Dinitz algorithm (cont.)

Layered network

Recall the main steps of the Dinitz algorithm.

The construction of the residue network $N'$ from $N$ was discussed in Lecture 5. Next, consider the construction of the layered network $N''$ from $N'$. It is very similar to applying the BFS algorithm to the network $N'$ starting at vertex $s$. More specifically, the constructed $N''$ can be described as follows:

- $V'' = \bigcup_{i=0}^{\ell} V_i$, where $V_i$ is the $i$-th layer, $\ell$ is the total number of layers;
- $E'' = \left\{ u\xrightarrow{e''} v \mid u \in V_i, v \in V_{i+1}, e'' \in E' \right\}$.
- For each $e'' \in E'$, $c''(e'') = c'(e'').$

(See an example in Lecture 5).

Pseudocode of Dinitz algorithm

The following scheme summarizes the Dinitz algorithm.

In Algorithm 1, the procedure $\text{Layers}(N', V_0, V_1, \ldots, V_{n-1}, \ell)$ constructs subsets of vertices (layers) $V_0, V_1, \ldots, V_{n-1}$, where $\ell$ is the total number of layers. The procedure $\text{Maximal}(N'', f'')$ finds a maximal flow $f''$ in $N''$. It is specified below.

---

1 Additional reading: Section 5.3 in the book of S. Even “Graph Algorithms”.
Input: Network $\mathcal{N}(\mathcal{G}(\mathcal{V}, \mathcal{E}), s, t, c)$
Output: Flow function $f$

1. for $e \in \mathcal{E}$ do
2.   $f(e) \leftarrow 0$;
3. end
4. $\mathcal{N}' \leftarrow \mathcal{N}$;
5. LAYERS($\mathcal{N}'$, $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{n-1}, \ell$);
6. while $f$ is not maximum do
7.   construct $\mathcal{N}''$;
8.   MAXIMAL($\mathcal{N}''$, $f''$);
9.   for all $e' \in \mathcal{E}''$ do
10.     if $e'$ corresponds to a forward edge $e \in \mathcal{E}$ then
11.        $f(e) \leftarrow f(e) + f''(e')$;
12.     end
13.     if $e'$ corresponds to a backward edge $e \in \mathcal{E}$ then
14.        $f(e) \leftarrow f(e) - f''(e')$;
15.     end
16. end
17. Construct $\mathcal{N}'$;
18. LAYERS($\mathcal{N}'$, $\mathcal{V}_0, \mathcal{V}_1, \ldots, \mathcal{V}_{n-1}, \ell$);
19. end

Algorithm 1: Dinitz Algorithm
Finding maximal flow in $N''$

Finding maximal flow in $N''$ is done by finding augmenting paths from $s$ to $t$ in it. Each edge has a label, which takes values unblocked and blocked. In the beginning of the phase, all edges are labeled unblocked. Then, once an augmenting path is found, and the flow pushed through it, – the edge which is saturated (the flow is equal to capacity) is labeled as blocked. If during the attempt to find an augmenting path, the algorithm gets stuck at some vertex (because all of its outgoing edges are blocked), it walks back through the last used edge and marks that edge blocked too. The processes continues while there exists an augmenting path from $s$ to $t$ in $N''$.

Thus, the edge $e = (u, v)$ is blocked in one of the two cases:

- $e$ is saturated;
- there is no unblocked edge out of $v$.

**Input:** Layered network $N''(G(V'', E''), s, t, c'')$

**Output:** Maximal flow function $f''$

1. for all $e \in E''$ do
   2. $f''(e) \leftarrow 0$;
   3. Label$(e) \leftarrow \text{"unblocked"}$;
4. end
5. Find_path($N''$, blocking_labels, $S$);
6. while $S$ is not empty do
   7. Increase_flow($N''$, $S$, $f''$, blocking_labels);
   8. Find_path($N''$, blocking_labels, $S$);
9. end

**Algorithm 2:** Procedure Maximal

Here $S$ is a stack that stores the current path from $s$ to $t$.

The procedure Find_path($N''$, blocking_labels, $S$) finds an augmenting path from $s$ to $t$ using non-blocked edges and stores it in $S$. The procedure Increase_flow($N''$, $S$, $f''$, blocking_labels) finds the maximal possible increment of flow in the augmenting path stored in $S$, and marks saturated edges as “blocked”.

3
Correctness of the algorithm

Intuitively, the Dinitz algorithm makes use of augmenting paths in a certain way. Thus, the run of the Dinitz algorithm can be viewed as a special case of the run of Ford-Fulkerson, which is known to find maximum flow.

**Lemma 1** If the Dinitz algorithm halts, then the resulting \( f \) is a legitimate flow in \( \mathcal{N} \).

**Proof.**

- The values of \( f'' \) represent a legitimate flow in \( \mathcal{N}'' \).
- When \( f \) is changed by adding (or subtracting) the value of \( f'' \) to (from) \( f \), the edge rule is maintained in the edges of \( \mathcal{N} \) (this is due to the way the residue network \( \mathcal{N}'' \) and the layered network \( \mathcal{N}'' \) are constructed.)
- The new \( f \) is a superposition (an edge-wise sum) of the two flow functions, each of them observes the vertex rule, and so the resulting \( f \) does.

□

**Lemma 2** When the Dinitz algorithm halts, the resulting \( f \) is the maximum flow in \( \mathcal{N} \).

**Proof.** The idea of the proof is similar to the analogous proof for Ford-Fulkerson. The algorithm halts because no augmenting path from \( s \) to \( t \) in \( \mathcal{N}' \) (and in \( \mathcal{N} \) exists. Assume that the maximum flow is not reached. Consider the set of vertices \( S \), to which there exist augmenting paths from \( s \). Then the cut \( (S : \bar{S}) \) is saturated, it must be a minimum cut, and the total flow is maximum. □

**Time complexity**

**Lemma 3** (Lemma 5.7 in the book of S. Even) If the \((k + 1)\)-st phase is not last, then \( \ell_{k+1} > \ell_k \), where \( \ell_i + 1 \) is the number of layers in phase \( i \).

For the proof of this lemma please refer to Section 5.3 in the book. The lemma shows that the number of layers in the layered network in the algorithm is monotonically increasing with the number of phases.

**Lemma 4** The time complexity of the Dinitz algorithm is \( O(|V|^2|E|) \).

**Proof.** Consider the main steps in the algorithm.

- Constructing \( \mathcal{N}' \) and \( \mathcal{N}'' \) takes \( O(|V| + |E|) \).
• During execution of MAXIMAL, an edge is blocked at least every $\ell \leq |V|$ steps (the algorithm either finds an augmenting path and pushes flow – at least one edge is saturated, or gets stuck and traverses back – the corresponding edge is blocked). The total number of edges is $\leq |E|$. Therefore, the total complexity of MAXIMAL is $O(|V||E|)$, and this is also the total complexity of one phase.

• Since the maximum number of layers is $\ell + 1 \leq |V|$, due to Lemma 3, there are at most $O(|V|)$ phases.

The total complexity of all phases is then $O(|V| \cdot |V||E|) = O(|V|^2|E|)$, and this is the complexity of the algorithm.