Advanced Graph Algorithms and Optimization Spring 2023

Classical Algorithms for Maximum Flow

R. Kyng & M. Probst Gutenberg *Problem Set 7 — Thursday, April 27th*

The exercises for this week will not count toward your grade, but you are highly encouraged to solve them all. This exercise sheet has exercises related to week 7. We encourage you to start early so you have time to go through everything.

To get feedback, you must hand in your solutions by 23:59 on May 4. Both hand-written and LaTeX solutions are acceptable, but we will only attempt to read legible text.

Exercise 1: Implementing Field Preservations for Cut-Link Tree Rotations

In Chapter 14, Section 14.3, we described the operations PLink(u, v) and PCut(u, v) by doing $O(\log n)$ tree rotations (in expectation). However, we omitted the details. Here we ask you to give the pseudo-code for the operation PTreeRotation(v, w) where it is assumed that on input v is the parent of w and the operation manipulates the tree over the path such that v and w change position as described in the script. For simplicity, you are allowed to assume that w is the left child of v in the treap $\mathcal{T}_p$, and that the nodes $left_{\mathcal{T}_p}(v), right_{\mathcal{T}_p}(v), left_{\mathcal{T}_p}(w), right_{\mathcal{T}_p}(w)$ exist. Accompany your pseudo-code with a brief analysis that confirms that the run-time is indeed $O(1)$.

Solution The pseudocode is given in algorithm 1. Clearly, all updates in the algorithm (changing pointers and updating Δmin and $\Delta cost$ values can be done in $O(1)$ hence the total runtime is $O(1)$.

Exercise 2: Max Flow in directed Graphs with Edge Capacities

Consider directed graph $G = (V, E, \boldsymbol{c})$ with arbitrary capacities $\boldsymbol{c} \geq \mathbf{0}$.

Let $\boldsymbol{B} \in \mathbb{R}^{E \times V}$ be the edge vertex incidence matrix of the graph, i.e. if $e \in E$ and $(u, v) = e$ then $\boldsymbol{B}(e, u) = 1$ and $\boldsymbol{B}(e, v) = -1$.

We let $\boldsymbol{\chi}_v \in \mathbb{R}^V$ denote the indicator of vertex v, i.e. $\boldsymbol{\chi}_v(v) = 1$ and $\boldsymbol{\chi}_v(u) = 0$ for $u \neq v$.

We let $s \in V$ denote the flow "source" and $t \in V$ the flow "sink".

The maximum flow problem is given by

$$\begin{aligned} \max_{\boldsymbol{f} \in \mathbb{R}^E, F \geq 0} \quad & F \\ \text{s.t. } \boldsymbol{B}\boldsymbol{f} &= F(-\boldsymbol{\chi}_s + \boldsymbol{\chi}_t) \\ \mathbf{0} \leq \boldsymbol{f} &\leq \boldsymbol{c} \end{aligned}$$

In the context of a given maximum flow problem, for a flow $\boldsymbol{f}$ satisfying $\boldsymbol{B}\boldsymbol{f} = F(-\boldsymbol{\chi}_s + \boldsymbol{\chi}_t)$, we define $\text{val}(\boldsymbol{f}) = F$.

Let $\boldsymbol{f}^*$ denote a feasible flow maximizing F, so that the maximum attainable flow value F is $\text{val}(\boldsymbol{f}^*)$.

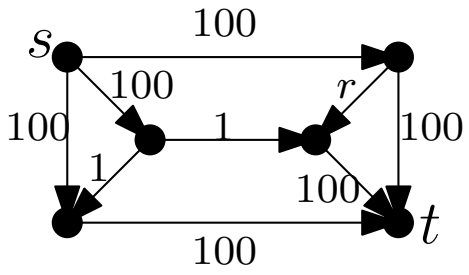
Algorithm 1: PTreeRotation(v,w)

