Métodos de Programação – Turma: A
Andre Garrido Damaceno – 15/0117531

TRABALHO 4:
RELATÓRIO DOS TESTES.

1) A: TESTE DE COBERTURA DE INSTRUÇÕES UTILIZANDO O GCOV

Os arquivos que mostram a cobertura de instruções do gcov encontram-se no diretório ‘./20_perguntas_MP/src/Cobertura Gcov’, sendo que cada arquivo dentro dessa pasta corresponde ao relatório de cada módulo do jogo. Todos os módulos tiveram a cobertura completa de todas as funções (com exceção de 2 linhas de código do módulo ‘Arvore.c’ e 1 linha de código do módulo ‘Funcs.c’, que são apenas executados quando um ‘malloc’ não é bem-sucedido).

1) B: TESTE DE DECISÕES COBRINDO TODAS AS POSSIBILIDADES BOOLEANAS

Módulo ‘Arvore.c’

- **Função: Constroi_Manual**

1) Condição da linha 78:

```
...
if(size < 20)  //Verifica se ja foram inseridas 20 perguntas em um nó
...
```

   - Quais as possibilidades:

     O valor ‘size’ pode ser menor que 20, ou igual a 20 (valores maiores que 20 não são aceitos pela assertiva de entrada). Quando ‘size’ é igual a 20, o nó da árvore é null (isso é feito no ‘else’ da condição).

   - Como os testes foram feitos:

     Os testes abrangem os valores que são menores e iguais a 20 (valores maiores que 20 não são aceitos pelas assertivas de entrada). Essa condição foi testada pelo teste ‘TEST_CASE("Creating a tree from user input", "Prove that the tree is created")’.
     O primeiro caso de teste faz uma inserção de um nó com o ‘size’ = 20, sendo que o teste passa, pois, o nó retorna como null (não é feita a inserção). No segundo caso de teste são feitas várias inserções com um nó ‘size’ menor que 20, sendo que o teste passa, pois, o nó retorna com uma árvore criada.

2) Do-while da linha 84:

```
...
do
   {
    /*
      Esse loop é feito para evitar que '\n' seja inserido como pergunta.
    */
    fgets (pergunta, 99, stdin);
   } while(strcmp(pergunta,"\n") == 0);
...
```

- Quais as possibilidades:

  É feito sempre pelo menos uma vez a execução da função ‘fgets’. Caso o que foi digitado pelo usuário seja qualquer coisa diferente de ‘\n’, o loop termina, caso contrário, o loop continua até que seja digitado algum caractere válido.

- Como os testes foram feitos:

  Os testes abrangem o caso de o usuário escrever um enter (ou seja, ‘\n’) e em seguida escrever normalmente uma pergunta, e o caso do usuário de primeira escrever uma pergunta válida. Esses casos foram testados pelo teste ‘TEST_CASE("Creating a tree from user input", "Prove that the tree is created")’.
  O primeiro caso de teste é apenas passado uma pergunta válida, e o do-while não faz o loop. No segundo caso, é digitado um enter (‘\n’), em seguida digitada uma pergunta válida, e é verificado que o do-while executa o loop uma vez.

3) Condição da linha 94:

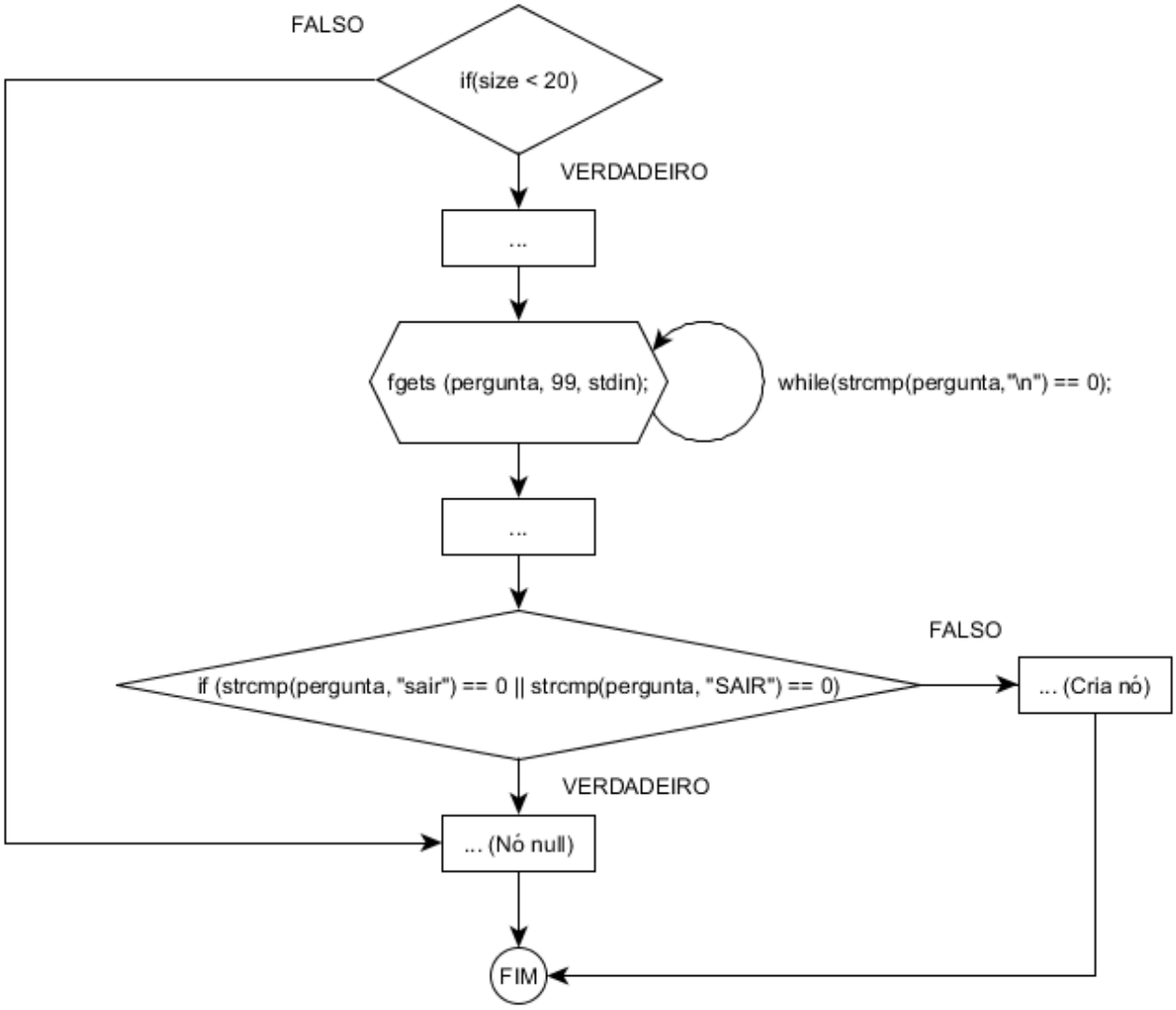
```
...
if (strcmp(pergunta, "sair") == 0 || strcmp(pergunta, "SAIR") == 0)
...
```

- Quais as possibilidades:

  A string ‘pergunta’ é equivalente a palavra ‘sair’ minúscula ou maiúscula executando a condição, ou a string ‘pergunta’ é qualquer outra palavra, executando assim o ‘else’ da condição.

