Dynamic Software Testing

QCS-22/23-001-1.0

João Silva
Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologias da
Universidade de Coimbra
Coimbra, Portugal
joaosilvba@student.dei.uc.pt

Inês Marçal
Departamento de Engenharia Informática
Faculdade de Ciências e Tecnologias da
Universidade de Coimbra
Coimbra, Portugal
inesmarcal@student.dei.uc.pt

Índice

1) Introdução 3
1.1) Software alvo 3
1.1.1) Descrição 3
1.1.2) Implementação do software 4
1.1.3) Parâmetros e execução do software 6
1.2) Propósito do plano de testes 7
1.3) Abordagens a utilizar 7
1.4) Considerações 9
2) Problemas relacionados com riscos 9
2.1) Aspetos críticos 9
2.2) Potenciais riscos 10
2.3) Ações de mitigação 10
2.4) Consequências de testes incorretamente efetuados 10
3) A ser testado 11
3.1) Lista de itens a serem testados e Vista técnica 11
3.2) Identificar as features do end-user 11
5) Estratégia de teste 12
5.1) Técnicas a aplicar 13
5.2) Especificação de cada técnica 13
5.2.1) Control Flow Testing 13
5.2.2) Data Flow Testing 14
5.2.3) Equivalence Classes Partitioning 14
5.2.4) Boundary Value Analysis 14

5.2.5) Fuzzing 14
5.3) Configurações a serem testadas 15
5.4) Ferramentas utilizadas 15
5.5) Métricas de avaliação 15
6) Aplicação do plano de testes 15
6.1) White-box testing 16
6.1.1) Control Flow Testing 16
6.1.1.1) compress(self, str) 16
6.1.1.2) decompress(self, name) 22
6.1.2) Data Flow Testing 26
6.1.2.1) compress(self, str) 26
6.1.2.2) decompress(self, name) 33
6.2) Black-box testing 38
6.2.1) Equivalence Classes Partitioning 38
6.2.1.1) compress(self, str) 38
6.2.1.2) decompress(self, name) 46
6.2.2) Boundary Value Analysis 51
6.2.2.1) compress(self, str) 51
6.2.2.2) decompress(self, name) 52
6.2.4) Fuzzing 53
6.2.5) Considerações e outros aspetos 53
6.3) Test Coverage 54
6.3.1) White-box testing (fase de control flow testing) 54
6.3.1.1) compress(self, str) 54
6.3.1.2) decompress(self, name) 54
6.3.2) Black-box testing 54
6.3.3) Considerações finais 54
7) O que o plano de testes entregou 55
8) Necessidades para este plano de teste 55
9) Staffing e responsabilidades 55
10) Conclusão 55
11) Referências 56

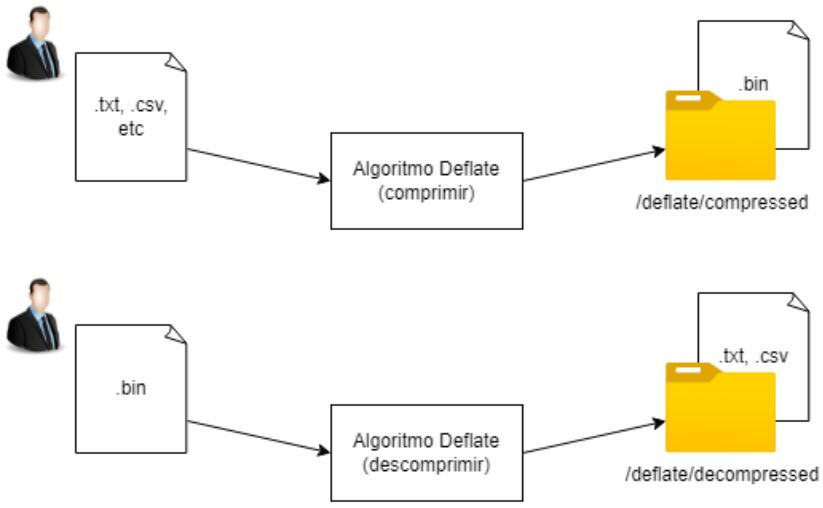
1) Introdução

Quando se fala em desenvolvimento de software, um dos tópicos mais importantes e relevantes a que se deve atender é a fase de testes do mesmo, já que é uma etapa fundamental para que o software venha a apresentar a qualidade e confiabilidade esperada. Tão importante como compreender esta fase é entender o seu propósito, assim como o alvo de testes e os requisitos que o mesmo deve seguir. Obtida esta informação torna-se, naturalmente, mais fácil definir o plano de testes, isto é, pontos alvo de análise, tipos de análise a efetuar, abordagens e ferramentas a utilizar.

1.1) Software alvo

1.1.1) Descrição

O software a ser testado será o algoritmo deflate (que no fundo combina os algoritmos LZ77 e Huffman Coding), um algoritmo de compressão/descompressão, neste caso aplicado a ficheiros de texto (.txt, .csv, etc). Por conseguinte, este é composto por duas funcionalidades, uma para comprimir e outra para descomprimir. De seguida, apresenta-se um esquema ilustrativo de como o mesmo funciona:



Ora, atendendo ao representado, quando um utilizador quer comprimir o seu ficheiro de texto, este insere o documento no algoritmo *deflate* (e as suas respectivas configurações) e o mesmo é comprimido num ficheiro binário, sendo guardado o resultado desta operação na diretoria "/deflate/compressed". Este último irá constar em 2 ficheiros binários, o texto comprimido e uma árvore *Huffman* gerada pelo algoritmo *Huffman Encoding*. Para descomprimir, o procedimento acaba por ser bastante semelhante. Neste caso, são providenciados como *input* as configurações (iguais às utilizadas no método de compressão) e os 2 ficheiros binários anteriormente armazenados. O algoritmo *deflate com base* nos *inputs* irá reconstruir o ficheiro original (consoante a extensão atribuída nas configurações), guardando o respetivo resultado na diretoria "/deflate/decompressed".

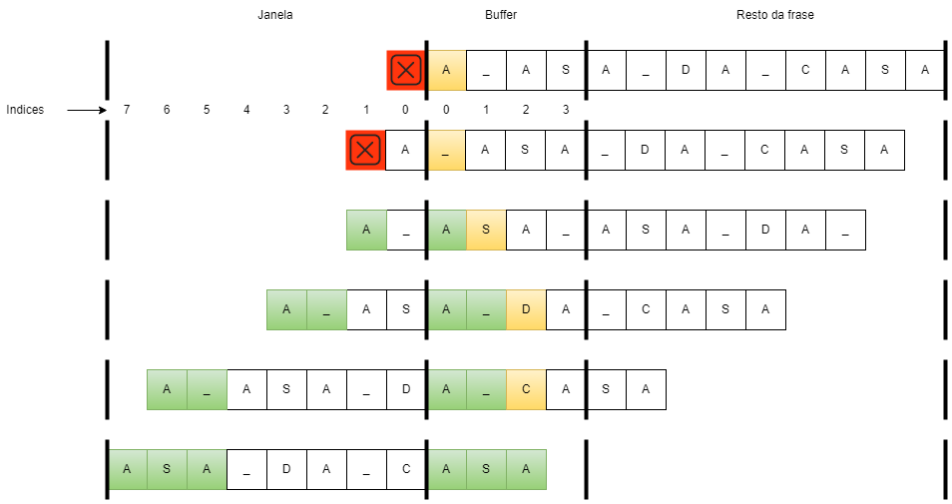
1.1.2) Implementação do software

Este tipo de algoritmos, normalmente, apresenta diversas implementações, variando principalmente nas operações efetuadas e parâmetros usados. É demonstrado, de seguida, o pseudo-código da implementação do método de compressão utilizado, onde *n* representa o tamanho do *buffer* e *m* o tamanho da janela:

```
Se o ficheiro não estiver vazio:
    dicionário = null
    buffer = n primeiros símbolos (tamanho n bytes)
    janela = null (tamanho m bytes)

    Enquanto existir símbolo para codificar:
        Se existir uma sequência presente no buffer que esteja também na janela:
            c = próximo símbolo depois da sequência
            Enviar código (i(sequência), len(sequencia), c)
            janela = janela + sequência + c
        Senão:
            c = primeiro símbolo no buffer
            Enviar código (0,0 c)
            janela = janela + c
        buffer = n primeiros símbolos depois de c

Aplicação do algoritmo Huffman Encoding ao resultado anterior
```

Este algoritmo demonstra ser bastante simples, sendo, tal como já mencionado, composto por 2 algoritmos: LZ77 e Huffman Enconding. De seguida, irá proceder-se à explicação dos mesmos detalhadamente.

O algoritmo LZ77 é composto por um buffer e uma janela, utilizados, por sua vez, para limitar a compressão que irá ser efetuada. Enquanto existam caracteres a codificar, é procurado por alguma sequência que esteja simultaneamente presente na janela e no buffer. Caso esta exista, o texto é codificado da seguinte forma: (índice do último caracter da sequência lida da direita para a esquerda na janela, tamanho da sequência encontrada, caracter que procede esta sequência no buffer). Caso a mesma não tenha sido encontrada, a codificação é efetuada da seguinte maneira: (0, 0, primeiro caracter do buffer lido da esquerda para a direita). Enquanto que na primeira opção, o buffer avança o número de caracteres da sequência encontrada mais 1 (caracter que segue a sequência), no segundo caso avança-se apenas para o próximo caracter. De seguida encontra-se esquematizado a aplicação do mesmo numa simples frase:

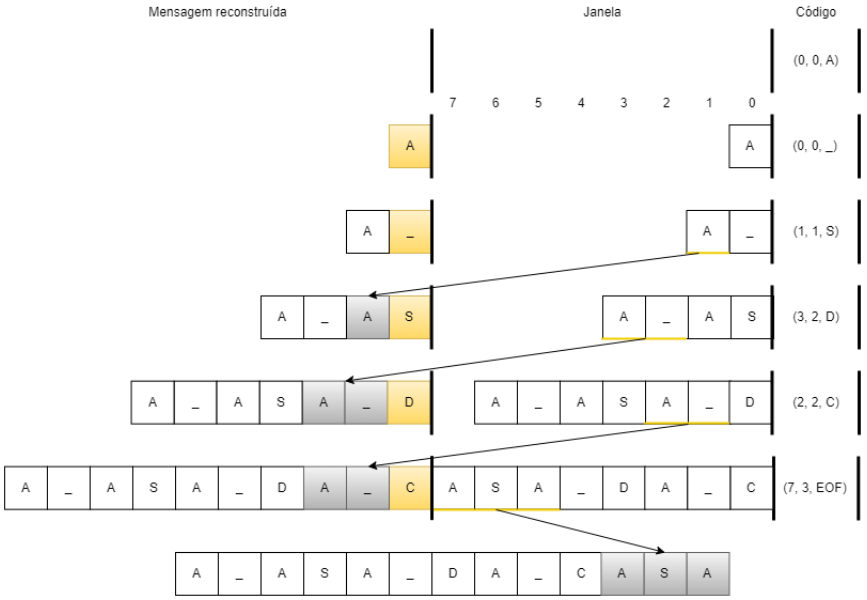


No exemplo acima, poderá observar-se que a frase "A_ASA_DA_CASA" é codificada no seguinte output: (0,0,A), (0,0, _), (1,1,S), (3,2,D), (2,2,C), (7,3,EOF).

Com vista a completar o processo de codificação utilizando o algoritmo LZ77 no contexto do Deflate, é necessário aplicar ainda o algoritmo Huffman Encoding. No software em causa, este é utilizado por meio de uma chamada a uma função externa, pelo que o modo de funcionamento do mesmo não irá ser especificado no plano de testes desenvolvido.

O processo de descompressão consistirá nas operações inversas às que foram anteriormente descritas. Em primeiro lugar, o texto é descodificado utilizando o descompressor do algoritmo Huffman Encoding, que, novamente, não irá ser especificado por se tratar de um método contido numa biblioteca externa.

Caso se pretenda reverter o algoritmo LZ77, o processo é bastante simples. Tendo como base o exemplo anterior:



Durante a descompressão procede-se ao seguinte:

- caso o comprimento da sequência codificada seja 0 (por exemplo, no tuplo (0,0,A) representa o valor do meio), a reconstrução da mensagem é simplesmente a adição da letra presente no tuplo (ou seja, no exemplo providenciado seria a letra A).
- perante o contrário, o primeiro número presente no tuplo indica o índice onde a sequência a reconstruir (presente na janela) irá começar, a qual terá um tamanho igual correspondente ao número central do tuplo. Para completar a reconstrução adiciona-se a letra presente no tuplo.

Para que seja possível compreender o modo de funcionamento e implementação do software, torna-se imprescindível entender o contexto de todas as operações executadas pelo mesmo, facilitando, em simultâneo, a criação do plano de testes a efetuar.

1.1.3) Parâmetros e execução do software

O software em causa é executado por meio da linha de comandos, onde também são fornecidos os argumentos para configurar o algoritmo Estes encontram-se detalhados de seguida:

- tamanho da janela: também conhecido como sliding window, determina o tamanho máximo dos dados comprimidos/codificados ao qual o algoritmo consegue referir. Este parâmetro é extremamente importante para o algoritmo LZ77 e deve ser especificado com um valor numérico.
- tamanho do buffer: também conhecido como lookahead buffer, indica a quantidade de dados que serão possivelmente comprimidos/codificados. Este parâmetro é de extrema importância para o algoritmo LZ77 e deve ser especificado com um valor numérico.
- flag a indicar operação: esta flag especifica se o ficheiro fornecido através do input irá ser comprimido ou descomprimido, sendo representada pelos valores "-compress" ou "-decompress".
- flag a on/off: permite indicar se o utilizador pretende ou não observar o output a comprimir ou descomprimir do passo efetuado pelo algoritmo LZ77, podendo tomar os valores "on" ou "off".
- nome do ficheiro: especifica o ficheiro a ser comprimido e descomprimido, o qual deverá ser representado por valores alfanuméricos.
- extensão: este parâmetro é utilizado apenas durante a descompressão de um ficheiro, de modo a indicar qual a extensão que se pretende para o mesmo. Este irá tomar valores alfanuméricos.

Compreender os parâmetros utilizados é um passo importante para a posterior elaboração de casos de teste, especialmente em abordagens como black-box testing, onde é necessário serem criadas classes de equivalência (válidas e inválidas). Este acaba por ser um ponto crucial também para determinar os possíveis fluxos de execução que um código pode seguir, tanto a nível de control flow testing como de data flow testing.

Para o software especificado ser executado deverá proceder-se à utilização dos seguintes comandos:

- Para comprimir:

```
[name-path]> python deflate.py <window-size> <buffer-size> -compress on/off <name-of-file>
```

- Para descomprimir:

```
[name-path]> python deflate.py <window-size> <buffer-size> -decompress on/off <name-of-file> <extension>
```

Como é possível observar, é a partir dos mesmos que são definidos os parâmetros do algoritmo. Estes comandos constituem, assim, os pontos de entrada para a execução de eventuais casos de teste que venham a ser criados.

1.2) Propósito do plano de testes

A criação de um plano de testes para o algoritmo *Deflate* tem como objetivo principal garantir a qualidade e confiabilidade do mesmo, já que este exige, durante a compressão e descompressão, uma implementação precisa, eficaz e sem falhas. Desenvolver um plano de testes permite, assim, validar o comportamento do *software* e garantir que o mesmo age de acordo com o esperado. Por conseguinte, torna-se possível avaliar se o software implementado apresenta um bom funcionamento a todos os níveis e, caso o mesmo falhe, se estão a ser utilizados os mecanismos corretos de recuperação.

