Constraint Logic Programming

What is Constraint Logic Programming?

◊ Is a combination of
  - » **Logic programming**
  - » **Optimization**
  - » **Artificial Intelligence**

Components

◊ Have a set of variables

» **Each variable ranges over a domain of values**

> **X in 1 .. 20**

– **X has values between 1 and 20 inclusive – finite domain**

> **[X , Y] ins 1 .. 20**

– **X and Y each have values between 1 and 20 inclusive**

> **X**

– **If the library loaded is the CLP on real numbers then X is any real number – infinite domain**

Components 2

- ◊ Have constraints on subsets of the variables
  - » **Y = X + 1**
  - » **Y = X + 1 , 2 * Y =< 8 – X**
  - » **Y = X + 1 , 2 * Y =< 8 – X , Z = 2 * X + 3 *Y**
    - > **Here we assume the base type of X, Y and Z are real numbers**
- ◊ Note that constraints can be on single variables to restrict the range, effectively defining the domain of values
  - » **X > 0 , X < 21**

Components 3

◊ Have built-in operators

» **maximize (Z)**

» **minimize (Z + 10 * Y)**

» **inf (Z , I)**

» **sup (Z – Y , S)**

Putting it together

- ◊ Assuming variables are in the real number domain
  - » **Try different variations of the following in CLP(R)**

```
{ X >= 2 ,                 -- Specify domain of X
  Y >= 2 ,                 -- Specify domain of Y
  Y = X + 1 ,              -- Constraint 1 on X & Y
  2 * Y =< 8 – X,          -- Constraint 2 on X & Y
  Z = 2 * X + 3 * Y } ,    -- Constrain Z wrt X and Y
  maximize (Z+Y) ,         -- Another constraint
  inf (Z , I) ,            -- I is the the infimum (minimum) of Z
  sup (Z–Y , S).           -- S is the supremum (maximum) of Z-Y
```

Purpose

- Satisfy the constraints
  - **Find an assignment of values to the variables such that all the constraints are simultaneously true**
  - **In optimization problems find the best assignment of values**
    - **Maximize, minimize, etc.**

What is in SWIPL

- SWI-prolog has various libraries that can be consulted
  - **[library(clpr)].**
    - **An implementation of CLP(R) with variables being real numbers with real arithmetic**
  - **[library(clpq)].**
    - **An implementation of CLP(Q) with variables being rational numbers (ratios of integers)**
  - **[library(clpfd)].**
    - **An implementation of CLP(FD) with variables being in finite domains**
  - **[library(clpqr)].**
    - **Combination of rationals and reals**

CLP(Q) CLP(R) comparison

- Try the following
  - **:- library(clpq)**
  - **{ X = 2 * Y , Y = 1 – X}.**

- Compare with what is done in CLP(R)
  - **:- library(clpr)**
  - **{ X = 2 * Y , Y = 1 – X}.**

CLP(R) Exercise

◊ Try the expression in slide CLP-6, adding one expression after another until the full slide is done

Fahrenheit <--> Celsius

- Consider a predicate to convert between Fahrenheit and Celsius
  - » **convert (Fahrenheit , Celsius) :-**
    Celsius is (Fahrenheit – 32) * 5 / 9.
    - > **Can only go in one direction because "is" requires Fahrenheit to be instantiated**

- Using CLP we can go both ways
  - » **convert (Fahrenheit , Celsius) :-**
    { Celsius = (Fahrenheit – 32) * 5 / 9 }.

  - » **convert (Fahrenheit , Celsius) :-**
    { Fahrenheit = Celsius* 5 / 9 + 32 }.

PERT & CPM

PERT == Program Evaluation and Review Technique

CPM == Critical Path Method

◊ Both are methods used in managing the complex scheduling of tasks that occur, for example, in building projects