- Como os testes foram feitos:

  Os testes abrangem o caso de o usuário escrever ‘sair’, verificando em seguida se a árvore é null (verificando assim o que ocorre dentro da condição), e o caso de o usuário escrever uma pergunta, verificando que o nó não é null (verificando assim o que ocorre na parte ‘else’ da condição). Esses casos foram testados pelo teste ‘TEST_CASE("Creating a tree from user input", "Prove that the tree is created")’.
  O primeiro caso de teste é apenas passado ‘sair’, assim verifica-se que o nó da árvore é null. No segundo caso, é digitado uma pergunta normal, e checado que o nó é criado normalmente.



- **Função: Constroi_TXT**

1) Condição da linha 164:
   ...
   if(arq!=NULL) //Verificação de arquivo .txt não existente
   ...

   - Quais as possibilidades:

     O arquivo 'arq' pode ser null (executando assim o 'else'), ou ser um arquivo válido.

   - Como os testes foram feitos:

     Os testes abrangem o caso de o usuário passar para função um arquivo válido (criando assim a árvore normalmente), e o caso de o usuário passar para função um arquivo inexistente (retornando assim uma árvore null). Esses casos foram testados pelos testes 'TEST_CASE("Creating a tree from a file", "Prove that the tree is created")', 'TEST_CASE("Trying to create a tree from an non existing file", "Prove that the tree is not created")'.
     O primeiro caso de teste é passado para a função um arquivo válido e existente, e o teste verifica todas as perguntas criadas na árvore e a estrutura da árvore (nós sim, não). No segundo caso, é apenas passado para a função um arquivo não existente, e verificado que a árvore retornada é null.

2) Condição da linha 168:

```
...
if(fgets (pergunta, 100, arq)!=NULL)  //Pega uma linha inteira do .txt (pergunta)
...
```

- Quais as possibilidades:

  A string lida em 'pergunta' pode existir ou ser o fim do arquivo (sendo null e executando o 'else').

- Como os testes foram feitos:

  Os testes abrangem o caso de no arquivo '.txt' ter perguntas existentes e a árvore ser criada normalmente, e de no final do arquivo '.txt', quando 'pergunta' é null, ser colocado 'null' no nó filho inexistente. Esses casos foram testados pelo teste 'TEST_CASE("Creating a tree from a file", "Prove that the tree is created")'.
  No caso de teste é passado para a função um arquivo '.txt' válido e existente, e o teste verifica todas as perguntas criadas na árvore e a estrutura da árvore (nós sim, não) e verifica-se que a árvore é criada normalmente, com os nós filhos inexistentes (do final da árvore) sendo null (que é o esperado).

3) Condição da linha 172:

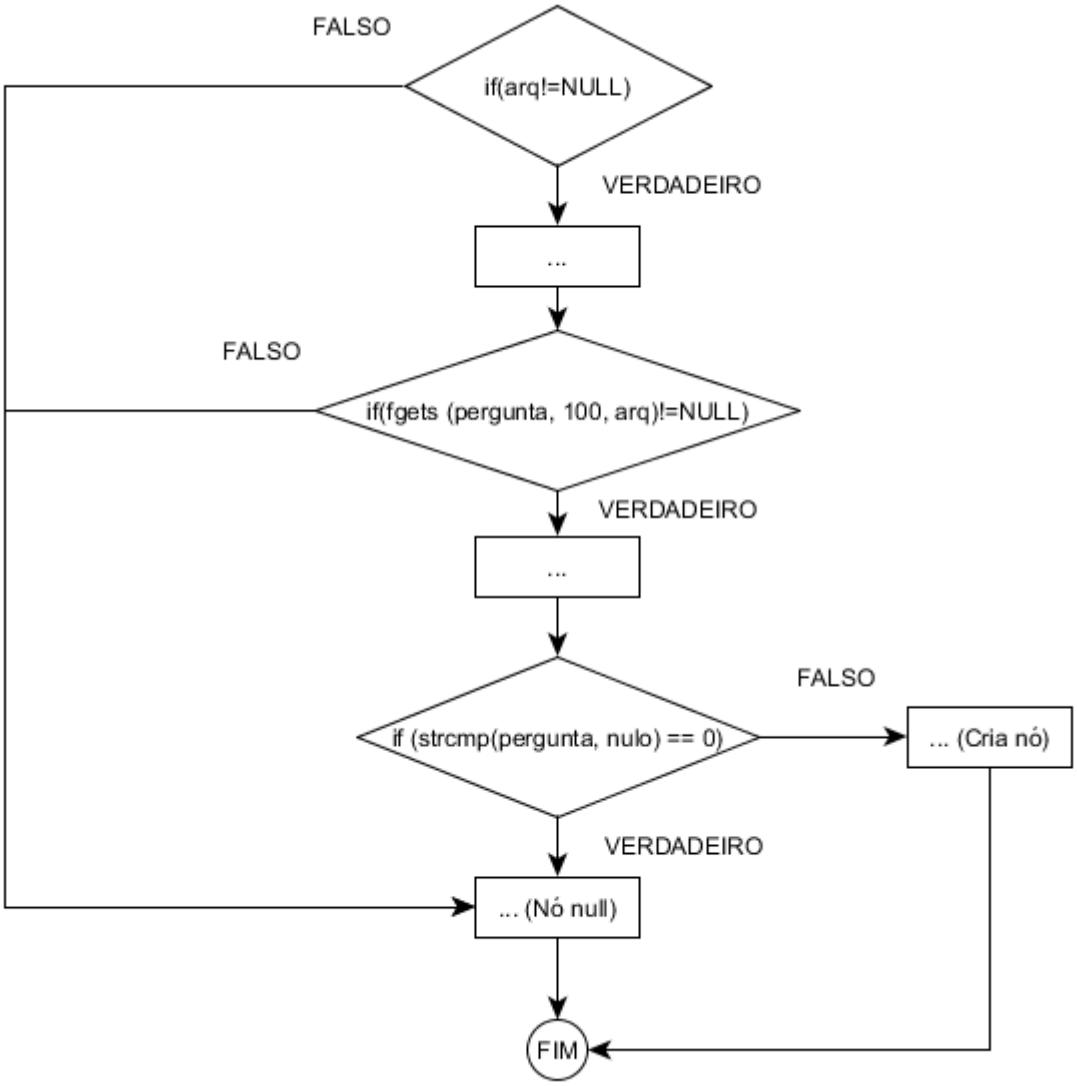
```
...
if (strcmp(pergunta, nulo) == 0)  //Caso haja um '.' no .txt, criar nó nulo
...
```

- Quais as possibilidades:

  A string 'pergunta' é equivalente a um ponto '.' (que é o conteúdo do char 'nulo') executando a condição, ou a string 'pergunta' é qualquer outra palavra, executando assim o 'else' da condição.

