



A-PDF Page Crop DEMO: Purchase from

UMELÁ INTELIGENCIA

prednášateľ: Prof. Ing. Pavol Návrat, PhD.

štúdium: bakalárske

študijný odbor: Informatika

nominálny ročník: III

Absolvovanie predmetu

Študent napĺňa podmienky absolvovania predmetu preukazovaním vlastných vedomostí. Odovzdanie práce prevzatej od iného, aj ak sa vhodne cituje, nevedie k naplneniu podmienok, pokiaľ študent súčasne nepreukázal vlastné vedomosti v dostatočnej miere.

Podmienky na získanie zápočtu:

- Vypracovanie a odovzdanie všetkých požadovaných zadanií s tým, že za každé jedno zadanie musí študent získať aspoň jeden bod a v súhrne zo všetkých požadovaných zadanií musí získať aspoň 17,5 bodu - t.j. polovicu možných bodov zo všetkých zadanií.
- Získanie aspoň 7 bodov z priebežného testu.

Podmienky na vykonanie skúšky:

Získanie zápočtu.

Podmienky na absolvovanie predmetu:

Získanie dostatočného počtu bodov (podľa čl. 4 študijného poriadku), ktorými sa hodnotí:

- zadania (max. 35 bodov)
- priebežný test (max. 20 bodov)
- záverečná skúška (max. 45 bodov)

Pričom je nutné zo záverečnej skúšky získať aspoň 18 bodov.

Literatúra:

Odporúčaná literatúra:

Návrat a kol.: Umelá Inteligencia, STU Bratislava, 2002, 2006. (Malé centrum)

Vzorová svetová literatúra:

1. Russell, Norvig: Artificial Intelligence: A Modern Approach. Prentice Hall, 1995. Tiež druhé vydanie 2002

Dostupná literatúra:

1. Kelemen a spol.: Základy umelej inteligencie. Alfa 1992.
2. I.M. Havel: Robotika. Úvod do teórie kognitívnych robotov. SNTL 1981.
3. Mařík a spol: Umělá inteligence (1), (2), (3) a (4), Academia Praha, 1993, 1997, 2000 a 2003.

Akademická bezúhonnosť

Odpisovanie je vedomé prezentovanie cudzej práce ako svoj vlastný výsledok. V tomto predmete sa nebude plagiát tolerovať.

Plagiát je prevzatá myšlienka bez priznania, že autorom je niekto iný (napr. citovaním). Môže ale nemusí byť vyjadrená totožným alebo podobným spôsobom ako v origináli. Často je to odpisaná alebo parafrázovaná alebo inak upravená pasáž (aj časť programu) z diela iného autora bez jej citovania.

V každej práci, ktorú predkladáme ako vlastný výsledok (napr. zadanie, program, projekt) treba uviesť všetky zdroje informácií, ktoré sme použili pri vypracovaní.

Nedodržanie sa podľa Štatútu FIIT STU posudzuje a rieši pred disciplinárnom komisiou.

Robocup: simulačná liga, 2D



Robocup: turnaj FIIT 2003



Robocup: turnaj FIIT 2010



Robocup: turnaj FIIT 2012

- niekedy v máji 2012
- tímové projekty, diplomové projekty, zadania UI
- alternatívna možnosť namiesto zadaní UI: z2, z3, z4
- http://www.fiit.stuba.sk/generate_page.php?page_id=2637

Robocup: liga humanoidov, detská veľkosť



Humanoid KidSize (Foto: D. Kriesel)

Robocup: liga humanoidov, týnedžerská veľkosť



Humanoid TeenSize (Foto: D. Kriesel)

Umelá inteligencia

- Cieľom UI je vytvoriť, zstrojiť inteligentné objekty a porozumieť im.
- Metóda UI je vo svojej podstate spätá s použitím výpočtových procesov.

ČO TO JE UMELÁ INTELIGENCIA?

- či sa skúma alebo sa usiluje o myšlienkové procesy a usudzovanie na jednej strane alebo o správanie sa na druhej strane,
- či sa hodnotí úspech podľa podobnosti s ľudským konaním alebo s ideálnou predstavou o inteligencii - tzv. rozumnosťou. Systém je rozumný, ak robí správnu vec.

Allan Newell

- (* 19. marec 1927, San Francisco, Kalifornia, USA - † 19. júl 1992, Pittsburgh, Pensylvánia, USA)
- 1949 – Bc, Stanford
- 1950 – MS matematika, Princeton
- PhD – CMU Tepper School of Business, školiteľ Herb Simon
- americký informatik a kognitívny vedec
- RAND Corporation
- výskum v umelej inteligencii a kognitívnej vede
- 1956 - Logic Theory Machine
- 1957 – General Problem Solver
- 1975 – Turingova cena spolu s Herbertom Simonom

A handwritten signature in cursive script that appears to read "Allan Newell".

Herbert Simon

- (June 15, 1916 Milwaukee, Wisc. – February 9, 2001, Pittsburgh, Penn.)
- 1936 Bac – U Chicago
- 1942 PhD administratívne rozhodovanie, U Chicago / U Cal Berkeley
- americký politológ, ekonóm, sociológ, psychológ
- zakladateľ / významné príspevky:
- kognitívna psychológia, verejná správa, ekonómia, informatika, manažment, politológia
- vedecké výsledky:
 - umelá inteligencia, spracovanie informácií, rozhodovanie, riešenie problémov, zložité systémy
- pôsobil na CMU 52 rokov
- 1975 Turingova cena
- 1978 – Nobelova cena za ekonómiu za priekopnícky výskum procesu rozhodovania v hospodárskych organizáciách



rôzne pohľady na UI

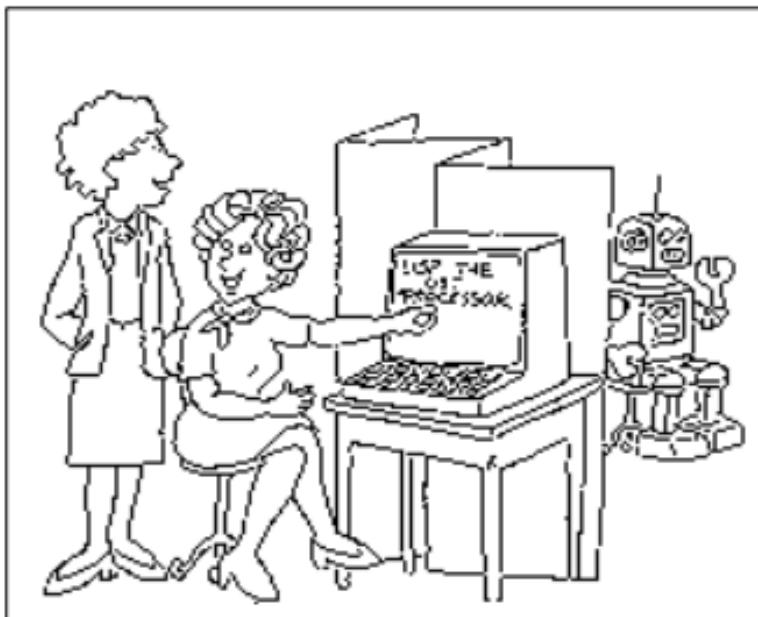
- systémy, ktoré myslia ako ľudia,
 - GPS (general problem solver - všeobecný riešič problémov) (Newell a Simon, 1961)
 - Kognitívna veda spája skúmanie výpočtových modelov z UI a experimentálnych metód psychológie s cieľom nájsť presné a overiteľné teórie fungovania ľudského rozumu.
- systémy, ktoré konajú ako ľudia,
 - Turingov test: systém koná ako človek (t.j. inteligentne), ak dokáže prekabátiť vyšetrovateľa tak, že ho nedokáže rozlišiť od človeka.
- systémy, ktoré myslia rozumne,
 - sylogizmy vyjadrujú vzory správneho myслenia
- systémy, ktoré konajú rozumne
 - systém koná tak, aby dosiahol svoje ciele s ohľadom na tvrdenia, ktorých pravdivosť predpokladá (ktorým verí).

Alan Turing

- 23. jún 1912 Maida Vale, London, Anglicko – 7. jún 1954 Wilmslow, Cheshire, Anglicko)
- 1934 – Bc matematika, King's College Cambridge
- 1938 – PhD matematika, Princeton (školiteľ Alonzo Church)
- anglický matematik, logik, kryptoanalytik, informatik
- formalizácia pojmov algoritmus a výpočet – Turingov stroj
- problém zastavenia
- 1950 – Môžu stroje myslieť? – Turingov test



Turingov test



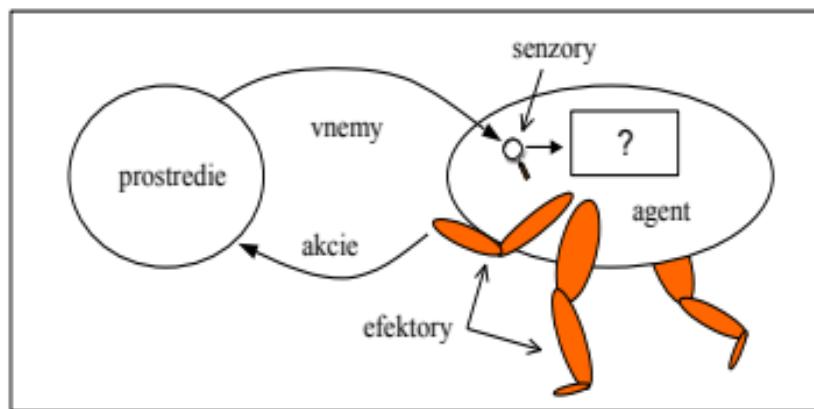
Umelá inteligencia

- disciplína, ktorá skúma rozumné konatele a spôsoby ich zstrojovania.
- Konatel' (agent) je systém, ktorý vníma a koná.
- Ústrednou hypotézou v tomto prístupe je chápanie inteligencie ako rozumného konania.

Rozumný konateľ

- Ideálny rozumný konateľ by mal pre ľubovoľnú možnú postupnosť vnemov vykonať na základe faktov získaných postupnosťou vnemov a všetkých znalostí, ktoré má v sebe zapísané takú akciu, od ktorej sa očakáva čo najväčšie ohodnotenie mierou úspešnosti.
 - vstup: postupnosť vnemov
 - výstup: konanie (akcia), ktoré je reakciou na postupnosť vnemov.
- Navrhnutý ideálny rozumný konateľ znamená špecifikovať, akú akciu má vykonať ako odpoveď na ľubovoľnú postupnosť vnemov.
- rozumný konateľ = program + technické zariadenie

Konatel' - agent



agent

- vnem: vstup, ktorý agent získa vnímaním
- postupnosť vnemov: úplná história všetkého, čo agent vnímal
- funkcia: zobrazenie ľubovoľnej postupnosti vnemov do akcie
- program: vykonáva sa na fyzickej architektúre agenta, realizuje jeho funkciu
- agent = architektúra + program

opis úlohy agenta

- úspešnosť (performance measure)
 - objektívna miera hodnotiaca úspešnosť konania agenta
- prostredie
- aktuátory
- senzory

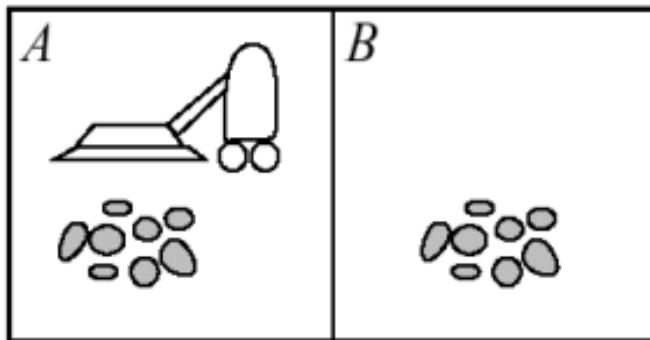
automatizovaný taxík

- úspešnosť:
 - bezpečnosť, dosiahnutie cieľa, zisk, dodržiavanie predpisov, pohodlie zákazníka,...
- prostredie:
 - európska cestná siet', iní účastníci cestnej premávky, chodci, počasie
- aktuátory:
 - volant, rýchlosný pedál, brzda, klaksón, displej,...
- senzory:
 - tachomer, otáčkomer, ďalšie snímače stavu motora, okolia, GPS,...

internetový kupujúci

- úspešnosť:
 - cena, kvalita, vhodnosť, efektívnosť,...
- prostredie:
 - súčasné aj budúce webové sídla, predajcovia, dodávateelia,...
- aktuátory:
 - displej pre používateľa, prechod na inú stránku podľa URL,...
- senzory:
 - HTML stránky (text, grafika, skripty),...

svet vysávača



- 2 miesta: miestnosť A, miestnosť B
- Agent vníma miesto a jeho stav
(čisté/špinavé) (dirty/not dirty)
- Akcie: doľava, doprava, vysávaj, no_op

vysávací agent

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
:	:

```
function REFLEX-VACUUM-AGENT([location, status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

- Aký je „správny“ spôsob vyplnenia tabuľky?
- ‘Správny’ spôsob robí agenta dobrým/inteligentným

Rozumnosť

- “robiť správnu vec”, formálnejšie:
- “Rozumný agent je taký, ktorý koná tak, aby dosahoval najlepší výsledok alebo, ak je neurčitosť, najlepší očakávaný výsledok.”
- otázky:
 - Čo to znamená ‘najlepší’?
 - Čo je výsledok?
 - Čo to stojí dosiahnuť výsledok?
 - Čo všetko treba na vypočítanie ‘očakávaného’ výsledku?

Rozumnosť

- Čo je rozumné závisí od:
 - kritéria úspešnosti
 - postupnosti vnemov
 - agentových apriorných znalostí o prostredí
 - akcií, ktoré dokáže vykonávať
- Rozumný agent: vyberá si akciu, ktorá maximalizuje jeho úspešnosť, na základe faktov daných postupnosťou vnemov a apriórnymi znalosťami o prostredí

mierka úspešnosti

- opatrne s voľbou!
 - Vysávací agent: merať úspešnosť množstvom špinív vyčistenej počas 8-hodinovej smeny
- navrhovať podľa toho, čo chceme dosiahnuť v prostredí, nie podľa toho, ako sa má agent správať

Je vysávací agent rozumný?

- áno za týchto predpokladov:
 - mierka úspešnosti: 1 bod za každú čistú miestnosť
 - pozná rozmiestnenie miestností ale nepozná ktoré sú špinavé ani ktorá má byť jeho začiatočná pozícia
 - čisté miestnosti zostávajú čisté, vysávanie čistí
 - pohyby doľava alebo doprava nezavedú agenta mimo prostredie
 - dostupné akcie: doľava, doprava, vysávaj, NoOp
 - agent vie, kde sa nachádza a je to miesto špinavé

Je vysávací agent rozumný?

- ale za iných predpokladov by vysávací agent neboli rozumný
 - mierka úspešnosti určená pokutou za zbytočný pohyb
 - ak sa čisté miesta môžu stať špinavými
 - ak celé prostredie nie je známe
 - ...

viac o rozumnosti

- rozumnosť nie je vševedúkosť
- rozumnosť nie je jasnovidnosť
- rozumnosť nemusí viesť k úspechu !
- rozumné správanie často vyžaduje
 - zbieranie informácií: skúmanie neznámeho prostredia
 - učenie sa: zistiť, ktorá akcia pravde podobne povedie k želanému výsledku (a získať spätnú väzbu z prostredia o úspechu)
- ...takže rozumný agent by mal byť autonómny
(nespolieha sa výlučne len na apriornu vedomosť jeho návrhára, učí sa zo svojej skúsenosti)

bližšie o prostredí

- prostredie môže byť
 - skutočné alebo umelé
 - jednoduché (napr. dopravníkový pás) alebo zložité (letový simulátor)
- rozhoduje zložitosť vzťahov medzi správaním sa robota, postupnosťou vnemov generovanou prostredím a mierou pre úspešnosť

vlastnosti prostredia

- úplná alebo čiastočná pozorovateľnosť
 - úplná: agentove senzory sprístupňujú úplný stav prostredia v každom okamihu
 - efektívne úplná: (stačí ak) senzory rozpoznajú všetky aspekty relevantné pre výber akcie (tak, ako určuje miera pre úspešnosť)
 - úplná: agent nepotrebuje vnútorný stav na reprezneotvanie stavu prostredia

vlastnosti prostredia

- Deterministické vs stochastické
 - Deterministické ak je nasledujúci stav prostredia úplne určený súčasným stavom a akciou, ktorú agent vykoná
 - čiastočne pozorovateľné prostredie môže sa javiť ako stochastické

vlastnosti prostredia

- Epizodické vs sekvenčné
 - Epizodické prostredie: agentova skúsenosť sa člení na atomické epizódy: každá epizóda pozostáva z vnímania a potom vykonania jednej akcie
 - epizódy sú nezávislé: ďalšia epizóda nezávisí od akcií vykonalých pri predchádzajúcich epizódach
 - napr: klasifikačná úloha: rozpoznanie chybnej súčiastky na montážnej linke
 - sekvenčné: súčasné rozhodnutie môže ovplyvniť všetky budúce rozhodnutia
 - napr. ťah v šachu

vlastnosti prostredia

- Statické vs dynamické
 - Dynamické: prostredie sa môže meniť počas toho, keď agent hľadá ďalšiu akciu
 - semidynamické: ohnodnotenie úspešnosti sa môže meniť v čase, hoci prostredie sa nemení (napr. hranie v šachy s hodinami)
- Diskrétné vs spojité
 - tento rozdiel sa môže vzťahovať na stav prostredia, spôsob práce s časom, vnemy alebo akcie

vlastnosti prostredia

- jeden agent vs viac agentov
 - Ako rozhodnúť, či nejaký iný objekt sa má chápať ako agent?
 - Je to agent alebo len stochasticky sa správajúci objekt (napr vlna naobreží)?
 - Základná otázka: dá sa jeho správanie opísť ako maximalizácia úspešnosti v závislosti od akcií „nášho“ agenta?
 - viackonateľské (multiagentové) prostredie sa dá klasifikovať ako (čiastočne) súťaživé a/alebo (čiastočne) spolupracujúce
 - napr taxíky sú čiastočne súťaživé a čiastočne spolupracujúce

príklady prostredia

- Solitér: pozorovateľné, deterministické, sekvenčné, diskrétne, statické, jednoagentové
- Backgammon: pozorovateľné, deterministické, sekvenčné, diskrétne, semi-statické, multiagentové
- Internetové nakupovanie: čiastočne pozorovateľné, čiastočne deterministické, sekvenčné, semi-statické, diskrétne, jednoagentové (okrem aukcií, napr ebay)
- jazdenie v taxíku (“skutočný svet”): čiastočne pozorovateľné, nedeterministické, sekvenčné, spojité, multiagentové

Príklady rozumných agentov

Druh agenta	Vnemy	Akcie	Ciele	Prostredie
lekársky dia-g-nos-tický systém	symptómy, nálezy, odpo-vede pacienta	otázky, testy, liečebné postupy	zdravý pa-cient, min. ná-klady	pacient, ne-mocnica
systém ana-lyzy sate-lit-úch sní-mok	body snímku súmej inten-sity a farby	vyliačenie osa-menia o-kate-go-ezicii scény	správna kate-gori-zácia	obrazy z obie-hajúceho sate-litu
robot na trié-de-nie súčas-tok	body snímku súmej inten-sity	uchopenie sú-čas-tky, umies-menie do koša	súčasťky sú v správnych koloch	bežiaci pás so súčasťami súmeho druhu
systém riade-nia rafinérie	zostávajúce hodnoty tep-loty a tlaku	otvorenie/unavre-tie ven-tilov, prs-spô-sobenie tep-loty	maximálna čislosť, výfa-lok a heape-l-nosť	rafinéria
vodič taxi	kamery, GPS, tachometer, mikrofón	smerovanie, bra-denie, zrýchlo-va-nie, komunika-cia s pasažierom	bezpečná, po-ho-dlná, le-gál-na, rýchla do-prava do cieľa	cesty, doprav-né značky, se-mafor, chodeci a zákaznice

systém na pod-poru učenia sa angličtiny	slová napi-sa-né kláves-ní-cou	vyliačenie evi-čení, návodov, optív	max. po-let bodov ita-denta na teste	množina ita-dentov
---	--------------------------------	-------------------------------------	--------------------------------------	--------------------

program rozumného agenta vo veľmi zjednodušenej podobe

```
function Agent-Kostra(vnem) returns akcia
```

static: *pamäť*, pamäť agenta o svete

*pamäť' ← Obnov-Pamäť'(*pamäť*', *vnem*)*

*akcia ← Vyber-Najlepšiu-Akciu(*pamäť*)*

*pamäť' ← Obnov-Pamäť'(*pamäť*', *akcia*)*

return *akcia*

Agent riadený tabuľkou