Espera-se que durante a realização do plano de testes sejam igualmente encontrados os potenciais fluxos de execução do *software*, de modo a que seja possível compreender melhor o seu funcionamento. Com base nestas informações e juntamente com os *inputs* definidos para o *software* de teste, pretende-se elaborar casos de teste que permitam avaliar o comportamento do mesmo.

É esperado que este plano de testes contemple a exploração dos casos limite dos parâmetros anteriormente definidos, como por exemplo, tipo de ficheiros que o algoritmo *Deflate* suporta. A partir disto, torna-se possível avaliar a performance do *software*, assim como perceber as taxas de compressão apresentadas por este. Por fim, irá efetuar-se uma análise minuciosa dos resultados obtidos, de modo a entender-se as suas limitações e identificar maneiras de as ultrapassar e corrigir.

Resumindo, pretende-se através deste plano de testes atingir os seguintes objetivos:

- Verificar a funcionalidade do sistema*:* certificar que todos os recursos, funções e *workflows* estão implementados de acordo com o esperado e se seguem os requisitos previamente definidos.
- Identificar problemas: detectar *bugs*, erros e inconsistências no *software* e possíveis soluções.
- Avaliar o desempenho: determinar qual o desempenho do *software* em determinadas condições e quais as limitações do mesmo.
- Mitigar riscos*:* identificar potenciais riscos e falhas que possam ocorrer, reduzir o seu impacto ou até mesmo proceder à erradicação por completo dos mesmos.

1.3) Abordagens a utilizar

Durante a definição e aplicação deste plano de testes será dada especial atenção às abordagens dinâmicas, ou seja, que envolvem a execução do *software*. Dentro deste tipo de técnicas existem vários tipos de testes que podem ser efetuados, como: *unit testing, integration testing, system testing, coverage criteria, black e white box testing,* entre outros. O plano de testes irá, neste caso,

focar-se apenas em *white box testing, black box testing* e *testing coverage,* visto serem os mais adequados para o software em questão e os que fazem mais sentido dentro do seu contexto.

Começando por definir *white-box testing*, esta é uma abordagem que se baseia numa visão detalhada da estrutura e lógica interna do *software*, o que envolve conhecimento total acerca do código fonte. Esta contém as seguintes características:

- Examinar cuidadosamente os detalhes da implementação com vista a entender a forma como o programa foi desenvolvido.
- Testar as diferentes estruturas de controlo e dados presentes no programa com o intuito de verificar se funcionam corretamente.
- Os critérios de avaliação são bastante objetivos e baseados em estruturas do programa (*loops*, condições, chamadas de função, entre outros), bem como nas estruturas de dados utilizadas (arrays, listas, estruturas de registo).
- Parte do pressuposto de que o código-fonte esteja completamente alinhado com a especificação, ou seja, tenha sido implementado de acordo com todos os requisitos definidos.

Estes constituem os pontos mais importantes a reter durante a realização deste tipo de testes.

Já em *black-box testing*, o *software* é testado como se não houvesse qualquer conhecimento acerca da estrutura interna do mesmo, sendo analisados os seus aspetos fundamentais. Ao contrário de *white-box testing*, é importante existir um bom conhecimento acerca do funcionamento do *software* alvo e não do seu código fonte.

Este apresenta os seguintes objetivos:

- A derivação de casos de teste (conjuntos de *inputs* e respectivos *outputs* esperados) que explorem completamente a funcionalidade, do ponto de vista externo.
- Definir casos de teste eficazes que cubram de forma adequada o amplo espaço de possíveis entradas, considerando sua natureza potencialmente infinita.

Através de *black-box testing* pretende-se ainda encontrar diferentes tipos de erros em comparação com o *white-box testing*. Por exemplo:

- funcionalidades ausentes
- problemas de usabilidade
- problemas de desempenho
- erros de concorrência e temporização
- erros de inicialização e término
- entre outros.

Deste modo, durante o *black-box testing*, são realizados testes tanto do ponto de vista funcional quanto não funcional, avaliando aspectos como desempenho, *error handling*, a capacidade do *software* de executar ações conforme o esperado, entre outros critérios relevantes.

Por fim, o *testing coverage* é um tipo de testes que envolve, geralmente, 3 tarefas: *code coverage, data coverage* e *functional coverage*. Este é considerado uma métrica que avalia a adequação de um teste relativamente a um determinado *software*. Estas 3 métricas podem definidas da seguinte forma:

- code coverage: mede o nível de cobertura dos testes em relação ao código fonte, auxiliando na identificação de áreas do *software* que não tenham sido testadas.

- data coverage: mede o quanto os *inputs*/parâmetros utilizados nos casos de teste cobrem o *software*, ou seja, se a maioria dos cenários são testados (seja com *inputs* válidos ou inválidos).
- functional coverage: mede o quanto cada funcionalidade do *software* é testada e se os testes abrangem o maior número possível de funcionalidades, recursos e requisitos.

Este será um tipo de teste a ser utilizado principalmente em *black-box testing*, de modo a avaliar a cobertura dos casos de teste elaborados.

1.4) Considerações

Através desta primeira introdução do plano de testes, foi possível realizar uma breve descrição do *software* alvo dos mesmos, ou seja, compreender a sua implementação, respectivos parâmetros e modo de funcionamento. De seguida, abordou-se o propósito do plano de testes e as suas diferentes fases. Os tipos de testes a serem realizados foram mencionados de forma geral, sendo que serão detalhados mais adiante em capítulos subsequentes

Assim, foi possível obter uma visão condensada do plano de testes a executar e uma ideia dos procedimentos que se pretendem efetuar.

2) Problemas relacionados com riscos

A partir deste tópico pretende-se delimitar quais são os aspetos mais críticos do algoritmo *Deflate* implementado e identificar os potenciais riscos associados. Com base nos riscos identificados, serão apresentados problemas relacionados com os mesmos e o seu possível impacto quando o *software* for integrado em outros sistemas. É também importante referir que devem ser avaliadas as consequências resultantes da elaboração de testes incorretos, a fim de medir o nível de exigência requisitado durante os mesmos. Posteriormente, tentar-se-ão planear estratégias de mitigação caso venham a ser detectados riscos.

2.1) Aspetos críticos

Durante a elaboração do plano de testes, devem ser tidos em conta os seguintes aspetos:

- Algoritmo de compressão: o algoritmo implementado neste *software* é o *Deflate*, que permite comprimir ou descomprimir texto. É crucial verificar se a sua implementação está correta e se o algoritmo se comporta conforme o esperado.
- Taxa de compressão e performance: sendo este um algoritmo de compressão, espera-se que realmente seja aplicada compressão aos ficheiros fornecidos como *input*. Portanto, durante os testes, será interessante variar os parâmetros de configuração para determinar as taxas de compressão e desempenho que o algoritmo é capaz de alcançar, assegurando sempre se o faz corretamente.
- Lidar com diferentes ficheiros de texto: visto que a implementação deste algoritmo *Deflate* apenas se encontra preparada para lidar com ficheiros de texto.
- Impacto na performance: um aspeto importante a ter em conta neste tipo de *software* é o seu tempo de execução e a quantidade de recursos consumidos, de modo a não degradar a performance do ambiente onde este se integra.

2.2) Potenciais riscos

De seguida, são apresentados os potenciais riscos associados ao *software* a ser avaliado, caso este não tenha sido corretamente implementado:

- Corrupção ou perda de dados: caso seja detetado, provavelmente, constitui um dos riscos mais graves, já que a perda e/ou corrupção de dados é algo que não deve ser tolerado neste *Software*. Durante os testes, é essencial ser-se rigoroso na identificação de qualquer comportamento que possa levar a estas situações.
- Problemas de compatibilidade: o algoritmo *Deflate* deve ser compatível com quaisquer tipos de ficheiros de texto, como tal é necessário averiguar se não apresenta nenhum comportamento inconsistente ou desvio dependendo do tipo de ficheiro de texto.
- Degradação de performance: é importante que o *software* não afete o funcionamento do meio em que se encontra integrado, podendo comprometer o sistema em questões de performance.
- Falhas nas funcionalidades: outro risco significativo ao qual se deverá atender é o mal funcionamento do *software* implementado. Por exemplo, ao comprimir determinado ficheiro de texto, o tamanho em vez de diminuir, aumentar, ou até mesmo o ficheiro resultante da descompressão apresentar um tamanho diferente do ficheiro original.

2.3) Ações de mitigação

Para cada risco identificado acima, é possível associar um teste como medida de mitigação. Identificar estes riscos precocemente permite tomar medidas para lidar com os mesmos de forma mais eficaz.

- Testar algoritmo: envolveria a comparação com implementações do mesmo algoritmo e comparar os resultados com estas, de modo a averiguar se o *software* segue a especificação corretamente.
- Testar compatibilidade: testar alguns formatos de texto e avaliar se existe algum tipo de irregularidade no comportamento do *software*, caso seja encontrado é necessário corrigir as possíveis causas para tal acontecer.
- Testar performance: avaliar a performance da implementação do *Deflate* efetuada, caso os resultados não sejam satisfatórios, será necessário rever o algoritmo e identificar áreas que possam ser aprimoradas.
- Testar mecanismos de error handling e recuperação: compreender se num contexto em que sejam utilizados *inputs* inválidos ou quaisquer dados corruptos (ficheiros), o *software* lida adequadamente com as falhas que possam surgir e recuperar das mesmas.

2.4) Consequências de testes incorretamente efetuados

Os testes devem ser adequadamente definidos e alinhados com o contexto do *software* a testar, como já mencionado várias vezes. As consequências associadas a testes incorretamente efetuados encontram-se diretamente relacionadas com os potenciais riscos e problemas identificados. Portanto, é crucial estabelecer um plano de testes preciso para evitar quaisquer imprecisões no software em teste.

Resumindo, caso os testes não sejam corretamente efetuados poderá esperar-se as seguintes consequências, no futuro, durante a utilização deste software:

- incorreta implementação do algoritmo de compressão *Deflate*
- corrupção ou perda de ficheiros

- problemas quando utilizados certos tipos de ficheiros de texto
- a performance, tanto a nível de tempo de execução como no consumo de recursos, fica aquém do esperado
- funcionalidades de compressão e descompressão apresentarem desvios no comportamento esperado para os mesmos
- o sistema em redor do *software* ser afetado de forma pejorativa
- falta de mecanismos de e*rror handling* e recuperação em caso de falhas
- taxa de compressão insatisfatória

3) A ser testado

Visto que o alvo de testes em causa pode ser algo bastante extenso, é crucial compreender claramente os aspetos do mesmo que devem ser testados, de modo a que não sejam despendidos recursos e esforços em áreas de menor relevância. Deste modo, este capítulo irá incluir os seguintes pontos: identificação da lista de itens a serem testados, apresentação de uma vista técnica e identificação das features do *end-user*. O objetivo de cada um destes tópicos é o seguinte:

- identificar a lista de itens a serem testados: identificar claramente os componentes, módulos ou unidades a serem testadas. Especificar as versões e quaisquer requisitos de configuração para os itens a serem testados.
- apresentar uma vista técnica: identificar as unidades ou módulos a serem testados com base nos diferentes níveis considerados no plano de teste.
- identificar as features do end-user: fornecer uma descrição *high-level* das funcionalidades do sistema na perspetiva de um *end-use*r, assim como destacar os recursos e *features* relevantes para o mesmo.

3.1) Lista de itens a serem testados e Vista técnica

O *software* a ser testado é significativamente mais pequeno comparado com a grande maioria dos *softwares* existentes, sendo apenas constituído por uma função *main* e uma classe que, por sua vez, contém 3 métodos (funções) dentro de si. Os 3 métodos representam as seguintes funcionalidades: compressão, descompressão e verificação do *input* de um utilizador.

Dentro deste plano, o foco incidirá nos dois primeiros métodos, uma vez que os mesmos englobam os pontos-chave do *software* em causa e permitem o seu funcionamento. Pretende-se compreender se estes são capazes de efetuar a compressão e descompressão de ficheiros de textos de acordo com as especificações previamente apresentadas.

Outro aspecto a ser testado é a possível existência de limitações nesta implementação devido às bibliotecas utilizadas no seu desenvolvimento. Portanto, também será realizada uma breve análise ao *software* com o intuito de verificar a presença das mesmas.

Quanto à versão a testar, ao que tudo indica esta foi a única versão desenvolvida para este *software*, como tal é a única que irá ser averiguada.

3.2) Identificar as features do end-user

Como *features*, o *end-user* poderá usufruir diretamente de 3 possibilidades, sendo que duas delas estão relacionadas, configuração do algoritmo *Deflate* e possibilidade de descomprimir ou comprimir um ficheiro. Embora estas funcionalidades já tenham sido abordadas na introdução, será, novamente, fornecida uma definição destas:

- compressão: transformar um ficheiro de texto existente num ficheiro que ocupe menos espaço de memória, comprimindo, assim, o mesmo.
- descompressão: refere-se à operação inversa da compressão, na qual um ficheiro originalmente comprimido é restaurado, sendo esperado que o mesmo recupere o tamanho que tinha antes da compressão.
- configuração do algoritmo: outra funcionalidade permitida é a configuração do algoritmo *Deflate* antes deste efetuar uma eventual compressão ou descompressão. Isto permite a manipulação dos parâmetros mencionados anteriormente, permitindo ao utilizador personalizar o processo de compressão/descompressão de acordo com as suas preferências.

Como mencionado anteriormente, existem pelo menos duas destas funcionalidades a serem usadas, já que durante a execução do *software* em questão, tanto para efetuar compressão como descompressão, é sempre necessário configurar o algoritmo *Deflate*.

4) Não tão testado

As funcionalidades de verificação presentes no *main* do programa e no método de verificação de números inteiros, da classe mencionada anteriormente, foram excluídas da análise. Embora tenham sido colocadas de lado, estas acabam por ser testadas quando se proceder à avaliação do mecanismo de tratamento de erros do *software*, entendendo-se apenas que, neste caso, as mesmas estão fora do *scope* da análise de *white-box testing* e grande parte do *black-box testing*.

De seguida, apresenta-se uma descrição destas mesmas funcionalidades:

- verificações no main: é através destas que são verificadas alguns pontos essenciais do *software*, como o número de parâmetros introduzidos e se os mesmos apresentam os valores corretos.
- verificação dos números introduzidos: parâmetros como o tamanho da janela e do *buffer* são números introduzidos pelo utilizador, convertido mais tarde para um inteiro. Como tal, o *software* testado apresenta um método para verificar se realmente o *input* introduzido pelo utilizador é um número.

Assim, ainda que estas funcionalidades sejam essenciais para o correto funcionamento do algoritmo *Deflate*, testes como *control flow testing* ou *data flow testing* não serão aplicados às mesmas. Tal acontece, visto o ponto fulcral de teste incidir nas funcionalidades principais do algoritmo, avaliando se estas manifestam o comportamento esperado e analisando a sua complexidade e performance.

5) Estratégia de teste

Neste capítulo, pretende-se fornecer uma visão mais detalhada da estratégia para o plano de testes, abordando de forma mais aprofundada o que será realizado e como será executado. Assim, de seguida, serão apresentados diversos pontos importantes a serem mencionados neste plano de testes, como, por exemplo, técnicas utilizadas em cada ponto anteriormente definido (*white-box*, *black-box*, *testing coverage*), requisitos de *coverage*, configurações do *software* testadas, ferramentas usadas e métricas de avaliação.

