Processo Seletivo CROSSBOTS – Programação - 067

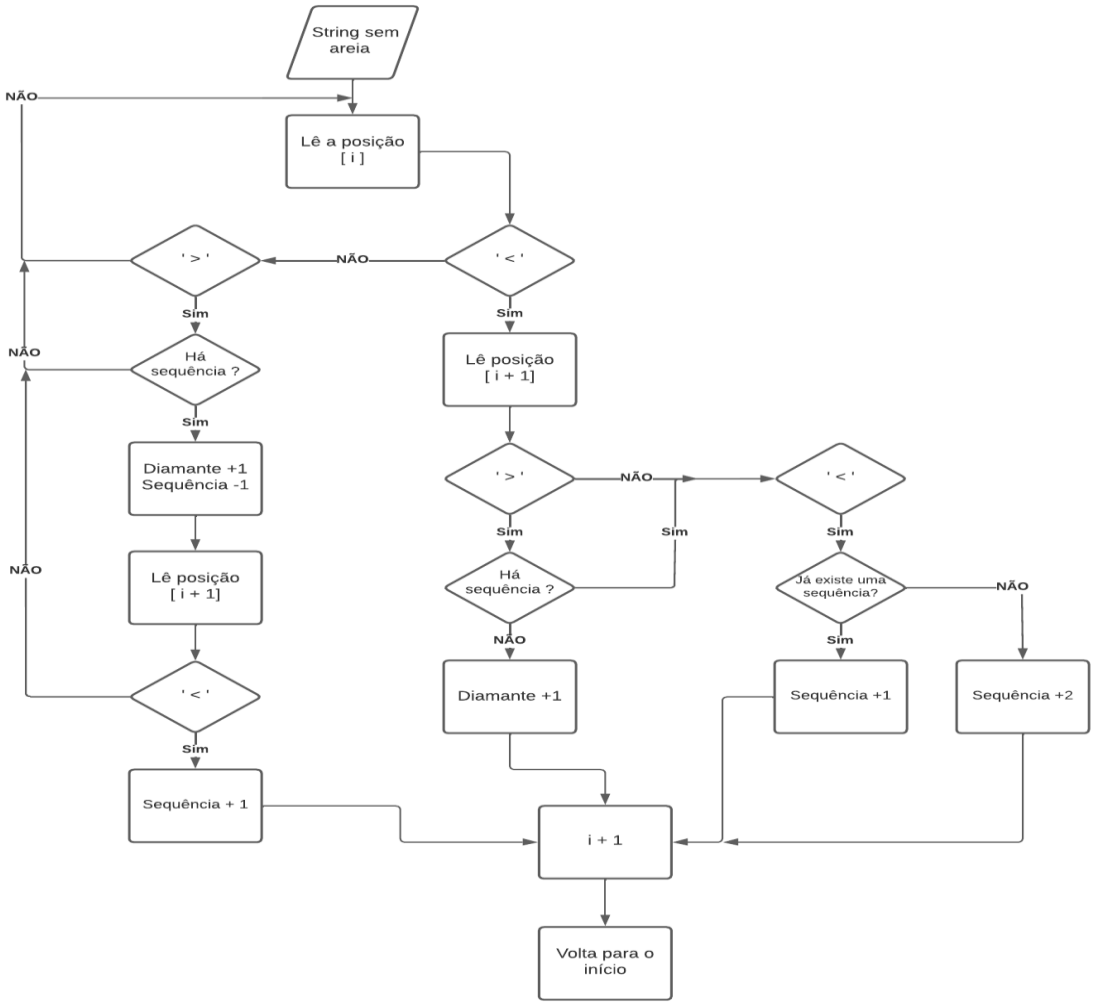
Problemas difíceis:

1) Para esse problema eu pensei à primeira vista em duas versões: inicialmente o problema é resolvido de maneira similar, as areias são retiradas com uma função e então mandadas para outra função que vai fazer a contagem, que é onde começa a diferença.

• A **Versão1** vai ler a linha somente uma vez e então “prever” as possíveis combinações levando em consideração possíveis diamantes sendo bloqueados por outros diamantes.

• A **Versão2** vai ler a linha várias vezes, e a cada vez que encontrar os diamantes vai elimina-los e reestruturar a string, fazendo então outra leitura, até que nenhum diamante seja mais encontrado. (Versão2 é mais simples e explicada no código .c)

Para melhor visualização da Versão1, veja o fluxograma a seguir:



A principal preocupação em “prever” diamantes ainda não formados é formar um programa que consiga lembrar de ‘ < ’ passados, que possam ser futuramente usados em possíveis ‘ > ’ caso um caminho entre eles seja liberado, sendo esse caminho o diamante formado ‘ <> ’. A “memória” dessa sequência é representada no fluxograma como ‘Sequência’, sendo que há seu incremento quando mais de um ‘ < ’ são encontrados.

