8.1 Recurrence Relations

Definition:

A *recurrence relation* for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more previous terms of the sequence $a_0, a_1, \ldots, a_{n-1}$. A sequence is called a solution of a recurrence relation if its terms satisfy the recurrence relation.

The recurrence relation together with the initial conditions uniquely determines a sequence, i.e., a solution.

e.g. Consider the recurrence relation (Fibonacci numbers):

$a_n = a_{n-1} + a_{n-2}$, $n \geq 2$
where $a_0 = 0$ and $a_1 = 1$.

Then $\{0,1,1,2,3,5,8,\ldots\}$ is the solution.

Question: Is it possible to find an explicit formula for a_n?
In some conditions yes!

8.2 Solving Recurrence Relations

Definition:

A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ where $c_1, c_2, \ldots, c_k$ are real numbers and $c_k \neq 0$.

8.2 Solving Recurrence Relations

Definition:

A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ where $c_1, c_2, \ldots, c_k$ are real numbers and $c_k \neq 0$.

Basic approach is to look for a solution of the form $a_n = r^n$.

$\Rightarrow r^n = c_1 r^{n-1} + c_2 r^{n-2} + \cdots + c_k r^{n-k}$

8.2 Solving Recurrence Relations

Definition:

A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ where $c_1, c_2, \ldots, c_k$ are real numbers and $c_k \neq 0$.

Basic approach is to look for a solution of the form $a_n = r^n$.

$\Rightarrow r^n = c_1 r^{n-1} + c_2 r^{n-2} + \cdots + c_k r^{n-k}$

Divide both sides by r^{n-k}

$\Rightarrow r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_{k-1} r - c_k = 0$

8.2 Solving Recurrence Relations

Definition:

A linear homogeneous recurrence relation of degree k with constant coefficients is a recurrence relation of the form

$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$ where $c_1, c_2, \ldots, c_k$ are real numbers and $c_k \neq 0$.

Basic approach is to look for a solution of the form $a_n = r^n$.

$\Rightarrow r^n = c_1 r^{n-1} + c_2 r^{n-2} + \cdots + c_k r^{n-k}$

Divide both sides by r^{n-k}

$\Rightarrow r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_{k-1} r - c_k = 0$: Characteristic equation

$\Rightarrow$ If r is a solution of the *characteristic equation*, then $\{a_n\}$ with $a_n = r^n$, is a solution of the recurrence relation.

Theorem:

Consider the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$. Suppose that $r^2 - c_1 r - c_2$ has two distinct roots r_1 and r_2. Then the solution is given by $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, where α_1 and α_2 are real constants.

Proof: See page 462 in the textbook 6th edition (page 499 in the textbook 7th edition).

e.g. Solve $f_n = f_{n\text{-}1} + f_{n\text{-}2}$, where $f_0 = 0$ and $f_1 = 1$, i.e., find an explicit formula for Fibonacci numbers.

The roots of the characteristic equation $r^2 - r - 1 = 0$ are

$r_1 = (1+\sqrt{5})/2$ and $r_2 = (1-\sqrt{5})/2$.

e.g. Solve $f_n = f_{n-1} + f_{n-2}$, where $f_0 = 0$ and $f_1 = 1$, i.e., find an explicit formula for Fibonacci numbers.

The roots of the characteristic equation $r^2 - r - 1 = 0$ are

$$r_1 = (1+\sqrt{5})/2 \text{ and } r_2 = (1-\sqrt{5})/2.$$

By the theorem, $f_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for some α_1 and α_2.

e.g. Solve $f_n = f_{n-1} + f_{n-2}$, where $f_0 = 0$ and $f_1 = 1$, i.e., find an explicit formula for Fibonacci numbers.

The roots of the characteristic equation $r^2 - r - 1 = 0$ are

$r_1 = (1+\sqrt{5})/2$ and $r_2 = (1-\sqrt{5})/2$.

