CSC375 Dinic's max flow algorithm)
Reference: R. Tarjan's Monograph

We are given a flow network $\mathcal{F} = (G, s, t, c)$. Dinic's algorithm can be viewed as an implementation of the generic Ford Fulkerson algorithm. It is a strongly polynomial time algorithm that runs in time $O(mn^2)$ where $G = (V, E)$ and $m = |E|, n = |V|$.

In a directed graph $G = (V, E)$ with distinguisheed source node s, we define $level(v)$, the level of node v, to be the length of the shortest path from s to v. The levelled graph $L = (V, E_L)$ associated with G is the graph obtained by taking the set of edges $E_L = \{(u, v) \in E$ such that $|level(v) = level(u) + 1\}$. When G is the underlying graph of a flow network, we can also view L as a flow network where the capacity of any edge in E_L is set to its capacity in E. We need one more (central) definition, namely the concept of a *blocking flow.* We say that a flow f' is a blocking flow for network $\mathcal{F} = (G, s, t, c)$ if every $s - t$ path Π in G is saturated by f'; that is, for at least one edge e in Π, $f'(e) = c(e)$. Finally, we recall that for a flow f in an flow network $\mathcal{F}$, we let G_f denote the residual graph which gives rise to the residual network $\mathcal{F}_f = (G_f, s, t, c_f)$ where $c_f(e) = c(e) - f(e)$.

Dinic's Algorithm

$f(e) := 0$ for all $e \in E$; $L =$ levelled graph associated with G

% Initialize with the trivial all zero flow and L the levelled graph for the given input network

While $level(t) < \infty inL$

find a blocking flow f' in L

$f := f + f'$;

construct $G_f; L := L_f =$ the levelled graph associated with G_f

End While

The Ford Fulkerson max flow-min cut implies that upon termination, Dinic's algorithm correctly computes a maximum flow in the input network $\mathcal{F}$.

The termination and efficiency of Dinic's algorithm follows from the following results:

Theorem 1: Dinic's algorithm terminates in $n - 1$ iterations (i.e. in $n - 1$ blocking steps)

Theorem 2: The residual graph G_f and its associated levelled graph L can be constructed in time $O(m)$

Theorem 3: A blocking flow in a levelled graph can be computed in time $O(mn)$ and hence Dinic's algorithm always terminates within time (mn^2)

The proof of Theorem 2 is quite obvious using breadth first search to compute L_f. Theorem 3 is achieved using depth first search as sketched in class. The more interesting theorem is Theorem 1. The crucial lemma (as discussed in class) needed for Theorem 1 is the following:

Lemma: Let L_i be the levelled graph at the end of iteration i and let $level_i(v)$ be its level function. Then $level_{i+1}(v) \geq level_i(v)$ for all $v \in V$ and $level_{i+1}(t) > level_i(t)$. That is, the level of the target node t must increase after a blocking step.

Theorem 1 follows immediately from the Lemma since $level(t) < \infty$ implies $level(t) \leq n - 1$.

Recall that a *unit network* is one in which all edges have capacities in $\{0, 1\}$ and for all nodes $v \neq s, t$, either v has at most one incoming edge of capacity 1 or at most one outgoing edge of capacity 1. In particular, the network associated with bipartite matching is a unit network.

The following results provide a significant improvement for the bounds in Dinic's algorithm when appiled to unit networks (and hence when applied to maximum matching in a bipartite graph). We note that for any flow f in a unit network, G_f is also a unit network.

Theorem 4: If $\mathcal{F}$ is a *unit network*, then Dinic's algorithm terminates within $2\sqrt{n}$ blocking steps.

Theorem 5: A blocking flow in a unit network can be computed (again by the same depth first search used for Theorem 3) in $O(m)$ steps. Hence a max flow in a unit network (and max matching in a bipartite graph) can can be computed in $O(m\sqrt{n})$ steps.

Sketch of proof:

After $\sqrt{n}$ blocking steps, we known by the creitical lemma that every $s - t$ path has length at least $\sqrt(n)$. If the f^* is a max flow and f is the flow after the first $\sqrt(n)$ blocking steps, then there is a (max) flow f' in G_f of value $val(f^*) - val(f)$. Since G_f is also a unit network, such a flow must be realized by node disjoint paths and hence there are at least $[val(f^*) - val(f)] \cdot \sqrt{n} \leq n$ nodes which implies that $val(f^*) - val(f) \leq \sqrt{n}$. Thus in at most an additional

$\sqrt{n}$ iterations (even if only a single augmenting path is found in each step), the algorithm will terminate.