Manual de utilização - task 3

Grupo constituinte

- Guilherme Gaspar n.º 2020218933
- Maria Carolina Fernandes n.º 2021218374
- Pedro Nuno Monteiro n.º 2021218544

Manual

Na pasta *Task 3* é possível encontrar 4 ficheiros *.py* (functions.py, menu.spy, draw.py e task.py) e uma pasta *networks* com 4 ficheiros de texto *.txt* que correspondem às 4 redes, obtidas da SNDLib

Ficheiro ***task.py***

- Este é o ficheiro principal, aquele que deve ser corrido na linha de comandos para iniciar o programa
- Antes de executar o programa, é necessário criar uma pasta com o nome "output", de forma a evitar erros durante a execução. Nesta pasta serão guardadas todas as imagens geradas pelo programa, incluindo a imagem da rede inicial, a imagem da rede final com os caminhos definidos e, no caso do algoritmo de Suurballe, todas as imagens correspondentes aos diferentes passos do processo.
- Ao correr o programa, é apresentado um menu com 6 opções
    - Escolher 1 das 4 redes apresentadas
    - Inserir, em texto, o nome da rede
    - Sair
- Ao escolher uma das opções 1 a 4, o programa irá apresentar uma lista com os nós disponíveis da rede selecionada, criando, ainda, um grafo para ajudar na visualização
    - Após esta escolha e dependendo do algoritmo selecionado, o programa é direcionado para a função correspondente
        - *find_best_paths* caso Two Step Approach
        - *suurbale* caso Suurballe

- Escolhendo a opção 5, o programa irá apenas pedir ao utilizador para escrever, incluindo a extensão *.txt*, o nome da rede: para isso basta acrescentar na pasta onde estão presentes as restantes redes com a extensão .txt - em seguida, basta escrever o nome do ficheiro adicionado, com a respetiva extensão.
- Ao escolher a opção 6, o programa é terminado

Ficheiro ***menus.py***

- Este ficheiro foi adicionado ao projeto para melhorar a organização do mesmo. Aqui, estão as funções que apresentam ao utilizador os diferentes menus existentes

1. *ask_network()*
    a. Solicita ao utilizador o nome do ficheiro da rede, devolvendo este mesmo nome

2. *ask_origin_destiny()*
    a. Pede ao utilizador os nós de origem e destino, apresentando, para isso, uma lista com todos os nós da rede, devolvendo estes mesmos valores

3. *ask_which_algorithm()*
    a. Apresenta ao utilizador o menu para escolher o algoritmo a utilizar

Ficheiro ***functions.py***

- Este ficheiro contém agora 5 funções que servem de suporte ao programa principal. Comparando com a versão anterior, as funções responsáveis por desenhar os grafos foram movidas para o ficheiro *draw.py*, para uma melhor organização do projeto.

1. *retrieve_data(data)*
    a. É a função que, ao receber a informação *data* (que corresponde ao conteúdo de um ficheiro *.txt*), vai buscar a informação relativa aos **nós** e aos **arcos** de cada rede.
    b. A função faz esta busca e adiciona diretamente no grafo G linha-a-linha, inicialmente para os nós e depois para os arcos
    c. Retorna o grafo **G** e a a tabela ***node_mapping*** serve como tabela de conversão, ou seja, atribui a cada nó, um número para que seja mais fácil para o utilizador escolher o nó de origem e nó de destino.

2. *find_best_paths(G, origem, destino)*
    a. Recebe como parâmetros o grafo G, o nó de origem e o nó destino
    b. Calcula o **caminho mais curto** entre os nós de origem e destino, através do método Djisktra
    c. Em seguida, copiamos o grafo original e removemos todos os nós que pertenciam ao primeiro caminho, e calculamos o **segundo** caminho mais curto
    d. Este segundo caminho mais curto pode não existir se o nó for de primeira ordem ou se for um nó de

3. *suurballe(G, origem_orig, destino_orig)*
    a. Recebe como parâmetros o **grafo G**, o nó de **origem** e o nó de **destino**.
    b. Aplica a técnica de **node splitting**, dividindo cada nó em dois (ex: A_in, A_out) para permitir caminhos disjuntos.
    c. Calcula o caminho mais curto entre origem e destino no grafo com node splitting, utilizando o algoritmo de **Dijkstra**.
    d. Copia o grafo e transforma-o:
        i. Calcula os **custos reduzidos** para todos os arcos com base nas distâncias anteriores.

