1. Dirbtinio intelekto sistema kaip produkcijų sistema, formalizmas, algoritmas PRODUCTION, pavyzdžiai. Dirbtinio intelekto samprata (remiantis skaityta literatūra).

Intelektualiais yra laikomi šie gebėjimai:
1) suvokti (perceive);
2) suprasti (understand);
3) prognozuoti (predict);
4) manipuliuoti – atlikti veiksmą (manipulate).
Pvz -žirgo kelionė.

2. Dirbtinio intelekto sistema kaip produkcijų sistema

Šiame skyriuje vadovaujamės (Nilsson 1982) 1.1 poskyriu.

Daugumoje dirbtinio intelekto sistemų galima išskirti tris esminius komponentus: duomenis, operacijas ir jų valdymą. Kitaip tariant, tokiose sistemose galima išskirti tam tikrą duomenų struktūrą, kurią vadinsime *globalia duomenų baze*, bei su ja atliekamus iš anksto veiksmus apibrėžtus veiksmus. Šie veiksmai paprastai valdomi remiantis tam tikra globalia valdymo strategija. Bendru atveju, tokios sistemos yra vadinamos produkcijų sistemomis. Taigi, produkcijų sistema vadinsime trejetą:

1. globali duomenų bazė (sutrumpintai – duomenų bazė arba GDB);
2. produkcijų aibė $\{\pi_1, \pi_2, ..., \pi_m\}$;
3. valdymo sistema.

Šis trejetas yra modelis. Juo remiantis dalykinės srities žinios gali būti klasifikuojamos pagal tai, kur yra vaizduojamos. *Deklaratyviosiomis žiniomis* vadinamos žinios, vaizduojamos globalioje duomenų bazėje. *Procedūrinėmis žiniomis* vadinamos žinios, vaizduojamos produkcijų aibėje. *Valdymo žiniomis* vadinamos žinios, vaizduojamos valdymo strategijoje.

Globali duomenų bazė – tai duomenų struktūra, kurią naudoja dirbtinio intelekto produkcijų sistema. Priklausomai nuo dalykinės srities, ji gali būti tiek įprasta sveikųjų skaičių matrica, tiek sudėtinga duomenų bazių valdymo sistema. Produkcija iš produkcijų aibės yra taikoma GDB ir po pritaikymo GDB yra pervedama iš vienos būsenos į kitą. Produkcija gali būti taikoma tik tada, jeigu GDB būsena tenkina tos produkcijos pradinę sąlygą. Kurią būtent produkciją iš produkcijų aibės parinkti ir taikyti, sprendžia valdymo sistema. Po kiekvienos produkcijos taikymo, valdymo sistema patikrina, ar GDB būsena yra terminalinė. Jeigu GDB tenkina terminalinės būsenos sąlygą, tai algoritmas baigia darbą. Jeigu GDB nėra terminalinė, o valdymo sistema nebegali taikyti nei vienos produkcijos, tai laikoma, kad uždavinio sprendinys neegzistuoja. Produkcijų sistemos algoritmas:

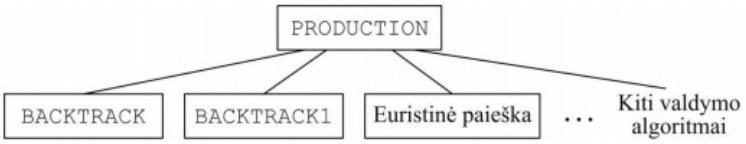
```
procedure PRODUCTION
{1} DATA := pradinė duomenų bazė;
{2} until DATA patenkina terminalinę sąlygą do
{3} begin
{4}     select  išrinkti produkciją π iš aibės
                tų produkcijų, kurias galima taikyti duomenų
                bazei DATA
{5}     DATA := π(DATA) {Rezultatas, gautas pritaikius π
                duomenų bazei DATA. Produkcija atvaizduoja
                (t. y. perveda) duomenų bazę iš vienos
                būsenos į kitą}
{6} end
```

2. Valdymo su grįžimais procedūros BACKTRACK ir BACKTRACK1. Valdymo su grįžimais esmė, užsiciklinimo galimybė. Pavyzdžiai. Euristikos samprata.

3. Valdymas su grįžimais ir procedūra BACKTRACK

Terminas „valdymas su grįžimais“ žymi vieną iš keleto valdymo sistemos algoritmų, tiksliau, vieną algoritmų šeimą. Ši šeima, aprašoma žemiau pateikiama procedūra BACKTRACK, ir jos modifikacijomis. Nilsonas ir kiti autoriai dar vartoja žodį strategija – valdymo strategija. Pastarasis terminas yra mažiau formalus negu algoritmas. Valdymo su grįžimais požymis yra suprantamas kaip DI principas.

Procedūra BACKTRACK yra atskiras PRODUCTION atvejis. Kitų algoritmų pavyzdžiai yra BACKTRACK1 bei euristinė paieška (3.1 pav.).



3.1 pav. Procedūra BACKTRACK valdymo algoritmų kontekste

```
recursive procedure BACKTRACK(DATA) {return list of rules}
{DATA - einamoji globalios duomenų bazės būsena}
    { 1}      if TERM(DATA) then return NIL;
    { 2}      if DEADEND(DATA) then return FAIL;
    { 3}      RULES := APPRULES(DATA);
    { 4}      LOOP:    if NULL(RULES) then return FAIL;
    { 5}               R     := FIRST(RULES);
    { 6}               RULES := TAIL(RULES);
    { 7}               RDATA := R(DATA);
    { 8}               PATH  := BACKTRACK(RDATA);
    { 9}               if PATH = FAIL then goto LOOP;
    {10}      return CONS(R, PATH);
end
```

Procedūra BACKTRACK turi esminių trūkumų. Pirmasis – BACKTRACK gali užsiciklinti, t. y. nesustoti. Priežastis – BACKTRACK nesaugo nesėkmingų būsenų.

Antrasis trūkumas – paieška gali nueiti pernelyg gylyn. Tada BACKTRACK sukurs tiek daug duomenų bazės būsenų, kad neužteks kompiuterio atminties arba bus viršytas tam tikras laiko limitas. Todėl yra tikslinga riboti rekursijos gylį.

