SCC-211
Lab. Algoritmos Avançados

Capítulo 6
Análise Combinatória

Adaptado por João Luís G. Rosa

Organização

- Introdução
- Técnicas de Contagem Elementares
- Relações de Recorrência
- Coeficientes Binomiais
- Seqüências
- Indução Matemática
- Algumas Fórmulas Úteis

Introdução

- **Análise combinatória** pode ser definida como um ramo da matemática dedicado à contagem de elementos ou eventos discretos e de suas possíveis combinações.
- Exemplo:
  - Quantas senhas são possíveis com 8 caracteres utilizando letras maiúsculas, minúsculas e dígitos?
- Para problemas de contagem como este acima, dentre tantos outros, existem **soluções fechadas**:
  - Fórmulas matemáticas resultantes da análise combinatória.

Introdução

Nota (Skiena & Revilla, 2003; p. 129):

The judge can't look into your heart or your program to see your intentions: it only checks the results.

- **Importância da Análise Combinatória:**
  - **Para Computação em Geral:**
    - Várias classes de problemas de contagem ocorrem recorrentemente em ciência da computação e programação.
    - Pode substituir um algoritmo com tempo de execução elevado e/ou codificação complexa (p. ex. *backtracking*!!!) por uma única chamada a uma simples fórmula.
  - **Para Competições de Programação:**
    - Envolve problemas que são ideais para competições, pois ao mesmo tempo que são aparentemente complexos, admitem soluções simples, desde que vistos da forma correta.
    - Permitem em alguns casos a obtenção de *look-up tables* para soluções *off-line* de forma muito mais rápida que via força bruta.

Técnicas de Contagem Elementares

- **Contagem de Associações:**
  - Uma associação é um arranjo de *n* itens, onde cada item pode ser escolhido de uma lista de *m* valores, com repetição.
  - Por exemplo, quantas formas diferentes existem de se pintar 4 casas utilizando 3 cores?
  - Logo, o número de possíveis associações é:

$$S(n,m) = m^n$$

  - Exemplo: Existem $S(4,3) = 3^4 = 81$ associações possíveis entre 4 casas e 3 cores.

Técnicas de Contagem Elementares

- **Contagem de Associações:**
  - **Outros exemplos de associações são:**
    - **Subconjuntos:** um subconjunto é uma seleção de elementos a partir de *n* itens, onde cada elemento é selecionado uma única vez.
    - Em outras palavras, trata-se de uma seleção **sem reposição**.
    - A seleção ou não de cada um dos *n* elementos pode ser representada de forma binária: arranjo binário de *n* posições.
    - Logo, o número de possíveis subconjuntos é $S(n,2)=2^n$.
    - Exemplo: Os $S(3,2) = 2^3 = 8$ subconjuntos dos nos. 1,2,3 são:

∅, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}.

Técnicas de Contagem Elementares

- **Contagem de Associações:**
  - **Outros exemplos de associações são:**
    - **Strings:** uma string é uma seleção de *n* dentre *m* itens, onde cada item pode ser selecionado mais de uma vez.
    - Em outras palavras, trata-se de uma **seleção com reposição**.
    - A string pode ser representada como um vetor: vetor de *n* posições, cada uma podendo assumir *m* valores.
    - Logo, o número de possíveis strings é $S(n,m) = m^n$.
    - Note que o número de strings binárias de tamanho *n* é igual ao número de subconjuntos de *n* elementos. Exemplo:
      - $2^3 = 8$ subconjuntos de 3 elementos.
      - $2^3 = 8$ strings binárias de tamanho 3.

Técnicas de Contagem Elementares

- **Contagem de Associações:**
  - **Outros exemplos de associações são:**
    - **Um computador de 32 bits é capaz de endereçar quantos gigabytes de memória (considerando o barramento de endereços = tamanho da palavra)?**

$$S(32,2) = 2^{32} = 4{,}294{,}967{,}296 = 4\text{GB}$$

    - **Quantas senhas diferentes é possível criar utilizando de 8 a 10 letras ou dígitos, considerando letras minúsculas e maiúsculas?**

$$S(8,62) + S(9,62) + S(10,62) = 852{,}836{,}452{,}414{,}603{,}776$$

Técnicas de Contagem Elementares

- **Permutações:**
  - Uma permutação é um arranjo de *n* itens, onde cada item aparece exatamente uma única vez.
  - O 1o elemento do arranjo pode assumir qualquer um dos *n* itens, o 2o pode assumir *n–1* itens (qualquer um dos *n* itens menos o assumido pelo primeiro elemento), etc.
  - Logo, o número de possíveis permutações é:

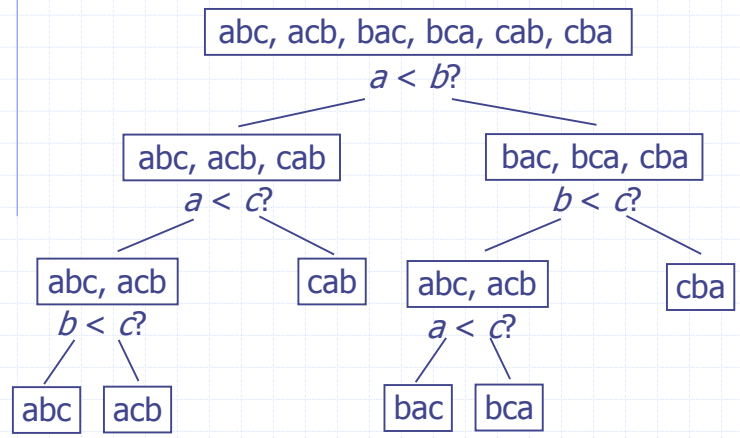
$$P(n) = n! = \prod_{i=1}^{n} i = n \times (n-1) \times \cdots \times 2 \times 1$$

  - Exemplo: As *n!* = 6 permutações dos números 1,2,3 são:
    - 123, 132, 213, 231, 312 e 321.