CPM & PERT Graph

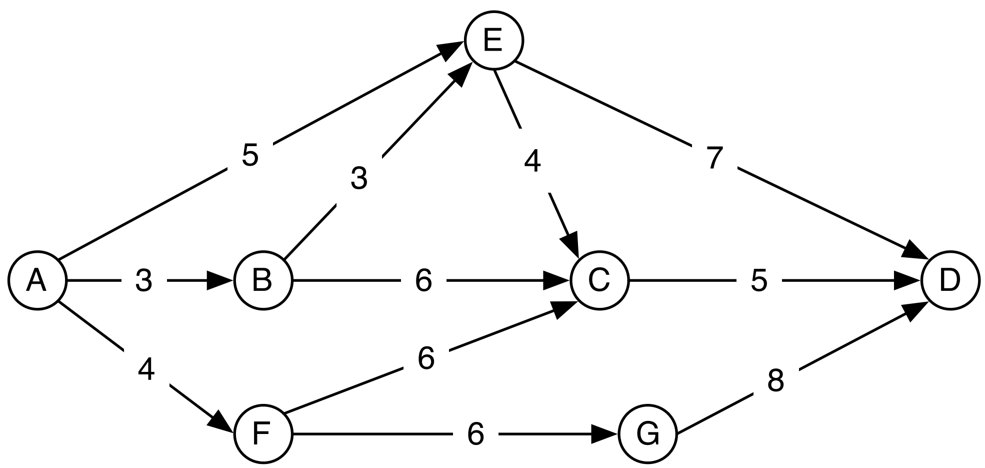
◊ Is a graph where

» **Nodes are end points for tasks**

> **Tasks begin or end at nodes**

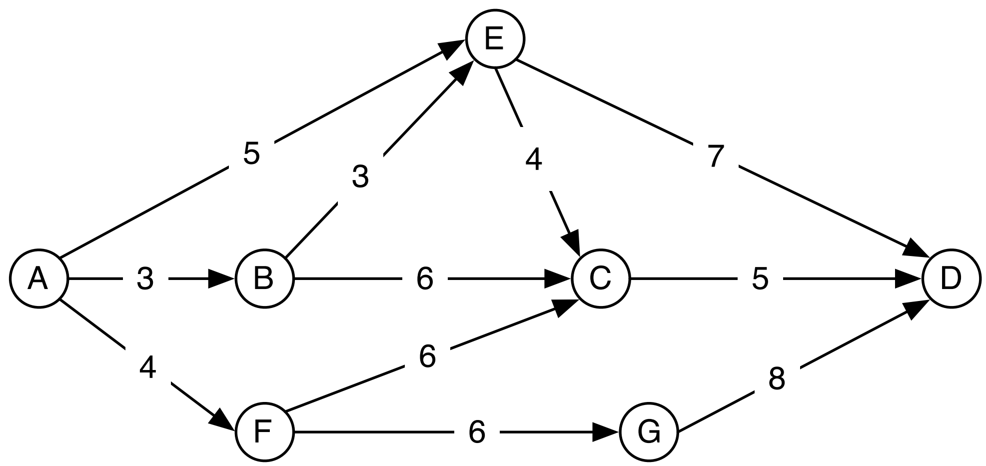
» **Arcs are duration time for tasks**

> **Have a duration time associated with them**



CPM & PERT Graph – 2

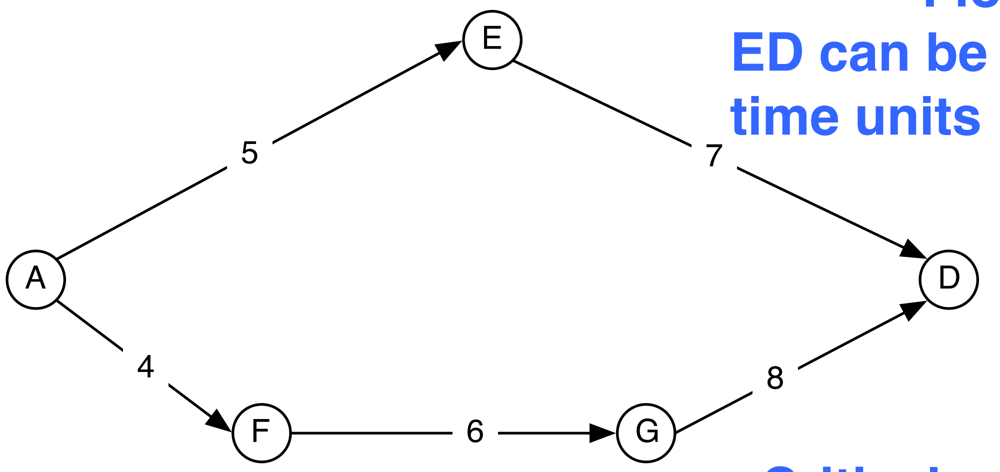
- A task cannot start until all its precedence tasks are completed
  - **E.G. Task CD must wait until tasks EC, BC and FC are completed before it can start**