By the theorem, $f_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for some α_1 and α_2.

Using the initial conditions $f_0 = 0$ and $f_1 = 1$,

$f_0 = \alpha_1 + \alpha_2 = 0$

$f_1 = \alpha_1 (1+\sqrt{5})/2 + \alpha_2 (1-\sqrt{5})/2 = 1$

$\Rightarrow \alpha_1 = 1/\sqrt{5}$ and $\alpha_2 = -1/\sqrt{5}$

$$\Rightarrow \quad f_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

Theorem:

Consider the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$. Suppose that $r^2 - c_1 r - c_2$ has only one root r_0 with multiplicity 2. Then the solution is given by $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$, where α_1 and α_2 are real constants.

Theorem:

Consider the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$. Suppose that $r^2 - c_1 r - c_2$ has only one root r_0 with multiplicity 2. Then the solution is given by $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$, where α_1 and α_2 are real constants.

e.g. Solve the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$ with $a_0 = 1$ and $a_1 = 6$.

Characteristic equation: $r^2 - 6r + 9 = 0$**,** which is $(r-3)^2 = 0 \Rightarrow r_0 = 3$

Solution is then given by $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$

$a_0 = \alpha_1 r_0^0 + \alpha_2 0 r_0^0 = \alpha_1 3^0 = \alpha_1 = 1$

$a_1 = \alpha_1 r_0^1 + \alpha_2 1 r_0^1 = \alpha_1 3^1 + \alpha_2 1 3^1 = 3\alpha_1 + 3\alpha_2 = 6$

$\Rightarrow \alpha_2 = 1$

$a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$

$a_n = 3^n + n 3^n$

8.4 Generating Functions

Using Generating Functions to solve recurrence relations

Example: Solve $a_n = 8a_{n-1} + 10^{n-1}$ with initial condition $a_0 = 1$.
(a_n corresponds to the number of valid code words of length n, supposing that a valid code word is an n-digit number in decimal notation containing an even number of 0s.)

Can't use characteristic equation in this case because the recurrence relation is not linear.

8.4 Generating Functions

Using Generating Functions to solve recurrence relations

Example: Solve $a_n = 8a_{n-1} + 10^{n-1}$ with initial condition $a_0 = 1$.
(a_n corresponds to the number of valid code words of length n, supposing that a valid code word is an n-digit number in decimal notation containing an even number of 0s.)

Can't use characteristic equation in this case because the recurrence relation is not linear.
One possible way is to use generating functions:

Definition:
The generating function for the sequence $a_0, a_1, \ldots, a_k, \ldots$ of real numbers is the infinite power series

$$G(x) = a_0 + a_1 x + \ldots + a_k x^k + \ldots = \sum_{k=0}^{\infty} a_k x^k$$

<u>Example</u>: Solve $a_n = 8a_{n-1} + 10^{n-1}$ with initial condition $a_0 = 1$.
(a_n corresponds to the number of valid code words of length n, supposing that a valid code word is an n-digit number in decimal notation containing an even number of 0s.)

<u>Solution:</u>

Let $G(x)$ be the generating function for the sequence $\{ a_n \}$, that is, $G(x) = \sum_{n=0}^{\infty} a_n x^n$

<u>Example</u>: Solve $a_n = 8a_{n-1} + 10^{n-1}$ with initial condition $a_0 = 1$.
(a_n corresponds to the number of valid code words of length *n*, supposing that a valid code word is an *n*-digit number in decimal notation containing an even number of 0s.)

