Mini-Projet: Challenge Unesco

Ce mini-projet s'étale sur deux TPs (séances 7 et 8). Il est noté et vous devrez présenter vos résultats et votre code lors d'une soutenance organisée le mercredi 5 décembre après-midi. Vous devrez également soumettre votre code **commenté** sur Moodle avant le mardi 4 décembre minuit (heure de Paris).

Vous pouvez travailler seul ou en binôme. Si vous travaillez en binôme, vous devez être capable d'expliquer l'ensemble du code, même pour des parties du code sur lesquelles vous n'avez pas travaillé.

L'ensemble des informations et fichiers nécessaires à ce mini-projet peuvent être récupérés sur Moodle.

1 Présentation

Vous êtes un fan de voyage, c'est la fin de l'année scolaire et vous avez trois semaines de vacances devant vous ! Vous décidez de saisir cette opportunité pour réaliser votre rêve : visiter les endroits les plus incroyables de la terre. Votre lieu de départ est connu sous forme LAT/LONG et vous disposez d'un hélicoptère parfait (y compris le pilote) qui voyage à la vitesse constante de 80km/h et possède un réservoir illimité. Vous décidez de vous limiter à la visite de sites UNESCO inscrits au patrimoine de l'humanité, qui sont de trois types : endroits culturels (tels que les centres historiques), les endroits naturels (tels que les parc nationaux) et les endroits mixtes qui représentent les deux. Vous décidez que vous voulez visiter autant de sites que possible, avec un nombre égal (à un près) de sites culturels et naturels (les sites mixtes comptent pour naturel et culturel).
Le challenge est de trouver un itinéraire, partant de n'importe quel endroit LAT/LONG, qui va vous permettre de visiter autant de sites que possible, et revenir à votre point de départ. Le temps maximum de votre voyage est de 3 semaines, et vous supposez que vous allez passer 6 heures par site, comprenant la visite, les repas/boissons et le repos (que vous pourrez aussi réaliser dans l'hélicoptère). Vous évaluerez la qualité de votre itinéraire de la façon suivante :

- Chaque site UNESCO visité compte pour 1 point
- Chaque pays différent visité compte pour 2 points
- Chaque site qui est listé comme "en danger" compte pour 3 points

Votre but est de trouver un itinéraire permettant d'obtenir le score le plus élevé.
Lors de la soutenance, un point de départ sera aléatoirement choisi et les trois binômes ayant trouvés les meilleurs scores seront récompensés.

Contraintes :

