CSE530: Algorithms & Complexity
Solutions to Exercise Set 2

Antoine Vigneron

March 30, 2018

Some of the exercises are taken from the textbook *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein.

1. For each of the relations below, write whether it is TRUE (T) or FALSE (F).

- $n^2 = O(n^2)$ TRUE
- $n^2 = o(n^2)$ FALSE
- $n^2 = \Omega(n^2)$ TRUE
- $n^2 = \Theta(n^2)$ TRUE
- $5 = O(1)$ TRUE
- $5 = O(0)$ FALSE
- $(-1)^n = O(1)$ TRUE
- $n^3 + 2n + 4 = O(n^4)$ TRUE
- $n^3 = O(n^3 - n + 2)$ TRUE
- $2^{n+\log n} = O(2^n)$ FALSE

2. Is the statement below true? Justify your answer.

- "If f is an increasing function and $f(0) = 0$, then $f(2n) = O(f(n))$"

Answer. No, it is not true. Take for instance $f(n) = n2^n$, which is an increasing function and satisfies $f(0) = 0$. So we have $f(2n) = n2^{2n+1}$. For sake of contradiction, suppose that $f(2n) = O(f(n))$. Then there exist constants n_0 and $c > 0$ such that $f(2n) \leqslant cf(n)$ for all $n \geqslant n_0$. So we have $n2^{2n+1} \leqslant cn2^n$, which is equivalent to $2^{n+1} \leqslant c$ for all $n \geqslant n_0$. This is not possible because $2^{n+1} \to \infty$ and c is constant.

3. Consider the Algorithm 1. Suppose that one execution of each line 2, 3, 4, 5 and 6 takes constant time c_2, c_3, c_4, c_5 and c_6, respectively. Give the running time $T(n)$ of Algorithm 1 as a function of n and these five constants, assuming that it returns NOT FOUND. Then express this running time using the Θ notation.

Algorithm 1

```
1: procedure DUPLICATE(A[1...n])
2:     for i ← 1, n − 1 do
3:         for j ← i + 1, n do
4:             if A[i] = A[j] then
5:                 return (i, j)
6:     return NOT FOUND
```

Answer. Line 2 is executed n times. The number of times line 3 is executed is

$$\sum_{i=1}^{n-1} n+1-i = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 = \frac{n^2}{2} + \frac{n}{2} - 1,$$

the number of times line 4 is executed is

$$\sum_{i=1}^{n-1} n-i = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2},$$

line 5 is never executed because we return NOTFOUND, so the running time is

$$\begin{aligned} T(n) &= c_2 n + c_3 \left(\frac{n^2}{2} + \frac{n}{2} - 1 \right) + c_4 \frac{n(n-1)}{2} + c_6 \\ &= \frac{c_3 + c_4}{2} n^2 + \frac{2c_2 + c_3 - c_4}{2} n - c_3 + c_6 \\ &= \Theta(n^2). \end{aligned}$$

4. You are given k types of coins with values $v_1, \ldots, v_k \in \mathbb{N}$ and a cost $C \in \mathbb{N}$. You may assume $v_1 = 1$ so that it is always possible to make any cost. You want to find the smallest number of coins required to sum to C exactly. For example, assume you have coins of values 1, 5, and 10. Then the smallest number of coins to make 26 is 4: take 2 coins of value 10, 1 coin of value 5, and 1 coin of value 1.

Give an algorithm for this problem. Its running time should be polynomial in $n = \max(k, C)$.

Answer. Let $S(C)$ denote the smallest number of coins needed to make cost C. Then it satisfies the relation

$$S(C) = \begin{cases} 0 & \text{if } C = 0 \\ 1 + \min\{S(C - v_j) \mid 1 \leqslant j \leqslant k \text{ and } v_j \leqslant C\} & \text{if } C > 0. \end{cases}$$

Based on this recurrence relation, we can compute $S(C)$ in $O(kC) = O(n^2)$ time with the following algorithm.

5. Using the definitions of flows and flow networks on Slides 5 and 6, Lecture 5, show the following: If $(u, v) \notin E$ and $(v, u) \notin E$, then $f(u, v) = 0$.

Answer. As $(u, v) \notin E$, we have $c(u, v) = 0$. So the capacity constraint implies that $f(u, v) \leqslant 0$.

As $(v, u) \notin E$, we also have $c(v, u) = 0$, so $f(v, u) \leqslant 0$. By skew symmetry, it implies that $f(u, v) = -f(v, u) \geqslant 0$.

