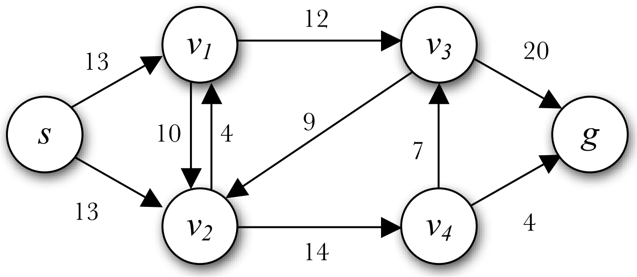
Maximum Flow

Maximum flow is an important problem in computer science. The Ford-Fulkerson method for solving it establishes a number of algorithmic techniques: augmenting paths, residual networks, and cuts. There are many applications that benefit from this solution, including network routing, highway design, path finding for multiple units, and circuit design. The Ford-Fulkerson algorithm builds on algorithms and data structures that we've studied so far, such as breadth-first search, queues (used in BFS), and graphs. For the final exam, you should know:

- *what* the max flow problem is
- that it can be solved in polynomial time
- the magnitude of the maximum flow is exactly equal to the flow across the minimum cut according to the max flow-min cut theorem
- that max flow is an example of an algorithm where the search order matters

1 The Maximum Flow Problem

Consider a direct graph G with one source ("start") *s* and one sink ("goal") *g*. For simplicity, assume that these are sources and sinks in the graph theory sense: there are no incoming edges at *s* and no outgoing edges at *g*. Each edge of the graph is labeled with its capacity. Let $c(u, v)$ be the capacity of the edge from u to v. The max flow problem is to compute a new set of edge labels. These new labels represent the steady state rate of flow on each edge, $f(u, v)$.



2 A Physical Analogy

It may help to think of edges as water pipes. In this case the source is a water spigot to which they are connected and the sink is literally a sink—it is a drain through which the water flows. Because water can only enter a pipe as fast as it leaves the pipe, the various paths through the pipeline are constrained by their bottlenecks. We want to compute the maximum flow of water possible through each pipe given all of the bottlenecks in the system. Similar (useful!) analogies can be made to current traveling through electric circuits, automobiles traveling on roads, and packets traveling on a computer network.

3 The Ford-Fulkerson Method

The Ford-Fulkerson method[1] computes maximum flow through a graph in polynomial time. The algorithm is simple:

```
label the flow of each edge as 0
for each path from source to sink:
    let residual = minimum of (capacity − flow) for all edges on the path
    increase the flow of each edge on the path by residual
```

[1] L.R. Ford and D.R. Fulkerson, Maximal flow through a network. *Canadian Journal of Mathematics*, vol. 8, pages 399-303, 1956

Expanding the Ford-Fulkerson method into more detailed pseudo-code gives:

```
for each edge from $u$ to $v$ in $p$:
    $f(u, v) = 0$
for each path $p$ from $s$ to $g$ in G:
    let $r = \infty$
    for each edge from $u$ to $v$ in $p$:
        $r = \min(r, c(u, v) - f(u, v))$
    for each edge from $u$ to $v$ in $p$:
        $f(u, v) = f(u, v) + r$
```

The two inner for-loops can't be combined because we have to compute the maximum possible flow along the whole path before we can update the labels.

The final flow labels depend on the order of iteration through the paths, however the total net flow and the minimum edges will be the same no matter how the iteration proceeds. One way to iterate through the edges is to perform depth-first or breadth-first search of the entire graph. The structure of the search itself is a tree (branching every time it hits a vertex), and every leaf of the search tree (that actually reaches the goal) corresponds to a unique path through the graph.

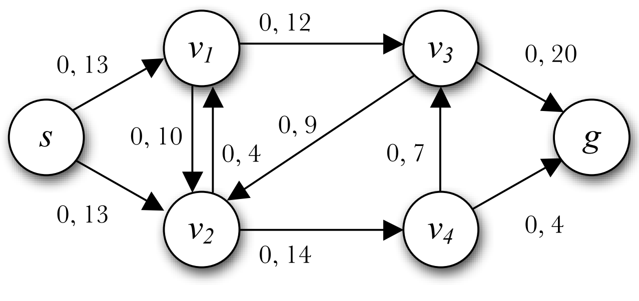
The challenge is that in a graph with cycles we must avoid sampling the infinite number of paths possible. Normally during a search one marks vertices (or edges) as visited to break cycles. However, we need to erase those visited marks every time around the iterator (the outer loop), so that we can sample multiple paths through a vertex during different iterations. One way to do this is to simply restart a search for each time around the path iterator but never generate the same path multiple times, giving the entire algorithm O(E * max flow) running time: generating a path takes at most O(E) time and we must generate at most the number of paths corresponding to the maximum flow through the graph.

