Lecture 8

Graphs IV – Flow Problems and Matching

Dominic Zimmer, Karl Bringmann,
Christoph Weidenbach

Saarland University

June, 2020

Overview

- Graphs I: Traversals and Shortest Paths
- Graphs II: DFS Applications and Friends
- Graphs III: Trees
- Graphs IV: Flow Problems and Matching

Flow Problems

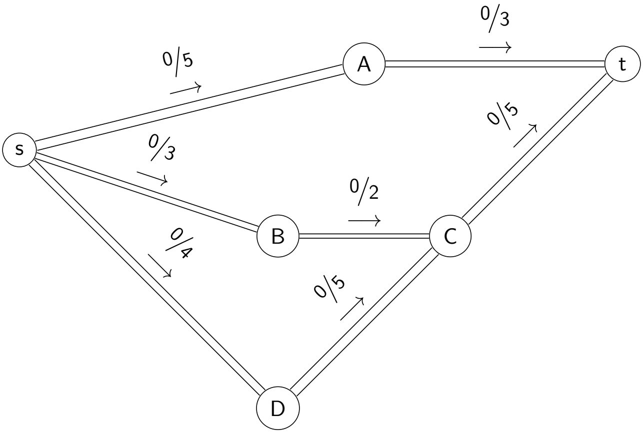
Problem: Water Flow

Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?

Flow Problems

Problem: Water Flow

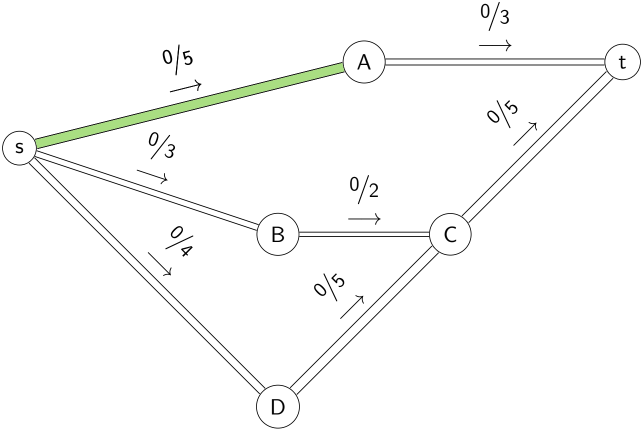
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

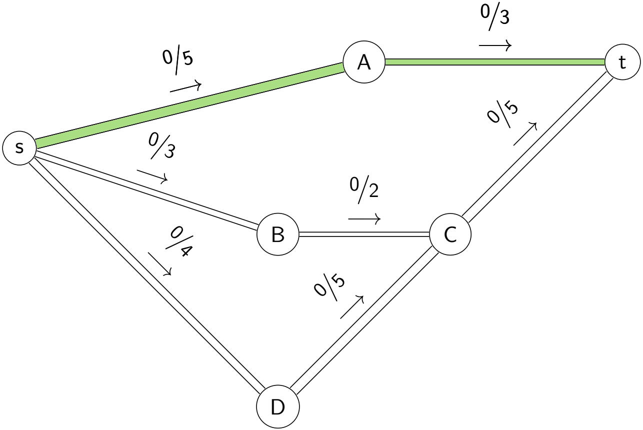
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

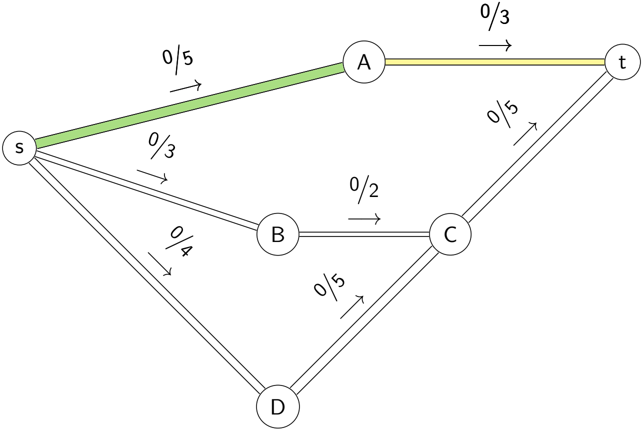
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

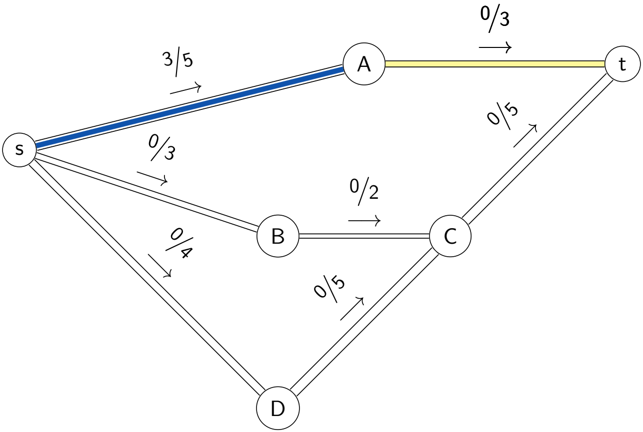
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

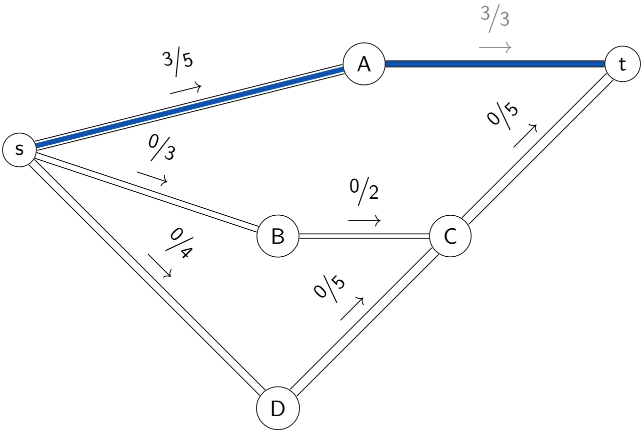
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

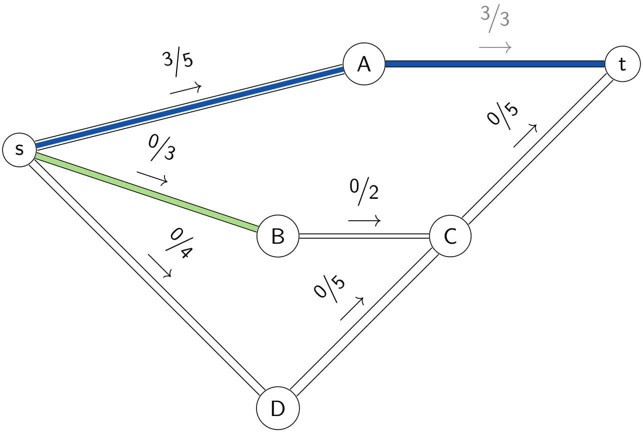
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

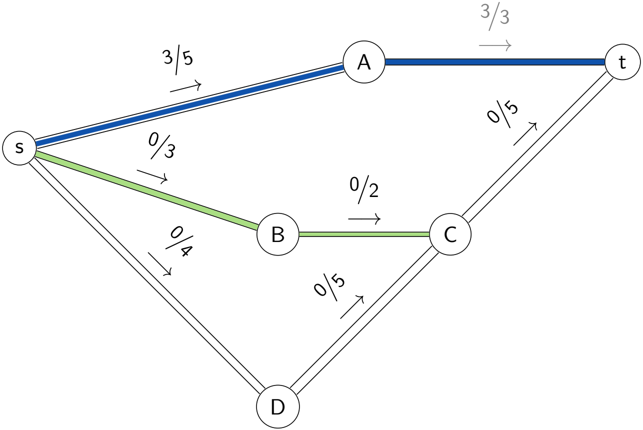
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

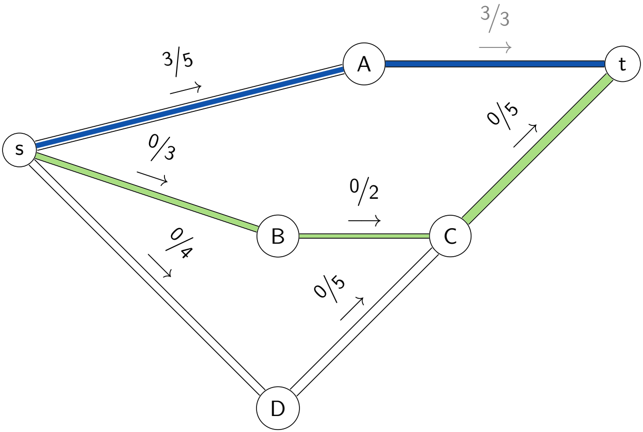
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

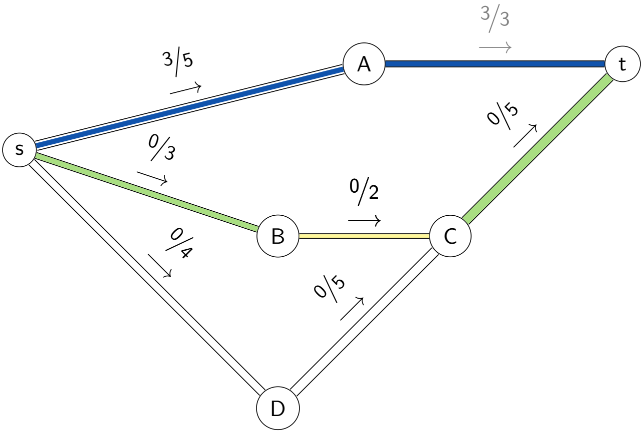
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

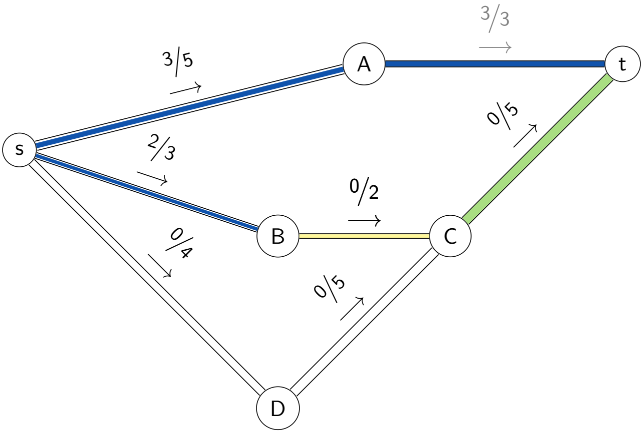
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

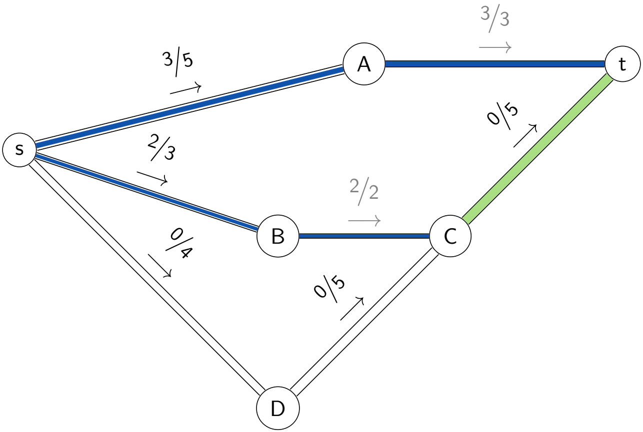
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

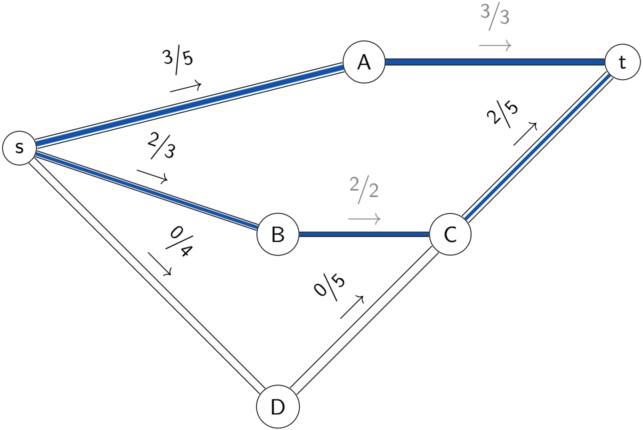
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

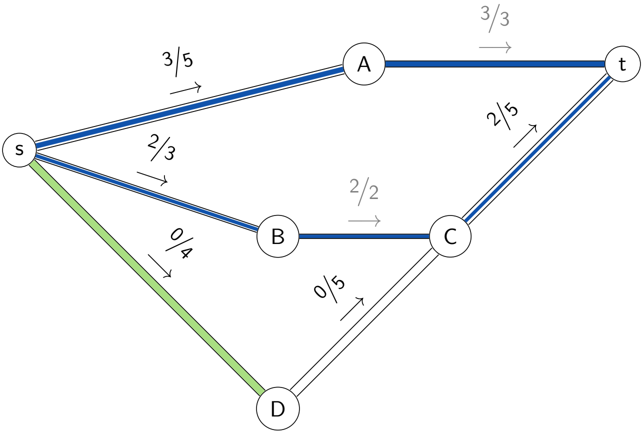
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

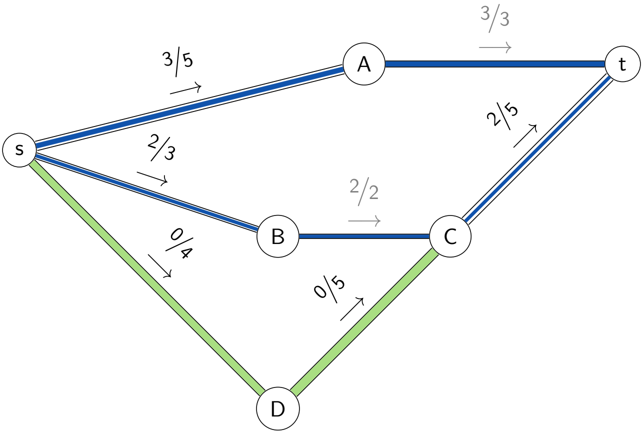
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

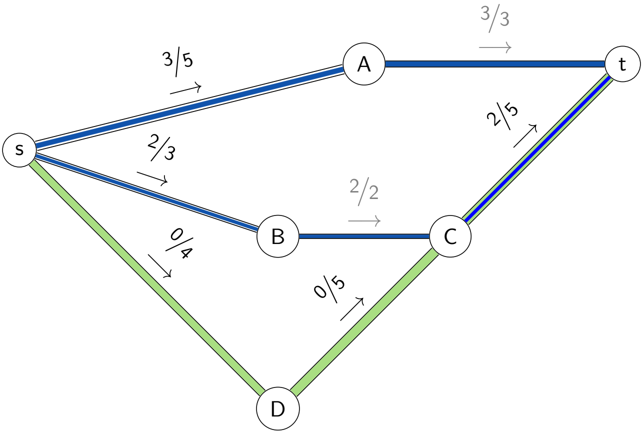
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

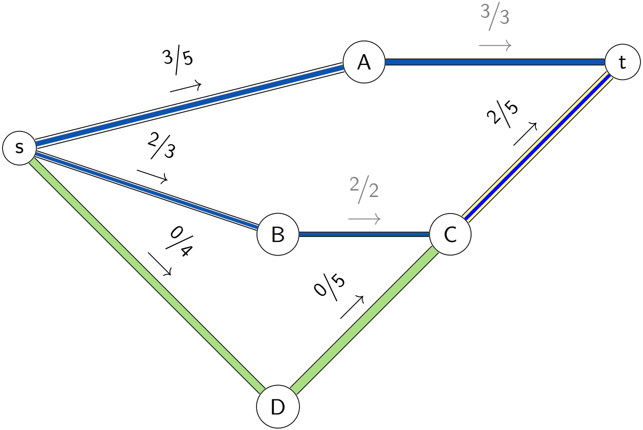
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

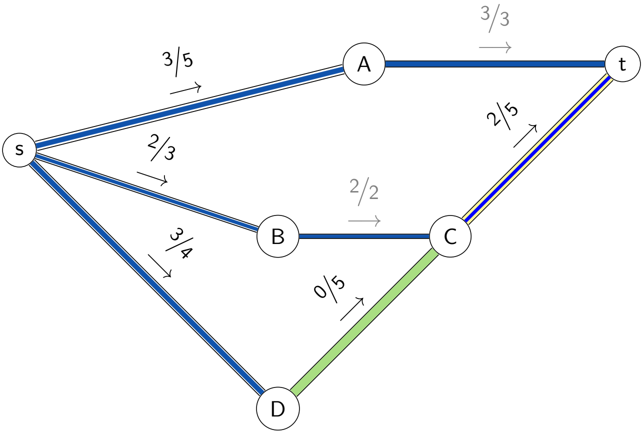
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

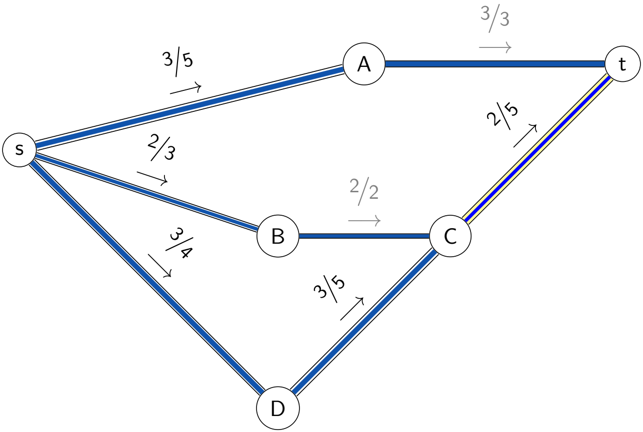
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

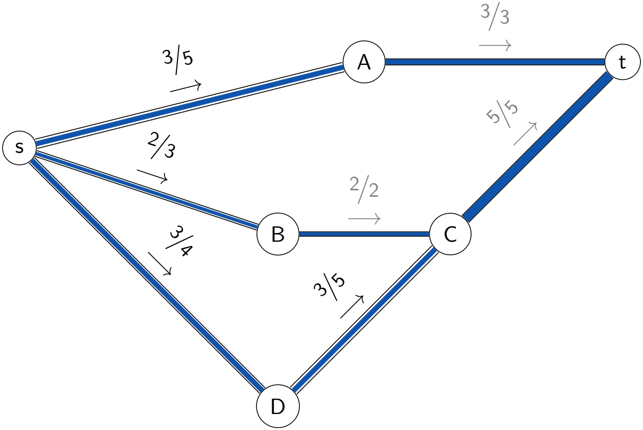
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Problem: Water Flow

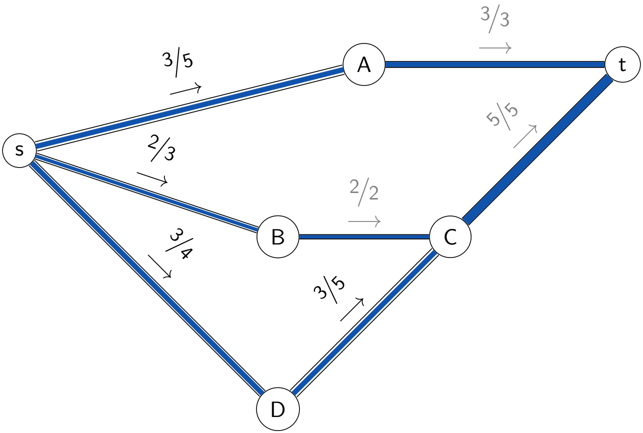
Given a network of unidirectional pipes, source and sink.
How much water over time can you pump from s to t?



Flow Problems

Solution: Water Flow

At most 8 units of water can flow from s to t simultaneously.



Working Definition: Maximum Flow

Given a directed Graph G with capacity c_e for every edge e, a source vertex s, and a sink vertex t, we call the maximum amount of *flow* (e.g. water) that can flow from s to t simultaneously the *Maximum Flow* of G.

