CORRIGE

ECUE «Introduction à la programmation »

13 juin 2012
sans document - durée 2 heures
Bruno Bouzy

On souhaite écrire un programme C `chemins.c` qui exécute des épisodes au cours desquels des agents se déplacent sur un damier pour atteindre leurs buts.

Il y a `5` agents et le damier possède `5x5 = 25` cases. Un agent est repéré par un numéro appartenant à `{0, 1, 2, 3, 4}`. Chaque agent occupe une case `(x, y)` du damier où `x` est le numéro de la colonne et `y` le numéro de la ligne. `x` et `y` appartiennent à `{0, 1, 2, 3, 4}`. A chaque itération, les agents se déplacent sur une case voisine ou reste sur la même case. Une case ne peut être occupée que par un seul agent. Une case est reliée à 2, 3 ou 4 voisines selon que la case est au coin, au bord ou au milieu du damier. Un exemple d'exécution d'un épisode du programme est représenté sur les 3 colonnes ci-dessous:

```
episode avec graine 1:

positions et buts:
---------------
.. .. 1. 2. ..
.3 .. .4 02 4.
.. 3. .0 .. ..
.. .. .. .. ..
.1 .. .. .. ..
---------------
h = 6;

episode: debut

iteration 1:
positions et buts:
---------------
.. 1. .. .. 4.
.3 .. 04 22 ..
3. .. .0 .. ..
.. .. .. .. ..
.1 .. .. .. ..
---------------
fini = 5

iteration 2:
positions et buts:
---------------
.. .. .. .. ..
33 1. .4 22 4.
.. .. 00 .. ..
.. .. .. .. ..
.1 .. .. .. ..
---------------
fini = 4

iteration 3:
positions et buts:
---------------
.. .. .. 2. ..
33 .. .4 42 ..
.. 1. 00 .. ..
.. .. .. .. ..
.1 .. .. .. ..
---------------
fini = 3

iteration 4:
positions et buts:
---------------
.. .. .. .. ..
33 .. 44 22 ..
.. .. .0 0. ..
.. 1. .. .. ..
.1 .. .. .. ..
---------------
fini = 2

iteration 5:
positions et buts:
---------------
.. .. .. .. ..
33 .. 44 22 ..
.. .. 00 .. ..
.. .. .. .. ..
.1 1. .. .. ..
---------------
fini = 1

iteration 6:
positions et buts:
---------------
.. .. .. .. ..
33 .. 44 22 ..
.. .. 00 .. ..
.. .. .. .. ..
11 .. .. .. ..
---------------
fini = 0

episode: fin.

longueur = 6
```

Les positions et les buts de agents sont affichés au début de l'épisode. Pour chaque case, le programme affiche 3 caractères:

le numéro de l'agent situé sur la case s'il en existe un, un '.' sinon,
le numéro de l'agent si la case est le but d'un agent, un '.' sinon,
un caractère espace ' '.

Par exemple, initialement, l'agent `3` est situé sur la case `(1, 2)` située sur la colonne `1` et la ligne `2`, et son but est sur la case `(0,1)` située sur la colonne `0` et la ligne `1`. A chaque itération, les agents se déplacent sur une case voisine ou restent immobiles. Par exemple, après l'itération `1` de l'épisode, l'agent `3` est allé sur la case `(0, 2)`. A l'itération `2`, l'agent `3` est arrivé sur son but. Les itérations se poursuivent tant que tous les agents n'ont pas tous atteints leurs buts. A l'itération `6`, tous les agents ont atteints leur but et l'épisode se termine. A la fin d'un épisode, le programme affiche la longueur de l'épisode. Après chaque itération, le programme estime le nombre d'itérations restantes et l'affiche.

Le but de cet exercice est de programmer `chemins.c`. La question 1 traite de la structure des données utilisées: des tableaux unidimensionnels. La question 2 traite des fonctions gérant la position des agents ou buts sur le damier et l'affichage du damier. La question 3 traite des fonctions estimant le nombre d'itérations restantes. La question 4 traite de l'exécution d'un pas de temps ou itération: souhaits des agents, évitements des collisions et déplacements effectifs. La question 5 traite du déroulement d'un épisode. Enfin, la question 6 est relative au programme principal.

Les tableaux à plusieurs dimensions n'étant pas au programme de cette UE, l'exercice n'utilise que des tableaux unidimensionnels.

