Problema de Fluxo máximo

Dupla:

- Lucas Moreira e Silva Alves - 20170027336
- Renan Almeida Goes Vieira de Melo - 20170021539

- Problema do fluxo máximo modelado a partir do problema de fluxo de custo mínimo
- Ambos consideram um fluxo através de uma rede, com capacidade limitada através de arcos (representam o caminho que o fluxo irá percorrer). Percorrendo a rede de um nó de entrada a um nó de saída. O problema envolve maximizar a quantidade de fluxo de uma origem S para um destino T. Tendo como entrada um nó de entrada, um de saída, e diferentes capacidades associadas a cada arco que conecta os nós

- Assim, o problema consiste na **maximização do fluxo** através de uma rede de nós conectados por arcos
- Temos que:

$$max \sum_{j} Xsj = \sum_{i} Xit$$

Aonde s é o nó de partida do fluxo e t representa o nó de destino, i e j representam os nós que serão percorridos

- Esse sistema deve respeitar as seguintes restrições

$$\sum_{i} Xik = \sum_{j} Xkj, \forall k$$

$$Xij <= Cij, \forall i, j$$

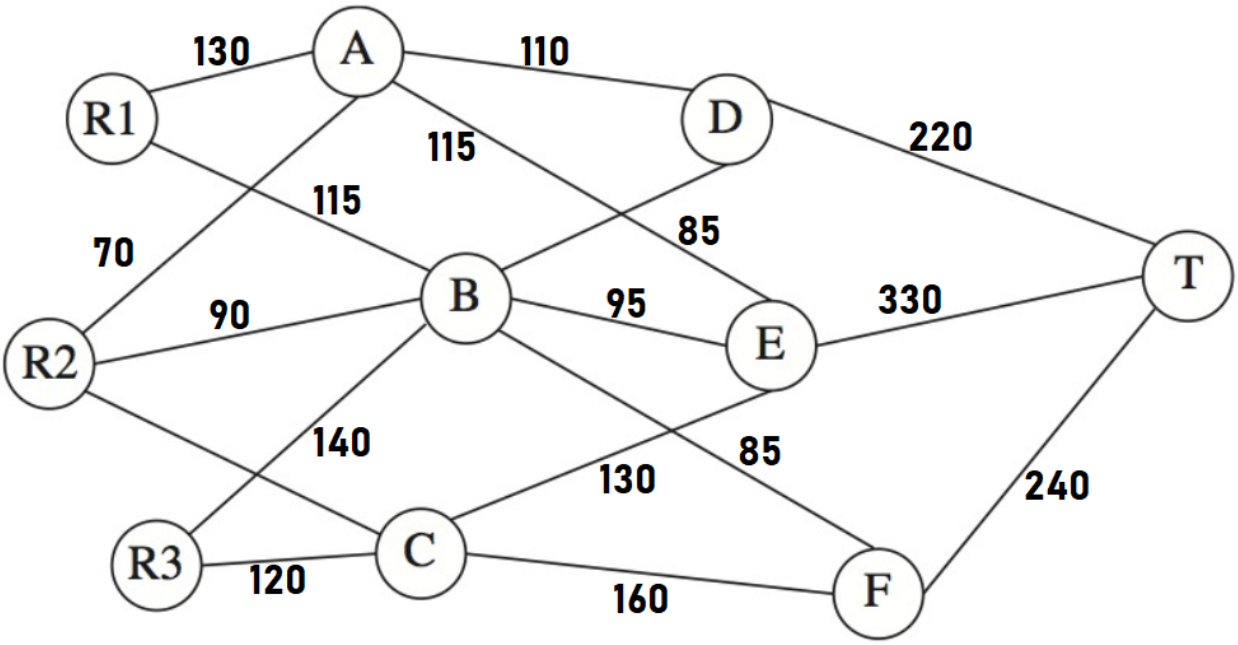
$$Xij >= 0, \forall i, j$$

As restrições estabelecidas definem que a soma do fluxo que entra em um nó deve ser igual a soma dos fluxos que saem deste, que o fluxo que passa no que o fluxo que passa através de um arco deve ser menor ou igual a capacidade desse arco, outrossim, o fluxo que passa através de cada arco devem ser maiores ou iguais a zero, nunca negativos

