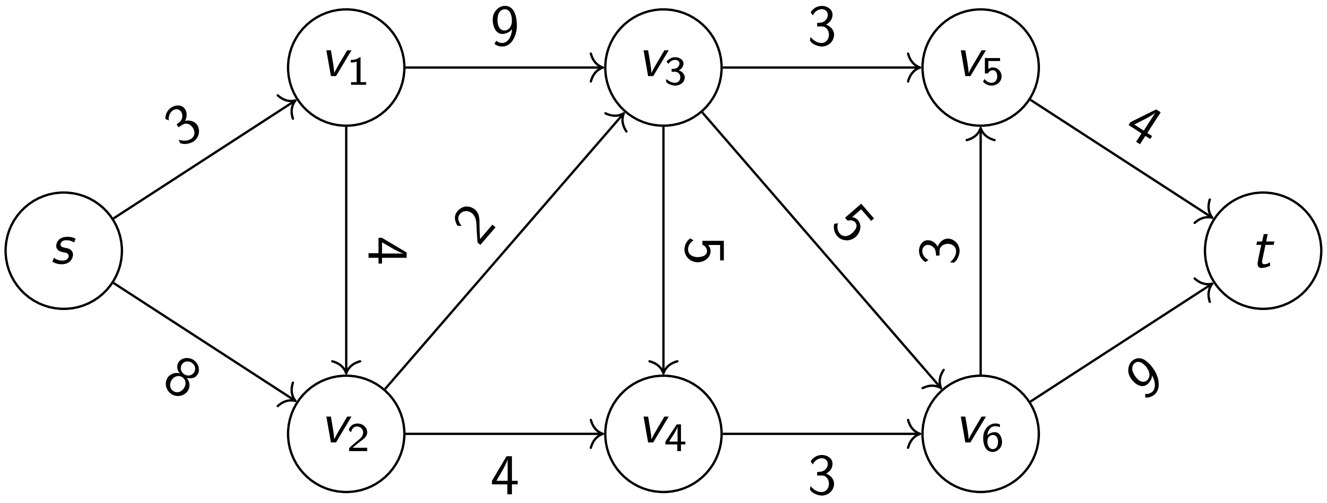
Contents Lecture 7

- The maximum flow problem
- The Ford-Fulkerson algorithm
- Maximum flows and minimum cuts
- The preflow-push maximum flow algorithm

A flow network

- A directed (or undirected) graph $G(V, E)$
- Each edge $e \in E$ has a nonnegative capacity $c(e)$
- A source node $s \in V$ with no predecessor
- A sink node $t \in V$ with no successor
- An example:



- A previous version of Lab 6 was about being an CCCP party member and solving a problem for railway transportations passing Minsk, using capacities estimated by US spies — hence book cover

Terminology

- An $st - cut$ is a partition (A, B) with $s \in A$ and $t \in B$. Also called simply a *cut*
- The **capacity** of a cut is

$$cap(A, B) = \sum_{e \text{ out from } A} c(e)$$

- For the previous graph, $cap(\{s\}, V - \{s\}) = 3 + 8$
- The **min-cut problem** is to find a cut of minimum capacity
- Useful information when bombing enemy railroads for instance
- Honest and respectful diplomacy towards a happy world is preferable

A flow

- A **flow** is a function f which says how much flows on each edge
- Often we want to use the edges to maximize the total flow from s to t
- The algorithm design techniques we have studied so far are insufficient to solve this problem
- The **capacity constraint** says: for each $e \in E$, $0 \leq f(e) \leq c(e)$
- For undirected graphs, we need to specify the direction of the flow
- One way to do that is to fix the order of the nodes and use:
    - flow from u to v is positive
    - flow from v to u is negative
    - u and v need to agree on what is meant by positive flow

Flow conservation constraint

- The flow coming in to a vertex v must equal the flow going out from v
- This **flow conservation constraint** does not apply to the source s and the sink t

$$v \in V - \{s, t\} : \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out from } v} f(e)$$

- A water hose (vattenslang) cannot store any water
- Water systems are a good mental model for network flow

The maximum flow problem

- The value of a flow f is $\sum_{e \text{ out from } s} f(e) = \sum_{e \text{ in to } t} f(e)$
- The **maximum flow problem** is to find a flow f with maximum value

