NOPT042 Constraint programming: Tutorial 4 – Search strategies

From last week:

- Solution to the Coin grid problem.
- Best model and solver for the problem? MIP, naturally expressed as an integer program
- Unsatisfiable instances - LP works well.
- For sparse solution sets heuristic approaches may be slow.

Example: N-queens

Place n queens on an $n \times n$ board so that none attack another. How to choose the decision variables?

- How large is the search space?
- Can we use symmetry breaking?
- Consider the *dual* model.

In [1]:
```
!time picat queens/queens-primal.pi 8
```

```
.....Q..
...Q....
.Q......
.......Q
....Q...
......Q.
Q.......
..Q.....

real	0m0.020s
user	0m0.015s
sys	0m0.005s
```

In [2]:
```
!cat queens/queens-primal.pi
```

```
% n-queens, primal model
import sat.

main([N]) =>
    N := to_int(N),
    queens(N, Q),
    solve(Q),
    if N <= 32 then
        output(Q)
    end.


queens(N, Q) =>
    Q = new_array(N),
    Q :: 1..N,
    all_different(Q),
    all_different([$Q[I] - I : I in 1..N]),
    all_different([$Q[I] + I : I in 1..N]).


output(Q) =>
    N = Q.length,
    foreach(I in 1..N)
        foreach (J in 1..N)
            if Q[I] = J then
                print("Q")
            else
                print(".")
            end
        end,
        print("\n")
    end.
```

In [3]:

```
!time picat queens/queens-dual.pi 64
```

```
Welcome to the CBC MILP Solver
Version: 2.10.7
Build Date: Feb 14 2022

command line - cbc __tmp.lp solve solu __tmp.sol (default strategy 1)
Continuous objective value is 0 - 0.07 seconds
Cgl0003I 0 fixed, 0 tightened bounds, 2 strengthened rows, 0 substitutions
Cgl0004I processed model has 379 rows, 4096 columns (4096 integer (4096 of which bin
ary)) and 20478 elements
Cbc0045I No integer variables out of 4096 objects (4096 integer) have costs
Cbc0045I branch on satisfied N create fake objective Y random cost Y
Cbc0038I Initial state - 125 integers unsatisfied sum - 26.3403
Cbc0038I Pass   1: suminf.   10.43351 (69) obj. 0 iterations 767
Cbc0038I Pass   2: suminf.    6.76111 (160) obj. 0 iterations 471
Cbc0038I Pass   3: suminf.    5.28571 (23) obj. 0 iterations 743
Cbc0038I Pass   4: suminf.    5.24859 (154) obj. 0 iterations 160
Cbc0038I Pass   5: suminf.    2.00000 (4) obj. 0 iterations 379
Cbc0038I Pass   6: suminf.    2.00000 (4) obj. 0 iterations 139
Cbc0038I Pass   7: suminf.    2.00000 (4) obj. 0 iterations 132
Cbc0038I Pass   8: suminf.    2.00000 (4) obj. 0 iterations 73
Cbc0038I Pass   9: suminf.   47.29014 (196) obj. 0 iterations 938
Cbc0038I Pass  10: suminf.   15.00000 (37) obj. 0 iterations 1146
Cbc0038I Pass  11: suminf.   13.00000 (26) obj. 0 iterations 78
Cbc0038I Pass  12: suminf.   13.00000 (26) obj. 0 iterations 270
Cbc0038I Pass  13: suminf.   52.01151 (186) obj. 0 iterations 982
Cbc0038I Pass  14: suminf.    9.72000 (51) obj. 0 iterations 1494
Cbc0038I Pass  15: suminf.    0.00000 (0) obj. 0 iterations 1390
Cbc0038I Solution found of 0
Cbc0038I Before mini branch and bound, 3418 integers at bound fixed and 0 continuous
Cbc0038I Mini branch and bound did not improve solution (1.15 seconds)
Cbc0038I After 1.15 seconds - Feasibility pump exiting with objective of 0 - took 0.
89 seconds
Cbc0012I Integer solution of 0 found by feasibility pump after 0 iterations and 0 no
des (1.15 seconds)
Cbc0001I Search completed - best objective 0, took 0 iterations and 0 nodes (1.15 se
conds)
Cbc0035I Maximum depth 0, 0 variables fixed on reduced cost
Cuts at root node changed objective from 0 to 0
Probing was tried 0 times and created 0 cuts of which 0 were active after adding rou
nds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding roun
ds of cuts (0.000 seconds)
Knapsack was tried 0 times and created 0 cuts of which 0 were active after adding ro
unds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding roun
ds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were active af
ter adding rounds of cuts (0.000 seconds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after adding r
ounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after adding ro
unds of cuts (0.000 seconds)

Result - Optimal solution found
```

```
Objective value:                0.00000000
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             1.17
Time (Wallclock seconds):       1.18

Total time (CPU seconds):       1.25   (Wallclock seconds):       1.26


real	0m1.707s
user	0m1.484s
sys	0m0.286s
```