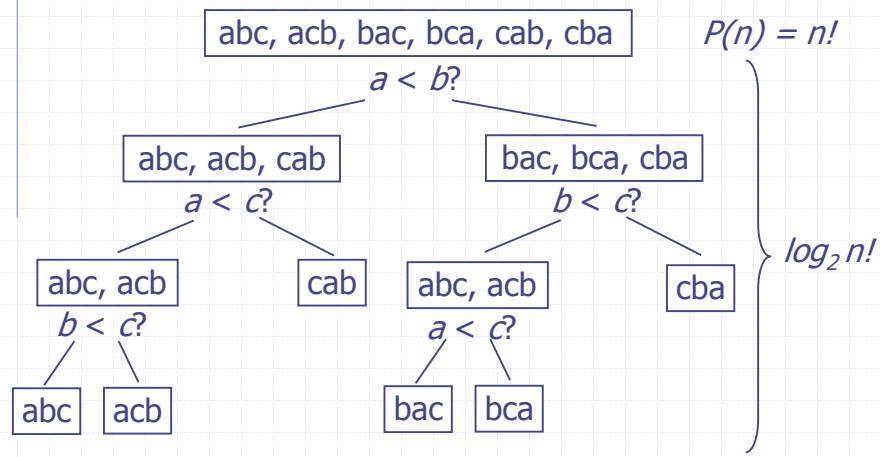
Técnicas de Contagem Elementares

- **Outros exemplos de permutações:**
  - Permutações são uma ferramenta na prova matemática que *algoritmos de ordenação de propósito geral* (ordenação baseada somente em comparações de chaves) podem levar no mínimo um tempo proporcional a *n log n* para ordenar *n* elementos.
  - Por exemplo, para um vetor de 3 elementos (*n* = 3), existem 6 possíveis permutações (*P(3)*), sendo que pelo menos uma delas é o vetor ordenado.
  - Qual delas? Um algoritmo ótimo possui a seguinte estratégia...

Técnicas de Contagem Elementares



Técnicas de Contagem Elementares



É possível mostrar que $O(\log_2 n!) = O(n \log_2 n)$

Técnicas de Contagem Elementares

- **Permutações II:**
  - Pode-se também estar interessado nas permutações de *m* dos *n* itens.
  - Por exemplo, em um campeonato com 10 equipes, quantas variações são possíveis para as 3 primeiras colocações? Existem 10 possibilidades para o primeiro colocado, 9 para o segundo e 8 para o terceiro, portanto 10 x 9 x 8 = 720.

$$P(n,m) = \frac{n!}{(n-m)!}$$

  - Observe que $P(n) = P(n,n)$.

Técnicas de Contagem Elementares

- **Outros exemplos de permutações II:**
  - Quantas senhas de 8 letras é possível construir com 62 possíveis caracteres (letras maiúsculas e minúsculas e números), se nenhuma letra pode ocorrer mais de uma vez?

$$P(62,8) = \frac{62!}{(62-8)!} = 136,325,893,334,400$$

  - Como calcular P(62,8) sem causar *overflow*? Note que:

$$P(n,m) = \frac{n!}{(n-m)!} = \prod_{i=n-m+1}^{n} i = n \times (n-1) \times \ldots \times (n-m+1)$$

Técnicas de Contagem Elementares

- **Combinações:**
  - Pode-se estar interessado em selecionar *m* de *n* itens, mas não há interesse na ordem desses *m* itens.
  - No exemplo do campeonato, pode-se querer saber os três primeiros colocados, sem ter interesse em saber a ordem dos três primeiros colocados.
  - Sabe-se que existem *P*(10,3) = 720. Sabe-se também que *P*(3) = 6. Portanto o número de combinações é 720/6 = 120.

$$\binom{n}{m} = \frac{P(n,m)}{P(m)} = \frac{n!}{(n-m)!m!}$$

Técnicas de Contagem Elementares

- **Outros exemplos de combinações:**
  - Qual é o número de mãos de *poker* diferentes que podem ser obtidas? Uma mão de *poker* possui 5 cartas de um universo de 52 cartas.
  - P(52,5) = 52!/(52-5)! = 311,875,200
  - Entretanto, não há interesse na ordem em que as cartas são recebidas. Como P(5) = 5! = 120, então

$$\binom{52}{5} = \frac{P(52,5)}{P(5)} = \frac{311{,}875{,}200}{120} = 2{,}598{,}960$$

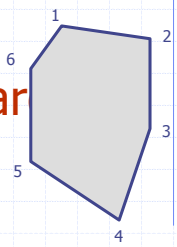
Técnicas de Contagem Elementares

- **Outros exemplos de combinações:**
  - Coeficientes da Expansão Binomial:

$$(a+b)^n = \underbrace{(a+b)\times(a+b)\times\cdots\times(a+b)}_{\text{n vezes}} = a^n + b^n + \sum_{m=1}^{n-1} c_m a^m b^{n-m}$$

