Rapport TP4/TP5 IA

Théo Gayant – Duc Viet Nguyen

> TP4 :

Pour la modélisation de notre taquin nous avons choisis un tableau à 1 dimension.

> Fonctions utilisées :

- int estTaquinBut(int Taquin[], int n) →
  Permet d'indiquer si le taquin passé en paramètre est un taquin but, selon sa taille nxn.

- int taquin3x3Solvable(int Taquin[]) →
  Indiquer si le taquin 3x3 passé en paramètre est solvable ou non. On compte la somme du nombre d'inversion pour chaque case (sauf la case vide), si cette somme est paire c'est solvable.

- int indiceCaseVide(int Taquin[], int n) →
  Retourne l'indice de la case vide dans le taquin de taille nxn.

- int ** taquinsSucceseur(int * taq) →
  Retourne les successeurs du taquin 3x3 passé en paramètre.
  On retourne un tableau de taille 4, contenant le successeur du taquin selon la direction, l'indice 0 pour le HAUT, l'indice 1 pour le BAS, l'indice 2 pour la GAUCHE, l'indice 3 pour la DROITE.
  Si un des déplacements n'est pas possible on remplace le taquin successeur par NULL.

- int h_malplacees(int taquin[]) →
  Retourne le résultat de la fonction heuristique h1, on retourne le nombre de case qui sont mal placées.

- int h_manhattan(int taquin[]) →
  Retourne le résultat de la fonction heuristique h2, on retourne la somme des déplacements nécessaire afin que chaque case soit à sa place but.

- int estRetour(char c1, char c2) →
  Permet d'indiquer si on ne fait pas un déplacement « bête », c1 est le dernier déplacement que l'on a fait et c2 est le déplacement que l'on souhaite faire.
  Par exemple si on vient d'aller en HAUT, on ne vas pas aller en BAS juste après.

- struct Element →
  Permet de stocker les infos d'un taquin, dans cette structure on stocke le taquin, le chemin qu'on a emprunté afin d'arrivé à ce taquin depuis le taquin initial, le résultat de la fonction h et g.

- Element * rechercheIDA(int * taquin, int (*h)(int *)) →
  Permet de résoudre un taquin 3x3 si celui est solvable,
  On passe en paramètre le taquin initial, et une pointeur vers la fonction heuristique voulu.
  La fonction retourne le taquin but obtenu, avec le chemin parcouru afin d'arrivé à celui ci.

- void taquinTxtToTaquin(char * taquinTxt, int * taquin) →
  Convertit un taquin sous forme de chaîne en un taquin sous forme de tableau.
  Exemple : '0 1 2 3 4 5 6 7 8' → {0,1,2,3,4,5,6,7,8}

- + Utilisation d’une pile

> Fonctionnement du programme :

Il faut avoir un fichier d’entré, « entree.txt » (uniquement ce nom d’entrée), avec le même format qui est décrit dans le sujet du TP4. Si ce fichier n’est pas présent ERREUR.

Ensuite le programme va créer un fichier « sortie.txt » dans lequel il va mettre le résultat pour chaque taquin du fichier en entré.

Puis le programme lis, ligne par ligne le fichier en entré.
Il convertit le taquin sous forme de texte en un tableau grâce à la fonction taquinTxtToTaquin.
Après le programme vérifie si ce taquin est solvable en appelant la fonction taquin3x3Solvable.
Si celui n’est pas solvable on l’indique dans le fichier de sortie.
Si il est solvable, on appelle la fonction rechercheIDA sur ce taquin, qui nous retourne un Element *e.
Grâce à cette élément on écrit dans le fichier de sortie le nombre de déplacement afin d’obtenir ce taquin depuis le taquin initial et le chemin parcouru.
Après il vous suffit de consulter le fichier de sortie afin d’analyser les résultats obtenu.

/!\ Il faut bien respecter la syntaxe du fichier en entrée, ici on a supposé qu’il n’y a aucune erreur d’écriture dans celui-ci /!\

Compiler :
$ gcc tp.c -o tp
Exécuter :
$ ./tp entree.txt

Un fichier d’entrée d’exemple est disponible dans le dossier tp4, vous pouvez le modifier tant que vous respecté la syntaxe du fichier.

> TP5 v1:

Les fonctions utilisées précédemment sont également utilisé ici et elles ont juste était adaptés pour le taquin 4x4.

