MC202 - Estruturas de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Recursão

- Um método é dito recursivo quando ele divide o problema em dois possíveis casos:
  - o caso trivial, em que uma solução é facilmente encontrada, finalizando a recursão, e
  - o caso em que o problema é subdividido em um ou mais subproblemas, cujas soluções obtidas de forma recursiva são combinadas para resolver o problema maior.

Recursão

- Um método é dito recursivo quando ele divide o problema em dois possíveis casos:
    - o caso trivial, em que uma solução é facilmente encontrada, finalizando a recursão, e
    - o caso em que o problema é subdividido em um ou mais subproblemas, cujas soluções obtidas de forma recursiva são combinadas para resolver o problema maior.
- A recursão é, portanto, uma estratégia de divisão e conquista bastante usada na solução de problemas.

Recursão

- Um método é dito recursivo quando ele divide o problema em dois possíveis casos:
  - o caso trivial, em que uma solução é facilmente encontrada, finalizando a recursão, e
  - o caso em que o problema é subdividido em um ou mais subproblemas, cujas soluções obtidas de forma recursiva são combinadas para resolver o problema maior.
- A recursão é, portanto, uma estratégia de divisão e conquista bastante usada na solução de problemas.
- A recursão está associada ao conceito de indução matemática, a qual pode ser fraca ou forte, direta ou indireta.

Agenda

Vamos estudar esses conceitos através de exemplos de problemas relacionados ao curso.

- Funções recursivas e árvores de recursão.
- Exemplo de recursão indireta.
- Ordenação de sequências de números.
- Busca binária na sequência ordenada.
- Backtracking.

Funções recursivas e árvores de recursão

- O segredo da recursão está em saber identificar o caso trivial e expressar a solução do problema em função das soluções dos subproblemas.

- O segredo da recursão está em saber identificar o caso trivial e expressar a solução do problema em função das soluções dos subproblemas.

- Considere, por exemplo, o fatorial de $n \geq 0$:

  $fat(n) = 1$ se $n \leq 1$, caso trivial.

  $fat(n) = n \times fat(n-1)$ no caso contrário.

- O segredo da recursão está em saber identificar o caso trivial e expressar a solução do problema em função das soluções dos subproblemas.

- Considere, por exemplo, o fatorial de $n \geq 0$:
  $fat(n) = 1$ se $n \leq 1$, caso trivial.
  $fat(n) = n \times fat(n-1)$ no caso contrário.

- Esta indução é fraca e direta, pois a função chama a si mesma para resolver um único subproblema.

Uma função recursiva baseada em indução fraca e direta apresenta o seguinte padrão para os casos trivial e recursivo.

```
tipo nome-da-função (<lista de argumentos>) {
   <declaração das variáveis locais>
   if (<condição de parada>) {
     <comandos finais>
     return(<resultado>)
   } else {
     <comandos iniciais>
     <chamada recursiva>
     <comandos finais>
     return(<resultado>)
   }
}
```

Note que a função pode ser simplificada de diversas formas.

```
/* Assumindo que n >= 0 */

long double fatorial_direta_fraca(unsigned long n){
  long double res; /* variável local */
  if (n <= 1) { /* condição de parada */
    res = 1.0; /* comando inicial */
    return(res);
  } else { /* n > 1 */
    /* sem comandos iniciais */
    res = fatorial_direta_fraca(n-1); /* chamada recursiva */
    res = res * n; /* comando final */
    return(res);
  }
}
```

http://www.pythontutor.com

Funções recursivas e árvores de recursão

- O mesmo problema pode ser resolvido por indução forte e direta.

Funções recursivas e árvores de recursão

- O mesmo problema pode ser resolvido por indução forte e direta.
- O fatorial de $n \geq 0$ pode resolver o caso trivial $fat(n) = 1$, se $n \leq 1$, ou calcular o produto $1 \times 2 \times \ldots n$ por recursão forte e direta, para $n > 1$.

