Universidade Federal de Goiás
Instituto de Informática

INF0286 – Algoritmos e Estruturas de Dados 1

Lista 2 – Recursividade

Prof. Dr. Aldo André Díaz Salazar

1 (⋆) Números naturais

Os números naturais são os que utilizamos para contagem:

$$\mathbb{N} = \{1, 2, 3, \dots\}.$$

Estes números podem ser expressos recursivamente da seguinte maneira:

$$\begin{aligned} n_1 &= 1 \\ n_{k+1} &= n_k + 1, \qquad k \in \{1, 2, \dots\} \end{aligned}$$

Ecreva um programa para mostrar os n primeiros números naturais de forma recursiva.

Entrada

A quantidade de números naturais a mostrar, $n \leq 5000$.

Saída

Os n primeiros números naturais.

Tabela 1: Exemplo – Números naturais.

Entrada	Saída
10	1 2 3 4 5 6 7 8 9 10
7	1 2 3 4 5 6 7

2 (⋆) Fibonacci recursivo

Aos 32 anos, Fibonacci publicou em 1202 *Liber Abaci* (Livro do Ábaco), um livro de receitas a respeito de como realizar cálculos e que foi o responsável pela disseminação dos números hindu-arábicos na Europa. Num trecho desta obra, é introduzida a "sequência de Fibonacci" por meio de um problema envolvendo coelhos:

"*Iniciando com um par de coelhos macho e fêmea, depois de um mês eles se tornam sexualmente adultos e produzem um par de filhotes, também um macho e uma fêmea. Um mês depois, estes*

coelhos se reproduzem e geram outro par macho-fêmea, os quais, por sua vez, também gerarão outro par macho-fêmea depois de um mês. A questão é, depois de um ano, quantos coelhos haverá?"

A resposta ao problema é obtida por meio do uso da sequência de Fibonacci, definida como:

$$\begin{aligned} s_0 &= 0 \\ s_1 &= 1 \\ s_k &= s_{k-1} + s_{k-2} \qquad , k \in \mathbb{N},\ k \geq 2 \end{aligned}$$

Usando o conceito de recursividade, elabore um programa para imprimir os primeiros n termos da Série de Fibonacci.

Entrada

O número de termos desejados, $1 \leq n \leq 1000$.

Saída

Os termos da série.

Tabela 2: Exemplo – Fibonacci recursivo.

Entrada	**Saída**
`1`	`0 1`
`15`	`0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610`

(⋆⋆) Provocação: Será possível projetar uma função que, dada a posição de um termo na sequência de Fibonacci, retorne seu valor sem a necessidade de calcular todos os anteriores?

3 (⋆) Função de Ackermann

Uma das versões mais comuns da *função de Ackermann* é definida da seguinte maneira:

$$f_{ack}(m,n) = \begin{cases} (n+1) & , \text{se } m = 0 \\ f_{ack}(m-1,1) & , \text{se } n = 0,\ m > 0 \\ f_{ack}(m-1, f_{ack}(m,n-1)) & , \text{se } n > 0,\ m > 0 \end{cases}$$

com m e n inteiros. Implemente sua própria versão desta função. *Dica:* Lembre-se dos passos estratégicos para projetar funções recursivas. Começe testando a função numericamente, no papel, para alguns valores de exemplo.

Entrada

Os parâmetros da função de Ackermann, m e n.

Saída

O valor de f_{ack}.

Tabela 3: Exemplo – Função de Ackermann.

Entrada	Saída
0 7	8
3 0	5
3 2	29
2 4	11

4 (⋆) Capicua

Todo número natural positivo possui um *capicua* ou número reverso correspondente. Por exemplo, o capicua de 321 é 123. Escreva uma função recursiva que seja capaz de determinar o capicua de um número.

Entrada

Um número natural positivo, $n \leq 10^6$.

Saída

O capicua de n.

Tabela 4: Exemplo – Capicua.

Entrada	Saída
411	114
1230	321
138000	831
2 4	11

5 (⋆⋆) Bits

Escreva um programa que converta um número natural para sua notação binária utilizando recursão.

Entrada

- O número de casos de teste a serem avaliados, $k \leq 1000$.
- O números naturias, em notação decimal, a serem convertidos.

Saída

A representação binária de cada número.

Tabela 5: Exemplo – Bits.