- Como os testes foram feitos:

  Os testes abrangem o caso de no arquivo '.txt' conter um ponto '.', verificando em seguida se a árvore é null (verificando assim o que ocorre dentro da condição), e o caso de no arquivo '.txt' conter uma pergunta válida normal, verificando que o nó não é null (verificando assim o que ocorre na parte 'else' da condição). Esses casos foram testados pelo teste 'TEST_CASE("Creating a tree from a file", "Prove that the tree is created")'.
  No caso de teste é passado para a função um arquivo '.txt' válido e existente, e o teste verifica todas as perguntas criadas na árvore e a estrutura da árvore (nós sim, não) e verifica-se que a árvore é criada normalmente, com os nós sendo null nos lugares que o arquivo '.txt' possui um ponto '.' escrito na linha.



- **Função: Salva_TXT**

  1) Condição da linha 232:
     ...
     if(arq != NULL) //Checa se o arquivo .txt existe
     ...

     - Quais as possibilidades:

       O arquivo 'arq' pode ser null (dessa forma a função não faz nada e ignora a chamada), ou ser um arquivo válido.

     - Como os testes foram feitos:

       Os testes abrangem o caso de o usuário passar para função um arquivo válido (salvando assim a árvore normalmente), e o caso de o usuário passar para função um arquivo inexistente (função faz nada e ignora a chamada). Esses casos foram testados pelos testes 'TEST_CASE ("Saving tree to NULL file", "Prove that the function does nothing and contains the program")', 'TEST_CASE ("Saving a tree to file", "Prove that the txt saves the tree")'.
       O primeiro caso de teste é passado para a função um arquivo válido e existente, e uma árvore válida e existente. No teste é salva a árvore em um arquivo '.txt', em seguida é aberto o arquivo, recriando a árvore pelo arquivo salvo, e verificado todas as perguntas criadas na árvore e a

estrutura da árvore (nós sim, não). No segundo caso, é apenas passado para a função um arquivo não existente, e verificado que a função não faz nada (não é criado nenhum arquivo novo).

2) Condição da linha 234:

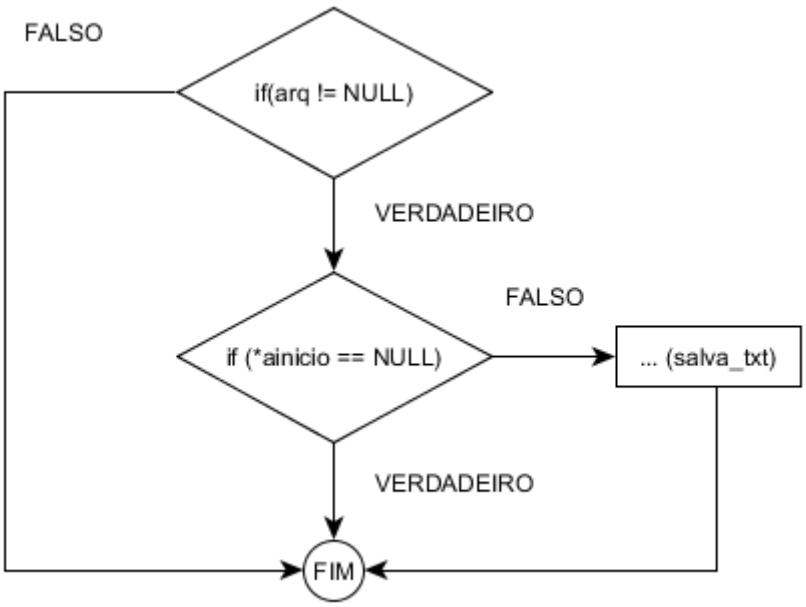
```
...
if (*ainicio == NULL)  //Caso o nó esteja em branco (NULL)
...
```

- Quais as possibilidades:

  O nó 'ainicio' é null, ou o nó 'ainicio' não é null (executando assim o 'else' da condição)

- Como os testes foram feitos:

  Os testes abrangem o caso de criação do arquivo '.txt' de acordo com a árvore, sendo colocado no arquivo ponto '.' nos locais em que a árvore é null. Esses casos foram testados pelos testes 'TEST_CASE ("Trying to navigate to save a NULL tree to file", "Prove that the txt saves '.'")', ' TEST_CASE ("Saving a tree to file", "Prove that the txt saves the tree")'.
  O primeiro caso de teste é passado para a função um arquivo válido e existente, e uma árvore válida e existente. No teste é salva a árvore em um arquivo '.txt', em seguida é aberto o arquivo, recriando a árvore pelo arquivo salvo, e verificado todas as perguntas criadas na árvore e a estrutura da árvore (nós sim, não) e as partes da árvore que devem ser null. No segundo caso, é passado para a função um arquivo válido e existente, e uma árvore null, e verificado que no arquivo '.txt' é salvo um ponto '.', e ao recriar a árvore pelo arquivo salvo, é criada uma árvore null.



- **Função: Le**

1) Condição da linha 275:

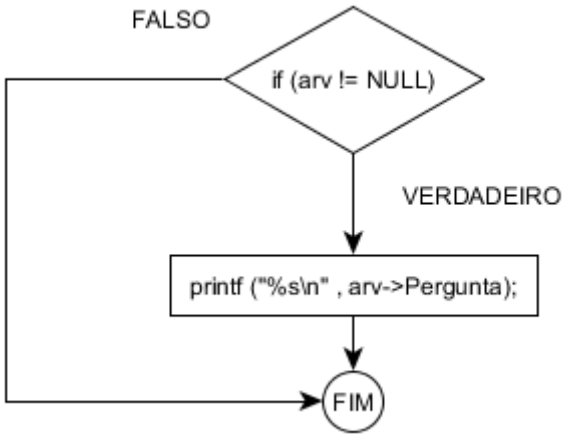
```
...
if (arv != NULL)  //Checa se o ponteiro é valido (diferente de NULL)
...
```

- Quais as possibilidades:

  O nó 'arv' é null (dessa forma a função não faz nada e ignora a chamada), ou o nó 'arv' não é null.

- Como os testes foram feitos:

  Os testes abrangem o caso de ser passado um nó null e um nó não null para a função. Esses casos foram testados pelos testes 'TEST_CASE("Reading a tree question", "tree is unmodified and question is read")', 'TEST_CASE("Trying to read NULL tree", "Program is contained and function does nothing")'.
  O primeiro caso é verificado que ao se passar um nó null a função não faz nada e nada é modificado. No segundo caso, ao passar um nó válido é verificado que a pergunta do nó é impressa na tela e nenhuma estrutura da árvore é alterada.



- **Função: NavegaSim**

1) Condição da linha 313:
   ...
   if((*pergunta) != NULL) //Checa se o nó é valido (não NULL)
   ...

   - Quais as possibilidades:

     O nó 'pergunta' é diferente de null (executando a condição) ou é null (dessa forma a função não faz nada e ignora a chamada).

   - Como os testes foram feitos:

     Os testes abrangem o caso de ser passado um nó null e o caso de se ter passado um nó não null. Esses casos foram testados pelos testes 'TEST_CASE("Trying to navigate to '->sim' and '->nao' and reading the question", "Tree goes to specific navigation and reads the question")', 'TEST_CASE("Trying to navigate to '->sim' and '->nao' whith NULL tree", "Program is contained and function returns 2")'.
     O primeiro caso é verificado que ao se passar um nó null a função faz nada. No segundo caso, ao passar um nó válido, a função continua sua execução normalmente e se passar na próxima condição o nó é navegado para seu filho 'sim'.

