15/20

Advanced Algorithms UE 6

by Maarten Behn
Group 5

Exercise 6.1 (Flows in strongly connected networks) (10 Points)

> Let $\mathcal{N} = (V, E, c, s, t)$ be an s-t network with the following properties:
>
> 1. The sub graph of (V, E) induced by $V \backslash s, t$ is strongly connected. That is, for every $u, v \in V \backslash s, t$ there is a directed path from u to v in $(V \backslash s, t, E)$.
> 2. We have that $\sum_{e \in \delta^+(s)} c(e) = \sum_{e \in \delta^-(t)} c(e) =: B$.
> 3. For all $e \in E \backslash (\delta^+(s) \cup \delta^-(t))$, we have $c(e) = 2B$.
>
> a) Prove that there exists a maximum flow f in $\mathcal{N}$ with $val(f) = B$.
> b) Moreover, give an algorithm that computes such a flow in time $O(|E|)$.

a)

10/10

The maximum flow $\mathcal{N}$ is never larger than B. -> 2. Statement ✓

Assumption

For every $\mathcal{N}$ there is a Network $\mathcal{N}' := (V', E', c', s, t)$
$V' := \{s, t, v'\}$
where we merge all vertices $v \in V \backslash \{s, t\}$ into v'.
$E' := \{(s, v'), (v', t)\}$ this is not a sentence? what do you mean?

$c'((s,v')) = c'((v',t)) = B$
therefore the maximum flow of N' is B

The maximum flow $\mathcal{N}'$ is never smaller than the the maximum flow of $\mathcal{N}$. why?

Proof

2. states that the maximum incoming and outgoing flow into $V\backslash\{s,t\}$ is B
3. states that $\forall v,u \in V\backslash\{s,t\}\ \ v \neq u$ have an edge (v,u) with capacity $2B$
   therefore
   $c((s,v)) \leq c((v,u))$
   $c((u,t)) \leq c((v,u))$

This means that for every s-t-path an edge (v,u) will never be the bottleneck edge.

because
$\sum_{e\in\delta^+(s)} c(e) = \sum_{e\in\delta^-(t)} c(e) = B < c((v,u)) = 2B$
we can merge all vertecies $v \in V\backslash\{s,t\}$ into v' and the maximum flow will not change.

$\Rightarrow$ The maximum flow of $\mathcal{N}$ is B.

b)

Idea

1. Choose one Vertex in $q \in V\backslash\{s,t\}$.
2. Push all the flow from s to q.

3. Push all the flow from q to t.

Algorithm

Input: $\mathcal{N} := \{V, E, c, s, t\}$, an empty flow f

1. Take $q \in V \backslash \{s, t\}$
2. Perform DFS from q to all neighbors of s on the incoming edges of each Vertex.
   When hitting a neighbor n_s of s set $f((s, n_s)) = c((s, n_s))$
   When going back up an incoming edge of (v, u) of Vertex u
   set $f((v, u)) = \sum_{e \in \delta^-(v)} f(e)$.
3. Perform DFS from q to all neighbors of t on the outgoing edges of each Vertex.
   When hitting a neighbor n_t of t set $f((n_t, t)) = c((n_t, t))$
   When going going back up an outgoing edge (v, u) of Vertex v
   set $f((v, u)) = f((v, u)) + \sum_{e \in \delta^+(u)} f(e)$.

Proof the Algorithm computes a flow of B

With the DFS in the second step we take all the flow from s and sum it up the tree till it is all pushed into q.
With the DFS in the third step we take that flow from q and split it down the DFS into neighbors of t so that all edges to t are satisfiyed.
Both DFS can at most push an flow of B over an edge.
An edge in $V \backslash \{s, t\}$ can never be a bottleneck edge because they have all a capacity of $2B$.

Runtime

1. $O(1)$
2. $O(|E|)$

3. 2. $O(|E|)$

$$\Rightarrow O(2|E|+1) = O(|E|)$$

Exercise 6.2 (The FIFO Preflow-Push Algorithm)

A FIFO (first-in first-out) queue Q is a data structure where elements are added (push) to the end of the queue and removed from the front (pop). Consider the following variant of the generic Preflow-Push algorithm:

Algorithm 1: The FIFO Algorithm

Input: An s-t-network $\mathcal{N} = (V, E, c, s, t)$

```
Init(N)
FIFO queue Q ← neighbors of s (all are active now)
while Q is not empty do
    u ← pop(Q)
    Discharge(u)
    Push all active vertices v ∈ V \ Q into Q.
return f
```

Advanced Algorithms UE 6-1748183803522.png

Init

Function `Init(`$\mathcal{N}$`)`**:**

- $d(s) = n, d(v) = 0$ for all $v \in V \setminus \{s\}$
- $f := 0$
- **for** $v \in V$ with $(s, v) \in E$ **do**
  - $f(s, v) \leftarrow c(s, v)$

Preflows-1747651077894.png

Discharge

Function `Discharge(`u`)`**:**

- **while** u is active and at least one edge $(u, v) \in E_f$ is eligible **do**
  - `Push(u,v)`
- **if** u is active **then**
  - `Relabel(`u`)`