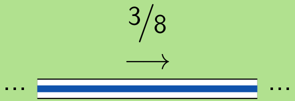
Flow is subject to the conditions that

- in every vertex, except in s and t, the in-flow equals the out-flow
- no edge can support more flow than its capacity

Flow Problems

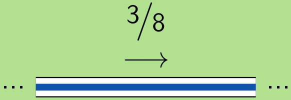
Ford-Fulkerson

Some Nomenclature



- edge capacity: 8
- current flow: 3
- residual capacity: 5

Some Nomenclature



- edge capacity: 8
- current flow: 3
- residual capacity: 5

Algorithm Idea: Ford-Fulkerson

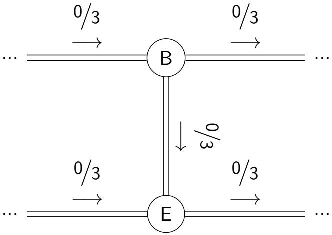
Find a path from s to t that can still take some water, a so-called *Augmenting Path*

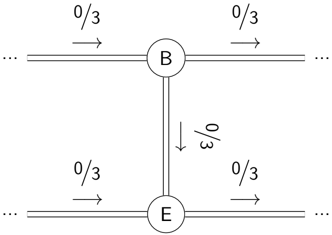
- The *bottleneck capacity* c is the minimum residual capacity along this augmenting path
- Augment the current flow of each edge on the path by c

The maximum flow equals the total flow added until no Augmenting Path exists

Flow Problems

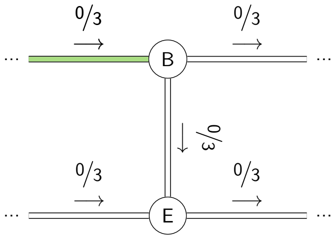
Ford-Fulkerson

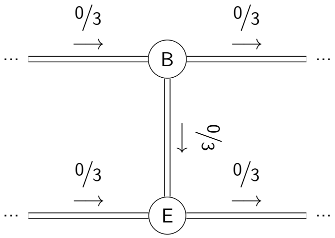




Flow Problems

Ford-Fulkerson



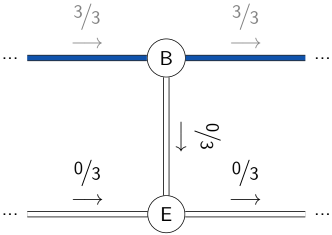


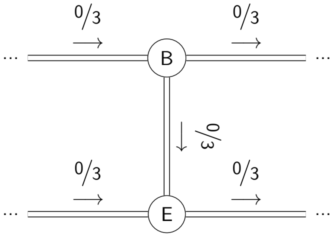
Flow Problems

Ford-Fulkerson

Flow Problems

Ford-Fulkerson



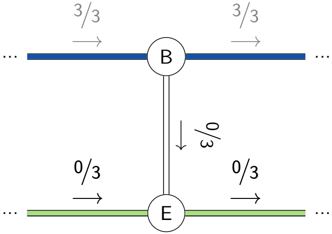


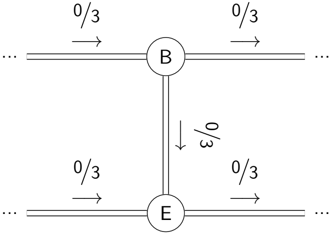
Flow Problems

Ford-Fulkerson

Flow Problems

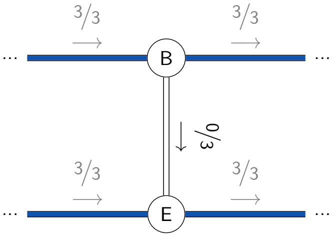
Ford-Fulkerson

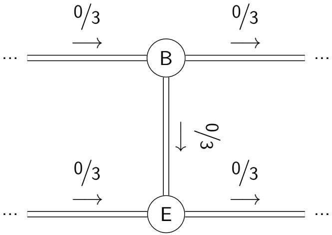




Flow Problems

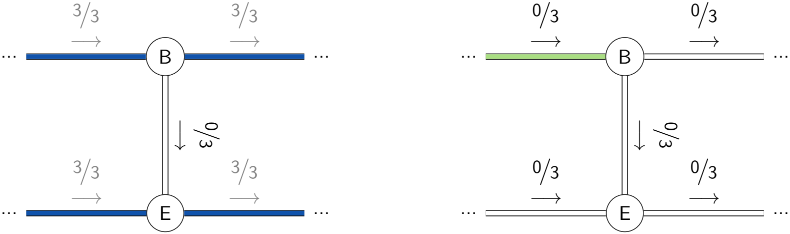
Ford-Fulkerson





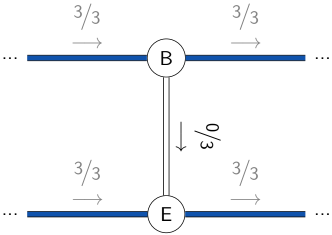
Flow Problems

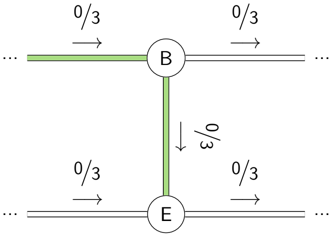
Ford-Fulkerson



Flow Problems

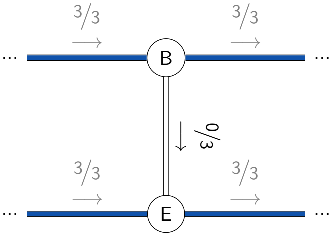
Ford-Fulkerson

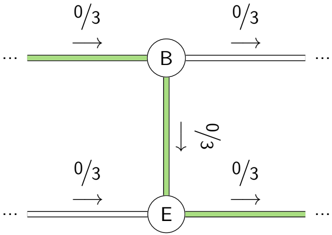




Flow Problems

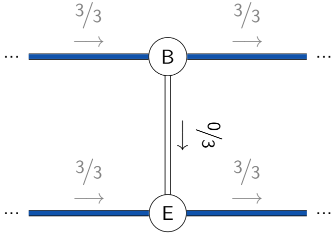
Ford-Fulkerson

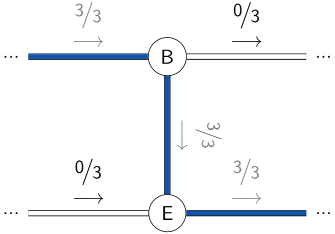




Flow Problems

Ford-Fulkerson





Flow Problems

Ford-Fulkerson

Flow Problems

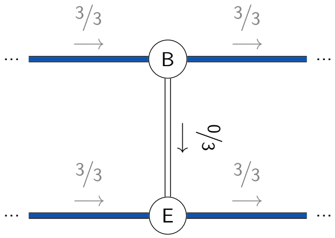
Ford-Fulkerson

Flow Problems

Ford-Fulkerson

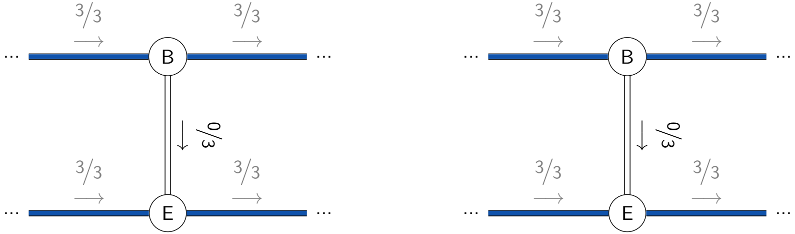
Flow Problems

Ford-Fulkerson



Flow Problems

Ford-Fulkerson

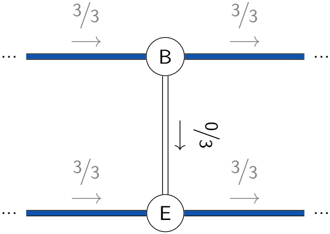


Remark: Back Edges

Flow can flow through back edges, *cancelling* prior flow. The residual backwards capacity is the current forwards flow.

Flow Problems

Ford-Fulkerson



Remark: Back Edges

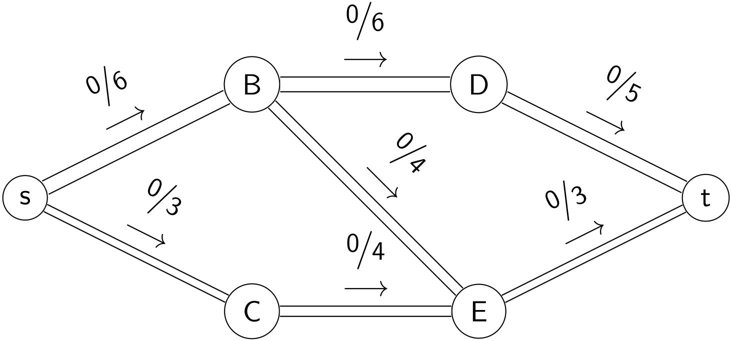
Flow can flow through back edges, *cancelling* prior flow. The residual backwards capacity is the current forwards flow.

Remark: Augmenting Paths

Several Augmenting Paths may be available to chose from. Any such path is sufficient for the correctness of Ford-Fulkerson.