Problema ∑

- O problema inicial é de Fluxo Máximo, ele delimita as capacidades de cada aqueduto, porém não determina o custo de cada arco ou demanda de cada nó

- O problema determina que o fluxo parte de três rios: *R1*, *R2* e *R3*, buscando chegar em um escoadouro *T*, passando pelos nós *A*, *B*, *C*, *D*, *E* e *F,* conforme pode ser observado na imagem abaixo:
- O objetivo é passar o maior fluxo possível dos vértices de origem até o escoadouro, obedecendo as capacidades de cada arco



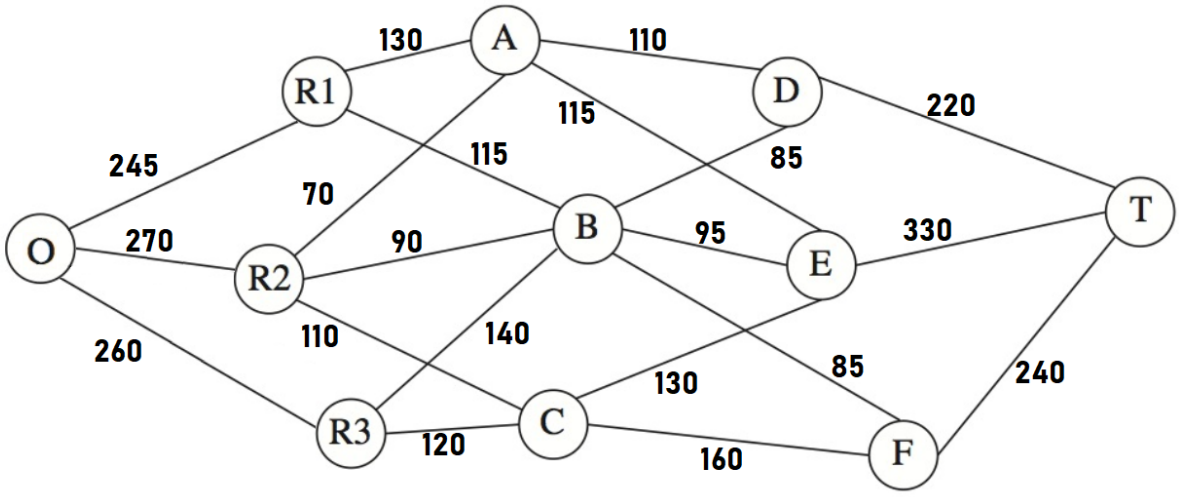
Transformando em um problema de fluxo de custo mínimo 💧

- Um problema de fluxo de custo mínimo se diferencia do problema do fluxo máximo na sua função objetivo por buscar minimizar o custo total, sendo assim:

$$min \sum_{ij \in \sigma} Xij$$

- Como o problema não oferecia custo para os arcos, o custo de todos os arcos foi considerado 1, além disso a demanda continuou sendo considerada zero

- Como é preciso especificar uma origem, foi criado um novo vértice com arcos de saída para os vértices *R1*, *R2* e *R3*. As capacidades que saem da origem devem ser iguais a soma das capacidades dos arcos que saem de cada vértice de entrada *R1*, *R2* e *R3*



- As restrições de capacidade e de conservação do fluxo foram mantidas:

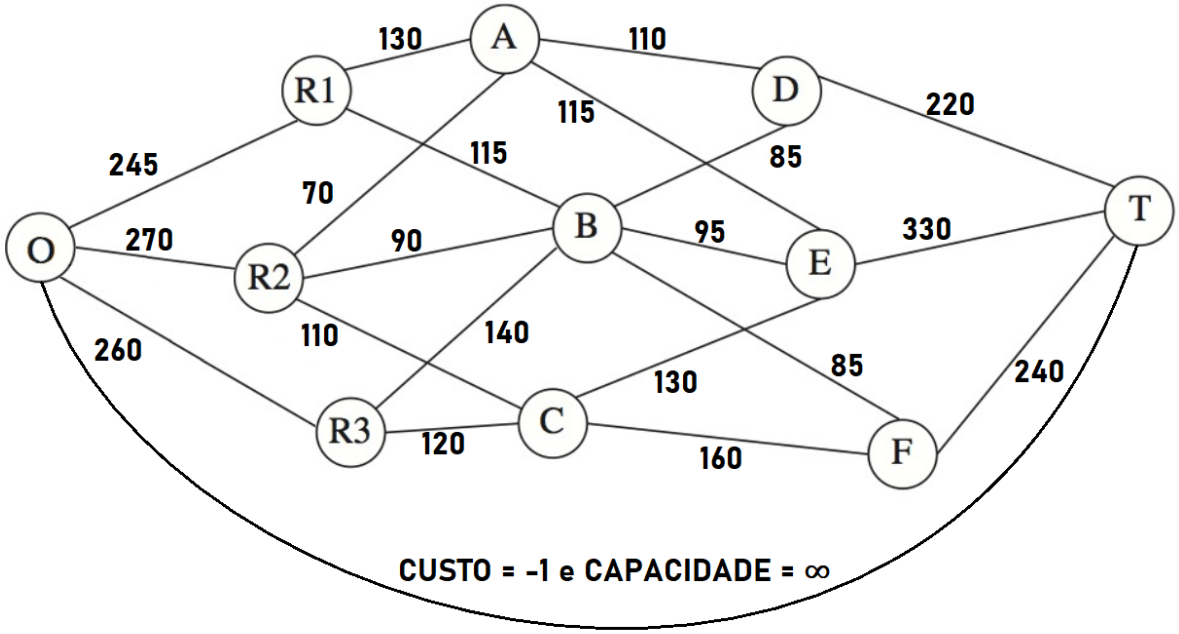
$$\sum_{ij \in \lambda -} Xij - \sum_{ij \in \lambda +} Xij = 0$$

$$Xij <= Cij, \forall i, j$$

$$Xij >= 0, \forall i, j$$

Adequando ao problema de fluxo máximo, com o $\lambda -$ sendo o conjunto de arcos de entrada e $\lambda +$ o conjunto de arcos de saída, aproximando-se à definição citada no início do relatório

- Como o objetivo do problema é minimizar o custo e todos os arcos permitem passar zero de fluxo, existiria a possibilidade de passar um fluxo de zero em todos os arcos, retornando o valor mínimo possível e garantindo que obedece a todas as restrições, sem ser uma solução válida. Para resolver isso foi necessário criar um arco fictício indo do nó origem para o escoadouro com custo -1 e cacapidade infinita, dessa forma o sistema buscará passar o máximo de fluxo possível por esse arco, dado que ao multiplicar por -1, quanto maior o fluxo menor o custo. Contudo, pela restrição de conservação é preciso que a soma dos arcos que chegam em *T* sejam iguais a zero, dessa forma os valores dos fluxos de outros arcos terão de ser diferente de zero, gerando o menor valor para uma solução válida.



