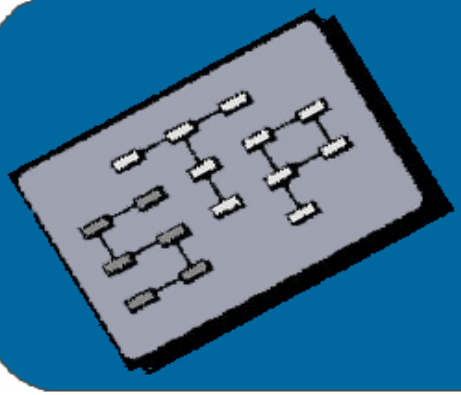
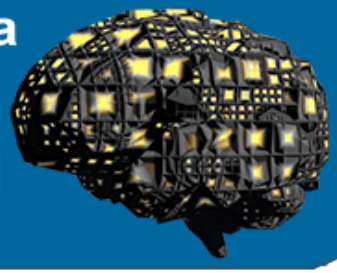


Datorzinātnes un informācijas tehnoloģijas fakultāte

Sistēmu teorijas un projektēšanas katedra

MĀKSLĪGĀ INTELEKTA PAMATI



2. Modulis "Neinformētas pārmeklēšanas stratēģijas stāvokļu telpā"

2.4. Tēma

Pārmeklēšanas rekursīva realizācija

Dr.habil.sc.ing., profesors **Jānis Grundspeņķis**, Dr.sc.ing., lektore **Alla Anohina**

Sistēmu teorijas un projektēšanas katedra
Datorzinātnes un informācijas tehnoloģijas fakultāte
Rīgas Tehniskā universitāte
E-pasts: *{janis.grundspenkis, alla.anohina}@rtu.lv*
Kontaktadrese: *Meža iela 1/4- {550, 545}, Rīga, Latvija, LV-1048*
Tālrunis: *(+371) 67089{581, 595}*

Tēmas mērķi un uzdevumi

Tēmas mērķis ir sniegt zināšanas par pārmeklēšanas rekursīvu realizāciju.

Pēc šīs tēmas apgūšanas Jūs:

- zināsiet, kāpēc rekursija ir piemērota pārmeklēšanas realizācijai;
- zināsiet, kas ir rekursīva procedūra;
- zināsiet, kā tiek izmantots saraksts OPEN pārmeklēšanas rekursīvā realizācijā.

Rekursijas izmantošanas pamatojums (1)

Rekursija ir īpaši piemērota stāvokļu telpas pārmeklēšanas algoritmu realizācijai, jo rekursija ir dabīga valodas konstrukcija datu struktūrām, kurām nav iepriekš noteikts apjoms. Pie šādām datu struktūrām pieder saraksti, grafi un koki.

Pārmeklēšanas dziļumā un plašumā realizācijas algoritmos tiek izmantoti divi saraksti: OPEN un CLOSED. Iepriekš nevar paredzēt, cik virsotnes saturēs katrs no sarakstiem, jo tas ir atkarīgs gan no pārmeklēšanas sākumstāvokļa, gan no stāvokļu telpas struktūras. Turklāt, pati stāvokļu telpa ir grafs.

Rekursijas izmantošanas pamatojums (2)

Bez tam, pārmeklēšana stāvokļu telpā pēc savas dabas ir rekursīvs process. Lai atrastu ceļu no tekošā stāvokļa uz mērķi, ir jāvirzās uz šī stāvokļa pēcteci, tālāk no pēcteča uz pēcteča pēcteci u.t.t. Ja atbilstošais pēctecis neved uz mērķi, tad ir jāapskata stāvoklis tajā pašā līmenī, kur atrodas tā priekštecis. Tādējādi, rekursija ļauj sadalīt lielu un sarežģītu problēmu (pārmeklēšanu visā stāvokļu telpā) mazākās daļās (atsevišķa stāvokļa pēcteču ģenerēšanu), un tad pielietot šo stratēģiju (rekursīvi) katram pēctecim.

Rekursīva pārmeklēšana (1)

Rekursīvas pārmeklēšanas pamatā ir ***rekursīva procedūra***, kas sastāv no:

1. ***Rekursīva soļa***, kurā procedūra izsauc pati sevi, lai atkārtotu darbību secību
2. ***Beigu nosacījuma***, kas paredzēts procedūras darbības apturēšanai, novēršot bezgalīgu procedūras izpildes ciklu

Rekursīva pārmeklēšana (2)

Iepriekš apskatītie algoritmi bija iteratīvi jeb balstījās uz cikla *while* izmantošanu. Taču, viss, kas var tikt izdarīts iteratīvi, var tikt izdarīts rekursīvi.

Tieša pāreja no cikla uz rekursijas izmantošanu

```
function pārmeklēšana_dziļumā;
begin
//beigu nosacījums
  if saraksts OPEN ir tukšs then return (neveiksme);
  tekošais_stāvoklis:= pirmais stāvoklis sarakstā OPEN;
//vēl viens beigu nosacījums
  if tekošais_stāvoklis = mērķa_stāvoklis then return (veiksme)
  else
    begin
       OPEN := OPEN atlikuša daļa;
       CLOSED := CLOSED + tekošais_stāvoklis;
       Katru tekošā stāvokļa pēcteci, kurš nav sarakstā OPEN vai CLOSED, ievieto saraksta OPEN  priekšā;
    end;
//rekursīvais solis
  function pārmeklēšana_dziļumā;
end;
```

OPEN un CLOSED ir globālie mainīgie

Rekursīva pārmeklēšana (3)

Iepriekš dotais algoritms rekursijas iespējas neizmanto pilnā mērā, jo tajā tiek lietots saraksts OPEN. Taču šo sarakstu var izslēgt.

Rekursīva realizācija, neizmantojot sarakstu OPEN

```
function pārmeklēšana_dziļumā (tekošais_stāvoklis);
begin
 if tekošais_stāvoklis = mērķa_stāvoklis then return (veiksme);
 pievienot tekošais_stāvoklis sarakstam CLOSED;
 while tekošajam_stāvoklim ir neapskatīti pēcteči do
  begin
   nākošais_stāvoklis:=nākošais neapskatītais pēctecis;
   if nākošais_stāvoklis neatrodas saraksta CLOSED then
     if  pārmeklēšana_dziļumā (nākošais_stāvoklis)=veiksme then
       return(veiksme);
  end;
 return(neveiksme);
end;
```

Rekursīva pārmeklēšana (4)

Algoritmā tiek izmantots globālais mainīgais, t.i. saraksts CLOSED. Taču, tekošā stāvokļa visu pēcteču atrašanas un ievietošanas sarakstā OPEN vietā algoritms apskata katru pēcteci atsevišķi. Katram no tiem rekursīvi tiek atrasti visi pēcteči. Ja kāds no pēctečiem ir mērķa stāvoklis, procedūra beidz savu darbību. Taču, ja kāds no pēctečiem noved strupceļa stāvoklī, tad tiek apskatīts šī pēcteča brālis (brāļi ir vienas un tās pašas virsotnes pēcteči), un visi tā pēcteči.