```
function Agent-Riadenny-Tabulka(vnem) returns akcia
```

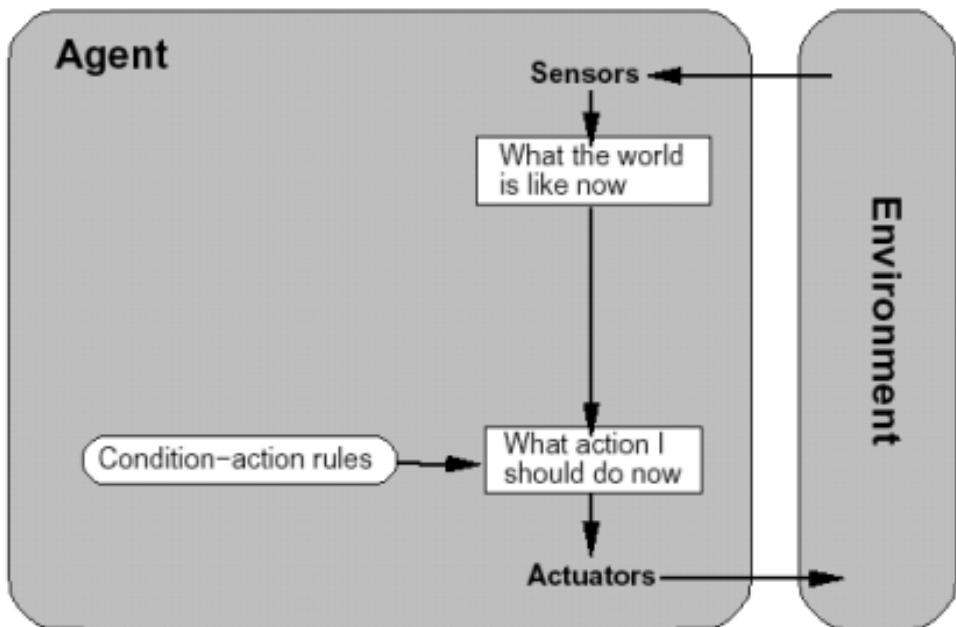
static: *vnemy*, postupnosť, na začiatku prázdna
tabuľka, položky sprístupniteľné podľa postupnosti vnemov,
na začiatku úplne špecifikovaná
pripoj *vnem* na koniec *vnemy*
akcia \leftarrow Vyhľadaj(*vnemy*, *tabuľka*)

return *akcia*

Pre šach by musela mať približne 35^{100} položiek

Ak by agent konal výlučne na základe v ňom zapísaných znalostí a vôbec by nepotreboval brat' do úvahy vnemy, tak by vôbec nebol autonómny.

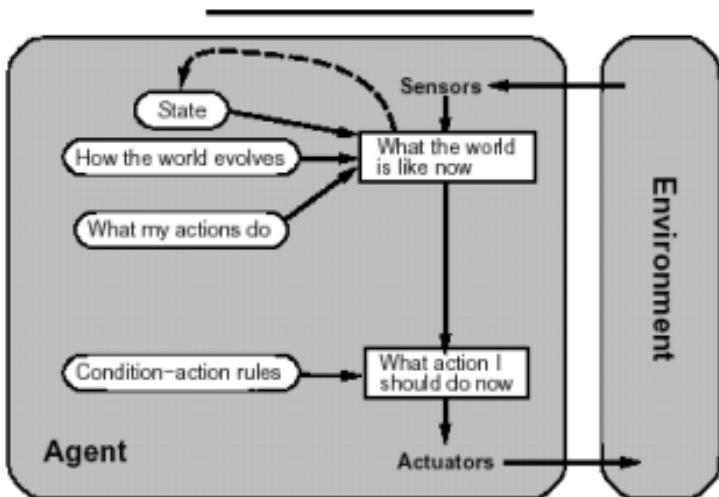
Agent s odrazom



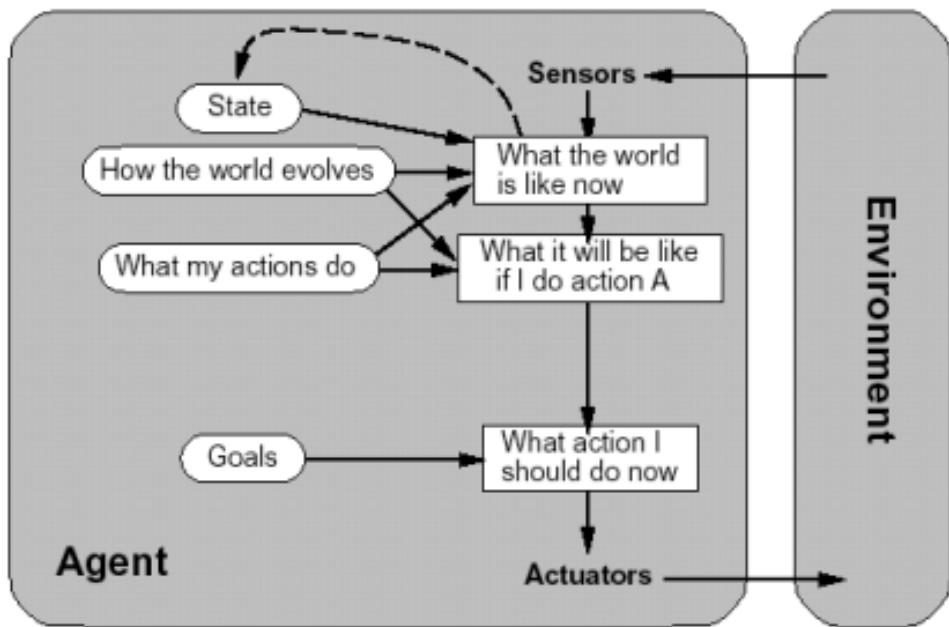
Agent s odrazom

```
function Agent-s-Odrazom(vnem) returns akcia
    static: stav, opis súčasného stavu prostredia
            pravidlá, množina pravidiel v tvare podmienka-akcia
    stav  $\leftarrow$  Obnov-Stav(stav, vnem)
    akcia  $\leftarrow$  Urči-Akciu(Nájdi-Pravidlo(stav, pravidlá))
    stav  $\leftarrow$  Obnov-Stav(stav, akcia)
    return akcia
```

Agent s odrazom založený na modeli



Agent uvažujúci cieľ



Agent uvažujúci ciel'

```
function Agent-Uvažujúci-Ciel'(vnem) returns akcia
```

static: stav, opis súčasného stavu prostredia

ciel', na začiatku prázdný

stav \leftarrow Obnov-Stav(stav, vnem)

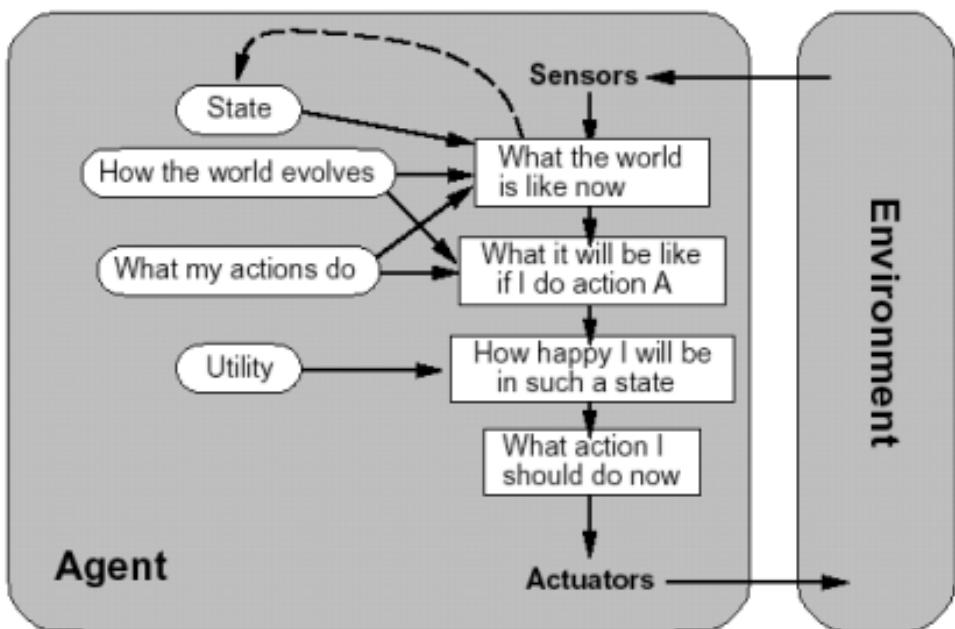
if ciel' je prázdný **then** ciel' \leftarrow Vyjadri-Ciel'(stav)

akcia \leftarrow Urči-Akciu(ciel', Nájdi-Nasledovníky(stav))

stav \leftarrow Obnov-Stav(stav, akcia)

```
return akcia
```

Agent uvažujúci užitočnosť



Agent uvažujúci užitočnosť

```
function Agent-Uvažujúci-Užitočnosť(vnem) returns akcia
```

static: *stav*, opis súčasného stavu prostredia

užitočnosť, funkcia oceňujúca užitočnosť stavu

stav \leftarrow Obnov-Stav(*stav*, *vnem*)

akcia \leftarrow Urči-Akciu(*užitočnosť*, Nájdi-Nasledovníky(*stav*))

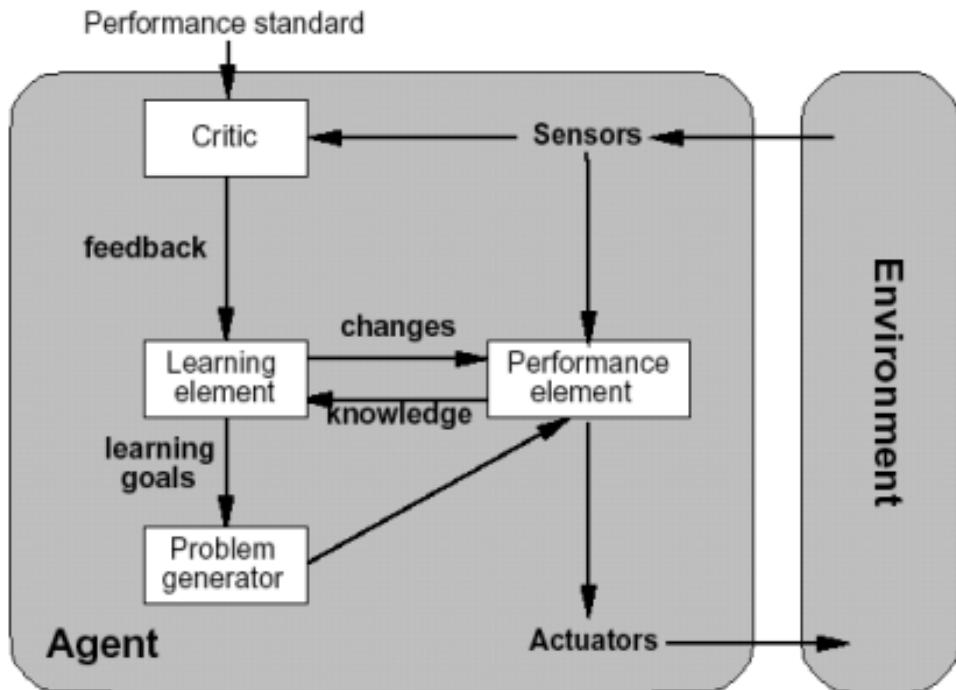
stav \leftarrow Obnov-Stav(*stav*, *akcia*)

return *akcia*

ohraničená rozumnosť

- ohraničenými výpočtovými prostriedkami (veľkosť pamäti, čas, dokedy treba rozhodnúť o ďalšom kroku),
- ohraničenými nákladmi na úsilie, ktoré možno vynaložiť na získanie údajov z prostredia (ohraničenie doby, ktorú získavanie môže najviac trvať, ohraničenie finančných nákladov získavania apod.),
- neúplnosťou a prípadnou protirečivosťou poznatkov v jeho báze,
- neurčitosťou niektorých poznatkov,
- nepresnosťou niektorých údajov.

učiaci sa agent



ohraničená rozumnosť

- ohraničenými výpočtovými prostriedkami (veľkosť pamäti, čas, dokedy treba rozhodnúť o ďalšom kroku),
- ohraničenými nákladmi na úsilie, ktoré možno vynaložiť na získanie údajov z prostredia (ohraničenie doby, ktorú získavanie môže najviac trvať, ohraničenie finančných nákladov získavania apod.),
- neúplnosťou a prípadnou protirečivosťou poznatkov v jeho báze,
- neurčitosťou niektorých poznatkov,
- nepresnosťou niektorých údajov.

Hľadanie riešenia problému

- Ak si rozumný agent prostredníctvom vnemu určí cieľ, môže problém vyriešiť vyhľadaním postupnosti akcií, vedúcich do cieľa.

