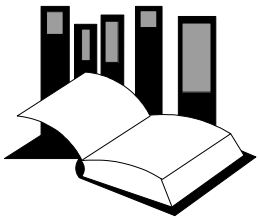
Algoritmos Exaustivos



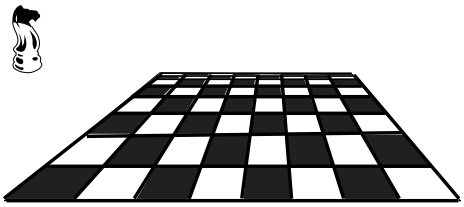
- Um problema interessante em programação é a *solução geral de um problema*
- A tarefa consiste em se determinar algoritmos destinados à busca de soluções para problemas específicos por meio de tentativa e erro
- Veremos três exemplos do emprego apropriado da técnica de recursão em algoritmos exaustivos:
  - O percurso do cavalo de xadrez
  - O problema das oito rainhas
  - O problema do casamento estável

Prof. Dr. José Augusto Baranauskas
DFM-FFCLRP-USP

1

Percurso do Cavalo de Xadrez

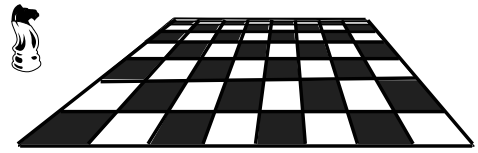
- Suponha que você tenha um cavalo de um jogo de xadrez
- ... e um tabuleiro de xadrez



2

Percurso do Cavalo de Xadrez

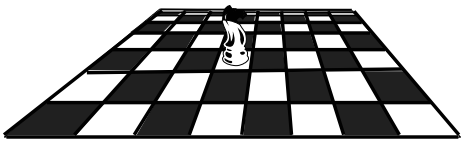
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



3

Percurso do Cavalo de Xadrez

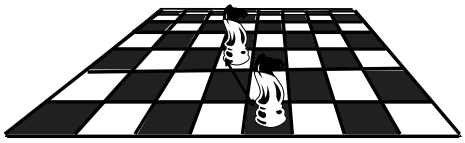
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



4

Percurso do Cavalo de Xadrez

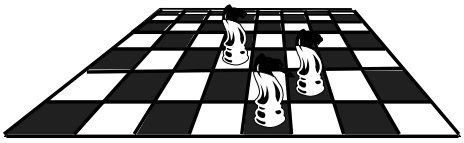
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



5

Percurso do Cavalo de Xadrez

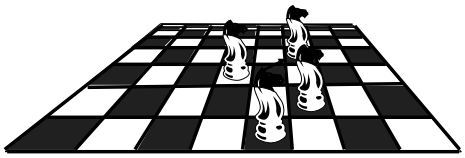
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



6

Percurso do Cavalo de Xadrez

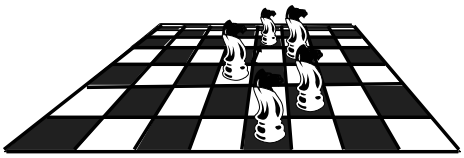
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



7

Percurso do Cavalo de Xadrez

- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



8

Percurso do Cavalo de Xadrez

- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...

9

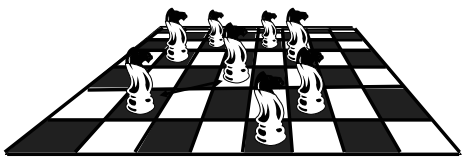
Percurso do Cavalo de Xadrez

- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...

10

Percurso do Cavalo de Xadrez

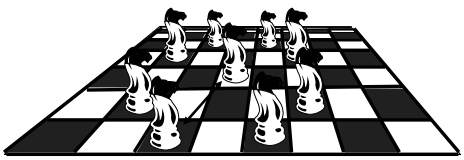
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



11

Percurso do Cavalo de Xadrez

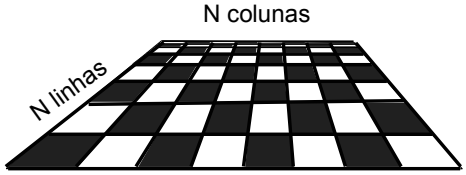
- É possível o cavalo percorrer todo o tabuleiro, tal que todas as células sejam percorridas exatamente uma vez?
- O cavalo deve se mover segundo as regras do jogo de xadrez...



12

Percurso do Cavalo de Xadrez

- O tamanho do tabuleiro pode variar...



