Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
SCC0218 - Algoritmo Avançados e Aplicações
Professor Gustavo Batista

Relatório: Trabalho 1

Eduardo Aguiar Pacheco (8632455)

1. IDEIA DO ALGORÍTMO

O trabalho propõe um algoritmo para solucionar um tabuleiro qualquer de Futoshiki através da técnica *backtracking*. Como requisitado, o algoritmo pode ser executado de três formas, sendo elas:

I. Sem poda, onde o tabuleiro é percorrido sequencialmente e um valor bom é proposto para cada posição, onde bom significa não violar nenhuma regra. A recursão segue da seguinte forma:
   A. Se não for possível solucionar o tabuleiro com esse valor, o próximo é escolhido.
   B. Se as opções acabarem o *backtracking* é aplicado, informando seu predecessor que não foi possível para o cenário atual.

II. Com poda, utilizando a inteligência da verificação adiante. Todas as células do tabuleiro armazenam uma lista de possibilidades ainda válida, e outros índices são mantidos pela inteligência interna. As funcionalidades são:
   A. Quando um valor é escolhido para uma célula todas as que sofrem com essa decisão são atualizadas, a saber, toda a linha, toda a coluna e toda a cadeia de celulas cujos valores são restritos àquele através dos sinais de menor que ou maior que.
   B. Se uma célula ficar sem possibilidades, mesmo que distante da fonte da alteração, o ramo de decisões é interrompido, retornando a impossibilidade de prosseguir com tal decisão.
   C. Cada possível valor para cada linha e cada coluna é mapeado em relação à célula que ainda pode recebê-lo.
      1. Se nenhuma célula for elegível para tal valor, a decisão tomada é cancelada.
      2. Se somente uma célula for elegível, todas as outras possibilidades de tal célula são removidas. Essa iminente decisão afeta as possibilidades das outras células: menores que, maiores que, posse do valor.
   D. Sempre que o menor ou o maior valor válido para uma célula são alterados, seja por consequência do comportamento A, B, C ou a recursão deste D, inicia-se a poda de todas as células com restrições (menor que, maior que) encadeadas a essa. De tal forma que, se uma célula X tem que ser menor que Y:

1. Poda alta: as possibilidades de X são limitadas pelo teto da maior possibilidade de Y
2. Poda baixa: as possibilidades de Y são limitadas pelo chão da menor possibilidade de X

III. Com verificação adiante e heurística Mínimos Valores Remanescentes (MVR). Todas as funcionalidades descritas no item II são mantidas, mas, além delas, adiciona-se um índice de células com o menor número de possibilidades. Dessa forma, optando pela célula com o menor número de opções, a árvore de decisões é consideravelmente diminuída.

2. DETALHES DA IMPLEMENTAÇÃO

A opção I do algoritmo foi implementada usando a mesma estrutura de dados, porém não faz uso do método *set_number*, pois esse é desencadeia a atualização dos índices e listas de possibilidades. Ao invés disso, só verifica as possibilidades de uma célula no momento de atribuir uma valor a ela.

As peculiaridades do código se iniciam no item II. O uso de bits como *flags* foi oficializado pelo *namespace bitflags*, em particular o tipo *wbitflags*, onde o w se refere ao tamanho *word*, que são 2 bytes. A lista de possibilidades de uma célula é representada por um *wbitflags*, onde cada bit setado (igual a um) representa um valor válido, onde o valor um é o bit zero (da direita para a esquerda).

O índice de células disponíveis para cada valor de uma linha ou coluna (*wbitflags **remaining*) também foi implementado utilizando *wbitflags*. Mais especificamente:

a) um vetor com 2*D posições, onde D é a largura do tabuleiro, representam cada coluna e depois as linhas

b) cada posição desse vetor aponta para uma lista com D valores, representam cada número possível para aquela linha/coluna

c) os valores são *wbitflags*, onde cada bit representa a célula que pode receber aquele valor.

O índice de prioridades, *bbitfags **priority_index*, é melhor visto como uma matrix, onde as linhas representam o número de possibilidades para uma celula, e as colunas representam as células. Dessa forma, se o bit 10 da segunda linha estiver setado, significa que a célula número dez só possui dois valores válidos. É simples assim, porem como o número máximo de células é 81, o que ultrapassa os 16 bits de *wbitflags*, essa estrutura foi construída como uma vetor de bytes *bbitflags*. Para acessar o bit que representa a célula X, faça: acesse byte (X / D), acesse bit (X mod D). Isso gera um comportamento curioso onde os bytes crescem para a direita, e os bit crescem para a esquerda, mas o Core i5 não tem preconceito com essas coisas.

A última estrutura peculiar é o histórico, *stack<uint8_t> history*. Essa pilha armazena bytes indiscriminadamente, então a lógica fica na responsabilidade dos métodos *history_push*, *history_pop*. Essa estrutura é necessária para exercer o *backtracking* utilizando a verificação adiante pois seria muito custoso recalcular as possibilidades de todas as células envolvidas em uma decisão quando ela é alterada. Dessa forma, sempre que uma célula é alterada seu vetor de possibilidades é salvo no histórico. Quando o

backtracking é feito basta estabelecer o *frame* anterior da pilha, retornando os valores anteriores das células e atualizando so índices.

3. RESULTADOS

Lookahead + MVR:
Total de clocks: 17735, equivalentes a 0 segundos
100 tabuleiros resolvidos e ocupam 17735 clocks.
0 tabuleiros fora do limite de atribuições e ocupam 0 clocks.

Lookahead somente:
Total de clocks: 12646577, equivalentes a 12 segundos
58 tabuleiros resolvidos e ocupam 1075568 clocks.
42 tabuleiros fora do limite de atribuições e ocupam 11571009 clocks.

Nenhuma poda:
Total de clocks: 12664395, que equivalentes a 12 segundos
58 tabuleiros resolvidos e ocupam 1059505 clocks.
42 tabuleiros fora do limite de atribuições e ocupam 11604890 clocks.