PERT & CPM Objectives

- Find the critical path of tasks such that if any task is delayed the entire project is delayed, hence resources are allocated to minimize delay
- Another objective is to find where there is float-time in the schedule so resources can be moved from non-critical tasks to critical tasks

Float-time at E
ED can be delayed in starting by 6 time units



Critical path is AFGD – 18 time units

Scheduling Example – Figure 7.1

◊ The textbook gives the following scheduling algorithm

» {Ta = 0 ,
Ta + 2 =< Tb ,
Ta + 2 =< Tc ,
Tb + 3 =< Td ,
Tc + 5 =< Tf ,
Td + 4 =< Tf } , minimize(Tf).

Note, you have to construct a final node F, with zero duration, and appropriate arcs to it.

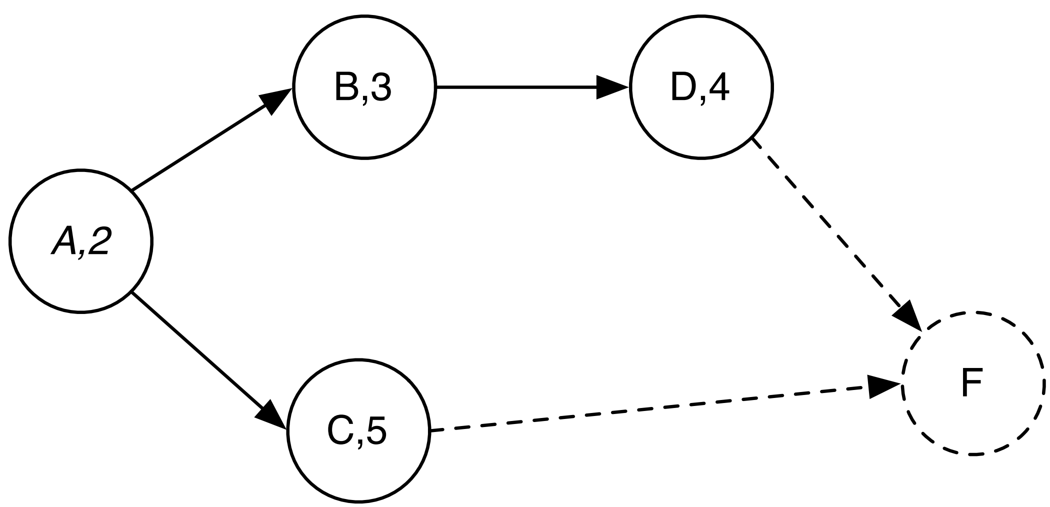
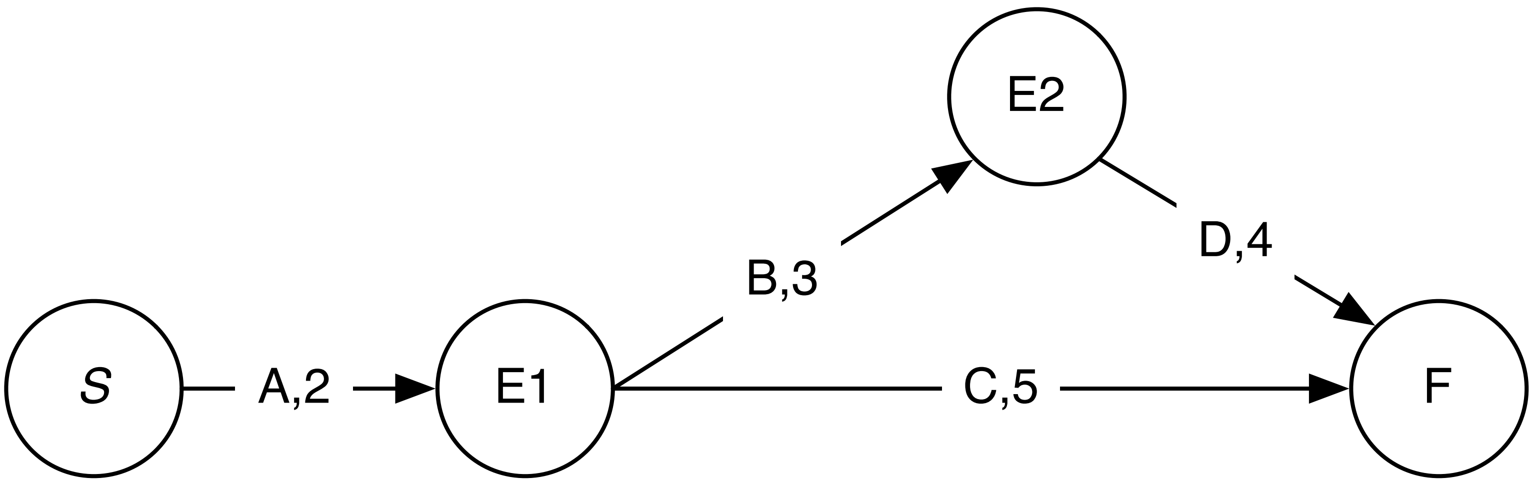