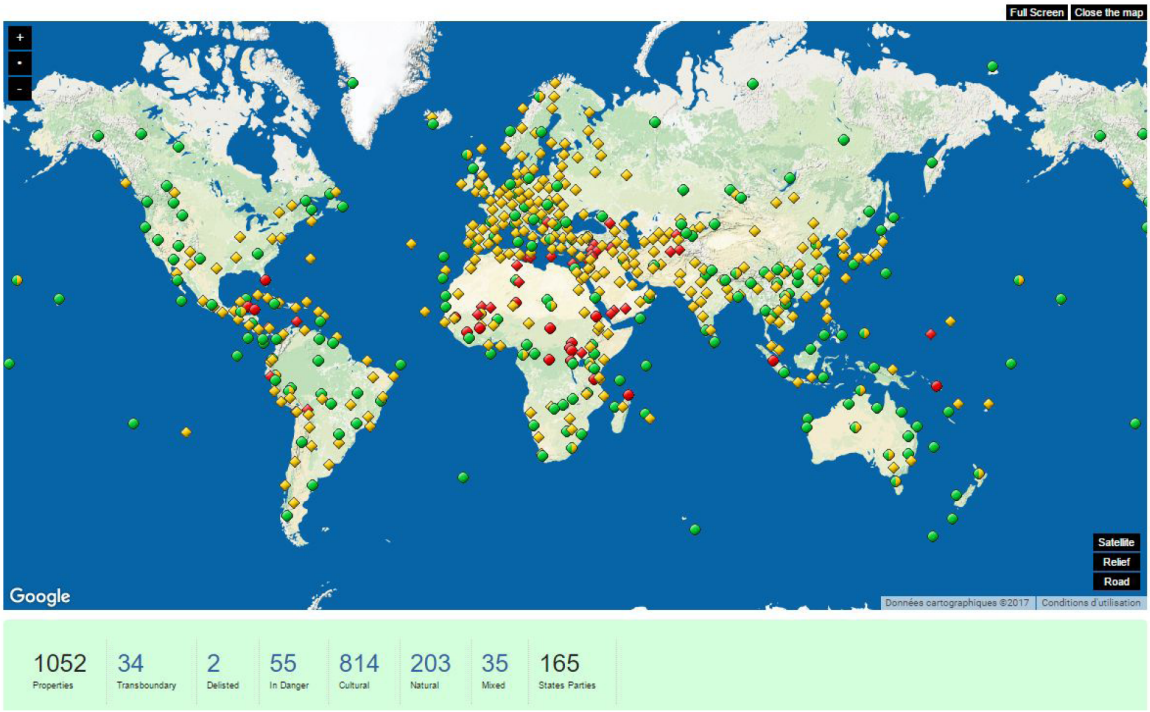
Le programme doit accepter en argument les coordonnées LAT et LONG du point départ (par exemple `./main 48.8464111 2.3548468`, si vous partez de la place Jussieu).
En sortie du programme, vous devez afficher l'itinéraire de voyage, y compris les noms des sites, leurs types (culturel, naturel ou mixtes) et le pays, ainsi que l'évaluation finale de l'itinéraire. Vous devez donc obtenir un résulat semblable à celui-ci :

```
Welcome to the Unesco travel challenge!
Generation:
******
TRAVEL ITENARY
0)Starting point = (48.8464111, 2.3548468)
1)Paris Banks of the Seine / Cultural / France
2)Dorset and East Devon Coast / Natural / United Kingdom of Great Britain and Northern Ireland
3)Liverpool â?? Maritime Mercantile City / Cultural / United Kingdom of Great Britain and Northern Ireland / Endangered
4)Mill Network at Kinderdijk-Elshout / Cultural / Netherlands
5)Stevns Klint / Natural / Denmark
6)HolaÅ¡ovice Historic Village / Cultural / Czechia
7)Historic Centre of the City of Salzburg / Cultural / Austria
8)Millenary Benedictine Abbey of Pannonhalma and its Natural Environment / Cultural / Hungary
9)Caves of Aggtelek Karst and Slovak Karst / Natural / Slovakia
10)Durmitor National Park / Natural / Montenegro
11)Medieval Monuments in Kosovo / Cultural / Serbia / Endangered
12)Archaeological Site of Cyrene / Cultural / Libya / Endangered
13)Abu Mena / Cultural / Egypt / Endangered
14)Ancient City of Damascus / Cultural / Syrian Arab Republic / Endangered
15)Old City of Jerusalem and its Walls / Cultural / Jerusalem (Site proposed by Jordan) / Endangered
16)Palestine: Land of Olives and Vines â?? Cultural Landscape of Southern Jerusalem Battir / Cultural / Palestine / Endangered
17)Simien National Park / Natural / Ethiopia / Endangered
18)Garamba National Park / Natural / Democratic Republic of the Congo / Endangered
19)Okapi Wildlife Reserve / Natural / Democratic Republic of the Congo / Endangered
20)Manovo-Gounda St Floris National Park / Natural / Central African Republic / Endangered
21)Air and TÃ0nÃ0rÃ0 Natural Reserves / Natural / Niger / Endangered
22)ComoÃ0 National Park / Natural / CÃ´te d'Ivoire / Endangered
23)Mount Nimba Strict Nature Reserve / Natural / CÃ´te d'Ivoire / Endangered
24)Niokolo-Koba National Park / Natural / Senegal / Endangered
25)Old Town of GhadamÃ¨s / Cultural / Libya / Endangered
26)Archaeological Site of Sabratha / Cultural / Libya / Endangered
27)Archaeological Site of Leptis Magna / Cultural / Libya / Endangered
28)San Marino Historic Centre and Mount Titano / Cultural / San Marino
29)Plitvice Lakes National Park / Natural / Croatia
30)Å kocjan Caves / Natural / Slovenia
31)Monte San Giorgio / Natural / Italy
32)Swiss Alps Jungfrau-Aletsch / Natural / Switzerland
33)Messel Pit Fossil Site / Natural / Germany
34)City of Luxembourg: its Old Quarters and Fortifications / Cultural / Luxembourg
35)Notre-Dame Cathedral in Tournai / Cultural / Belgium
36)Starting point = (48.8464111, 2.3548468)

Number of cultural sites: 18
Number of natural sites: 17
Number of mixed sites: 0
Distance travelled = 23125,22 km
Time travelled = 499,07 hours (max = 504 hours)
Evaluation:
        35 destinations x 1 point
        29 countries x 2 points
        18 endangered x 3 points
Overall score: 147
```

2 Lecture des données et génération d'un premier itinéraire

Il y a plus de 1000 sites Unesco dans le monde, qui sont représentés sur l'image ci-dessous. Vous pouvez trouver plus d'informations sur les sites ici : `http://whc.unesco.org/en/list/`.



