Partie théorique (30 points ; 45 min)

Question 1 — 16 (=3+10+3) points

(1) Exposer l'idée de l'algorithme du ***tri rapide*** (***quicksort***).

(2) Ecrire une version ***récursive*** de cet algorithme pour une liste de chaînes de caractères que l'on souhaite trier par ordre alphabétique.

(3) On trie la liste

('I','N','F','O','R','M','A','T','I','Q','U','E')

à l'aide de l'algorithme précédent. Quels changements cette liste subit-elle au cours de la <u>première</u> exécution de la procédure de partitionnement (division) ?

Question 2 — 7 points

Ecrire une ***version itérative*** de l'algorithme de recherche dichotomique.

Question 3 — 7 (=4+1+2) points

Soit la fonction suivante **d** qui prend comme paramètre une liste d'entiers :

```
function d(lbA:TListBox):integer;
var i,a,b:integer;
begin
  a:=strtoint(lbA.Items[0]);
  b:=strtoint(lbA.Items[0]);
  for i:=1 to lbA.Items.Count-1 do
    if strtoint(lbA.Items[i])>a then a:=strtoint(lbA.Items[i])
    else if strtoint(lbA.Items[i])<b then b:=strtoint(lbA.Items[i]);
  result:=a-b
end;
```

(1) Supposons que la liste **lbA** contienne les éléments :

('3','8','5','1','9','7','-3','0','7').

Calculer **d(lbA)** en suivant l'évolution des variables **a** et **b** dans un tableau.

(2) Que calcule la fonction **d** en général.

(3) Lorsqu'on appelle la fonction **d** avec une ***liste vide*** comme paramètre, on obtient un message d'erreur. Pourquoi ? Modifier la fonction afin de remédier à cet inconvénient.

Partie pratique (30 points ; 70 min)

Un voyageur de commerce doit faire le tour de n villes. Il cherche un circuit fermé de longueur totale minimale, passant une et une seule fois par toutes les villes (à l'exception de la ville de départ qui est égale à la ville d'arrivée). Ce problème, dit du voyageur de commerce, est plus compliqué qu'il n'y paraît ; on ne connaît pas de méthode de résolution permettant d'obtenir des solutions exactes en un temps raisonnable si le nombre de villes est grand. Nous nous limiterons dans cette simulation Delphi à calculer la longueur d'un chemin choisi « manuellement » par le voyageur de commerce sans essayer de trouver le circuit optimal.

(1) Créer l'interface graphique contenant les éléments suivants (**6 points**):

- un ***stringgrid*** **`sgMap`** à 21 lignes et 21 colonnes dont 1 ligne et une colonne fixes, sur lequel seront placées au hasard 8 villes : **`'A'`**, **`'B'`**, … , **`'H'`** (cf. (3));
- un ***stringgrid*** **`sgDist`** à 9 lignes et 9 colonnes, dont 1 ligne et une colonne fixes, qui contiendra les distances entre les différentes villes (cf. (4));
- une ***editbox*** **`edtTour`**, dans lequel l'utilisateur écrira le circuit dont il cherche à calculer la longueur ; cette boîte d'édition contiendra par défaut le circuit **`'ABCDEFGHA'`**;
- la longueur d'un circuit sera écrite dans un ***label*** **`lblResult`** qui affiche « **Résultat :** » au lancement de l'application ;
- trois boutons avec les mentions respectives : « **Placer les villes au hasard** », « **Remplir la matrice des distances** » et « **Calculer la longueur du cricuit** » ;
- des ***labels*** explicatifs comme sur la *figure 1* ci-dessous :

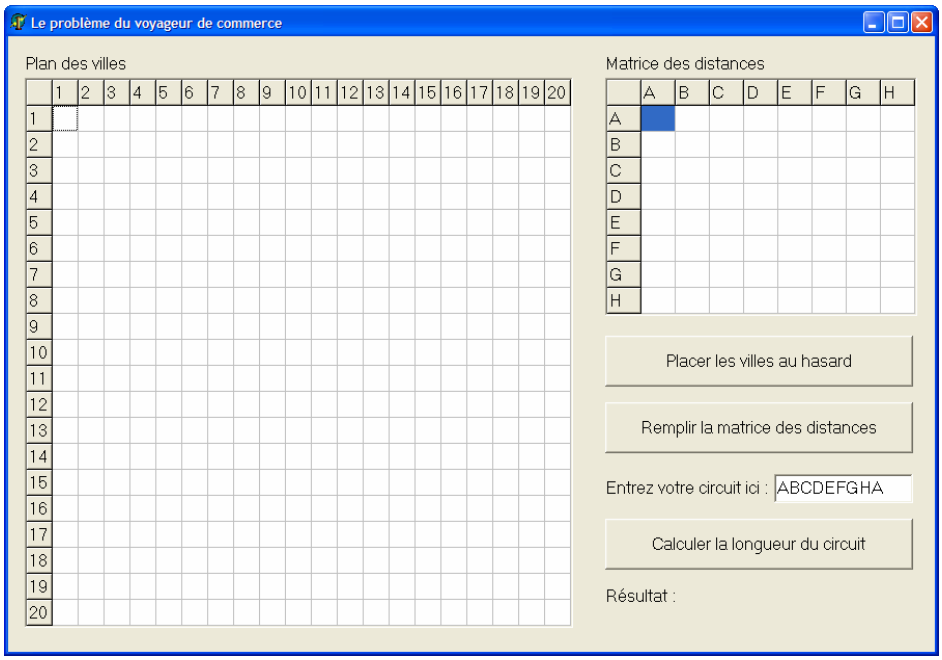

figure 1 : interface graphique au démarrage de l'application

(2) Au démarrage de l'application, les colonnes et lignes fixes des deux stringgrids devront être initialisés comme le montre la *figure 1*. En outre, on initialisera le générateur de nombres aléatoires. On utilisera l'événement « ***OnCreate*** » du formulaire. (**4 points**)

