UNIVERSITATEA POLITEHNICA BUCURESTI
FACULTATEA AUTOMATICA SI CALCULATOARE

SOKOBAN

- proiect Analiza Algoritmilor -

Onisor Teodor
Cojocaru Marian
CA331

Ce este Sokoban?

Sokoban este un joc clasic inventat in Japonia. Versiunea originala a jocului a fost scrisa de Hiroyuki Imabayashi, castigand cu acesta si un concurs de programare in 1980. Termenul Sokoban inseamna “supraveghetorul depozitului” (in engleza “warehouse keeper”). Sokoban este unul din cele mai indragite jocuri de logica, nivele speciale fiind si astazi create.

Scopul jocului Sokoban este de a duce cutii pe zone tinta prin impingerea lor. Utilizatorul controleaza miscare unui Sokoban. El se poate deplasa sus, jos, stanga dreapta si nu poate trece prin ziduri. Poate sa impinge doar o cutie in acelasi timp (nu o poate trage). Pentru a rezolva jocul trebuie ca toate cutiile sa fie duse pe zone tinta. Pare simplu dar si in labirinte mici unele probleme foarte dificile pot aparea. Gasirea solutie de om poate dura ore sau zile.

Sokoban are unele proprietati importante. De exemplu, impingerea cutiilor catre zonele tinta nu inseamna ca vom avea o solutie. In unele cazuri, miscari neintuitive trebuiesc facute pt a ajunge la o solutie. Ordinea de impingere a cutiilor este mereu importanta. Uneori inlaturarea unei cutii dintr-o zona tinta este necesara si alteori, desi la inceput impingerea blocurilor pe zonele tinta este destul de simpla, ultimul bloc sau chiar Sokoban ar putea sa nu aiba loc pentru a finaliza jocul. Solutia este, deci, nebanala.

Descrierea jocului

Sokoban a fost implementat in Java sub forma unui applet.
Semnificatia casutelor este urmatoarea:

 Sokoban;

 Cutie aflata pe o zona libera;

 Cutie aflata pe o zona tinta;

 Zona libera;

 Zona tinta;

 Zid.

Nivelul poate fi rezolvat de catre utilizator prin folosirea tastelor directionale sus, jos, dreapta, stanga sau poate fi rezolvat de catre calculator prin apasarea tastei INSERT.

Algoritmul de rezolvare

Sokoban are cateva reguli de gradul I simple:

- Sokoban se poate deplasa sus, jos, dreapta, stanga;
- Sokoban nu poate trece prin ziduri;
- Daca Sokoban vrea sa se miste inspre o cutie incearca s-o impinga;
- Daca o cutie are un zid sau o alta cutie in directia de impingere, Sokoban nu o poate misca;
- Fiecare cutie ar trebui impinsa pe zone tinta;

Aceste reguli pot fi satisfacute usor. Dar avand in vedere aceste reguli, cateva proprietati fac ca gasirea solutiei sa fie foarte dificila. Cateva din aceste proprietati sunt:

- Algoritmul de aflare a solutiei este necunoscut (asemanator sahului)
- Ordinea de impingere a cutiilor si directia de impingere sunt necunoscute.
- Nu se cunoaste adancimea solutiei.

Se poate demonstra ca Sokoban poate fi folosit sa emuleze un automat marginit liniar (o Masina Turing cu banda finita). In particular, o constructie este data avand o solutie daca si numai daca Masina Turing corespunzatoare se opreste la input in starea de accept. Mai mult, daca Masina Turing se opreste si accepta, atunci avansatorul va face $\Theta(n + t(n))$ miscari si impingeri, unde n este numarul de simboluri pe banda de input si t(n) este numarul de tranzitii facut de Masina Turing in timpul de calcul. Aceasta constructie arata ca rezolvarea unui nivel Sokoban este P-completa.

Descrierea algoritmului folosit

Algoritmul de baza este un algoritm de cautare prin extindere a configuratiilor, asemanator algoritmului A*. A* tine o lista cu toate pasurile urmatoare posibile, apoi alege urmatorul pas care va conduce cel mai probabil la solutie in timpul minim. O data ce pasul a fost ales, se marcheaza pentru a nu permite reintoarcerea la el.

Cum algoritmul de baza este foarte incet si nu foarte destept, o serie de optimizari au fost necesare pentru a reduce timpul de executie. Aceste optimizari sunt:

- **tabele de dispersie**

  Hash tables (sau Transposition Tables) sunt o modalitate de a evita revizitarea configuratiilor care pot fi atinse pe diferite cai. Tabelele de dispersie inlatura si ciclurile care ar putea aparea la executie. Rezulta o micsorare a spatiului de memorie folosit, prin inlaturarea configuratiilor identice;