Código 💻

- Foi utilizado `ortools` como bibilioteca para realizar as operações com programação linear. Utilizando o resolvedor `GLOP`
- A priori, o arquivo de entrada com as instâncias é lido e algumas restrições são estabelecidas

```
for arc in self.arcs_data:
    origin_node, destiny_node, capacity = arc.split()
    arc_name = f'x{origin_node}{destiny_node}'

    try:
        self.vertices_arcs[origin_node]["origin_arcs"].append(arc_name)
    except:
        self.vertices_arcs[origin_node] = {'origin_arcs': [arc_name], 'destiny_arcs': []}

    try:
        self.vertices_arcs[destiny_node]["destiny_arcs"].append(arc_name)

    except:
        self.vertices_arcs[destiny_node] = {'origin_arcs': [], 'destiny_arcs': [arc_name]}

    * self.arcs[arc_name] = self.solver.NumVar(0, float(capacity), arc_name)

** self.arcs[f'x{self.source}{self.sink}'] = self.solver.NumVar(0, self.solver.infinity(), f'x{self.source}{self.sink}')
```

*Para cada vértice são estabelecidos os arcos de entrada e saída dele, aplicando o limite da capacidade do arco na linha marcada com (*). Também é criado um arco fictício na linha (**) que conecta o primeiro com o nó de entrada com o nó de saída, aplicando um limite inferior de 0 e superior de infinito*

- A posteriori, fora aplicada as restrição de conservação

```
**def set_constraints(self):
        for vertice, arcs in self.vertices_arcs.items():
            * conservation_constraint = self.solver.Constraint(0, 0)
            if vertice != self.source and vertice != self.sink:

                for origin_arc in arcs['origin_arcs']:
                    ** conservation_constraint.SetCoefficient(self.arcs[origin_arc], 1)
                for destiny_arc in arcs['destiny_arcs']:
                    *** conservation_constraint.SetCoefficient(self.arcs[destiny_arc], -1)

            elif vertice == self.sink:

                for arc in arcs['destiny_arcs']:
                    **** conservation_constraint.SetCoefficient(self.arcs[arc], -1)

                ***** conservation_constraint.SetCoefficient(self.arcs[f'x{self.source}{self.sink}'], 1)
```

*A primeira restrição (*) é a restrição de conservação que obriga que todo fluxo que entra em um nó saia por completo deste. Já (**)* aplica um coeficiente positivo (equivalente ao custo) a todos os arcos que saem do nó e o *(***) estabelece que todos os arcos que saem do nó terão coeficiente negativo, assim a soma dos fluxos que saem do nó devem ser iguais a soma dos fluxo que entram. Já o (****) comina que todos os arcos que entram no nó escoadouro tem um coeficiente negativo e (*****) impõe um coeficiente positivo ao arco fictício com a entrada e escoadouro, obrigando, por (*) que este valor recebido pelo arco seja escoado*

```
def set_objective_function(self):
        objective = self.solver.Objective()

        * objective.SetCoefficient(self.arcs[f'x{self.source}{self.sink}'], -1)

        ** objective.SetMinimization()
```

*A função objetivo é definida pelo fluxo do arco fictício, em (*) esse valor é multiplicado por -1 para torná-lo positivo, como esse valor é o oposto do que chega no escoadouro pelos outros arcos, a solução ótima acabará sendo o valor (positivo) que chega no escadouro. Em (**) é estabelecido que a função é de minimização, ou seja, o valor obtido será o mínimo possível*

Como rodar ⚙️

- Clone o repositório `git clone https://github.com/lucasmsa/po-ufpb.git`
- Vá para a pasta raíz do projeto `cd po-ufpb`
- Instale as dependências `pip install -r requirements.txt`
- Rode com uma das instâncias dentro da pasta `input/instancias` ou alguma outra instância que tenha sida colocada na pasta `input/instancias` que obedeça o padrão estabelecido, para qualquer arquivo `.txt` dentro desta pasta rode o comando:

`python src/maximum_flow.py <nome_da_instancia_sem_.txt>`

- Exemplo: `python src/maximum_flow.py instance1`
- O resultado estará disponível na pasta `output` com um pdf gerado `maximum_flow_results.pdf` com o fluxo que passou por cada arco, bem como o valor da solução ótima e o grafo gerado na resolução do problema. O grafo se encontra no pdf e como uma imagem em `flow_graph.png`