CSOR W4246 - Summer 2021
Homework #3

Joseph High - `jph2185`

June 8, 2021

Problem 1

(i.) From the conditions provided in the problem, each node $v \in V$ must satisfy the following demand constraint:

$$f^{\text{in}}(v) - f^{\text{out}}(v) = d(v)$$

By the flow conservation condition for feasible flows: $f^{in}(v) = f^{out}(v)$

$$\implies \sum_{v\in V} f^{\text{in}}(v) = \sum_{v\in V} f^{\text{out}}(v) \iff \sum_{v\in V} f^{\text{in}}(v) - \sum_{v\in V} f^{\text{out}}(v) = 0$$

We then have that

$$\sum_{v\in V} d(v) \; = \; \sum_{v\in V} f^{\text{in}}(v) - f^{\text{out}}(v) \; = \; \sum_{v\in V} f^{\text{in}}(v) - \sum_{v\in V} f^{\text{out}}(v) \; = \; 0$$

$$\implies \sum_{v\in V} d(v) = 0$$

$$\implies \sum_{v\in V} d(v) = \underbrace{\sum_{d(v)>0} d(v)}_{=\sum_{v\in T} d(v)} + \underbrace{\sum_{d(v)<0} d(v)}_{=\sum_{v\in S} d(v)} + \underbrace{\sum_{d(v)=0} d(v)}_{=0 \;\; (\forall v\in V\setminus S\cup T)} = 0$$

$$\implies \sum_{v\in T} d(v) + \sum_{v\in S} d(v) + \cancelto{0}{\sum_{v\in V\setminus S\cup T} d(v)} = 0$$

$$\implies \sum_{v\in T} d(v) + \sum_{v\in S} d(v) = 0$$

$$\implies \sum_{v\in T} d(v) \; = \; -\sum_{v\in S} d(v)$$

Hence, a necessary condition for a feasible circulation with demands to exist is

$$\sum_{v \in T} d(v) \; = \; -\sum_{v \in S} d(v) \qquad \text{and/or} \qquad \sum_{v \in V} d(v) = 0$$

Furthermore, the demand at nodes in S or T must not exceed the capacity of the edges leaving or entering the nodes in S or T, respectively. However, this condition is captured by the combination of the capacity constraints and the demand constraints.

(ii.) Let $G = (V, E)$ be the flow network given in the problem. The reduction is as follows:
Input: (G, c, d)

- Input an instance (G, c, d) as an adjacency list encoded as a binary string x.
- G is the flow network/circulation.
- c is the function of capacity constraints for each edge in the graph.
- d is the function of demand constraints for each node in G.

Transform an instance $x = (G, c, d)$ to an instance $y = (G', c', d', s, t)$.

Reduction Transformation: From G, construct a new network $G' = (V', E')$ where $V' = V \cup \{s, t\}$ and $E' = E \cup E_N$. The node s is a new, additional source node in G' and t is a new, additional sink in G'. The new set of edges E_N is a set of directed edges from s to all nodes in the set S and directed edges from all nodes in T to t. That is, $E_N = \{(s, v), (u, t) : v \in S \text{ and } u \in T\}$. For each $(u, t) \in E_N$, with $u \in T$, assign a capacity $c_{(u,t)} = d(u)$. For each $(s, v) \in E_N$, with $v \in S$, assign a capacity $c_{(s,v)} = -d(v)$, since $d(v) < 0$ for all $v \in S$ and capacities must be positive.

Equivalence: For any feasible circulation in G, the maximum flow can be achieved in G' by assigning a flow value $f(u, t) = c_{(u,t)} = d(u)$ for all $(u, t) \in E_N$ and $u \in T$, and a flow value $f(s, v) = c_{(s,v)} = -d(v)$ for all $(s, v) \in E_N$ and $v \in S$. That is, assign flow values equal to the capacity of the corresponding edge. Then clearly, the max flow value will be $\sum_{v \in T} d(v)$. On the other hand, if the max flow is $\sum_{v \in T} d(v)$ in G', then this implies that every $e \in E_N$ is fully saturated at capacity (i.e., $f(e) = c_e$), which gives a feasible solution to the max flow problem on the original network G, and thus a feasible circulation in G.

Efficiency/Complexity: The input size $|x| = O(m \log n)$. The reduction transformation can be done in polynomial-time. The max-flow problem can be solved in polynomial-time if we use the Ford-Fulkerson algorithm (using BFS), and we only make one call the black box for Max Flow. Therefore, the original problem can be solved in polynomial-time. Hence, the reduction requires polynomial-time.

Problem 2

Note: My solution to this problem is inspired by experience with similar problems from previous optimization/financial engineering courses.

