# Milk problem (spilling + not spilling) and n jars - a graphs solution

GONZALO GARCÍA ALARCÓN ESTRADA, **********
Nov / 16 / 2017
University of Toronto, Shameless Mathematica

**Milk problem:**

A milkman has n jars, each with integer capacity. The jar with the maximum capacity is full of milk. Can the milkman measure out L liters? He has no other way to measure besides using these jars. (He may or may not spill milk in the process.)

**Challenge**:

Draw the state graph of this problem (spilling OR no spilling allowed)

****** USER INPUT ******

```
full = {8, 5, 3, 2}; (* give a list with (at least 2) integers,
these are the jar capacities *)
spilling = False;  (* set to True or False *)
target = 4; (* what amount of milk is desired to be measured *)
```

** END user input **

```
n = Length[full];
var = Map[x#&, Range[n]];
If[spilling === True, eq = (Plus @@ var ≤ Max[full]), eq = (Plus @@ var == Max[full])]
```

$x_1 + x_2 + x_3 + x_4 == 8$

```
States =
 var /. Solve[(eq) && (And @@ Map[(full[[#]] ≥ var[[#]] ≥ 0) &, Range[n]]), var, Integers]
```

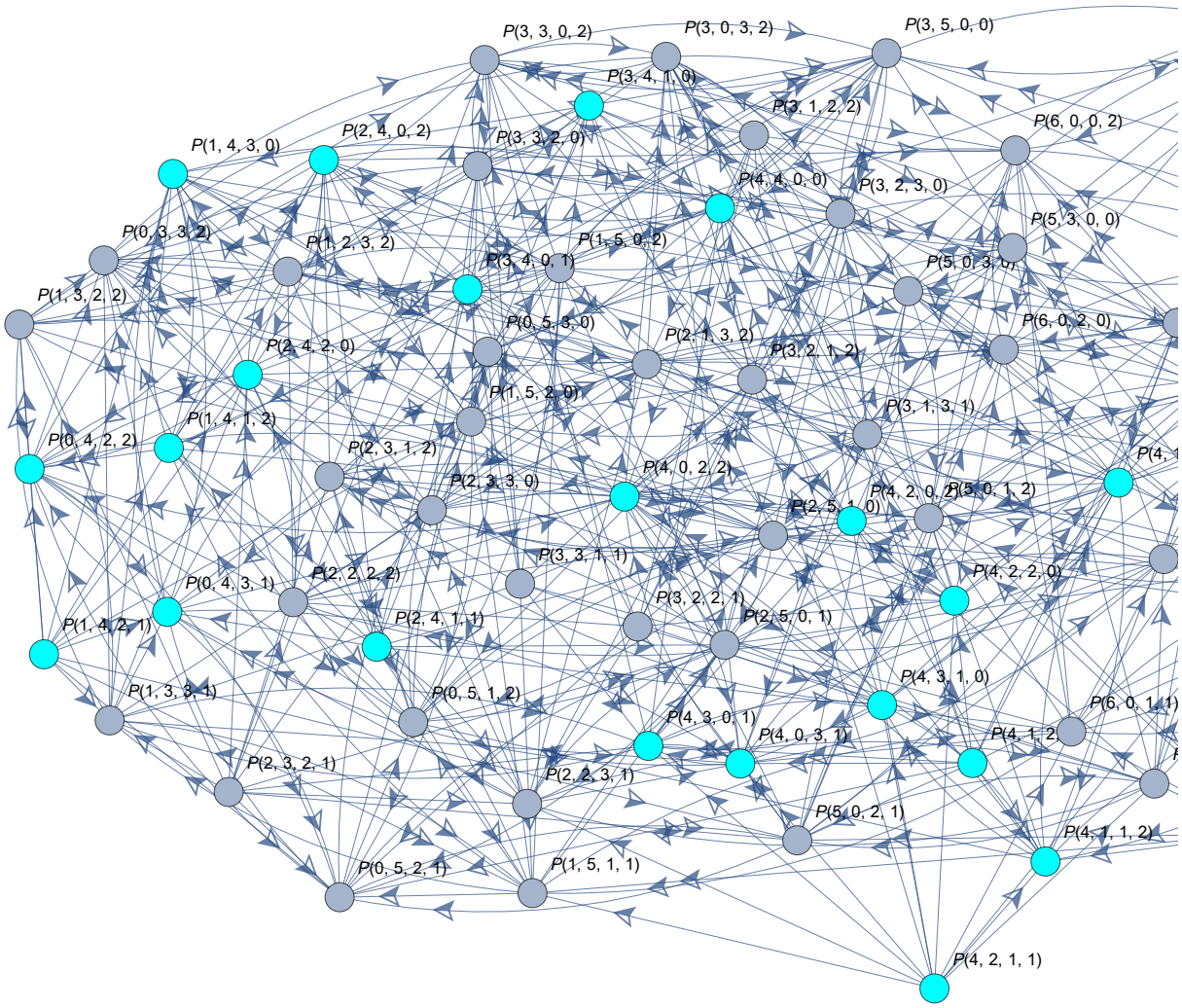
```
{{0, 3, 3, 2}, {0, 4, 2, 2}, {0, 4, 3, 1}, {0, 5, 1, 2}, {0, 5, 2, 1}, {0, 5, 3, 0}, {1, 2, 3, 2},
 {1, 3, 2, 2}, {1, 3, 3, 1}, {1, 4, 1, 2}, {1, 4, 2, 1}, {1, 4, 3, 0}, {1, 5, 0, 2}, {1, 5, 1, 1},
 {1, 5, 2, 0}, {2, 1, 3, 2}, {2, 2, 2, 2}, {2, 2, 3, 1}, {2, 3, 1, 2}, {2, 3, 2, 1},
 {2, 3, 3, 0}, {2, 4, 0, 2}, {2, 4, 1, 1}, {2, 4, 2, 0}, {2, 5, 0, 1}, {2, 5, 1, 0},
 {3, 0, 3, 2}, {3, 1, 2, 2}, {3, 1, 3, 1}, {3, 2, 1, 2}, {3, 2, 2, 1}, {3, 2, 3, 0},
 {3, 3, 0, 2}, {3, 3, 1, 1}, {3, 3, 2, 0}, {3, 4, 0, 1}, {3, 4, 1, 0}, {3, 5, 0, 0},
 {4, 0, 2, 2}, {4, 0, 3, 1}, {4, 1, 1, 2}, {4, 1, 2, 1}, {4, 1, 3, 0}, {4, 2, 0, 2},
 {4, 2, 1, 1}, {4, 2, 2, 0}, {4, 3, 0, 1}, {4, 3, 1, 0}, {4, 4, 0, 0}, {5, 0, 1, 2},
 {5, 0, 2, 1}, {5, 0, 3, 0}, {5, 1, 0, 2}, {5, 1, 1, 1}, {5, 1, 2, 0}, {5, 2, 0, 1},
 {5, 2, 1, 0}, {5, 3, 0, 0}, {6, 0, 0, 2}, {6, 0, 1, 1}, {6, 0, 2, 0}, {6, 1, 0, 1},
 {6, 1, 1, 0}, {6, 2, 0, 0}, {7, 0, 0, 1}, {7, 0, 1, 0}, {7, 1, 0, 0}, {8, 0, 0, 0}}
```

```
(*compare .... -1 out, 0 equal, +1 in*)
lzg[x_, y_] := If[x > y, -1,
  If[x == y, 0,
   If[x < y, +1]]]

(* now for sets.... actually lists *)
lzgSet[A_, B_] := Map[(lzg[A[[#]], B[[#]]]) &, Range[Min[Length[A], Length[B]]]]