ii. **Remove** os arcos do primeiro caminho que vão no sentido inverso à origem.
iii. **Inverte** os arcos do primeiro caminho (com custo 0).

e. Calcula um **segundo** caminho mais curto no grafo transformado:
i. Este caminho pode não existir se a estrutura da rede não permitir dois caminhos disjuntos

f. **Remove** sobreposições entre os dois caminhos
g. **Reconstrói** os caminhos finais e converte os nós divididos de volta para os nomes originais.
h. Retorna os dois caminhos disjuntos entre origem e destino.

4. *split_nodes(G, source_orig, target_orig)*
   a. Recebe como parâmetros o **grafo G**, o nó de **origem** e o nó de **destino originais**.
   b. Cria um novo grafo direcionado H, onde cada nó do grafo original é dividido em dois: **nó de entrada** (A_in) e **nó de saída** (A_out).
   c. Adiciona uma aresta com **custo nulo** (cost = 0) entre A_in e A_out para cada nó original.
   d. Para cada **aresta (u, v)** do grafo original, cria uma nova aresta entre u_out e v_in com o mesmo custo.
   e. Define o novo nó de origem (s) como **source_out** e o novo destino (t) como **target_in**.
   f. Retorna o novo grafo H com os nós divididos, a nova **origem s** e o novo **destino t**

5. *merge_split_path(split_path)*
   a. Recebe como parâmetro uma lista de nós **split_path** correspondente a um caminho no grafo com node splitting
   b. Cria uma nova lista chamada **original_path**, onde vai armazenar o caminho convertido para os nomes dos nós originais.
   c. Para cada nó no caminho dividido:
      i. Remove o sufixo **_in** ou **_out**, recuperando o nome original do nó.
      ii. Adiciona esse nome à lista original_path, garantindo que não haja repetições consecutivas.
   d. **Retorna** o caminho convertido com os nomes dos nós originais

Ficheiro ***draw.py***

- Este ficheiro embarca as diferentes funções utilizadas para os desenhos dos grafos - é uma junção de algumas funções previamente definidas noutros ficheiros, com a adição da nova função para desenho do suurballe

1. *draw_empty_network(G, node_mapping)*
   a. Recebe como argumentos o grafo G e a tabela de conversão node_mapping
   b. Antes de ser pedido os nós origem e destino, é apresentado o grafo G, com os respectivos nós já numerados, de forma a que os utilizadores possam antes de escolher, observar a rede na sua totalidade.

2. *draw_network(G, node_mapping, origem, destino, caminho1, caminho2, algoritmo)*
    a. Recebe como parâmetros: **grafo** direcionado **G**, o **node_mapping, nó de origem** e **destino,** os **caminhos** destacados o pelos algoritmos e o qual o **algoritmo** que o utilizador queira usar.
    b. Cria rótulos dos nós no formato "1: Nome", usando o node_mapping
    c. Converte os índices de origem e destino nos nomes correspondentes.
    d. Define a cor de cada nó: verde para a origem, vermelho para o nó destino e azul claro para os restantes.
    e. Desenha todas as arestas a vermelho, sendo que o primeiro caminho +e representado a verde e o segundo caminho a azul, se existir.
    f. Adiciona rótulos às arestas com os seus custos.
    g. Guardao grafo como imagem em "output/rede_final.png"

3. *draw_suurballe(G, origem_split, destino_split, caminho1_split, caminho2_split, filename)*
    a. Recebe como parâmetros: o **grafo G**, a **origem_split,** o **destino_split,** lista de nós do **primeiro** e **segundo caminho** e o nome do **arquivo** para guardar o gráfico.
    b. Para facilitar essa visualização, utilizamos a função draw_suurballe, que gera gráficos interativos e guarda as imagens representando cada passo do algoritmo.
        i. Mostra o estado do grafo após cada etapa de execução do Suurballe
        ii. Destaca os caminhos encontrados durante o cálculo
        iii. Ilustra a transformação do grafo, incluindo a divisão dos nós no formato _in/_out