cs473 Algorithms

Problem Set #4

Prof. Michael A. Forbes Due: *Fri., 2024-02-23 17:00*

Submissions must obey the following guidelines when using *reductions* to solve problems, in particular when using algorithms for computing maximum flows as a black-box.

- *Construction:* Submissions must completely describe the relevant capacitated graph by describing the vertices, the edges (with direction), and capacities. The source s and sink t must also be identified.
- *Forward reduction:* Submissions must describe an algorithm to, given the original input, construct the above mentioned capacitated graph. This includes a correctness and complexity analysis.
- *Backwards reduction:* Submissions must describe an algorithm to, given a maximum flow in the constructed capacitated graph, efficiently solve the original problem. This includes a correctness and complexity analysis. An argument for correctness will typically require two directions. First, if the resulting maximum flow is *large* in value, then the produced solution to the original problem is correct. Second, if the resulting maximum flow is *small* in value, then the produced solution to the original problem is correct.
- *Complexity:* The overall complexity of the entire algorithm must be specified as a function of the *original* input (**not** as a function of the constructed capacitated graph).
- Submissions should assume that maximum flows can be computed in $O(nm)$ time on graphs with n vertices and m edges, and in particular should **not** reproduce a maximum flow algorithm unless required.

Analogous guidelines exist for reductions to other problems whose algorithms were presented in lecture or auxiliary reading, such as shortest paths with negative edge lengths, or maximum bipartite matching.

All problems are of equal value.

1. Decreasing the flow. Kleinberg-Tardos Chapter 7, Problem #12.
2. Disaster allocation. Kleinberg-Tardos Chapter 7, Problem #9.
3. Give an analysis (correctness and complexity) of the following algorithm for maximum flow, expressing the runtime in terms of the number of vertices n, edges m, and sum of capacities $F = \sum_e c_e$.

```
rounding-FF(G = (V, E), (c_e)_{e∈E}):
    if all capacities zero, return the zero flow
    define capacities c' by (c')_e = ⌊c_e/2⌋, for e ∈ E.
    recursively compute f =rounding-FF(G,c')
    f_e ← 2 · f_e for e ∈ E.
    initialize the residual graph G^f
    while exists s ⇝ t path p in G^f
        augment f ← f + p
        update G^f ← G^{f+p}
    return f
```