Užsiciklinimas demonstruojamas kelio paieška labirinte kitame skyriuje 7.3 pav.

Siekiant išvengti minėtų trūkumų, procedūra BACKTRACK yra modifikuojama, įvedant aplankytų būsenų sąrašą bei rekursijos gylį, ir pavadinama BACKTRACK1. Ji žemiau pateikiama iš (Nilsson 1985) 2.1 poskyrio:

```
procedure BACKTRACK1(DATALIST) returns PRODLIST;
{DATALIST - globalios duomenų bazės būsenų sąrašas}
{PRODLIST - procedūros rezultatas: produkcijų sąrašas}
    { 1}    DATA := FIRST(DATALIST);
    { 2}    if MEMBER(DATA, TAIL(DATALIST)) then
               return FAIL;
    { 3}    if TERM(DATA) then return NIL;
    { 4}    if DEADEND(DATA) then return FAIL;
    { 5}    if LENGTH(DATALIST) > BOUND then return FAIL;
    { 6}    RULES := APPRULES(DATA);
    { 7}    LOOP:    if NULL(RULES) then return FAIL;
    { 8}    R := FIRST(RULES);
    { 9}    RULES := TAIL(RULES);
    {10}    RDATA := R(DATA);
    {11}    RDATALIST := CONS(RDATA, DATALIST);
    {12}    PATH := BACKTRACK1(RDATALIST);
    {13}    if PATH = FAIL then goto LOOP;
    {14}    return CONS(R, PATH);
end procedure;
```

Euristika – tai praktikos padiktuota pasirinkimo taisyklė (*rule of thumb*), skirta rasti sprendinį be varginančios paieškos (paprastai tai perrinkimas). Būdvardis **euristinis** (*heuristic*) apibūdina uždavinio sprendimo būdą, kuris pagerina sprendimo vidutinę charakteristiką, bet nebūtinai blogiausio atvejo charakteristiką (Russell, Norvig 2003, p. 94). Euristikos terminas kildinamas iš graikiško žodžio „heuriskein“ (atrasti). Legenda byloja, kad senovės graikų filosofas Archimedas sušuko „eureka“ (atradau), kai suvokė apie jėgą, kuri išstumia kietą kūną, panardintą į skystį. Dabar ši jėga vadinama *Archimedo jėga*.

Anglų kalboje „heuristic“ vartojamas kaip būdvardis ir daiktavardis. Žodžio „euristika“ reikšmė gali būti aiškinima remiantis anglų kalbos žodynu *Oxford English Dictionary* (žr. http://www.oed.com/):

euristinis (*heuristic*)

a. Skirtas išsiaiškinti arba atrasti (Serving to find out or discover) …

c. *kompiuteriai* … Su „euristinio“ programavimo procedūra kompiuteris atlieka paiešką tarp galimų sprendinių kiekvienoje programos stadijoje, įvertina šiai stadijai „gerą“ sprendimą ir pereina prie kitos stadijos. Iš esmės euristinis programavimas yra panašus į uždavinių sprendimą bandymų ir klaidų būdais, kuriais mes naudojamės kasdienybėje (T. W. McRae 1964).

euristika (*heuristic*)

b. Euristinis procesas arba metodas mėginant uždavinio sprendinį; taisyklė arba informacijos vienetas naudojamas tokiame procese (A. Newell et al. 1957). … Procesas, kuriuo galimas, bet negarantuojamas uždavinio išsprendimas. Trumpai sakant, „euristika“ yra „euristinis procesas“ sinonimas.

3. Uždavinys apie labirintą. Paieškos valdymo su grįžimais ir bangos būdais algoritmai ir programos. Apibūdinti depth-first search ir breath-first search iš esmės.

4. Prefiksinė ir postfiksinė medžio apėjimo tvarka. Binariniai medžiai, bendro pavidalo medžiai. Parašykite procedūras darbui su bendro pavidalo medžiais: a) medžio įvedimui, b) apėjimui prefiksine tvarka ir c) apėjimui postfiksine tvarka.

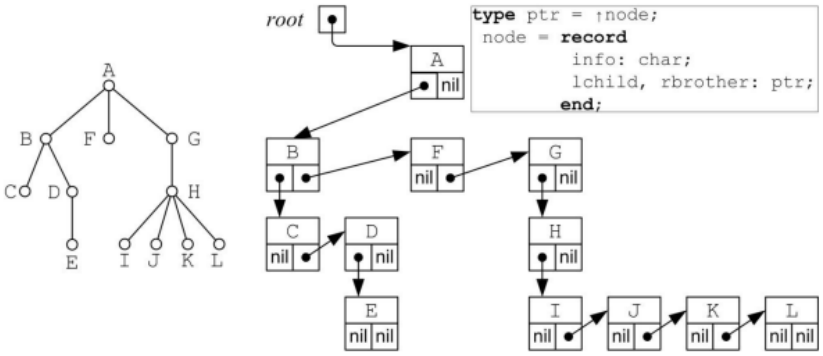
```
program traversal(input, output);
type ptr = ↑node;
     node = record
              info : char;
              llink, rlink : ptr;
            end;
var root : ptr;
    ch :   char;

procedure enter(var p:ptr);
begin { enter }
   read(ch);
   write(ch);
   if(ch ≠ '.') then
   begin
      new(p);
      p↑.info:=ch;
      enter(p↑.llink);
      enter(p↑.rlink);
   end;
   else p := nil;
end; { enter }

procedure preorder(p:ptr);
begin
   if p ≠ nil then
   begin
      write(p↑.info);
      preorder(p↑.llink);
      preorder(p↑.rlink);
   end;
end; { preorder }

procedure inorder(p:ptr);
begin
   if p ≠ nil then
   begin
      inorder(p↑.llink);
      write(p↑.info);
      inorder(p↑.rlink);
   end;
end; { inorder }

procedure postorder(p:ptr);
begin
   if p ≠ nil then
   begin
      postorder(p↑.llink);
      postorder(p↑.rlink);
      write(p↑.info);
   end;
end; { postorder }
```



12.5 pav. Bendro pavidalo medis ABC.DE...F.GHI.J.K.L ir jo vaizdavimas kompiuterio atmintyje

5. Paieška į gylį ir į plotį medyje. Algoritmas paieškai į plotį grafe, kai grafo briaunos neturi kainos (trumpiausias kelias tarp dviejų viršūnių). Programa paieškai į plotį labirinte.

