Mesterséges intelligencia

gyakorlat

1. feladat

(állapottér reprezentáció)

Készítette:

Szathmáry László
II. PM.

Feladat: a feladatgyűjteményből egy általam tetszőlegesen kiválasztott feladat állapottér reprezentációjának az elkészítése:

10. Lóugrásban I.

Írjunk rekurzív programot, amely egy adott mezőről indulva a lóugrás szabályainak megfelelően bejár egy 5X5-ös sakktáblát, minden mezőt pontosan egyszer érintve!

I. Állapottér megadása

Az állapottér egy 5X5-ös tábla, melyet egy kétdimenziós tömbbel, azaz mátrixszal reprezentálunk. A mátrix egy-egy mezője a következő állapotok valamelyikét veheti fel egy időben: ures, foglalt, lo.

Ures: azon a mezőn még nem jártunk, üres.

Foglalt: azon a mezőn már jártunk, így oda már nem léphetünk, hiszen minden mezőt pontosan egyszer lehet csak érinteni.

Lo: az a pozíció, ahol a ló éppen tartózkodik, az aktuális pozíció.

```
#define  N 5
enum elem {ures, foglalt, lo};
elem tomb[N][N];
```

A C-ben a kétdimenziós tömböt egydimenziós tömbök tömbjeként kell kezelni.

II. Kezdőállapot megadása

A feladat szövege szerint egy adott mezőről kell indulni. Ezt kérjük be a felhasználótól. Itt arra kell ügyelni, hogy C-ben az indexelés 0-tól kezdődik.

```
void kezdoallapot()
{
  int x,y, i,j;
  printf("Feladat: egy %dX%d-os sakktabla bejarasa lougrasban.\n"
         "Adja meg a kezdopoziciot (a szamozas 0-tol kezdodik!):\n\n", N,N);
  printf("Sor    (0-%d): ", N-1); scanf("%d", &x);
  printf("Oszlop (0-%d): ", N-1); scanf("%d", &y);
  for (i=0; i<N; ++i)
     for (j=0; j<N; ++j)
         tomb[i][j]=ures;
  tomb[x][y]=lo;
}
```

Vagyis: kezdetben üres a tábla, ezért minden pozíciót **ures**-re állítunk. A felhasználótól bekérjük, hogy kezdetben hova helyezi a lovat, ezt a pozíciót **lo** állapotba állítjuk, s ezzel kész a kezdőállapot megadása.

III. Végállapot megadása

Akkor jutunk el a végállapotba, ha egyetlen üres mező sincs, vagyis valamennyit érintettük. Ez azt jelenti, hogy az egyik állapota **lo**, az összes többié pedig **foglalt**.

```
int vege_van()
{
  int i,j;
  for (i=0; i<N; ++i)
    for (j=0; j<N; ++j)
        if (tomb[i][j]==ures) return 0;
  return 1;
}
```

Mivel C-ben nincs logikai típus, ezért azt integrálissal helyettesítjük: 1-igaz, 0-hamis. Végig kell menni a táblán (kétdimenziós tömb), s ha valahol üres pozícióval találkozunk, akkor már felesleges is továbbmenni: nincs végállapotban, térjen vissza 0-val (azaz hamis értékkel).

A **return 1;** részhez már csak akkor jut el, ha a táblán nem talált **ures** pozíciót, vagyis végállapotban vagyunk.

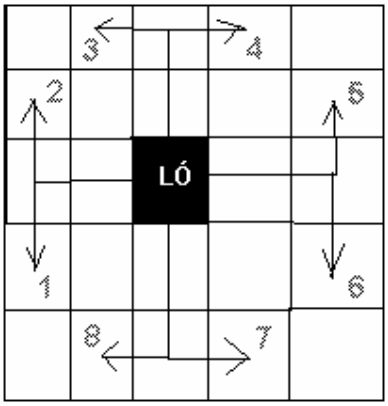
IV. Operátorok

Mivel egy lóval 8 irányba tudunk lépni, ezért 8 operátorra, s az ezekhez tartozó 8 darab előfeltételre lesz szükségünk. Mi a közös ezekben?

Előfeltétel: mindegyik kap egy x,y pozíciót, s meg kell vizsgálnia, hogy innen a megfelelő irányba lehet-e lépni. Mivel csak lóval lehet lépni a táblán üres pozícióra, ezért a kapott x,y pozícióban **lo**-nak kell lennie, s ahova lépünk vele, annak a pozíciónak nem szabad a táblán kívülre esnie, ill. üresnek kell lennie (**ures**).