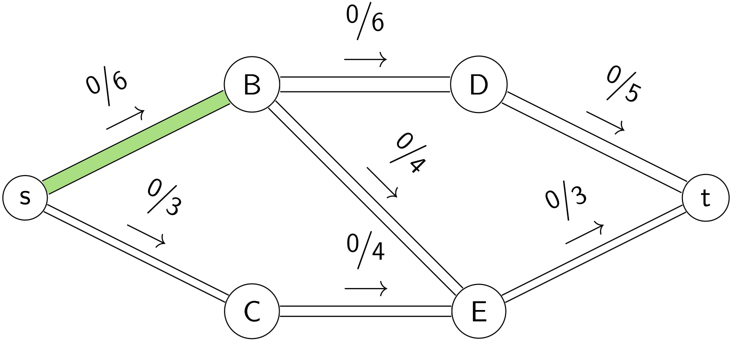
Flow Problems

Ford-Fulkerson: Example Run



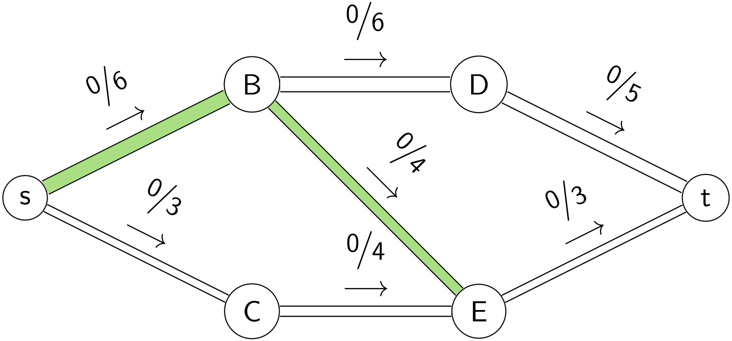
Flow Problems

Ford-Fulkerson: Example Run



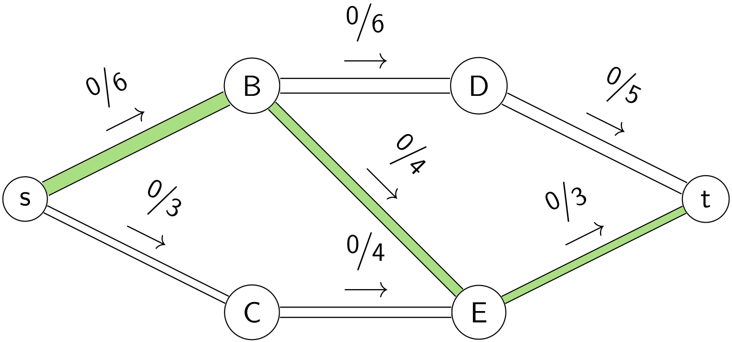
Flow Problems

Ford-Fulkerson: Example Run



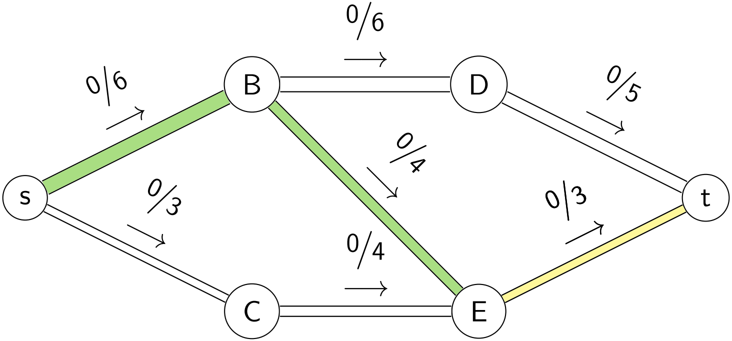
Flow Problems

Ford-Fulkerson: Example Run



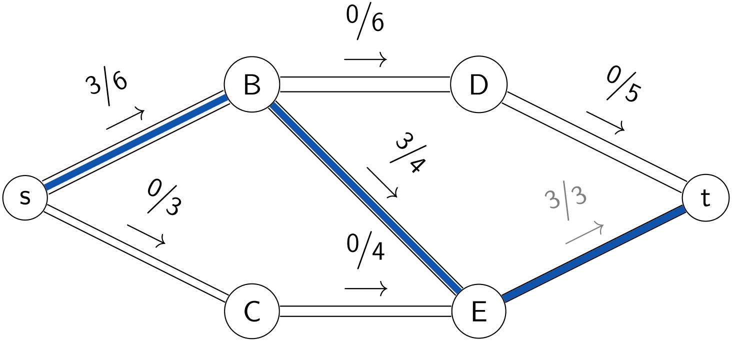
Flow Problems

Ford-Fulkerson: Example Run



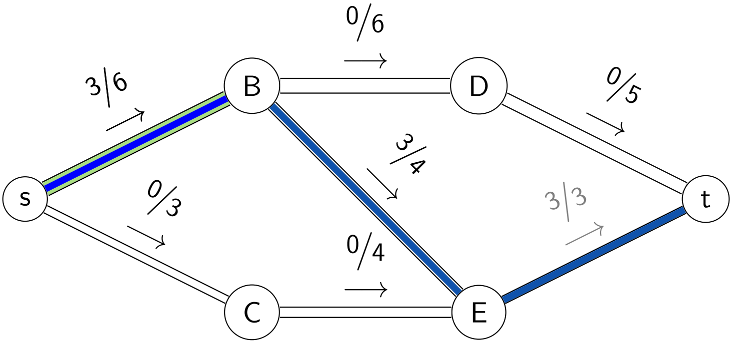
Flow Problems

Ford-Fulkerson: Example Run



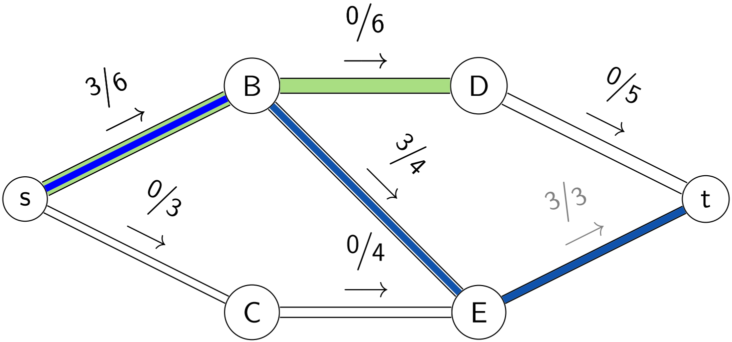
Flow Problems

Ford-Fulkerson: Example Run



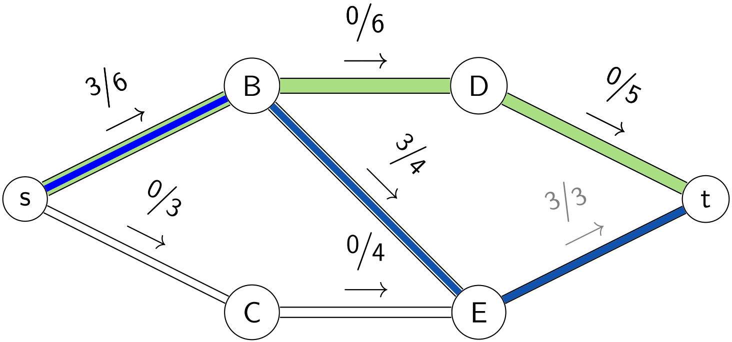
Flow Problems

Ford-Fulkerson: Example Run



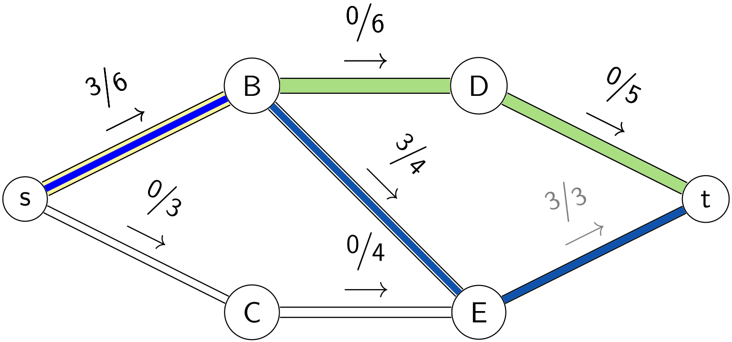
Flow Problems

Ford-Fulkerson: Example Run



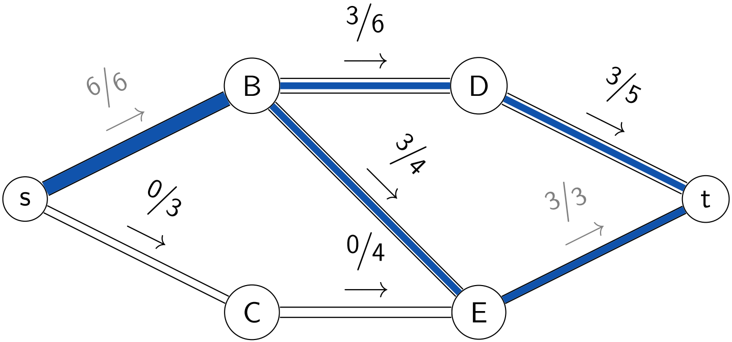
Flow Problems

Ford-Fulkerson: Example Run



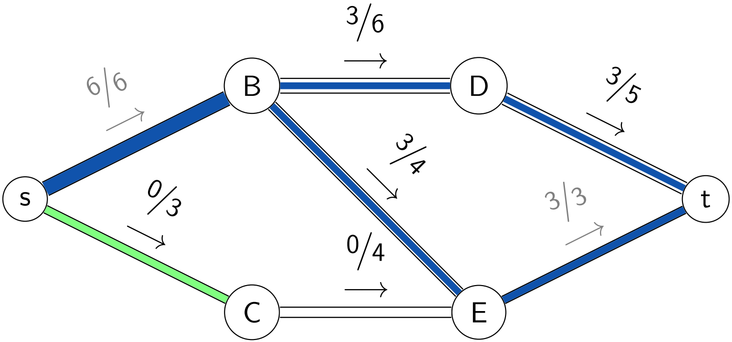
Flow Problems

Ford-Fulkerson: Example Run



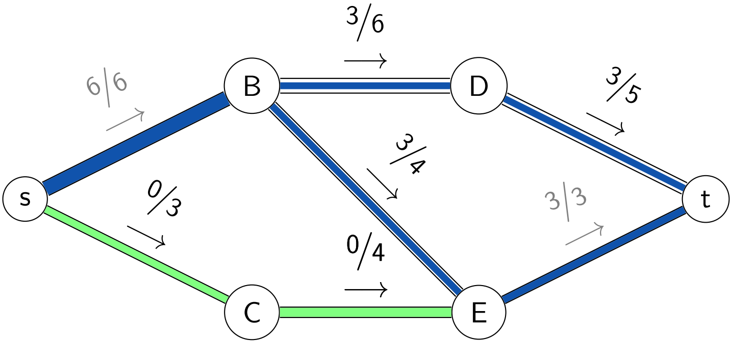
Flow Problems

Ford-Fulkerson: Example Run



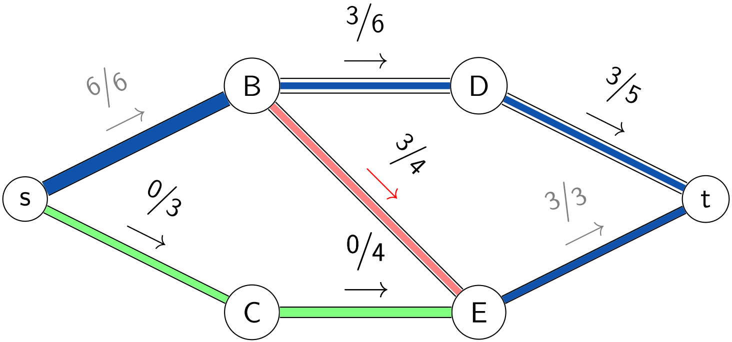
Flow Problems

Ford-Fulkerson: Example Run



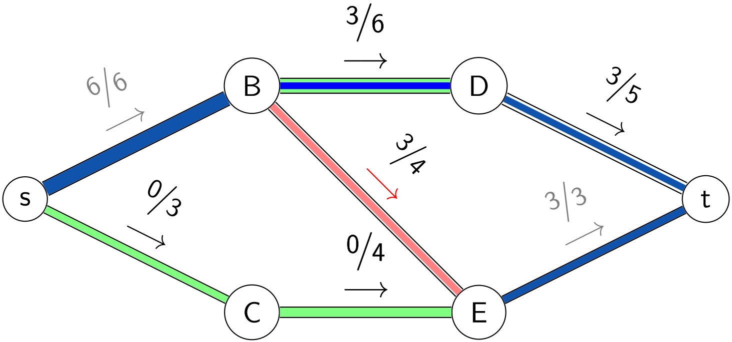
Flow Problems

Ford-Fulkerson: Example Run



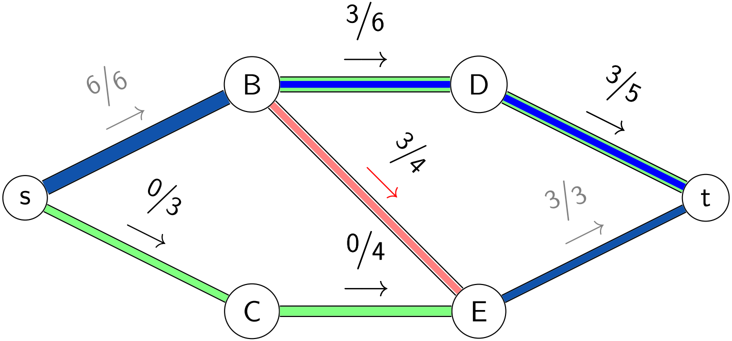
Flow Problems

Ford-Fulkerson: Example Run



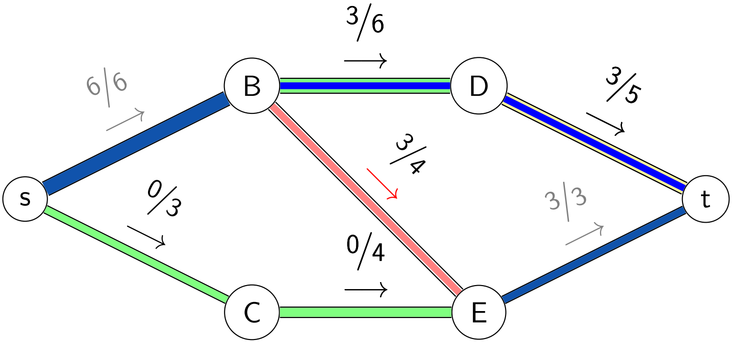
Flow Problems

Ford-Fulkerson: Example Run



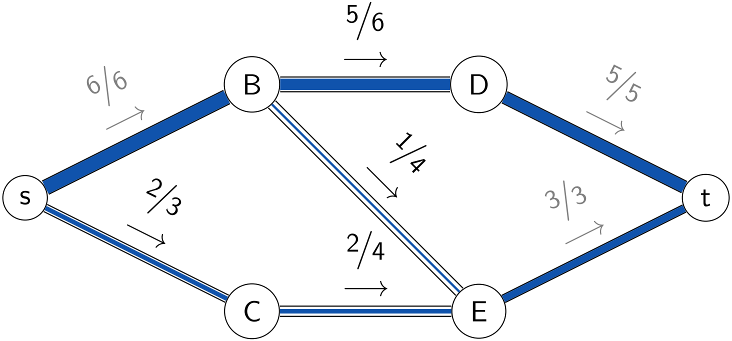
Flow Problems

Ford-Fulkerson: Example Run



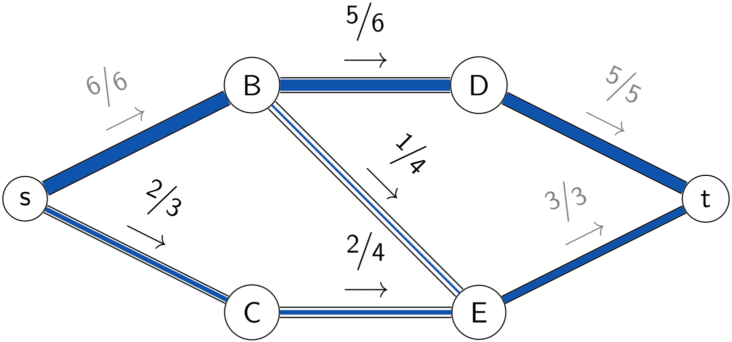
Flow Problems

Ford-Fulkerson: Example Run



Flow Problems

Ford-Fulkerson: Example Run

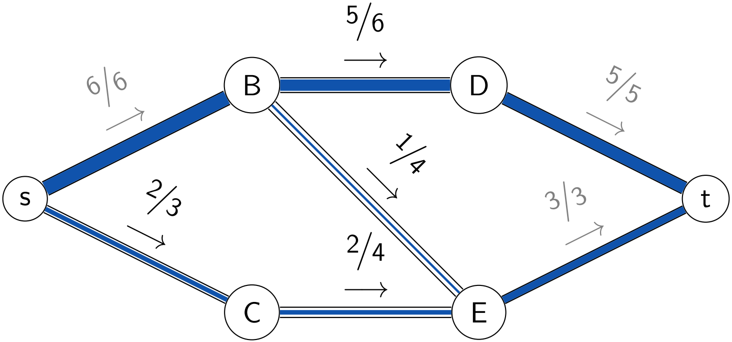


Result

Since no water gets lost in the system, the maximum flow can be read at s and t.

Flow Problems

Ford-Fulkerson: Example Run

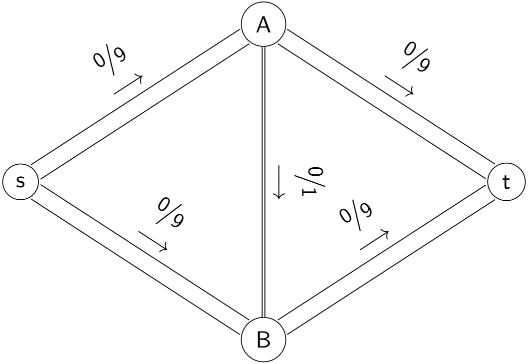


Result

Since no water gets lost in the system, the maximum flow can be read at s and t. The Max Flow value is $6 + 2 = 5 + 3 = 8$.

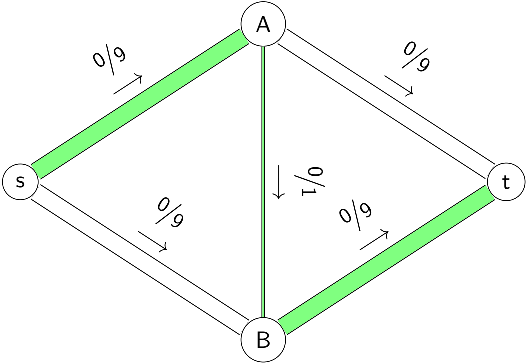
Flow Problems

Ford-Fulkerson: Analysis



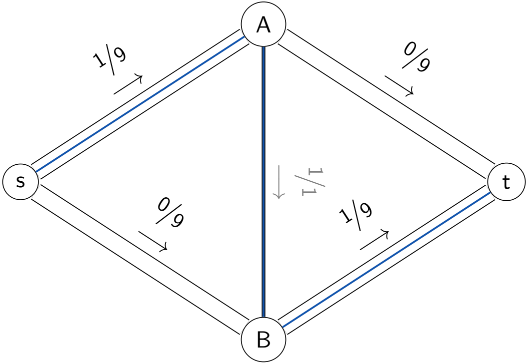
Flow Problems

Ford-Fulkerson: Analysis



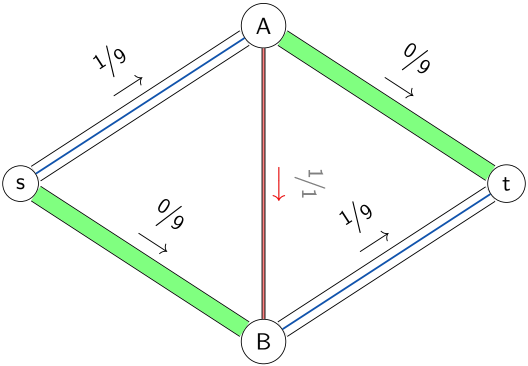
Flow Problems

Ford-Fulkerson: Analysis



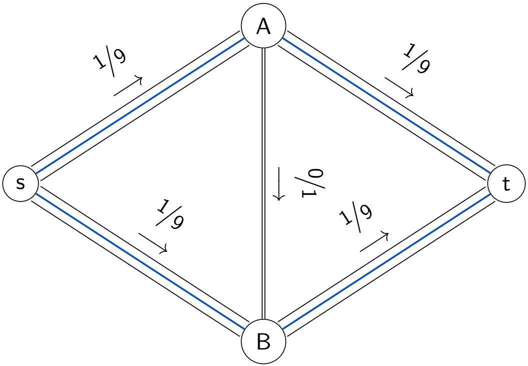
Flow Problems

Ford-Fulkerson: Analysis



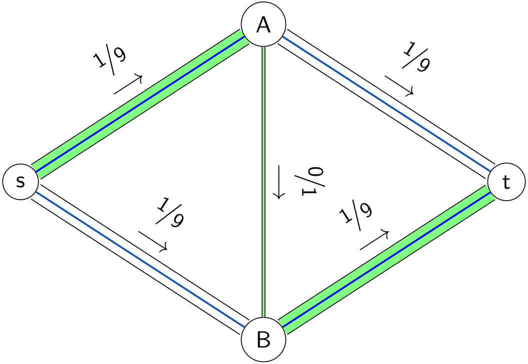
Flow Problems

Ford-Fulkerson: Analysis



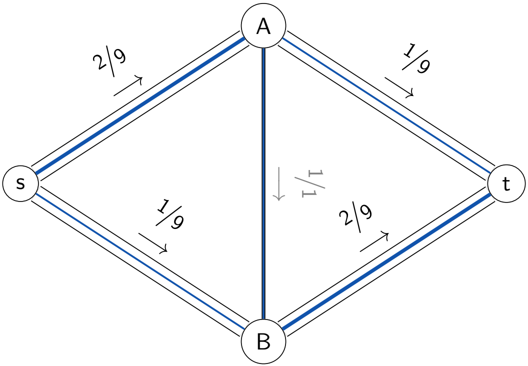
Flow Problems

Ford-Fulkerson: Analysis



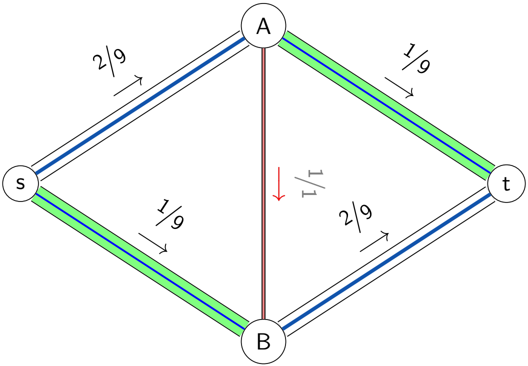
Flow Problems

Ford-Fulkerson: Analysis



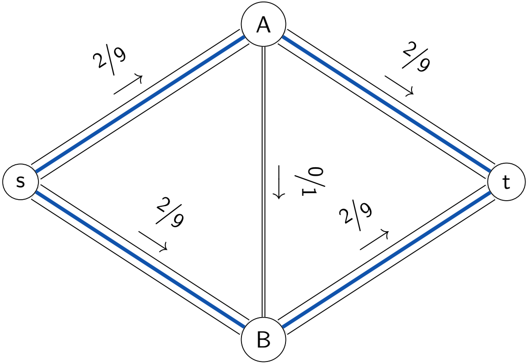
Flow Problems

Ford-Fulkerson: Analysis



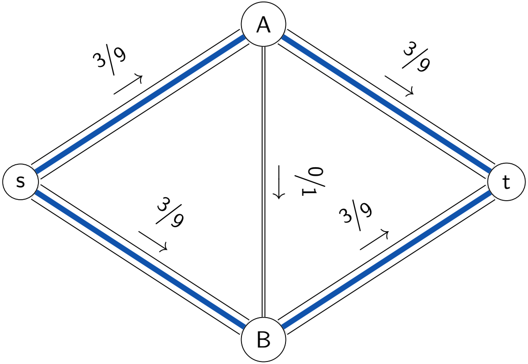
Flow Problems

Ford-Fulkerson: Analysis



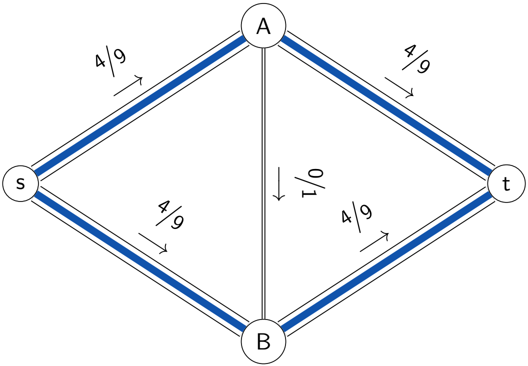
Flow Problems

Ford-Fulkerson: Analysis

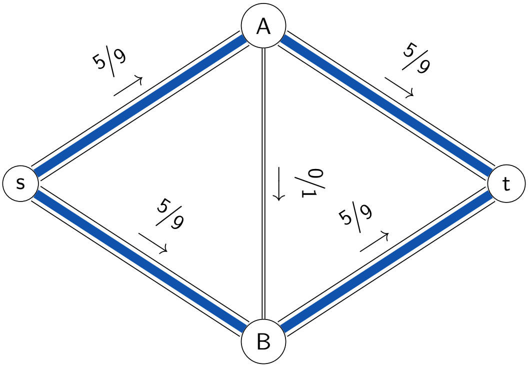


Flow Problems

Ford-Fulkerson: Analysis

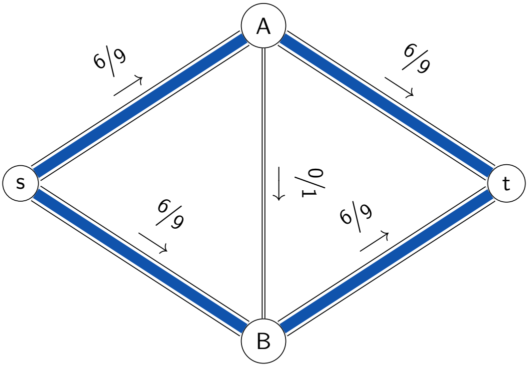


Flow Problems
Ford-Fulkerson: Analysis



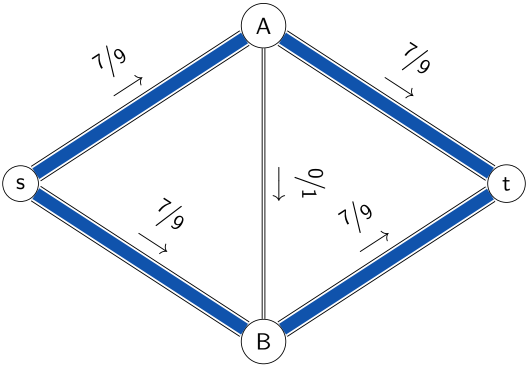
Flow Problems

Ford-Fulkerson: Analysis



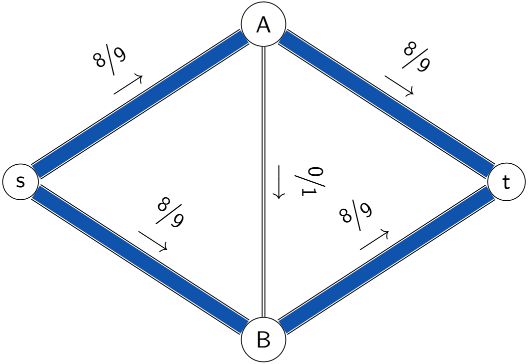
Flow Problems

Ford-Fulkerson: Analysis



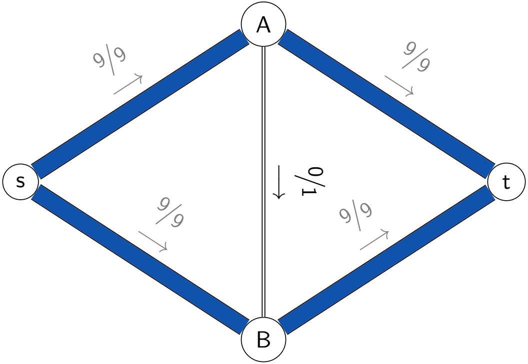
Flow Problems

Ford-Fulkerson: Analysis



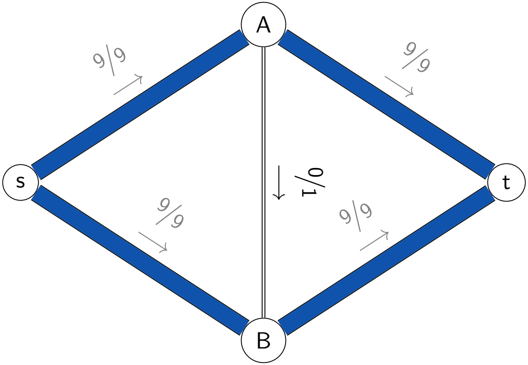
Flow Problems

Ford-Fulkerson: Analysis



Flow Problems

Ford-Fulkerson: Analysis



Analysis of Ford-Fulkerson

FF runs in $\mathcal{O}(\mathcal{F}^* \cdot E)$ where $\mathcal{F}^*$ is the maximum flow
$\longrightarrow$ Find a better way of finding Augmenting Paths

Insight: Ford-Fulkerson

Picking arbitrary Augmenting Paths can lead to $\mathcal{F}^*$ many iterations.

Insight: Ford-Fulkerson

Picking arbitrary Augmenting Paths can lead to $\mathcal{F}^*$ many iterations.

Insight (Edmonds & Karp)

Always pick the shortest Augmenting Path – in terms of hops.

Flow Problems

Edmonds-Karp

Insight: Ford-Fulkerson

Picking arbitrary Augmenting Paths can lead to $\mathcal{F}^*$ many iterations.

Insight (Edmonds & Karp)

Always pick the shortest Augmenting Path – in terms of hops.

$\longrightarrow$ use BFS

Insight: Ford-Fulkerson

Picking arbitrary Augmenting Paths can lead to $\mathcal{F}^*$ many iterations.

Insight (Edmonds & Karp)

Always pick the shortest Augmenting Path – in terms of hops.
$\longrightarrow$ use BFS

They showed that, after $\mathcal{O}(VE)$ iterations, no Augmenting Paths exist.

Flow Problems

Edmonds-Karp

Insight: Ford-Fulkerson

Picking arbitrary Augmenting Paths can lead to $\mathcal{F}^*$ many iterations.

Insight (Edmonds & Karp)

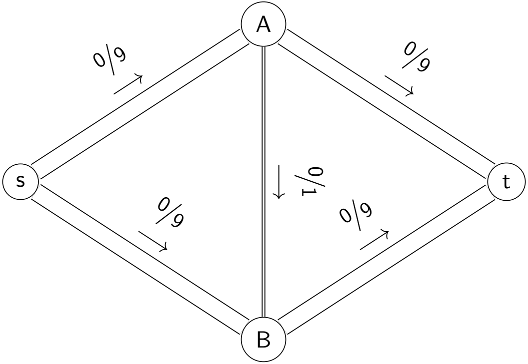
Always pick the shortest Augmenting Path – in terms of hops.
$\longrightarrow$ use BFS

They showed that, after $\mathcal{O}(VE)$ iterations, no Augmenting Paths exist.

Each BFS runs in $\mathcal{O}(E)$, totalling to $\mathcal{O}(VE^2)$.

Flow Problems

Edmonds & Karp's Algorithm

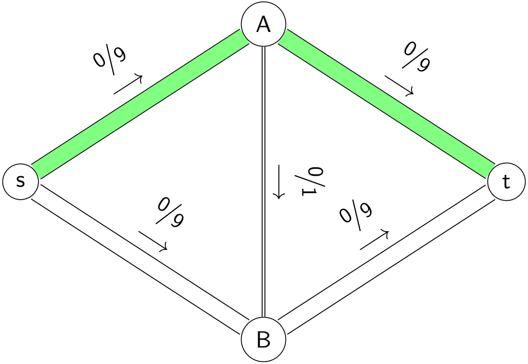


Insight (Edmonds & Karp)

Pick the shortest Augmenting Paths – in terms of hops.

Flow Problems

Edmonds & Karp's Algorithm

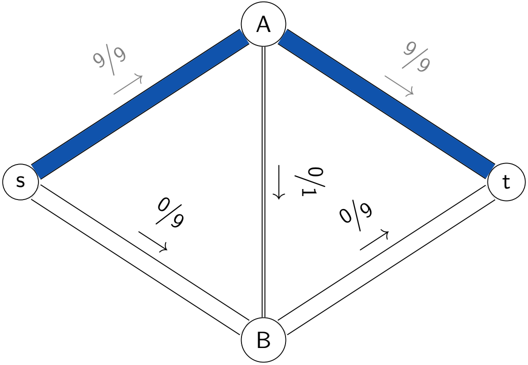


Insight (Edmonds & Karp)

Pick the shortest Augmenting Paths – in terms of hops.

Flow Problems

Edmonds & Karp's Algorithm

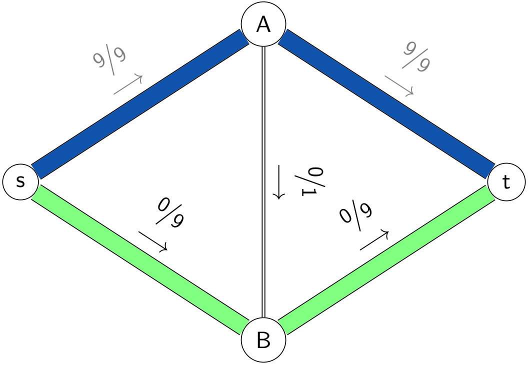


Insight (Edmonds & Karp)

Pick the shortest Augmenting Paths – in terms of hops.

Flow Problems

Edmonds & Karp's Algorithm

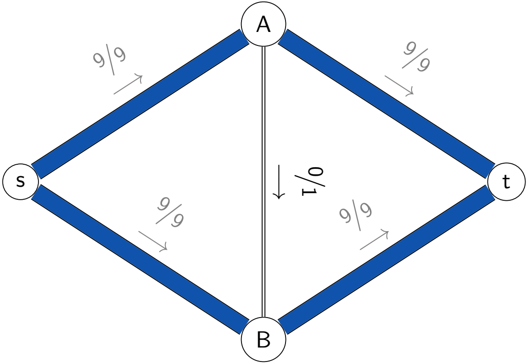


Insight (Edmonds & Karp)

Pick the shortest Augmenting Paths – in terms of hops.

Flow Problems

Edmonds & Karp's Algorithm



Insight (Edmonds & Karp)

Pick the shortest Augmenting Paths – in terms of hops.