9. Paieška į plotį grafe be kainų

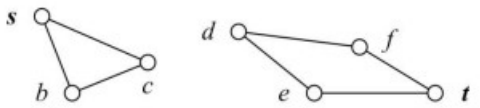
Pateiksime algoritmą rasti kelią grafe, kurio briaunos neturi kainos. Turime grafą, pradinę viršūnę ir vieną terminalinę viršūnę. Paieška į plotį randamas trumpiausias kelias – mažiausiai briaunų. Algoritmas pateikiamas pagal E. Hunt knygos (1978) 10.1.1 poskyrį.

Paieškos į plotį algoritmas grafe be kainų

ĮĖJIMAS: 1) grafas be kainų G, 2) pradinė viršūnė s, 3) terminalinė viršūnė t.
IŠĖJIMAS: trumpiausias kelias iš s į t.

Sąrašai ATIDARYTA ir UŽDARYTA yra tušti.

1. Patalpinti pradinę viršūnę s į sąrašą ATIDARYTA.
2. Jeigu ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesėkmės požymį ir baigti. Taip atsitinka, kai grafas *nesusijęs*, t. y. jame yra nepersikertantys viršūnių poaibiai tokie, kad pradinė viršūnė yra viename poaibyje, o terminalinė kitame, ir šių poaibių viršūnių nejungia jokia briauna (pavyzdys 9.1 pav.).
3. Uždaryti **pirmąją** viršūnę n iš sąrašo ATIDARYTA, t. y. perkelti iš ATIDARYTA į UŽDARYTA. Jeigu n yra terminalinė, tai kelias rastas. Surinkti kelią atgal ir baigti.
4. Paimti viršūnės n incidentinių viršūnių (gretimų) aibę $S(n)$. Į ATIDARYTA **pabaigą** patalpinti tas viršūnes iš $S(n)$, kurių nėra nei ATIDARYTA, nei UŽDARYTA. Formaliai ATIDARYTA := ATIDARYTA $\cup$ $S(n)$/(ATIDARYTA$\cup$UŽDARYTA)
5. Pereiti prie 2 žingsnio.



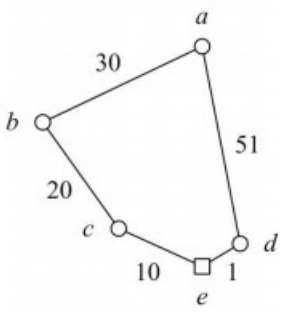
9.1 pav. Nesusijusio grafo pavyzdys. Kelias iš s į t neegzistuoja

6. Algoritmas paieškai grafe, kai grafo briaunos turi kainą

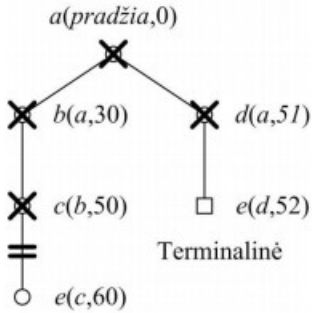
10. Paieška grafe su kainomis

Algoritmas pateikiamas vadovaujantis (Hunt 1978, 10.1.2 sk.). Tai variacija olandų informatiko Edsger Dijkstra 1956 m. sukurtam algoritmui, skirtam rasti trumpiausius kelius nuo vienos viršūnės iki kitų svoriniame grafe su neneigiamais svoriais; žr. http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm. Kelio kaina tarp dviejų viršūnių yra kraštinių kainų suma tame kelyje. Paieška iliustruojama grafu 10.1 pav. Paieškos medis parodytas 10.2 pav.

Kai kiekvienos briaunos kaina yra vienetas, tai uždavinys ekvivalentus rasti kelią su mažiausiai briaunų, kuris išnagrinėtas praeitame skyriuje.



10.1 pav. Grafas su kainomis. Pradinė viršūnė *a*, terminalinė – *e*



10.2 pav. Paieškos medis rasti kelią iš *a* į *e* grafe 10.1 pav.

ĮĖJIMAS: 1) grafas G, kurio briaunos turi neneigiamas kainas; 2) pradinė viršūnė s; 3) terminalinė viršūnė t.

IŠĖJIMAS: trumpiausias kelias nuo s iki t.

1. Patalpinti pradinę viršūnę s į sąrašą ATIDARYTA.
2. Jeigu sąrašas ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesėkmės požymį ir baigti. Tokia situacija, kai grafas nesusijęs.
3. Uždaryti tokią viršūnę n, kurios kelio nuo pradinės viršūnės kaina yra **mažiausia**: perkelti n iš ATIDARYTA į UŽDARYTA. Tokiu būdu sąrašas ATIDARYTA rūšiuojamas. Jeigu n terminalinė, tai kelias rastas. Surinkti kelią atgal ir baigti.
4. Paimti viršūnės n incidentinių viršūnių aibę $S(n)$. Kiekvienai viršūnei n^* iš $S(n)$, kurios nėra sąraše UŽDARYTA, apskaičiuoti naują kelio kainą. Formaliai, $\forall n^* \in S(n)$/UŽDARYTA priskirti KAINA(s,n^*) := KAINA(s,n) + BRIAUNA(n,n^*). Į ATIDARYTA įtraukti tas $S(n)$ viršūnes, kurių ATIDARYTA dar nėra. Formaliai, jeigu $n^* \notin$ ATIDARYTA, tai ATIDARYTA := ATIDARYTA$\cup n^*$. Jeigu $n^* \in S(n)$ jau priklauso ATIDARYTA, tai palyginti seno ir naujo kelio kainas. Jeigu naujas kelias yra geresnis, tai įtraukti n^* į ATIDARYTA su geresne kaina ir nauja nuoroda.
5. Pereiti prie 2 žingsnio.