In [4]:

```
!cat queens/queens-dual.pi
```

```
% n-queens, dual model
import mip.

main([N]) =>
    N := to_int(N),
    queens(N, Board),
    solve(Board),
    if N <= 32 then
        output(Q)
    end.


queens(N, Board) =>
    Board = new_array(N, N),
    Board :: 0..1,

    sum([Board[I, J] : I in 1..N, J in 1..N]) #= N,

    % rows
    foreach(I in 1..N)
        sum([Board[I, J] : J in 1..N]) #<= 1
    end,

    % cols
    foreach(J in 1..N)
        sum([Board[I, J] : I in 1..N]) #<= 1
    end,

    % diags
    foreach(K in 1-N..N-1)
        sum([Board[I,J] : I in 1..N, J in 1..N, I-J = K ]) #<= 1
    end,
    foreach(K in 2..2*N)
        sum([Board[I,J] : I in 1..N, J in 1..N, I+J = K ]) #<= 1
    end.


output(Board) =>
    N = Board.length,
    foreach(I in 1..N)
        foreach (J in 1..N)
            if Board[I, J] = 1 then
                print("Q")
            else
                print(".")
            end
        end,
        print("\n")
    end.
```

Sometimes it is best to model the problem in both ways and add *channelling constraints*. (Here it does not help.)

In [5]:
```
!time picat queens/queens-channeling.pi 128
```

```
real	0m3.207s
user	0m3.086s
sys	0m0.121s
```

In [6]:
```
!cat queens/queens-channeling.pi
```

```
% n-queens, primal model
import sat.

main([N]) =>
    N := to_int(N),
    queens(N, Q, Board),
    solve(Q ++ Board),
    if N <= 32 then
        output(Q)
    end.


queens(N, Q, Board) =>
    % primal
    Q = new_array(N),
    Q :: 1..N,
    all_different(Q),
    all_different([$Q[I] - I : I in 1..N]),
    all_different([$Q[I] + I : I in 1..N]),

    % dual
    Board = new_array(N, N),
    Board :: 0..1,
    sum([Board[I, J] : I in 1..N, J in 1..N]) #= N,
    foreach(I in 1..N)
        sum([Board[I, J] : J in 1..N]) #<= 1
    end,
    foreach(J in 1..N)
        sum([Board[I, J] : I in 1..N]) #<= 1
    end,
    foreach(K in 1-N..N-1)
        sum([Board[I, J] : I in 1..N, J in 1..N, I - J = K ]) #<= 1
    end,
    foreach(K in 2..2*N)
        sum([Board[I, J] : I in 1..N, J in 1..N, I + J = K ]) #<= 1
    end,

    % channeling
    foreach(I in 1..N, J in 1..N)
        (Board[I,J] #= 1) #<=> (Q[I] #= J)
    end.


output(Q) =>
    N = Q.length,
    foreach(I in 1..N)
        foreach (J in 1..N)
            if Q[I] = J then
                print("Q")
            else
                print(".")
            end
        end,
        print("\n")
    end.
```

Can the models be improved using symmetry breaking?

Search strategies

And other solver options: see Picat guide (Section 12.6) and the book (Section 3.5)

In [7]:
```
%load_ext ipicat
```

```
Picat version 3.5#5
```

In [8]:
```
%%picat -n queens
import cp. %try sat, try also mip with the other model

queens(N, Q) =>
    Q = new_array(N),
    Q :: 1..N,
    all_different(Q),
    all_different([$Q[I] - I : I in 1..N]),
    all_different([$Q[I] + I : I in 1..N]).
```

The predicate `time2` also outputs the number of backtracks during the search - a good measure of complexity.

In [9]:
```
%%picat
main =>
    N = 32,
    queens(N, Q),
    time2(solve(Q)).
```

```
CPU time 28.775 seconds. Backtracks: 11461548
```

Which search strategy could work well for our model?

