AULA 26

Problema do passeio do cavalo

"*Creating a program to find a knight's tour is a common problem given to computer science students*"

PF 12
http://www.ime.usp.br/~pf/algoritmos/aulas/enum.html
http://en.wikipedia.org/wiki/Knight's_tour

Problema do passeio do cavalo

Problema: Suponha dado um tabuleiro de xadrez n-por-n. Determinar se é possivel que um cavalo do jogo de xadrez parta da posição (1,1) e complete um passeio por todas as n^2 posições.

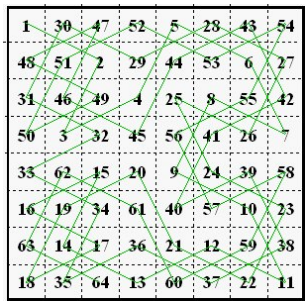

Imagem: http://www.magic-squares.net/knighttours.htm/

Soluções

	1	2	3	4	5
1	1	10	19	4	25
2	18	5	2	9	14
3	11	20	15	24	3
4	6	17	22	13	8
5	21	12	7	16	23

Soluções

	1	2	3	4	5	6
1	1	14	11	28	7	4
2	12	27	2	5	10	29
3	15	20	13	8	3	6
4	26	33	24	19	30	9
5	21	16	35	32	23	18
6	34	25	22	17	36	31

Soluções

	1	2	3	4	5	6	7
1	1	18	27	6	11	16	13
2	28	7	2	17	14	5	10
3	19	26	29	8	3	12	15
4	40	45	20	25	30	9	4
5	37	34	39	44	21	24	31
6	46	41	36	33	48	43	22
7	35	38	47	42	23	32	49

Movimentos do cavalo

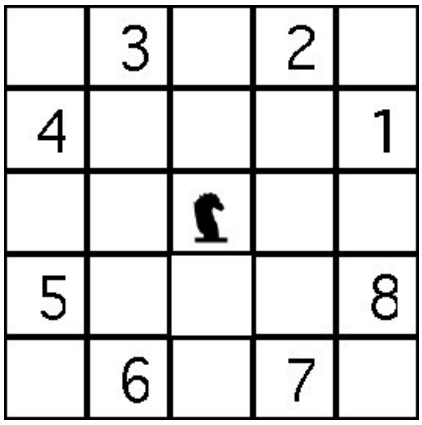

Imagem: http://www.mactech.com/articles/mactech/Vol.14/14.11/TheKnightsTour/index.html

Problema do passeio do cavalo

Cada movimento possível é de um dos tipos $1,\ldots,8$ mostrados. De uma maneira grosseira existem 8^{63} sequência de movimentos em um tabuleiro 8×8.

$$8^{63} \approx 7.9 \times 10^{56}$$

Supondo que conseguimos verificar cada sequência em 10^{-6} segundos, o tempo para verificar todas as sequências seria

7.9×10^{50} seg $\approx 1.310^{49}$ min $\approx 2.1 \times 10^{47}$ horas $\approx$

Deixa para lá ...

Algoritmo passeioCavalo

A função `passeioCavalo` a seguir imprime, caso exista, uma possível solução para o problema do passeio do cavalo em uma tabuleiro $n \times n$.

A função mantém no início da cada iteração a seguinte relação invariante

(i0) $s[1..k-1]$ são os movimentos de uma solução parcial do problema

Cada iteração procura estender essa solução fazendo mais um movimento

Algoritmo passeioCavalo

A função utiliza a função auxiliar e a struct

```
/* Imprime o tabuleiro com as rainhas em
s[1..i] */
void mostreTabuleiro(int n, int **tab);

typedef struct {
  int i;
  int j;
} Movimento;
```

Algoritmo passeioCavalo

```
void passeioCavalo (int n) {
  int **tab;
  int i, j; /* posicao atual */
  int iProx,jProx; /* coluna candidata */
  int nMovimentos = 0; /* num.  de mov */
  int *s = malloc((n*n+1)*sizeof(int));
  /* s[t] = movimento no passo t*/
  int k; /* passo atual */
```

