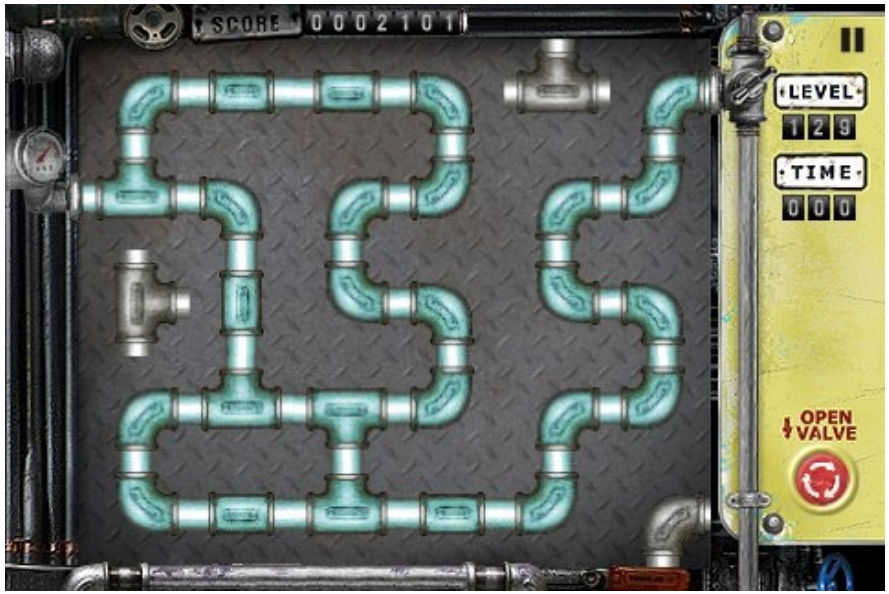
Graphs: Maximum Flows

Jordi Cortadella and Jordi Petit
Department of Computer Science

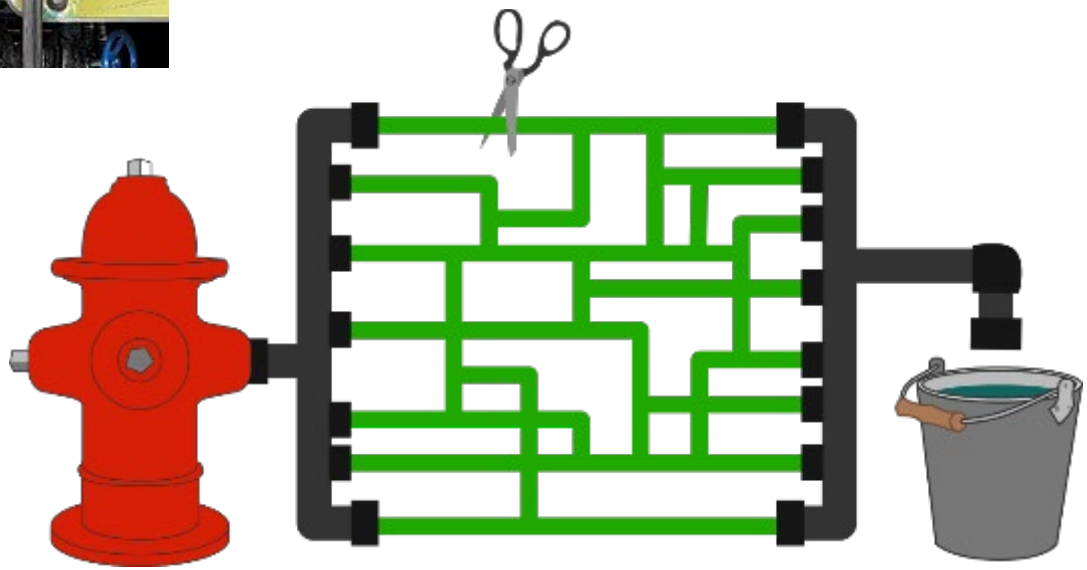
Max-flow/min-cut problems



OpenValve, by JAE HYUN LEE

How much water can you pump from source to target?

What is the fewest number of green tubes that need to be cut so that no water will be able to flow from the hydrant to the bucket?



Max-flow/Min-cut algorithm. Brilliant.org. https://brilliant.org/wiki/max-flow-min-cut-algorithm/

Max-flow/min-cut problems: applications

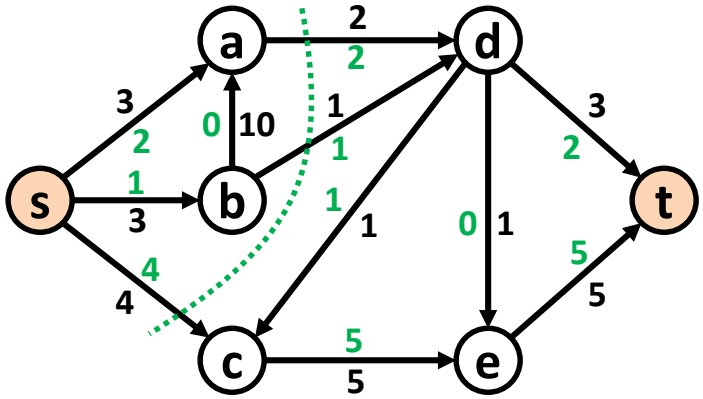
- Networks that carry data, water, oil, electricity, cars, etc.
  - How to maximize usage?
  - How to minimize cost?
  - How to maximize reliability?

- Multiple application domains:
  - Computer networks
  - Image processing
  - Computational biology
  - Airline scheduling
  - Data mining
  - Distributed computing
  - …

Max-flow problem

Model:

- A directed graph $G = (V, E)$.
- Two special nodes $s, t \in V$.
- Capacities $c_e > 0$ on the edges.



