MTH6412B - Projet voyageur de commerce

Lien vers notre github : https://github.com/nathanemac/Projet/tree/phase2

Phase 2

Victor Darleguy - Nathan Allaire

Importation des résultats de la phase 1

In [1]:
```
include("../Phase 1/node.jl")
include("../Phase 1/edge.jl")
include("../Phase 1/graph.jl")
include("../Phase 1/read_stsp.jl")
include("../Phase 1/main.jl")

include("utils.jl")
include("PriorityQueue.jl")
```

```
get_priority
```

Question 1)

Choisir et implémenter une structure de données pour les composantes connexes d’un graphe

In [2]:
```
import Base.isless, Base.==, Base.popfirst!
```

In [3]:
```


# AbstractPriorityItem est une structure de données abstraite pour les élément
"""
    AbstractPriorityItem{T}

Structure de base abstraite pour les éléments qui seront utilisés dans une fil
`T` est le type de donnée encapsulé dans l'élément de priorité.
"""
abstract type AbstractPriorityItem{T} end

# PriorityItem est une implémentation concrète d'un élément de file de priorit
"""
    mutable struct PriorityItem{T} <: AbstractPriorityItem{T}

Implémente un élément de priorité avec une `priorité` et des `données`.
- `priority` est un `Number` indiquant la priorité de l'élément.
- `data` est la donnée de type `T` stockée dans l'élément.
"""
mutable struct PriorityItem{T} <: AbstractPriorityItem{T}
  priority::Number
  data::T
```

```julia
end

# PriorityItem est le constructeur pour créer un nouvel élément de priorité.
"""
    PriorityItem(priority::Number, data::T)

Constructeur pour `PriorityItem`. Prend en compte une `priorité` et des `donné
La priorité est ajustée pour ne jamais être inférieure à zéro.
"""
function PriorityItem(priority::Number, data::T) where T
  PriorityItem{T}(max(0, priority), data)
end

# priority retourne la priorité d'un PriorityItem.
"""
    priority(p::PriorityItem)

Renvoie la priorité de l'élément de priorité `p`.
"""
priority(p::PriorityItem) = p.priority

# priority! modifie la priorité d'un PriorityItem.
"""
    priority!(p::PriorityItem, priority::Number)

Modifie la valeur de la priorité de l'élément `p` avec la nouvelle `priorité`.
La nouvelle priorité est ajustée pour ne jamais être inférieure à zéro.
"""
function priority!(p::PriorityItem, priority::Number)
  p.priority = max(0, priority)
  p
end

# Redéfinition des opérateurs isless et == pour PriorityItem.
isless(p::PriorityItem, q::PriorityItem) = priority(p) < priority(q)
==(p::PriorityItem, q::PriorityItem) = priority(p) == priority(q)

# AbstractQueue est une structure de données abstraite pour les files.
"""
    AbstractQueue{T}

Structure de base abstraite pour les files qui seront utilisées pour gérer les
`T` est le type spécifique des éléments de priorité utilisés dans la file.
"""
abstract type AbstractQueue{T} end

# PriorityQueue est une implémentation concrète d'une file de priorité.
"""
    mutable struct PriorityQueue{T <: AbstractPriorityItem} <: AbstractQueue{T

Implémente une file de priorité qui utilise un `Vector` pour stocker les éléme
"""
mutable struct PriorityQueue{T <: AbstractPriorityItem} <: AbstractQueue{T}
  items::Vector{T}
end

# PriorityQueue est le constructeur pour créer une nouvelle file de priorité.
PriorityQueue{T}() where T = PriorityQueue(T[])
```

```
# pop_lowest! retire et renvoie l'élément avec la plus faible priorité.
"""
    pop_lowest!(q::PriorityQueue)

Retire et renvoie l'élément ayant la plus faible priorité de la file de priori
"""
function pop_lowest!(q::PriorityQueue)
  lowest = q.items[1]
  for item in q.items[2:end]
    if item < lowest
      lowest = item
    end
  end
  idx = findfirst(x -> x == lowest, q.items)
  deleteat!(q.items, idx)
  lowest
end

# update_priority! modifie la priorité d'un élément dans la file.
"""
    update_priority!(q::PriorityQueue, item_data, new_priority)

Modifie la priorité d'un élément spécifique dans la file de priorité `q`.
`item_data` est la donnée de l'élément à modifier.
`new_priority` est la nouvelle priorité à attribuer à l'élément.
"""
function update_priority!(q::PriorityQueue, item_data, new_priority)
  for pi in q.items
    if pi.data == item_data
      priority!(pi, new_priority)
      return
    end
  end
end

# get_priority renvoie la priorité d'un élément spécifique ou Inf si non trouv
"""
    get_priority(q::PriorityQueue, item_data)

Renvoie la priorité d'un élément spécifique dans la file de priorité `q`.
`item_data` est la donnée de l'élément dont la priorité est demandée.
Renvoie `Inf` si l'élément n'est pas trouvé dans la file.
"""
function get_priority(q::PriorityQueue, item_data)
  for pi in q.items
    if pi.data == item_data
      return pi.priority
    end
  end
  return Inf # Si l'élément n'est pas trouvé dans la file de priorité
end
```

get_priority

Tests unitaires de l'implémentation une structure de données pour les composantes connexes

In [4]:

```julia
using Test
# Test pour le constructeur PriorityItem
@testset "PriorityItem Constructor Tests" begin
    item = PriorityItem(-5, "Data")
    @test item.priority == 0
    @test item.data == "Data"
end

# Test pour la fonction priority
@testset "PriorityItem Functions Tests" begin
    item = PriorityItem(10, "Data")
    @test priority(item) == 10
end

# Test pour la fonction priority!
@testset "PriorityItem Mutator Tests" begin
    item = PriorityItem(10, "Data")
    priority!(item, 15)
    @test item.priority == 15

    priority!(item, -5)
    @test item.priority == 0  # La priorité ne peut pas être négative
end

# Test pour les opérateurs isless et ==
@testset "PriorityItem Operators Tests" begin
    item1 = PriorityItem(10, "Data1")
    item2 = PriorityItem(15, "Data2")
    @test item1 < item2
    @test item1 != item2
    item2.priority = 10
    @test item1 == item2
end

# Test pour update_priority!
@testset "PriorityQueue Update Priority Test" begin
    queue = PriorityQueue{PriorityItem{String}}()
    item1 = PriorityItem(10, "Item1")
    push!(queue.items, item1)

    update_priority!(queue, "Item1", 20)

    # Trouver l'élément mis à jour dans la queue
    item1_updated_index = findfirst(pi -> pi.data == "Item1", queue.items)
    item1_updated = queue.items[item1_updated_index]  # Utilisez l'index pour

    @test item1_updated !== nothing
    @test item1_updated.priority == 20
end
```

```
Test Summary:                  | Pass  Total  Time
PriorityItem Constructor Tests |    2      2  0.2s
Test Summary:                | Pass  Total  Time
PriorityItem Functions Tests |    1      1  0.0s
Test Summary:              | Pass  Total  Time
PriorityItem Mutator Tests |    2      2  0.0s
Test Summary:                | Pass  Total  Time
PriorityItem Operators Tests |    3      3  0.0s
Test Summary:                      | Pass  Total  Time
```

```
PriorityQueue Update Priority Test |      2       2  0.0s
Test.DefaultTestSet("PriorityQueue Update Priority Test", Any[], 2, false, false, true, 1.699282515302e9, 1.699282515339e9, false)
```

In [5]:

```
n1 = Node("1", [1.0, 3.0], nothing)
n2 = Node("2", [2.0, 1.0])
n3 = Node("3", [1.0, 2.0], nothing)
n4 = Node("4", [1.0, 2.0])
n5 = Node("5", [2.0, 3.9])

e1 = Edge(n1, n2, 5.6)
e2 = Edge(n2, n3, 1.0)
e3 = Edge(n1, n3, 2.0)
e4 = Edge(n2, n4, 2.0)
e5 = Edge(n1, n4, 0.5)
e6 = Edge(n4, n5, 3.0)
```

```
Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("4", [1.0, 2.0], nothing, 0), Node{Vector{Float64}}("5", [2.0, 3.9], nothing, 0), 3.0)
```

In [6]:

```
connexcomp1 = ConnexComponent("connex component 1", [n1, n2])
connexcomp2 = ConnexComponent("connex component 2", [n3, n4])
connexcomp3 = ConnexComponent("connex component 3", [n5])

graph_test = ConnexGraph("graph test", [connexcomp1, connexcomp2])
```

```
ConnexGraph{Vector{Float64}}("graph test", ConnexComponent{Vector{Float64}}[ConnexComponent{Vector{Float64}}("connex component 1", Node{Vector{Float64}}[Node{Vector{Float64}}("1", [1.0, 3.0], nothing, 0), Node{Vector{Float64}}("2", [2.0, 1.0], nothing, 0)], Edge{Vector{Float64}}[], nothing), ConnexComponent{Vector{Float64}}("connex component 2", Node{Vector{Float64}}[Node{Vector{Float64}}("3", [1.0, 2.0], nothing, 0), Node{Vector{Float64}}("4", [1.0, 2.0], nothing, 0)], Edge{Vector{Float64}}[], nothing)])
```