7. Paieškos į gylį algoritmas grafe be kainų. Pavyzdys. Sprendėjas ir planuotojas.

11.2. Sprendėjas ir planuotojas

Iš pradžių priminsime paieškos į gylį ir į plotį principus, o paskui paaiškinsime *sprendėjo* ir *planuotojo* sąvokas. Paieška į gylį siejama su trumpalaikės atminties, t. y. steko principu – vėliau gimusi viršūnė yra uždaroma anksčiau. Paieška į plotį siejama su sąrašo principu, t. y. vėliau gimusi viršūnė yra ir uždaroma vėliau. Sąrašus ATIDARYTA ir UŽDARYTA galime palyginti su gyvaisiais ir mirusiaisiais. Gyvųjų eilė gali būti peržiūrima. O mirusiųjų jau niekas netrikdo.

Algoritmas į gylį arba į plotį pasirenkamas atsižvelgiant į skirtingus motyvus. Paieška į gylį paprastai naudojasi uždavinio *sprendėjas* (*problem-solver*). Sinonimas yra *sprendžiantis agentas* (*solving agent*). Įprasta kad sprendėjo uždavinio prigimtis yra spręsti jį vieną kartą. Pavyzdžiui, ištrūkti iš labirinto. Sprendėjas neturi prieš akis labirinto žemėlapio. Ištrūkti iš šalto labirinto yra jo gyvybės ar mirties klausimas.

Paieška į plotį paprastai naudojasi *planuotojas* (*planner*). Sinonimas yra *planuojantis agentas* (*planning agent*). Racionalu, kad planuotojas prieš akis turi žemėlapį. Pavyzdžiui, planuotojas sėdi patogiame krėsle šiltame kambaryje ir ieško trumpiausio maršruto kroviniui gabenti. Rastas kelias savo esme yra tam tikro uždavinio sprendimo planas. Vieną kartą rastas planas paprastai taikomas daug kartų. Todėl yra prasminga ieškoti trumpiausio plano. Paieška į plotį randa trumpiausią kelią, kuris, kaip minėjome, yra suprantamas kaip planas. Pavyzdžiui, grafe rastu trumpiausiu keliu vežėjui yra prasminga daug kartų gabenti krovinį iš pradinio miesto *s* į terminalinį *t*. Planavimas paprastai atima daugiau planuotojo resursų, negu sprendėjo. Bet planavimo algoritmas vykdomas tik vieną kartą. O vežėjas gabendamas trumpiausiu keliu sutaupo kiekvieną kartą.

Paiešką į gylį galima palyginti su pasivaikščiojimu neturint tikslo rasti trumpiausią kelią. O paieškos į plotį tikslas ne tik vaikščioti, bet ir rasti trumpiausią kelią.

Galima iškelti klausimą, kuris algoritmas geresnis? Toks klausimo kėlimas nekorektiškas, nes sprendėjas ir planuotojas paprastai naudojami esant skirtingoms aplinkybėms. Keliant palyginimo klausimą turi būti apibrėžtas kriterijus. Paieška į plotį naudojama ieškant trumpiausio kelio. Paieška į gylį siejama su intelekto sąvoka. Laikoma, kad žmogaus trumpalaikė atmintis veikia paieškos į gylį principu. Tokiu būdu vienoje situacijoje gali geriau tikti vienas algoritmas, pavyzdžiui, kai iš anksto nežinomas dalykinės srities būsenų grafas, kitoje – kitas, kai žinomas.

Įprasta, kad sprendėjas ir planuotojas naudojami skirtingose situacijose:

	Sprendėjas	Planuotojas
1	Neturi žemėlapio	Turi žemėlapį
2	Randa nebūtinai trumpiausią kelią	Gali rasti trumpiausią kelią
3	Paieška į gylį	Paprastai paieška į plotį
4	Keliu pasinaudoja vieną kartą	Kelias naudojamas daug kartų

8. Bendras paieškos grafe algoritmas GRAPHSEARCH. Neinformuotos procedūros, euristinė paieška. BACKTRACK1 ir GRAPHSEARCH skirtumai; kontrpavyzdys.

13. Bendras paieškos grafe algoritmas GRAPHSEARCH

Ankstesnėse temose išnagrinėjome algoritmus „į gylį“ ir „į plotį“. Apjunkime šias paieškas į vieną procedūrą. Ši procedūra sukuria išreikštinį paieškos medį grafe. Pastarasis grafas gali būti tiek išreikštinis, tiek neišreikštinis. Neišreikštinio grafo pavyzdžiais gali būti žaidimų, tokių kaip 8 kauliukai, kryžiukai-nuliukai, šaškės, šachmatai ir kt. būsenų medžiai.

ĮĖJIMAS: 1) grafas; 2) pradinė viršūnė *s*; 3) terminalinė viršūnė (jų gali būti keletas).
IŠĖJIMAS: kelias nuo pradinės viršūnės *s* iki terminalinės (nebūtinai trumpiausias).

Sąrašai ATIDARYTA ir UŽDARYTA yra tušti.

1. Patalpinti pradinę viršūnę *s* į sąrašą ATIDARYTA ir sukurti paieškos medį *T*, susidedantį iš vienos viršūnės *s*.
2. Jeigu ATIDARYTA tuščias, tai kelias neegzistuoja. Grąžinti nesėkmės požymį ir baigti.
3. Uždaryti **pirmąją** viršūnę *n* iš ATIDARYTA: perkelti ją į UŽDARYTA.
4. Jeigu *n* yra terminalinė, tai sėkmė. Sekant nuorodomis atgal nuo *n* iki *s* surinkti kelią ir baigti.
5. Išskleisti *n*, t. y. paimti *n* incidentinių viršūnių aibę $S(n)$. Tas $n^* \in S(n)$, kurių nėra nei ATIDARYTA, nei UŽDARYTA, t. y. $n^* \in (S(n)/(\text{ATIDARYTA} \cup \text{UŽDARYTA}))$, patalpinti į ATIDARYTA ir į *T* formuojant nuorodas $n^*(n)$ atgal. Formaliai ATIDARYTA:= ATIDARYTA $\cup$ $S(n)/(\text{ATIDARYTA} \cup \text{UŽDARYTA})$
6. Kiekvienai viršūnei $n^* \in S(n)$, kuri priklausė ATIDARYTA (senajai būsenai, žr. dešiniąją priskyrimo pusę aukščiau), t. y. $n^* \in (S(n) \cap \text{ATIDARYTA})$, nuspręsti, ar perorientuoti nuorodą. Nuspręsti būtina nes yra senas kelias ir naujas kelias. Likusių n^*, t. y. $n^* \in (S(n) \cap \text{UŽDARYTA})$, nenagrinėti.
7. Sutvarkyti ATIDARYTA pagal kokią nors schemą arba euristiką. Pavyzdžiui,
   a) pagal kainą;
   b) **„į gylį“,** t. y. $S(n)$/UŽDARYTA patalpinti į ATIDARYTA **pradžią**, o besidubliuojančias viršūnes iš ATIDARYTA pabaigos pašalinti, kad nesikartotų. Taip sutvarkoma **steko** principu, t. y. LIFO (Last In First Out).
   c) **„į plotį“,** t. y. $S(n)$/UŽDARYTA patalpinti į naujojo ATIDARYTA **pabaigą**, o besidubliuojančias viršūnes iš ATIDARYTA **pradžios** pašalinti. Taip sutvarkoma **eilės** principu, t. y. FIFO (First In First Out).
8. Pereiti prie 2 žingsnio.

