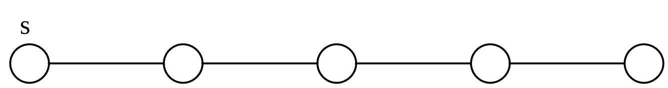
In previous lectures we discussed admission control algorithms on general undirected graphs that were $O(\log m)$-competitive. That is, the number of paths accepted by our algorithm was within a factor of $O(\log m)$ of the optimal number of paths. However, our algorithms depended on the assumption that each edge could be shared by $\Omega(\log m)$ paths. Can we still get a competitive algorithm without this restriction? Unfortunately, a couple of lower bounds exist for general undirected graphs:

- Deterministic $O(\log m)$-competitive admission control algorithms need to require that edges may be shared by $\Omega(\log m)$ paths.
- If paths cannot share edges at all, no admission control algorithm can be better than $O(n)$-competitive (i.e. there is an $\Omega(n)$ lower bound on the competitive ratio - see homework #2).

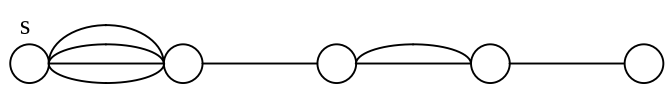
In this lecture we consider admission control algorithms on special kinds of graphs, such as trees, meshes, and random graphs, where algorithms exist that are competitive without requiring that an edge may be shared by $\Omega(\log m)$ paths.

<u>Paths with parallel copies and a single source node</u>

First we consider admission control on graphs that look like paths:



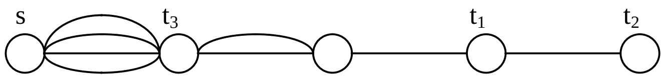
We will allow each edge to contain multiple "parallel copies":



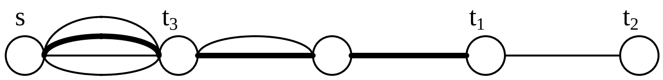
In addition, we require that all paths start from the leftmost node (which we call "s"). For this problem, the following algorithm is optimal:

Algorithm: Accept a path if there is room for the path.

For example, suppose that on the following graph we get requests for paths (s, t_1), (s, t_2), and (s, t_3) (in that order):



The path (s, t_1) will be accepted:



(s, t_2) will be rejected, though, because one of the edges on the way from s to t_2 is already full (we say an edge is "full" if all of its parallel copies have been taken). (s, t_3) will be accepted.

We claim this algorithm is optimal. That is, it accepts the maximum possible number of paths.

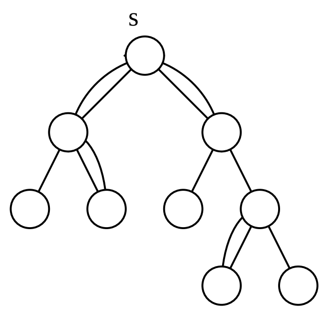
*Proof***.** Consider the easy case where the algorithm is able to accept all the requested paths. This is obviously optimal - we can’t accept any more paths than were requested. Now consider the case where the algorithm had to reject some paths. Look at the state of the graph after the algorithm has finished, and let "e" be the closest full edge to s. If some request (s, t_i) was rejected, than t_i must lie beyond e (otherwise, there would be space for (s, t_i) so it would have been accepted). Divide all of the connection requests into two sets:

- Let c_1 be the connections that do not cross e. There is room for all of these connections, so our algorithm accepts all connections in c_1.
- Let c_2 be the connections that cross e. Let $cap(e)$ be the number of parallel copies contained by e. Our algorithm accepts $cap(e)$ edge copies from c_2, because e is full in the final configuration of the graph, so every parallel copy on it has been taken by one connection from c_2.

Even the optimal algorithm cannot accept more than $cap(e)$ connections out of c_2 - there wouldn’t be any room on edge e for any more connections. And the optimal algorithm can accept at most $|c_1|$ of the paths in c_1, so the optimal number of paths is no more than $|c_1| + cap(e)$. But our algorithm accepts exactly $|c_1| + cap(e)$ paths, since it takes all the connections in c_1 and $cap(e)$ of the connections in c_2. So our algorithm is optimal.

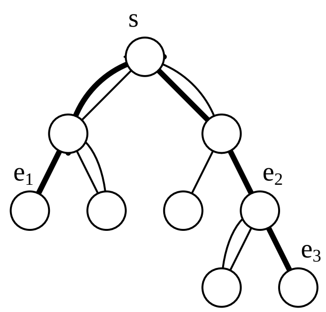
Binary trees with parallel copies and a single source node

Now we consider binary trees, where all connection requests are of the form (s, t_i), where s is the root node of the tree. Edges may contain multiple parallel copies:



Again, we claim that the simple greedy algorithm (for any connection request (s, t_i), accept the request if room is available) is optimal.