Flow Problems

Edmonds & Karp's Algorithm

```
vector<vector<int>> capacity; // adjacency matrix
vector<vector<int>> adj;      // adjacency list
vector<int> parent;           // used in BFS

int main() {
  int s, t, V, E;
  cin >> V >> E >> s >> t;
  adj.resize(V);
  capacity.assign(V, vector<int>(V, 0));
  int u, v, c;
  for (int i = 0; i < E; i++) {
    cin >> u >> v >> c;
    adj[u].push_back(v);
    adj[v].push_back(u);
    capacity[u][v] += c; // forward capacity
  }
  int out = maxflow(s, t);
  cout << out << endl;
}
```

- Capacity could be an adjacency list
  $\longrightarrow$ Code will be in CMS

Flow Problems

Edmonds & Karp's Algorithm

```
int maxflow(int s, int t) {
  int totalflow = 0, u;
  while (true) {
    bfs(s); // build bfs tree
    if (parent[t] == -1) break; // unreachable
    int bottleneck = INF;
    u = t; // find bottleneck capacity
    while (u != s) {
      int v = parent[u];
      bottleneck = min(bottleneck, capacity[v][u]);
      u = v;
    }
    u = t; // update capacities along path
    while (u != s) {
      int v = parent[u];
      capacity[v][u] -= bottleneck;
      capacity[u][v] += bottleneck;
      u = v;
    }
    totalflow += bottleneck;
  }
  return totalflow;
}
```

Flow Problems

Edmonds & Karp's Algorithm

```cpp
void bfs(int s) {
  parent.assign(adj.size(), -1);
  parent[s] = -2; // s is visited

  queue<int> Q;
  Q.push(s);
  while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int v : adj[u])
      if (parent[v] == -1 and capacity[u][v] > 0) {
        Q.push(v);
        parent[v] = u;
      }
  }
}
```

- Almost plain BFS. Tracks parent pointers and checks if capacity is nonzero

Flow Problems

Recap

Maximum Flow Algorithms

Algorithm	Concept	Running Time
Ford-Fulkerson	Augmenting Paths	$\mathcal{O}(\mathcal{F}^* E)$
Edmonds-Karp	Augmenting Paths	$\mathcal{O}(VE^2)$
Dinic	Blocking Flow	$\mathcal{O}(V^2 E)$

Dinic's Algorithm is worth a glance and useful to have at hand, but Edmonds-Karp's is usually sufficient for competitive programming.

Modeling Flow Problems

The Golden Hammer of Network Flow

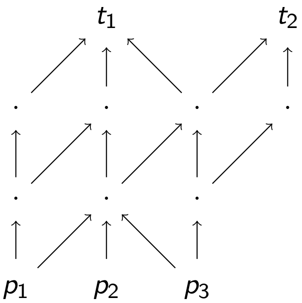
Once you know Edmonds-Karp's algorithm, every problem starts to look like a Max Flow instance.

Modeling Flow Problems

Hiking

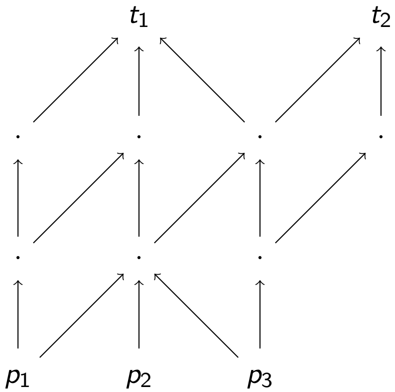
Problem: Hiking

You are a big fan of hiking. Your favorite hike starts any of the parking lots p_1, p_2 or p_3 and ends at either one of the tops t_1 or t_2. Sadly, you get bored very easily. How many different hikes starting at any parking lot can you find such that you visit no path segment twice?



Modeling Flow Problems

Hiking: Modeling

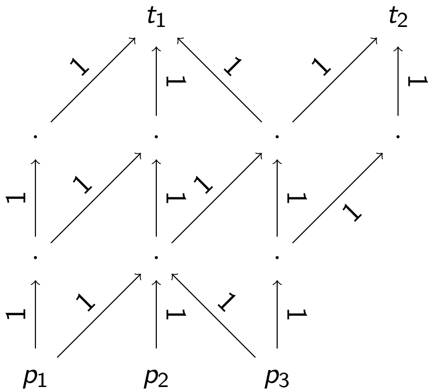


Hiking: Idea

See Hiking as a Max Flow instance

Modeling Flow Problems

Hiking: Modeling



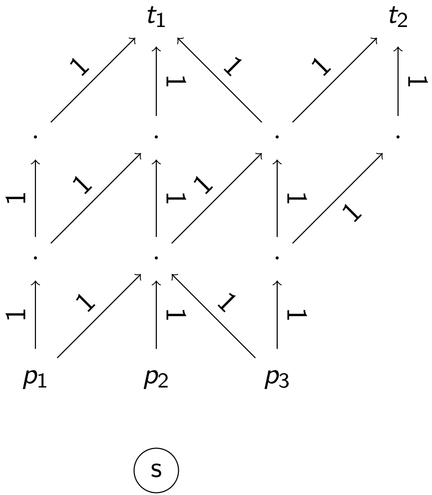
Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities

Modeling Flow Problems

Hiking: Modeling



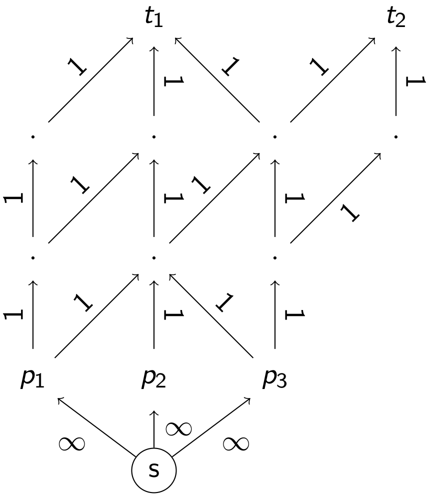
Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities

Modeling Flow Problems

Hiking: Modeling



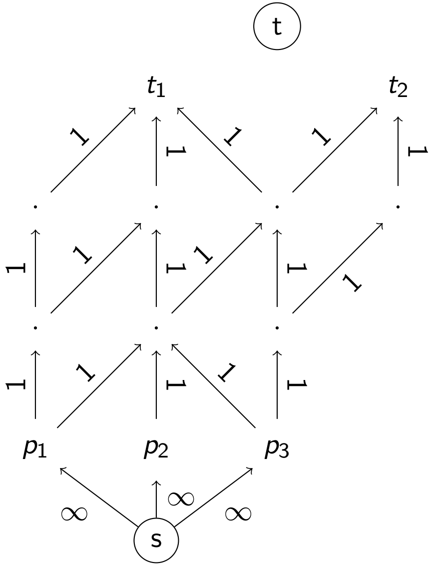
Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i

Modeling Flow Problems

Hiking: Modeling



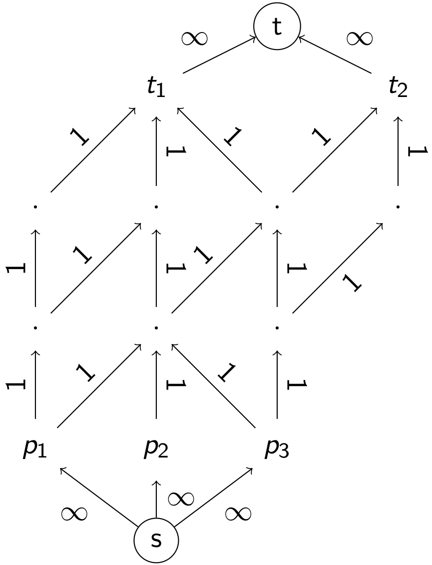
Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i

Modeling Flow Problems

Hiking: Modeling



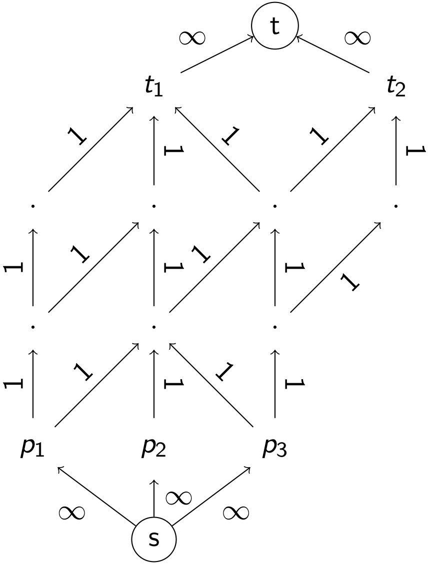
Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

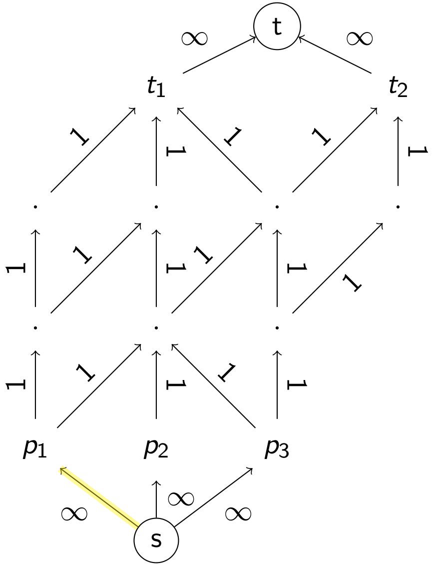
See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

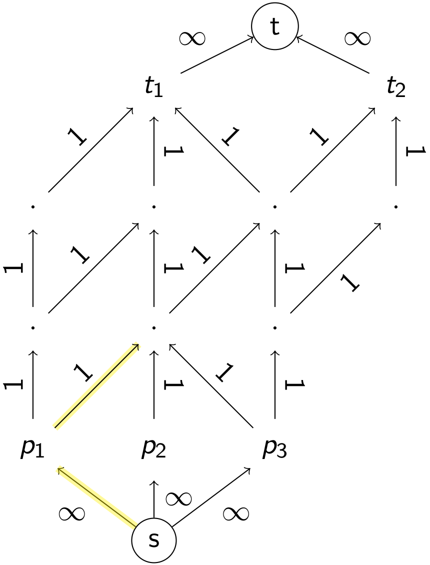
See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_j

Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

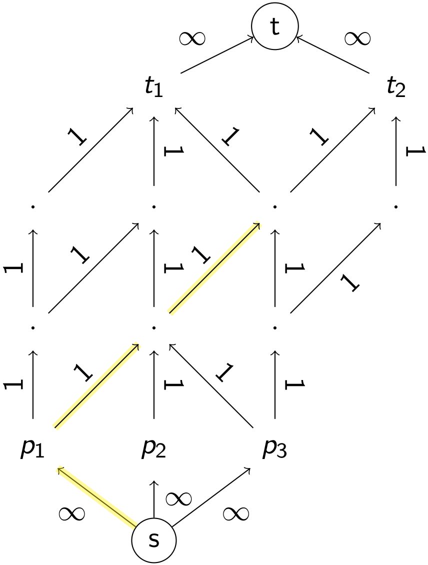
See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

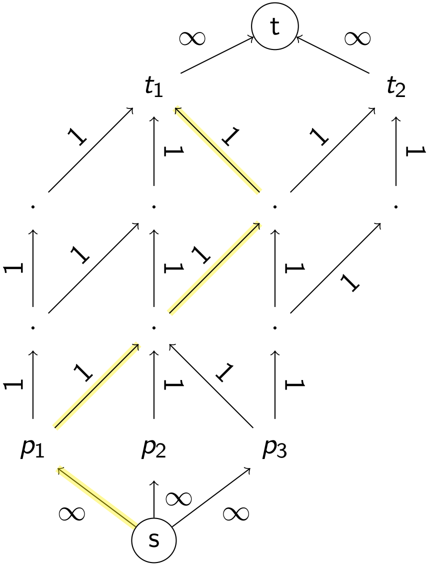
See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

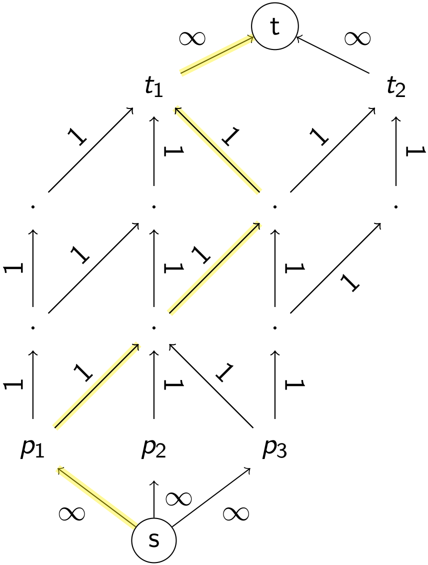
See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

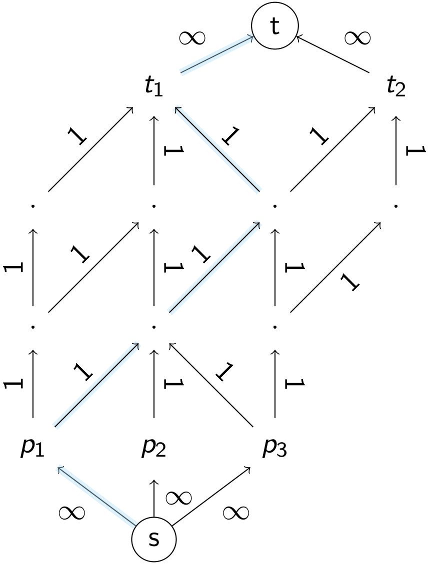
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

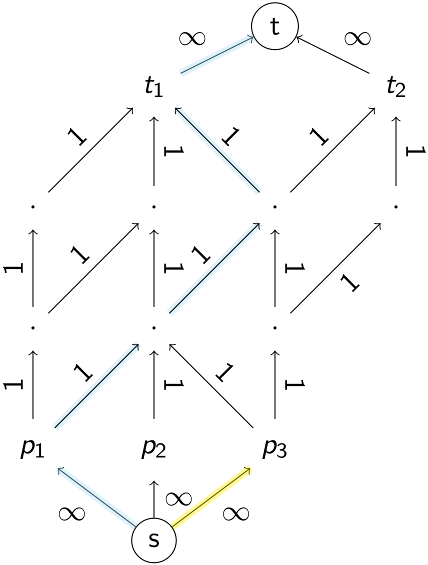
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

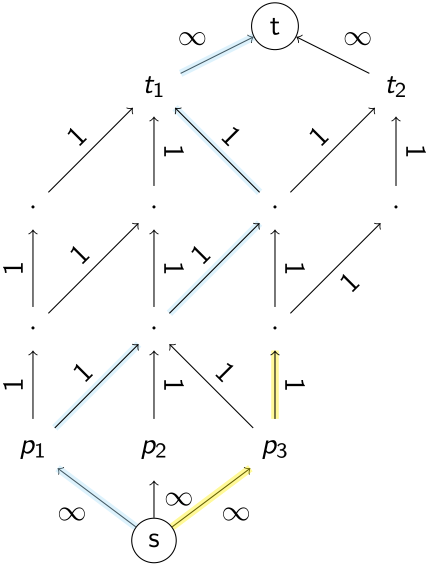
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

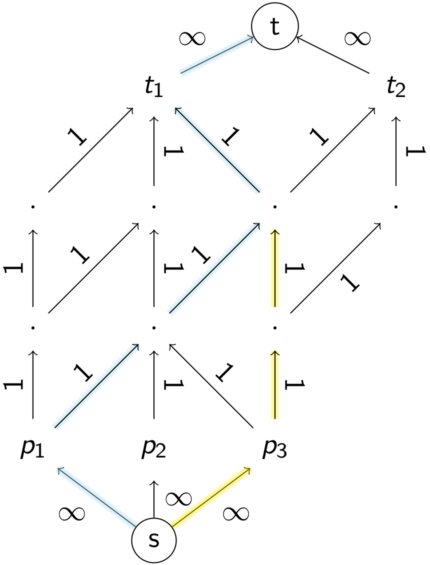
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

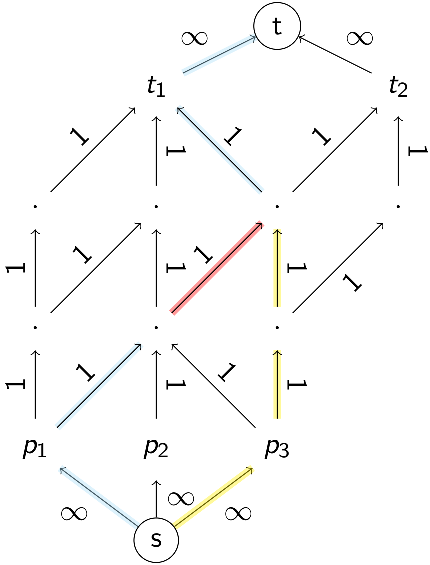
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

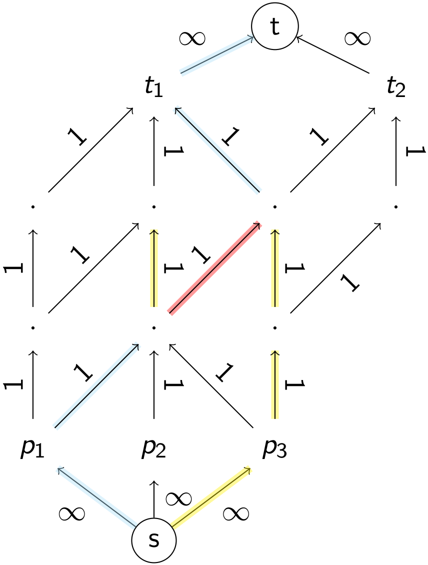
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

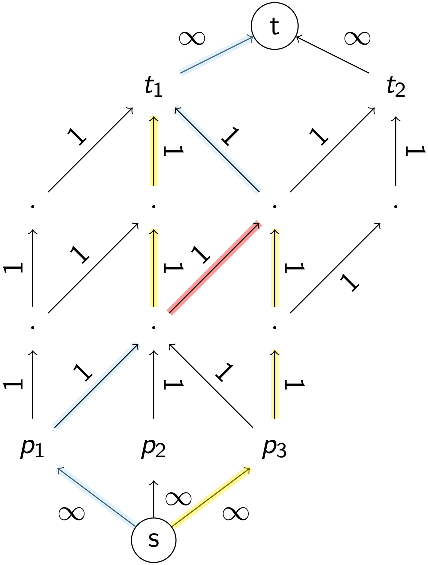
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

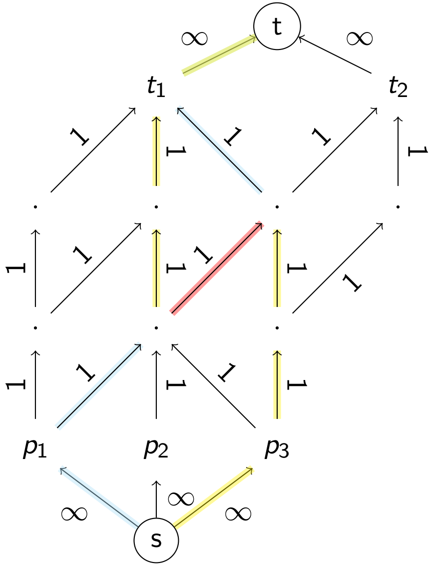
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

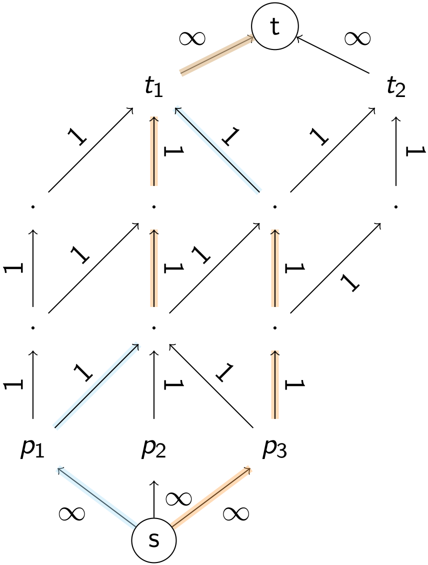
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

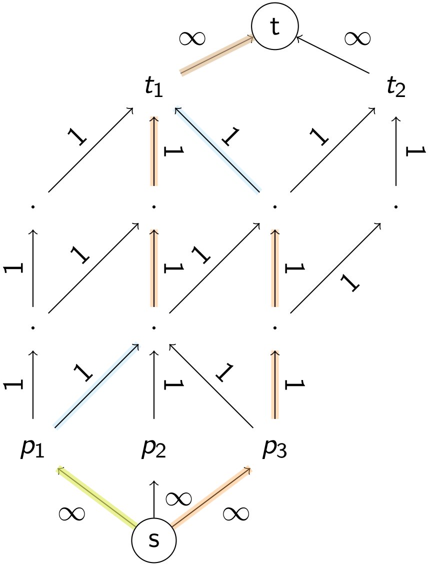
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

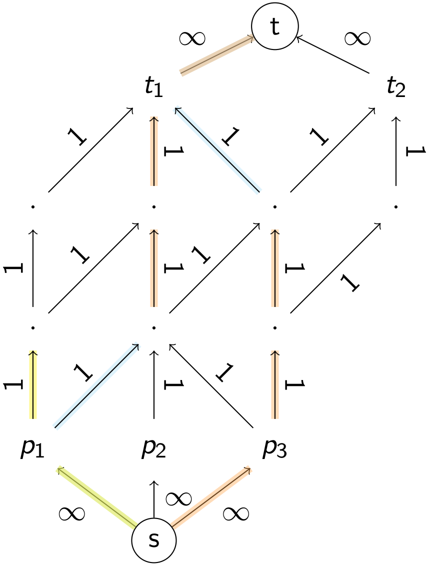
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