13

1ª Versão

```
void TryNextMove()
{ iniciar seleção dos movimentos;
  do
  { selecionar o próximo candidato da lista de movimentos;
    if (aceitável)
    { registrar movimento;
      if (tabuleiro não preenchido completamente)
      { TryNextMove();
        if (insucesso)
          eliminar movimento;
      }
    }
  } while (movimento sem sucesso && há mais candidatos);
}
```

14

1ª Versão: Refinamentos

- O tabuleiro será representado por uma matriz de inteiros na qual os índices utilizados variam de 1 até o número de linhas/colunas, ou seja, h[1..8][1..8]:
  - `const int Size = 8;`
  - `int h[Size+1][Size+1];`
- onde:
  - h[x][y] == 0 indica que a célula (x,y) não foi ocupada
  - h[x][y] == i indica que a célula (x,y) foi ocupada no movimento i (1 <= i <= $\mathbf{n}^2$)
  - **n** indica o tamanho do tabuleiro (1 <= **n** <= Size)

15

1ª Versão: Refinamentos

- Parâmetros do procedimento
  - (x,y): coordenadas onde o cavalo se encontra
  - i: número do movimento (para fins de registro)
  - sucesso: variável booleana que indica que o movimento teve sucesso
- (u,v) serão duas variáveis locais que representam as coordenadas dos possíveis pontos de destino dos movimentos, determinados conforme a lei de movimentação do cavalo

16

1ª Versão: Refinamentos

- "tabuleiro não preenchido completamente"
  - i < n^2
- "aceitável"
  - 1 <= u <= n e 1 <= v <= n e h[u][v] == 0
- "registrar movimento"
  - h[u][v] = i
- "eliminar movimento"
  - h[u][v] = 0

17

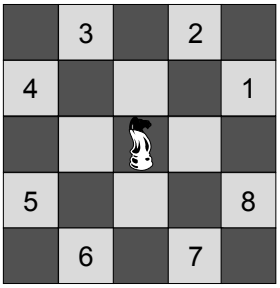
2ª Versão

```
void TryNextMove(int i, int x, int y, bool &s)
{ int u,v;
  bool sucesso;
  iniciar seleção dos movimentos;
  do
  { seja (u,v) ponto de destino do próximo movimento;
    if (1<=u && u<=n && 1<=v && v<=n && h[u][v]==0)
    { h[u][v] = i;
      if (i < n*n)
      { TryNextMove(i+1,u,v,sucesso);
        if (! sucesso)
          h[u][v] = 0;
      }
      else
        sucesso = true;
    }
  } while (! sucesso && há mais candidatos);
  s = sucesso;
}
```

18

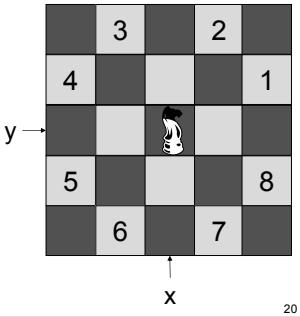
2ª Versão: Refinamentos

- Até o momento, o programa foi desenvolvido de maneira completamente independente das regras que regem os movimentos do cavalo
- Dado um par de coordenadas (x,y), existem oito possíveis células cadidatas (u,v) de destino



Os 8 Movimentos Possíveis

- (x,y) = posição atual
- (u,v) = nova posição
  - 1: (x+2,y+1)
  - 2: (x+1,y+2)
  - 3: (x-1,y+2)
  - 4: (x-2,y+1)
  - 5: (x-2,y-1)
  - 6: (x-1,y-2)
  - 7: (x+1,y-2)
  - 8: (x+2,y-1)



2ª Versão: Refinamentos

- Um método simples para obter (u,v) a partir de (x,y) é através da adição de diferenças de coordenadas, que serão armazenadas nos vetores dx e dy, que serão iniciados com valores apropriados
- Um índice k será utilizado para numerar o próximo candidato
- O procedimento recursivo é iniciado por uma chamada, tendo como parâmetros as coordenadas iniciais (i,j) fornecidas pelo usuário; a esta posição no tabuleiro deve ser atribuído o valor 1:
  - `h[i][j] = 1;`
  - `TryNextMove(2,i,j,q);`

Versão Final (1/2)

```
#include <iostream>
#include <iomanip>
using namespace std;

const int Size = 8;
int dx[Size+1] = {0,2,1,-1,-2,-2,-1,1,2},
    dy[Size+1] = {0,1,2,2,1,-1,-2,-2,-1},
    h[Size+1][Size+1],
    n;

//----------------------------------------
void Print()
{ int i,j;

  for(i=1; i<=n; i++)
  { for(j=1; j<=n; j++)
      cout << setw(5) << h[i][j];
    cout << endl;
  }
  cout << endl;
}
//----------------------------------------
```

