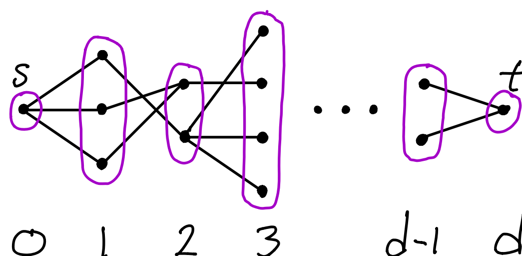
The Shortest Augmenting Paths (Edmonds-Karp #2) algorithm had a running time of $O(nm^2)$, and worked by using a breadth-first search to find the augmenting path with the fewest edges at each iteration. We can improve this algorithm to $O(n^2m)$ by reorganizing the computation. This is called Dinic's algorithm[1].

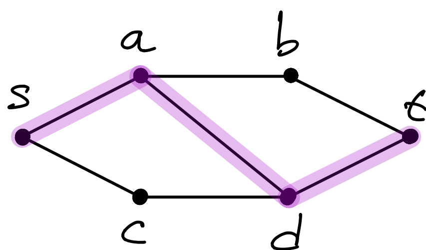
Dinic's Algorithm (Optional)

If we examine what E-K#2 does, we see that it generates a sequence of augmenting paths of non-decreasing length. What Dinic's algorithm does is it finds all the necessary augmenting paths of a given length with one *blocking flow* computation. It then augments the graph by this blocking flow, and repeats. The number of passes needed is at most $n-1$, because each one increases the length of the paths being searched for. So at the beginning of each pass the algorithm spends $O(m)$ time to explicitly build the layered graph. This is done via a simple breadth-first-search starting from s.

The layered graph L only includes the edges that go from one level to the next higher level. The schematic below illustrates a layered graph. All edges go from left to right.



So the crux of the algorithm is to find a flow in L from s to t which is *blocking*, which means that every path from from s to t that steps from level to level (always increasing) has a saturated edge on it. Note that this is not the same as a maximum flow, as illustrated in the figure below.



In the above diagram all the edges go from left to right. Level 0 is comprised of $\{s\}$, level 1 is comprised of $\{a, c\}$, level 2 is $\{b, d\}$, and level 3 is $\{t\}$. All edges are of capacity 1, and directed from a level i to level $i+1$.

A flow of one unit along the highlighted path from left to right is a blocking flow. In the residual

[1]The idea of finding the blocking flow, i.e., maximum flow possible in the level graph G_{level}, and then iterating this as the distance from s to t gets bigger, is due to Yefim Dinitz. He gave this algorithm while getting his M.Sc. in Soviet Russia in 1969 under G. Adel'son Vel'sky (of AVL trees fame), in response to a exercise in an Algorithms class, and published it in 1970. The iron curtain meant this was not known in the west for a few more years, and then only as Dinic's [*sic*] algorithm. Dinitz talks about his algorithm and some history here.

graph that results, there is no path of positive capacity from s to t that starts at level 0 and proceeds to level 1, 2, and then 3. However there *is a path* of positive capacity from s to t, namely $[s, c, d, a, b, t]$. This path will be discovered later when the algorithm is working on paths of 5 edges.

Here is some pseudo code for computing a blocking flow.

```
Dinic's Algorithm to find a blocking flow in a layered graph L.  The
edges of this graph only go from a layer to the next higher layer.

The graph is represented as follows:

degree[v] : the number of edges out from v to the next level.

N[v][i]   : The ith neighbor of vertex v.  (Of course these vertices are
            all in the next layer.)  The index i ranges from 0 to degree[v]-1.

Cap[v][i] : The capacity of the edge from v to N[v][i].
            These change over time as we push flow through.

The adjusted capacities are how the algorithm returns its result.

The algorithm also makes use of the following variable:

C[v]      : The index of the "current" neighbor of v.
            N[v][C[v]] is that neighbor.  Initially this
            is 0, and eventually it could reach degree[v].

    Dinic(L) {
        Build the representation of L described above.
        flow <- 0;
        while(true) do{
            f <- Search(s);
            if f = 0 then return flow;
            else {
                flow = flow + f;
                Augment the current path from s to t by f.
                (i.e. reduce all these capacities by f)
                This path is obtained by simply following
                current edges.
            }
        }
    }

    Search(v) {
    // Search for a path from v to t of positive capacity.
    // If there is no such path, return 0.
    // As a side-effect, this may increase C[v]

        if v=t then return infinity;
        while(true) do {
           if C[v] = degree[v] then return 0; (call v "dead")
           if Cap[v][C[v]] = 0 then C[v] <- C[v] + 1
           else {
               f <- Search(N[v][C[v]])
               if f > 0 then return min (f, Cap[v][C[v]])
               else C[v] <- C[v] + 1
            }
        }
    }
```

Discussion of Correctness

Claim: once a vertex v is labeled "dead" then there is no path of positive capacity from v to t.

