

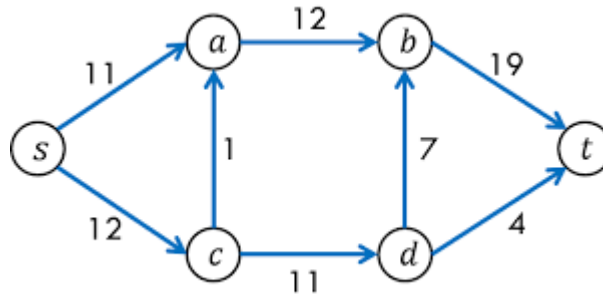
TUGAS MATA KULIAH
OPTIMISASI
MAX-FLOW PROBLEM USING JULIA



Dosen Pengampuh:
Ir. Novalio Daratha, S.T., M.Sc., Ph.D.
Disusun Oleh
Zizki Wahyudi (G1D021052)

PROGRAM STUDI TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS BENGKULU
2024

1. Carilah aliran arus maksimal pada Aliran Jaringan (Network Flow) berikut!



2. Modelkan bentuk Aliran Jaringan di atas ke dalam bentuk matriks 6x6, sebagai berikut.

$A = \begin{bmatrix} 0 & 11 & 0 & 12 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 19 & 0 \\ 0 & 1 & 0 & 0 & 11 & 0 \\ 0 & 0 & 7 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

3. Buatlah program Julia untuk menyelesaikannya berdasarkan matriks yang telah dibuat.

```

julia> using JuMP
julia> import HiGHS
julia> A = [
0 11 0 12 0 0
0 0 12 0 0 0
0 0 0 0 19 0
0 1 0 0 11 0
0 0 7 0 0 4
0 0 0 0 0 0
end Matrix{Int64};
julia> n = size(A)[1]
6
julia> max_flow = Model(HiGHS.Optimizer)
julia> JuMP.Model{<:Model{HiGHS.Optimizer}}(max_flow)
julia> JuMP.set_optimizer_attribute(max_flow, :solver, :HiGHS)
julia> JuMP.set_optimizer_attribute(max_flow, :objective_sense, :FEASIBILITY_SENSE)
julia> JuMP.set_optimizer_attribute(max_flow, :num_variables, 0)
julia> JuMP.set_optimizer_attribute(max_flow, :num_constraints, 0)
julia> JuMP.registered_names(max_flow)
Names registered in the model: none

```

4. Tahap pertama dalam membuat program ini adalah mengimpor paket

```
julia> using JuMP
julia> using HiGHS
julia>
```

5. Kemudian mendefinisikan graf dengan matriks

```
julia> A = [
    0 11 0 12 0 0
    0 0 12 0 0 0
    0 0 0 0 0 19
    0 1 0 0 11 0
    0 0 7 0 0 4
    0 0 0 0 0 0
]
6×6 Matrix{Int64}:
 0  11   0  12   0   0
 0   0  12   0   0   0
 0   0   0   0   0  19
 0   1   0   0  11   0
 0   0   7   0   0   4
 0   0   0   0   0   0
```

Matriks A mendefinisikan graf dengan biaya aliran antara node. Elemen $A[i,j]$ menunjukkan biaya aliran dari aliran dari node i ke node j . Jika biayanya 0, berarti tidak ada aliran langsung antara node tersebut.

6. Memebuat jumlah node, bagian ini menghitung jumlah node dalam graf A dengan mengambil ukuran dimensi pertama dari matriks A, dalam contoh ini, n akan bernilai 6 karena matriks A berukuran 6x6.

```
julia> n=size(A)[1]
6
```