Figure 7.1as a CPM / PERT graph

» { Start = 0 ,
Start + 2 =< E1,
E1+ 3 =< E2,
E2+ 4 =< F,
E1+ 5 =< F} , minimize(F).

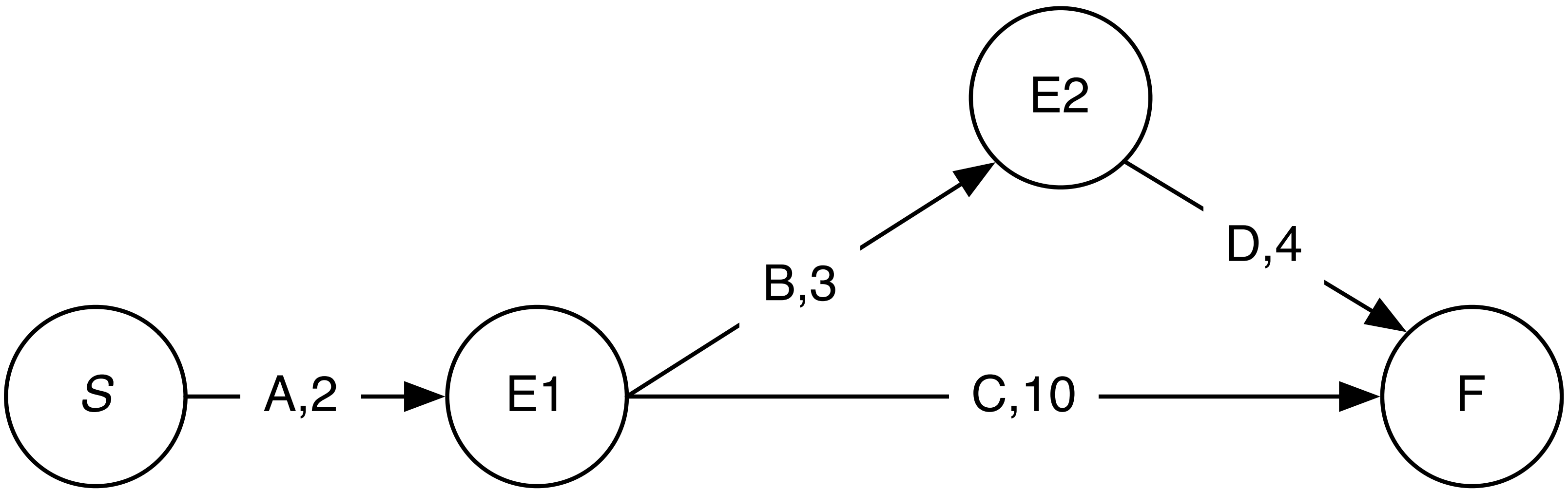
Nodes are start/stop task events. Edges are tasks, with duration.



Critical path is S,E1,E2,F.
Task C has a float of 2 time units.

Showing D with delayed start time

» { Start = 0 ,
Start + 2 =< E1,
E1+ 3 =< E2,
E2+ 4 =< F,
E1+ 5 =< F} , minimize(F), maximize(E2).



