Recorrências

Prof. João B. Oliveira

Recursividade e modelagem

O estudo de soluções recursivas é uma das áreas mais complicadas para o novato na informática. Em geral ele aprende exemplos simples e depois usa recursividade para percorrer estruturas de dados, mudando o mínimo que puder os algoritmos que foram apresentados em aula para não complicar o que já é confuso. Infelizmente, nesta trajetória ignora-se completamente o poder de usar a recursão para **modelar** problemas que podem ser muito complicados sem ela.

Este texto tem o objetivo de explorar o uso de recursividade na modelagem de problemas, mostrando que tipos de problemas se prestam para soluções recursivas e como são suas implementações. No decorrer do texto as palavras recursão, recursividade e recorrência serão usadas com o mesmo sentido, ou seja, formas de tentar resolver um problema transformando-o em problemas menores.

Os exemplos de sempre

Se você estava acordado na disciplina de programação básica, deve ter visto os dois exemplos de recursividade que são clássicos e dos quais ninguém pode escapar: fatorial e os números de Fibonacci. Mesmo que sejam completamente sem graça, ainda podemos aprender algo com eles. Vamos examiná-los.

Fatorial

O número $n!$ é conhecido como "n fatorial" ou "fatorial de n" e pode ser definido matematicamente como

$$n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n > 0 \end{cases}$$

e esta maneira de definir o fatorial pode ser traduzida imediatamente para uma linguagem de programação, tornando-se algo parecido com isto:

```
define f(n) {
  if ( n == 0 ) return 1;
  return n * f(n-1);
}
```

O código ao lado foi escrito para o programa `bc`, que pode ser usado em um sistema GNU/Linux. O `bc` é uma calculadora bastante interessante (e peculiar). Tente.

Usando o mesmo `bc` podemos calcular os primeiros valores para $n!$:

```
for(i=0; i<=10; i++) f(i);
1
1
2
6
24
120
720
5040
40320
362880
3628800
```

É bonito ver como a tradução da definição para o código é imediata, mas os mesmos resultados são obtidos pelo algoritmo abaixo, que muitos considerariam mais simples:

```
res = 1;
i = 0;
while (i <= 10) {
  res;
  i = i + 1;
  res = res * i;
}
```

Este algoritmo também pode ser usado no mesmo `bc`.

É nessa hora que tudo se complica. Precisamos nos convencer de que o código alternativo que usa apenas um `while` sem recursão **não** é mais simples do que o código recursivo. Veja que a versão recursiva é a tradução direta da definição, enquanto a segunda versão teve que ser trabalhada para não usar recursividade. Também temos uma lição importante, que não vai ser demonstrada neste texto: qualquer algoritmo recursivo pode ser transformado em não-recursivo e vice-versa, e a prática ensina que muitas vezes um deles é bem mais simples do que o outro.

Fibonacci

Os números de Fibonacci são representados geralmente por F_n e a sua definição matemática também usa recursão:

$$F_n = \begin{cases} 1, & n = 0 \\ 1, & n = 1 \\ F_{n-1} + F_{n-2}, & n > 1 \end{cases}$$

Desta vez, são necessárias duas condições iniciais para fazer com que a recursão termine. Isto é preciso por que a expressão geral (a última e mais importante da definição) precisa do valor $F_{n-2,}$ que está duas posições à frente de F_n na sequencia de Fibonacci. Esta definição também se traduz imediatamente em código:

```
define f(n) {
  if ( n == 0 ) return 1;
  if ( n == 1 ) return 1;
  return f(n-1) + f(n-2);
}
```

E é claro que podemos calcular facilmente os primeiros números de Fibonacci:

```
for(i=0; i<=10; i++) f(i);
1
1
2
3
5
8
13
21
34
55
89
```

Como no fatorial, os mesmos resultados são obtidos pelo algoritmo não recursivo abaixo:

```
f1 = f2 = 1;
f1; f2;
i = 2;
while (i <= 10) {
  f3 = f2 + f1;
  f3;
  i = i + 1;
  f1 = f2;
  f2 = f3;
}
```

De brinde, você já pode ficar sabendo que mais adiante você aprenderá uma técnica que pode encontrar uma fórmula fechada para os números de Fibonacci, ou seja, uma fórmula que usa n como variável e retorna F_n.

O que aprendemos

Até agora temos dois exemplos de números que são definidos de forma recursiva e podem ser programados facilmente com a recursão e também poderiam ser programados sem ela. Ou seja, não temos motivos muito convincentes para acreditar que recursão seja algo mais do que uma técnica bonitinha que se usa para confundir os alunos. É verdade que quando você estudar árvores binárias ou grafos poderá usar recursão para tarefas muito mais interessantes, mas neste momento não queremos chegar a tanto. Queremos nos convencer de que recursão realmente serve para fazer algo que não conseguimos fazer de outra maneira.

A senha secreta

Imagine que sua vó perdeu a senha do banco e pediu a sua ajuda. Ela sempre usa senhas com as letras A, B, C e D, mas nunca coloca dois C juntos por que acha que isso dá azar. Ela também lembra que a senha tem 12 letras, e é claro que não lembra quais eram. Você resolveu fazer um programinha que teste todas as senhas e gostaria de saber quantas são para prever quanto tempo vai levar. Vamos começar organizando o problema:

- É bom dar nome ao que estamos procurando. Por exemplo, S_n pode representar o número de senhas com n letras. Neste caso, queremos saber o valor de S_{12}.
- Sabe-se algo sobre outros valores de S_n? Quando trabalhamos com recorrências é sempre bom começar com casos simples e ir acostumando com o problema através deles. Por exemplo, sabemos que se a senha tivesse apenas uma letra, teríamos 4 possibilidades: A, B, C, D. Por isso temos $S_1 = 4$.
- Se a senha tem duas letras, temos $4 \times 4 - 1$ possibilidades, que são todos os pares de letras excetuando o par CC que é inválido. Ou seja, sabemos que $S_2 = 15$.
- Um conhecimento extra: se não houvesse a proibição do caso CC, teríamos sempre 4 opções para cada letra da senha e neste caso a resposta seria imediata: $S_{12} = 4^{12}$. Com a restrição dos CC este número tem que diminuir, portanto sabemos que $S_n \leq 4^n$. A igualdade só acontece com $n = 1$, pois com n maior começam a aparecer casos CC que devem ser retirados.
- Podemos procurar S_3 manualmente mas neste caso não estaremos fazendo nada muito inteligente, apenas trabalho braçal. Em algum ponto devemos procurar uma regra geral que fará o papel de regra da recursão. É claro que esta geralmente é a parte difícil.

A regra geral de uma recursão costuma ser a parte mais complicada de obter e o erro mais comum que um iniciante pode cometer é ser hipnotizado pelos números. Por exemplo, ele pode escrever um programinha[1] para obter os primeiros valores de S_n para dar uma olhada e terá os valores abaixo:

$$\begin{aligned} S_1 &= 4 \\ S_2 &= 15 \\ S_3 &= 57 \\ S_4 &= 216 \end{aligned}$$

Depois disso, é fácil ficar horas imaginando como esses números podem se combinar para produzir os números seguintes e tentar as formas mais estranhas para ver se funcionam. Isso não tem muita chance de dar certo se o fenômeno que estamos estudando for complicado e mesmo que por um acaso feliz achemos a regra certa não saberemos **por que** ela funciona. Ou seja, não saberemos como ela expressa o fenômeno que queremos entender, só saberemos que ela parece funcionar para os poucos números que temos. Não é uma grande certeza.

Vamos parar de pensar nos números e pensar no problema: se queremos usar recursão precisamos obter um mecanismo para transformar um problema de um certo tamanho n em problema(s) de tamanho menor. Traduzindo, como podemos transformar um problema que fala de uma senha de tamanho n em um problema que trata de senhas menores?

Uma conclusão fácil é que uma senha de n letras é a combinação de uma senha de $n - 1$ letras com mais uma letra, e existem 4 opções para essa última letra, o que quadruplica o número de senhas. Se for assim, temos uma recorrência para S_n:

[1] Pergunta: como você faria este programa?

$$S_n = \begin{cases} 4, & n = 1 \\ 4 * S_{n-1}, & n > 1 \end{cases}$$

Deve ser claro que esta recorrência está errada, pois em nenhum momento ficamos impedidos de ter as letras CC. (Você também pode pensar um pouco mais e se convencer de que a solução para esta recorrência é $S_n = 4^n$, o que já sabemos que é mais do que desejamos). Então, como devemos evitar o caso de senhas que incluem CC?

O problema é que não podemos simplesmente adicionar um C no final de uma senha com $n-1$ letras por que não sabemos qual a última letra da senha, e se ela já for um C teremos problemas. Por isso temos que encarar duas alternativas:

1. Ter uma forma de saber qual era a última letra da senha com $n-1$ letras e agir de acordo: se não era C então podemos adicionar 4 letras, se era C só podemos adicionar 3 letras;

2. Não saber qual era a última letra, mas garantir (de algum jeito) que a colocação de novas letras não crie o par CC, independentemente do que havia antes.

As duas formas de pensar podem levar a recursões que funcionam e em geral as pessoas tentam a primeira alternativa, ou seja, saber qual era a última letra e agir de acordo. Neste caso, ajuda pensar que "carregamos" junto um booleano que informa se a última letra foi C, e então estaremos calculando $S_{n,b}$ onde b é o booleano. Vamos juntar alguns fatos:

- Sabemos que $S_{1,b} = 4$ independentemente do valor de b por que não havia letra anterior.

- Sabemos que se $b = True$ então a última letra adicionada foi C e não podemos adicionar outro C mas temos outras três opções de letras. Então, precisamos de uma regra parecida com
$$S_{n,b} = 3 * S_{n-1,b}$$

- Sabemos que se $b = False$ podemos adicionar qualquer letra sem medo de criar CC.
$$S_{n,b} = 4 * S_{n-1,b}$$

- Pergunta: antes desejávamos calcular S_n. Agora que trabalhamos com $S_{n,b}$ desejamos calcular $S_{n,True}$, ou $S_{n,False}$ ou o quê? Pense a respeito.

Tentando juntar as peças, podemos tentar algo assim:

$$S_{n,b} = \begin{cases} 4, & n = 1 \\ 3 * S_{n-1,b}, & n > 1,\ b = True \\ 4 * S_{n-1,b}, & n > 1,\ b = False \end{cases}$$

Infelizmente há algo errado com esta maneira de pensar, pois ao tentarmos calcular o que acontece com apenas 2 letras temos $S_{2,True} = 12$ e $S_{2,False} = 16$ quando já sabemos que a resposta é 15. Os dois casos são reveladores se pensarmos no que eles realmente significam: $S_{2,True} = 12$ deveria ser o número de senhas de duas letras quando a primeira letra foi C. Esse resultado não pode ser 12 por que se a primeira letra é C, se a senha tem 2 letras e se o par CC é impossível, então apenas 3 senhas são possíveis (CA, CB e CD). O certo seria $S_{2,True} = 3$. O resultado $S_{2,False} = 16$ sofre do mesmo problema, pois se a primeira senha não inicia com C só pode iniciar com A, B ou D, e depois segue-se outra letra qualquer, o que dá 12 senhas no total. Parece que regra correta para $S_{2,False}$ deveria ser a regra usada para $S_{2,True}$, e a regra verdadeira para $S_{2,True}$ ainda é desconhecida. Podemos procurar uma nova regra para esta situação, mas existe uma alternativa mais interessante: analisar o caso "óbvio" em que $n = 1$. As duas regras que criamos terminam nesse caso, e talvez fosse melhor separar em duas situações, mesmo que

sejam pouco intuitivas. Por exemplo, podemos ter uma situação que informa quantas senhas de uma letra terminam com C: $S_{1,True} = 1$ e outra informando quantas senhas de uma letra não terminam com C: $S_{1,False} = 3$. Se tentarmos isso, teremos

$$S_{n,b} = \begin{cases} 1, & n = 1,\ b = True \\ 3, & n = 1,\ b = False \\ 3 * S_{n-1,b}, & n > 1,\ b = True \\ 4 * S_{n-1,b}, & n > 1,\ b = False \end{cases}$$

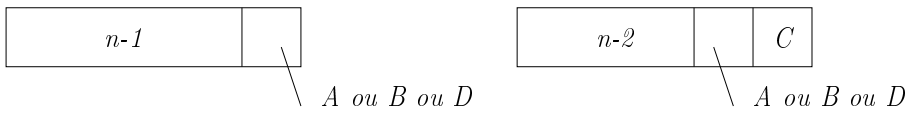
Fazendo isso teremos uma separação completa entre os cálculos de acordo com o valor do booleano, e finalmente teremos $S_{2,True} = 3$ e $S_{2,False} = 12$. Infelizmente a dedução ficou mais confusa e não é fácil convencer-se de que o raciocínio está certo (será que está mesmo? Ele é convincente? Passa em mais testes? Temos uma prova?). A dificuldade é que estamos tentando reduzir o problema de n letras para $n-1$, mas ao mesmo tempo precisamos da informação de qual foi a última letra colocada na senha de tamanho $n-1$ e isso não é tão simples de conseguir. Seja como for, a adição de um booleano deixou a situação mais complicada. Talvez seja a hora de jogar tudo isso para baixo do tapete e tentar a outra alternativa: garantir que o par CC nunca seja obtido quando se adiciona outra letra.