GRAPHSEARCH algoritmas pateiktas (Nilsson 1982, 2.2.2 sk.)

14. Skirtumai tarp BACKTRACK1 ir GRAPHSEARCH

Pavyzdžiais pademonstruosime skirtumą tarp algoritmų BACKTRACK1 ir GRAPHSEARCH_Į_GYLĮ. Kaip minėta anksčiau, BACKTRACK1 įvardinamas sinonimu BACKTRACK_SU_STEKU. Čia steko vaidmenį atlieka procedūros parametras DATALIST. Jo tipas yra sąrašas, bet iš rekursijos yra grįžtama į ankstesnę sąrašo būseną. Tokiu būdu DATALIST „kvėpuoja“ kaip stekas. Algoritmų skirtumas apibūdinamas dviem teiginiais.

1. Procedūra BACKTRACK1 grindžiama sąvokomis a) produkcijos ir b) globalios duomenų bazės būsenos, o GRAPHSEARCH_Į_GYLĮ – a) atidarytos viršūnės ir b) uždarytos viršūnės.
2. Procedūra BACKTRACK1 aplink „salas“ eina *keletą* kartų, nes sąraše DATALIST kaip steke saugo tik einamąjį kelią. Procedūra GRAPHSEARCH_Į_GYLĮ aplink salą eina tik *vieną* kartą; už laiką ji moka atmintimi.

Skirtingų kriterijų, pvz., laiko ir atminties – pusiausvyra yra informatikos principas.

9. Algoritmo A* samprata. Pavyzdys, kai euristinė funkcija yra Manheteno atstumas.

17. Algoritmas A*

Algoritmas A* (tariama A su žvaigždute) yra GRAPSHEARCH algoritmas, kai naudojama tokia viršūnės *n* įvertinimo funkcija:

$$f(n) = g(n)+h(n)$$

Čia *g(n)* yra trumpiausias kelias nuo pradinės viršūnės iki *n*, o *h(n)* euristinė funkcija įvertinti atstumui nuo *n* iki terminalinės viršūnės. Pavyzdžiui, žemėlapyje *h(n)* gali būti: 1) atstumas tiesia linija (t. y. oru) nuo miesto *n* iki galinio miesto arba 2) Manheteno atstumas. Agentas (ir sprendėjas, ir planuotojas) nemato viso žemėlapio ir vadovaujasi euristine funkcija *h(n)*.

Algoritmą A* N. Nilsonas su bendrautoriais publikavo 1968 m, o vėliau pateikė vadovėlyje (Nilsson 1982); žr. taip pat kituose DI vadovėliuose, pvz., (Russell, Norvig 2003; Luger 2005) ir Vikipedijoje (http://en.wikipedia.org/wiki/A*_search_algorithm). Algoritmas iš pradžių įvardijamas A, o vėliau dėl tam tikro optimalumo pavadinamas A*.

Dirbtiniame intelekte skiriamos dvi paieškos strategijos: informuota paieška ir neinformuota paieška. *Neinformuota paieška* – kai algoritmui nėra žinoma globali informacija apie sprendinį, t. y. nei žingsnių skaičius, nei kelio įvertinimas nuo einamosios būsenos iki terminalinės. Pavyzdžiai: paieška „į gylį“ bei „į plotį“ ir grafe be kainų ir su kainomis.

Informuota paieška – kai algoritmas remiasi euristiniu vertinimu atstumo nuo einamosios būsenos iki ternimalinės. Informuotos paieškos pavyzdžiai yra „kopimas į kalną“ ir A*.

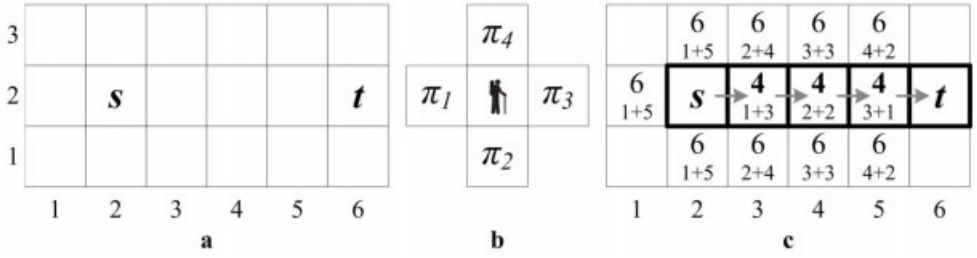
Informuota paieška yra efektyvesnė. Paieškos medyje yra mažiau viršūnių, bet efektyvumas pasiekiamas euristinio įvertinimo kaina.