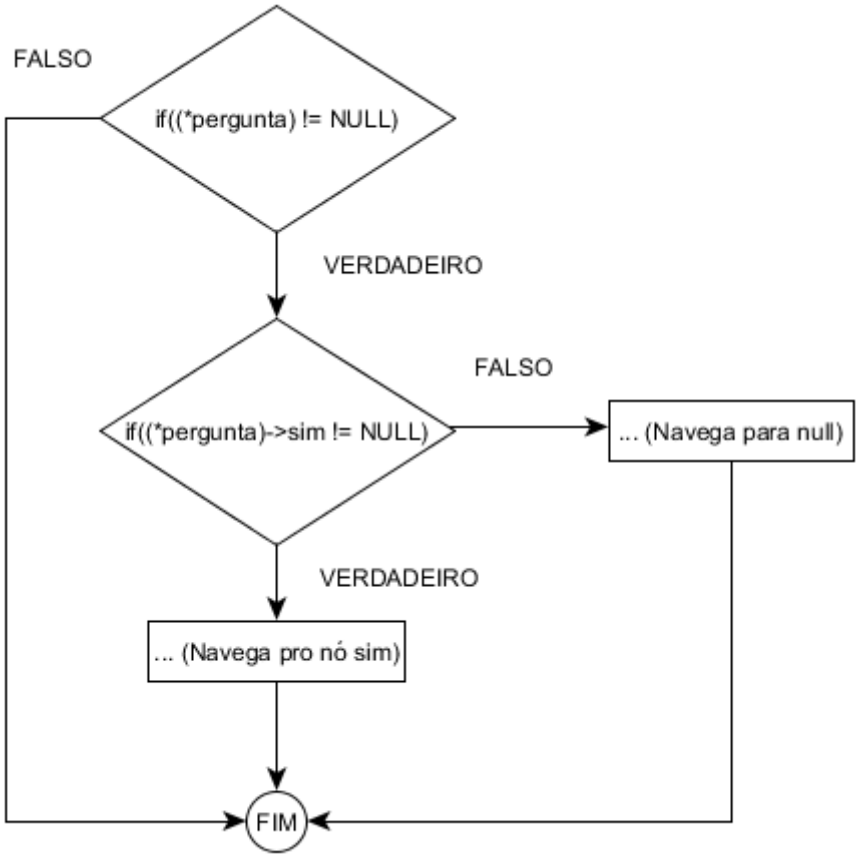
2) Condição da linha 315:
   ...
   if((*pergunta)->sim != NULL) //Checa se o nó->sim é valido (não NULL)
   ...

   - Quais as possibilidades:

     O nó 'pergunta->sim' é diferente de null (executando a condição) ou é null (dessa forma a função navega para null).

- Como os testes foram feitos:

  Os testes abrangem o caso de ser passado um nó não null e o nó->sim não null, e o caso de se ter passado um nó não null e o nó->sim null. Esses casos foram testados pelos testes 'TEST_CASE("Trying to navigate to '->sim' and '->nao' and reading the question", "Tree goes to specific navigation and reads the question")'.
  O primeiro caso é verificado que ao se passar um nó não null e o nó->sim não null a função navega para o nó->sim normalmente. No segundo caso, ao passar um nó não null e o nó->sim null, a função navega para null.



- **Função: NavegaNao**

1) Condição da linha 358:
   ...
   if((*pergunta) != NULL) //Checa se o nó é valido (não NULL)
   ...

   - Quais as possibilidades:

     O nó 'pergunta' é diferente de null (executando a condição) ou é null (dessa forma a função não faz nada e ignora a chamada).

   - Como os testes foram feitos:

     Os testes abrangem o caso de ser passado um nó null e o caso de se ter passado um nó não null. Esses casos foram testados pelos testes 'TEST_CASE("Trying to navigate to '->sim' and '->nao' and reading the question", "Tree goes to specific navigation and reads the question")', 'TEST_CASE("Trying to navigate to '->sim' and '->nao' whith NULL tree", "Program is contained and function returns 2")'.

O primeiro caso é verificado que ao se passar um nó null a função faz nada. No segundo caso, ao passar um nó válido, a função continua sua execução normalmente e se passar na próxima condição o nó é navegado para seu filho 'nao'.

2) Condição da linha 360:

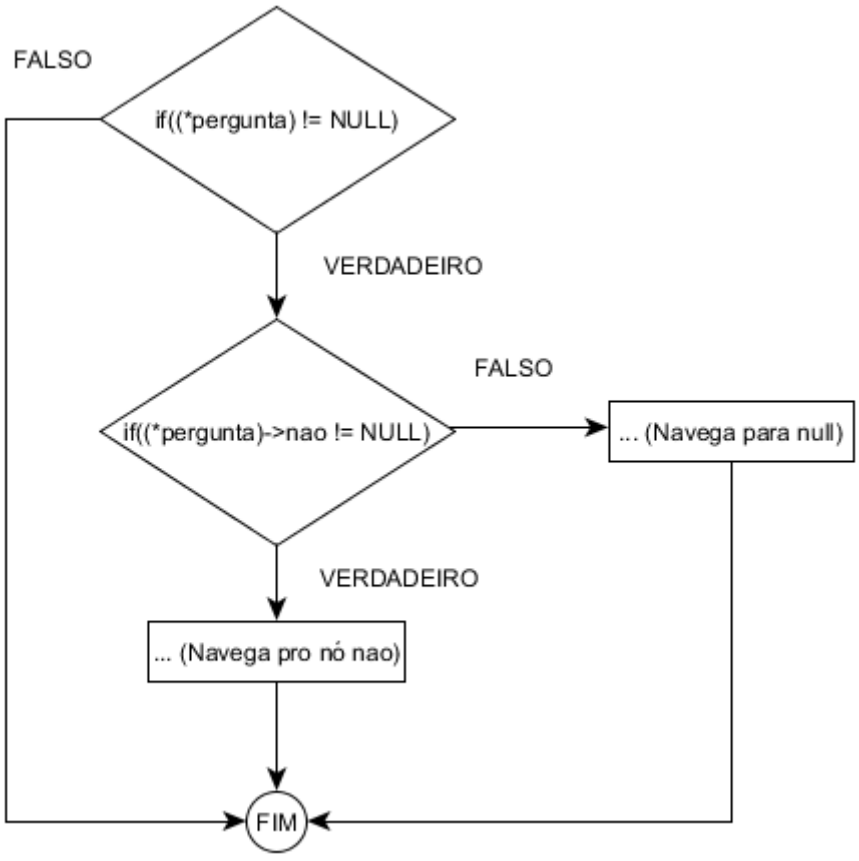
```
...
if((*pergunta)->nao != NULL)  //Checa se o nó->'nao' é valido (não NULL)
...
```

- Quais as possibilidades:

  O nó 'pergunta->nao' é diferente de null (executando a condição) ou é null (dessa forma a função navega para null).

