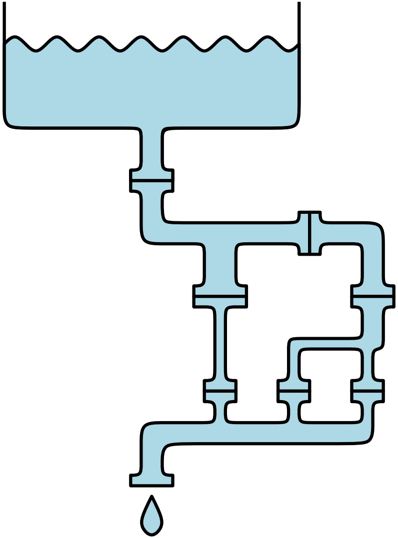
SOI Workshop 2015
Network Flows

Daniel Graf

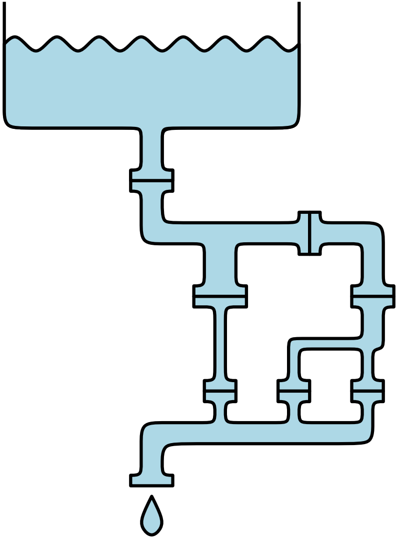
ETH Zürich

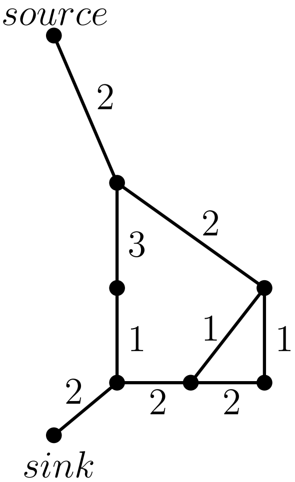
November 8, 2015

Network Flow: Example



Network Flow: Example

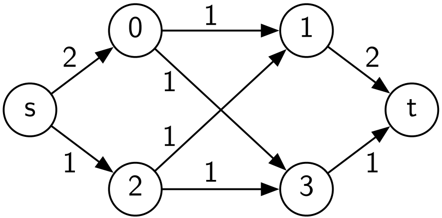




Network Flow: Problem Statement

Input: A flow network consisting of

- directed graph $G = (V, E)$
- source and sink $s, t \in V$
- edge capacity $c : E \to \mathbb{N}$.

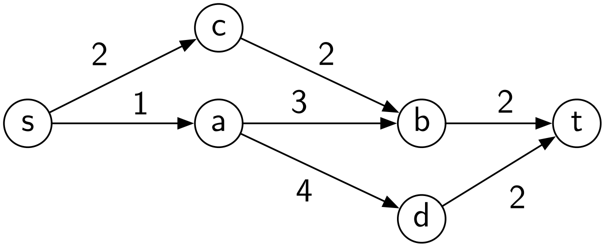


Output: A flow function $f : E \to \mathbb{N}$ such that:

- all capacity constraints are satisfied: $\forall u, v \in V : 0 \leq f(u, v) \leq c(u, v)$ (no pipe is overflowed)
- flow is conserved at every vertex: $\forall u \in V \setminus \{s, t\}$: $\sum_{(v,u) \in E} f(v, u) = \sum_{(u,v) \in E} f(u, v)$ (no vertex is leaking)
- the total flow is maximal: $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) = \sum_{u \in V} f(u, t) - \sum_{u \in V} f(t, u)$

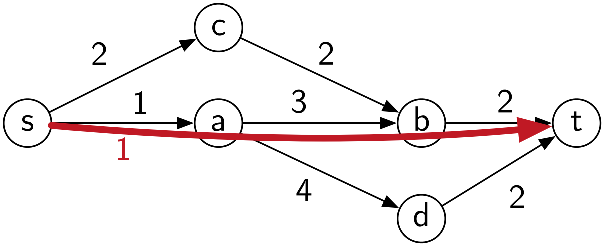
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Take any s-t-path and increase the flow along it.
- Update capacities and repeat as long as we can.
- Problem: We can get stuck at a local optimum.