Entrada	Saída
5 1 2 3 4 5	1 10 11 100 101
3 321 753 255	101000001 1011110001 11111111
1 373728	1011011001111100000

6 (⋆⋆) Fatorial duplo

Seja função *fatorial duplo*, $f(n)$, com $n \in \mathbb{N}$, definida como o produto dos números ímpares de 1 até n, incluindo o último quando ímpar. Por exemplo:

$$
\begin{aligned}
f(1) &= 1 \\
f(2) &= f(1) \\
f(3) &= f(1) \times 3 = 3 \\
f(4) &= f(3) = 3 \\
f(5) &= f(3) \times 5 = 15 \\
f(6) &= f(5) = 15 \\
f(7) &= f(5) \times 7 = 105 \\
&\vdots
\end{aligned}
$$

Escreva uma função recursiva que implemente a função fatorial duplo.

Entrada

O argumento da função, $n \leq 100$.

Saída

O valor de $f(n)$.

Tabela 6: Exemplo – Fatorial duplo.

Entrada	Saída
1	1
7	105
10	945

7 ($\star\star\star$) Banco inteligente

Os caixas automáticos são muito úteis mas, às vezes precisamos de dinheiro trocado e a máquina nos entrega notas de $R\$\,100,00$. Noutras vezes, desejamos sacar um valor maior e gostaríamos de receber notas de $R\$\,100,00$, mas a máquina nos entrega um monte de notas de $R\$\,20,00$.

Para conquistar clientes, o `Banco Inteligente` (`BI`) está tentando minimizar este problema dando aos clientes a possibilidade de escolher o valor das notas na hora do saque. Para isso, eles precisam da sua ajuda para saber a resposta para a seguinte questão:

"Dado um valor de saque e a quantidade de notas de cada valor que a máquina tem, quantas maneiras distintas há para entregar o valor ao cliente?"

Sabe-se que nos caixas do `BI` há escaninhos para notas de 2, 5, 10, 20, 50 e de 100 reais. Por exemplo, suponha que um cliente deseja retirar $R\$\,22,00$ e que a quantidade de notas N_k de cada valor k presente no caixa momento do saque é:

$$N_2 = 5$$
$$N_5 = 4$$
$$N_{10} = 3$$
$$N_{20} = 10$$
$$N_{50} = 0$$
$$N_{100} = 10$$

Então, há 4 maneiras distintas de entregar o valor do saque solicitado:

1 : 1 nota de $R\$\,20,00$ e 1 nota de $R\$\,2,00$ (2 notas)

2 : 2 notas de $R\$\,10,00$ e 1 nota de $R\$\,2,00$ (3 notas)

3 : 1 nota de $R\$\,10,00$, 2 notas de $R\$\,5,00$ e 1 nota de $R\$\,2,00$ (4 notas)

4 : 4 notas de $R\$\,5,00$ e 1 nota de $R\$\,2,00$ (5 notas)

Escreva um programa que determine as maneiras possíveis de atender uma solicitação de saque.

Entrada

- O valor do saque desejado, $x \leq 5000$.
- O número de notas de 2, 5, 10, 20, 50 e 100 reais disponíveis no caixa, $N_k \leq 500$.

Saída

A quantidade de maneiras distintas de atender ao saque solicitado.

Tabela 7: Exemplo – Banco inteligente.

Entrada	Saída
22 5 4 3 10 0 10	4
1000	

20 20 20 20 20 20	34201
50 1 1 1 1 0 1	0
50 2 2 2 2 2 2	4

8 ($\star\star\star$) Tróia

A *Guerra de Tróia* foi um grande conflito entre gregos e troianos, possivelmente ocorrido entre 1.300 a.C. e 1.200 a.C. Recentemente, foram encontradas inscrições numa caverna a respeito de sobreviventes deste conflito. Arqueólogos descobriram que as incrições descreviam relações de parentesco numa certa população daquela região. Cada item da inscrição indica que as duas pessoas pertenciam a uma mesma família.

O problema dos arqueólogos é determinar quantas famílias distintas existiam naquela população. Eles desejam fazer isto "automaticamente", e para isso contratam a você como programador para realizar estas atividades de agrupamento.

Entrada

- O número de pessoas na população, $1 \leq n \leq 5 \times 10^4$.
- A quantidade de registros a ingressar, $1 \leq m \leq 10^5$.
- Os pares de pessoas daquela população. Cada linha indica que as duas pessoas pertenciam a uma mesma família.

Saída

A contagem do número de famílias identificadas naquela população.

Tabela 8: Exemplo – Tróia.

