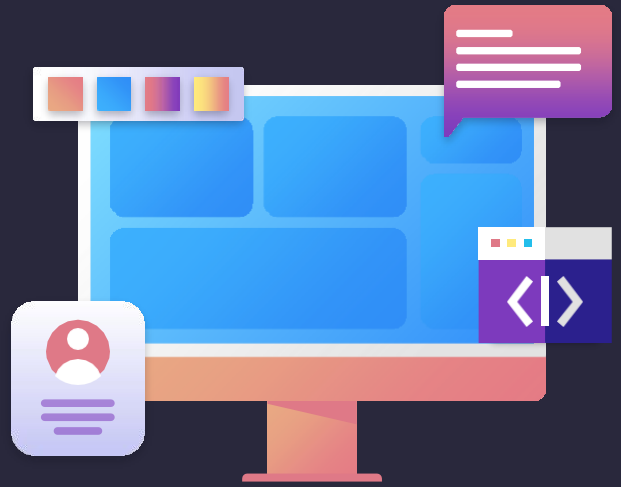
G88

PROJETO DA-2

Artur Oliveira - 202108663
Gonçalo Pinho- 202108672
Tomás Sucena Lopes- 202108701



C++, CMAKE, DOXYGEN

/INDÍCE

* Identificação do trabalho e elementos do grupo;

* Descrição do problema;

* Descrição da solução, identificação de algoritmos relevantes;

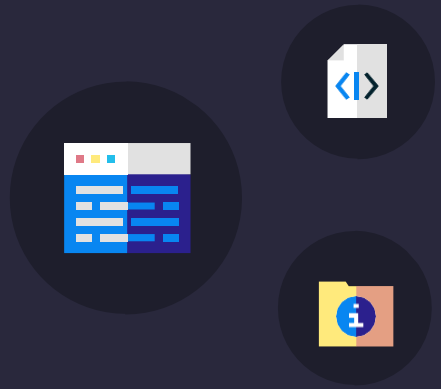
* Destaque de alguma funcionalidade;

/PROBLEMA

Quer-se um programa para correr e visualizar soluções para o problema do Travelling Salesman Problem (TSP) em vários grafos.

Durante a execução do trabalho, deparámo-nos com diversas dificuldades, nomeadamente:

* Desenvolvimento de algoritmos;
* Calcular trajetos;
* Display das informações;
* Otimização de algoritmos.



/SOLUÇÃO

* Uso de algoritmos e conceitos dados nas aulas (Prim, Dijkstra,…);

* Criou-se um grafo para mais fácil utilização dos dados disponibilizados nos ficheiros csv;

* Uso da libraria libfort para mais fácil display;

* Refactoring do código para otimização;

* Usou-se o Git para facilitar o trabalho em equipa.

C++, CMAKE, DOXYGEN

/FUNCIONALIDADES

/MENU

```
Hello! How can I be of assistance?

* Change
* Display
* Run
* Toggle
```

Ao executar o programa, é pedido ao utilizador que escolha uma operação a executar. O utilizador volta ao menu após cada operação.

/RUN TSP

Nesta funcionalidade, o programa irá correr 1 de 3 algoritmos para resolver o TSP.

```
These are the results of my computation:
```

N	Source	Destination	Distance	Total Distance
1	0	3	450	450
2	3	2	500	950
3	2	1	450	1400
4	1	4	450	1850
5	4	0	750	2600

```
Total distance: 2600 m
Execution time: 0ms
```

/FUNCIONALIDADES/TSP

Esta funcionalidade permite encontrar soluções para o TSP

```
std::list<std::pair<int, double>> TSPGraph::backtracking(int src){
    std::list<std::pair<int, double>> bestPath;
    double minDistance = INF;

    std::vector<int> indices;
    for (int i = 1; i <= countVertices(); ++i)
        if (i != src) indices.push_back(i);

    if (matrix.empty()) matrix = toMatrix();

    do {
        int prev = src;

        std::list<std::pair<int, double>> currPath;
        double currDistance = 0;

        for (int i : indices) {
            if (matrix[prev][i] < 0)
                matrix[prev][i] = distance( src: prev,  dest: i);
```

/Backtracking

Computa uma solução do TSP, com recurso a brute-force e backtracking

/FUNCIONALIDADES/TSP

```
std::list<std::pair<int, double>> TSPGraph::triangularInequality(int src) {
    std::list<Edge *> MST = getMST( root: src);
    if (matrix.empty()) matrix = toMatrix();

    // set up the algorithm
    for (Edge *e: edges)
        e->valid = false;

    for (Edge *e: MST) {
        e->valid = true;
        (*this)[e->getSrc()].valid = true;
    }

    // compute the path using DFS
    std::list<std::pair<int, double>> path;
    double (TSPGraph::*dist)(int, int) = isReal ? &TSPGraph::haversine : &TSPGraph::distance;

    std::stack<int> s;
    s.push( x: src);

    (*this)[src].valid = false;
    int prev = src;

    while (!s.empty()) {
        int curr = s.top();
        s.pop();
```

/Triangular

Computa uma solução aproximada para o TSP, com base na heurística Triangular Inequality

/FUNCIONALIDADES/TSP

```cpp
std::list<std::pair<int, double>> TSPGraph::other(int src) {
    if (matrix.empty()) matrix = toMatrix();

    // set up the algorithm
    resetAll();

    // execute Nearest-Neighbours to obtain the initial path
    double distance;
    std::vector<int> initialPath = nearestNeighbours(src, &distance);

    // use 2-opt to optimize the path
    twoOpt(&initialPath, &distance);

    // compute the final path
    std::list<std::pair<int, double>> path;

    int curr = src;
    for (int i : initialPath) {
        path.emplace_back(a: i, b: matrix[curr][i]);
        curr = i;
    }

    path.emplace_back(a: src, b: matrix[curr][src]);
    return path;
}
```

/Other

Computa uma solução aproximada para o TSP, com recurso ao Nearest-Neighbours e ao 2-opt

apresentacao.eof();