JEU DE LA LETTRE QUI SAUTE

Auteurs :

- Leo PERARD
- Salla DIAGNE

Listing des fichiers et répertoires du projet

- *bin/* : contient les fichiers compilés (.class)
- *src/* : contient les fichiers source (.java)
- *test/* : contient les tests unitaires

Fonctionnement du programme

- Se placer à la racine du projet (dossier tp1_perard-diagne/)
- exécuter la commande : java -jar lettre.jar mot1 mot2
  - Ce programme recherche et affiche le chemin le plus court entre mot1 et mot2 dans le graphe Dicos.dico4

Réponses aux questions

Exercice 1 : Modélisation et initialisation du jeu

Graphe par liste d'adjacence

Voir classes `MotGraphe` et `Graphe` : * constructeur `MotGraphe(...)` * constructeur `Graphe(...)`

Création des listes de successeurs

Voir classe `Graphe` : * méthode `ajouteArete(int s, int d)` * méthode `initListeSuccesseursMot(int indiceMot)` * méthode `initListeSuccesseursTousMots()`

Exercice 2 : Calcul des composantes connexes

Parcours en profondeur : préparation

Voir classe `Graphe`. * constructeur `Graphe(...)`

Parcours en profondeur : en partant d'un nœud

Voir classe `Graphe` : * méthode dfs(int x)

```
public static void main(String[] args) {
```

```
    final String[] dico3Court = { "gag", "gai", "gaz", "gel", "gks",
         "gin",
         "gnu", "glu", "gui", "guy", "gre", "gue", "ace", "acm", "agi",
         "ait", "aie", "ail", "air", "and", "alu", "ami", "arc", "are",
         "art", "apr", "avr", "sur", "mat", "mur" };
    final Graphe graphe = new Graphe(dico3Court, 0, 1);
    graphe.dfs(0);
```

```
$ gag gai gaz gui guy gue gre are ace acm aie ait ail air apr avr art arc
```

Parcours en profondeur : visiter tous les nœuds

Voir classe `Graphe` : * méthode visit()

Le graphe composé des mots de `Dicos.dico4` comporte 138 composantes connexes. La composante connexe de lion et peur est : drap (...) peur (...) lion (...) trot

Exercice 3 : Calcul de chemins

Arborescence

Voir classe `Graphe` : * méthode dfs(int x) * méthode visit()

Retrouver le chemin du parcours

Voir classe `Graphe` : * méthode chemin(int s, int d, boolean verbose)

Un chemin de lion à peur : lion pion paon pain vain vais mais hais haie hait haut vaut vaux veux veuf oeuf neuf nerf cerf serf sera aera aere here hele tele fele fela fele hele hale hall halo hale hase vase jase jade cade cage cake cane cafe came lame lama lame pame page sage sake sape sapa saga sana sang sans sais jais jars gars pars paru parc pari sari sali soli joli poli polo solo silo kilo kilt tilt tint vint vent vend venu tenu tetu fetu feru fera fora fort font foot flot flou clou clot ilot plot plut peut veut vert vers sers sens cens cene cent sent sert sort mort mont vont voit toit tort tors mors mord moud moue mode rode roda rode role mole mile mike mise miss mess mens mans mars mare mure bure buse muse mule male gale gala gata hata hate hote hate have rave race rame pame pane pale pape papa pama rama rame dame dome tome toge tige pige pire aire aine aile aide aine aile aise aise vise vice vive vite mite mita mite site sire lire lice lise vise bise base jase jasa nasa basa base bale pale pate gate gate gite rite rate rare tare taie trie tria trio brio bric eric cric crin coin foin loin loir voir vois voix voie vote hote home houe houx toux tous mous vous nous bous bois buis puis cuis cris fris frit fret pret pres ores ores tres gres gras gros gris iris pris pois poix noix noir soir soif soin sain sait lait laid caid raid rail bail bain gain nain nais nait nuit fuit fuir ouir ouie suie suif suis suit huit cuit coit cout aout aout tout tour four cour jour pour peur

Exercice 3 : Calcul de plus courts chemins

Parcours en largeur

Voir classe `Graphe` : * méthode bfsIteratif(int s, int d, boolean verbose)

Plus court chemin

Plus court chemin de lion à peur : lion pion paon pain paix poix poux pour peur

Exercice 5 : Généralisation

Graphe orienté généralisé

Voir classes `MotGraphe` et `Graphe` : * constructeur `MotGraphe(...)` * constructeur `Graphe(...)`

Excentricité