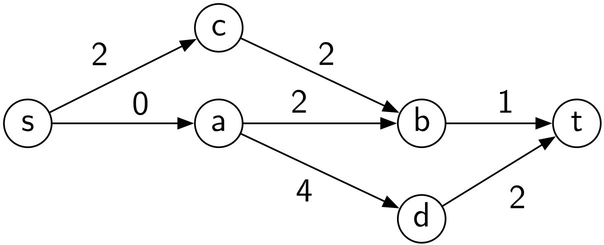
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Take any s-t-path and increase the flow along it.
- Update capacities and repeat as long as we can.
- Problem: We can get stuck at a local optimum.



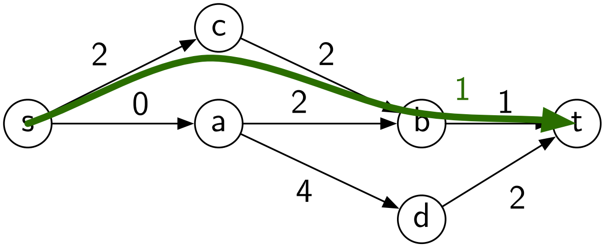
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Take any s-t-path and increase the flow along it.
- Update capacities and repeat as long as we can.
- Problem: We can get stuck at a local optimum.



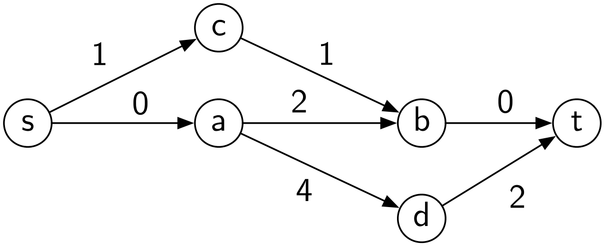
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Take any s-t-path and increase the flow along it.
- Update capacities and repeat as long as we can.
- Problem: We can get stuck at a local optimum.



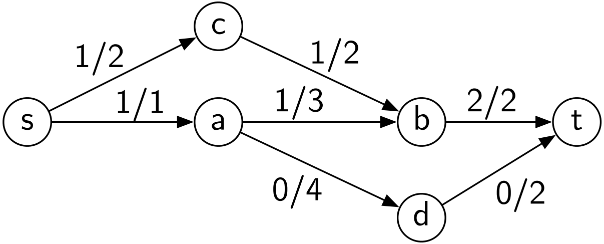
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Take any s-t-path and increase the flow along it.
- Update capacities and repeat as long as we can.
- Problem: We can get stuck at a local optimum.



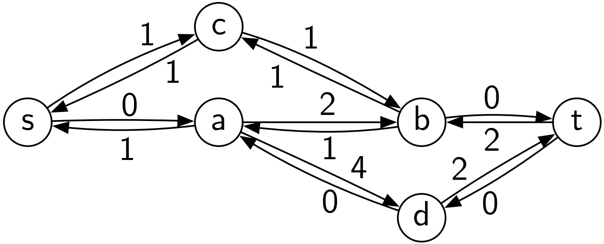
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



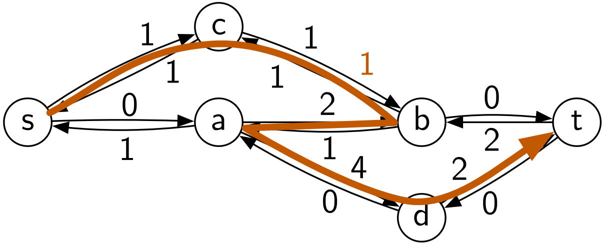
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



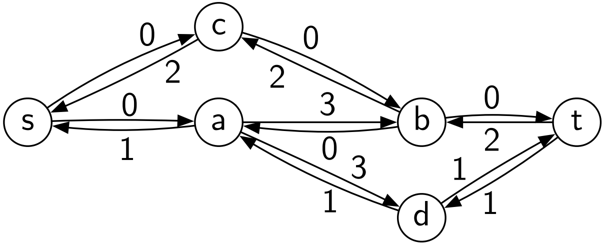
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