The proof is by induction. The only way a vertex is said to be dead is if every neighbor w on the next level either (1) gives a return value of zero on `Search(w)`, or (2) the edge from v to w has zero capacity. This shows that there is no path of positive capacity from v to t.

Discussion of Running Time

Lemma 1 *The running time of Dinic's blocking flow algorithm is $O(nm)$.*

Proof: Each path augmentation causes the residual capacity of an edge to go to zero. This can happen at most m times. The work of all of these augmentations is at $O(nm)$.

What about the time of `Search()`? The running time is proportional to the total number of calls to the `Search()` function. Every time `Search()` returns a zero value, it causes `C[v]` (for some v) to increase. This can happen at most m times, in total. When it returns a positive value, this causes a cascading sequence of returns of positive value all the way back to the initial call of `Search(s)`. These calls to search can be paid for by the lengths of all the augmenting paths, which total to at most nm. ■

In practice this algorithm seems much faster. We can prove this in a few special cases.

Lemma 2 *If all edges in the original graph have capacity one, then Dinic's blocking flow algorithm runs in $O(m)$ time. (And the whole algorithm runs in $O(mn)$ time.)*

Proof: First note that the unit capacity property is preserved in all residual graphs.

In the proof of Lemma 1 above we used the observation that the total length of all augmenting paths is at most nm. Since now all edges have capacity one, an edge can only be used once in an augmenting path. Thus the total length of all augmenting paths is at most m. As before the total cost of all the calls to `Search()` is also $O(m)$. ■

Continuing along the same lines, a *unit network* is one in which all edges have capacity one, and additionally for each vertex the total capacity exiting it **or** the total capacity entering it is at most one. Either way, the amount of flow that an be put through such a vertex is at most one. Being a unit network is preserved in all residual graphs. This is because the residual transformation that occurs at a vertex v simply swaps an incoming edge and an outgoing edge at v. Thus the *number* of incoming edges and outgoing edges is preserved at every vertex (excluding s and t).

Note that the flow graph that we used to solve the matching problem is a unit network.

Lemma 3 *In a unit network, the running time of Dinic's algorithm is $O(m\sqrt{n})$.*

Proof: It follows from Lemma 2 that Dinic's algorithm finds a blocking flow in a unit network in time $O(m)$. We will complete the proof by showing that the number of iterations of the blocking flow algorithm is at most $O(\sqrt{n})$.

Consider the situation after $\sqrt{n}$ iterations. Each iteration increases the number of layers in the layered network by one, so in the next, and all subsequent iterations the number of layers will be at least $\sqrt{n}$.

Let's look at the residual graph G^R that exists right after the $\sqrt{n}$th iteration. Divide G^R into layers, but keep all the edges, even ones within a layer. We'll show that there must be a cut in G^R

with capacity at most $\sqrt{n}$. This shows that there can be at most an additional $\sqrt{n}$ iterations of the algorithm.

If the number of layers is at least $\sqrt{n}$ then there must be a layer with at most $\sqrt{n}$ nodes in it (since the total number of nodes is n). Call that layer ℓ.

We'll construct a cut (A, B) as follows. A will contain all layers $0, 1, \ldots \ell - 1$. B will contain layers $\ell + 1, \ell + 2 \ldots$. Consider a vertex v in layer ℓ. If v has just one edge entering it, then we'll put it into the set B. Otherwise we put it into set A. Note that in this case it must have at most one edge exiting it.

Observe that no edges can go directly from a layer before ℓ to one after ℓ. So the only edges that go from A to B must use one of the vertices of layer ℓ. And each of these contributes at most one to the capacity of the cut. So the cut (A, B) has capacity at most $\sqrt{n}$. This completes the proof. ■

Lemma 4 *When Dinic's algorithm is used for the maximum matching problem the first blocking flow has value at least $\lceil M/2 \rceil$, where M is the size of the maximum matching.*

Proof: Conisder the blocking flow in the middle of page 1 of these lecture notes. Notice the way in which a flow of one can block two paths. In general a flow of one can block *at most two* paths. The result follows. ■

So for matchings, the flow graph satisfies the hypothesis of Lemma 2, which means that a blocking flow is found in $O(m)$ time. Now we see that this blocking flow finds a matching at least half the size of the maximum one.