Question 1 (2 points)

1) a) Définir 2 constantes `N_AGENTS`, et `TAILLE` dont la valeur est `5`.

```
#define TAILLE 5
#define N_AGENTS 5
```

Pour représenter les positions des agents on utilise deux tableaux unidimensionnels `pos_x` et `pos_y` d'entiers (`int`) de taille `N_AGENTS`. `pos_x[a]` et `pos_y[a]` contiennent respectivement le numéro `x` de la colonne et le numéro `y` de la ligne de la position de l'agent de numéro `a`. De manière analogue, `dep_x` et `dep_y` sont des tableaux contenant les points de départs des agents. `but_x` et `but_y` sont des tableaux contenant les buts des agents.

1) b) Déclarer les 6 tableaux `pos_x`, `pos_y`, `dep_x`, `dep_y`, `but_x` et `but_y`.

```
  int dep_x[N_AGENTS], dep_y[N_AGENTS];
  int pos_x[N_AGENTS], pos_y[N_AGENTS];
  int but_x[N_AGENTS], but_y[N_AGENTS];
```

1) c) Donner le contenu des 6 tableaux correspondant au damier du début de l'épisode donné en exemple.

L'énoncé est ambigu selon que l'on prend le premier diagramme de la sortie du programme ou le second (après le debut de l'épisode)

si on prend

```
---------------
.. .. 1. 2. ..
.3 .. .4 02 4.
.. 3. .0 .. ..
.. .. .. .. ..
.1 .. .. .. ..
---------------
```

alors:

```
dep_x: 3 2 3 1 4
dep_y: 1 0 0 2 1
pos_x: 3 2 3 1 4
pos_y: 1 0 0 2 1
but_x: 2 0 3 0 2
but_y: 2 4 1 1 1
```

(les positions == les departs)

si on prend

```
---------------
.. 1. .. .. 4.
.3 .. 04 22 ..
3. .. .0 .. ..
.. .. .. .. ..
.1 .. .. .. ..
---------------
```

alors:

```
dep_x: 3 2 3 1 4
dep_y: 1 0 0 2 1
pos_x: 2 1 3 0 4
pos_y: 1 0 1 2 0
but_x: 2 0 3 0 2
but_y: 2 4 1 1 1
```

Question 2 (5 points)

2) a) Ecrire une fonction `agentSurXY` retournant le numéro de l'agent situé sur la case `(x, y)` et ayant la signature suivante:
`int agentSurXY(int x, int y, int *tx, int * ty);`
`tx` et `ty` sont des pointeurs vers les tableaux des positions des agents. Si aucun agent n'est situé sur la case `(x, y)` la fonction retourne `-1`.

```
int agentSurXY(int x, int y, int *tx, int * ty) {
  int a; for (a=0; a<N_AGENTS; a++) if ((tx[a]==x) && (ty[a]==y))
return a;
  return -1;
}
```

2) b) Ecrire une fonction `afficher` affichant le damier avec la position des agents et leurs buts et ayant la signature suivante:
`void afficher(int * tx, int * ty, int * bx, int * by);`
`tx` et `ty` sont des pointeurs vers les tableaux des positions. `bx` et `by` sont des pointeurs vers les tableaux des buts . L'affichage respectera celui de l'épisode exemple.

```
void afficher(int * tx, int * ty, int * bx, int * by) {
  int l, c;
  for (c=0; c<TAILLE; c++) printf("---");
  printf("\n");
  for (l=0; l<TAILLE; l++) {
    for (c=0; c<TAILLE; c++) {
      int a = agentSurXY(c, l, tx, ty);
      int b = agentSurXY(c, l, bx, by);
      if (a==-1) printf(".");
      else       printf("%d", a);
      if (b==-1) printf(".");
      else       printf("%d", b);
      printf(" ");
    }
    printf("\n");
  }
  for (c=0; c<TAILLE; c++) printf("---");
  printf("\n");
}
```

2) c) Ecrire une fonction `placerEntite` qui place une entité sur une case du damier tirée au hasard, en vérifiant que la case n'est pas déjà occupée par une autre entité. Une entité est soit un agent, soit un départ, soit un but. La signature de la fonction est:
`void placerEntite(int e, int *tx, int * ty);`

`e` est le numéro de l'entité à placer, `tx` et `ty` sont les tableaux de positions d'entités déjà placées sur le damier. La fonction `placerEntite` fera itérativement un tirage aléatoire d'une case `(x, y)` en utilisant la fonction `rand` tant que la case `(x, y)` est occupée. Pour tester si une case est occupée on utilisera la fonction `agentSurXY`. Après le traitement itératif, la fonction affecte l'entité à la case choisie.

```
void placerEntite(int e, int *tx, int * ty) {
  int x, y;
  do {
    x = rand() % TAILLE;
    y = rand() % TAILLE;
  } while (agentSurXY(x, y, tx, ty)!=-1);
  tx[e] = x; ty[e] = y;
}
```