/* Recall we assume $left_{\mathcal{T}_p}(v) = w$; and w and v both have two children. */
/* We first change the parent pointer for the parent that supports the subtree (if there is one). */
if $parent_{\mathcal{T}_p}(v) \neq NULL$ **then**
  $p \leftarrow parent_{\mathcal{T}_p}(v)$
  if $left_{\mathcal{T}_p}(p) = v$ **then**
    $left_{\mathcal{T}_p}(p) \leftarrow w$
  end
  else
    $right_{\mathcal{T}_p}(p) \leftarrow w$
  end
end
/* Compute roots of the subtrees rooted at the children of v and w. */
$a \leftarrow left_{\mathcal{T}_p}(w)$.
$b \leftarrow right_{\mathcal{T}_p}(w)$.
$c \leftarrow right_{\mathcal{T}_p}(v)$.
/* Change the tree structure according to the rotation. */
$right_{\mathcal{T}_p}(w) \leftarrow v$.
$left_{\mathcal{T}_p}(v) \leftarrow b$.
$\Delta c(v) \leftarrow \Delta cost(v)$.
$\Delta c(w) \leftarrow \Delta min(w) + \Delta cost(w)$.
$\Delta m(b) \leftarrow \Delta min(w) + \Delta min(b)$.
$\Delta m(c) \leftarrow \Delta min(c)$.
/* Recompute the fields at v and w. */
$\Delta min(w) \leftarrow \Delta min(v)$. ; // The old minimum of v is now the minimum of w.
$\Delta cost(w) \leftarrow \Delta c(w)$.; // We set the cost of w to be the pre-computed cost.
/* The new minimum of v is the minimum in subtrees rooted at b or c or the element v itself. */
$\Delta min(v) \leftarrow \min\{\Delta m(b), \Delta m(c), \Delta c(v)\}$.
$\Delta cost(v) \leftarrow \Delta c(v) - \Delta min(v)$.

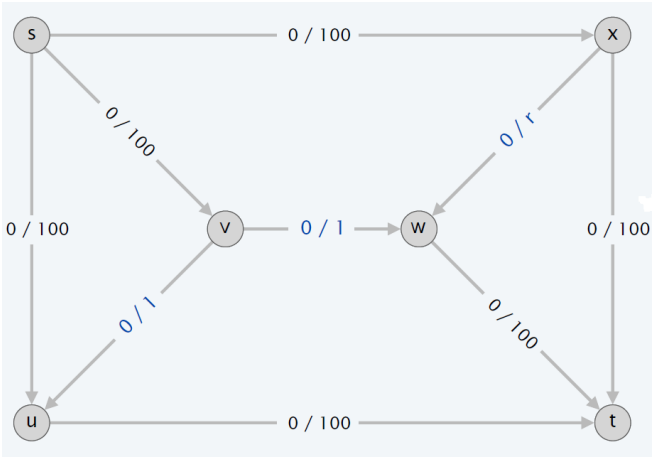
Exercise 2.A: Convergence of Ford-Fulkerson

Show that the Ford-Fulkerson algorithm may not terminate; moreover, it may converge a value not equal to the value of the maximum flow.

Hint: You might use the graph below with the given capacities, where $r = \frac{\sqrt{5}-1}{2}$ *(which implies that* $r^2 = 1 - r$*).*

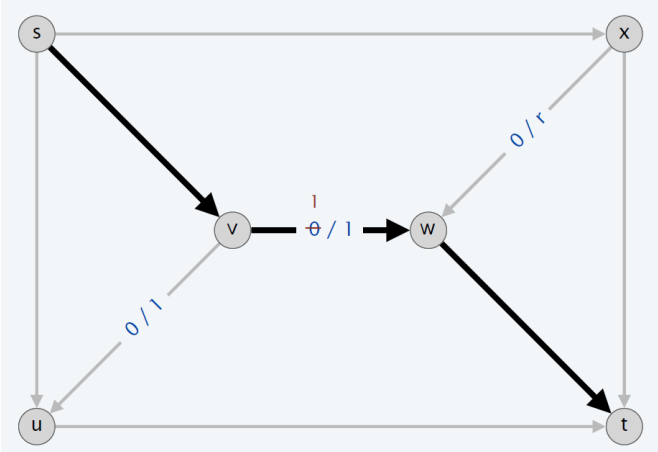


Solution. Consider the graph below with the given capacities.[1] The goal is to find a sequence of augmenting paths such that the algorithm converges to a value which is much smaller than the value of maximum flow.