Essa alternativa é bem mais simples de desenvolver. Imagine uma senha válida e que tem $n-1$ letras e pense nas duas situações:

1. Se adicionarmos A, B ou D teremos uma nova senha de tamanho n sem problemas.

2. Se adicionarmos C, precisamos saber que a senha antiga não terminava com C, e esse foi o problema da abordagem anterior, que agora precisamos evitar. Como podemos garantir que a adição de um C não crie um CC? Bom, um jeito é colocar uma letra **extra** antes do C, que o protege e não deixa que ele encoste no que havia antes. Só que neste caso estamos adicionando **duas** letras, e no final temos uma senha de tamanho $n+1$, não tamanho n como desejávamos. Isso pode ser resolvido se colocarmos esss duas letras em uma senha de tamanho $n-2$ e não $n-1$.

A figura abaixo ilustra as situações possíveis para criar senhas de tamanho n:



Pensando assim percebemos que existem seis formas de criar uma senha com tamanho n: colocar uma de três letras (A, B, D) em uma senha de tamanho $n-1$ ou colocar duas letras (AC, BC ou DC) em uma senha de tamanho $n-2$. Agora precisamos ter duas certezas:

1. A certeza de que nenhuma senha possível foi esquecida, o que produziria um resultado menor do que o verdadeiro.

2. A certeza de que nenhuma senha foi contabilizada duas vezes, o que daria um resultado maior.

Temos a primeira certeza por que a nova senha de tamanho n tem todas as letras finais possíveis: A, B, C, D e portanto nenhuma foi esquecida. E como os finais são colocados em senhas menores que eram diferentes, podemos ter certeza de que não há repetições. Então podemos nos convencer de que se o número inicial da recursão está correto ($S_1 = 4$) os números seguintes estarão certos. A nova recursão é dada por

$$S_n = \begin{cases} 4, & n = 1 \\ 3 * S_{n-1} + 3 * S_{n-2}, & n > 1 \end{cases}$$

Ela parece bem convincente até percebermos que nada pode ser calculado: ao tentar calcular S_2 por exemplo a recursão pede S_1 e S_0, que não temos. Está faltando mais uma condição inicial na recursão e duas formas simples de adicionar condições extras estão abaixo:

$$S_n = \begin{cases} 4, & n = 1 \\ 15, & n = 2 \\ 3 * S_{n-1} + 3 * S_{n-2}, & n > 2 \end{cases}$$

Neste caso, adicionamos o caso $n = 2$ que já sabíamos. A regra geral também foi alterada para ser válida para valores maiores do que 2.

Qualquer uma das duas recursões pode ser programada facilmente, por exemplo assim:

```
define s(n) {
  if ( n == 0 ) return 1;
  if ( n == 1 ) return 4;
  return 3 * s(n-1)+ 3 * s(n-2);
}
```

Podemos calcular valores, como fizemos ao lado. O valor de S_{12} que desejávamos é 9221121.

$$S_n = \begin{cases} 1, & n = 0 \\ 4, & n = 1 \\ 3 * S_{n-1} + 3 * S_{n-2}, & n > 1 \end{cases}$$

Aqui adicionamos o caso $n = 0$. Por que? Por que muitas vezes é um caso simples de obter. Pense um pouco e entenda por que o valor dado para $n = 0$ é 1.

```
for(i=0; i<=12; i++) s(i);
1
4
15
57
216
819
3105
11772
44631
169209
641520
2432187
9221121
```

A escada

Imagine que existe uma escada de n degraus e você pode subir estes degraus de um em um, dois em dois ou três em três. A pergunta que queremos responder é simples: de quantas formas podemos subir a escada? Por exemplo, uma escada de 3 degraus pode ser subida de várias maneiras: 1+1+1 degraus, 1+2 degraus, 2+1 ou 3 degraus, completando 4 maneiras no total. Para dar um nome ao que estamos procurando, vamos chamar o número de maneiras de subir n degraus de M_n. O que sabemos sobre os números representados por M_n?

- Sabemos que $M_1 = 1$, pois só há uma forma de subir uma escada de 1 degrau.
- Sabemos que $M_2 = 2$, pois há duas formas de subir uma escada de 2 degraus: 1+1 e 2 degraus.
- Já sabemos que $M_3 = 4$.

Neste momento começamos a acreditar que os números 1, 2 e 4 não são concidência. Talvez seja verdade que $M_n = 2^{n-1}$. Infelizmente um pouco de exploração confirma que $M_4 = 7$ em vez de 8 e nossa ideia não deu certo. Mesmo assim, podemos guardar essa coleção de casos iniciais e tentar pensar em uma regra geral cujo objetivo é reduzir um problema de tamanho n para problemas de tamanhos menores.

Uma inspiração surge quando percebemos que subir uma escada de n degraus tem sempre um último passo. Este passo pode ser de 1, 2 ou 3 degraus, e isso reduz o tamanho da escada anterior para $n-1$, $n-2$ e $n-3$ respectivamente. Cada uma destas formas é uma forma válida

de terminar de subir os n degraus, e todas pode ser contabilizadas. Isto sugere a recorrência

$$M_n = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ 4, & n = 3 \\ M_{n-1} + M_{n-2} + M_{n-3}, & n > 3 \end{cases}$$

Perceba que ela já está com 3 casos iniciais por que a regra geral precisa recuar três termos. Podemos programar facilmente e obter valores:

```
define m(n) {
  if ( n == 1 ) return 1;
  if ( n == 2 ) return 2;
  if ( n == 3 ) return 4;
  return m(n-1) + m(n-2) + m(n-3);
}
for(i=1; i<=12; i++) m(i);
1
2
4
7
13
24
44
81
149
274
504
927
```

Existe uma simplificação que pode ser feita na recorrência acima. Ela se baseia em pensar assim: se a recorrência vai sempre reduzir o valor de n para $n-1$, $n-2$ ou $n-3$, então este valor vai acabar sendo zero ou negativo. Nestes casos:

- Quantas formas existem de subir uma escada de zero degraus? Resposta: uma forma, que é ficando parado.
- Quantas formas existem de subir uma escada com número negativo de degraus? Resposta: não há maneira, ou seja, zero maneiras.

Reescrevendo a recorrência anterior, temos

$$M_n = \begin{cases} 0, & n < 0 \\ 1, & n = 0 \\ M_{n-1} + M_{n-2} + M_{n-3}, & n > 0 \end{cases}$$

Mais simples, com casos mais elementares e dando os mesmos resultados. Faça um teste.

O que aprendemos

Você pode pensar que nos dois últimos exemplos, da senha e da escada, ainda seria possível fazer um programa que não seja recursivo e calcule os números corretamente. É verdade, você pode usar as recursões que estão apresentadas e a partir delas fazer programas não recursivos que calculam os mesmos valores, como nós mesmos já fizemos para o fatorial e para os números de Fibonacci. No entanto, a pergunta mais importante é **outra**: sem ter as recursões que mostram como os números crescem, como seriam estes programas? O que você usaria para construí-los?

Por exemplo, como seria um programa que examina o problema das escadas? Provavelmente você pensaria "Bem, eu saio do degrau zero, e subo um, dois ou três de cada vez, e sempre que chegar no degrau n aumento um contador, no fim escreve o contador...". Não seria algo parecido? Tente implementar e você vai ver que é praticamente a mesma recursão que já escrevemos, só que você não vai estar **contando**, e sim **simulando** o que acontece, por isso vai ter mais trabalho.

Em outro exemplo, voltemos ao problema das senhas. Imagine que em vez de ter 12 letras a senha da sua vó tem 24 letras por que ela é maníaca por segurança. Como seria seu programa que conta quantas senhas são válidas? Parecido com o da escada, certo? Você começa com zero letras, vai adicionando uma a mais, evita o caso CC, sempre que chegar no tamanho 24

incrementa o contador, no final imprime o contador... deu pra entender, né? Bom, e as recursões? Depois de programar as recorrências elas rodaram no `bc` e acharam o resultado: 81325141303041 senhas, em **meio segundo**. Isso são mais de 81 **trilhões** de senhas. Perceba a diferença: enquanto o seu programa cria cada uma dessas senhas, a recursão apenas conta quantas são sem criar nada.

Esta é a moral da história: recorrências e recursões não são apenas formas de implementar, mas de pensar sobre o problema e achar uma forma simples de resolver.

Uma nova versão da escada

Voltando ao problema da escada, vamos pensar em uma versão mais realista: você cansa logo se subir degraus de 3 em 3, e por isso quer saber quantas maneiras tem para subir a escada de n degraus se puder subir 3 degraus apenas k vezes. Por exemplo, se não permitirmos nenhuma subida de três degraus, temos $k = 0$. Se permitirmos apenas uma vez, temos $k = 1$ e assim por diante. Devemos estar procurando algo similar ao M_n de antes, mas agora k também deve estar sendo considerado, portanto devemos procurar por algo chamado $M_{n,k}$. Vamos ver o que sabemos:

1. Um problema verdadeiro deve calcular $M_{n,k}$ com n sendo o tamanho da escada e k o número de subidas de três degraus que são permitidas. Essa é a chamada recursiva que deve ser feita no início do problema.