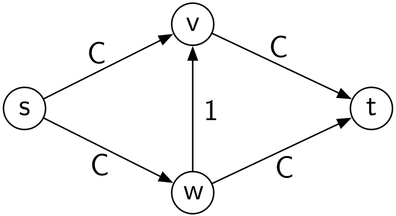
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



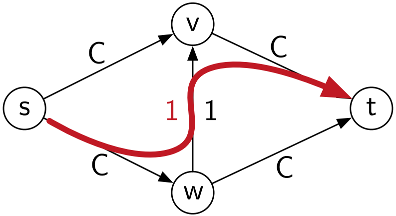
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



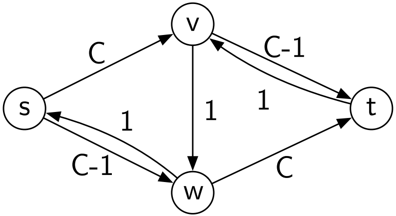
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



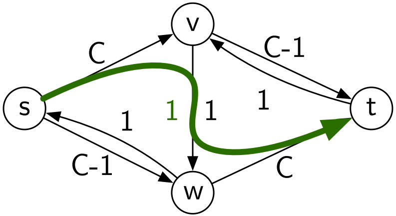
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



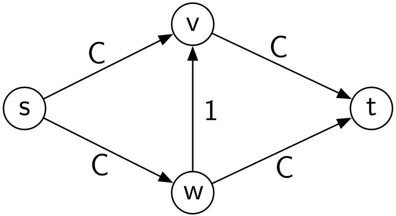
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



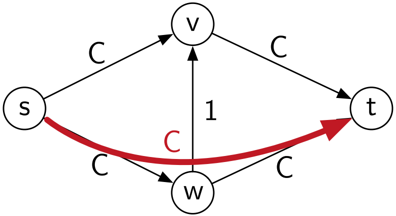
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



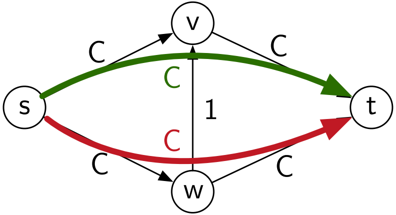
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- Solution: Keep track of the flow and allow paths that *reroute* units of flow. These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson: Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp: Repeatedly take the shortest augmenting path: running time: best of $\mathcal{O}(m|f|)$, $\mathcal{O}(nm \max c)$ and $\mathcal{O}(nm^2)$. [BGL-Doc].



Edmonds-Karp Maximum Flow Algorithm

The headers and edge struct:

```
// Sample implementation of the Edmonds Karp Algorithm
// Daniel Graf, grafdan@ethz.ch, 7.11.2015
#include <iostream>
#include <vector>
#include <cassert>
#include <queue>

#define INF 1000000000
typedef long long int in;
using namespace std;

struct Edge {
    in from, to, flow, cap, rev;
    in residual_capacity() {
        return cap-flow;
    }
};
...
```

Edmonds-Karp Maximum Flow Algorithm

Graph struct, add_edge function:

```
...
struct Graph {
    in s, t;
    vector<vector<Edge> > E; // adjacency-list of edges
    vector<Edge*> P; // predecessor map for the BFS

    Graph(in N) {
        E = vector<vector<Edge> >(N);
    }

    void add_edge(in from, in to, in cap) {
        if(from==to) return;
        E[from].push_back({from,to,0,cap,(in)E[to].size()});
        E[to].push_back({to,from,0,0,(in)E[from].size()-1});
    }
...
```

Edmonds-Karp Maximum Flow Algorithm

Reset all flow values:

```
...
void reset_flow() {
    for(in v=0; v<E.size(); v++) {
        for(in e=0; e<E[v].size(); e++) {
            E[v][e].flow = 0;
        }
    }
}
...
```

