Solutions to assignment 3

Exercise 1

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 0.7 British pound, 1 British pound buys 9.5 French francs, and 1 French franc buys 0.16 U.S. dollar. Then by converting currencies, a trader can start with 1 U.S. dollar and buy $0.7 \times 9.5 \times 0.16 = 1.064$ U.S. dollars, thus turning a profit of 6.4 percent.

Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i,j]$ units of currency c_j.

(a) Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ such that

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

Analyze the running time of your algorithm.

(b) Give an efficient algorithm to print out such a sequence if one exists. Analyze the running time of your algorithm.

Hint: $\prod_{i=1}^{k} x_i = e^{\sum_{i=1}^{k} \ln x_i}$. Furthermore, $\prod_{i=1}^{k} x_i > 1$ iff $1/\prod_{i=1}^{k} x_i < 1$.

Solution:

This problem can be interpreted as a graph problem: Each currency is a node and each possibility of exchange between two currencies is an edge. The edges are weighted by the exchange rate. The question is: Does there exist a cycle in the graph, such that the product of the edge weights is greater than 1? Since the known graph algorithms are designed to minimize the sum of the edge weights instead of maximizing their product, we use the following fact:

$$
\begin{aligned}
x_1 \cdot x_2 \cdots x_k &> 1 && (1)\\
\Leftrightarrow \frac{1}{x_1} \cdot \frac{1}{x_2} \cdots \frac{1}{x_k} &< 1 && (2)\\
\Leftrightarrow \ln(\frac{1}{x_1} \cdot \frac{1}{x_2} \cdots \frac{1}{x_k}) &< \ln(1) && (3)\\
\Leftrightarrow \ln(\frac{1}{x_1}) + \ln(\frac{1}{x_2}) + \cdots + \ln(\frac{1}{x_n}) &< 0 && (4)
\end{aligned}
$$

If the edge weights x are replaced by $\ln(\frac{1}{x})$, then the problem reduces to finding a negative cycle.

a) We can use a variant of the Floyd-Warshall-Algorithm for determining the existence of negative cycles:

(*continues on the next page*)

```
procedure hasNegCyc((V,E,c) : WeightedGraph) : boolean
    n := card(V)
    distance : Array [0,...n][0,...n] of Real

    // Fill up matrix with default values
    for i:=0 to n − 1 do
        for j:=0 to n − 1 do
            if (i, j) ∈ E then distance[i][j]:=c(i,j)
            else distance[i][j]:=+oo

    // Loop all nodes
    for k:=0 to n − 1 do
        // Loop all pairs of nodes
        for i:=0 to n − 1 do
            for j:=0 to n − 1 do
                // Can node k help in constructing the path from i to j ?
                if distance[i][j]>distance[i][k]+distance[k,j] then
                    distance[i][j] := distance[i][k]+distance[k][j]

    // Check out negative values at position (i,i)
    for i:=0 to n − 1 do
            if distance[i][i] < 0 then return true
    return false
```

This algorithm uses three nested loops over n nodes. Hence, its running time is $O(n^3)$.

b) In order to determine the nodes of the negative cycle, the above routine has to be modified such that it memorizes the shortest path:

```
procedure negCyc((V,E,c) : WeightedGraph) : List of List of Node
    n := card(V)
    distance : Array [0,...n][0,...n] of Real
    // This array tells where to go next in order to go from node i to node j
    nextNode : Array[0,...n − 1][0,...n − 1] of Node

    // Fill up matrices with default values
    for i:=0 to n − 1 do
        for j:=0 to n − 1 do
            if (i, j) ∈ E then
                distance[i][j]:=c(i,j)
                nextNode[i][j]:=j
            else
                distance[i][j]:=+oo
                nextNode[i][j]:=nil

    // Loop all nodes
    for k:=0 to n − 1 do
        // Loop all pairs of nodes
        for i:=0 to n − 1 do
            for j:=0 to n − 1 do
                // Can node k help in constructing the path from i to j ?
                if distance[i][j] > distance[i][k]+distance[k,j] then
                    distance[i][j] := distance[i][k]+distance[k][j]
                    nextNode[i][j] := nextNode[i][k]
```

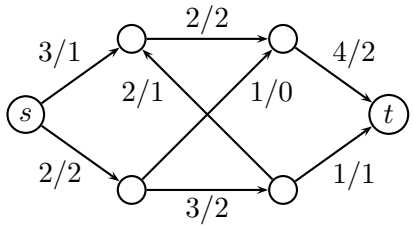
```
// Construct result
result : List of List of Node := ∅
// Check out negative values at position (i,i)
for i:=0 to n − 1 do
        if distance[i][i]<0 then
            negcyc : List of Node := < i >
            // Follow nextNodes until i has been reached again
            runnode := i
            repeat
                runnode := nextNode[runnode][i]
                negcyc.pushBack(runnode)
            until runnode==i
            result.pushFront(negcyc)

return result
```

The running time of this algorithm is still $O(n^3)$, because the nextNode-loop loops n times at a maximum.