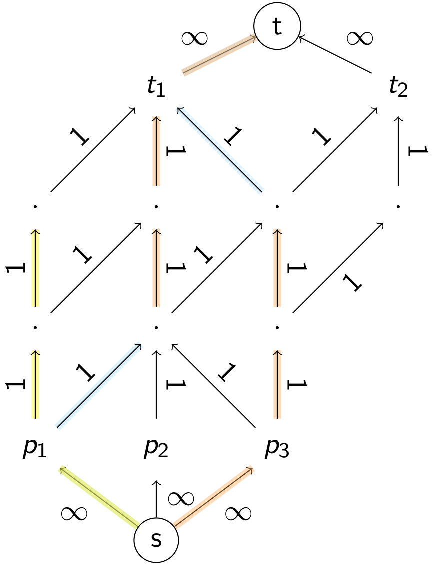
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

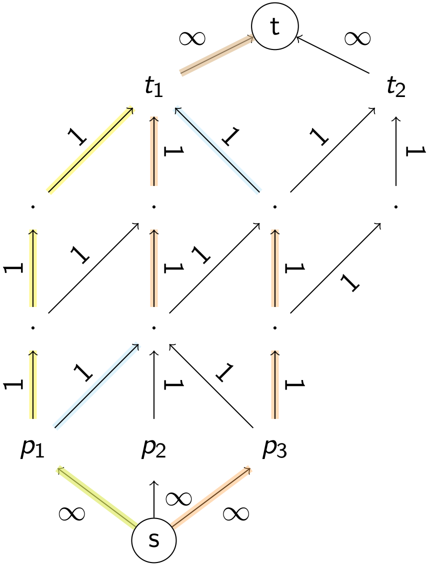
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

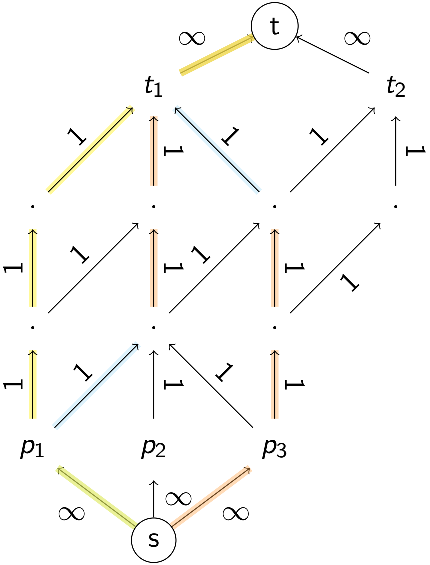
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

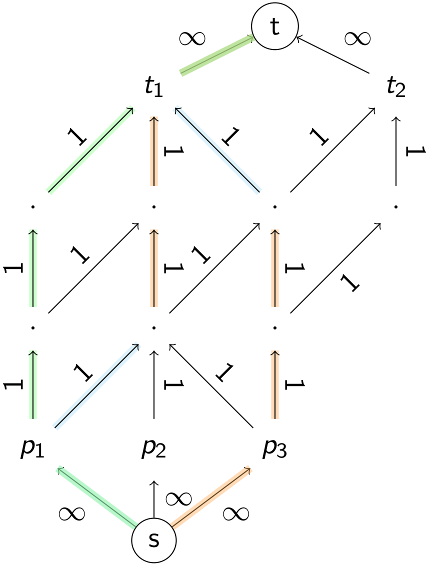
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

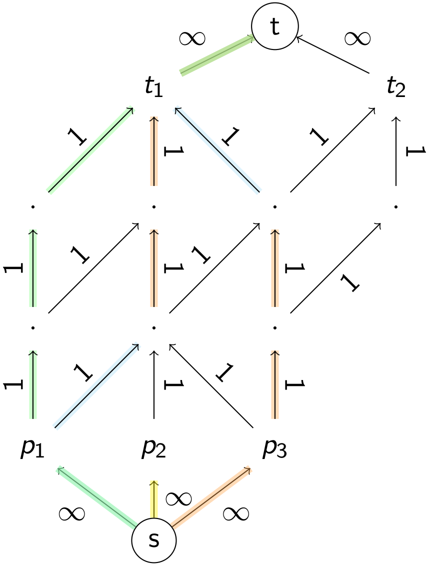
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

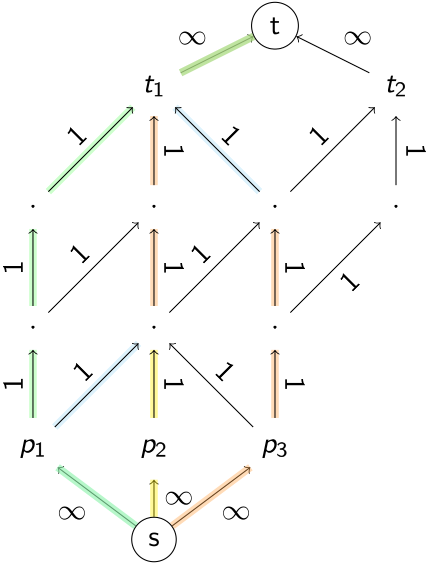
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

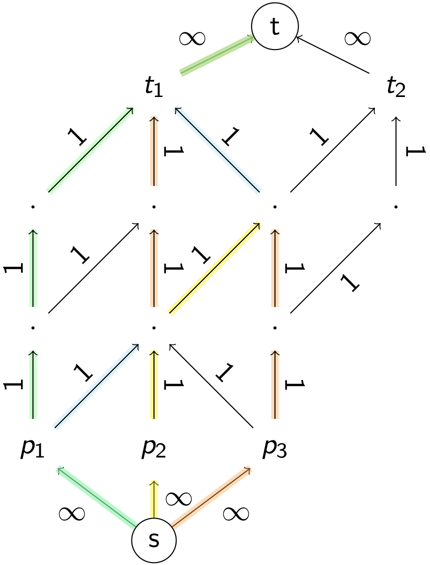
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_j

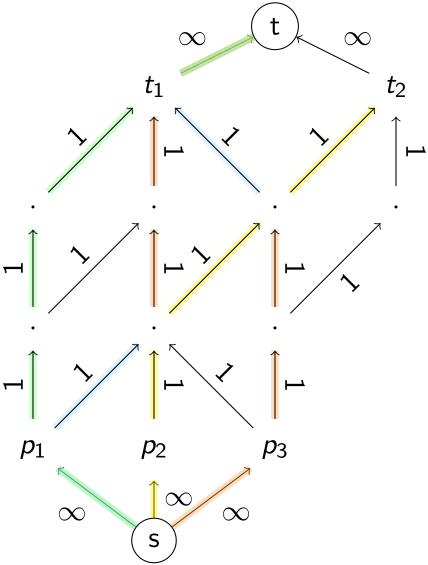
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

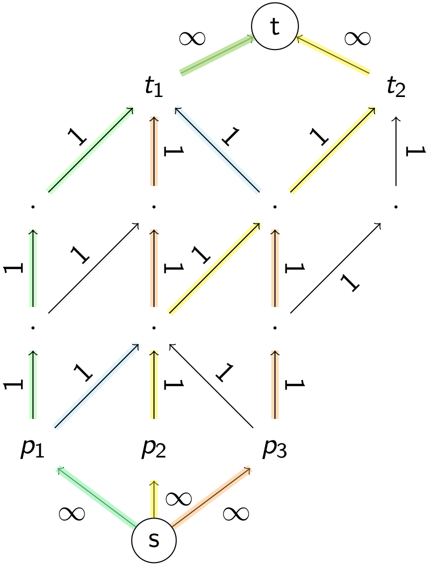
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_i

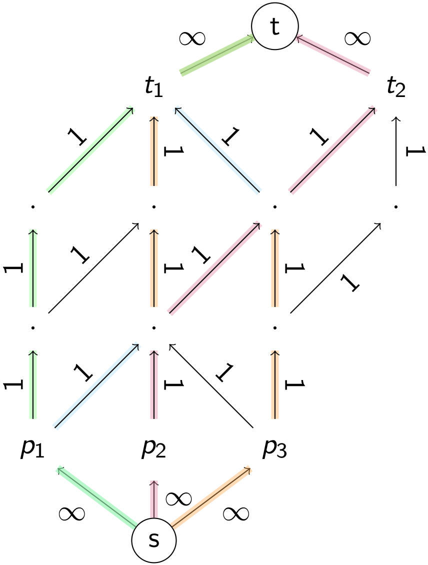
Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

There are four edge-disjoint paths from the parking lots to the summits.

Modeling Flow Problems

Hiking: Modeling



Hiking: Idea

See Hiking as a Max Flow instance

- Edges have unit capacities
- Add super source, connect to p_i
- Add super sink, connect to s_j

Finally, MaxFlow(s, t) computes the number of edge-disjoint paths between p_i to s_j

Hiking: Solution

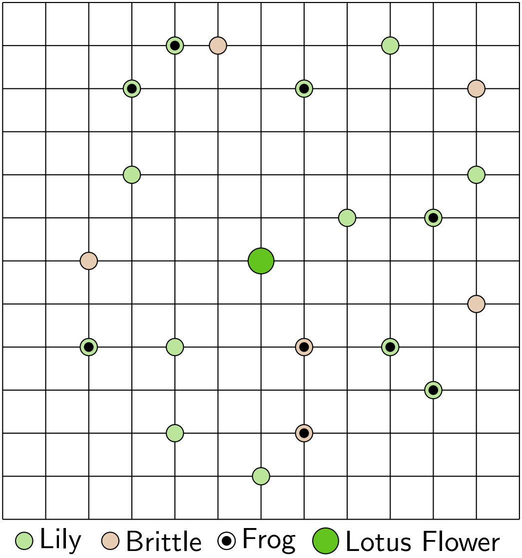
There are four edge-disjoint paths from the parking lots to the summits.

Problem: Lily Pads

A bunch of frogs are sitting on lily pads. They want to jump from leaf to leaf in order to reach a big lotus flower in the center (0,0). Each frog can jump a distance of up to three units. However, some leaves are brittle and can only be jumped off once. Given the coordinates of the frogs and lily pads, how many frogs can meet at the flower?

Modeling Flow Problems

Lily Pads: Modeling



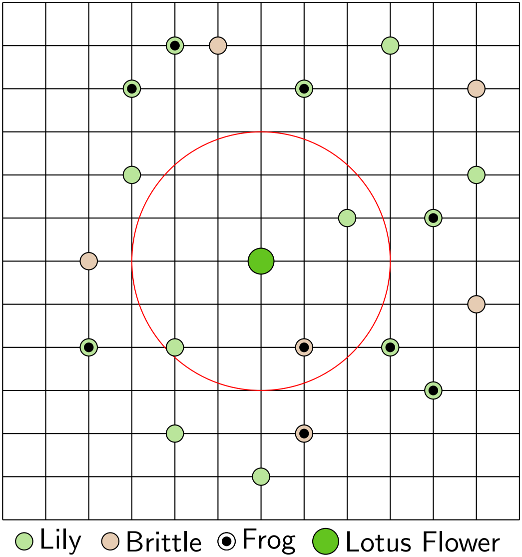
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



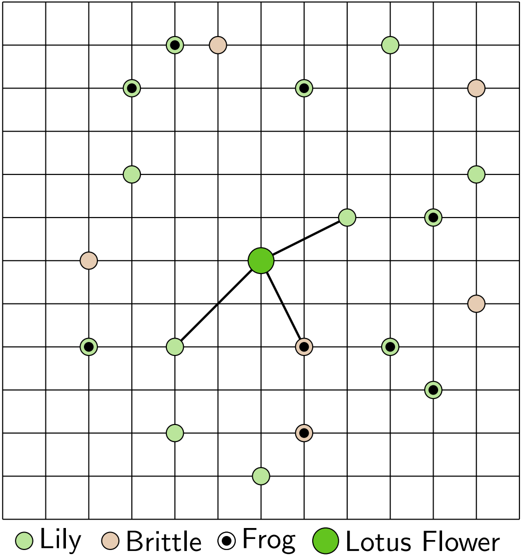
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



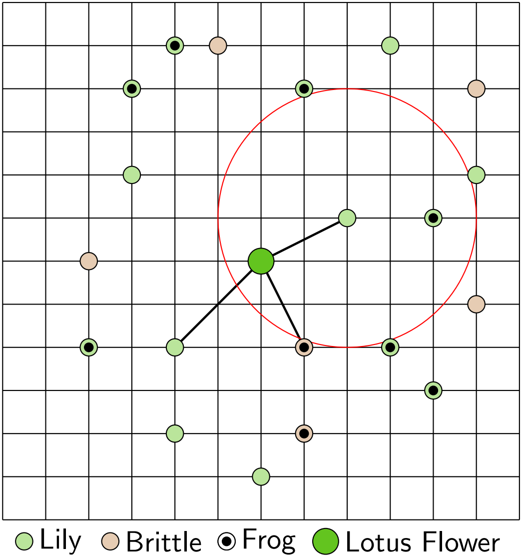
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



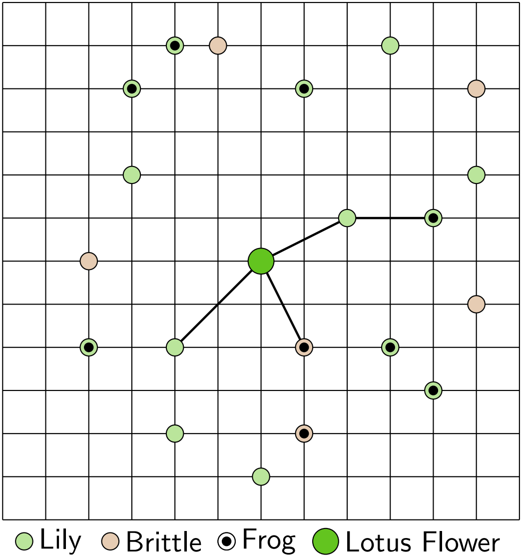
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



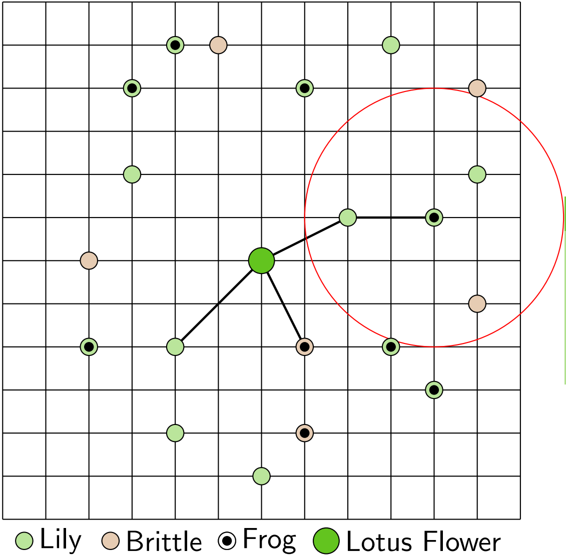
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



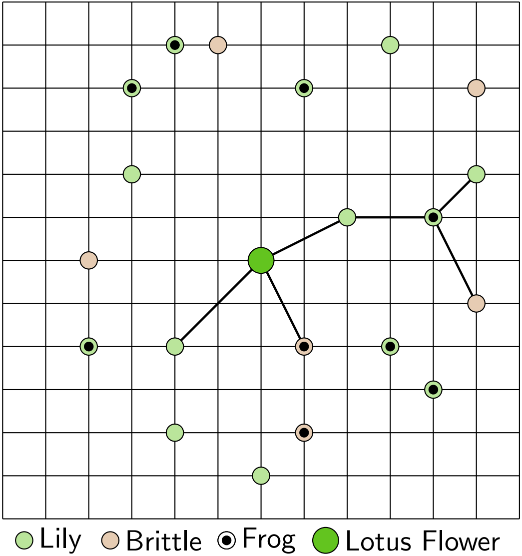
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



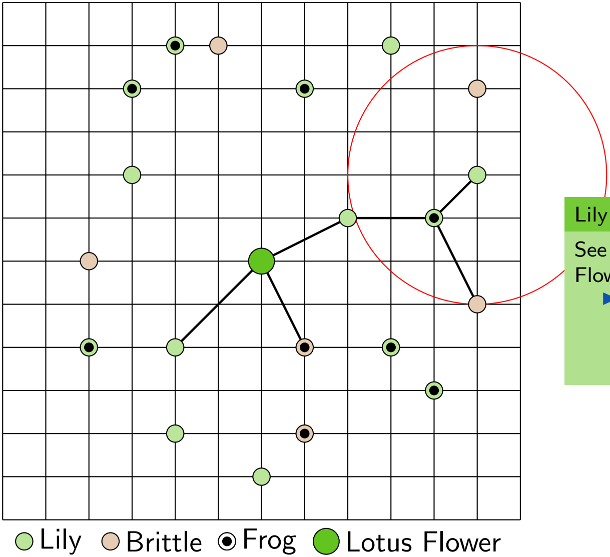
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



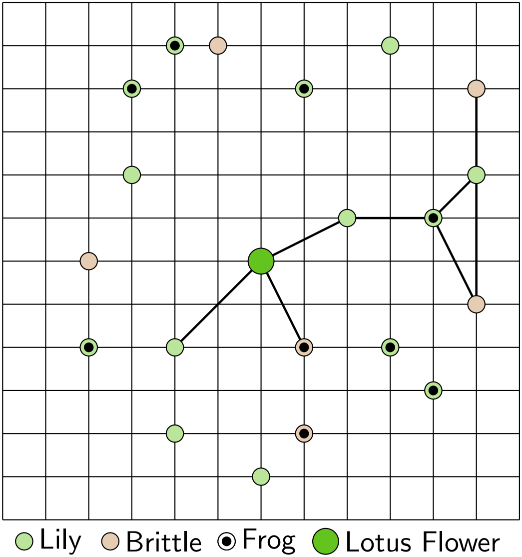
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



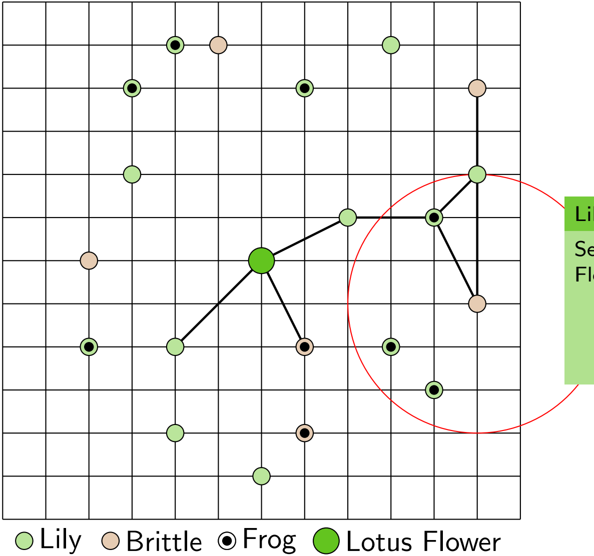
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



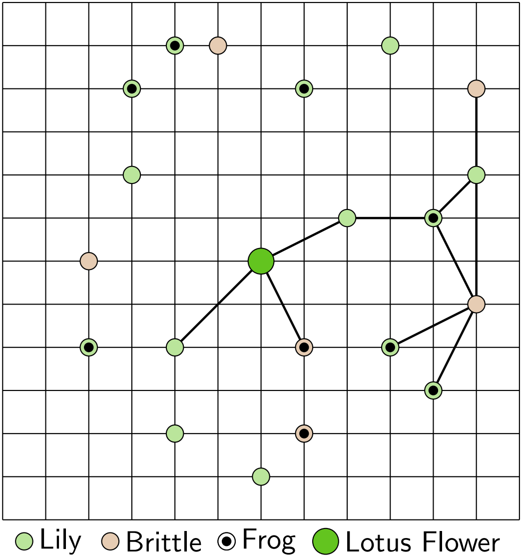
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



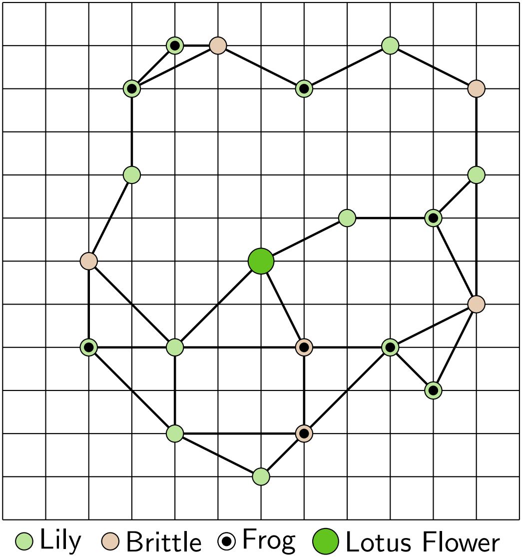
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



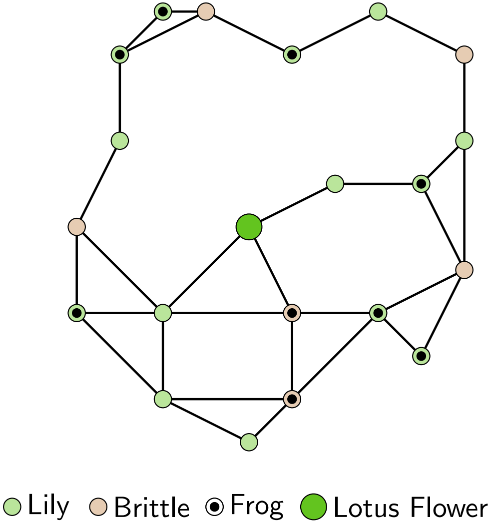
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