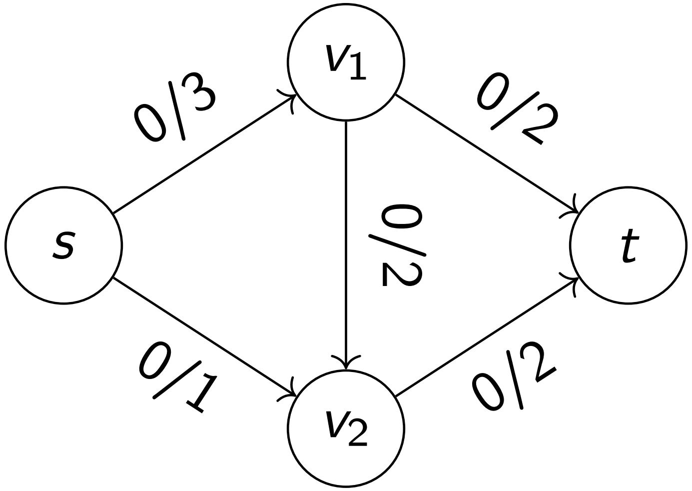
The Ford-Fulkerson algorithm: overview

- The basic idea is very simple
  1. Start with a flow $f(e) = 0$ for every $e \in E$
  2. Look for a simple path p from s to t such that on every edge (u, v) in p we can increase the flow in the direction from u to v
  3. If we could not find any such path, we have the maximum flow
  4. Let each edge $e = (u, v)$ on p have a value $\delta(e)$, which means room for improvement, or how much we can increase the flow on that edge
  5. Let Δ be the minimum of all $\delta(e)$ on p
  6. Increase the flow by Δ along the path p
  7. `goto` 2
- The risky part is number 3: how can we be sure of that?
- We will prove it is correct

Some more details

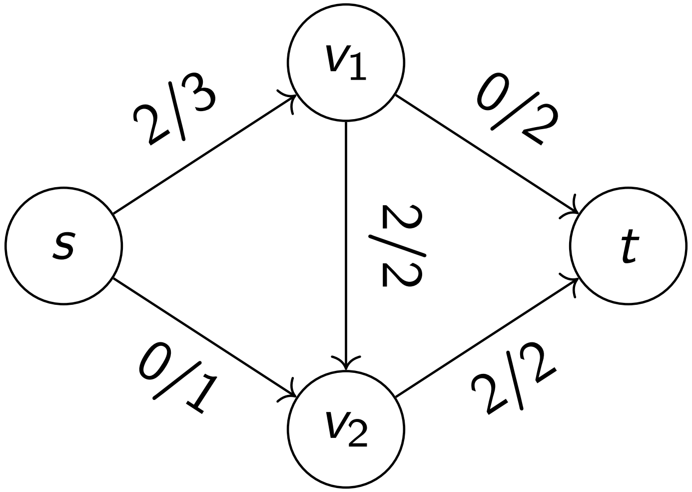
- What does it mean that $\delta(u, v) > 0$?
- Answer: $f(u, v) < c(u, v)$
- It is clear that if we find such a path p we can increase the flow on each edge of that path p by Δ
- From what we have so far, we cannot decrease the flow of any edge, so we still easily can get stuck
- But consider an edge $e = (u, v)$ with a flow $f(e)$
- To decrease this flow, we can instead increase the flow of a new edge (v, u) by up to the amount $f(e)$
- We thus need additional edges and therefore create a new graph G_f for that

An example



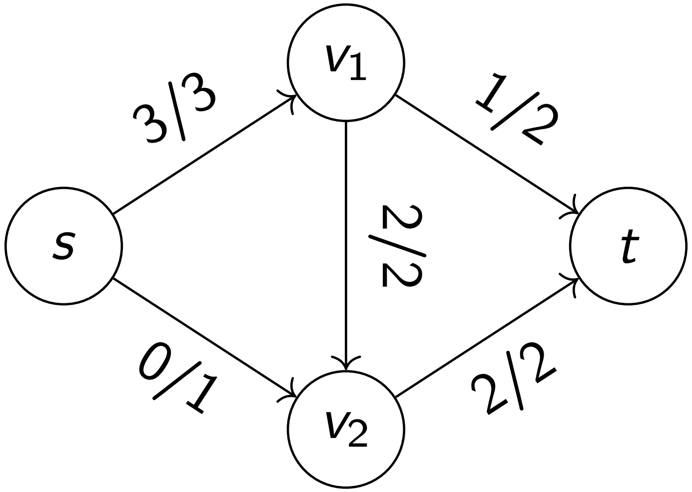
- Using BFS to find an $s - t$ path is a good idea
- Let the first BFS find the path: $p_1 = (s, v_1, v_2, t)$ with $\delta = 2$.