(* RELATION A→B i.e. B is pourable from A *)
Pourable[A_, B_] := Module[{change},
  change = lzgSet[A, B];
  If[Sort[change] == Sort[Join[ConstantArray[0, n - 2], {-1, 1}]],
   out = Position[change, -1][[1]][[1]]; in = Position[change, +1][[1]][[1]];
   If[B[[out]] == 0 ∨ B[[in]] == full[[in]], True, False], False]]


Spillable[A_, B_] := Module[{change},
  change = lzgSet[A, B];
  If[Sort[change] == Sort[Join[ConstantArray[0, n - 1], {-1}]],
   out = Position[change, -1][[1]][[1]];
   If[B[[out]] == 0, True, False], False]]

(* apply relation to states *)
SxS = Pair @@@ Tuples[States, 2];
Pour = Select[SxS /. Pair[i_, j_] :> i ↦ j /; Pourable[i, j], Head[#] == DirectedEdge &];
Spill = Select[SxS /. Pair[i_, j_] :> i ↦ j /; Spillable[i, j], Head[#] == DirectedEdge &];

(* ----- PREPARATION FOR GRAPH ------ *)

ver = P @@@ States;

edgP = Pour /. a_ ↦ b_ :> P @@ a ↦ P @@ b;
edgS = Spill /. a_ ↦ b_ :> P @@ a ↦ P @@ b;
edg = edgP ⋃ edgS;


Start = {P @@ full /. x_ :> 0 /; x ≠ Max[full]};
```

```
Obj = Select[ver, MemberQ[List @@ #, target] &]
```

```
{P[0, 4, 2, 2], P[0, 4, 3, 1], P[1, 4, 1, 2], P[1, 4, 2, 1], P[1, 4, 3, 0],
 P[2, 4, 0, 2], P[2, 4, 1, 1], P[2, 4, 2, 0], P[3, 4, 0, 1], P[3, 4, 1, 0],
 P[4, 0, 2, 2], P[4, 0, 3, 1], P[4, 1, 1, 2], P[4, 1, 2, 1], P[4, 1, 3, 0], P[4, 2, 0, 2],
 P[4, 2, 1, 1], P[4, 2, 2, 0], P[4, 3, 0, 1], P[4, 3, 1, 0], P[4, 4, 0, 0]}
```

```
G = Graph[ver, edg, VertexLabels → "Name", ImageSize → 900,
    EdgeShapeFunction → GraphElementData["ShortCarvedArrow", "ArrowSize" → 0.02]];

HighlightGraph[G, {Style[Start, Blue], Style[Obj, Cyan], Style[edgS, LightGray]}]
```




```
(* find reachable objectives *)
reachPaths = Map[FindShortestPath[G, Start[[1]], #] &, Obj];
```

```
NotObj = Select[Obj, ! MemberQ[Flatten[reachPaths], #] &]
YesObj = Complement[Obj, NotObj]
```

```
{P[1, 4, 2, 1], P[2, 4, 1, 1], P[4, 1, 2, 1], P[4, 2, 1, 1]}
```

```
{P[0, 4, 2, 2], P[0, 4, 3, 1], P[1, 4, 1, 2], P[1, 4, 3, 0], P[2, 4, 0, 2],
 P[2, 4, 2, 0], P[3, 4, 0, 1], P[3, 4, 1, 0], P[4, 0, 2, 2], P[4, 0, 3, 1], P[4, 1, 1, 2],
 P[4, 1, 3, 0], P[4, 2, 0, 2], P[4, 2, 2, 0], P[4, 3, 0, 1], P[4, 3, 1, 0], P[4, 4, 0, 0]}
```

```
HighlightGraph[G, {Style[Start, Blue], Style[YesObj, Cyan], Style[NotObj, LightCyan]} ⋃
  Map[PathGraph[#, DirectedEdges → True] &, reachPaths]]
```