Goal: assign a flow f_e to each edge e of the network satisfying:

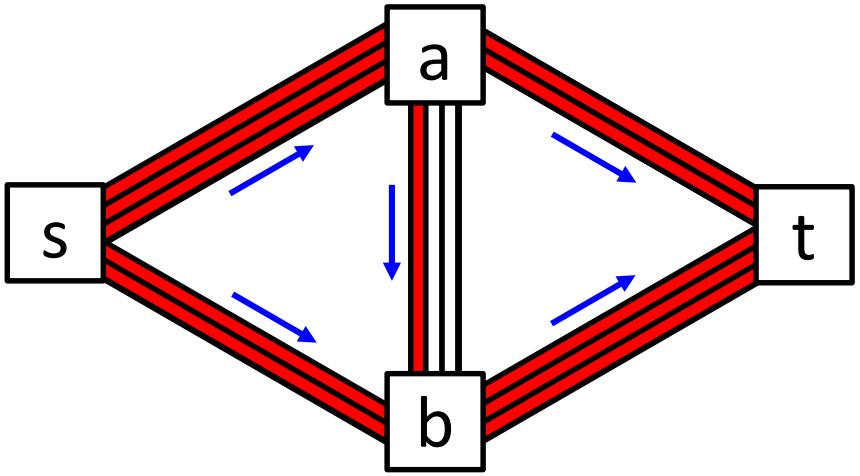
- $0 \leq f_e \leq c_e$ for all $e \in E$ (edge capacity not exceeded)
- For all nodes u (except s and t), the flow entering the node is equal to the flow exiting the node:

$$\sum_{(w,u)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}\,.$$

Size of a flow: total quantity sent from s to t (equal to the quantity leaving s):

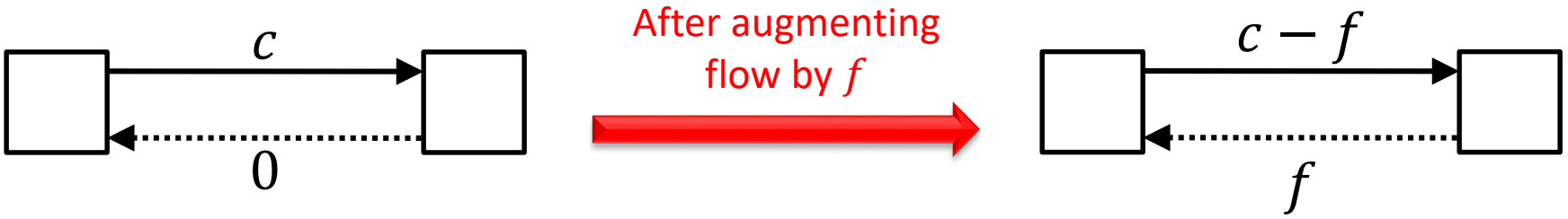
$$\text{size}(f) = \sum_{(s,u)\in E} f_{su}$$

