COMS 4995 Project Proposal

Parallelization of Ford Fulkerson Algorithm

Jiayuan Li (jl5957)

Summary:

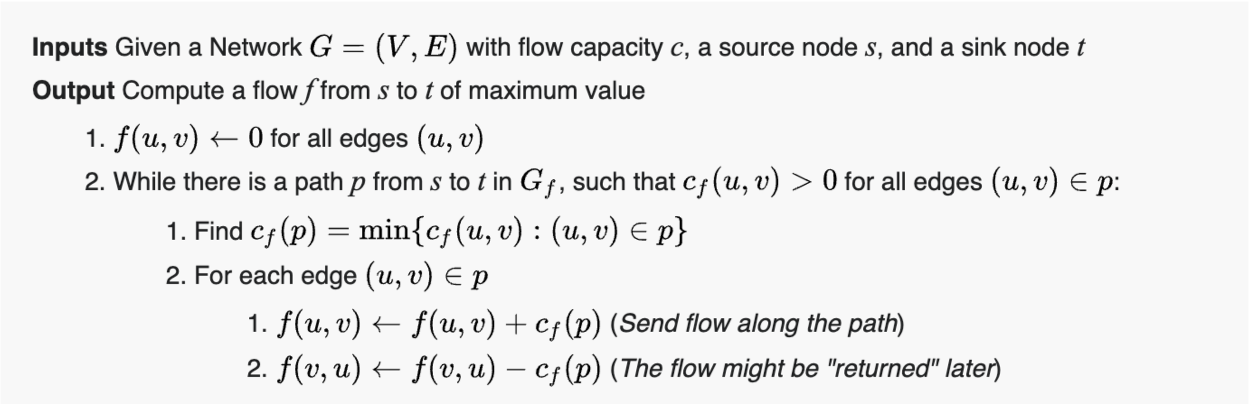
A maximum flow problem can be defined as following:

Given a graph N=(V,E), where each edge e comes with a capacity c. We have the following constraints:

- The flow sending through each edge e cannot exceed the capacity c
- The flow sending into each node should equal to the flow getting out from the node

Then, given a source node s and a sink node t, we want to figure out what is the maximum flow between s and t.

The Ford-Fulkerson Algorithm is a well-known algorithm for solving max-flow / min-cut problems. The algorithm is as following:



```
**Inputs** Given a Network $G = (V, E)$ with flow capacity $c$, a source node $s$, and a sink node $t$
**Output** Compute a flow $f$ from $s$ to $t$ of maximum value
    1. $f(u, v) \leftarrow 0$ for all edges $(u, v)$
    2. While there is a path $p$ from $s$ to $t$ in $G_f$, such that $c_f(u, v) > 0$ for all edges $(u, v) \in p$:
        1. Find $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$
        2. For each edge $(u, v) \in p$
            1. $f(u, v) \leftarrow f(u, v) + c_f(p)$ (*Send flow along the path*)
            2. $f(v, u) \leftarrow f(v, u) - c_f(p)$ (*The flow might be "returned" later*)
```

(source: from wikipedia)

Plan:

For the project, I plan to use Haskell to implement the following 3 stages:

- A sequential version of Ford Fulkerson algorithm
- A parallel version of Ford Fulkerson algorithm
- Parallel Ford Fulkerson algorithm with accelerated by GPU (Nvidia CUDA)

Dataset:

Specifically, the program to be implemented will be designed to solve a max-flow problem with the following input format:

```
<src> <dst> <capacity>
<src> <dst> <capacity>
…
```

The program will output the max-flow between node 1 and node N-1 where N is the largest node ID in the graph. The dataset and a C++ implementation of the program can be found in this github repo. The number of nodes in the graph can range from 50 to 10000.

Reference:

The sequential algorithm is well-explained in the wikipedia page.
A parallel approach of the algorithm can be found in this paper.