An example



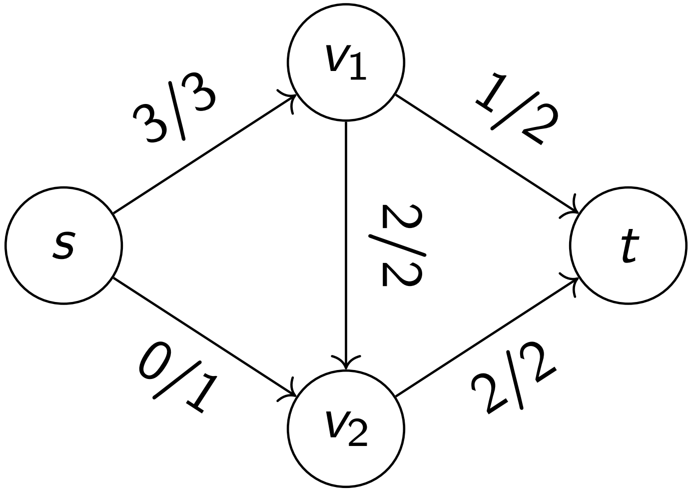
- In the second BFS there is "no edge" between v_2 and t since its flow cannot be increased
- So BFS cannot reach t going through v_2
- The path $p_2 = (s, v_1, t)$ with $\delta = 1$ is found.

An example



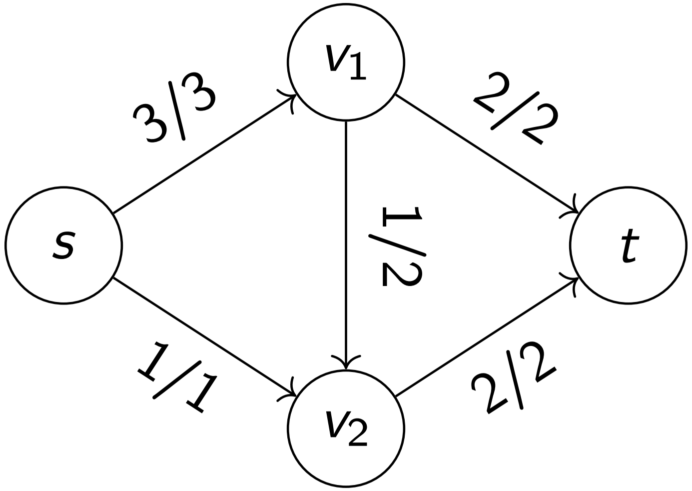
- Now no path can be found!
- What to do?

An example



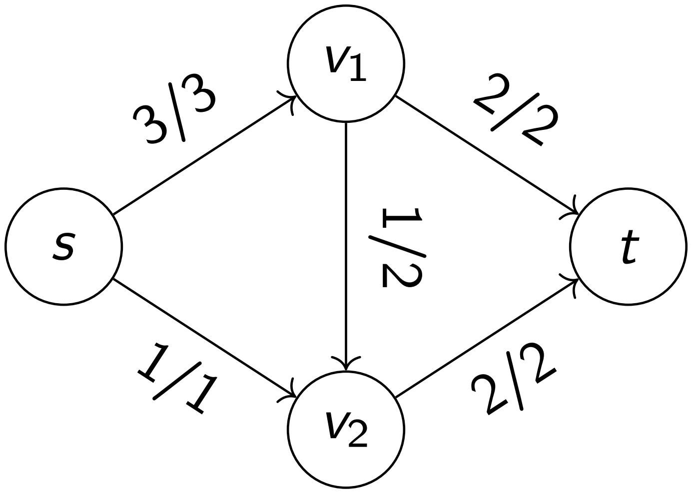
- Can we somehow send flow using $p_3 = (s, v_2, v_1, t)$?

An example



- We have in some sense reduced the flow from v_1 to v_2
- Note we only changed the flow along $p_3 = (s, v_2, v_1, t)$?

An example



- This is the maximum flow
- We need a simple and systematic approach for this
- The "residual graph" has the same nodes but edges correspond to where we can increase or decrease flow.
- In this graph an original edge is called a forward edge
- To reduce flow a backward edge is created

The residual graph

- We create a **residual graph** G_f with the same nodes as G
- An edge in G becomes either one or two edges in G_f (one of them with reversed direction)
- Edges in G_f have the capacities:

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E \quad \text{a forward edge} \\ f(v,u) & \text{if } (v,u) \in E \quad \text{a backward edge} \\ 0 & \text{otherwise} \end{cases}$$

- If $c(u,v) = a$, $f(u,v) = b$, and $a > b$, two edges are created in G_f:
  - One forward edge (u,v) with capacity $a - b$ since this is how much we can increase the flow
  - One backward edge (v,u) with capacity b since this is how much we can decrease the flow
- If $f(u,v) = c(u,v)$ in G then only a backward edge is created in G_f
- With $f(u,v) = c(u,v)$ in G we can only decrease the flow on (u,v)

The Ford-Fulkerson algorithm

```
procedure $ford\_fulkerson(G, s, t, c)$
    for each $e \in E$ do $f(e) \leftarrow 0$
    $G_f \leftarrow$ create initial residual graph
    while $(p \leftarrow find\_path(G_f)) \neq null$ do
        update $G_f$ according to previous slide
```

Ford-Fulkerson algorithm or method?