    - Note que o coeficiente c_m corresponde ao número de possíveis diferentes combinações de m a's selecionados dentre os n disponíveis para compor o produto com os $n - m$ b's dos binômios restantes.
    - Exemplo: $(a+b)^2 = (a+b)(a+b) = a^2 + \boxed{2ab} + b^2$
    - Logo, por definição, tem-se: $c_m = \binom{n}{m}$

Técnicas de Contagem Elementar

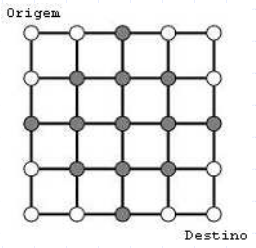


- **Outros exemplos de combinações:**
  - Triângulos em Polígonos Convexos:
    - Qual o número N de triângulos internos e compartilhando vértices com um polígono convexo de n vértices?
    - Resposta: como o polígono é convexo, qualquer trio de vértices formará um triângulo interno. Logo, tem-se:

$$N = \binom{n}{3}$$

    - Nota: todas as permutações redundantes são contadas apenas uma vez na equação acima.

Técnicas de Contagem Elem



- **Outros exemplos de combinações:**
  - Caminhos através de uma grade:
    - Quantas são as diferentes possíveis formas de caminhar em uma grade *n* x *m* a partir do canto superior esquerdo e alcançar o canto inferior direito caminhando apenas para baixo e para a direita?
    - Note que cada caminho é necessariamente constituído de um conjunto de *n* + *m* passos, *n* para baixo e *m* para a direita.
    - Necessariamente, dois caminhos distintos diferem na ordem de um ou mais dos *n* passos para baixo, dentro dos *n* + *m* passos totais.
    - Exemplo (*grid* 2x2):
      - baixo – direita – baixo – direita (ordens 1 e 3)
      - baixo – baixo – direita – direita (ordens 1 e 2)
    - Logo, a resposta é: $\binom{n+m}{n}$

Coeficientes Binomiais

- Cálculo dos Coeficientes Binomiais:
  - Pode-se calcular os coeficientes binomiais por meio da equação:

$$\binom{n}{m} = \frac{n!}{(n-m)!m!} = \frac{n \times (n-1) \times \ldots \times (n-m+1)}{m \times (m-1) \times \ldots \times 1}$$

  - No entanto, o uso desta equação para computar os coeficientes pode facilmente levar a *overflows* durante os cálculos intermediários das multiplicações.

Coeficientes Binomiais

- Cálculo dos Coeficientes Binomiais:
  - Pode-se calcular os coeficientes binomiais por meio da equação:

$$\binom{n}{m} = \frac{n!}{(n-m)!m!} = \frac{n \times (n-1) \times \ldots \times (n-m+1)}{m \times (m-1) \times \ldots \times 1}$$

```
long binominal_coefficient(n, m)
int n, m;
{
    long c;
    int i;
    c = 1;
    for (i = n; i > n-m; i--)
        c *= i;
    for (i = 2; i <= m; i++)
        c /= i;
    return c;
}
```

  - No entanto, o uso desta eq
    pode facilmente levar a *ove*
    intermediários das multiplic

Coeficientes Binomiais

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$$

- Para calcular qualquer coeficiente usando essa recorrência, precisamos de valores base conhecidos. Para tanto toma-se:

$$\binom{n}{0} = 1 \text{ maneira de escolher 0 dentre } n \text{ itens} \rightarrow \varnothing$$

$$\binom{n}{n} = 1 \text{ maneira de escolher } n \text{ dentre } n \text{ itens} \rightarrow \mathbf{U}$$

- Exemplos:

$$\binom{2}{1} = \binom{1}{0} + \binom{1}{1} = 2; \quad \binom{3}{1} = \binom{2}{0} + \binom{2}{1} = 3$$

$$\binom{3}{2} = \binom{2}{1} + \binom{2}{2} = 3; \quad \binom{4}{1} = \binom{3}{0} + \binom{3}{1} = 4$$

Coeficientes Binomiais

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$$

- A execução desse procedimento para k=0, ..., n e n=0, ... leva a uma matriz triangular conhecida como **Triângulo de Pascal**:

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
      ⋮
```

- Notas:
  - A (n+1)-ésima linha representa os coeficientes $\binom{n}{m}$ para $0 \le k \le n$.
  - As fronteiras (elementos unitários) representam os valores base.
  - Os elementos intermediários são sempre dados pela soma do elemento imediatamente acima e o seu predecessor à esquerda.
  - A propriedade $\binom{n}{m} = \binom{n}{n-m}$ fica clara observando a matriz.

Triângulo de Pascal

$$\binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

	0	1	2	3	4	5	
0	1						2^0
1	1	1					2^1
2	1	2	1				2^2
3	1	3	3	1			2^3
4	1	4	6	4	1		2^4
5	1	5	10	10	5	1	2^5

Cada número do triângulo de Pascal é igual à soma do número imediatamente acima e do antecessor do número de cima:

$$\binom{4}{2} + \binom{4}{3} = \binom{5}{3}$$

$$6 + 4 = 10$$

A soma de uma linha no triângulo de Pascal é igual a 2^n

Coeficientes Binomiais

$$\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$$

- Código para obtenção do triângulo de Pascal:

```
#define MAXN     100   /* largest n or m */

long binomial_coefficient(n,m)
int n,m;
{
    int i,j;
    long bc[MAXN][MAXN];

    for (i=0; i<=n; i++) bc[i][0] = 1;

    for (j=0; j<=n; j++) bc[j][j] = 1;

    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
            bc[i][j] = bc[i-1][j-1] + bc[i-1][j];

    return( bc[n][m] );
}
```

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
      ⋮
```


Relações de Recorrência

- Uma relação de recorrência é uma equação definida em termos de si mesma (recorrente).
- Trata-se de um conceito matemático intimamente ligado ao conceito de recursão em computação.
- Por exemplo, qualquer polinômio:
  - $p_n(x) = c_0 + c_1x + \ldots + c_nx^n$

  pode ser escrito recorrentemente (**regra de Horn**) como
  - $p_i(x) = c_{n-i} + x\,p_{i-1}(x) \;;\; p_0(x) = c_n$

  e computado recursivamente em tempo $O(n)$.

Relações de Recorrência

- Importância:
  - Muitas recorrências podem ser expressas simplificadamente através de funções:
    - Isso permite o cálculo analítico de qualquer termo de seqüências numéricas recorrentes, independente da quantidade de termos precedentes.
    - Exemplo (**Números de Fibonacci**):
      - $F_n = F_{n-1} + F_{n-2}$; $F_0 = 0$, $F_1 = 1$
      - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
      - $F_n = sqrt(5) * (((1 + sqrt(5))/2)^n - ((1 - sqrt(5))/2)^n)$

Algoritmos Avançados – Capítulo 6

Relações de Recorrência

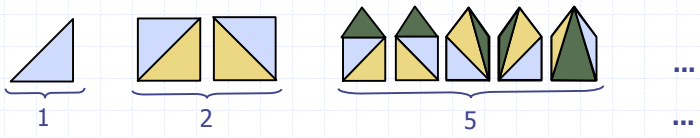
- Importância (cont.):
  - Muitas funções podem ser facilmente expressas como recorrências:
    - Exemplos:
      - $a_n = 2^n \rightarrow a_n = 2 \times a_{n-1}$, $a_0 = 1$
      - $a_n = n! \rightarrow a_n = n \times a_{n-1}$, $a_0 = 1$
  - Isso permite simplificar o cálculo computacional de algumas dessas funções.
    - Exemplos:
      - Polinômios (regra de Horn)
      - **Coeficientes Binomiais**

Seqüências

- Além da seqüência de Fibonacci vista anteriormente, existem outras seqüências numéricas de particular interesse por aparecerem em uma variedade grande de aplicações (e problemas de programação).
- **Números de Catalan**:

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{2n}{n}$$

  - Para $C_0 = 1$ tem-se C_n, n>0, como: 1, 2, 5, 14, 42, 132, 429, 1430 ...
  - Exemplo (triangulações de polígonos convexos):



Seqüências

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{2n}{n}$$

- **Números de Catalan** (cont.):
  - Quantas são as possíveis formas de construir uma fórmula balanceada com *n* pares de parênteses ?
    - Exemplo (n=3): ((())), ()(()), (())(), (()()), ()()() → 5 formas
  - Observe que o parêntesis mais à esquerda ***l*** deve fazer par com algum parêntesis direito ***r***, o que divide os demais pares em dois grupos (possivelmente nulos):
    - aqueles entre ***l*** e ***r*** : **((())**), ( )(()), **(())**(), **(()())**, ( )()()
    - aqueles à direita de ***r*** : ((())), ( )**(())**, (())**()**, (()()), ( )**()()**
  - Se a parte interna contém *k* pares, a parte à direita deve necessariamente conter (*n* – 1) – *k* pares.
  - Obviamente, ambas devem constituir fórmulas balanceadas.

Seqüências

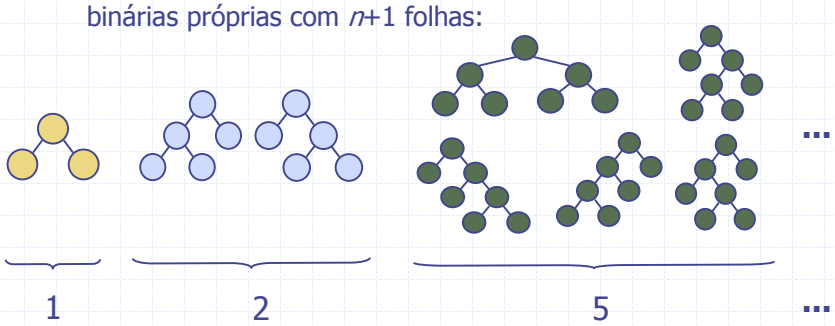
$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{2n}{n}$$

- **Números de Catalan** (cont.):
  - Definindo C_n como o no. de possíveis formas de construir uma fórmula balanceada com n pares de parênteses, tem-se que:
    - Existem C_k possíveis formas de construir a fórmula balanceada interna aos parênteses ***l*** e ***r***.
    - Analogamente, existem C_{n-1-k} possíveis formas de construir a fórmula balanceada à direita de ***r***.
    - Para cada forma interna existem C_{n-1-k} possíveis formas à direita.
    - Logo: $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
    - Obs: $C_0 = 1$ pois só existe uma forma de construir uma fórmula balanceada com 0 pares de parênteses, o nulo (conjunto vazio).