7. Kemudian kita dapat membuat model

```
C:\Users\alyas\AppData\Local...
Precompiling project...
8 dependencies successfully precompiled in 20 seconds. 39 already precompiled.

julia> using JuMP

julia> import HiGHS

julia> A = [
    0 11 0 12 0 0
    0 0 12 0 0 0
    0 0 0 0 19 0
    0 1 0 0 11 0
    0 0 7 0 0 4
    0 0 0 0 0 0
]
6×6 Matrix{Int64}:
 0 11  0 12  0  0
 0  0 12  0  0  0
 0  0  0  0 19  0
 0  1  0  0 11  0
 0  0  7  0  0  4
 0  0  0  0  0  0

julia> n=size(A)[1]
6

julia> max_flow = Model(HiGHS.Optimizer)
A JuMP Model
├ solver: HiGHS
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none
```

```
julia>

julia> @variable(max_flow, f[1:n, 1:n] >= 0)
6×6 Matrix{VariableRef}:
 f[1,1] f[1,2] f[1,3] f[1,4] f[1,5] f[1,6]
 f[2,1] f[2,2] f[2,3] f[2,4] f[2,5] f[2,6]
 f[3,1] f[3,2] f[3,3] f[3,4] f[3,5] f[3,6]
 f[4,1] f[4,2] f[4,3] f[4,4] f[4,5] f[4,6]
 f[5,1] f[5,2] f[5,3] f[5,4] f[5,5] f[5,6]
 f[6,1] f[6,2] f[6,3] f[6,4] f[6,5] f[6,6]

julia>
6×6 Matrix{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}}, MathOptInterface.LessThan{Float64}}, ScalarShape}:
 f[1,1] <= 0 f[1,2] <= 11 ... f[1,5] <= 0 f[1,6] <= 0
 f[2,1] <= 0 f[2,2] <= 0 ... f[2,5] <= 0 f[2,6] <= 0
 f[3,1] <= 0 f[3,2] <= 0 ... f[3,5] <= 0 f[3,6] <= 19
 f[4,1] <= 0 f[4,2] <= 1 f[4,3] <= 11 f[4,4] <= 0
 f[5,1] <= 0 f[5,2] <= 0 f[5,3] <= 0 f[5,4] <= 4
 f[6,1] <= 0 f[6,2] <= 0 ... f[6,5] <= 0 f[6,6] <= 0

julia>
@constraint(max_flow, [i = 1:n; i != 1 && i != 6], sum(f[i, :]) == sum(f[:, i]))
JuMP.Containers.SparseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}}, MathOptInterface.EqualTo{Float64}}, ScalarShape}, 1, Tuple{Int64}} with 4 entries:
 [2] = f[2,1] - f[1,2] - f[3,2] - f[4,2] - f[5,2] - f[6,2] + f[2,3] + f[2,4] + f[2,5] + f[2,6] == 0
 [3] = f[3,1] + f[3,2] - f[1,3] - f[2,3] - f[4,3] - f[5,3] - f[6,3] + f[3,4] + f[3,5] + f[3,6] == 0
 [4] = f[4,1] + f[4,2] + f[4,3] - f[1,4] - f[2,4] - f[3,4] - f[5,4] - f[6,4] + f[4,5] + f[4,6] == 0
 [5] = f[5,1] + f[5,2] + f[5,3] + f[5,4] - f[1,5] - f[2,5] - f[3,5] - f[4,5] - f[6,5] + f[5,6] == 0
```

8. Mendefinisikan variable biner X untuk setiap arc dalam graf. Variable ini akan bernilai 1 jika arc tersebut termasuk dalam jalur dan variable akan bernilai 0 jika arc tersebut tidak termasuk dalam jalur.

```

julia>
julia> @variable(max_flow, f[1:n, 1:n] >= 0)
6×6 Matrix{VariableRef}:
 f[1,1] f[1,2] f[1,3] f[1,4] f[1,5] f[1,6]
 f[2,1] f[2,2] f[2,3] f[2,4] f[2,5] f[2,6]
 f[3,1] f[3,2] f[3,3] f[3,4] f[3,5] f[3,6]
 f[4,1] f[4,2] f[4,3] f[4,4] f[4,5] f[4,6]
 f[5,1] f[5,2] f[5,3] f[5,4] f[5,5] f[5,6]
 f[6,1] f[6,2] f[6,3] f[6,4] f[6,5] f[6,6]

julia>
julia> @constraint(max_flow, [i = 1:n, j = 1:n], f[i, j] <= A[i, j])
6×6 Matrix{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}}, MathOptInterface.LessThan{Float64}}, ScalarShape}:
 f[1,1] <= 0 f[1,2] <= 11 ... f[1,5] <= 0 f[1,6] <= 0
 f[2,1] <= 0 f[2,2] <= 0 ... f[2,5] <= 0 f[2,6] <= 0
 f[3,1] <= 0 f[3,2] <= 0 ... f[3,5] <= 0 f[3,6] <= 19
 f[4,1] <= 0 f[4,2] <= 1 ... f[4,5] <= 11 f[4,6] <= 0
 f[5,1] <= 0 f[5,2] <= 0 ... f[5,5] <= 0 f[5,6] <= 4
 f[6,1] <= 0 f[6,2] <= 0 ... f[6,5] <= 0 f[6,6] <= 0

julia>
julia> @constraint(max_flow, [i = 1:n; i != 1 && i != 6], sum(f[i, :]) == sum(f[:, i]))
JuMP.Containers.SparseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}}, MathOptInterface.EqualTo{Float64}}, ScalarShape}, 1, Tuple{Int64}} with 4 entries:
 [2] = f[2,1] - f[1,2] - f[3,2] - f[4,2] - f[5,2] - f[6,2] + f[2,3] + f[2,4] + f[2,5] + f[2,6] == 0
 [3] = f[3,1] + f[3,2] - f[1,3] - f[2,3] - f[4,3] - f[5,3] - f[6,3] + f[3,4] + f[3,5] + f[3,6] == 0
 [4] = f[4,1] + f[4,2] + f[4,3] - f[1,4] - f[2,4] - f[3,4] - f[5,4] - f[6,4] + f[4,5] + f[4,6] == 0
 [5] = f[5,1] + f[5,2] + f[5,3] + f[5,4] - f[1,5] - f[2,5] - f[3,5] - f[4,5] - f[6,5] + f[5,6] == 0