- Ford and Fulkerson did not specify how the path should be found
- Different options result in different time complexity and therefore it is sometimes called a **method** and not an algorithm — such as in Cormen, Leiserson, Rivest and Stein *Introduction to Algorithms* — i.e. CLRS (about 1300 pages)
- If breadth first search is used, it is called the Edmond-Karp algorithm
- We will use the name Ford-Fulkerson algorithm

Correctness of the Ford-Fulkerson algorithm

- We need to show that after updating G_f it still satisfies the two constraints for being a network flow, the capacity and conservation constraints
- We also need to prove that it actually terminates — maybe it does not?

Termination of the Ford-Fulkerson algorithm

- Will it eventually terminate?
- It depends. If we use infinite precision of the representation of the capacities and flows, and the capacities are carefully selected irrational numbers, it will not terminate
- Showing this is beyond the scope of the course
- In practise this is not a problem because real numbers are represented as floating point numbers which means they really are rational numbers
- If the capacities are integers, then all flows will also be integers and the algorithm clearly will terminate since it improves the flow at least by one each iteration (exactly by Δ)
- The sum C of capacities out from s is an upper bound on the maximum flow so it will terminate after at most C iterations

Running time of the Ford-Fulkerson algorithm

- As usual, n is the number of nodes and m the number of edges in G
- Assume all capacities are integers
- Let the sum of capacities out from s be C
- We assume $m \geq n$ to make our analysis simpler

Lemma

The Ford-Fulkerson algorithm can be implemented to run in $O(Cm)$ time

Proof.

At most C iterations to find a path are needed. Finding a path using e.g. breadth-first search and an adjacency list representation, can be done in $O(n+m)$ and by our assumption this is equal to $O(m)$. Updating G and G_f using the path also needs $O(m)$ time □

Cuts

- Recall a partitioning of V into A and B means
  - $V = A \cup B$, and
  - $A \cap B = \emptyset$
- A cut is a partitioning (A, B) such that $s \in A$ and $t \in B$
- How are cuts and flows related?

Flows and cuts

- The value of a flow is denoted by $v(f)$
- Consider any cut (A, B) with $s \in A$ and $t \in B$
- $f^{\text{in}}(s) = 0$
- $v(f) = f^{\text{out}}(s)$
- So $v(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$
- For all nodes $u \in V - \{s, t\}$ we have $f^{\text{out}}(u) = f^{\text{in}}(u)$
- Thus for all nodes $u \in A - \{s\}$ we have $f^{\text{out}}(u) - f^{\text{in}}(u) = 0$
- Therefore we can write: $v(f) = f^{\text{out}}(s) = \sum_{u \in A} f^{\text{out}}(u) - f^{\text{in}}(u)$
- See next slide

Edges, flows, and cuts

- Again $v(f) = \sum_{u \in A} f^{\text{out}}(u) - f^{\text{in}}(u)$
- Consider any edge $e \in E$. We have four cases:
  1. No end in A: The edge does not affect the flow in A
  2. From B to A: the flow will be counted only as $-f^{\text{in}}(u)$
  3. From A to B: the flow will be counted only as $f^{\text{out}}(u)$
  4. Both ends in A: the flow will be counted both as $f^{\text{out}}(u)$ and as $f^{\text{in}}(u)$ above and thus cancels (by different terms in the sum)
- Thus: $v(f) = \sum_{u \in A} f^{\text{out}}(u) - f^{\text{in}}(u) = \sum_{e \text{ out of } A} f^{\text{out}}(e) - \sum_{e \text{ in to } A} f^{\text{in}}(e)$
- We have just shown:

Lemma

Let f be any $s - t$ flow and (A, B) any $s - t$ cut. Then $v(f) = f^{out}(A) - f^{in}(A)$

Viewing the flow from t

- The value of a flow f can also be written $v(f) = f^{\text{in}}(t)$
- This is clear but we can also see it follows from what we just saw
- Since the edges out of A are the edges in to B, we have $f^{\text{out}}(A) = f^{\text{in}}(B)$
- And since the edges out of B are the edges in to A we have $f^{\text{out}}(B) = f^{\text{in}}(A)$
- Therefore $v(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$
- Since $f^{\text{out}}(t) = 0$ we have with $B = \{t\}$ the expected $v(f) = f^{\text{in}}(t)$