5.1) Técnicas a aplicar

Como referido, este plano estará dividido essencialmente em 3 abordagens, *white-box*, *black-box* e *testing coverage*, as quais já foram previamente referenciadas e explicadas. Dentro de cada uma destas estratégias irão ser aplicadas técnicas específicas para avaliar e validar o alvo de testes em questão.

Uma vez contidas no *white-box testing* serão utilizadas as seguintes técnicas:

- control flow testing: é uma estratégia de teste estrutural que utiliza o fluxo de controlo do programa como modelo. O objetivo do *control flow testing* é garantir a cobertura adequada dos diferentes caminhos dentro do código-fonte, a fim de identificar possíveis falhas, erros lógicos ou comportamentos indesejados no programa.
- data flow testing: é uma técnica de teste de *software* que se concentra na análise e verificação do fluxo dos dados dentro de um programa ou sistema, ou seja, baseia-se na compreensão de como os valores dos mesmo são obtidos, modificados e usados ao longo da execução do *software*.

Relativamente a *black-box testing*, serão utilizadas as seguintes técnicas:

- equivalence classes partitioning: é uma técnica usada para selecionar os casos de teste a partir de um conjunto potencialmente infinito de possíveis casos de teste. O objetivo é reduzir a quantidade de testes necessários, mantendo a cobertura adequada do programa.
- boundary value analysis: é uma estratégia que se concentra em testar os limites dos *inputs* de um *software*, assim como os valores próximos a esses limites. Através desta abordagem será possível identificar falhas e erros que possam ocorrer perto destas regiões críticas.
- fuzzing: utilização de uma grande quantidade de *inputs* inseridos num determinado programa ou *software*, de modo a testar o comportamento do mesmo face aos parâmetros utilizados.

Finalmente, irá abordar-se a questão de *testing coverage*, o qual já foi amplamente explicitado em capítulos anteriores. Portanto, não será novamente explicado, uma vez não haver mais nada a acrescentar acerca do conceito.

5.2) Especificação de cada técnica

Para cada técnica enumerada será especificado em mais detalhe tanto os objetivos que se pretendem atingir com a mesma, como também a definição anteriormente fornecida. Por conseguinte, este capítulo visa eliminar quaisquer dúvidas que possam ter surgido durante a elaboração do plano de testes.

5.2.1) Control Flow Testing

O *control flow testing* favorece caminhos mais simples e diretos em detrimento de caminhos complexos e menos frequentes. Isto ocorre, porque a abordagem utilizada pelos primeiros tende a facilitar a sua compreensão e, consequentemente, a abranger um maior número de casos de teste, aumentando a probabilidade de encontrar erros.

Ao utilizar o *control flow testing*, é possível identificar áreas críticas do código que não são acionadas durante a execução normal do programa, o que pode indicar a presença de eventuais

falhas. Além disso, esta estratégia de teste também pode ajudar a entender melhor o fluxo de execução do programa, melhorando a qualidade do código e facilitando a sua manutenção.

5.2.2) Data Flow Testing

Nesse tipo de teste, o grafo de fluxo de controlo do programa é anotado com informações sobre como as variáveis do programa são definidas e usadas. Isto permite identificar as dependências entre as variáveis e rastrear o fluxo de dados de uma instrução para outra. Este tem como objetivo encontrar possíveis problemas, como variáveis não inicializadas, uso incorreto de valores de variáveis ou dependências não atendidas.

Ainda dentro do *data flow testing* existem diversas técnicas que podem ser aplicadas, mais especificamente *all-defs*, *all-c-uses*, *all-p-uses* e *all-du-paths*. De seguida, é explicitado o significado de cada uma destas:

- all-defs: identificar todos os pontos onde variáveis são definidas.
- all-p-uses: identificar onde são utilizadas variáveis dentro de condições e assegurar que as mesmas foram previamente inicializadas.
- all-c-uses: identificar onde são modificadas variáveis e assegurar que as mesmas foram previamente inicializadas.
- all-du-paths: para cada variável determinar o caminho em que a mesma é definida, modificada ou usada.

5.2.3) Equivalence Classes Partitioning

A ideia é agrupar os casos de teste em conjuntos de "classes de equivalência". Cada classe contém um conjunto de casos de teste que são considerados equivalentes, ou seja, espera-se que o programa processe todos esses casos de maneira semelhante, seguindo o mesmo caminho através do código.

Durante esta técnica é importante considerar diversos fatores, como limites de intervalos, valores válidos e inválidos, requisitos específicos e comportamentos esperados em diferentes situações. Além disso, é necessário validar se os casos de teste escolhidos são realmente representativos das classes de equivalência.

5.2.4) Boundary Value Analysis

A ideia por detrás desta estratégia é que muitos dos erros encontrados em *software* ocorrem frequentemente quando os *inputs* estão bastante próximos dos seus limites, visto que isso pode desencadear comportamentos inesperados. Portanto, testar estes limites aumenta a probabilidade de encontrar falhas no *software* em questão.

5.2.5) Fuzzing

O processo de *fuzzing* consiste em enviar uma variedade de *inputs* válidos e inválidos. Estes *inputs* são criados de forma aleatória, modificando os *inputs* já existentes ou gerando completamente novos. O objetivo é sobrecarregar o programa com uma grande quantidade de entradas inesperadas e observar o seu comportamento em busca de falhas.

Para realizar o *fuzzing*, são utilizadas ferramentas específicas de teste de *fuzz*, que automatizam o processo de geração e envio de *inputs* malformados para o *software*.

5.3) Configurações a serem testadas

Este ponto acaba por ter sido referenciado, em parte, em outros capítulos, já que as configurações do software em questão estão contidas no *input* fornecido pelo próprio *end-user*. Assim, as configurações a serem testadas são os próprios parâmetros do algoritmo, que serão avaliados para identificar possíveis anomalias no *software* alvo.

5.4) Ferramentas utilizadas

Para a realização deste plano de testes foi apenas utilizada uma ferramenta ao nível *white-box testing*, mais especificamente na fase de *control flow testing*, chamada ***pycfg***. Através desta ferramenta foi possível obter os grafos de fluxo de controlo das duas funcionalidades a serem testadas (*compress* e *decompress*).

Quanto às restantes fases não foi possível encontrar quaisquer ferramentas para as mesmas, pelo que foram testadas manualmente:

- *white-box testing*, fase de *data flow testing*
- *black-box testing*
- *test-coverage*

5.5) Métricas de avaliação

Para a realização deste plano de testes foram utilizadas diversas métricas de avaliação. As mesmas encontram-se especificadas de seguida:

- *white-box testing*
  - *control flow testing*
    - V(G): fornece o número de caminhos independentes e, consequentemente, um limite superior para o número de casos de teste necessários para testar a execução do *software*
  - *data flow testing* (esta métricas já foram anteriormente explicitadas)
    - *all-defs*
    - *all-c-uses*
    - *all-p-uses*
    - *all-du-paths*
- *black-box testing*
  - número de casos de teste passados
  - número de casos de teste falhados
- *test coverage*
  - *line coverage*: número mínimo de linhas cobertas
  - *method coverage*: número mínimo de métodos/funcionalidades cobertas
  - *data coverage*: número mínimo de variáveis cobertas

6) Aplicação do plano de testes

Após a definição do plano de testes, ferramentas e métricas, chegou o momento de o colocar em prática. Para tal, será necessário ter em consideração todos os detalhes previamente mencionados, bem como os testes anteriormente definidos. Por conseguinte, irá prosseguir-se para a aplicação dos métodos de *white-box testing, black-box testing* e *testing coverage.*

6.1) White-box testing

6.1.1) Control Flow Testing

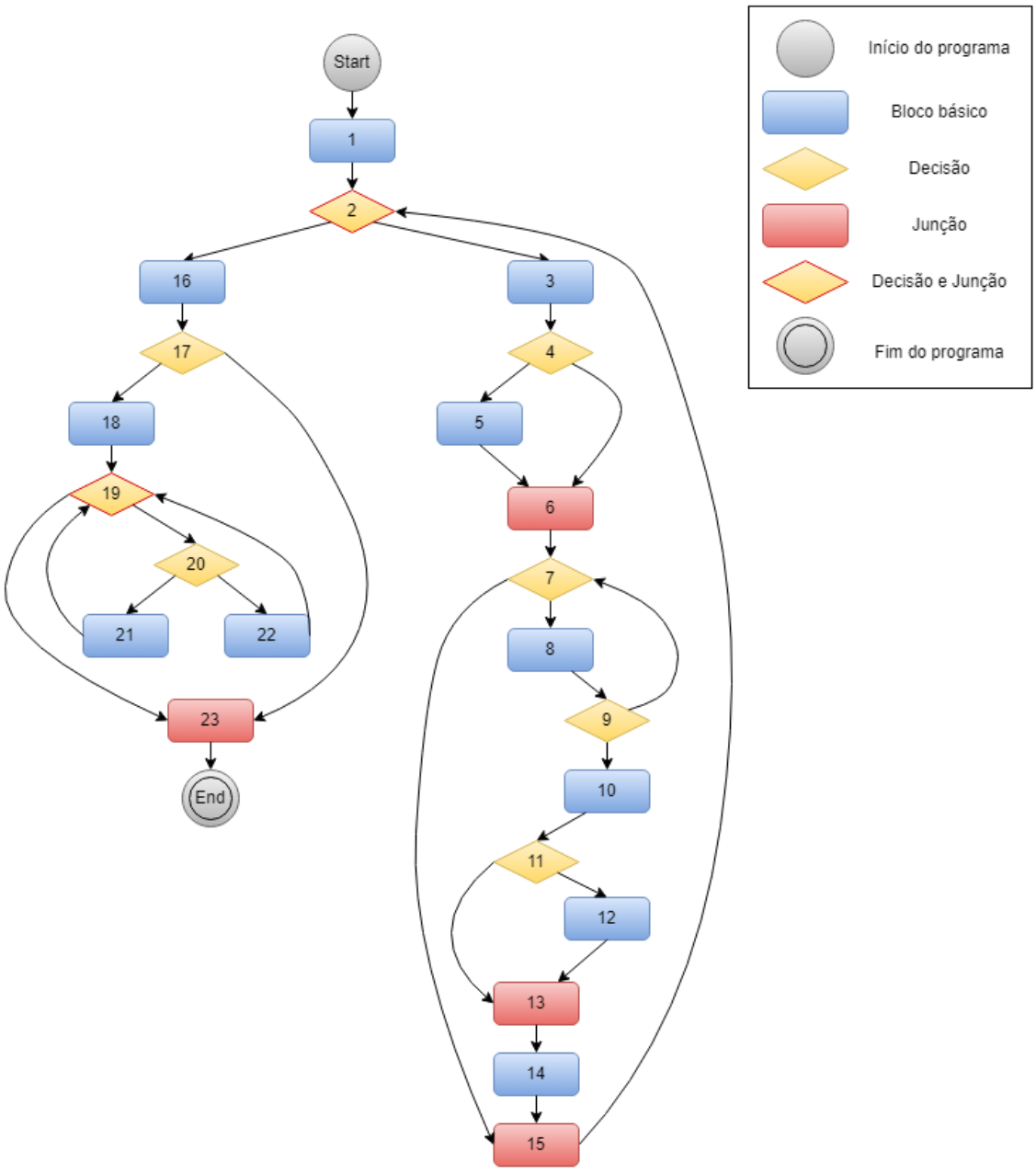
De modo a aplicar esta técnica foi criado o grafo de fluxo de controlo para ambas as funcionalidades a testar. Esse grafo permitirá extrair métricas que ajudarão a determinar o número de caminhos independentes presentes em cada um dos casos, V(G) e consequentemente criar casos de teste.

6.1.1.1) compress(self, str)

```
def compress(self, str):
    with open(str, "rb") as file:                                   1
        text = file.read()


    text = bytearray(text)


    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):                                            2
        i_start_window = i-self.window                              3
        if i_start_window < 0:                                      4
            i_start_window = 0                                      5

        window = text[i_start_window:i]                             6
        buffer = text[i:i+self.buffer]


        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):              7
            posW = window.rfind(buffer[0:size])                     8
            if posW >=0:                                            9
                char = 0                                            10
                pos = len(window) - posW -1

                if i+size < len(text):                              11
                    char = text[i+size]                             12

                toskip=size                                         13
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break                                               14
```

```
        i += toskip+1                                                    15
        BIN += binary

    BIN = bytearray(BIN)                                                 16

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)


    if self.preview==1:                                                  17
        print("\nPREVIEW OF COMPRESSION:")                               18
        for i in range(0, len(BIN), 3):                                  19
            if BIN[i+2]==0:                                              20
                print((BIN[i], BIN[i+1], ''))                            21
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))                 22
        print()                                                          23
```

Primeiramente, dividiu-se o código a testar em pequenos blocos, de modo a facilitar a elaboração do grafo de fluxo de controlo. Este encontra-se apresentado de seguida:



Através deste grafo é possível calcular a complexidade ciclomática, também conhecida como V(G). Esta métrica, como anteriormente referido, irá permitir identificar o número de caminhos independentes existentes no código deste *software*, significando que para testar o mesmo serão necessários pelo menos n caminhos (n = V(G)). Seguidamente, apresenta-se o cálculo de V(G) utilizando duas maneiras distintas:

V(G) = nº de arestas - nº de nós + 2 = 30 - 23 + 2 = 9

V(G) = nº de nós predicativos + 1 = 8 + 1 = 9

Os 9 caminhos linearmente independentes encontram-se representados de seguida:

ID	Caminho (a negrito, os nós novos em relação aos caminhos já encontrados)
CInd1	**1-2-16-17-23**
CInd2	1-2-16-17-**18-19**-23
CInd3	1-2-**3-4-6-7-15**-2-16-17-23
CInd4	1-2-3-4-**5**-6-7-15-2-16-17-23
CInd5	1-2-3-4-5-6-7-**8-9**-7-15-2-16-17-23
CInd6	1-2-3-4-5-6-7-8-9-**10-11-13**-14-15-2-16-17-23
CInd7	1-2-3-4-5-6-7-8-9-10-11-**12**-13-14-15-2-16-17-23
CInd8	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-2-16-17-18-19-**20-21**-19-**23**
CInd9	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-2-16-17-18-19-20-**22**-19-20-21-19-23

É de referir que alguns dos caminhos descritos poderiam ser incluídos dentro de outros, mas, de modo a ter uma visão mais abrangente do conjunto de possibilidades que o software poderia seguir, procedeu-se à separação dos mesmos. Além disso, caminhos mais simples facilitam a tarefa de criação de casos de testes.

- Algumas notas sobre os caminhos encontrados:
- Apesar dos caminhos CInd8 e CInd9 serem praticamente os mesmos, acabam por ser necessários para a cobertura dos testes efetuados ao software em questão. É de reparar que a diferença entre estes está na adição da subsequência de nós 19-20-21 depois da subsequência 19-20-22. Isto é algo inevitável, já que para a 1ª subsequência ser executada, a 2ª terá de ter ocorrido.
- A repetição de nós em alguns dos caminhos é algo que não se consegue evitar, já que por vezes algumas instruções, como loops, provocam tal comportamento.
- CInd(n): identificador do caminho independente encontrado, significando "caminho independente número n".

