15.5 / 20

Advanced Algorithms UE 5

by Maarten Behn
Group 5

Exercise 5.1 (Ford-Fulkerson with thickest paths) (5 Points)

> Let $\mathcal{N}$ be an integer s-t-network and f be a feasible flow in $\mathcal{N}$.
> A thickest s-t-path in $\mathcal{N}_f$ is an s-t-path with a maximal bottleneck capacity.
>
> (a) Prove that the number of augmentations needed by the variant of the Ford-Fulkerson algorithm that always augments along a thickest s-t-path in $\mathcal{N}_f$ for a network $\mathcal{N}$ with n vertices, m edges, and integer capacities in $\{0, 1, \ldots, C\}$ is in $O(m\ log(nC))$.
> *Hint: use Exercise 4.3 and the fact that* $(1 - \frac{1}{x})^x \leq \frac{1}{e}$ *for all* $x > 1$.
>
> (b) Describe an efficient algorithm to find a thickest s-t-path in $\mathcal{N}_f$, and analyze its running time.
> *Hint: recall Dijkstra's algorithm.*

a)

Let f^* be a maximum Flow in $\mathcal{N}$.

3/6 From a) we can say that a flow can be decomposed into the sum k flows.

$$f^* = \sum_{i=0}^{k} f_i$$

The Ford-Fulkerson algorithm chooses always the "thickest" s-t-path. why?
Let f_i be the flow of of the s-t-path from the ith iteration.

$$R_i := |f^*| - \sum_{j=0}^{i} |f_j|$$

We know $k \leq m$ therefore

$|f_i| \geq \frac{R_i}{m}$ why?

$$R_i - |f_i| \leq R_i - \frac{R_i}{m} = R_i\left(1 - \frac{1}{m}\right)$$

Thus R gets per iteration reduced by at least $\left(1 - \frac{1}{m}\right)$.

$R_i \leq |f^*|\left(1 - \frac{1}{m}\right)^i$

The algorithm stops when $R_i < 1$.

$|f^*|\left(1 - \frac{1}{m}\right)^i < 1$

$\iff \left(1 - \frac{1}{m}\right)^i < \frac{1}{|f^*|}$

$\iff i \cdot log(\left(1 - \frac{1}{m}\right)) < log(1) - log(|f^*|) = -log(|f^*|)$

$\iff i > \frac{-log(|f^*|)}{log((1-\frac{1}{m}))} \geq \frac{-log(|f^*|)}{\frac{1}{m}} = i \in O(m \cdot log(|f^*|))$

what are you doing in the last step?

i>i seems like a contradiction?

$|f^*|$ can be at most $n \cdot C$

$\Rightarrow O(m \cdot log(nC))$

b)

The original Dijkstra's algorithm written in pseudo code:

```
Input: G = (V, E), w(u, v) ≥ 0, s

Inital:
    ∀ v ∈ V: dist(v) := ∞
    dist(s) := 0
    pred(v) := undefiniert
    Q := min-priority queue storing (v, dist(v))

Algorithm:
```

```
    while Q ≠ ∅ do
        u := Get-Min(Q)
        for each (u, v) ∈ E do
            if dist(v) > (dist(u) + w(u, v)) then
                dist(v) := dist(u) + w(u, v)
                pred(v) := u
                Set(Q, v, dist(v))

Output:
    dist(v): length of path
    pred(v): linked list of reverse path
```

We can modify the algorithm
by using the maximum bottelneck capacity as the value instead of the length.
Also we will use a max-priority queue insetad of an min-priority queue to find the path with the maximum bottleneck capacity.

```
Input: N_f = (V, E, c, s, t)

Inital:
    ∀ v ∈ V: bottleneck(v) := 0
    bottleneck(s) := ∞
    pred(v) := undefiniert
    Q := max-priority queue storing (v, bottleneck(v))

Algorithm:
    while Q ≠ ∅ do
        u := Get-Max(Q)
        for each (u, v) ∈ E do
            cap := c(u, v)
            new_b := min(bottleneck(u), cap)
            if new_b > bottleneck(v) then
```

```
                 bottleneck(v) := new_b
                 pred(v) := u
                 Set(Q, v, bottleneck(v))

Output:
    bottleneck(t): bottleneck capacity of the s-t-path
    pred(v): linked list of reverse s-t-path
```

4/4

Using a binary heap as priority queue,
Dijkstra's algorithm has a runtime of $O(m \log n)$.
The modifications don't change anything that effects the runtime,
therfore the modified algorithm has the same runtime.

Exercise 5.2 (Blocking vs Maximum) (5 Points)

> Let $\mathcal{N}$ be an s-t-network.
> Prove or disprove the following statements.
> (a) Every maximum flow in $\mathcal{N}$ is also a blocking flow in $\mathcal{N}$.
> (b) Every blocking flow in $\mathcal{N}$ is also a maximum flow in $\mathcal{N}$.

a)

A blocking flow is a flow that saturates at least one edge on every s-t-path.
So every residual Graph where there is no path from s to t, is considered blocking.

