Again, let me reiterate the importance of this result. We can now use the fact that sets of \( m \) indicies of the matrix \( A \) give rise to vertices of the feasible polytope.

**Overview of Simplex Method** The simplex method, then, moves from vertex to vertex on the feasible polytope by adding and subtracting indicies from the subset \( B \) to move between vertices. At every stage, it moves to another vertex with a non-increasing objective.

"Presolve" (Remove redundant rows from \( A \))
"Phase 1" Find a feasible vertex / basic point
"Simplex" Find the dual variables / Lagrange multipliers for that point
While KKT conditions are not true
Move to another point
Find the dual variables / Lagrange multipliers for new point

The first step of finding a feasible vertex or basic point is called a Phase 1 solution. We'll cover that soon.

**Columns to basic feasible points and Lagrange multipliers** What we now need to address is how to find the dual variables or Lagrange multipliers at a basic feasible point (vertex). If we can get these (and we can) then we can determine when to stop the simplex method.

Let \( x \) be a basic feasible point. Let \( B \) be the subset of columns at any basic feasible point. Then \( Bx_B = b \) and we can permute \( x \) into \( [x_B^T \ x_N^T]^T \), where \( x_B \) are non-zero elements ("the basic elements") of \( x \) and \( x_N = 0 \) be the zero elements ("the non-basic elements"). Partition \( s \) (multipliers) and \( c \) (objective) conformally into \( s_B, s_N \) and \( c_B, c_N \).

We can write:

\[
Ax = Bx_B + Nx_N = b
\]

where \( x_B = B^{-1}b \) and \( x_B \geq 0 \). We set \( s_B = 0 \) because all the elements of \( x_B \) handle that portion of the constraint. We now use the remaining KKT conditions to find \( \lambda \) and \( s_N \). Note that

\[
B^T \lambda = c_B \text{ and } N^T \lambda + s_N = c_N.
\]

The first equation defines \( \lambda \) because \( B^T \) is non-singular. The second equation can then be solved for \( s_N \). Consequently, we can find the dual variables for a solution. However, \( s \) need not be non-negative.

The material here is from Chapter 13 in Nocedal and Wright, but some of the geometry comes from Griva, Sofer, and Nash.
function simplex_point(s::SimplexState)
    binds = state.bset # basic variable indices
    ninds = setdiff(1:size(A,1),binds) # non-basic
    B = state.A[:,binds]
    N = state.A[:,ninds]
    cb = state.c[binds]
    cn = state.c[ninds]
    if rank(B) != m
        return (:Infeasible, SimplexPoint(eltype(c), B, N))
    end
    xb = B\b
    x = zeros(eltype(xb), n)
    x[binds] = xb
    x[ninds] = zero(eltype(xb))
    lam = B'\cb
    sn = cn - N'*lam
    if any(xb .< 0)
        return (:Infeasible, SimplexPoint(x, binds, ninds, lam, sn, B, N))
    else
        if all(sn .≥ 0)
            return (:Solution, SimplexPoint(x, binds, ninds, lam, sn, B, N))
        else
            return (:Feasible, SimplexPoint(x, binds, ninds, lam, sn, B, N))
        end
    end
end

function simplex_step!(state::SimplexState)
    stat,p::SimplexPoint = simplex_point(state)
    if stat == :Solution
        return stat, p
    elseif state == :Infeasible
        return :Breakdown, p
    else # we have a BFP
        qn = indmin(p.sn) # take the Dantzig index to add to basic
        q = p.ninds[qn] # translate index
        @assert all(state.A[:,q] == p.N[:,qn])
        d = p.B \ state.A[:,q]
        xq = p.x[binds]/d
        xq[d .< eps(eltype(xq))] = Inf
        pb = indmin(xq)
        pind = p.binds[pb] # translate index
        @assert state.bset[pb] == pind
        state.bset[pb] = q
        return stat, p
    end
end