OLPR0001 – Introdução a Programação por Restrições

(2 de novembro de 2014) – 6a.lista – Grafos, Seleção de Variáveis e Domínios

Entrega digital, se houver gráficos e figuras, deposite também em separado no Dropbox

Leia com atenção as instruções de entrega

Entrega: 15 dias

1 Explicando a Combinatória dos Caminhos

Recentemente o prof. Claudio se defrontou de um simples e antigo problema de caminhos. Mas, o que chama atenção é que ele tem uma natureza de explicar o que uma combinatória. Veja a figura 1 como um exemplo de entrada.

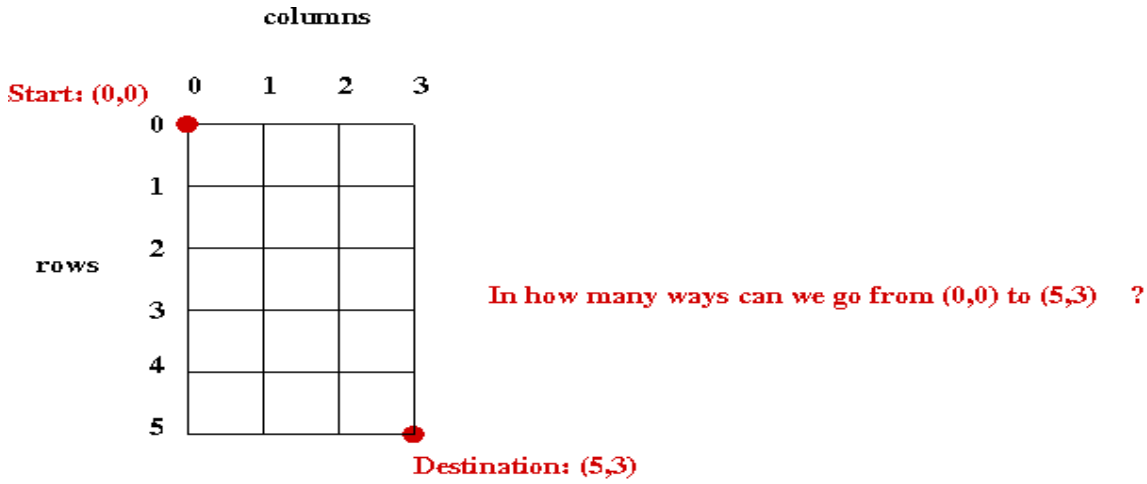

Figura 1: Quantos caminhos possíveis existem (sem ciclos) entre estes dois pontos?

Inicialmente dado um quadrado $n \times n$ dispostos como os dois primeiros exemplos da figura 2. Considerando sempre o mesmo ponto de partida (p) e o de chegada (c). A proposta é encontrar quantos caminhos possíveis, sem repetições entre nós e arcos, existem entre estes 2 pontos. Para os exemplos da figura 2, o número de caminhos está indicado sob a figura.

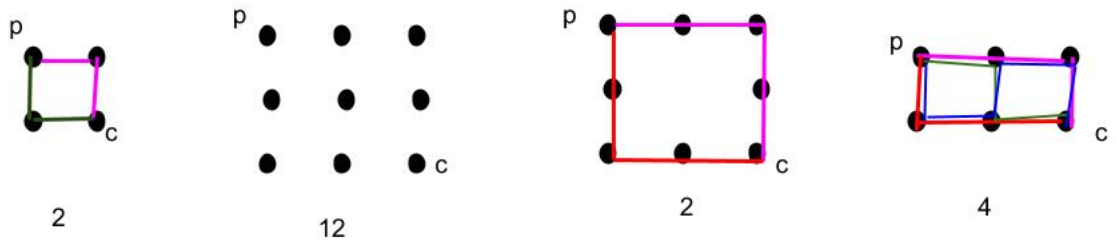

Figura 2: Caminhos Possíveis em Grafos Reticulados

Veja que o último exemplo, o quadrado tornou-se genérico: $m \times n$, bem como com alguns pontos ausentes.

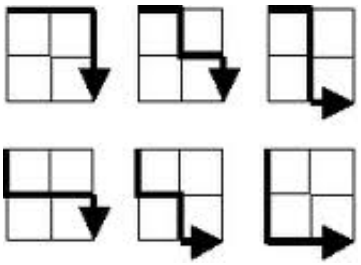

Figura 3: Exemplos de caminhos possíveis neste reticulado

Considere que este grafo se encontra no formato de uma matriz de adjacência, e que os caminhos devem respeitar os vizinhos imediatos. Exemplos de caminhos podem ser vistos em 3.

Experimentem implementar algum algoritmo que resolva este problema com os recursos vistos até o momento, e depois partam para restrições mais específicas e dirigidas a este fim. Exemplo: restrição *circuit* e a biblioteca *library(graph_algorithms)* do Eclipse. Esta observação vale para o próximo problema.

2 Caminho Mínimo

Seja o grafo da figura 4 tomado como exemplo. Implemente um algoritmo para um caminho mínimo entre dois pontos.

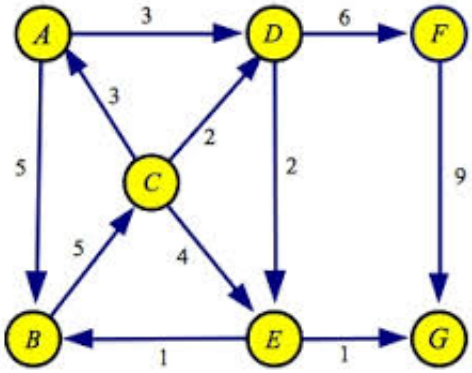

Figura 4: Simples Grafo

Teste o algoritmo com vários pontos de partida e chegada, incluindo as exceções. Funcionando, assuma um exemplo mais complexo como o da figura 5.