Les données relatives aux différents sites sont données dans un fichier joint au projet (fichier "unesco.csv"). Dans ce fichier, vous trouverez pour chaque site :

- Nom
- Latitude
- Longitude
- Catégorie (Cultural, Natural ou Mixed)
- Pays
- Continent
- En danger ou pas sous forme booléenne (1 si le site est en danger, 0 sinon)

Exercice 1 – Lecture des données

Q 1.1 Structure de données associée à un site Unesco
Pour sauvegarder les informations liées à un site Unesco, vous pourrez utiliser la structure suivante, que vous créerez dans un fichier `site.h`.

```
typedef struct site{
    char* nom;
    float LAT;
    float LONG;
    char* categorie; //cultural,natural,mixed
    char* pays;
    int enDanger; //0,1
```

```
} Site;
```

Dans un fichier `site.c` ajoutez une méthode de signature `Site construireSite(char* nom,float LAT,float LONG,char* categorie,char* pays,int enDanger);` permettant de construire une variable de type `Site` à partir des informations données en arguments.
Ajoutez également une methode `void affichageSite(Site s);` permettant d'afficher les caractéristiques d'un site.
Testez et validez vos méthodes sur de petits exemples.

Q 1.2 Lecture des données du fichier "unesco.csv"
Nous vous conseillons ensuite de construire un tableau de `Site` de taille égale au nombre de sites présents dans le fichier.
Chaque case du tableau contiendra les sites présents dans le fichier "unesco.csv". Pour lire ce fichier, vous pouvez vous aider des différentes fonctions présentes dans les fichiers "lectureFichiers.c" et "lectureFichiers.h" à télécharger sur Moodle. Les fonctions `GetChaine()` (lire la prochaine chaîne de caractères avant une virgule ou une fin de ligne), `GetReel` (lire le prochain réel) et `GetEntier` (lire le prochain entier) sont particulièrement utiles.

Exercice 2 – Distance entre sites

Pour calculer la distance entre deux points du globe terrestre, vous utiliserez la formule de Haversine (`https://fr.wikipedia.org/wiki/Formule_de_haversine`). L'implémentation de cette méthode vous est donnée ci-dessous et dans les fichiers "harversine.c" et "haversine.h" téléchargeables sur Moodle.

```c
#include <math.h>
#include "haversine.h"

double toRad(double angleD){
    return (angleD * M_PI / 180.0);
}

double haversin(double val){
    return pow(sin(val / 2), 2);
 }

double calculDistance(double nLat1, double nLon1, double nLat2, double nLon2){
    double dLat  = toRad(nLat2 - nLat1);
    double dLong = toRad(nLon2 - nLon1);
    double startLat = toRad(nLat1);
    double endLat = toRad(nLat2);
    double a = haversin(dLat) + cos(startLat) * cos(endLat) * haversin(dLong);
    double c = 2 * atan2(sqrt(a),sqrt(1 - a));
    return RAYON_TERRE * c;
}
```

Q 2.1 Validation du calcul de distance
Vérifiez que vous obtenez bien une distance de 182.27 km entre les points (50,3) et (49,5) et une distance de 18806.70 km entre les points (-85.7,-170.6) et (83.2,165.3).

Q 2.2 Sauvegarde des distances
Créez une matrice de distances que vous remplirez avec les distances entre les différents sites. Notez qu'il est également utile de sauvegarder les distances entre le point de départ et les différents sites.