Capacities, flows, and cuts

- The capacity of a cut (A, B) is $\sum_{e \text{ out of } A} c(e)$ and it is denoted $c(A, B)$
- We have:

$$\begin{aligned} v(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \\ &\leq f^{\text{out}}(A) \\ &= \sum_{e \text{ out of } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &= c(A, B) \end{aligned}$$

- Therefore:

Lemma

The value of any flow is limited by the capacity of any cut: $v(f) \leq c(A, B)$

Exploiting $v(f) \leq c(A, B)$

- If we can show that the flow f found by the Ford-Fulkerson algorithm is equal to the capacity of any cut (A, B) then we know the algorithm finds the maximum flow since the flow must pass every cut

Lemma

If there is an $s - t$ flow f in G such that there is no $s - t$ path in G_f then f has the maximum flow in G

- See the next slides for the proof

No s-t path in G_f means $f(e) = c(e)$ for e crossing cut

Proof.

- Let A be the set of nodes reachable from s in G_f and $B = V - A$
- Since s is reachable from itself, $s \in A$ and therefore A is not empty
- By assumption, there is no $s - t$ path in G_f and therefore $t \in B$ and B is not empty
- Thus (A, B) is both a partition and a cut
- For any edge $e = (u, v)$ such that $u \in A$ and $v \in B$ we will next see that $f(e) = c(e)$
- Assume in contradiction that $f(e) < c(e)$. Since $c(e) - f(e) > 0$ there exists a forward edge e in G_f with $c_f(e) > 0$. Since $u \in A$ there is a path from s to v in G_f. Since this is a contradiction, $f(e) = c(e)$

☐

No s-t path in G_f means no flow back across cut

Proof.

- For any edge $e = (v, u)$ such that $v \in B$ and $u \in A$ we will next see that $f(e) = 0$
- Assume in contradiction that $f(e) > 0$. Since $f(e) > 0$ there exists a backward edge $e' = (u, v)$ with $c(e') > 0$ in G_f. But e' makes v reachable from s in G_f which is a contradiction, and therefore $f(e) = 0$
- We have showed that all edges e out from A have $f(e) = c(e)$ and all edges e in to A have $f(e) = 0$

□

Proving optimality of the Ford-Fulkerson algorithm

Proof.

$$\begin{aligned} v(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \\ &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to of } A} f(e) \\ &= \sum_{e \text{ out of } A} c(e) - 0 \\ &= c(A, B) \end{aligned}$$

☐

- We have shown that the flow computed by the Ford-Fulkerson algorithm is equal to a cut, and this means it is optimal

The max-flow min-cut theorem

- Recall: *the value of any flow is limited by the capacity of any cut:* $v(f) \leq c(A, B)$

Theorem

The maximum flow is equal to the minimum cut

Proof.

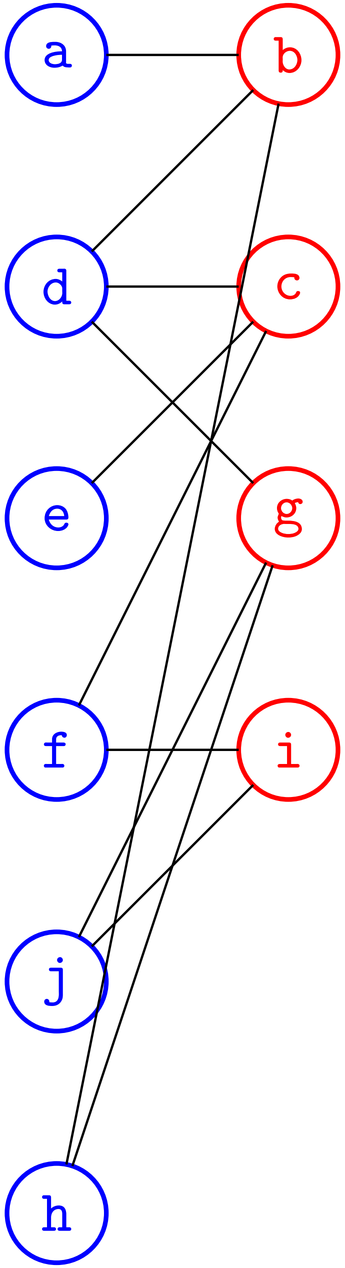
- Consider any flow f and cut (A, B) such that $v(f) = c(A, B)$
- Assume in contradiction there exists a flow $v'(f) > v(f)$
- This would contradict $v(f') \leq c(A, B)$, and therefore f is maximum
- Also assume in contradiction there exists a cut (A', B') with $c(A', B') < c(A, B)$
- Again this would contradict $v(f) \leq c'(A, B)$, and therefore c is minimum