2. Como k afeta apenas as subidas de três degraus, não afeta os casos para $n = 1$ e $n = 2$ que já conhecemos. No entanto k afeta o valor para o caso $n = 3$: nos 4 casos que contabilizamos, um deles inclui uma subida de 3 degraus, e ela só pode ser contada se $k > 0$. Se $k = 0$ ela não pode ser considerada por que não temos mais subidas de 3 degraus para usar. Neste caso, podemos tentar iniciar a recursão assim:

$$M_{n,k} = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ 3, & n = 3,\ k = 0 \\ 4, & n = 3,\ k > 0 \end{cases}$$

Agora vem a parte mais complicada, buscar a equação geral. A boa notícia é que a forma de pensar para o caso $n = 3$ mostrada acima também serve para a regra geral: se tivermos $k = 0$, temos que contar apenas formas que usam 1 e 2 degraus, e se tivermos $k > 0$ contamos as mesmas formas de 1 e 2 degraus e mais uma opção de subir 3 degraus, apenas não podemos esquecer de que neste caso gastamos um dos pulos de três degraus e por isso k deve diminuir. Juntando tudo, temos

$$M_{n,k} = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ 3, & n = 3,\ k = 0 \\ 4, & n = 3,\ k > 0 \\ M_{n-1,k} + M_{n-2,k} & n > 3,\ k = 0 \\ M_{n-1,k} + M_{n-2,k} + M_{n-3,k-1} & n > 3,\ k > 0 \end{cases}$$

A última linha é a mais interessante e merece atenção especial. Nela podemos gastar um passo de três degraus, e ao darmos esse passo k deve diminuir em 1. Para os casos de 1 e 2 degraus, k não diminui por que não gastamos um dos preciosos passos de 3 degraus nestes casos. Agora pense um pouco para se convencer de que a versão reduzida pode ser escrita assim:

$$M_{n,k} = \begin{cases} 0, & n < 0 \text{ } ou \text{ } k < 0 \\ 1, & n = 0 \\ M_{n-1,k} + M_{n-2,k}, & n > 0,\ k = 0 \\ M_{n-1,k} + M_{n-2,k} + M_{n-3,k-1}, & n > 0,\ k > 0 \end{cases}$$

Só para mostrar como recorrências podem ser interessantes e dar espaço para criatividade, pense um pouco para explicar por que a recorrência acima pode ser reduzida a algo ainda menor:

$$M_{n,k} = \begin{cases} 0, & n < 0 \; ou \; k < 0 \\ 1, & n = 0 \\ M_{n-1,k} + M_{n-2,k} + M_{n-3,k-1}, & n > 0 \end{cases} \quad (1)$$

Nesta versão não nos importamos com o valor de k sendo zero ou não quando ainda temos degraus a subir. Programando, obtemos alguns resultados:

```
m(3,0) = 3  // O caso de n=3 onde não podemos dar passos de 3 degraus.
m(3,1) = 4  // O caso onde podemos
m(3,2) = 4  // Só para três degraus não adianta dar mais pulos.
m(6,0) = 13 // Indo mais longe
m(6,1) = 23
m(6,2) = 24 // Qual caso tem a mais?
m(15,0) = 987
m(15,1) = 3272
m(15,2) = 5108
m(25,0) = 121393
m(25,1) = 609793
m(25,2) = 1414021
m(25,4) = 2450343
m(25,8) = 2555757
m(50,5) = 6883599386621
m(50,10) = 10555638249256
```

Pensando mais um pouco A recursão que foi implementada para gerar os números acima foi a recursão (1). Perceba que ela chegou a números grandes como `m(50,10)` através da decomposição do problema em probleminhas menores até chegar nas regras mais simples da recursão. Até aqui não tem novidade, mas pense que para `m(50,10)` o resultado foi um número de 14 dígitos obtido somando-se apenas zeros e uns, que são os valores dos casos mais simples. Ou seja, sabemos que chegamos 10555638249256 vezes na regra que retorna 1 e um número desconhecido de vezes na regra que retorna zero. Algumas perguntas surgem imediatamente:

1. O que poderíamos fazer para descobrir quantas vezes usamos a regra que retorna zero?
2. Esse cálculo recursivo deve ter gasto uma quantidade de tempo enorme. Quanto tempo? O processo pode ser acelerado? Como esse cálculo foi feito **de verdade**?
3. O que o número que obtemos significa? O valor `m(50,10)` representa quantas formas temos de subir uma escada de 50 degraus com **exatamente** 10 pulos de 3 degraus, com **no máximo** 10 pulos ou o quê? Pense um pouco e descubra qual dessas opções é verdadeira. Depois mude a recorrência para calcular a outra opção.

Dois problemas mais avançados

Problema 1 Suponha que você tem uma urna com bolinhas brancas e pretas. Digamos, a bolinhas brancas e b bolinhas pretas. Você vai fazer um sorteio retirando k bolinhas lá de dentro e deseja saber a probabilidade de retirar exatamente n bolinhas brancas. O enunciado do problema não é muito complicado e rapidamente podemos tirar várias conclusões simples:

- Existem 4 variáveis envolvidas:
  - a, número de bolinhas brancas;

  - b, número de bolinhas pretas;
  - k, número de bolinhas que serão retiradas;
  - n, número de bolinhas brancas que desejamos (e por isso também desejamos $k-n$ bolinhas pretas);

- O problema não assume números iguais de brancas e pretas. Podemos ter uma urna com 5 bolinhas brancas, 200 bolinhas pretas e perguntar qual a probabilidade de tirar as 5 brancas se retirarmos 5 bolinhas lá de dentro;

- Não poderemos obter mais do que k bolinhas brancas, que é o máximo que pode ser retirado, ou seja, a probabilidade é zero se $n > k$.

- A qualquer momento, não é possível tirar mais bolinhas brancas do que as que temos, ou seja, a probabilidade é zero se $n > a$.

- Um sorteio mal sucedido termina assim:
  - quando podemos retirar $k = 0$ bolinhas e desejamos $n > 0$ bolinhas brancas;
  - quando se $n > k$ (perceba que esta condição é mais abrangente que a anterior e pode substituí-la!);
  - quando $n > a$;

- Um sorteio bem sucedido termina quando temos que retirar $k = 0$ bolinhas e desejamos $n = 0$ bolinhas brancas;

A esta altura podemos dar um nome para a quantidade que desejamos:

$$P(n, k, a, b)$$

E as condições que descobrimos já permitem escrever

$$P(n,k,a,b) = \begin{cases} 1, & n = k = 0 \\ 0, & n > k \\ \ldots & \end{cases}$$

O problema mais sério, claro, são os pontinhos ... na recorrência. Eles tratam do caso do sorteio que ainda está na metade, ou seja, quando ainda não chegamos a uma situação final. Nesse caso temos duas situações para considerar, com probabilidades que devem ser somadas:

- Retiramos uma bolinha branca

  Se temos $a + b$ bolinhas, a probabilidade da que foi retirada ser branca é $a/(a + b)$. Além disso, teremos "queimado" uma de nossas retiradas, passando a ter $k-1$ retiradas e queremos agora $n-1$ bolinhas brancas, afinal uma delas acaba de aparecer. A quantidade de bolinhas brancas para serem retiradas também passa a ser $a - 1$.

- Retiramos uma bolinha preta

  Temos $a+b$ bolinhas, por isso a probabilidade dela ser preta é $b/(a+b)$. Também teremos "queimado" uma de nossas retiradas, passando a ter $k - 1$ retiradas mas ainda queremos n bolinhas brancas. A quantidade de bolinhas pretas para serem retiradas passa a ser $b - 1$.

Juntando tudo temos

$$P(n,k,a,b) = \begin{cases} 1, & n = k = 0 \\ 0, & n > k \\ \frac{a}{a+b} * P(n-1,k-1,a-1,b) + \frac{b}{a+b} * P(n,k-1,a,b-1) & \end{cases}$$

A parte mais complicada e talvez mais interessante de todo o problema é perceber que somente as duas condições de parada que temos são suficientes para todas as situações. Depois de programar, podemos começar a obter resultados:

- $P(1,1,1,1)$ é a probabilidade de retirar uma bolinha branca quando temos uma branca e uma preta e vamos retirar uma bolinha só. O resultado é 0.5, exatamente como deveria ser.
- $P(2,1,1,1)$ é a probabilidade de retirar duas bolinhas brancas se temos uma branca e uma preta e vamos retirar uma bolinha só. O resultado é 0. Não surpreende...
- $P(1,2,1,1)$ é a probabilidade de retirar uma bolinha branca quando temos uma branca e uma preta e vamos retirar duas bolinhas. O resultado é 1. Faz sentido.
- $P(1,1,1,2)$ é a probabilidade de retirar uma bolinha branca se temos uma branca e duas pretas e vamos retirar uma bolinha. O resultado é $0.3333\ldots$. Parece muito com $1/3$, não?
- $P(1,2,2,2)$ é a probabilidade de retirar uma bolinha branca quando temos duas brancas e duas pretas e vamos retirar duas bolinhas. O resultado é $0.6666\ldots$. Parece muito com $2/3$, e é um número inesperadamente alto. Faça os casos à mão se estiver com dúvida, vale a pena.
- $P(2,2,2,2) = 1/6$ e tem ligação direta com o caso anterior. Se você fez os casos à mão, o motivo deve ser claro.
- $P(1,1,1,99) = 0.01$. Interprete este caso...
- $P(10,20,30,40) = 0.157322\ldots$ mas $P(10,20,30,80) = 0.0112584\ldots$. Ou seja, tendo mais bolinhas pretas a probabilidade das brancas aparecerem vai diminuindo rapidamente. Só que $P(10,20,80,30) = 0.0112584\ldots$ **também**, ou seja, ter bolinhas brancas em excesso também vai diminuir a probabilidade de aparecerem exatamente as n que desejamos. Seu desafio agora pode ser mostrar que $P(n,k,a,b) = P(n,k,b,a)$. Será que é verdade?

Como pergunta extra você pode responder o que estará sendo calculado se simplesmente retirarmos as probabilidades $a/(a+b)$ e $b/(a+b)$ na recorrência. Isso também tem um significado.

Problema 2 Imagine que seu irmão menor está fazendo um álbum de figurinhas. O álbum tem 300 figurinhas e no início tudo é muito fácil por que cada envelope de figurinhas traz figurinhas novas, mas à medida que o álbum vai sendo preenchido as probabilidades de um envelope trazer uma figura nova vão ficando cada vez menores. Nessa hora você entende que seu irmão vai ter que comprar muito mais do que 300 figurinhas para completar o álbum (ou trocar com os amigos, mas isso é outra história). Você resolve analisar o assunto mais a fundo:

- Vamos iniciar supondo que o álbum tem $T = 300$ figurinhas. (Não custa nada ser genérico para poder trocar o valor de T se der vontade.)
- Os envelopes de figurinhas são um problema e você resolve se livrar deles, pensando que as figurinhas são compradas uma de cada vez.

- À medida que o tempo passa seu irmãozinho terá comprado cada vez mais figurinhas. Vamos chamar esse número acumulado de k. O valor de k vai crescendo à medida que ele faz novas compras e é claro que para preencher o álbum precisamos ter $k \geq T$.

- Pensando em casos pequenos, você percebe que a primeira figurinha que seu irmão comprar certamente será uma figurinha nova e pode ser colada no álbum. Já a segunda figurinha **provavelmente** será uma figurinha nova mas existe uma pequena probabilidade de ser uma repetição da primeira. Você nota que nocaso da segunda figurinha essa probabilidade de repetir é de uma em T (ou seja, $1/T$) e a probabilidade de ser nova é $(T-1)/T$. Depois de ter duas figurinhas no álbum a probabilidade da terceira ser repetida são de $2/T$ e de ser nova são $(T-2)/T$ e assim por diante.

- À medida que novas figuras são encontradas a probabilidade de repetição aumenta até o caso extremo em que só falta uma para completar o álbum. Nesse caso a probabilidade de acertar é de apenas 1 em T e as de errar são de $T-1$ em T, ou seja, $(T-1)/T$.

- O que você deseja saber é a probabilidade de ter n figurinhas novas depois de ter comprado k figurinhas. Ou seja, você quer saber $P(n,k)$. Analisando as situações possíveis, podemos ter:

  - Com certeza não podemos ter n figurinhas novas se $n > k$. Neste caso, $P(n,k) = 0$.
  - Também não faz sentido falar em $n < 0$ ou $k < 0$, e para estes casos temos $P(n,k) = 0$ outra vez.
  - Se comprarmos apenas uma figurinha teremos exatamente uma figurinha nova, portanto $P(1,1) = 1$ e $P(n,1) = 0$ para qualquer $n \neq 1$. Ou seja, se $k = 1$ não podemos ter 0 nem 2, 3, 4,... figurinhas novas.

Depois de coletar todas essas informações, o mais complicado é (como sempre) encontrar uma regra para o caso geral em que acabamos de comprar a figurinha k e não sabemos muito sobre o que havia antes. A figurinha k pode ser nova ou repetida, mas com qual probabilidade? Para saber isso não podemos depender apenas do valor de k, pois dependemos também do número n de novas que já encontramos. Voltando ao objetivo de calcular $P(n,k)$, temos

- Desejamos ter n figurinhas novas, mas a figurinha k pode ser nova ou repetida. A probabilidade de termos n novas depois de comprar k figurinhas é a soma de duas probabilidades: a probabilidade da figurinha k ser nova quando tínhamos $n-1$ novas até aquele momento mais a probabilidade dela ser repetida quando já tínhamos n novas. Ou seja, é a soma de $P(n-1,k-1)$ e $P(n,k-1)$. Agora temos que levar em conta a probabilidade de cada um desses casos acontecer.