```
function JEDNODUCHÝ-KONATEĽ-RIEŠIACI-PROBLÉM(vnem) returns akcia
    static: akcie, postupnosť akcií, na začiatku prázdna
            stav, nejaký opis súčasného stavu sveta
            cieľ, cieľ, na začiatku prázdný
            problém, vyjadrenie problému
    stav ← OBNOV-STAV(stav, vnem)
    if akcie je prázdna then
        cieľ ← VYJADRI-CIEĽ(stav)
        problém ← VYJADRI-PROBLÉM(stav, cieľ)
        akcie ← HĽADAJ(problém)
        akcia ← VYBER-PRVÚ(akcie, stav)
        akcie ← ZVÝŠOK-AKCIÍ(akcie, stav)
    return akcia
```

Hľadanie riešenia

Hľadanie riešenia je prístup k riešeniu problémov, pri ktorom nevychádzame z algoritmu riešenia problému.

Bud' ho nepoznáme (možno preto, že ani neexistuje), alebo ho poznáme, ale pre svoju neefektívnosť je prakticky nepoužiteľný. Namiesto toho vychádzame z algoritmu, ako riešenie hľadať.

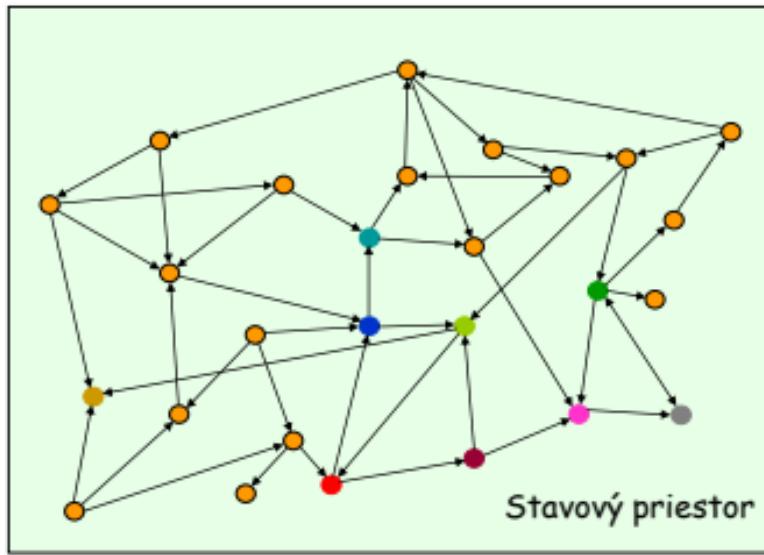
Definícia typu problému

- Na vyjadrenie problému treba poznať niekoľko základných informácií:
 - Začiatočný stav
 - Množinu operátorov
 - Množinu všetkých stavov
 - Cieľový test
 - Cenu cesty

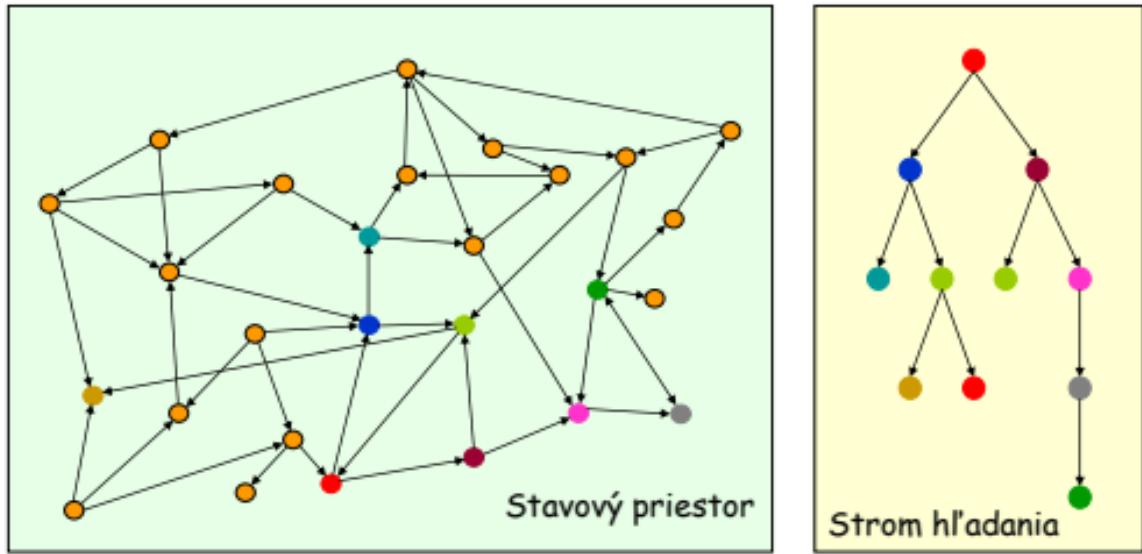
datatype PROBLÉM

components: STAVY, ZAČIATOČNÝ-STAV, OPERÁTORY,
CIEĽOVÝ-TEST, CENA-CESTY

Stavový priestor



Strom hľadania

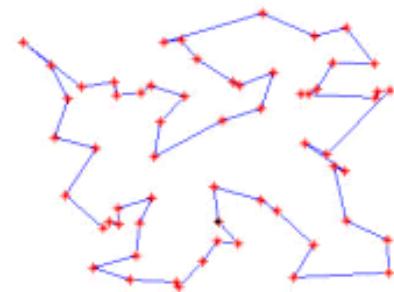
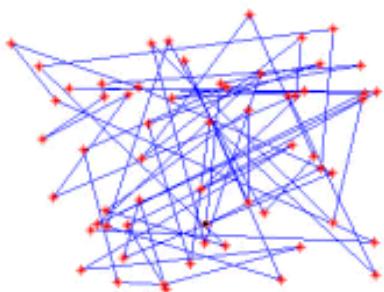


všimnime si, že pri hľadaní
sa niektoré stavy môžu
navštíviť viac ráz

Reálne problémy – problém nájdenia cesty

- **problém naplánovania najvhodnejšej cestovej trasy z mesta A do mesta B**
 - **stavy**: mestá, ktoré sa uvažujú pri hľadani;
 - **začiatočný stav**: mesto *A*;
 - **operátory**: možné presuny z jedného mesta do druhého (existuje cesta na mape);
 - **cieľový test**: "Sme v meste *B*?";
 - **cena cesty**: aplikácia operátora, t.j. presun z jedného mesta do druhého, má cenu rovnajúcu sa vzdialenosť medzi týmito mestami .

Reálne problémy – problém obchodného cestujúceho

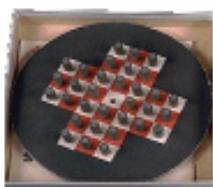
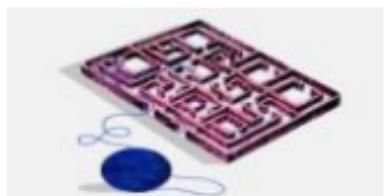
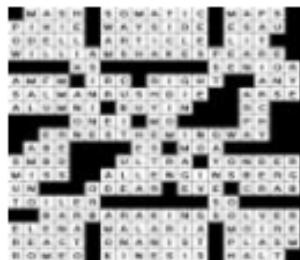


http://en.wikipedia.org/wiki/Traveling_salesman_problem

Reálne problémy – autonómne roboty

- Autonómny robot pri svojej činnosti rieši množstvo problémov:
 - Rozhodovanie, ktorú z možných akcií je treba vykonať
 - Predchádzanie koliziám
 - Plánovanie trajektórií
 - Interpretácia veľkého množstva numerických dát, poskytovaných senzormi do kompaktnnej zmysluplnnej symbolickej reprezentácie
 - Diagnostikovanie, prečo niečo nedopadlo podľa očakávaní
 - Atď ...
- Na riešenie týchto problémov je nevyhnutné používať rôzne metódy prehľadávania, pričom v jednom časovom okamihu sa môže vykonávať viacero prehľadávanií súčasne

Hračkové problémy



Hračkové problémy – 8 hlavolam

1	8	3
6	2	7
4		5

Začiatočný stav

1	2	3
8		4
7	6	5

Cieľový stav

Hračkové problémy – 8 hlavolam

- **stavy:** všetky možné konfigurácie políčok na tabuli. Opis stavu obsahuje údaje o umiestnení každého z 8 políčok na tabuli.
- **začiatočný stav:** pre každý zvláštny prípad problému musí byť zadaný začiatočný stav, t.j. východisková konfigurácia na tabuli
- **operátory:** výmena prázdnego miesta s poličkou vpravo, vľavo, hore, dolu
- **cieľový test:** súčasný stav opisuje konfiguráciu, ktorá sa zhoduje s danou cieľovou konfiguráciou
- **cena cesty:** každý krok má cenu 1, takže cena cesty je jednoducho jej dĺžka

Hračkové problémy – 15 hlavolam

- Sam Loyd, ktorý sám seba označil za najväčšieho experta na puzzle v Amerike, v roku 1878 ponúkol prvému človeku, ktorý vyrieši tento hlavolam, odmenu 1000 dolárov.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	



1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

Hračkové problémy – 15 hlavolam



NIKOMU SA TO VŠAK NEPODARILO ☺

Hračkové problémy – 15 hlavolam

- Aby bolo možné hlavolam vyriešiť, je nutné, aby bola hodnota **N mod 2** pre oba stavy rovnaká, pričom:

$N \text{ mod } 2 = n_2 + n_3 + \dots + n_{15} + \text{číslo riadku prázdnego políčka}$

n_i - počet všetkých tých políčok j, pre ktoré platí $i < j$ a zároveň sú umiestnené pred políčkom i

1	2	3	4
5	10	7	8
9	6	11	12
13	14	15	

$$n_2 = 0 \quad n_3 = 0 \quad n_4 = 0$$

$$n_5 = 0 \quad n_6 = 0 \quad n_7 = 1$$

$$n_8 = 1 \quad n_9 = 1 \quad n_{10} = 4$$

$$n_{11} = 0 \quad n_{12} = 0 \quad n_{13} = 0$$

$$n_{14} = 0 \quad n_{15} = 0$$

$$\rightarrow N = 7 + 4$$

Hračkové problémy – 15 hlavolam

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	



1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

$$N = 4$$

$$4 \bmod 2 = 0$$

$$0 \neq 1$$

$$N = 5$$

$$5 \bmod 2 = 1$$

- Druhý stav teda nie je z prvého dosiahnutelný a Sam Loyd sa o svoje peniaze nemusel báť.

Hračkové problémy – (n^2-1) hlavolam

- Aká je veľkosť stavového priestoru pre (n^2-1) hlavolam ?

	Počet stavov	Čas (10^8 stavov/sekunda)
9 hlavolam	$9! = 362,880$	0.036 sekundy
15 hlavolam	$16! \sim 2.09 \times 10^{13}$	~ 55 hodín
24 hlavolam	$25! \sim 10^{25}$	> 109 rokov

- Ale iba **POLOVICA** týchto stavov je dosiahnuteľných z ľubovoľného stavu.

Hračkové problémy – šach, dáma, go

- Už v minulosti boli hry, ktorých úspešné vyriešenie vyžadovalo preskúmanie rôznych alternatív výzvou pre ľudskú inteligenciu.
 - Šach – pôvod Perzia, India, pred 4000 rokmi
 - Dáma – prvé zmienky na starých egyptských maľbách spred 3600 rokov
 - Go – pôvod Čína, pred 3000 rokmi

Hračkové problémy – šach

- Veľkosť stavového priestoru 10^{120}
 - $10^{120} >$ počet všetkých atómov vo vesmíre
- 200 miliónov pozícii za sekundu = 10^{100} rokov na vyhodnotenie všetkých možných hier
 - Vesmír existuje iba 10^{10} rokov
- 1957 – Newell a Simon: „Do **desiatich rokov** sa počítač stane šachovým veľmajstrom“
- predpoved' im celkom nevyšla. podcenili potrebný čas, ale podcenili aj umelú inteligenciu (dnes je už počítačový program s umelou inteligenciou nielen šachový veľmajster, ale dokonca poráža majstra sveta).

Hračkové problémy – šach



Kasparov

165 cm
80 kg
34 rokov
50 miliárd neurónov
2 pozícií/s
Obrovské
Electrické/chemické
Enormné

Výška
Hmotnosť
Vek
Počítače
Rýchlosť
Znalosti
Napájanie
Ego

Deep Blue

195 cm
1,1 tony
4 roky
32 RISC procesorov
+ 256 VLSI šach. "enginov"
200,000,000 pozícií/s
Primitívne
Electrické
Žiadne

Deep Blue vyhráva po 3 výhrach, 1 prehre a 2 remízach

šach

- 10. 2. 1996 Philadelphia: Deep Blue porazil Kasparova v normálnej partii = vôbec prvý raz zvíťazil počítač nad úradujúcim majstrom sveta (celkovo ale zápas na 6 partií Kasparov vyhral 3 a 2 remizoval, 4:2)
- **11.5.1997** – Gary Kasparov prehráva s počítačom Deep Blue zápas na 6 partií: $3\frac{1}{2}$ - $2\frac{1}{2}$, prehral druhú a poslednú rozhodujúcu partiu. V poslednej spravil jasnú chybu, v druhej sa mu zdal počítač príliš tvorivý, IBM poprela ľudskú intervenciu.
- 2.8.2003 – Gary Kasparov remizuje s programom Deep Junior
 - cena je približne 100 dolárov
 - 3 milióny pozícii/s
 - knižnica otvorení
 - zaujímavejšie ťahy sa hlbšie prehľadávajú
 - modelovanie protihráča

šach

- 2002: Kramnik – Deep Fritz 4:4
- 2003: Kasparov – Deep Fritz 2:2
- 2006: Kramnik – Deep Fritz 2:4
 - program Fritz dnes (2011, 2012) stojí ~~99.90 €~~
~~49.90 €~~
 - minimálne požiadavky: Pentium 300 MHz, 64 MB RAM, Windows Vista or XP (SP 2), DVD ROM drive, Windows Media Player 9.

Rybka

- vyhral viacero turnajov šachových programov, vrátane majstrovstiev sveta 2007, 2008, 2009, 2010
- vyhral partie s veľmajstrami, ktorí dostali výhodu pešiaka
- vyhral zápas s veľmajstrom, ktorý mal všetky možné výhody okrem pešiaka: dvojnásobný čas na rozmyšľanie, naopak Rybka databázu otvorení obmedzenú len na 3 ťahy, obmedzenie na heš tabuľku 1/2 GB, bez databázy koncových hier. 4a1/2:1a1/2

Rybka

- verzie
 - vývoj začal 2003
 - 1 beta 2005 ELO=2885
 - 2.2 ELO=3110
 - 3 2008 ELO= cca +100 (v 2010 najvyššie vyhodnotený šachový program s ELO=3227)
 - 4 2010
 - 4+ cluster
 - 5 vraj sa pripravuje na 2012

Rybka

- α/β hľadanie
- používa reprezentáciu stavu pomocou bitových dosákov
 - bitová doska (bitboard) dátová štruktúra, zvláštny prípad bitovej množiny
 - jedna bitová doska má 64 bitov (toľko má šachovnica poličok)
- stav partie sa reprezentuje:
 - dvanásťimi bitovými doskami:
 - biele/čierne pešiaky, strelnici, jazdci, veže, dáma, kráľ
 - niekoľkými stavovými bitmi:
 - b/c dostať šach,
 - b/c ešte môže robiť rošádu.
- tím:
 - Vaclav „Vasik“ Rajlich, jr. medzinárodný majster v šachu
 - Iweta Radziewicz Rajlich, medzinárodná majsterka v šachu
 - Larry Kaufman, majster sveta v šachu 2008 (vyhodnocovacia funkcia)
 - Jeroen Noomen, Dag Nielsen (otvorenia)

Vasik Rajlich

- 1971 Cleveland
- MIT
- medzinárodný majster v šachu
- autor šachového programu
Rybka
<http://www.rybkachess.com/>
- &&&&&&&&&&&&&
- VÚMS Praha, U Michigan, CMU
- Wayne State U
- profesor, softvérové inžinierstvo



Rybka

- podľa stavu 2010: najvyššie hodnotený program, Elo = 3227
- vyhral viacero turnajov šachových programov, vrátane majstrovstiev sveta World Computer Chess Championships (WCCC): 2007, 2008, 2009, 2010
- June 29, 2011 at 1:03 pm:
 - Rybka has been disqualified and banned for the plagiarizing of two other chess engines, Crafty and Fruit.
 - International Computer Games Association (ICGA):
 - anulovala výsledky Rybky späťne za roky 2006-2010
 - odobrala tituly majstra sveta 2007-2010
 - V Rajlich má doživotný zákaz súťažiť na akomkoľvek turnaji organizovanom ICGA
 - musí vrátiť sošky a peniaze

Rybka - spor

- obvinenie:
 - VR nedodržal pravidlo č. 2 pre turnaje ICGA. Vyžaduje, aby autor šachového programu v prihláške na turnaj uviedol všetky zdroje, ktoré použil pri tvorbe svojho programu.
– "2. Each program must be the original work of the entering developers. Programming teams whose code is derived from or including game-playing code written by others must name all other authors, or the source of such code, in the details of their submission form. Programs which are discovered to be close derivatives of others (e.g., by playing nearly all moves the same), may be declared invalid by the Tournament Director after seeking expert advice. For this purpose a listing of all game-related code running on the system must be available on demand to the Tournament Director."
- argumentácia:
 - nemusí ísiť o presné, doslovné skopírovanie
 - treba priznať aj to, že nejaká časť je odvodnenia (derivative) od niečo iného bez ohľadu na zhodu/podobnosť na úrovni zdrojového programu
 - rozsah: nedá sa určiť presne (6? 12? 20? 200? riadkov), rozumné posúdenie funkcionality
- ak by uviedol, že zdrojom pre časť Rybky je program Fruit (autor Letouzey):
 - Letouzey by sa musel vyjadriť ako spoluautor Rybky, ktorý z jeho programov (Fruit alebo Rybka) má súťažiť

Rybka - spor

- obrana:

- VR tvrdil, že je úplne nevinný,
- obvinil vyšetrovanie, že nebolo nestranné
- odmietol spolupracovať pri vyšetrovaní aj brániť sa.
- verejne prehlásil iba:
 - *Rybka neobsahuje časť programu, ktorá rieši ako hrať (game-playing code), ktorú by napísal niekto iný okrem štandardných výnimiek, ktoré sa nedajú chápať ako programovanie hry.*

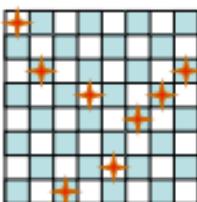
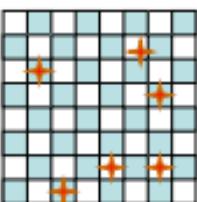
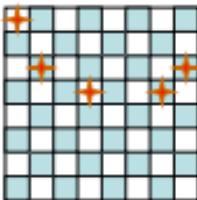
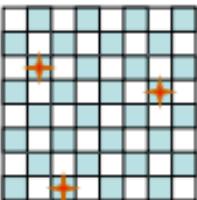
- Søren Riis:

- implementácia podobných koncepcíí a algoritmov vyhodnocovania v šachovom programe povedie spravidla ku podobnostiam v zdrojových programoch aj ak sa zdrojové časti nekopírovali. Rybka implementuje koncepcie a algoritmy, získané z programu Fruit.

šach

- v súčasnosti (2012) sa za najlepší šachový program považuje Houdini (autor Robert Houdart)
- Rybka sa predáva...

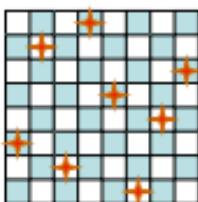
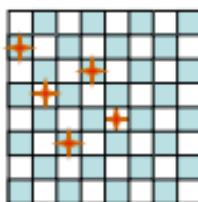
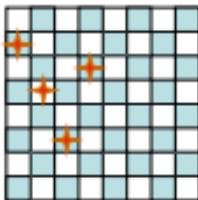
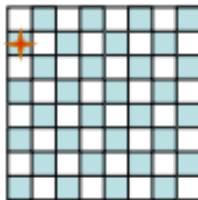
Hračkové problémy – 8 dám (1.formulácia)



- **Stavy:** všetky možné konfigurácie ľubovoľného možného (0-8) počtu dám na šachovnici
- **Počiatočný stav:** žiadna dáma na šachovnici
- **Operátory:** položenie dámy na ľubovoľné políčko šachovnice
- **Cena cesty:** 0
- **Cieľový test:** 8 dám na šachovnici umiestnených tak, že sa navzájom neohrozujú

→ $64 \times 63 \times \dots \times 57 \sim 3 \times 10^{14}$ stavov

Hračkové problémy – 8 dám (2. formulácia)



- **Stavy:** všetky možné konfigurácie ľubovoľného možného (0-8) počtu dám na šachovnici také, že ani jedna z dám nie je ohrozená
- **Počiatočný stav:** žiadna dáma na šachovnici
- **Operátory:** položenie dámy na ľubovoľné políčko v najľavejšom prázdnom stĺpci také, že ju na ňom neohrozuje žiadna iná dáma
- **Cena cesty:** 0
- **Cieľový test:** 8 dám na šachovnici umiestnených tak, že sa navzájom neohrozujú

→ 2057 stavov

Hračkové problémy – kryptografické problémy

$$\begin{array}{r} \text{SEND} \\ + \text{MORE} \\ \hline \text{MONEY} \end{array} \quad \begin{array}{r} 9567 \\ +1085 \\ \hline 10652 \end{array}$$

- **stavy:** všetky možné zápisy hlavolamu (zápis výrazu a hodnoty), v ktorých sa vyskytujú písmená a/alebo číslice
- **začiatočný stav:** hlavolam (zápis výrazu a hodnoty), v ktorom sa vyskytujú iba písmená
- **operátory:** náhrada všetkých výskytov nejakého písmena za číslicu, ktorá sa ešte medzi číslicami v hlavolame nevyskytuje
- **cieľový test:** hlavolam obsahuje iba číslice a je aritmeticky správny
- **cena cesty:** 0, pretože všetky riešenia sú rovnako dobré

Charakteristiky problémov

- riešením problému je stav alebo cesta
- problém rozložiteľný na samostatne riešiteľné podproblémy
- problémy s ignorovateľnými krokmi riešenia
- problémy s odčiniteľnými krokmi riešenia
- problémy s neodčiniteľnými krokmi riešenia

Hľadanie riešenia

Hľadanie riešenia je prístup k riešeniu problémov, pri ktorom nevychádzame z algoritmu riešenia problému.

Budť ho nepoznáme (možno preto, že ani neexistuje), alebo ho poznáme, ale pre svoju neefektívnosť je prakticky nepoužiteľný. Namiesto toho vychádzame z algoritmu, ako riešenie hľadať.

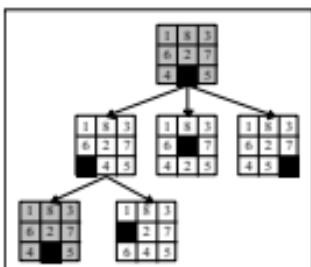
Hľadanie riešenia - algoritmus

```
function VŠEOBECNÉ-HĽADANIE(problém, stratégia)
    returns riešenie alebo neúspech

    inicializuj strom hľadania použitím začiatočného stavu z problém
    loop do
        if nie sú nerozvité uzly then return neúspech
        vyber list za uzol na rozvitie podľa stratégia
        if uzol predstavuje cieľový stav then return zodpovedajúce riešenie
        else rozví uzel a pripíš vygenerované uzly do stromu hľadania
    end
```

Stavový priestor a graf (strom) hľadania

- Reprezentácia uzla:

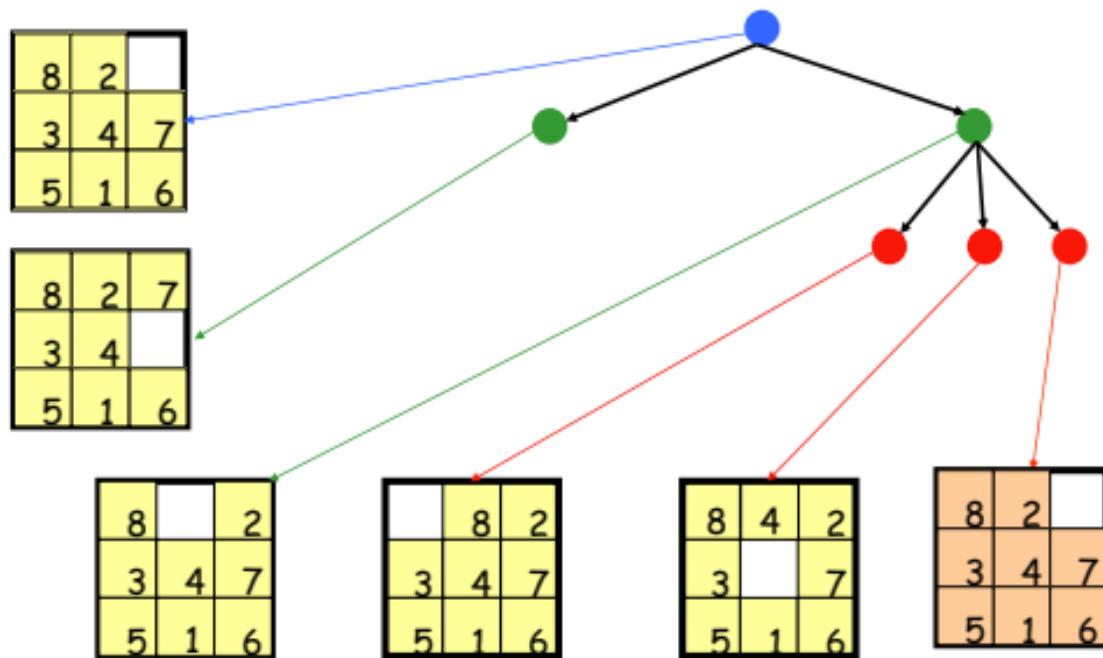


- zodpovedajúci stav zo stavového priestoru,
- uzol v strome hľadania, z ktorého sa daný uzol vygeneroval,
- operátor, ktorý sa aplikoval pri generovaní uzla (rodičovský uzol),
- počet uzlov na ceste z koreňa do daného uzla (hĺbka uzla),
- cena cesty zo začiatočného uzla do daného uzla.

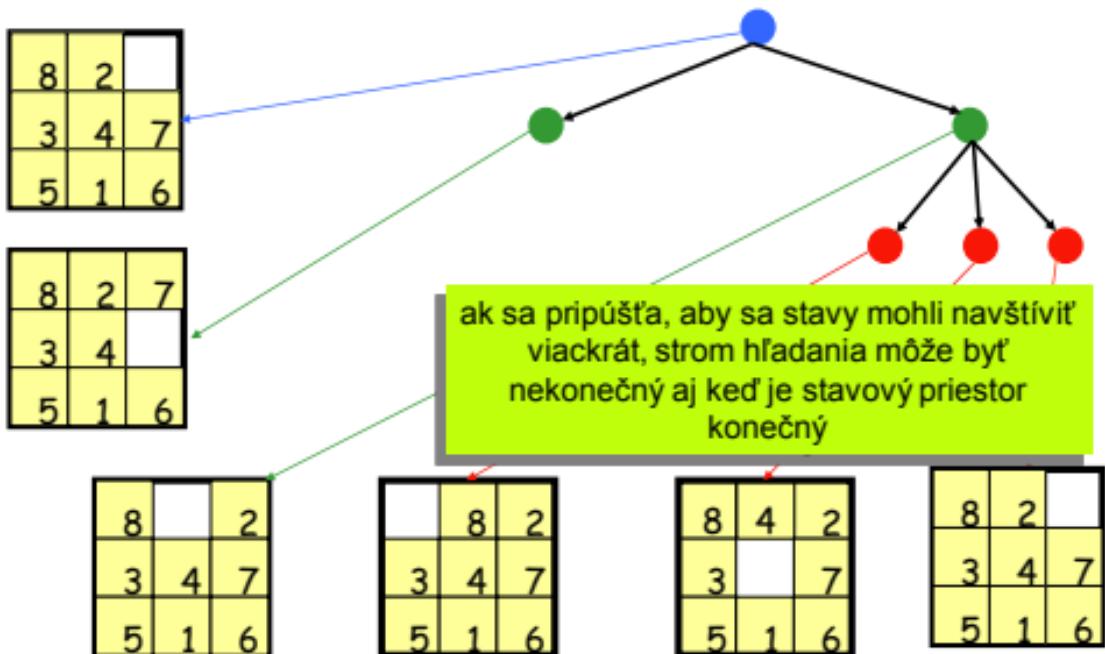
datatype UZOL

components: STAV, RODIČOVSKÝ-UZOL, OPERÁTOR,
HĽBKA, CENA-CESTY

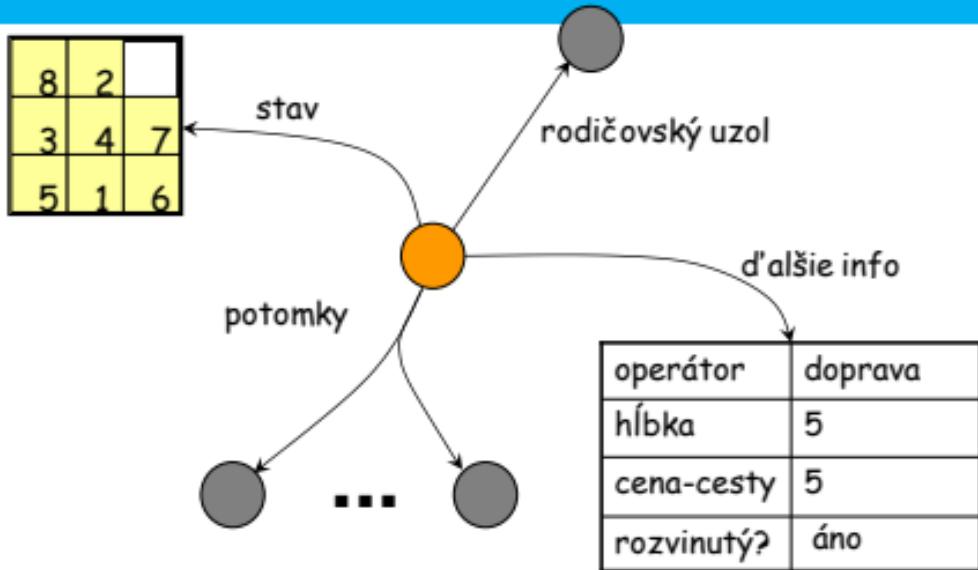
Uzly v strome hľadania ≠ stavy



Uzly v strome hľadania \neq stavy



dátová štruktúra pre uzol



hĺbka uzla N
= dĺžka cesty z koreňa do N
(hĺbka koreňa = 0)

rozvinutie uzla

rozvinutie uzla N v strome hľadania
pozostáva z:

- 1) vyhodnotenia funkcie nasledovníka
na $\text{STAV}(N)$
- 2) vygenerovania
potomka/nasledovníka uzla N pre
každý stav, ktorý vráti funkcia
nasledovníka

generovanie uzla \neq rozvinutie uzla

8	2
3	4
5	1



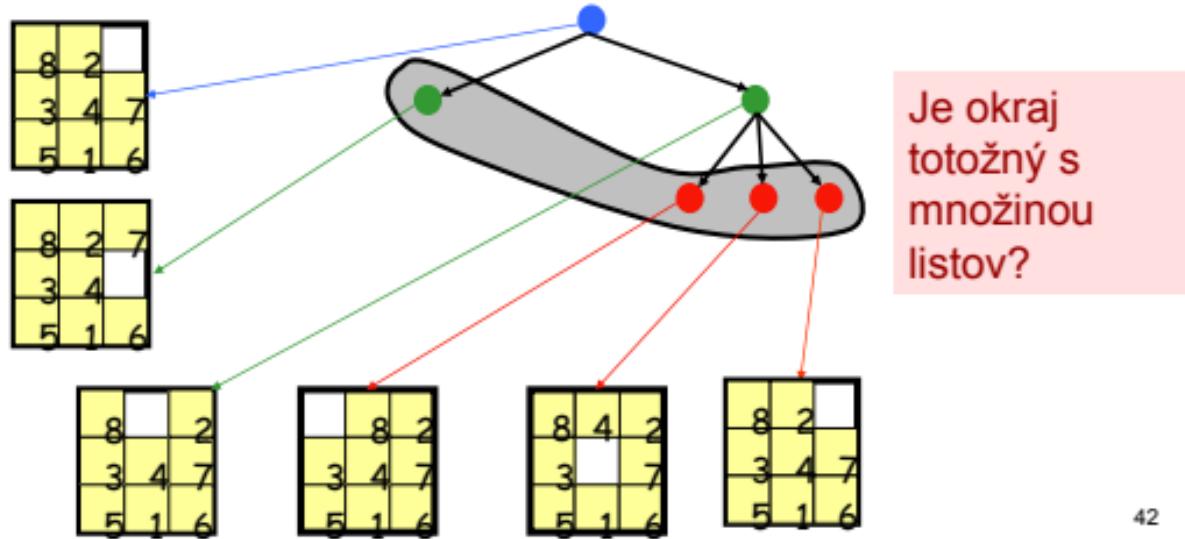
	8	2
3	4	7
5	1	6

8	4	2
3		7
5	1	6

8	2	
3	4	7
5	1	6

Okraj stromu hľadania

- okraj je množina všetkých uzlov (v strome hľadania), ktoré ešte nie sú rozvinuté



Front – štruktúra na zápis množiny uzlov

- Nad frontom definujeme tieto operácie:
 - VYTVOR-FRONT(*prvky*) vytvorí front s danými prvkami;
 - PRÁZDNY(*front*) vráti true práve vtedy, ak front neobsahuje žiadne prvky;
 - VYBER(*front*) odstráni prvok z frontu a vráti ho;
 - ZARAĎ-DO-FRONTU(*prvky, front*) vráti front po zaradení prvkov do pôvodného frontu. Rôzne druhy tejto funkcie určujú rôzne algoritmy hľadania.

Všeobecný algoritmus hľadania