OBS: como o programa lê a posição atual e a seguinte, no caso de dois ou mais ‘ < ’ é necessário o caso de ‘+1’ e ´+2´ para sequência (sendo o +2 quando há uma nova sequência e ela é vista como um par), já que no caso de ‘ << ’ o programa leria [i = 1] e a seguinte posição, percebendo a sequência e aumentando em ‘+2’, ao ler [i = 2] e sua seguinte posição, ele pode não encontrar outra ‘ < ’ e portanto não aumentando novamente a sequência, porém em ‘<<<’, ele leria [i = 1] e sua seguinte posição aumentando em ‘+2’, e então ao ler [i = 2] e sua seguinte posição, ele aumentaria em ‘+1’, e ao final em [i = 3] não aumentaria.

Outro caso importante a se denotar é quando há uma nova possível sequência após o diamante, sendo preciso que ela seja incluída na contagem de sequência, assim como visto no canto inferior esquerdo do fluxograma.

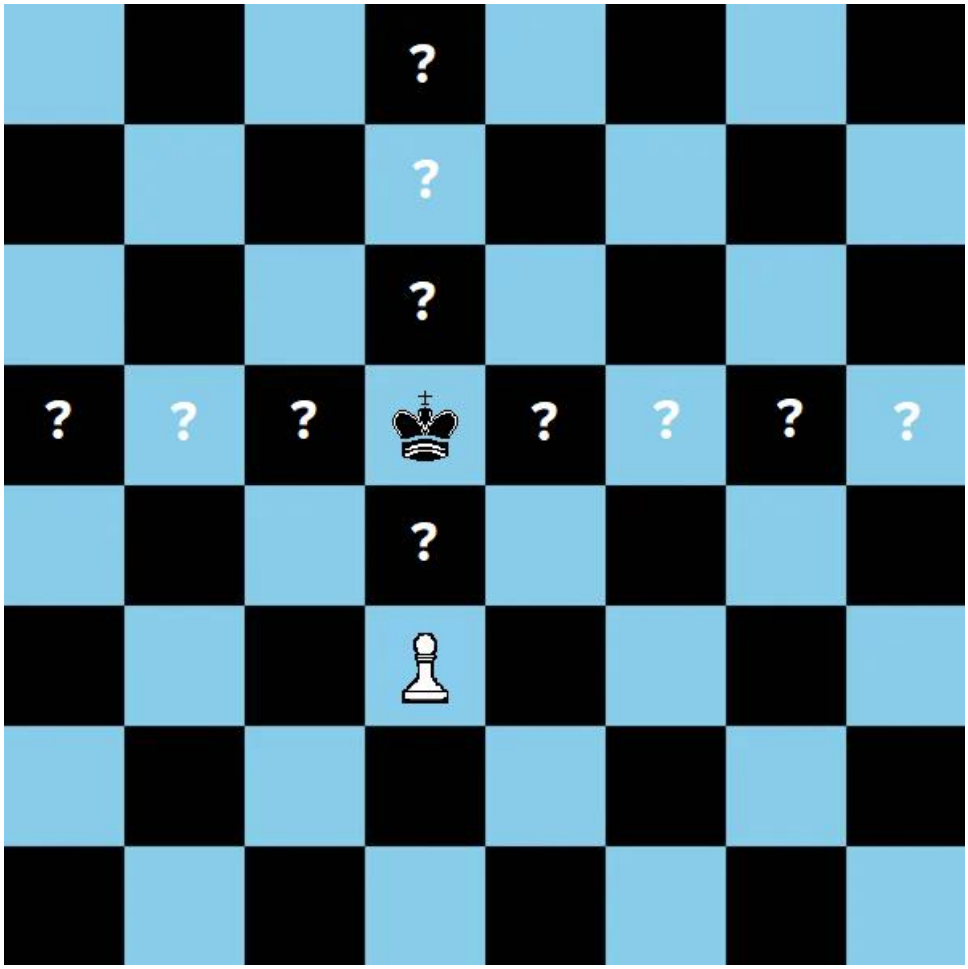
EX : ´ << <> <<´ Sequência = 4.

2) Para esse problema eu também pensei em duas possíveis soluções, sendo ela relativamente diferentes em princípio, uma versão faz a análise da posição de cada peça e seus alcances e verifica se o rei está presente em uma delas, a outra versão é o oposto, ela faz a análise a partir do rei e verifica todas posições que uma peça pode se posicionar para ataca-lo.

