Data Structure & Algorithms in Coluna.jl

Problem

```
struct Problem <: AbstractProblem
    original_formulation::Union{Nothing, Formulation}
    re_formulation::Union{Nothing, Reformulation}
    form_counter::Counter # 0 is for original form
    default_optimizer_builder::Function
end
```

Formulation

```
struct Formulation{Duty <: AbstractFormDuty}  <: AbstractFormulation
    uid::Int
    var_counter::Counter
    constr_counter::Counter
    parent_formulation::Union{AbstractFormulation, Nothing}
    optimizer::AbstractOptimizer
    manager::FormulationManager
    obj_sense::Type{<:AbstractObjSense}
    buffer::FormulationBuffer
end
```

Reformulation

```
struct Reformulation <: AbstractFormulation
    strategy::GlobalStrategy
    parent::Union{Nothing, AbstractFormulation}
    master::Union{Nothing, Formulation}
    dw_pricing_subprs::Vector{AbstractFormulation} # vector of Formulation or
Reformulation
    benders_sep_subprs::Vector{AbstractFormulation}
    dw_pricing_sp_lb::Dict{FormId, Id} # Attribute has ambiguous name
    dw_pricing_sp_ub::Dict{FormId, Id}
end

function optimize!(reformulation::Reformulation)
```

```
    opt_result = apply!(GlobalStrategy, reformulation)
    opt_result.primal_sols = [proj_cols_on_rep(
        getbestprimalsol(opt_result), getmaster(reformulation)
    )]
    return opt_result
end
```

Variable

```
 struct VarData <: AbstractVcData
    cost::Float64
    lb::Float64
    ub::Float64
    kind::VarKind
    sense::VarSense
    inc_val::Float64
    is_active::Bool
    is_explicit::Bool
end

struct Variable <: AbstractVarConstr
    id::Id{Variable}
    name::String
    duty::Type{<: AbstractVarDuty}
    perene_data::VarData
    cur_data::VarData
    moirecord::MoiVarRecord
end
```

Constraint

```
 struct ConstrData <: AbstractVcData
    rhs::Float64
    kind::ConstrKind
    sense::ConstrSense
    inc_val::Float64
    is_active::Bool
    is_explicit::Bool
end

struct Constraint <: AbstractVarConstr
    id::Id{Constraint}
    name::String
    duty::Type{<: AbstractConstrDuty}
```

```
    perene_data::ConstrData
    cur_data::ConstrData
    moirecord::MoiConstrRecord
end
```

VC Id

```
 struct Id{VC <: AbstractVarConstr}
    uid::Int
    form_uid::Int
    proc_uid::Int
    sequence_nb::Int
    _hash::Int
end
```

Manager

```
struct FormulationManager
    vars::VarDict
    constrs::ConstrDict
    coefficients::MembMatrix # rows = constraints, cols = variables
    primal_sp_sols::VarMatrix # rows = variables, cols = solutions
    dual_sp_sols::ConstrMatrix # rows = constraints, cols = solutions
    expressions::VarMatrix  # rows = expressions, cols = variables
end
```

Members Vector / Matrix

```
struct MembersVector{I,K,T} <: AbstractMembersContainer
    elements::Dict{I, K}
    records::Dict{I, T}
end

struct MembersMatrix{I,K,J,L,T} <: AbstractMembersContainer
    cols::MembersVector{I,K,MembersVector{J,L,T}}
    rows::MembersVector{J,L,MembersVector{I,K,T}}
end
```

Algorithms

```
function prepare!(T::Type{<:AbstractAlgorithm}, formulation, node, strategy_rec,
parameters)
```

```
   ...
end

function run!(T::Type{<:AbstractAlgorithm}, formulation, node, strategy_rec,
parameters)
    ...
end

function apply!(A::Type{<:AbstractAlgorithm}, formulation, node, strategy_rec,
parameters)
    prepare!(formulation, node)
    setalgorithm!(strategy_rec, A)
    TO.@timeit _to string(A) begin
        TO.@timeit _to "prepare" begin
            prepare!(A, formulation, node, strategy_rec, parameters)
        end
        TO.@timeit _to "run" begin
            record = run!(A, formulation, node, strategy_rec, parameters)
        end
    end
    set_algorithm_record!(node, A, record)
    return record
end
```

Strategy

```
struct GlobalStrategy <: AbstractStrategy
    conquer::Type{<:AbstractConquerStrategy}
    divide::Type{<:AbstractDivideStrategy}
    tree_search::Type{<:AbstractTreeSearchStrategy}
end

struct SimpleBnP <: AbstractConquerStrategy end

function apply!(::Type{SimpleBnP}, reform, node, strategy_rec::StrategyRecord,
params)
    colgen_rec = apply!(ColumnGeneration, reform, node, strategy_rec, params)
    if colgen_rec.proven_infeasible
        node.status.proven_infeasible = true
        return
    end
    ip_gap(colgen_rec.incumbents) <= 0 && return
    mip_rec = apply!(MasterIpHeuristic, reform, node, strategy_rec, params)
    return
end
```

Solution

```
struct PrimalSolution{S <: AbstractObjSense} <: AbstractSolution
    bound::PrimalBound{S}
    sol::MembersVector{Id{Variable}, Variable, Float64}
end

struct DualSolution{S <: AbstractObjSense} <: AbstractSolution
    bound::DualBound{S}
    sol::MembersVector{Id{Constraint}, Constraint, Float64}
end
```

Automated Decomposition

```
function reformulate!(prob::Problem, annotations::Annotations,
                      strategy::GlobalStrategy)
    decomposition_tree = annotations.tree
    root = BD.getroot(decomposition_tree)
    reform = Reformulation(prob, strategy)
    set_re_formulation!(prob, reform)
    buildformulations!(prob, annotations, reform, reform, root)
end
```

Demo

```
module GeneralizedAssignment
    using JuMP, BlockDecomposition
    include("data.jl")
    include("model.jl")
end
function generalized_assignment_tests()
    data = CLD.GeneralizedAssignment.data("play2.txt")

    coluna = JuMP.with_optimizer(Coluna.Optimizer,
                                 params = CL.Params(
                                     global_strategy =
CL.GlobalStrategy(CL.SimpleBnP,
CL.SimpleBranching, CL.DepthFirst)
                                 ),
                                 default_optimizer =
with_optimizer(GLPK.Optimizer)
                                 )

    problem, x, dec = CLD.GeneralizedAssignment.model(data, coluna)
```

```
    JuMP.optimize!(problem)
end
```