Entrada	**Saída**
4 4 1 2 2 3 3 4 4 1	1
8 10 1 2 2 3 3 6 6 5 5 4 4 3 6 7 7 8	1

8 1 1 5	
5 4 1 2 2 3 4 5 5 4	2

9 ($\star\star\star$) Torre de Hanói

A *Torre de Hanói* é uma espécie de quebra-cabeça que consiste em uma base contendo três hastes. Em uma delas estão dispostos discos, um sobre o outro, em ordem decrescente de tamanho. O problema consiste em passar todos os discos de uma haste para outra qualquer usando a haste sobrante como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor, em nenhuma situação. A torre mais simples contém apenas três discos.

Faça um programa recursivo que resolva a Torre de Hanói. Suponha que os pinos se chamam "O" (origem), "D" (destino), e "A" (auxiliar).

Entrada

A quantidade de discos no pino de origem, $n \leq 1000$. A numeração dos discos indica seu tamanho.

Saída

Os movimentos a serem realizados para resolver a Torre de Hanói. Cada movimento deve estar em uma linha no formato de par ordenado `<haste de origem>, <haste de destino>`.

Tabela 9: Exemplo – Torre de Hanói.

Entrada	Saída
2	(O,A) (O,D) (A,D)
3	(O,D) (O,A) (D,A) (O,D) (A,O) (A,D) (O,D)

10 ($\star\star\star\star$) Setas

Gabriel é um garoto que gosta muito de um jogo eletrônico onde há várias letras num tabuleiro que fica sobre o piso e o jogador precisa pisar rapidamente nas letras corretas, de acordo com as

instruções que aparecem na tela de projeção que está à sua frenta, seguindo uma música ao fundo.

Cansado de vencer o "jogo", Gabriel inventou um novo: Agora temos um tabuleiro quadrado, em que cada célula possui uma seta que aponta para uma das quatro posições vizinhas (►,◄,▲,▼). Primeiro, o jogador escolhe uma célula inicial para se posicionar e, quando a música começa, ele deve caminhar na direção para onde a seta em que ele está naquele momento apontar. Quando a seta atual manda sair do tabuleiro dizemos que a célula é *insegura*, pois leva a um caminho que termina fora do tabuleiro. Ganhará o jogo quem pisar em mais setas corretas durante um determinado período de tempo previamente fixado.

A Fig. 1 mostra dois tabuleiros, um de tamanho 3×3 e outro de 4×4, com 8 e 11 células *seguras*, respectivamente:

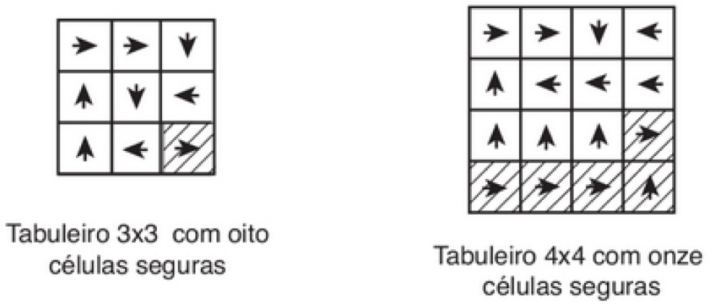

Figura 1: Tabuleiros.

As células seguras de cada tabuleiro são as seguintes:

3×3 – todas, exceto a (3, 3);

4×4 – (1,1); (1,2); (1,3); (1,4); (2,1); (2,2); (2,3); (2,4); (3,1); (3,2) e (3,3).

Sua tarefa é ajudar a Gabriel construindo um programa que indique, a partir de uma dada configuração do tabuleiro fornecida, quantas células são *seguras* para ele iniciar o jogo.

Entrada

- O tamanho do tabuleiro, $1 \leq n \leq 500$.
- As direções das setas:
  - `'v'` : Aponta para a célula da linha abaixo, na mesma coluna
  - `'<'` : Aponta para a célula à esquerda, na mesma linha
  - `'>'` : Aponta para a célula à direita, na mesma linha
  - `'a'` : Aponta para a célula da linha acima, na mesma coluna

Saída

O número de células seguras no tabuleiro.

Tabela 10: Exemplo – Setas.

Entrada	**Saída**

```
   3        8
> > v
a v <
a < >
-----------
   4       11
> > v <
a < < <
a a a >
> > > a
-----------
   4        0
v > > >
v > v <
> a > v
< < v <
-----------
   5       25
> > v < <
v > v v a
v > > > a
> > a a <
> > a > a
-----------
```

11 ($\star\star\star\star$) Batalha naval

No jogo *batalha naval*, dois jogadores possuem tabuleiros retangulares que representam o campo de batalha, onde:

- Cada posição pode conter água, ou parte de um navio.
- É proibido que navios distintos tenham lados em comum.
- Para que um navio seja destruído é necessário que o oponente acerte todas as partes dele.