We have just proved that $0 \leqslant f(u, v) \leqslant 0$, thus $f(u, v) = 0$.

Algorithm 2

```
1: procedure CHANGE(C, V[1...k])                    ▷ V[i] is v_i
2:     S[0...C] ← new array
3:     S[0] ← 0
4:     for i ← 1, C do
5:         S[i] ← 1 + S[i − 1]
6:         for j ← 2, k do
7:             if v_j ⩽ i then
8:                 S[i] ← min(S[i], 1 + S[i − v_j])
9:     return S[C]
```

6. Let f be a flow network $G(V, E)$ and let α be a real number. The *scalar flow product* αf is a function from $V \times V$ to $\mathbb{R}$ defined by

$$(\alpha f)(u, v) = \alpha \cdot f(u, v).$$

Prove that the flows in a network form a *convex set*. That is, show that, if f_1 and f_2 are flows, then $\alpha f_1 + (1 - \alpha) f_2$ is a flow for any $0 \leqslant \alpha \leqslant 1$.

Answer. We need to check that $f = \alpha f_1 + (1 - \alpha) f_2$ obeys the three flow properties: the capacity constraint, the skew symmetry, and the flow conservation.

- Capacity constraint: For any $u, v \in V$, we have $f_1(u, v) \leqslant c(u, v)$. Since $c(u, v) \geqslant 0$, it implies that $\alpha \cdot f_1(u, v) \leqslant \alpha \cdot c(u, v)$. Similarly, we have $(1 - \alpha) f_2(u, v) \leqslant (1 - \alpha) c(u, v)$. Thus

$$\begin{aligned} f(u, v) &= \alpha \cdot f_1(u, v) + (1 - \alpha) f_2(u, v) \\ &\leqslant \alpha \cdot c(u, v) + (1 - \alpha) c(u, v) \\ &= c(u, v). \end{aligned}$$

- Skew symmetry: For any $u, v \in V$

$$\begin{aligned} f(v, u) &= \alpha \cdot f_1(v, u) + (1 - \alpha) f_2(v, u) \\ &= -\alpha \cdot f_1(u, v) - (1 - \alpha) f_2(u, v) \qquad \text{(by skew symmetry of } f_1 \text{ and } f_2\text{)} \\ &= -f(u, v). \end{aligned}$$

- Flow conservation: For all $u \in V \setminus \{s, t\}$,

$$\begin{aligned} \sum_{v \in V} f(u, v) &= \sum_{v \in V} \alpha \cdot f_1(u, v) + (1 - \alpha) f_2(u, v) \\ &= \alpha \sum_{v \in V} f_1(u, v) + (1 - \alpha) \sum_{v \in V} f_2(u, v) \end{aligned}$$

  By flow conservation of f_1 and f_2, we have $\sum_{v \in V} f_1(u, v) = \sum_{v \in V} f_2(u, v) = 0$, so it follows that $\sum_{v \in V} f(u, v) = 0$.

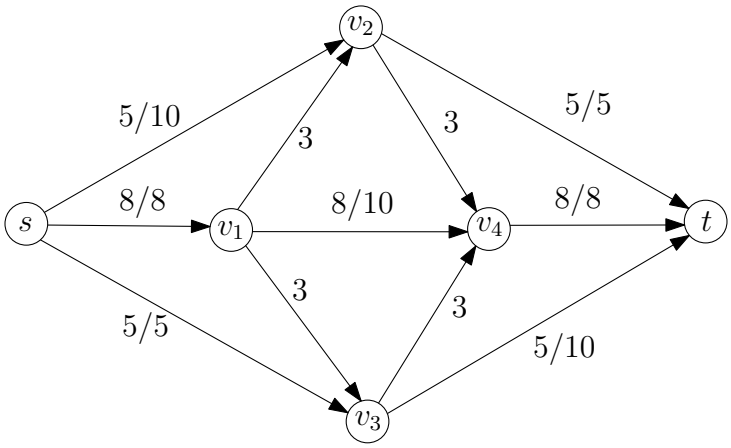


Figure 1: The flow network G and the flow f from Question 7.

7. Figure 1 shows a flow network G on which a flow f has been computed. The notation is the same as in Lecture 5: for instance, the label 5/10 on edge (s, v_2) means that $f(s, v_2) = 5$ and $c(s, v_2) = 10$.

(a) Is f a maximum flow? Justify your answer.

(b) Find a minimum cut in G. Justify your answer.

Answer a. No, f is not a maximum flow. The flow in Figure 2 has value 21, while the value of f is only 18.

