Dynamic programming

Goldsmiths Computing

Motivation

Technique for applying memoization to optimization problems.

- not really "dynamic";
- not really "programming" (as we understand it today).

Marketing!

Definition

The bottom-up application of memoization (stored computation) to solve problems searching for an optimum (shortest, smallest, ...) of a set of possibilities, where the optimum can be described in terms of subproblems.

Example: factorial

$$n! = \begin{cases} 1 & n < 2 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

```
function FACT(n)
    if n < 2 then
        return 1
    else
        return n × FACT(n−1)
    end if
end function
```

Complexity

time $\Omega(N)$

space $\Omega(N)$

Example: factorial (memoized)

```
T ← new Vector(1000)
for 0 ≤ i < 1000 do
    T ← 0
end for
function FACTMEMO(n)
    if T[n] > 0 then
        return T[n]
    else if n < 2 then
        T[n] ← n; return T[n]
    else
        T[n] ← n × FACTMEMO(n−1); return T[n]
    end if
end function
```

Complexity

time $\Omega(N)$ (first time); $\Theta(1)$ (subsequent times)

space $\Omega(N)$

Example: factorial (dynamic programming)

function FACTDP(n)
  T ← new Vector(n+1)
  T[0] ← 1
  for $0 < i \leq n$ **do**
    T[i] ← n × T[i-1]
  end for
  return T[n]
end function

Example: Fibonacci

$$u_n = \begin{cases} n & n < 2 \\ u_{n-1} + u_{n-2} & \text{otherwise} \end{cases}$$

```
function Fib(n)
    if n < 2 then
        return n
    else
        return Fib(n−1) + Fib(n−2)
    end if
end function
```

Complexity

time $\Omega(\varphi^N)$

space $\Omega(\varphi^N)$

Example: Fibonacci (memoized)

```
T ← new Vector(1000)
for 0 ≤ i < 1000 do
    T ← −1
end for
function FibMemo(n)
    if T[n] ≥ 0 then
        return T[n]
    else if n < 2 then
        T[n] ← n
        return T[n]
    else
        T[n] ← FibMemo(n−1) + FibMemo(n−2)
        return T[n]
    end if
end function
```

Example: Fibonacci (dynamic programming)

```
function FibDP(n)
    T ← new Vector(n+1)
    T[0] ← 0
    T[1] ← 1
    for 1 < i ≤ n do
        T[i] ← T[i-1] + T[i-2]
    end for
    return T[n]
end function
```

Example: coins

Given a collection of denominations {D}, how many coins does it take to make a particular value v?

- extension: in what way can we make v using the smallest number of coins?

Example: coins

```
function GREEDY(D,v)
    if v = 0 then
        return 0
    else
        cs ← {c|c ∈ D ∧ c ≤ v}
        c ← MAX(cs)
        return 1 + GREEDY(D,v-c)
    end if
end function
```

Example: coins

```
function OPT(D,v)
    if v ∈ D then
        return 1
    else if v < MIN(D) then
        return false
    else
        cs ⟵ {OPT(D,v-c)|c ∈ D ∧ Opt(c) ≠ false }
        return 1 + MIN(cs)
    end if
end function
```

Example: coins

```
function LOOKUP(T,i)
    if i < 0 then
        return ∞
    else
        return T[i]
    end if
end function
function OPTDYNAMICPROGRAMMING(D,v)
    T ← new Vector(v)
    T[0] ← 0
    for 0 < i ≤ v do
        cs ← {1 + LOOKUP(T,i-c)|c ∈ D}
        T[i] ← MIN(cs)
    end for
    return T[v]
end function
```

Example: image seam carving

Assume some "energy" measurement for pixels E(i,j)

$$c(i,j) = \begin{cases} E(i,j) & j = 0 \\ E(i,j) + \min(c(i-1,j-1), c(i,j-1), c(i+1,j-1)) & \text{otherwise} \end{cases}$$

```
function SEAM(I)
    w ← WIDTH(I); h ← HEIGHT(I)
    T ← new Array(w+2, h)
    for 0 ≤ i < w do
        T[i+1,j] ← (E(I,i,j),NIL)
    end for
    for 0 ≤ j < h do
        T[0,j] ← (∞,NIL); T[w+1,j] ← (∞,NIL)
    end for
    for 0 < j < h do
        for 0 ≤ i < w do
            T[i+1,j] ← MIN1((T[i,j-1],i), (T[i+1,j-1],i+1), (T[i+2,j-1],i+2))
        end for
    end for
end function
```

Example: edit distance

Operations needed to edit one string into another:

insertion	insert a character into the string (cost: ci)
deletion	delete a character from the string (cost: cd)
substitution	substitute one character for another (cost: cs)

```
function EditDistance(S,Z)
    if length(S) = 0 then
        return ci × length(Z)
    else if length(Z) = 0 then
        return cd × length(S)
    else
        ins ← ci + EditDistance(Z[0]S, Z)EditDistance(S, Z[1..])
        del ← cd + EditDistance(S[1..], Z)
        if Z[0] = S[0] then
            sub ← EditDistance(S[1..], Z[1..])
        else
            sub ← cs + EditDistance(S[1..], Z[1..])
        end if
        return min(ins, del, sub)
    end if
end function
```

Example: edit distance

```
function EditDistanceDP(S,Z)
    ls ← length(S); lz ← length(Z)
    T ← new Array(ls+1, lz+1)
    for 0 ≤ i ≤ ls do
        T[i,0] ← i × cd
    end for
    for 0 ≤ j ≤ lz do
        T[0,j] ← j × ci
    end for
    for 0 < i ≤ ls do
        for 0 < j ≤ lz do
            if S[i-1] = Z[j-1] then
                T[i,j] ← T[i-1,j-1]
            else
                ins ← ci + T[i,j-1]
                del ← cd + T[i-1,j]
                sub ← cs + T[i-1,j-1]
                T[i,j] ← min(ins, del, sub)
            end if
        end for
    end for
    return T[ls,lz]
end function
```

Dynamic programming and memoization

memoization

- small modification of natural recursive definition
- introduction of a cache to store intermediate results
- start from problem, work on progressively smaller cases

dynamic programming

- more substantial rewrite of recursive definition
- introduction of a table to store successive results
- start from base case, work on progressively larger cases

Work

1. Reading
    - CLRS, chapter 15
    - DPV, chapter 6
2. Exercises and Problems

   Exercises from CLRS 15.1-5

   Exercises from DPV 6.1, 6.2

   CLRS 15-4 Printing neatly

   CLRS 15-5 Edit distance