Seqüências

$$C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k} = \frac{1}{n+1}\binom{2n}{n}$$

- **Números de Catalan** (cont.):
  - Uma outra situação onde observa-se essa mesma série de números é na contagem do número de diferentes árvores binárias próprias com n+1 folhas:



  - Nota: Pode-se obter números iniciais por *backtracking*.

Algoritmos Avançados – Capítulo 6

Seqüências

$\left\langle {n \atop k} \right\rangle$

- **Números de Euler** :
  - Os números de Euler (ou Eulerianos) contam qual o número de *permutações* de *n* itens com exatamente *k ascendentes*.
    - Uma permutação é um rearranjo de uma lista ordenada de itens
    - Um **ascendente** de uma permutação $\{a_1, a_2, \ldots, a_n\}$ é um par (a_i, a_{i+1}) tal que $a_i < a_{i+1}$ (de acordo com a ordem estabelecida)
  - Exemplo:
    - As permutações de {1,2,3} são:

      {1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1}
    - As permutações com exatamente 1 ascendente são:

      {**1**,**3**,2}, {2,**1**,**3**}, {**2**,**3**,1}, {3,**1**,**2**} $\rightarrow \left\langle {3 \atop 1} \right\rangle = 4$

Seqüências

- **Números de Euler** (cont.) :
  - Pode-se demonstrar que os números de Euler respeitam a seguinte relação de recorrência ($n > 0$ e $0 < k < n-1$):

$$\left\langle {n \atop k} \right\rangle = k\left\langle {n-1 \atop k} \right\rangle + (n-k+1)\left\langle {n-1 \atop k-1} \right\rangle$$

Observe a similaridade com os Coef. Binomiais e o Triângulo de Pascal

  - Os valores base conhecidos são ($k = 0$ e $k = n-1$):

$$\left\langle {n \atop 0} \right\rangle = 1 \quad \text{(permutação descendente inversa)}$$

$$\left\langle {n \atop n-1} \right\rangle = 1 \quad \text{(permutação ascendente original)}$$

n

```
1
1  1
1  4  1
1  11  11  1
1  26  66  26  1
1  57  302  302  57  1
1  120  1191  2416  1191  120  1
```

k

Indução Matemática

- Dada uma seqüência ou relação de recorrência e valor(es) base conhecido(s), como obter uma expressão fechada para o *n*-ésimo termo?
- Por exemplo, $T_n = 2T_{n-1} + 1;\ T_0 = 0$

n	0	1	2	3	4	5	6	7
T_n	0	1	3	7	15	31	63	127

- Uma das metodologias matemáticas para solucionar esse tipo de problema é **indução**.
- Essa metodologia requer uma hipótese, usualmente obtida a partir da observação de um conjunto de valores iniciais conhecidos e, possivelmente, ajustes por tentativa-e-erro.

Exige Experiência !!!

- No exemplo acima, uma hipótese óbvia é $T_n = 2^n - 1$

n	0	1	2	3	4	5	6	7
T_n	0	1	3	7	15	31	63	127

$T_n = 2T_{n-1} + 1;\ T_0 = 0$

Indução Matemática

- **Três Passos da Prova por Indução**:
  - Mostre que a hipótese satisfaz o valor base: $T_0 = 2^0 - 1 = 0$
  - Assuma que a hipótese é válida para qualquer *n*: $T_n = 2^n - 1$
  - Use esta hipótese para generalizar para os elementos seguintes:

$$T_{n+1} = 2T_n + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 2 + 1 = 2^{n+1} - 1$$

- As razões da validade deste tipo de prova estão intimamente ligadas às razões do funcionamento de programas recursivos:
  - Garante-se o funcionamento para caso(s) base (*boundary conditions*)
  - Obtém-se o caso geral (*general conditions*) garantindo que este funciona como uma função do caso imediatamente anterior.

Indução Matemática

- **Exemplo:**
  - Seqüência: $\sum_{k=0}^{n} k = ?$
  - Caso base: $\sum_{k=0}^{0} k = 0$
  - Hipótese: $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$ ➡
    - Validade no Caso Base: $\frac{0(0+1)}{2} = 0$
  - Validade no Caso Geral:

$$\sum_{k=0}^{n+1} k = \sum_{k=0}^{n} k + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{(n+1)(n+2)}{2}$$

Algumas Fórmulas Úteis

- **Série Aritmética**: $\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$
- **Soma de Quadrados**: $\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$
- **Soma de Cubos**: $\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$

Para Saber Mais...

- Graham, R., Knuth, D. & Patashnik, O., *Concrete Mathematics*, Addison-Wesley, 1989.
- Skiena, S. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica,* Addison-Wesley, 1990.
- A Enciclopédia On-line de Sequências de Inteiros:

http://www.research.att.com/~njas/sequences/index.html?language=portuguese

- Cormen, T. H., Leiserson, C. E. & Rivest, R. L. *Introduction to Algorithms*. MIT Press, 2nd. Ed., 2001.
- ...