Edmonds-Karp Maximum Flow Algorithm

Can we find a path from s to t?

```
bool find_flow() {
    P = vector<Edge*>(E.size(),NULL);
    // Breadth First Search through the edges with remaining capacity
    queue<in> Q; Q.push(s);
    while(!Q.empty() && P[t]==NULL) {
        in v = Q.front(); Q.pop();
        for(in e=0; e<E[v].size(); e++) {
            if(E[v][e].residual_capacity()==0) {
                continue;
            }
            in w = E[v][e].to;
            if(P[w]==NULL) {
                P[w] = &(E[v][e]);
                Q.push(w);
            }
        }
    } ...
```

Edmonds-Karp Maximum Flow Algorithm

How much can we fit through that path from s to t?

```
...
    // Check if there is a path to t
    if(P[t] == NULL) {
        return 0;
    }
    // Check the minimum capacity
    in flow = INF;
    in pos = t;
    while(pos != s) {
        flow = min(flow, P[pos]->residual_capacity());
        pos = P[pos]->from;
    }
    return flow;
}
```

Edmonds-Karp Maximum Flow Algorithm

Increase the flow along the path and reduce the reverse edges.

```
void update_flow(int flow) {
    int pos = t;
    while(pos != s) {
        // cout << "update at vertex " << pos << endl;
        P[pos]->flow += flow;
        E[P[pos]->to][P[pos]->rev].flow -= flow;
        pos = P[pos]->from;
    }
}
```

Edmonds-Karp Maximum Flow Algorithm

Repeatedly search for the shortest augmenting path while one exists.

```
    in edmonds_karp_max_flow(in _s, in _t) {
        s = _s;
        t = _t;
        reset_flow();
        in flow = 0;
        in new_flow;
        do {
            new_flow = find_flow();
            flow += new_flow;
            if(new_flow > 0) {
                update_flow(new_flow);
            }
        } while(new_flow > 0);
        return flow;
    }
}; // end of the Graph struct
```

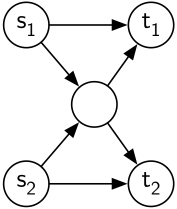
Edmonds-Karp Maximum Flow Algorithm

Read graph from input and call the algorithm.

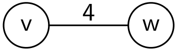
```
void read_graph_from_stdin(Graph &G) {
    in N,M;
    cin >> N >> M;
    G = Graph(N);
    for(in m=0; m<M; m++) {
        in a,b,c;
        cin >> a >> b >> c;
        G.add_edge(a,b,c);
    }
}
int main() {
    Graph G(0);
    read_graph_from_stdin(G);
    in s, t; cin >> s >> t;
    in res = G.edmonds_karp_max_flow(s,t);
    cout << "max␣flow:␣" << res << endl;
}
```

Common tricks

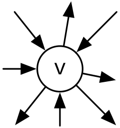
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

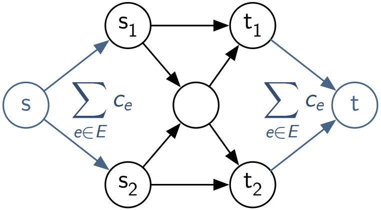


Minimum Flow per Edge

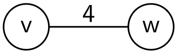
[Exercise]

Common tricks

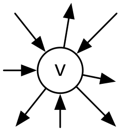
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

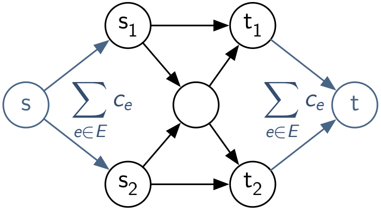


Minimum Flow per Edge

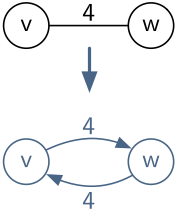
[Exercise]

Common tricks

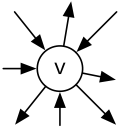
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

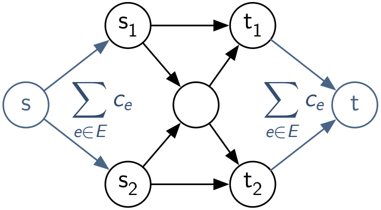


Minimum Flow per Edge

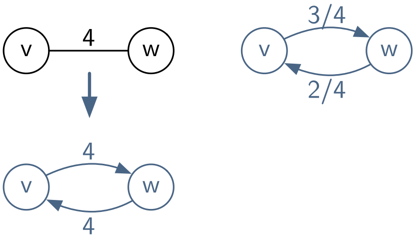
[Exercise]