Advanced Algorithms UE 6-1748181667161.png

Push

Function `Push(u, v)`:
  Precondition : $(u, v) \in E_f$, u is active, and (u, v) is eligible
  if $(u, v) \in E$ **then**
    $f(u, v) \leftarrow f(u, v) + \min\{e_f(u), c_f(u, v)\}$
  if $(u, v) \in \overleftarrow{E}$ **then**
    $f(v, u) \leftarrow f(v, u) - \min\{e_f(u), c_f(u, v)\}$

Preflows-1747650858745.png

Relabel

Function `Relabel(u)`:
  Precondition : u is active and no edge out of u is eligible
  $d(u) \leftarrow d(u) + 1$

Preflows-1747650866913.png

Definition (Eligible edges and active vertices)

An edge $(u, v) \in E_f$ is eligible w.r.t. a preflow f and a height d if

1. $e_f(u) > 0$, and
2. $d(u) = d(v) + 1$.

Moreover, we call a vertex $v \in V \setminus \{s, t\}$ active if $e_f(v) > 0$.

Preflows-1747650845491.png

Prove that the FIFO Algorithm computes a maximum flow in time $O(n^3)$, where $n = |V|$
To do so, you can use the following hints:

Split the execution of the algorithm into phases, where the Phase 1 is the processing of the neighbors of s, Phase 2 is the processing of those vertices that are added to Q during Phase 1, Phase 3 is the processing of those vertices that are added to Q during Phase 2, and so on.

Your goal is to show that the number of non-saturating Push operation is in $O(n3)$ (why?).
Such a bound follows from the number of phases (why?).

> To analyze the number of phases, use the potential function $\Phi = max\{d(v)|visactive\}$. Moreover, distinguish between phases without any relabeling and phases with at least one relabeling.
>
> So für den Fail noch ein Tipp zum bounden der Anzahl an Phases: In einer Phase ohne Relabelings wird jeder zu Beginn der Phase aktive Knoten inaktiv. Da alle Pushes downward sind, bedeutet das, dass die Höhe der neu aktiv gewordenen Knoten wie ist?
>
> Und Phasen mit mindestens einem Relabel kann es höchstens wie viele geben?

Proof

From Lecture

The Corollary states that the number of Relabel operations is at most $2n^2 \Rightarrow O(n^2)$
The Lemma states that the number of saturating pushes is at most $2nm$
m is at most n^2 therefore it is bound by $O(n^3)$. √

5/10

Goal

So we need to show that number of non-saturating Push operation is also $O(n^3)$
to show that the Algorithm has a runtime of $O(n^2 + 2n^3) = O(n^3)$.

Passes Idea

We can split the Algorithm into passes (phase) $P_0, P_1, \ldots, P_k$ $k \leq n^2$
The first pass is all the iterations where the initial neighbors of s are popped.
The next pass is all the iterations where all the vertices are popped

that where added in the last pass.
This is possible because Q is a FIFO Queue.
For an example how the interactions are split, see the example below.

Maximum number of Iterations

Yes, there are at most n items in the FIFO queue, but how much time does the discharge function take?

Every pass can contain at most n iterations.
So we have to prove that there are at most n^2 passes.

Pass with no relabeling operation

If a pass P_i contains no relabeling operation then all processed vertices in the pass are inactive after the pass and will therefore not be processed in the nex pass P_{i+1}.

A push operation pushes from vertex v to a vertex u is only performed if $d(v) = d(u) + 1$
$\Rightarrow max(d(P_i)) = max(d(P_{i+1})) + 1$ explain your notation
The maximum height a pass can be at most $2n^2$ because the maximum possible height of a vertex is $2n^2$
$max(d(P_i)) \leq 2n^2$
So there can be at most $2n^2$ passes containing no relabeling operation. $\Rightarrow O(n^2)$

You are missing explanation about the runtime of the while loop in the discharge function

Pass with relabeling operation

The number of passes with a relabeling operation is at most $2n^2$ because there can be at most $2n^2$ relabel operations. $\Rightarrow O(n^2)$