Self-describing Sequence

Popularidade: C, Sucess rate: high, Level: 2

Solomon Golomb's *self-describing sequence* $\langle f(1), f(2), f(3),\ldots\rangle$ is the only non-decreasing sequence of positive integers with the property that it contains exactly $f(k)$ occurrences of k for each k. A few moment's thought reveals that the sequence must begin as follows:

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
f(n)	**1**	**2**	**2**	**3**	**3**	**4**	**4**	**4**	**5**	**5**	**5**	**6**

In this problem you are expected to write a program that calculates the value of ***f(n)*** given the value of ***n***.

Self-describing Sequence

Input

The input may contain multiple test cases. Each test case occupies a separate line and contains an integer n ($1 \leq n \leq 2{,}000{,}000{,}000$). The input terminates with a test case containing a value 0 for n and this case must not be processed.

Output

For each test case in the input, output the value of $f(n)$ on a separate line.

Self-describing Sequence

Sample Input	Sample Output
100	21
9999	356
123456	1684
1000000000	438744
0	

Self-describing Sequence

- Possíveis soluções:
  - Encontrar uma equação para *f(n)*:

Self-describing Sequence

- Possíveis soluções:
  - Encontrar uma equação para *f(n)*:

$$f'(n) = \varphi^{(2-\varphi)} n^{(\varphi-1)} + E(n)$$

  - Sendo:

$$\varphi = \frac{1+\sqrt{5}}{2}$$

$$E(n) = O\left(\frac{n^{\varphi-1}}{\log n}\right)$$

Pétermann, Y.-F. S. *On Golomb's self describing sequence II*, Arch. Math., Vol. 67, 473-477 (1996).

Self-describing Sequence

n: 1, f'(n): 1.20, E(n): -----
n: 2, f'(n): 1.84, E(n): 2.21
n: 3, f'(n): 2.37, E(n): 1.79
n: 4, f'(n): 2.83, E(n): 1.70
n: 5, f'(n): 3.25, E(n): 1.68
n: 6, f'(n): 3.64, E(n): 1.69
n: 7, f'(n): 4.00, E(n): 1.71
n: 8, f'(n): 4.34, E(n): 1.74
n: 9, f'(n): 4.67, E(n): 1.77
n: 10, f'(n): 4.99, E(n): 1.80

n: 11, f'(n): 5.29, E(n): 1.84
n: 12, f'(n): 5.58, E(n): 1.87
n: 13, f'(n): 5.87, E(n): 1.90
n: 14, f'(n): 6.14, E(n): 1.94
n: 15, f'(n): 6.41, E(n): 1.97
n: 16, f'(n): 6.67, E(n): 2.00
n: 17, f'(n): 6.92, E(n): 2.03
n: 18, f'(n): 7.17, E(n): 2.06
n: 19, f'(n): 7.42, E(n): 2.10

n: 1991, f'(n): 131.45, f(n) = 132, E(n) = 14.40

Self-describing Sequence

- Possíveis soluções:
  - Encontrar uma equação para *f(n)*.
  - Encontrar uma relação de recorrência.

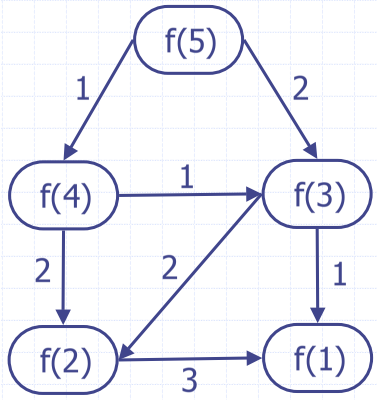
Self-describing Sequence

- Possíveis soluções:
  - Encontrar uma equação para *f(n)*.
  - Encontrar uma relação de recorrência.
    - *f(1) = 1;*
    - *f(n) = 1 + f(n-f(f(n-1))).*

Self-describing Sequence

- Possíveis soluções:
  - Encontrar uma equação para *f(n)*.
  - Encontrar uma relação de recorrência.
    - *f(1) = 1;*
    - *f(n) = 1 + f(n-f(f(n-1))).*
  - Essa relação poderia ajudar a calcular *f(n)*, mas...
    - Seria muito custoso, ou;
    - Envolveria armazenar valores de *f* menores que *n*.

Self-describing Sequence



Self-describing Sequence

```
#include <stdio.h>
#include<stdlib.h>

unsigned int sg(unsigned int n) {
    if (n == 1)
       return 1;
    else
       return 1+sg(n-sg(sg(n-1)));
}

main() {
       int n;

       for (n=1; n < 100; n++)
           printf("sg(%u) = %u\n", n, sg(n));
       system("pause");
}
```

Self-describing Sequence

```c
#include <stdio.h>
#include <stdlib.h>

unsigned int v[100];

unsigned long long sg_memoization(int n) {
    if (v[n] != 0)
       return v[n];
    if (n == 1)
       v[n] = 1;
    else
       v[n] = 1+sg_memoization(n-sg_memoization(sg_memoization(n-1)));
    return v[n];
}

main() {
       int n;

       for (n=1; n < 100; n++) v[n] = 0;
       for (n=1; n < 100; n++)
           printf("sg(%u) = %u\n", n, sg_memoization(n));
       system("pause");
}
```


Self-describing Sequence

- É possível armazenar os 2×10^9 valores de *f(n)*?

Self-describing Sequence

- É possível armazenar os 2×10^9 valores de *f(n)*?