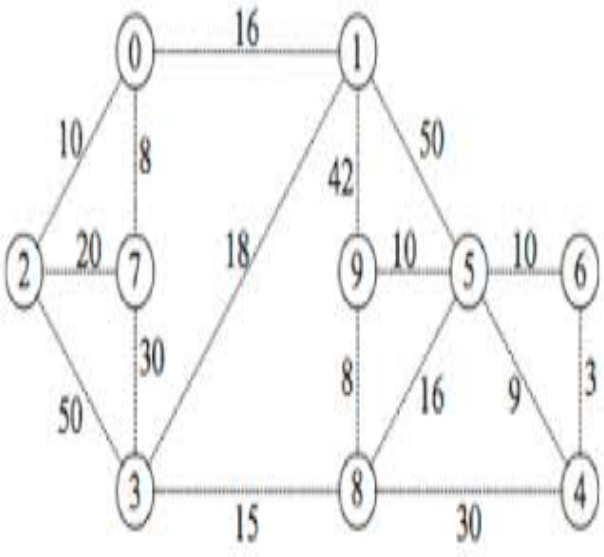

Figura 5: Grafo 2

Tanto para este problema como o anterior, o aluno deve estudar o problema de coloração de vértices, os quais foram discutidos em duas versões em sala, e que servem como base.

3 Problemas do K-Cavalos – Vale um TCC

Este problema apresenta muitas variações e tão interessante no estudo da PR como o problema das N-rainhas, apresentado em aula. Neste problema, há um aspecto a mais que no problema da N-rainhas clássico não apresentava: a otimização.

Para o nosso problema, o problema é simples. Quantos cavalos K (nossos cavalos tem os mesmos movimentos permitidos ao jogo de xadrez) são necessários para cobrir um tabuleiro $N \times N$? Diferentemente ao problema das rainhas, aqui os cavalos podem se atacar mutuamente, veja uma solução exemplo na figura 6.

Assim, sua tarefa é encontrar o menor número de cavalos necessários para atacar todas as células deste tabuleiro.

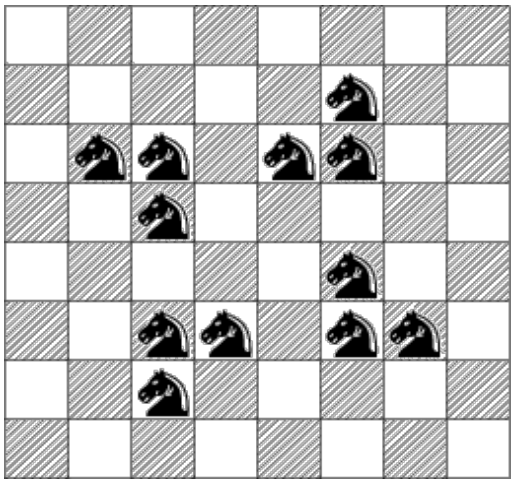

Figura 6: Cavalos atacando todas as células

Implemente este problema, contudo, rode para algumas instâncias ($N = 5, 10, 20, 40, 80$ – como não conheço o número de instâncias plausíveis, creio que até $N = 20$ teremos respostas em tempos aceitáveis) e procure preencher a tabela com os diversos tipos de escolha de variáveis e de domínio. Siga como exemplo a tabela abaixo (referência: tabelas 1 e 2) como guia, e escolha uns 5 testes do quais voce entende que para este problema vai alcançar um bom e ruim desempenho. Veja no exemplo das n-rainhas, slides do professor, alguns parâmetros e variações.

O Minizinc oferece menos parâmetros, e eventualmente com algum outro nome. Exemplo no seu código:

```
solve :: int_search(K_cavalos, input_order, indomain_min, complete)
%% satisfy;   este funciona
minimize;
```

Ou ainda algo mais genérico (sequência de search):

```
solve :: seq_search([
int_search(K_cavalos, smallest, indomain_min, complete),
int_search(K_cavalos, input_order, indomain_min, complete)
]) minimize end;
```

Descreva a característica da máquina que fizeste os testes, bem como uma execução por vez é o suficiente, já que a idéia é a minimização.

Conclua o experimento, respondendo:

Tabela 1: Tempos de execução com manipulação dos parâmetros do *search*. $N = ...$

Sel. Variável / Atr. Dominio	`first_fail`	`anti_first_fail`	`occurrence`
`indomain_min`	....	....	....
`indomain_max`	....	....	....
`indomain_median`	....	....	....
`indomain_split`	....	....	....
`indomain_random`	....	....	....
`indomain_interval`	....	....	....

Tabela 2: Continuação da tabela anterior. $N = ...$

Sel. Variável / Atr. Dominio	`most_constrained`	`max_regret`
`indomain_min`	....	....
`indomain_max`	....	....
`indomain_median`	....	....
`indomain_split`	....	....
`indomain_random`	....	....
`indomain_interval`	....	....

1. Porquê ocorreram os piores e melhores resultados para este problema em específico?
2. O que poderia ser feito para ser melhorado?

Se quiser estender o experimento acima e fazer uma *verdadeiro experimento científico*[1], fique confortável e será contabilizado na nota.

3.1 Material de apoio

1. Outros detalhes deste problema começam em: `http://mathworld.wolfram.com/KnightsProblem.html`
2. `http://www.imada.sdu.dk/~marco/DM87/Exercises/mzn-tutorial.pdf`, veja seção 1.5 as opções do search.

[1] Aumentar N, variar heurísticas, etc.

4 Considerações Finais:

- Leia e siga as instruções de entrega
- Faça vários testes. Em geral ninguém faz, mas, é para fazer vários testes de I/Os
- Assuma e justifique os dados que faltarem.