O jogo consiste em "disparos" alternados entre os jogadores. Cada disparo tem como alvo um quadrado do tabuleiro do oponente. Para fazer um disparo, um jogador informa ao outro as coordenadas do alvo (linha e coluna). Considere que os jogadores lembram todos seus disparos anteriores (atiram apenas uma vez no mesmo lugar).

Escreva um programa que simule uma partida deste jogo e determine o número de navios do oponente que foram destruídos, a partir da configuração do tabuleiro e de uma sequência de disparos feitos por um dos jogadores.

Entrada

- O número de linhas e colunas do tabuleiro.
- O tabuleiro do jogo, descrito linha por linha. Se o caractere for '`a`', a posição contém água. Se o caractere for '`#`', a posição contém parte de um navio.
- O número de disparos feitos por um jogador.

- As coordenadas (linha e coluna) dos disparos.

Saída

O número de navios destruídos do oponente.

Tabela 11: Exemplo – Batalha naval.

Entrada	Saída
5 5 aa#a# #aaaa aaa#a #aaaa aaa#a 5 1 3 1 4 1 5 2 1 3 4	4
5 5 aa### aaaaa ##### aaaaa #a##a 5 5 1 5 2 1 3 1 4 1 5	2
7 7 a#aaaa# ###-a## a#aaaa# aaaa#a# a#aa#a# a####a# aaaaaaa 8 1 1 1 2 2 1 2 2 2 3 3 2	1

```
5 2
6 2
```

Provocação

Se você domina alguma biblioteca gráfica (*e.g.*, SDL, OpenGL, Unity), ou linguagem de programação que permita interfazes GUI (*Graphic User Interface*), que tal desenvolver este jogo com ela?

12 ($\star\star\star\star$) Labirinto v1.0.

Considere um jogo de labirinto como uma matriz, onde cada casa da matriz possui a coordenada da próxima casa onde você deverá ir. A saída do labirinto é sempre a posição $(0, 0)$.

Por exemplo, observando a Fig. 2 temos que você está na casa vermelha com posição $(0, 1)$, e dela você irá para a posição amarela $(1, 2)$. Da posição amarela, você irá para a posição verde $(0, 0)$, indicando que você conseguiu sair do labirinto e, portanto, venceu o jogo.

	0	1	2
0	0 , 0	1 , 2	1 , 1
1	0 , 2	2 , 2	0 , 0
2	2 , 2	0 , 0	0 , 2

Figura 2: Labirinto 1, primeiro exemplo.

No exemplo da Fig. 3, você inicia o jogo na posição $(1, 0)$. Neste caso, você sairia da posição vermelha $(1, 0)$ e iria para a posição azul $(0, 2)$. De lá, você iria para a posição amarela $(1, 1)$ e então iria para a posição verde $(2, 2)$. Da posição verde, você voltaria para a posição azul $(0, 2)$, o que caracteriza que você entrou em *looping*. Assim, partindo da posição $(1, 0)$ não é possível chegar à saída do labirinto, ou seja, é impossível ganhar o jogo a partir dela.

	0	1	2
0	0 , 0	1 , 2	1 , 1
1	0 , 2	2 , 2	0 , 0
2	2 , 2	0 , 0	0 , 2

Figura 3: Labirinto 1, segundo exemplo.

Escreva um programa para simular este jogo de acordo com as especificações a seguir.

Entrada

- As dimensões da matriz (labirinto), $m \leq 100$, $n \leq 100$.
- Os pares de coordenadas de cada célula da matriz.
- As coordenadas da posição inicial a partir de onde o jogo começará.

Saída

- A palavra `vence`, se for possível ganhar o jogo, ou seja, sair do labirinto a partir de uma certa posição inicial.
- A palavra `preso`, caso seja impossível ganhar o jogo.

Tabela 12: Exemplo – Labirinto v.1.0.