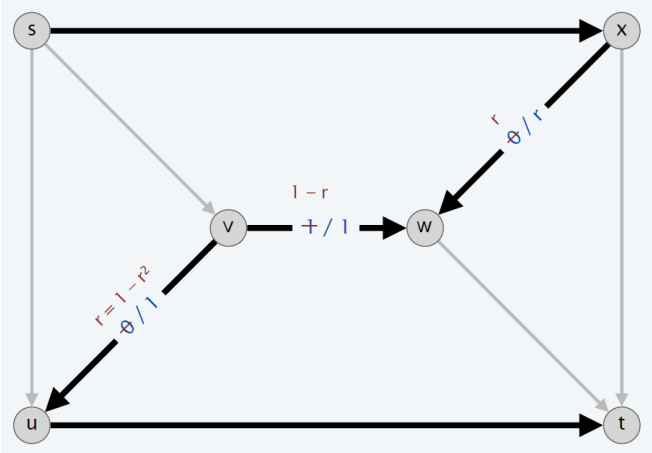


First, we augment Path 1: s, v, w, t.

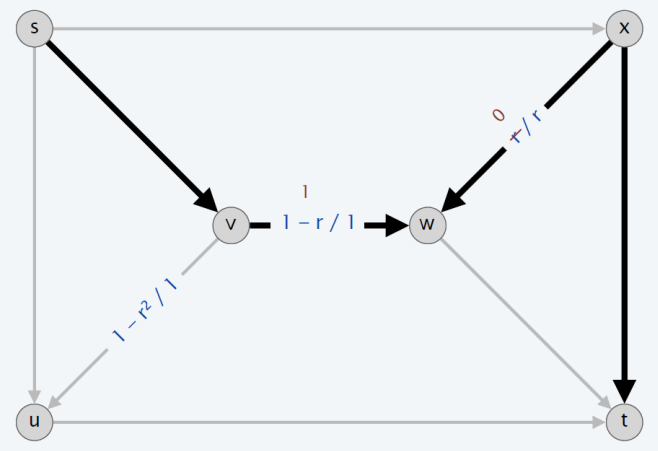
[1] All the figures used in this exercise are borrowed from the slides of the course Theory of Algorithms by Kevin Wayne, Princeton University.



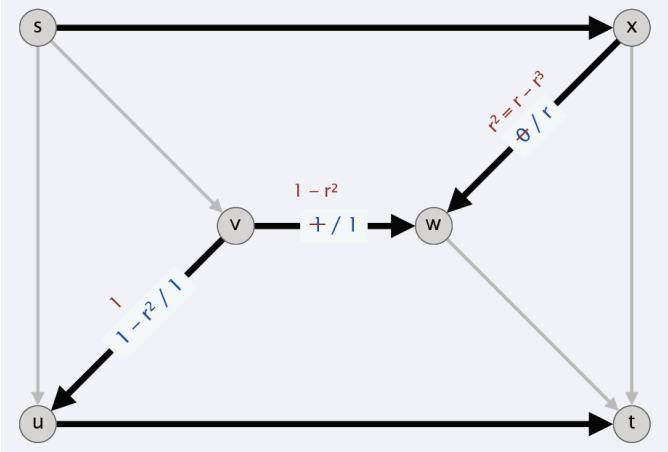
Then, we augment Path 2: s, x, w, v, u, t.



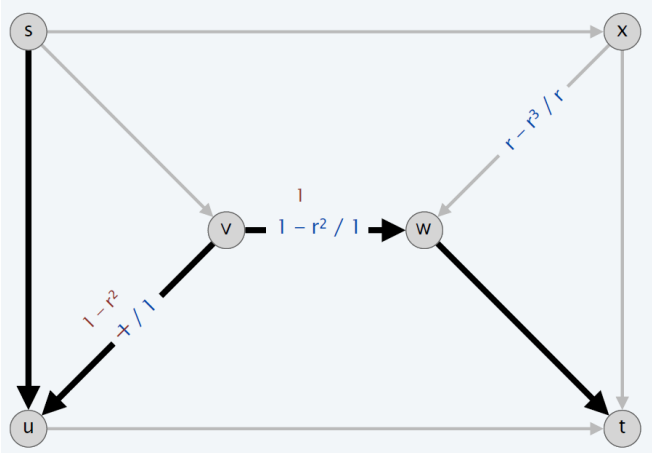
Then, we augment Path 3: s, v, w, x, t.