A partir do grafo anteriormente desenvolvido, é possível identificar certos caminhos que, do ponto de vista da execução do software, não são viáveis, como por exemplo:

ID	Caminho
CInv1	1-2-16-17-18-19-20-21-19-23
CInv2	1-2-16-17-18-19-20-22-19-23
CInv3	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-2-16-17-18-19-20-21-19-23

Algumas observações sobre os caminhos considerados como inviáveis:

- A razão para CInv1 e CInv2 não serem possíveis é a seguinte: o facto de o nó 16 se seguir ao 2 acontece devido ao comprimento do texto presente no ficheiro ser 0, ou seja, o ciclo "for" presente no nó 19 não efetuará qualquer iteração. Assim, as sequências de nós 19-20-21 ou 19-20-22, neste contexto, nunca serão possíveis de ocorrer.
- Outra razão pela qual o caminho CInv2, juntamente com o caminho CInv3, não poderem ocorrer é o facto de ambos implicarem que a sequência 19-20-21 seja executada sem que a sequência 19-20-22 não o seja.
- CInv(n): identificador do caminho inválido encontrado, significando "caminho inválido número n".

O número de caminhos linearmente independentes fornece, portanto, um upper bound relativamente à quantidade de casos de testes a serem criados para testar por completo todo o programa. De seguida, são, assim, demonstrados os casos de teste criados (*inputs* escolhidos) para avaliar cada um dos caminhos anteriormente referidos:

ID	Caminhos	Inputs (valor dado a cada parâmetro)
T1	CInd1	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off tipo de ficheiro = ficheiro de texto sem qualquer conteúdo
T2	CInd2	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = on tipo de ficheiro = ficheiro de texto sem qualquer conteúdo
T3	CInd3 e CInd4	tamanho da janela = (não interessa) tamanho do buffer = 0 flag a indicar operação = -compress flag a on/off = (não interessa) tipo de ficheiro = ficheiro de texto com algum conteúdo
T4	CInd5	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = (não interessa) tipo de ficheiro = ficheiro de texto em que cada caracter esteja escrito uma e uma só vez
T5	CInd6, CInd7	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off tipo de ficheiro = ficheiro de texto com algum conteúdo
T6	CInd8 e CInd9	tamanho da janela = (não interessa)

		tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = on tipo de ficheiro = ficheiro de texto com algum conteúdo

Nota: quando é indicado que um valor para um determinado parâmetro "não interessa", assume-se que o mesmo deve encontrar-se dentro dos limites da variável.

ID	Output esperado	Output obtido
T1	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) perto de 0 KB	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho perto de 0 KB
T2	**Terminal:** "PREVIEW OF COMPRESSION:" **Ficheiros:** 2 ficheiros de tamanho (conjunto) perto de 0 KB	**Terminal:** "PREVIEW OF COMPRESSION:" **Ficheiros:** 2 ficheiros de tamanho perto de 0 KB
T3	**Terminal:** se a flag on/off = off (nada) se a flag on/off = on "PREVIEW OF COMPRESSION: (0,0,1º caracter), (0,0,2º caracter), …", este padrão mantém-se para os restantes caracteres **Ficheiros:** 2 ficheiros de tamanho (conjunto) triplo do tamanho original	**Terminal:** se a flag on/off = off (nada) se a flag on/off = on "PREVIEW OF COMPRESSION: (0,0,1º caracter), (0,0,2º caracter), …", este padrão mantém-se para os restantes caracteres **Ficheiros:** 2 ficheiros de tamanho (conjunto) de menos 8%
T4	**Terminal:** se a flag on/off = off (nada) se a flag on/off = on "PREVIEW OF COMPRESSION: (0,0,1º caracter), (0,0,2º caracter), …", este padrão mantém-se para os restantes caracteres **Ficheiros:** 2 ficheiros de tamanho (conjunto) triplo do tamanho original	**Terminal:** se a flag on/off = off (nada) se a flag on/off = on "PREVIEW OF COMPRESSION: (0,0,1º caracter), (0,0,2º caracter), …", este padrão mantém-se para os restantescaracteres **Ficheiros:** 2 ficheiros de tamanho (conjunto) de menos 8%
T5	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável
T6	**Terminal:** "PREVIEW OF COMPRESSION: <preview>" **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável	**Terminal:** "PREVIEW OF COMPRESSION: <preview>" **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável

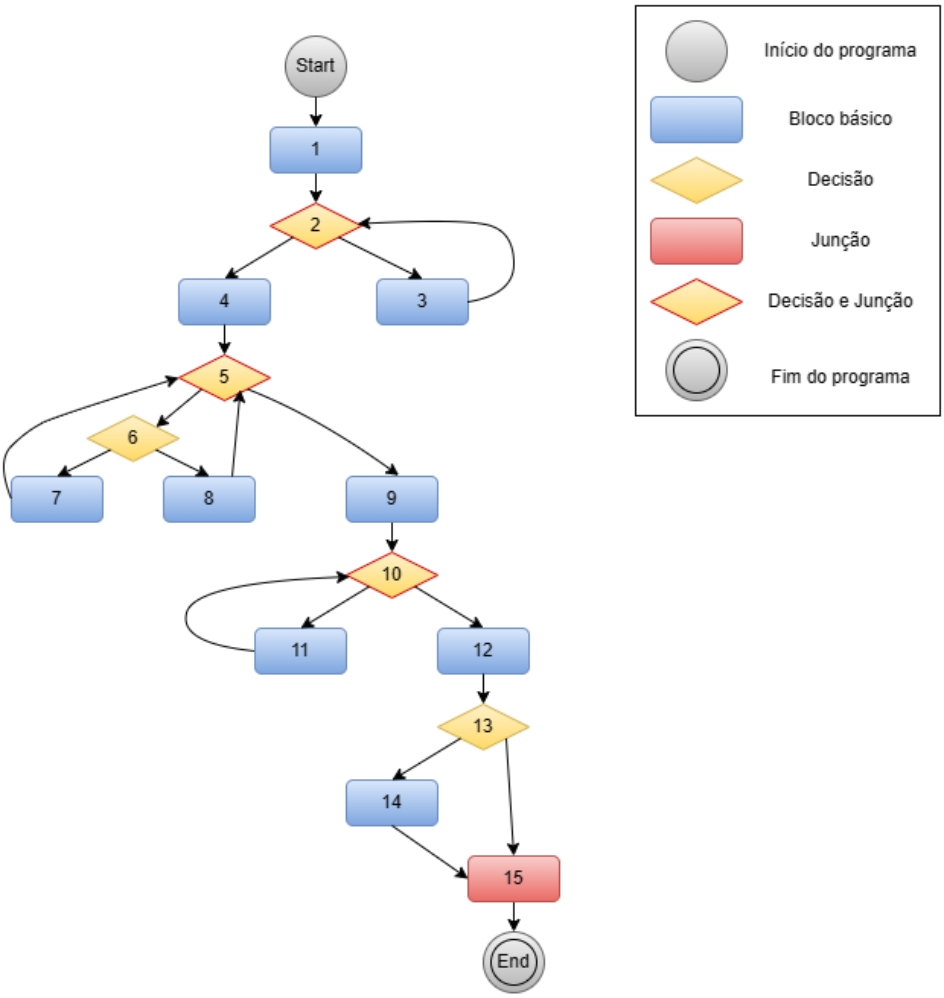
Como se pode observar, com exceção dos testes T3 e T4, os testes efetuados foram bem sucedidos e produziram o resultado esperado. A razão pela qual os testes T3 e T4 não estão completamente corretos é a seguinte: ao examinar a preview, ambos os casos satisfazem perfeitamente o output esperado, contudo, acontece que o programa armazena os dados utilizando os métodos bytes() e bytearray() do Python. Ora, isto leva a que os mesmos sejam guardados nas estruturas do programa com um tamanho menor e, consequentemente, à diminuição do tamanho do ficheiro original, ainda que teoricamente fosse de esperar que o mesmo triplicasse.

6.1.1.2) decompress(self, name)

```
def decompress(self, name):
        with open('huffman_aux.bin', 'rb') as freq:                    1
            codec = pickle.load(freq)


        with open(name, 'rb') as encodedFile:
            dic = encodedFile.read()


        text = bytearray('', 'utf-8')
        for char in codec.decode(dic):                                 2
            text += bytes([char])                                      3

        aux = []                                                       4
        for i in range(0,len(text),3):                                 5
            if text[i+2] == 0:                                         6
                aux.append((text[i], text[i+1], ''))                   7
            else:
                aux.append((text[i], text[i+1], chr(text[i+2])))       8

        text = aux                                                     9


        text2=""
        for elem in text:                                              10
            start = len(text2)-(elem[0]+1)                             11
            text2 += text2[start:start+elem[1]]+elem[2]

        splitted = name.split("_")                                     12
        toremove = splitted[0]
        towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension


        with open(towrite, "w") as decodedFile:
            decodedFile.write(text2)
```

```
if self.preview==1:  13
    print("\nPREVIEW OF DECOMPRESSION:")  14
    print(text)
print()  15
```

Novamente, dividiu-se o código a testar em pequenos blocos, de modo a facilitar a elaboração do grafo de fluxo de controlo. Este encontra-se apresentado de seguida:



De seguida, é calculado o V(G) utilizando duas maneiras distintas:
V(G) = nº de arestas - nº de nós + 2 = 19 - 15 + 2 = 6
V(G) = nº de nós predicativos + 1 = 5 + 1 = 6

Os 6 caminhos linearmente independentes encontram-se representados na seguinte tabela:

ID	Caminho (a negrito, os nós novos em relação aos caminhos já encontrados)
CInd1	**1-2-4-5-9-10-12-13-15**
CInd2	1-2-**3**-2-4-5-9-10-12-13-15
CInd3	1-2-3-2-4-5-**6**-**8**-5-9-10-12-13-15
CInd4	1-2-3-2-4-5-6-8-5-6-**7**-5-9-10-12-13-15
CInd5	1-2-3-2-4-5-9-10-**11**-10-12-13-15
CInd6	1-2-3-2-4-5-6-8-5-9-10-11-10-12-13-**14**-15

Tal como mencionado na função compress(self, str), alguns dos caminhos descritos poderiam ser incluídos dentro de outros. Assim, de modo a simplificar a tarefa de criação de casos de testes, estes caminhos encontram-se separados. Isto, permitirá também uma visão mais abrangente do conjunto de possibilidades que o software poderia seguir.

A partir do grafo desenvolvido, volta a ser possível encontrar caminhos que, do ponto de vista da execução do software, não são viáveis, como por exemplo:

ID	Caminho
CInv1	1-2-3-2-4-5-6-7-5-9-10-12-13-15
CInv2	1-2-3-2-4-5-9-10-12-13-15

Algumas observações sobre os caminhos considerados como inviáveis:

- Ambos os caminhos encontrados acabam por ser definidos como inválidos devido ao contexto de execução desta funcionalidade. Neste programa existem 3 ciclos “for”, onde ou todos são executados ou nenhum é executado, correspondentes às sequências 2-3-2, 5-6–8 e 10-11-10. Visto que os caminhos acima não apresentam simultaneamente as sequências de nós mencionadas, os mesmos são considerados inválidos.
- O caminho CInv1 também não será considerado válido devido à sequência 5-6-7 ser apresentada sem que antes tenha sido executada a sequência 5-6-8. Este volta a ser um problema a nível da execução do programa.

Como referido, a complexidade ciclomática (também conhecida como número de caminhos independentes) fornece um upper bound relativamente ao número de casos de teste a criar para que o software seja testado por completo. Ao contrário do que foi observado para a função de compressão, os caminhos linearmente independentes não poderão ser completamente testados, já que alguns destes são inválidos. Por conseguinte, irão ser testados os seguintes caminhos:

- C1 = 1-2-4-5-9-10-12-13-15
- C2 = 1-2-3-4-5-6-8-5-9-10-11-10-12-13-15
- C3 = 1-2-3-4-5-6-8-5-9-10-11-10-12-13-14-15
- C4 = 1-2-3-4-5-6-8-5-6-7-5-9-10-11-10-12-13-15

- C5 = 1-2-3-4-5-6-8-5-6-7-5-9-10-11-10-12-13-14-15
- C6 = 1-2-4-5-9-10-12-13-14-15

De seguida, são apresentados os casos de teste criados para avaliar cada um dos caminhos mencionados anteriormente:

ID	Caminhos	Inputs (valor dado a cada parâmetro)
T1	C1	flag a indicar operação = -decompress flag a on/off = off ficheiro principal sem qualquer conteúdo e ficheiro huffman com conteúdo original extensão = correspondente ao ficheiro original
T2	C1	flag a indicar operação = -decompress flag a on/off = off tipo de ficheiro = ficheiro principal com conteúdo original ficheiro huffman sem qualquer conteúdo extensão = correspondente ao ficheiro original
T3	C6	flag a indicar operação = -decompress flag a on/off = on tipo de ficheiro = ficheiro principal sem qualquer conteúdo ficheiro huffman sem qualquer conteúdo extensão = correspondente ao ficheiro original
T4	C6	flag a indicar operação = -decompress flag a on/off = on tipo de ficheiro = ficheiro principal sem qualquer conteúdo ficheiro huffman resultante da compressão de um ficheiro vazio extensão = correspondente ao ficheiro original
T5	C2 e C4	flag a indicar operação = -decompress flag a on/off = off tipo de ficheiro = ficheiro principal com algum conteúdo extensão = correspondente ao ficheiro original
T6	C3 e C5	flag a indicar operação = -decompress flag a on/off = on tipo de ficheiro = ficheiro principal com algum conteúdo extensão = correspondente ao ficheiro original

Nota: quando é indicado que um valor para um determinado parâmetro "não interessa", assume-se que o mesmo deve encontrar-se dentro dos limites da variável.

ID	Output esperado	Output obtido
T1	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho de 0 KB	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho de 0 KB

T2	**Terminal:** Mensagem de erro (emitida pelo Python e não pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo Python e não pelo programa) **Ficheiros:** Nenhum
T3	**Terminal:** "PREVIEW OF DECOMPRESSION:" **Ficheiros:** 1 ficheiro com tamanho de 0 KB	**Terminal:** Mensagem de erro (emitida pelo Python) **Ficheiros:** Nenhum
T4	**Terminal:** "PREVIEW OF DECOMPRESSION:" **Ficheiros:** 1 ficheiro com tamanho de 0 KB	**Terminal:** "PREVIEW OF DECOMPRESSION:" **Ficheiros:** 1 ficheiro com tamanho de 0 KB
T5	**Terminal:** "PREVIEW OF DECOMPRESSION: <preview>" **Ficheiros:** 1 ficheiros com tamanho original antes de ser comprimido	**Terminal:** "PREVIEW OF DECOMPRESSION: <preview>" **Ficheiros:** 1 ficheiros com tamanho ligeiramente superior ao original
T6	**Terminal:** (nada) **Ficheiros:** 1 ficheiros com tamanho original antes de ser comprimido	**Terminal:** (nada) **Ficheiros:** 1 ficheiros com tamanho ligeiramente superior ao original

Ao testar a função de descompressão foram detectados alguns erros e incoerências para os testes T3, T5 e T6, enquanto que os restantes apresentaram o output correto.