Common tricks

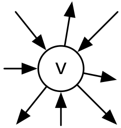
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

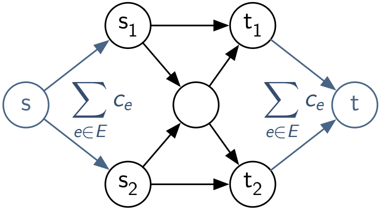


Minimum Flow per Edge

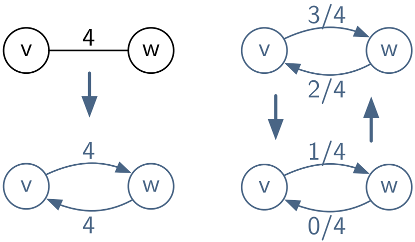
[Exercise]

Common tricks

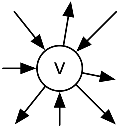
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

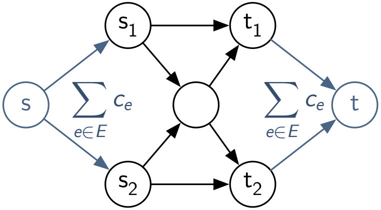


Minimum Flow per Edge

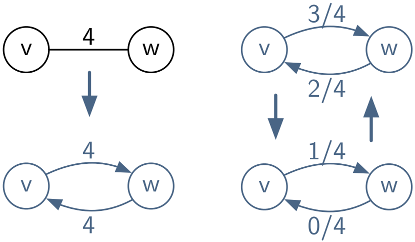
[Exercise]

Common tricks

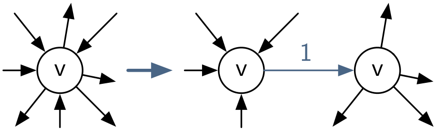
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

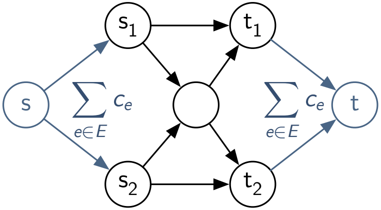


Minimum Flow per Edge

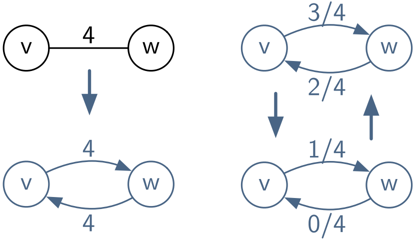
[Exercise]

Common tricks

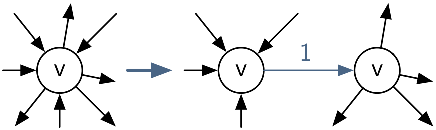
Multiple sources/sinks:



Undirected Graphs



Vertex Capacities

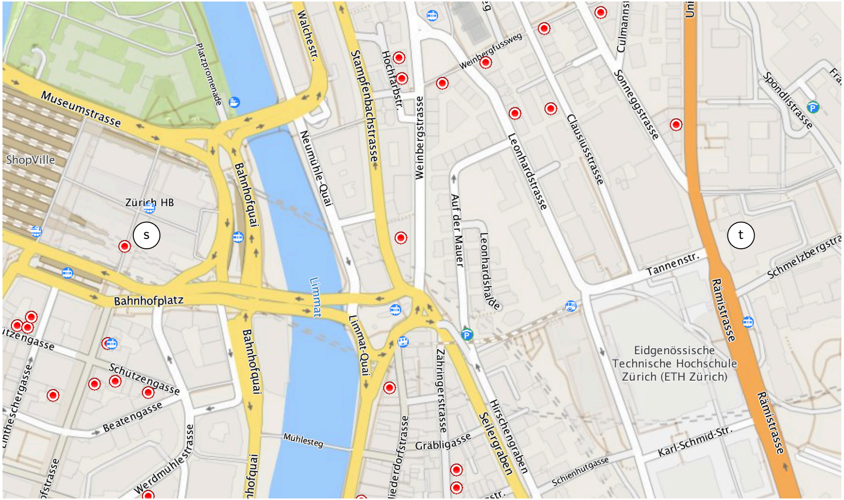


Minimum Flow per Edge

[Exercise]