<u>Solution:</u>

Let $G(x)$ be the generating function for the sequence $\{ a_n \}$, that is, $G(x) = \sum_{n=0}^{\infty} a_n x^n$

$$\Rightarrow \quad G(x) = 1 + \sum_{n=1}^{\infty} a_n x^n = 1 + \sum_{n=1}^{\infty} (8a_{n-1}x^n + 10^{n-1}x^n) = 1 + 8x\sum_{n=1}^{\infty} a_{n-1}x^{n-1} + x\sum_{n=1}^{\infty} 10^{n-1}x^{n-1}$$

Example: Solve $a_n = 8a_{n-1} + 10^{n-1}$ with initial condition $a_0 = 1$.
(a_n corresponds to the number of valid code words of length n, supposing that a valid code word is an n-digit number in decimal notation containing an even number of 0s.)

Solution:

Let $G(x)$ be the generating function for the sequence $\{a_n\}$, that is, $G(x) = \sum_{n=0}^{\infty} a_n x^n$

$$\Rightarrow \quad G(x) = 1 + \sum_{n=1}^{\infty} a_n x^n = 1 + \sum_{n=1}^{\infty} (8a_{n-1}x^n + 10^{n-1}x^n) = 1 + 8x\sum_{n=1}^{\infty} a_{n-1}x^{n-1} + x\sum_{n=1}^{\infty} 10^{n-1}x^{n-1}$$

$$= 1 + 8x\sum_{n=0}^{\infty} a_n x^n + x\sum_{n=0}^{\infty} 10^n x^n = 1 + 8x \cdot G(x) + x/(1-10x)$$

since by geometric series: $1/(1-rx) = \sum_{k=0}^{\infty} r^k x^k$

$$\Rightarrow \quad G(x) = \frac{1-9x}{(1-8x)(1-10x)} = \frac{1}{2}\left(\frac{1}{1-8x}\right) + \frac{1}{2}\left(\frac{1}{1-10x}\right)$$

$$= \frac{1}{2}\left(\sum_{n=0}^{\infty} 8^n x^n + \sum_{n=0}^{\infty} 10^n x^n\right) = \sum_{n=0}^{\infty} \frac{1}{2}(8^n + 10^n)x^n$$

$$\Rightarrow \quad a_n = \frac{1}{2}(8^n + 10^n)$$

<u>Example</u>: Solve $a_k = 3a_{k-1}$ with initial condition $a_0 = 2$.
(You could solve this recurrence relation by using its characteristic equation, but we'll try using generating functions.)

Example: Solve $a_k = 3a_{k-1}$ with initial condition $a_0 = 2$.
(You could solve this recurrence relation by using its characteristic equation, but we'll try using generating functions.)

Let $G(x)$ be the generating function for the sequence $\{a_k\}$, that is, $G(x) = \sum_{k=0}^{\infty} a_k x^k$.

$G(x) = \sum_{k=0}^{\infty} a_k x^k = 2 + \sum_{k=1}^{\infty} 3a_{k-1} x^k$ (by the recurrence relation $a_k = 3a_{k-1}$)

$$= 2 + 3x \sum_{k=1}^{\infty} a_{k-1} x^{k-1} = 2 + 3x \cdot G(x)$$

$$\Rightarrow \quad G(x) - 3x \cdot G(x) = (1-3x) \cdot G(x) = 2$$
$$\Rightarrow \quad G(x) = 2/(1-3x)$$

Using the identity, $1/(1-rx) = \sum_{k=0}^{\infty} r^k x^k$,

$$G(x) = 2 \cdot \sum_{k=0}^{\infty} 3^k x^k = \sum_{k=0}^{\infty} 2 \cdot 3^k x^k \quad \therefore \quad a_k = 2 \cdot 3^k$$

8.3 Divide-and-Conquer Recurrence Relations

Many algorithms divide a problem into one or more smaller problems.

Recall binary search: (from Chapter 3)
Search x in the list $a_0, a_1, \ldots, a_{n-1}$ where $a_0 < a_1 < \ldots < a_{n-1}$.

1. Compare x with the middle term of the sequence, a_m, where $m = \lfloor (n-1) / 2 \rfloor$.
2. **If** $x > a_m$, search x on the second half $\{a_{m+1}, a_{m+2}, \ldots a_n\}$
   else search x on the first half $\{a_1, a_2, \ldots a_m\}$
3. **Repeat** the first two steps until a list with one single term is obtained.
4. Determine whether this one term is x or not.