Here's how we can test multiple search strategies (code adapted from the book):

In [10]:
```
%%picat

% Variable selection
selection(VarSels) =>
    VarSels = [backward,constr,degree,ff,ffc,ffd,forward,inout,leftmost,max,min,up]

% Value selection
choice(ValSels) =>
    ValSels = [down,reverse_split,split,up,updown].

main =>
    selection(VarSels),
    choice(ValSels),
    Timeout = 1000, % Timeout in milliseconds
```

```
%Timeout = 10000, % Timeout in milliseconds
Ns = [64, 128, 256],

foreach (N in Ns, VarSel in VarSels, ValSel in ValSels)
    queens(N,Q),
    time2(time_out(solve([VarSel,ValSel], Q),Timeout,Status)),
    println([N,VarSel,ValSel,Status])
end.
```

```
CPU time 0.993 seconds. Backtracks: 349094

[64,backward,down,time_out]

CPU time 0.99 seconds. Backtracks: 0

[64,backward,reverse_split,time_out]

CPU time 1.976 seconds. Backtracks: 0

[64,backward,split,time_out]

CPU time 0.996 seconds. Backtracks: 226141

[64,backward,up,time_out]

CPU time 0.995 seconds. Backtracks: 524393

[64,backward,updown,time_out]

CPU time 0.995 seconds. Backtracks: 345490

[64,constr,down,time_out]

CPU time 0.993 seconds. Backtracks: 0

[64,constr,reverse_split,time_out]

CPU time 0.987 seconds. Backtracks: 0

[64,constr,split,time_out]

CPU time 0.992 seconds. Backtracks: 186115

[64,constr,up,time_out]

CPU time 0.989 seconds. Backtracks: 471112

[64,constr,updown,time_out]

CPU time 0.996 seconds. Backtracks: 333211

[64,degree,down,time_out]

CPU time 2.001 seconds. Backtracks: 0

[64,degree,reverse_split,time_out]

CPU time 1.0 seconds. Backtracks: 0

[64,degree,split,time_out]

CPU time 1.0 seconds. Backtracks: 197097

[64,degree,up,time_out]
```

```
CPU time 0.996 seconds. Backtracks: 471578

[64,degree,updown,time_out]

CPU time 0.002 seconds. Backtracks: 695

[64,ff,down,success]

CPU time 0.003 seconds. Backtracks: 0

[64,ff,reverse_split,success]

CPU time 0.003 seconds. Backtracks: 0

[64,ff,split,success]

CPU time 0.002 seconds. Backtracks: 382

[64,ff,up,success]

CPU time 0.0 seconds. Backtracks: 115

[64,ff,updown,success]

CPU time 0.002 seconds. Backtracks: 695

[64,ffc,down,success]

CPU time 0.002 seconds. Backtracks: 0

[64,ffc,reverse_split,success]

CPU time 0.003 seconds. Backtracks: 0

[64,ffc,split,success]

CPU time 0.003 seconds. Backtracks: 382

[64,ffc,up,success]

CPU time 0.0 seconds. Backtracks: 115

[64,ffc,updown,success]

CPU time 0.001 seconds. Backtracks: 136

[64,ffd,down,success]

CPU time 0.002 seconds. Backtracks: 0

[64,ffd,reverse_split,success]

CPU time 0.0 seconds. Backtracks: 0

[64,ffd,split,success]
```

```
CPU time 0.001 seconds. Backtracks: 75

[64,ffd,up,success]

CPU time 0.01 seconds. Backtracks: 9120

[64,ffd,updown,success]

CPU time 1.001 seconds. Backtracks: 353758

[64,forward,down,time_out]

CPU time 1.0 seconds. Backtracks: 0

[64,forward,reverse_split,time_out]

CPU time 1.0 seconds. Backtracks: 0

[64,forward,split,time_out]

CPU time 1.001 seconds. Backtracks: 209680

[64,forward,up,time_out]

CPU time 1.0 seconds. Backtracks: 468253

[64,forward,updown,time_out]

CPU time 1.0 seconds. Backtracks: 528571

[64,inout,down,time_out]

CPU time 1.0 seconds. Backtracks: 0

[64,inout,reverse_split,time_out]

CPU time 1.0 seconds. Backtracks: 0

[64,inout,split,time_out]

CPU time 1.0 seconds. Backtracks: 361647

[64,inout,up,time_out]

CPU time 1.001 seconds. Backtracks: 618063

[64,inout,updown,time_out]

CPU time 1.001 seconds. Backtracks: 365186

[64,leftmost,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[64,leftmost,reverse_split,time_out]
```

```
CPU time 1.001 seconds. Backtracks: 0

[64,leftmost,split,time_out]

CPU time 1.001 seconds. Backtracks: 220229

[64,leftmost,up,time_out]

CPU time 1.0 seconds. Backtracks: 520309

[64,leftmost,updown,time_out]

CPU time 1.001 seconds. Backtracks: 499376

[64,max,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[64,max,reverse_split,time_out]

CPU time 0.0 seconds. Backtracks: 0

[64,max,split,success]

CPU time 0.0 seconds. Backtracks: 53

[64,max,up,success]

CPU time 0.0 seconds. Backtracks: 363

[64,max,updown,success]

CPU time 0.001 seconds. Backtracks: 82

[64,min,down,success]

CPU time 0.001 seconds. Backtracks: 0

[64,min,reverse_split,success]

CPU time 1.001 seconds. Backtracks: 0

[64,min,split,time_out]