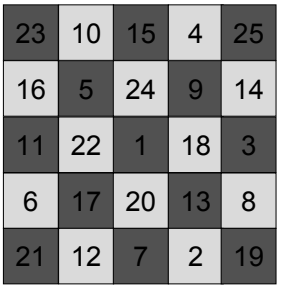
```
void TryNextMove(int i,int x,int y,bool &s)
{ int u,v,k;
  bool sucesso;

  k = 0;
  sucesso = false;
  do
  { k++;
    u = x + dx[k];
    v = y + dy[k];
    if (1<=u && u<=n && 1<=v && v<=n &&
        h[u][v]==0)
    { h[u][v] = i;
      if (i < n*n)
      { TryNextMove(i+1,u,v,sucesso);
        if (!sucesso)
          h[u][v] = 0;
      }
      else
        sucesso = true;
    }
  } while (!sucesso && k < Size);
  s = sucesso;
}
```


Versão Final (2/2)

```
int main()
{ int i,j;
  bool q;

  cout << "Tamanho do tabuleiro (1-8): ";
  cin >> n;
  for(i=1; i<=n; i++)
    for(j=1; j<=n; j++)
      h[i][j] = 0;
  cout << "Posicao inicial do cavalo (x,y): ";
  cin >> i >> j;
  h[i][j] = 1;
  TryNextMove(2,i,j,q);
  if (q)
    Print();
  else
    cout << "Caminho nao encontrado" << endl;
  return 0;
}
```

```
Tamanho do tabuleiro (1-8): 5
Posicao inicial do cavalo (x,y): 3 3
   23   10   15    4   25
   16    5   24    9   14
   11   22    1   18    3
    6   17   20   13    8
   21   12    7    2   19

Tamanho do tabuleiro (1-8): 5
Posicao inicial do cavalo (x,y): 4 2
   23   10   13    4   21
   12    5   22    9   14
   17   24   11   20    3
    6    1   18   15    8
   25   16    7    2   19

Tamanho do tabuleiro (1-8): 6
Posicao inicial do cavalo (x,y): 1 1
    1   16    7   26   11   14
   34   25   12   15    6   27
   17    2   33    8   13   10
   32   35   24   21   28    5
   23   18    3   30    9   20
   36   31   22   19    4   29
```


Exemplo de Percurso



Backtracking

- Qual o padrão que este exemplo exibe, que seja típico para este tipo de algoritmo de solução de problemas?
- O recurso característico é que passos próximos da solução total são obtidos e armazenados para uso futuro e eliminados da memorização quando se descobre que o passo corrente não terá, seguramente, nenhuma possibilidade de conduzir à solução total procurada, mas, em lugar disso, levará a um beco sem saída
- Esta conduta é conhecida como ***backtracking***

25

Backtracking: Padrão Geral

```
void Try()
{ iniciar seleção de candidatos;
  do
  { próxima seleção;
    if (aceitável)
    { gravar;
      if (solução incompleta)
      { Try();
        if (insucesso)
          cancelar gravação;
      }
    }
  } while (sem sucesso && há mais candidatos);
}
```

26

Backtracking

- Na prática, os programa assumem diversos padrões, derivados do padrão geral
- Uma forma padrão usualmente encontrada utiliza um parâmetro explícito, que indica o grau de profundidade de recursão e que permite a utilização de uma condição simples de término
- Se, além disso, em cada passo o número de candidatos a serem avaliados for fixo (por exemplo, **m**), então o padrão seguinte é aplicável, sendo ativado pelo comando **Try(1)**

27

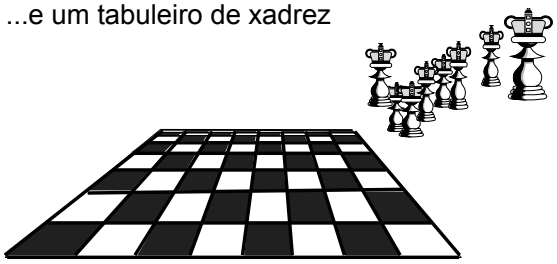
Backtracking: Padrão Geral

```
void Try(int i)
{ int k;
  k = 0;
  do
  { k = k + 1;
    selecionar o k-ésimo candidato;
    if (aceitável)
    { gravar;
      if (i < n)
      { Try(i+1);
        if (insucesso)
          cancelar gravação;
      }
    }
  } while (sem sucesso && k < m);
}
```

28

O Problema das Oito Rainhas

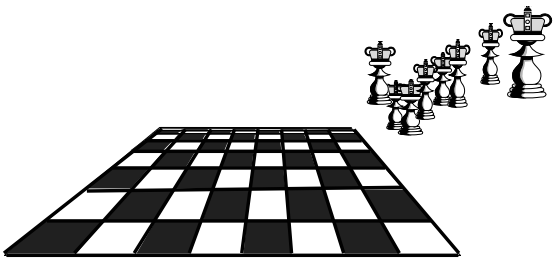
- Suponha que você tenha 8 rainhas de um jogo de xadrez...
- ...e um tabuleiro de xadrez



29

O Problema das Oito Rainhas

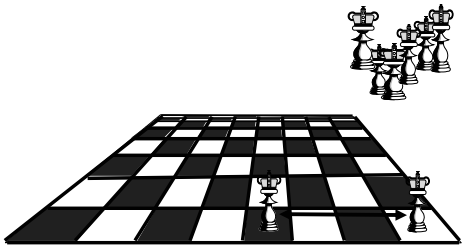
- As rainhas podem ser dispostas no tabuleiro de modo que duas rainhas não se ataquem?