Question 2)

a. Implémenter l’algorithme de Kruskal

In [7]:

```
# find_component recherche la composante connexe d'un noeud donné.
"""
    find_component(components, node)

Recherche et renvoie la composante connexe à laquelle appartient le noeud `nod
- `components` est un tableau de composantes connexes.
- `node` est le noeud dont la composante connexe est recherchée.

Renvoie la composante connexe trouvée ou `nothing` si le noeud n'est pas trouv
"""
function find_component(components, node)
    for component in components
        if node in component.nodes
            return component
        end
    end
    return nothing
end
```

```
# merge_components! fusionne deux composantes connexes en une seule.
"""
    merge_components!(components, component1, component2)

Fusionne deux composantes connexes `component1` et `component2` en une seule.
- `components` est le tableau contenant toutes les composantes.
- `component1` et `component2` sont les composantes à fusionner.

Supprime `component2` du tableau `components` après fusion.
"""
function merge_components!(components, component1, component2)
    for node in component2.nodes
        push!(component1.nodes, node)
    end
    deleteat!(components, findfirst(x -> x == component2, components))
end

# L'algorithme de Kruskal permet de trouver l'arbre couvrant minimal d'un grap
"""
    Kruskal(graph::ExtendedGraph)

Implémente l'algorithme de Kruskal pour trouver l'arbre couvrant minimal d'un
- `graph` est un graphe étendu avec des sommets et des arêtes.

Crée un `ExtendedGraph` représentant l'arbre couvrant minimal trouvé.
"""
function Kruskal(graph::ExtendedGraph)
    A0 = typeof(graph.edges[1])[]
    A = graph.edges
    S_connex = ConnexGraph("Graphe connexe", graph)
    S = graph.nodes

    res = ExtendedGraph("res Kruskal", S, A0)

    for n in S
        add_connex_component!(S_connex, ConnexComponent("", [n]))
    end

    A_sorted = sort(A, by = e -> e.weight)
    Components = S_connex.components

    for a in A_sorted
        start_component = find_component(Components, a.start_node)
        end_component = find_component(Components, a.end_node)

        if start_component !== end_component
            push!(A0, a)
            merge_components!(Components, start_component, end_component)
        end
    end
    return res
end
```

```
Kruskal
```

In [8]:

```
using Test
```

```julia
n1 = Node("1", [1.0, 3.0], nothing)
n2 = Node("2", [2.0, 1.0])
n3 = Node("3", [1.0, 2.0], nothing)
n4 = Node("4", [1.0, 2.0])
n5 = Node("5", [2.0, 3.9])

e1 = Edge(n1, n2, 5.6)
e2 = Edge(n2, n3, 1.0)
e3 = Edge(n1, n3, 2.0)
e4 = Edge(n2, n4, 2.0)
e5 = Edge(n1, n4, 0.5)
e6 = Edge(n4, n5, 3.0)

connexcomp1 = ConnexComponent("connex component 1", [n1, n2])
connexcomp2 = ConnexComponent("connex component 2", [n3, n4])
connexcomp3 = ConnexComponent("connex component 3", [n5])

@testset "Priority Queue Tests" begin

    @testset "Find Component Tests" begin
        components = [connexcomp1, connexcomp2, connexcomp3]

        @test find_component(components, n1) === connexcomp1
        @test find_component(components, n2) === connexcomp1
        @test find_component(components, n3) === connexcomp2
        @test find_component(components, n4) === connexcomp2
        @test find_component(components, n5) === connexcomp3

        n6 = Node("6", [3.0, 3.0])
        @test find_component(components, n6) === nothing
    end

    @testset "Merge Components Tests" begin
        components = [connexcomp1, connexcomp2, connexcomp3]

        merge_components!(components, connexcomp1, connexcomp2)

        @test length(components) == 2
        @test all(node in connexcomp1.nodes for node in [n1, n2, n3, n4])
        @test !any(c -> c == connexcomp2, components)
    end

    @testset "Kruskal Function Tests" begin
        nodes_test = [n1, n2, n3, n4, n5]
        edges_test = [e1, e2, e3, e4, e5, e6]
        graph_test = ExtendedGraph("graph test", nodes_test, edges_test)

        result = Kruskal(graph_test)

        @test typeof(result) == ExtendedGraph{Vector{Float64}, Float64}
        @test length(result.edges) == 4
    end
end
```

```
Test Summary:        | Pass  Total  Time
Priority Queue Tests |   11     11  0.4s
Test.DefaultTestSet("Priority Queue Tests", Any[Test.DefaultTestSet("Find Compone
nt Tests", Any[], 6, false, false, true, 1.699282516788e9, 1.699282516818e9, fals
```

e), Test.DefaultTestSet("Merge Components Tests", Any[], 3, false, false, true, 1.699282516818e9, 1.699282516901e9, false), Test.DefaultTestSet("Kruskal Function Tests", Any[], 2, false, false, true, 1.699282516901e9, 1.699282517158e9, false)], 0, false, false, true, 1.699282516788e9, 1.699282517158e9, false)

b. Tester sur l'exemple des notes de laboratoire

In [9]:

```
a, b, c, d, e, f, g, h, i = Node("a", 1.0), Node("b", 1.0), Node("c", 1.0), Nc
e1 = Edge(a, b, 4.)
e2 = Edge(b, c, 8.)
e3 = Edge(c, d, 7.)
e4 = Edge(d, e, 9.)
e5 = Edge(e, f, 10.)
e6 = Edge(d, f, 14.)
e7 = Edge(f, c, 4.)
e8 = Edge(f, g, 2.)
e9 = Edge(g, i, 6.)
e10 = Edge(g, h, 1.)
e11 = Edge(a, h, 8.)
e12 = Edge(h, i, 7.)
e13 = Edge(i, c, 2.)
e14 = Edge(b, h, 11.)
G_cours = ExtendedGraph("graphe du cours", [a, b, c, d, e, f, g, h, i],[e1, e2
```

```
ExtendedGraph{Float64, Float64}("graphe du cours", Node{Float64}[Node{Float64}("a", 1.0, nothing, 0), Node{Float64}("b", 1.0, nothing, 0), Node{Float64}("c", 1.0, nothing, 0), Node{Float64}("d", 1.0, nothing, 0), Node{Float64}("e", 1.0, nothing, 0), Node{Float64}("f", 1.0, nothing, 0), Node{Float64}("g", 1.0, nothing, 0), Node{Float64}("h", 1.0, nothing, 0), Node{Float64}("i", 1.0, nothing, 0)], Edge{Float64, Float64}[Edge{Float64, Float64}(Node{Float64}("a", 1.0, nothing, 0), Node{Float64}("b", 1.0, nothing, 0), 4.0), Edge{Float64, Float64}(Node{Float64}("b", 1.0, nothing, 0), Node{Float64}("c", 1.0, nothing, 0), 8.0), Edge{Float64, Float64}(Node{Float64}("c", 1.0, nothing, 0), Node{Float64}("d", 1.0, nothing, 0), 7.0), Edge{Float64, Float64}(Node{Float64}("d", 1.0, nothing, 0), Node{Float64}("e", 1.0, nothing, 0), 9.0), Edge{Float64, Float64}(Node{Float64}("e", 1.0, nothing, 0), Node{Float64}("f", 1.0, nothing, 0), 10.0), Edge{Float64, Float64}(Node{Float64}("d", 1.0, nothing, 0), Node{Float64}("f", 1.0, nothing, 0), 14.0), Edge{Float64, Float64}(Node{Float64}("f", 1.0, nothing, 0), Node{Float64}("c", 1.0, nothing, 0), 4.0), Edge{Float64, Float64}(Node{Float64}("f", 1.0, nothing, 0), Node{Float64}("g", 1.0, nothing, 0), 2.0), Edge{Float64, Float64}(Node{Float64}("g", 1.0, nothing, 0), Node{Float64}("i", 1.0, nothing, 0), 6.0), Edge{Float64, Float64}(Node{Float64}("g", 1.0, nothing, 0), Node{Float64}("h", 1.0, nothing, 0), 1.0), Edge{Float64, Float64}(Node{Float64}("a", 1.0, nothing, 0), Node{Float64}("h", 1.0, nothing, 0), 8.0), Edge{Float64, Float64}(Node{Float64}("h", 1.0, nothing, 0), Node{Float64}("i", 1.0, nothing, 0), 7.0), Edge{Float64, Float64}(Node{Float64}("i", 1.0, nothing, 0), Node{Float64}("c", 1.0, nothing, 0), 2.0), Edge{Float64, Float64}(Node{Float64}("b", 1.0, nothing, 0), Node{Float64}("h", 1.0, nothing, 0), 11.0)])
```

In [10]:

```
graph_cours_kruskal = Kruskal(G_cours)
show(graph_cours_kruskal)
```

```
Graph res Kruskal has 9 nodes and 8 edges.
Nodes:
Node a, data: 1.0, parent: No parent for node
Node b, data: 1.0, parent: No parent for node
Node c, data: 1.0, parent: No parent for node
Node d, data: 1.0, parent: No parent for node
Node e, data: 1.0, parent: No parent for node
```

```
Node f, data: 1.0, parent: No parent for node
Node g, data: 1.0, parent: No parent for node
Node h, data: 1.0, parent: No parent for node
Node i, data: 1.0, parent: No parent for node
Edges:
Edge from g to h, weight: 1.0
Edge from f to g, weight: 2.0
Edge from i to c, weight: 2.0
Edge from a to b, weight: 4.0
Edge from f to c, weight: 4.0
Edge from c to d, weight: 7.0
Edge from b to c, weight: 8.0
Edge from d to e, weight: 9.0
```

c. Tester l'implémentation sur diverses instances de TSP symétrique.