```
function VŠEOBECNÉ-HĽADANIE(problém, ZARAĎ-DO-FRONTU)
    returns riešenie alebo neúspech
    static: front, front obsahujúci vygenerované a nerozvité uzly,
            na začiatku prázdny
            uzol, uzol stromu hľadania

    front ← VYTVOR-FRONT(VYTVOR-UZOL(ZAČIATOČNÝ-STAV[problém]))
    loop do
        if front je prázdny then return neúspech
        uzol ← VYBER(front)
        if CIELOVÝ-TEST[problém] aplikovaný na STAV(uzol) je úspešný
            then return VYBER-RIEŠENIE(uzol)
        front ← ZARAĎ-DO-FRONTU(ROZVI(uzol, OPERÁTORY[problém]), front)
    end
```

- **Neinformované (slepé)**

- Nemajú k dispozícii nejakú doplňujúcu informáciu o probléme
- Poradie generovania stavov závisí iba od informácií získaných hľadaním a nie je ovplyvnené ani nepreskúmanou časťou grafu ani vlastnosťami cieľového stavu.

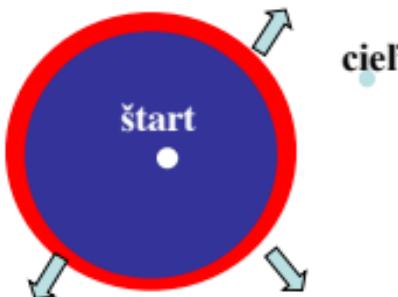
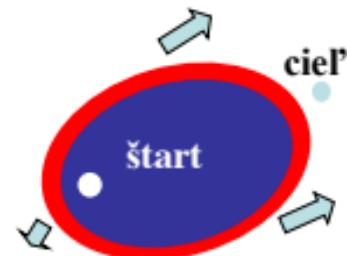
- **Informované (heuristické)**

- Majú k dispozícii nejakú doplňujúcu informáciu o probléme
- Heuristická informácia sa často využíva na to, aby sa zvýšila efektívnosť hľadania (t.j. znížila časová a/alebo pamäťová zložitosť) aj za cenu, že nebude dodržané ďalšie kritérium, a sice prípustnosť a/alebo úplnosť

heuristika

- ηὕρηκα [heuréka] = našiel (objavil) som to – Archimedes
- ηύπισκω = nájsť, objaviť
- spôsob riešenia problému, pre ktorý nemáme algoritmus alebo presný postup ⇒ heuristiké riešenie problémov
- Polya: Ako to vyriešiť. 1954:
 - ak nerozumiete riešenému problému, skúste si ho nakresliť
 - ak neviete nájsť riešenie, predstavte si, že ho máte a pozrite sa, či z neho neviete odvodiť postup (pracovať odzadu)
 - ak je problém abstraktný, skúste najprv riešiť konkrétny príklad
 - skúste najprv riešiť všeobecnejší problém (paradox vynálezcu: ambicioznejší plán môže mať lepšie výhliadky na jeho vyriešenie)
- heuristika v informatike: postup, ktorý zvyčajne vedie k dobrému riešeniu, avšak nezaručuje, že sa nájde najlepšie riešenie, ani že sa nájde v krátkom čase, ani že sa vôbec nájde.

Stratégie hľadania

Neinformované hľadanie	Informované hľadanie
	

- **Úplnosť**

Zaručuje hľadanie s danou stratégiou, že sa nájde riešenie, ak existuje?

- **Časová zložitosť**

Ako dlho trvá, kým sa nájde riešenie?

- **Pamäťová zložitosť**

Koľko pamäti treba na vykonanie hľadania?

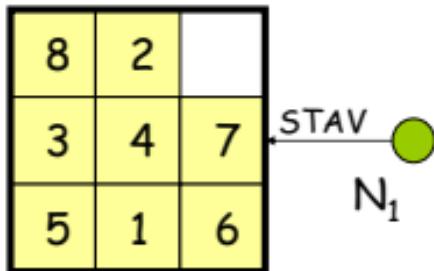
- **Prípustnosť**

Nájde sa pomocou danej stratégie najlepšie riešenie, ak existuje aspoň jedno riešenie?

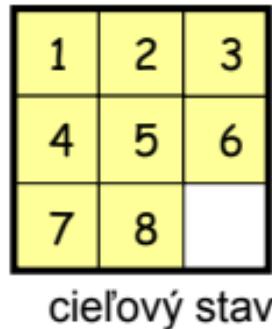
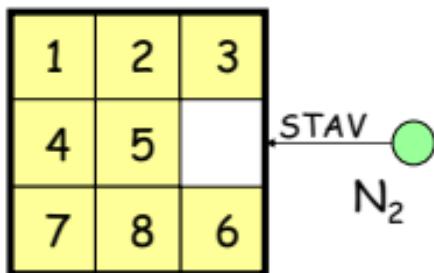
Neinformované a informované hľadanie

- slepé (neinformované) stratégie nevyužívajú opisy stavov na to, aby usporiadali okraj. využívajú iba polohu uzlov v strome hľadania.
- heuristické (informované) stratégie využívajú opisy stavov na to, aby usporiadali okraj. najsľubnejšie uzly sa umiestnia na začiatok okraja.

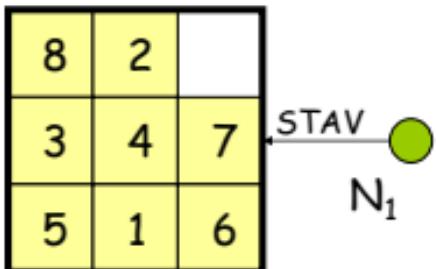
príklad



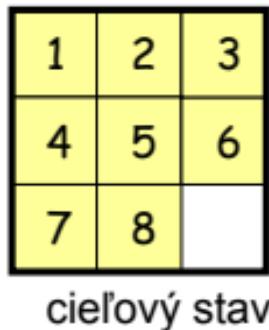
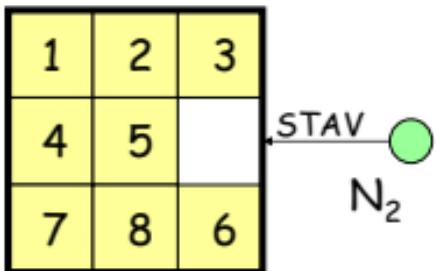
pre slepú stratégiu, N_1 a N_2 sú len dva uzly (s nejakou polohou v strome hľadania)



príklad



pre heuristickú stratégiu, počítajúc počet kameňov, ktoré nie sú na svojom mieste, N₂ je sľubnejší uzol než N₁



poznámka

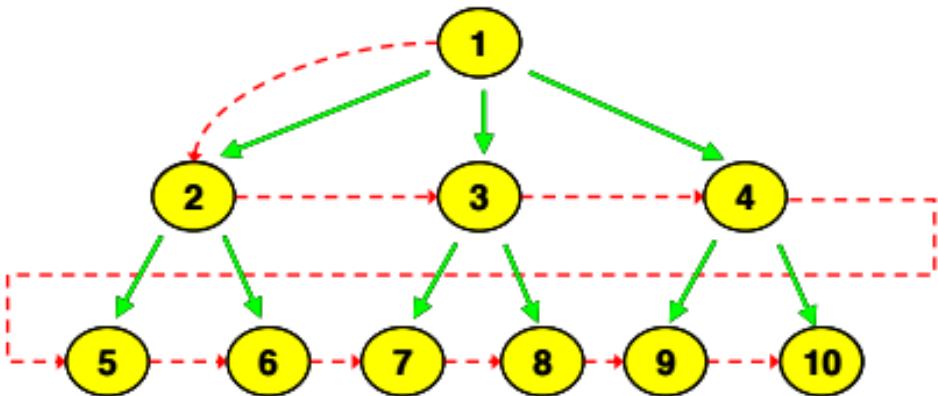
- problémy, ktoré uvažujeme, ako napr. (n^2-1) -hlavolam, sú NP-ťažké
 - neočakávajme, že budeme vedieť vyriešiť ľubovoľnú (t.j. každú) inštanciu takého problému v čase lepšom než exponenciálnom
 - môžeme sa usilovať vyriešiť každú inštanciu čo najefektívnejšie
- to je účelom stratégie hľadania

slepé stratégie

- do šírky
 - obojsmerne
 - do hĺbky
 - obmedzené
 - iteratívne sa prehlbujúce
 - do hĺbky s návratom
 - rovnoramerná cena
(varianta do šírky)
-
- cena hrany = 1
- cena hrany
= $c(\text{operátor}) \geq \varepsilon > 0$

Hľadanie do šírky

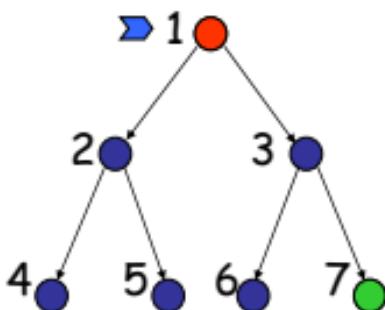
- úplné, prípustné, exponenciálna zložitosť



```
function HĽADANIE-DO-ŠÍRKY(problém) returns riešenie alebo neúspech
    return VŠEOBECNÉ-HĽADANIE(problém, ZARAĎ-NA-KONIEC)
```

Hľadanie do šírky

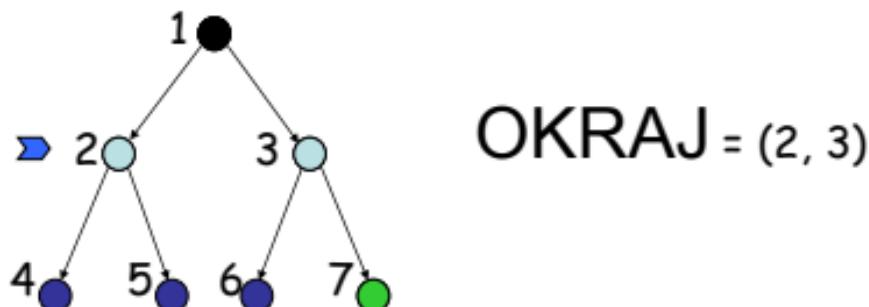
Nové uzly sa pridávajú na koniec OKRAJA



OKRAJ = (1)

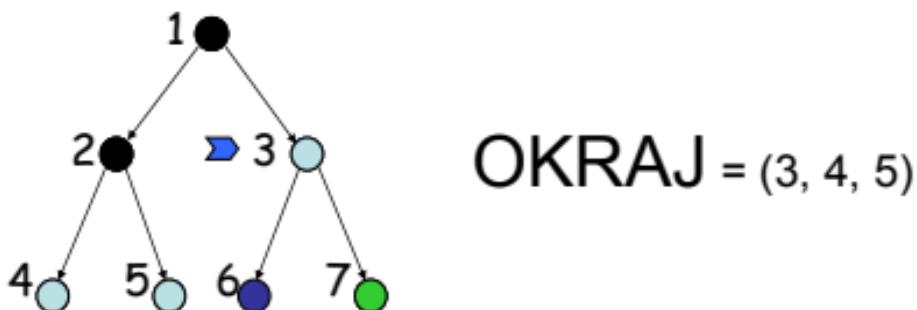
Hľadanie do šírky

Nové uzly sa pridávajú na koniec OKRAJa



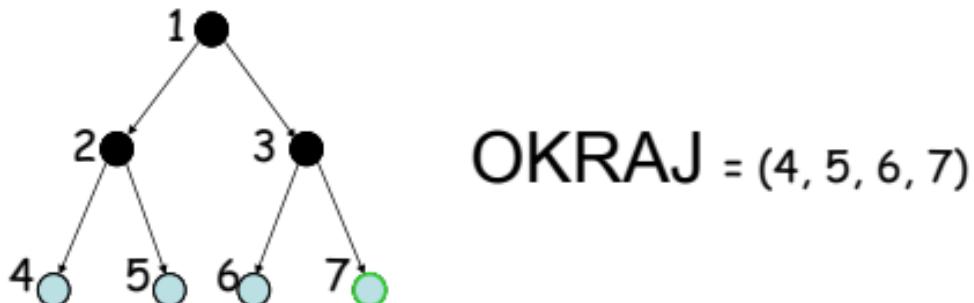
Hľadanie do šírky

Nové uzly sa pridávajú na koniec OKRAJA



Hľadanie do šírky

Nové uzly sa pridávajú na koniec OKRAJa



Dôležité parametre

- 1) Maximálny počet nasledovníkov ktoréhokoľvek stavu
→ faktor vetvenia **b** prehľadávaného stromu
- 2) Minimálna dĺžka (\neq cena) cesty medzi počiatočným a cieľovým stavom
→ hĺbka **d** najplytšieho cieľového uzla v strome

Vyhodnotenie

- **b:** Vettviaci faktor
- **d:** hĺbka najplytšieho cieľového uzla
- Hľadanie do šírky je:
 - úplné
 - optimálne, ak je krok 1
- Počet vygenerovaných uzlov
???

Vyhodnotenie

- **b**: Vetyaci faktor
- **d**: hĺbka najplytšieho cieľového uzla
- Hľadanie do šírky je:
 - úplné
 - optimálne ak je krok 1
- Počet vygenerovaných uzlov
 $1 + b + b^2 + \dots + b^d = ???$

Vyhodnotenie

- **b**: Vetyiaci faktor
- **d**: hĺbka najplytšieho cieľového uzla
- Hľadanie do šírky je:
 - úplné
 - optimálne ak je krok 1
- Počet vygenerovaných uzlov
$$1 + b + b^2 + \dots + b^d = (b^{d+1}-1)/(b-1) = O(b^d)$$
- → Časová a priestorová zložitosť je **O(b^d)**

Časové a pamäťové nároky hľadania do šírky

Hĺbka	Počet uzlov	Čas	Pamäť
0	1	0.01 milisekundy	100 slabík
1	35	0.3 milisekundy	3.4 kiloslabík
2	1225	0.01 sekundy	119 kiloslabík
3	42 875	0.4 sekundy	4 megaslabíky
4	1.5×10^6	15 sekúnd	143 megaslabík
5	52×10^6	8.7 minúty	4.8 gigaslabík
6	1.8×10^9	5 hodín	171 gigaslabík
7	64×10^9	7 dní	5.8 teraslabík
8	2.2×10^{12}	261 dní	204 teraslabík
9	78×10^{12}	25 rokov	7 168 teraslabík
10	2.7×10^{15}	874 rokov	250 888 teraslabík
12	3.3×10^{18}	10^6 rokov	8.7×10^6 teraslabík
...
20	7.6×10^{30}	2.4×10^{18} rokov	6.9×10^{20} teraslabík

Predpoklady: faktor vetvenia 35, 100000 uzlov / sekunda, 100 slabík / uzol

Poznámka

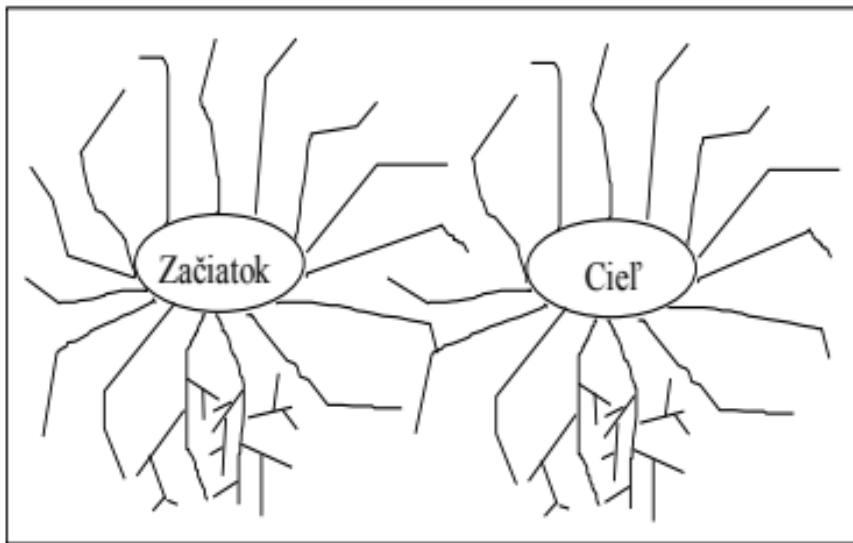
Ak problém nemá riešenie, hľadanie do šírky sa môže vykonávať *donekonečna* (ak stavový priestor je nekonečný alebo stavy môžu byť znova navštívené ľubovoľný počet ráz)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	



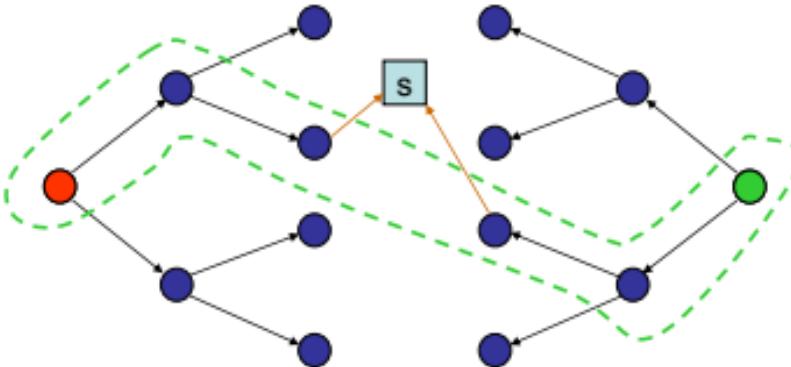
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Obojsmerné hľadanie



Obojsmerná stratégia

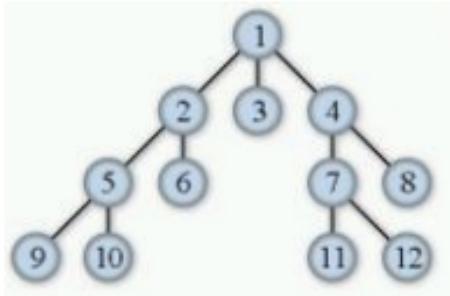
fronty dvoch okrajov: OKRAJ1 a OKRAJ2



Časová a priestorová zložitosť je $O(b^{d/2}) \ll O(b^d)$
ak oba stromy majú rovnaký vetviaci faktor b

Otázka: Čo sa stane ak vetviaci faktor je rôzny
od každého smeru?

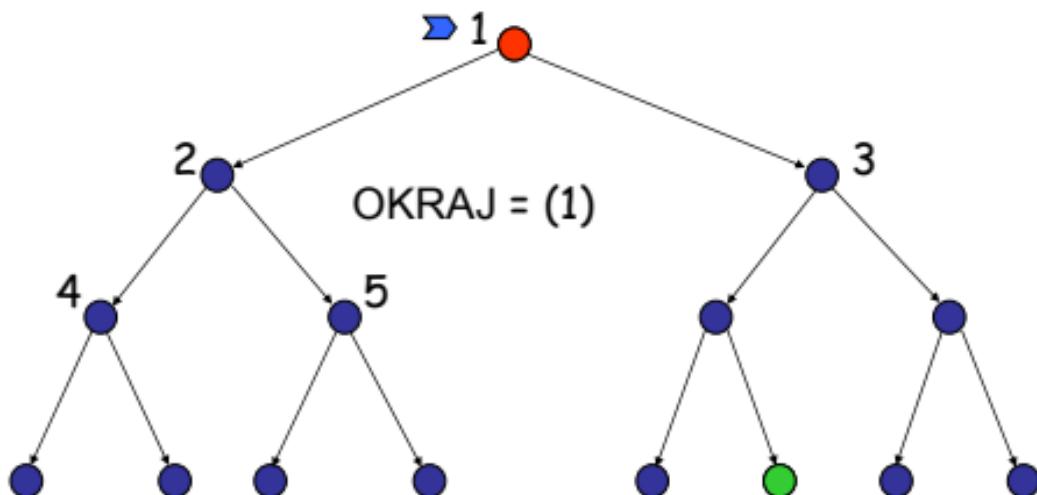
Hľadanie do hĺbky