17.1. A* pavyzdys su euristine funkcija h(n) Manheteno atstumu

Langelių lentoje agentas keliauja iš pradinio langelio *s* į terminalinį *t* (17.1 pav.). Lentoje gali būti kliūčių. Vertinimo funkcijoje $f(n)=g(n)+h(n)$ dėmuo *g(n)* yra nueito kelio nuo *s* iki *n* ilgis. Tegu euristinė funkcija *h(n)* yra Manheteno atstumas nuo *n* iki *t*. Tegu keturi agento ėjimai (produkcijos) $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ yra surūšiuoti kaip parodyta 17.1 b pav. Manheteno atstumas $d(n_1,n_2)$ tarp langelių $n_1=(x_1,y_1)$ ir $n_2=(x_2,y_2)$ yra apibrėžiamas šitaip:

$$d(n_1,n_2)=|x_2-x_1|+|y_2-y_1|$$

Manheteno atstumas yra trumpiausias atstumas, kai keliaujama tik vertikaliai ir horizontaliai – keturiomis produkcijomis $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ – t. y. draudžiama eiti įstrižai.

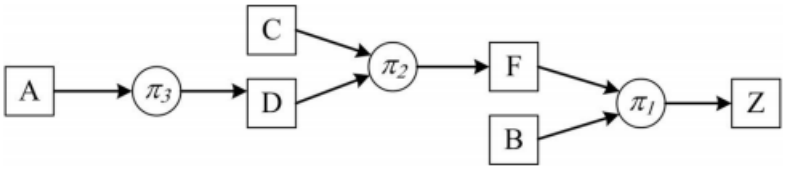


17.1 pav. a) Agentas keliauja iš *s* į *t*. b) Keturi ėjimai. c) Uždaromos ir atidaromos viršūnės (frontas) bei rastas kelias iš *s* į *t*

10. Tiesioginis išvedimas ir atbulinis išvedimas produkcijų sistemoje. Semantinis grafas. Programų sintezės elementai. Išvedimo sudėtingumas.
Tiesioginio išvedimo sudėtingumas N^2

18.1. Tiesioginio išvedimo algoritmas

Pradedama nuo pradinės GDB būsenos, t. y. nuo faktų. Produkcijos perrenkamos iteracijomis. Kiekvienoje iteracijoje produkcijos perrenkamos iš eilės. Jeigu paimtą produkciją galima taikyti, tai ji yra taikoma ir pažymima, kad daugiau nebūtų taikoma kitose iteracijose; valdymas perduodamas kitai iteracijai. Priešingu atveju imama kita produkcija. Jeigu produkcijos išsemtos, tai nesėkmė – ieškoma produkcijų seka neegzistuoja.



18.5 pav. Semantinis grafas, vaizduojantis tikslo kintamojo Z išvedimą iš {A,B,C} produkcijų sistemoje (18.1). Kelias yra seka $\langle \pi_3, \pi_2, \pi_1 \rangle$

3 Atbulinis išvedimas

3.1 Pseodokodas

```
BackwardChain(G)
if( G yra darbinėje atmintyje ) return true
if( nėra tokios taisyklės su konsekventu, kuris sutaptu su G ) return false
for each rule {
        if( taisyklės konsekventas sutampa su G){
                if ( visos pasirinktos taisykles prielados p BackwardChain(p) = true ){
                        return true
                }
        }
}
return false
```

Produkcijas atitinka procedūros algoritminėje kalboje. Todėl šiame pavyzdyje ir tiesioginio, ir atbulinio išvedimo rezultatas yra produkcijų seka $\langle \pi_3, \pi_2, \pi_1 \rangle$, kuri atitinka šitokią *sintezuotą programą*:

```
call P3;
call P2;                    (18.4)
call P1
```

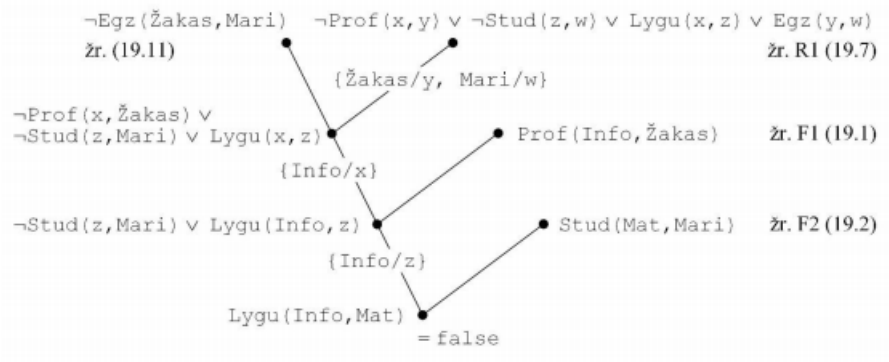
11. Tiesioginė ir atbulinė dedukcija pagal rezoliucijos taisyklę.

Rezoliucijos taisyklė paprasčiausia forma užrašoma šitaip (grafiškai parodyta 19.2 pav.):

$$\frac{P,\quad \neg P \vee Q}{Q}\ \sigma \qquad \textbf{Rezoliucijos taisyklė} \text{ (paprasčiausia forma)} \qquad (19.8)$$

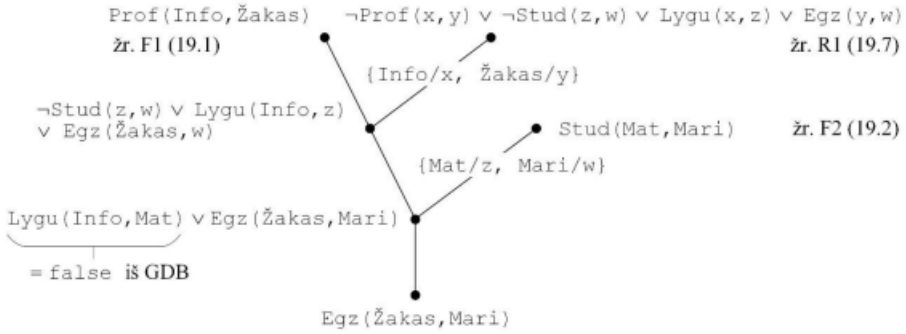
Bendru atveju rezoliucijos taisyklė užrašoma šitaip, kur $n \geq 0$:

$$\frac{P \vee H,\quad \neg P \vee Q}{H \vee Q}\ \{A_1/x_1,\ A_2/x_2, \ldots A_n/x_n\} \qquad \textbf{Rezoliucijos taisyklė} \text{ (bendresnė)} \qquad (19.10)$$



