// BUT = 1; DT = 2; SB = 3;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SETS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 int n = ...; // number of windfarm nodes

 {int} N = asSet(0..n); // all windfarm nodes incl. the depot

 int lastN = last(N);

 int firstN = first(N);

 {int} NW = asSet(1..n-1); // all windfarm nodes

 int m = ...; // number of maintenance task

 {int} M = asSet(0..m); // All maintenance tasks

 int lastM = last(M);

 int firstM = first(M);

 {int} Mloc[i in N] = ...; // maintenance tasks at location i in N

 int v = ...; // number of vessels

 {int} V = asSet(1..v); // All vessels

 int s = ...; // number of shifts

 {int} S = asSet(1..s); // all shifts in the planning period

 //range Wind = 0..30;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constants \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Time

 float Tday = ...;//number of Time units in a day

 float Tshift[S] = ...; // length of shift s in S

 float Tit[i in NW] = ...; //Average time to travel between turbines for vessel v

 float Speed[v in V] = ...;

// NEW Transport costs GEODESIC DISTANCES:

 float Cit[v in V] = ...; //Internal Transportation Costs for vessel v travelling inside windfarm to maintenance task

 float latitude[N] = ...;

 float longitude[N] = ...;

 float d[i in N][j in N];

 float PI;

execute

 {

 PI=Math.PI;

 }

 float pointsRadianlat[i in N] = latitude[i]\*PI/180;

 float pointsRadianlong[i in N] = longitude[i]\*PI/180;

 float earthRadiusKms = 6376.5;

execute compute\_distances

 {

 for(i in N) for(j in N)

 {

 d[i][j]=earthRadiusKms\*Math.acos(

 Math.sin(pointsRadianlat[i])\*Math.sin(pointsRadianlat[j])+

 Math.cos(pointsRadianlat[i])\*Math.cos(pointsRadianlat[j])\*

 Math.cos(pointsRadianlong[j]-pointsRadianlong[i]));

}

}

 float Cop[v in V] = ...; // operation costs in $$ time unit of transport of Vessel v

 float Tt[i in N][j in N][v in V]= d[i,j]/Speed[v]; //Transportation time between node i in N and Node j in N for vessel v in V

 float Ct[i in N][j in N][v in V] = Tt[i,j,v]\*Cop[v]; // transportation costs between node i in N and node j in N for vessel v in V

// Maintenance tasks costs - downtime costs according to current availability:

 float Tmt[m in M][s in S] = ...; //Duration of task m in M in hrs

 float Avail[i in NW] = ...; // Availability at Windfarm i in NW

// SWL Functions for Downtime costs regards to availability

 float WA[i in NW] = ...; // Warranted Availability

 float IT2[i in NW] = ...; // start of IT2

 float CostComp[i in NW] = ...; // Hourly rate in Comp

 float CostIT1[i in NW] = ...; // Hourly rate in IF1

 float CostIT2[i in NW] = ...; // Hourly rate in IF2

 stepFunction fCostAv[i in NW] = stepwise{CostComp[i]->WA[i]; CostIT1[i]->IT2[i];CostIT2[i]}; // Stepwise function of the DT costs per hour as function of current availability

 float CostAv[i in NW] = fCostAv[i](Avail[i]); // Downtime cost per hour, depending on the availability

 float Cdt[i in NW][m in M][s in S] = fCostAv[i](Avail[i])\*Tmt[m][s]; //Downtime cost of maintenance task m depending on the current availability

// NEW --- cost due to lost production

 float U[i in NW] = ...; // Wind speed at windpark i

 float elecpr = ...; // electricity price / production based rate of service provider

 pwlFunction fpow = piecewise{0->3.66; 459.32->12; 0->25;-3830.7288->26;0} (3.66,0); // pwl funct. of SWT-3.6 -->powercurve

 float produc[i in NW] = fpow(U[i]); // production at windpark i at the respective windspeed wndsd[i] in MWh

 float loPro[i in NW] = produc[i] \* elecpr; // lost production based rate (incentive) per hour downtime in wind park i

 float lostProMa[i in NW][m in M] = loPro[i] \* Tmt[m][s]; // total lost production based rate of maintenance task m in M

// Penalty for not completed or started tasks

 float CpenDT[i in NW][m in M][s in S] = (CostAv[i]) \* (Tmt[m][s]+Tshift[s]+(Tday-Tshift[s])); // Down time costs for duration of maintenance task plus at least 24 hours until beginning of next shift

 float CpenLP[i in NW][m in M][s in S] = loPro[i] \* (Tmt[m][s]+Tshift[s]+(Tday-Tshift[s])); // Lost Production for the duration of not started maintenance tasks

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Decision Variables \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 dvar boolean x[m in M, v in V, s in S] in 0..1; // 1 if maintenance task m in M is performed by vessel v in Vduring shift s in S, 0 otherwise

 dvar boolean y[i in N, j in N, v in V, s in S] in 0..1; // 1 if vessel travels directly between node i in N and j in N, i != j, during shift s in S

 //dvar float+ l[m in M, s in S]; // time counter for how long the turbine where maintenance tassk m in M is located is shut down during shift s in S.