```
function HĽADANIE-DO-HĽBKY(problém) returns riešenie alebo neúspech
    return VŠEOBECNÉ-HĽADANIE(problém, ZARAĎ-NA-ZAČIATOK)
```

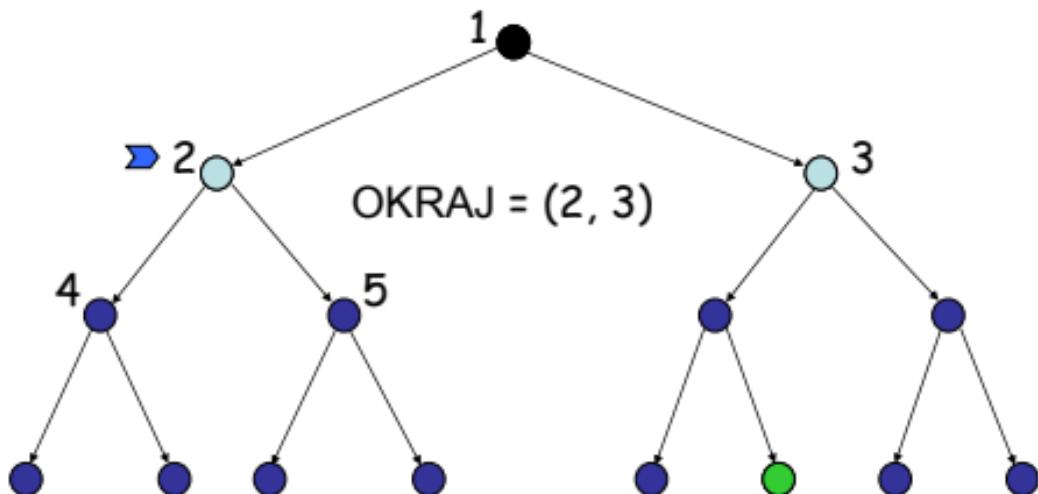
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



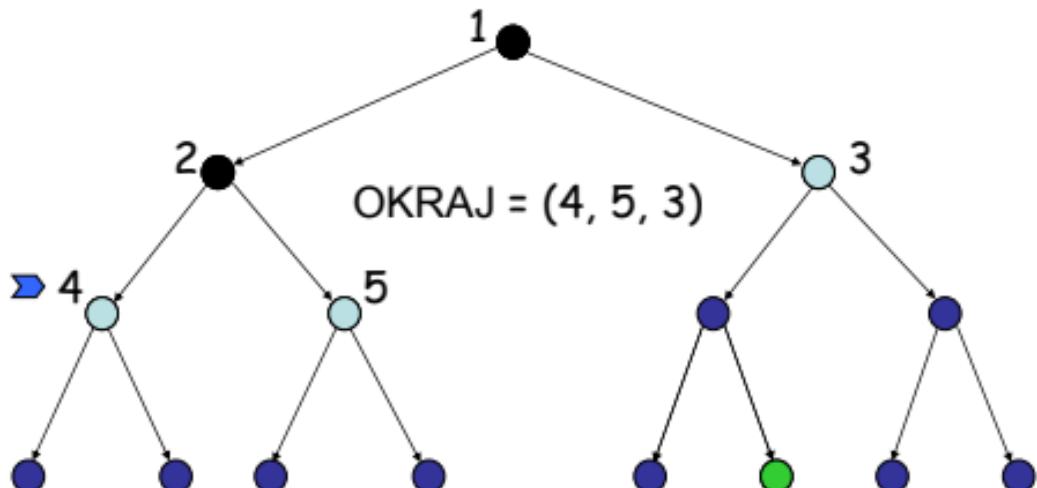
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



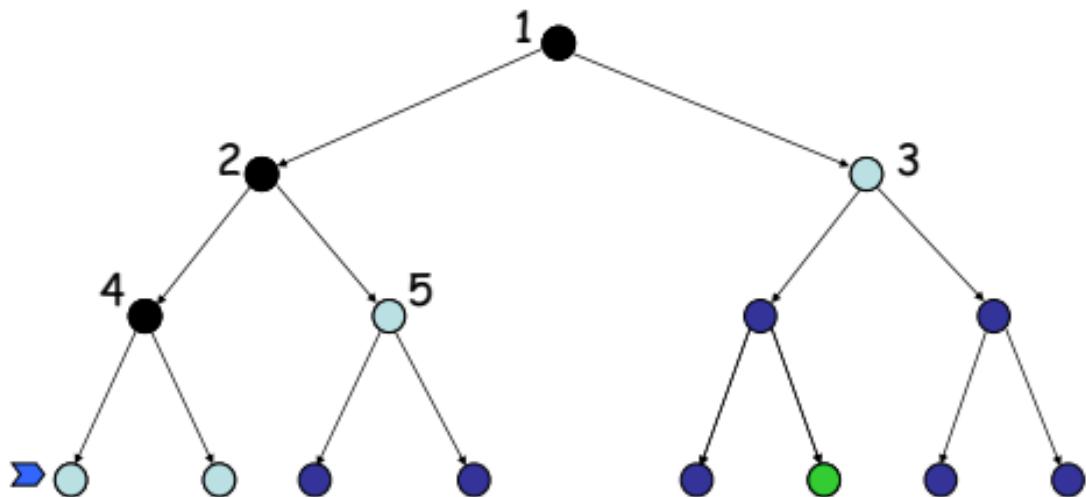
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



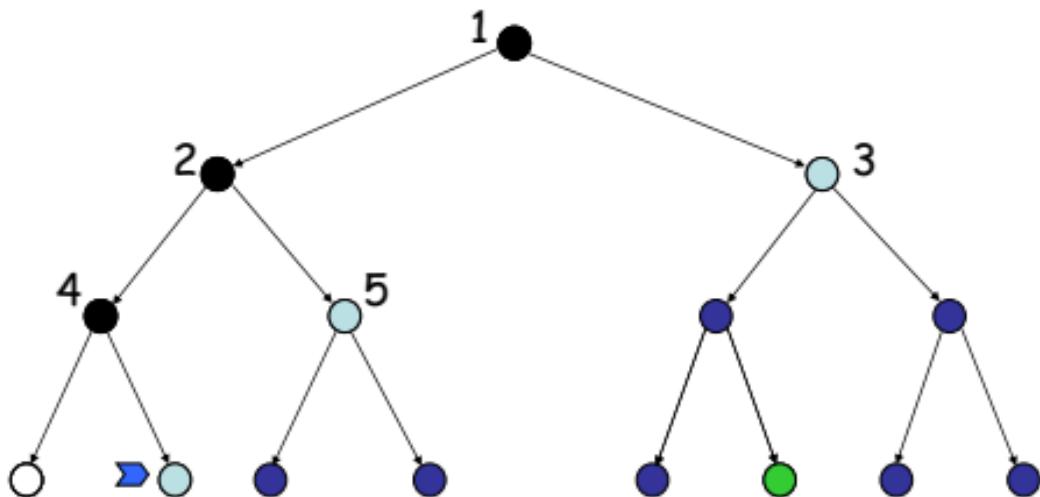
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



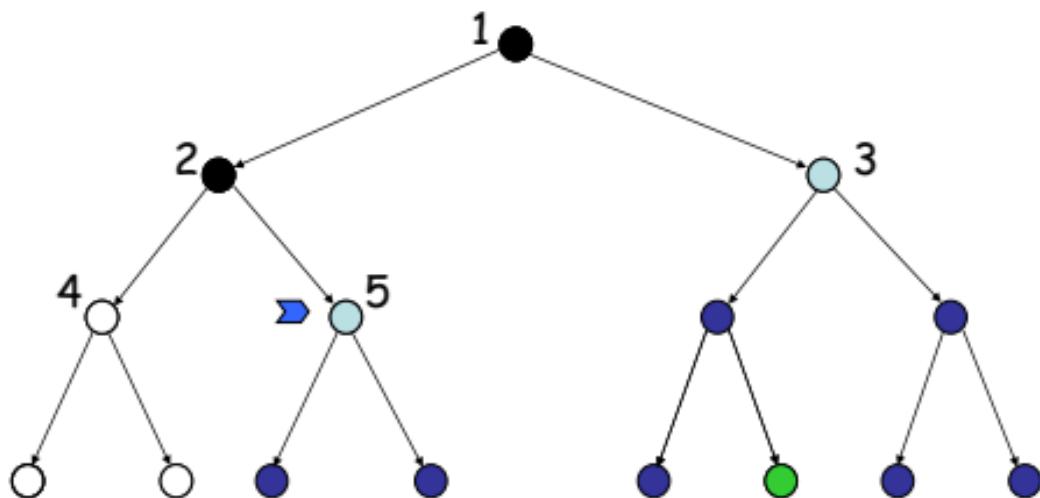
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



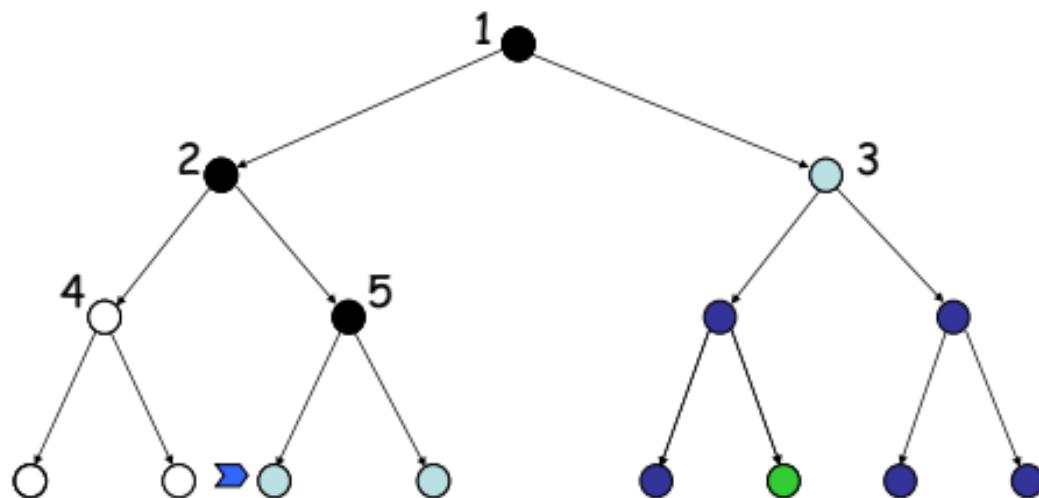
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJA



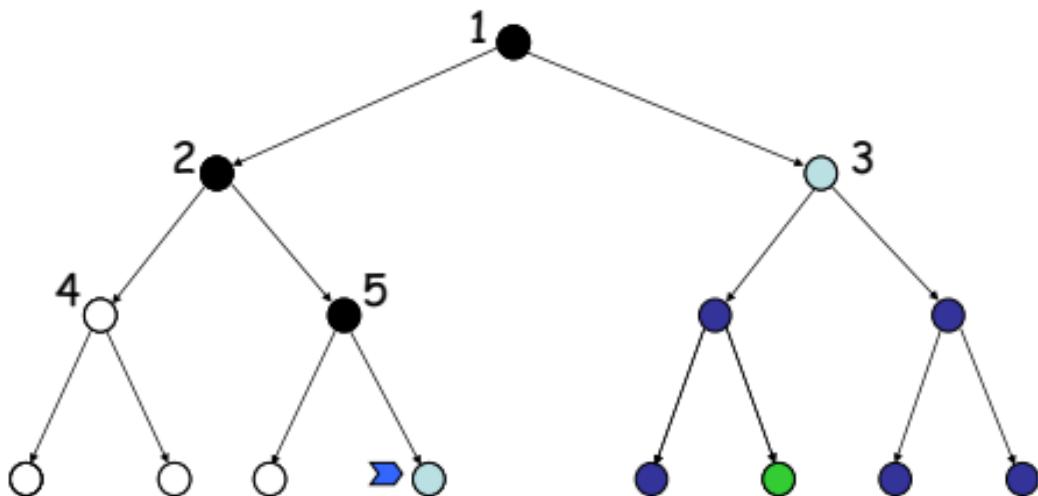
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



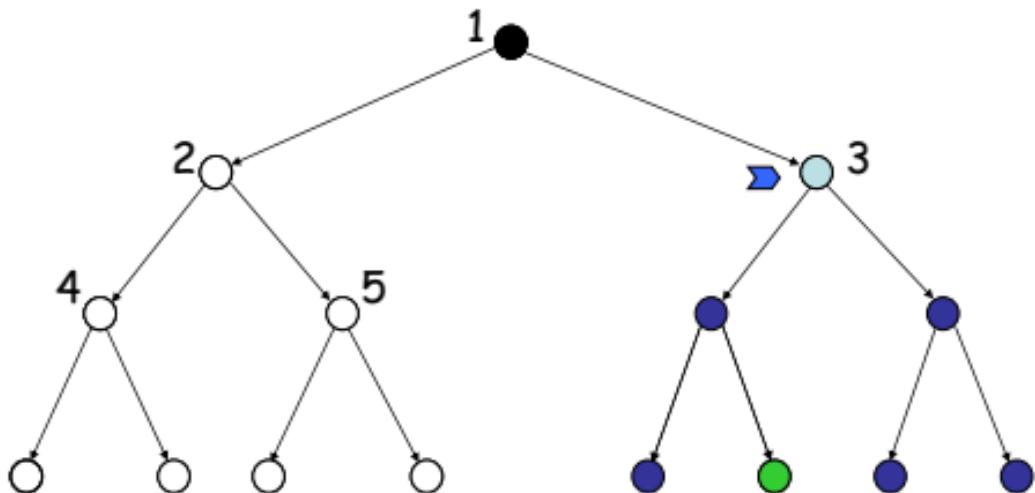
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJA



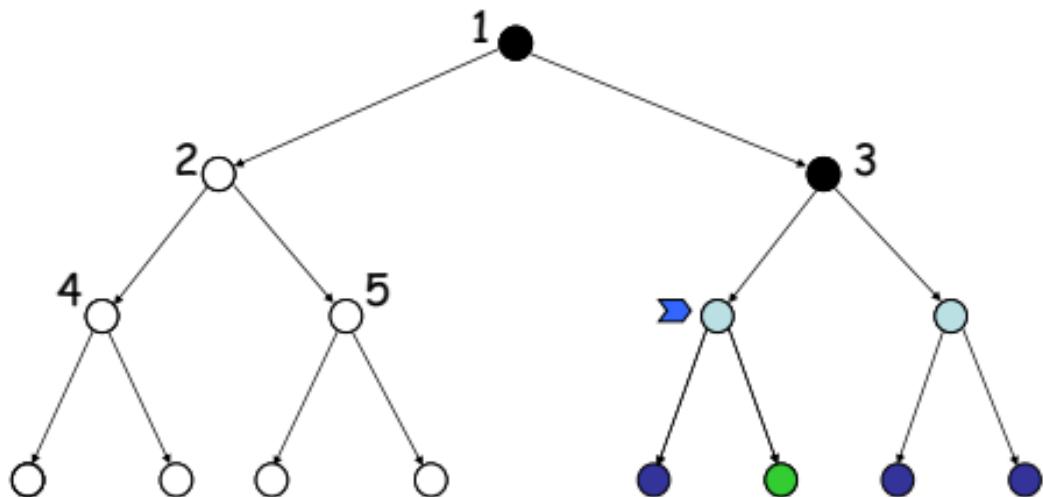
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJA



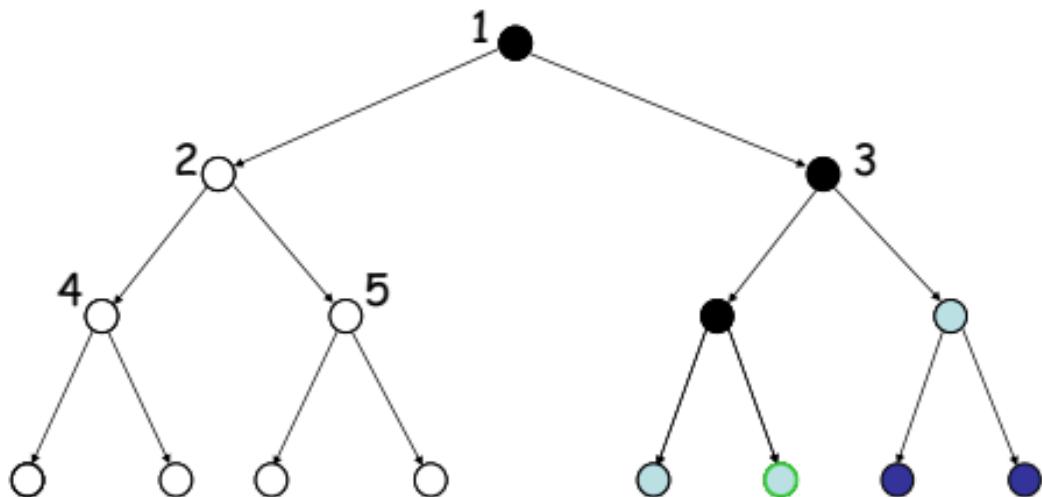
Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



Hľadanie do hĺbky

Nové uzly sa vkladajú na začiatok OKRAJa



Vyhodnotenie

- **b**: vetviaci faktor
- **d**: hĺbka najplytšieho cieľového uzla
- **m**: maximálna hĺbka listového uzla
- Hľadanie do hľbky je:
 - úplné?
 - optimálne?

Vyhodnotenie

- **b**: vetviaci faktor
 - **d**: hĺbka najplytšieho cieľového uzla
 - **m**: maximálna hĺbka listového uzla
 - Hľadanie do hľbky je:
 - úplné iba pre konečný strom hľadania
 - nie je optimálne
 - Počet vygenerovaných uzlov (najhorší prípad) :
$$1 + b + b^2 + \dots + b^m = O(b^m)$$
 - Časová zložitosť: $O(b^m)$
 - Priestorová zložitosť: $O(bm)$ [alebo $O(m)$]
- [pripomienka: 'Vyhľadávanie do šírky vyžaduje $O(b^d)$ čas a pamäť']

Cyklicky sa prehlbujúce hľadanie