Flow Application: Edge Disjoint Paths

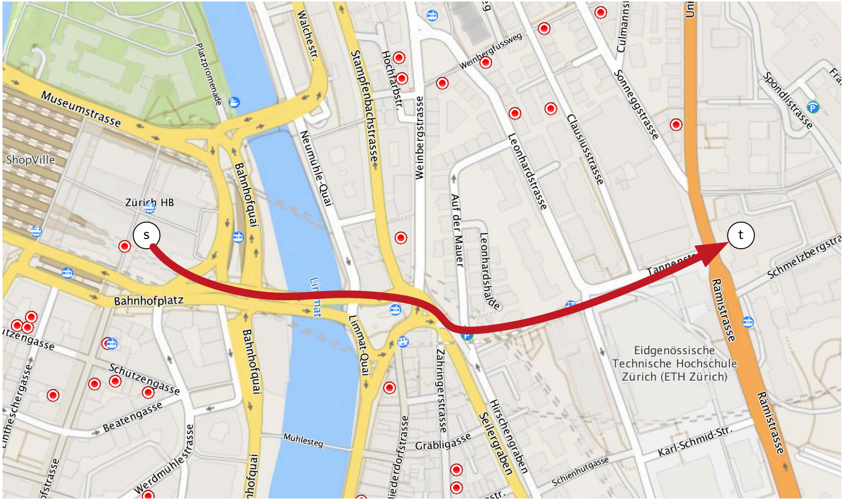
How many ways are there to get from HB to CAB without using the same street twice?



Map: search.ch, TomTom, swisstopo, OSM

Flow Application: Edge Disjoint Paths

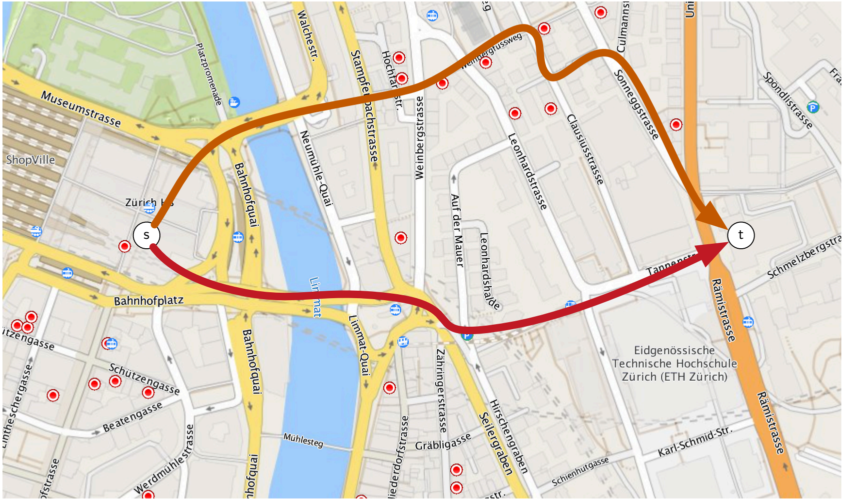
How many ways are there to get from HB to CAB without using the same street twice?