Question 3 (2 points)

3) a) Ecrire une fonction `distanceManhattan` retournant la distance de Manhattan de deux cases. La signature de la fonction est:
`int distanceManhattan(int ax, int ay, int bx, int by);`
`(ax, ay)` et `(bx, by)` représentent les deux cases. La distance de Mahattan entre les deux cases est la somme des distances entre `ax` et `bx` d'une part et `ay` et `by` d'autre part.

```
int distanceManhattan(int ax, int ay, int bx, int by) {
  int dx = ax - bx; if (dx<0) dx = -dx;
  int dy = ay - by; if (dy<0) dy = -dy;
  return dx+dy;
}
```

3) b) Ecrire une fonction `calculHeuristique` retournant une estimation du nombre d'itérations restantes pour que tous les agents atteignent leur buts. La signature de la fonction est:
`int calculHeuristique(int * dx, int * dy, int * bx, int * by);`
`dx` et `dx` contiennent les positions des agents, `bx` et `by` contiennent les buts. La fonction `calculHeuristique` retourne le maximum des distances entre un agent et son but calculées pour chaque agent avec la fonction `distanceManhattan`.

```
int calculHeuristique(int * dx, int * dy, int * bx, int * by)
{
  int max = -1, i;
  for (i=0; i<N_AGENTS; i++) {
    int d = distanceManhattan(dx[i], dy[i], bx[i], by[i]);
    if (max<d) max=d;
  }
  return max;
}
```

Question 4 (4 points)

On suppose déjà programmée une fonction `souhaitAgent` dont la signature est:
`void souhaitAgent(int ax, int ay, int bx, int by, int * sx, int * sy);`
La fonction `souhaitAgent` donne la position `(*sx, *sy)` souhaitée par agent de position `(ax, ay)` et de but `(bx, by)`. Son contenu est hors examen, mais elle est appelée par la fonction `souhaitsAgents`.

4) a) Ecrire une fonction `souhaitsAgents` dont la signature est:
`void souhaitsAgents(int * ax, int * ay, int * bx, int * by, int * sx, int * sy);`
`ax` et `ay` pointent vers les positions des agents, `bx` et `by` vers les buts des agents. La fonction `souhaitsAgents` appelle `souhaitAgent` pour tous les agents. A la fin, les tableaux pointés par `sx` et `sy` contiennent les souhaits des agents. (NB: Puisque les agents effectuent leurs souhaits individuellement, après l'exécution de `souhaitsAgents`, plusieurs agents peuvent avoir des souhaits identiques).

```
void souhaitsAgents(
int * ax, int * ay, int * bx, int * by, int * sx, int * sy)
{
  int a;
  for (a=0; a<N_AGENTS; a++)
    souhaitAgent(ax[a], ay[a], bx[a], by[a], sx+a, sy+a);
}
```

4) b) Ecrire une fonction `souhaitsCompatibles` dont la signature est:
`int souhaitsCompatibles(int * sx, int * sy);`
`sx` et `sy` pointent vers les tableaux des cases souhaitées. La fonction retourne `-1` si deux souhaits sont identiques et `0` si tous les souhaits sont différents. On utilisera deux boucles `for` imbriquées.

```
int souhaitsCompatibles(int * sx, int * sy)
{
  int a, b;
  for (a=0; a<N_AGENTS; a++)
    for (b=a+1; b<N_AGENTS; b++)
      if ((sx[a]==sx[b]) && (sy[a]==sy[b])) return -1;
  return 0;
}
```

4) c) Ecrire une fonction `executeUnPasDeTemps` dont la signature est:
`void executeUnPasDeTemps(int * ax, int * ay, int * bx, int * by);`
`ax` et `ay` contiennent les positions des agents, `bx` et `by` les buts des agents. La fonction déclare deux tableaux `sx` et `sy` destinés à contenir les cases souhaitées des agents. De manière itérative, la fonction demande les souhaits des agents en appelant `souhaitsAgents`, puis elle teste si les souhaits sont compatibles en appelant `souhaitsCompatibles`. (NB: La fonction `souhaitAgent` contient du hasard, c'est pourquoi on peut appeler `souhaitsAgents` plusieurs fois en obtenant des résultats différents). Lorsque tous les souhaits sont compatibles, la fonction

