DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Instituto Superior Técnico

Algoritmos e Estruturas de Dados

MEEC (2019/2020 – 1º Sem)

Relatório do Projeto

Campista

Grupo: 76

Ana Catarina Rodrigues Grilo

Aluno nº: 930003 **e-mail:** ana.catarina.grilo@tecnico.ulisboa.pt

Tomás De Sousa Lopes Barata Salgueiro

Aluno nº: 93192 **e-mail:** tomas.b.salgueiro@tecnico.ulisboa.pt

ÍNDICE

Descrição do problema

De um modo geral o problema do projeto de AED consiste na resolução de um mapa (matriz) que contem algumas árvores (caracter 'A') e espaços livres (caracter '.') em que é pedido a colocação de um certo número de tendas (caracter 'T'). Cada linha e coluna do mapa tem um número máximo de tendas que é necessário cumprir. Para a colocação das tendas é necessário cumprir certas restrições, tais como: cada tenda necessita de estar ao pé de uma árvore, não contando com os espaços diagonais a árvore, cada árvore só pode dar "sombra" a uma e só uma tenda, as tendas não podem estar juntas, isto é, quando colocada uma tenda esta não pode ter tendas em espaços adjacentes incluindo os diagonais, e verificar se é possível colocar a tenda nessa linha e coluna de acordo com o número de lotação para estas.

Abordagem ao problema

Para a implementação deste projeto decidimos dividir o projeto em quatro partes. A primeira parte encontra-se na função Ficheiro que lê e faz uma primeira validação dos dados e determina se o problema é admissível de forma mais superficial (através dos vetores de linha e coluna) chamando para isso outras funções. A segunda parte que se encontra na função resolução consiste em testes de admissibilidade mais a fundo onde já analisamos o conteúdo da matriz nesta parte também tentamos diminuir o espaço de procura para ser mais fácil e rápida a resolução do problema descobrindo quais as árvores que tem espaços disponíveis para colocar uma tenda. A terceira parte do projeto, a solução do mapa, alocamos uma estrutura que irá auxiliar a procura da solução e criamos duas funções onde este processo acontece a função Solução e a Colocar_Tenda. A última parte consiste na escrita da resposta ao problema no ficheiro de saída.

Descrição das Estruturas de Dados

Estrutura, Campismo, que guarda os dados do problema dado:

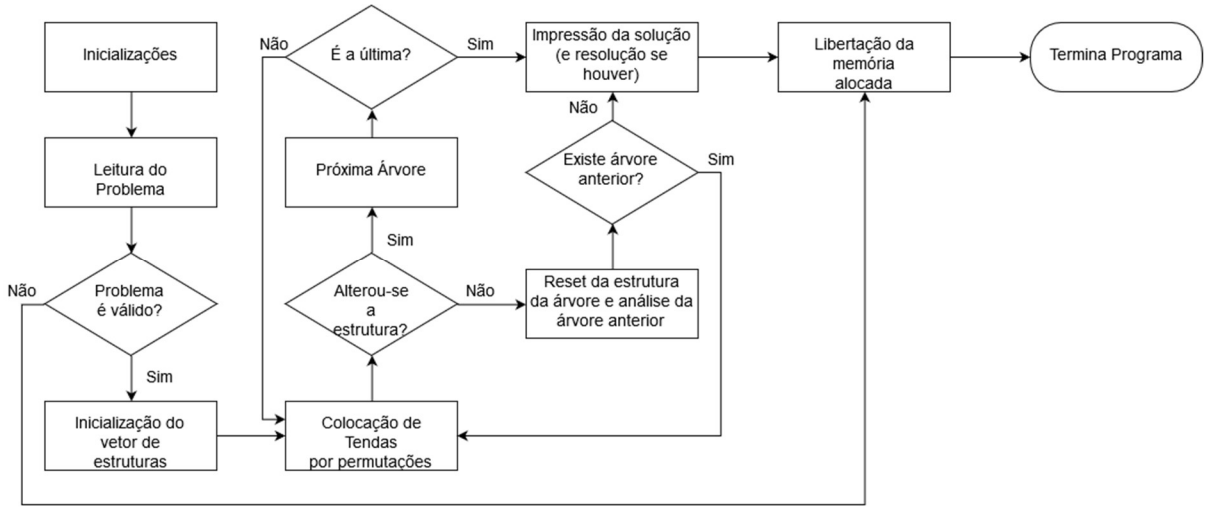
1) Dois inteiros que guardam o número de Linhas e Colunas
2) Dois apontadores de inteiros que vão apontar para os vetores que iram ser alocados das Linhas e Colunas
3) Um apontador de apontador de inteiro que ira apontar para a matriz (mapa dado) que irá ser alocado