4 bytes por inteiro $\times$ $2 \times 10^9 \approx$ 8 GB.

Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

v1:	1	2	2	3	3	4	4	4	5	5	f(n)
	1	2	3	4	5	6	7	8	9	10	n

Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

v1:	1	2	2	3	3	4	4	4	5	5	f(n)
	1	2	3	4	5	6	7	8	9	10	n

v2:	1	2	4	6	9	12	16	20	24	29
	1	2	3	4	5	6	7	8	9	10

Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

v1:	1	2	2	3	3	4	4	4	5	5	f(n)
	1	2	3	4	5	6	7	8	9	10	n

v2:	1	2	4	6	9	12	16	20	24	29	n
	1	2	3	4	5	6	7	8	9	10	f(n)

p/ i=1: v1[i] = v2[i] = 1;

p/ i > 1: v2[i] = v1[i-1]+v2[i-1]

Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

v1:

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

f(n) / n

v2:

1	2	4	6	9	12	16	20	24	29
1	2	3	4	5	6	7	8	9	10

n / f(n)

$n = 10$, $f(n) = ?$

Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

v1:

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

f(n) / n

v2:

1	2	4	6	9	12	16	20	24	29
1	2	3	4	5	6	7	8	9	10

n / f(n)

$n = 10$, $f(n) = 5$

Self-describing Sequence

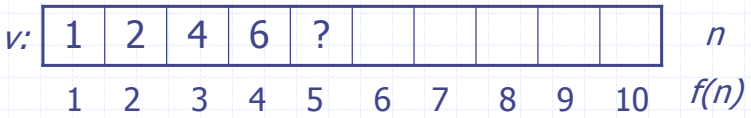
Sample Input	Sample Output
100	21
9999	356
123456	1684
1000000000	438744
0	

Self-describing Sequence

- A pergunta principal é:
  - É possível calcular todos os valores sem TLE (*time limit exception*)*?*
  - A resposta é sim, mas o algoritmo precisa ser *O(n)*.

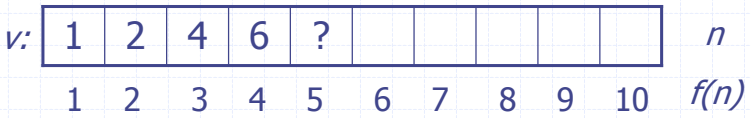
Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6



Self-describing Sequence

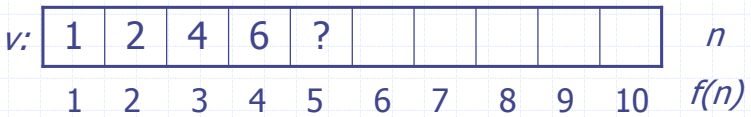
n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6



v[5] = f(4) + f(3) + f(2) + f(1) + 1

Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

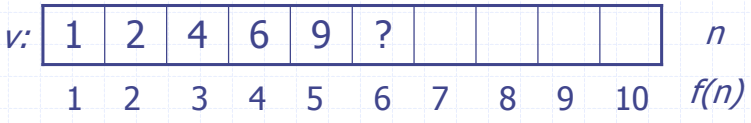


v[5] = f(4) + f(3) + f(2) + f(1) + 1

v[5] = 3 + 2 + 2 + 1 + 1 = 9

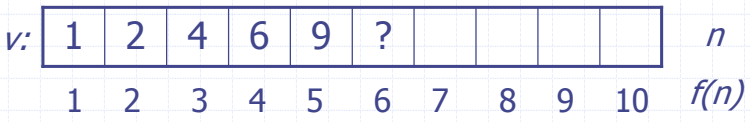
Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6



Self-describing Sequence

n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6

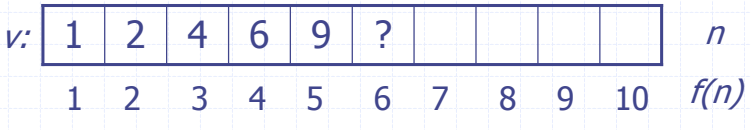


v[6] = f(5) + v[5]

v[i] = f(i-1) + v[i-1]

Self-describing Sequence

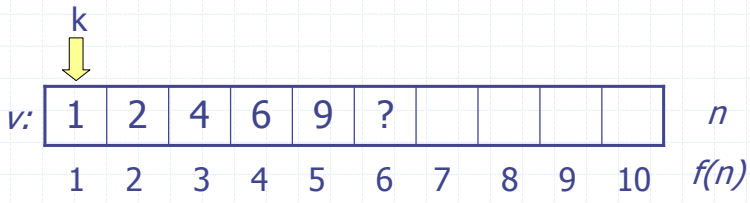
n	1	2	3	4	5	6	7	8	9	10	11	12
f(n)	1	2	2	3	3	4	4	4	5	5	5	6



v[6] = f(5) + v[5] = 3 + 9 = 12

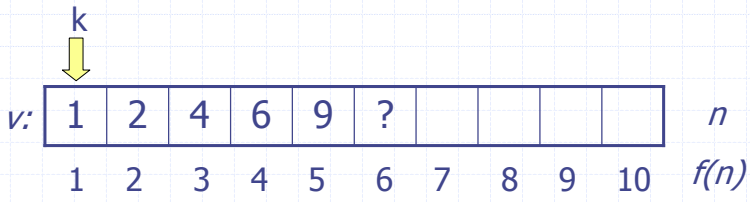
v[i] = f(i-1) + v[i-1]