- Como os testes foram feitos:

  Os testes abrangem o caso de ser passado um nó não null e o nó->nao não null, e o caso de se ter passado um nó não null e o nó->nao null. Esses casos foram testados pelos testes 'TEST_CASE("Trying to navigate to '->sim' and '->nao' and reading the question", "Tree goes to specific navigation and reads the question")'.
  O primeiro caso é verificado que ao se passar um nó não null e o nó-> nao não null a função navega para o nó-> nao normalmente. No segundo caso, ao passar um nó não null e o nó-> nao null, a função navega para null.



- **Função: Desconstroi**

1) Condição da linha 402:

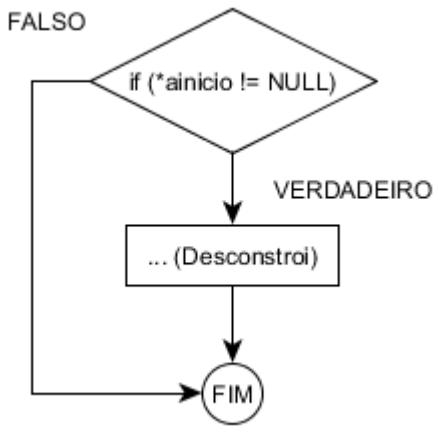
```
...
if (*ainicio != NULL)  //Checa se o nó é valido (não NULL)
...
```

- Quais as possibilidades:

  O nó ‘ainicio’ pode ser diferente de null (entrando na condição) ou ser igual a null (não entrando na condição).

- Como os testes foram feitos:

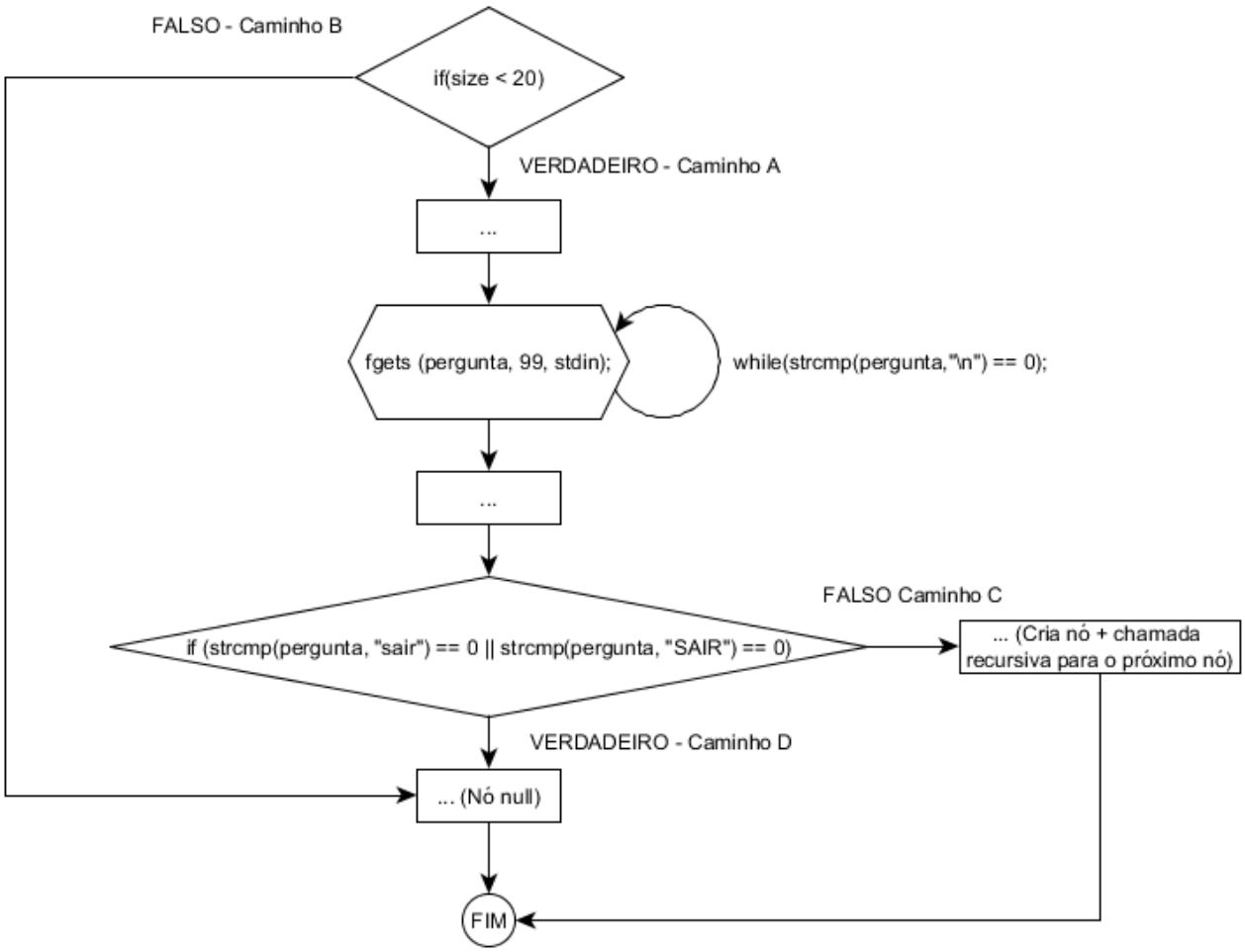
  Os testes abrangem o caso de entrar na função um nó não null e entrar na função um nó null. Esses casos foram testados pelos testes ‘TEST_CASE ("Freeing an existing tree", "the tree is freed")’, ‘TEST_CASE ("Freeing a NULL tree", "the program is contained")’.
  O primeiro caso é verificado que ao se passar um nó não null, deve-se ter um nó null no final (pois a árvore será apagada). No segundo caso, ao se passar um nó null, verifica-se que o nó permanece sendo null (pois nada foi feito).



1) C: TESTE DE CAMINHOS

Módulo 'Arvore.c'