Max-flow problem: intuition

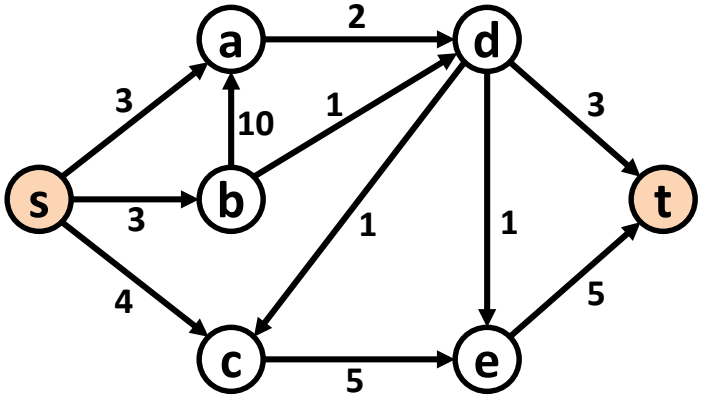


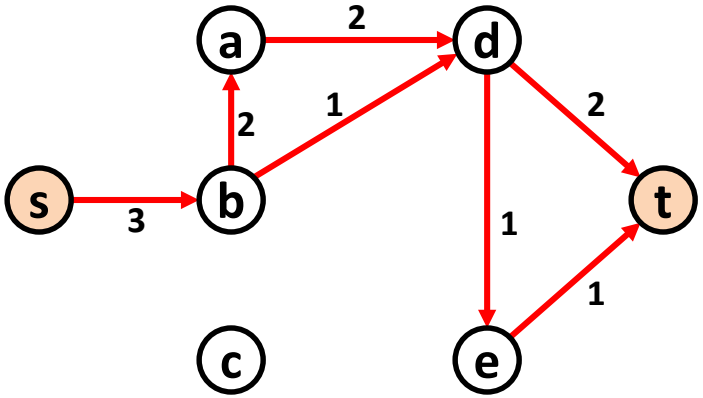
Find an augmenting path

An augmenting path may reverse some of the flow previously assigned



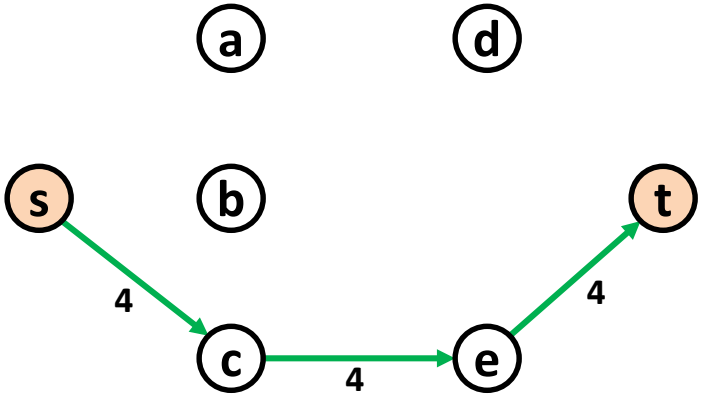
Augmenting paths



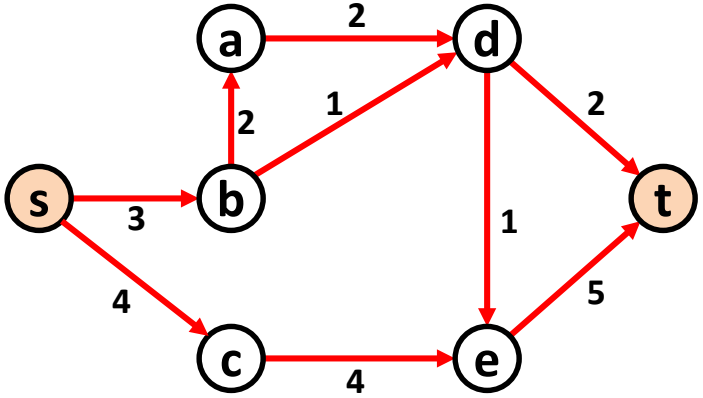


Flow

Given a flow, an ***augmenting path*** represents a feasible additional flow from s to t.

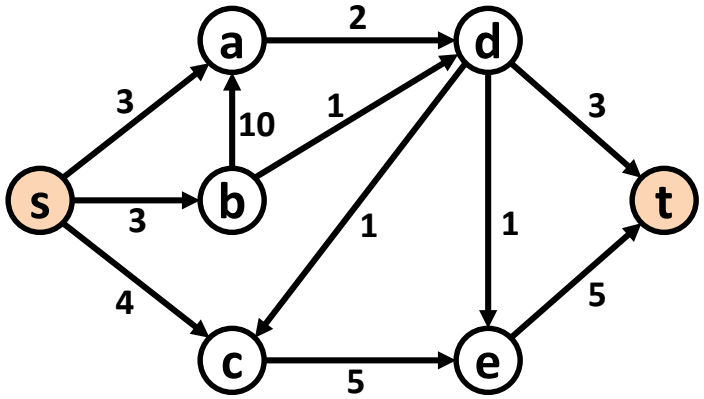


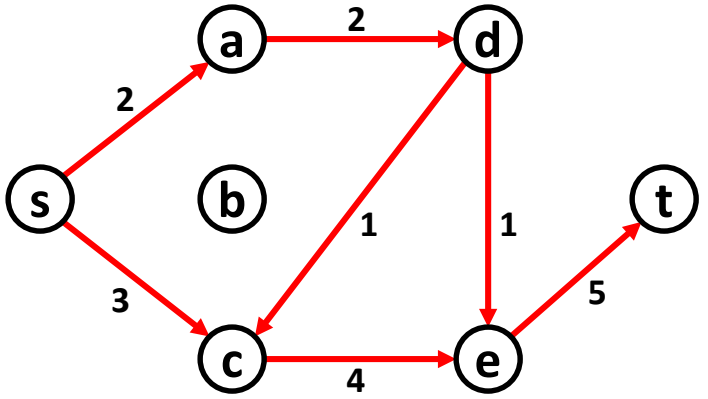
Augmenting path



New flow

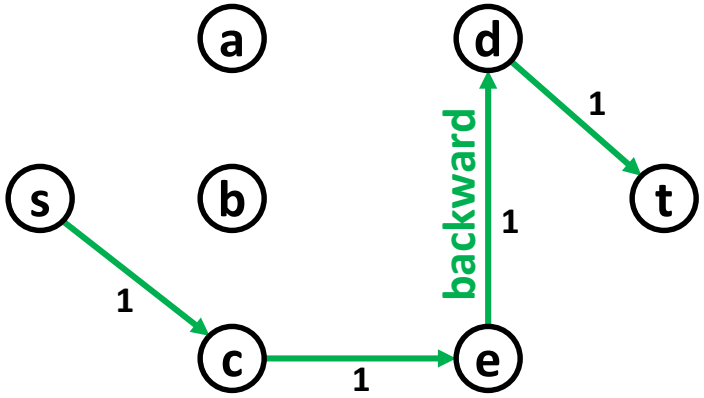
Augmenting paths



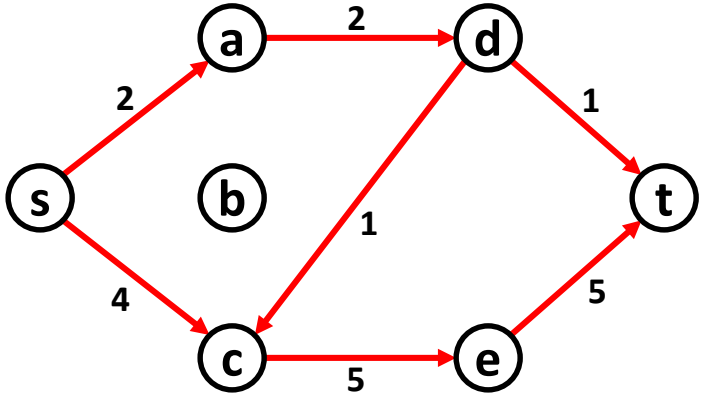


Flow

Augmenting paths can have *forward* and *backward* edges.



