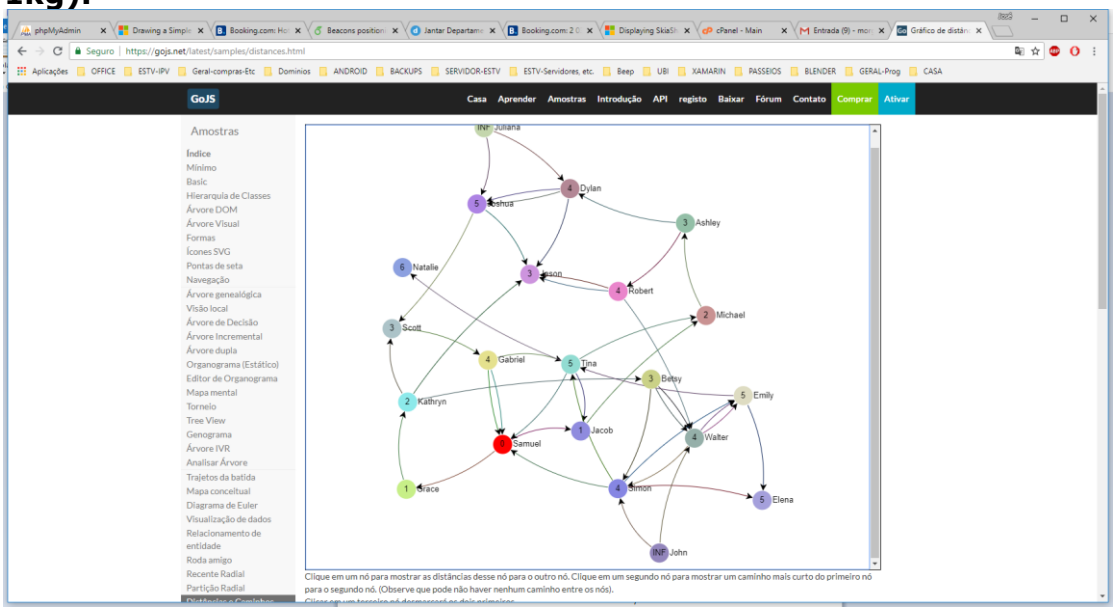
Projeto Prático – 2018/2019
Grafos

Uma das situações atuais é a vinda para a Europa de muitas pessoas de países Africanos, assim pretende-se um programa que permita monitorizar as movimentações das pessoas. As pessoas quando saem dos seus países, têm de atravessar vários tipos de fronteiras.

- **Considere que os vértices do grafo são fronteiras de países que estão em guerra, essas fronteiras podem ser oficiais ou são controladas por melícias.**
- **Existem 3 tipos de Fronteiras (Fronteira Oficial, Fronteira tipo1, Fronteira Tipo 2)**

Quando uma pessoa passa pela "Fronteira Oficial", recebe algum tipo de informação do país (não pagando nada), quando passa pela Tipo1 tem de pagar um dado valor para passar a fronteira, quando passa pela do Tipo2 tem de pagar e levar consigo uma dada mercadoria para entregar no destino! (Considere que esta mercadoria tem sempre 1kg).



É fornecido no projeto os seguintes ficheiros de dados:

- "Pessoas.txt", ficheiro de pessoas que pretendem atravessar dos seus países de origem para a europa.
- "grafo_1.txt", "grafo_2.txt", etc. São os dados das várias fronteiras por onde pode passar.

Nas funcionalidades que abaixo são pedidas, são apresentados os protótipos dos métodos a implementar, assumindo que tem de implementar a classe Grafo (pode implementar outras que considere necessárias!!!)

Atenção!: Uma pessoa nunca pode transportar mais de 10 Kg!, isto quer dizer que não pode passar em mais de 10 fronteiras…. Que obrigam a levar 1Kg!