Map: search.ch, TomTom, swisstopo, OSM

Flow Application: Edge Disjoint Paths

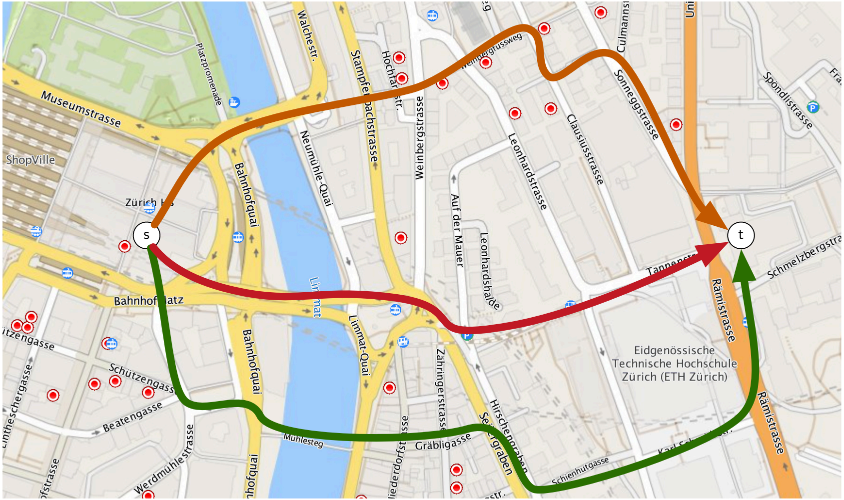
How many ways are there to get from HB to CAB without using the same street twice?



Map:
search.ch,
TomTom,
swisstopo,
OSM

Flow Application: Edge Disjoint Paths

How many ways are there to get from HB to CAB without using the same street twice?



Map: search.ch, TomTom, swisstopo, OSM

Flow Application: Edge Disjoint Paths

How many ways are there to get from HB to CAB without using the same street twice?

- Is this a flow problem? No.
- Can it be turned into a flow problem? Maybe.
- Build directed street graph by adding edges in both directions.
- Set all capacities to 1.

Lemma

In a directed graph with unit capacities, the maximum number of edge-disjoint s-t-paths is equal to the maximum flow from s to t.

Flow Application: Edge Disjoint Paths

How many ways are there to get from HB to CAB without using the same street twice?

- Is this a flow problem? No.
- Can it be turned into a flow problem? Maybe.
- Build directed street graph by adding edges in both directions.
- Set all capacities to 1.

Lemma

In a directed graph with unit capacities, the maximum number of edge-disjoint s-t-paths is equal to the maximum flow from s to t.

Flow Application: Edge Disjoint Paths

How many ways are there to get from HB to CAB without using the same street twice?

- Is this a flow problem? No.
- Can it be turned into a flow problem? Maybe.
- Build directed street graph by adding edges in both directions.
- Set all capacities to 1.

Lemma

In a directed graph with unit capacities, the maximum number of edge-disjoint s-t-paths is equal to the maximum flow from s to t.

Flow Application: Edge Disjoint Paths

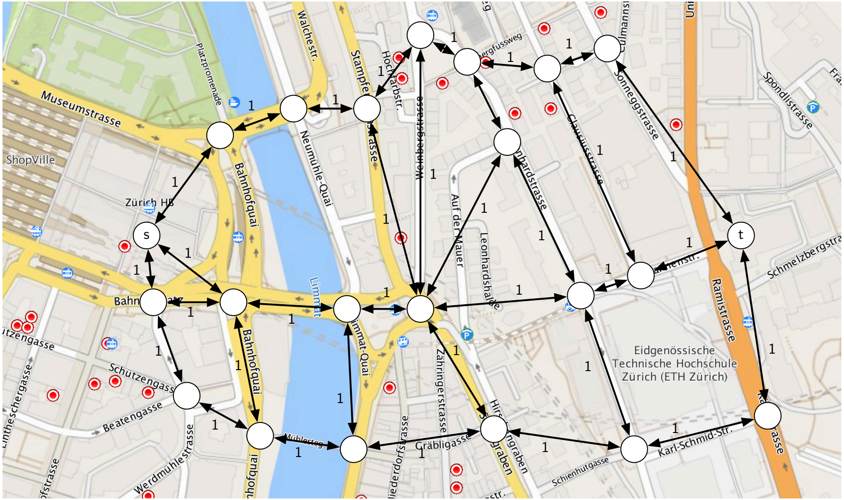
How many ways are there to get from HB to CAB without using the same street twice?

- Is this a flow problem? No.
- Can it be turned into a flow problem? Maybe.
- Build directed street graph by adding edges in both directions.
- Set all capacities to 1.

Lemma

In a directed graph with unit capacities, the maximum number of edge-disjoint s-t-paths is equal to the maximum flow from s to t.

Flow Application: Edge Disjoint Paths



Map:
search.ch,
TomTom,
swisstopo,
OSM