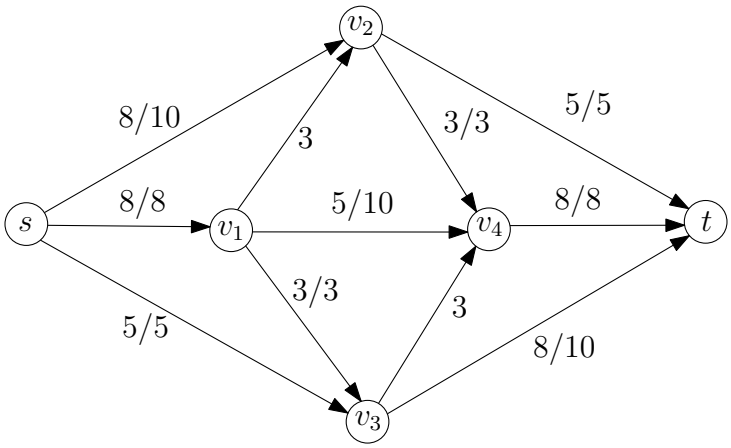


Figure 2: Answer.

Answer b. The cut $(S, T) = (\{s, v_2\}, \{v_1, v_3, v_4, t\})$ has capacity 21. In addition, the flow f_0 in Figure 2 has value 21. So by the max-flow min-cut theorem, f_0 and (S, T) are a maximum flow and a minimum cut.

8. Find a minimum $s - t$ cut in the graph drawn in Figure 3. Justify your answer.

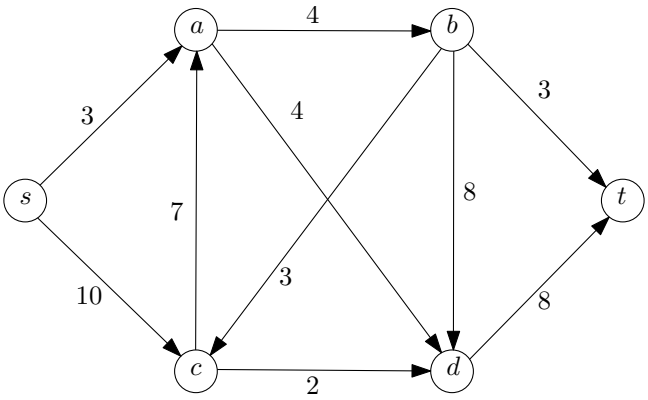


Figure 3: Flow network from Question 8.

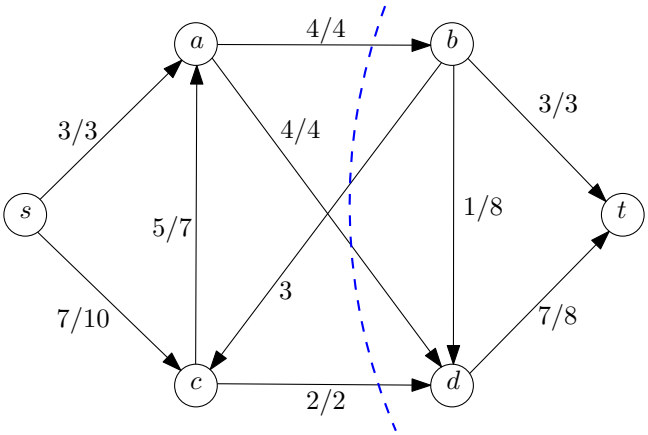


Figure 4: A minimum cut and a maximum flow with value 10.

Answer. A flow with value 10 and the cut $(\{s, a, b\}, \{b, d, t\})$ with capacity 10 are shown in Figure 4. By the max-flow min-cut, these must be a maximum flow and a minimum cut.

9. Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible *base stations*. We will suppose that there are n clients, with the position of each client specified by its (x, y) coordinates. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a *range parameter* r–a client can only be connected to a base station that is within distance r. There is also a *load parameter* L—no more than L clients can be connected to a single base station.

Your goal is to design an efficient algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether all clients can be connected simultaneously to a base station, subject to the range and load conditions in the previous paragraph.

Give a polynomial-time algorithm for this problem and prove that it returns a correct answer.

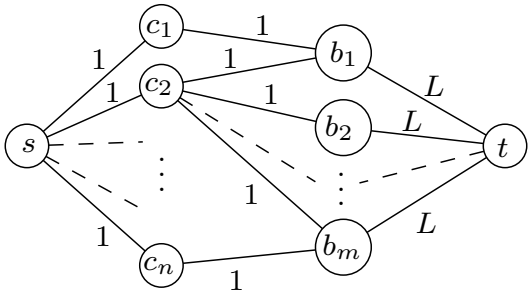


Figure 5: Answer to Question 4.

