EECS 477: Introduction to algorithms.

Lecture 8

Prof. Igor Guskov
guskov@eecs.umich.edu

October 1, 2002

Lecture outline

- Recurrencies: inhomogeneous case
- Master theorem
- Examples

Recursive Fibonacci

- ```
  Integer fib_rec(unsigned n) {
      if(n<2)
          return 1;
      else
          return fib_rec(n-1) + fib_rec(n-2);
  }
  ```
- $t(n) = c_1 n + t(n-1) + t(n-2)$: a linear recurrence
- $t(0) = t(1) = c_0$
  ```

Linear recurrencies

- General form: $a_0 t(n) + a_1 t(n-1) + \ldots + a_k t(n-k) = f(n)$ plus initial conditions on $t(0), \ldots, t(k-1)$
- a_k are constants
- Start with homogeneous case: $f(n) = 0$: solutions form linear space (can add and scale them)

Linear recurrencies: characteristic polynomial

- Consider solution of exponential kind $t(n) = x^n$, substitute into equation to get

$$a_0 x^n + a_1 x^{n-1} + \dots a_k x^{n-k} = 0$$

or

$$a_0 x^k + a_1 x^{k-1} + \dots a_k = 0$$

- Find roots of the above and assume they are different $r_1, \dots, r_k$. Then

$$t(n) = c_1 r_1^n + c_2 r_2^n + \dots + c_k r_k^n$$

is a general solution form, constants from initial conditions

- for a root r of multiplicity m we get m fundamental solutions

$$r^n,\ nr^n, \dots,\ n^{m-1} r^n$$

Linear recurrencies: inhomogeneity

- Inhomogeneous are important!

$$a_0 t(n) + a_1 t(n-1) + \ldots + a_k t(n-k) = b^n p(n),$$

restricted version where $p(n)$ is a polynomial of degree d.

- Solution involves forming the *implied homogeneous recurrence*:

$$(a_0 x^k + a_1 x^{k-1} + \ldots a_k)(x-b)^{d+1} = 0$$

- General solution is $t(n) = \sum_i \sum_{j=0}^{m_i - 1} c_{ij} n^j r_i^n$ then substitute into the original recurrence and initial condition

Linear recurrencies: general inhomogeneity

- More general inhomogeneity

  $$a_0t(n) + a_1t(n-1) + \ldots + a_kt(n-k) = b_1^n p_1(n) + b_2^n p_2(n) + \ldots,$$

  restricted version where $p_i(n)$ is a polynomial of degree d_i.

- Solution involves forming the *implied homogeneous recurrence*:

  $$(a_0x^k + a_1x^{k-1} + \ldots a_k)(x - b_1)^{d_1+1}(x - b_2)^{d_2+1} \ldots = 0$$

- General solution is $t(n) = \sum_i \sum_{j=0}^{m_i-1} c_{ij}n^j r_i^n$ then substitute into the original recurrence and initial condition

Merge sort

- !!
- ```
  void merge_sort(data* inlist, data* outlist, unsigned n) {
      data* temp = new data[n];
      unsigned half = n/2;
      merge_sort(inlist, temp, half);
      merge_sort(inlist+half, temp+half, n-half);
      merge_lists(temp, half, temp+half, n-half, outlist);
  }
  ```
- Recurrence relation?
- What if we split into d parts?
- Is it going to improve the performance?
  ```

Master theorem

- Divide and conquer tool
- $T(n) = aT(n/b) + f(n)$
- $a \geq 1$ and $b > 1$ are constants, $f(n)$ is eventually positive, can have $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$
- Three cases ():

  $f(n) = O(n^{log_b a - \epsilon})$ for some $\epsilon > 0$ then $T(n) = \Theta(n^{log_b a})$;

  $f(n) = \Theta(n^{log_b a})$ then $T(n) = \Theta(n^{log_b a} log n)$;

  $f(n) = \Omega(n^{log_b a + \epsilon})$ some $\epsilon > 0$ and $af(n/b) \leq cf(n)$ for some constant $c < 1$ and sufficiently large n then $T(n) = \Theta(f(n))$.

Master theorem: examples

- $T(n) = 9T(n/3) + n$
- $a = 9$, $b = 3$, $\log_b a = \log_3 9 = 2$
- $f(n) = O(n^{2-\epsilon})$ holds
- Then $T(n) = \Theta(n^2)$

Master theorem: merge sort

- $T(n) = dT(n/d) + n$
- $a = d$, $b = d$, $\log_b a = \log_d d = 1$
- $f(n) = \Theta(n)$ holds
- Then $T(n) = \Theta(n \log n)$

Master theorem: misc

- if instead of $T(n) = \ldots$ we have $T(n) \leq \ldots$ then we can only make $O(\ldots)$ claims
- Third case condition: $af(n/b) \leq cf(n) - f(N)$ should grow steadily, e.g. $f(n) = n^2(1 + n^2 sin^2(n))$ will not work.
- There are gaps in the theorem like in case 1 $f(n) = O(n^{log_b a})$ is not enough to conclude anything...
- Proof is optional

Master theorem: more examples

- $T(n) = T(2n/3) + 1$: case 2, $T(n) = \Theta(\log n)$
- $T(n) = 3T(n/4) + n \log n$: case 3, $T(n) = \Theta(n \log n)$
- $T(n) = 2T(n/2) + n \log n$: case 3 does not apply!!!

Change of variable

- Often helps
- $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$
- $n = 2^m$
- $S(m) = T(2^m)$ so that $S(m) = 2S(m/2) + m$
- now go back to n to obtain $T(n) = \log n \log\log n$

Change of range

- $T(1) = 1/3, T(n) = nT^2(n/2)$
- $n = 2^m$ leads to $S(m) = 2^m S^2(m-1)$
- change of range: $U(m) = \log S(m)$