```
function CYKLICKY-SA-PREHLBUJÚCE-HĽADANIE(problém)
returns riešenie alebo neúspech

    for hĺbka  $\leftarrow 0$  to  $\infty$  do
        if OBMEDZENÉ-HĽADANIE(problém, hĺbka) je úspešné
            then return jeho riešenie
    end
    return neúspech
```

Obmedzené prehľadávanie do hĺbky

- hľadanie do hĺbky s odseknutím v hĺbke k
 - hĺbka, za ktorou sa uzly nerozvíjajú
- Tri možné prípady
 - Riešenie
 - Zlyhanie – žiadne riešenie
 - Odseknutie hĺbky – nebolo by riešenia bez odseknutia

Cyklicky sa prehlbujúce hľadanie

Poskytuje to najlepšie z hľadania do šírky a do hĺbky

Hlavná idea:

Úplne desivé !

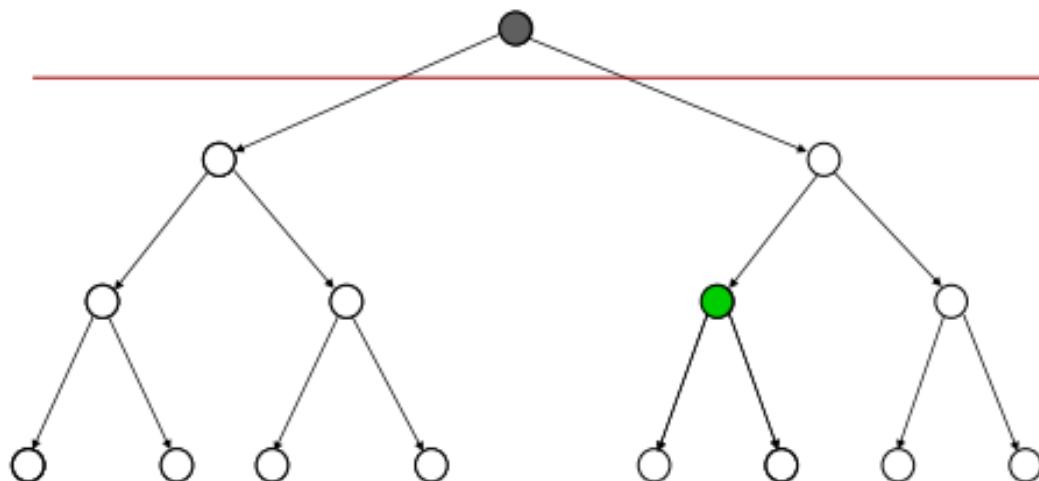
IDS

Pre $k = 0, 1, 2, \dots$ do:

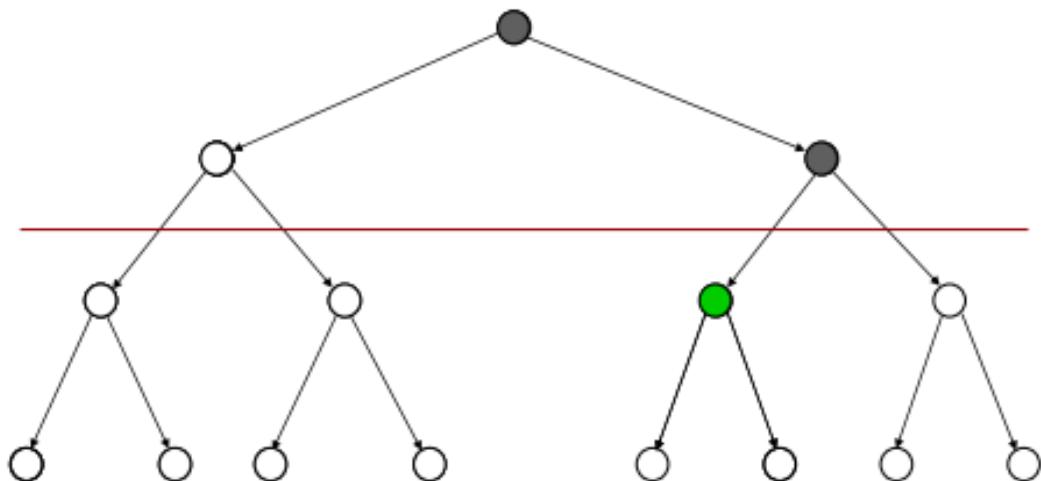
Vykonaj hľadanie do hĺbky s odseknutím v hĺbke k

(napr., generuj iba uzly s hĺbkou $\leq k$)

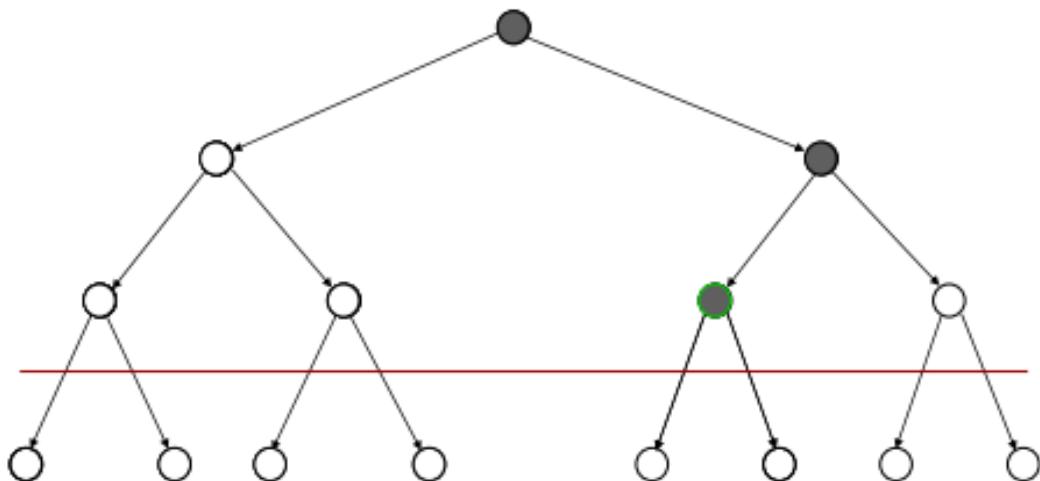
Cyklicky sa prehlbujúce hľadanie



Cyklicky sa prehlbujúce hľadanie



Cyklicky sa prehlbujúce hľadanie



Vyhodnotenie

- Cyklicky sa prehlbujúce hľadanie je:
 - úplné
 - optimálne ak cena kroku = 1
- Časová zložitosť:
$$(d+1)(1) + db + (d-1)b^2 + \dots + (1)b^d = O(b^d)$$
- Priestorová zložitosť: $O(bd)$ alebo $O(d)$

Výpočet

$$\begin{aligned} & db + (d-1)b^2 + \dots + (1) b^d \\ &= b^d + 2b^{d-1} + 3b^{d-2} + \dots + db \\ &= (1 + 2b^{-1} + 3b^{-2} + \dots + db^{-d}) \times b^d \\ &\leq (\sum_{i=1, \dots, \infty} i b^{(1-i)}) \times b^d = b^d (b/(b-1))^2 \end{aligned}$$

Počet generovaných uzlov (hľadanie do šírky a cyklické prehlbovanie)

$$d = 5 \text{ a } b = 2$$

do šírky	cykl. prehlb.
1	$1 \times 6 = 6$
2	$2 \times 5 = 10$
4	$4 \times 4 = 16$
8	$8 \times 3 = 24$
16	$16 \times 2 = 32$
32	$32 \times 1 = 32$
63	120

$$120/63 \sim 2$$

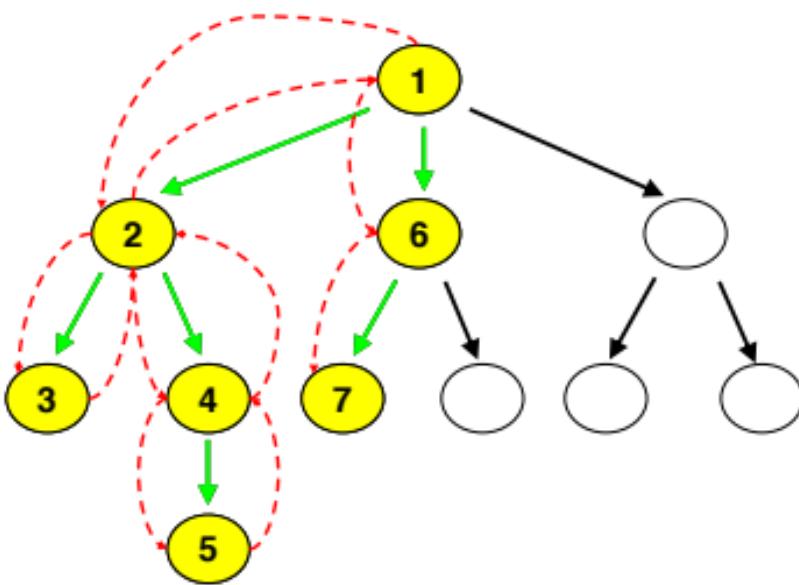
Počet generovaných uzlov (hľadanie do šírky a cyklické prehlbovanie)

$d = 5$ a $b = 10$

do šírky	cykl. prehlb.
1	6
10	50
100	400
1,000	3,000
10,000	20,000
100,000	100,000
111,111	123,456

$$123,456/111,111 \sim 1.111$$

hľadanie do hĺbky s návratom



Hľadanie do hĺbky s návratom

```
function HLADANIE-DO-HLBKY-S-NAVRATOM(problém)
    returns riešenie alebo neúspech
    static: front, front obsahujúci vygenerované a úplne nerozvité uzly,
            na začiatku prázdný
            uzol, uzol stromu hľadania

    front ← VYTVOR-FRONT(VYTVOR-UZOL(ZAČIATOČNÝ-STAV[problém]))
    loop do
        if front je prázdný then return neúspech
        uzol ← SPRÍSTUPNI-PRVÝ(front)
        if CIELOVÝ-TEST[problém] aplikovaný na STAV(uzol,front) je úspešný
            then return VYBER-RIEŠENIE(uzol)
        if uzol má ešte nepreskúmané nasledovníky then
            front ← ZARAĎ-NA-ZAČIATOK(
                ĎALŠÍ-NASLEDOVNÍK(uzol, OPERÁTORY[problém]), front)
        else VYBER(uzol, front)
    end
```

Porovnanie stratégií

- Hľadanie do šírky je úplné a optimálne, ale má vysokú pamäťovú zložitosť
- Hľadanie do hĺbky je pamäťovo efektívne, ale nie je úplné ani optimálne
- Cyklické prehlbovanie je úplné, optimálne, s rovnakou pamäťovou zložitosťou ako prehľadávanie do hĺbky a má skoro rovnakú časovú zložitosť ako prehľadávanie do šírky

Znovunavštívené stavy

Žiadne

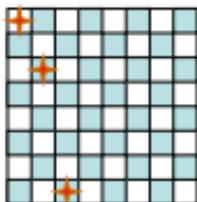
Málo

Veľa

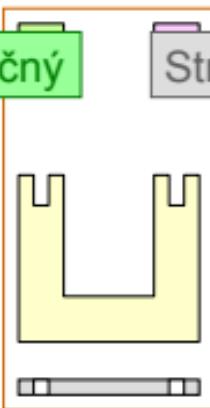
Strom je konečný

Strom je nekonečný

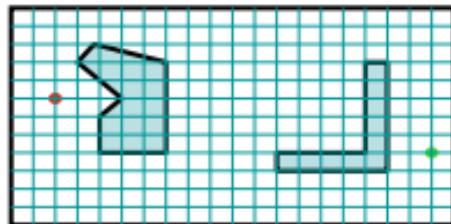
1	2	3
7	8	6



8 dám



plánovanie
montáže



8-puzľa a navigácia robota

Vyhýbanie sa znovunavštíveným stavom

- Vyžaduje porovnávanie opisov stavov
- Hľadanie do šírky:
 - Ulož všetky stavy združené s generovanými uzlami do NAVSTIVENE
 - Ak stav nového uzla je v NAVSTIVENE, tak zruš uzol

Vyhýbanie sa znovunavštíveným stavom

- Vyžaduje porovnávanie opisov stavov
- Hľadanie do šírky:
 - Ulož všetky stavy združené s generovanými uzlami do NAVSTIVENE
 - Ak stav nového uzla je v NAVSTIVENE, tak zruš uzol

Implementovať ako rozptylová tabuľka

Vyhýbanie sa znovunavštíveným stavom

- hľadanie do hĺbky:

Riešenie 1:

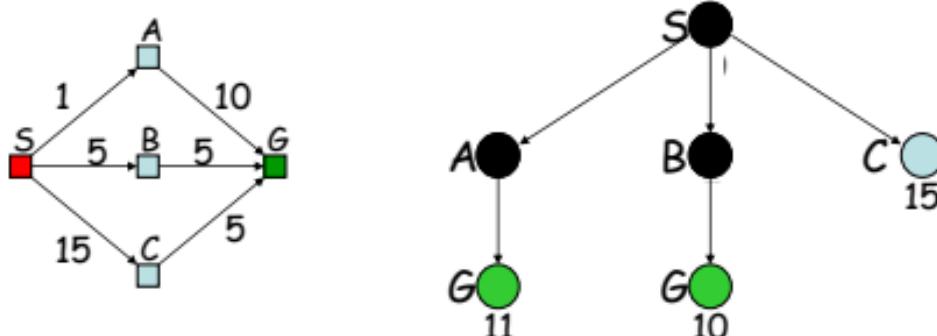
- Ukladaj všetky stavy asociované s uzlami v aktuálnej ceste do NAVSTIVENE
- Ak stav nového uzlu je v NAVSTIVENE, tak zruš uzol
- **tým sa iba vyhneme slučkám**

Riešenie 2:

- Ukladaj všetky generované stavy do NAVSTIVENE
- Ak stav nového uzlu je v NAVSTIVENE, tak zruš uzol
- **Rovnaká pamäťová zložitosť ako pri hľadaní do šírky!**

Stratégia rovnomernej ceny

- Každá hrana ma nejakú kladnú cenu $c \geq \varepsilon > 0$
- Cena cesty do ľubovoľného uzla N
 - $g(N) = \sum$ cien hrán pozdĺž cesty
- Ciel' je generovať cestu riešenia s minimálnou cenou
- Uzly N vo fronte OKRAJ sú usporiadané podľa stúpajúceho $g(N)$



- Nutnosť zmeniť algoritmus

Stratégia rovnomernej ceny

hľadanie#2

1. VLOZ(začiatočný-uzol,OKRAJ)

2. Opakuj:

a. Ak prázdny(OKRAJ) tak vráť **neúspech**

b. **N** \leftarrow VYBER(OKRAJ)

c. **s** \leftarrow STAV(**N**)

➤ d. Ak CIEL?(**s**) tak vráť **cestu alebo cieľový stav.**

e. Pre každý stav **s'** v NASLEDOVNÍKY(**s**)

i. Vytvor uzol **N'** ako nasledovník **N**

ii. VLOZ(**N'**,OKRAJ)

Cieľový test sa aplikuje na uzol vtedy, keď sa tento uzol rozvinie, nie už vtedy, keď sa generuje

Vyhýbanie sa znovunavštíveným stavom pri stratégii rovnomernej ceny

- pre ľubovoľný stav S, keď prvý uzol N taký, že $STAV(N)=S$, sa rozvinie, cesta do N je tiež najlepšou cestou z počiatočného stavu do S
- Takže:
 - Keď uzol je rozvinutý, ulož jeho stav do ZATVORENÉ
 - Keď sa vygeneruje nový uzol N:
 - Ak $STAV(N)$ je v ZATVORENÉ, zruš N
 - Ak existuje uzol N' v OKRAJi taký, že $STAV(N') = STAV(N)$, zruš uzol – N alebo N' – s najvyššou cenou cesty

Porovnanie neinformovaných stratégii hľadania

Kritérium	Do šírky	Rovnomernej ceny	Do hĺbky	Obmedzené do hĺbky	Cyklicky sa prehlbujúce	Obojsmerné
Čas	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Pamäť	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Prípustná?	áno	áno	nie	nie	áno	áno
Úplná?	áno	áno	nie	áno, ak $l \geq d$	áno	áno

- b je faktor vetvenia,
- d je hĺbka riešenia,
- m je maximálna hĺbka stromu hľadania,
- l je hraničná hĺbka (pri obmedzenom hľadaní do hĺbky)

Hľadanie najskôr najlepší

```
function HLADANIE-NAJSKÔR-NAJLEPŠÍ(problém,  
VYHODNOCOVACIA-FUNKCIA) returns riešenie alebo neúspech
```

```
ZARAĎOVACIA-FUNKCIA ← funkcia, ktorá zaradí uzly (1. arg) podľa  
hodnôt, ktoré vráti VYHODNOCOVACIA-  
FUNKCIA po aplikácii na ne do frontu  
(2. arg) tak, aby ostalo zachované  
usporiadanie
```

```
return VŠEOBECNÉ-HĽADANIE(problém, ZARAĎOVACIA-FUNKCIA)
```

- vyhodnocovacia funkcia vyjadruje iba odhad, takže na rozvitie sa vyberá uzol, ktorý sa iba zdá byť najlepším

hľadanie najlepší najsôkôr

Treba pripomenúť, že poradie stavov v

OKRAJi definuje stratégia hľadania

```
hľadanie-najskôr-najlepší(záčiatočný-s)
    OKRAJ <- vytvor-front(záčiatočný-s,
    h(záčiatočný-s))
        while (not(prázdny(OKRAJ)))
            uzol ← vyber(OKRAJ)
            s ← stav(uzol)
            if cieľový-test(s) then return s alebo cesta do
            uzol
            for each stav s' v nasledovníky(s)
                OKRAJ <- zarad'-do-frontu(s', h(s'))
            return failure
```

hľadanie najlepší najsíkôr

- Využíva opis stavov na odhadnutie ako „dobrý“ bude vyhľadávací uzol
- vyhodnocovacia funkcia f zobrazuje každý uzol N stromu hľadania na reálne číslo $f(N) \geq 0$
[Zvyčajne, $f(N)$ je očakávaná cena, takže čím menšie $f(N)$, tým je uzol N sľubnejší]
- **hľadanie najlepší najsíkôr usporadúva OKRAJ podľa stúpajúcej hodnoty f** [poradie uzlov s rovnakou hodnotou f je ľubovoľné]

hľadanie najlepší najskôr

- Využíva popis stavov na odhadnutie ako „dobrý“ bude vyhľadávací uzol
- výhodnocovacia funkcia f zobrazuje každý uzol N stromu hľadania na reálne číslo $f(N) \geq 0$
[Zvyčajne, $f(N)$ je očakávané číslo krokov, ktoré sú potrebné pre dosiahnutie cieľu z uzolu N]
- **hľadanie najlepší podľa stúpajúcej hodnotou f** je ľubovoľné

“Najlepší” nepredstavuje kvalitu generovanej cesty.

Vo všeobecnosti best-first prehľadávanie negeneruje optimálne cesty.

Ako výrobiť f?

- $f(N)$ odhaduje:
 - Bud' **cenu cesty riešenia cez N**
Potom $f(N) = g(N) + h(N)$, kde
 - $g(N)$ je cena cesty z počiatočného uzla do N
 - $h(N)$ je odhad ceny cesty z N do cieľového bodu
 - Alebo cenu cesty z N do cieľového bodu
 - Potom $f(N) = h(N)$ → lačné hľadanie
- Nie sú tu avšak žiadne limity na F. Ľubovoľná funkcia vášho výberu je akceptovateľná. Pomôže to však vyhľadávaniu?

Ako výrobiť f?

- $f(N)$ odhaduje:

- Bud' **cenu riešenej cesty cez N**

Potom $f(N) = g(N) + h(N)$, kde

- $g(N)$ je cena cesty z počiatocného uzla do N
 - $h(N)$ je odhad ceny cesty z N do cieľového bodu

- Alebo cena cesty z N do cieľového bodu

- Potom $f(N) = h(N)$

→ ~~Izbačné (nepodľahľacie) hľadanie~~

Heuristická funkcia

- Nie sú tu avšak žiadne limity na F. Ľubovoľná funkcia vášho výberu je akceptovateľná. Pomôže to však vyhľadávaniu?

Informované hľadanie

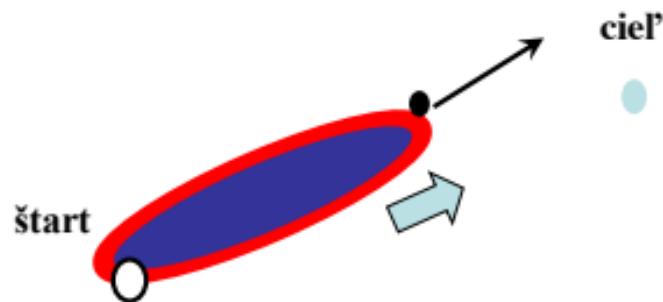
- Vyhodnocovacia funkcia musí zahŕňať aj nejaký odhad ceny cesty z daného uzla do najbližšieho cieľového uzla.
- Dva prístupy:
 - rozvitie uzla, ktorý sa zdá byť podľa vyhodnocovacej funkcie najbližšie k cieľu
 - rozvitie uzla na najlacnejšej ceste riešenia

Lačné hľadanie

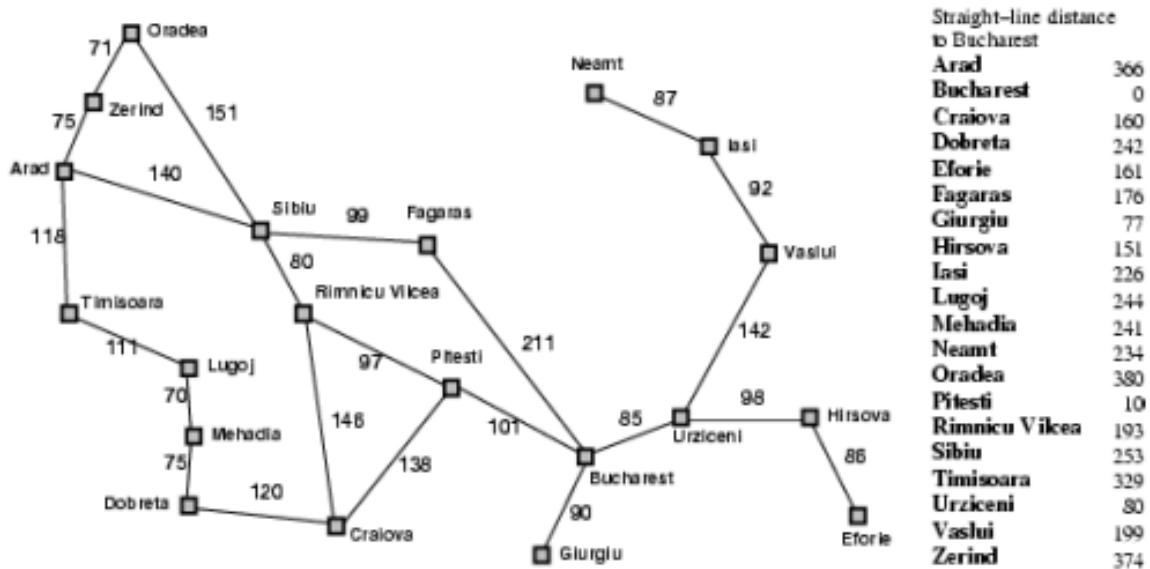
- minimalizácia odhadovanej ceny dosiahnutia cieľa
- $h(u)$ = odhad ceny najlacnejšej cesty z uzla u do cieľového uzla
- $h(c) = 0$, ak c je cieľový uzol
- Hodnotí sa cena cesty do cieľa – čím lacnejšie, tým lepšie
- Pripomína hľadanie do hĺbky
- Nie je úplné, neoptimalizuje riešenie