An alternative way of generating paths is to run a normal search for generating paths, but not marking vertices as visited in a global data structure. Instead, when a path is to be extended with a new edge, first check if that edge (or destination vertex) is already in the path. Using a HashSet to store the edges in the path allows an O(1)-time check, and by storing the visited information in the path instead of in the graph we do not need to erase that information between paths.

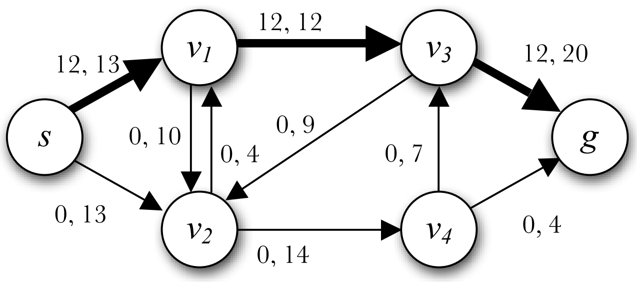
Ford and Fulkerson used depth-first search. Edmonds and Carp showed[2] that by using breadth-first search (measuring path distance as number of edges, regardless of capacity) the bound can be O(VE^2), which is lower than Ford and Fulkerson's bound for sparse graphs.

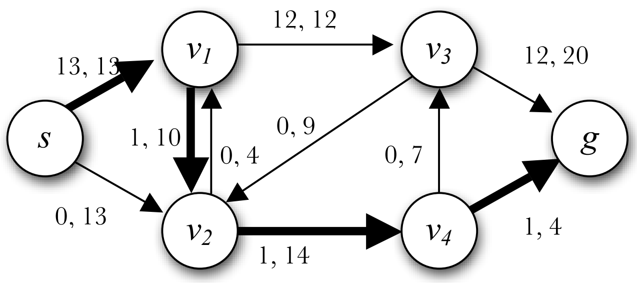
4 Example

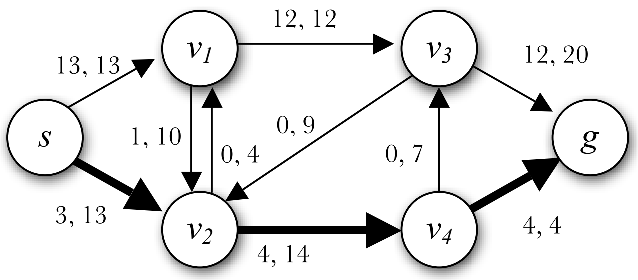
This section shows the computation of flow through a graph using the Ford-Fulkerson algorithm. Edges are labeled with f, c and the path to augment the flow in each step is shown in bold. The labels are those for *after* flow along the path is added.

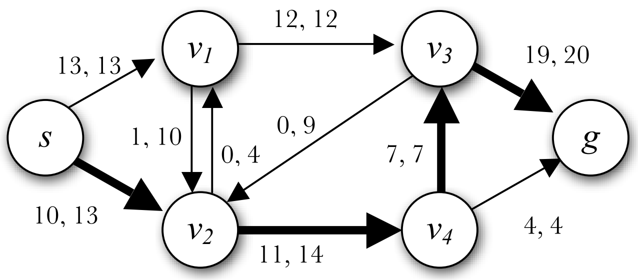


[2] J. Edmonds and R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, vol. 19, no. 2, pages 248—264, 1972









At this point, there is no way of transmitting more flow through the graph, since edges v_1v_3, v_4v_3 and v_4g are all saturated (N.B. those edges form a minimum cut.) The magnitude of the maximum flow through the network is therefore the amount of flow leaving the source, which is equal to the flow entering the sink, which is equal to the flow across the minimum cut. In this case, that value is 23. If we chose the paths in a different order, the values on edges not on the minimum cut might have been different, but the flow magnitude would still have been the same.

5 The Residual Network

The preferred implementation of the Ford-Fulkerson/Edmonds-Carp and other max flow algorithms does not actually operate on the original graph G. Instead, these algorithms use a *residual network*, which has a single label per edge. The labels in the residual network correspond to capacity − flow (i.e., residual) in the original graph. The residual network originally is identical to the original graph, however its labels and structure change during the solving process. Additional edges appear during solving that run opposite the original directed edges; these represent "backwards" or negative flow, which is a useful intermediate tool.