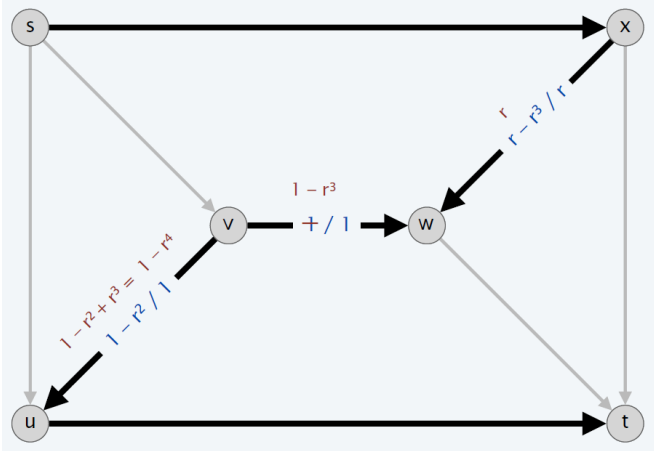
Then, we augment Path 4: s, x, w, v, u, t.



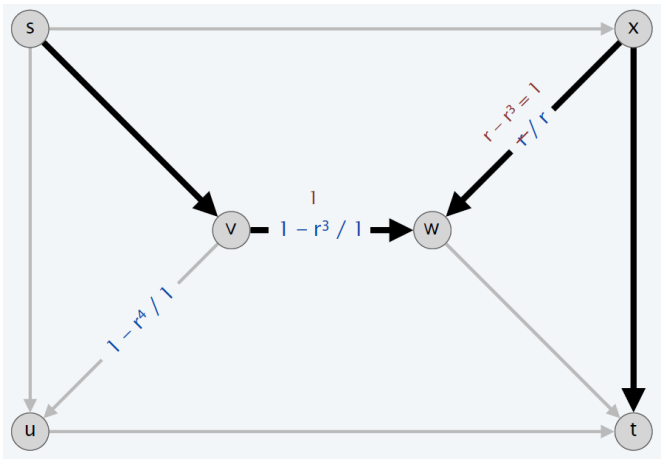
Then, we augment Path 5: s, u, v, w, t.



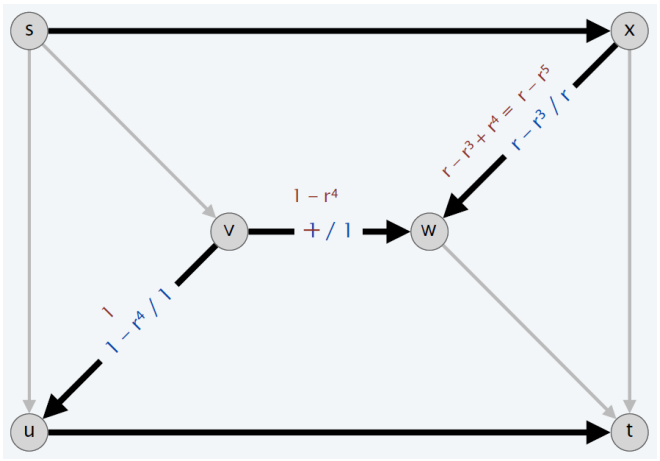
Then, we augment Path 6: s, x, w, v, u, t.



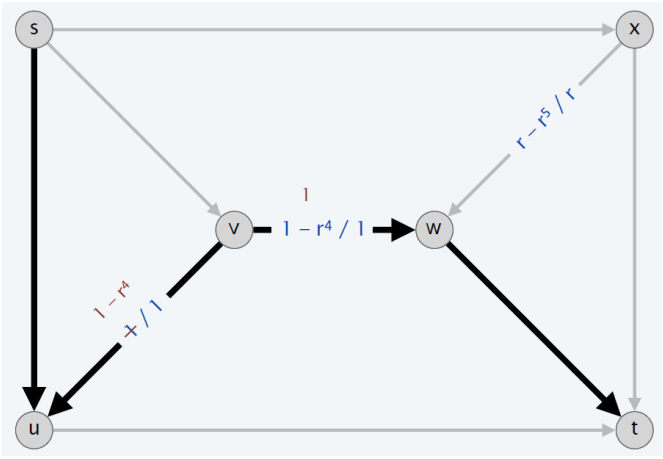
Then, we augment Path 7: s, v, w, x, t.



Then, we augment Path 8: s, x, w, v, u, t.



Then, we augment Path 9: s, u, v, w, t.



After augmenting Path 1 the flow is 1.

After augmenting Path 5 the flow is $1 + 2r + 2r^2$.

After augmenting Path 9 the flow is $1 + 2r + 2r^2 + 2r^3 + 2r^4$.