`executeUnPasDeTemps` affecte les agents aux cases souhaitées.

```
void executeUnPasDeTemps(int * ax, int * ay, int * bx, int * by)
{
  int sx[N_AGENTS], sy[N_AGENTS], sc;
  do {
    int a;
    for (a=0; a<N_AGENTS; a++) {
      sx[a] = sy[a] = -1;
    }
    souhaitsAgents(ax, ay, bx, by, sx, sy);
    sc = souhaitsCompatibles(sx, sy);
  } while (sc==-1);
  int a;
  for (a=0; a<N_AGENTS; a++) {
    ax[a] = sx[a]; sx[a] = -1;
    ay[a] = sy[a]; sy[a] = -1;
  }
}
```

Question 5 (4 points)

5) a) Ecrire une fonction `initEntites` créant un problème avec des départs et des buts et dont la signature est:
`void initEntites(int g, int * dx, int * dy, int * bx, int * by);`
`g` est une graine pour appeler `srand`. `dx` et `dy` pointent vers les positions de départs des agents, `bx` et `by` vers les buts des agents. La fonction appelle 5 fois `placerEntite` pour placer aléatoirement les positions de départ et 5 fois `placerEntite` pour placer aléatoirement les buts. Elle est appelée par le `main` avant l'exécution d'un épisode.

```
void initEntites(int g, int * dx, int * dy, int * bx, int * by) {
  srand(g);
  printf("\nepisode avec graine %d: \n", g);
  int i;
  for (i=0; i<N_AGENTS; i++) {
     dx[i] = dy[i] = bx[i] = by[i] = -1;
  }
  for (i=0; i<N_AGENTS; i++) placerEntite(i, dx, dy);
  for (i=0; i<N_AGENTS; i++) placerEntite(i, bx, by);
  printf("positions et buts:\n"); afficher(dx, dy, bx, by);
}
```

5) b) Ecrire une fonction `initEpisode` dont la signature est:
`void initEpisode(int * dx, int * dy, int * px, int * py);`
Cette fonction initialise les tableaux de positions `px` et `py` avec les tableaux de départs `dx` et `dy`.

```
void initEpisode(int * dx, int * dy, int * px, int * py) {
  int i;
  for (i=0; i<N_AGENTS; i++) {
    px[i] = dx[i];
    py[i] = dy[i];
  }
}
```

5) c) Ecrire une fonction `executeEpisode` dont la signature est:
`int executeEpisode(int * pos_x, int * pos_y, int * but_x, int * but_y);`
Cette fonction appelle itérativement `executeUnPasDeTemps` et `calculHeuristique` jusqu'à ce que l'épisode soit terminé. Elle retourne le nombre de pas de temps de l'épisode.

```
int executeEpisode(
int * pos_x, int * pos_y, int * but_x, int * but_y) {
  printf("episode: debut\n");
  int fini, n=0;
  do {
    ++n; printf("iteration %d:\n", n);
    executeUnPasDeTemps(pos_x, pos_y, but_x, but_y);
    printf("positions et buts:\n");
    afficher(pos_x, pos_y, but_x, but_y);
    fini = calculHeuristique(pos_x, pos_y, but_x, but_y);
    printf("fini = %d\n", fini);
  } while (fini>0);
  printf("episode: fin.\n");
  return n;
}
```

Question 6 (3 points)

6) a) Ecrire le `main`. Il déclare les 6 tableaux de la question 1. Il initialise les départs et les buts avec `initEntites`. Il calcule une estimation de la durée de l'épisode avec `calculHeuristique`. Il appelle itérativement `initEpisode` et `executeEpisode` tant que la durée effective de l'épisode est strictement supérieure à la durée estimée ou qu'un nombre maximal d'épisodes n'est pas dépassé.

```
int main() {
  int dep_x[N_AGENTS], dep_y[N_AGENTS];
  int pos_x[N_AGENTS], pos_y[N_AGENTS];
  int but_x[N_AGENTS], but_y[N_AGENTS];
  char c, c2;
  int h, i, l;
  initEntites(n, dep_x, dep_y, but_x, but_y);
  h = calculHeuristique(dep_x, dep_y, but_x, but_y);
  printf("h = %d; \n", h);
  i = 0;
  do {
      initEpisode(dep_x, dep_y, pos_x, pos_y);
      l = executeEpisode(pos_x, pos_y, but_x, but_y);
      printf("%d ", l);
  } while ((++i<N_ITERATIONS) && (l>h));
   return 0;
}
```

6) b) Dessiner un graphe dont les noeuds sont les fonctions, les arcs sont les appels des fonctions et le noeud initial est le `main`.

Trop long de dessiner le graphe avec éditeur...
A faire...