NOPT042 Constraint programming: Tutorial 11 - Tabling

In [1]:
```
%load_ext ipicat
```

```
Picat version 3.5#5
```

Dynamic programming with tabling

The "t" in Picat stands for "tabling": storing and resusing subcomputations, most typically used in dynamic programming (divide & conquer). We have already seen the following classical example of usefulness of tabling:

Example: Fibonacci sequence

In [2]:
```
%%picat -n fib
fib(0, F) => F = 0.
fib(1, F) => F = 1.
fib(N, F), N > 1 => fib(N - 1, F1), fib(N - 2, F2), F = F1 + F2.
```

In [3]:
```
%%picat -n fib_tabled
table
fib_tabled(0, F) => F = 0.
fib_tabled(1, F) => F = 1.
fib_tabled(N, F), N > 1 => fib_tabled(N - 1, F1), fib_tabled(N - 2, F2), F = F1 + F
```

Compare the performance:

In [4]:
```
%%picat
main =>
    time(fib_tabled(42, F)),
    println(F),
    time(fib(42, F)),
    println(F).
```

```
CPU time 0.0 seconds.

267914296

CPU time 26.975 seconds.

267914296
```

Example: shortest path

Find the shortest path from source to target in a weighted digraph. Code from the book:

```
table(+,+,-,min)

sp(X,Y,Path,W) ?=>
  Path = [(X,Y)],
  edge(X,Y,W).

sp(X,Y,Path,W) =>
  Path = [(X,Z)|Path1],
  edge(X,Z,Wxz),
  sp(Z,Y,Path1,W1),
  W = Wxz+W1.
```

Recall that `?=>` means a backtrackable rule. Consider the following simple instance:

```
index (+,-,-)
edge(a,b,5).
edge(b,c,3).
edge(c,a,9).

source(a).
target(c).
```

In [5]:
```
!time picat shortest-path/shortest-path.pi instance2.pi
```

```
path = [(1,2),(2,4),(4,8),(8,6)]
w = 20

real	0m0.019s
user	0m0.006s
sys	0m0.007s
```

In [6]:
```
!cat shortest-path/shortest-path.pi
```

```
/***********************************************************
  Adapted from
  sp1.pi
  from Constraint Solving and Planning with Picat, Springer
  by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
***********************************************************/

main([Filename]) =>
  cl(Filename),
  source(S),
  target(T),
  sp(S,T,Path,W),
  println(path = Path),
  println(w = W).


table(+,+,-,min)

sp(X,Y,Path,W) ?=>
  Path = [(X,Y)],
  edge(X,Y,W).

sp(X,Y,Path,W) =>
  Path = [(X,Z)|Path1],
  edge(X,Z,Wxz),
  sp(Z,Y,Path1,W1),
  W = Wxz+W1.
```

Table mode declaration

We can tell Picat what to table using a *table mode declaration*:

```
table(s1,s2,...,sn)
my_predicate(X1,...,Xn) => ...
```

where `si` is one of the following:

- `+` : input, the row/column/etc. where to store
- `-` : output, the value to store
- `min` or `max` : objective, only store outputs with smallest/largest value of this
- `nt`: not tabled, as if this argument was not passed; last coordinate only, you can use this for global data that do not change in the subproblems, or for arguments functionally dependent (1-1, easily computable) on the + arguments

For example:

```
table(+,+,-,min)
sp(X,Y,Path,W)
```

means for every X and Y store (only) the Path with minimum weight W (only rewrite Path if its W is smaller).

Index declaration

The *index declaration* `index (+,-,-)` does not change semantics but facilitates faster lookup when unifying e.g. terms `edge(a,X,W)`, see Wikipedia. The `+` means that the corresponding coordinate is indexed ("an input"), `-` means not indexed ("an output"). There can be multiple index patterns, e.g. an undirected graph can be given as:

```
index (+,-) (-,+)
edge(a,b).
edge(a,c).
edge(b,c).
edge(c,b).
```

if we want to traverse the edges in both ways. (This example is from the guide.)

In [7]:
```
!cat table-mode-example.pi
```

```
/************************************************************
  table_mode.pi
  from Constraint Solving and Planning with Picat, Springer
  by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
************************************************************/
main ?=>
    p(a,Y),
    println("Y" = Y).

table(+,max)
index (-,+)
p(a,2).
p(a,1).
p(a,3).
p(b,3).
p(b,4).
```

In [8]:
```
!picat table-mode-example.pi
```

```
Y = 3
```

Exercise: shortest shortest path

Modify the above example so that among the minimum-weight paths, only one with minimum *length*, meaning number of edges, is chosen.