□

Improving the running time

- Recall that the flow f is incremented by the smallest value of $c_f(u, v)$ on the $s - t$ path in the residual graph G_f (which we called Δ)
- Therefore it is useful to find a path with a high Δ
- If $C = \sum_{e \text{ out from } s} c(e)$ is a huge number this is particularly important
- In this and many other situations it is not worthwhile to find the optimal value of a parameter (here Δ) used to speed up an algorithm
- We can look for paths with $\Delta \geq C/2^i$ for $i = 0, 1, 2, \ldots$
- Another idea is to search for paths with the fewest number of edges
- We will instead soon look at a completely different approach: preflow-push

Bipartite graph matching



- In a bipartite graph the nodes can be partitioned in two sets
- No edge between nodes in the same set
- We seek a matching of blue and red nodes
- Blue can be employees and red can be tasks
- An edge says somebody can perform a task
- We want to find a maximal matching
- We want as many tasks performed as possible
- Quiz: how can we solve this with Ford-Fulkerson?

The preflow-push maximum network flow algorithm

- Variants of the preflow-push maximum network flow algorithm, we will study next, are the fastest algorithms for finding the maximum network flow
- For brevity we simply call it the preflow-push algorithm
- The preflow-push algorithm also uses of the residual graph
- Instead of maintaining a valid flow which satisfies both the conservation constraint and the capacity constraint, it uses a weaker type of flow which only satisfies the capacity constraint
- The weaker flow is called a **preflow**
- At algorithm termination, the preflow will have become a valid flow
- In addition, it uses a **height** function for each node

The preflow

- For each edge $e \in E$ we have $0 \leq f(e) \leq c(e)$
- Thus the capacity constraint is always satisfied
- Instead of the conservation constraint, a node $u \neq s$ is allowed to have more incoming flow than outgoing
- Thus for each node $u \in V - \{s\}$ we have

$$\sum_{e \text{ into } u} f(e) \geq \sum_{e \text{ out from } u} f(e)$$

- The **excess preflow** of a node u is

$$e_f(u) = \sum_{e \text{ into } u} f(e) - \sum_{e \text{ out from } u} f(e)$$

- Only s has a negative excess preflow

The height function

- There is a height function $h : V \to \mathbb{N}$
- $h(s) = n$
- $h(t) = 0$
- For s and t the heights cannot change and for other nodes they start at 0 and can increase
- The preflow on an edge (u, v) can only increase if $h(u) = h(v) + 1$
- As we will see, $0 \leq h(u) \leq 2n - 1$ for $u \neq s$

Compatible h and f

- Recall: $(v, w) \in E_f$ if the flow on (v, w) can be increased
- The height function h and a preflow f are **compatible** if the following conditions are satisfied:
    1. $h(s) = n$ and $h(t) = 0$
    2. For all edges $(v, w) \in E_f$ we have $h(v) \leq h(w) + 1$, or $h(w) \geq h(v) - 1$
- In G_f a simple path $p = v_1, v_2, \ldots, v_k$ we have v_i at most one higher than v_{i+1}
- Consider a simple path $v_0, v_1, v_2, \ldots, v_k$ in G_f with $v_0 = s$
- $h(v_0) = n$, $h(v_1) \geq n - 1$, $h(v_2) \geq n - 2$, ..., $h(v_k) \geq n - k$.
- In Ford-Fulkerson we look for an $s - t$ path
- Quiz: can there be an $s - t$ path in G_f here?

Preflow paths in G_f

Lemma

There can be no $s - t$ *path in* G_f *for a preflow* f *compatible with* h

Proof.

- Assume in contradiction there is a simple $s - t$ path p in G_f
- Let $p = v_0, v_1, v_2, \ldots, v_k$, i.e. $s = v_0$ and $t = v_k$
- Then $h(t) \geq n - k$ and since $h(t) = 0$ it must be the case that $k = n$, and that the length of p is n.
- This path cannot be simple. A contradiction.

□

Finding a maximum flow using h

Lemma

If an $s-t$ flow f is compatible with a height function h, then f is a maximum flow.

Proof.