Augmenting path



New flow

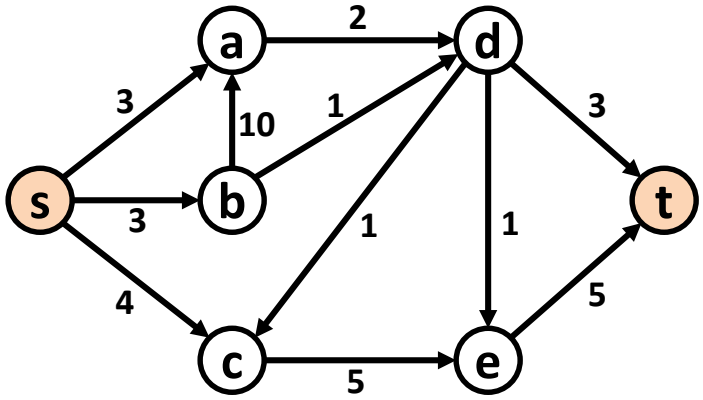
Augmenting paths

Given a flow f, an augmenting path is a directed path from s to t, which consists of edges from E, but not necessarily in the same direction. Each of these edges e satisfies exactly one of the following two conditions:

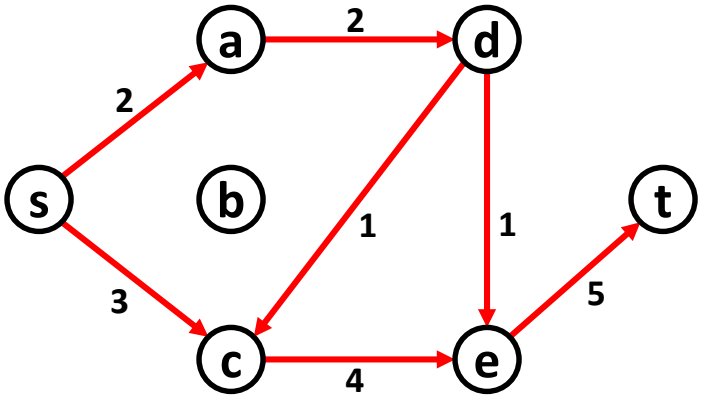
- e is in the same direction as in E (forward) and $\mathrm{f_e} < c_e$. The difference $c_e - f_e$ is called the *slack* of the edge.

- e is in the opposite direction (backward) and $f_e > 0$. It represents the fact that some flow can be borrowed from the current flow.