30

O Problema das Oito Rainhas

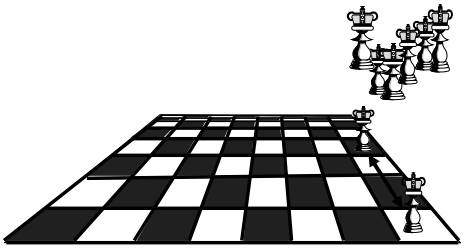
- Duas rainhas não são permitidas na mesma linha...



31

O Problema das Oito Rainhas

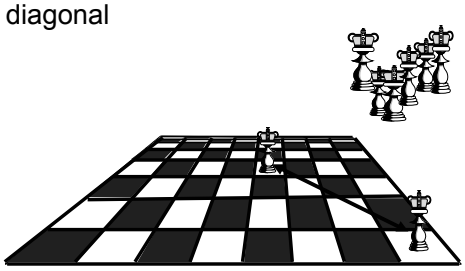
- Duas rainhas não são permitidas na mesma linha, ou na mesma coluna...



32

O Problema das Oito Rainhas

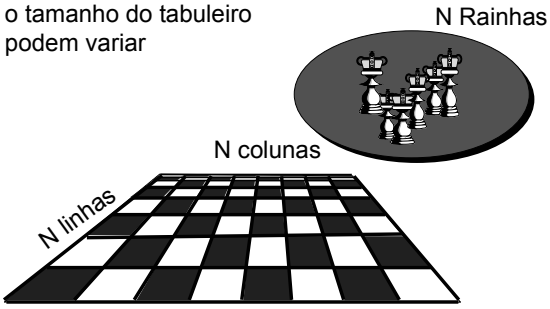
- Duas rainhas não são permitidas na mesma linha, ou na mesma coluna, ou na mesma diagonal



33

O Problema das N Rainhas

- O número de rainhas e o tamanho do tabuleiro podem variar



34

O Problema das N Rainhas

```
void Try(int i)
{ iniciar seleção de posição para a elecionar o i-ésima rainha;
  do
  { fazer próxima seleção;
    if (rainha está salva)
    { posicionar rainha;
      if (i < n)
      { Try(i+1);
        if (insucesso)
          remover rainha;
      }
    }
  } while (sem sucesso && há mais posições);
}
```

35

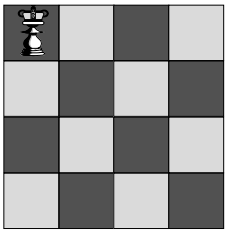
Estruturação dos Dados

- A rainha ataca peças que se encontrem na mesma linha, coluna ou diagonal do tabuleiro
  - Cada coluna deve conter uma e só uma rainha
  - A escolha da posição da i-ésima rainha deve restringir-se às casas da i-ésima coluna
- O parâmetro **i** é o índice da coluna
- A escolha da linha **j** limita-se a escolher uma dentre as n linhas disponíveis (1 <= j <= n)

36

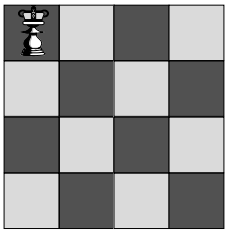
Como o Algoritmo Funciona

- Cada vez que o programa coloca uma rainha no tabuleiro, sua posição é guardada em x[i]



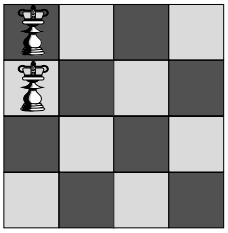
Como o Algoritmo Funciona

- Cada vez que o programa coloca uma rainha no tabuleiro, sua posição é guardada em x[i]
- Uma nova chamada recursiva é efetuada, Try(i+1)



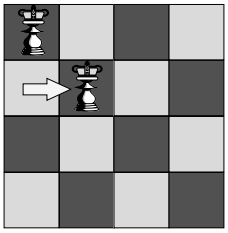
Como o Algoritmo Funciona

- Cada vez que o programa coloca uma rainha no tabuleiro, sua posição é guardada em x[i]
- Uma nova chamada recursiva é efetuada, Try(i+1), iniciando outro nível de recursão...