In [11]:

```
graph = build_graph("../Phase 1/instances/stsp/bays29.tsp", "Graph_Test")
show(graph)
```

```
Graph Graph_Test has 29 nodes and 435 edges.
Nodes:
Node 1, data: [1150.0, 1760.0], parent: No parent for node
Node 2, data: [630.0, 1660.0], parent: No parent for node
Node 3, data: [40.0, 2090.0], parent: No parent for node
Node 4, data: [750.0, 1100.0], parent: No parent for node
Node 5, data: [750.0, 2030.0], parent: No parent for node
Node 6, data: [1030.0, 2070.0], parent: No parent for node
Node 7, data: [1650.0, 650.0], parent: No parent for node
Node 8, data: [1490.0, 1630.0], parent: No parent for node
Node 9, data: [790.0, 2260.0], parent: No parent for node
Node 10, data: [710.0, 1310.0], parent: No parent for node
Node 11, data: [840.0, 550.0], parent: No parent for node
Node 12, data: [1170.0, 2300.0], parent: No parent for node
Node 13, data: [970.0, 1340.0], parent: No parent for node
Node 14, data: [510.0, 700.0], parent: No parent for node
Node 15, data: [750.0, 900.0], parent: No parent for node
Node 16, data: [1280.0, 1200.0], parent: No parent for node
Node 17, data: [230.0, 590.0], parent: No parent for node
Node 18, data: [460.0, 860.0], parent: No parent for node
Node 19, data: [1040.0, 950.0], parent: No parent for node
Node 20, data: [590.0, 1390.0], parent: No parent for node
Node 21, data: [830.0, 1770.0], parent: No parent for node
Node 22, data: [490.0, 500.0], parent: No parent for node
Node 23, data: [1840.0, 1240.0], parent: No parent for node
Node 24, data: [1260.0, 1500.0], parent: No parent for node
Node 25, data: [1280.0, 790.0], parent: No parent for node
Node 26, data: [490.0, 2130.0], parent: No parent for node
Node 27, data: [1460.0, 1420.0], parent: No parent for node
Node 28, data: [1260.0, 1910.0], parent: No parent for node
Node 29, data: [360.0, 1980.0], parent: No parent for node
Edges:
Edge from 1 to 1, weight: 0.0
Edge from 1 to 2, weight: 107.0
Edge from 1 to 3, weight: 241.0
Edge from 1 to 4, weight: 190.0
Edge from 1 to 5, weight: 124.0
Edge from 1 to 6, weight: 80.0
Edge from 1 to 7, weight: 316.0
Edge from 1 to 8, weight: 76.0
```

```
Edge from 1 to 9, weight: 152.0
Edge from 1 to 10, weight: 157.0
Edge from 1 to 11, weight: 283.0
Edge from 1 to 12, weight: 133.0
Edge from 1 to 13, weight: 113.0
Edge from 1 to 14, weight: 297.0
Edge from 1 to 15, weight: 228.0
Edge from 1 to 16, weight: 129.0
Edge from 1 to 17, weight: 348.0
Edge from 1 to 18, weight: 276.0
Edge from 1 to 19, weight: 188.0
Edge from 1 to 20, weight: 150.0
Edge from 1 to 21, weight: 65.0
Edge from 1 to 22, weight: 341.0
Edge from 1 to 23, weight: 184.0
Edge from 1 to 24, weight: 67.0
Edge from 1 to 25, weight: 221.0
Edge from 1 to 26, weight: 169.0
Edge from 1 to 27, weight: 108.0
Edge from 1 to 28, weight: 45.0
Edge from 1 to 29, weight: 167.0
Edge from 2 to 2, weight: 0.0
Edge from 2 to 3, weight: 148.0
Edge from 2 to 4, weight: 137.0
Edge from 2 to 5, weight: 88.0
Edge from 2 to 6, weight: 127.0
Edge from 2 to 7, weight: 336.0
Edge from 2 to 8, weight: 183.0
Edge from 2 to 9, weight: 134.0
Edge from 2 to 10, weight: 95.0
Edge from 2 to 11, weight: 254.0
Edge from 2 to 12, weight: 180.0
Edge from 2 to 13, weight: 101.0
Edge from 2 to 14, weight: 234.0
Edge from 2 to 15, weight: 175.0
Edge from 2 to 16, weight: 176.0
Edge from 2 to 17, weight: 265.0
Edge from 2 to 18, weight: 199.0
Edge from 2 to 19, weight: 182.0
Edge from 2 to 20, weight: 67.0
Edge from 2 to 21, weight: 42.0
Edge from 2 to 22, weight: 278.0
Edge from 2 to 23, weight: 271.0
Edge from 2 to 24, weight: 146.0
Edge from 2 to 25, weight: 251.0
Edge from 2 to 26, weight: 105.0
Edge from 2 to 27, weight: 191.0
Edge from 2 to 28, weight: 139.0
Edge from 2 to 29, weight: 79.0
Edge from 3 to 3, weight: 0.0
Edge from 3 to 4, weight: 374.0
Edge from 3 to 5, weight: 171.0
Edge from 3 to 6, weight: 259.0
Edge from 3 to 7, weight: 509.0
Edge from 3 to 8, weight: 317.0
Edge from 3 to 9, weight: 217.0
Edge from 3 to 10, weight: 232.0
Edge from 3 to 11, weight: 491.0
Edge from 3 to 12, weight: 312.0
Edge from 3 to 13, weight: 280.0
Edge from 3 to 14, weight: 391.0
Edge from 3 to 15, weight: 412.0
```

```
Edge from 3 to 16, weight: 349.0
Edge from 3 to 17, weight: 422.0
Edge from 3 to 18, weight: 356.0
Edge from 3 to 19, weight: 355.0
Edge from 3 to 20, weight: 204.0
Edge from 3 to 21, weight: 182.0
Edge from 3 to 22, weight: 435.0
Edge from 3 to 23, weight: 417.0
Edge from 3 to 24, weight: 292.0
Edge from 3 to 25, weight: 424.0
Edge from 3 to 26, weight: 116.0
Edge from 3 to 27, weight: 337.0
Edge from 3 to 28, weight: 273.0
Edge from 3 to 29, weight: 77.0
Edge from 4 to 4, weight: 0.0
Edge from 4 to 5, weight: 202.0
Edge from 4 to 6, weight: 234.0
Edge from 4 to 7, weight: 222.0
Edge from 4 to 8, weight: 192.0
Edge from 4 to 9, weight: 248.0
Edge from 4 to 10, weight: 42.0
Edge from 4 to 11, weight: 117.0
Edge from 4 to 12, weight: 287.0
Edge from 4 to 13, weight: 79.0
Edge from 4 to 14, weight: 107.0
Edge from 4 to 15, weight: 38.0
Edge from 4 to 16, weight: 121.0
Edge from 4 to 17, weight: 152.0
Edge from 4 to 18, weight: 86.0
Edge from 4 to 19, weight: 68.0
Edge from 4 to 20, weight: 70.0
Edge from 4 to 21, weight: 137.0
Edge from 4 to 22, weight: 151.0
Edge from 4 to 23, weight: 239.0
Edge from 4 to 24, weight: 135.0
Edge from 4 to 25, weight: 137.0
Edge from 4 to 26, weight: 242.0
Edge from 4 to 27, weight: 165.0
Edge from 4 to 28, weight: 228.0
Edge from 4 to 29, weight: 205.0
Edge from 5 to 5, weight: 0.0
Edge from 5 to 6, weight: 61.0
Edge from 5 to 7, weight: 392.0
Edge from 5 to 8, weight: 202.0
Edge from 5 to 9, weight: 46.0
Edge from 5 to 10, weight: 160.0
Edge from 5 to 11, weight: 319.0
Edge from 5 to 12, weight: 112.0
Edge from 5 to 13, weight: 163.0
Edge from 5 to 14, weight: 322.0
Edge from 5 to 15, weight: 240.0
Edge from 5 to 16, weight: 232.0
Edge from 5 to 17, weight: 314.0
Edge from 5 to 18, weight: 287.0
Edge from 5 to 19, weight: 238.0
Edge from 5 to 20, weight: 155.0
Edge from 5 to 21, weight: 65.0
Edge from 5 to 22, weight: 366.0
Edge from 5 to 23, weight: 300.0
Edge from 5 to 24, weight: 175.0
Edge from 5 to 25, weight: 307.0
Edge from 5 to 26, weight: 57.0
```

```
Edge from 5 to 27, weight: 220.0
Edge from 5 to 28, weight: 121.0
Edge from 5 to 29, weight: 97.0
Edge from 6 to 6, weight: 0.0
Edge from 6 to 7, weight: 386.0
Edge from 6 to 8, weight: 141.0
Edge from 6 to 9, weight: 72.0
Edge from 6 to 10, weight: 167.0
Edge from 6 to 11, weight: 351.0
Edge from 6 to 12, weight: 55.0
Edge from 6 to 13, weight: 157.0
Edge from 6 to 14, weight: 331.0
Edge from 6 to 15, weight: 272.0
Edge from 6 to 16, weight: 226.0
Edge from 6 to 17, weight: 362.0
Edge from 6 to 18, weight: 296.0
Edge from 6 to 19, weight: 232.0
Edge from 6 to 20, weight: 164.0
Edge from 6 to 21, weight: 85.0
Edge from 6 to 22, weight: 375.0
Edge from 6 to 23, weight: 249.0
Edge from 6 to 24, weight: 147.0
Edge from 6 to 25, weight: 301.0
Edge from 6 to 26, weight: 118.0
Edge from 6 to 27, weight: 188.0
Edge from 6 to 28, weight: 60.0
Edge from 6 to 29, weight: 185.0
Edge from 7 to 7, weight: 0.0
Edge from 7 to 8, weight: 233.0
Edge from 7 to 9, weight: 438.0
Edge from 7 to 10, weight: 254.0
Edge from 7 to 11, weight: 202.0
Edge from 7 to 12, weight: 439.0
Edge from 7 to 13, weight: 235.0
Edge from 7 to 14, weight: 254.0
Edge from 7 to 15, weight: 210.0
Edge from 7 to 16, weight: 187.0
Edge from 7 to 17, weight: 313.0
Edge from 7 to 18, weight: 266.0
Edge from 7 to 19, weight: 154.0
Edge from 7 to 20, weight: 282.0
Edge from 7 to 21, weight: 321.0
Edge from 7 to 22, weight: 298.0
Edge from 7 to 23, weight: 168.0
Edge from 7 to 24, weight: 249.0
Edge from 7 to 25, weight: 95.0
Edge from 7 to 26, weight: 437.0
Edge from 7 to 27, weight: 190.0
Edge from 7 to 28, weight: 314.0
Edge from 7 to 29, weight: 435.0
Edge from 8 to 8, weight: 0.0
Edge from 8 to 9, weight: 213.0
Edge from 8 to 10, weight: 188.0
Edge from 8 to 11, weight: 272.0
Edge from 8 to 12, weight: 193.0
Edge from 8 to 13, weight: 131.0
Edge from 8 to 14, weight: 302.0
Edge from 8 to 15, weight: 233.0
Edge from 8 to 16, weight: 98.0
Edge from 8 to 17, weight: 344.0
Edge from 8 to 18, weight: 289.0
Edge from 8 to 19, weight: 177.0
```

```
Edge from 8 to 20, weight: 216.0
Edge from 8 to 21, weight: 141.0
Edge from 8 to 22, weight: 346.0
Edge from 8 to 23, weight: 108.0
Edge from 8 to 24, weight: 57.0
Edge from 8 to 25, weight: 190.0
Edge from 8 to 26, weight: 245.0
Edge from 8 to 27, weight: 43.0
Edge from 8 to 28, weight: 81.0
Edge from 8 to 29, weight: 243.0
Edge from 9 to 9, weight: 0.0
Edge from 9 to 10, weight: 206.0
Edge from 9 to 11, weight: 365.0
Edge from 9 to 12, weight: 89.0
Edge from 9 to 13, weight: 209.0
Edge from 9 to 14, weight: 368.0
Edge from 9 to 15, weight: 286.0
Edge from 9 to 16, weight: 278.0
Edge from 9 to 17, weight: 360.0
Edge from 9 to 18, weight: 333.0
Edge from 9 to 19, weight: 284.0
Edge from 9 to 20, weight: 201.0
Edge from 9 to 21, weight: 111.0
Edge from 9 to 22, weight: 412.0
Edge from 9 to 23, weight: 321.0
Edge from 9 to 24, weight: 221.0
Edge from 9 to 25, weight: 353.0
Edge from 9 to 26, weight: 72.0
Edge from 9 to 27, weight: 266.0
Edge from 9 to 28, weight: 132.0
Edge from 9 to 29, weight: 111.0
Edge from 10 to 10, weight: 0.0
Edge from 10 to 11, weight: 159.0
Edge from 10 to 12, weight: 220.0
Edge from 10 to 13, weight: 57.0
Edge from 10 to 14, weight: 149.0
Edge from 10 to 15, weight: 80.0
Edge from 10 to 16, weight: 132.0
Edge from 10 to 17, weight: 193.0
Edge from 10 to 18, weight: 127.0
Edge from 10 to 19, weight: 100.0
Edge from 10 to 20, weight: 28.0
Edge from 10 to 21, weight: 95.0
Edge from 10 to 22, weight: 193.0
Edge from 10 to 23, weight: 241.0
Edge from 10 to 24, weight: 131.0
Edge from 10 to 25, weight: 169.0
Edge from 10 to 26, weight: 200.0
Edge from 10 to 27, weight: 161.0
Edge from 10 to 28, weight: 189.0
Edge from 10 to 29, weight: 163.0
Edge from 11 to 11, weight: 0.0
Edge from 11 to 12, weight: 404.0
Edge from 11 to 13, weight: 176.0
Edge from 11 to 14, weight: 106.0
Edge from 11 to 15, weight: 79.0
Edge from 11 to 16, weight: 161.0
Edge from 11 to 17, weight: 165.0
Edge from 11 to 18, weight: 141.0
Edge from 11 to 19, weight: 95.0
Edge from 11 to 20, weight: 187.0
Edge from 11 to 21, weight: 254.0
```

```
Edge from 11 to 22, weight: 103.0
Edge from 11 to 23, weight: 279.0
Edge from 11 to 24, weight: 215.0
Edge from 11 to 25, weight: 117.0
Edge from 11 to 26, weight: 359.0
Edge from 11 to 27, weight: 216.0
Edge from 11 to 28, weight: 308.0
Edge from 11 to 29, weight: 322.0
Edge from 12 to 12, weight: 0.0
Edge from 12 to 13, weight: 210.0
Edge from 12 to 14, weight: 384.0
Edge from 12 to 15, weight: 325.0
Edge from 12 to 16, weight: 279.0
Edge from 12 to 17, weight: 415.0
Edge from 12 to 18, weight: 349.0
Edge from 12 to 19, weight: 285.0
Edge from 12 to 20, weight: 217.0
Edge from 12 to 21, weight: 138.0
Edge from 12 to 22, weight: 428.0
Edge from 12 to 23, weight: 310.0
Edge from 12 to 24, weight: 200.0
Edge from 12 to 25, weight: 354.0
Edge from 12 to 26, weight: 169.0
Edge from 12 to 27, weight: 241.0
Edge from 12 to 28, weight: 112.0
Edge from 12 to 29, weight: 238.0
Edge from 13 to 13, weight: 0.0
Edge from 13 to 14, weight: 186.0
Edge from 13 to 15, weight: 117.0
Edge from 13 to 16, weight: 75.0
Edge from 13 to 17, weight: 231.0
Edge from 13 to 18, weight: 165.0
Edge from 13 to 19, weight: 81.0
Edge from 13 to 20, weight: 85.0
Edge from 13 to 21, weight: 92.0
Edge from 13 to 22, weight: 230.0
Edge from 13 to 23, weight: 184.0
Edge from 13 to 24, weight: 74.0
Edge from 13 to 25, weight: 150.0
Edge from 13 to 26, weight: 208.0
Edge from 13 to 27, weight: 104.0
Edge from 13 to 28, weight: 158.0
Edge from 13 to 29, weight: 206.0
Edge from 14 to 14, weight: 0.0
Edge from 14 to 15, weight: 69.0
Edge from 14 to 16, weight: 191.0
Edge from 14 to 17, weight: 59.0
Edge from 14 to 18, weight: 35.0
Edge from 14 to 19, weight: 125.0
Edge from 14 to 20, weight: 167.0
Edge from 14 to 21, weight: 255.0
Edge from 14 to 22, weight: 44.0
Edge from 14 to 23, weight: 309.0
Edge from 14 to 24, weight: 245.0
Edge from 14 to 25, weight: 169.0
Edge from 14 to 26, weight: 327.0
Edge from 14 to 27, weight: 246.0
Edge from 14 to 28, weight: 335.0
Edge from 14 to 29, weight: 288.0
Edge from 15 to 15, weight: 0.0
Edge from 15 to 16, weight: 122.0
Edge from 15 to 17, weight: 122.0
```

```
Edge from 15 to 18, weight: 56.0
Edge from 15 to 19, weight: 56.0
Edge from 15 to 20, weight: 108.0
Edge from 15 to 21, weight: 175.0
Edge from 15 to 22, weight: 113.0
Edge from 15 to 23, weight: 240.0
Edge from 15 to 24, weight: 176.0
Edge from 15 to 25, weight: 125.0
Edge from 15 to 26, weight: 280.0
Edge from 15 to 27, weight: 177.0
Edge from 15 to 28, weight: 266.0
Edge from 15 to 29, weight: 243.0
Edge from 16 to 16, weight: 0.0
Edge from 16 to 17, weight: 244.0
Edge from 16 to 18, weight: 178.0
Edge from 16 to 19, weight: 66.0
Edge from 16 to 20, weight: 160.0
Edge from 16 to 21, weight: 161.0
Edge from 16 to 22, weight: 235.0
Edge from 16 to 23, weight: 118.0
Edge from 16 to 24, weight: 62.0
Edge from 16 to 25, weight: 92.0
Edge from 16 to 26, weight: 277.0
Edge from 16 to 27, weight: 55.0
Edge from 16 to 28, weight: 155.0
Edge from 16 to 29, weight: 275.0
Edge from 17 to 17, weight: 0.0
Edge from 17 to 18, weight: 66.0
Edge from 17 to 19, weight: 178.0
Edge from 17 to 20, weight: 198.0
Edge from 17 to 21, weight: 286.0
Edge from 17 to 22, weight: 77.0
Edge from 17 to 23, weight: 362.0
Edge from 17 to 24, weight: 287.0
Edge from 17 to 25, weight: 228.0
Edge from 17 to 26, weight: 358.0
Edge from 17 to 27, weight: 299.0
Edge from 17 to 28, weight: 380.0
Edge from 17 to 29, weight: 319.0
Edge from 18 to 18, weight: 0.0
Edge from 18 to 19, weight: 112.0
Edge from 18 to 20, weight: 132.0
Edge from 18 to 21, weight: 220.0
Edge from 18 to 22, weight: 79.0
Edge from 18 to 23, weight: 296.0
Edge from 18 to 24, weight: 232.0
Edge from 18 to 25, weight: 181.0
Edge from 18 to 26, weight: 292.0
Edge from 18 to 27, weight: 233.0
Edge from 18 to 28, weight: 314.0
Edge from 18 to 29, weight: 253.0
Edge from 19 to 19, weight: 0.0
Edge from 19 to 20, weight: 128.0
Edge from 19 to 21, weight: 167.0
Edge from 19 to 22, weight: 169.0
Edge from 19 to 23, weight: 179.0
Edge from 19 to 24, weight: 120.0
Edge from 19 to 25, weight: 69.0
Edge from 19 to 26, weight: 283.0
Edge from 19 to 27, weight: 121.0
Edge from 19 to 28, weight: 213.0
Edge from 19 to 29, weight: 281.0
```

```
Edge from 20 to 20, weight: 0.0
Edge from 20 to 21, weight: 88.0
Edge from 20 to 22, weight: 211.0
Edge from 20 to 23, weight: 269.0
Edge from 20 to 24, weight: 159.0
Edge from 20 to 25, weight: 197.0
Edge from 20 to 26, weight: 172.0
Edge from 20 to 27, weight: 189.0
Edge from 20 to 28, weight: 182.0
Edge from 20 to 29, weight: 135.0
Edge from 21 to 21, weight: 0.0
Edge from 21 to 22, weight: 299.0
Edge from 21 to 23, weight: 229.0
Edge from 21 to 24, weight: 104.0
Edge from 21 to 25, weight: 236.0
Edge from 21 to 26, weight: 110.0
Edge from 21 to 27, weight: 149.0
Edge from 21 to 28, weight: 97.0
Edge from 21 to 29, weight: 108.0
Edge from 22 to 22, weight: 0.0
Edge from 22 to 23, weight: 353.0
Edge from 22 to 24, weight: 289.0
Edge from 22 to 25, weight: 213.0
Edge from 22 to 26, weight: 371.0
Edge from 22 to 27, weight: 290.0
Edge from 22 to 28, weight: 379.0
Edge from 22 to 29, weight: 332.0
Edge from 23 to 23, weight: 0.0
Edge from 23 to 24, weight: 121.0
Edge from 23 to 25, weight: 162.0
Edge from 23 to 26, weight: 345.0
Edge from 23 to 27, weight: 80.0
Edge from 23 to 28, weight: 189.0
Edge from 23 to 29, weight: 342.0
Edge from 24 to 24, weight: 0.0
Edge from 24 to 25, weight: 154.0
Edge from 24 to 26, weight: 220.0
Edge from 24 to 27, weight: 41.0
Edge from 24 to 28, weight: 93.0
Edge from 24 to 29, weight: 218.0
Edge from 25 to 25, weight: 0.0
Edge from 25 to 26, weight: 352.0
Edge from 25 to 27, weight: 147.0
Edge from 25 to 28, weight: 247.0
Edge from 25 to 29, weight: 350.0
Edge from 26 to 26, weight: 0.0
Edge from 26 to 27, weight: 265.0
Edge from 26 to 28, weight: 178.0
Edge from 26 to 29, weight: 39.0
Edge from 27 to 27, weight: 0.0
Edge from 27 to 28, weight: 124.0
Edge from 27 to 29, weight: 263.0
Edge from 28 to 28, weight: 0.0
Edge from 28 to 29, weight: 199.0
Edge from 29 to 29, weight: 0.0
```

