Algorithms (2IL15): Homework Set B.2

Exercises for Lecture 7

B.2-1 (1/2 + 1/2 point) Consider the flow network $G = (V, E)$ in Fig. 1, with source $s$ and sink $t$, and the given capacities and flow $f$.

Figure 1: A flow network. Flows and capacities are indicated using the notation flow/capacity.

(i) What is the value of the flow $f$? Show that this flow is a maximum flow by specifying a cut $(S, T)$ whose capacity equals the value of the flow $f$.

(ii) Draw the residual network for this network with the given flow.

B.2-2 (1/2 point) Suppose that we have a flow network $G = (V, E)$ in which not only the edges have capacities, but also the vertices. More precisely, every vertex $v \in V \setminus \{s, t\}$ has a capacity $c(v)$. We now require that a valid flow also respects the vertex capacities, meaning that the flow through any vertex $v \in V \setminus \{s, t\}$ is at most $c(v)$.

Explain how to modify $G$ into a normal flow network $G'$—that is, one without vertex capacities—such that the maximum flow in $G$ is equal to the maximum flow in $G'$.

B.2-3 (1 1/2 point) In a flow network the source node $s$ can have incoming edges. Prove that these edges are in fact not needed, that is, prove that there exists a maximum flow $f^*$ such that $f^*(u, s) = 0$ for all $u \in V$.

B.2-4 (1+1 point) Consider a flow network $G = (V, E)$ with source $s$ and sink $t$ such that $G$ forms a tree with root $s$ if we remove $t$ and its incoming edges, and such that the only incoming edges in $t$ are edges from the leaves of this tree. We call such a flow network a flow tree; see Fig. 2 for an example.

(i) Give pseudocode for a recursive algorithm that computes a maximum flow in a flow tree in $O(|V|)$ time. (Don’t try to adapt the Ford-Fulkerson method, but give a different algorithm that is targeted to flow trees.)

Figure 2: Example of a flow tree.
(ii) Let \( M \) be the value of the flow computed by your algorithm. Prove by induction that there is a cut whose capacity is equal to \( M \). (Note: By the Max-Flow Min-Cut Theorem, this implies that your algorithm correctly computes a maximum flow.)

Exercises for Lecture 8

B.2-5 \((\frac{1}{2} + \frac{1}{2} + 1\) point) Consider the flow network \( G = (V, E) \) in Fig. 3. The network consists of a path \( s, u_1, \ldots, u_m \), for some \( m \geq 1 \) and a collection \( v_1, \ldots, v_k \) of \( k \geq 1 \) vertices that each have an incoming edge from \( u_m \) and an outgoing edge to \( t \).

\[
\begin{array}{c}
s \rightarrow u_1 \rightarrow \cdots \rightarrow u_m \\
\vdots \\
\vdots \\
\rightarrow t
\end{array}
\]

Figure 3: A flow network.

(i) Give an example of a set of capacities on the edges such that the Ford-Fulkerson method finishes after a single iteration. Explain your example.

(ii) Give an example of a set of capacities on the edges such that the Ford-Fulkerson method needs at least \( k \) iterations to finish. Explain your example.

(iii) Prove that the Ford-Fulkerson method finishes after at most \( k \) iterations for \( \text{any set of capacities in this network, no matter how the augmenting paths are chosen. Your proof should be based on the concept of critical edge (as defined in the proof of number of iterations performed by the Edmonds-Karp algorithm).} \)

B.2-6 \((1 + \frac{1}{2} + \frac{1}{2} \) point) Let \( P = \{p_1, \ldots, p_n\} \) be a set of \( n \) points in the plane, and let \( R = \{r_1, \ldots, r_m\} \) be a set of \( m \) rectangles. Each rectangle \( r_j \in R \) has a budget \( b(r_j) \), where all budgets are positive and integral. The point-to-rectangle problem is to decide if \( P \) can be partitioned into \( m \) subsets \( P_1, \ldots, P_m \)—thus, each point in \( P \) must appear in exactly one subset—such that

- The points from \( P_j \) all lie inside the rectangle \( r_j \), for all \( 1 \leq j \leq m \).
- The total number of points in \( P_j \) does not exceed the budget of \( r_j \), that is, \( |P_j| \leq b(r_j) \), for all \( 1 \leq j \leq m \).

See Fig. 4 for an example. We want to solve this problem by formulating it as a flow problem and then solving the flow problem using the Ford-Fulkerson method.

(i) Explain how to construct a flow network \( G = (V, E) \) for a given instance of the point-to-rectangle problem, such that the point-to-rectangle problem can be solved by computing a maximum flow on the network using the Ford-Fulkerson algorithm. You should explicitly explain how, after running the Ford-Fulkerson method on the network \( G \), you

\[
\begin{array}{c}
r_1 \\
p_1 \\
p_2 \\
p_4 \\
p_8 \\
r_2 \\
p_3 \\
p_6 \\
p_5 \\
p_9 \\
r_3 \\
b(r_1) = 3 \\
b(r_2) = 4 \\
b(r_3) = 2
\end{array}
\]

Figure 4: An instance of a point-to-rectangle problem. A solution for this instance would be \( P_1 = \{p_1, p_2, p_4\} \), \( P_2 = \{p_3, p_5, p_7, p_8\} \), and \( P_3 = \{p_6, p_9\} \).
can decide if the point-to-rectangle problem has a valid solution—that is, a partitioning of $P$ into subsets $P_1, \ldots, P_m$ that satisfies the two requirements stated above—and, if so, how to construct a solution.

(ii) Suppose you solve the point-to-rectangle assignment in this manner, where you use the Edmonds-Karp algorithm as your implementation of the Ford-Fulkerson method. What is then the running time of the algorithm, as a function of $n$ and $m$?

(iii) Suppose someone gives you a flow $f_{\text{max}}$ for $G$. The flow $f_{\text{max}}$ is a maximum flow for $G$, but it has not been computed by the Ford-Fulkerson method, but by some completely different method. Can you use $f_{\text{max}}$ to decide if the point-to-rectangle problem has a valid solution? Explain your answer.

B.2-7 (1 point) You have sweets of different types and want to distribute them into packages such that no package contains more than one sweet of a type (Your friends will appreciate the variety). Assume you have $M$ different types of sweets and $s_i$ sweets of type $i$. Further assume that you have $N$ packages and the $j$th package can hold $p_j$ sweets.

Explain how to construct a flow network $G = (V, E)$ for a given instance of this problem, such that the sweet packaging problem can be solved by computing a maximum flow on the network using the Ford-Fulkerson algorithm. You should explicitly explain how, after running the Ford-Fulkerson method on the network $G$, you can decide if you can package your sweets such that no sweets are left—and, if so, how to construct a solution. You do not need to analyze the running time.