Greedy - Example 2 Continued

Start:D Finish:L Greedy will chose D,G,L. Total distance of 330km, optimum

Start:L Finish:D Greedy will chose L,C,D. Total distance of 370km, sub-optimum

The Travelling Salesperson Problem

This is a classic problem in Computer Science, and is essentially a variant on the above: The Salesperson must visit all the cities on the network exactly once, without "retracing his/her steps". Given the distances, calculate the shortest such route.

The Coins Problem

In a certain country, coins are available in denominations of $c_1, c_2, c_3, \ldots, c_d$. Find the least number of coins that are

necesssary to make up some fixed amount n, assuming we have a large number (∞ if you want) of each coin.

Greedy Solution to coins problem

At each step, choose largest coin that does not excede remaining amount.

Example 1 $c_i = 1, 2, 5, 10, 20, 50$ and $n = 28$:

step number	1	2	3	4
coin chosen	20	5	2	1
running total	20	25	27	28

so, a total of 4 coins: this is the best answer in this case.

Example 2 $c_i = 4, 5$ and $n = 20$:

step number	1	2	3	4
coin chosen	5	5	5	5
running total	5	10	15	20

again the greedy strategy gets the best answer.

Greedy Solution to coins problem - Continued

What about the cases

1. $c_i = 4, 5$ and $n = 19$
2. $c_i = 4, 5$ and $n = 18$
3. $c_i = 4, 5$ and $n = 17$

The (0/1) Knapsack Problem

In this problem, we have a bag/container of (weight) capacity K kilograms. We have n distinct items with weights $w(1), w(2), w(3), \ldots w(n)$ (kg) and corresponding values $v(1), v(2), v(3), \ldots v(n)$ (euro). The aim is to put as many items as possible in to our bag, in order to obtain maximum value.
Note:

- There is only one of each item (which is why it is called the 0/1 knapsack problem)

- If $w(1) + w(2) + w(3) + \cdots + w(n) < K$ then the problem is trivial (i.e. easy to solve): just take everything!

Knapsack Example

Suppose our bag capacity is $K = 4$ kg, and we have $n = 4$ items A, B, C, D, with weights $w(i)$ of 2,3,4,1 and associated values $v(i)$ 5,6,7,2.

	A	B	C	D
weight (kg)	2	3	4	1
value (euro)	5	6	7	2

Clearly all the items don't fit in the bag (since 2+3+4+1 > 4), so which items should we choose?

Brute Force Solution to 0/1 Knapsack Problem

Consider/Enumerate all possibilities:

- Choose just one item: 4 possibilities
- Choose any two items: 6 possibilities
- Choose any three items: 4 possibilities
- Choose four items: 1 possibility

(In actual fact, we don't need to look at all these options: In this example, we can't pick 4 items: We can't even fit any 3 items in the bag.)

For any item, we can either take it or not, i.e. 2 choices. 2 choices each for n items means 2^n possibilities. If for example we had 50 items, 2^{50} is about 1,000,000,000,000,000. Even if we have to consider a fraction of these possibilities, this algorithm would be unworkable for anything but a supercomputer. The Brute Force Solution is $\mathbf{O}(2^n)$.

Greedy Solution to 0/1 Knapsack Problem

Greedy on weight We at each stage pick the lightest item: So we pick D, then A, then terminate with final value of 7 euro in the bag.

Greedy on value We at each stage pick the most valuable item: So we pick C, then terminate with final value of 7 euro in the bag.

Greedy on euro/kg We at each stage pick the most value per kilogram item: So we pick A, then D, then terminate with final value of 7 euro in the bag.

Algorithm Strategies: Recursion

(Sometimes this is viewed as a *methodology* rather than a strategy - i.e. it can be used within other strategies.) Problems which allow a recursive solution have two features:

1. They can be (easily) solved for some small "size" (n) of the problem (typically n is 0 or 1). This is called the base case.
2. The general problem of size n can be broken up (re-written) in terms of smaller problem(s) of size p where $p < n$:
   - Typically p is $n - 1$, or $n/2$, or $n - r$ for some r.
   - The way in which n gets reduced to p must ensure we reach the base case.

   This is called the recursive step.

Recursion - Example 1

Consider the calculation of $n!$ (the factorial of a number):

Base Case Do we know how to solve it for a small value of n? Yes, we know $0! = 1$, or $1! = 1$

Recursive Step Do we know how to write $n!$ in terms of the factorial of a smaller number? Yes - $n!$ is just n multiplied by $(n-1)!$. So we can re-write the problem in terms of one of smaller size p, where in this case p is $n-1$.