Proof. Look at the graph after the completion of the algorithm. Let $\tilde{E}$ be the set of all edges that are full and have no other full edges between them and s. For instance, in the following graph e_1, e_2, and e_3 are full, and $\tilde{E}$ is the set $\{e_1, e_2\}$:



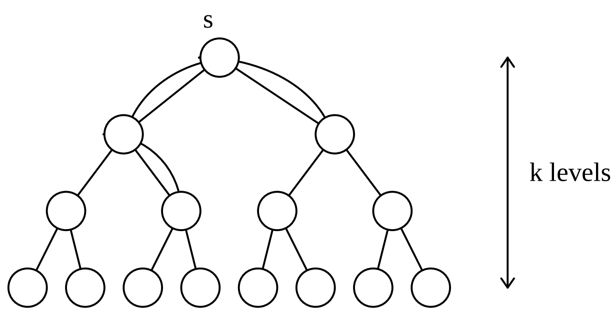
Again, we divide the requested connections into two sets:

- Let c_1 be the set of connection requests that don't pass through any edge in $\tilde{E}$.
- Let c_2 be the set of connection requests that pass through some edge in $\tilde{E}$.

The connections in c_1 do not go through any full edges, so our algorithm accepts all of the requests in c_1. The optimal algorithm can accept no more than $|c_1|$ paths from c_1. Our algorithm accepts exactly $\sum_{e \in \tilde{E}} cap(e)$ paths from c_2, because every parallel copy of an edge in $\tilde{E}$ is taken by one path, and no single path goes through more than one edge in $\tilde{E}$. The optimal path could not accept more than $\sum_{e \in \tilde{E}} cap(e)$ paths from c_2, because there would be no room for more. So our algorithm accepts no fewer paths than the optimal algorithm.

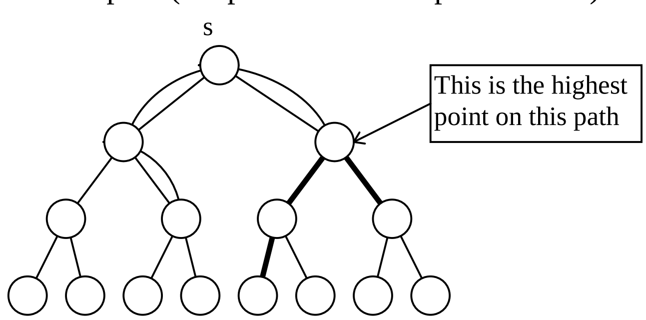
<u>Full binary trees with parallel copies and arbitrary connections</u>

Now we look at full binary trees (again with parallel copies) and we allow connections between arbitrary nodes. Say the tree has k levels:



We present an algorithm and show that it is $O(k)$-competitive (note: $O(k)$-competitive is the best we can do for an on-line algorithm (this is a lower bound, although we do not prove this here). The off-line version of this problem is NP-complete and the best known off-line approximation algorithm is 2-competitive).

Algorithm: At the beginning of the algorithm, select a level i at random. This i will remain constant while the algorithm runs. Now for each connection request that comes in, there is one obvious path that connects the two nodes in the connection. Find the highest point in this path (the point where the path "turns"). For example:



- If the highest point is on level i, use a greedy admission policy.
- Otherwise, always reject.

Any solution produced by this algorithm is feasible (it satisfies our constraints). Now we want to argue that it is $O(k)$-competitive.

Proof. Let $\mathcal{O}$ be the set of connections accepted by the optimal algorithm. Divide $\mathcal{O}$ into $\mathcal{O} = \mathcal{O}_1 \cup ... \cup \mathcal{O}_k$, where all the paths in $\mathcal{O}_j$ turn on level j. Let $\mathcal{A}_i$ be the set of connections accepted by our algorithm (the i here is the level our algorithm selected when it started). Of course, all the paths in $\mathcal{A}_i$ turn on level i because our algorithm rejects paths that don't turn on level i.

> Claim: $|\mathcal{A}_i| \geq \frac{1}{2} |\mathcal{O}_i|$
>
> *Proof.* We didn't actually do the proof in class. The proof should proceed along the lines of the earlier tree proof, dividing the connection requests into c_1 and c_2, where c_2 paths go through at least one $\tilde{E}$ edge (note that a path could go through as many as two $\tilde{E}$ edges this time!) and we wind up accepting $\geq \frac{1}{2}\sum_{e\in\tilde{E}} cap(e)$ the connections in c_2.

Using this claim, we can prove our competitiveness. Our algorithm is randomized, so we look at the expected number of connections that it accepts, averaged over all k possible choices of i. The expected number of accepted connections is:

$$
\begin{aligned}
\sum_i \tfrac{1}{k} |\mathcal{A}_i| &= \tfrac{1}{k} \sum_i |\mathcal{A}_i| \\
&\geq \tfrac{1}{2k} \sum_i |\mathcal{O}_i| \\
&= \tfrac{1}{2k} |\mathcal{O}|
\end{aligned}
$$

So our the expected number of accepted connections is within a factor of 2k of optimal.

<u>Other results for trees</u>

- $O(\log n)$ -competitive algorithms exist for arbitrary trees with parallel copies and arbitrary connections
- Even better, there is an $O(\log \mathit{diameter})$ -competitive algorithm, where *diameter* is the diameter of the tree

See the Web page for references to these two papers