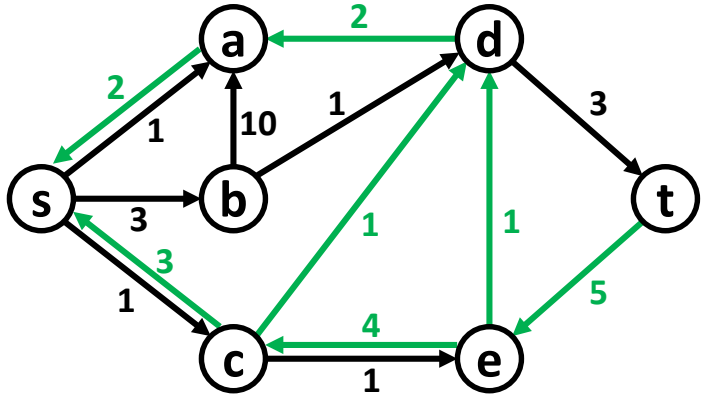
Residual graph



Graph



Flow

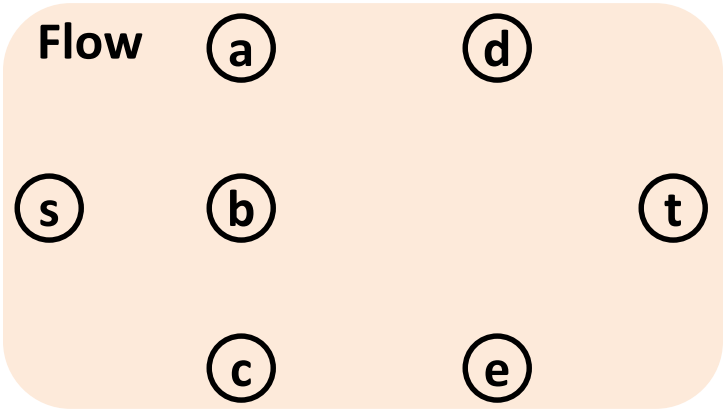


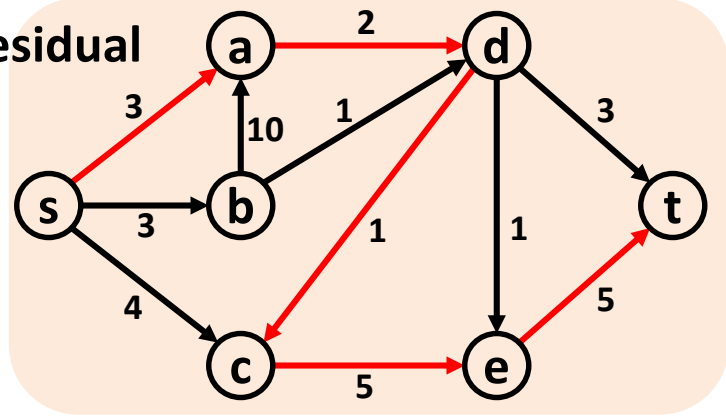
Residual graph

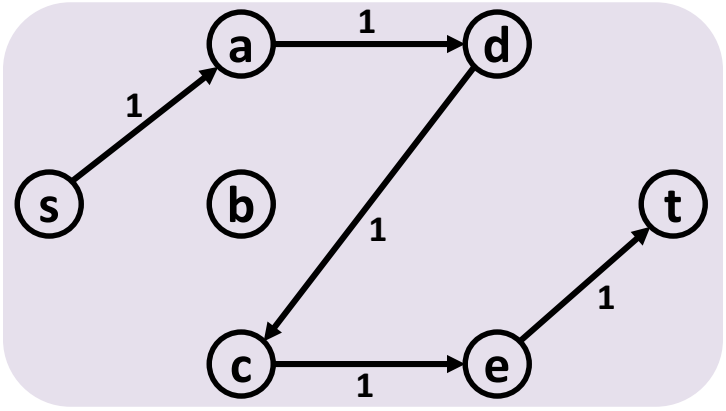
forward

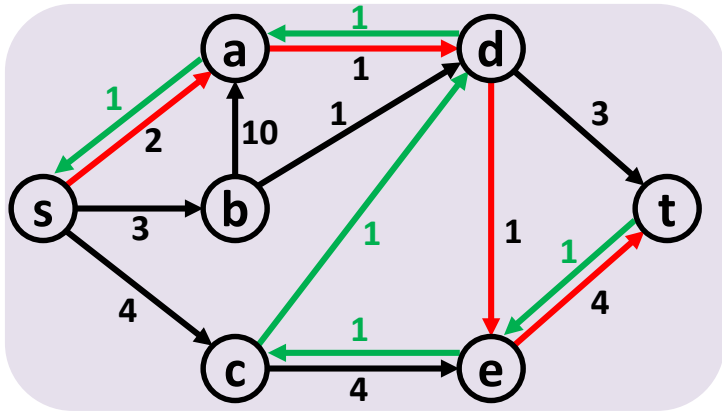
backward

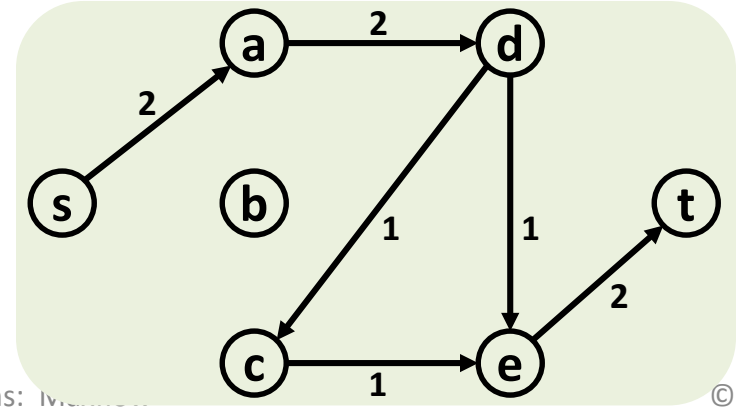
Ford-Fulkerson algorithm: example

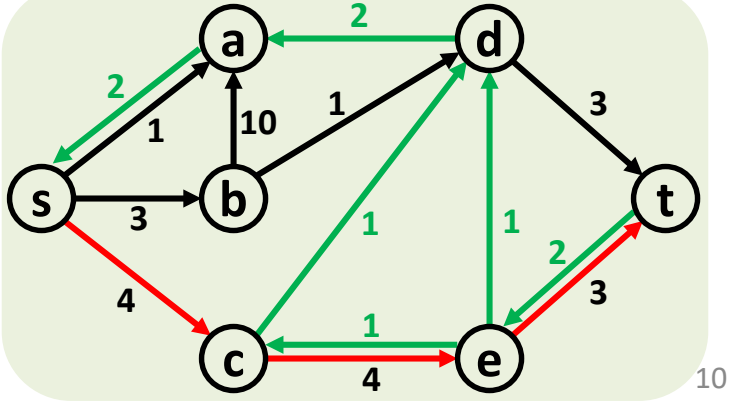




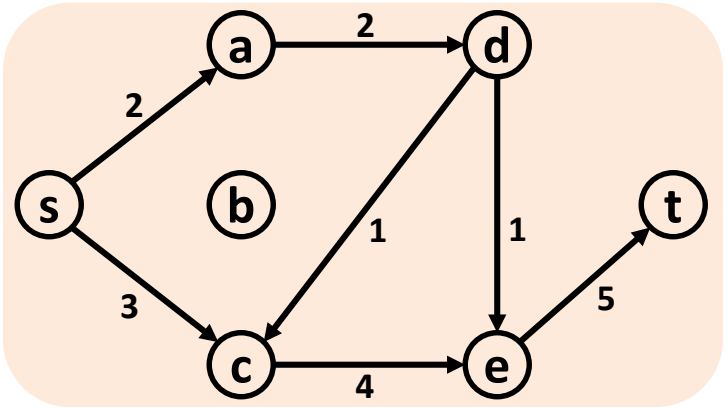


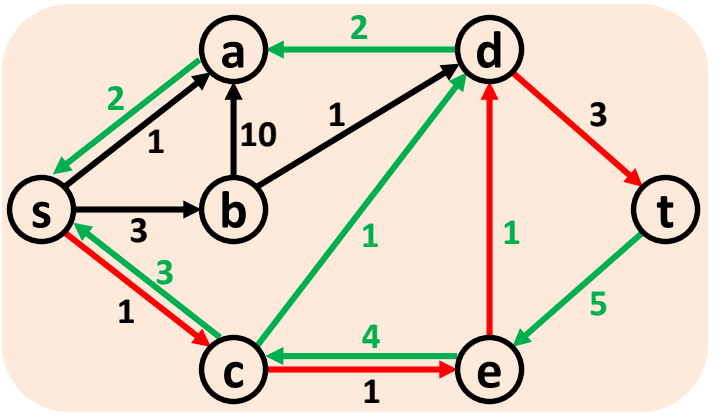


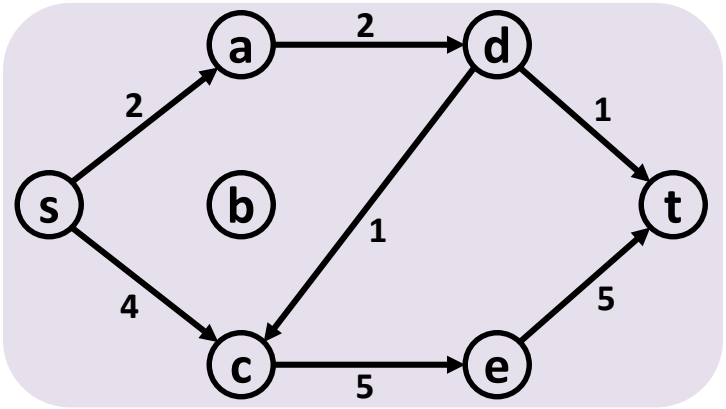


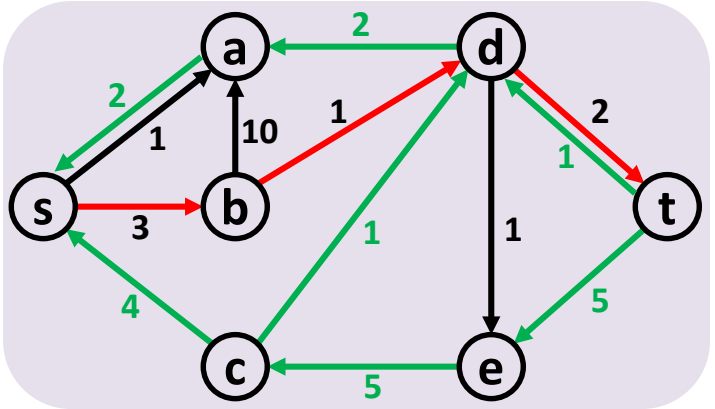


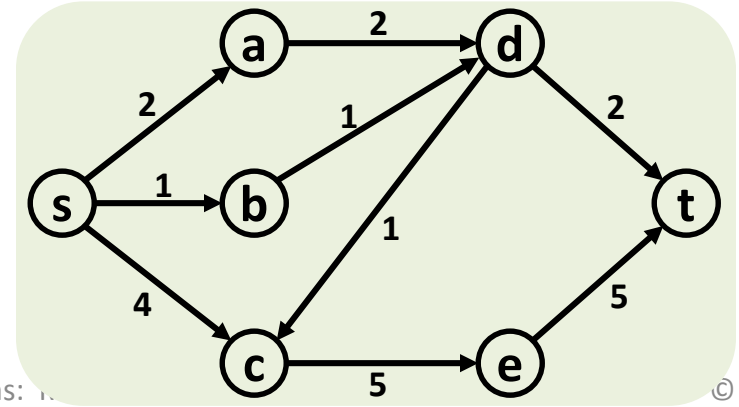
Ford-Fulkerson algorithm: example

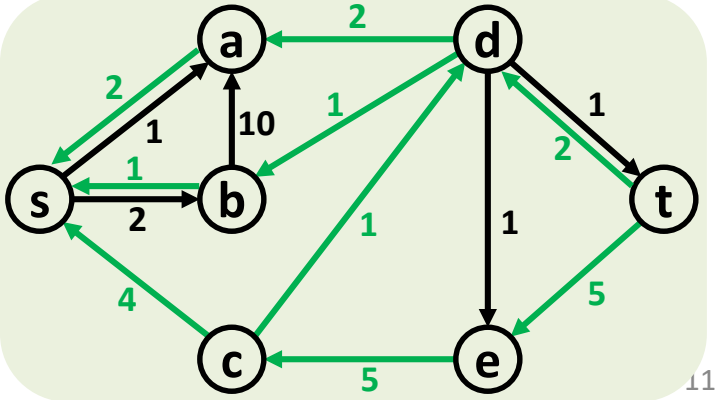












Ford-Fulkerson algorithm

```
def Ford-Fulkerson(G, s, t) → f:
  """Input: A directed Graph G(V,E) with edge capacities c_e.
            s and t and the source and target of the flow.
     Output: A flow f that maximizes the size of the flow.
             For each (u,v) ∈ E, f(v,u) represents its flow.
  """
  for all (u,v) ∈ E:
    f(u,v) = c(u,v)   # Forward edges
    f(v,u) = 0        # Backward edges

  while there exists a path p = s ⇝ t in the residual graph:
    f(p) = min{f(u,v): (u,v) ∈ p}
    for all (u,v) ∈ p:
      f(u,v) = f(u,v) − f(p)
      f(v,u) = f(v,u) + f(p)
```

Ford-Fulkerson algorithm: complexity

- Finding a path in the residual graph requires $\mathrm{O}(|E|)$ time (using BFS or DFS).

- How many iterations (augmenting paths) are required?
  - The worst case is really bad: $\mathrm{O}(C \cdot |E|)$, with C being the largest capacity of an edge (if only integral values are used).
  - By selecting the path with fewest edges (using BFS) the maximum number of iterations is $\mathrm{O}(|V| \cdot |E|)$.
  - By carefully selecting *fat* augmenting paths (using some variant of Dijkstra's algorithm), the number of iterations can be reduced.