We can write the recursive calculation for $n!$ in pseudocode as follows:

```
FACT(i)
IF i=1 THEN FACT ← 1 ELSE FACT ← i*FACT(i-1)
BEGIN
IN(n)
```

```
OUT FACT(n)
END
```

Recursion - Towers of Hanoï

We can denote by `H(i,a,b,c)` our function which will tell us how to move `i` disks from peg `a` to peg `b` (with peg `c` as spare).

Base Case Do we know how to solve it for a small value of n?
Yes, for $n = 1$ we know

$$\texttt{H}(1, \texttt{a}, \texttt{b}, \texttt{c}) \equiv$$

"Move the top disk from peg a to peg b"

Recursion - Towers of Hanoï Continued

Recursive Step Do we know how to write `H(n,a,b,c)` in terms of some other `H` function(s) `H(p,a,b,c)` with $p < n$? Yes -

$$\begin{aligned} H(n,a,b,c) \equiv \quad & H(n-1,a,c,b) \\ & H(1,a,b,c) \\ & H(n-1,c,b,a) \end{aligned}$$

In this recursive step, we re-write the problem of size n in terms of three new problems, of sizes $n-1$, 1, and $n-1$ respectively.

Fibonacci Numbers

The Fibonacci sequence is defined by

$$\begin{aligned} F_n &= 1 && \text{if } n < 3 \\ &= F_{n-1} + F_{n-2} && \textit{otherwise} \end{aligned}$$

which gives us the sequence

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Note that here, the *definition* of the numbers gives us directly the

base case(s) i.e. $F_1 = 1$ and $F_2 = 1$.

recursive step i.e. $F_n = F_{n-1} + F_{n-2}$. Here we re-write a problem of size n in terms of *two* new problems of size $n - 1$ and $n - 2$ respectively.

Recursion - Fibonacci Numbers

We can denote by `fib(i)` our function which will tell us how to calculate the *i*th Fibonacci Number

Base Case Do we know how to solve it for a small value of *i*? Yes, for $i = 1$ we know $f(1) = 1$. Also for $i = 2$ we know $f(2) = 1$

Recursive Step Do we know how to write `fib(i)` in terms of some other function(s) `fib(p)` with `p < i`? Yes, we can re-write `fib(i)` in terms of two functions `fib(p)` with $p = i - 1$ and $p = i - 2$ respectively:

$$\texttt{fib(i)} = \texttt{fib(i} - 1\texttt{)} + \texttt{fib(i} - 2\texttt{)}$$

Th algorithm in pseudocode for the recursive solution to the Fibonacci problem is:

```
FIB(i)
IF i<3 THEN FIB ← 1
     ELSE FIB ← FIB(i-1) + FIB(i-2)

BEGIN
IN(n)
OUT FIB(n)
END
```

Is this efficient? Let $N(n)$ denote the number of operations (additions) we do in this recursive solution. Then

$$N(n) = N(n-1) + N(n-2)$$

We can apply this process repeatedly to get

$$
\begin{aligned}
N(n) &= 1 + N(n-1) + N(n-2) \\
&= 1 + [1 + N(n-2) + N(n-3)] + N(n-2) \\
&= 2 + 2N(n-2) + N(n-3) \\
&= 2 + 2[1 + N(n-3) + N(n-4)] + N(n-3) \\
&= 4 + 3N(n-3) + 2N(n-4) \\
&= 4 + 3[1 + N(n-4) + N(n-5)] + 2N(n-4) \\
&= 7 + 5N(n-4) + 3N(n-5) \\
&= 7 + 5[1 + N(n-5) + N(n-6)] + 3N(n-5) \\
&= 12 + 8N(n-5) + 5N(n-6) \\
&= \ldots \\
&= [f(i+2) - 1] + f(i+1)N(n-i) \\
&\qquad + f(i)N(n-i-1) \qquad \text{in the } i\text{th iteration}
\end{aligned}
$$

Note the coefficients here are the fibonacci numbers themselves. When do we terminate (*base case*)? Well, we know N(1) is 0 as is N(2) (no additions required): So we get that this recursive solution is $\mathbf{O}(fin(n))$.
Where does this fit in our list of functions? Consider

1.5^n i.e. $1, 1.5, 2.2, 3.4, 5.1, 7.6, 11.4, 17.1, 25.6, \ldots$

2^n i.e. $1, 2, 4, 8, 16, 32, 64, 128, 256, 512, \ldots$

The fibonacci sequence falls between these two, so we know it is about $\mathbf{O}(a^n)$ where a is some number between 1.5 and 2.