Se o problema for admissível então para auxiliar a procura de existência de solução utilizamos uma estrutura, Vect_Tendas, que ajuda o processo de colocação de tendas:

1) Cinco inteiros que vão servir de flags de colocação de tendas (já experimentado ou não) para as posições a volta da árvore (incluindo sem tenda, época baixa)
2) Dois inteiros que vão guardar a posição da árvore
3) Dois inteiros que vão guardar a posição da tenda colocada ao pé da árvore

A procurar da solução de um mapa admissível faz-se, com ajuda da estrutura anteriormente descrita, através da estrutura dados apreendida nas aulas teóricas que são as árvores sendo que neste não esta explicita no sentido que não alocamos memória para todas as possibilidades de um mapa. A árvore assim é implícita mas com o auxílio da estrutura Vect_Tendas sabemos quais as possibilidades exploradas ao longo da procura.

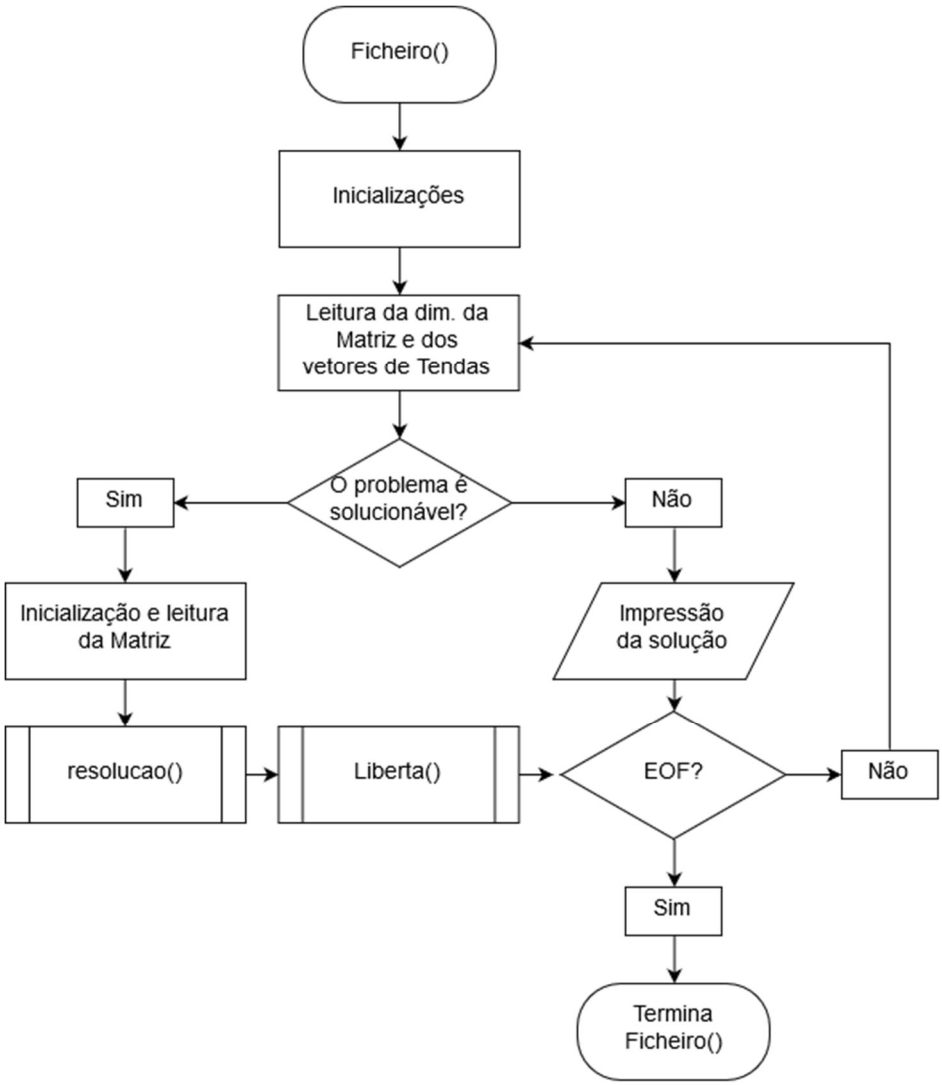
Fluxograma Geral



Descrição dos algoritmos

1) Descrição das Funções Principais