 //dvar boolean z[m in M, n in M, v in V, s in S]; // if vessel v performs maintenance task n in M directly after maintenance task m in M during shift s in S

 //dvar float+ t[m in M, v in V, s in S]; // time vessel v in V[m] starts maintenance task m in M during shift s in S

 //dvar boolean f[m in M, s in S]; // 1 if task m in M is completed before the end of shift s in S, 0 otherwise

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Objective Function \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

minimize sum(i in N, j in N, v in V, s in S:i!=j) Ct[i,j,v]\*y[i,j,v,s] // Tansportation cost between Nodes

 + sum(m in M, v in V, s in S: m!=firstM && m!=lastM) Cit[v]\*x[m,v,s] // Internal transportation costs to maintenance task

 + sum(i in NW, m in Mloc[i], v in V,s in S) Cdt[i,m,s]\*x[m,v,s] //Downtime costs according to current availability level

 + sum(i in NW, m in Mloc[i], v in V, s in S: i== 1) lostProMa[i,m]\*x[m,v,s] //Costs due to lost production / production based rate during maintenance task - only applicable for BUT -> : i==1

 + sum(i in NW, m in Mloc[i], v in V, s in S) CpenDT[i,m,s]\*(1-x[m,v,s]) //Penalty & costs for Maintenance tasks, that are not done - incentive to do as many tasks as possible

 + sum(i in NW, m in Mloc[i], v in V, s in S: i == 1) CpenLP[i,m,s]\*(1-x[m,v,s]); // Penalty costs due to lost production as a result of maintenance tasks that are not done

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constraints \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

subject to {

//Vessel (ctv) constraints:

forall (v in V, s in S)

 sum(j in N) y[firstN,j,v,s] == 1; // (2) all vessels must leave the start depot node

forall (v in V, s in S)

 sum(i in N) y[i,lastN,v,s] == 1; // (3) all vessels must arrive at the end depot node at the end of the shift

forall (j in NW, v in V, s in S)

 sum(i in N) y[i,j,v,s] == sum(i in N) y[j,i,v,s]; // (5) conserves node flow for CTVs by ensuring that when a CTV visits a wind farm node during a shift, it also leaves the node during the same shift

forall (v in V, s in S)

 sum(i in N, j in N) y[i,j,v,s] <= 2; // (6) CTVs cannot travel betwen windfarms during shifts, they can travel at maximum twice during each shift

// Constraints for the execution of tasks:

forall (m in M diff {firstM,lastM}, s in S)

 sum(v in V) x[m,v,s] <= 1; // (13) Tasks can be performed by maximum one vessel each shift

forall (v in V, s in S, m in M)

 x[lastM,v,s] == 1; // (14) Forces the CTVs to perform the depot task in each shift

forall (v in V, s in S, m in M)

 x[firstM,v,s] == 1; // (eigene) Forces CTVs to perform the start depot task first in each shift

forall (i in NW, m in Mloc[i], v in V, s in S)

 x[m,v,s] <= sum(j in N) y[j,i,v,s]; // (17) Maintenance task at wind farm i can only be performed by a vessel that is located at this wind farm

//forall (m in M, v in V, s in S) // (eigene) Evtl. überflüssig jetzt?! All maintenance tasks have to be performed ??!!

 // sum(m in M) x[m,v,s] >= 4;

// Own Constraints

// Time Management:

//forall (i in NW, j in NW, m in M, v in V, s in S) // der ist scheiße weil er simultane maintenance task bearbeitung verhindert /bzw. nicht nutzt

 // sum(m in M) x[m,v,s]\*(Tmt[m,s]+Tit[i])+ sum(i in N, j in N) y[i,j,v,s]\*Tt[i,j] <= Tshift[s]; // (eigene) total duration of maintenance tasks performed can not be larger than 12 hours (incl. transfer time in park)

 forall (i in NW, j in NW, m in Mloc[i], v in V, s in S) // die summe der transferzeiten im park zwischen den einzelnen Tasks in dem park + die Transferzeit zwischen Depot und Park darf nicht größer sein als die Shift length

 sum(m in Mloc[i]) x[m,v,s]\*(Tit[i]) + 2\* sum(i in N, j in N) (y[i,j,v,s]\*Tt[i,j,v])<= Tshift[s];

// forall (i in NW, j in NW, m in M, v in V, s in S)

 // sum(m in M) x[m,v,s]\*tmax[m];

// Subtours and Depot to Depot touren

forall (v in V, s in S)

 sum(i in N, j in N) y[firstN,lastN,v,s] == 0; // Vessels cant go from start depot to finish depot directly without visiting a windfarm

forall (v in V, s in S)

 sum(i in N, j in N:j==i) y[i,j,v,s] == 0; // Eliminate circle tours at one node: ÜBERFLÜSSIG!

//forall (m in M, s in s)

 // Tmt[m] - sum(v in V)sum(h==1..s) t{m,v,h]+(Tshift[s]\*f[m,s]);

// Constraints for the time Management:

/\*

forall (m in M, v in V, s in S)

 t[m,v,s] <= Tshift\*x[m,v,s]; // (27) forces the start time of tasks that are not performed to 0

 \*/

// Constraints for the precedence of tasks:

/\*

forall (m in M, v in V, s in S)

 x[m,v,s] == sum(n in M) z[m,n,v,s]; // (36) if task m is performed, exactly one task is performed directly after m if m is not performed, zero tasks are performed directly after

forall (m in M, v in V, s in S)

 x[m,v,s] == sum(n in M: n != m) z[n,m,v,s]; // (37) same as 36 just for tasks performed directly before

// Constraints for the Down Time:

forall (m in M, s in S)

 l[m,s]>= Tday\*(1-f[m,s]);

\*/

};

// assert forall(i in N, j in N, v in V, s in S) ((x[m,v,s]!=0)=>(y[i,j,v,s]==0));