Advanced Algorithms. Assignment 2

Exercise 3.
Let $G = (V, E)$ be an undirected(!) and connected graph with two distinguished nodes $s, t \in V$, and with weighted edges. We want to determine a set $F \subset E$ of edges, with minimum total weight, such that the removal of F disconnects s and t.
This looks almost like the Minimum Cut problem, however, G is undirected. A tempting idea is now to replace every undirected edge $e \in E$ with two opposite directed edges that have the same weight as e, and solve the Minimum Cut problem in the resulting directed graph G', taking the weights as edge capacities. This is plausible, but the construction might have unwanted side effects.

Prove that the proposed algorithm is indeed correct. That is, prove that a minimum cut in G' yields a minimum cut in G.

Give logically precise and complete arguments, without handwaving. Since equality of the minimum cut capacities c and c' (in G and G', respectively) must be shown, it might even be convenient to split the proof in two parts showing $c \leq c'$ and $c' \leq c$.

Find Exercise 4 on the reverse page.

Exercise 4.
The floor of a storage hall is partitioned like a chess board into black and white unit squares, say of 1×1 meters. Many of the squares are occupied by heavy objects that cannot be easily moved. That is, only a certain set F of the squares are free. Nothing special is assumed about the shape of F.
Now a number of boxes of size 1×2 meters shall be stored. Every box must be placed exactly on two neighbored free squares, but the boxes may be rotated (stand in N-S or W-E direction). The problem is to place as many boxes as possible on the given set F of squares.

4.1. Solve this problem efficiently, by reducing it to Maximum Flow.
Give a time bound. Most importantly, prove that you get an optimal number of boxes. (Remember that an equivalence proof has two directions. It is not enough to show that a maximum flow yields a maximum number of boxes; the converse is needed, too.)

4.2. Suppose that you have already computed an optimal solution (as it was done in 4.1), and later on, one more square becomes free. That is, one square is added to F somewhere. This provides a chance to place more boxes. In order to decide correctly whether now a better solution exists or not, you may, of course, re-compute an optimal solution from scratch. But can you update the solution more efficiently?
Give a method and a time bound.
Prove that your method cannot miss an optimal solution. Make sure that you consider all cases in the proof; it is easy to forget some. In particular, it must become clear how many further boxes you can add, and why.