- O caso $P(n,k-1)$ significa que a figurinha k é repetida, então já tinhamos encontrado n novas antes e a probabilidade dela ser repetida é n/T.

- O caso $P(n-1,k-1)$ significa que a figurinha k é nova, então já tinhamos encontrado $n-1$ novas e a probabilidade de ser uma figura nova é de $(T-(n-1))/T$.

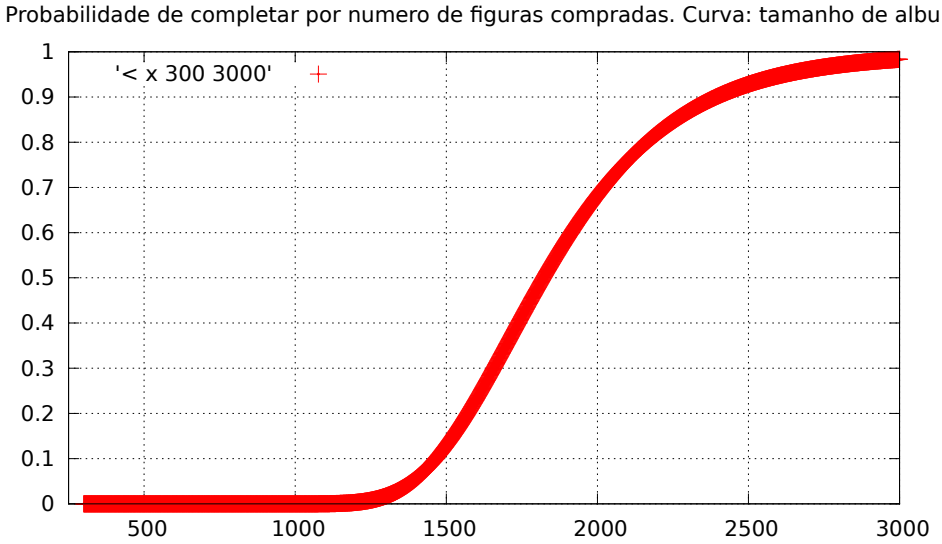
Juntando tudo, temos a recorrência final:

$$P(n,k) = \begin{cases} 0, & n \leq 0 \; ou \; k \leq 0 \; ou \; n > k \\ 1, & n = 1 \; e \; k = 1 \\ 0, & n \neq 1 \; e \; k = 1 \\ P(n-1,k-1) * \frac{T-(n-1)}{T} + P(n,k-1) * \frac{n}{T} & \end{cases}$$

Depois disso tudo, pense mais um pouco e entenda como a recorrência acima pode ser ser simplificada para

$$P(n,k) = \begin{cases} 0, & n < 0 \, ou \, k < 0 \\ 1, & n = 0 \, e \, k = 0 \\ P(n-1,k-1) * \frac{T-(n-1)}{T} + P(n,k-1) * \frac{n}{T} & \end{cases}$$

Agora vem as notícias ruins para o seu irmão: depois de programar a recorrência você produz um gráfico como o que está a seguir, mostrando como a probabilidade de ter 300 figurinhas novas vai aumentando à medida que elas são compradas.



Com ele você entende várias coisas: sabe que depois de comprar 3000 figurinhas ainda existe uma pequena probabilidade de não ter completado o álbum. sabe como a editora do álbum ganha a maior parte do seu dinheiro e também sabe até que ponto comprar figurinhas e depois desistir das compras e tentar trocar com os amigos.

Recorrências mútuas

Às vezes alguns problemas não podem (ou não devem) ser atacados com apenas uma recorrência e existem situações que levam naturalmente a recorrências que estão associadas, que também são chamadas de recorrências mútuas. Um exemplo deste tipo de problema está descrito a seguir.

O planeta dos coelhinhos

Você estava viajando pelo espaço (isso se passa no ano 2335) quando sua nave cargueira teve uma pane e foi forçada a pousar em um planeta fora das rotas comerciais, e você pode ficar abandonado lá por bastante tempo. Entre as cargas existem muitas coisas úteis para a sobrevivência: seus mp3, vários equipamentos e até mesmo um casal de bebês-coelho que deviam ser entregues para um laboratório de pesquisa. Junto com os coelhos, há um folhetinho com as informações abaixo:

1. Estes coelhinhos nasceram no ano 2335.
2. Durante o primeiro ano de vida, os coelhos são bebês.
3. No segundo ano de vida, os coelhos são jovens.

4. A partir do terceiro ano de vida, os coelhos são adultos.

5. Um casal de coelhos adultos se reproduz uma vez ao ano.

6. A reprodução gera um novo casal de bebês-coelho.

7. Estes coelhos foram tratados geneticamente e não morrem.

Você já está pensando em jantar os coelhinhos hoje mesmo quando resolve fazer as contas e prever quantos coelhos terá no futuro se deixar que eles se reproduzam, afinal pode levar um tempo até a nave de resgate aparecer. Você começa com uma tabela assim:

	2335	2336	2337	2338	2339
Bebês	2				
Jovens	0				
Adultos	0				

Não é difícil saber o que você vai ter no ano que vem, afinal os coelhinhos se tornarão jovens:

	2335	2336	2337	2338	2339
Bebês	2	0			
Jovens	0	2			
Adultos	0	0			

No ano 2337 as coisas serão mais interessantes, pois você vai ter seu primeiro casal de coelhos adultos e eles fornecerão um casal de bebês:

	2335	2336	2337	2338	2339
Bebês	2	0	2		
Jovens	0	2	0		
Adultos	0	0	2		

Em 2338 a situação fica ainda mais interessante pois o seu novo casal de bebês passa a ser jovem e o casal de adultos fornece mais um par de bebês:

	2335	2336	2337	2338	2339
Bebês	2	0	2	2	
Jovens	0	2	0	2	
Adultos	0	0	2	2	

Para terminar as previsões você imagina que em 2339 terá uma situação mais complicada: o casal de bebês passa a ser jovem, o casal de jovens passa a adulto e fornece mais um par de bebês, o casal de adultos continua lá e fornece mais outro par de bebês. O resultado por enquanto é

	2335	2336	2337	2338	2339
Bebês	2	0	2	2	4
Jovens	0	2	0	2	2
Adultos	0	0	2	2	4

Neste ponto você cansa de fazer continhas e resolve descobrir quantos coelhos vai ter em um ano qualquer. A primeira coisa que você percebe é que poderia tentar calcular C_n, que seria o total de coelhos no ano n, mas talvez seja mais simples calcular quantidades separadas:

- B_n é o número de bebês no ano n;

- J_n é o número de jovens no ano n;
- A_n é o número de adultos no ano n;

E se você tiver essas três quantidades, é claro que terá C_n pois $C_n = B_n + J_n + A_n$. Vamos juntar o que você sabe até o momento:

- No ano 1 em que você está no planeta, tem apenas um casal de bebês e nenhum jovem ou adulto (e é bem melhor usar ano 1 do que ano 2335...). Então, $B_1 = 2$ e $J_1 = A_1 = 0$.
- Os jovens em um ano são os bebês que havia no ano passado, portanto $J_n = B_{n-1}$.
- Os adultos de um ano são os mesmos adultos do ano anterior (pois eles não morrem) mais os jovens daquele ano, pois eles passaram a ser adultos. Portanto, $A_n = A_{n-1} + J_{n-1}$.
- Agora você tem que descobrir quantos bebês terá em um certo ano. Isso não é difícil, pois de um ano para o outro cada casal de adultos e jovens em um ano fornece um outro casal de bebês no ano seguinte, e por isso $B_n = A_{n-1} + J_{n-1}$. Juntando todos os resultados temos

$$
\begin{aligned}
B_n &= \begin{cases} 2, & n = 1 \\ A_{n-1} + J_{n-1}, & n > 1 \end{cases} \\
J_n &= \begin{cases} 0, & n = 1 \\ B_{n-1}, & n > 1 \end{cases} \\
A_n &= \begin{cases} 0, & n = 1 \\ A_{n-1} + J_{n-1}, & n > 1 \end{cases}
\end{aligned}
$$

Pelas equações acima notamos que $B_n = A_n$ quando $n > 1$, pois as duas quantidades são iguais a $A_{n-1} + J_{n-1}$. Isso permite algumas simplificações, mas vamos evitar simplificar por enquanto para ir direto para a programação.

Podemos escrever um algoritmo não recursivo como este para calcular os valores ao lado:

```
ano = 1;
b = 2;
j = a = 0;
while (ano <= 15) {
  print b, " ", j, " ", a, "\n";
  proxb = a + j;
  proxj = b;
  proxa = a + j;
  b = proxb;
  j = proxj;
  a = proxa;
  ano = ano + 1;
}
```

```
2 0 0
0 2 0
2 0 2
2 2 2
4 2 4
6 4 6
10 6 10
16 10 16
26 16 26
42 26 42
68 42 68
110 68 110
178 110 178
288 178 288
466 288 466
754 466 754
1220 754 1220
1974 1220 1974
3194 1974 3194
5168 3194 5168
```

Como exercício, escreva um algoritmo recursivo para calcular os mesmos valores que este algoritmo apresenta.

Os primeiros resultados conferem com nossa tabela e parece que a população de coelhos cresce rapidamente depois de alguns anos. Todas essas contas foram feitas para calcular o número de coelhos, mas na verdade poderiam medir o número de **casais** de coelhos: a única

mudança a ser feita é dizer que no ano 1 temos um casal de bebês, ou seja, $B_1 = 1$. Também podemos mudar o programa anterior para apresentar C_n, que agora será a quantidade total de casais de coelhos.

Executando as mudanças sugeridas, temos

```
1 0 0 1
0 1 0 1
1 0 1 2
1 1 1 3
2 1 2 5
3 2 3 8
5 3 5 13
8 5 8 21
13 8 13 34
21 13 21 55
34 21 34 89
55 34 55 144
89 55 89 233
144 89 144 377
233 144 233 610
377 233 377 987
610 377 610 1597
987 610 987 2584
1597 987 1597 4181
2584 1597 2584 6765
```

Agora cumpra as seguintes missões:

1. Volte para as equações que definem B_n, J_n e A_n e faça todas as simplificações que puder.
2. Examine a tabela ao lado e chegue a uma conclusão sobre os números C_n.
3. Prove sua conclusão. Não é difícil.

As senhas secretas (mais uma vez)

Pois é, sua vó perdeu a senha de novo. Só que desta vez ela tem mais superstições do que antes e a senha formada por A, B, C, D e E não pode ter CC nem DD juntos (perceba também que ela adicionou a letra E para dar mais segurança). Com a senha perdida, você tem o mesmo problema de antes: calcular quantas senhas podem existir com comprimento n. Você tenta modelar o problema com a técnica que já conhecia, que impedia que houvesse CC mas descobre que é mais complicado evitar CC e DD ao mesmo tempo e ainda permitir coisas como CDCDCD. Para este tipo de situação, você resolve tentar controlar as letras individualmente, ou seja, sabendo quantas senhas terminam com A, com B, com C, com D e com E separadamente e depois dar um jeito de juntar tudo. Você sabe que:

- Uma letra A pode ser colocada em uma senha com qualquer final, gerando uma nova senha.
- O mesmo acontece com as letras B e E, que não sofrem restrições.
- Uma letra C pode ser colocada em qualquer senha, exceto as que já terminam com C.
- A mesma regra de C vale para a letra D.

Se voce resolveu separar as situações pelas letras finais, pode começar a modelar com as senhas que terminam com a letra A, que podem ser colocadas em qualquer senha com uma letra a menos:

$$A_n = \begin{cases} 1, & n = 1 \\ A_{n-1} + B_{n-1} + C_{n-1} + D_{n-1} + E_{n-1}, & n > 1 \end{cases}$$

E se for assim mesmo, as situações para B e E são praticamente iguais:

$$B_n \;=\; E_n \;=\; \begin{cases} 1, & n = 1 \\ A_{n-1} + B_{n-1} + C_{n-1} + D_{n-1} + E_{n-1}, & n > 1 \end{cases}$$

E se for assim, como ficam as recorrências para C e D, que não podem vir duplicadas? É muito simples, basta bloquear essa possibilidade:

$$C_n \;=\; \begin{cases} 1, & n = 1 \\ A_{n-1} + B_{n-1} + D_{n-1} + E_{n-1}, & n > 1 \end{cases}$$

$$D_n \;=\; \begin{cases} 1, & n = 1 \\ A_{n-1} + B_{n-1} + C_{n-1} + E_{n-1}, & n > 1 \end{cases}$$

Agora você pode responder a algumas perguntas extras:

1. Você está interessado em saber qual resultado?
2. É possível tirar proveito de $A_n = B_n = E_n$ e ter algo mais simples?
3. É possível generalizar a resposta para p letras que podem repetir e q letras que não podem repetir?