In [12]:

```
graph_kruskal = Kruskal(graph)
show(graph_kruskal)
```

```
Graph res Kruskal has 29 nodes and 28 edges.
Nodes:
Node 1, data: [1150.0, 1760.0], parent: No parent for node
```

```
Node 2, data: [630.0, 1660.0], parent: No parent for node
Node 3, data: [40.0, 2090.0], parent: No parent for node
Node 4, data: [750.0, 1100.0], parent: No parent for node
Node 5, data: [750.0, 2030.0], parent: No parent for node
Node 6, data: [1030.0, 2070.0], parent: No parent for node
Node 7, data: [1650.0, 650.0], parent: No parent for node
Node 8, data: [1490.0, 1630.0], parent: No parent for node
Node 9, data: [790.0, 2260.0], parent: No parent for node
Node 10, data: [710.0, 1310.0], parent: No parent for node
Node 11, data: [840.0, 550.0], parent: No parent for node
Node 12, data: [1170.0, 2300.0], parent: No parent for node
Node 13, data: [970.0, 1340.0], parent: No parent for node
Node 14, data: [510.0, 700.0], parent: No parent for node
Node 15, data: [750.0, 900.0], parent: No parent for node
Node 16, data: [1280.0, 1200.0], parent: No parent for node
Node 17, data: [230.0, 590.0], parent: No parent for node
Node 18, data: [460.0, 860.0], parent: No parent for node
Node 19, data: [1040.0, 950.0], parent: No parent for node
Node 20, data: [590.0, 1390.0], parent: No parent for node
Node 21, data: [830.0, 1770.0], parent: No parent for node
Node 22, data: [490.0, 500.0], parent: No parent for node
Node 23, data: [1840.0, 1240.0], parent: No parent for node
Node 24, data: [1260.0, 1500.0], parent: No parent for node
Node 25, data: [1280.0, 790.0], parent: No parent for node
Node 26, data: [490.0, 2130.0], parent: No parent for node
Node 27, data: [1460.0, 1420.0], parent: No parent for node
Node 28, data: [1260.0, 1910.0], parent: No parent for node
Node 29, data: [360.0, 1980.0], parent: No parent for node
Edges:
Edge from 10 to 20, weight: 28.0
Edge from 14 to 18, weight: 35.0
Edge from 4 to 15, weight: 38.0
Edge from 26 to 29, weight: 39.0
Edge from 24 to 27, weight: 41.0
Edge from 2 to 21, weight: 42.0
Edge from 4 to 10, weight: 42.0
Edge from 8 to 27, weight: 43.0
Edge from 14 to 22, weight: 44.0
Edge from 1 to 28, weight: 45.0
Edge from 5 to 9, weight: 46.0
Edge from 6 to 12, weight: 55.0
Edge from 16 to 27, weight: 55.0
Edge from 15 to 18, weight: 56.0
Edge from 15 to 19, weight: 56.0
Edge from 5 to 26, weight: 57.0
Edge from 10 to 13, weight: 57.0
Edge from 14 to 17, weight: 59.0
Edge from 6 to 28, weight: 60.0
Edge from 5 to 6, weight: 61.0
Edge from 1 to 21, weight: 65.0
Edge from 16 to 19, weight: 66.0
Edge from 1 to 24, weight: 67.0
Edge from 19 to 25, weight: 69.0
Edge from 3 to 29, weight: 77.0
Edge from 11 to 15, weight: 79.0
Edge from 23 to 27, weight: 80.0
Edge from 7 to 25, weight: 95.0
```