Exercise 2

Consider the graph G and the flow f given on the right.



(a) Is f a blocking flow?

(b) Give the residual graph G_f.

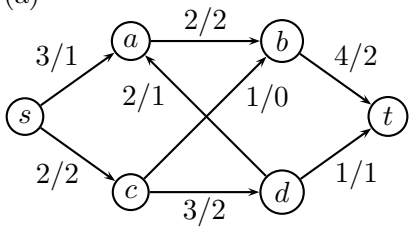
(c) Give the layered subgraph L_f of G_f.

(d) Find an augmenting path and give the resulting augmented flow. Repeat until the flow is maximum.

(e) Give a saturated (s,t)-cut for the maximum flow.

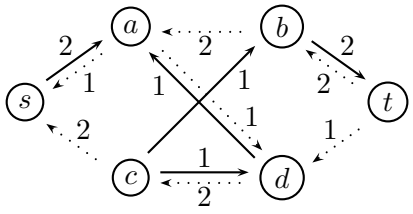
Solution:

(a)

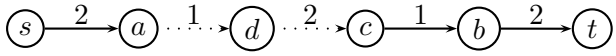


Every path from s to t matches either the pattern $\langle s, c, \ldots, t\rangle$ or the pattern $\langle s, a, b, \ldots, t\rangle$, where the edge (s, c) and the edge (a, b) are saturated. Hence, the flow f is *blocking* because on every path from s to t one edge is saturated.

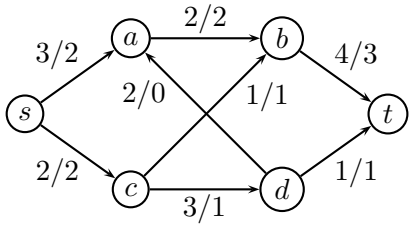
(b)



(c)



(d) augmenting path: $\langle s, a, d, c, b, t\rangle$, $\delta = 1$
resulting augmented flow:

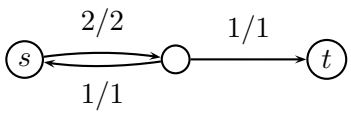


(e) $S = \{s, a\}$

Exercise 3
Which of the following claims are true and which are false. Justify your answer by giving either a (short) proof or a counterexample.

(a) In any maximum flow there are no cycles that carry positive flow. (A cycle $\langle e_1, \ldots, e_k\rangle$ carries positive flow iff $f(e_1) > 0, \ldots, f(e_k) > 0$.)

Solution
False. Counterexample:

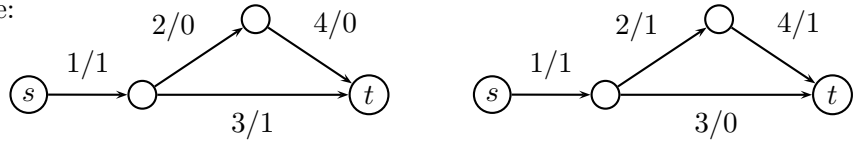


(b) There always exists a maximum flow without cycles carrying positive flow.

Solution
True. Let f be a maximum flow and let C be a cycle with positive flow. Let $\delta = \min_{e \in C} f(e)$. Reducing the flow of each edge in C by δ maintains the value of the flow and sets the flow $f(e)$ of at least one of the edges $e \in C$ to zero.

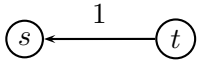
(c) If all edges in a graph have distinct capacities, there is a unique maximum flow.

Solution
False. Counterexample:



(d) In a directed graph with at most one edge between each pair of vertices, if we replace each directed edge by an undirected edge, the maximum flow value remains unchanged.

Solution
False. Counterexample:

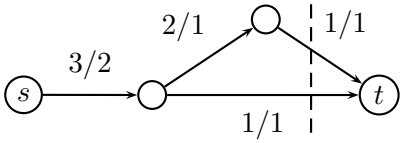


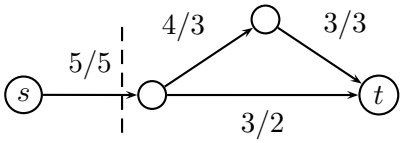
(e) If we multiply all edge capacities by a positive number λ, the minimum cut remains unchanged.

Solution
True. The value of every cut gets multiplied by λ. Thus, the relative order of cuts does not change.

(f) If we add a positive number λ to all edge capacities, the minimum cut remains unchanged.

Solution
False. Counterexample: $\lambda = 2$





(g) If we add a positive number λ to all edge capacities, the maximum flow increases by a multiple of λ.

Solution
False. In the example above, the maximum flow changes by 3.