Fibonacci – Ordinary Recursion

◊ Following is a recursive definition of the Fibonacci series. For reference here are the first few terms of the series

Index –	0	1	2	3	4	5	6	7	8	9	10	11	12
Value –	1	1	2	3	5	8	13	21	34	55	89	144	233

Fibonacci (N) = Fibonacci (N – 1)
+ Fibonacci (N – 2).

```
fib ( 0 , 1 ).
fib ( 1 , 1 ).
fib ( N , F )  :-  N1  is  N – 1 ,  N2  is  N – 2
                 ,  fib ( N1 , F1 ) , fib ( N2 , F2 )
                 ,  F  is  F1 + F2.
```

◊ Does not work for queries **fib (N , 8)** and **fib (N , F)**

» **Values for is operator are undefined.**

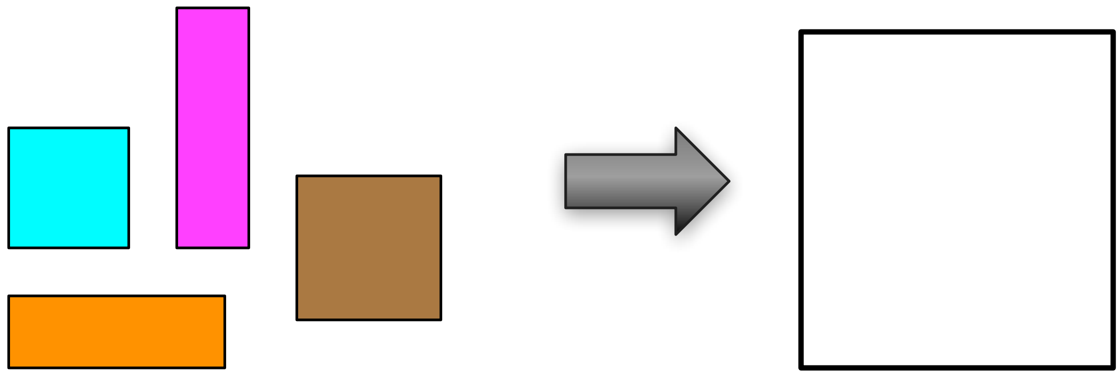
Fibonacci with CLP

```
fib_clp(N , F) :-
        { N = 0 , F = 1 }
;
        { N = 1 , F = 1 }
;
        { N >= 2 ,
          F = F1 + F2 ,
          N1 = N – 1 ,
          N2 = N – 2 ,

          F1 >= N1 ,
          F2 >= N2 }

        fib_clp ( N1, F1) , fib_clp ( N2 , F2).
```

With accumulators we will see another solution

Add for computational needs, not logical needs.

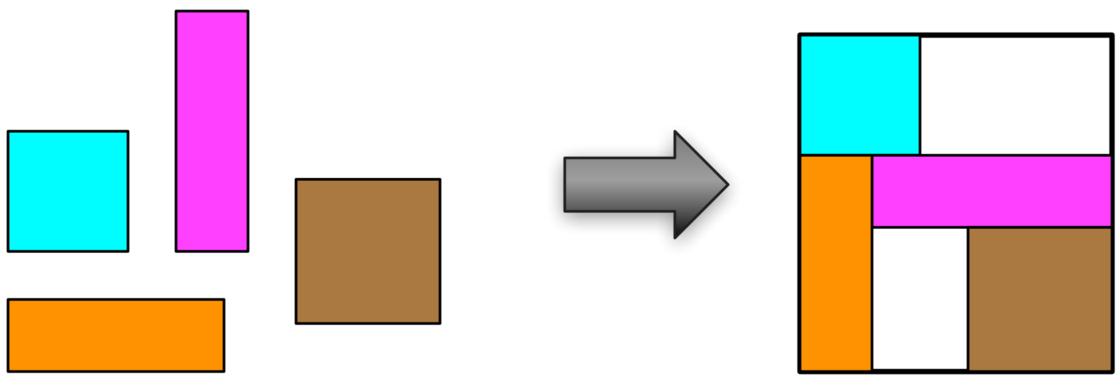
Packing blocks into boxes



- Constraints
  - **All objects are rectangular in two dimensional space**
  - **Sides of rectangles are parallel to the axes**
  - **Rectangles have a height and width**