Algoritmo passeioCavalo

```
  Movimento movimento[NMOV+1] = {
         {0,0}, /* fica parado */
         {-1,+2}, /* movimento[1] */
         {-2,+1}, /* movimento[2] */
         {-2,-1},
         {-1,-2},
         {+1,-2},
         {+2,-1},
         {+2,+1},
         {+1,+2} /* movimento[8] */
  };
  int mov;
```

Algoritmo passeioCavalo

```
/* linha 0 e coluna 0 do tabuleiro nao
serao usadas */
tab = malloc((n+1)*sizeof(int*));
for (i = 1; i <= n; i++)
   tab[i] = calloc(n+1,sizeof(int));
i = j = 1; /* posicao inicial */
k = 1; /* passo inicial */
mov = 1; /* movimento inicial */
tab[i][j] = 1; /* tabuleiro inicial */
iProx = jProx = 0; /* compilador feliz */
```

Algoritmo passeioCavalo

```
while (0 < k && k < n*n) {
   int achouMov = FALSE;
   nMovimentos++;
   while(mov <= NMOV && !achouMov){
       iProx = i + movimento[mov].i;
       jProx = j + movimento[mov].j;
       if (  (0 < iProx && iProx <= n)
           && (0 < jProx && jProx <= n)
           && tab[iProx][jProx] == 0)
           achouMov = TRUE;
       else
           mov++;
   }
```

Algoritmo passeioCavalo

```
   if (mov < NMOV) { /* AVANCA */
       i = iProx;
       j = jProx;
       s[k] = mov;
       tab[i][j] = ++k;
       mov = 1;
   } else { /* BACKTRACKING */
       tab[i][j] = 0;
       mov = s[--k];
       i -= movimento[mov].i;
       j -= movimento[mov].j;
       mov++;
   }
}
```

Algoritmo passeioCavalo

```
   if (k == n*n) {
       /* uma solucao foi encontrada */
       mostreTabuleiro(n,tab);
   } else printf("\n NAO TEM SOLUCAO\n");
   /* libera memoria alocada */
   free(s);
   for (i = 1; i <= n; i++)
       free(tab[i]);
   free(tab);
   printf("Num.  mov.=%d\n", nMovimentos);
}
```

Mais *backtracking*

O esquema a seguir tenta descrever o método *backtracking*.

Suponha que a solução de um problema pode ser vista como uma sequência de decisões

$$x[1], x[2], \ldots, x[n]$$

Por exemplo, cada $x[k]$ pode ser a posição de uma rainha ou para qual posição mover o cavalo.

A relação invariante chave do método é algo como

> *no início de cada iteração $x[1\,..\,k\text{-}1]$ é uma "solução parcial" (que pode ou não ser parte de uma solução)*

Mais *backtracking*

```
k ← 1
enquanto k > 1 faça
   procure valor para x[k] que ainda não
   foi testado e tal que x[1..k] é
   solução parcial
   se encontrou candidato para x[k] então
       k ← k + 1 (avança)
       se k = n então
           encontramos uma solução
           devolva x[1..n]
           k ← k − 1 (continua)
       senão k ← k − 1 (volta)
```

Enumeração de subsequências

PF 12
http://www.ime.usp.br/~pf/algoritmos/aulas/enum.html

Enumeração de subsequências

Problema: Enumerar todas as subsequências de 1,2,...,n, ou seja, fazer uma lista em que cada subsequência aparece uma e uma só vez.
Exemplo: para n= 3 as subsequências são

```
1
1   2
1   2   3
1   3
2
2   3
3
```

Enumeração de subsequências

Exemplo: para n= 4 as subsequências são

```
1
1   2
1   2   3
1   2   3   4
1   2   4
1   3
1   3   4
1   4
2
2   3
2   3   4
2   4
3
3   4
4
```

subseqLex

A função subseqLex recebe n e imprime todas as subsequências não vazias de 1 . . n.

```
void subseqLex (int n) {
  int *s, k;
  s = mallocSafe((n+1) * sizeof(int));
  s[0] = 0;
  k = 0;
```

subseqLex

```
  while(1) {
      if (s[k] < n) {
          s[k+1] = s[k] + 1;
          k += 1;
      } else {
          s[k-1] += 1;
          k -= 1;
      }
      if (k == 0) break;
      imprima(s, k);
  }
  free(s);
}
```

Comentários finais

MACO122 Princípios de Desenvolvimento de Algoritmos

Edição 2013

Livros

Nossa referência básica **foi** o livro

PF = Paulo Feofiloff,
Algoritmos em linguagem C,

Este livro é baseado no material do sítio

Projeto de Algoritmos em C.

