Chapter 44 : Dynamic programming: the Knapsack problem with repetition.

We are given

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

A collection of N items [0..N)

Each item has a weight W[0..N) and a value V[0.N)

Our task is to maximise the value we can fit in the knapsack given the weight constraint. We are also told that there are (sufficiently) many instances of each item available.

We should define a function

K: weight --> value

which returns the maximum value that can be accommodated in a knapsack of a particular capacity. How shall we define it?

* (0) K.0 = 0

We make the reasonable assumption that all of the items have a strictly positive weight. So, if the knapsack can not carry any weight then we can not fit any items into it and so the maximum value of the items it contains will be 0.

* (1) $K.m = \langle \uparrow i : 0 \leq i < N \ \wedge \ W.i \leq m : K.(m - W.i) + V.i \rangle$, $0 < m \leq M$

Here we consider every item which could have been put into a knapsack of weight m and ask which one of those items would have resulted in the best value had it been added.

Let us focus on (1) and ask how many recursive calls there are on the RHS? We actually don't know because that will depend on the actual problem instance. However, we do know that each recursive call to K will involve a smaller argument than m. Yet each of these could involve more calls, probably we are looking at potentially O(m!) or more. Yet, for a given value m the calls will only be to m smaller problems. Why would we want to have this complexity when we could record the values of the function K for all the values smaller than m?

This is a key insight. Perhaps we can ensure that at the point where we want to compute K.m that K.0....K.(m-1) are already available to us. This suggests an invariant, suppose we had an array H[0..M] of int and we imposed these invariants.

P0 : $\langle \forall j : 0 \leq j \leq m : H.j = K.j \rangle$
P1: $0 \leq m \leq M$

Establish Invariants.

(m := 0). P0
= { text substitution}
$\langle \forall j : 0 \leq j \leq 0 : H.j = K.j \rangle$
= { 1-point}
H.0 = K.0
= { (0) }
H.0 = 0

So we can establish the invariants using the assignment

$$m, H.0 := 0, 0$$

Termination.

Observe.

$$\begin{array}{ll} & (m := M).\ P0 \\ = & \quad \{ \text{ text substitution } \} \\ & \langle\ \forall\ j : 0 \leq j \leq\ M : H.j = K.j \rangle \\ \Rightarrow & \quad \{ \text{ Instantiate } j := M \} \\ & H.M = K.M \end{array}$$

So, if we maintain the invariants and reach the point where m = M then H.M contains the value we want.

Guard.

$$m \neq M$$

vf.

$$M-m$$

Loop body.

Within the loop we will decrease the variant by increasing m by 1. We explore this.

$$\begin{array}{ll} & (m := m + 1).P0 \\ = & \quad \{ \text{ text substitution} \} \\ & \langle\ \forall\ j : 0 \leq j \leq\ m+1 : H.j = K.j \rangle \\ = & \quad \{ \text{ split off } j = m+1 \text{ term} \} \\ & \langle\ \forall\ j : 0 \leq j \leq\ m : H.j = K.j \rangle\ \wedge\ H.(m+1) = K.(m+1) \\ = & \quad \{ P0 \} \\ & H.(m+1) = K.(m+1) \\ = & \quad \{(1)\} \\ & H.(m+1) = \langle \uparrow i : 0 \leq i < N\ \wedge\ W.i \leq m+1 : K.((m+1) - W.i) + V.i \rangle \end{array}$$

As the right hand side of this equality is a quantified expression we are not going to achieve this in a single assignment. So, we propose another invariant.

Q0: $s = \langle \uparrow i : 0 \leq i < n\ \wedge\ W.i \leq m+1 : K.((m+1) - W.i) + V.i \rangle$
Q1: $0 \leq n \leq N$

Establish Invariants.

n, s := 0, Id↑

Guard.

n ≠ N

Variant.

N-n

Loop body.

```
        (n, s := n+1, E).Q0
=               { text substitution }
        E = ⟨ ↑ i : 0 ≤ i < n+1  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩
=               { split off i=n case }
        E = ⟨ ↑ i : 0 ≤ i < n  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩ ↑
            ⟨ ↑ i : i =n  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩
=               { case analysis, W.n > m+1, empty range}
        E = ⟨ ↑ i : 0 ≤ i < n  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩ ↑ Id↑
=               { Q0}
        E = s ↑ Id↑
=               { algebra }
        E = s
```

```
        (n, s := n+1, E).Q0
=               { text substitution }
        E = ⟨ ↑ i : 0 ≤ i < n+1  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩
=               { split off i=n case }
        E = ⟨ ↑ i : 0 ≤ i < n  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩ ↑
            ⟨ ↑ i : i =n  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩
=               { case analysis, W.n ≤ m+1, 1-point }
        E = ⟨ ↑ i : 0 ≤ i < n  ∧  W.i ≤ m+1 : K.((m+1) - W.i) + V.i⟩ ↑ K.((m+1) - W.n) + V.n
=               {Q0,  P0 because (m+1)-W.i is in then range 0..m }
        E = s ↑ H.((m+1) - W.n) + V.n
```

Algorithm.

```
        m, H.0 := 0, 0
        ; do m  ≠  M -->
                n, s := 0, Id↑
                ;do n  ≠  N -->
                        if  W.n > m+1 -->  n, s := n+1, s
                        []  W.n ≤ m+1 -->  n, s := n+1, s ↑ H.((m+1) - W.n) + V.n
```

```
                fi
            od
            ; H.(m+1) := s
            ; m := m+1
    od
    ; r := H.M
```

The algorithm has temporal complexity O(N*M) and spacial complexity O(M).