- **Função: Constroi_Manual**



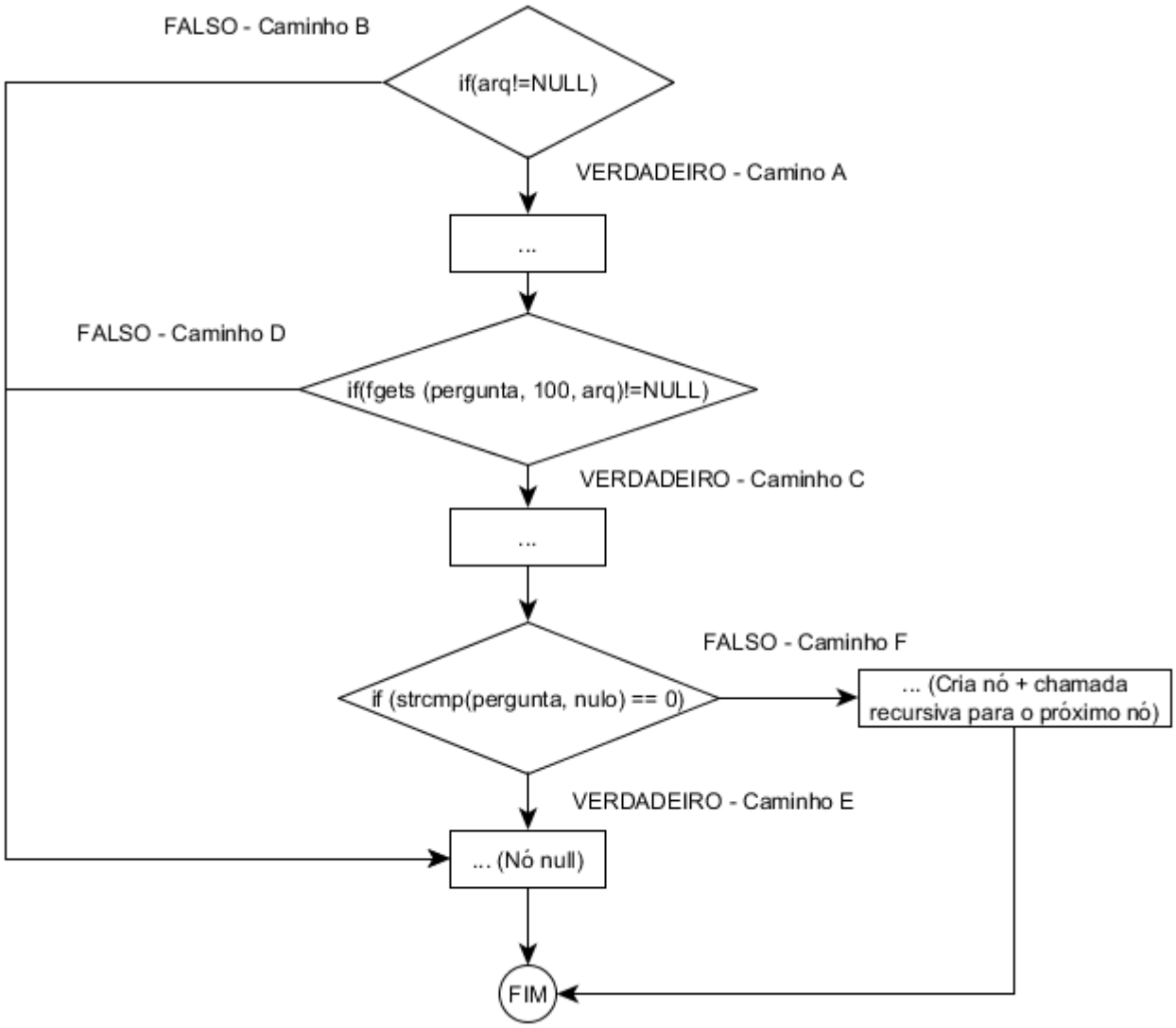
De acordo com o esquema acima, são possíveis os seguintes caminhos:

- B
- A – D
- A – C – B
- A – C – A – D
- Outras variações recursivas

O teste que cobre todas as possibilidades mencionadas acima é o seguinte: 'TEST_CASE("Creating a tree from user input", "Prove that the tree is created")'.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis que a função pode ter (como chamadas recursivas). No teste, são populadas 7 árvores binárias diferentes (com a função constrói_manual), e testado cada caminho especificado acima, verificando os nós criados (se realmente foram criados ou se são nulls), o tamanho e a estrutura da árvore. Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na construção da árvore binária, testando grande parte das possíveis ocorrências na criação de uma árvore do jogo, para verificar se o comportamento da função é igual ao comportamento da especificação.

- **Função: Constroi_TXT**



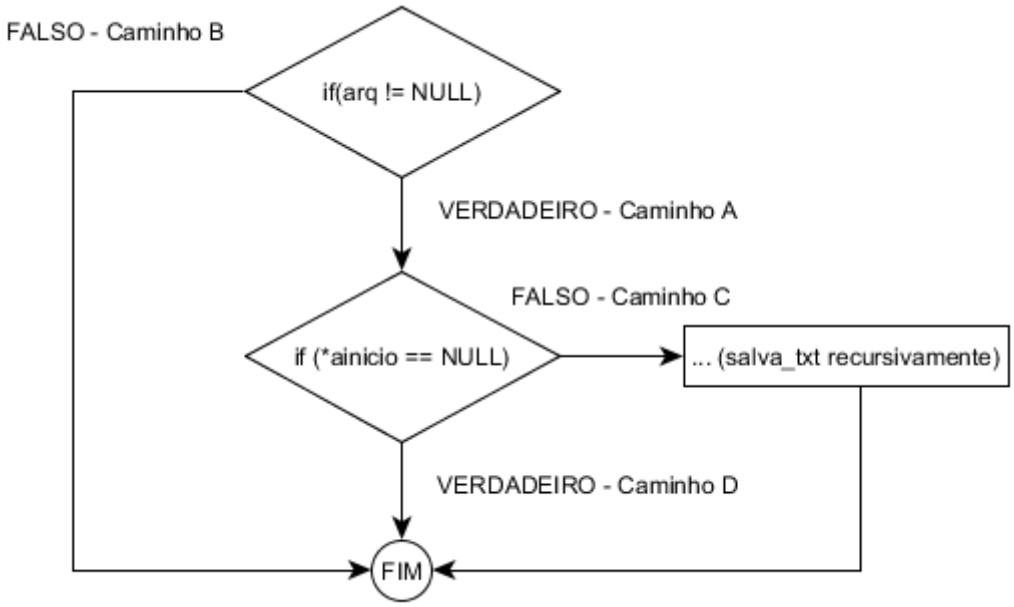
De acordo com o esquema acima, são possíveis os seguintes caminhos:

- B
- A – D
- A – C – E
- A – C – F – A – C – E
- A – C – F – A – D
- Outras variações recursivas

Os testes que cobrem todas as possibilidades mencionadas acima são os seguintes: 'TEST_CASE("Creating a tree from a file", "Prove that the tree is created")', 'TEST_CASE("Trying to create a tree from an non existing file", "Prove that the tree is not created")'.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis que a função pode ter (como chamadas recursivas). No teste, são populadas 2 árvores binárias diferentes (com a função constrói_txt), e testado cada caminho especificado acima, verificando os nós criados (se realmente foram criados ou se são nulls), o tamanho, estrutura da árvore e as perguntas de todos os nós. Também foi testado o comportamento da função ao ter um arquivo null (caminho B). Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na construção da árvore binária a partir de um arquivo '.txt', testando grande parte das possíveis ocorrências na criação de uma árvore do jogo, para verificar se o comportamento da função é igual ao comportamento da especificação.