19.5 pav. Teoremos `Egz(Žakas,Mari)` atbulinio išvedimo medis: nuo tikslo paneigimo iki prieštaros

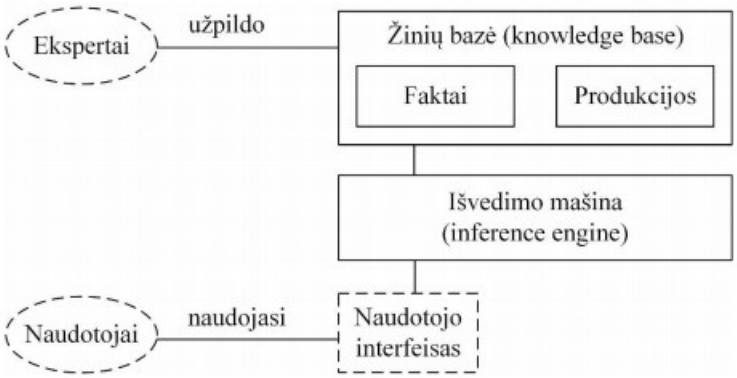
Tiesioginis įrodymas – nuo faktų prie tikslo. Duomenų bazėje yra du faktai – F1 (19.1) bei F2 (19.2) – ir viena produkcija R1 (19.7). Įrodyti `Egz(Žakas,Mari)`.



19.6 pav. Teoremos `Egz(Žakas,Mari)` tiesioginio išvedimo medis – nuo faktų prie tikslo

12. Ekspertinė sistema kaip dirbtinio intelekto sistema; jos architektūra, demonstracinis pavyzdys.

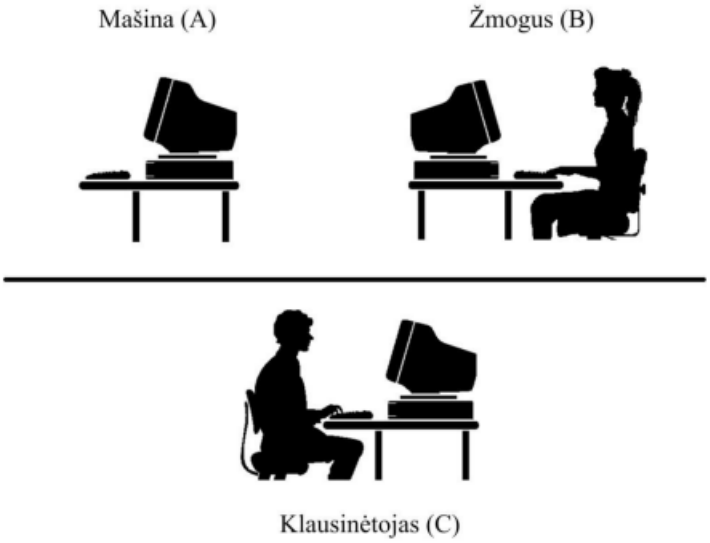
Bendroji ekspertinės sistemos architektūra yra parodyta 20.1 pav.



20.1 pav. Bendroji ekspertinės sistemos architektūra

13. Tiuringo testas ir dirbtinio intelekto filosofija (remiantis skaityta literatūra)
Kinų kambario argumentas (Chinese Room Argument)

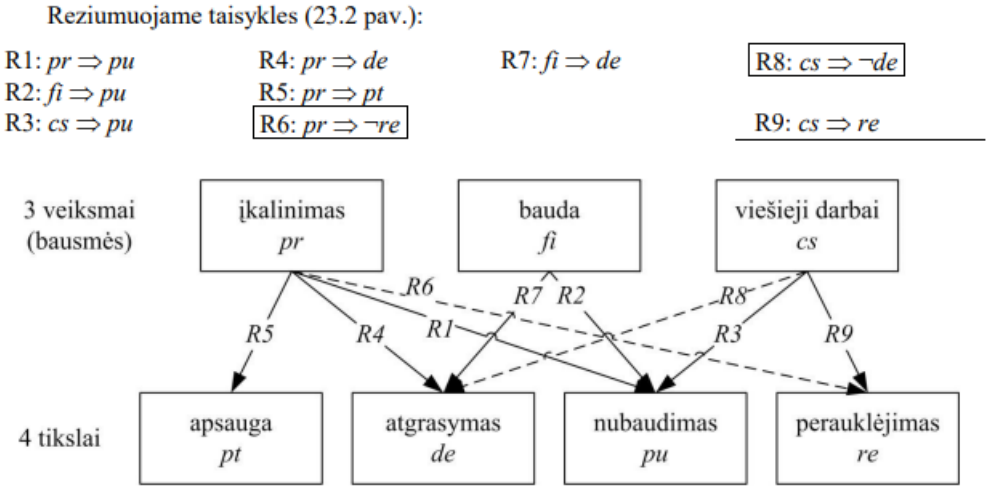
Pastebėsime, kad naujoje formuluotėje nebedalyvauja moteris, tačiau žaidėjų tikslai tie patys (22.2 pav.). Taigi Tiuringo testo paskirtis yra ne išsiaiškinti, ar konkreti mašina A gali imituoti moterį B imitaciniame žaidime, bet išsiaiškinti, ar mašina apskritai gali imituoti žmogų. Daugumoje vėlesnių Tiuringo testo formuluočių lytis yra ignoruojama ir laikoma, kad žaidime dalyvauja mašina A, žmogus B ir klausinėtojas C. Klausinėtojo tikslas yra nustatyti, kuris iš A ir B yra žmogus ir kuris mašina.



22.2 pav. Tiuringo testas yra suprantamas kaip imitacinis žaidimas. Klausinėtojas siekia išsiaškinti, kuris iš A ir B yra žmogus ir kuris mašina. Mašina A siekia apkvailinti klausinėtoją, kad ji žmogus

14. Internetinės parduotuvės specifikacija pagal Russell & Norvig vadovėlį.
Rekursyvus web-crawler

15. Neįmanomumas pasiekti keletą tikslų. Nusikaltėlio nubaudimo pavyzdys.



23.2 pav. Priežastiniai ryšiai tarp veiksmų ir tikslų. Punktyrinės briaunos *R6* ir *R8* vaizduoja neigiamą poveikį

16. Ekstensionalinė reliacinė struktūra, pasaulis, intensionalinis santykis, intensionalinė reliacinė struktūra, ekstensionalinė pirmos eilės struktūra (kalbos modelis), intensionalinė pirmos eilės struktūra (ontologinis įsipareigojimas), numatomi modeliai ir ontologija.