Recorrências numéricas

Recorrências (ou equações recorrentes) são comumente encontradas a partir de problemas recursivos, mas podem surgir em outras aplicações não-recursivas, dependendo da abordagem usada. Em geral, recorrências expressam um conhecimento parcial sobre algum resultado, que não pode ser completamente calculado sem que se calculem também outras quantidades. Por exemplo, a tradicional definição de x^n:

$$x^n = \begin{cases} 1, & n = 0 \\ x * x^{n-1}, & n > 0 \end{cases} \tag{2}$$

Com a definição acima não é possível calcular diretamente x^5, tendo-se primeiro de calcular x^4, x^3 e assim sucessivamente. Desta forma, uma equação recorrente não pode ser avaliada diretamente e geralmente temos de resolver outros casos da mesma recorrência até acharmos a resposta adequada. É interessante notar que pode haver mais de uma recorrência que calcule a mesma coisa. Por exemplo, alterando a recorrência anterior podemos chegar a uma outra versão que também serve para calcular x^n, dada a seguir:

$$x^n = \begin{cases} 1, & n = 0 \\ x, & n = 1 \\ x^{\lfloor n/2 \rfloor} * x^{\lceil n/2 \rceil}, & n > 1 \end{cases} \tag{3}$$

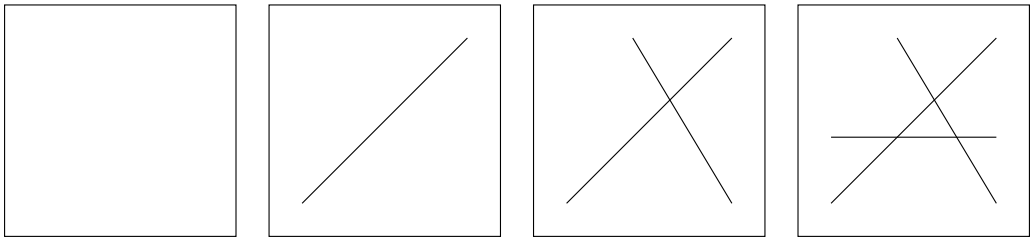
Nesta nova recorrência, $\lceil\ \rceil$ e $\lfloor\ \rfloor$ significam arredondamento e truncamento, respectivamente.

Pergunta

1. Compare as recorrências (2) e (3) acima, decidindo qual das duas é mais eficiente. Faça alguns testes e verifique se os resultados são diferentes e se o funcionamento é diferente.

Exemplo 1

Para perceber como recorrências podem surgir em aplicações práticas, considere que você pode ter n linhas cortando em um plano, como na figura abaixo:



Zero retas | Uma reta | Duas retas | Tres retas

Neste caso, podemos chamar de L_n o maior número possível de regiões definidas pelas n retas. Evidentemente, pelo desenho, temos então

$$\begin{aligned} L_0 &= 1 \\ L_1 &= 2 \\ L_2 &= 4 \\ L_3 &= 7. \end{aligned}$$

Depois deste conhecimento inicial podemos fazer a pergunta: como L_n se comporta para um n qualquer? Podemos perceber que uma nova reta (reta n) corta as $n-1$ retas anteriores no máximo em $n-1$ pontos, e com isso são definidas n novas áreas. A partir deste raciocínio temos que

$$L_n = L_{n-1} + n$$

Infelizmente esta recorrência está incompleta, pois se tentarmos calcular L_n a partir dela precisaremos de L_{n-1}, que precisará de L_{n-2} e assim sucessivamente sem que esta recursão seja terminada. Para que esta recorrência esteja completa, no entanto, basta adicionar um termo já conhecido:

$$L_n = \begin{cases} 1, & n = 0 \\ L_{n-1} + n, & n > 0 \end{cases} \tag{4}$$

Com esta nova informação servindo de condição de parada, podemos calcular L_7, por exemplo, calculando para isto L_6, L_5, L_4, L_3, L_2, L_1 e L_0.

Perguntas

1. Na recorrência (4) foi preciso adicionar um termo para fazer a recursão parar. Podem haver casos em que seja necessário adicionar mais termos? Dê um exemplo.

2. Imagine agora duas novas quantidades: A_n é o número máximo de regiões abertas definidas pelas n retas, e F_n é o número máximo de regiões fechadas definidas pelas n retas. Então é evidente que $L_n = A_n + F_n$. A partir disto desenvolva os valores iniciais e as fórmulas de recorrência para A_n e F_n.

Exemplo 2

Outro exemplo pode ser o número de bactérias em um ambiente. Em geral, muitos microorganismos se reproduzem a uma taxa constante (por exemplo, duplicam a cada 25 minutos ou a cada hora, enquanto as condições do ambiente forem ideais). Podemos supor que temos uma população de bactérias que dobra a cada hora, a partir de uma população inicial x. Neste caso, podemos estar interessados em saber quantas teremos na hora n, chamando este número de B_n. Para este caso a modelagem é bastante simples, e temos

$$B_n = \begin{cases} x, & n = 1 \\ 2B_{n-1}, & n > 1 \end{cases}.$$

Para estudantes de informática não deve ser difícil determinar a solução da recorrência acima. No entanto, se quisermos fazer um modelo mais complicado, uma solução direta é bem mais difícil: suponha que a cada hora colocamos outras y bactérias no ambiente, e portanto o modelo passa a ser

$$B_n = \begin{cases} x, & n = 1 \\ 2B_{n-1} + y, & n > 1 \end{cases} .$$

Com o que você já sabe de somatórios (e um pouco de paciência) ainda é possível resolver esta recorrência através de um dos métodos que mostraremos abaixo. Porém, se formos mais longe e tentarmos incluir no modelo o fato de que as bactérias vivem apenas 5 horas, teremos uma situação mais complicada:

$$B_n = \begin{cases} x, & n = 1 \\ 2B_{n-1} + y - B_{n-4}, & n > 1 \end{cases} .$$

A recorrência acima parece ser ainda bastante simples, pois apenas retira do ambiente as bactérias que existiam algum tempo atrás. Infelizmente a recorrência está **errada**, pois não seremos capazes de calcular o valor de B_2, por exemplo (por quê? Quais valores além desse não podem ser obtidos?). Neste caso, temos de ter maneiras de contornar o problema (existem pelo menos duas maneiras. Quais são?).

Perguntas

1. O modelo para o crescimento das bactérias prevê que elas dobrem de número a cada hora. O que acontece se a taxa de aumento não for o dobro? E se pensarmos que elas triplicam?

2. O modelo também prevê que as bactérias vivem sempre 5 horas. O que acontece se usarmos uma abordagem diferente e assumirmos que 85% das bactérias continuam vivas de uma hora para a seguinte? Altere o modelo para representar esta situação. Note que as duas hipóteses são diferentes, e explique as diferenças.

Resolvendo recorrências

Assuma que você tenha recebido a recorrência abaixo:

$$P(n) = \begin{cases} 3, & n = 0 \\ 2 * P(n-1) + n - 2, & n > 0 \end{cases} \tag{5}$$

A partir de agora o problema é tentar resolvê-la através de uma fórmula fechada, evitando ter de calcular casos sucessivos como para a recorrência (4). Geralmente, a primeira coisa a fazer é coletar informações sobre o comportamento da recorrência, como na tabela a seguir:

n	0	1	2	3	4	5	6	7	8	9	10
$P(n)$	3	5	10	21	44	91	186	377	760	1527	3062

Os valores parecem crescer a cada novo valor de n, mas não formam uma imagem clara de como foram construídos a partir da recorrência inicial. Para tentar formar um quadro de como se compõem os valores, podemos alterar alguns termos na recorrência e transformá-la em uma recorrência algébrica:

$$P(n) = \begin{cases} a, & n = 0 \\ 2 * P(n-1) + n + b, & n > 0 \end{cases} \tag{6}$$

Desenvolvendo a mesma tabela para esta nova recorrência, temos os valores decompostos:

n	$P(n)$			n	$P(n)$		
0	a			5	$32a$	$+31b$	$+57$
1	$2a$	$+b$	$+1$	6	$64a$	$+63b$	$+120$
2	$4a$	$+3b$	$+4$	7	$128a$	$+127b$	$+247$
3	$8a$	$+7b$	$+11$	8	$256a$	$+255b$	$+502$
4	$16a$	$+15b$	$+26$	9	$512a$	$+511b$	$+1013$

A partir destes valores é mais fácil reconhecer os coeficientes de a e de b, embora o número adicionado a cada fórmula ainda seja difícil de reconhecer. A partir destes dados já podemos nos sentir confiantes que $P(n)$ obedece à fórmula geral

$$P(n) = 2^n a + (2^n - 1)b + C(n), \tag{7}$$

onde $C(n)$ é uma constante ainda a ser determinada. (Não foi **provado** que esta fórmula geral está correta, apenas estamos razoavelmente convencidos após examinar a forma de $P(n)$). Ainda temos de descobrir o valor de $C(n)$, que é desconhecido. Felizmente não é difícil reconhecer que $C(n)$ cresce de forma semelhante a 2^{n+1}, com um valor apenas um pouco menor. Depois de alguma tentativa e erro, obtemos $C(n) = 2^{n+1} - n - 2$. Com este resultado temos a solução da recorrência (6), ou seja, independentemente dos valores que a e b possam ter. Para achar a solução de nossa recorrência original, inserimos seus valores antigos de a e b (ou seja, 3 e 2 respectivamente):

$$\begin{aligned} P(n) &= 3 * 2^n - 2 * (2^n - 1) + 2^{n+1} - n - 2 & (8)\\ &= 3 * 2^n - n & (9) \end{aligned}$$

Pergunta

1. Até este momento não **provamos** que a solução encontrada em (9) é mesmo a solução, apenas ela parece convincente. Como podemos fazer para provar que temos mesmo a solução? Bom, se temos a solução devemos poder encaixá-la de volta no problema original, ou seja, em (5), e ela deve funcionar. Pense em como fazer isto e descubra que é muito simples, bem mais do que encontrar a resposta.

Sendo mais esperto

Evidentemente, não é desejável depender de intuição e sorte para reconhecer os termos de uma recorrência e poder resolvê-la. Tentando ser mais metódicos na solução vamos experimentar resolver a mesma recorrência com o **método da iteração**.

Este método baseia-se em avaliar sucessivamente a recorrência para alguns valores de n e reconhecer o padrão para cada coeficiente, mas de forma mais metódica e disciplinada, mesmo que um pouco mais trabalhosa. Refazendo as avaliações de $P(n)$ sem realizar nenhuma operação aritmética, temos:

$$\begin{aligned} P(0) &= a\\ P(1) &= 2a + b + 1\\ P(2) &= 2 * (2a + b + 1) + b + 2\\ &= 4a + b + 2b + 2 * 1 + 2\\ P(3) &= 2 * (4a + b + 2b + 2 * 1 + 2) + b + 3\\ &= 8a + b + 2b + 4b + 2 * 2 * 1 + 2 * 2 + 3\\ P(4) &= 2 * (8a + b + 2b + 4b + 2 * 2 * 1 + 2 * 2 + 3) + b + 4\\ &= 16a + b + 2b + 4b + 8b + 2 * 2 * 2 * 1 + 2 * 2 * 2 + 2 * 3 + 4\\ &= 16a + b * (1 + 2 + 4 + 8) + 2^3 * 1 + 2^2 * 2 + 2^1 * 3 + 2^0 * 4 \end{aligned}$$

A partir destas primeiras avaliações fica claro que o coeficiente de a tem de ser 2^n, pois a cada nova avaliação o coeficiente é multiplicado por dois. O coeficiente de b e o termo independente são um pouco mais difíceis de identificar, porém logo se chega a

$$P(n) = 2^n a + b\sum_{i=0}^{n-1} 2^i + \sum_{i=0}^{n-1} 2^i(n-i).$$

Resolvendo os somatórios através do método da perturbação, temos facilmente a forma geral

$$P(n) = 2^n a + (2^n - 1)b + 2^{n-1} - n - 2,$$

exatamente a mesma resposta obtida anteriormente. Desta vez, apesar do maior trabalho algébrico, temos mais certeza de ter obtido a resposta correta e sem recorrer à adivinhação.