Exercice 3 – Génération d'un premier itinéraire

Un itinéraire est composé d'une suite de sites à visiter. Le nombre de sites à visiter n'étant pas fixé, il vous est demandé d'utiliser une liste chaînée pour sauvegarder les informations d'un itinéraire. De plus, comme il est utile de pouvoir insérer des élements en début et fin de liste rapidement, il est préférable d'utiliser une liste doublement chaînée. Vous pourrez par exemple utiliser la réprésentation suivante :

```
1  typedef struct celluleLDC{
2    Site s; // Site
3    //Ou int i; //Indice du site dans un tableau contenant l'ensemble des sites
4    struct celluleLDC* prec; /* pointeur sur l'element precedent de la liste */
5    struct celluleLDC* suiv; /* pointeur sur l'element suivant de la liste */
6  } CelluleLDC;
7
8  typedef struct {
9    CelluleLDC* premier;  /* Pointeur sur element en tete */
10   CelluleLDC* dernier;  /* Pointeur sur element en fin */
11 } LDC;
```

Q 3.1 Gestion de la liste chaînée
Ajoutez les fonctions permettant de créer une cellule (ou "maillon") de votre liste chaînée, d'insérer un élément en tête et en fin, de désallouer l'espace mémoire occupé par la liste, d'afficher la liste, etc. Testez et validez vos méthodes sur de petits exemples.

Q 3.2 Méthode du plus proche voisin
Implémentez une première méthode permettant d'obtenir un itinéraire rapidement, basée sur l'algorithme du "plus proche voisin". Cett méthode fonctionne de la manière suivante : on construit un itinéraire en choisissant à chaque fois le site le plus proche du site que l'on est train de visiter. Attention, différentes conditions seront à vérifier :

- On doit toujours être capable de revenir à notre point de départ (c'est-à-dire avoir assez de temps pour retourner à notre point de départ après la visite d'un site ; le temps total de l'itinéraire étant limité à 3 semaines)
- On doit visiter autant de sites culturels que naturels (à un près). Une façon simple de gérer cette contrainte est d'alterner les visites naturels/culturels (ou mixtes car les sites mixtes comptent pour naturels et culturels)
- Il faut bien sûr éviter de visiter un site deux fois

Q 3.3 Affichage de la solution obtenue
Affichez l'itinéraire obtenu avec cette méthode, et vérifiez qu'il respecte bien les contraintes du problème.

Q 3.4 Calcul du score
Calculez et affichez le score obtenu avec cette méthode.

Exercice 4 – Visualisation d'un itinéraire

Il peut être pratique de visualiser (sur une carte) un itinéraire.
Pour cela, une petite application écrite en `Java` est disponible sur Moodle. Cette application a besoin en entrée d'un fichier nommé "Tour.txt" contenant les coordonnées des différents sites de l'itinéraire, du point de départ et du point d'arrivée (égal au point de départ). Par exemple :

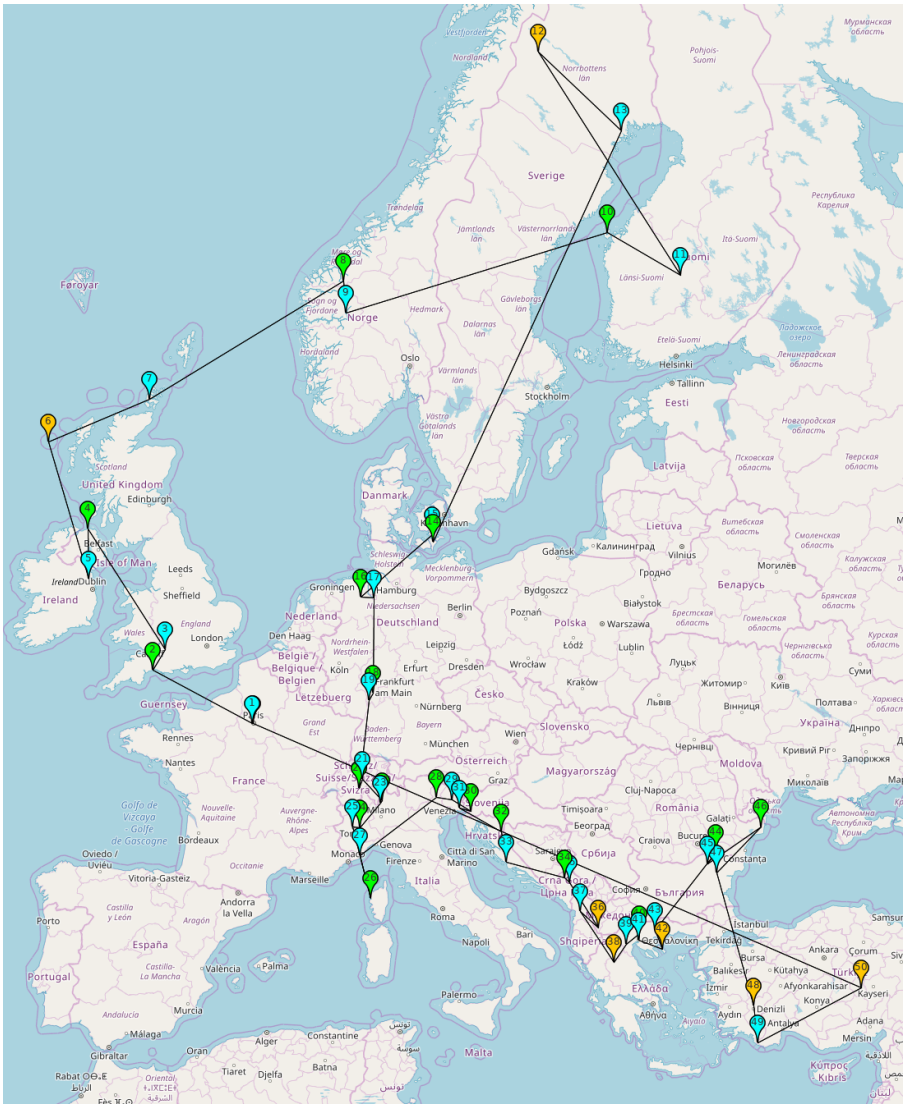
Tour.txt :

```
48.846413, 2.354847
48.858334, 2.294167, Cultural
50.705555, -2.989889, Natural
51.381390, -2.358611, Cultural
55.250000, -6.485278, Natural
48.846413, 2.354847
```