Answer. This exercise is similar to the reduction from maximum flow to maximum bipartite matching in Lecture 5: we will reformulate this mobile computing problem as a maximum flow problem. The difference is that several clients can be assigned to one base station, while in the matching problem, we only match pairs of vertices.

Let $C = \{c_1, \ldots, c_n\}$ denote the set of clients and $B = \{b_1, \ldots, b_k\}$ denote the set of base stations. We construct a graph $G(V, E)$ where $V = C \cup B \cup \{s, t\}$. (See Figure 5.) The set of edges E is defined by:

$$E = \{(s, c_i) \mid c_i \in C\} \cup \{(c_i, b_j) \in C \times B \mid (c_i, b_j) \leqslant r\} \cup \{(b_j, t) \mid b_j \in B\}.$$

Thus, we connect a client to a base if their distance is at most r, we connect the source s to all the clients, and we connect each base to the sink t. As we want to model the problem as a maximum flow problem, we also need to assign capacities to the edges. We assign a capacity 1 to each edge (s, c_i) or (c_i, b_j), and we assign a capacity L to each edge (b_j, t).

Our algorithm is the following: we run the Edmonds-Karp algorithm on this flow network. If the value $|f^*|$ of the optimal flow f^* returned by this algorithm is equal to n, then all clients can be assigned to a base station. Otherwise, it is impossible to perform such an assignment.

Observe that G has $nk + n + k$ edges, so our algorithm runs in polynomial time; more precisely, it runs in time $O(n^2k^2(n + k))$. We still need to prove that it correctly decides whether all clients can be assigned to a base station. So we will first prove that, (i) if each client can be connected to a base station, then our algorithm says that it is the case, and then we will prove that, (ii) if our algorithm says that each client can be connected to a base, then it is the case.

Part (i). So we assume that each client can be assigned to a base station. We consider one such assignment, and we denote by E' the set of pairs (c_i, b_j) in this assignment. We denote my m_j the number of clients connected to b_j in this assignment. Then we consider a flow f' defined as follow:

- $f'(c_i, b_j) = 1$ and $f'(b_j, c_i) = -1$ for each $(c_i, b_j) \in E'$.
- $f'(s, c_i) = 1$ and $f'(c_i, s) = -1$ for each $c_i \in C$.
- $f'(b_j, t) = m_j$ and $f'(t, b_j) = -m_j$ for each $b_j \in B$.

We first need to argue that f is indeed a flow. Skew symmetry and the capacity constraint are clear from our construction. Flow conservation at each client c_i follows from the fact that each client appears exactly once in E', hence for each client c_i, there is one unit of flow entering c_i from s, and one unit of flow leaving c_i to some b_j. Similarly, flow conservation is enforced at

each base station b_j because m_j units of flow arrive from the m_j clients connected to b_j, and m_j units of flow leave towards t.

Since f' is a flow, and its value is $|f'| = n$, then the Edmonds-Karp algorithm returns a flow f^* with value $|f^*| \geqslant n$. On the other hand, the cut $(\{s\}, C \cup B \cup \{t\})$ has capacity n, so by the Max-flow Min-cut theorem, we have $|f^*| \leqslant n$. Hence, $|f^*| = n$ and our algorithm correctly decided that all clients can be connected to a base station.

Part (ii). We now assume that our algorithm says that all clients can be connected to a base station. That is, the optimal flow f^* on G returned by the Edmonds-Karp algorithm has value $|f^*| = n$.

As all the capacities are integers, by the Integrality Theorem, we know that all the values $f^*(u, v)$ are integers as well. As the capacity of each edge (c_i, b_j) is 1, then $f(c_i, b_j)$ is 0 or 1. So we consider the following assignment of clients to base stations: client c_i is assigned to b_j if and only if $f(c_i, b_j) = 1$. That is, we consider the assignment $E' = \{(c_i, b_j) \mid f^*(c_i, b_j) = 1\}$.

We first need to argue that E' obeys the range condition. Let (c_i, b_j) be a pair in E'. Then $f^*(c_i, b_j) = 1$, so the edge (c_i, b_j) is present in E, which implies that $d(c_i, b_j) \leqslant r$.

We then need to argue that the load constraint is enforced. Let us consider a base station b_j. There is only one outgoing edge at b_j with capacity L, so there are at most L units of flow entering b_j. By our construction, a client c_i may only be assigned to b_j if $f(c_i, b_j) = 1$, so there are at most L clients connected to b_j.

