

Lecture 6: Recurrences

Agenda:

- Recurrence relations
 - Merge sort as an example
- Solving recurrences
 - Key method
 - iterated substitution/replacement
 - Merge sort as an example
 - More examples & notes

Reading:

- Textbook pages 62 – 67, Appendix A.

Recurrence relations — Recurrences

- *Recurrence relation*:

A relation defined recursively — in terms of itself. e.g.,

$$f(n) = \begin{cases} 1, & \text{if } n = 1 \\ n + f(n - 1), & \text{if } n \geq 2 \end{cases}$$

- Must have *base case* and *general case*.
- Arise in the analysis of Divide-and-Conquer algorithms
- How are recurrences derived?
- How are recurrences solved?
 - iterated substitution/replacement method
 1. particular cases: solve small examples exactly
 2. general case: guess the answer, prove by induction
 - recurrence tree method (future lectures)
 - master theorem method (future lectures)

Iterated substitution: an easy example

- $f(n) = \begin{cases} 1, & \text{if } n = 1 \\ n + f(n-1), & \text{if } n \geq 2 \end{cases}$

- Particular cases:

n	1	2	3	4	5	6	7
$f(n)$	1	$2 + 1$	$3 + 3$	$4 + 6$	$5 + 10$	$6 + 15$	$7 + 21$
	1	3	6	10	15	21	28

- General case:

$$\begin{aligned}
 f(n) &= n + f(n-1) \\
 &= n + (n-1) + f(n-2) \\
 &= n + (n-1) + (n-2) + f(n-3) \\
 &= n + (n-1) + (n-2) + (n-3) + f(n-4) \\
 &= \dots \\
 &= n + (n-1) + (n-2) + (n-3) + \dots + 2 + f(1) \\
 &= \sum_{i=1}^n i
 \end{aligned}$$

Therefore, we guess that $f(n) = \sum_{i=1}^n i$

(this is NOT a proof, yet).

Prove it by induction.

Iterated substitution: an easy example (cont'd)

- Prove that $f(n) = \sum_{i=1}^n i$ by induction.
 - Base case: $n = 1$
 $f(1) = 1$ according to guessed, which is the same as defined. So it holds in base case.
 - Inductive step:
 Assume $f(k) = \sum_{i=1}^k i$, $k \geq 1$. Want to show $f(k+1) = \sum_{i=1}^{k+1} i$ by using the recurrence relation (only).

$$f(k+1) = (k+1) + f(k) = (k+1) + \sum_{i=1}^k i = \sum_{i=1}^{k+1} i. \text{ Done!}$$
 - So, for all $n \geq 1$, $f(n) = \sum_{i=1}^n i$.
- So,

$$f(n) = \sum_{i=1}^n i = \frac{n(n+1)}{2}, n \geq 1.$$

Another math induction (exercise).

- $\frac{n(n+1)}{2}$ is the *closed form* for the recurrence.
- You NEED to get the closed forms, as simple as possible!!!

Recurrence relations — merge sort analysis

- Merge sort recall:
 - Divide the whole list into 2 sublists of equal size;
 - Recursively merge sort the 2 sublists;
 - Combine the 2 sorted sublists into a sorted list.
- Assumptions:
 - n — number of keys in the whole list — a power of 2
 - $T(n)$ — WC running time
 - Operations under consideration: KC
- Deriving recurrence relation:
 - Merge sort on 2 sublists $2 \times T(\frac{n}{2})$
 - Assembling needs $n - 1$ KC (in the WC)
 - $T(n) = (n - 1) + 2 \times T(\frac{n}{2})$
 - Base case: $T(1) = 0$.
- Solving recurrence relation:

Merge sort analysis — solving the recurrence relation

- Particular case:

$$T(1) = 0,$$

$$T(2) = 1,$$

...