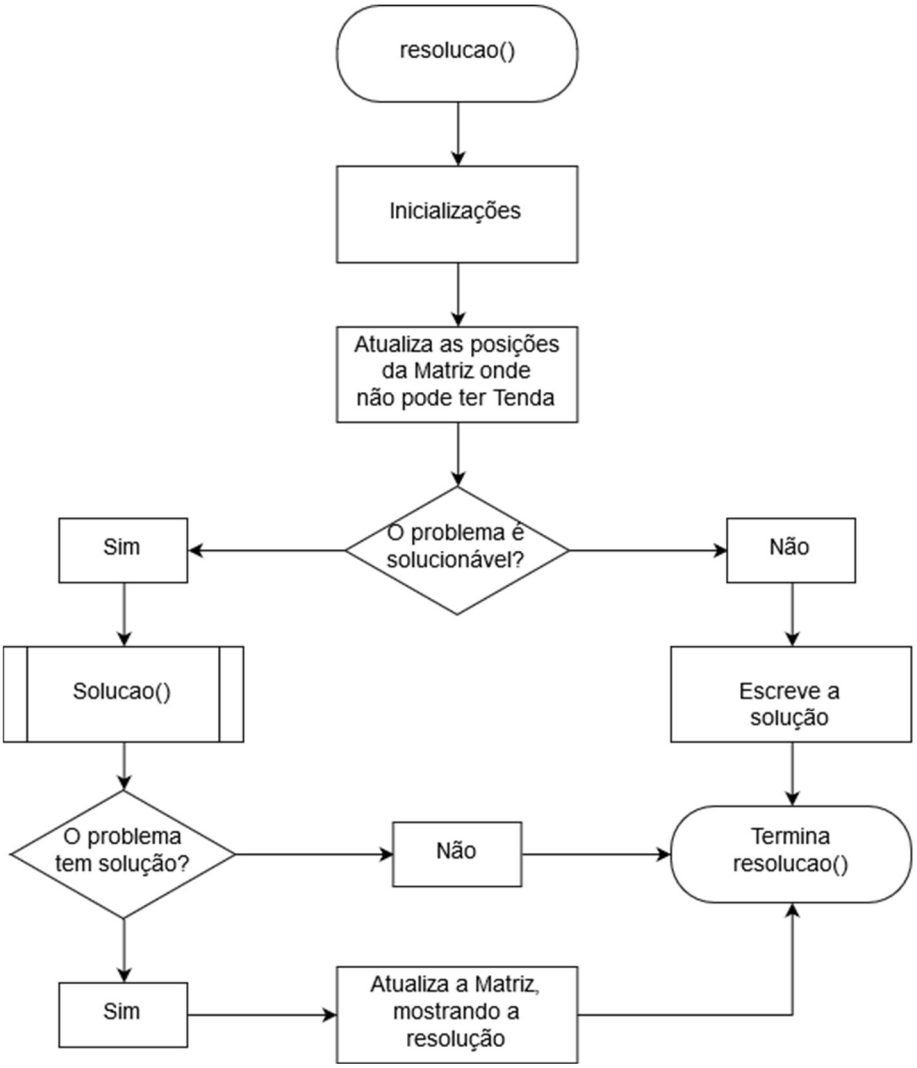
1) **Ficheiro**



O objetivo da função Ficheiro() é produzir o ficheiro final e realizar uma primeira análise do problema.
Depois das inicializações e da criação do novo ficheiro, a função começa a guardar informações do problema. Assim que os vetores das Linhas e das Colunas são guardados, verificamos a sua validade ou se produzem um problema anormal.
Quanto à validade, verificamos se: algum dos inteiros é negativo, se algum dos inteiros é maior do que o número de tendas possiveis segundo a dimensão dada ou se a soma de cada vetor fôr diferente. Se alguma destas situações se verificar, o programa imprime de imediato a resposta, não precisando de alocar memória para o mapa de árvores.
No que toca a problemas abnormais, consiste unicamente no caso em que tanto a soma das linhas e das colunas são ambas zero. Nesta situação, o problema encontra-se já resolvido, pelo que, não só imprimimos a solução, como também o mapa desta, que corresponde simplesmente ao mapa fornecido.