Finally, we need to verify that each client is connected to a base station. As the sum of the capacities of the edges (s, c_i) is exactly n, and $|f^*| = n$, then all the edge (s, c_i) must be saturated by the flow f^*, hence we have $f(s, c_i) = 1$ for each c_i. Each c_i only has outgoing edges to some base stations, so the unit of flow that enters c_i from s must sent from c_i to a base station b_j, and thus $f(c_i, b_j) = 1$. It follows that c_i is assigned to this station b_j, and only this station.

Remark. Your proof should make use of the Integrality Theorem. Otherwise, you cannot rule out the possibility that the max-flow algorithm returns a flow where, for instance, $f(c_1, b_1) = 1/2$ and $f(c_1, b_2) = 1/2$.

11. We consider the *image segmentation* problem: Given a digital image, we want to partition its pixels into two classes. For instance, we may want to separate an object from the background.

The problem is formalized as follows. The image is a set of pixels $E = \{1, \ldots, n\}$, to be partitioned into two classes A and B. For each pixel i, we are given the likelihoods $a_i > 0$ and $b_i > 0$ that it belongs to class A and B, respectively. In order to minimize the size of the boundary between A and B, we are also given a penalty $p_{ij} > 0$ for each pair of pixels: We need to pay the penalty $p_{ij} = p_{ji}$ if i and j belong to different classes. The goal is to maximize the sum of the likelihoods, minus the penalties:

$$\text{Maximize} \quad q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in A \times B} p_{ij} \quad \text{over all partitions } \{A, B\} \text{ of } E.$$

Give a polynomial-time algorithm for this problem.

Answer. We will reduce this problem to the min-cut problem. So first we need to make it a minimization problem. Let

$$M = \sum_{i \in E} a_i + b_i.$$

Then we have

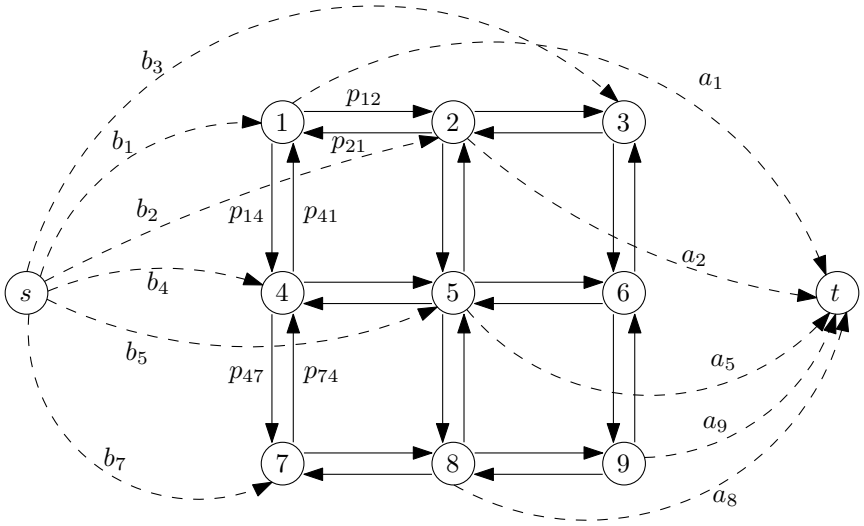
$$q(A,B) = M - \sum_{i\in A} b_i - \sum_{j\in B} a_j - \sum_{(i,j)\in A\times B} p_{ij}.$$

So our image segmentation problem is equivalent to the following minimization problem:

$$\text{Minimize} \quad q'(A,B) = \sum_{i\in A} b_i + \sum_{j\in B} a_j + \sum_{(i,j)\in A\times B} p_{ij} \quad \text{over all partitions } \{A,B\} \text{ of } E.$$

We can see this minimization problem as the min-cut problem in the graph described below.

The set of vertices is $V = \{s,t\} \cup E$. We draw an edge with weight b_i between s and each vertex $i \in E$, and we draw an edge with weight a_i between each $i \in E$ and t. We also draw an edge with weight p_{ij} between any two distinct $i, j \in E$. (See figure below; not all edges could be drawn.)



Any partition A, B of E yields a cut $A' = A \cup \{s\}, B' = B \cup \{t\}$ in this graph. The capacity of this cut is $q'(A,B)$, so minimizing $q'(A,B)$ reduces to the min-cut problem in this graph. It can be solved in polynomial time by the Edmonds-Karp algorithm, for instance.