(Considere Vértice ⇔ Fronteira)

Funcionalidades a implementar (cotação: 1 Valor cada alínea):

1. Carregar os dados de ficheiros (do grafo e de pessoas);
   bool Grafo::Load(const string &fich_grafo, const string &fich_pessoas);

2. Contar o número de nós/fronteiras do grafo;
   int Grafo::ContarNos();

3. Contar o número de arestas/arcos do grafo;
   int Grafo::ContarArcos();

4. Determinar toda a memória ocupada;
   int Grafo::Memoria();

5. Determinar qual o nó/fronteira que tem mais arcos/aresta, se existirem vários devolve uma lista deles;
   List<int> *Grafo::NoMaisArcos();

6. Verificar se um nó é adjacente de outro nó do Grafo;
   bool Grafo::Adjacencia(int v1, int v2);

7. Determinar um caminho(não interessa se é o mais rápido!, é qualquer um!) de um nó/fronteira para outro.... devolvendo o custo total;
   List<int> *Grafo::Caminho(int v1, int v2, double &custo_total);

8. Determinar quais os vértices que estão isolados (Um vértice é isolado se não existe nenhum caminho até ele!);
   List<int> *Grafo::VerticesIsolados();

9. Verificar se um dado vértice existe
   bool Grafo::Search(int v);

10. Remover um dado vértice, também será necessário remover todas as arestas;
    bool Grafo::RemoverVertice(int v);

11. Remover a aresta que liga 2 vértices;
    bool Grafo::RemoverAresta(int v1, int v2);

12. Gravar para ficheiro em formato XML todas as informações do Grafo;
    void Grafo::EscreverXML(const string &s);

13. Ler de um ficheiro em formato XML todo o Grafo (antes de ler deve ser apagado tudo o que estiver no Grafo); Se leu corretamente devolve true, senão devolve false;
    bool Grafo::LerXML(const string &s);

14. Devolver uma lista de todos os vértices/fronteiras de um dado tipo;
List<int> *Grafo::DevolveVerticesTipo(const string &tipo);

15. Determinar o caminho mínimo entre 2 nós, devolvendo o custo total;
List<int> *Grafo::CaminhoMinimo(int v1, int v2, double &custo_total);

16. Determinar o caminho máximo entre 2 nós (passando somente uma vez em cada vértice), devolvendo o custo total;
List<int> *Grafo::CaminhoMaximo(int v1, int v2, double &custo_total);

17. Será possível ir de uma fronteira(v1) a outra(v2), passando somente por fronteiras de um dado tipo?
bool Grafo::PossivelCaminho(int v1, int v2, int TipoFronteira);

18. Assumindo que todas as pessoas (do ficheiro) andaram em viagem, qual foi a fronteira com menos tráfego/visitas? (No caso de serem várias, deve devolver uma lista com as fronteiras);
void Grafo::FronteirasMenosVisitadas(List<int> &Lv);

19. Implemente o destrutor do Grafo;
Grafo::~Grafo();

20. Dada uma fronteira (v1), determinar todas as fronteiras onde não é possível chegar apartir de v1. Deve devolver uma lista dessas fronteiras!;
Grafo::SitiosInacessiveisAPartirDe(int V1, list<int> &Lv);

Observações:

- Os alunos não devem alterar os métodos apresentados acima!.
- Podem e devem criar novos métodos (sempre privados ou protegidos) com os nomes que acharem por bem!
- Devem seguir sempre uma abordagem orientada aos objetos!.

Avaliação / Observações:

- Todos os programas serão corridos automaticamente por um programa feito pelos docentes da disciplina, se no processo automático a avaliação for inferior a 9 valores, o trabalho é rejeitado!, não sendo avaliado pelos docentes!
- Se a gestão da memória, não estiver correta, haverá uma penalização de 5 valores;
- Se for detetado "copianço", trabalho anulado!
- Os grupos são constituídos por 1, 2 ou 3 alunos;
- Eventuais dúvidas serão esclarecidas e anexadas em documento

colocado no moodle.

Entrega:

- Época Normal – 06/01/2019;
- Época Recurso – 04/02/2019;