Capital Budgeting Problem

"The purpose of this OPL model is to help a firm invest on a set of proposed projects over time. Each of the projects brings a reward in each period once chosen. The objective is to maximize the total net present value (NPV) of the initial investment and final revenue. The model enables restrictions on the selection of projects, such as: one project must be chosen among a set of mutually exclusive projects. The solution to the model indicates the projects that will be selected and the timing of selection. A minimum balance constraint in each period can be ensured."

To load this project, search for capital budgeting in project loader.

Now we explain the model in detail.

Define number of periods for investment and number of project sets in which one project must be chosen.

```
int T=...;

int NbMustTakeOne=...;
```

Define the range of periods

```
range Periods = 1..T;
```

Load project sets.

```
{string} AllProjects = ...;
```

Load must-choose-one project subsets.

```
// Subsets of projects; must take one

{string} MustTakeOne[1..NbMustTakeOne] = ...;
```

Define discount rate for calculating present value

```
//Discount rate (interest rate)

float Rate = …;
```

Load setup cost for each project at each period

```
// Setup cost of projects

float SetupCost[AllProjects][Periods] = …;
```

Load reward for each project at each period

```
// Rewards (= Revenue - Cost) of projects in each period

float Reward[AllProjects][Periods] = …;
```

Load minimum balance required for each period

```
// Minimum balance in each period

float MinBal[Periods]=…;
```

Load initial balance

```
float InitBal = …;
```

Define decision variable for balance (initial balance is used for period zero)

```
// Account balance at the end of each period

dvar float Bal[0..T];
```

Define Boolean variables for when starting each project.

```
// Selection activities

dvar boolean doProj[AllProjects][Periods]; // 1 if project is selected to start at time t; 0
otherwise
```

Define selection variable for each project at each period; a project is selected if its started before this period.

```
// Indicate if a project has been selected up to period t
dvar boolean SelectedProj[AllProjects][Periods];
```

Define objective to be the net present value of the final balance minus initial balance

```
// Maximize the total NPV of selected projects
dexpr float Objective =
  Bal[T] /pow(1+Rate,T) - Bal[0];
maximize
   Objective;
```

Define the constraint for the initial balance

```
  // Initial Balanced
  Bal[0] == InitBal;
```

Define the balance flow constraints: balance at the end of each period is equal to balance from last period with interest minus setup cost at this period plus total rewards from selected projects in this period

```
  // Cash flows balance constraints
  //  no reward for starting doing project at last period
  // Current money = Money left from previous period - Cost of selected projects
  //in current period  +  Rewards of the ongoing projects
  forall(t in Periods)
     ctBal: Bal[t] == (1+Rate)*(Bal[t-1] - sum(i in AllProjects) SetupCost[i][t]*doProj[i][t]
     + sum(i in AllProjects) Reward[i][t]*SelectedProj[i][t]);
```

Define constraints that balance at each period must meet minimum values

```
// Minimum Balance constraint in each period
forall(i in Periods)
  ctMinBal: Bal[i] >= MinBal[i];
```

Define constraints that a project is selected if it has started in any previous period

```
// Selected Projects
forall(i in AllProjects, t in Periods)
  SelectedProj[i][t] == sum(s in 1..t-1) doProj[i][s];
```

Define constraints that each project can only start at most once through the whole periods

```
// All project are selected at most once
forall(i in AllProjects)
  ctMost: SelectedProj[i][T] <= 1;
  // sum(s in 1..T-1) doProj[i][s] <=1
```

Define constraints that one project must be chosen for each project set.

```
// Must-take-one group -- select one project from the MustTakeOne set
forall(i in 1..NbMustTakeOne)
  ctMust: sum(p in MustTakeOne[i]) SelectedProj[p][T] == 1;
  // sum(p in MustTakeOne[i], t in Periods ) doProj[p][t] == 1;
```