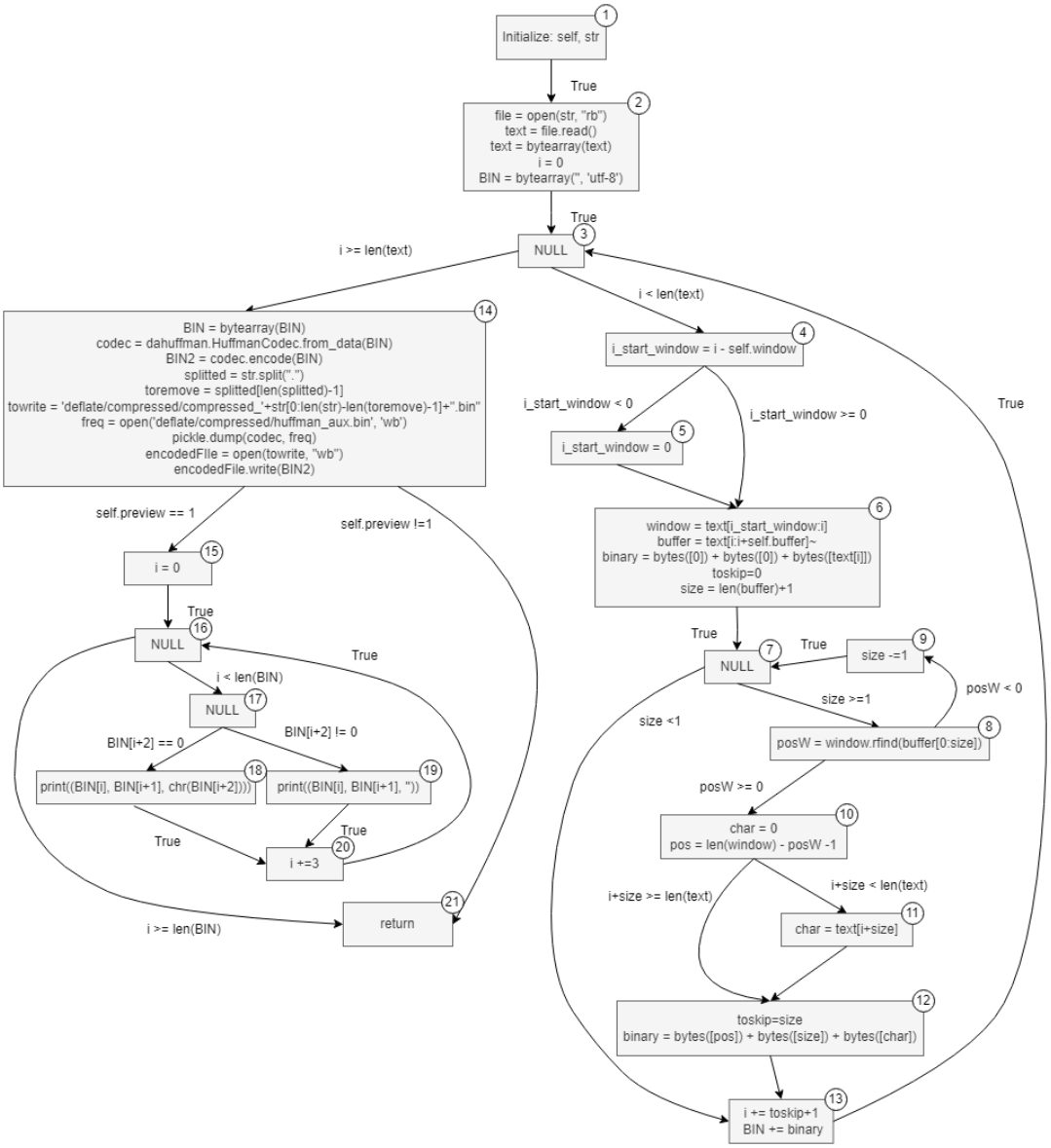
Observando primeiramente o teste T3, seria de esperar que o ficheiro principal comprimido fosse transformado num ficheiro descomprimido vazio. No entanto, como o ficheiro gerado pelo algoritmo de Huffman Enconding está vazio, ocorre uma exceção e, consequentemente, uma falha no programa.

Um comportamento curioso encontrado nos testes T5 e T6 é o facto do algoritmo de descompressão, apesar de mostrar um output correto, originar um ficheiro com tamanho superior ao original.

6.1.2) Data Flow Testing

Durante a aplicação desta abordagem, foram realizadas várias etapas, incluindo a criação de um grafo de fluxo de dados para cada uma das funcionalidades a serem testadas. Este grafo permitirá compreender melhor o caminho que cada variável segue durante a execução deste software, isto é, quando é inicializada, utilizada, redefinida, etc. Com base nesta informação e em outras métricas relevantes, pretende-se elaborar alguns casos de teste.

6.1.2.1) compress(self, str)



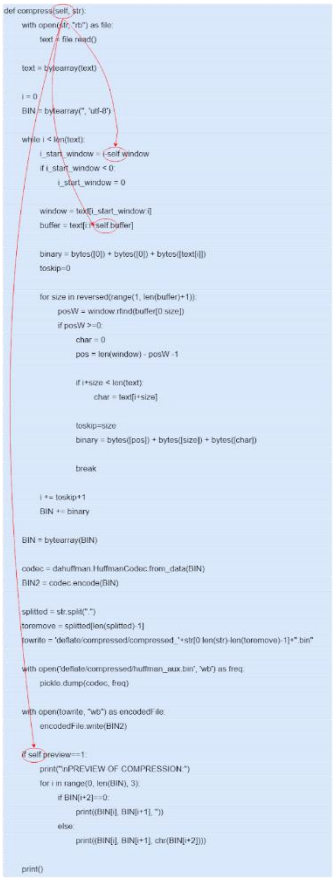
A partir deste grafo, torna-se possível compreender mais facilmente o fluxo dos dados durante a execução desta funcionalidade e identificar possíveis problemas relacionados com a definição e o uso de variáveis. É ainda possível extrair algumas “métricas” mencionadas anteriormente, como: all-defs, all-c-uses, all-p-uses, all-du-paths.

Segue-se a análise a all-defs, all-c-uses e all-p-uses:

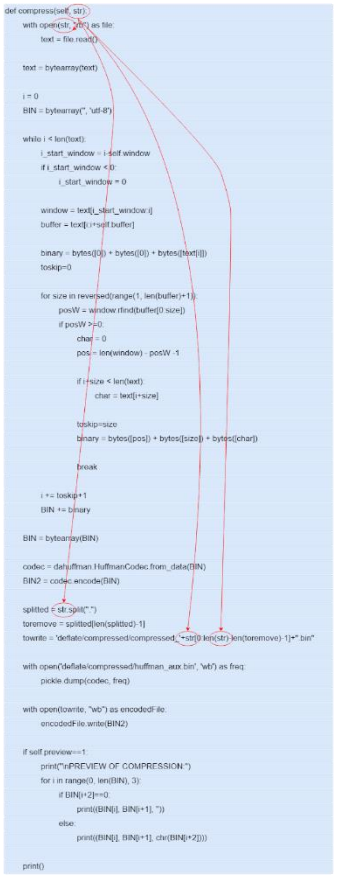
	Definitions	C-Uses	P-Uses
def compress(self, str):	self, str		
with open(str, "rb") as file:	file	str	
text = file.read()	text	file	
text = bytearray(text)	text	text	
i = 0	i		
BIN = bytearray('', 'utf-8')	BIN		
while i < len(text):			i, text
i_start_window = i-self.window	i_start_window	i, self	
if i_start_window < 0:			i_start_window
i_start_window = 0	i_start_window		
window = text[i_start_window:i]	window	text, i_start_window, i	
buffer = text[i:i+self.buffer]	buffer	text, i, self	
binary = bytes([0]) + bytes([0]) + bytes([text[i]])	binary	text, i	
toskip=0	toskip		
for size in reversed(range(1, len(buffer)+1)):	size	size	size, buffer
posW = window.rfind(buffer[0:size])	posW	window, buffer, size	
if posW >=0:			posW
char = 0	char		
pos = len(window) - posW -1	pos	window, posW	
if i+size < len(text):			i, size, text
char = text[i+size]	char	text, i, size	
toskip=size	toskip	size	
binary = bytes([pos]) + bytes([size]) + bytes([char])	binary	pos, size, char	
break			
i += toskip+1	i	toskip, i	
BIN += binary	BIN	BIN, binary	
BIN = bytearray(BIN)	BIN	BIN	
codec = dahuffman.HuffmanCodec.from_data(BIN)	codec	BIN	
BIN2 = codec.encode(BIN)	BIN2	codec, BIN	
splitted = str.split(".")	splitted	str	
toremove = splitted[len(splitted)-1]	toremove	splitted	
towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"	towrite	str, toremove	
with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:	freq		
pickle.dump(codec, freq)		codec, freq	
with open(towrite, "wb") as encodedFile:	encodedFile	towrite	
encodedFile.write(BIN2)		encodedFile, BIN2	
if self.preview==1:			self
print("\nPREVIEW OF COMPRESSION:")			
for i in range(0, len(BIN), 3):	i	i	i, BIN
if BIN[i+2]==0:			BIN, i
print((BIN[i], BIN[i+1], ''))		BIN,i	
else:			
print((BIN[i], BIN[i+1], chr(BIN[i+2])))		BIN, i	
print()			

Após analisar todas as métricas, pode-se garantir que este programa não apresentará falhas devido à falta de inicialização de variáveis. Isto ocorre porque, antes de um C-Use ou P-Use de uma variável, há sempre uma definição prévia da mesma. As variáveis identificadas neste programa foram as seguintes: “self”, “str”, “file”, “text”, “i”, “BIN”, “i_start_window”, “window”, “buffer”, “binary”, “toskip”, “size”, “posW”, “char”, “pos”, “codec”, “BIN2”, “splitted”, “toremove”, “towrite”, “freq”, “encodedFile”.

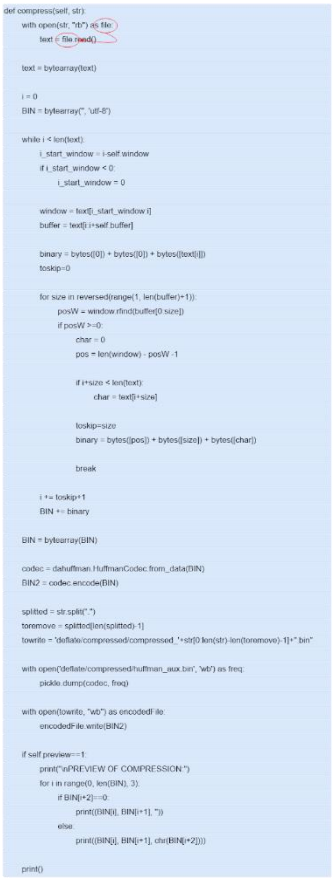
De modo a compreender o caminho que cada variável toma durante a execução foram avaliados todos os caminhos DU. Estes foram criados pela ordem com que as variáveis se encontram anunciadas em cima.



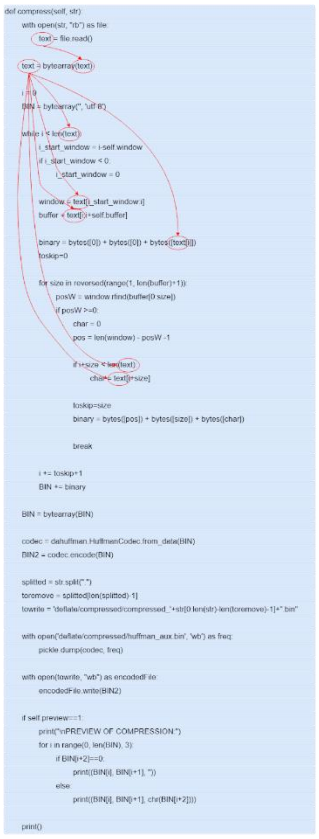
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



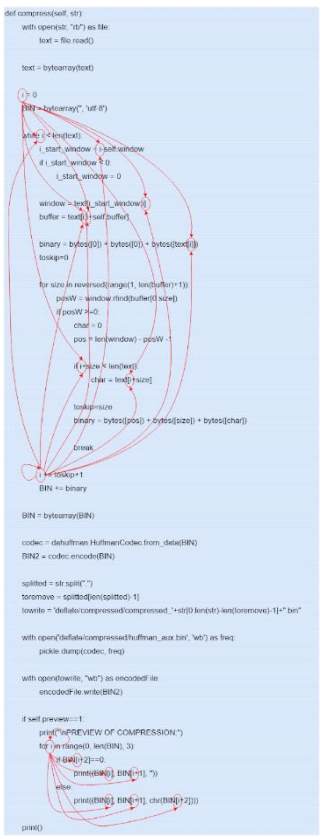
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



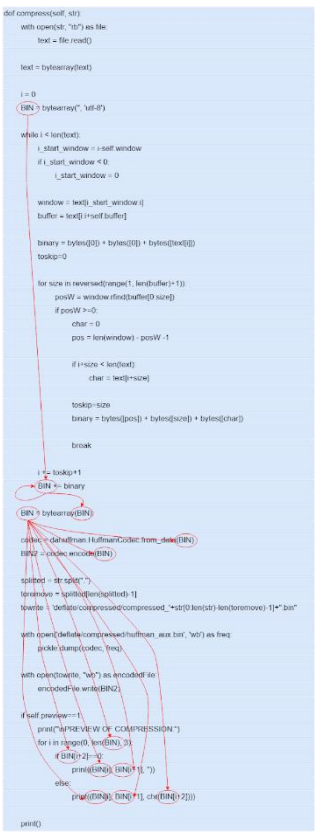
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

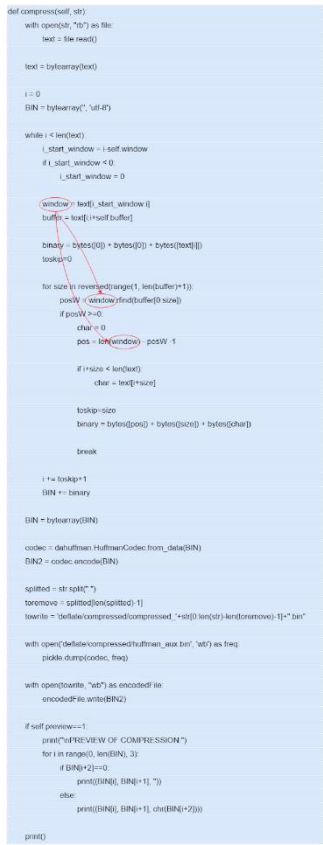


```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



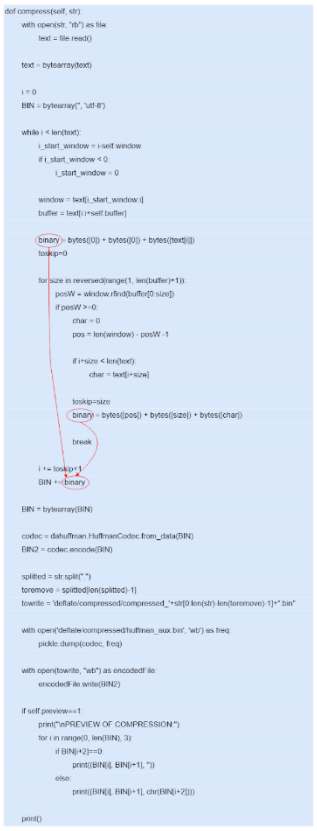
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



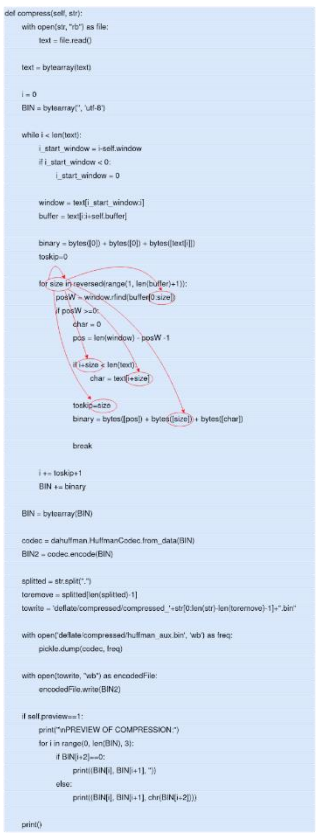
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

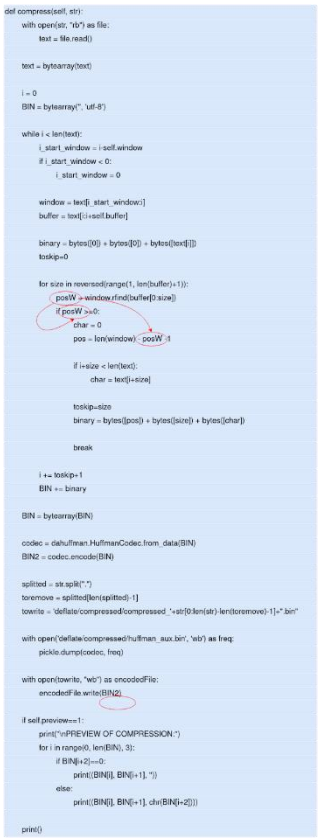


