Passeio do cavalo

Diogo Tuler Chaves

Para a resolução do trabalho, meu primeiro passo foi entender mais sobre o passeio do cavalo em si. Ao entender o problema, recordei um vídeo sobre função recursiva do professor Sérgio Campos que foi passado como material complementar na matéria de Programação e Desenvolvimento do Software 1. no qual é apresentado um algoritmo de busca por força bruta conhecido como backtracking para a resolução de um labirinto. Logo, notei uma similaridade no problema do labirinto e no do Passeio do Cavalo e busquei fazer uma adaptação do algoritmo mostrado no vídeo para essa nova situação.

Pesquisando mais sobre o backtracking consegui fazer minha primeira versão do código. Ela era bem simples,contendo uma função chamada “caminho” que simplesmente ia testando uma rota e, caso ficasse sem movimentos, voltava e testava um novo. Infelizmente, dependendo do lugar de onde o cavalo começava no tabuleiro, o código demorava muito para rodar, teve até um caso que demorou a noite inteira.Testei e com tabuleiros menores que 8x8 quase todas as casas que o cavalo começasse dava certo praticamente instantaneamente. Dessa forma, percebi que seria necessário alguma heurística que tornasse meu código mais rápido.

Logo, me deparei com a "Warnsdorff 's Rule” que, intuitivamente, fazia bastante sentido. Ela é uma forma de determinar, dentre vários possíveis movimentos, qual seria a melhor escolha. Nesse caso, a melhor escolha é sempre o deslocamento do cavalo que gera a menor quantidade de possíveis próximos movimentos ao ser feito. Tendo a lógica comecei a implementação, foi um pouco difícil pois tive que mudar bastante a lógica, mesmo mantendo o backtracking.

Após várias horas de implementação e testes, ficou pronta essa versão do meu código que, felizmente, acabou funcionando .Nessa implementação, na maioria dos casos em tabuleiros 8x8, o código acaba não retrocedendo nenhuma vez, rodando praticamente de forma instantânea em todos os meus testes. Além disso, que ele funciona para certos casos até mesmo com tabuleiros bem maiores,como 70x70,mostrando a importância dessa heurística para a execução do código.

Depois do meu código funcionar, fiz adaptações para que a forma de entrada e saída fosse a desejada. Além disso, dei uma lapidada no código para que o entendimento fosse mais fácil.Logo abaixo, irei listar as partes do meu código e explicar cada uma delas.

- `#define MAX`
  - O tamanho máximo do tabuleiro.

- `#define MIN`
  - A primeira coluna/linha.

- `int tabuleiro[MAX][MAX];`
  - Uma matriz que representa o tabuleiro em si.

- `typedef struct movimentos_possiveis {} cavalo;`
  - Uma struct para conseguir armazenar os possíveis movimentos para o cavalo.

- `cavalo movimentos[8]`
  - Os possíveis movimentos para o cavalo.

- `typedef struct ponto_com_rank {}ponto`
  - Uma struct para armazenar um possível movimento para o cavalo e o seu rank.
    - No meu código, rank é a quantidade de possíveis próximos movimentos que o deslocamento armazenado no struct gera.

- `int caminho(int linha_atual,int coluna_atual,int casa_atual, int* casas_visitadas,int* recursividade);`
  - Essa é a função principal onde o backtracking ocorre. Inicialmente ela não envolvia rank e funcionava como um backtracking normal. Na versão final do meu código, essa função mudou, e agora é dividida em duas partes. Primeiramente a função avalia qual os possíveis movimentos, seu rank e os ordena, utilizando as funções abaixo. Depois disso,ocorre de fato o backtracking, no qual o cavalo vai fazendo os movimentos, priorizando os com menor rank sempre, e retrocedendo quando ficar sem.

- `void rank_da_funcao(ponto *analisando);`
  - Essa função, adicionada depois no meu código, recebe a casa que o cavalo irá parar depois de um movimento específico do cavalo e fornece um “rank” para ela, o armazenando no variável rank do struct ponto. No final, após várias chamadas dessa função, teremos um vetor contendo os possíveis movimentos e seus ranks, que será ordenado pela função abaixo.

        - Vale ressaltar novamente que rank é a quantidade de possíveis próximos movimentos que o deslocamento armazenado no struct gera.Ou seja, um movimento de rank 1 significa que ao fazer esse movimento vai existir apenas uma opção de deslocamento.

- `void ordenar_por_rank ponto (ordenando[],int tamanho);`
    - Essa função é bastante simples, e serve apenas para ordenar os possíveis pontos pelo seu rank, para que, seguindo a "Warnsdorff 's Rule”, a escolha seja sempre o deslocamento do cavalo que gera a menor quantidade de possíveis próximos movimentos ao ser feito.

- `int averiguar(int linha, int coluna);`
    - Essa função serve para testar se o ponto que será analisado está dentro dos limites do tabuleiro, delimitados por MAX e MIN.

- `void preencher_matriz();`
    - Essa função simplesmente preenche o tabuleiro com 0s.

- `void print_matriz();`
    - Essa função serve para printar a matriz apenas.

- `void passeio(int x, int y)`
    - Essa é a função chamada pela main que não faz nada além de ler as entradas, chamar a função “caminho” e organizar a saída, chamando a função “print_matriz” quando necessário.

Links utilizados para a resolução do trabalho:

- https://www.youtube.com/watch?v=TBowEHtwjMo
- http://warnsdorff.com/
- https://www.youtube.com/watch?v=07yVVGltD7w