Using the given sequence of augmenting paths, after $(1 + 4k)$-th such path, the value of the flow is equal to

$$F = 1 + 2\sum_{i=1}^{k} r^i \leq 1 + 2\sum_{i=1}^{\infty} r^i = 3 + 2r \leq 5. \tag{1}$$

We observe that the value of maximum flow is equal to $200 + 1 = 201$. Therefore, we can conclude that the Ford-Fulkerson algorithm may not terminate; moreover, it may converge a value not equal to the value of the maximum flow.

Exercise 2.B: Iterative Refinement for Maximum Flow

Suppose we have an algorithm FlowRefine, which given a maximum flow instance $G = (V, E, \boldsymbol{c})$ with source $s \in V$ and sink $t \in V$ returns a feasible s-t flow $\tilde{\boldsymbol{f}}$, i.e. $\boldsymbol{B}\tilde{\boldsymbol{f}} = F(-\boldsymbol{\chi}_s + \boldsymbol{\chi}_t)$ for some F, and $\boldsymbol{0} \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c}$, and $\tilde{\boldsymbol{f}}$ is guaranteed to route at least half the maximum flow, i.e. $F = \text{val}(\tilde{\boldsymbol{f}}) \geq 0.5\,\text{val}(\boldsymbol{f}^*)$.

Suppose that the running time of FlowRefine is $O(|E|^c)$ for some constant $c \geq 1$.

Explain how we can use FlowRefine to find a flow $\hat{\boldsymbol{f}}$ that routes at least $(1 - \epsilon)\,\text{val}(\boldsymbol{f}^*)$ in time $O(|E|^c \log(1/\epsilon))$.

Solution. Assume that we are given a maximum flow instance $G = (V, E, \boldsymbol{c})$ with source $s \in V$ and sink $t \in V$. Consider the flow $\boldsymbol{f} = \boldsymbol{0}$. We run the algorithm FlowRefine on graph G. We know that, it will return a feasible s-t flow $\boldsymbol{f}_1$ such that $\text{val}(\boldsymbol{f}_1) \geq 0.5\,\text{val}(\boldsymbol{f}^*_G)$, where $\boldsymbol{f}^*_G$ is a maximum flow in G. Let $G_{\boldsymbol{f}_1}$ be the residual graph of $\boldsymbol{f}_1$ and set $\boldsymbol{f} = \boldsymbol{f} + \boldsymbol{f}_1$. Let's run FlowRefine this time on graph $G_{\boldsymbol{f}_1}$. It should return a flow $\boldsymbol{f}_2$. Now, we set $\boldsymbol{f} = \boldsymbol{f} + \boldsymbol{f}_2$. Thus, we get

$$\begin{aligned}
\text{val}\,(\boldsymbol{f}) &= \text{val}\,(\boldsymbol{f}_1) + \text{val}\,(\boldsymbol{f}_2) \\
&\geq \text{val}\,(\boldsymbol{f}_1) + \frac{1}{2}\,\text{val}\left(\boldsymbol{f}^*_{G_{\boldsymbol{f}_1}}\right) \\
&= \text{val}\,(\boldsymbol{f}_1) + \frac{1}{2}\,\text{val}\,(\boldsymbol{f}^*_G) - \frac{1}{2}\,\text{val}\,(\boldsymbol{f}_1) \\
&= \frac{1}{2}\,\text{val}\,(\boldsymbol{f}_1) + \frac{1}{2}\,\text{val}\,(\boldsymbol{f}^*_G) \\
&\geq \frac{1}{4}\,\text{val}\,(\boldsymbol{f}^*_G) + \frac{1}{2}\,\text{val}\,(\boldsymbol{f}^*_G) \\
&\geq \frac{3}{4}\,\text{val}\,(\boldsymbol{f}^*_G)
\end{aligned}$$

where we used $\text{val}(\boldsymbol{f}^*_{G_{\boldsymbol{f}_1}}) + \text{val}(\boldsymbol{f}_1) = \text{val}(\boldsymbol{f}^*_G)$ based on Lemmas 2.6 and 2.7 from Lecture 10.

By repeating the same argument, we can conclude that after $k = \log(1/\epsilon)$ iterations

$$\text{val}\,(\boldsymbol{f}) \geq \left(1 - \frac{1}{2^k}\right)\text{val}\,(\boldsymbol{f}^*_G) = (1 - \epsilon)\,\text{val}\,(\boldsymbol{f}^*_G)$$

We know that the running time of FlowRefine is $O(|E|^c)$ for some constant $c \geq 1$ and one can generate a residual graph in $O(|E|)$. Therefore, the above algorithm runs in time $O(|E|^c \log(1/\epsilon))$.