```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

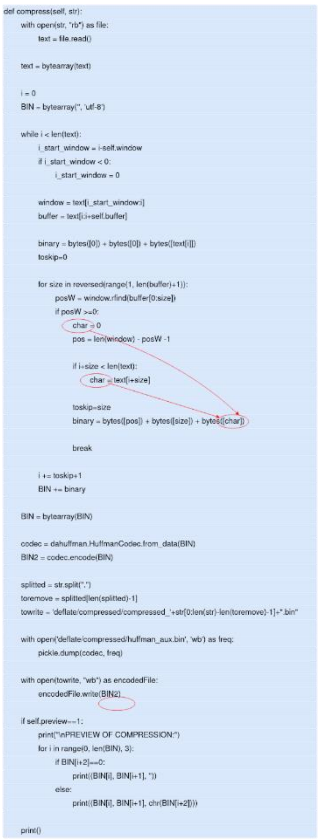
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



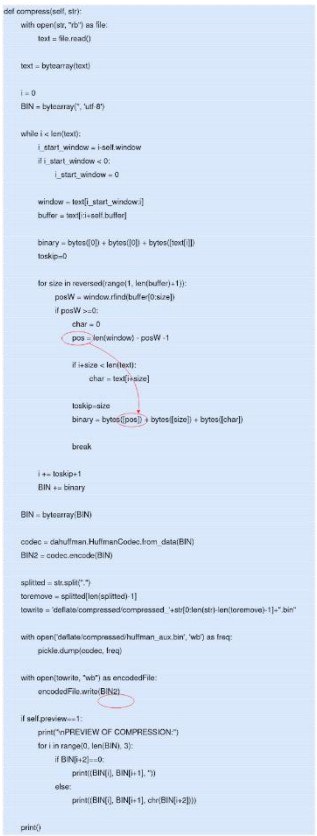
```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```


```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```



```python
def compress(self, str):
    with open(str, "rb") as file:
        text = file.read()

    text = bytearray(text)

    i = 0
    BIN = bytearray('', 'utf-8')

    while i < len(text):
        i_start_window = i-self.window
        if i_start_window < 0:
            i_start_window = 0

        window = text[i_start_window:i]
        buffer = text[i:i+self.buffer]

        binary = bytes([0]) + bytes([0]) + bytes([text[i]])
        toskip=0

        for size in reversed(range(1, len(buffer)+1)):
            posW = window.rfind(buffer[0:size])
            if posW >=0:
                char = 0
                pos = len(window) - posW -1

                if i+size < len(text):
                    char = text[i+size]

                toskip=size
                binary = bytes([pos]) + bytes([size]) + bytes([char])

                break

        i += toskip+1
        BIN += binary

    BIN = bytearray(BIN)

    codec = dahuffman.HuffmanCodec.from_data(BIN)
    BIN2 = codec.encode(BIN)

    splitted = str.split(".")
    toremove = splitted[len(splitted)-1]
    towrite = 'deflate/compressed/compressed_'+str[0:len(str)-len(toremove)-1]+".bin"

    with open('deflate/compressed/huffman_aux.bin', 'wb') as freq:
        pickle.dump(codec, freq)

    with open(towrite, "wb") as encodedFile:
        encodedFile.write(BIN2)

    if self.preview==1:
        print("\nPREVIEW OF COMPRESSION:")
        for i in range(0, len(BIN), 3):
            if BIN[i+2]==0:
                print((BIN[i], BIN[i+1], ''))
            else:
                print((BIN[i], BIN[i+1], chr(BIN[i+2])))

    print()
```

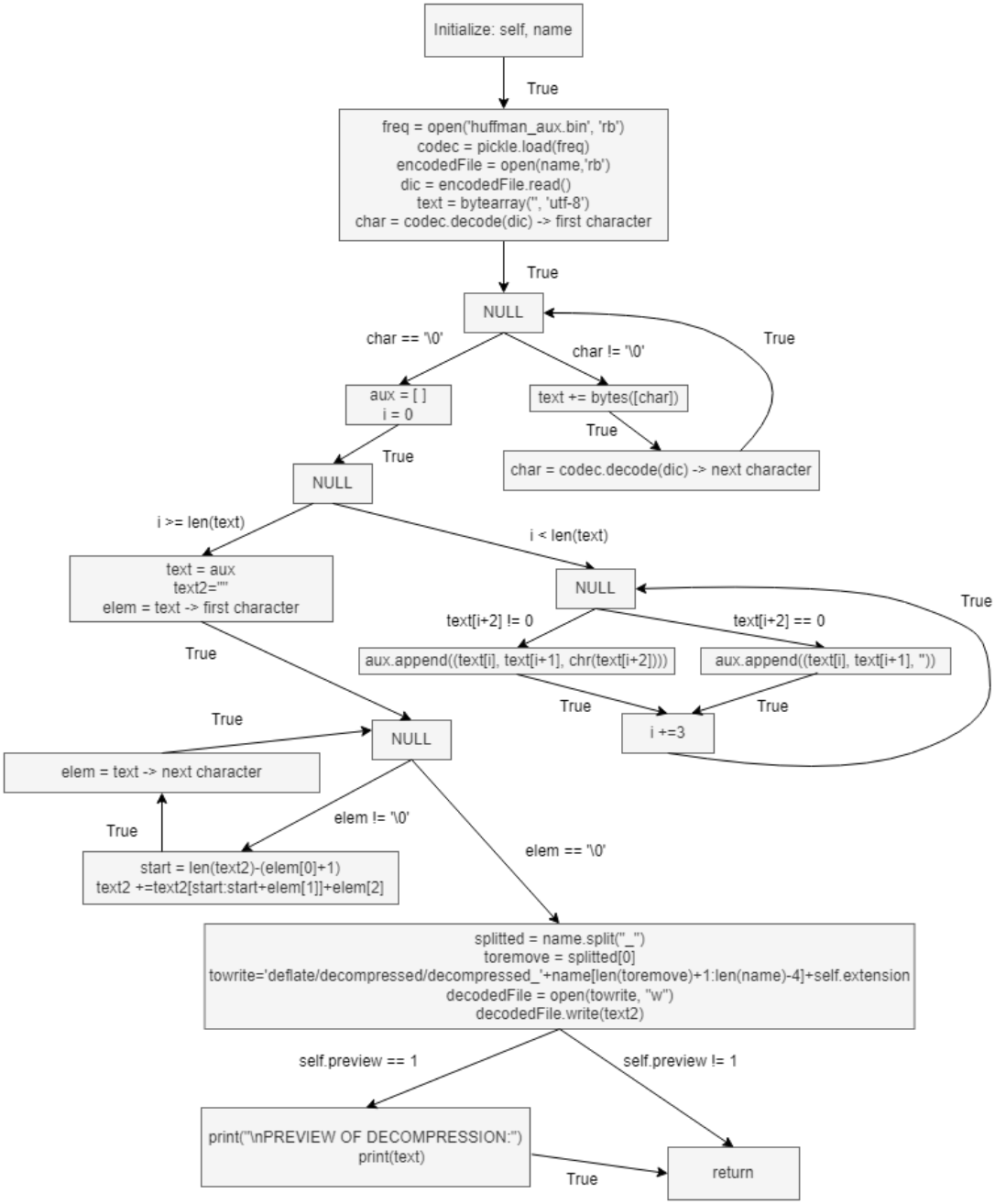
Após terem sido identificados todos os caminhos DU, correspondentes às diferentes variáveis ao longo da execução da funcionalidade de compressão, procedeu-se à sua contabilização na tabela seguinte:

Variável	Número de caminhos	Variável	Número de caminhos
self	3	size	6
str	4	posW	2
file	1	char	2
text	7	pos	1
i	24	codec	2
BIN	12	BIN2	1
i_start_window	3	splitted	2
window	2	toremove	1
buffer	2	towrite	1
binary	2	freq	1
toskip	2	encodedFile	1

Como se pode observar, a variável “i” é a que apresenta um maior número de caminhos, o que é expectável, visto ser utilizada em ciclos (“while” e “for”). A variável "BIN" ocupa a segunda posição em termos de número de caminhos, o que é esperado, uma vez que é responsável por armazenar a maior parte do resultado da compressão realizada pelo algoritmo Deflate. As demais variáveis são encontradas com menor frequência, pois a maioria delas é utilizada em operações de menor escala.

No entanto, não serão executados quaisquer casos de teste para a fase em questão. Os testes efetuados, durante o “Control flow testing”, já apresentam evidências suficientes para perceber que as variáveis desta funcionalidade apresentam um papel importante para a sua execução.

6.1.2.2) decompress(self, name)

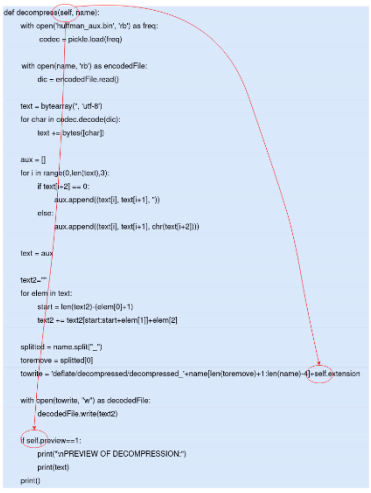


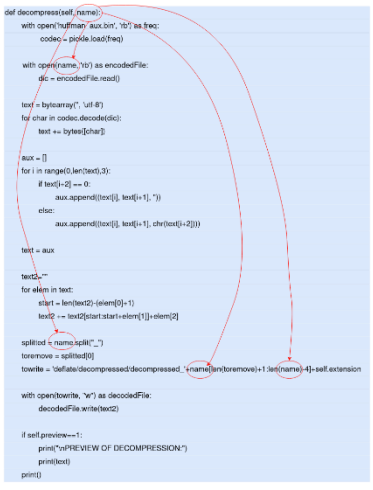
Novamente, a partir do grafo acima, torna-se possível compreender melhor qual é o fluxo dos dados durante a execução da funcionalidade aqui representada(descompressão) e, provavelmente, identificar possíveis problemas relacionados com a definição e uso de variáveis. É ainda possível extrair as métricas mencionadas anteriormente, como: all-defs, all-c-uses, all-p-uses, all-du-paths. Segue-se a análise a all-defs, all-c-uses e all-p-uses:

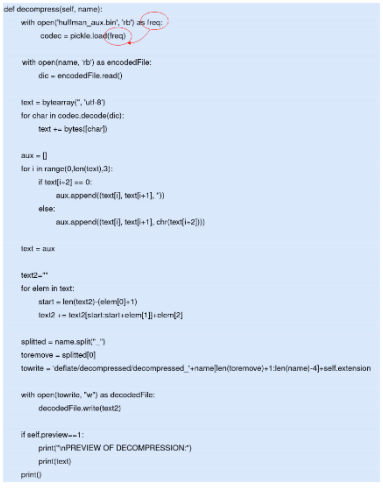
	Definitions	C-Uses	P-Uses
def decompress(self, name):	self, name		
with open('huffman_aux.bin', 'rb') as freq:	freq		
codec = pickle.load(freq)	codec	freq	
with open(name, 'rb') as encodedFile:	encodedFile	name	
dic = encodedFile.read()	dic	encodedFile	
text = bytearray('', 'utf-8')	text		
for char in codec.decode(dic):	char	char, dic	char, dic
text += bytes([char])	text	text, char	
aux = []	aux		
for i in range(0,len(text),3):	i	i	i, text
if text[i+2] == 0:			text
aux.append((text[i], text[i+1], ''))	aux	aux,text,i	
else:			
aux.append((text[i], text[i+1], chr(text[i+2])))	aux	aux,text,i	
text = aux	text	aux	
text2=""	text2		
for elem in text:	elem	elem,text	elem,text
start = len(text2)-(elem[0]+1)	start	text2,elem	
text2 += text2[start:start+elem[1]]+elem[2]	text2	text2,elem,start	
splitted = name.split("_")	splitted	name	
toremove = splitted[0]	toremove	splitted	
towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension	towrite	name,toremove, self	
with open(towrite, "w") as decodedFile:	decodedFile	towrite	
decodedFile.write(text2)		decodedFile, text2	
if self.preview==1:			self
print("\nPREVIEW OF DECOMPRESSION:")			
print(text)		text	
print()			

Ao analisar as métricas mencionadas, garante-se que este programa também não apresenta qualquer risco de falha devido à falta de inicialização de variáveis, já que antes de qualquer C-Use ou P-Use de uma variável, existe uma definição da mesma. As variáveis identificadas neste programa foram as seguintes: “self”, “name”, “freq”, “codec”, “encodedFile”, “dic”, “text”, “char”, “aux”, “i”, “text2”, “elem”, “start”, “splitted”, “toremove”, “towrite”, “decodedFile”.