24.1 apibrėžtis. Ekstensionalinė reliacinė struktūra yra pora $S = (D, \mathbf{R})$, kur

- D yra aibė, vadinama diskurso universumu (*universe of discourse*)
- $\mathbf{R}$ yra santykių virš D aibė. (Guarino et al. 2009, 2.1 apibrėžtis) □

Kiekvienas R elementas yra ekstensionalinis santykis, t.y. matematinis santykis, dekartinės sandaugos poaibis.

24.5 apibrėžtis. Intensionalinis santykis (dar vadinama koncepcinis santykis ar ryšys) $\rho^{(n)}$ virš (D, W) yra atvaizdavimas $W \rightarrow powerset(D^n)$. Kitais žodžiais, $\rho^{(n)}$: $w_i \rightarrow \rho^{(n)}(w_i)$, t.y. $\rho^{(n)}$ atvaizduoja pasaulio būseną w_i į dekartinės sandaugos D^n poaibį $\rho^{(n)}(w_i)$ – į kortežų iš D^n aibę, kitaip sakant, į ekstensionalinę reliacinę struktūrą. (Guarino et al. 2009, 2.3 apibrėžtis) □

Čia $powerset(A)$ žymi aibės A poaibių aibę. Dar žymima 2^A. Pavyzdžiui, tegu $A = \{a, b\}$. Tada $powerset(A) = 2^A = \{\ \{\}, \{a\}, \{b\}, \{a, b\}\ \}$.

24.6 apibrėžtis. Intensionalinė reliacinė struktūra (dar vadinama konceptualizacija) yra trejetas $\mathbf{C} = (D, W, \mathcal{R})$, kur

- D – diskurso universumas
- W – pasaulis
- $\mathcal{R}$ – aibė intensionalinių santykių virš (D, W). (Guarino et al. 2009, 2.4 apibr.) □

24.8 apibrėžtis (ekstensionalinė pirmosios eilės struktūra $M = (S, I)$). Tegu

- $\mathbf{L}$ – pirmosios eilės kalba su žodynu $\mathbf{V}$
- $S = (D, \mathbf{R})$ – ekstensionalinė reliacinė struktūra

Ekstensionalinė pirmosios eilės struktūra (dar vadinama **kalbos L modelis**) yra pora $M = (S, I)$, kur I (vadinama **ekstensionalinė interpretacijos funkcija**) yra atvaizdavimas I: $\mathbf{V} \rightarrow D \cup \mathbf{R}$, toks, kad konstantų simboliai atvaizduojami į aibę D ir predikatų simboliai į $\mathbf{R}$. (Guarino et al. 2009, 3.1 apibrėžtis) □

24.9 apibrėžtis (intensionalinė pirmos eilės struktūra $\mathbf{K} = (\mathbf{C}, \mathcal{I})$). Tegu

- $\mathbf{L}$ – pirmosios eilės kalba su žodynu $\mathbf{V}$
- $\mathbf{C} = (D, W, \mathcal{R})$ – intensionalinė reliacinė struktūra, t.y. konceptualizacija

Intensionalinė pirmos eilės struktūra (dar vadinama **ontologinis įsipareigojimas**) kalbai $\mathbf{L}$ yra pora $\mathbf{K} = (\mathbf{C}, \mathcal{I})$, kur $\mathcal{I}$ (vadinama **intensionalinė interpretacijos funkcija**) yra atvaizdavimas $\mathcal{I}$: $\mathbf{V} \rightarrow D \cup \mathcal{R}$, toks kad konstantų simboliai atvaizduojami į aibę D ir predikatų simboliai į aibę $\mathcal{R}$. (Guarino et al. 2009, 3.2 apibrėžtis) □

24.11 apibrėžtis (numatomi modeliai $\mathbf{I_K(L)}$). Tegu

- $\mathbf{C} = (D, W, \mathcal{R})$ – intensionalinė reliacinė struktūra (konceptualizacija)
- **L** – pirmosios eilės kalba su žodynu **V**
- $\mathbf{K} = (\mathbf{C}, \mathcal{I})$ – intensionalinė pirmosios eilės struktūra (ontologinis įsipareigojimas)

Modelis $M = (S, I)$, kur $S = (D, \mathbf{R})$, yra vadinamas numatomu kalbos **L** modeliu pagal **K** tada ir tik tada, kai

1. Kiekvienai konstantai $c \in \mathbf{V}$ turime $I(c) = \mathcal{I}(c)$
2. Egzistuoja toks pasaulis $w \in W$, kad kiekvienam predikatiniam simboliui $v \in \mathbf{V}$ egzistuoja intensionalinis santykis $\rho^{(n)} \in \mathcal{R}$, toks kad $\mathcal{I}(v) = \rho^{(n)}$ ir $I(v) = \rho^{(n)}(w)$

Numatomi modeliai $\mathbf{I_K(L)}$ yra aibė visų numatomų kalbos **L** modelių pagal **K**.
(Guarino et al. 2009, 3.3 apibrėžtis) □

24.12 apibrėžtis (ontologija $\mathbf{O_K}$). Tegu

- $\mathbf{C} = (D, W, \mathcal{R})$ – intensionalinė reliacinė struktūra (konceptualizacija)
- **L** – pirmos eilės kalba su žodynu **V**
- $\mathbf{K} = (\mathbf{C}, \mathcal{I})$ – intensionalinė pirmosios eilės struktūra (ontologinis įsipareigojimas)

Ontologija $\mathbf{O_K}$ intensionalinei reliacinei struktūrai **C**, žodynui **V** ir ontologiniam įsipareigojimui **K** yra loginė teorija, sudaryta iš kalbos **L** tokios formulių aibės, kuri kiek galima geriau specifikuoja numatomų modelių aibę $\mathbf{I_K(L)}$. (Guarino et al. 2009, 3.4 apibrėžtis) □