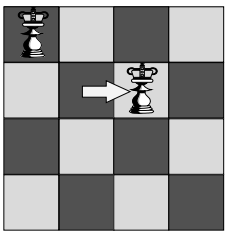
Como o Algoritmo Funciona

- Se há conflito o programa muda a nova rainha para a próxima coluna à direita



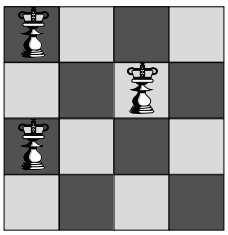
Como o Algoritmo Funciona

- Quando não há mais conflitos o programa tenta colocar a próxima rainha no tabuleiro, ativando Try(i+1)



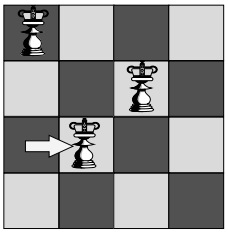
Como o Algoritmo Funciona

- Vamos olhar na terceira linha. A primeira posição que tentamos tem um conflito...



Como o Algoritmo Funciona

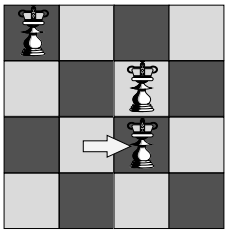
- Vamos olhar na terceira linha. A primeira posição que tentamos tem um conflito...
- ...então mudamos para a coluna 2. Mas outro conflito surge...



43

Como o Algoritmo Funciona

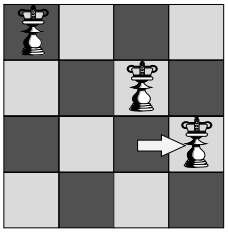
- Vamos olhar na terceira linha. A primeira posição que tentamos tem um conflito...
- ...então mudamos para a coluna 2. Mas outro conflito surge...
- ...e mudamos para a terceira coluna. Ainda há conflito...



44

Como o Algoritmo Funciona

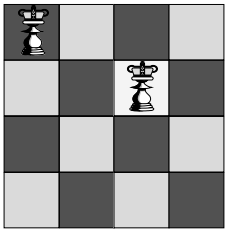
- Vamos olhar na terceira linha. A primeira posição que tentamos tem um conflito...
- ...então mudamos para a coluna 2. Mas outro conflito surge...
- ...e mudamos para a terceira coluna. Ainda há conflito...
- ...e mudamos para a quarta coluna, ainda há conflito e não existem mais colunas disponíveis



45

Como o Algoritmo Funciona

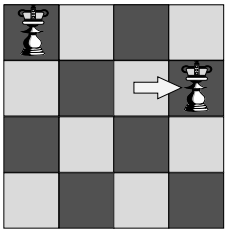
- Quando não há mais espaço em uma linha, o algoritmo retorna para a recursão de nível anterior, ou seja, do nível 3 para o 2



46

Como o Algoritmo Funciona

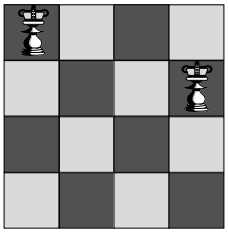
- Quando não há mais espaço em uma linha, o algoritmo retorna para a recursão de nível anterior, ou seja, do nível 3 para o 2
- ...e continua trabalhando na linha 2, mudando a rainha para a direita



47

Como o Algoritmo Funciona

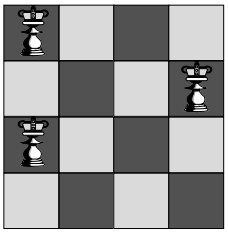
- Essa posição não apresenta conflitos, então podemos ativar o próximo nível de recursão, Try(i+1)



48

Como o Algoritmo Funciona

- Na linha 3, começamos novamente pela primeira coluna, reiniciando todo o processo



Estruturação dos Dados

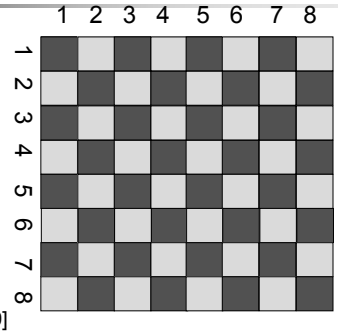
- A escolha óbvia para representar as rainhas no tabuleiro é uma matriz quadrada
- Entretanto, esta escolha acarretaria a necessidade de operações exaustivas para a verificação de disponibilidade de posições
- Assim, representaremos tão diretamente quanto possível a informação que for realmente relevante e muito utilizada
  - Não é o caso da posição das rainhas mas sim se a rainha já tenha sido colocada corretamente em uma dada posição (em uma linha, coluna ou diagonal)
  - Note que somente uma rainha deve ser colocada em cada coluna k (1 <= k <= i)