Outros livros **foram**

S = Robert Sedgewick,
Algorithms in C, vol. 1

CLRS = Cormen-Leiserson-Rivest-Stein,
Introductions to Algorithms

MAC0122

MAC0122 **foi** uma disciplina introdutória em:

Projeto de algoritmos:

- recursão: torres de Hanoi,..., EP2, EP5...
- divisão-e-conquista: Mergesort, Quicksort, EP5
- pré-processamento: Heapsort, Boyer-Moore
- heurísticas: Boyer-Moore, EP5
- algoritmos de enumeração: n Rainhas, Passeio do Cavalo, EP1

MAC0122

MAC0122 **foi** uma disciplina introdutória em:

Correção de algoritmos:

- relações invariantes: vários problemas nas aulas

MAC0122

MAC0122 **foi** uma disciplina introdutória em:

Eficiência de algoritmos:

- consumo de tempo: vários problemas nas aulas
- notação assintótica O: vários problemas nas aulas
- análise experimental: vários problemas nas aulas
- consumo de espaço: Mergesort usa espaço extra $O(n)$, Quicksort usa espaço extra $O(\lg n)$

MAC0122

MAC0122 **foi** uma disciplina introdutória em:

Estruturas de dados:

- listas lineares encadeadas, listas encadeadas ciculares, listas com e sem cabeça: EP2, EP3, EP4 e EP5
- filas: distâncias, EP2, EP3
- pilhas: EP2, EP3, EP4
- heaps

MAC0122

MAC0122 **combinou** conceitos e recursos de programação:

- recursão: EP2, EP5
- strings: todos os EPs?
- endereços e ponteiros: EP2, EP3, EP4 e EP5
- registros e structs: EP2, EP3, EP4, e EP5
- alocação dinâmica de memória: EP2, EP3, EP4, e EP5
- interfaces: (EP1), EP2, EP3, EP4, e EP5

que nasceram de aplicações cotidianas em ciência da computação.

Principais tópicos

Alguns dos tópicos de MAC0122 foram:

- recursão;
- busca em um vetor;
- busca (binária) em vetor ordenado;
- listas encadeadas;
- listas lineares: filas e pilhas;
- algoritmos de enumeração;
- busca de palavras em um texto;
- algoritmos de ordenação: bubblesort, heapsort, mergesort,...; e

Tudo isso regado a muita análise de eficiência de algoritmos e invariantes.

C

"... There is an important pedagogical issue in choosing a language for our examples. Just as no language solves all problems equally well, no single language is best for presenting all topics. Higher-level languages preempt some design decisions. If we use a lower-level language, we get to consider alternative answers to the questions; by exposing more of the details, we can talk about them better. Experience shows that even when we use the facilities of high-level languages, it's invaluable to know how they relate to lower-level issues; without that insight, it's easy to run into performance problems and mysterious behavior. So we will often use C for our examples, even though in practice we might choose something else..."

The Practice of Programming

Brian W. Kernigham e Rob Pike

Pausa para nossos comerciais

- EP5: segunda-feira, 2/DEZ
- Prova 3: terça-feira, 3/DEZ
- Aula 06/DEZ: cancelada
- Linux: vocês devem usar
- Página do BCC: http://bcc.ime.usp.br/

Próximos anos

MAC0122 **foi** um primeiro passo para

- MAC0323 Estruturas de Dados
- MAC0328 Algoritmos de Grafos
- MAC0338 Análise de Algoritmos

Entretanto, várias outras disciplina se apoiam em MAC0122.