As stated in the problem, an arbitrage opportunity exists within currency exchange rates if

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] = \prod_{t=1}^{k-1} R[i_t, i_{t+1}] \cdot R[i_k, i_1] > 1$$

or equivalently,

$$\sum_{t=1}^{k-1} \ln R[i_t, i_{t+1}] + \ln R[i_k, i_1] > \ln(1) = 0 \iff -\sum_{t=1}^{k-1} \ln R[i_t, i_{t+1}] - \ln R[i_k, i_1] < 0$$

Then we can apply a modified version of either the Bellman-Ford algorithm or the identical dynamic programming algorithm presented in class to a graph with vertices that represent the currencies and edge weights equal to $-\ln R[i, j]$ for any two currencies c_i and c_j, where $i \neq j$ to detect whether or not the graph has negative cycles. In particular, we create a directed graph G with two anti-parallel edges between every pair of currencies, c_i and c_j, to represent the fact that an exchange can be done in either direction. More specifically, letting $v_i \in V$ represent currency c_i, the edge set is such that both $(v_i, v_j) \in E$ and $(v_j, v_i) \in E$, for all $i, j \in \{1, \ldots, n\}$ and $i \neq j$. That is, the graph is a complete directed graph in the sense that every pair of nodes is an in-neighbor and out-neighbor of one another. Finally, for every $(v_i, v_j) \in E$ let the associated edge weight be $w_{i,j} = -\ln R[i, j]$. The algorithm will solely be used to detect whether any negative cycles exist in the graph, and not to find a shortest path, and so a modified version of the Bellman-Ford algorithm will be used to accomplish this. If the algorithm detects a negative cycle, it will return `True` and `False` otherwise, which is the opposite of what the original Bellman-Ford algorithm returns for negative-cycle detection.

Pseudocode is provided on the next page (not enough room on this page).

Running Time: Developing the graph takes $O(n^2)$ time. The Bellman-Ford algorithm requires $O(nm)$ time, but $m = O(n^2)$ since the graph is complete; thus, Bellman-Ford requires $O(n^3)$ time. Therefore, the total running time of the algorithm is $O(n^3)$.

Correctness: If the graph contains negatives cycles, one run of the Bellman-Ford algorithm will correctly detect it since the graph is complete (all nodes are reachable from one another). More, correctness of negative cycle detection follows from correctness of the Bellman-Ford algorithm (as correctness has already been proven in class).

```
ArbitrageDetect(c = [c_1, c_2, ..., c_n], R):
    G = ExchangeRateGraph(c, R)                         // Construct graph G.
    Fix any s ∈ V
    Run Bellman-Ford-Modified(G, w, s)
end

Bellman-Ford-Modified(G, w, s):
    Note: Only the modified portion shown
    ⋮
    //Add the following to the end of the Bellman-Ford algorithm
    for any v ∈ V do
        if M[n, v] < M[n − 1, v] then
            return True                       // Indicative of negative cycles if true
        else return False                     // Returns false if no negative cycles
        end if
    end for
end

ExchangeRateGraph(c = [c_1, c_2, ..., c_n], R):      // Constructs graph.
    Declare G = (V, E, w)
    G.V = c
    for i = 1 to n do
        for j = 1 to n do
            G.E[i, j] = (G.V[i], G.V[j])
            G.w[i, j] = − ln R[i, j]
        end for
    end for
    return G = (V, E, w)
end
```

Problem 3

Let $A = \sum_{i=1}^{n} a_i$ and let $OPT(i, S_I, S_J)$ be a boolean indicator which indicates whether or not there exists a partitioning of the first i integers $a_1, \ldots, a_i$ into I, J, and K such that $\sum_{i \in I} a_i = S_I$ and $\sum_{j \in J} a_j = S_J$ and $\sum_{k \in K} a_k = A - S_I - S_J$. Let M be a DP table that represents the values of $OPT(i, S_I, S_J)$. Then $M[i, S_I, S_J] = 1$ if there exists a partitioning of the first i integers $a_1, \ldots, a_i$ into I, J, and K such that $\sum_{i \in I} a_i = S_I$ and $\sum_{j \in J} a_j = S_J$ and $\sum_{k \in K} a_k = A - S_I - S_J$, and $M[t, S_I, S_J] = 0$ otherwise.

Boundary conditions:

- $OPT(0, 0, 0) = 1$, since it is vacuously true that zero integers can be partitioned into three subsets such that the sum of each is equal to 0.
- $OPT(0, S_I, S_J) = 0$ for $S_I, S_J > 0$. Indeed, it is impossible to partition zero integers such that the sum of at least one partition is a positive integer.