Estruturação dos Dados

- int x[1..8]
  - x[i] indica a posição da rainha da i-ésima coluna
- bool a[1..8]
  - a[j] significa “nenhuma rainha está há j-ésima coluna”
- bool b[2..16]
  - b[k] significa “nenhuma rainha ocupa a k-ésima diagonal do tipo /”
  - k = i + j
- bool c[2..16]
  - c[r] significa “nenhuma rainha ocupa a r-ésima diagonal do tipo \”
  - r = i – j (-7 <= r <= 7, índice assume valores negativos)
  - r = i – j + 9 (2 <= r <= 16)

O Problema das N Rainhas

- “posicionar rainha”
  - x[i] = j;
  - a[j] = false;
  - b[i+j] = false;
  - c[i-j+9] = false;
- “remover rainha”
  - a[j] = true;
  - b[i+j] = true;
  - c[i-j+9] = true;
- “rainha está salva”
  - a[j] && b[i+j] && c[i-j+9]



O Problema das N Rainhas

```
#include <iostream>
#include <iomanip>
using namespace std;

bool a[9],b[17],c[17];
int n,x[9];

void Print()
{ int i;

  for(i=1; i<=n; i++)
    cout << "(" << i
         << "," << x[i]
         << ") ";
  cout << endl;
}
```

```
void Try(int i, bool &s)
{ int j;

  j = 0;
  s = false;
  do
  { j++;
    if (a[j] && b[i+j] && c[i-j+9])
    { x[i] = j;
      a[j]=b[i+j]=c[i-j+9]=false;
      if (i < n)
      { Try(i+1,s);
        if (!s)
          a[j]=b[i+j]=c[i-j+9]=true;
      }
      else
        s = true;
    }
  } while (!s && j < n);
}
```


O Problema das N Rainhas

```
int main()
{ int i;
  bool q;

  cout << "Tamanho do tabuleiro (1-8): ";
  cin >> n;
  for(i=1; i<=8; i++)
    a[i] = true;
  for(i=2; i<=16; i++)
  { b[i] = true;
    c[i] = true;
  }
  Try(1,q);
  if (q)
    Print();
  else
    cout << "Nao foi possivel posicionar as rainhas" << endl;
  return 0;
}
```

O Problema das N Rainhas

- Uma extensão do algoritmo consiste em se encontrar não apenas uma solução, mas *todas* as soluções
- A geração de candidatos deve ocorrer de forma sistemática, de maneira que nenhum deles seja gerado mais de uma vez
- O algoritmo seguinte encontrar todas as soluções e é surpreendente que ele seja mais simples do que o algoritmo para a busca de uma única solução

55

O Problema das N Rainhas

```
void Try(int i)
{ int j;

  for(j=1; j<=n; j++)
  { selecionar j-ésimo candidato;
    if (aceitável)
    { gravar;
      if (i < n)
        Try(i+1);
      else
        imprimir a solução;
      cancelar gravação;
    }
  }
}
```

56

O Problema das N Rainhas

```
void Try(int i)
{ int j;

  for(j=1; j<=n; j++)
    if (a[j] && b[i+j] && c[i-j+9])
    { x[i] = j;
      a[j]=b[i+j]=c[i-j+9]=false;
      if (i < n)
        Try(i+1);
      else
        Print();
      a[j]=b[i+j]=c[i-j+9]=true;
    }
}
```

57

O Problema das Oito Rainhas

- O algoritmo estendido determina as 92 soluções do problema das oito rainhas
- Na realidade, existem apenas 12 soluções significativamente diferentes (a versão apresentada não detecta soluções simétricas)

58

Oito Rainhas: As Doze Soluções

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
1	5	8	6	3	7	2	4
1	6	8	3	7	4	2	5
1	7	4	6	8	2	5	3
1	7	5	8	2	4	6	3
2	4	6	8	3	1	7	5
2	5	7	1	3	8	6	4
2	5	7	4	1	8	6	3
2	6	1	7	4	8	3	5
2	6	8	3	1	4	7	5
2	7	3	6	8	5	1	4
2	7	5	8	1	4	6	3
2	8	6	1	3	5	7	4

59

O Problema do Casamento Estável

- Dados dois conjuntos disjuntos A e B de tamanho n, por exemplo, seja A um conjunto de homens e B um conjunto de mulheres
- Cada homem e cada mulher estabelecem preferências distintas para seus possíveis consortes
- Sendo escolhidos n casais
  - Se um homem e uma mulher que não estão casados um com o outro mas preferem um ao outro em lugar dos seus atuais parceiros, então a atribuição é **instável**
  - Caso não exista tal casal, a atribuição é **estável**

60

O Problema do Casamento Estável

- Essa situação caracteriza muitos problemas de relações em que devam ser feitas atribuições de acordo com preferências
  - Escolha de uma escola por estudantes
  - Escolha dos recrutas pelas diferentes áreas no serviço militar
  - Atribuição de disciplinas a docentes
- Uma solução consiste em se tentar construir pares de elementos dos dois conjuntos, um após o outro, até que se tenha varrido ambos os conjuntos