> Nouvelles fonctions :

- int deplacementG2(int taquin[], char c) →
  C’est la fonction G2, elle indique si on à déplacé une case qui appartient au taquin but partiel actuellement utilisée. ‘c’ est le dernier déplacement du taquin.

- int h_manhattan_partiel(int taquin[], int taquin_partiel[]) →
  Retourne le résultat de la fonction heuristique Manhattan, appliqué uniquement au case spécifié dans le taquin partiel.

> Fonctionnement du programme :

Le fonctionnement est identique au tp4, vous devez mettre vos taquins initiale dans le fichier ‘entree.txt’ (ici on a retiré la ligne du taquin but), il faut écrire son taquin sur une ligne avec un seule espace entre chaque nombre. Vous trouverez un exemple de fichier en entrée avec déjà plusieurs taquins.
Vous pouvez donc en essayer d’autre en ajoutant des lignes.
Pour le reste c’est le même procédé.

Compiler :
$ gcc tp.c -o tp
Exécuter :
$ ./tp entree.txt

Résultat disponible dans le fichier ‘resultat.txt’.

> Étude expérimentale :

bp1 = {0,-1,-1,3,-1,-1,-1,7,-1,-1,-1,11,12,13,14,15};
bp2 = {0,1,2,-1,4,5,6,-1,8,9,10,-1,-1,-1,-1,-1};

Fichier entree.txt →
1 10 2 3 4 5 9 7 8 6 14 12 11 13 0 15
1 2 0 3 4 5 6 7 8 9 10 11 12 13 14 15
2 1 0 3 4 5 6 7 8 9 10 11 12 13 14 15

Fichier sortie.txt →
Taquin initial : 1 10 2 3 4 5 9 7 8 6 14 12 11 13 0 15
Distance de Manhattan : 17
BP1 - G1 : 27 GGHDBDHDBGHGBDDHGHGGBBDHGHH
BP1 - G2 : 15 HDBGGGHDDHGGBDBGHHDDBGBDDHGGGHH
BP2 - G1 : 19 HHDHGGBBDHHDBBGHGHG
BP2 - G2 : 12 DHHHGGBBDHDBBGGHDHHDBBGHGHG

Taquin initial : 1 2 0 3 4 5 6 7 8 9 10 11 12 13 14 15
Distance de Manhattan : 2
BP1 - G1 : 2 GG
BP1 - G2 : 0 GG
BP2 - G1 : 2 GG
BP2 - G2 : 2 GG

Taquin initial : 2 1 0 3 4 5 6 7 8 9 10 11 12 13 14 15
Pas Solvable

Analyse :
En comparant le max entre BP1-G1 & BP2-G1 et la somme de BP1-G2 et BP2-G2,
On remarque qu’ils sont égalent.

Problème rencontré :
Si l'on met des taquins très dure à résoudre, cela peut prendre énormément de temps.
Cela est probablement dû à une mauvaise gestion de la mémoire et des choix algorithmique implémenté par nous-même.

> TP5 v2:

Version du tp5 avec les 4 taquins but partiels décrit dans le nouveau sujet du TP.
On a ajouté et modifié les taquins but précédent afin d'avoir les 4 taquins but partiels demandé.

> Fonctionnement du programme :

Ce programme fonctionne de la même manière que le TP5 V1.
On utilise juste plus la fonctions G1, uniquement la fonction G2.

Analyse :

bp1 = {0,-1,-1,3,-1,-1,-1,7,-1,-1,-1,11,12,-1,-1,-1};
bp2 = {0,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,13,14,15};
bp3 = {0,-1,2,-1,-1,5,6,-1,-1,9,10,-1,-1,-1,-1,-1};
bp4 = {0,1,-1,-1,4,-1,-1,-1,8,-1,-1,-1,-1,-1,-1,-1};

Exemple de resultat :
Taquin initial : 4 5 6 2 0 3 10 9 12 11 14 8 7 13 1 15
Distance de Manhattan : 27
BP1 - G2 : 11 HDDDBBGGBDDHGGBGHDDBGHDBGHGHHDDBGHDDBGBGGHHDDDBGGGH
BP2 - G2 : 3 DBBDHDHHGGGBBBDHDDHHGGG
BP3 - G2 : 7 DBDHDBBGGGHHDDDBBGGHHDHDBBBGGGHHDHG
BP4 - G2 : 8 DDBDHGGGBBDDDHHGGBDDHHGGBGBDBDDHHGGBDDHHGGBGH

On remarque la somme des coût de chaque taquins but partiel, est supérieur à la distance de Manhattan.