Perguntas

1. Verifique como um somatório pode sempre ser transformado em uma recorrência.
2. Use o método acima para obter uma solução para a recorrência (5). Verifique se este valor é o mesmo obtido em (9).
3. Use o mesmo método acima para obter uma solução para a recorrência (4). Em seguida use o mesmo método para resolver as recorrências para A_n e F_n da pergunta 2.

Um método mais poderoso

O próximo método de resolução de recorrências é mais sofisticado e mais trabalhoso, mas evita a dificuldade de ter de expandir os termos da recorrência para procurar padrões que podem ser bastante difíceis de identificar. Como bônus adicional, o método resolve muito mais recorrências do que os anteriores, que dependem apenas da percepção do usuário. Essencialmente, este novo método baseia-se em achar partes da solução através de equações que são obtidas de dentro da recorrência[2]. Para demonstrar o mecanismo, vamos fazer a resolução da recorrência abaixo[3]:

$$R_n = \begin{cases} 0, & n = 1 \\ 2, & n = 2 \\ 3R_{n-1} - R_{n-2} + n - 4, & n > 2. \end{cases} \tag{10}$$

Avaliando os primeiros termos, temos a tabela a seguir:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R_n	0	2	5	13	35	94	250	660	1735	4551	11925	31232	81780	214118

A primeira coisa a fazer é trabalhar com o termo geral de nossa recorrência, ou seja, com aquele que gera os novos termos. Este termo é manipulado deixando-se todos os termos de recorrência no lado esquerdo e outros componentes no lado direito:

$$R_n = 3R_{n-1} - R_{n-2} + n - 4 \tag{11}$$

$$R_n - 3R_{n-1} + R_{n-2} = n - 4. \tag{12}$$

[2] A teoria que explica como este método funciona não será explicada nesta apostila.

[3] Esta recorrência é bem mais complicada do que as anteriores, e servirá para demonstrar como o método é poderoso, mas por ser mais difícil os cálculos também são mais complicados. Seria praticamente impossível identificar padrões neste caso.

Esta equação é chamada de **equação particular** da recorrência. Para começar desprezaremos completamente o lado direito da equação acima, criando o que chamamos de **equação homogênea**, ou seja, com o lado direito igual a zero:

$$R_n - 3R_{n-1} + R_{n-2} = 0. \tag{13}$$

No momento estamos interessados em resolver esta equação. Seremos totalmente arbitrários e decretaremos que $R_n = ax^n$ provavelmente resolve a equação. Não estamos preocupados com os valores de a ou de x neste momento, vamos apenas pensar neles como números desconhecidos e seguir em frente: com esta definição para R_n podemos reescrever (13) e tentar descobrir que valores de x podem resolvê-la. Teremos

$$\begin{aligned} R_n - 3R_{n-1} + R_{n-2} &= 0 \\ ax^n - 3ax^{n-1} + ax^{n-2} &= 0 \\ ax^{n-2}(x^2 - 3x + 1) &= 0 \end{aligned}$$

Existe mais de uma maneira de resolver a equação acima: com $a = 0$ ou $x = 0$ (alternativas desinteressantes porque não nos levam mais longe) ou com as raízes de $x^2 - 3x + 1 = 0$, que é chamada de **equação característica** da recorrência. Estas raízes são

$$x = \frac{3 \pm \sqrt{5}}{2}.$$

Como havíamos decretado que $R_n = ax^n$, já temos valores para x e podemos melhorar nossa resposta, escrevendo

$$R_n = ax^n \tag{14}$$

$$R_n = a_1 * \left(\frac{3+\sqrt{5}}{2}\right)^n + a_2 * \left(\frac{3-\sqrt{5}}{2}\right)^n. \tag{15}$$

Note-se que porque temos dois valores possíveis para x colocamos os dois dentro da resposta e trocamos a constante a usada originalmente por duas constantes a_1 e a_2, permitindo que elas assumam valores diferentes[4].

Agora que resolvemos a equação homogênea (13) vamos nos dedicar à resolução da equação particular (12). A equação particular tem um pequeno polinômio no lado direito, e vamos (mais uma vez) decretar que a resposta R_n é um polinômio de mesmo grau, ou seja, $R_n = a_3 n + a_4$. Os coeficientes a_3 e a_4 deste polinômio ainda são desconhecidos e é nossa tarefa descobri-los, substituindo em (12):

$$\begin{aligned} R_n - 3R_{n-1} + R_{n-2} &= n - 4 \\ a_3 n + a_4 - 3(a_3(n-1) + a_4) + a_3(n-2) + a_4 &= n - 4 \\ a_3 n + a_4 - 3a_3 n + 3a_3 - 3a_4 + a_3 n - 2a_3 + a_4 &= n - 4 \\ (-a_3)n + (a_3 - a_4) &= n - 4 \\ (-1 - a_3)n + (a_3 - a_4 + 4) &= 0 \end{aligned}$$

Esta é uma das etapas mais delicadas deste método: para que a solução seja independente de n, devemos ter cada coeficiente acima igual a zero. Ou seja, precisamos garantir que

$$\begin{aligned} -1 - a_3 &= 0 \\ a_3 - a_4 + 4 &= 0. \end{aligned}$$

Com isto, temos imediatamente que $a_3 = -1$ e $a_4 = 3$. Usando estes valores em nossa suposição de que $R_n = a_3 n + a_4$, temos

$$R_n = -n + 3. \tag{16}$$

[4]Se estamos na pista certa, já dá pra ver que a solução não tem uma aparência simples...

Depois de resolver as equações homogênea e particular devemos combinar os resultados para descobrir a solução final da recorrência. Isto é feito somando-se as duas respostas obtidas, (15) e (16):

$$R_n = a_1 * \left(\frac{3+\sqrt{5}}{2}\right)^n + a_2 * \left(\frac{3-\sqrt{5}}{2}\right)^n - n + 3. \tag{17}$$

Ainda não sabemos quais são as constantes a_1 e a_2 que devem ser utilizadas, mas é relativamente simples determiná-las usando as condições iniciais de nossa recorrência. Afinal, elas ainda não foram usadas para nada...

1. Usando a primeira condição, $R_1 = 0$, teremos

$$\begin{aligned} 0 &= a_1 * \left(\frac{3+\sqrt{5}}{2}\right)^1 + a_2 * \left(\frac{3-\sqrt{5}}{2}\right)^1 - 1 + 3. \\ -4 &= 3 * (a_1 + a_2) + \sqrt{5}(a_1 - a_2). \end{aligned}$$

2. Usando a segunda condição, $R_2 = 2$, teremos

$$\begin{aligned} 2 &= a_1 * \left(\frac{3+\sqrt{5}}{2}\right)^2 + a_2 * \left(\frac{3-\sqrt{5}}{2}\right)^2 - 2 + 3. \\ 4 &= a_1 * (14 + 6\sqrt{5}) + a_2 * (14 - 6\sqrt{5}) \\ 2 &= 7 * (a_1 + a_2) + 3\sqrt{5}(a_1 - a_2) \end{aligned}$$

Resolvendo as duas equações acima, temos

$$\begin{aligned} a_1 &= \frac{8\sqrt{5} - 10}{15 + 5\sqrt{5}} = \frac{17\sqrt{5} - 35}{10}, \\ a_2 &= \frac{8\sqrt{5} + 10}{5\sqrt{5} - 15} = -\frac{17\sqrt{5} + 35}{10}. \end{aligned}$$

Reconstruindo a recorrência (17) com estes valores, obtemos

$$R_n = \frac{17\sqrt{5} - 35}{10} * \left(\frac{3+\sqrt{5}}{2}\right)^n - \frac{17\sqrt{5} + 35}{10} * \left(\frac{3-\sqrt{5}}{2}\right)^n - n + 3.$$

Por mais difícil que possa parecer, esta é realmente a solução da recorrência (10), e um teste com qualquer software algébrico pode confirmar que esta fórmula fechada calcula os mesmos valores (é interessante notar como as frações e os termos com $\sqrt{5}$ desaparecem, deixando apenas números inteiros. Com certeza, ficar tentando achar padrões como nas abordagens anteriores não nos teria levado longe neste caso.)

Perguntas

1. Faça um programa que liste alguns dos valores da recorrência e valores calculados pela fórmula obtida, comparando-os. Não esqueça de que os valores calculados estão sujeitos a erro de arredondamento, e portanto podem haver pequenas diferenças.

2. Dada a solução para R_n, analise-a e verifique que o primeiro termo torna-se mais e mais importante à medida que n cresce, enquanto o segundo termo é cada vez menos significativo. Mostre o motivo e deduza que para n grande podemos ter uma fórmula aproximada[5]

$$R_n \approx \frac{17\sqrt{5} - 35}{10} * \left(\frac{3+\sqrt{5}}{2}\right)^n - n + 3.$$

Use um software algébrico ou sua calculadora e verifique que a diferença entre as duas expressões diminui rapidamente, tão rápido que ela é cerca de 3×10^{-6} para $n = 15$.

[5]Note o sinal $\approx$, indicando aproximação.

Detalhes do método

Apesar de parecer bastante complicado, o método é relativamente simples (a recorrência que resolvemos é que era complicada...). Agora resolveremos outros exemplos, mais simples, mas onde aparecem detalhes que podem criar problemas. À medida que estas situações forem ocorrendo, mostraremos a forma de contornar cada uma delas.

Exemplo 1

Vamos resolver algo simples:

$$S_n = \begin{cases} 0, & n = 0 \\ 2S_{n-1} + 3, & n > 0. \end{cases}$$

Listando alguns valores, temos

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_n	0	3	9	21	45	93	189	381	765	1533	3069	6141	12285	24573	49149	98301

Primeiro, produzimos a equação particular

$$S_n - 2S_{n-1} = 3.$$

Desta retiramos a equação homogênea, que é resolvida supondo-se que $S_n = ax^n$:

$$\begin{aligned} ax^n - 2ax^{n-1} &= 0 \\ ax^{n-1}(x-2) &= 0. \end{aligned}$$

A solução é $x = 2$, e portanto $S_n = a2^n$. Agora partimos para a equação particular. Nesta etapa do exemplo (10) foi feita a suposição de que $S_n = a_3 n + a_4$, porque o lado direito de (12) era um polinômio. Agora, como o lado direito é apenas uma constante, tentaremos uma constante também. Portanto, assumindo $S_n = b$,

$$\begin{aligned} b - 2b &= 3 \\ b &= -3. \end{aligned}$$

e portanto $S_n = -3$. Somando as duas soluções, temos imediatamente

$$S_n = a2^n - 3.$$

Para descobrir o valor de a devemos trabalhar com a condição inicial:

$$\begin{aligned} 0 &= a2^0 - 3 \\ a &= 3. \end{aligned}$$

Com isto temos a solução definitiva: $S_n = 3 * 2^n - 3 = 3 * (2^n - 1)$. Calcule alguns dos valores obtidos com esta fórmula e verifique que eles coincidem com os apresentados na tabela.