  Un obiect HashTable contine o serie de configuratii date de pozitia lui Sokoban si pozitia cutiilor. Au fost definite functiile de inserare, stergere, gasire bazate pe codul hash si pe metoda equals din clasa Config.
  Functia hash returneaza codul hash pentru o configuratie data. Formula utilizata este:

hash(s)=(hash1(s)+i*hash2(s))%dim,

unde i = numarul de conflicte gasite la inserare;

dim – dimensiunea tabelei (numar prim la mijlocul distantei intre doua puteri ale lui 2, dat de functia getDim);

hash1(s) = $\sum_{k=1}^{nr.cutii} i_k \cdot 2^{j_k}$ unde i_k, j_k sunt coordonatele cutiei k

hash2(s) = $\sum_{k=1}^{nr.cutii} j_k \cdot 2^{i_k}$ unde i_k, j_k sunt coordonatele cutiei k

- eliminarea miscarilor inutile ale lui Sokoban

In algoritmul de baza sunt luate in considerare toate configuratiile care modifica pozitia lui Sokoban, fara a modifica si pozitia cutiilor. Aceasta duce la un numar extrem de mare de configuratii redundante. Pentru rezolvarea acestei probleme s-au introdus in lista doar configuratiile in care Sokoban poate sa impinga o cutie, toate celelate configuratii fiind echivalente cu acestea. Pentru aceasta Sokoban a fost plasat intr-o pozitie alaturata cutiei si in urmatorul pas Sokoban impinge cutia in directia respectiva. Gasirea cutiilor/pozitiilor accesibile este realizata de functia fillhh() care realizeaza o parcurgere in latime a grafului asociat labirintului din pozitia curenta a lui Sokoban si care umple matricea hh cu costurile deplasarii. Deplasarea lui Sokoban se face apoi cu functia mutaSoko().

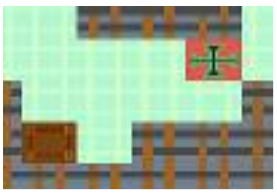
Complexitatea fillhh este data in cazul cel mai defavorabil de dimensiunea hartii.

- eliminarea zonelor inaccesibile

Se elimina toate cutiile din harta si apoi se apeleaza fillhh pentru a afla care zone sunt inaccesibile din pozitia actuala a lui Sokoban.

- eliminarea zonelor nefavorabile

In fiecare harta de Sokoban exista pozitii pe care daca s-ar muta cutii, acestea ar ramane blocate acolo sau nu ar putea fi duse in continuare pe o zona tinta. Exemple de astfel de zone:



In functia fillZoneDefavorizate() se umple matricea pozne[][] cu 1 pentru astfel de zone nefavorabile. Se porneste cu o multime vida de pozitii si apoi se extinde avand in vedere anumite conditii (de exemplu sa aiba numai doua iesiri pentru colturi, sau pentru un perete – sa nu aiba nici o iesire si langa el sa nu se afle nici-o zona tinta). In continuare in algoritmul de rezolvare se evita configuratiile in care o cutie se muta intr-o astfel de zona.

Pe langa aceste optimizari s-au mai implementat si alte functii cum ar fi:

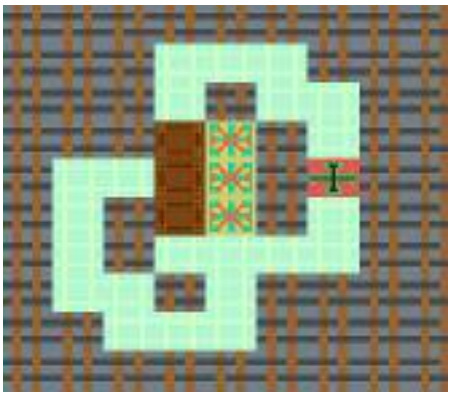
- **SokoSus, SokoJos, SokoDreapta, SokoStanga** – il muta pe Sokoban in directia respectiva (realizand si mutarea de cutii) si modifica imaginea buffer a appletului pentru a reflecta modificarea. Daca nu este posibil returneaza false.
- **SokoSusf, SokoJosf, SokoDreaptaf, SokoStangaf** realizeaza acelasi lucru fara updatarea imaginii.
- **exsol** reconstituie solutia afisand si pe ecran miscarea lui Sokoban.
- **solve** – implementeaza algoritmul de rezolvare.
- **scan** realizeaza importarea nivelelor scrise in format standard, retinand caracteristicile nivelelor in matricile h[][],ci[],cj[] etc.

Formatul standard reprezinta o secventa de caractere Ascii care reflecta dispunerea lui Sokoban, a cutiilor, a zonelor tinta sau a zidurilor. Semnificatia caracterelor este urmatoarea:

#	Zid
	Zona libera
@	Sokoban pe zona libera
$	Cutie pe zona libera
.	Zona tinta
+	Sokoban pe zona tinta
*	Cutie pe zona tinta

Un exemplu de nivel:

```
  #####
 ##   ##
### #  #
###$.# #
#  $.#@#
# #$.# #
# #    #
#  # ###
##   ##
 #####
```



Bibliografie

1. Giumale, C.A. – "*Introducere in analiza algoritmilor"* Ed. Polirom iasi 2004

2. Sokoban @ University of Alberta
   http://www.cs.ualberta.ca/%7Egames/Sokoban/

3. Culberson, J. – "*Sokoban is PSPACE complete"* Technical Report 1997

4. Sokoban Puzzle Solver *http://sps.gotdns.org/*