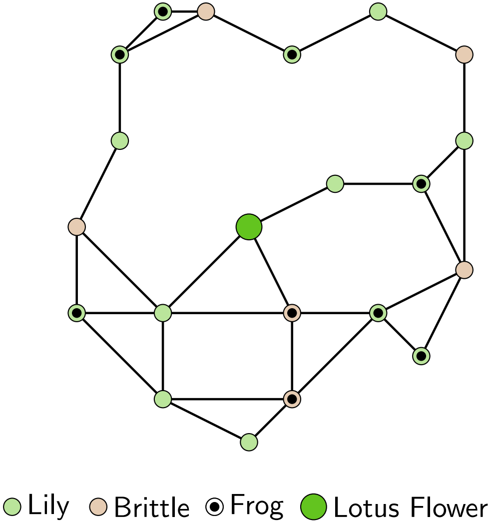
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph

Modeling Flow Problems

Lily Pads: Modeling



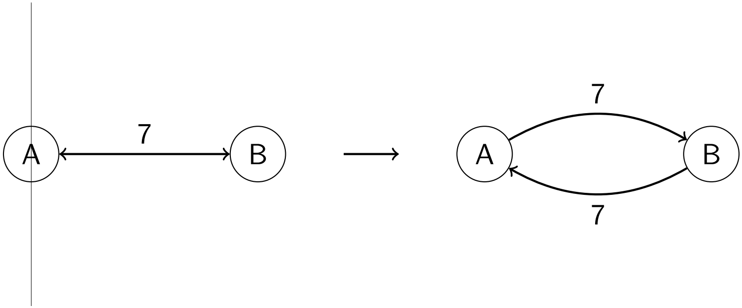
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞

Modeling Flow Problems

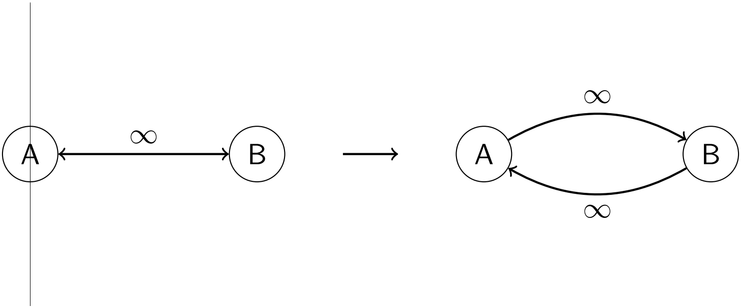
Modeling Undirected Edges




```
int main() {
  // setting up capacities
    capacity[u][v] = 7; // forward capacity
    capacity[v][u] = 7; // backward capacity
  // ...
}
```

Modeling Flow Problems

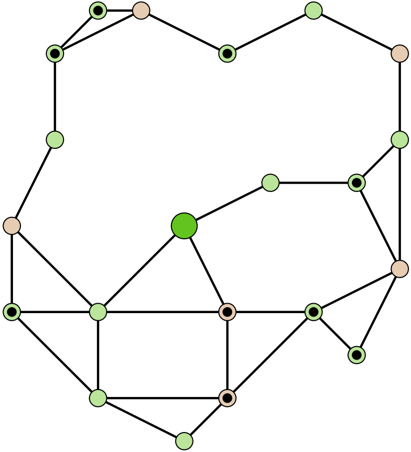
Modeling Undirected Edges




```
int main() {
  // setting up capacities
    capacity[u][v] = INF / 2; // prevent overflow when
    capacity[v][u] = INF / 2; //   updating flow
  // ...
}
```

Modeling Flow Problems

Lily Pads: Modeling



Lily Brittle Frog Lotus Flower

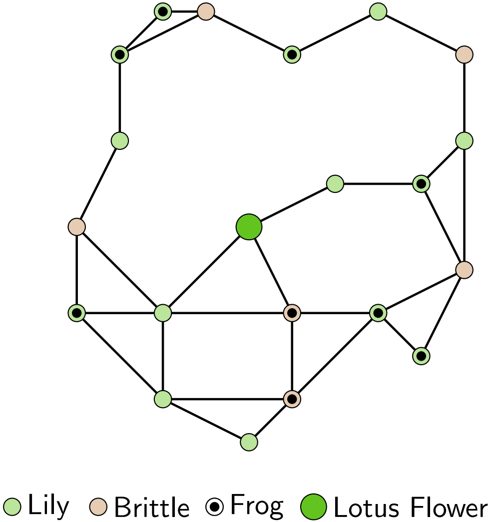
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞

Modeling Flow Problems

Lily Pads: Modeling



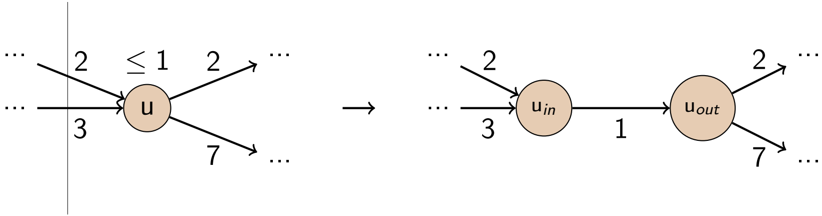
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities

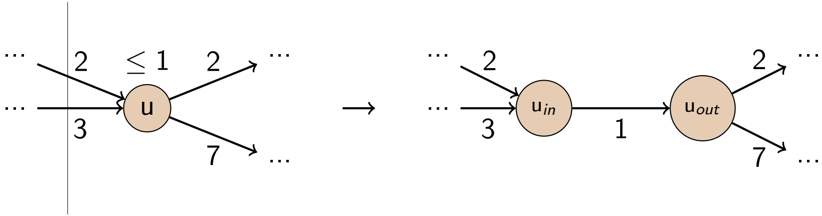
Modeling Flow Problems

Modeling Vertex Capacities



Modeling Flow Problems

Modeling Vertex Capacities

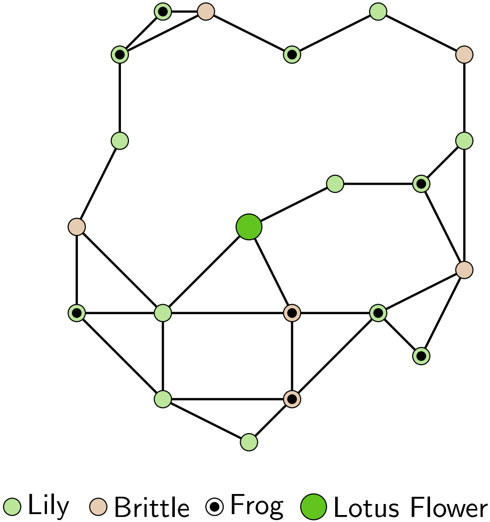



```
vector<vector<int>> adj(2*V);
int u, v;
// access to duplicate nodes
int u_in = u; int u_out = u + V;
int v_in = v; int v_out = v + V;
// edge u -> v
adj[u_out].push_back(v_in);
adj[v_in].push_back(u_out);

// vertex capacity
capacity[u_in][u_out] = 1;
capacity[u_out][u_in] = 0;
```

Modeling Flow Problems

Lily Pads: Modeling



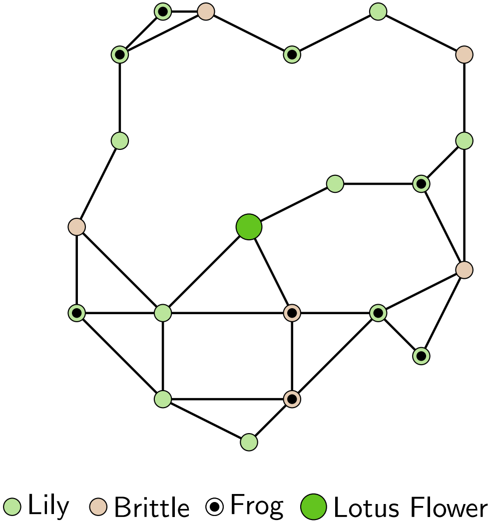
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities

Modeling Flow Problems

Lily Pads: Modeling



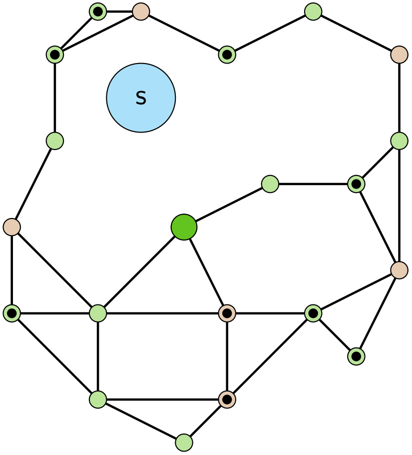
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities
- Add super source $\rightarrow$Capacity 1 per frog

Modeling Flow Problems

Lily Pads: Modeling



Lily Brittle Frog Lotus Flower

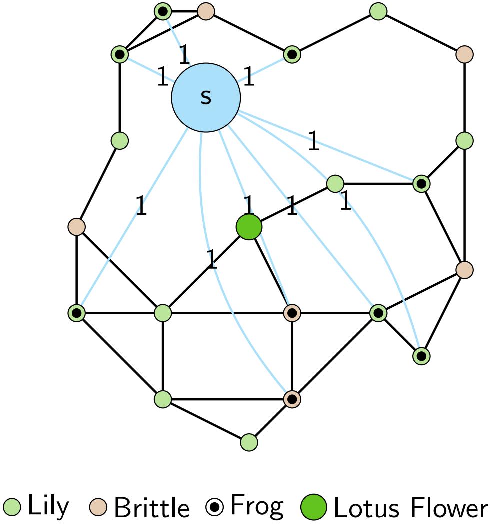
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves →Vertex Capacities
- Add super source →Capacity 1 per frog

Modeling Flow Problems

Lily Pads: Modeling



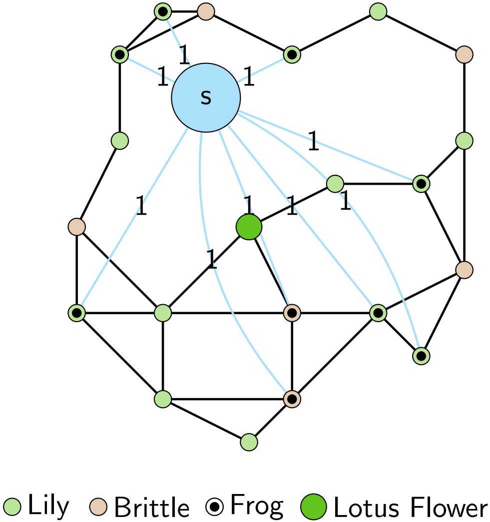
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities
- Add super source $\rightarrow$Capacity 1 per frog

Modeling Flow Problems

Lily Pads: Modeling



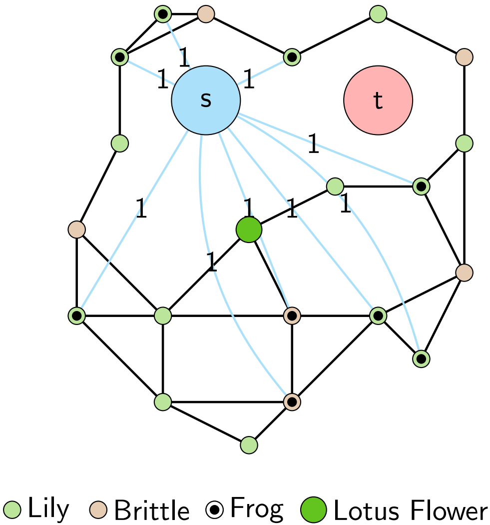
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities
- Add super source $\rightarrow$Capacity 1 per frog
- Add super sink $\rightarrow$Capacity ∞

Modeling Flow Problems

Lily Pads: Modeling



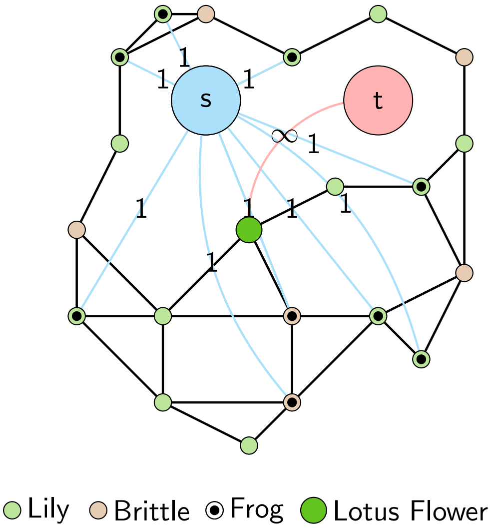
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities
- Add super source $\rightarrow$Capacity 1 per frog
- Add super sink $\rightarrow$Capacity ∞

Modeling Flow Problems

Lily Pads: Modeling



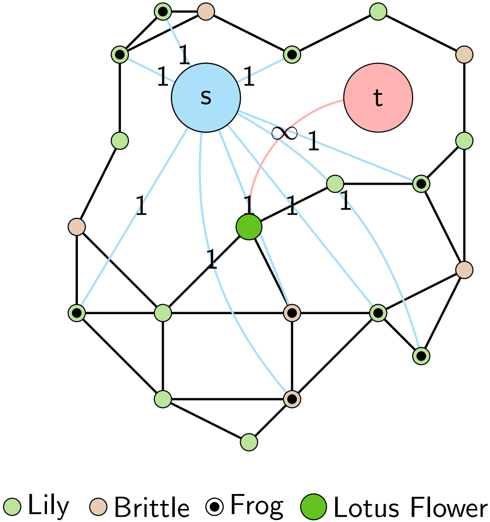
Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities
- Add super source $\rightarrow$Capacity 1 per frog
- Add super sink $\rightarrow$Capacity ∞

Modeling Flow Problems

Lily Pads: Modeling



Lily Pads: Idea

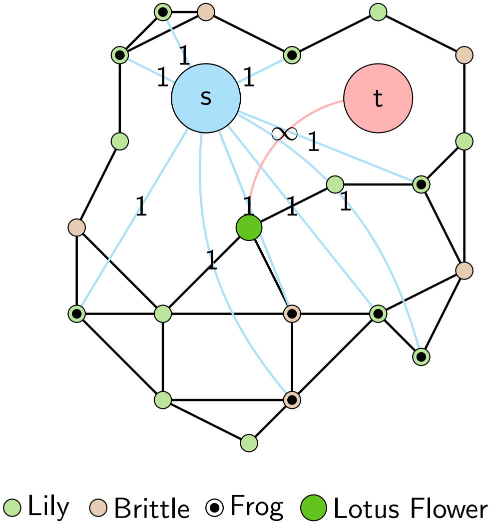
See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves
  $\rightarrow$Vertex Capacities
- Add super source
  $\rightarrow$Capacity 1
  per frog
- Add super sink
  $\rightarrow$Capacity ∞

Run MaxFlow(s, t)

Modeling Flow Problems

Lily Pads: Modeling



Lily Pads: Idea

See Lily Pads as a Max Flow instance

- Model Graph
- Undirected Edges of capacity ∞
- Brittle Leaves $\rightarrow$Vertex Capacities
- Add super source $\rightarrow$Capacity 1 per frog
- Add super sink $\rightarrow$Capacity ∞

Run MaxFlow(s, t)

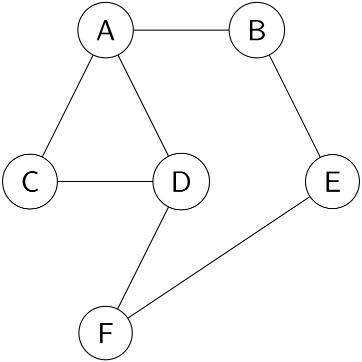
Lily Pads: Solution

At most seven frogs can make it to the lotus flower.

Matching

Matching

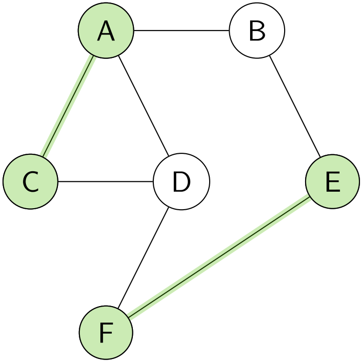
A matching of a graph G is a subset $M \subseteq E$ such that no two edges in M share a common vertex.



Matching

Matching

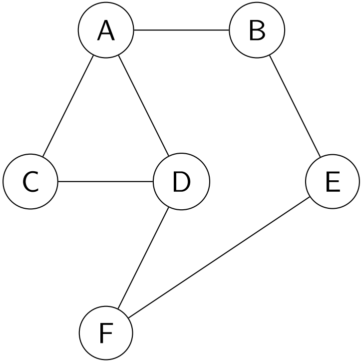
A matching of a graph G is a subset $M \subseteq E$ such that no two edges in M share a common vertex.



Matching

Maximum Cardinality Matching

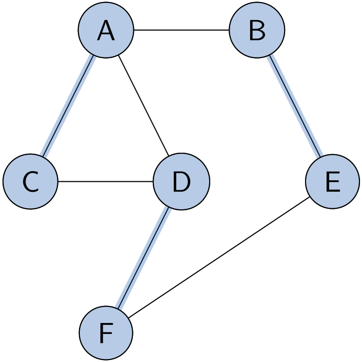
A matching M is called a Maximum Cardinality Matching if among all matchings M' of G, $|M| \geq |M'|$.



Matching

Maximum Cardinality Matching

A matching M is called a Maximum Cardinality Matching if among all matchings M' of G, $|M| \geq |M'|$.

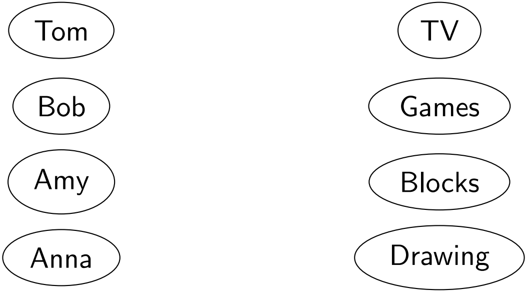


Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?

Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?

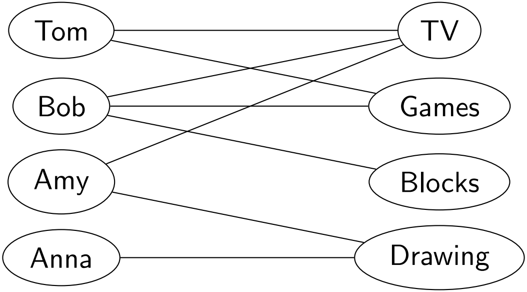


Matching

Bipartite Matching

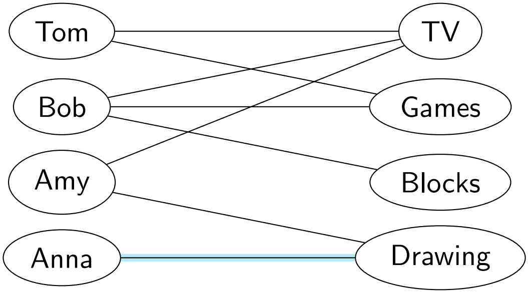
Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?



Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?

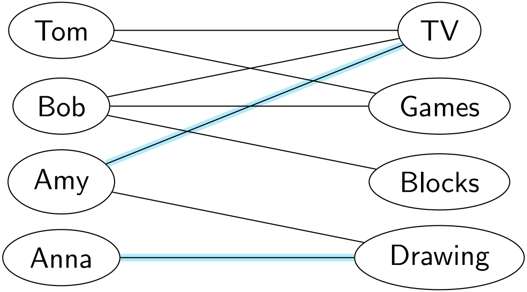


Matching

Bipartite Matching

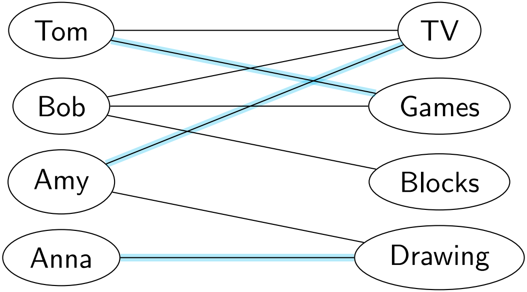
Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?



Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?

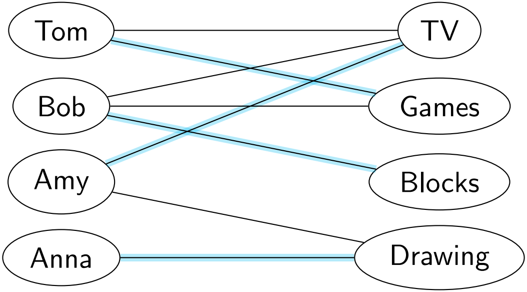


Matching

Bipartite Matching

Baby-Sitting

You are baby-sitting a bunch of kids. If you can, you try to keep them busy by themselves. How many kids can you have enjoy their favorite activities at the same time?

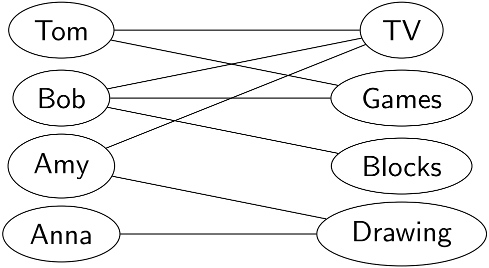


Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.

Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.

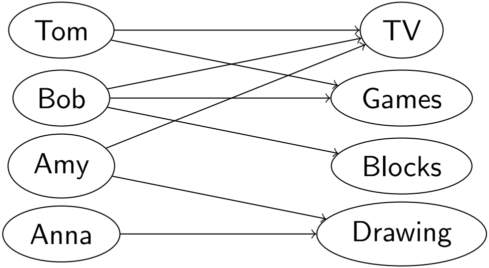


Matching

Bipartite Matching

Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.

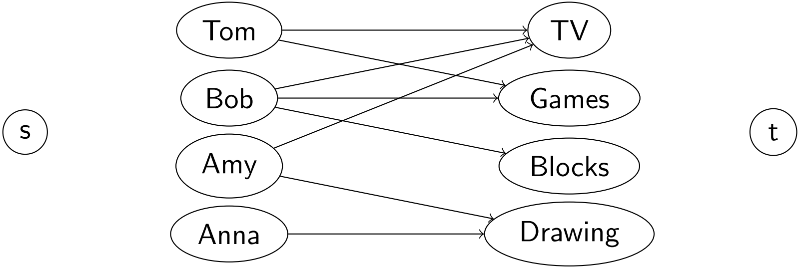


Matching

Bipartite Matching

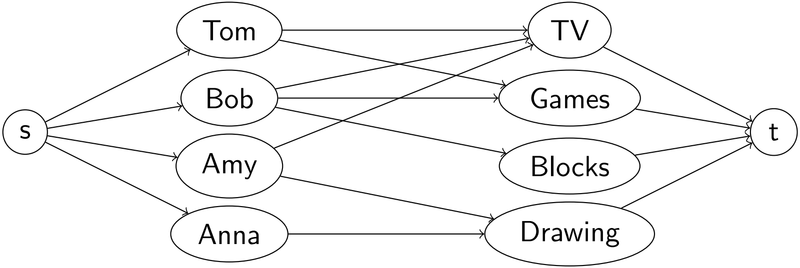
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.

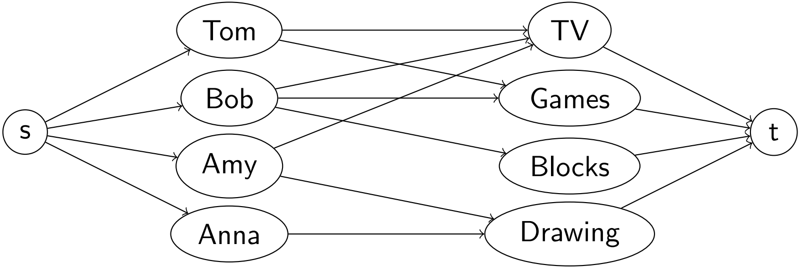


Matching

Bipartite Matching

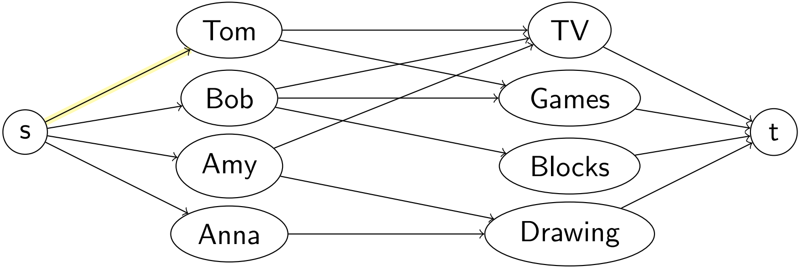
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

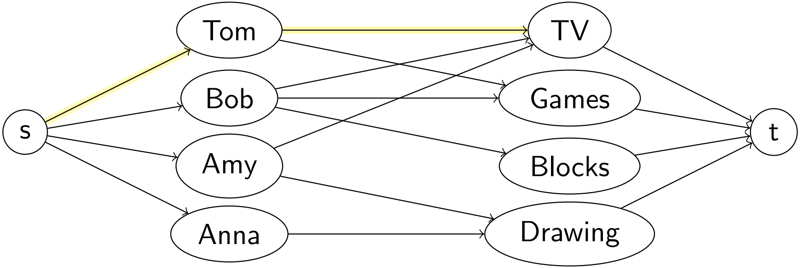


Matching

Bipartite Matching

Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

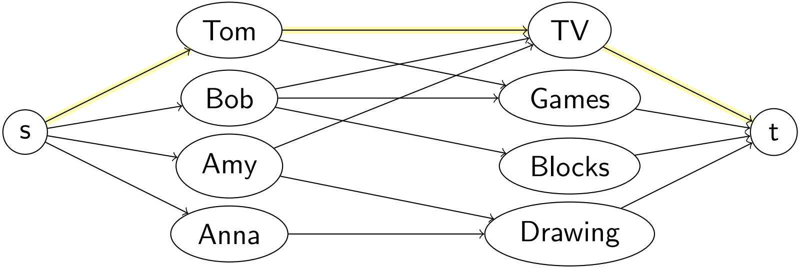


Matching

Bipartite Matching

Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

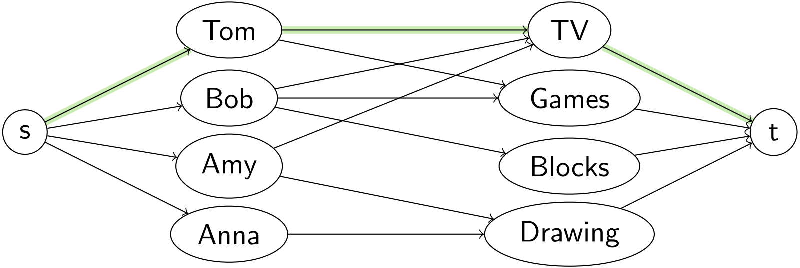


Matching

Bipartite Matching

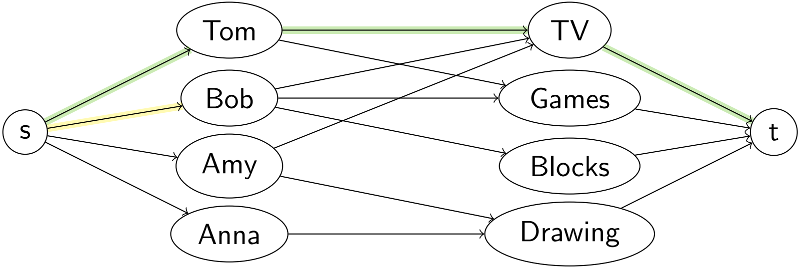
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

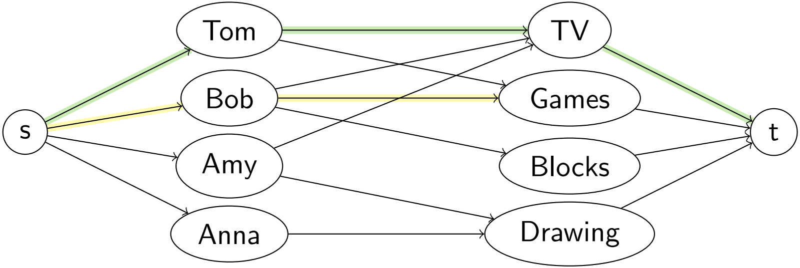


Matching

Bipartite Matching

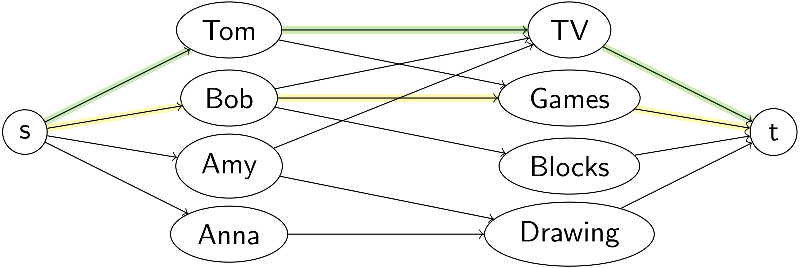
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

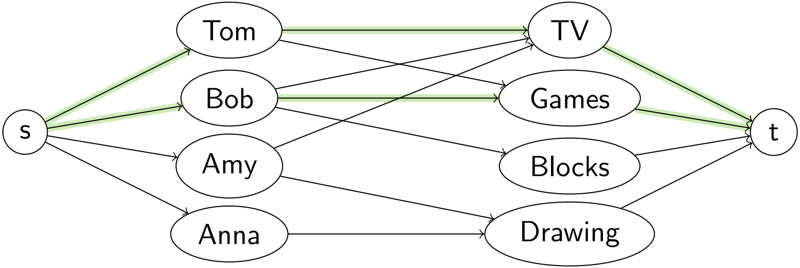


Matching

Bipartite Matching

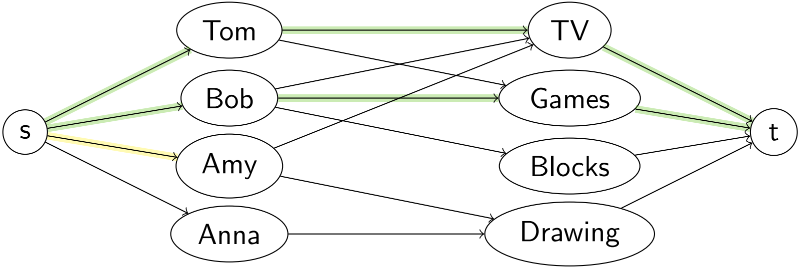
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



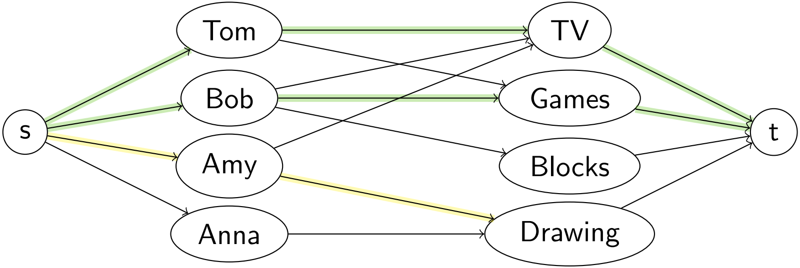
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

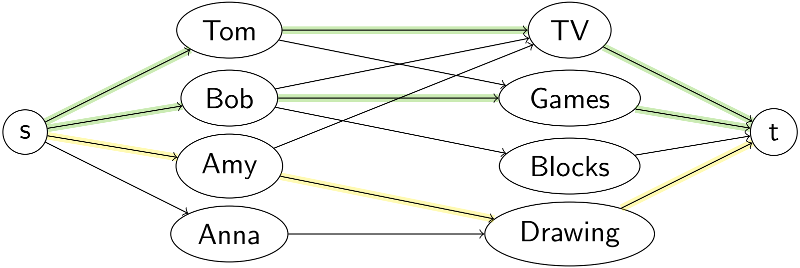
MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Matching
Bipartite Matching

Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

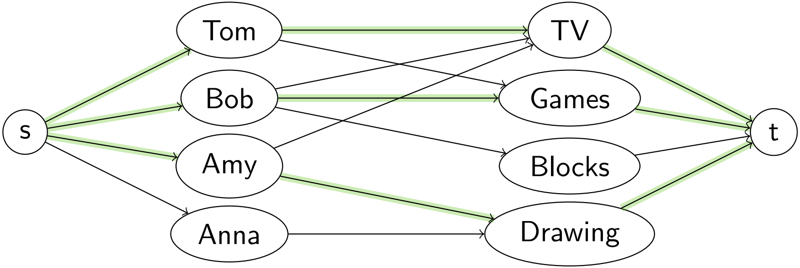


Matching

Bipartite Matching

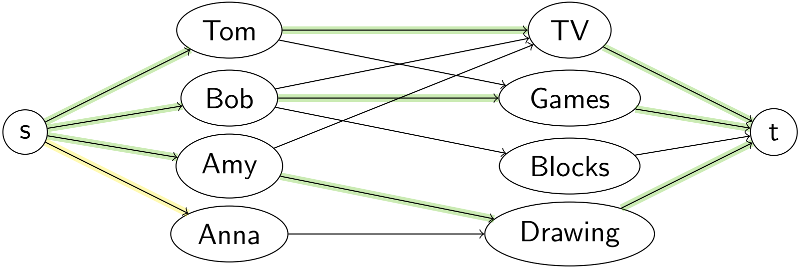
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



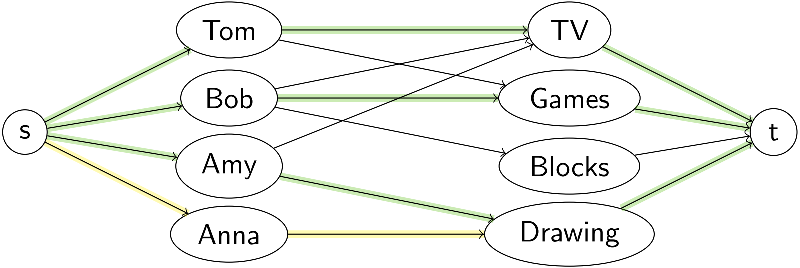
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

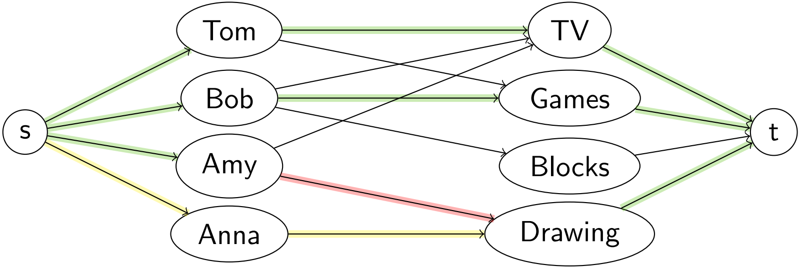


Matching

Bipartite Matching

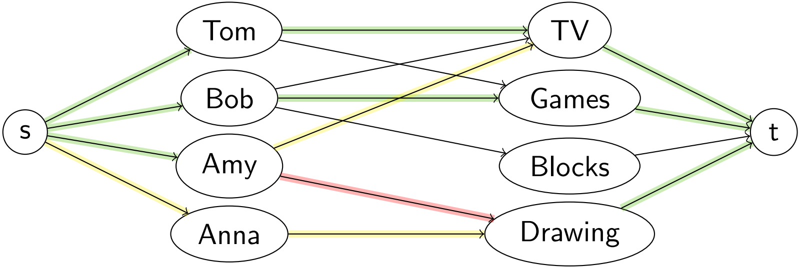
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



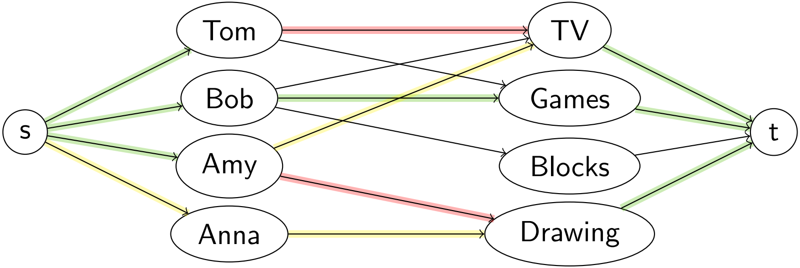
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



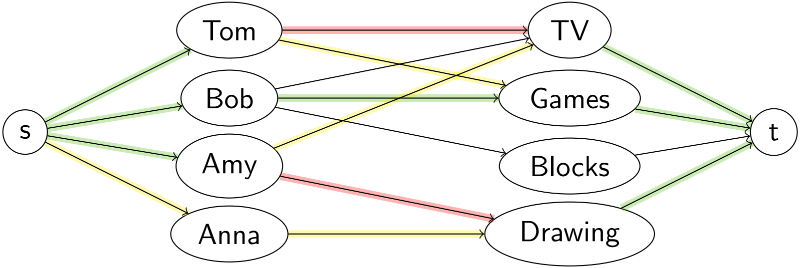
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

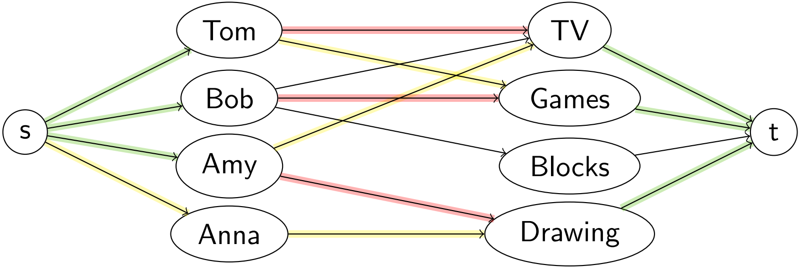


Matching

Bipartite Matching

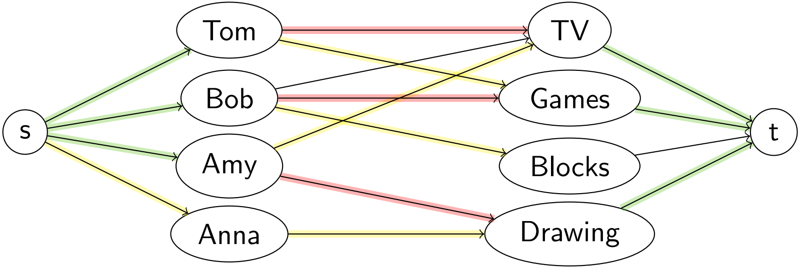
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



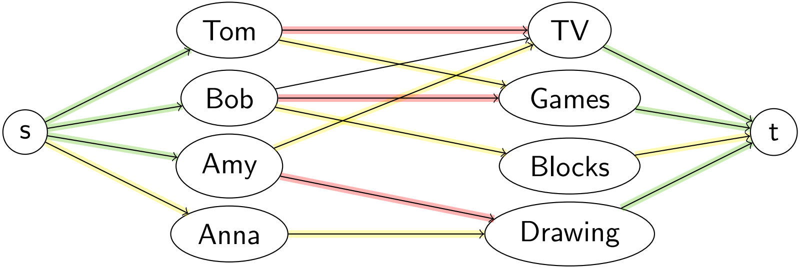
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



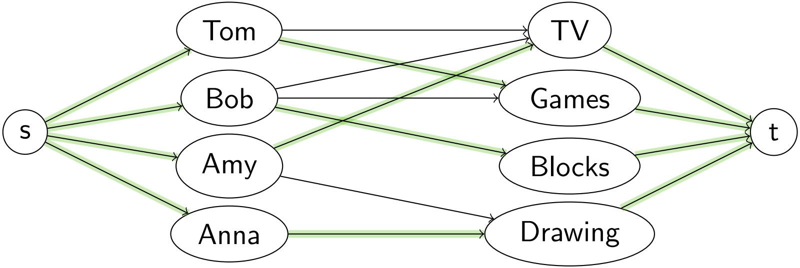
Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.



Maximum Cardinality Bipartite Matching

MCBM can be encoded as a Max Flow instance.
Set the capacity of every edge to 1.

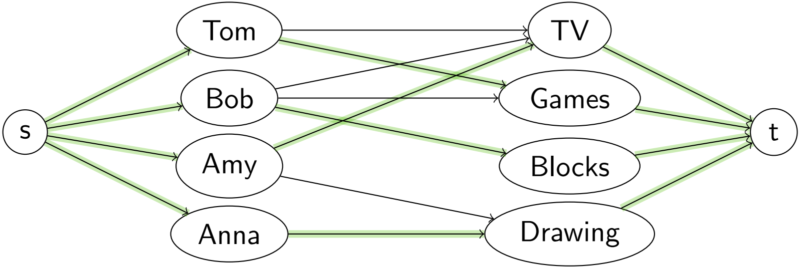


Matching

Bipartite Matching

Problem: Baby-Sitting

How many kids can keep themselves busy?
$\longrightarrow$ The MCBM's size is equal to the Maximum Flow from s to t

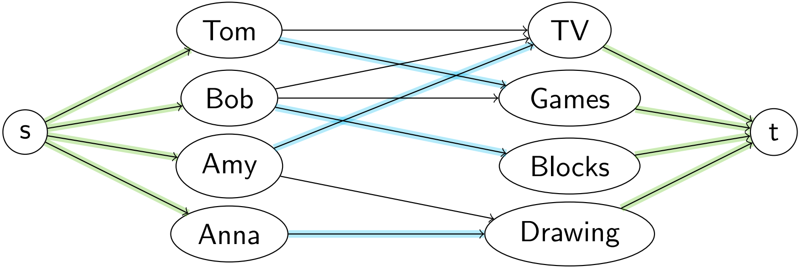


Matching

Bipartite Matching

Problem: Baby-Sitting

How many kids can keep themselves busy?
$\longrightarrow$ The MCBM's size is equal to the Maximum Flow from s to t



Remark: Matchings

Assignment problems – in particular bipartite matching – are abundant in competitive programming.