- O mesmo problema pode ser resolvido por indução forte e direta.
- O fatorial de $n \geq 0$ pode resolver o caso trivial $fat(n) = 1$, se $n \leq 1$, ou calcular o produto $1 \times 2 \times \ldots n$ por recursão forte e direta, para $n > 1$.
- Neste caso, $produto(x_1, x_n) = x_1 x_2 \ldots x_n$ fica $produto(x1, x_n) = x_n$, se $x_1 = x_n$ (caso trivial), ou $produto(x1, x_n) = produto(x_1, \frac{x_1+x_n}{2}) \times produto(\frac{x_1+x_n}{2} + 1, x_n)$, para $x_1 < x_n$.

Funções recursivas e árvores de recursão

No caso da indução forte e direta com divisão em dois subproblemas, o padrão fica assim.

```
tipo nome-da-função (<lista de argumentos>) {
  <declaração das variáveis locais>
  if (<condição de parada>) {
    <comandos finais>
    return(<resultado>)
  } else {
    <comandos iniciais>
    <chamada recursiva subproblema 1>
    <chamada recursiva subproblema 2>
    <comandos finais>
    return(<resultado>)
  }
}
```

Note que a função pode ser simplificada de diversas formas.

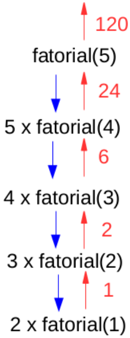
```c
/* Assumindo que n >= 0 */

long double fatorial(unsigned long n)
{
  if (n <= 1) { /* caso trivial */
    return(1.0);
  } else { /* n > 1 */
    return(produto_direta_forte(1,n));
  }
}
```

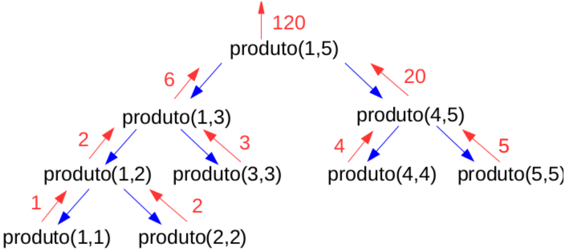
```c
/* Assumindo que x1 <= xn */

long double produto_direta_forte(unsigned long x1, \
                                 unsigned long xn)
{
  unsigned long xm; /* variáveis locais */
  long double res1, res2;

  if (x1 == xn){ /* condição de parada */
    return(xn);
  } else { /* x1 < x n */
    xm   = (x1+xn)/2; /* comando inicial */
    res1 = produto_direta_forte(x1,xm);   /* chamada 1 */
    res2 = produto_direta_forte(xm+1,xn); /* chamada 2 */
    res2 = res2 * res1; /* comando final */
    return(res2);
  }
}
```

http://www.pythontutor.com

Árvores de recursão

A ida e a volta das chamadas recursivas podem ser ilustradas com a topologia de uma árvore binária (indução forte) ou de sua degeneração em um lista (indução fraca).





Recursão indireta

- A título de ilustração apenas, na recursão indireta, a solução de um problema pode ser escrita em função de instâncias menores da solução de outro problema, e assim por diante, até que o último problema seja resolvido em função de instâncias menores do primeiro.

Recursão indireta

- A título de ilustração apenas, na recursão indireta, a solução de um problema pode ser escrita em função de instâncias menores da solução de outro problema, e assim por diante, até que o último problema seja resolvido em função de instâncias menores do primeiro.

- Um exemplo simples é a recursão mútua entre funções que respondem se um dado número $n \in Z^+$ é par/ímpar por contagem regressiva.

Recursão indireta

- A título de ilustração apenas, na recursão indireta, a solução de um problema pode ser escrita em função de instâncias menores da solução de outro problema, e assim por diante, até que o último problema seja resolvido em função de instâncias menores do primeiro.

- Um exemplo simples é a recursão mútua entre funções que respondem se um dado número $n \in Z^{+}$ é par/ímpar por contagem regressiva.

Recursão indireta

- A título de ilustração apenas, na recursão indireta, a solução de um problema pode ser escrita em função de instâncias menores da solução de outro problema, e assim por diante, até que o último problema seja resolvido em função de instâncias menores do primeiro.

- Um exemplo simples é a recursão mútua entre funções que respondem se um dado número $n \in Z^+$ é par/ímpar por contagem regressiva.

Um exemplo mais complexo seria a análise sintática que o compilador realiza para determinar se uma expressão matemática é válida.