• A Versão1 possuí dois tabuleiros, um char de 64 posições que armazena a entrada e um int de 64 posições que armazena as posições “marcadas” como território que cada peça alcança, a partir da leitura e comparação dos dois tabuleiros, é verificado se o rei está numa posição “marcada” como a do inimigo.

• A Versão2 foi criada através da ideia de fileiras e colunas, sendo cada fileira denominada por ´a´, ´b´, ´c´, etc. e cada coluna como ´1´, ´2´, ´3´, etc... Ao se armazenar a entrada na char, faz-se a procura da posição do rei, e a partir de sua posição, todas possíveis posições que cada determinada peça inimiga pode estar e o ameaçar são analisadas. (O código em si é bem simples e auto explicativo no código e de relativa fácil identificação já que foi usado a ideia de coordenadas com linhas e colunas ‘a1’, ´b5´, etc...) **** Eu me empolguei com o problema e fiz essa versão mais por diversão e curiosidade, então não ligue para a quantidade de linhas****

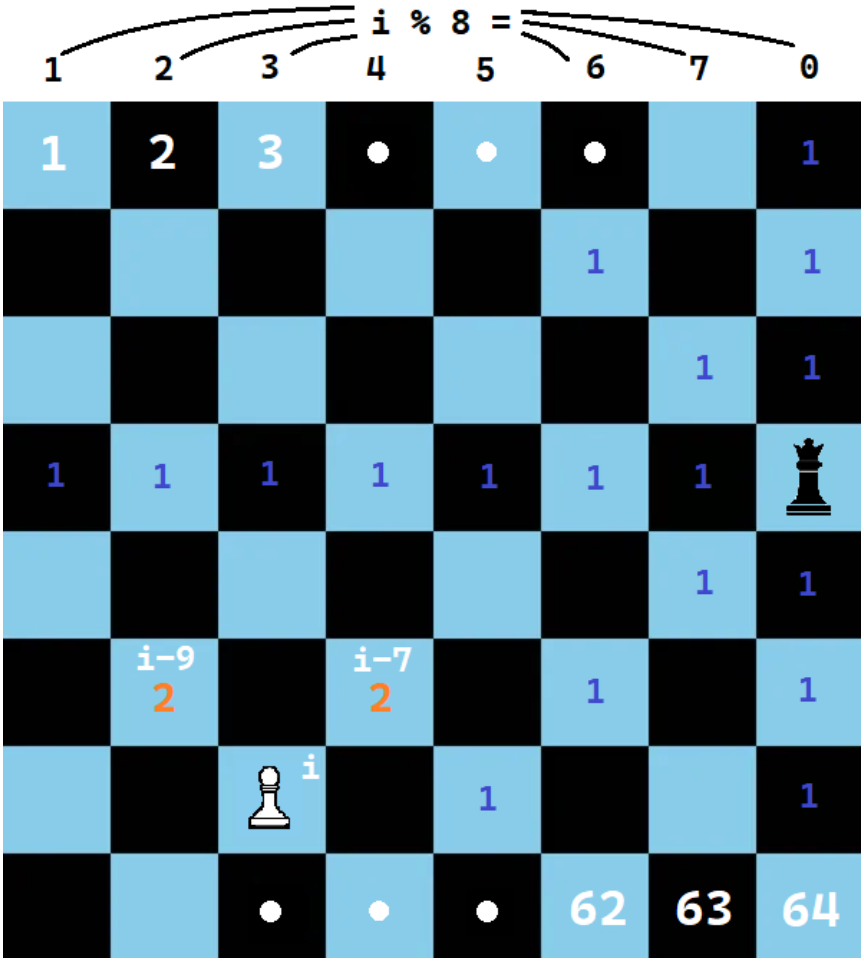
Exemplo da procura do rei por uma possível torre inimiga (Versão2):



A ideia da procura e marcação das peças na Versão1 é um pouco menos intuitiva já que o tabuleiro não está separado em linhas e colunas, mas sim em “uma linha contínua de 64 caracteres”.

Para melhor visualização do processo usado no código, segue a imagem de um exemplo de marcação:

OBS: note que há um padrão quando um número na mesma coluna é dividido por 8, eles possuem um resto assim como indicado na parte superior da imagem a seguir



Como o tabuleiro tem tamanho definido, e cada peça tem ataques definidos, tais comandos como “i-9” e “i-7” podem ser feitos para defini-los, e o mesmo é executado com cada peça, marcando todas posições possíveis no tabuleiro int formado no começo, com as constantes ‘1’ para ataque preto e ‘2’ para ataque branco.

Após todas marcações, a posição dos reis que foram inicialmente descobertas são comparadas com cada posição nesse tabuleiro int com os ataques, e caso um rei esteja numa posição de ataque inimigo, o cheque é indicado.