- General case:

$$\begin{aligned} T(n) &= (n - 1) + 2 \times T\left(\frac{n}{2}\right) \\ &= (n - 1) + 2 \times \left(\left(\frac{n}{2} - 1\right) + 2 \times T\left(\frac{n}{4}\right)\right) \\ &= \dots \end{aligned}$$

$$\begin{aligned} &T(2^k) \\ &= (2^k - 1) + 2 \times T(2^{k-1}) \\ &= (2^k - 1) + 2 \times ((2^{k-1} - 1) + 2 \times T(2^{k-2})) \\ &= (2^k - 1) + (2^k - 2) + 2^2 \times T(2^{k-2}) \\ &= (2^k - 1) + (2^k - 2) + 2^2 \times ((2^{k-2} - 1) + 2 \times T(2^{k-3})) \\ &= (2^k - 1) + (2^k - 2) + (2^k - 2^2) + 2^3 \times T(2^{k-3}) \\ &= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + 2^3 \times T(2^{k-3}) \\ &= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + (2^k - 2^3) + 2^4 \times T(2^{k-4}) \\ &= \dots \\ &= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + \dots + (2^k - 2^{k-1}) + 2^k \times T(2^{k-k}) \\ &= (2^k - 2^0) + (2^k - 2^1) + (2^k - 2^2) + \dots + (2^k - 2^{k-1}) \\ &= k \times 2^k - \sum_{i=0}^{k-1} 2^i \\ &= (k - 1)2^k + 1 \end{aligned}$$

Since $n = 2^k$, we have $k = \lg n$. So, $T(n) = n(\lg n - 1) + 1$.

- Notes:
 1. Variable substitution makes guessing easy ...
 2. Later on recurrence solving always assume n being some power, whenever necessary (ignore floor and ceiling).
 3. Prove by induction.
 4. Need to transform back to original variable.

Closed form proof by induction:

- Recurrence:

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ (n - 1) + 2 \times T(\frac{n}{2}) & \text{if } n \geq 2 \end{cases}$$

Guessed closed form:

$$T(n) = n(\lg n - 1) + 1, n \geq 1$$

- Assuming $n = 2^k, k \geq 0$

- Base case:

According to guessed, $T(1) = 0$.

Holds in base case.

- Inductive step:

Assuming that $T(2^k) = 2^k(k - 1) + 1, k \geq 0$, want to show $T(2^{k+1}) = 2^{k+1}k + 1$.

By recurrence relation,

$$\begin{aligned} T(2^{k+1}) &= (2^{k+1} - 1) + 2 \times T(2^k) \\ &= (2^{k+1} - 1) + 2^{k+1}(k - 1) + 2 \\ &= k2^{k+1} + 1. \end{aligned}$$

Done!

- Conclusion: merge sort WC running time is $\Theta(n \log n)$.
- What is the worst case? or, what are those instances on which mergesort performs exactly WC number of KC?
- Question: BC/AC running time, in terms of KC?

Conclusions

- Divide-and-conquer algorithm often recursive
- Analysis of recursive algorithm $\implies$ solving recurrence

An exercise:

- Examine the running time of $QZ(n)$ (uniform cost RAM)

```

Proc QZ(n)

  if n > 1 then
    a ← n × n + 37
    b ← a × QZ( $\frac{n}{2}$ )
    return QZ( $\frac{n}{2}$ ) × QZ( $\frac{n}{2}$ ) + n
  else
    return n × n
    
```

- $A(n)$ — during $QZ(n)$, number of additions
 - $M(n)$ — during $QZ(n)$, number of multiplications
 - $T(n) = A(n) + M(n)$
 - *Claim:* $QZ(n)$ running time $\in \Theta(T(n))$
 - $A(n) = ?$ $M(n) = ?$ $T(n) = ?$
 - Hint: $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3QZ(\frac{n}{2}) + 5 & \text{if } n \geq 2 \end{cases}$
- Solve $T(n)$!!!

Lecture 6: Recurrences

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	iterated substitution (IS) method
☐	☐	☐	closed form guessing
☐	☐	☐	proof by math induction
☐	☐	☐	recurrence deriving
☐	☐	☐	variable substitution
☐	☐	☐	solving recurrence by IS