Com vista a entender o caminho de cada variável ao longo da execução, irá analisar-se todos os caminhos DU que este programa apresenta. Estes foram determinados seguindo a ordem com que as respectivas variáveis foram mencionadas anteriormente:







```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

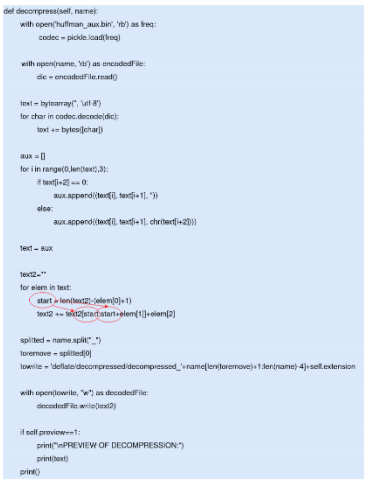
```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

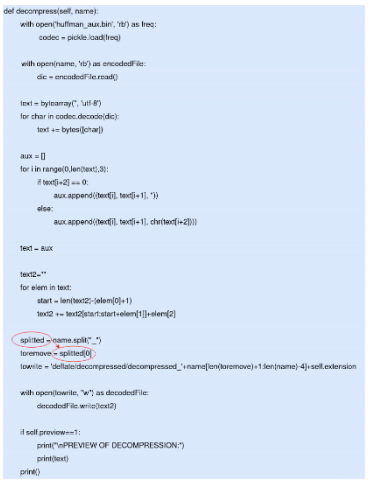
```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

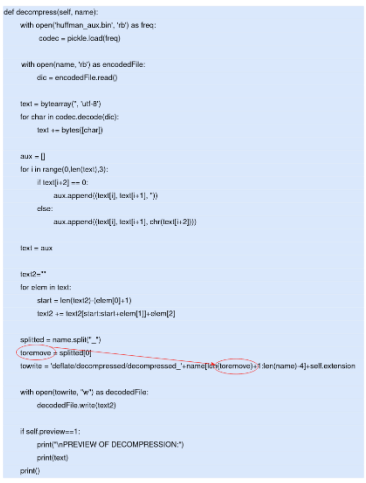
```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

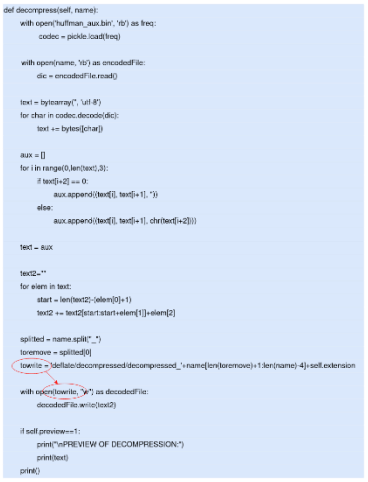
```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```

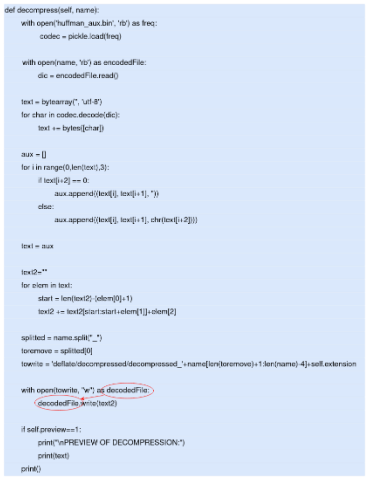
```
def decompress(self, name):
    with open('huffman_aux.bin', 'rb') as freq:
        codec = pickle.load(freq)

    with open(name, 'rb') as encodedFile:
        dic = encodedFile.read()

    text = bytearray('', 'utf-8')
    for char in codec.decode(dic):
        text += bytes([char])

    aux = []
    for i in range(0,len(text),3):
        if text[i+2] == 0:
            aux.append((text[i], text[i+1], ''))
        else:
            aux.append((text[i], text[i+1], chr(text[i+2])))

    text = aux

    text2=""
    for elem in text:
        start = len(text2)-(elem[0]+1)
        text2 += text2[start:start+elem[1]]+elem[2]

    splitted = name.split("_")
    toremove = splitted[0]
    towrite = 'deflate/decompressed/decompressed_'+name[len(toremove)+1:len(name)-4]+self.extension

    with open(towrite, "w") as decodedFile:
        decodedFile.write(text2)