Recursion:

$$OPT(i, S_I, S_J) = OPT(i-1, S_I - a_i, S_J) \textbf{ OR } OPT(i-1, S_I, S_J - a_i) \textbf{ OR } OPT(i-1, S_I, S_J)$$

The algorithm recursively builds to the desired output of $M\left[n, \frac{A}{3}, \frac{A}{3}\right]$, where M is filled in from the top down (row-by-row).

```
Partition-Sum(a = [a_1, ..., a_n]):
    Set A = sum_{i=1}^{n} a_i
    Declare DP table M
    Initialize M[0,0,0] = 1                                  // Boundary condition
    for (i ≥ 1) and (j ≥ 1) do                               // Boundary conditions
        Initialize M[0,i,j] = 0
    end for
    if A/3 ∉ ℤ then
        return False
    end if
    for i = 1 to n do
        for j = 1 to A/3 do
            for k = 1 to A/3 do
                M[i,j,k] = M[i-1, j-a_i, k] OR M[i-1, j, k-a_i] OR M[i-1, j, k]
            end for
        end for
    end for
    if M[n, A/3, A/3] == 1 then
        return True
    else return False
    end if
end
```

Space: The algorithm has an additional space requirement of $\Theta(nA^2)$ for the DP matrix M.

Running Time: The sum at line 2 requires $O(n)$ time. Initializing M requires $O(nA^2)$ time. The recursion iterates $O(nA^2)$ times, each of which takes constant time since each of the values were computed in earlier steps. Therefore, the total running time of the algorithm is then $O(nA^2)$.

Problem 4

This decision/feasibility problem can be transformed to a Max-Flow problem by modeling the given parameters on a flow network and subsequently solving the Max-Flow problem on the constructed flow network. In particular, construct a directed graph $G = (V, E)$ where V consists of a vertex for each injured person, which we denote by v_i, $i = 1, \ldots, n$; a vertex for each hospital, which we denote by h_j, $j = 1, \ldots, k$; and a source node, and a sink node. For each person node v_i add a directed edge from v_i to all h_j that the person v_i is able to arrive at within the half-hour driving time from their current location to the edge set E. For all $j = 1, \ldots, k$, add a directed edge from hospital node h_j to the sink node t. For each $i = 1, \ldots, n$, add a directed edge from the sink node to injured patient node v_i. Letting c be the capacity constraint function, the following edge capacities will be set:

- For each edge from the source to injured person nodes, set the capacity to be $c(s, v_i) = 1$ (Since each patient adds at most one to the count.)
- For each edge (v_i, h_j) (from person i to hospital j), set the capacity to be $c(v_i, h_j) = 1$ (To constrain the source node to sending at most one unit of flow from $s \to v_i \to h_j \to t$ for a given pair i, j.)
- For each edge from hospital j to the sink node, set the capacity to be $c(h_j, t) = \lceil \frac{n}{k} \rceil$ (To capture hospital load constraint.)

Solve `Max Flow`(D) on input $(G = (V, E), s, t)$; answer **yes** if $\max |f| = n$ and **no** if $\max |f| < n$.

On the flow network constructed above, run the Ford-Fulkerson algorithm to solve the Max Flow problem. If the maximum flow value is equal to n, then this indicates that all n injured people were brought to a hospital. Otherwise, if the max flow less than n, then it is not feasible for all n people to be sent to the k hospitals. Accordingly, if $\max |f| = n$ the algorithm returns `True`; otherwise, the algorithm returns `False`.

`Hospital-Flow`$(P = \{p_1, \ldots, p_n\}, H = \{h_i, \ldots, h_k\}, L = \{l_1, \ldots, l_n\})$:
  //L denotes the set of n locations for the n people.
  Construct flow network on input (P, H, L) // Using construction described above
  Let G be the constructed flow network
  Run `Ford-Fulkerson`(G)
  if $\max |f| == n$ **then**
    return `True`
  else if $\max |f| < n$ **then**
    return `False`
  end if
end

Correctness/Equivalence: If the $\max |f| = n$, this implies that all n injured people are able to be sent to a hospital. Indeed, $|f| = f^{in}(t)$, and since there is exactly one edge going from each hospital to t, this implies that when the max flow is achieved, the hospitals received all n patients. On the other hand, if there exists a flow f' in the network in which all n people can be sent to a hospital, the maximum flow can be achieved by saturating all edges leaving the source, and subsequently send flow down f' to the sink node. Hence, the Max-Flow problem is a yes instance if and only if the feasibility problem of sending patients to hospitals is a yes instance.

Running Time: The flow network has $|V| = n + k$ nodes and contains at most $nk + n + k$ edges; indeed, there are exactly n edges from the source node and injured people, at most nk edges from people to hospitals, and k nodes from hospitals to the sink node. Therefore, constructing the flow network requires $O(nk)$ time. Running Ford-Fulkerson (using BFS) requires $O(|V| \cdot |E|) = O((n + k) \cdot nk) = O(n^2k + nk^2)$ time. Therefore, the total running time of the algorithm is $O(n^2k + nk^2)$. Hence, the algorithm/reduction requires polynomial time.

Problem 5

Unfortunately, I did not have time to complete Problem 5.