Question 3

a. Implémenter les deux heuristiques d'accélération

Dans l'implementation des algortihmes ci-dessous, nous ne procédons pas à l'intégration des heuristiques car cela n'est pas explicitement demandé dans les consignes de cette phase 2 de projet. Cependant, il serait facile d'ajouter un argument dans les fonctions Prim et Kruskal, afin de laisser la liberté à l'utilisateur d'activer ou non ces heuristiques.

In [13]:

```
"""
    find_root!(CC::ConnexComponent)

Trouve la racine de la composante connexe `CC`. Si la racine est déjà détermin
la fonction la retourne simplement. Sinon, elle la détermine en cherchant le p
nœud sans parent et en le définissant comme racine. Ensuite, elle fait pointer
les nœuds de la composante directement vers la racine.

### Arguments
- `CC` : une composante connexe de type `ConnexComponent`.

### Retourne
La racine de la composante connexe.

### Modification en place
Les nœuds de la composante connexe sont modifiés pour pointer vers la racine t
"""
function find_root!(CC::ConnexComponent)

  # Renvoie la racine de la composante si elle n'est pas déjà déterminée
  if CC.root !== nothing
    return CC.root
  end

  root = CC.nodes[1]
  for node in CC.nodes
    # On s'arrête dès que l'on trouve un noeud sans parent
    if node.parent === nothing
      root = node
      root.rank += 1
      CC.root = root
      break
    end
  end

  # On fait pointer les nœuds directement vers la racine.
  for node in CC.nodes
    if node === root
      continue
    else
      node.parent = root
    end
  end
  root
end


"""
```

```
    union_roots!(root1::AbstractNode, root2::AbstractNode)

Lier deux racines d'arbres en union-find. La racine de rang inférieur est liée
à la racine de rang supérieur. Si les rangs sont égaux, une des racines est li
à l'autre et son rang est augmenté de 1.

### Arguments
- `root1` : le premier noeud racine.
- `root2` : le second noeud racine.

### Modification en place
Les parents et les rangs des racines sont potentiellement modifiés pour reflét
"""
function union_roots!(root1::AbstractNode, root2::AbstractNode)

  # Lie la racine de rang inférieur à la racine de rang supérieur
  if root1.rank > root2.rank
      root2.parent = root1
  elseif root1.rank < root2.rank
      root1.parent = root2
  else
      # Si les deux racines ont le même rang, lie l'une à l'autre et augmentez
      root2.parent = root1
      root1.rank += 1
  end
  return
end

"""
    union_all!(CC1::ConnexComponent, CC2::ConnexComponent)

Unir deux composantes connexes. Trouve les racines des deux composantes
et les unit en utilisant `union_roots!`. Ensuite, les nœuds de la composante
de rang inférieur sont ajoutés à ceux de la composante de rang supérieur.

### Arguments
- `CC1` : la première composante connexe.
- `CC2` : la seconde composante connexe.

### Retourne
La liste des nœuds de la composante connexe qui a absorbé l'autre.

### Modification en place
Les nœuds de la composante connexe de rang inférieur sont déplacés vers celle
"""
function union_all!(CC1::ConnexComponent, CC2::ConnexComponent)
  # Trouver les racines des deux composantes
  root1 = find_root!(CC1)
  root2 = find_root!(CC2)

  # Si les racines sont déjà les mêmes, rien à faire
  if root1 === root2
      return
  end

  # Sinon, unir les deux racines
  union_roots!(root1, root2)

  # Ajoute les noeuds de la composante fille aux noeuds de la composante mère
```

```
    if root1.rank > root2.rank
      append!(CC1.nodes, CC2.nodes)
      empty!(CC2.nodes)
      return CC1.nodes
    else
      append!(CC2.nodes, CC1.nodes)
      empty!(CC2.nodes)
      return CC2.nodes
    end
  end
```

```
union_all!
```

