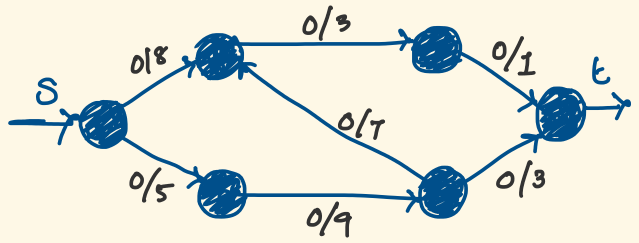
Network flow

Tuesday, October 31, 2023 6:15 AM

- A flow graph is a network of nodes each having a specific capacity
- It consists of a sink node (receiver) & a source node (sender)



Max flow w/o exceeding the capacity.

Applications

- roads with cars (traffic flow)
- water pipes
- Electric wires

max flow is the bottleneck value of the amount of traffic your n/w can handle

Ford Fulkerson method

- To find the max flow (& min-cut) the ford fulkerson method repeatedly finds augmenting paths through the residual graph & augments the flow until no more augmenting paths can be found.

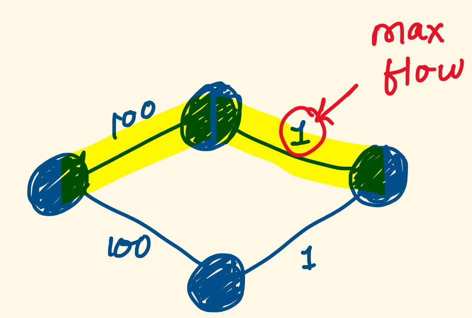
augmenting paths: path in a residual graph with unused capacity > 0 from 's' to 't'.

- Every augmenting path has a bottleneck value which is the smallest edge along the path.
- When augmenting a path we update the flow value of the edges along the augmenting path.
- For forward edges we increase the flow val, & decrease for backward (residual edges) by the bottleneck value.
  - undo an bad choices taken while augmenting.

* flow can be -ve.

* capacity is always +ve.

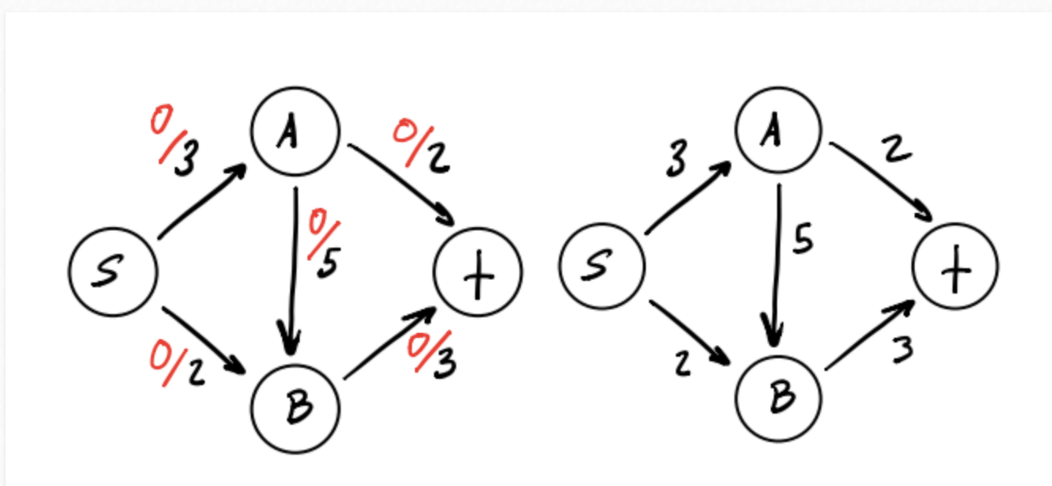
- ford fulkerson method keeps finding the augmenting paths until no more augmenting paths can be found.
- max flow = sum (bottlenecks)



https://downey.io/blog/max-flow-ford-fulkerson-algorithm-explanation/ ← see

This algorithm will look pretty similar to the one we laid out earlier, with one key difference. We will be constructing a residual graph for the flow network and searching for s-t paths across it instead!

1. Initially set the flow along every edge to 0.
2. Construct a residual graph for this network. It should look the same as the input flow network.



1. Use a pathfinding algorithm like depth-first search (DFS) or breadth-first search (BFS) to find a path P from s to t that has available capacity **in the residual graph**.
2. Let *cap(P)* indicate the maximum amount of stuff that can flow along this path. To find the capacity of this path, we need to look at all edges e on the path and subtract their current flow, f_e, from their capacity c_e. We'll set *cap(P)* to be equal to the smallest value of $c_e - f_e$ since this will bottleneck the path.
3. We then **augment the flow** across the forward edges in the path P by adding *cap(P)* value. For flow across the back edges in the residual graph, we subtract our *cap(P)* value.
4. Update the residual graph with these flow adjustments.
5. Repeat the process from step 2 until there are no paths left from s to t in the **residual graph** that have available capacity.

```
FordFulkerson(Graph G, Node s, Node t):
    Initialize flow of all edges e to 0.
    while(there is augmenting path(P) from s to t
    in the residual graph):
        Find augmenting path between s and t.
        Update the residual graph.
        Increase the flow.
    return
```

```
def search_path(s, t, parent):
    visited = [0] * n
    queue = deque()

    queue.append(s)
    visited[s] = True

    while queue:
        u = queue.popleft()

        for ind, val in enumerate(matrix[u]):
            if not visited[ind] and val > 0: # non zero capacity edge
                queue.append(ind)
                visited[ind] = True
                parent[ind] = u

    return True if visited[t] else False

def ford_fulkerson(source, sink):
    parent = [-1] * n
    max_flow = 0

    while search_path(source, sink, parent):
        flow = inf
        s = sink

        while s != source:
            flow = min(flow, matrix[ parent[s] ][s])
            s = parent[s]

        max_flow += flow

        v = sink
        while v != source:
            u = parent[v]
            matrix[u][v] -= flow
            matrix[v][u] += flow
            v = parent[v]

    return max_flow
```

we use adj. matrix as it is easy to update backward edges

finding Bottleneck

updating fwd & bckwd edges