- Recall: if there is an $s-t$ flow f in G such that there is no $s-t$ path in G_f then f has the a maximum flow in G
- Since a flow f also satisfies the conservation constraint, f is a preflow.
- Therefore for a flow f compatible with a height function h, there cannot be an $s-t$ path in G_f (from previous slide)
- And no s-t path in G_f means f is maximal

☐

- If we can transform a preflow to a flow compatible with a height function h, we have found a maximal flow

Overview

- We start with a preflow f which, as we will see, is not a flow since it violates the conservation constraint
- The preflow f is compatible with the height function h and thus there is no $s - t$ path in G_f
- We will maintain the preflow so it remains compatible with an h
- The preflow will be modified until it becomes a flow f which then will be a maximum flow
- Instead of maintaining valid but suboptimal flows which are improved, we will work towards a valid optimal flow
- The height of a node $u \in V - \{s, t\}$ can be at most $2n - 1$

Initial preflow and height function

- Each edge (s, u) is assigned the initial preflow $f(s, u) = c(s, u)$
- For all other edges $f(u, v) = 0$
- $h(s) = n$ and $h(u) = 0$ for every node $u \neq s$

push

- Three conditions must be satisfied for a push:
  1. $e_f(v) > 0$
  2. $h(v) > h(w)$
  3. $(v, w) \in G_f$

```
procedure push(f, h, v, w)
    assert $e_f(v) > 0$ and $h(v) > h(w)$ and $(v, w) \in G_f$
    $e \leftarrow (v, w)$
    if $e$ is a forward edge then
        $\delta \leftarrow \min(e_f(v), c(e) - f(e))$
        increase $f(e)$ by $\delta$
    else
        $e \leftarrow (w, v)$
        $\delta \leftarrow \min(e_f(v), f(e))$
        decrease $f(e)$ by $\delta$
```

relabel

- The purpose of a relabel is to increase the height of a node
- It is done when the node has excess flow but nowhere to push it due to neighbors have too high height

```
procedure relabel(f, h, v)
    assert $e_f(v) > 0$ and for all edges $(v, w) \in E_f$ we have $h(w) \geq h(v)$
    $h(v) \leftarrow h(v) + 1$
```

The preflow push algorithm

```
function $preflow\_push(G, s, t)$
    $h(s) \leftarrow n$
    for each node $u \neq s$ do $h(u) \leftarrow 0$
    for each edge $(s, v)$ do $f(s, v) \leftarrow c(s, v)$
    for each edge $(u, v)$ such that $u \neq s$ do $f(u, v) \leftarrow 0$
    while there is a node $v \neq t$ with $e_f(v) > 0$ do
        if there is a node $w$ such that $h(v) > h(w)$ and $(v, w) \in G_f$ then
            $push(h, f, v, w)$
        else
            $relabel(h, f, v)$
    return $f$
```

Correctness of the preflow-push algorithm

- Initially the preflow f and height function h are compatible
- Each push satisfies the capacity constraints due to how the δ is calculated
- Each relabel increases the height of a node v by one.
- This could violate the compatibility of f and h
- The relevant condition for compatibility is:

$$\textit{For all edges } (v, w) \in E_f \textit{ we have } h(v) \leq h(w) + 1$$

- If it is the case $h(v) > h(w)$ then a push and not a relabel is performed
- In the other case, $h(v) \leq h(w)$ the height of v is incremented by one, and this still satisfies the condition
- Therefore after a relabel, f and h remain compatible

Correctness of the preflow-push algorithm

- The algorithm terminates when only $e_f(t) > 0$
- When this happens the preflow is a flow and as proved earlier, this is a maximum flow

Paths to s in G_f

Lemma

A node v with $e_f(v) > 0$ has a path in G_f to s

Proof.

- Let A be the set of nodes with a path to s in G_f, and $B = V - A$.
- $s \in A$
- An edge (v, w) with $v \in A$ and $w \in B$, (v, w) cannot have flow since that would create a backward edge (w, v) in G_f so that then $w \in A$, which contradicts the assumption that $w \in B$
- The sum of excess flow of nodes in B is nonnegative (since only $s \in A$ has negative excess flow) and can be written:

$$0 \leq \sum_{w \in B} e_f(w) = \sum_{w \in B} f^{\text{in}}(w) - \sum_{w \in B} f^{\text{out}}(w)$$

☐

Paths to s in G_f

Proof.

- From the previous slide

$$0 \leq \sum_{w \in B} e_f(w) = \sum_{w \in B} f^{\text{in}}(w) - \sum_{w \in B} f^{\text{out}}(w)$$

- Considering edges which contribute to the above sums we have different cases.
- For an edge (u, v) with $u, v \in B$ these cancel.
- For an edge (u, v) with $u \in A$ and $v \in B$ its flow is 0 as shown on the previous slide.
- Only edges (u, v) with $u \in B$ and $v \in A$ remain

$$0 \leq \sum_{w \in B} e_f(w) = - \sum_{w \in B} f^{\text{out}}(w)$$

□

Paths to s in G_f

Proof.

- From the previous slide

$$0 \leq \sum_{w \in B} e_f(w) = -\sum_{w \in B} f^{\text{out}}(w)$$

- But flows are nonnegative which implies they are all zero.
- Therefore, all nodes with excess are in the set A and the claim follows.