- Numerous MCBM algorithms exist
- Solving MCBM with Max Flow is usually sufficient

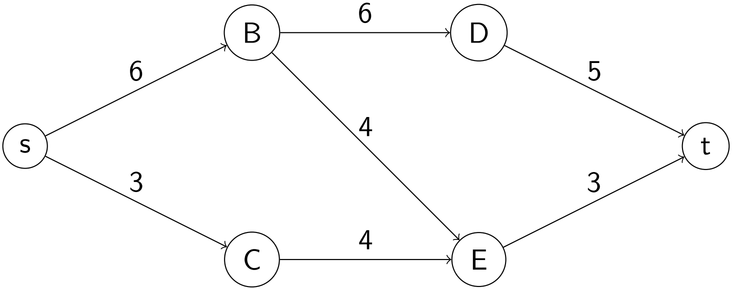
Solving Flow Problems

- Determine that the problem is indeed a flow problem
- Model the flow network
- Run Edmonds-Karp's Algorithm

Minimum Cuts

Problem: Sabotage

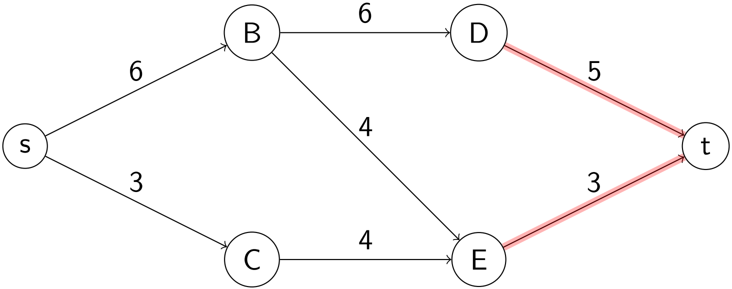
You want to prevent sabotages to your road system. In particular, you need to guarantee traffic from s to t. Some roads are easier to sabotage than others, thus every road has a sabotage cost c. Which roads would an attacker sabotage to stop traffic flow?



Minimum Cuts

Problem: Sabotage

You want to prevent sabotages to your road system. In particular, you need to guarantee traffic from s to t. Some roads are easier to sabotage than others, thus every road has a sabotage cost c. Which roads would an attacker sabotage to stop traffic flow?



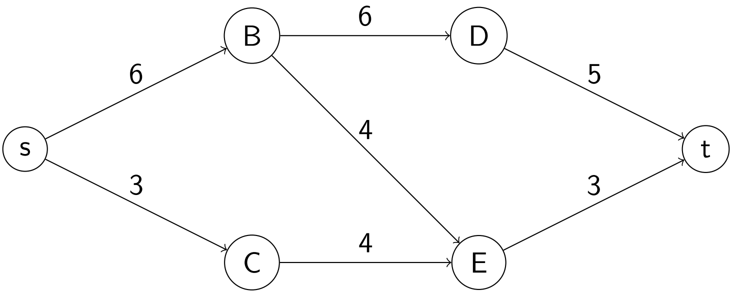
Minimum Cuts

Definition: s-t-Cut

Let s and t be two vertices of a graph. An *s-t-cut* is a set of edges in a graph, whose removal disconnects s and t.

Definition: Minimum s-t-Cut

In a weighted graph, an s-t-Cut C is called a minimum cut if the sum of the edge weights of C is the minimum among all s-t-cuts.



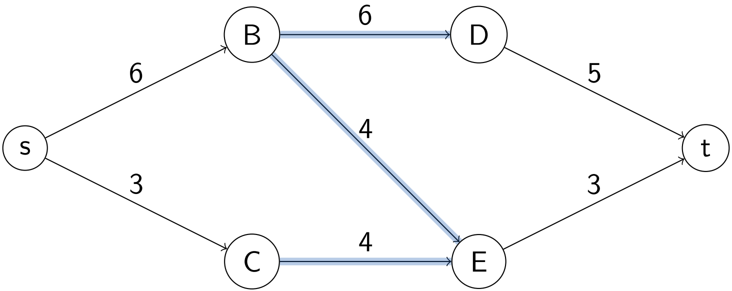
Minimum Cuts

Definition: s-t-Cut

Let s and t be two vertices of a graph. An *s-t-cut* is a set of edges in a graph, whose removal disconnects s and t.

Definition: Minimum s-t-Cut

In a weighted graph, an s-t-Cut C is called a minimum cut if the sum of the edge weights of C is the minimum among all s-t-cuts.



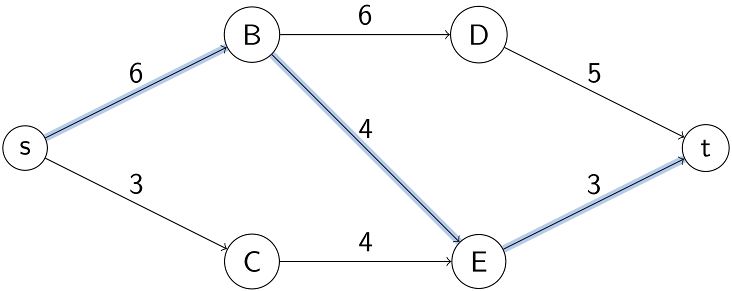
Minimum Cuts

Definition: s-t-Cut

Let s and t be two vertices of a graph. An *s-t-cut* is a set of edges in a graph, whose removal disconnects s and t.

Definition: Minimum s-t-Cut

In a weighted graph, an s-t-Cut C is called a minimum cut if the sum of the edge weights of C is the minimum among all s-t-cuts.



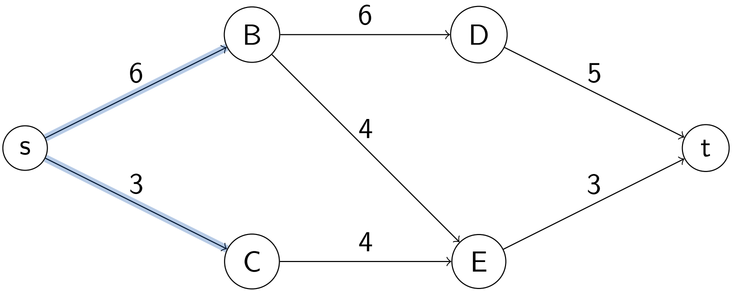
Minimum Cuts

Definition: s-t-Cut

Let s and t be two vertices of a graph. An *s-t-cut* is a set of edges in a graph, whose removal disconnects s and t.

Definition: Minimum s-t-Cut

In a weighted graph, an s-t-Cut C is called a minimum cut if the sum of the edge weights of C is the minimum among all s-t-cuts.



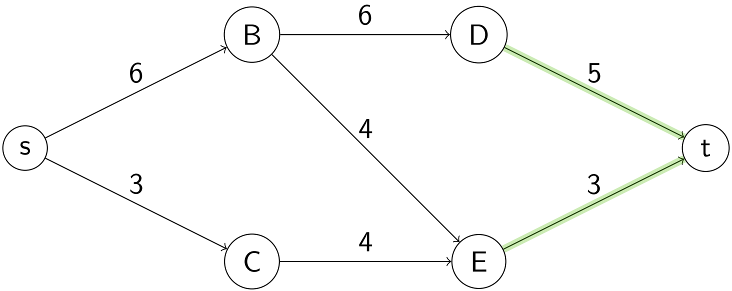
Minimum Cuts

Definition: s-t-Cut

Let s and t be two vertices of a graph. An *s-t-cut* is a set of edges in a graph, whose removal disconnects s and t.

Definition: Minimum s-t-Cut

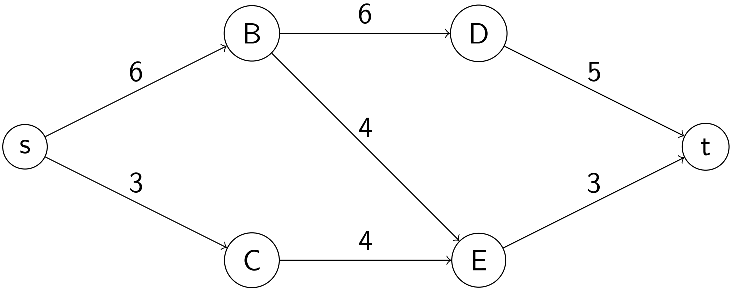
In a weighted graph, an s-t-Cut C is called a minimum cut if the sum of the edge weights of C is the minimum among all s-t-cuts.



Minimum Cuts

Theorem: Min Cut - Max Flow

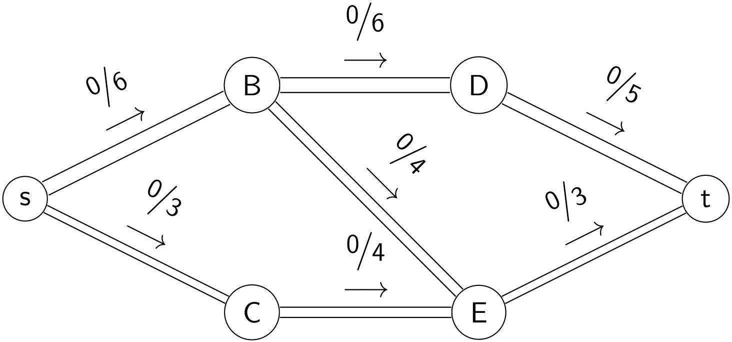
The value of the minimum s-t-cut is equal to the maximum flow from s to t.



Minimum Cuts

Theorem: Min Cut - Max Flow

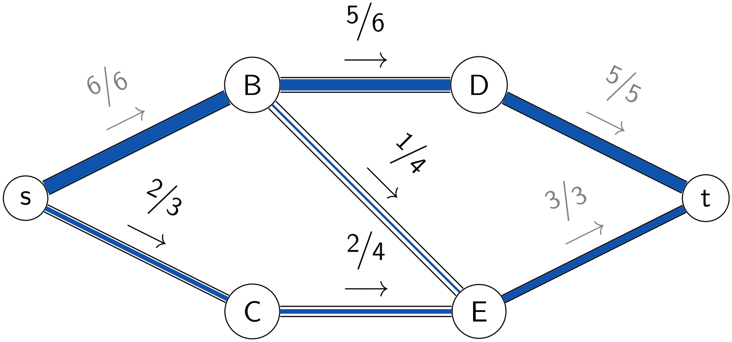
The value of the minimum s-t-cut is equal to the maximum flow from s to t.



Minimum Cuts

Theorem: Min Cut - Max Flow

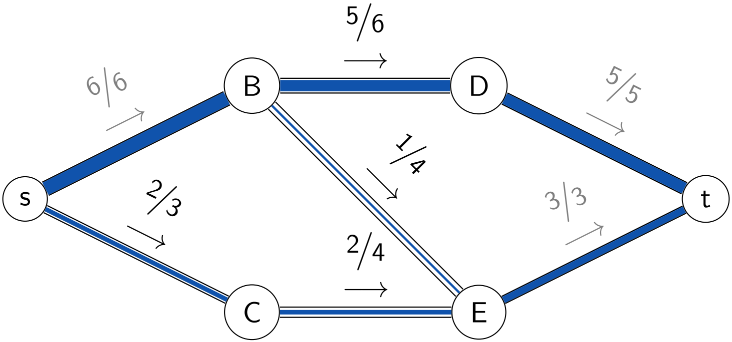
The value of the minimum s-t-cut is equal to the maximum flow from s to t.



Minimum Cuts

Finding Minimum Cuts

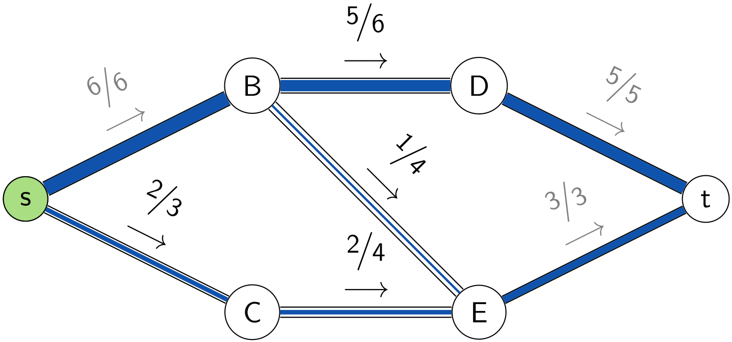
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

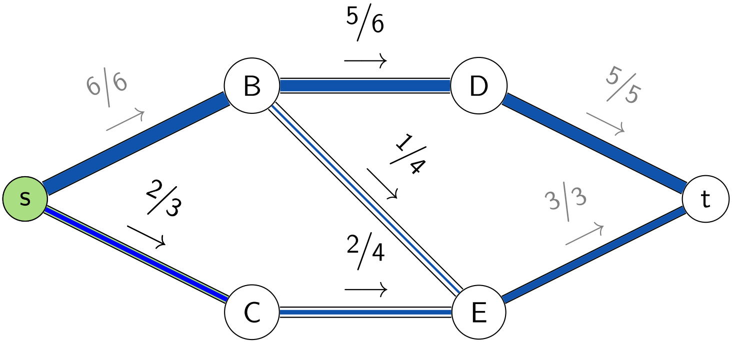
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

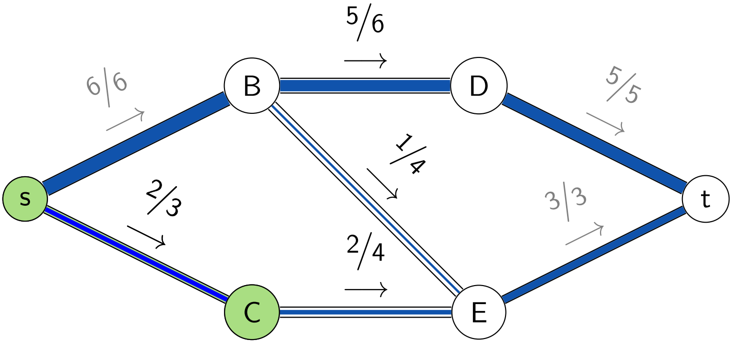
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

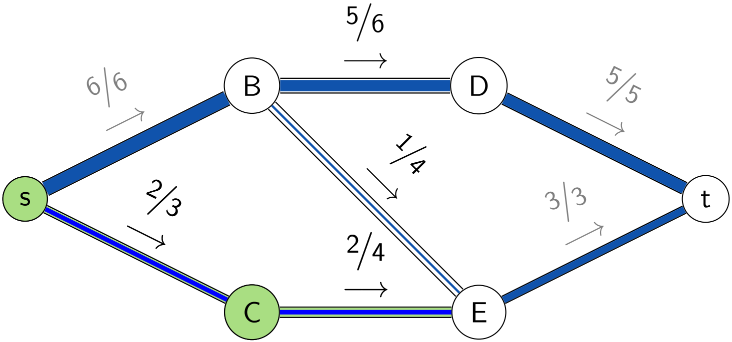
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

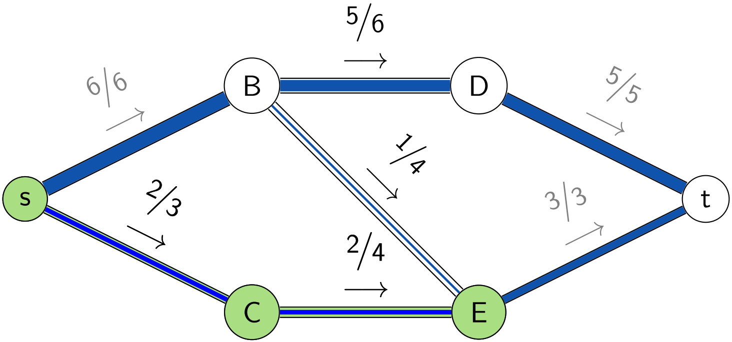
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

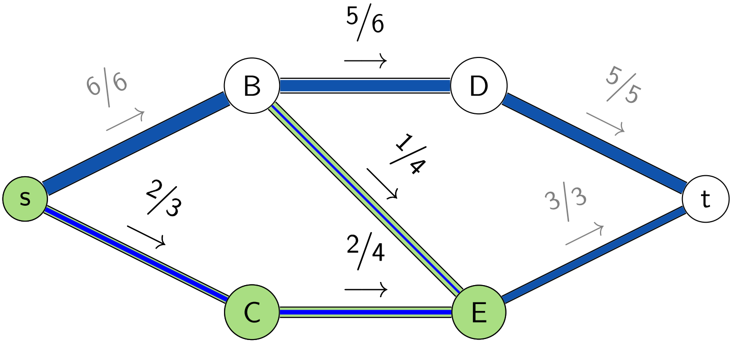
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

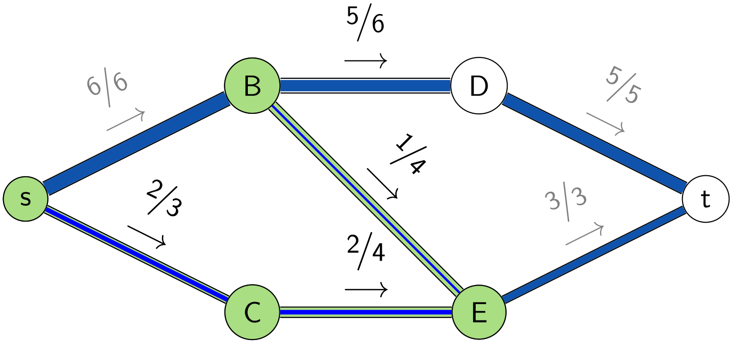
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

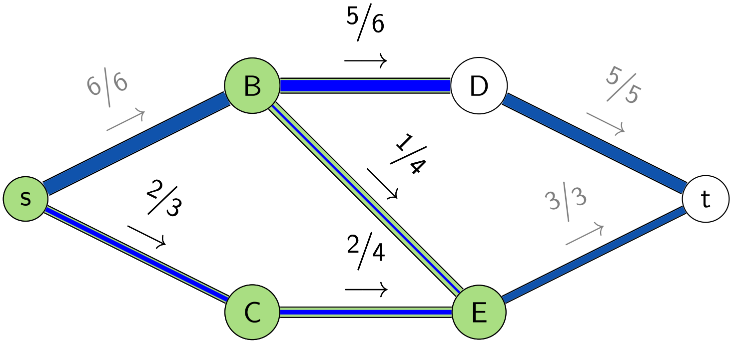
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges

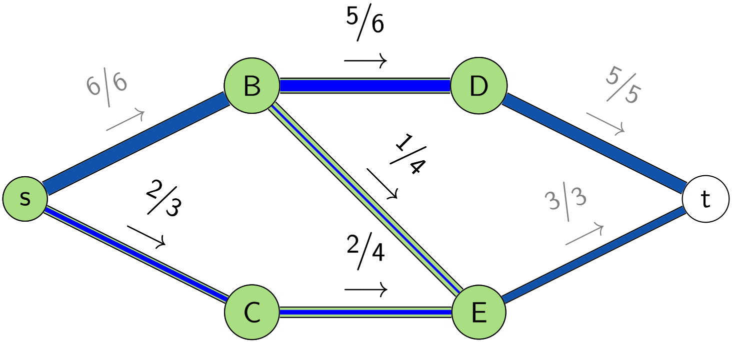


max planck institut informatik

Minimum Cuts

Finding Minimum Cuts

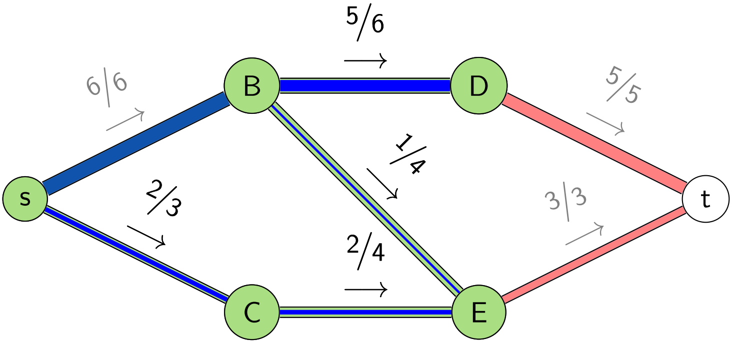
- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

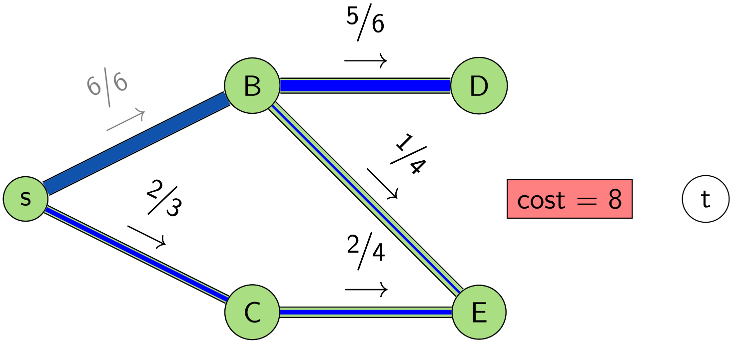
- Explore all vertices reachable from *s* using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Minimum Cuts

Finding Minimum Cuts

- Explore all vertices reachable from s using only edges with positive residual capacity (in the direction of traversal)
- All edges with zero residual capacity leaving this set are cut edges



Flow Problems & Matching

Today's Recap

We discussed

- Maximum Flow
- Algorithms of Ford-Fulkerson & Edmonds-Karp
- How to model Flow Networks
- Bipartite Matchings
- Minimum Cuts

Conclusion

End of the graph lecture series. You are now familiar with

- Graph Traversals and Applications
  - DFS and BFS
  - Computing Toposort
  - Finding Bridges & Articulation Points
  - Finding (Strongly) Connected Components
  - Determining LCAs
  - Solving MaxFlow and Matching
- Shortest Path Algorithms
  - BFS for unweighted graphs
  - Dijkstra, Bellman Ford, SPFA
  - Floyd Warshall
- Special Types of Graphs
  - DAGs (recall Toposorts)
  - Bipartite Graphs
  - Eulerian Graphs
  - Planar Graphs
  - Trees and Spanning Trees
  - Flow Graphs (and how to model)