Reduces the search in a sequence of size n to a search in a sequence of size $n/2$, assuming n is even.

8.3 Divide-and-Conquer Recurrence Relations

Many algorithms divide a problem into one or more smaller problems.

Recall binary search: (from Chapter 3)
Search x in the list $a_0, a_1, \ldots, a_{n-1}$ where $a_0 < a_1 < \ldots < a_{n-1}$.

1. Compare x with the middle term of the sequence, a_m, where $m = \lfloor (n-1) / 2 \rfloor$.
2. **If** $x > a_m$, search x on the second half $\{a_{m+1}, a_{m+2}, \ldots a_n\}$
   else search x on the first half $\{a_1, a_2, \ldots a_m\}$
3. **Repeat** the first two steps until a list with one single term is obtained.
4. Determine whether this one term is x or not.

Reduces the search in a sequence of size n to a search in a sequence of size $n/2$, assuming n is even.

Two comparisons are needed at each iteration (one to determine which half of the list to use and the other to determine whether any terms of the list remain).
$\Rightarrow f(n) = f(n/2) + 2, \quad f(1) = 2$

$f(n)$: # of operations (comparisons) for a search sequence of size n.

Definition:
The recurrence relation

$$f(n) = a \cdot f(n/b) + g(n)$$

is called a *divide-and-conquer* recurrence relation.

Sometimes we aren’t really interested in solving a given recurrence relation, but we rather want to find the complexity of the function.

e.g. The problem of finding the maximum element of a sequence, $a_1, a_2, ..., a_n$, can be solved using a divide-and-conquer algorithm:

- If $n = 1$ then a_1 is the maximum.
- If $n > 1$, split the sequence into two sequences. The overall maximum is maximum of the maximum elements of these two sub-sequences. The problem is hence reduced to finding the maximum of each of the two smaller sequences.

e.g. The problem of finding the maximum element of a sequence, $a_1, a_2, \ldots, a_n$, can be solved using a divide-and-conquer algorithm:

- If $n = 1$ then a_1 is the maximum.
- If $n > 1$, split the sequence into two sequences. The overall maximum is maximum of the maximum elements of these two sub-sequences. The problem is hence reduced to finding the maximum of each of the two smaller sequences.

```
int MAX(int a[], int i, int j) {

if (i == j)
    p = a[i];
else{
    max1 = MAX(a, i,(i+j)/2));
    max2 = MAX(a,((i+j)/2)+1,j);
    if (max1 > max2) p = max1;
    else p = max2;
    }

return p;
}
// initially i=0,j=n-1
```

Complexity analysis:

Let $f(n)$ be the number of comparisons to find the maximum of the sequence with n elements. Using two comparisons (one to compare the current maxima, and one to determine whether any terms of the list remain at each iteration),

$$f(n) = 2f(n/2) + 2, \quad f(1) = 1, \text{ where } n \text{ is even.}$$

```
int MAX(int a[], int i, int j) {

if (i == j)
    p = a[i];
else{
    max1 = MAX(a, i,(i+j)/2));
    max2 = MAX(a,((i+j)/2)+1,j);
    if (max1 > max2) p = max1;
     else p = max2;
     }

return p;
}
// initially i=0,j=n-1
```

<u>Theorem</u>: Let f be an increasing function that satisfies the recurrence relation:

$$f(n) = a \cdot f(n/b) + c$$

where n is divisible by b, $a \geq 1$, b is an integer greater than 1 and c is a positive real number. Then

$$f(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > 1 \\ O(\log n) & \text{if } a = 1 \end{cases}$$

<u>Theorem</u>: Let f be an increasing function that satisfies the recurrence relation:

$$f(n) = a \cdot f(n/b) + c$$

where n is divisible by b, $a \geq 1$, b is an integer greater than 1 and c is a positive real number. Then

$$f(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > 1 \\ O(\log n) & \text{if } a = 1 \end{cases}$$

e.g. **Finding the maximum** with $f(n) = 2 \cdot f(n/2) + 2$, n is even

By the theorem $f(n)$ is $O(n^{\log_2 2}) = O(n)$.