CPU time 1.001 seconds. Backtracks: 349667

[64,min,up,time_out]

CPU time 0.021 seconds. Backtracks: 10314

[64,min,updown,success]

CPU time 1.001 seconds. Backtracks: 369763

[64,up,down,time_out]
```

```
CPU time 1.0 seconds. Backtracks: 0

[64,up,reverse_split,time_out]

CPU time 1.0 seconds. Backtracks: 0

[64,up,split,time_out]

CPU time 1.001 seconds. Backtracks: 223145

[64,up,up,time_out]

CPU time 1.001 seconds. Backtracks: 518559

[64,up,updown,time_out]

CPU time 1.001 seconds. Backtracks: 185339

[128,backward,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,backward,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,backward,split,time_out]

CPU time 1.0 seconds. Backtracks: 118316

[128,backward,up,time_out]

CPU time 1.0 seconds. Backtracks: 244467

[128,backward,updown,time_out]

CPU time 1.001 seconds. Backtracks: 184359

[128,constr,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,constr,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,constr,split,time_out]

CPU time 1.001 seconds. Backtracks: 110319

[128,constr,up,time_out]

CPU time 1.001 seconds. Backtracks: 230922

[128,constr,updown,time_out]
```

```
CPU time 1.001 seconds. Backtracks: 183134

[128,degree,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,degree,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,degree,split,time_out]

CPU time 1.001 seconds. Backtracks: 110355

[128,degree,up,time_out]

CPU time 1.002 seconds. Backtracks: 229615

[128,degree,updown,time_out]

CPU time 1.001 seconds. Backtracks: 158976

[128,ff,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,ff,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,ff,split,time_out]

CPU time 1.001 seconds. Backtracks: 74788

[128,ff,up,time_out]

CPU time 1.001 seconds. Backtracks: 412546

[128,ff,updown,time_out]

CPU time 1.001 seconds. Backtracks: 152097

[128,ffc,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,ffc,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[128,ffc,split,time_out]

CPU time 1.001 seconds. Backtracks: 74570

[128,ffc,up,time_out]
```

```
CPU time 1.001 seconds. Backtracks: 411677

[128,ffc,updown,time_out]

CPU time 0.002 seconds. Backtracks: 4

[128,ffd,down,success]

CPU time 0.002 seconds. Backtracks: 0

[128,ffd,reverse_split,success]

CPU time 0.002 seconds. Backtracks: 0

[128,ffd,split,success]

CPU time 0.002 seconds. Backtracks: 3

[128,ffd,up,success]

CPU time 1.002 seconds. Backtracks: 345023

[128,ffd,updown,time_out]

CPU time 1.001 seconds. Backtracks: 179279

[128,forward,down,time_out]

CPU time 0.997 seconds. Backtracks: 0

[128,forward,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,forward,split,time_out]

CPU time 1.002 seconds. Backtracks: 107898

[128,forward,up,time_out]

CPU time 1.001 seconds. Backtracks: 226512

[128,forward,updown,time_out]

CPU time 1.001 seconds. Backtracks: 455866

[128,inout,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,inout,reverse_split,time_out]

CPU time 1.998 seconds. Backtracks: 0

[128,inout,split,time_out]
```

```
CPU time 1.001 seconds. Backtracks: 340319

[128,inout,up,time_out]

CPU time 1.001 seconds. Backtracks: 265005

[128,inout,updown,time_out]

CPU time 1.001 seconds. Backtracks: 180417

[128,leftmost,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[128,leftmost,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[128,leftmost,split,time_out]

CPU time 0.993 seconds. Backtracks: 108888

[128,leftmost,up,time_out]

CPU time 1.002 seconds. Backtracks: 220702

[128,leftmost,updown,time_out]

CPU time 1.002 seconds. Backtracks: 190178

[128,max,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[128,max,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[128,max,split,time_out]

CPU time 1.002 seconds. Backtracks: 153272

[128,max,up,time_out]

CPU time 1.002 seconds. Backtracks: 232148

[128,max,updown,time_out]

CPU time 1.002 seconds. Backtracks: 215633

[128,min,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[128,min,reverse_split,time_out]
```

```
CPU time 1.002 seconds. Backtracks: 0

[128,min,split,time_out]

CPU time 1.002 seconds. Backtracks: 133941

[128,min,up,time_out]

CPU time 1.001 seconds. Backtracks: 199093

[128,min,updown,time_out]

CPU time 1.002 seconds. Backtracks: 156845

[128,up,down,time_out]

CPU time 0.998 seconds. Backtracks: 0