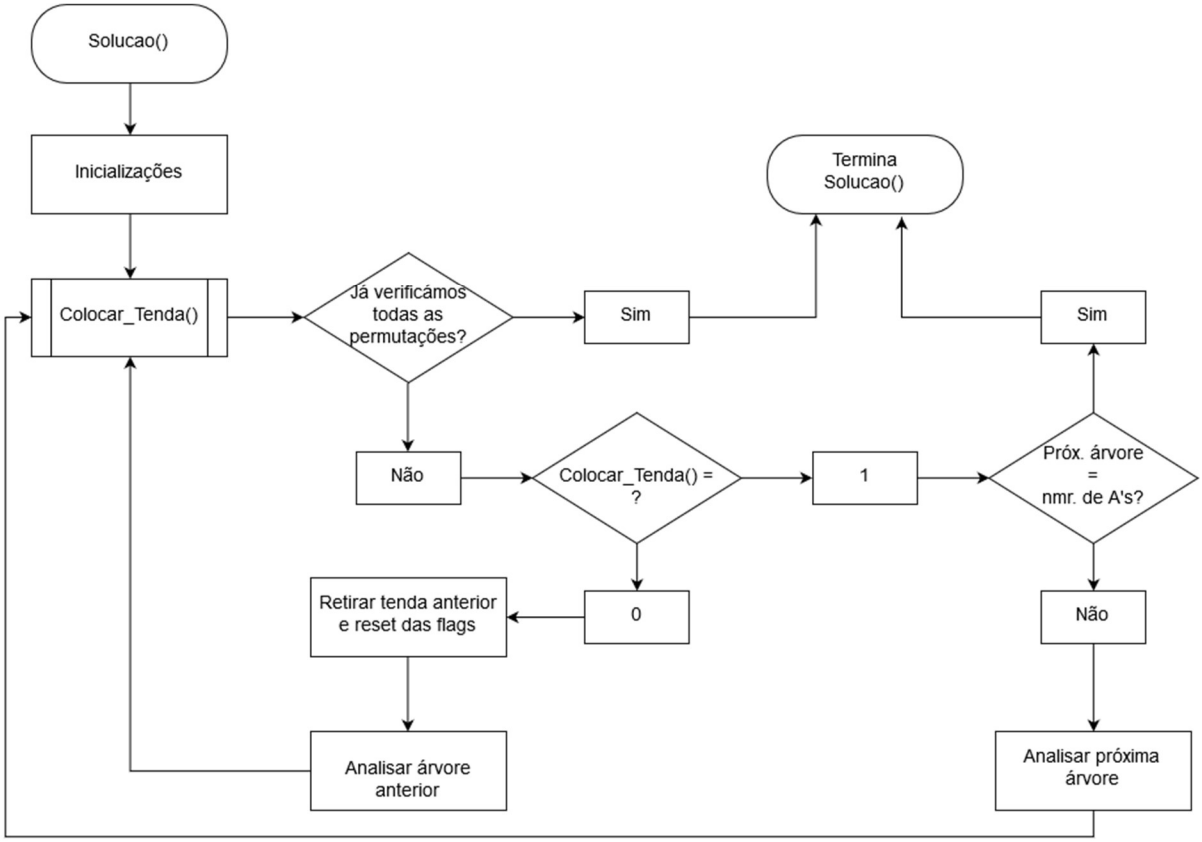
Se nenhuma destas situações ocorrer, o problema corre como deve, sendo alocada memória para o mapa, que depois é fornecido como argumento para a função resolucao(), que tenta por sua vez dar resposta ao problema. A verificação e o chamamento da função resolucao() encontram-se num loop, uma vez que um único ficheiro pode conter diversos problemas.