Self-describing Sequence



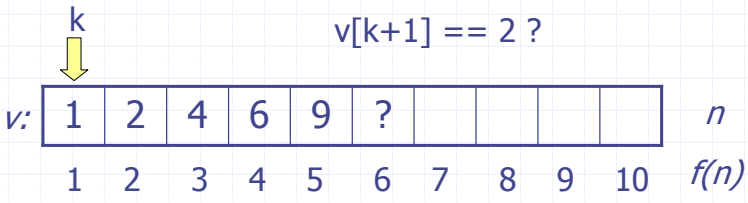
v[2] = f(1) + v[1]

Self-describing Sequence



v[2] = f(1) + v[1] = k + v[1] = 2

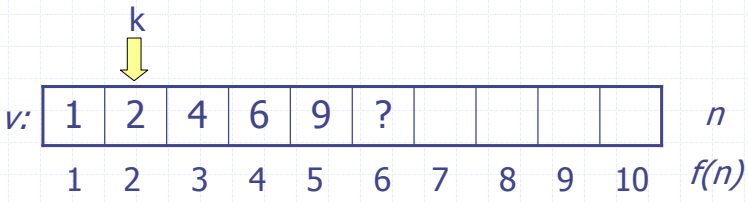
Self-describing Sequence



v[2] = f(1) + v[1] = k + v[1] = 2

v[3] = f(2) + v[2]

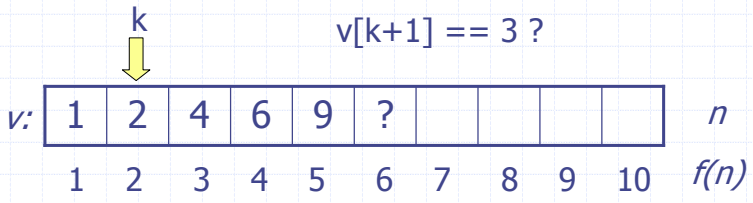
Self-describing Sequence



v[2] = f(1) + v[1] = k + v[1] = 2

v[3] = f(2) + v[2] = k + v[2] = 4

Self-describing Sequence

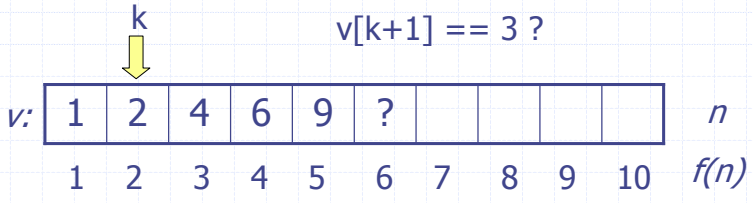


v[2] = f(1) + v[1] = k + v[1] = 2

v[3] = f(2) + v[2] = k + v[2] = 4

V[4] = f(3) + v[3]

Self-describing Sequence



v[2] = f(1) + v[1] = k + v[1] = 2

v[3] = f(2) + v[2] = k + v[2] = 4

v[4] = f(3) + v[3] = k + v[3] = 6

Exercícios para Nota

- Fibonacci Numbers (10579)
- Bits (11645)

Fibonacci Numbers (10579)

- A Fibonacci sequence is calculated by adding the previous two members of the sequence, with the first two members being both 1.

  $f(1) = 1, f(2) = 1, f(n > 2) = f(n - 1) + f(n - 2)$

- **Input and Output**
  - Your task is to take numbers as input (one per line), and print the corresponding Fibonacci number.

Fibonacci Numbers (10579)

- **Sample Input**

  3
  100
- **Sample Output**

  2
  354224848179261915075

- **Note:** No generated fibonacci number in excess of 1000 digits will be in the test data, i.e. $f(20) = 6765$ has 4 digits.

Bits (11645)

- A **bit** is a binary digit, taking a logical value of either "1" or "0" (also referred to as "true" or "false" respectively). And every decimal number has a binary representation which is actually a series of bits. If a bit of a number is "1" and it's next bit is also "1" then we can say that the number has a 1 adjacent bit. And you have to find out how many times this scenario occurs for all numbers up to **N**.

Bits (11645)

- Examples:

Number	Binary	Adjacent Bits
12	1100	1
15	1111	3
27	11011	2

- Input
  - For each test case, you are given an integer number (0 <= N <= ((2^63)-2)), as described in the statement. The last test case is followed by a negative integer in a line by itself, denoting the end of input file.

Bits (11645)

- Output
  - For every test case, print a line of the form "Case X: Y", where X is the serial of output (starting from 1) and Y is the cumulative summation of all adjacent bits from 0 to N.
- Sample Input

```
0
6
15
20
21
22
-1
```

Bits (11645)

- Output for Sample Input

```
Case 1: 0
Case 2: 2
Case 3: 12
Case 4: 13
Case 5: 13
Case 6: 14
```


Referências

- Batista, G. & Campello, R.
  - Slides disciplina *Algoritmos Avançados*, ICMC-USP, 2007.
- Skiena, S. S. & Revilla, M. A.
  - *Programming Challenges – The Programming Contest Training Manual*. Springer, 2003.
- Wikipedia
  - http://pt.wikipedia.org/wiki/Tri%C3%A2ngulo_de_Pascal