```

9. Constraint : Arc dengan biaya 0. Constraint ini memastikan bahwa arc dengan biaya nol tidak termasuk dalam jalur.

```

julia>
julia> @variable(max_flow, f[1:n, 1:n] >= 0)
6×6 Matrix{VariableRef}:
 f[1,1] f[1,2] f[1,3] f[1,4] f[1,5] f[1,6]
 f[2,1] f[2,2] f[2,3] f[2,4] f[2,5] f[2,6]
 f[3,1] f[3,2] f[3,3] f[3,4] f[3,5] f[3,6]
 f[4,1] f[4,2] f[4,3] f[4,4] f[4,5] f[4,6]
 f[5,1] f[5,2] f[5,3] f[5,4] f[5,5] f[5,6]
 f[6,1] f[6,2] f[6,3] f[6,4] f[6,5] f[6,6]

julia>
julia> @constraint(max_flow, [i = 1:n, j = 1:n], f[i, j] <= A[i, j])
6×6 Matrix{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}}, MathOptInterface.LessThan{Float64}}, ScalarShape}:
 f[1,1] <= 0 f[1,2] <= 11 ... f[1,5] <= 0 f[1,6] <= 0
 f[2,1] <= 0 f[2,2] <= 0 ... f[2,5] <= 0 f[2,6] <= 0
 f[3,1] <= 0 f[3,2] <= 0 ... f[3,5] <= 0 f[3,6] <= 19
 f[4,1] <= 0 f[4,2] <= 1 ... f[4,5] <= 11 f[4,6] <= 0
 f[5,1] <= 0 f[5,2] <= 0 ... f[5,5] <= 0 f[5,6] <= 4
 f[6,1] <= 0 f[6,2] <= 0 ... f[6,5] <= 0 f[6,6] <= 0

julia>
julia> @constraint(max_flow, [i = 1:n; i != 1 && i != 6], sum(f[i, :]) == sum(f[:, i]))
JuMP.Containers.SparseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}}, MathOptInterface.EqualTo{Float64}}, ScalarShape}, 1, Tuple{Int64}} with 4 entries:
 [2] = f[2,1] - f[1,2] - f[3,2] - f[4,2] - f[5,2] - f[6,2] + f[2,3] + f[2,4] + f[2,5] + f[2,6] == 0
 [3] = f[3,1] + f[3,2] - f[1,3] - f[2,3] - f[4,3] - f[5,3] - f[6,3] + f[3,4] + f[3,5] + f[3,6] == 0
 [4] = f[4,1] + f[4,2] + f[4,3] - f[1,4] - f[2,4] - f[3,4] - f[5,4] - f[6,4] + f[4,5] + f[4,6] == 0
 [5] = f[5,1] + f[5,2] + f[5,3] + f[5,4] - f[1,5] - f[2,5] - f[3,5] - f[4,5] - f[6,5] + f[5,6] == 0

```

10. Tujuan dari model ini adalah untuk meminimalkan total biaya aliran. Ini dilakukan dengan menjumlahkan hasil perkalian elemen-elemen matriks biaya A dengan variabel x.

```
julia> @objective(max_flow, Max, sum(f[1, :]))
f[1,1] + f[1,2] + f[1,3] + f[1,4] + f[1,5] + f[1,6]
```

11. Optimasi dan Hasil. Bagian ini menjalankan optimasi model dan menampilkan hasilnya. `objective_value(model)` memberikan nilai objektif dari solusi optimal, dan `value.(x)` memberikan nilai dari variabel `x` dalam solusi optimal.

```
julia> @objective(max_flow, Max, sum(f[1, :]))
f[1,1] + f[1,2] + f[1,3] + f[1,4] + f[1,5] + f[1,6]
```

```
julia> optimize!(max_flow)
```

Running HiGHS 1.7.2 (git hash: 5ce7a2753): Copyright (c) 2024 HiGHS under MIT licence terms

Coefficient ranges:

| | |
|--------|----------------|
| Matrix | [1e+00, 1e+00] |
| Cost | [1e+00, 1e+00] |
| Bound | [0e+00, 0e+00] |
| RHS | [1e+00, 2e+01] |

Presolving model

| | |
|----------------------------|----|
| 1 rows, 4 cols, 3 nonzeros | 0s |
| 0 rows, 0 cols, 0 nonzeros | 0s |

Presolve : Reductions: rows 0(-40); columns 0(-36); elements 0(-76) - Reduced to empty

Solving the original LP from the solution after postsolve

| | |
|-----------------|---------------------|
| Model status | : Optimal |
| Objective value | : 2.30000000000e+01 |
| HiGHS run time | : 0.01 |

```
julia> @assert is_solved_and_feasible(max_flow)
```

```
julia> objective_value(max_flow)
```