In [14]:

```
CC1 = ConnexComponent("cc1", [Node("1", 0.5)])
for i = 2:10
  push!(CC1.nodes, Node("n$i", rand(1)[1]))
end
CC2 = ConnexComponent("cc2", [Node("11", 0.8)])
for i = 2:10
  push!(CC2.nodes, Node("n$(i+10)", rand(1)[1]))
end

union_all!(CC1, CC2)

CC3 = ConnexComponent("cc3", [Node("21", 1.5)])
for i = 2:5
  push!(CC3.nodes, Node("n$(i+20)", rand(1)[1]))
end

union_all!(CC1, CC3)
```

```
25-element Vector{Node{Float64}}:
 Node{Float64}("1", 0.5, nothing, 2)
 Node{Float64}("n2", 0.7234475007599427, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n3", 0.29164408483474036, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n4", 0.7592378675006993, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n5", 0.10727113169499436, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n6", 0.149914987447492, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n7", 0.18258738685811726, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n8", 0.7747372110090457, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n9", 0.7531284038568216, Node{Float64}("1", 0.5, nothing, 2), 0)
 Node{Float64}("n10", 0.010014518103752312, Node{Float64}("1", 0.5, nothing, 2),
0)
 ⋮
 Node{Float64}("n17", 0.29720123908756546, Node{Float64}("11", 0.8, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("n18", 0.5905288416545136, Node{Float64}("11", 0.8, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("n19", 0.3151726880913611, Node{Float64}("11", 0.8, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("n20", 0.14380374291564624, Node{Float64}("11", 0.8, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("21", 1.5, Node{Float64}("1", 0.5, nothing, 2), 1)
 Node{Float64}("n22", 0.5533709467367504, Node{Float64}("21", 1.5, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("n23", 0.9125732922208051, Node{Float64}("21", 1.5, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("n24", 0.11955485867571669, Node{Float64}("21", 1.5, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
 Node{Float64}("n25", 0.105183536265677, Node{Float64}("21", 1.5, Node{Float64}
("1", 0.5, nothing, 2), 1), 0)
```

```
( 1 , 0.5, nothing, 2), 1), 0)
```

b.1. Montrons que le rang d’un nœud sera toujours inférieur à $|S| - 1$.

Soit $|S|$ le nombre d'éléments dans l'ensemble. Le rang d'un nœud est défini comme la hauteur de l'arbre.

À l'initialisation, chaque nœud est son propre parent et a un rang de 0.

Lorsque deux ensembles de même rang r sont unis, le rang du nouveau parent augmente de 1, devenant $r + 1$.

Pour qu'un nœud ait un rang r, il doit y avoir eu r unions où deux arbres de même hauteur ont été unis. Pour que cela se produise, il doit y avoir 2^r nœuds dans chaque ensemble avant l'union, car chaque union double le nombre d'éléments que l'arbre peut potentiellement avoir.

Donc, pour obtenir un rang r, il faut un minimum de 2^r éléments dans l'ensemble.

Si le rang d'un nœud était $|S| - 1$, cela impliquerait que l'ensemble contient au moins $2^{(|S|-1)}$ éléments. Mais c'est impossible, car cela signifierait que l'ensemble contiendrait plus d'éléments qu'il n'en existe dans l'ensemble total S, ce qui est un non-sens.

Ainsi, le rang d'un nœud doit être strictement inférieur à $|S| - 1$, car pour avoir un rang $|S| - 1$, l'ensemble devrait contenir au moins $2^{(|S|-1)}$ éléments, ce qui dépasse le nombre total d'éléments dans l'ensemble S.

b.2. Montrons ensuite que ce rang sera en fait toujours inférieur à $\lfloor log_2(|S|) \rfloor$

Induction de Base:

Initialement, tous les éléments sont dans des ensembles distincts, donc le rang de chaque nœud est 0. Puisque $log(1) = 0$, l'affirmation est vraie pour $|S| = 1$.

Induction Hypothèse:

Supposons que l'affirmation soit vraie pour tous les ensembles de taille k, c'est-à-dire que le rang de n'importe quel nœud est inférieur à $\lfloor log_2(|k|) \rfloor$.

Induction Étape:

Considérons deux ensembles de taille maximale k qui sont unis. Par hypothèse d'induction, le rang de chaque arbre est au plus $\lfloor log_2(|k|) \rfloor$.

Lors de l'union, si les rangs des arbres sont différents, le rang du nouvel arbre reste le même que celui de l'arbre avec le plus grand rang, qui est $\lfloor log_2(|k|) \rfloor$ ou moins. Si les rangs sont les mêmes, disons r, alors après l'union, le rang du nouvel arbre est $r + 1$. Mais pour que l'arbre ait un rang r, il doit y avoir 2^r éléments. En unissant deux de ces arbres, nous avons $2 \times 2^r = 2^{r+1}$ éléments dans le nouvel ensemble.

Maintenant, montrons que le rang est toujours inférieur à $\lfloor log_2(|S|) \rfloor$:

Après l'union, le nombre total d'éléments est 2^{r+1}. Le rang du nouvel ensemble est $r + 1$, qui est le log2 de 2^{r+1}. Donc, $r + 1 = log_2(2^{r+1})$. Par conséquent, $r + 1 \leq \lfloor log_2(|2k|) \rfloor$ car $2k = 2^{r+1}$. Mais $2k$ est au plus $\mid S \mid$ après l'union, donc $r + 1 \leq \lfloor log_2(|S|) \rfloor$. En conclusion, à chaque étape de l'union par rang, le rang d'un nœud est limité par le logarithme en base 2 du nombre d'éléments dans l'ensemble, arrondi à l'entier inférieur. Cela prouve que le rang d'un nœud dans une structure Union-Find avec union par rang est toujours inférieur à $\lfloor log_2(|S|) \rfloor$.

Question 4

a. Implémenter l'algorithme de Prim

In [16]:

```
"""
    neighbours_node(n::AbstractNode, graph::ExtendedGraph)

Retourne un ensemble d'arêtes du graph `graph` contenant le nœud `n`, triées p
Cela permet de déterminer les voisins d'un nœud dans le cadre d'algorithmes gr

### Arguments
- `n` : Le nœud pour lequel les voisins sont recherchés.
- `graph` : Le graphe de type `ExtendedGraph` contenant le nœud `n`.

### Retourne
Un vecteur des arêtes voisines du nœud `n`, trié par ordre croissant de poids.

### Exemples
```julia
neighbours = neighbours_node(monNode, monGraphe)
"""
function neighbours_node(n::AbstractNode, graph::ExtendedGraph)
  E = graph.edges
  neighbours = [] # vecteur qui contiendra les arêtes voisines

  for e in E
    if e.start_node == n || e.end_node == n # si n appartient à l'arête e
      push!(neighbours, e)
    end
  end
  return sort(neighbours, by=edge -> edge.weight)
end

"""
    Prim(graph::ExtendedGraph; st_node::AbstractNode = graph.nodes[1])

Implémente l'algorithme de Prim pour un graphe donné. Cet algorithme construit
couvrant de poids minimum à partir d'un graphe pondéré. Le noeud de départ par
nœud du graphe, mais un noeud de départ différent peut être spécifié.

### Arguments
- `graph` : Le graphe sur lequel appliquer l'algorithme de Prim
```
```

```
- `graph` : Le graphe sur lequel appliquer l'algorithme de Prim.
- `st_node` : Le noeud de départ pour l'algorithme. Par défaut, c'est le premi

### Retourne
Le graphe résultant après application de l'algorithme de Prim, sous forme d'un

### Exemples
```julia
grapheResultant = Prim(monGraphe)
grapheResultant = Prim(monGraphe, st_node=monNode)
"""
function Prim(graph::ExtendedGraph; st_node::AbstractNode=graph.nodes[1])
  N = graph.nodes
  E = graph.edges
  graph_res = ExtendedGraph("res Prim", N, typeof(E[1])[])

  # On recherche st_node dans le graphe donné
  idx = findfirst(x -> x == st_node, N)
  if idx === nothing
    @warn "starting node not in graph"
    return
  end

  # Création de la file de priorité pour traiter les noeuds
  q = PriorityQueue([PriorityItem(Inf, n) for n in N])
  priority!(q.items[idx], 0)
  visited = Set{AbstractNode}() # Set pour vérifier les noeuds visités
  parent_map = Dict{AbstractNode, AbstractNode}() # Map pour les parents des n