    if self.preview==1:
        print("\nPREVIEW OF DECOMPRESSION:")
        print(text)
    print()
```











Identificados todos os caminhos DU, correspondentes às diferentes variáveis presentes ao longo da execução da funcionalidade de compressão, procedeu-se ao registo dos resultados obtidos na tabela seguinte:

Variável	Número de caminhos	Variável	Número de caminhos
self	2	i	7
name	3	text2	7
freq	1	elem	3
codec	1	start	2
encodedFile	1	splitted	1
dic	1	toremove	1
text	11	towrite	1

char	1	decodedFile	1
aux	9		

Comparativamente com a funcionalidade anterior, é possível observar que o número de caminhos DU é muito menor. Isto seria algo expectável e facilmente perceptível, já que esta função é consideravelmente mais simples do que a anterior. Não é surpreendente que as variáveis "text" e “aux” apresentem o maior número de caminhos, uma vez que elas são amplamente modificadas ao longo do programa para a descompressão de um ficheiro. A terceira variável com maior número de caminhos é a variável "i", o que também é compreensível, pois ela é uma variável de indexação num loop.

No entanto, não serão executados quaisquer casos de teste para a fase em questão. Os testes realizados durante o "Control flow testing" já fornecem evidências suficientes para entender o papel importante das variáveis nesta funcionalidade e a sua influência na execução do programa. É apenas de apontar um possível erro na variável “freq” quando esta lê um ficheiro huffman com conteúdo vazio, devido ao uso do método pickle.load().

6.2) Black-box testing

6.2.1) Equivalence Classes Partitioning

Para esta técnica procedeu-se à criação de diversos conjuntos de classes que representassem inputs válidos ou inválidos, consoante os valores que um determinado parâmetro pudesse assumir. Estes conjuntos foram organizados em tabelas com vista a facilitar a visualização dos mesmos e, posteriormente, elaborados casos de teste.

6.2.1.1) compress(self, str)

De modo a simplificar a escrita das classes de equivalência, os parâmetros do *software* foram simplificados da seguinte forma:

- tamanho da janela: TJ, no código constitui a variável “self.window” presente na classe *deflate*
- tamanho do *buffer*: TB, no código constitui a variável “self.buffer” presente na classe *deflate*
- flag a indicar operação: FIO, no código constituiu a variável “sys.argv[3]” presente no *main*
- flag a on/off: FOO, no código constituiu a variável “self.preview” presente na classe *deflate*
- nome do ficheiro: NF, no código constituiu a variável “str” que se encontra na função compress contida na classe *deflate*
- conteúdo do ficheiro: CF

Input	Equivalence classes válidas	Equivalence classes inválidas
Um número inteiro TJ, tal que 0 <= TJ <= 2**63-1	[0, 1000] [1001, 10000] [10001, 2**62-1] 2**62 [2**62+1, 2*63-10001]	<0 >2**63-1 Números malformados Strings “Lixo”

	[2**63-10000, 2**63-1001] [2**63-1000, 2**63-1]	Valores vazios
Um número inteiro TB, tal que 0 <= TJ <= 256	[0,100] [101, 127] 128 [129, 155] [156,256]	<0 >256 Números malformados Strings “Lixo” Valores vazios
Uma string FIO, tal que FIO = “-compress”	-compress	Números Números malformados Lixo Valores vazios Outra string que não “-compress”
Uma string FOO, tal que FOO = “on” ou FOO = “off”	on off	Números Números malformados Lixo Valores vazios Outra string que não “on” ou “off”
Uma string NF, tal que NF apresenta o nome com uma extensão de formato de texto	ficheiros com extensão .txt ficheiros com extensão .csv ficheiros com extensão .js ficheiros com extensão .py ficheiros com extensão .xml ficheiros com extensão .json ficheiros com extensão .log	ficheiros com extensão .png ficheiros com extensão .pdf ficheiros com extensão .docx ficheiros com extensão .jpeg ficheiros com extensão .zip ficheiros com extensão .mp4
Um conjunto de strings CF, tal que apresente caracteres ASCII	conjunto de letras conjunto de número conjunto de letras e número conjunto de letras, números e caracteres especiais	quaisquer caracteres que não sejam ASCII

De seguida, são apresentados os testes realizados para avaliar as diferentes equivalence classes válidas e inválidas. Primeiramente, é indicado o tipo de teste realizado e, posteriormente, são fornecidos os resultados obtidos para cada caso de teste.

ID	Inputs (valor dado a cada parâmetro)
T1	tamanho da janela = [0, 1000] tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T2	tamanho da janela = [1001, 10000] tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = ficheiro formato de texto conteúdo do ficheiro = (não interessa)
T3	tamanho da janela = [2**63-10000, 2**63-1] tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T4	tamanho da janela = [10001, 2*63-10001] tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T5	tamanho da janela = < 0 tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T6	tamanho da janela = > 2**63-1 tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T7	tamanho da janela = Número malformados tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)

T8	tamanho da janela = Strings tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T9	tamanho da janela = Floats tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T10	tamanho da janela = (não interessa) tamanho do buffer = [0, 100] flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T11	tamanho da janela = (não interessa) tamanho do buffer = [101, 155] flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T12	tamanho da janela = (não interessa) tamanho do buffer = [156, 256] flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T13	tamanho da janela = (não interessa) tamanho do buffer = < 0 flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T14	tamanho da janela = (não interessa) tamanho do buffer = > 256 flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T15	tamanho da janela = (não interessa) tamanho do buffer = Número malformados

	flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T16	tamanho da janela = (não interessa) tamanho do buffer = Strings flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T17	tamanho da janela = (não interessa) tamanho do buffer = Floats flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T18	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = Números flag a on/off = off nome do ficheiro = ficheiro formato de texto conteúdo do ficheiro = (não interessa)
T19	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = Outra string que não “-compress” flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T20	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = on nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T21	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = Números nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T22	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = Outra string que não “on” ou “off”

	nome do ficheiro = (não interessa) conteúdo do ficheiro = (não interessa)
T23	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = ficheiro com extensão .txt conteúdo do ficheiro = (não interessa)
T24	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = ficheiro com extensão .csv conteúdo do ficheiro = (não interessa)
T25	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = ficheiro com extensão .py conteúdo do ficheiro = (não interessa)
T26	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = ficheiro com extensão .png conteúdo do ficheiro = (não interessa)
T27	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = ficheiro com extensão .pdf conteúdo do ficheiro = (não interessa)
T28	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = conjunto de letras, números e caracteres especiais
T29	tamanho da janela = (não interessa) tamanho do buffer = (não interessa) flag a indicar operação = -compress flag a on/off = off nome do ficheiro = (não interessa) conteúdo do ficheiro = quaisquer caracteres que não sejam ASCII

Nota: quando é indicado que um valor para um determinado parâmetro "não interessa", assume-se que o mesmo deve encontrar-se dentro dos limites da variável.

ID	Output esperado	Output obtido
T1	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T2	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T3	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T4	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T5	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T6	**Terminal:** Mensagem de erro (emitida pelo Python e não pelo programa) **Ficheiros:** Nenhum(conjunto) variável	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T7	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T8	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T9	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T10	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T11	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor

T12	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T13	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T14	**Terminal:** Mensagem de erro (emitida pelo Python e não pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo Python e não pelo programa) **Ficheiros:** Nenhum
T15	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T16	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T17	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T18	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Nenhum **Ficheiros:** Nenhum
T19	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Nenhuma (se a string for "-decompress" indica uma mensagem a explicar o usage do programa) **Ficheiros:** Nenhum
T20	**Terminal:** "PREVIEW OF COMPRESSION: <preview>" **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** "PREVIEW OF COMPRESSION: <preview>" **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T21	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T22	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T23	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor

T24	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T25	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T26	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) ligeiramente superior
T27	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) ligeiramente superior
T28	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor
T29	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 2 ficheiros de tamanho (conjunto) variável menor

Como é possível observar, o programa é capaz de detetar a maioria dos erros (graças ao método mencionado anteriormente, que está exterior à função). No entanto, ainda existem alguns comportamentos que não deveriam ser permitidos, mais especificamente o uso de extensões que não são textuais, como .png e .pdf, que a funcionalidade de compressão apenas aceita e tenta comprimir de alguma forma. Além disso, outro comportamento que a função de compressão não leva em consideração é o uso de caracteres que não estão contidos na tabela ASCII, resultando na sua compressão inadequada. Algo de importante notar acerca destes dois ficheiros (extensões não textuais e caracteres não ASCII), é que, apesar de ser permitida a sua compressão, não é possível efetuar, posteriormente, a sua descompressão. Finalmente, em T6 é possível observar que não existe qualquer limitação para o tamanho máximo da janela, não sendo apresentado qualquer erro pelo *Python* quando o mesmo seria de esperar.

Um pequeno problema que também foi encontrado foi o facto deste programa falhar silenciosamente caso seja usada uma *flag* que não a "-compress", sendo que o comportamento mais adequado seria exibir uma mensagem de erro. Com base nos testes realizados, pode-se concluir que este compressor não realiza verificações quanto ao conteúdo comprimido nem às extensões utilizadas. Relativamente aos testes de classes de equivalência válidas, o mesmo reproduz o resultado esperado.

6.2.1.2) decompress(self, name)

De modo a simplificar a escrita das classes de equivalência, os parâmetros do *software* foram simplificados da seguinte forma:

- flag a indicar operação: FIO, no código constituiu a variável "sys.argv[3]" presente no *main*

- flag a on/off: FOO, no código constitui a variável “self.preview” presente na classe *deflate*
- nome do ficheiro principal: NFP, no código constitui a variável “name” que se encontra na função *decompress* integrada na classe *deflate*
- extensão: E, no código constituiu a variável “self.extension” presente na classe *deflate*
- conteúdo do ficheiro principal: CFP
- conteúdo do ficheiro huffman: CFH

Input	Equivalence classes válidas	Equivalence classes inválidas
Uma string FIO, tal que FIO = “-decompress”	-decompress	Números Números malformados Lixo Valores vazios Outra string que não “-decompress”
Uma string FOO, tal que FOO = “on” ou FOO = “off”	on off	Números Números malformados Lixo Valores vazios Outra string que não “on” ou “off”
Uma string NF, tal que NF apresenta o nome com uma extensão .bin	ficheiros com extensão .bin	ficheiros com extensão .txt ficheiros com extensão .csv ficheiros com extensão .js
Uma string E, tal que a mesma represente a extensão de um ficheiro	extensão .txt (desde que seja a extensão original) extensão .py (desde que seja a extensão original) extensão .csv (desde que seja a extensão original)	extensão que não seja a original
Um conjunto de strings CFP, tal que apresente caracteres binários	conjunto de caracteres binários	conjunto de caracteres que não sejam binários
Um conjunto de strings CFH, tal que apresente caracteres binários	conjunto de caracteres binários	conjunto de caracteres que não sejam binários

De seguida, apresentam-se os testes efetuados de modo a testar as diversas equivalence classes válidas e inválidas, onde primeiramente é apresentado o tipo de teste efetuado e, posteriormente, os resultados para os mesmos.

ID	Inputs (valor dado a cada parâmetro)
T1	flag a indicar operação = -decompress flag a on/off = on nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T2	flag a indicar operação = -decompress flag a on/off = Números nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T3	flag a indicar operação = -decompress flag a on/off = Outra string que não “on” ou “off” nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T4	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = ficheiro com extensão .bin extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T5	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = ficheiro com extensão .txt extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T6	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = ficheiro com extensão .csv extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T7	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = .txt (desde que seja a extensão original) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)

T8	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = .py (desde que seja a extensão original) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T9	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = extensão que não seja a original conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)
T10	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = conjunto de caracteres binários conteúdo do ficheiro huffman = (não interessa)
T11	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = conjunto de caracteres que não sejam binários conteúdo do ficheiro huffman = (não interessa)
T12	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = conjunto de caracteres binários
T13	flag a indicar operação = -decompress flag a on/off = off nome do ficheiro principal = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = conjunto de caracteres que não sejam binários
T14	flag a indicar operação = Outra string que não “-decompress” flag a on/off = off nome do ficheiro = (não interessa) extensão = (não interessa) conteúdo do ficheiro principal = (não interessa) conteúdo do ficheiro huffman = (não interessa)

Nota: quando é indicado que um valor para um determinado parâmetro "não interessa", assume-se que o mesmo deve encontrar-se dentro dos limites da variável.

ID	Output esperado	Output obtido
T1	**Terminal:** "PREVIEW OF DECOMPRESSION: <preview>" **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido	**Terminal:** "PREVIEW OF DECOMPRESSION: <preview>" **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T2	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T3	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum
T4	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T5	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T6	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T7	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T8	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T9	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T10	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T11	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum **Ficheiros:** Nenhum	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho menor que o original antes de ser comprimido

T12	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido	**Terminal:** (nada) **Ficheiros:** 1 ficheiro com tamanho original antes de ser comprimido
T13	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Mensagem de erro (emitida pelo Python e não pelo programa) **Ficheiros:** Nenhum
T14	**Terminal:** Mensagem de erro (emitida pelo programa) **Ficheiros:** Nenhum	**Terminal:** Nenhuma (se a string for "-compress" indica uma mensagem a explicar o usage do programa) **Ficheiros:** Nenhum

Foi possível observar que, à semelhança da funcionalidade de compressão, não existem verificações a nível do conteúdo dos ficheiros utilizados, neste caso a descomprimir, nem às extensões utilizadas. Atendendo ao teste T5, T6, T9, pôde-se verificar que o ficheiro é descomprimido ainda que a extensão não seja a original, preservando, contudo, o seu conteúdo original. Todavia, o comportamento mais correto seria verificar a extensão utilizada aquando a compressão, de modo a evitar a corrupção de ficheiros. Já em T11, o mesmo descomprime o ficheiro, ainda que este não possua um conteúdo binário, sendo de notar que, obviamente, o ficheiro obtido apresenta o conteúdo corrompido. Finalmente, em T13 é observada uma falta de *handle* de erros, já que o erro apresentado devia ser gerado pelo próprio programa e não pelo *stderr*.

Quanto aos restantes casos não há nada a apontar, o programa apresenta o *outcome* esperado sem qualquer incorreção.

6.2.2) Boundary Value Analysis

Para esta técnica procedeu-se à criação de diversos conjuntos de casos que representassem alguns dos *boundary cases*, consoante os valores que um determinado parâmetro pudesse assumir. Estes conjuntos foram organizados em tabelas com vista a facilitar a visualização dos mesmos e, posteriormente, elaborados casos de teste.

6.2.2.1) compress(self, str)

De modo a simplificar a escrita dos boundary cases, os parâmetros do *software* foram simplificados da seguinte forma:

- tamanho da janela: TJ, no código constitui a variável "self.window" presente na classe *deflate*
- tamanho do *buffer*: TB, no código constitui a variável "self.buffer" presente na classe *deflate*
- flag a indicar operação: FIO, no código constituiu a variável "sys.argv[3]" presente no *main*
- flag a on/off: FOO, no código constituiu a variável "self.preview" presente na classe *deflate*
- nome do ficheiro: NF, no código constituiu a variável "str" que se encontra na função compress contida na classe *deflate*
- conteúdo do ficheiro: CF

Input	Boundary Cases
Um número inteiro TJ, tal que 0 <= TJ <= 2**63-1	-1, 0, 1, 1000, 1001, 2**62-1, 2**62, 2**62+1, 2**63-10001, 2**63-10000, 2*63-1001, 2*63-1000, 2*63-2, 2**63-1, 2**63
Um número inteiro TB, tal que 0 <= TJ <= 256	-1,0, 1, 100, 101, 127, 128, 129, 155, 155, 255, 256, 257
Uma string FIO, tal que FIO = "-compress"	-compress, [a-zA-Z0-9]+"-compress"[a-zA-Z0-9]+
Uma string FOO, tal que FOO = "on" ou FOO = "off"	on, [a-zA-Z0-9]+"on"[a-zA-Z0-9]+ off, [a-zA-Z0-9]+"off"[a-zA-Z0-9]+
Uma string NF, tal que NF apresenta o nome com uma extensão de formato de texto	ficheiros com extensão .txt ou [a-zA-Z0-9]+".txt"[a-zA-Z0-9]+ ficheiros com extensão .csv ou [a-zA-Z0-9]+".csv"[a-zA-Z0-9]+ ficheiros com extensão .js ou [a-zA-Z0-9]+".js"[a-zA-Z0-9]+
Um conjunto de strings CF, tal que apresente caracteres ASCII	conjunto de letras, números e caracteres especiais conjunto de letras, números, caracteres especiais e caracteres UTF-8 fora do conjunto ASCII

6.2.2.2) decompress(self, name)

De modo a simplificar a escrita do boundary cases, os parâmetros do *software* foram simplificados da seguinte forma:

- flag a indicar operação: FIO, no código constituiu a variável "sys.argv[3]" presente no *main*
- flag a on/off: FOO, no código constitui a variável "self.preview" presente na classe *deflate*
- nome do ficheiro principal: NFP, no código constitui a variável "name" que se encontra na função *decompress* integrada na classe *deflate*
- extensão: E, no código constituiu a variável "self.extension" presente na classe *deflate*
- conteúdo do ficheiro principal: CFP
- conteúdo do ficheiro huffman: CFH

Input	Boundary Cases
Uma string FIO, tal que FIO = "-decompress"	-decompress, [a-zA-Z0-9]+"-decompress"[a-zA-Z0-9]+
Uma string FOO, tal que FOO = "on" ou FOO = "off"	on, [a-zA-Z0-9]+"on"[a-zA-Z0-9]+ off, [a-zA-Z0-9]+"off"[a-zA-Z0-9]+

Uma string NF, tal que NF apresenta o nome com uma extensão .bin	ficheiros com extensão .bin ou [a-zA-Z0-9]+“.bin”[a-zA-Z0-9]+
Uma string E, tal que a mesma represente a extensão de um ficheiro	extensão .txt (desde que seja a extensão original) extensão [a-zA-Z0-9]+“.txt”[a-zA-Z0-9]+ extensão .csv (desde que seja a extensão original) extensão [a-zA-Z0-9]+“.csv”[a-zA-Z0-9]+
Um conjunto de strings CFP, tal que apresente caracteres binários	conjunto de caracteres binários misturado com alguns caracteres ASCII
Um conjunto de strings CFH, tal que apresente caracteres binários	conjunto de caracteres binários misturado com alguns caracteres ASCII

6.2.4) Fuzzing

A utilização desta técnica acaba por estar englobada nos testes que foram efetuados previamente, já que foi utilizada uma grande quantidade de *inputs* de modo a testar o programa em questão. Como tal, neste capítulo não será abordada mais nenhuma técnica específica.

6.2.5) Considerações e outros aspetos

É de notar que tópicos como taxa de compressão, eficiência do algoritmo de compressão/descompressão e manifestação de riscos não foram especificamente testados, mas observando os resultados dos testes é possível tirar algumas conclusões.

No que diz respeito à corrupção ou perda de dados, é importante observar que, durante a descompressão, se um dos ficheiros (principal ou *huffman*) contiver um conteúdo inválido, o ficheiro descomprimido resultante apresentar-se-à corrompido e, portanto, os dados nele contidos serão perdidos. Já a nível de compatibilidade, o algoritmo *Deflate* mostrou-se extremamente útil para quaisquer ficheiros de texto (.txt, .csv, etc). Em termos de degradação do sistema onde o mesmo se encontra a ser executado, não foram encontrados quaisquer problemas. Contudo, foram detetadas algumas falhas relativamente às funcionalidades:

- por vezes a comprimir/descomprimir o ficheiro obtido não apresenta um tamanho esperado (no caso da compressão obtém-se um ficheiro maior do que esperado, no caso de descompressão obtém-se um ficheiro com tamanho diferente do original)
- apesar de constituir uma limitação para o programa, seria muito mais interessante (e obteria-se melhores resultados) caso o tamanho do *buffer* não fosse tão limitado.

No final acaba por ser um software no qual são encontradas algumas falhas e erros que facilmente poderiam ser corrigidos, mas que não constituem um enorme risco para o uso do mesmo. É de destacar que o mesmo é constituído por alguns handlings de erros, mas nem sempre são suficientes para segurar todos os erros possíveis neste programa. Ainda assim é um programa que cumpre bem o seu papel e quando bem configurado apresenta boas taxas de compressão e eficiência (algo concluído pelos testes efetuados).

6.3) Test Coverage

Neste capítulo pretende-se avaliar a performance dos testes efetuados ao longo deste plano e perceber o quão os mesmos cobriram as funcionalidades implementadas. De seguida, apresentam-se as métricas calculadas (anteriormente referenciadas) para cada um dos testes:

6.3.1) White-box testing (fase de control flow testing)

6.3.1.1) compress(self, str)

ID	Line coverage (%)	Method Coverage (%)	Data Coverge (%)
T1	(18/46 * 100) = 39.1	(9/14 * 100) = 64.3	(13/22 * 100) = 59.1
T2	(20/46 * 100) = 43.5	(10/14 * 100) = 71.4	(13/22 * 100) = 59.1
T3	(29/46 * 100) = 63.0	(13/14 * 100) = 92.9	(19/22 * 100) = 86.4
T4	(30/46 * 100) = 65.2	(14/14 * 100) = 100	(20/22 * 100) = 90.9
T5	(36/46 * 100) = 78.3	(14/14 * 100) = 100	(22/22 * 100) = 100
T6	(46/46 * 100) = 100	(14/14 * 100) = 100	(22/22 * 100) = 100

6.3.1.2) decompress(self, name)

ID	Line coverage (%)	Method Coverage (%)	Data Coverge (%)
T1	(18/27 * 100) = 66.7	(10/13 * 100) = 76.9	(16/17 * 100) = 94.1
T2	(18/27 * 100) = 66.7	(10/13 * 100) = 76.9	(16/17 * 100) = 94.1
T3	(20/27 * 100) = 74.1	(10/13 * 100) = 76.9	(16/17 * 100) = 94.1
T4	(20/27 * 100) = 74.1	(10/13 * 100) = 76.9	(16/17 * 100) = 94.1
T5	(24/27 * 100) = 88.9	(13/13 * 100) = 100	(17/17 * 100) = 100
T6	(27/27 * 100) = 100	(13/13 * 100) = 100	(17/17 * 100) = 100

6.3.2) Black-box testing

Os testes efetuados neste capítulo acabam por testar na íntegra as funcionalidades do *software*, pelo que as métricas anteriormente calculadas para os mesmos serão de 100% sempre.

6.3.3) Considerações finais

Como é possível constatar, os testes efetuados cobrem por completo o *software* testado, com exceção daqueles que percorrem caminhos mais curtos (por exemplo, em *white-box testing*). O motivo pelo qual o *software* foi amplamente testado é devido à sua pequena escala, portanto, os resultados obtidos nas métricas retiradas não são surpreendentes.

Pode-se, assim, concluir que, satisfatoriamente, os testes efetuados apresentam uma boa cobertura das funcionalidades testadas nas suas 3 vertentes: quantidade de código, quantidade métodos/funcionalidades e quantidade de dados.

7) O que o plano de testes entregou

Juntamente com este plano de testes pretende-se entregar:

- grafos desenvolvidos
- testes efetuados
- *software* testado
- ferramentas utilizadas
- cálculos das diferentes métricas

Não foram desenvolvidos quaisquer *scripts* ou ficheiros de *logs* durante este plano de testes, pelo que os mesmos não fazem parte da entrega

8) Necessidades para este plano de teste

As únicas especificações para execução deste plano de testes foi a utilização da linguagem *python* e suas respectivas bibliotecas. Estas encontram-se descritas de seguida:

- *sys* (para o código ser testado)
- *dahuffman* (para o código ser testado)
- *pickle* (para o código ser testado)
- *pycfg* (para a utilização da ferramenta *pycfg*)
- *argparse* (para a utilização da ferramenta *pycfg*)
- *tkinter* (para a utilização da ferramenta *pycfg*)
- *PIL* (para a utilização da ferramenta *pycfg*)
- *pygraphviz* (para a utilização da ferramenta *pycfg*)
- *Anaconda* (para a utilização da ferramenta *pycfg*)

9) Staffing e responsabilidades

Pessoa	Funções Únicas
Inês Marçal	6.1.1.2), 6.1.2.2), 6.2.1.2), 6.2.2.2), 6.3.1.2)
João Silva	6.1.1.1), 6.1.2.1), 6.2.1.1), 6.2.2.1), 6.3.1.1)

Notas:

- Os restantes pontos foram efetuados em conjunto
- O elemento Inês Marçal ajudou nos pontos 6.1.2.1) e 6.2.2.1), visto que a função de compressão era maior que a de descompressão

10) Conclusão

Através deste trabalho, foi possível compreender a importância da elaboração de um plano de testes no desenvolvimento/teste de um *software* específico, visando minimizar o número de falhas.

Durante o processo, foram realizadas várias etapas para obter um melhor entendimento do *software* em teste.

Começou-se, inicialmente, por obter um pouco de contexto do programa em questão, incluindo o seu modo de funcionamento, parâmetros, riscos e qual o fundamento para a aplicação deste plano de testes. De seguida, definiu-se um *scope* em torno do ambiente encontrado, de modo a limitar as funcionalidades e aspetos que iriam ser testados acerca das mesmas.

Após a recolha da informação acima referida, estabeleceu-se uma metodologia de testes, a qual inclui os domínios aplicados *(white-box, black-bo*x e *test coverage)* e as suas respectivas técnicas. Com base nisto, definiram-se as métricas e as ferramentas que iriam ser utilziadas. Por fim, o plano de testes criado foi implementado para recolher métricas e avaliar diferentes aspectos do *software*, incluindo sua execução, manipulação de dados e funcionamento. Os diferentes resultados foram discutidos nos respetivos capítulos de teste.

Como resultado, pode-se concluir que o *software* testado apresentou algumas falhas no funcionamento do mesmo, sobretudo na funcionalidade descompressão. Recomenda-se, como uma melhoria futura do *software*, focar nos seguintes pontos:

- *handling* mais robusto de possíveis exceções;
- verificação mais assertiva do conteúdo de ficheiros;
- verificações extra nos parâmetros utilizados no algoritmo de *Deflate* implementado;
- possível melhoramento na eficiência do algoritmo e nas estruturas de dados utilizadas.

11) Referências

- Slides da disciplina de Qualidade e Confiabilidade de Software
- https://www.geeksforgeeks.org/draw-control-flow-graph-using-pycfg-python/
- https://github.com/pygraphviz/pygraphviz/blob/main/INSTALL.txt
- https://github.com/mahkoCosmo/GraphViz_x64/blob/master/graphviz-2.38_x64.tar.gz
- http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-estatistica/codificador-qm/deflate/
- http://multimedia.ufp.pt/codecs/compressao-sem-perdas/codificacao-baseada-em-dicionarios/lz77/
- https://pt.wikipedia.org/wiki/LZ77
- https://www.researchgate.net/publication/228922323_An_Evaluation_of_Test_Coverage_Tools_in_Software_Testing
- https://airccse.org/journal/ijsea/papers/1011ijsea04.pdf
- https://www.airccse.org/journal/ijesa/papers/2212ijesa04.pdf
- https://wwjmrd.com/upload/a-research-paper-on-white-box-testing_1519477825.pdf
- https://www.asciitable.com
- https://app.diagrams.net