Exemplo 2

Desta vez vamos apelar para algo mais complicado:

$$S_n = \begin{cases} 2, & n = 0 \\ 1, & n = 1 \\ 4S_{n-2} + n, & n > 1. \end{cases}$$

Primeiro produzimos a equação particular:

$$S_n - 4S_{n-2} = n.$$

Retiramos a equação homogênea, que é resolvida supondo-se que $S_n = ax^n$:

$$\begin{aligned} ax^n - 4ax^{n-2} &= 0 \\ ax^{n-2}(x^2 - 4) &= 0. \end{aligned}$$

Para satisfazer a equação $x^2 - 4 = 0$, temos $x = \pm 2$. Então S_n tem a forma $S_n = a_1 2^n + a_2(-2)^n$. (Isto serve para mostrar que todas as raízes encontradas devem ser usadas na solução homogênea). Agora vamos para a solução particular: o lado direito é n, e somos tentados a assumir também que $S_n = a_3 n$:

$$\begin{aligned} a_3 n - 4a_3(n-2) &= n \\ -(3a_3 + 1)n + 8a_3 &= 0. \end{aligned}$$

Se fizermos isso, infelizmente não teremos mais para onde ir. Afinal, quando tentamos zerar os coeficientes teremos para um deles que $a_3 = -1/3$, e para o outro $a_3 = 0$, teremos uma contradição evidente. Para evitar o problema, temos de usar um polinômio **denso** como lado direito (ou seja, com todos os elementos). Fazemos $S_n = a_3 n + a_4$ e obtemos

$$\begin{aligned} a_3 n + a_4 - 4(a_3(n-2) + a_4) &= n \\ -(3a_3 + 1)n - 3a_4 + 8a_3 &= 0. \end{aligned}$$

Igualando cada coeficiente a zero, temos que $a_3 = -1/3$ e $a_4 = -8/9$. Portanto, temos $S_n = -n/3 - 8/9$. Em seguida somamos as duas soluções encontradas, ainda dependendo de parâmetros:

$$S_n = a_1 2^n + a_2(-2)^n - n/3 - 8/9.$$

Descobrimos os parâmetros usando as duas condições iniciais, como feito anteriormente:

$$\begin{aligned} 2 &= a_1 + a_2 - 8/9 \\ 1 &= 2a_1 - 2a_2 - 1/3 - 8/9. \end{aligned}$$

Imediatamente temos $a_1 = 2$ e $a_2 = 8/9$, e a resposta final obtida é

$$S_n = 2^{n+1} + \frac{8}{9}(-2)^n - n/3 - 8/9.$$

Este exemplo é importante por duas razões: primeiro, para mostrar que todas as raízes achadas na solução homogênea devem entrar na resposta final. Em segundo lugar, para mostrar a importância de escolher um lado direito coerente para resolver a equação particular. Especialmente, no caso do lado direito ser um polinômio deve-se usar um polinômio **denso** como S_n.

Para acabar com quaisquer dúvidas que possamos ter, um teste comparando a recorrência original e a fórmula fechada dá os mesmos resultados:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_n	2	1	10	7	44	33	182	139	736	565	2954	2271	11828	9097	47326	36403

Talvez o mais interessante de tudo seja o fato de que a solução fechada gera apenas números inteiros, mesmo tendo coeficientes fracionários.

A forma da solução particular

É interessante notar que a forma da solução particular não tem nada a ver com a ordem da recorrência, dependendo apenas da forma do lado direito segundo a tabela abaixo:

Lado direito	Solução particular
17 (constante)	a (constante)
12^n (constante na n-ésima potência)	$a12^n$ (constante na n-ésima potência)
$2^n + 12^n + 4$ (combinação linear)	$a2^n + b12^n + c$ (mesma combinação linear)
$12n^3$ (polinômio em n)	$an^3 + bn^2 + cn + d$ (polinômio denso)
$5n^3 - 2$ (polinômio em n)	$an^3 + bn^2 + cn + d$ (polinômio denso)
$7n^2 5^n$ (combinação)	$5^n(an^2 + bn + c)$ (combinação)

Exemplo 3

Este exemplo é quase idêntico ao anterior, com apenas um sinal trocado. No entanto, este sinal terá muitas consequências:

$$S_n = \begin{cases} 2, & n = 0 \\ 1, & n = 1 \\ -4S_{n-2} + n, & n > 1. \end{cases}$$

Produzimos outra vez a equação particular:

$$S_n + 4S_{n-2} = n.$$

Retiramos a equação homogênea, resolvida supondo-se que $S_n = ax^n$:

$$\begin{aligned} ax^n + 4ax^{n-2} &= 0 \\ ax^{n-2}(x^2 + 4) &= 0. \end{aligned}$$

A equação é agora $x^2 + 4 = 0$, e temos raízes complexas $x = \pm 2i$. Mesmo que estas raízes sejam complexas, S_n ainda tem a forma $S_n = a_1(2i)^n + a_2(-2i)^n$. Vamos para a solução particular: já sabemos que o polinômio deve ser denso, e fazemos $S_n = a_3 n + a_4$:

$$\begin{aligned} a_3 n + a_4 + 4(a_3(n-2) + a_4) &= n \\ (5a_3 - 1)n + 5a_4 - 8a_3 &= 0. \end{aligned}$$

Igualando cada coeficiente a zero, temos que $a_3 = 1/5$ e $a_4 = 8/25$. Portanto, temos $S_n = n/5 + 8/25$. Em seguida somamos as duas soluções encontradas, ainda dependendo de parâmetros:

$$S_n = a_1(2i)^n + a_2(-2i)^n + n/5 + 8/25.$$

Descobrimos os parâmetros usando as duas condições iniciais, como feito anteriormente:

$$\begin{aligned} 2 &= a_1 + a_2 + 8/25 \\ 1 &= 2a_1 i - 2a_2 i + 1/5 + 8/25. \end{aligned}$$

Logo obtemos $a_1 = (21 - 3i)/25$ e $a_2 = (21 + 3i)/25$, e a resposta final obtida é

$$S_n = \frac{21 - 3i}{25}(2i)^n + \frac{21 + 3i}{25}(-2i)^n + n/5 + 8/25.$$

Este exemplo mostra que mesmo raízes complexas da equação característica devem ser incluídas na solução definitiva. Apesar desta aparência ameaçadora, ao calcularmos a recorrência as partes complexas se cancelam e a resposta é sempre um número real:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_n	2	1	−6	−1	28	9	−106	−29	432	125	−1718	−489	6884	1969	−27522	−7861

Exemplo 4

Vamos complicar mais um pouco:

$$S_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ 4S_{n-1} - 4S_{n-2} + n^2 + 1, & n > 1. \end{cases}$$

A equação particular é

$$S_n - 4S_{n-1} + 4S_{n-2} = n^2 + 1$$

e a equação homogênea é

$$S_n - 4S_{n-1} + 4S_{n-2} = 0.$$

Para resolvê-la, assumimos mais uma vez que $S_n = ax^n$, e vamos em frente:

$$\begin{aligned} ax^n - 4ax^{n-1} + 4ax^{n-2} &= 0 \\ ax^{n-2}(x^2 - 4x + 4) &= 0. \end{aligned}$$

Resolvendo a equação mais uma vez, descobrimos que a solução é $x = 2$, uma raiz dupla. No caso de raízes duplas (ou de multiplicidade maior), **não** podemos fazer $S_n = a2^n$. Pela raiz ser dupla, ela será colocada duas vezes e uma delas será multiplicada por n. Então, $S_n = a_1 2^n + a_2 2^n n$. Estamos prontos para atacar a equação particular, e como o lado direito é um polinômio de grau 2, assumiremos o mesmo tipo de valor para S_n, ou seja, $S_n = bn^2 + cn + d$. Temos

$$\begin{aligned} bn^2 + cn + d - 4(b(n-1)^2 + c(n-1) + d) + 4(b(n-2)^2 + c(n-2) + d) &= n^2 + 1 \\ (b-1)n^2 + (c - 8b)n + 12b - 4c + d - 1 &= 0. \end{aligned}$$

Igualando os coeficientes de um lado e os do outro (que são todos zero) temos $b = 1$, $c = 8$ e $d = 21$. Somando as duas soluções encontradas, teremos

$$S_n = a_1 2^n + a_2 2^n n + n^2 + 8n + 21.$$

Em seguida achamos os valores de a_1 e a_2 usando as condições iniciais dadas:

$$\begin{aligned} 0 &= a_1 + 21 \\ 1 &= 2a_1 + 2a_2 + 30. \end{aligned}$$

É imediato que $a_1 = -21$ e $a_2 = 13/2$ e já podemos montar a solução definitiva:

$$\begin{aligned} S_n &= -21 * 2^n + \frac{13}{2} * 2^n n + n^2 + 8n + 21 \\ &= 2^n \left(\frac{13n}{2} - 21 \right) + n^2 + 8n + 21 \\ &= 2^{n-1}(13n - 42) + n^2 + 8n + 21. \end{aligned}$$

Neste exemplo, temos de notar que raízes duplas **não** podem ser unificadas na expressão para S_n. Na verdade, elas formam uma situação especial: se tivermos uma raiz r de multiplicidade k esta deve entrar na solução como

$$a_1 r^n + a_2 r^n n + a_3 r^n n^2 + \cdots + a_k r^n n^{k-1} = r^n(a_1 + a_2 n + \cdots + a_k n^{k-1}).$$

Fazendo testes entre as duas versões da recorrência, temos os mesmos valores:

n	0	1	2	3	4	5	6	7	8	9	10	11	12
S_n	0	1	9	42	139	454	1257	3262	8085	19374	45257	103654	233733

Recorrências de história completa

Algumas recorrências podem usar *todos* os valores anteriores, e não apenas alguns deles. Estas recorrências são chamadas de recorrências de história completa, pois precisa-se conhecer toda a história da recorrência para calcular um novo valor. Os métodos apresentados não servem para estes casos, pois é fácil notar que cada valor a ser calculado tem um número diferente de valores iniciais (todos os já calculados até ele). Embora a solução geral destas recorrências seja muito complexa e fuja ao escopo desta disciplina, algumas recorrências de história completa podem ser resolvidas com um pouco de manipulação e imaginação. A seguir são mostrados alguns exemplos que podem (e devem) ser resolvidos:

$$T_n = \begin{cases} 1, & n = 1 \\ 1 + \sum_{i=1}^{n-1} T_i & n > 1 \end{cases} \qquad T_n = \begin{cases} 1, & n = 1 \\ n^2 + \sum_{i=1}^{n-1} T_i & n > 1 \end{cases}$$

$$T_n = \begin{cases} 1, & n = 1 \\ k + \sum_{i=1}^{n-1} T_i & n > 1 \end{cases} \qquad T_n = \begin{cases} 1, & n = 1 \\ 1 + 2\sum_{i=1}^{n-1} T_i & n > 1 \end{cases}$$

Perguntas

1. O que acontece com a equação característica se a recorrência usar três, quatro, cinco ou seis termos anteriores?

2. Monte a recorrência e determine uma fórmula fechada para os números de Fibonacci (F_n). Analise-a e determine uma aproximação para valores grandes de n. Teste sua aproximação.

3. Seja U_n o número de bits 1 quando n é escrito em binário. Por exemplo, aqui vão alguns exemplos de valores de U_n:

n	$(n)_2$	U_n
2	10	1
5	101	2
9	1001	2
19	10011	3

   Com estes dados monte uma recorrência para avaliar U_n, mas não tente resolvê-la pois esta recorrência não tem solução fechada. Pode ser que sua recorrência tenha de usar regras diferentes das habituais para poder ser escrita.