1.5/3

this statement is incorrect (see the green path that i drew in the residual graph of your example from (b))

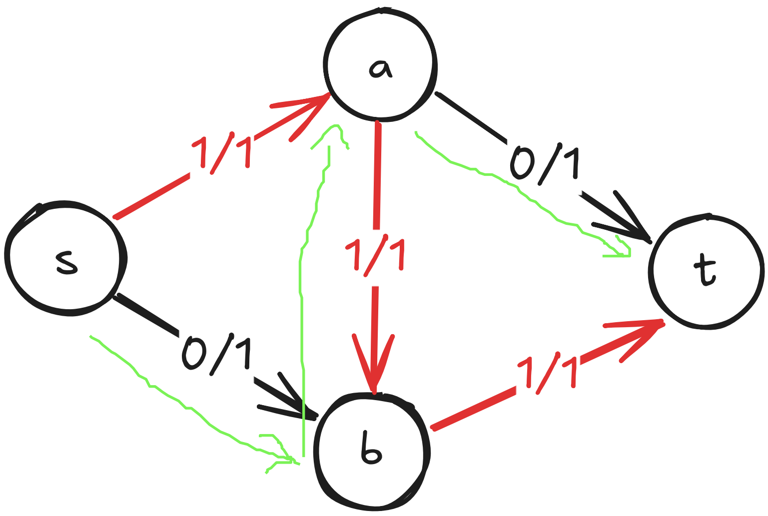
A maximum Flow is a flow that has no augmenting s-t-path to increase the flow.
So in the residual Graph there is no path from s to t.

Therefore every maximum Flow is a blocking flow.

b)

No, here is an example:

2/2



This flow is blocking, because all three s-t-paths $s - a - t$, $s - b - t$ and $s - a - b - t$ have an saturated edge.

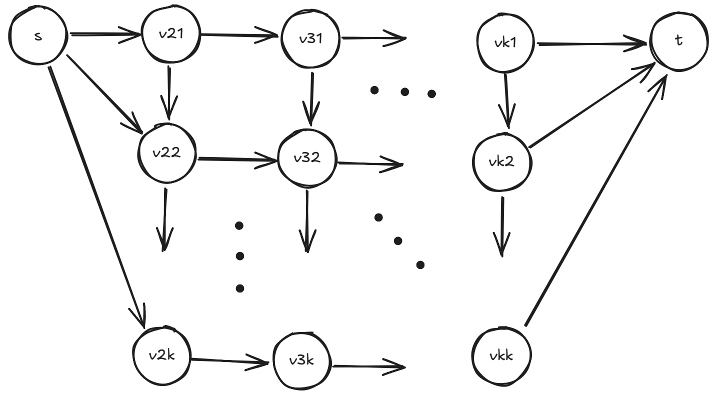
But it is not maximum.
A maximum flow in this Graph has a value of two.
$s - a - t$ and $s - b - t$

Exercise 5.3 (Number of Iterations of Dinitz Algorithm) (5 Points)

5/5

> For every natural number k, describe an s-t-network on which Dinitz algorithm needs at least k iterations, i.e., computes at least k successive blocking flows.
> Explain why at least k iterations are needed

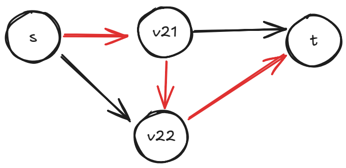
Construct the graph like this:

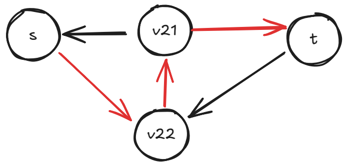


Blocking flows a chosen like this:

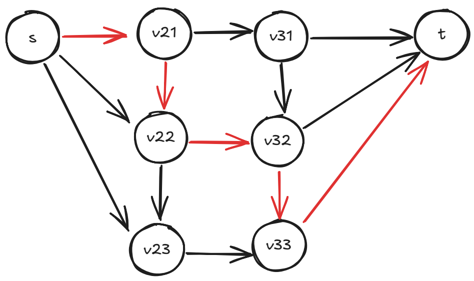
k=1

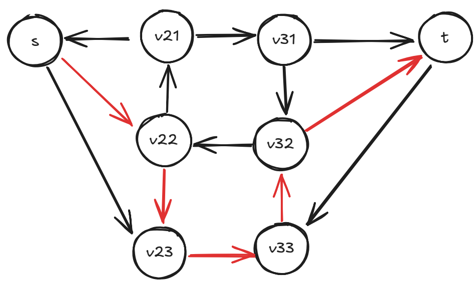
k=2

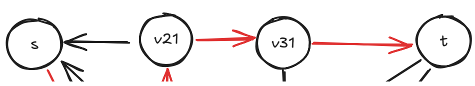




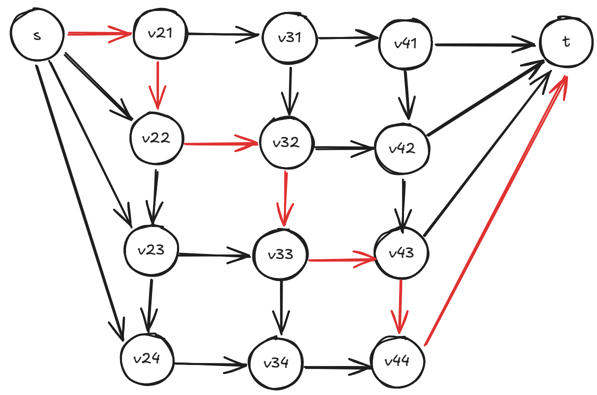
k=3

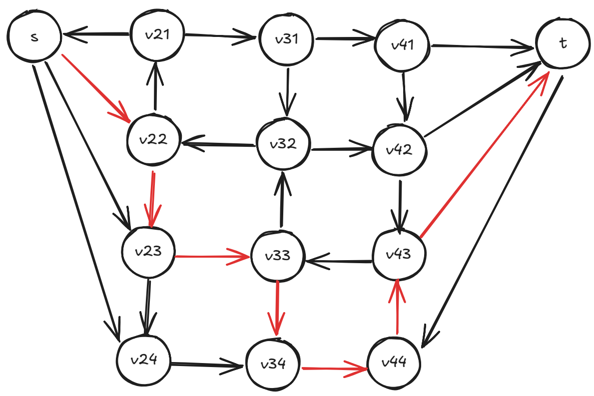


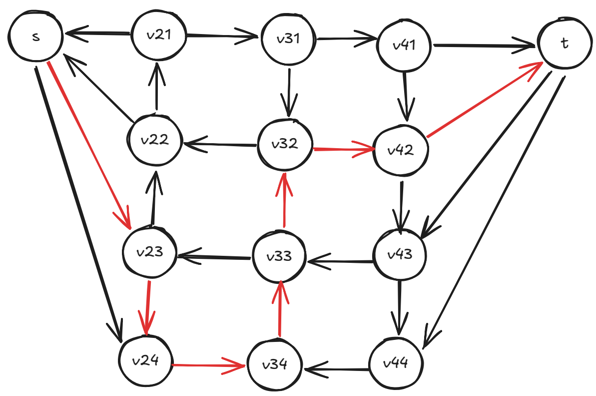


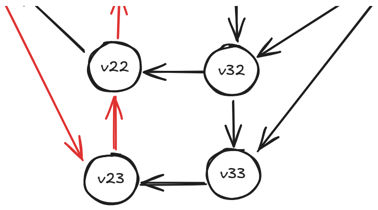


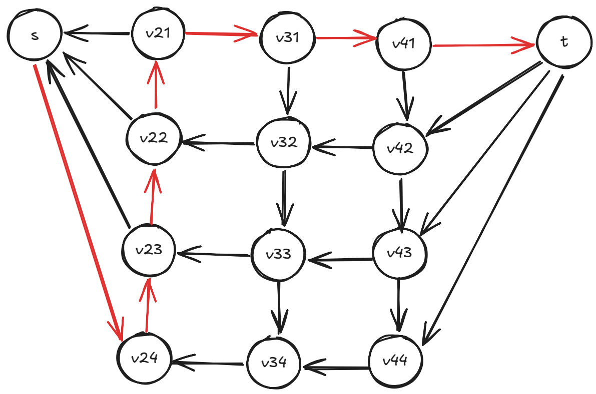
k=4











All edes have a capacity of one.
The maximum flow of these graph setup is k.
Because there are k disjunkt paths from s to t:

$$\begin{array}{l}
s - v_{21} - v_{31} - \ldots - v_{k1} \quad - t \\
s - v_{22} - v_{32} - \ldots - v_{k2} \quad - t \\
\qquad\quad \vdots \qquad \ddots \qquad \vdots \\
s - v_{2k} - v_{3k} - \ldots - v_{kk} \quad - t
\end{array}$$

Note the Graph has for every k k rows but only $k-1$ collums of v vertecies.

In this Graph setup it is possible to always choose a blocking flow that only adds one to the flow sum.
Therfore it takes k iterations to compute the maximum flow of k.

The blocking flow is choosen after this pattern:

1. Choose the first outgoing edge from s. (top to bottom)
   $s - v_{2a}$ where $!\exists (s, v_{2b}) \in E_f \ \ b < a$
2. Choose the edges in a stair pattern down (starting with a vertical edge)
   $v_{2a} - v_{2a+1} -$
   $v_{3a+1} - v_{3a+2} -$
   $\vdots$
   $v_{l-1k-1} - v_{l-1k} - v_{lk}$

3. Choose every edge straight up till it is possible go right.
   $v_{lk} - v_{lk-1} - \ldots - v_{lj}$ where $\exists(v_{jl}, v_{jl+1}) \in E_f$
4. Choose all horizontal edges till hitting t
   $v_{l+1j} - v_{l+2j} - \ldots - t$

The s-v edge and the stair case pattern block all other s-t-paths.
Its always possible to find a path fom v_{lk} to t because the previous stair case pattern produce vertical paths up to a j where it is possible to go horizontaly over to t.