61

1ª Versão

```
void Try(int m) // m = homem
{ int r; // rank

  for(r=1; r<=n; j++)
  { escolher a r-ésima preferência do homem m;
    if (aceitável)
    { registrar casamento;
      if (m não é o último homem)
        Try(sucessor(m));
      else
        imprimir a solução;
      cancelar casamento;
    }
  }
}
```

62

Refinamentos

- Supõe-se que Try(m) ative o algoritmo para encontrar uma parceira para o homem **m** e a busca prossegue na ordem indicada pela lista de preferências do homem
- Os dados iniciais são representados por duas matrizes que indicam as preferências dos homens e mulheres, respectivamente:
- wmr[1..n][1..n]
  - wmr[m][r] é a mulher que ocupa a **r**-ésima posição na lista de preferências (rank) do homem **m**
- mwr[1..n][1..n]
  - mwr[w][r]: é o homem que ocupa a **r**-ésima posição na lista de preferências (rank) da mulher **w**

63

Refinamentos

- Exemplo de dados para *wmr* e *mwr*
- wmr[m][r] é a mulher que ocupa a **r**-ésima posição na lista de preferências (rank) do homem **m**
- mwr[w][r]: é o homem que ocupa a **r**-ésima posição na lista de preferências (rank) da mulher **w**

wmr

r =	1	2	3	4	5	6	7	8
m = 1	7	2	6	5	1	3	8	4
2	4	3	2	6	8	1	7	5
3	3	2	4	1	8	5	7	6
4	3	8	4	2	5	6	7	1
5	8	3	4	5	6	1	7	2
6	8	7	5	2	4	3	1	6
7	2	4	6	3	1	7	5	8
8	6	1	4	2	7	5	3	8

mwr

r =	1	2	3	4	5	6	7	8
w = 1	4	6	2	5	8	1	3	7
2	8	5	3	1	6	7	4	2
3	6	8	1	2	3	4	7	5
4	3	2	4	7	6	8	5	1
5	6	3	1	4	5	7	2	8
6	2	1	3	8	7	4	6	5
7	3	5	7	2	4	1	8	6
8	7	2	8	4	5	6	3	1

64

Refinamentos

- O resultado será representado por um vetor de mulheres x[1..n], tal que x[m] denota a parceira do homem m
- Para manter a simetria entre homens e mulheres, y[1..n] é tal que y[w] denota o parceiro da mulher w
- Na verdade, y[w] é redundante, já que é valido para todos os **m** e **w** casados:
  - x[y[w]] = w
  - y[x[m]] = m
- Entretanto, o vetor y aumenta visivelmente a eficiência do algoritmo

65

Refinamentos

- A informação representada pelos vetores **x** e **y** é necessária para se determinar a estabilidade de um conjunto proposto de casamentos
- Já que tal conjunto é construído passo a passo, promovendo-se o casamento de indivíduos e testando-se a estabildiade após cada casamento proposto, **x** e **y** são necessários mesmo antes que todos os seus componentes tenham sido definidos
- O estado civil de um homem pode ser determinado pelo valor de **m**, ou seja, todos os homens k estão casados se **k**<=m
- O estado civil de uma mulher será determinado pelo vetor booleano single[1..n], onde single[w] implica que a mulher **w** é solteira e !single[w] que a mulher **w** está casada

66

2ª Versão

```
void Try(int m)
{ int r; // rank
  int w; // mulher

  for(r=1; r<=n; j++)
  { w = wmr[m][r];
    if (single[w] && estável)
    { x[m] = w; y[w] = m; single[w] = false;
      if (m < n)
        Try(sucessor(m));
      else
        imprimir a solução;
      single[w] = true;
    }
  }
}
```


Refinamentos

- Não é possível determinar estabilidade por uma expressão tão simples como ocorreu no caso do programa das N-Rainhas
- Deve-se ter em mente que a estabilidade é definida a partir de comparações de ordens de classificação
- Tais ordens, de homens e mulheres, estão explicitamente disponíveis na coleção de dados estabelecida até este ponto
- A classificação da mulher **w** na mente do homem **m** pode ser calculada, mas somente por meio de uma busca onerosa de **w** em wmr[**m**]

Refinamentos

- Como o cálculo da estabilidade é uma operação que ocorre com muita freqüência, é aconselhável fazer com que essa informação seja mais diretamente acessível
- rmw[1..n][1..n]
  - rmw[m][w] denota a classificação da mulher **w** na lista de preferências do homem **m**
- rwm[1..n][1..n]
  - rwm[w][m] denota a classificação do homem **m** na lista de preferências da mulher **w**

Estabilidade

- O objetivo é casar **m** com **w**, onde wmr[m][r], isto é, a r-ésima escolha do homem **m**
- Sendo otimista, presume-se que a estabilidade se mantenha
- Com essa hipótese, começa-se a verificar quais são as possíveis fontes de problemas:
  - Haveria uma mulher **pw**, preferida com relação a **w** por **m**, e ela (**pw**), por sua vez, prefere **m** no lugar do seu marido?
  - Haveria um homem **pm**, preferido com relação a **m** por **w**, e ele (**pm**), por sua vez, prefere **w** no lugar da sua esposa?