[128,up,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[128,up,split,time_out]

CPU time 1.002 seconds. Backtracks: 106459

[128,up,up,time_out]

CPU time 1.001 seconds. Backtracks: 207676

[128,up,updown,time_out]

CPU time 2.0 seconds. Backtracks: 154446

[256,backward,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,backward,reverse_split,time_out]

CPU time 1.003 seconds. Backtracks: 0

[256,backward,split,time_out]

CPU time 0.998 seconds. Backtracks: 51610

[256,backward,up,time_out]

CPU time 1.002 seconds. Backtracks: 116811

[256,backward,updown,time_out]

CPU time 1.001 seconds. Backtracks: 83778

[256,constr,down,time_out]
```

```
CPU time 1.002 seconds. Backtracks: 0

[256,constr,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,constr,split,time_out]

CPU time 1.002 seconds. Backtracks: 51075

[256,constr,up,time_out]

CPU time 1.002 seconds. Backtracks: 112026

[256,constr,updown,time_out]

CPU time 1.001 seconds. Backtracks: 83973

[256,degree,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,degree,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,degree,split,time_out]

CPU time 1.001 seconds. Backtracks: 103480

[256,degree,up,time_out]

CPU time 1.001 seconds. Backtracks: 108323

[256,degree,updown,time_out]

CPU time 1.002 seconds. Backtracks: 78978

[256,ff,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,ff,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[256,ff,split,time_out]

CPU time 1.001 seconds. Backtracks: 38723

[256,ff,up,time_out]

CPU time 1.001 seconds. Backtracks: 155757

[256,ff,updown,time_out]
```

```
CPU time 1.002 seconds. Backtracks: 77067

[256,ffc,down,time_out]

CPU time 1.001 seconds. Backtracks: 0

[256,ffc,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,ffc,split,time_out]

CPU time 1.002 seconds. Backtracks: 37551

[256,ffc,up,time_out]

CPU time 1.002 seconds. Backtracks: 156916

[256,ffc,updown,time_out]

CPU time 0.021 seconds. Backtracks: 2223

[256,ffd,down,success]

CPU time 0.023 seconds. Backtracks: 0

[256,ffd,reverse_split,success]

CPU time 0.022 seconds. Backtracks: 0

[256,ffd,split,success]

CPU time 0.022 seconds. Backtracks: 1172

[256,ffd,up,success]

CPU time 0.994 seconds. Backtracks: 148227

[256,ffd,updown,time_out]

CPU time 1.002 seconds. Backtracks: 85084

[256,forward,down,time_out]

CPU time 0.994 seconds. Backtracks: 0

[256,forward,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,forward,split,time_out]

CPU time 0.998 seconds. Backtracks: 51445

[256,forward,up,time_out]
```

```
CPU time 1.001 seconds. Backtracks: 112556

[256,forward,updown,time_out]

CPU time 1.001 seconds. Backtracks: 274662

[256,inout,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,inout,reverse_split,time_out]

CPU time 1.001 seconds. Backtracks: 0

[256,inout,split,time_out]

CPU time 1.001 seconds. Backtracks: 202544

[256,inout,up,time_out]

CPU time 1.002 seconds. Backtracks: 158491

[256,inout,updown,time_out]

CPU time 1.002 seconds. Backtracks: 84051

[256,leftmost,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,leftmost,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,leftmost,split,time_out]

CPU time 1.003 seconds. Backtracks: 51272

[256,leftmost,up,time_out]

CPU time 1.002 seconds. Backtracks: 112490

[256,leftmost,updown,time_out]

CPU time 1.002 seconds. Backtracks: 77082

[256,max,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,max,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,max,split,time_out]
```

```
CPU time 0.998 seconds. Backtracks: 85995

[256,max,up,time_out]

CPU time 1.002 seconds. Backtracks: 188784

[256,max,updown,time_out]

CPU time 0.998 seconds. Backtracks: 110535

[256,min,down,time_out]

CPU time 0.999 seconds. Backtracks: 0

[256,min,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,min,split,time_out]

CPU time 1.003 seconds. Backtracks: 45505

[256,min,up,time_out]

CPU time 1.001 seconds. Backtracks: 78160

[256,min,updown,time_out]

CPU time 1.003 seconds. Backtracks: 82843

[256,up,down,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,up,reverse_split,time_out]

CPU time 1.002 seconds. Backtracks: 0

[256,up,split,time_out]

CPU time 1.002 seconds. Backtracks: 49771

[256,up,up,time_out]

CPU time 1.002 seconds. Backtracks: 109682

[256,up,updown,time_out]
```