In [9]:
```
!cat shortest-path/instance.pi
```

```
index (+,-,-)

edge(a,b,5).
edge(b,c,3).
edge(c,a,9).

source(a).
target(c).
```

In [10]:
```
!picat shortest-path/shortest-shortest-path.pi instance2.pi
```

```
path = [(1,2),(2,4),(4,8),(8,6)]
w = (20,4)
```

In [11]:
```
!cat shortest-path/shortest-shortest-path.pi
```

```
/**********************************************************
  Adapted from
  sp2.pi
  from Constraint Solving and Planning with Picat, Springer
  by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
**********************************************************/

main([Filename]) =>
  cl(Filename),
  source(S),
  target(T),
  ssp(S,T,Path,W),
  println(path = Path),
  println(w = W).

table(+,+,-,min)

ssp(X,Y,Path,WL) ?=>
  Path = [(X,Y)],
  WL = (Wxy,1),
  edge(X,Y,Wxy).

ssp(X,Y,Path,WL) =>
  Path = [(X,Z)|Path1],
  edge(X,Z,Wxz),
  ssp(Z,Y,Path1,WL1),
  WL1 = (Wzy,Len1),
  WL = (Wxz+Wzy,Len1+1).

% The order in `WL = (Weight, Length)` matters, otherwise we would choose minimum-we
ight path among minimum-edges paths.
```

Exercise: edit distance

Find the (length of the) shortest sequence of edit operations that transform `Source` string to `Target` string. There are two types of edit operations allowed:

- insert: insert a single character (at any position)
- delete: delete a single character (at any position)

Once you can compute the distance, try also outputing the sequence of operations.

In [12]:
```
# this should output 4
!picat edit-distance/edit2.pi saturday sunday
```

```
dist = 4
[del(2,a),del(2,t),ins(3,n),del(4,r)]
```

In [13]:
```
!cat edit-distance/edit2.pi
```

```
/************************************************************
        Adapted from
        edit.pi
        from Constraint Solving and Planning with Picat, Springer
        by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
************************************************************/
main([Source, Target]) =>
        edit(Source, Target, Distance, Seq, 1),
        writeln(dist=Distance),
        writeln(Seq).

table(+,+,min)

% base
edit([],[],D,Seq, I) =>
        D=0,
        Seq=[].

% match
edit([X|P],[X|T],D,Seq,I) =>
        edit(P,T,D,Seq,I+1).

% insert
edit(P,[X|T],D,Seq,I) ?=>
        edit(P,T,D1,Seq1,I+1),
        Seq=[$ins(I,X)|Seq1],
        D=D1+1.

% delete
edit([X|P],T,D,Seq,I) =>
        edit(P,T,D1,Seq1,I),
        Seq=[$del(I,X)|Seq1],
        D=D1+1.
```

Exercise: knapsack

Write a dynamic program for the knapsack problem.

In [14]:
```
!cat knapsack/knapsack.pi
```

```
/**********************************************************
  Code adapted from
  knapsack.pi
  from Constraint Solving and Planning with Picat, Springer
  by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
**********************************************************/

main([Filename]) =>
    cl(Filename),
    instance(ItemNames, Capacity, Values, Weights),
    Items = [(ItemNames[I], Values[I], Weights[I]) : I in 1..ItemNames.length],
    knapsack(Items, Capacity, ChosenItems, TotalValue),
    output(ChosenItems, TotalValue).


table(+,+,-,max)

knapsack(_, Capacity, ChosenItems, Value), Capacity =< 0 =>
  ChosenItems = [], Value = 0.

knapsack([ _ | RemainingItems], Capacity, ChosenItems, Value) ?=>
  % Don't take the item
  knapsack(RemainingItems, Capacity, ChosenItems, Value).

knapsack([Item@(ItemName, ItemValue, ItemWeight) | RemainingItems], Capacity, Chosen
Items, Value), Capacity >= ItemWeight =>
  % Take the item
  ChosenItems = [Item | PrevChosenItems],
  knapsack(RemainingItems, Capacity - ItemWeight, PrevChosenItems, PrevValue),
  Value = PrevValue + ItemValue.

output(ChosenItems, TotalValue) =>
    println(total=TotalValue),
    foreach(Item in ChosenItems)
      println(Item)
    end.
```

In [15]:

```
# !picat knapsack/knapsack.pi instance.pi
```

In [16]:

```
# !cat knapsack/knapsack.pi
```

Homework: Maximum path in a triangle

We are given a triangle filled with positive integers. Find a maximum-sum path from the top vertex to the bottom side: at every step we can go either down or down and right. This is Problem 67 in Project Euler. Read the input from a file (each level is one line line), see the attached two instances. The easy one is:

```
3
7 4
```

```
2 4 6
8 5 9 3
```

Its solution is to go down, down+right, down+right: 3 + 7 + 4 + 9 = 23. Running

```
picat triangle.pi small.txt
```

should output `23`. The optimal value for the instance `big.txt` is `7273`. Also output some representation of the path: which type of step to take at every level, here e.g. `[0,1,1]`.

Try to write a Picat program that uses dynamic programming with tabling. Optionally, you can create a constraint model and use the cp solver.