- **Função: Salva_TXT**



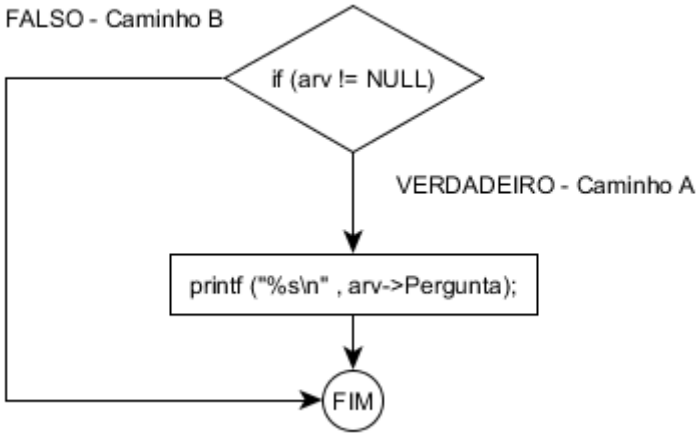
De acordo com o esquema acima, são possíveis os seguintes caminhos:

- B
- A – D
- A – C – A – D
- A – C – A – C – A – D
- Outras variações recursivas

Os testes que cobrem todas as possibilidades mencionadas acima são os seguintes: ‘TEST_CASE ("Trying to navigate to save a NULL tree to file", "Prove that the txt saves '.'")’, ‘TEST_CASE ("Saving a tree to file", "Prove that the txt saves the tree")’, ‘ TEST_CASE ("Saving tree to NULL file", "Prove that the function does nothing and contains the program")’.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis que a função pode ter (como chamadas recursivas). No teste, são populadas 3 árvores binárias diferentes, em seguida foram salvas essas árvores (com a função salva_txt), reaberta as árvores com a função constrói_txt e testado cada caminho especificado acima, verificando os nós da árvore salva e reaberta (se realmente foram criados ou se são nulls), o tamanho, estrutura da árvore e as perguntas de todos os nós, verificando assim que a árvore é salva corretamente. Também foi testado o comportamento da função ao ter um arquivo null (caminho B). Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na hora de salvar a árvore binária em um arquivo ‘.txt’, testando grande parte das possíveis ocorrências na hora de salvar uma árvore do jogo, para verificar se o comportamento da função é igual ao comportamento da especificação e garantir que a árvore seja salva corretamente, para que na hora de reabrir o jogo tudo seja feito de forma correta.

- **Função: Le**



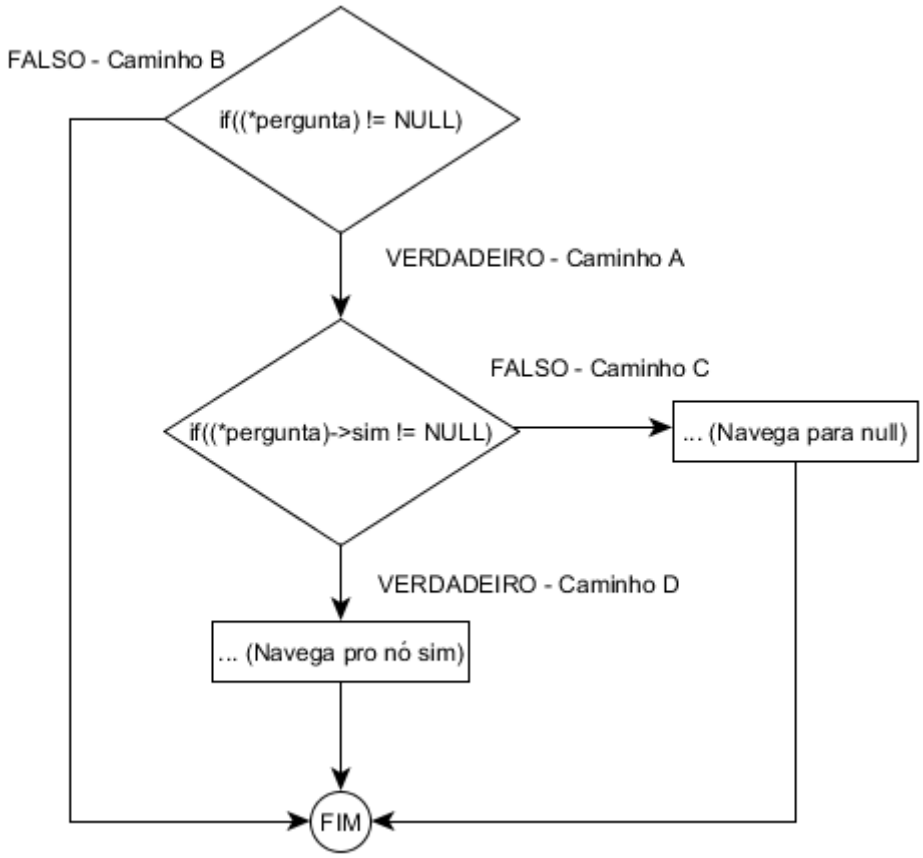
De acordo com o esquema acima, são possíveis os seguintes caminhos:

  - A
  - B

Os testes que cobrem todas as possibilidades mencionadas acima são os seguintes: 'TEST_CASE("Reading a tree question", "tree is unmodified and question is read")', 'TEST_CASE("Trying to read NULL tree", "Program is contained and function does nothing")'.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis observados pela estrutura da função. No teste, são populadas 2 árvores binárias diferentes (uma existente e uma null), em seguida foi chamada a função 'Le' para cada uma dessas árvores. Foi verificado que para a árvore existente a função leu a pergunta normalmente, e para a árvore null a função não fez nada, como esperado. Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na hora ler a pergunta da árvore binária, testando grande parte das possíveis ocorrências na hora de imprimir uma pergunta da árvore do jogo, para verificar se o comportamento da função é igual ao comportamento da especificação e garantir que a pergunta da árvore seja lida corretamente.

- **Função: NavegaSim**



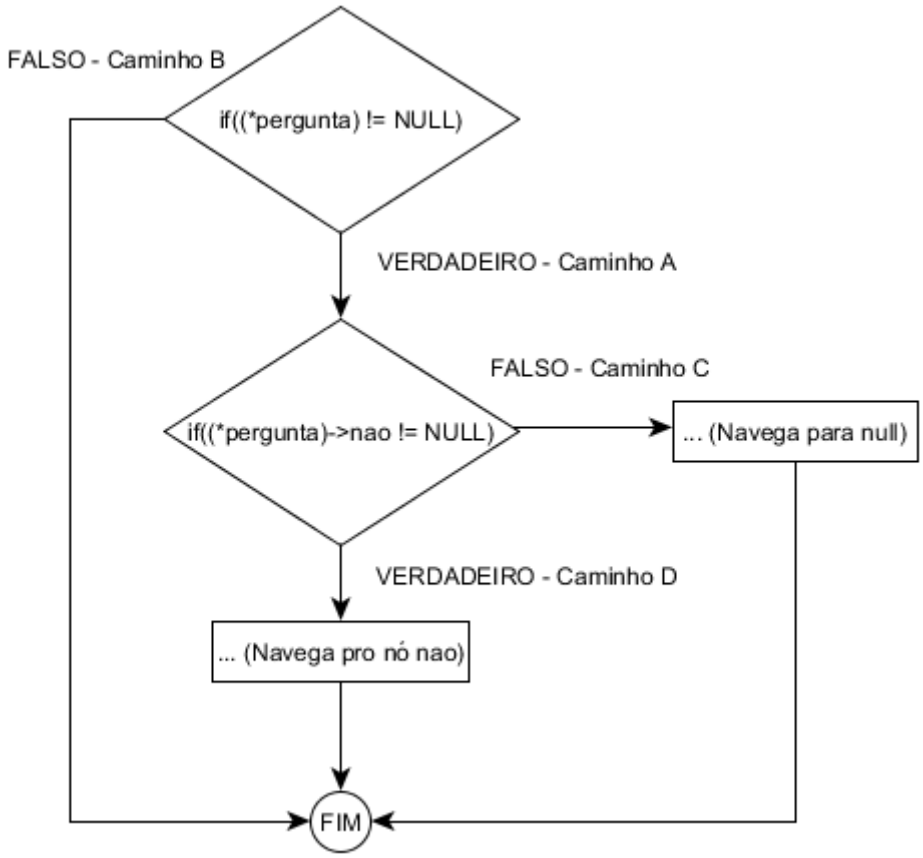
De acordo com o esquema acima, são possíveis os seguintes caminhos:

