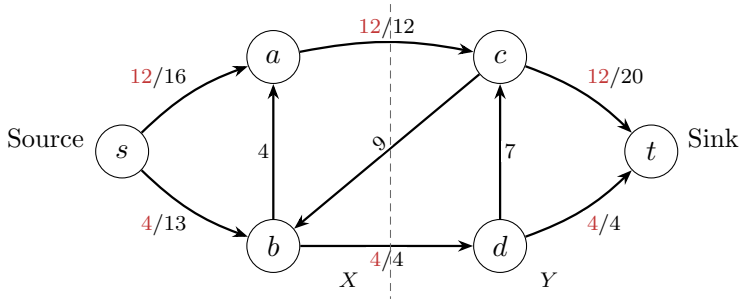
Flow Problems

November 16, 2026



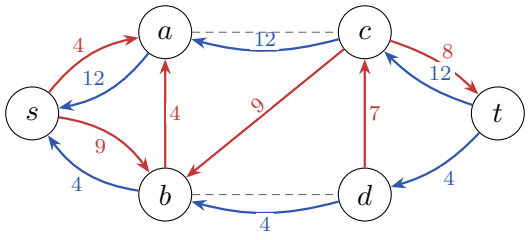
Labels on edges with flow are flow/capacity; labels with one number are capacity only.

Key Questions & Ideas:

- Find the **bottleneck** along a path: the smallest remaining capacity along that path.
- What is a good algorithm to figure out maximum flow? Find an $s \to t$ path with room for more flow, push its bottleneck amount, and repeat.
- Use this problem to solve other problems, such as bipartite matching.

Residual Graph (G_f) Intuition:

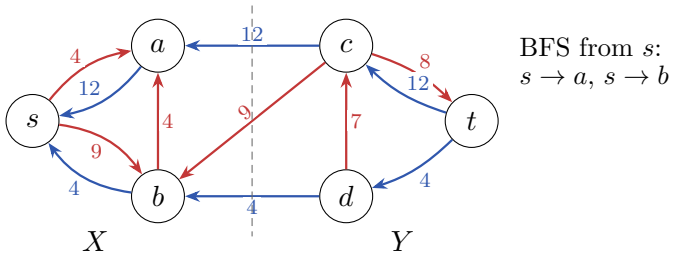
- **Blue (undo):** Backward edges let us undo flow already pushed, with remaining capacity $f(e)$.
- **Red (remaining):** Forward edges show what more can be pushed, with remaining capacity $c(e) - f(e)$.
- **Black:** Shows the path taken in the handwritten example. Dashed lines are retained as reference edges; only residual edges with positive capacity can be used in a new path.



Valid Flow Conditions:

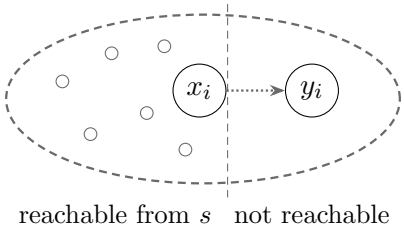
1. **Capacity:** $0 \leq f(e) \leq c(e)$ for every edge e.
2. **Flow conservation:** Flow entering v equals flow leaving v for every $v \notin \{s, t\}$.

Vertices reachable from s (BFS)



Points reachable from s: $X = \{s, a, b\}$. The other vertices form $Y = \{c, d, t\}$.

See sets X and Y:



If x_i is reachable from s, a positive-capacity residual edge $x_i \to y_i$ would make y_i reachable too. So there cannot be a positive-capacity residual edge crossing from X to Y.

Here, the original edges from X to Y are $a \to c$ (capacity 12) and $b \to d$ (capacity 4). Both are full, and there is no remaining $s \to t$ path. The total flow is

$$\boxed{12 + 4 = 16.}$$

Ford–Fulkerson Algorithm

Algorithm:

While there is an $s \to t$ path in G_f:
  Push the bottleneck amount of flow along the path.
  Update G_f.

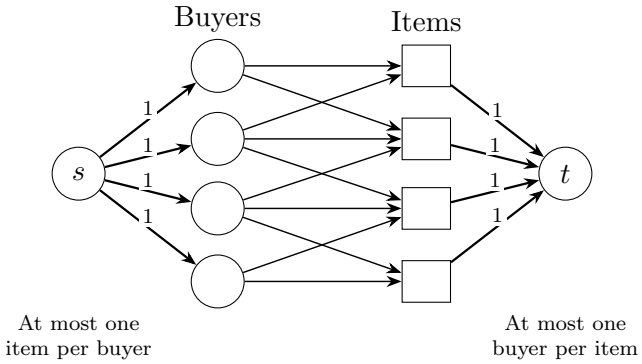
Let n be the number of vertices, m the number of edges, and C the maximum flow value (for integer capacities).

Runtime: With integer capacities, there are at most C augmentations. The running time is $O\big(C(n + m)^2\big)$.

Using BFS to find a path takes $O((n + m)^2)$ time per search. With adjacency lists, each search takes $O(n + m)$ time, giving a total of $O(C(n + m))$.

Bipartite Matching

Problem: Given n buyers and m items to sell, each buyer is interested in only a subset of the items and wants to buy at most one item.



All arrows point left to right; each has capacity 1

Equivalence:

- Given an assignment of k different items to k buyers, we can send 1 unit of flow on $s \to$ buyer $\to$ item $\to t$ for each pair. The total flow is k.
- Conversely, an integer flow of value k gives k buyer–item pairs, with no buyer or item used more than once.
- Thus finding the largest assignment is the same as finding the largest flow in this network.

Runtime Analysis:

- Constructing the graph takes $O(n+m+nm)$ time: there are $n+m+2$ vertices and at most $n+m+nm$ edges.
- If the maximum-flow algorithm takes $f(n)$ time, then the total time for bipartite matching is

$$\boxed{O(n+m+nm)+f(n).}$$

  Here $f(n)$ is shorthand for the time to solve max flow on the constructed network.

- If a maximum-flow algorithm takes $O(N^3)$ time on a network with N vertices, then bipartite matching can also be solved in $O(n^3)$ time when there are n buyers and n items. The constructed network has $N = 2n + 2 = O(n)$ vertices, and constructing it takes $O(n^2)$ time. So the total is

$$O(n^2) + O(n^3) = \boxed{O(n^3)}.$$