Exercises

Exercise: Magic square

Arrange numbers $1, 2, \ldots, n^2$ in a square such that every row, every column, and the two main diagonals all sum to the same quantity.

- Try to find the best model, solver and search strategy.
- How many magic squares are there for a given n?
- Allow also for a partially filled instance.

Exercise: Minesweeper

Identify the positions of all mines in a given board. Try the following instance (from the book):

```
Instance = {
    {_,_,2,_,3,_},
    {2,_,_,_,_,_},
    {_,_,2,4,_,3},
    {1,_,3,4,_,_},
    {_,_,_,_,_,3},
    {_,3,_,3,_,_}
}.
```

Exercise: Graph-coloring

1. Write a program that solves the (directed) graph 3-coloring problem with a given number of colors and a given graph. The graph is given by a list of edges, each edge is a 2-element list. We assume that vertices of the graph are $1, \ldots, n$ where n is the maximum number appearing in the list.
2. Generalize your program to graph k-coloring where k is a positive integer given on the input.
3. Modify your program to accept the incidence matrix (a 2D array) instead of the list of edges.
4. Add the flag `-n` to output the minimum number of colors (the chromatic number) of a given graph. For example:

```
picat graph-coloring.pi [[1,2],[2,3],[3,4],[4,1]]
picat graph-coloring.pi [[1,2],[2,3],[3,1]] 4
picat graph-coloring.pi "{{0,1,1},{1,0,1},{1,1,0}}" 4
picat graph-coloring.pi -n [[1,2],[2,3],[3,4],[4,1]]
```

Knapsack

There are two common versions of the problem: the general **knapsack** problem:

> Given a set of items, each with a weight and a value, determine **how many of each item** to include in a collection so that the total

> weight is less than or equal to a given limit and the total value is as large as possible.

And the **0-1 knapsack** problem:

> Given a set of items, each with a weight and a value, determine **which items** to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

(In a general knapsack problem, we can take any number of each item, in the 0-1 version we can take at most one of each.)

Example of an instance:

A thief breaks into a department store (general knapsack) or into a home (0-1 knapsack). They can carry 23kg. Which items (and how many of each, in the general version) should they take to maximize profit? There are the following items:

- a TV (weighs 15kg, costs $500),
- a desktop computer (weighs 11kg, costs $350)
- a laptop (weighs 5kg, costs $230),
- a tablet (weighs 1kg, costs $115),
- an antique vase (weighs 7kg, costs $180),
- a bottle of whisky (weighs 3kg, costs $75), and
- a leather jacket (weighs 4kg, costs $125).

This instance is given in the file `data.pi`.

In [11]:
```
!cat knapsack/data.pi
```

```
instance(Items, Capacity, Values, Weights) =>
    Items = {"tv", "desktop", "laptop", "tablet", "vase", "bottle", "jacket"},
    Capacity = 23,
    Values = {500,350,230,115,180,75,125},
    Weights = {15,11,5,1,7,3,4}.
```