Memoization

Goldsmiths Computing

Motivation

We've seen a trade-off between space and time in various places so far. Is there a systematic way of thinking about it?

Definition

Memoization is the use of some data structure to store the results of previous computations, particularly when those results will be re-used. (Similar: cacheing)

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 (accumulator)

save stack space: use accumulator instead

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

Complexity

time $\Omega(N)$

space $\Omega(1)$

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: 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
```

Work

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

   Exercises from CLRS 15.1-1, 15.1-4