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 0 0 2 0 1	vence
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 0 0 2 1 0	preso

13 (⋆⋆⋆⋆) Labirinto v2.0

Na segunda versão, o labirinto ficou um pouco mais complicado, pois a posição inicial não é dada. Escreva um programa que dadas as dimensões do labirinto e as coordenadas de cada célula (igual que no problema anterior), seja capaz de calcular a quantidade de células onde é possível chegar à saída. Ou seja, quantas casas permitem que, iniciando-se a partir dela, seja possível ganhar o jogo. Por exemplo, no tabuleiro da Fig. 4, estão pintadas de vermelho todas as células que, iniciando nelas, atinge-se a saída. Neste caso, o programa retorna 4.

	0	1	2
0	0 , 0	1 , 2	1 , 1
1	0 , 2	2 , 2	0 , 0
2	1 , 2	1 , 0	0 , 2

Figura 4: Labirinto 2.

Entrada

- Aa dimensões da matriz (labirinto), $m \leq 100$, $n \leq 100$.
- Os pares de coordenadas de cada célula da matriz.

Saída

A quantidade de casas a partir da qual é possível alcançar a saída.

Tabela 13: Exemplo – Labirinto v2.0.

Entrada	Saída
3 3 0 0 1 2 1 1 0 2 2 2 0 0 1 2 1 0 0 2	4
3 3 0 0 1 2 1 1 0 2 2 2 0 0 2 2 0 1 1 2	9
5 5 0 0 2 0 3 0 1 0 1 1 0 4 3 1 0 2 2 4 2 1 4 4 2 3 1 3 4 2 4 1 0 0 1 4 3 4 2 2 4 3 0 1 4 0 3 2 0 3 1 2	13

14 ($\star\star\star\star$) Pegar e escapar

Considere que lhe é fornecido um "vetor" de números naturais. Escreva um programa que seja capaz de escolher k elementos deste vetor, de maneira tal que, aplicando-se a operação lógica `XOR` entre todos os elementos escolhidos, obtenha-se o máximo valor possível. Aplique a operação `XOR` utilizando a representação binária de cada número.

Entrada

- O número de casos de teste a serem avaliados, $1 \leq t \leq 100$.
- Os casos de teste, onde cada caso contem:
  - O tamanho do vetor e a quantidade de elementos a selecionar.
  - O valor de cada elemento do vetor, cujo valor está no intervalo [1, 10000].

Saída

O valor máximo obtido para a aplicação da operação `XOR` para cada caso.

Tabela 14: Exemplo – Pegar e escapar.

Entrada	Saída
2 5 3 1 2 3 4 5	7 7

```
5 3
3
4
5
7
4
```

```
2          0
10 2       1
10000
10000
10000
10000
10000
10000
10000
10000
10000
10000
10 2
0
0
0
0
0
1
1
1
1
1
```

15 ($\star\star\star\star\star$) Altas aventuras

Incentivado por um filme de animação, vovô Renato resolveu realizar seu sonho de criança, fazer sua pequena casa voar amarrada a balões cheios de gás hélio. Considere que a quantidade de gás em cada balão, n, varia de maneira discreta e que todos os balões possuem igual resistência em relação à quantidade de gás hélio que suportam. Vovô Renato comprou k balões coloridos para fazer alguns testes, com $k < n$, e começou a planejar a sua grande aventura.

A primeira tarefa é determinar qual é a quantidade máxima de gás hélio que pode ser injetada em cada balão, de maneira tal que ele não estoure. É claro que vovô Renato poderia testar todas as possibilidades de enchimento dos balões. Evidentemente este tipo de solução não é a mais apropriada.

Por exemplo, suponha $n = 5$ e $k = 2$. Nesse caso, a melhor solução seria testar o primeiro balão com a quantidade de gás hélio igual a 3. Caso o balão estoure, vovô Renato só teria mais um balão, e então teria de testar os valores $n = 1$ e $n = 2$, no pior caso, somando ao todo três testes. Caso o balão não estoure com o primeiro valor (ou seja, o valor 3 neste caso), o vovô poderia testar os valores 4 e depois 5 (ou 5 e depois 4), também somando três testes ao todo.

Escreva um programa que, dados a capacidade máxima da bomba disponível para enchimento dos balões, n, e o número de balões, k, indique "número mínimo de testes" que devem ser feitos, no pior caso, para determinar o ponto em que um balão estoura.

Entrada

Os valores de n e k, $1 < k \leq n \leq 10^6$.

Saíd

O número mínimo de testes que devem ser feitos, no pior caso, para determinar o ponto em que o balão estoura.

Tabela 15: Exemplo – Altas aventuras.

Entrada	**Saída**
5 2	3
2	0
10 2	1
20 2	5
11 5	4
13 7	4
100 19	7