check is the number of outgoing edges of v. Hence, there is an $O(n)$ preprocessing step to find the initial set of zero indegree vertices. Then, there is n iterations and the total number of degree checks in all iterations is the number of edges m. Therefore, the running time is $O(n + m)$.

Problem 4

Let $D = (V, A)$ be a directed *acyclic* graph, i.e., there exists no directed cycle in D, and let $w : A \to \mathbb{R}$ be arc weights. Assume that you are given a topological sort of the vertices. Show how the above ordering can be used to compute single-source shortest paths, with respect to w, in $O(m)$ arithmetic operations.

Solution:
One can compute a topological sort on the vertices as in Problem 3 if there are no negative cycles. Let $s \in V$ be the vertex that we want to compute the shortest paths from. Consider the vertices in the order of the topological sort, starting from s. For each vertex, relax all its outgoing edges. That means if we consider $v \in V$, then for all $(v, u) \in A$ set $d(u) = \min\{d(u), d(v) + w(v, u)\}$. Recall that there are no edges from v to u if u proceeds v in the ordering. Thus, the vertices on any path starting from s obey the ordering. Hence, the sizes of the shortest paths are found by considering each arc at most once.

Problem 5

Let $D = (V, A)$ be a directed graph, $w : A \to \mathbb{R}$ be arc weights and $s \in V$. Suppose that there exists a path from s to each other node of V.
Consider the folowing linear program:

$$\begin{aligned} \max \quad & \sum_{v \in V \setminus \{s\}} x_v \\ \text{s.t.} \quad & x_v - x_u \leq w(u,v), \quad \forall (u,v) \in A \\ & x_s \leq 0. \end{aligned} \tag{1}$$

Show the following:

a) This LP is feasible if and only if D has no negative cycle;

b) If D has no negative cycle, then (1) has a unique optimal solution.

Solution:

For each $v \in V$, denote with $d(s,v)$ the length of the shortest path from s to v in D, subject to w.

a) ($\Leftarrow$) If there are no negative cycles in D, then we have for each $(u,v) \in A$:

$$d(s,v) \leq d(s,u) + w(u,v) \Leftrightarrow d(s,v) - d(s,u) \leq w(u,v).$$

Thus, the assignment $x_v^* = d(s,v)$, $\forall v \in V$ with $x_s^* = 0$ is feasible for (1).

($\Rightarrow$) Let $\bar{x}$ be a feasible solution to (1) and let C be a directed cycle in D, denote its length with $w(C)$. One has:

$$w(C) = \sum_{(u,v) \in C} w(u,v) \geq \sum_{(u,v) \in C} (\bar{x}_v - \bar{x}_u) = 0.$$

The last equality comes from the fact that every vertex contained in C arises in the summation exactly twice, once with the positive and once with the negative sign.

b) Consider an arbitrary vertex $v \in V$ and let $s = u_0 \to u_1 \to \cdots \to u_k = v$ be a shortest length path from s to v, subject to w. By adding up the constraints $x_{u_1} - x_s \leq w(s, u_1)$ and $x_s \leq 0$ we obtain that $x_{u_1} \leq w(s, u_1)$ is valid for (1). Further adding $x_{u_2} - x_{u_1} \leq w(u_1, u_2)$ gives $x_{u_2} \leq w(s, u_1) + w(u_1, u_2)$. If we continue the process in the same manner, then we have that

$$\underbrace{x_v}_{x_{u_k}} \leq \sum_{i=1}^{k} w(u_{i-1}, u_i) = d(s,v)$$

For each $v \in V$, this inequality has to be satisfied by every feasible solution of (1). Thus, the maximum of $\sum_{v \in V \setminus \{s\}} x_v$ is uniquely attained by the solution

$$x_v^* = d(s,v), \quad \forall v \in V \setminus \{s\}.$$

Note that, if there is no negative cycle in D, there has to be an (outgoing) arc $(s,v) \in A$ such that $d(s,v) = w(s,v)$. Therefore, $x_s^* \geq x_v^* - w(s,v) = 0$, and by $x_s^* \leq 0$ one gets $x_s^* = 0$. As seen in part (a), the solution x^* is feasible for (1).

Problem 6

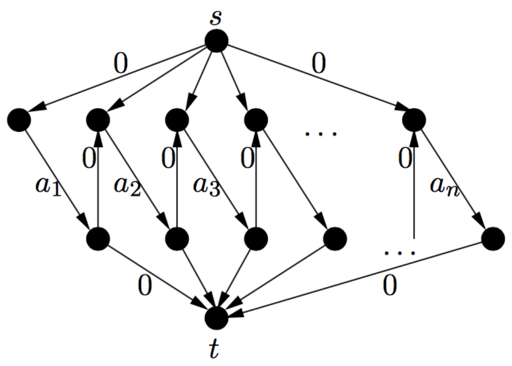
Given n numbers $a_1, \ldots, a_n$ find indices i and j, $1 \leq i \leq j \leq n$, such that $\sum_{k=i}^{j} a_k$ is minimized. We will develop two algorithms for this problem that run in linear time, *i.e.*, the number of (arithmetic) operations is linear in n.

(a) Solve the problem using Bellman-Ford as a subroutine. In particular, construct a graph such that a shortest path in this graph yields the optimal solution to the above problem. Show that the graph can be generated in linear time and that Bellman-Ford can be implemented to run in linear time on this graph.

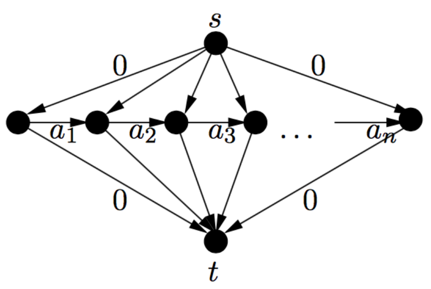
(b) Define $d(j) = \min_{1 \leq i \leq j} \sum_{k=i}^{j} a_k$. Conclude that the above problem is equivalent to computing $\min_{1 \leq j \leq n} d(j)$. Show how this can be done in linear time.

Solution:

(a) Consider the graph in Figure 1(a). The shortest path between s and t gives the respective



(a) General instance



(b) Instances with at least one negative number.

result. Clearly, this graph is acyclic and it has $O(n)$ edges so by Problem 4, one can compute the shortest $s - t$ path in $O(n)$ operations.

If there is at least one negative number, we can also consider the simpler graph in Figure 1(b). Note that instances with only non-negative numbers are not very interesting. Then, the solution is $i = j$ where a_i is the smallest number.

(b) It is easy to see that $d(j+1) = \min\{d(j) + a_{j+1}, a_{j+1}\}$. Setting $d(1) = a_1$ we can thus compute $d(2), \ldots, d(n)$ subsequently. Each successor computation takes constant time, hence the total computation takes $O(n)$ time.

The optimal pair (i, j) sums the numbers from a_i to a_j. Hence, this sequence ends at j and the value of $d(j) = \sum_{k=i}^{j} a_k$. Thus, after computing the values of d we can select the minimum value of the values in d which yields the optimal value. Again, this can be done in linear time.