2) Resolução



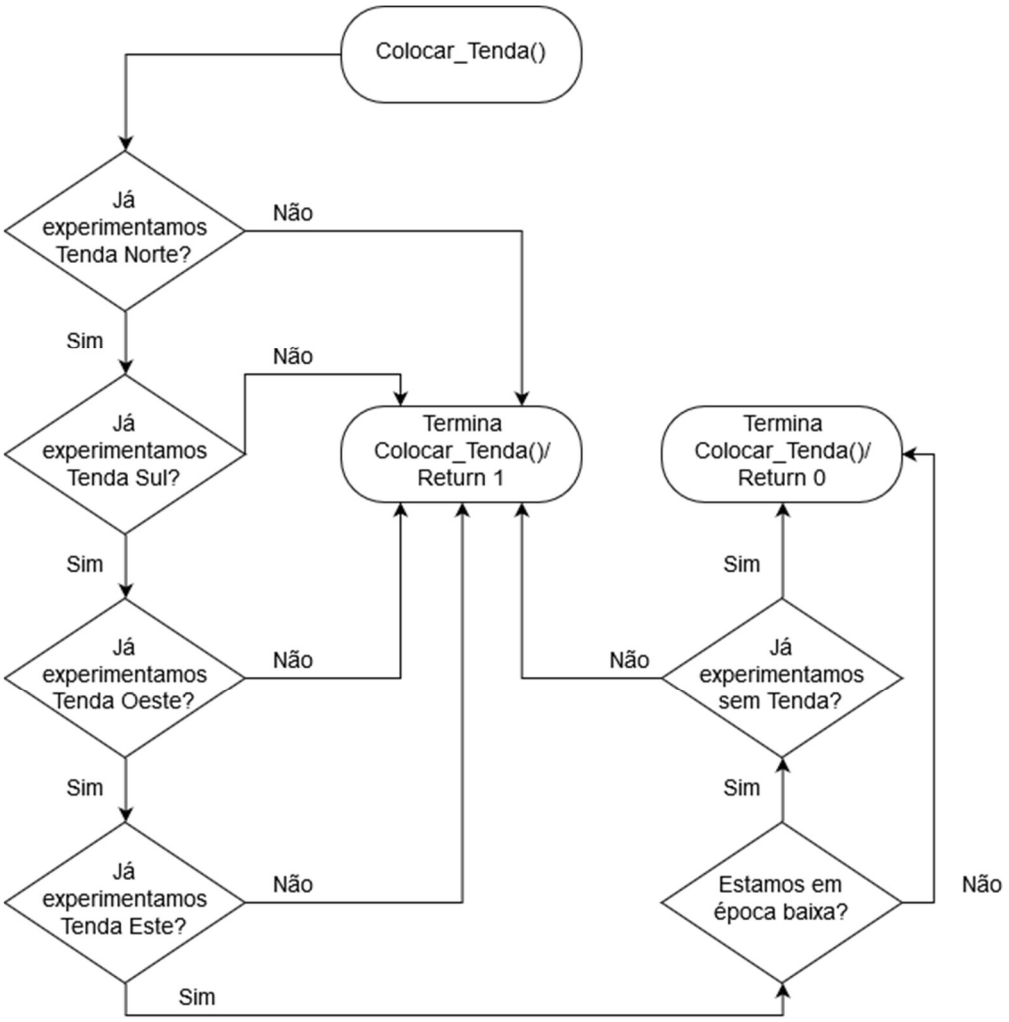
Nesta função verificamos se o mapa tem ou não solução ao nível do seu conteúdo, isto é, começamos por verificar se as árvores existentes chegam para o número de tendas pedidas uma vez que cada tenda necessita de uma árvore e cada árvore só pode ter uma e somente uma tenda. Se o número de árvores for menor ou número de tendas pedidas então o problema não tem solução.
Se a condição anterior se não se verificar, o número de árvores é maior (época baixa) ou igual (época alta) ao número de tendas, passamos para um segundo testes que consiste em contabilizar os espaços livre no mapa, ou seja, quantos são os lugares que provavelmente poderão ser tenda, isto é concretizado pela função ProblemaB, que irá analisar os vetores de linhas e colunas e os espaços adjacentes relativamente a uma localização na mapa para verificar as restrições da colocação de tendas do problema. Se o número de espaços livres for menor ao número de tendas então o problema não tem solução.
Em simultâneo ao teste anteriormente referido, sempre que encontramos uma árvore no mapa analisamos se esta é uma árvore valida, isto é, se esta tem espaços livres nas quatro principais células adjacentes (Norte, Sul, Oeste, Este). Se a árvore não tiver espaços livres não a contabilizamos e "eliminamos" do mapa substituindo 'A' por 'I', caso a árvore tenha alguma das quatro principais células adjacentes como espaço livre contabilizamos. Se o número de árvores validas for menor que o número de tendas pedidas o problema não irá ter solução.
Se o número de árvores disponíveis for maior, e pela subtração das árvores validas as tendas sabemos quantas árvores sem tenda são permitidas (época baixa), ou igual as tendas pedidas o problema pode ter solução e este último teste irá reduzir o espaço de procura da solução ou inexistência desta.
Se os testes anteriormente referidos falharem então é necessário procurar a solução ou inexistência desta, para isso chamos a função Solução que ira dar a resposta ao problema.
De seguida então é escrito no ficheiro de saída a resposta, se tiver solução escrevemos também o mapa com a respetiva solução.

