Chapter 45 : Dynamic Programming: Knapsack without repetition.

We are given

A knapsack which can hold a maximum weight of M Kg.

A collection of N items [1..N]

Each item has a weight W[1..N] and a value V[1..N]

Our task is to maximise the value we can fit in the knapsack given the weight constraint. We are told that there is just one instance of each item available.

We need to define a function which gives us the maximum value which can be contained in a knapsack with weight capacity m. However, in choosing whether to include an item in the knapsack we must know whether it has already been included or not. So our function will need to take another parameter which records the items which may have been chosen.

K : weight x set_of_items $\rightarrow$ value

* (0) K.m.0 = 0

If there are no items to choose from then the maximum achievable value is 0.

* (1) K.0.n = 0

If the capacity of the knapsack is 0 then the maximum achievable value is also 0.

* (2) K.m.n = K.m.(n-1) <= $m < W.n$

If the knapsack does not have the capacity to fit item n, then the maximum value is that which is achievable choosing items up to but not including item n.

* (3) K.m.n = K.m.(n-1) $\uparrow$ K.(m-W.n).(n-1) + V.n <= $W.n \leq m$

Include item n if it increases the value and otherwise don't include it.

Given this definition our postcondition can now be written.

Post: r = K.M.N

Looking at the shape of (2), (3) and at the shape of the recursive calls, it seems that K.m.n can be computed if we already have the solutions for K.x.(n-1) where $0 \leq x \leq m$. Those solutions in turn will have depended on having solutions to K.x.(n-2) etc. So I am going to propose that we use a 2 dimensional array to store these results.

We declare H[0..M, 0..N] of integer.

We propose some invariants.

P0 : $\langle \forall j : 0 \le j \le M : H.j.n = K.j.n \rangle$
P1: $0 \le n \le N$

Establish Invariants.

```
        (n := 0).P0
=               {text substitution }
        ⟨ ∀ j : 0 ≤ j ≤ M : H.j.0 = K.j.0⟩
=               {(0)}
        ⟨ ∀ j : 0 ≤ j ≤ M : H.j.0 = 0⟩
```

So, we can achieve this with a simple loop

```
        n, i := 0, 0
        ;do i ≠ M+1 →
                i, H.i.n := i +1, 0
        od
```

Termination.

```
        (n := N).P0
=               {text substitution}
        ⟨ ∀ j : 0 ≤ j ≤ M : H.j.N = K.j.N⟩
=>              { Instantiate j := M}
        H.M.N = K.M.N
```

Guard.

$n \ne N$

Variant.

N-n

Loop body.

```
        (n := n+1).P0
=               {text substitution}
        ⟨ ∀ j : 0 ≤ j ≤ M : H.j.(n+1) = K.j.(n+1)⟩
```

A single assignment is not going to change P0 into this. So, we propose a new invariant.

Q0: $\langle \forall j : 0 \le j \le m : H.j.(n+1) = K.j.(n+1) \rangle$
Q1: $0 \le m \le M$

Establish Invariants.

$(m := 0).Q0$

= {text substitution}

$\langle \forall j : 0 \le j \le 0 : H.j.(n+1) = K.j.(n+1)\rangle$

= { 1-point}

$H.0.(n+1) = K.0.(n+1)$

= {(1)}

$H.0.(n+1) = 0$

So we can establish by

$m, H.0.(n+1) := 0, 0$

Termination.
Observe.

$(m := M).Q0$

= {text substitution}

$\langle \forall j : 0 \le j \le M : H.j.(n+1) = K.j.(n+1)\rangle$

So, $Q0 \wedge Q1 \wedge m = M \Rightarrow (n := n+1).\ P0$.

Guard.
$m \ne M$

Variant.
$M-m$

Loop body.

$(m := m+1).Q0$

= {text substitution}

$\langle \forall j : 0 \le j \le m+1 : H.j.(n+1) = K.j.(n+1)\rangle$

= {split off j = m+1 term}

$\langle \forall j : 0 \le j \le m : H.j.(n+1) = K.j.(n+1)\rangle \wedge H.(m+1).(n+1) = K.(m+1).(n+1)$

= {Q0}

$H.(m+1).(n+1) = K.(m+1).(n+1)$

= { case $m+1 < W.(n+1)$ (2)}

$H.(m+1).(n+1) = K.(m+1).n$

= {P0}

$H.(m+1).(n+1) = H.(m+1).n$

$(m := m+1).Q0$

= {text substitution}

$\langle \forall j : 0 \le j \le m+1 : H.j.(n+1) = K.j.(n+1)\rangle$

= {split off j = m+1 term}

$\langle \forall j : 0 \le j \le m : H.j.(n+1) = K.j.(n+1)\rangle \wedge H.(m+1).(n+1) = K.(m+1).(n+1)$

= {Q0}

$H.(m+1).(n+1) = K.(m+1).(n+1)$

= {case $W.(n+1) \le m+1$ (3) }

$H.(m+1).(n+1) = K.(m+1).n \uparrow (K.((m+1) - W.(n+1)).n + V.(n+1))$

= {P0}

$H.(m+1).(n+1) = H.(m+1).n \uparrow (H.((m+1) - W.(n+1)).n + V.(n+1))$

Algorithm.

```
n, i := 0, 0
;do i ≠ M+1 →
        i, H.i.n := i +1, 0
od
;do n ≠ N →
        m, H.0.(n+1) := 0, 0
        ; do m ≠ M →
                if m+1 < W.(n+1)  →   m, H.(m+1).(n+1) = m+1, H.(m+1).n
                [] W.(n+1) ≤ m+1  →   m, H.(m+1).(n+1) := m+1,
                                         H.(m+1).n ↑( H.((m+1) - W.(n+1)).n + V.(n+1))
                fi
        od
        ;n := n+1
od
;r := H.M.N
```

The temporal complexity is O(M*N) and the spatial complexity is also O(M*N).