$\square$

Maximum height of a node and relabel operations

Lemma

$h(u) \leq 2n - 1$

Proof.

- A height is increased by a relabel operation, which is applicable to nodes other than s and t
- As was proved by the previous lemma, a node u with $e_f(u) > 0$ has a simple path p to s in G_f
- The length of this path is at most $n - 1$.
- For a compatible h and f the heights on this path decrease at most by the length of the path, i.e. at most $n - 1$
- Since $h(s) = n$ we have $h(u) - h(s) \leq n - 1$ i.e. $h(u) \leq 2n - 1$
- Since each node can have height at most $2n - 1$ and there are n nodes, the number of relabel operations is less than $2n^2$

□

Push operations

- A push operation increases the flow along an edge (v, w)
- As much excess flow $e_f(v)$ as possible is added to $f(v, w)$
- There are two limits:
  1. At most $e_f(v)$ can be used since excess flow can never be negative
  2. The capacity of the edge cannot be exceeded
- A push at an edge (v, w) is **saturating** if the only limit was edge capacity:
  1. (v, w) is a forward edge and $\delta = c(v, w) - f(v, w)$, and
  2. (v, w) is a backward edge and $\delta = f(v, w)$.
- Note *only*: we assume v still has excess flow after a saturating push
- All other push operations are **nonsaturating** and were limited by the amount of excess flow for v
- After a nonsaturating push, v no longer has any excess flow: $e_f(v) = 0$

Saturating push operations

Lemma

The number of saturating push operations is less than $2nm$.

Proof.

- Consider any two nodes v and w such that they have an edge (v, w)
- At a saturating push at the edge (v, w) we have $h(v) = h(w) + 1$
- Before a new push at the same edge, the height of w must be increased by 2. Since the height of any node always is less than $2n - 1$, any node can increase by 2 at most $n - 1$ times.
- Counting both v and w the number of saturating pushes between them is less than $2n$.
- Since there are m edges the total number of saturating pushes is less than $2nm$

□

Nonsaturating push operations

Lemma

The number of nonsaturating push operations is at most $4n^2m$.

Proof.

- This lemma is proved using the **potential function** method.
- For a given preflow f and height function h we define

$$\Phi(f, h) = \sum_{v : e_f(v) > 0} h(v)$$

- Initially $\Phi(f, h) = 0$ since $h(s) > 0$ but $e_f(s) < 0$
- $\Phi(f, h) \geq 0$ since no negative heights

□

Approach to counting nonsaturating push operations

- Both relabel and saturating push increase Φ
- We can find the max value of Φ
- Nonsaturating push decrease Φ
- If we find the minimal decrease for each nonsaturating push, we can calculate an upper bound on their number (max Φ / minimal decrease)

Effects on $\Phi(f, h)$ of different operations

Proof.

- A relabel increases $\Phi(f, h)$ by one and $2n^2$ relabels so at most $+2n^2$
- A saturating push (v, w) may increase $e_f(w)$ from 0 and therefore increase Φ by at most $2n - 1$, and with at most $2nm$ saturating push operations, at most $+4n^2m$
- After a nonsaturating push Φ is reduced by $h(v)$ since v no longer has any excess flow
- After that Φ is incremented by $h(w)$ if $e_f(w) = 0$ before the push and not incremented if w already had excess flow
- So at least -1 by each nonsaturating push but possibly reduced more
- $\Phi(f, h) \geq 0$ so at most $4n^2m$ nonsaturating pushes

□

Maximum flow running times

Ford-Fulkerson	$O(Cm)$
Preflow-push	$O(4mn^2)$

- Both relabel and push take constant time
- The theoretical limitation of preflow-push is the number of nonsaturating push operations
- The preflow-push algorithm has $O(mn^2)$ nonsaturating push operations
- In a dense graph this is $O(n^4)$
- It can be shown that if we always take the node v with $e_f(v) > 0$ and maximum height $h(v)$ the number of nonsaturating push operations is at most $4n^3$

Parallel preflow push

- EDAN26 multicore programming in LP1
- IBM POWER8 computer with 80 hardware threads
- Lots of synchronizations
- Two phases
- I. Henckel and D. Söderberg: if the sum of capacities in to t is less than the sum out from s and the graph is undirected, it can be useful to let s and t switch roles.
- Nils Ceberg: the algorithm can terminate when $-e_f(s) = e_f(t)$ which is especially useful in a distributed implementation (such as with Scala/Akka) so no need to maintain a set of nodes with $e_f(v) > 0$ and check that it is empty as in the sequential algorithm
- Called Ceberg preflow-push termination (since 24/4 2023)