3) Solução



Nesta função começamos por alocar a estrutura Vect_Tendas, que vai ser um vetor com a dimensão do número de árvores válidas, servindo como auxilio a procura da solução, cada índice no vetor irá guardar a localização de uma árvore valida no mapa, as flags (N, S, O, E, V), que registam quais as localizações possíveis para cada tenda já foi testadas ou não, por fim também guarda a localização da tenda colocada.
Para encontrarmos e guardarmos as árvores validas percorremos o mapa e inicializamos as flags a 0.
De seguida entramos num ciclo onde se chama a função colocar_tenda para cada índice do vetor. Com a resposta desta função decidimos que passo se deve tomar.
Se a resposta for negativa, ou seja não foi possível colocar tenda, e estamos no índice 0 do vetor é porque o problema não tem solução, caso o índice seja diferente de 0 então é necessário "andar para trás" para mudar a localização da última tenda colocada, primeiro inicializamos as flags do índice atual, de seguida necessitamos e ir ao índice anterior e se estivermos em época alta apagamos a tenda que foi colocada, se estivermos em época baixa e a flag V (significa que a última tentativa de colocar tenda foi não colocar) for diferente de 1 (possibilidade ainda não testada) então é porque colocamos tenda e é necessário apaga-la e subtraímos 1 a variável n que contabiliza em que posição do vetor me encontro e voltamos a chamar a função colocara_tenda.
Se a resposta da função colocar_tenda for positiva então incrementamos a variável n e chamamos a função colocar_tenda para esse índice do vetor de estrutura.
Quando a variável n for igual ao número de árvores disponíveis é porque já conseguimos colocar tenda em todas as árvores o que significa que o problema tem solução e já foi encontrada.

4) Colocar_Tenda



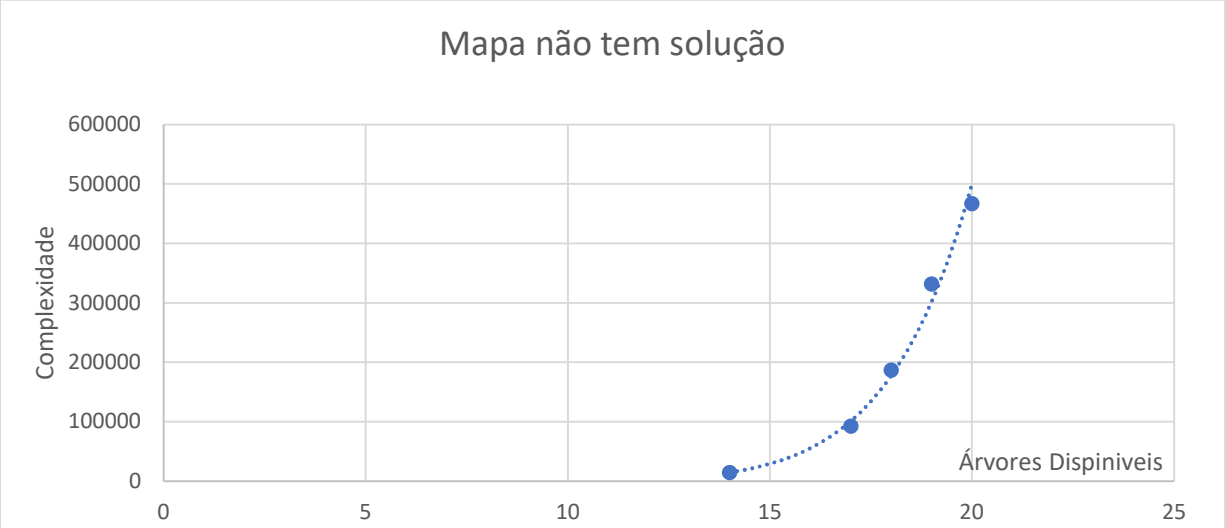
O objetivo da função Colocar_Tenda() resume-se em analisar as posições em redor da árvore em questão e decidir onde colocar uma Tenda. A função analisa se pode colocar tenda num padrão já predefinido, sendo ele: Norte, Sul, Oeste, Este e, se em época baixa, sem pôr Tenda.