Operátor: kap egy x,y pozíciót, mely azt jelenti, hogy onnan kell lépnie a megfelelő irányba. Ahova lép, aktuálisan ott lesz a ló, tehát annak az állapota **lo** lesz, míg ahonnan ellépett: **foglalt** lesz.

A következő ábra a 8 lehetséges lépést szemlélteti:



1.: balra le (2 balra, 1 le)
2.: balra fel (2 balra, 1 fel)
3.: fel balra (2 fel, 1 balra)
4.: fel jobbra (2 fel, 1 jobbra)
5.: jobbra fel (2 jobbra, 1 fel)
6.: jobbra le (2 jobbra, 1 le)
7.: le jobbra (2 le, 1 jobbra)
8.: le balra (2 le, 1 balra)

Általános megjegyzés:

A könnyebb áttekinthetőség kedvéért hívjuk a 0. indexszel jelölt sort (oszlopot) első sornak (oszlopnak), a 4. indexszel jelölt sort (oszlopot) pedig utolsó sornak (oszlopnak), stb.

Az előfeltételekhez:

Mivel a C rövidzár kiértékelést vall, ezért az ÉS-sel (&&) összekötött feltételek esetén ha bármelyik feltétel hamis, akkor hamis értékkel (0) fog a függvény visszatérni.

/* 1. */

```
int balra_le_ef(int x, int y)
{
  if (tomb[x][y]==lo && x<=N-2 && y>=2 && tomb[x+1][y-2]==ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e balra le lépni (azaz x<= mint az utolsó előtti sor ÉS y>= mint a harmadik oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void balra_le(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x+1][y-2]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 2. */

```
int balra_fel_ef(int x, int y)
{
  if (tomb[x][y]==lo && x>=1 && y>=2 && tomb[x-1][y-2]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e balra fel lépni (azaz x>= mint a második sor ÉS y>= mint a harmadik oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void balra_fel(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x-1][y-2]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 3. */

```
int fel_balra_ef(int x, int y)
{
  if (tomb[x][y]==lo && x>=2 && y>=1 && tomb[x-2][y-1]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e fel balra lépni (azaz x>= mint a harmadik sor ÉS y>= mint a második oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void fel_balra(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x-2][y-1]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 4. */

```
int fel_jobbra_ef(int x, int y)
{
  if (tomb[x][y]==lo && x>=2 && y<=N-2 && tomb[x-2][y+1]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e fel jobbra lépni (azaz x>= mint a harmadik sor ÉS y<= mint az utolsó előtti oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void fel_jobbra(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x-2][y+1]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 5. */

```
int jobbra_fel_ef(int x, int y)
{
  if (tomb[x][y]==lo && x>=1 && y<=N-3 && tomb[x-1][y+2]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e jobbra fel lépni (azaz x>= mint a második sor ÉS y<= mint a harmadik oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void jobbra_fel(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x-1][y+2]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 6. */

```
int jobbra_le_ef(int x, int y)
{
  if (tomb[x][y]==lo && x<=N-2 && y<=N-3 && tomb[x+1][y+2]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e jobbra le lépni (azaz x<= mint az utolsó előtti sor ÉS y<= mint a harmadik oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void jobbra_le(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x+1][y+2]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 7. */

```
int le_jobbra_ef(int x, int y)
{
  if (tomb[x][y]==lo && x<=N-3 && y<=N-2 && tomb[x+2][y+1]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e le jobbra lépni (azaz x<= mint a harmadik sor ÉS y<= mint az utolsó előtti oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void le_jobbra(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x+2][y+1]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.

/* 8. */

```
int le_balra_ef(int x, int y)
{
  if (tomb[x][y]==lo && x<=N-3 && y>=1 && tomb[x+2][y-1]== ures) return 1;
  else return 0;
}
```

Megvizsgálandó, hogy egyáltalán lóval akarunk-e lépni. Ha igen, akkor onnan lehet-e le balra lépni (azaz x<= mint a harmadik sor ÉS y>= mint a második oszlop), vagyis a célpozíció a táblán belül van-e, s ha ez is teljesül, akkor a célpozíciónak üresnek kell lennie.

```
void le_balra(int x, int y)
{
  tomb[x][y]=foglalt;
  tomb[x+2][y-1]=lo;
}
```

Ahova lépünk, annak a pozíciónak az állapota **lo** lesz, míg ahonnan léptünk: **foglalt** lesz.