4. Resolva a recorrência abaixo, sabendo que existe mais de uma maneira de resolvê-la. Pense primeiro e ache a forma mais fácil.

$$D_n = \begin{cases} 1, & n = 0 \\ 4 - D_{n-1}, & n > 0 \end{cases}$$

5. Resolva a recorrência abaixo:

$$R_n = \begin{cases} 1, & n = 0 \\ 2R_{n-1} + 2^n, & n > 0 \end{cases}$$

6. Sejam as duas recorrências abaixo:

$$B_n = \begin{cases} 2, & n = 1 \\ 4 * B_{n-1} + 2, & n > 1 \end{cases} \qquad C_n = \begin{cases} 1, & n = 1 \\ 4 * C_{n-1} + 1, & n > 1. \end{cases}$$

Resolva as duas recorrências da maneira que achar mais fácil, depois mostre que $B_n + C_n = 2^{2n} - 1$ (existe mais de uma maneira de mostrar este fato, e até mesmo pensar em binário ajuda.). Em seguida mostre que se D_n for definido segundo a recorrência abaixo, então $D_n = B_n + C_n$:

$$D_n = \begin{cases} 3, & n = 1 \\ 4 * D_{n-1} + 3, & n > 1. \end{cases}$$

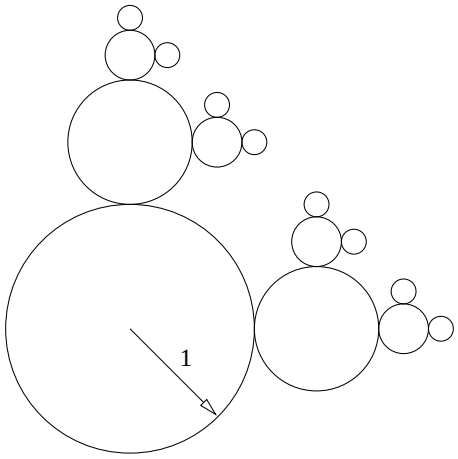
7. Você tem uma subrotina que recebe um número n e gasta $2n$ operações aritméticas, para em seguida chamar a si mesma duas vezes com parâmetro $n - 1$. A recursão termina quando $n = 1$, e neste caso são gastas 21 operações aritméticas. A partir disso, resolva:

   (a) Monte a recorrência e determine uma fórmula fechada para o número de operações em função de n;

   (b) Refaça o problema se a rotina tem de ser modificada para chamar a si mesma três vezes;

   (c) Refaça o problema se agora a rotina não gasta mais $2n$ operações, mas sim k operações a cada passo;

8. Resolva o problema do ataque dos Mickeys. Neste problema o Mickey M_1 é um simples círculo de raio 1. O Mickey M_2 é o mesmo círculo, mas com outros dois anexados acima e à direita, tendo a metade do raio e assim sucessivamente. A figura abaixo, por exemplo, mostra M_4.



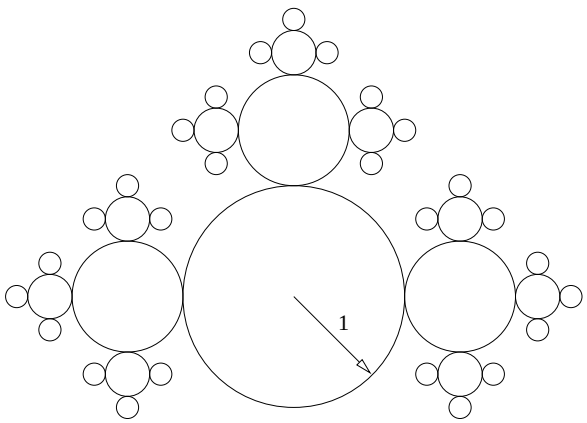
   A partir daí determine as seguintes quantidades:

   (a) C_n, o número de círculos que compõem M_n.

   (b) A_n, que é a área do Mickey M_n.

   (c) P_n, que é o perímetro desse mesmo Mickey.

9. Resolva o problema do ataque dos Mickeys Mutantes: M_1 ainda é um círculo de raio 1, mas M_2 é o mesmo círculo com outros três círculos anexados acima, à direita e à esquerda, tendo a metade do raio e assim sucessivamente. A figura abaixo mostra M_4.



   A partir daí determine as seguintes quantidades:

   (a) C_n, o número de círculos que compõem M_n.

   (b) A_n, que é a área do Mickey M_n.

   (c) P_n, que é o perímetro desse mesmo Mickey.

10. Você recebeu uma corrente pela paz mundial que diz assim: "Mande esta mensagem para outras cinco pessoas dentro de no máximo três dias, e envie um real para a ONU poder pagar suas missões pelo mundo". (A corrente foi iniciada pelo secretário geral da ONU, que descobriu que o orçamento está apertado e mandou a primeira carta em primeiro de janeiro). Partindo daí, continue:

    (a) Monte uma recorrência para o problema, para avaliar o número de pessoas que tem a mensagem num dado instante. Assuma que ninguém recebe a mensagem mais de uma vez.

    (b) Monte uma nova situação onde as pessoas acabam sempre recebendo em média 4 mensagens de seus amigos, conhecidos, vizinhos, etc. Você deve perceber um problema para os valores iniciais. Explique-o.

    (c) Algumas pessoas não conseguem lembrar de cinco amigos para repassar a corrente. Refaça a recorrência em (10a) supondo que as pessoas em média se lembram de k amigos.

    (d) Se a população do planeta é de 5.5 bilhões de habitantes, quanto tempo leva para que todos recebam a corrente? Faça o cálculo para as situações (10a) e (10c).

    (e) Calcule aproximadamente quanto dinheiro a ONU recebe nos dois casos, assumindo que as pessoas só mandam dinheiro uma vez.

    (f) Se esta corrente fosse mandada em mails de tamanho 2k, quanto espaço estaria sendo ocupado por ela a cada instante? (Assuma que os mails não são deletados.) Há diferença para os casos (10a) e (10c)?

11. Você foi mandado pela Federação para colonizar o planeta Zovirax III, e sua nave recebeu vários mísseis fertilizantes. Estes mísseis se abrem em pleno ar e despejam 12 hiperminhocas sobre o planeta. As minhocas vão comendo tudo o que encontram (rocha, areia, etc.) e deixando o planeta em condições de sustentar vida vegetal para limpar o ar e manter futuros colonizadores. O plano é mandar um míssil por mês para o planeta, mas você também sabe que as minhocas se reproduzem a cada mês de vida (cada minhoca gera mais outra minhoca). A Federação acredita que Zovirax III esteja habitável quando houverem 23.200.000 minhocas no planeta. Seu problema é prever em quantos meses isto acontecerá. Não procure soluções fechadas, mas ache a formulação adequada e resolva através de programação.

12. Você sai em um cruzeiro pelo mundo e deixa um saldo devedor de R$ 1.00 no seu cartão de crédito. A administradora cobra 8% de juros ao mês sobre o saldo devedor, mais uma taxinha de administração de R$ 2.00 mensais. Ao voltar bronzeadinho dois anos depois, quanto você está devendo? E se você ficou mais dois anos na Índia pra meditar?

13. Este é um caso real: um programa de computador precisava armazenar uma tabela de $2^{28} + 1$ números inteiros, cada um com 2 bytes. Desta forma, o programa precisava de mais de 1Gb somente para armazenar estes números. A tabela é inicializada pelo seguinte código:

```
ELEM = 2^28;
short int tab[ELEM+1]; // inteiros de 2 bytes!
tab[0] = tab[1] = 1;
for(i=2; i<ELEM; i+=2) {
     tab[i] = tab[i/2];
     tab[i+1] = tab[i/2] + tab[i/2 + 1];
    }
tab[ELEM] = 1;
```

Seu problema é adaptar o código para rodar em uma máquina que tenha apenas 64Mb de memória. Se você fez bem feitinho, pode mostrar também que seu algoritmo pode ir até mais do que 2^{28} se for preciso.

14. Resolva a recorrência abaixo, em função de c e d, e aproveite para verificar como S_n se comporta no caso especial $c = 1$, $d = 2$:

$$S_n = \begin{cases} c, & n = 1 \\ d, & n = 2 \\ S_{n-2} + n, & n > 2 \end{cases}$$

15. Você acidentalmente infectou um arquivo com um novo vírus de computador, que age de forma estranha: durante três dias ele não faz nada. No quarto dia ele escreve uma mensagem muito mal-educada na tela e em seguida se espalha em outros três arquivos ainda não contaminados. Depois de fazer isso, ele se retira do arquivo original mas também pendura sua máquina, pra você ser mais cuidadoso da próxima vez e deixar de usar sistemas que são malfeitos e penduram por qualquer coisa. Agora responda:

    (a) Quantos arquivos infectados estarão por aí depois de um mês (por **sua** culpa)?

    (b) Durante este mês, quantas vezes alguém teve de parar seu trabalho porque sua máquina pendurou (de novo por **sua** culpa)?

    (c) Qual o prejuízo que você causou se cada máquina fica parada 20 minutos para detecção e limpeza e o custo da máquina parada é R$ 1.00/hora, o custo da pessoa que vai fazer a limpeza é de R$ 10.00/hora e o custo do trabalho não feito é R$ 2.00/hora?

    (d) Como isto tudo muda se o vírus se espalha em cinco arquivos?

    (e) Se você fez suas contas, deve ter descoberto que o prejuízo é bastante grande. Por que estes valores não parecem coerentes? O que acontece na vida real que foi desconsiderado no seu modelo? Os fatores não considerados podem ser incluídos em seu modelo da situação?

16. Você tem um armazém onde são guardadas peças para máquinas. As prateleiras tem três metros de comprimento e as peças estocadas são guardadas em caixas de três tamanhos diferentes: caixas de 10 cm, de 20 cm e de 30 cm. De quantas formas você pode preencher uma prateleira com estas caixas? (Nota: se preferir, **não** resolva a recorrência pelos métodos apresentados, pois o polinômio característico tem raízes complicadas. Em vez disso escreva a recorrência, programe-a e calcule a resposta, que tem doze dígitos.)

17. Quantos números escritos em base decimal com n ou menos dígitos tem o dígito 3? Determine uma recorrência em função do número de dígitos e resolva-a. Conclua a partir dela que, para n grande o bastante, praticamente **todos** os números vão conter um 3. (Você pode até mostrar que com $n = 44$ dígitos mais de 99% dos números contém um 3, e que com $n = 66$ são mais de 99,9%). Refaça os cálculos usando seus conceitos de probabilidade (arranjos, combinações, etc.), e verifique se há coincidência.

18. Você acaba de criar uma nova bactéria que come lixo, para livrar o mundo de milhões de metros cúbicos de lixo malcheiroso. Suas bactérias se dividem todos os dias, portanto dobrando a cada dia o número de bactérias existentes no meio ambiente. Além disso, você consegue criar mais uma no laboratório todos os dias, soltando-a no meio ambiente. Infelizmente as bactérias vivem apenas três dias, morrendo no quarto dia. Nestas circunstâncias, determine:

    (a) Uma recorrência para o número de bactérias a cada dia.

(b) Um programa calculando a quantidade de bactérias a cada dia, já que a fórmula fechada é bastante complicada. Voce terá de usar um software como Maple para esta tarefa, devido aos números muito grandes.

(c) Se cada bactéria come 1 miligrama de lixo por dia, quanto tempo suas bacteriazinhas levarão para comer todo o lixo do mundo, imaginando que seja um trilhão de toneladas? Quanto tempo levaria se o lixo no planeta fosse equivalente à massa da Lua, ou seja, perto de 7×10^{22} kg?

19. Resolva a recorrência abaixo (sabendo que o método do polinômio característico não pode ser usado, e portanto você terá de resolvê-la de outra forma):

$$Q_n = \begin{cases} \alpha, & n = 0 \\ \beta, & n = 1 \\ (1 + Q_{n-1})/Q_{n-2}, & n > 1 \end{cases}$$

Referências

[1] Brassard, G.; Bratley, P.: "**Fundamentals of Algorithmics**", Prentice Hall, Englewood Cliffs, NJ, 1996. Registro na biblioteca: 004.0151 B823f.

[2] Cormen, T. H.; Leiserson, E. C.; Rivest, R. L.: **"Introduction to Algorithms"**. Mc-Graw Hill Book Co., The MIT Electrical Engineering and Computer Science Series, Cambridge, 1990. Registro na biblioteca: 005.1 C811{i,ib,ic}.

[3] Graham, R. L.; Knuth, D. E.; Patashnik, O.: **"Concrete mathematics"**. Addison-Wesley, Reading, 1989. Registro na biblioteca: 004.0151 G741c.

[4] Moret, B. M.; Shapiro, H. D.: "**Algorithms from P to NP: design and efficiency**". Addison-Wesley, Reading, 1990. Registro na biblioteca: 005.1 M844a.

[5] Lueker, G. S.: "*Some techniques for solving recurrences*". Computing Surveys, Vol. 12, no. 4, dezembro 1980.