A função baseia-se à volta de colocar uma tenda e assinalar onde se colocou, para mais tarde ser colocada noutra posição (se necessário). O processo de análise de uma posição consiste em várias condições para verificar se pode ou não colocar tenda sem quebrar as regras. Analisa se nunca lá foi colocada uma tenda numa permutação anterior, se essa posição existe (devido às árvores no perímetro do mapa), se não existem tendas adjacentes a essa posição ou se o vetor das linhas ou das colunas permite que uma tenda seja colocada na posição em questão. Se todas as condições se verificarem, é colocada uma Tenda na posição, e é retornado "1".

Se estivermos em época baixa, temos mais uma opção, não colocar Tenda. Neste caso, verificamos também se, em permutações anteriores, não experimentá-mos sem Tenda. Em seguida, verificamos se, ao colocar uma Tenda, não excede o número de àrvores sem Tendas associadas. Só depois de verificadas as condições é que podemos retornar "1". Em ambas as épocas, se não for possível verificar uma das opções, aí é retornado "0".

Complexidade

- Quando o mapa não tem solução (sendo admissível e chegando a fase de procura)
  A complexidade irá ser 5^n em época baixa uma vez que cada árvore disponível irá ter 5 possibilidades para a colocação de tenda e 4^n quando estamos em época alta.



Exemplo

É de interesse comtemplar um exemplo possível e verificar como o código se comporta ao tentar resolvê-lo. Para o efeito, veremos o problema seguinte:

4 4

0 1 1 1

0 0 2 1

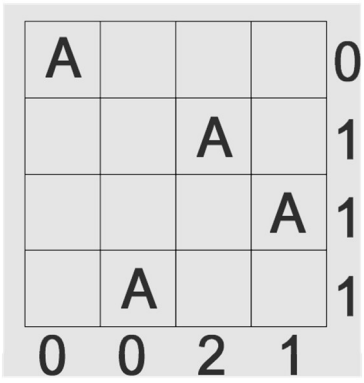
A...

..A.

...A

.A..

Cuja matriz, em teoria, estaria representada da seguinte forma:



Em primeiro lugar, o algorítmo altera a matriz, de forma a ser mais fácil a identificação das árvore "disponíveis"(isto é, árvores à qual uma tenda pode ser associada). A árvore na posição (0,0) será então substituída por I e as posições nas quais não se pode colocar uma tenda são marcadas como N:

I	N	N	N	0
N	N	A		1
N	N		A	1
N	A			1
0	0	2	1	

Como próximo passo, o algorítmo começa por colocar tendas na posição (1,3) (associada à árvore (1,2)). Porém, quando verificamos os redores da árvore (2,3), nenhuma das posições pode ter uma tenda, pelo que o programa volta à tenda (1,2) e verifica se pode colocar numa outra posição.

I	N	N	N	0
N	N	A	x	1
N	N	T	A	1
N	A		x	1
0	0	2	1	

I	N	N	N	0
N	N	A	T	1
N	N		A	1
N	A			1
0	0	2	1	

Porém, quando mais uma vez, avançamos para a árvore (2,3), verificamos outra vez que não existem posições viáveis onde colocar uma tenda, e uma vez que já extinguimos todas as opção de colocar tenda da primeira árvore "disponível" do mapa, admitimos que o problema não tem solução.