Pour lancer cette application à partir d'un code C, il suffit d'ajouter les instructions suivantes :

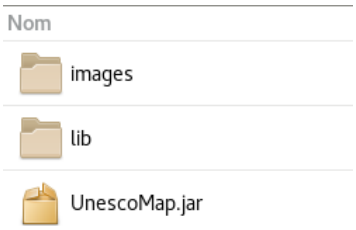
```
const char* commandLine="java -jar UnescoMap.jar";
system(commandLine);
```

Vous pourrez ainsi créer une fonction d'affichage d'itinéraire où vous créerez dans un premier temps le fichier "Tour.txt", et ensuite lancerez l'application de visualisation.
Pour la méthode du plus plus proche voisin, en partant de la place Jussieu, vous devriez obtenir un itinéraire ressemblant à cela :



Les points verts correspondent à des sites naturels, bleus à des sites culturels et oranges à des sites mixtes.
Pour info, le score obtenu par cette solution est de 93 (50 sites visités, 20 pays visités, 1 site en danger visité), mais il est possible de faire bien mieux !

Remarque :
Pour pouvoir lancer l'application de visualisation, il est nécessaire que les dossiers "images" et "lib" téléchargeables sur Moodle soient dans le même dossier que l'application `UnescoMap.jar`, comme illustré sur la figure ci-dessous :



Il est aussi nécessaire d'être connecté à Internet pour que l'affichage de la carte fonctionne.

3 Optimisation : A vous de jouer !

Vous remarquerez que dans l'algorithme du plus proche voisin, on ne tient pas compte du fait que les sites en danger rapportent trois points et que le nombre de pays visité rapporte deux points.
La séquence de visites est également loin d'être optimale, puisque des croisements entre routes reliants les sites sont présents.
De plus, avec cette méthode, on a tendance à s'éloigner du point de départ, sans tenir compte du fait qu'il faudra y revenir à la fin du tour.
Des adaptations sont donc nécéssaires pour augmenter le score de l'itinéraire, et ainsi pouvoir profiter au maximum de vos vacances de trois semaines.
Attention : votre méthode doit généré rapidement un itinéraire (c'est-à-dire en moins de 30 secondes).

4 Conseils

- Décomposez votre code en différents fichiers, et utilisez un `makefile` pour compiler l'ensemble des fichiers
- Testez et validez chaque nouvel élément de votre code avant d'aller plus loin
- Commencez par implémenter des méthodes simples pour trouver un itérinaire, que vous pourrez améliorer par la suite
- Avant de sortir de la fonction principale, n'oubliez pas de désallouer la mémoire qui a été allouée dynamiquement (pour chaque malloc, un free !)
- En cas de bogue (ce qui risque d'arriver !), n'hésitez pas à utiliser les outils de débogage comme `gdb`, `ddd`, ou `valgrind` vus au cours.
- N'oubliez pas de commenter votre code