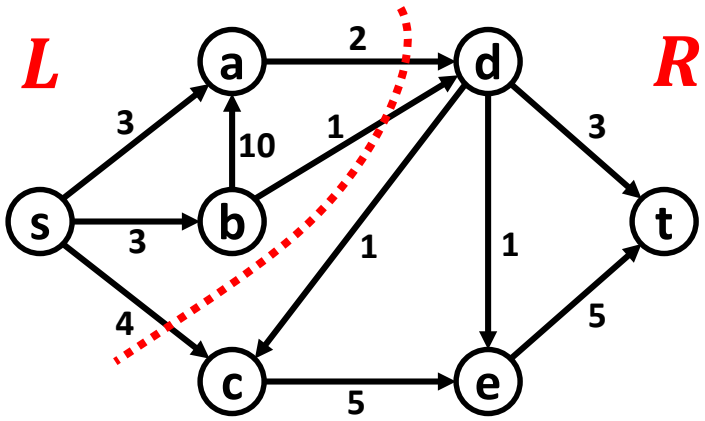
- Ford-Fulkerson algorithm is $\mathrm{O}(|V| \cdot |E|^2)$ if BFS is used to select the path with fewest edges (Edmonds-Karp algorithm).

Max-flow problem

Cut: An (s, t)-cut partitions the nodes into two disjoint groups, L and R, such that $s \in L$ and $t \in R$.

For any flow f and any (s, t)-cut (L, R):

$$\text{size}(f) \leq \text{capacity}(L, R).$$



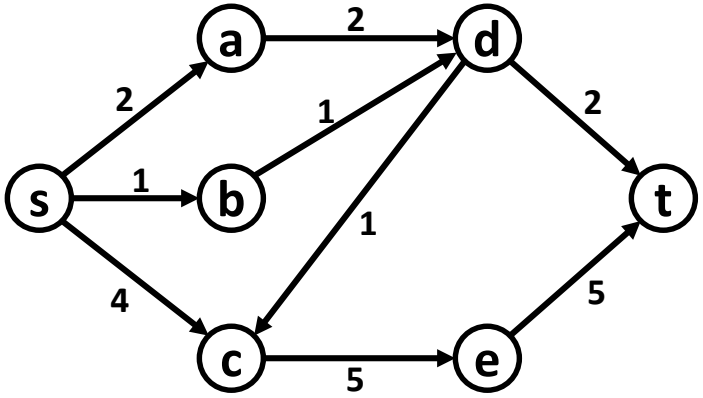
The max-flow min-cut theorem:

The size of the maximum flow equals the capacity of the smallest (s, t)-cut.

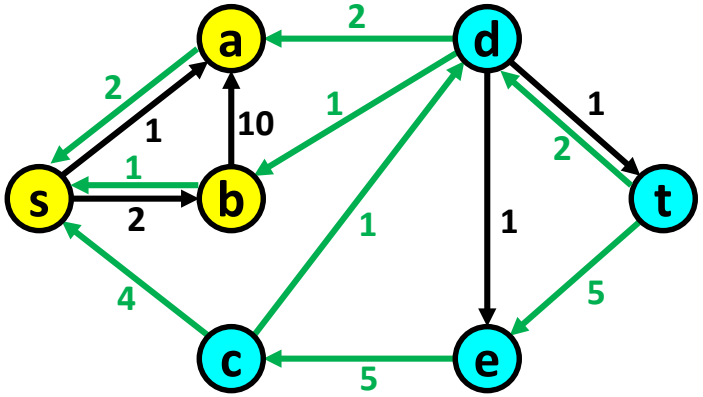
The augmenting-path theorem:

A flow is maximum iff it admits no augmenting path.

Min-cut algorithm



Flow

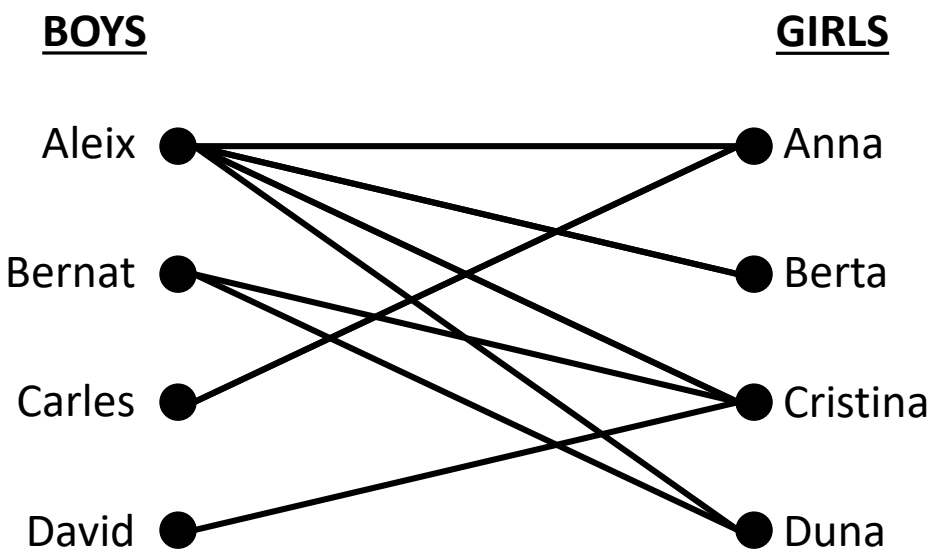


Residual graph

Finding a cut with minimum capacity:

1. Solve the max-flow problem with Ford-Fulkerson.
2. Compute L as the set of nodes reachable from s in the residual graph.
3. Define $R = V - L$.
4. The cut (L, R) is a min-cut.