Estabilidade

- Para atacar a primeira fonte de problemas, comparam-se as classificações de rwm[pw][m] e rwm[pw][y[pw]] para todas as mulheres preferidas com relação a **w** por **m**, isto é, para toda pw=wmr[m][i], tal que i < r
- Sabe-se que todas essas mulheres candidatas já estão casadas porque se alguma delas estivesse ainda solteira, **m** a teria escolhido antes
- O progresso descrito pode ser formulado por uma busca linear simples, onde **s** denota estabilidade

```
s = true; i = 1;
while(i < r && s)
{ pw = wmr[m][i];
  i = i + 1;
  if(!single[pw])
    s = rwm[pw][m] > rwm[pw][y[pw]];
}
```


Estabilidade

- Para combater a segunda fonte de problemas, deve-se investigar todos os candidatos pm que sejam preferidos por w em relação aos seus companheiros atuais, isto é, todos os homens preferidos pm=mwr[w][i], tal que **i** < rwm[w][m]
- Analogamente ao que ocorre com a primeira fonte de problemas, é necessária a comparação entre as classificações rmw[pm][w] e rmw[pm][x[pm]]
- Entretanto, é preciso ter cuidado de omitir comparações em envolvam x[pm] em que **pm** está ainda solteiro; isso pode ser obtido pelo teste **pm<m**, uma vez que se sabe que todos os homens que precedem **m** já estão casados

Versão Final

```
#include <iostream>
#include <iomanip>
using namespace std;

const int n=8;
int wmr[n+1][n+1],
    mwr[n+1][n+1],
    rmw[n+1][n+1],
    rwm[n+1][n+1],
    x[n+1],
    y[n+1];
bool single[n+1];
//-----------------------------------------------------------
void Print()
{ int m, rm=0, rw=0;

  for(m=1; m<=n; m++)
  { cout << setw(4) << x[m];
    rm = rm + rmw[m][x[m]];
    rw = rw + rwm[x[m]][m];
  }
  cout << setw(8) << rm << setw(4) << rw << endl;
}
```

73

Versão Final

```
bool stable(int m, int w, int r)
{ int pm,pw,i,lim,rank;
  bool s;

  s = true; i = 1;
  while(i < r && s)
  { pw = wmr[m][i];
    i = i + 1;
    if(!single[pw])
      s = rwm[pw][m] > rwm[pw][y[pw]];
  }
  i = 1;  lim = rwm[w][m];
  while(i < lim && s)
  { pm = mwr[w][i];
    i = i + 1;
    if (pm < m)
      s = rmw[pm][w] > rmw[pm][x[pm]];
  }
 return s;
}
```

74

Versão Final

```
void Try(int m)
{ int r; // rank
  int w; // mulher

  for(r=1; r<=n; r++)
  { w = wmr[m][r];
    if (single[w] && stable(m,w,r))
    { x[m] = w; y[w] = m; single[w] = false;
      if (m < n)
        Try(m+1);
      else
        Print();
      single[w] = true;
    }
  }
}
```

75

Versão Final

```
int main()
{ int m,w,r,h;

  for(m=1; m<=n; m++)
    for(r=1; r<=n; r++)
    { cin >> wmr[m][r];
      rmw[m][wmr[m][r]] = r;
    }
  for(w=1; w<=n; w++)
  { single[w] = true;
    for(r=1; r<=n; r++)
    { cin >> mwr[w][r];
      rwm[w][mwr[w][r]] = r;
    }
  }
  Try(1);
  return 0;
}
```

76

Solução

- rm = soma das classificações de todas as mulheres na lista de preferência de seus maridos
- rw = soma das classificações de todas os homens na lista de preferência de suas esposas
- Solução 1 = solução ótima masculina
- Solução 9 = solução ótima feminina

	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	rm	rw
1	7	4	3	8	1	5	2	6	16	32
2	2	4	3	8	1	5	7	6	22	27
3	2	4	3	1	7	5	8	6	31	20
4	6	4	3	8	1	5	7	2	26	22
5	6	4	3	1	7	5	8	2	35	15
6	6	3	4	8	1	5	7	2	29	20
7	6	3	4	1	7	5	8	2	38	13
8	3	6	4	8	1	5	7	2	34	18
9	3	6	4	1	7	5	8	2	43	11

77