A Pictorial Solution

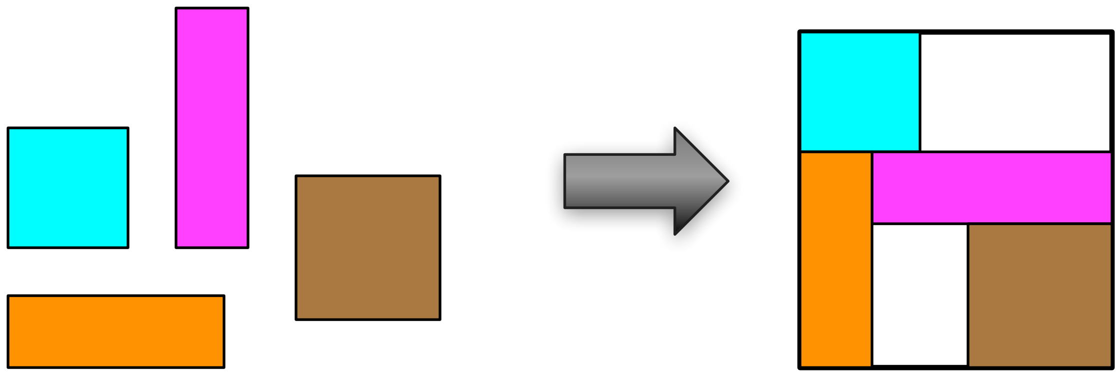
◊ Blocks can be rotated by 90 degrees within the box.



» **What needs to be done to get a solution in Prolog?**

A Pictorial Solution – 2

◊ Blocks can be rotated by 90 degrees within the box.



» **What needs to be done to get a solution in Prolog?**

» **Is all of the work unique to Prolog?**

DONALD + GERALD = ROBERT

◊ Crypt arithmetic puzzles are like the following, where digits 0..9 replace the letters

	DONALD			526485
+	GERALD	→	+	197485
	ROBERT			723970

DONALD + GERALD = ROBERT – 2

```
solve( [D,O,N,A,L,D] , [G,E,R,A,L,D] , [R,O,B,E,R,T])  :-
    Vars = [D,O,N,A,L,G,E,R,B,T],        % All variables in the puzzle
    Vars ins 0..9,                        % They are all decimal digits
    all_different( Vars),                 % They are all different
    100000*D + 10000*O + 1000*N + 100*A + 10*L + D  +
    100000*G + 10000*E + 1000*R + 100*A + 10*L + D  #=
    100000*R + 10000*O + 1000*B + 100*E + 10*R + T,
%   labeling( [], Vars).
    label(Vars).                          % Use default labeling
```

You can time predicate execution

» stats (Time) :-
```
stats ( Time ) :-
        statistics ( runtime , _ ) ,
        solve ( _ , _ , _ ) ,
        statistics ( runtime , [ _ , Time ] ).
```

SEND + MORE = MONEY

```
solve( [S,E,N,D] + [M,O,R,E] = [M,O,N,E,Y] )  :-
    Vars = [S,E,N,D,M,O,R,Y],            % All variables in the puzzle
    Vars ins 0..9,                        % They are all decimal digits
    all_different(Vars),                  % They are all different
    1000*S + 100*E + 10*N + D  +
    1000*M + 100*O + 10*R + E  #=
    10000*M + 1000*O + 100*N + 10*E + Y ,
    M #\= 0 , S #\= 0 ,
/* Systematically try out values for the finite domain variables
    in the set Vars until all of them are ground. */
    labeling( [], Vars).
```

Replacement for page 194

- maximize (indomain (X) , Y) does not exist in swipl
  - **Replace with the following**
  - **X in 1 .. 20 , Y #= X * (20 – X) ,
    once (labeling ([max (Y)] , [X , Y])).**
  - **[X ,Y] ins 1 .. 20 , 2 * X + Y #=< 40 ,
    once (labeling ([max (X * Y)] , [X , Y])).**

Replacement for page 194 – 2

◊ Compare the following with schedule1 in CLP(R)

» **Replace with the following**

» **schedule1 (A , B , C , D , F) :-**
```
StartTimes = [ A , B , C , D , F ] ,
StartTimes ins 0 .. 20 ,
A + 2 #=< B ,
A + 2 #=< C ,
B + 3 #=< D ,
C + 5 #=< F ,
D + 4 #=< F,
once ( labeling ( [ min ( F ) ] , [ A , B , C , D , F ] ) ).
```