```
function LAČNÉ-HĽADANIE(problém, h) returns riešenie alebo neúspech
    return HĽADANIE-NAJSKÔR-NAJLEPŠÍ(problém, h)
```

Lačné hľadanie

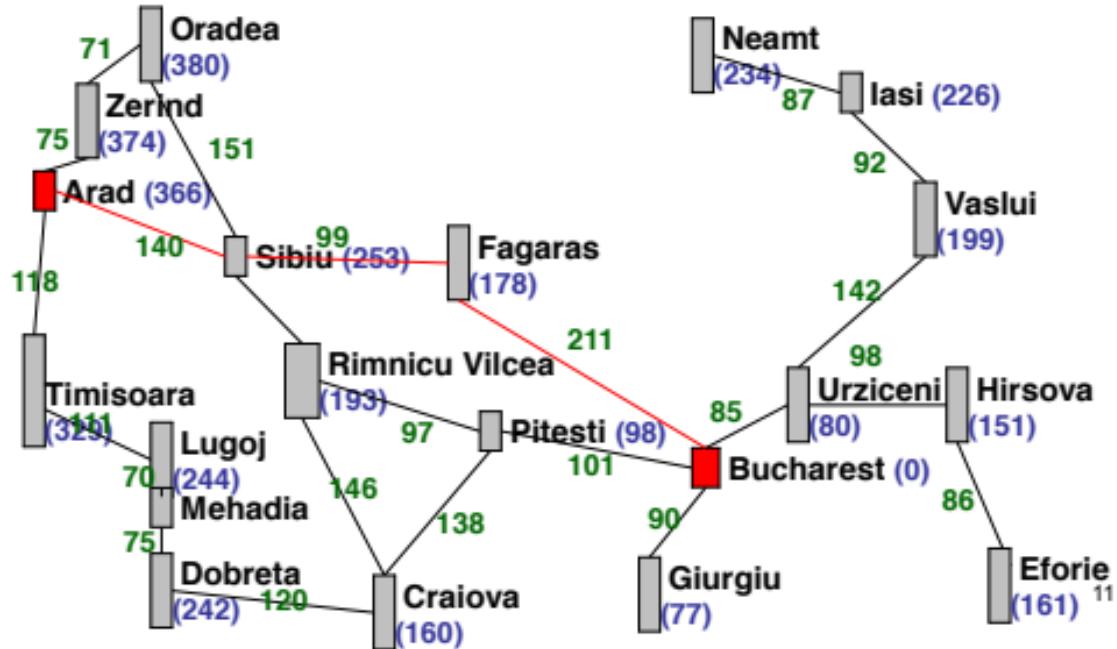


príklad: Rumunsko, cestná siet'



Heuristika: príklad Rumunsko

- cestovanie: $h(n) = \text{odhad-vzdialenosť}(n, \text{cieľ})$
- ako odhad sa použije vzdušná vzdialenosť



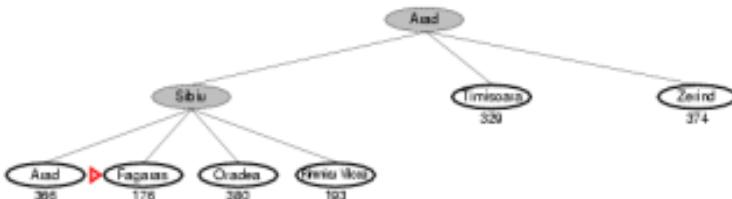
lačné hľadanie najlepší najsôkôr



Iačné hľadanie najlepší najskôr



Iačné hľadanie najlepší najskôr



Iačné hľadanie najlepší najskôr



Heuristická funkcia

- Heuristická funkcia $h(N) \geq 0$ odhaduje cenu cesty zo STAVu(N) do cieľového stavu
- Jej hodnota je nezávislá na **aktuálnom prehľadávanom strome**. Závisí iba na STAVe(N) a cieľovom teste GOAL?
- Príklad:

5		8
4	2	1
7	3	6

STAV(N)

1	2	3
4	5	6
7	8	

Cieľový stav

$h_1(N) =$ počet zle umiestnených očíslovaných doštičiek = 6

[Prečo je to odhad vzdialenosťi ku cieľu?]

Iné príklady

5		8
4	2	1
7	3	6

STAV(N)

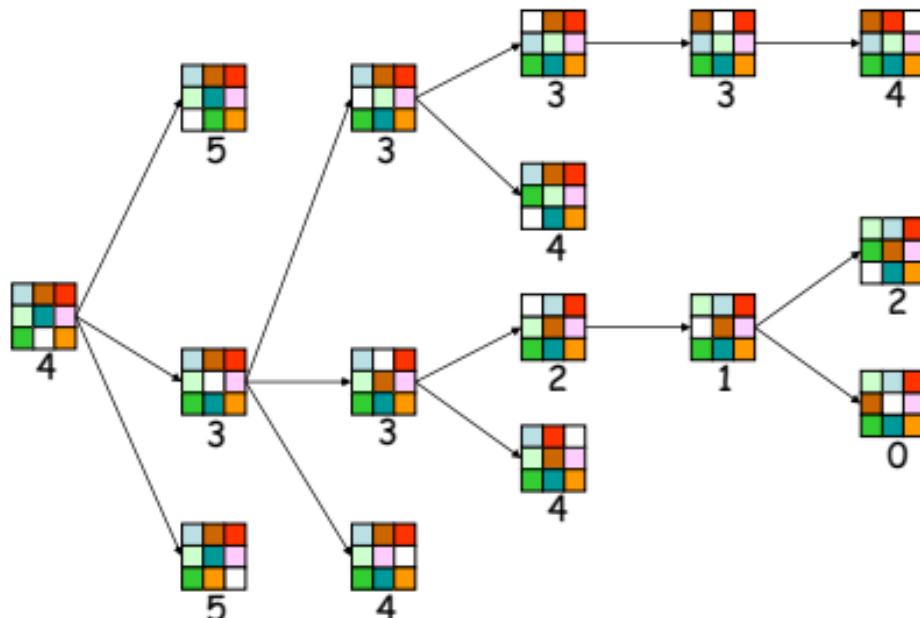
1	2	3
4	5	6
7	8	

Cieľový stav

- $h_1(N) =$ počet zle umiestnených očíslovaných doštičiek = 6
- $h_2(N) =$ súčet (manhattanských) vzdialostí každej očíslovej doštičky do jej cieľovej pozície
 $= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$
- $h_3(N) =$ súčet inverzií permutácií
 $= n_5 + n_8 + n_4 + n_2 + n_1 + n_7 + n_3 + n_6$
 $= 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0$
 $= 16$
- Ak doštička obsahujúca číslo i sa nachádza pred n doštičkami obsahujúcimi čísla menšie než i , tak dochádza k inverzii rádu $n =$ označíme n_i .

8-hlavolam

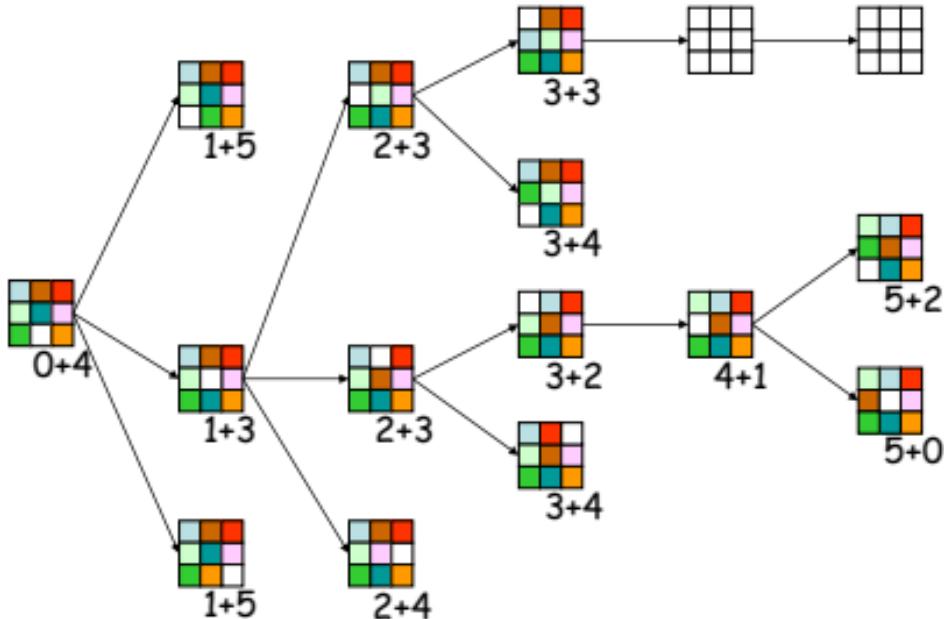
$f(N) = h(N) = \text{počet zle umiestnených ofarbených doštičiek}$



8-hlavolam

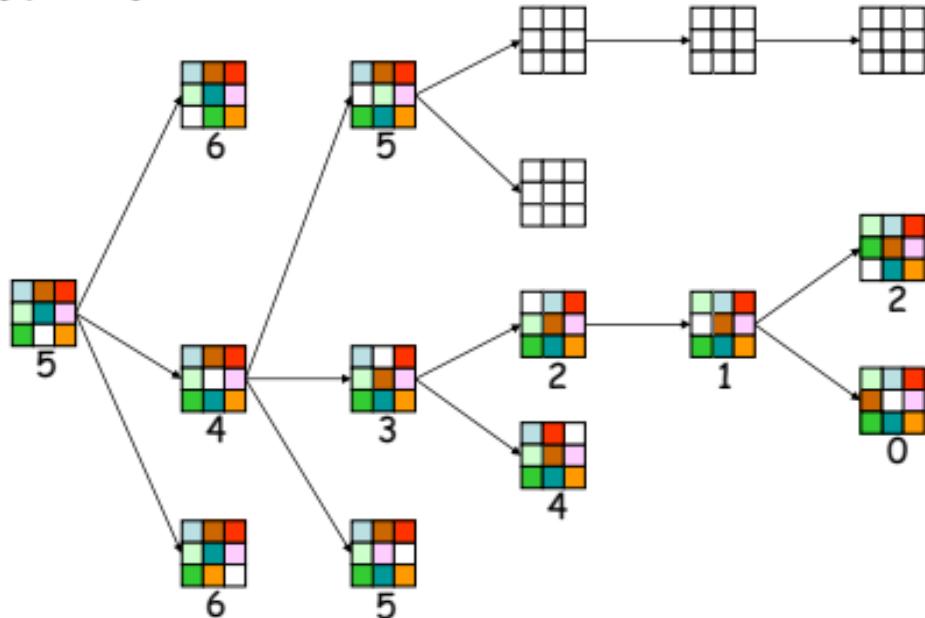
$$f(N) = g(N) + h(N)$$

kde $h(N)$ = počet zle umiestnených ofarbených doštičiek

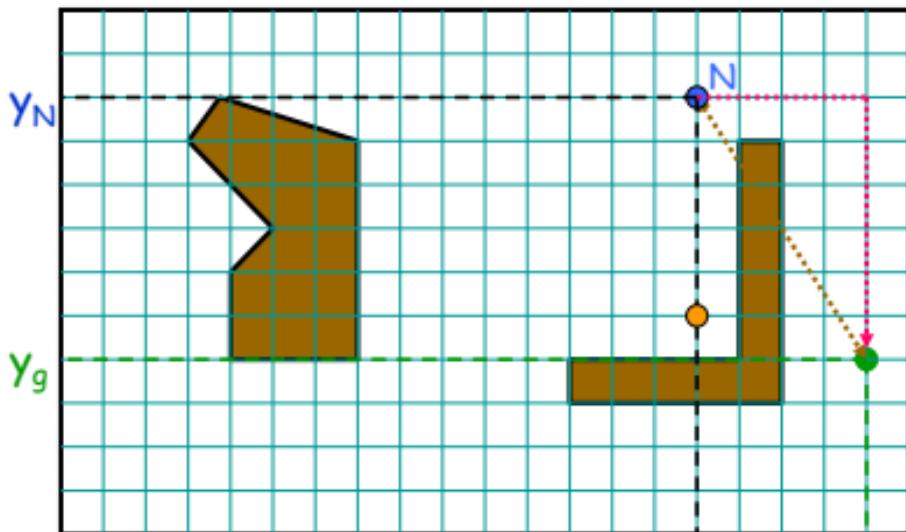


8-hlavolam

$f(N) = h(N) = \sum$ vzdialenosť ofarbených doštičiek do ich cieľovej polohy



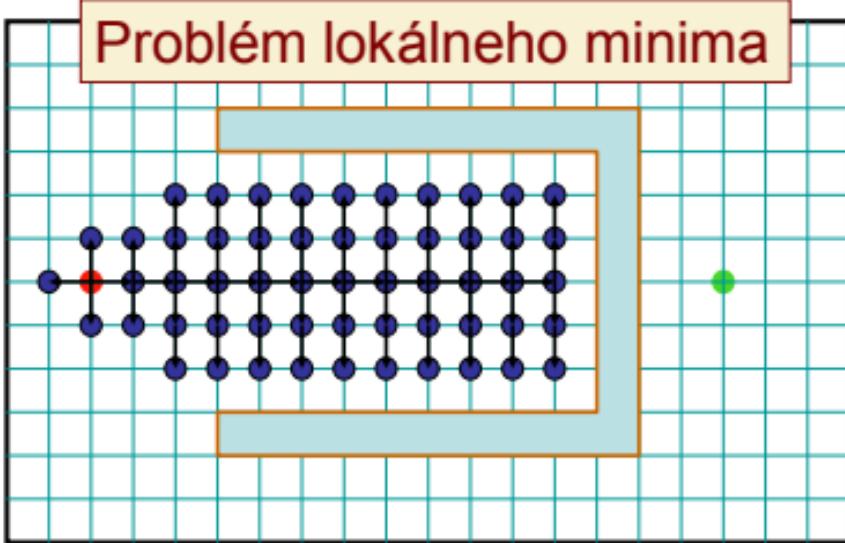
Navigácia robota



$$h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2} \quad (\text{L}_1 \text{ alebo euklidovská vzd.})$$

$$h_2(N) = |x_N - x_g| + |y_N - y_g| \quad (\text{L}_2 \text{ alebo manhattanská vzd.})$$

Best-First → Efektívnosť



$f(N) = h(N)$ = priama vzdialenosť do cieľa

Môžeme niečo dokázať?

- Ak stavový priestor je nekonečný, vo všeobecnosti hľadanie nie je úplné
- Ak stavový priestor je konečný a nezrušíme uzly, ktoré znova navštívili stavy, vo všeobecnosti hľadanie nie je úplné
- Ak stavový priestor je konečný a zrušili sme uzly, ktoré znova navštievujú stavy, hľadanie je úplné, ale vo všeobecnosti nie je optimálne

Prípustná heuristika

- Nech $h^*(N)$ je cena optimálnej cesty z N do cieľového uzla
- Heuristická funkcia $h(N)$ je prípustná ak:
$$0 \leq h(N) \leq h^*(N)$$
- Prípustná heuristická funkcia je vždy optimistická!

Prípustná heuristika

- Nech $h^*(N)$ je cena optimálnej cesty z N do cieľového uzla
- Heuristickej funkcie $h(N)$ je prípustná ak:
$$0 \leq h(N) \leq h^*(N)$$
- Prípustná heuristickej funkcie je vždy optimistická!



G je cieľový uzol $\rightarrow h(G) = 0$

Heuristiky pre 8-hlavolam

5		8
4	2	1
7	3	6

STAV(N)

1	2	3
4	5	6
7	8	

cieľový stav

- $h_1(N) = \text{počet zle uložených doštičiek} = 6$
je ???

Heuristiky pre 8-hlavolam

5		8
4	2	1
7	3	6

STAV(N)

1	2	3
4	5	6
7	8	

cieľový stav

- $h_1(N) = \text{počet zle uložených doštičiek} = 1+1+2+1+3+1+4=0+5=1+6=1+7=0+8=1=6$
je **prípustná**
- $h_2(N) = \text{súčet (manhattanských) vzdialostí každej doštičky do jej cieľa}$
 $= 1+3+2+1+3+3+4=0+5=2+6=1+7=0 = 10$
je **???**

Heuristiky pre 8-hlavolam

5		8
4	2	1
7	3	6

STAV(N)

1	2	3
4	5	6
7	8	

cielový stav

- $h_1(N)$ = počet zle uložených doštičiek = 6
je **prípustná**
- $h_2(N) = \text{súčet (manhattanských) vzdialenosí každej doštičky do jej cieľa}$
 $= 1+3+2+1+3+3+4=0+5=2+6=1+7=0 = 10$
je **prípustná**
- $h_3(N) = \text{súčet inverzií permutácií}$
 $= 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0 = 16$
je **???**

prípustnosť heuristik pre 8-hlavolam

5		8
4	2	1
7	3	6

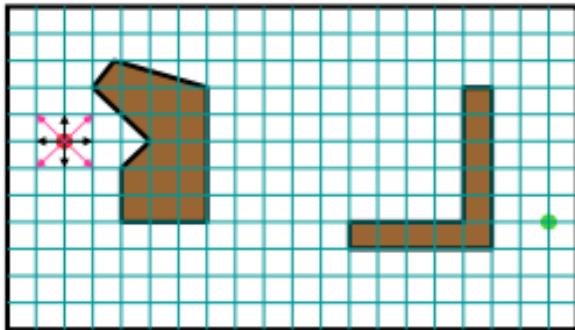
STAV(N)

1	2	3
4	5	6
7	8	

cieľový stav

- $h_1(N) =$ počet zle uložených doštičiek = 6
je **prípustná**
- $h_2(N) =$ súčet (manhattanských) vzdialenosí každej doštičky do jej cieľa
 $= 1+3+2+1+3+3+4=0+5=2+6=1+7=0 = 10$
je **prípustná**
- $h_3(N) =$ súčet inverzií permutácií
 $= 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0 = 16$
nie je prípustná

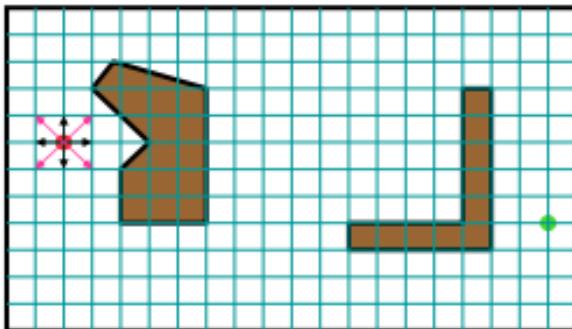
Navigácia robota



Cena horizontálneho alebo vertikálneho kroku = 1
Cena diagonálneho kroku = $\sqrt{2}$

$$h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2} \quad \text{Je prípustná}$$

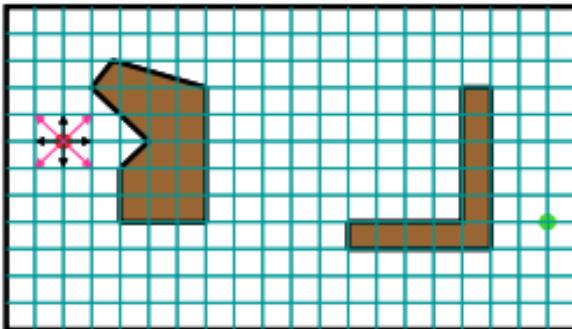
Navigácia robota



Cena horizontálneho alebo vertikálneho kroku = 1
Cena diagonálneho kroku = $\sqrt{2}$

$$h_2(N) = |x_N - x_g| + |y_N - y_g| \text{ je } ???$$

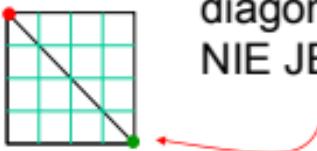
Navigácia robota



Cena horizontálneho alebo vertikálneho kroku = 1
Cena diagonálneho kroku = $\sqrt{2}$

$h_2(N) = |x_N - x_g| + |y_N - y_g|$ Je prípustná ak pohyb po diagonálach je zakázaný, inak NIE JE prípustná

$$h^*(I) = 4\sqrt{2}$$
$$h_2(I) = 8$$



Ako vytvoriť prípustnú h?

- Prípustná heuristika môže obyčajne chápaná ako cena optimálneho riešenia **voľnejšieho (všeobecnejšieho) problému** (takého, ktorý vznikne odstránením niektorých ohraničení v pôvodnom probléme)
- Príklad navigácie robota:
 - Manhattanská vzdialenosť zodpovedá tomu, že sa odstránili prekážky (*budovy atď., ulice a triedy tvoria úplnú mriežku*)
 - Euklidovská vzdialenosť zodpovedá tomu, že sa odstránili aj prekážky aj ohraničenie, že robot sa môže pohybovať iba po mriežke

príklad uvoľnenia (relaxovania) problému

- cena optimálneho riešenia voľnejšieho problému je prípustná heuristika pre pôvodný problém
- ak sa pravidlá 8-hlavolamu zvoľnia tak, že doštička sa môže presunúť hocikde, tak $h_1(n)$ určuje najkratšie riešenie
- ak sa pravidlá 8-hlavolamu zvoľnia tak, že doštička sa môže presunúť na hociktoré susedné poličko, tak $h_2(n)$ určuje najkratšie riešenie

A* hľadanie

(najpopulárnejší algoritmus umelej inteligencie)

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

$g(u)$ = cena cesty do uzla u

$h(u)$ = odhad ceny cesty z uzla u k riešeniu

$f(u)$ = odhad ceny najlacnejšej cesty riešenia vedúcej cez uzol u

- Pre všetky hrany: $c(u,u') \geq \epsilon > 0$
- Použije sa algoritmus najlepší najskôr

→ Hľadanie najskôr najlepší s vyhodnocovacou funkciou f a prípustnou heuristikou h sa nazýva **A* hľadanie**.