(3) Le voyageur cliquera d'abord sur le bouton « **Placer les villes au hasard** », dont l'effet est de vider d'abord les stringgrids **`sgMap`** et **`sgDist`**, puis de placer les huit lettres **`'A'`**, **`'B'`**, **`'C'`**, **`'D'`**, **`'E'`**, **`'F'`**, **`'G'`**, et **`'H'`** dans des cellules choisies au hasard sur le stringgrid **`sgMap`**. Chaque cellule ne devra contenir qu'une lettre au plus. Le voyageur devra aussi pouvoir placer les 8 villes « manuellement » sur le stringgrid **`sgMap`**. ***Indication algorithmique*** : stocker en même temps les « coordonnées » de chaque ville (c.-à-d. les indices de la colonne et de la ligne où elle se trouve) dans un stringgrid caché sur le formulaire. (**8 points**)

(4) Le voyageur cliquera ensuite sur le bouton « **Remplir la matrice des distances** », dont l'effet est de remplir le stringgrid **`sgDist`** avec les distances entre les villes. Afin de faciliter les calculs, la distance (dite de « Manhattan ») entre les villes **`'A'`** et **`'B'`** sera calculée par la formule :

$$\left|\ \text{colonne}(\texttt{'A'}) - \text{colonne}(\texttt{'B'})\ \right| + \left|\ \text{ligne}(\texttt{'A'}) - \text{ligne}(\texttt{'B'})\ \right|$$

où : colonne(**`'A'`**) est l'indice de la colonne dans laquelle se trouve **`'A'`** et ligne(**`'A'`**) est l'indice de la ligne dans laquelle se trouve **`'A'`**.

Dans l'exemple d'exécution de la *figure 2*, cette distance entre les villes **`'A'`** et **`'B'`** est égale à $|2-14|+|11-8| = 12+3 = 15$. (**6 points**)

(5) Finalement le joueur entrera un chemin dans la boîte d'édition **`edtTour`** (pas obligatoirement fermé et contenant autant de villes qu'il veut) et cliquera sur le bouton « **Calculer la longueur du cricuit** ». Le programme affichera alors la longueur totale du chemin (somme des distances des différentes étapes) dans le libellé **`lblResult`**. (**6 points**)

Tourner s.v.p

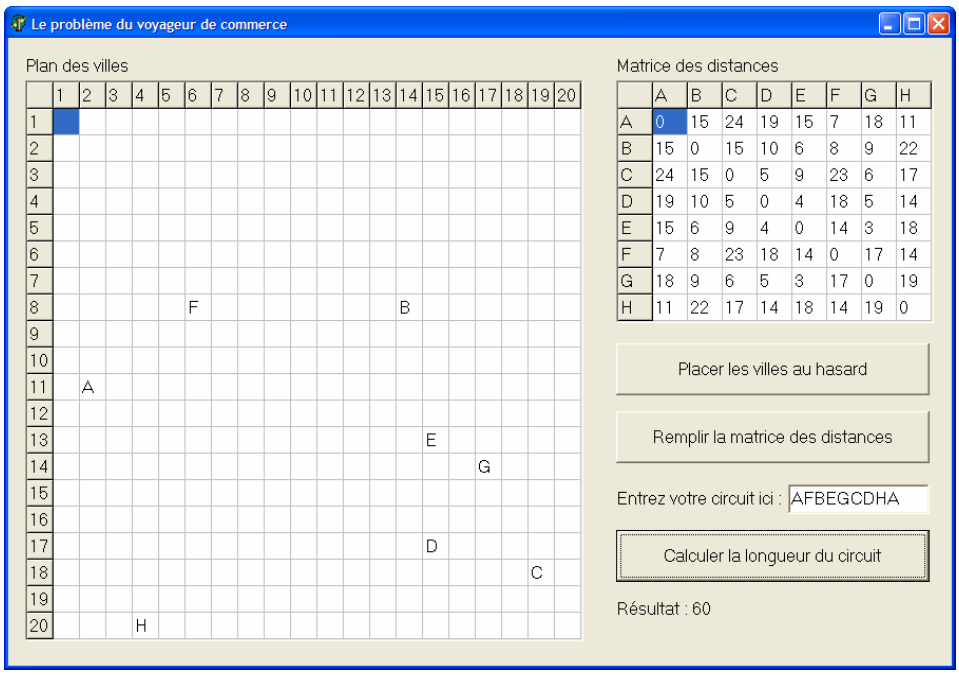

figure 2 : exemple d'exécution

G. Lorang