- B
- A – C
- A – D

Os testes que cobrem todas as possibilidades mencionadas acima são os seguintes: 'TEST_CASE("Trying to navigate to '->sim' and '->nao' and reading the question", "Tree goes to specific navigation and reads the question")', 'TEST_CASE("Trying to navigate to '->sim' and '->nao' whith NULL tree", "Program is contained and function returns 2")'.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis observados pela estrutura da função. No teste, são populadas 2 árvores binárias diferentes (uma existente e uma null), em seguida foi chamada a função 'NavegaSim' para cada uma dessas árvores (mais de uma vez para a árvore existente). Foi verificado que para a árvore existente a função navegou para o nó sim normalmente (ao comparar o retorno da função com o nó->sim da árvore), assim como no caso em que o nó->sim é null, e para a árvore null a função retornou null, como esperado. Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na hora de navegar para o nó sim da árvore binária, testando grande parte das possíveis ocorrências na hora de navegar para o nó sim da árvore do jogo, para verificar se o comportamento da função é igual ao comportamento da especificação e garantir que a árvore sempre navegue para o nó sim corretamente.

- **Função: NavegaNao**



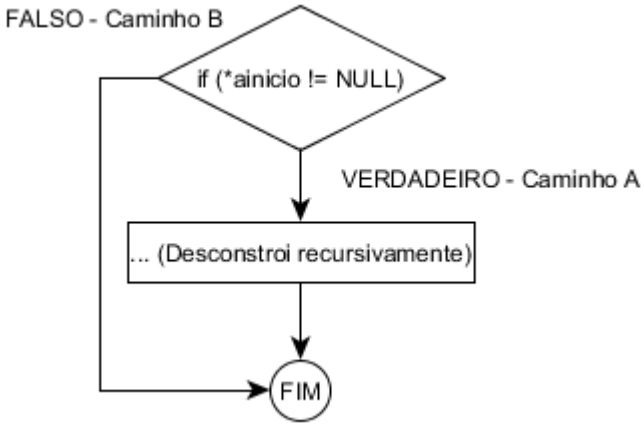
De acordo com o esquema acima, são possíveis os seguintes caminhos:

- B
- A – C
- A – D

Os testes que cobrem todas as possibilidades mencionadas acima são os seguintes: 'TEST_CASE("Trying to navigate to '->sim' and '->nao' and reading the question", "Tree goes to specific navigation and reads the question")', 'TEST_CASE("Trying to navigate to '->sim' and '->nao' whith NULL tree", "Program is contained and function returns 2")'.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis observados pela estrutura da função. No teste, são populadas 2 árvores binárias diferentes (uma existente e uma null), em seguida foi chamada a função 'NavegaNao' para cada uma dessas árvores (mais de uma vez para a árvore existente). Foi verificado que para a árvore existente a função navegou para o nó nao normalmente (ao comparar o retorno da função com o nó->nao da árvore), assim como no caso em que o nó->nao é null, e para a árvore null a função retornou null, como esperado. Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na hora de navegar para o nó nao da árvore binária, testando grande parte das possíveis ocorrências na hora de navegar para o nó nao da árvore do jogo, para verificar se o comportamento da função é igual ao comportamento da especificação e garantir que a árvore sempre navegue para o nó nao corretamente.

- **Função: Desconstroi**



De acordo com o esquema acima, são possíveis os seguintes caminhos:

- B
- A – B
- A – A – B
- Outras variações recursivas

O teste que cobre todas as possibilidades mencionadas acima são os seguintes: 'TEST_CASE ("Freeing an existing tree", "the tree is freed")', 'TEST_CASE ("Freeing a NULL tree", "the program is contained")'.

Pelos caminhos especificados na foto, o teste foi definido pelas possibilidades de todas as condições, como mencionado no item 1-B desse relatório e pelos caminhos possíveis que a função pode ter (como chamadas recursivas). No teste, são populadas 2 árvores binárias diferentes (uma existente e uma null), em seguida foi chamada a função 'Desconstroi' para cada uma dessas árvores. Foi verificado que para a árvore existente a função deletou todos os nós que a árvore possuía, e para a árvore null a função retornou null, como esperado. Esses testes têm o propósito (significado) de verificar todos os caminhos possíveis que o código pode percorrer na hora de deletar uma árvore binária, testando grande parte das possíveis ocorrências na hora de deletar a árvore do jogo (parcialmente ou completamente ao fim do jogo), para verificar se o comportamento da função é igual ao comportamento da especificação.

2) TABELA DE DECISÃO

Legenda:

condições
ações

***Código pode ser encontrado em './20_perguntas_MP/src/Arvore.c'.**

Módulo 'Arvore.c'

- **Função: Constroi_Manual**

'size' < 20	V	V	F
'pergunta' == sair	F	V	-
pergunta é pêga do usuário	V	V	F
arvore será null	F	V	V
arvore é criada com a pergunta pêga do usuário	V	F	F
chamada recursiva da função	V	F	F

*Do-while não se encontra na tabela tendo em vista que ele sempre é executado, e serve apenas para um auxílio quando o programa está em execução (quando o usuário clica no enter ('\n') acidentalmente e não insere nenhuma pergunta).

- **Função: Constroi_TXT**

'arq' != null	V	V	V	F
'pergunta' != null	V	V	F	-
'pergunta' != '.'	V	F	-	-
pergunta é pêga do arquivo '.txt'	V	V	F	F
arvore será null	F	V	F	V
arvore é criada com a pergunta do arquivo '.txt'	V	F	F	F
chamada recursiva da função	V	F	F	F

- **Função: Salva_TXT**

'arq' != null	V	V	F
'ainicio' == null	F	V	-
salva '.' no txt	F	V	F
salva a pergunta da arvore no txt	V	F	F
chamada recursiva da função	V	F	F

- **Função: Le**

'arv' != null	V	F
imprime pergunta da arvore	V	F

- **Função: NavegaSim**

'pergunta' != null	V	V	F
'pergunta->sim' != null	V	F	-
navega para 'pergunta->sim'	V	F	F
'pergunta' será igual a null	F	V	V

- **Função: NavegaNao**

‘pergunta’ != null	V	V	F
‘pergunta->nao’ != null	V	F	-
navega para ‘pergunta->nao’	V	F	F
‘pergunta’ será igual a null	F	V	V

- **Função: Desconstroi**

‘ainicio’ != null	V	F
chamada recursiva da função	V	F
libera memória alocada	V	F