Contents

1 Main program 1

2 Formatting rules for HPF/F90 files 4
1 Main program

This program is an example of how to use the spanning-tree routines in the modules *Graph Algorithms* and *Network Spanning Trees*, and it also performs some timing of these, in particular, it compares the time it takes to build an MST with the greedy Kruskal approach from scratch as opposed to updating it using a non-greedy approach when the weights change randomly by at most disorder percent (so disorder = 0 means the weights did not change at all, while disorder = 1 means the weights changed almost completely). The perturbations in the weights are drawn from a uniform distribution here, but that can of course be changed to study the influence of large (but rare) weight fluctuations:
"WEAVE.f90" 1.1:

PROGRAM Time_MST
USE Precision  // Kind parameters
USE Error_Handling
USE System_Monitors
USE Random_Numbers
USE Sorting_Ranking
USE Initialization_Termination
USE Network_Data_Structures
USE Lattice_Geometry
USE Network_Geometry
USE Network_Graphics
USE Lattice_Network_Optimization
USE Graph_Algorithms
USE Network_Spanning_Trees
USE Power_Cost_Parameters
IMPLICIT NONE

REAL (KIND = r_wp), DIMENSION (:), ALLOCATABLE :: weights_changes, arcs_weights
INTEGER (KIND = i_wp) :: arc, node, head, tail, n_pivoted
REAL (KIND = r_wp) :: disorder, total_weight

CALL StartProgram
CALL InitializeLatticeNetworkProblem
CALL CreateLatticeNetworkProblem
ALLOCATE (arcs_weights (-n_special_arcs : n_arcs), weights_changes (-n_special_arcs : n_arcs))
dbg_graph_algs = T

OPEN (FILE = "MST.dat", UNIT = 101, STATUS = "UNKNOWN", ACTION = "WRITE", POSITION = "APPEND")
OPEN (FILE = "MST.total.dat", UNIT = 201, STATUS = "UNKNOWN", ACTION = "WRITE",
POSITION = "APPEND")

WRITE (UNIT = 101, FMT = "(2I5,15)", ADVANCE = "NO") lengths, PRODUCT(lengths)
WRITE (UNIT = 201, FMT = "(I15)", ADVANCE = "NO") PRODUCT(lengths)
arcs_weights = arcs_cost_parameters1,

CALL ResetTimer(1)
CALL StartTimer(1)
CALL CreateSpanningTree ( tree_type = min_cost_tree, arcs_weights = arcs_weights,
weights_distribution = costs_distribution, tree_timer = 2, thread_timer = 3 )
CALL StopTimer(1)
WRITE(*, *) "Making the MST tree from scratch took:", ReadTimer(1)
WRITE(*, *) "_________________________ Tree building:", ReadTimer(2)
WRITE(*, *) "_________________________ Thread building:", ReadTimer(3)
WRITE (UNIT = 101, FMT = "(3E10.3)", ADVANCE = "NO") ReadTimer(2), ReadTimer(3),
ReadTimer(1)
// These do not depend on the disorder
WRITE (UNIT = 201, FMT = "(E10.3)", ADVANCE = "NO") ReadTimer(1)

WRITE(*, *) "Enter the disorder:
READ(*, *) disorder
WRITE (UNIT = 101, FMT = "(F6.2)", ADVANCE = "NO") disorder

CALL RandomUniform(weights_changes, range = (/ 1.0_wp - disorder, 1.0_wp + disorder /))
arcs_weights = weights_changes + arcs_weights  // Change the weights of the arcs
CALL ResetTimer(10)
CALL StartTimer(10)
CALL UpdateSpanningTree (tree_type = min_cost_tree, arcs_weights = arcs_weights,
    n_pivots = n_pivoted, tree_timer = 15, thread_timer = 16)
CALL StopTimer(10)

WRITE(*, *) "Rebuilding the MST tree took ":, ReadTimer(10)
WRITE(*, *) "___________________ Tree rebuilding:", ReadTimer(15)
WRITE(*, *) "___________________ Thread building:", ReadTimer(16)
WRITE (UNIT = 101, FMT = "(3E10.3, I10)", ADVANCE = "NO") ReadTimer(15), ReadTimer(16),
     ReadTimer(10), n_pivoted
WRITE (UNIT = 201, FMT = "(1E10.3)", ADVANCE = "NO") ReadTimer(10)

WRITE(101, *)  // A new line
CLOSE (UNIT = 101)
WRITE(201, *)  // A new line
CLOSE (UNIT = 201)

CALL DestroyLatticeNetworkProblem
CALL EndProgram

END PROGRAM Time_MST
2 Formatting rules for HPF/F90 files

These are just the auxiliary formatting rules and useful macros I use from time to time.

```plaintext
@m _SIZE(array, _kind, ...)  
  sifelse (#0, 0, int(_SIZE(array), KIND=_kind), int(_SIZE(array,#), KIND=_kind))
@m _MAXLOC(array, _kind, ...)  
  sifelse (#0, 0, int(_MAXLOC(array), KIND=_kind), int(_MAXLOC(array,#), KIND=_kind))
@m _MINLOC(array, _kind, ...)  
  sifelse (#0, 0, int(_MINLOC(array), KIND=_kind), int(_MINLOC(array,#), KIND=_kind))
@m _LBOUND(array, _kind, ...)  
  sifelse (#0, 0, int(_LBOUND(array, DIM=1), KIND=_kind), int(_LBOUND(array, #), KIND=_kind))
@m _UBOUND(array, _kind, ...)  
  sifelse (#0, 0, int(_UBOUND(array, DIM=1), KIND=_kind), int(_UBOUND(array, #), KIND=_kind))
@m _generic_name  
  interface generic_name  
  module procedure #.  
  end interface generic_name
@m _Declare_1_1(word(...)

  integer :: #.
@m _Declare_1_2(wp(...)

  integer (kind = i_wp) :: #.
@m _Declare_1_3(wp(...)

  real (kind = r_wp) :: #.
@m _Declare_1_4(sp(...)

  real (kind = r_sp) :: #.
@m _Declare_1_5(dp(...)

  real (kind = r_dp) :: #.
@m _fullExtent(_rank) :: DO (DIM, 2, _rank) { , : }
@m _VarSequence(variable, _start, _end)

  _variable##start_{DO (DIM, _SCALE (_start + 1), _end) { , _variable@DIM }
@m _NestedLoopStart(_variable, _array, _rank, _kind)

  _DO (_DIM, _rank, 1, -1) { _DO _variable@DIM = _LBOUND(_array, _kind, DIM = _DIM),  
  _UBOUND(_array, _kind, DIM = _DIM) }
@m _NestedLoopEnd(_rank) _DO (_DIM, 1, _rank) { end DO }
@m _Dummy(...)
@m _DisplayArray(message, array)

  if (_size(array) <= 20) then  
    write(message, print_unit, "(A)" message  
    write(message, print_unit, "(2005.2)") array  
  end if
```