  # Boucle principale :
  while !isempty(q.items)
    u = pop_lowest!(q)
    push!(visited, u.data)

    # Vérifie si le noeud a un parent et ajoute l'arête correspondante dans gr
    if haskey(parent_map, u.data)
      edge_idx = findfirst(e -> (e.start_node == u.data && e.end_node == paren
      edge = E[edge_idx]
      push!(graph_res.edges, edge)
    end

    neighbours = neighbours_node(u.data, graph)

    for edge in neighbours
      v = edge.start_node === u.data ? edge.end_node : edge.start_node
      if !(v in visited) && edge.weight < get_priority(q, v)
        parent_map[v] = u.data # Utilisez parent_map au lieu de v.parent
        update_priority!(q, v, edge.weight)
      end
    end
  end

  return graph_res
end
```

```
Prim
```

b. Le tester sur l’exemple des notes de cours
```

In [17]:

```
a, b, c, d, e, f, g, h, i = Node("a", 1.0), Node("b", 1.0), Node("c", 1.0), Nc
e1 = Edge(a, b, 4.)
e2 = Edge(b, c, 8.)
e3 = Edge(c, d, 7.)
e4 = Edge(d, e, 9.)
e5 = Edge(e, f, 10.)
e6 = Edge(d, f, 14.)
e7 = Edge(f, c, 4.)
e8 = Edge(f, g, 2.)
e9 = Edge(g, i, 6.)
e10 = Edge(g, h, 1.)
e11 = Edge(a, h, 8.)
e12 = Edge(h, i, 7.)
e13 = Edge(i, c, 2.)
e14 = Edge(b, h, 11.)
G_cours = ExtendedGraph("graphe du cours", [a, b, c, d, e, f, g, h, i],[e1, e2
```

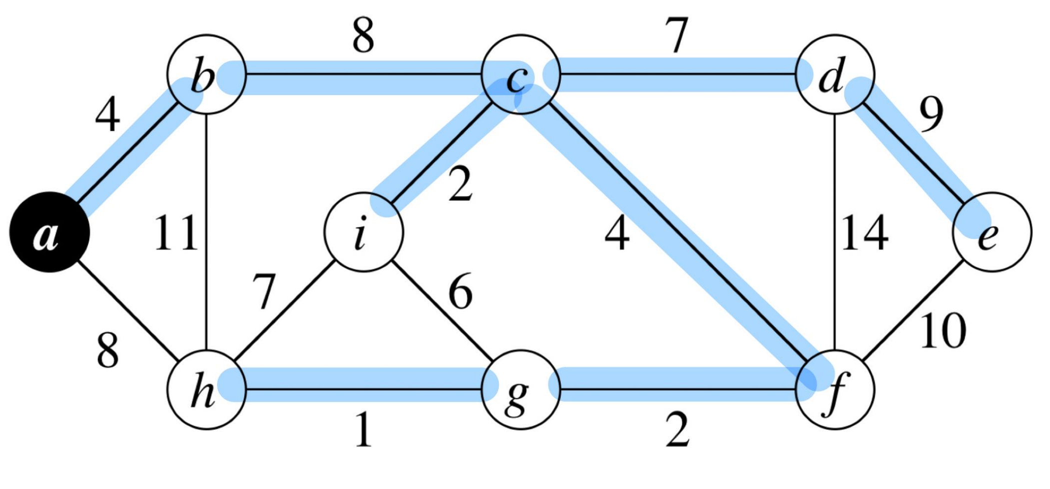
```
ExtendedGraph{Float64, Float64}("graphe du cours", Node{Float64}[Node{Float64}
("a", 1.0, nothing, 0), Node{Float64}("b", 1.0, nothing, 0), Node{Float64}("c",
1.0, nothing, 0), Node{Float64}("d", 1.0, nothing, 0), Node{Float64}("e", 1.0, no
thing, 0), Node{Float64}("f", 1.0, nothing, 0), Node{Float64}("g", 1.0, nothing,
0), Node{Float64}("h", 1.0, nothing, 0), Node{Float64}("i", 1.0, nothing, 0)], Ed
ge{Float64, Float64}[Edge{Float64, Float64}(Node{Float64}("a", 1.0, nothing, 0),
Node{Float64}("b", 1.0, nothing, 0), 4.0), Edge{Float64, Float64}(Node{Float64}
("b", 1.0, nothing, 0), Node{Float64}("c", 1.0, nothing, 0), 8.0), Edge{Float64,
Float64}(Node{Float64}("c", 1.0, nothing, 0), Node{Float64}("d", 1.0, nothing,
0), 7.0), Edge{Float64, Float64}(Node{Float64}("d", 1.0, nothing, 0), Node{Float6
4}("e", 1.0, nothing, 0), 9.0), Edge{Float64, Float64}(Node{Float64}("e", 1.0, no
thing, 0), Node{Float64}("f", 1.0, nothing, 0), 10.0), Edge{Float64, Float64}(Nod
e{Float64}("d", 1.0, nothing, 0), Node{Float64}("f", 1.0, nothing, 0), 14.0), Edg
e{Float64, Float64}(Node{Float64}("f", 1.0, nothing, 0), Node{Float64}("c", 1.0,
nothing, 0), 4.0), Edge{Float64, Float64}(Node{Float64}("f", 1.0, nothing, 0), No
de{Float64}("g", 1.0, nothing, 0), 2.0), Edge{Float64, Float64}(Node{Float64}
("g", 1.0, nothing, 0), Node{Float64}("i", 1.0, nothing, 0), 6.0), Edge{Float64,
Float64}(Node{Float64}("g", 1.0, nothing, 0), Node{Float64}("h", 1.0, nothing,
0), 1.0), Edge{Float64, Float64}(Node{Float64}("a", 1.0, nothing, 0), Node{Float6
4}("h", 1.0, nothing, 0), 8.0), Edge{Float64, Float64}(Node{Float64}("h", 1.0, no
thing, 0), Node{Float64}("i", 1.0, nothing, 0), 7.0), Edge{Float64, Float64}(Node
{Float64}("i", 1.0, nothing, 0), Node{Float64}("c", 1.0, nothing, 0), 2.0), Edge
{Float64, Float64}(Node{Float64}("b", 1.0, nothing, 0), Node{Float64}("h", 1.0, n
othing, 0), 11.0)])
```

In [18]:

```
graph_cours_prim = Prim(G_cours, st_node = a)
show(graph_cours_prim)
```

```
Graph res Prim has 9 nodes and 8 edges.
Nodes:
Node a, data: 1.0, parent: No parent for node
Node b, data: 1.0, parent: No parent for node
Node c, data: 1.0, parent: No parent for node
Node d, data: 1.0, parent: No parent for node
Node e, data: 1.0, parent: No parent for node
Node f, data: 1.0, parent: No parent for node
Node g, data: 1.0, parent: No parent for node
Node h, data: 1.0, parent: No parent for node
Node i, data: 1.0, parent: No parent for node
Edges:
Edge from a to b, weight: 4.0
Edge from b to c, weight: 8.0
Edge from i to c, weight: 2.0
```

```
Edge from f to c, weight: 4.0
Edge from f to g, weight: 2.0
Edge from g to h, weight: 1.0
Edge from c to d, weight: 7.0
Edge from d to e, weight: 9.0
```