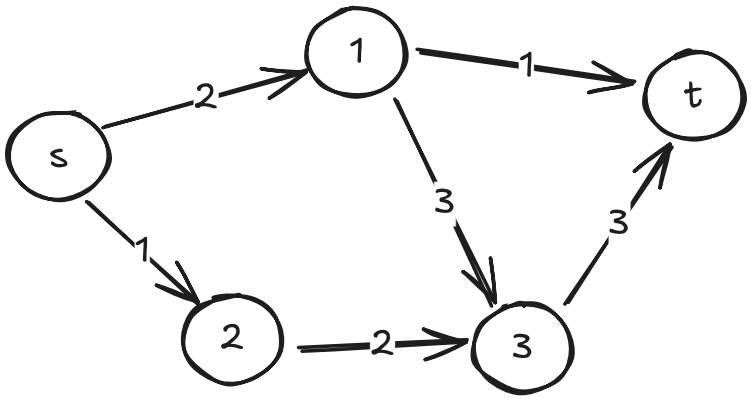
Conclusion

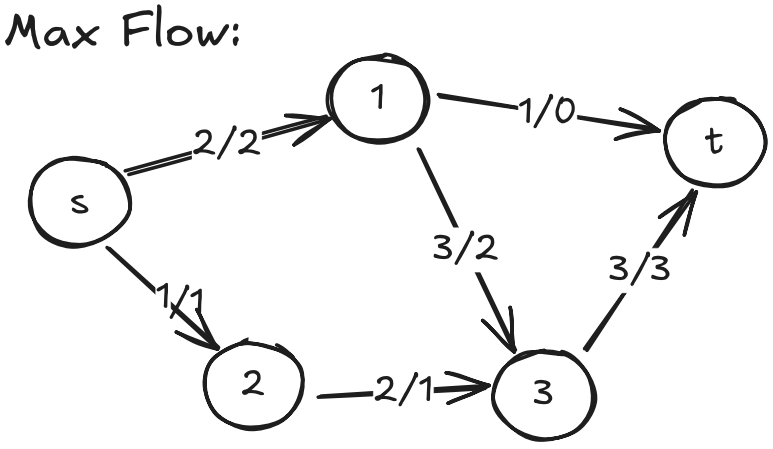
Therefore the number of Iterations is bound by $O(n \cdot 2n^2) \Rightarrow O(n^3)$
If the number of Iterations is bound by $O(n^3)$

then the number of non-saturating Push operation is also bound by $O(n^3)$.

$\Rightarrow$ the FIFO Algorithm computes a maximum flow in $O(n^3)$.

Example to understand the algorithm





Init:

$Q := (1, 2)$
$d := \{(s, 5), (1, 0), (2, 0), (3, 0)(t, 0)\}$
$f := 0$
$f(s, 1) \leftarrow 2$
$f(s, 2) \leftarrow 1$
$active : \{1, 2\}$
$excess : \{(s, 0), (1, 2), (2, 1), (3, 0)(t, 0)\}$

Phase: 1

1:

$u \leftarrow 1$
$Q \leftarrow (2)$

Discharge:

1 is active
$(1, 3)$ is not eligible
$(1, t)$ is not eligible

RELABEL

$d \leftarrow \{(s, 5), (1, 1), (2, 0), (3, 0)(t, 0)\}$

Push all active vertices into Q

$Q \leftarrow (2, 1)$

2:

$u \leftarrow 2$
$Q \leftarrow (1)$

Discharge:

2 is active
$(2, 3)$ is not eligible

RELABEL

$d \leftarrow \{(s, 5), (1, 1), (2, 1), (3, 0)(t, 0)\}$

Push all active vertices into Q

$Q \leftarrow (1, 2)$

Phase: 2

3:

$u \leftarrow 1$
$Q \leftarrow (2)$

Discharge:

1 is active
$(1, 3)$ is eligible

PUSH

$e(1) = 2$
$c(1, 3) = 3$
$f(1, 3) \leftarrow 2$
$excess : \{(s, 0), (1, 0), (2, 1), (3, 2)(t, 0)\}$
$active : \{2, 3\}$

$(1, t)$ is not eligible

Push all active vertecies into Q

$Q \leftarrow (2, 3)$

4:

$u \leftarrow 2$
$Q \leftarrow (3)$

Discharge:

2 is active
$(2, 3)$ is not eligible

RELABLE

$d \leftarrow \{(s, 5), (1, 1), (2, 2), (3, 0)(t, 0)\}$

Push all active vertecies into Q

$Q \leftarrow (3, 2)$

Phase: 3

5:

$u \leftarrow 3$
$Q \leftarrow (2)$

Discharge:

3 is active
$(3, t)$ is not eligible

RELABEL

$d \leftarrow \{(s, 5), (1, 1), (2, 2), (3, 1)(t, 0)\}$

Push all active vertices into Q

$Q \leftarrow (2, 3)$

6:

$u \leftarrow 2$
$Q \leftarrow (3)$

Discharge:

2 is active
$(2, 3)$ is eligible

Push

$e(2) = 1$
$c(2, 3) = 2$
$f(1, 3) \leftarrow 1$
$excess : \{(s, 0), (1, 0), (2, 0), (3, 3)(t, 0)\}$
$active : \{3\}$

Phase: 4

7:

$u \leftarrow 3$
$Q \leftarrow ()$

Discharge:

3 is active
$(3, t)$ is eligilbe

Push

$e(3) = 3$
$c(3, t) = 3$
$f(1, 3) \leftarrow 3$
$excess : \{(s, 0), (1, 0), (2, 0), (3, 0)(t, 3)\}$
$active : \{\}$

Push all active vertices into Q

$Q \leftarrow ()$

done

Summary

- Phase 1
  - relabel 1
  - relabel 2
- Phase 2:
  - push 1
  - relabel 2
- Phase 3
  - relabel 3
  - push 2
- Phase: 4
  - push 3