A ordem de chamada dessas funções não altera o resultado.

```
char Par(unsigned int n)
{
    if (n == 0)
        return(1);
    else
        return(Impar(n - 1));
}

char Impar(unsigned int n)
{
    if (n == 0)
        return(0);
    else
        return(Par(n - 1));
}
```

Ordenação por recursão

Seja A uma sequência de números de tamanho n, armazenada em um vetor. Podemos ordenar A usando dois tipos de indução.

Seja A uma sequência de números de tamanho n, armazenada em um vetor. Podemos ordenar A usando dois tipos de indução.

- Indução Fraca: O tempo de ordenação será proporcional ao quadrado do tamanho da sequência, $O(n^2)$.

Ordenação por recursão

Seja A uma sequência de números de tamanho n, armazenada em um vetor. Podemos ordenar A usando dois tipos de indução.

- Indução Fraca: O tempo de ordenação será proporcional ao quadrado do tamanho da sequência, $O(n^2)$.
- Indução Forte: O tempo de ordenação será proporcional ao tamanho da sequência multiplicado por seu logaritmo na base 2, $O(n \log n)$.

Ordenação por recursão

Seja A uma sequência de números de tamanho n, armazenada em um vetor. Podemos ordenar A usando dois tipos de indução.

- Indução Fraca: O tempo de ordenação será proporcional ao quadrado do tamanho da sequência, $O(n^2)$.
- Indução Forte: O tempo de ordenação será proporcional ao tamanho da sequência multiplicado por seu logaritmo na base 2, $O(n \log n)$.

Em ambos os casos, a recursão é direta, pois a solução do problema é escrita em função de instâncias menores do mesmo problema.

Ordenação por indução fraca

A ordenação por indução fraca apresenta o seguinte padrão.

```
void Ordena(int *A, int n)
{
    if (condição) {
        comandos iniciais;
        Ordena(A, n − 1);
        comandos finais;
    }
}
```

Exemplos são ordenação por seleção, inserção, e permutação (`https://visualgo.net/pt`).

Ordenação por seleção

Exemplo 1: Recursão em cauda (Como fica a árvore de recursão?).

void OrdenaPorSelecao(int *A, int n)
{
  int i;
   if ($n > 1$) {
      i = IndiceDoMaior(A, n);
      Troca(&$A[i]$,&$A[n-1]$);
      OrdenaPorSelecao($A, n-1$);
   }
}

Ordenação por inserção

Exemplo 2:

void OrdenaPorInsercao(int *A, int n)
{
  int i;
    if ($n > 1$) {
      OrdenaPorInsercao($A, n - 1$);
      $i = n - 1$;
      while ((($i > 0$)&&($A[i] < A[i-1]$)) {
        Troca(&$A[i]$,&$A[i-1]$);
        $i - -$;
      }
    }
}

Exemplo 3: Recursão em cauda.

```
void OrdenaPorPermutacao(int *A, int n)
{
  int i;
    if (n > 1) {
        for (i = 0; i < n − 1; i + +) {
            if (A[i] > A[i + 1])
                Troca(&A[i],&A[i + 1]);
        }
        OrdenaPorPermutacao(A, n − 1);
    }
}
```

A ordenação por indução forte apresenta o seguinte padrão, em que p e q são índices que delimitam a porção do vetor a ser ordenada.

```
void Ordena(int *A, int p, int q)
{
  int r, s;
   if (condição) {
       comandos iniciais: divisão de p até r e de s até q;
       Ordena(A, p, r);
       Ordena(A, s, q);
       comandos finais: conquista;
   }
}
```

Exemplos são *merge sort* e *quick sort* (`https://visualgo.net/pt`).

Ordenação por *merge sort*

Exemplo 1: *Merge sort* ordena as metades da sequência e depois intercala as subsequências ordenadas (Como fica a árvore de recursão?).

```
void OrdenaPorIntercalacao(int *A, int p, int q)
{
  int r;
    if (p < q) {
        r = (p + q)/2;
        OrdenaPorIntercalação(A, p, r);
        OrdenaPorIntercalação(A, r + 1, q);
        Intercala(A, p, r, q);
    }
}
```

Intercala manipula os números dos subvetores ordenados de p a r e de $r+1$ a q de modo que o vetor de p a q fique ordenado.