Theorem: Let f be an increasing function that satisfies the recurrence relation:

$$f(n) = a \cdot f(n/b) + c$$

where n is divisible by b, $a \geq 1$, b is an integer greater than 1 and c is a positive real number. Then

$$f(n) = \begin{cases} O(n^{\log_b a}) & \text{if } a > 1 \\ O(\log n) & \text{if } a = 1 \end{cases}$$

e.g. **Finding the maximum** with $f(n) = 2 \cdot f(n/2) + 2$, n is even

By the theorem $f(n)$ is $O(n^{\log_2 2}) = O(n)$.

e.g. **Binary search** with $f(n) = f(n/2) + 2$, n is even

From the theorem, $f(n)$ is $O(\log n)$.

Proof: Let $n = b^k$

$$\Rightarrow f(n) = a \cdot f(n/b) + c$$

$$= a^2 \cdot f(n/b^2) + ac + c$$

$$\vdots$$

$$= a^k \cdot f(n/b^k) + \sum_{j=0}^{k-1} a^j c$$

$$\Rightarrow f(n) = a^k \cdot f(1) + c\sum_{j=0}^{k-1} a^j$$

Case i:

Let a =1, then $f(n) = f(1) + ck = f(1) + c \cdot \log_b n$ $\quad \therefore \quad$ $f(n)$ is $O(\log n)$

When n is not a power of b, we have $b^k < n < b^{k+1}$ for some k.
Since f is an increasing function,

$$f(n) \le f(b^{k+1}) = f(1) + c(k+1) = f(1) + c + ck$$

$$\le f(1) + c + c \cdot \log_b n \qquad \therefore f(n) \text{ is } O(\log n)$$ in both cases.

Case ii:

Let $a > 1$ and $n = b^k$

$f(n) = a^k \cdot f(1) + c \cdot (a^k - 1)/(a-1)$ (from geometric series, see Section 2.4)

$= a^k \left[f(1) + c/(a-1) \right] - c/(a-1)$

$= C_1 \cdot n^{\log_b a} + C_2$ (Note that in general $n^{\log a} = a^{\log n}$)

where $C_1 = f(1) + c/(a-1)$ and $C_2 = -c/(a-1)$.

$\therefore$ $f(n)$ is $O(n^{\log_b a})$

Suppose that $n \neq b^k$, then $b^k < n < b^{k+1}$

$\Rightarrow$ $f(n) \le f(b^{k+1}) = C_1 a^{k+1} + C_2$

$\le (C_1 a) a^{\log_b n} + C_2$ since $k \le \log_b n < k+1$

$\le (C_1 a) n^{\log_b a} + C_2$

Hence *f(n)* is $O(n^{\log_b a})$.

Master Theorem: Let $f(n)$ be an increasing function that satisfies

$$f(n) = a \cdot f(n/b) + cn^d$$

where $n = b^k$, k is a positive integer, $a \geq 1$, b is an integer greater than 1, c and d are positive real numbers. Then

$$f(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

See the textbook for the proof.

Master Theorem: Let $f(n)$ be an increasing function that satisfies

$$f(n) = a \cdot f(n/b) + cn^d$$

where $n = b^k$, k is a positive integer, $a \geq 1$, b is an integer greater than 1, c and d are positive real numbers. Then

$$f(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n) & \text{if } a = b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

See the textbook for the proof.

e.g., **Merge Sort**

$f(n) = 2f(n/2) + n$, which is $O(n \log n)$.