Bipartite matching



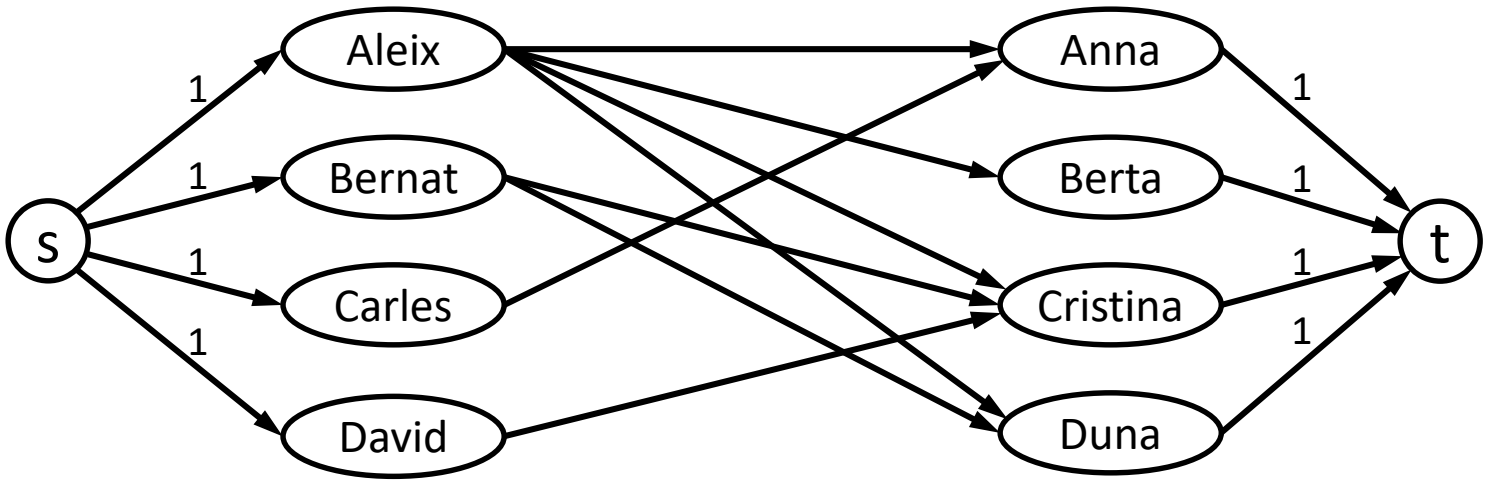
There is an edge between a boy and a girl if they like each other.

Can we pick couples so that everyone has exactly one partner that he/she likes?

Bad matching: if we pick (Aleix, Anna) and (Bernat, Cristina), then we cannot find couples for Berta, Duna, Carles and David.

A ***perfect matching*** would be: (Aleix, Berta), (Bernat, Duna), (Carles, Anna) and (David, Cristina).

Bipartite matching



Reduced to a max-flow problem with $c_e = 1$.

Question: can we always guarantee an integer-valued flow?

Property: if all edge capacities are integer, then the optimal flow found by Ford-Fulkerson's algorithm is integral. It is easy to see that the flow of the augmenting path found at each iteration is integral.

Extensions of Max-Flow

- **Max-Flow with Edge Demands**
  - Each edge e has a demand $d(e)$. The flow f must satisfy $d(e) \leq f(e) \leq c(e)$.

- **Node Supplies and Demands**
  - An extra flow $x(v)$ can be injected (positive) or extracted (negative) at every vertex v. The flow must satisfy:

$$\sum_{u \in V} f(u \to v) - \sum_{w \in V} f(v \to w) = x(v)\,.$$

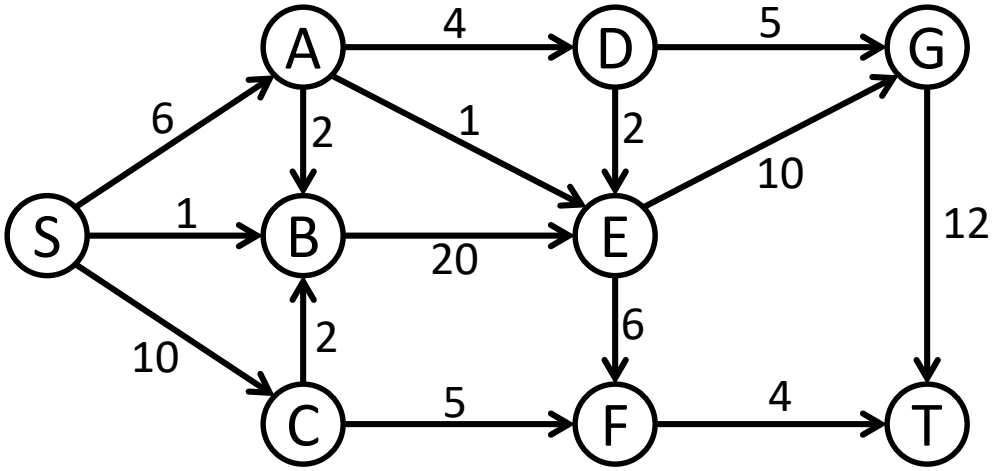
- **Min-cost Max-Flow**
  - Each edge e has a weight w_e. Compute a max-flow of minimum cost:

$$\text{cost}(f) = \sum_{e \in E} w_e \cdot f(e)$$

- **Max-Weight Bipartite Matching**
  - Each edge e has a weight w_e. Find a maximum cardinality matching with maximum total weight.

EXERCISES

Flow network (from [DVP2008])



- Find the maximum flow from S to T. Give a sequence of augmenting paths that lead to the maximum flow.
- Draw the residual graph after finding the maximum flow.
- Find a minimum cut between S and T.