Exemplo 2: *Quick sort* particiona a sequência em relação a um *pivot* r de modo que todos os elementos à esquerda de r sejam menores que os elementos à direita de r.

```
void OrdenaPorParticao(int *A, int p, int q)
{
  int r;
   if (p < q) {
      r = Particao(A, p, q);
      OrdenaPorParticao(A, p, r - 1);
      OrdenaPorParticao(A, r + 1, q);
   }
}
```

O tempo médio é $O(n \log n)$, mas no pior caso é $O(n^2)$.

Partição

```
int Partição(int *A, int p, int q)
{
  int x = A[p], i = p, j = q;
    do {
        while(A[i] ≤ x) i++;
        while(A[j] > x) j--;
        if (i < j) {
            Troca(&A[i],&A[j]);
            i++; j--;
        }
    } while (i < j);
    Troca(&A[p],&A[j]);
    return(j);
}
```

Partição

int Partição(int $*A$, int p, int q)
{
  int $x = A[p]$, $i = p$, $j = q$;
    do {
      while($A[i] \leq x$) $i++$;
      while($A[j] > x$) $j--$;
      if ($i < j$) {
        Troca(&$A[i]$,&$A[j]$);
        $i++$; $j--$;
      }
    } while ($i < j$);
    Troca(&$A[p]$,&$A[j]$);
    return(j);
}

Exercício: Desenhe a árvore de recursão da OrdenaçãoPorPartição.

Busca binária

- Uma vez ordenada a sequência de números, a busca por qualquer número na sequência pode ser realizada em $O(\log n)$.

Busca binária

- Uma vez ordenada a sequência de números, a busca por qualquer número na sequência pode ser realizada em $O(\log n)$.

- Uma aplicação é quando carregamos um índice (vetor de chaves e deslocamentos em bytes) de acesso aos registros de um arquivo grande em disco.

Busca binária

- Uma vez ordenada a sequência de números, a busca por qualquer número na sequência pode ser realizada em $O(\log n)$.

- Uma aplicação é quando carregamos um índice (vetor de chaves e deslocamentos em bytes) de acesso aos registros de um arquivo grande em disco.

- Mantendo o índice ordenado por chave, a busca binária pode ser usada para encontrar o deslocamento para acesso a um dado registro em disco.

Busca binária

- Uma vez ordenada a sequência de números, a busca por qualquer número na sequência pode ser realizada em $O(\log n)$.
- Uma aplicação é quando carregamos um índice (vetor de chaves e deslocamentos em bytes) de acesso aos registros de um arquivo grande em disco.
- Mantendo o índice ordenado por chave, a busca binária pode ser usada para encontrar o deslocamento para acesso a um dado registro em disco.
- A busca binária por indução forte acessa o elemento intermediário e, se ele não for a chave procurada, continua a busca de forma recursiva à esquerda ou à direita dele.

Busca binária

Retorna *true* e a posição da chave em pos, caso ela seja encontrada, ou *false* no caso contrário.

```
bool BuscaBinaria(int *A,int p,int q,int ch,int *pos)
{
  if (p <= q){
    int r = (p+q)/2;
    if (A[r]==ch){
      *pos = r;
      return(true);
    } else{
      if (A[r] < ch)
        return(BuscaBinaria(A,r+1,q,ch,pos));
      else
        return(BuscaBinaria(A,p,r-1,ch,pos));
    }
  }

  return(false);
}
```

Backtracking

Backtracking é uma estratégia para resolver problemas computacionais por busca exaustiva, que baseada em restrições é capaz de eliminar muitas soluções sem examiná-las.

```
bool Backtrack(<solução candidata>)
{
  <variáveis locais>
  if (<solução encontrada>) {
    < processa a solução >
  } else {
    < gera uma lista de candidatas que satisfazem as restrições >
    < executa Backtrack para cada solução candidata>
  }
  return(< true/false >)
}
```

Considere o labirinto do arquivo texto labirinto.txt em que 'E' indica entrada, 'S' saída, 'X' posição proibida, e 'P' posição permitida. Vamos completar o código de backtrack.c para resolver o labirinto, partindo da posição de entrada com deslocamentos de um ao longo da horizontal ou vertical em busca de candidatas.