Question 6

In [19]:
```
graph = build_graph("../Phase 1/instances/stsp/bays29.tsp", "Graph_Test")
```

```
ExtendedGraph{Vector{Float64}, Float64}("Graph_Test", Node{Vector{Float64}}[Node
{Vector{Float64}}("1", [1150.0, 1760.0], nothing, 0), Node{Vector{Float64}}("2",
[630.0, 1660.0], nothing, 0), Node{Vector{Float64}}("3", [40.0, 2090.0], nothing,
0), Node{Vector{Float64}}("4", [750.0, 1100.0], nothing, 0), Node{Vector{Float6
4}}("5", [750.0, 2030.0], nothing, 0), Node{Vector{Float64}}("6", [1030.0, 2070.
0], nothing, 0), Node{Vector{Float64}}("7", [1650.0, 650.0], nothing, 0), Node{Ve
ctor{Float64}}("8", [1490.0, 1630.0], nothing, 0), Node{Vector{Float64}}("9", [79
0.0, 2260.0], nothing, 0), Node{Vector{Float64}}("10", [710.0, 1310.0], nothing,
0)  …  Node{Vector{Float64}}("20", [590.0, 1390.0], nothing, 0), Node{Vector{Floa
t64}}("21", [830.0, 1770.0], nothing, 0), Node{Vector{Float64}}("22", [490.0, 50
0.0], nothing, 0), Node{Vector{Float64}}("23", [1840.0, 1240.0], nothing, 0), Nod
e{Vector{Float64}}("24", [1260.0, 1500.0], nothing, 0), Node{Vector{Float64}}("2
5", [1280.0, 790.0], nothing, 0), Node{Vector{Float64}}("26", [490.0, 2130.0], no
thing, 0), Node{Vector{Float64}}("27", [1460.0, 1420.0], nothing, 0), Node{Vector
{Float64}}("28", [1260.0, 1910.0], nothing, 0), Node{Vector{Float64}}("29", [360.
0, 1980.0], nothing, 0)], Edge{Vector{Float64}, Float64}[Edge{Vector{Float64}, Fl
oat64}(Node{Vector{Float64}}("1", [1150.0, 1760.0], nothing, 0), Node{Vector{Floa
t64}}("1", [1150.0, 1760.0], nothing, 0), 0.0), Edge{Vector{Float64}, Float64}(No
de{Vector{Float64}}("1", [1150.0, 1760.0], nothing, 0), Node{Vector{Float64}}
("2", [630.0, 1660.0], nothing, 0), 107.0), Edge{Vector{Float64}, Float64}(Node{V
ector{Float64}}("1", [1150.0, 1760.0], nothing, 0), Node{Vector{Float64}}("3", [4
0.0, 2090.0], nothing, 0), 241.0), Edge{Vector{Float64}, Float64}(Node{Vector{Flo
at64}}("1", [1150.0, 1760.0], nothing, 0), Node{Vector{Float64}}("4", [750.0, 110
0.0], nothing, 0), 190.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}
("1", [1150.0, 1760.0], nothing, 0), Node{Vector{Float64}}("5", [750.0, 2030.0],
nothing, 0), 124.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("1", [1
150.0, 1760.0], nothing, 0), Node{Vector{Float64}}("6", [1030.0, 2070.0], nothin
g, 0), 80.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("1", [1150.0,
1760.0], nothing, 0), Node{Vector{Float64}}("7", [1650.0, 650.0], nothing, 0), 31
6.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("1", [1150.0, 1760.0],
nothing, 0), Node{Vector{Float64}}("8", [1490.0, 1630.0], nothing, 0), 76.0), Edg
e{Vector{Float64}, Float64}(Node{Vector{Float64}}("1", [1150.0, 1760.0], nothing,
0), Node{Vector{Float64}}("9", [790.0, 2260.0], nothing, 0), 152.0), Edge{Vector
```

```
{Float64}, Float64}(Node{Vector{Float64}}("1", [1150.0, 1760.0], nothing, 0), Nod
e{Vector{Float64}}("10", [710.0, 1310.0], nothing, 0), 157.0)  …  Edge{Vector{Flo
at64}, Float64}(Node{Vector{Float64}}("26", [490.0, 2130.0], nothing, 0), Node{Ve
ctor{Float64}}("26", [490.0, 2130.0], nothing, 0), 0.0), Edge{Vector{Float64}, Fl
oat64}(Node{Vector{Float64}}("26", [490.0, 2130.0], nothing, 0), Node{Vector{Floa
t64}}("27", [1460.0, 1420.0], nothing, 0), 265.0), Edge{Vector{Float64}, Float64}
(Node{Vector{Float64}}("26", [490.0, 2130.0], nothing, 0), Node{Vector{Float64}}
("28", [1260.0, 1910.0], nothing, 0), 178.0), Edge{Vector{Float64}, Float64}(Node
{Vector{Float64}}("26", [490.0, 2130.0], nothing, 0), Node{Vector{Float64}}("29",
[360.0, 1980.0], nothing, 0), 39.0), Edge{Vector{Float64}, Float64}(Node{Vector{F
loat64}}("27", [1460.0, 1420.0], nothing, 0), Node{Vector{Float64}}("27", [1460.
0, 1420.0], nothing, 0), 0.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float6
4}}("27", [1460.0, 1420.0], nothing, 0), Node{Vector{Float64}}("28", [1260.0, 191
0.0], nothing, 0), 124.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}
("27", [1460.0, 1420.0], nothing, 0), Node{Vector{Float64}}("29", [360.0, 1980.
0], nothing, 0), 263.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("2
8", [1260.0, 1910.0], nothing, 0), Node{Vector{Float64}}("28", [1260.0, 1910.0],
nothing, 0), 0.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("28", [12
60.0, 1910.0], nothing, 0), Node{Vector{Float64}}("29", [360.0, 1980.0], nothing,
0), 199.0), Edge{Vector{Float64}, Float64}(Node{Vector{Float64}}("29", [360.0, 19
80.0], nothing, 0), Node{Vector{Float64}}("29", [360.0, 1980.0], nothing, 0), 0.
0)])
```

Nous observons ci-dessous que pour un graph à n noeud, nous obtenons bien n-1 arêtes.

In [20]:

```
graph_kruskal = Kruskal(graph)
show(graph_kruskal)
```

```
Graph res Kruskal has 29 nodes and 28 edges.
Nodes:
Node 1, data: [1150.0, 1760.0], parent: No parent for node
Node 2, data: [630.0, 1660.0], parent: No parent for node
Node 3, data: [40.0, 2090.0], parent: No parent for node
Node 4, data: [750.0, 1100.0], parent: No parent for node
Node 5, data: [750.0, 2030.0], parent: No parent for node
Node 6, data: [1030.0, 2070.0], parent: No parent for node
Node 7, data: [1650.0, 650.0], parent: No parent for node
Node 8, data: [1490.0, 1630.0], parent: No parent for node
Node 9, data: [790.0, 2260.0], parent: No parent for node
Node 10, data: [710.0, 1310.0], parent: No parent for node
Node 11, data: [840.0, 550.0], parent: No parent for node
Node 12, data: [1170.0, 2300.0], parent: No parent for node
Node 13, data: [970.0, 1340.0], parent: No parent for node
Node 14, data: [510.0, 700.0], parent: No parent for node
Node 15, data: [750.0, 900.0], parent: No parent for node
Node 16, data: [1280.0, 1200.0], parent: No parent for node
Node 17, data: [230.0, 590.0], parent: No parent for node
Node 18, data: [460.0, 860.0], parent: No parent for node
Node 19, data: [1040.0, 950.0], parent: No parent for node
Node 20, data: [590.0, 1390.0], parent: No parent for node
Node 21, data: [830.0, 1770.0], parent: No parent for node
Node 22, data: [490.0, 500.0], parent: No parent for node
Node 23, data: [1840.0, 1240.0], parent: No parent for node
Node 24, data: [1260.0, 1500.0], parent: No parent for node
Node 25, data: [1280.0, 790.0], parent: No parent for node
Node 26, data: [490.0, 2130.0], parent: No parent for node
Node 27, data: [1460.0, 1420.0], parent: No parent for node
Node 28, data: [1260.0, 1910.0], parent: No parent for node
Node 29, data: [360.0, 1980.0], parent: No parent for node
Edges:
```

```
Edges:
Edge from 10 to 20, weight: 28.0
Edge from 14 to 18, weight: 35.0
Edge from 4 to 15, weight: 38.0
Edge from 26 to 29, weight: 39.0
Edge from 24 to 27, weight: 41.0
Edge from 2 to 21, weight: 42.0
Edge from 4 to 10, weight: 42.0
Edge from 8 to 27, weight: 43.0
Edge from 14 to 22, weight: 44.0
Edge from 1 to 28, weight: 45.0
Edge from 5 to 9, weight: 46.0
Edge from 6 to 12, weight: 55.0
Edge from 16 to 27, weight: 55.0
Edge from 15 to 18, weight: 56.0
Edge from 15 to 19, weight: 56.0
Edge from 5 to 26, weight: 57.0
Edge from 10 to 13, weight: 57.0
Edge from 14 to 17, weight: 59.0
Edge from 6 to 28, weight: 60.0
Edge from 5 to 6, weight: 61.0
Edge from 1 to 21, weight: 65.0
Edge from 16 to 19, weight: 66.0
Edge from 1 to 24, weight: 67.0
Edge from 19 to 25, weight: 69.0
Edge from 3 to 29, weight: 77.0
Edge from 11 to 15, weight: 79.0
Edge from 23 to 27, weight: 80.0
Edge from 7 to 25, weight: 95.0
```

In [21]:

```
graph_prim = Prim(graph, st_node = graph.nodes[1])
show(graph_prim)
```

```
Graph res Prim has 29 nodes and 28 edges.
Nodes:
Node 1, data: [1150.0, 1760.0], parent: No parent for node
Node 2, data: [630.0, 1660.0], parent: No parent for node
Node 3, data: [40.0, 2090.0], parent: No parent for node
Node 4, data: [750.0, 1100.0], parent: No parent for node
Node 5, data: [750.0, 2030.0], parent: No parent for node
Node 6, data: [1030.0, 2070.0], parent: No parent for node
Node 7, data: [1650.0, 650.0], parent: No parent for node
Node 8, data: [1490.0, 1630.0], parent: No parent for node
Node 9, data: [790.0, 2260.0], parent: No parent for node
Node 10, data: [710.0, 1310.0], parent: No parent for node
Node 11, data: [840.0, 550.0], parent: No parent for node
Node 12, data: [1170.0, 2300.0], parent: No parent for node
Node 13, data: [970.0, 1340.0], parent: No parent for node
Node 14, data: [510.0, 700.0], parent: No parent for node
Node 15, data: [750.0, 900.0], parent: No parent for node
Node 16, data: [1280.0, 1200.0], parent: No parent for node
Node 17, data: [230.0, 590.0], parent: No parent for node
Node 18, data: [460.0, 860.0], parent: No parent for node
Node 19, data: [1040.0, 950.0], parent: No parent for node
Node 20, data: [590.0, 1390.0], parent: No parent for node
Node 21, data: [830.0, 1770.0], parent: No parent for node
Node 22, data: [490.0, 500.0], parent: No parent for node
Node 23, data: [1840.0, 1240.0], parent: No parent for node
Node 24, data: [1260.0, 1500.0], parent: No parent for node
Node 25, data: [1280.0, 790.0], parent: No parent for node
Node 26, data: [490.0, 2130.0], parent: No parent for node
Node 27, data: [1460.0, 1420.0], parent: No parent for node
```

```
Node 28, data: [1260.0, 1910.0], parent: No parent for node
Node 29, data: [360.0, 1980.0], parent: No parent for node
Edges:
Edge from 1 to 28, weight: 45.0
Edge from 6 to 28, weight: 60.0
Edge from 6 to 12, weight: 55.0
Edge from 5 to 6, weight: 61.0
Edge from 5 to 9, weight: 46.0
Edge from 5 to 26, weight: 57.0
Edge from 26 to 29, weight: 39.0
Edge from 1 to 21, weight: 65.0
Edge from 2 to 21, weight: 42.0
Edge from 2 to 20, weight: 67.0
Edge from 10 to 20, weight: 28.0
Edge from 4 to 10, weight: 42.0
Edge from 4 to 15, weight: 38.0
```