```
function A*-HLADANIE(problém, g, h) returns riešenie alebo neúspech
    return HLADANIE-NAJSKÔR-NAJLEPŠÍ(problém, g+h)
```

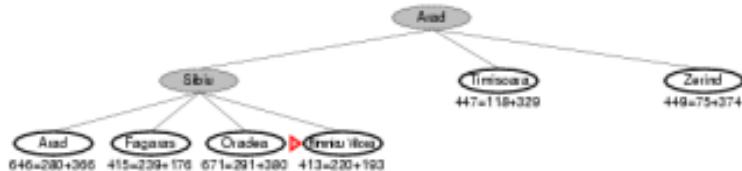
príklad A* hľadania

► Arad
365=0+365

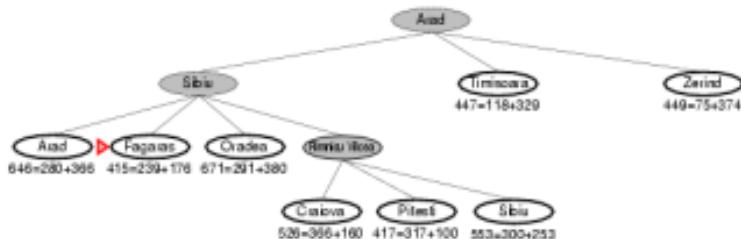
príklad A* hľadania



príklad A* hľadania



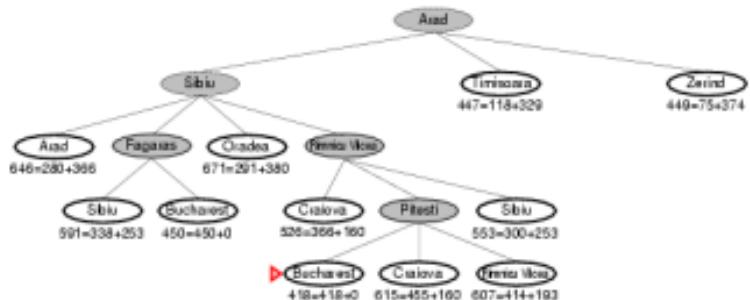
príklad A* hľadania



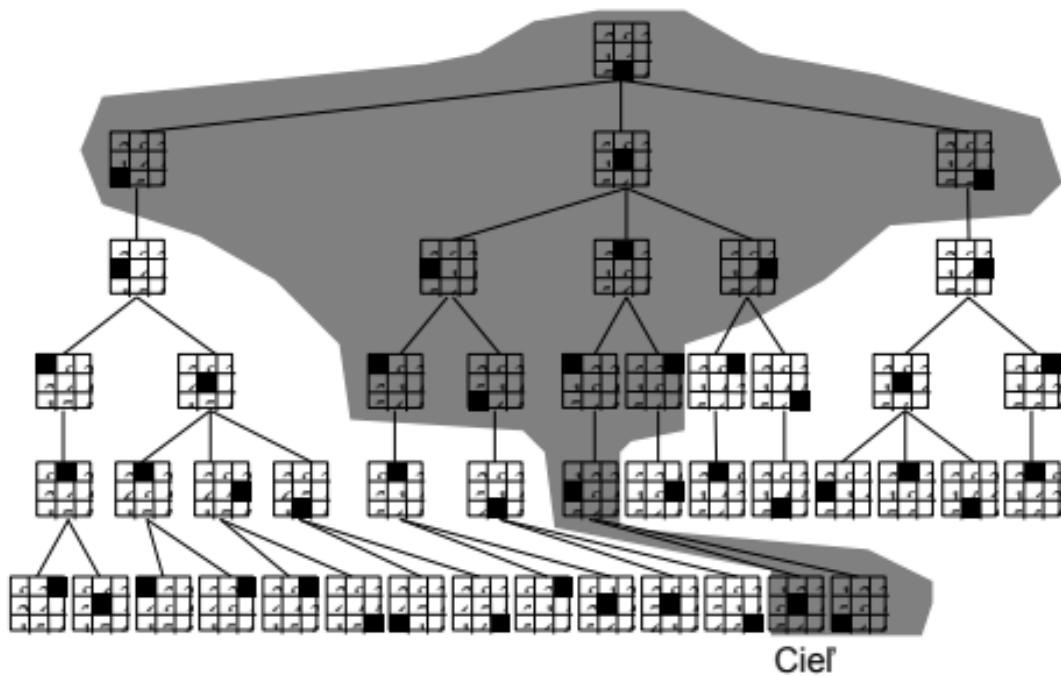
príklad A* hľadania



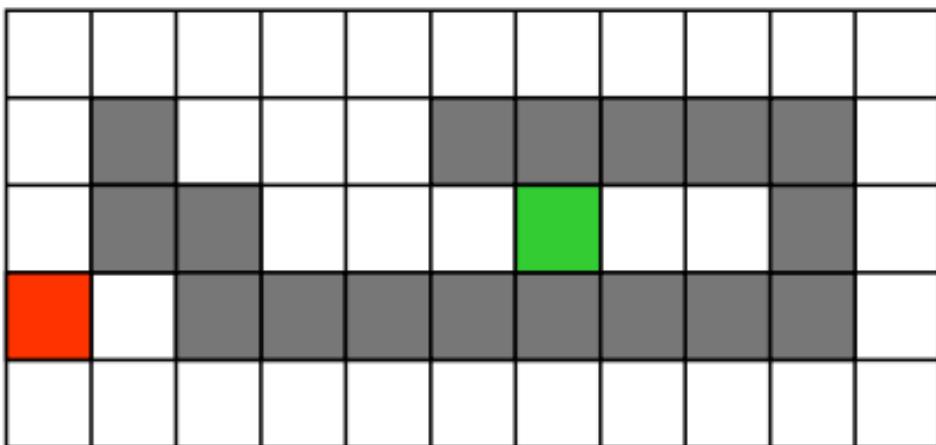
príklad A* hľadania



Porovnanie A* hľadania s hľadaním do šírky



Navigácia robota



Navigácia robota

$f(N) = h(N)$, kde $h(N)$ = manhattanská vzdialenosť do cieľa
(nie A*)

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Navigácia robota

$f(N) = h(N)$, kde $h(N)$ = manhattanská vzdialenosť do cieľa
(nie A*) – navštívené stavy

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

Navigácia robota

$f(N) = g(N) + h(N)$, kde $h(N)$ = manhattanská vzdialenosť do cieľa (A^*) – navštívené stavy

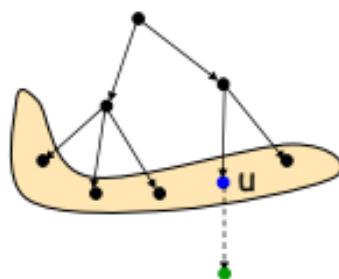
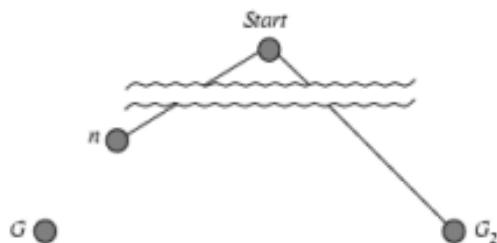
8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10	4	5	6
7+2		5+6	4+7	3+8						5
6+1			3	2+9	1+10	0+11	1	2		4
7+0	6+1									5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6

tvrdenie o A*

A* je úplný a prípustný

dôkaz prípustnosti (optimálnosti) A*

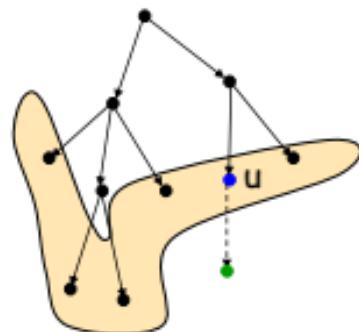
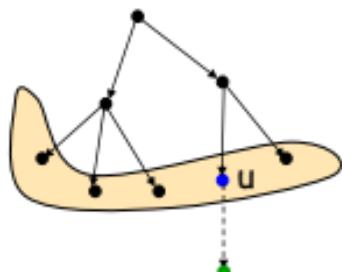
- nech C^* je cena optimálnej cesty.
- Predpokladajme: hľadanie skončí v cieľovom stave c ($h(c)=0$), pre ktorý $f(c) = g(c) > C^*$ (to je ale **suboptimálne** riešenie! – viedeme dôkaz sporom)
- uvažujme uzol u na okraji (čiže nie je cieľový), ležiaci na optimálnej ceste (existuje určite? áno, inak by sa už dosiahol cieľ na optimálnej ceste a hľadanie by skončilo tam a nie v c)



- legenda: uzol $u = n$, cieľ $c = G_2$

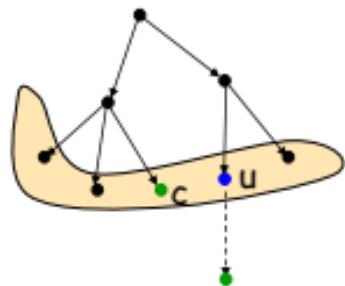
dôkaz prípustnosti (optimálnosti) A*

- uvažujme uzol u na okraji (čiže nie je cieľový), ležiaci na optimálnej ceste (existuje určite? áno, inak by sa už dosiahol cieľ na optimálnej ceste a hľadanie by skončilo tam a nie v c)
- každé rozvinutie nejakého uzla zvyšuje dĺžku nejakej cesty, takže skôr či neskôr musí prísť na rozvíjanie uzla u . Iba ak by sa už skôr našlo riešenie na inej ceste.

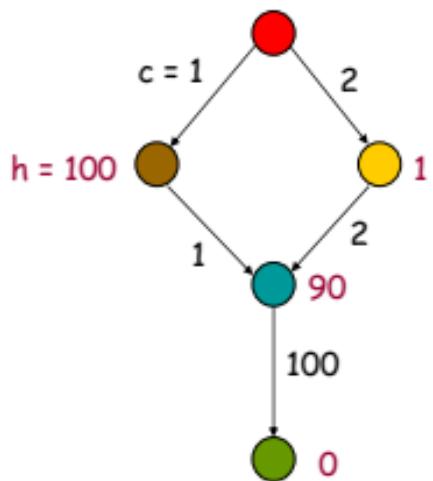


dôkaz prípustnosti (optimálnosti) A*

- $C^* \geq f(u)$ lebo h je prípustná
- $f(u) \geq f(c)$ lebo u sa nevybral dosiaľ na rozvitie
- $C^* \geq f(c)$ priamo z predchádzajúcich
- $C^* \geq g(c)$ lebo $h(c) = 0$ a teda $f(c) = g(c)$
- $C^* \geq g(c)$ je spor s predpokladom $g(c) > C^*$.
- nie je pravda, že A* vyberie neoptimálny cieľ.
Naopak, A* vyberie optimálny cieľ, t.j. cieľ na najlepšej ceste riešenia.

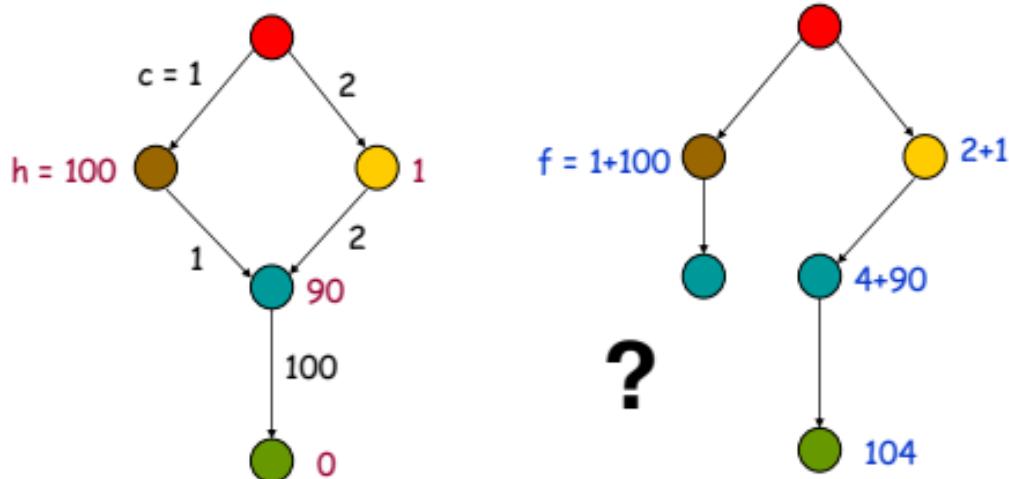


čo robiť s opäťovne navštívenými stavmi?



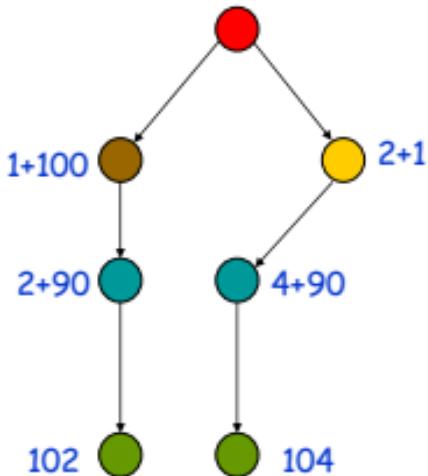
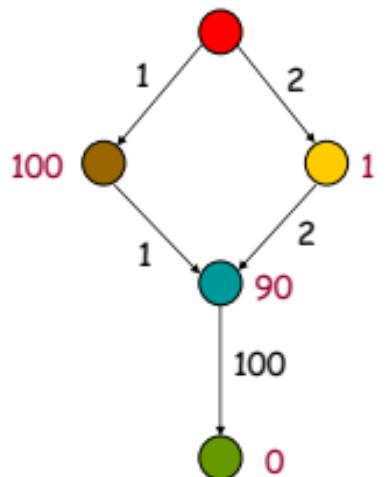
heuristika je zjavne
prípustná

čo robiť so opäťovne navštívenými stavmi?



ak zahodíme tento nový uzol, tak algoritmus rozvinie v ďalšom kroku cielový uzol a vráti neoptimálne riešenie

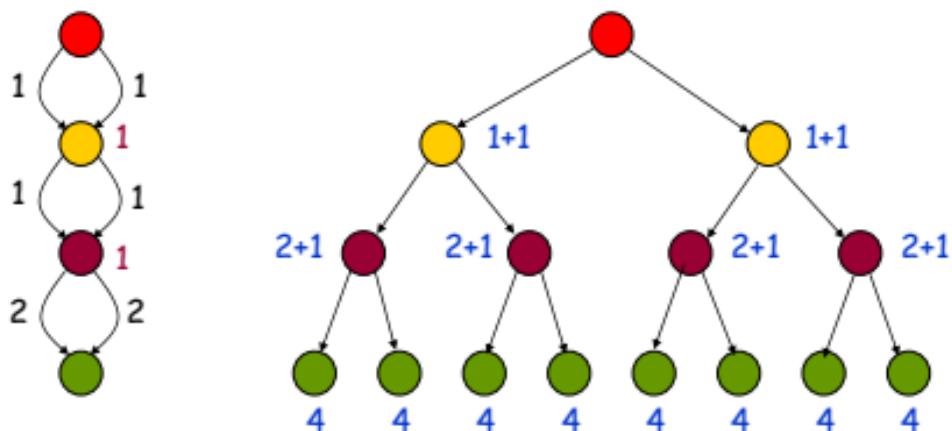
čo robiť s opäťovne navštívenými stavmi?



ak namiesto toho nezahadzujeme uzly s
opäťovne navštívenými stavmi, hľadanie skončí s
optimálnym riešením.

ale ...

ak nezahadzujeme uzly so znovunavštívenými stavmi, tak veľkosť stromu hľadania môže byť exponenciálna v závislosti od počtu navštívených stavov

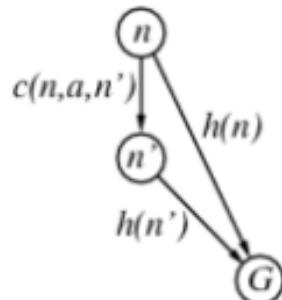


odhadzovanie uzlov, reprezentujúcich znovunavštívené stavy

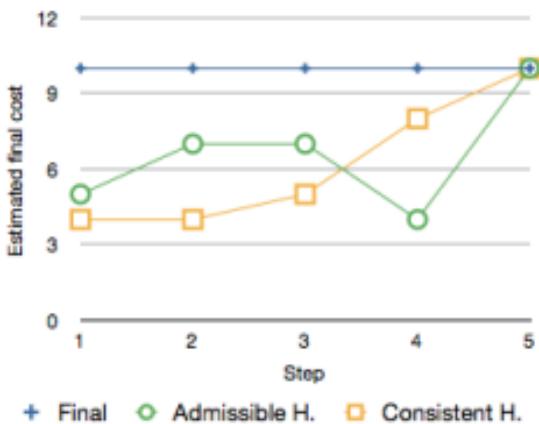
- odhadzovanie uzlov, reprezentujúcich znovunavštívené stavy, neškodí, ak cena novej cesty do toho uzla nie je menšia než cena doterajšej cesty
 - čiže napr možno odhodiť uzol, ktorý znovunavštíva stav, navštívený niektorým jeho predchadcom
- pre veľkú triedu prípustných heuristik, tzv konzistentných heuristik existuje omnoho efektívnejší spôsob spracovania znovunavštívených stavov

konzistentná heuristika

- heuristika je konzistentná, ak pre každý uzol n a pre každý jeho nasledovník n' generovaný aplikovaním operátora a ,
- $$h(n) \leq c(n,a,n') + h(n')$$
- ak h je konzistentná, platí
- $$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$
- t.j. $f(n)$ je neklesajúca pozdĺž ľubovoľnej cesty.
- Veta:** Ak je $h(n)$ konzistentná, A^* je optimálny.



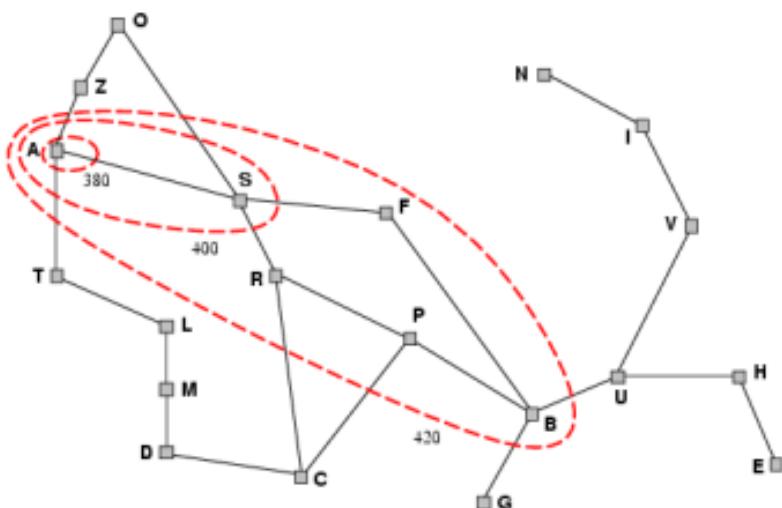
prípustná a konzistentná heuristika



- na obrázku je $f(n)$
- heuristika:
 - konzistentná \Rightarrow prípustná
 - prípustná \rightarrow konzistentná

prípustnosť A*

- A* rozvíja uzly v poradí podľa zvyšujúcej sa hodnoty f
- postupne pridáva obrysy uzlov "f-obrysy"
- obrys i má všetky uzly s hodnotou vyhodnocovacej funkcie $f=f_i$, kde $f_i < f_{i+1}$
-



úplnosť A*

- A* rozvíja uzly v poradí zvyšujúcej sa hodnoty vyhodnocovacej funkcie f (*najprv porozvíja všetky uzly s menším ohodnením a až potom prejde na uzly s najbližším vyšším ohodnením*)
- skôr či neskôr, ale po konečnom počte krokov, musí dôjsť na rozvitie cieľového uzla.
- alebo nie? áno, ak nie je v grafe nekonečne veľa uzlov s $f(u) < C^*$.
- čo je možné vtedy, ak
 - sú uzly s nekonečným faktorom vetvenia (počtom aplikovateľných operátorov)
 - (alebo) existuje cesta c s konečnou cenou ale nekonečným počtom uzlov pozdĺž nej*.

– * = Zenónov paradox dichotómie

Zenonov paradox dichotómie

- Zenonov paradox dichotómie: „To, čo sa pohybuje, musí najprv dospiť do polovice cesty skôr, než dospeje do konca. Ale aby to dospelo do polovice cesty, musí to najprv dospiť do polovice tejto polovice, a tak ďalej. Takže pohyb sa vlastne ani nemôže začať.“
- Tento paradox poukazuje na to, že prejsť konečnú dĺžku znamená prejsť nekonečne veľa bodov, a tieto dve veci sa nám intuitívne zdajú v rozpore. Tento rozpor sa však výrazne zmierní (alebo dokonca celkom zanikne) ak si uvedomíme, že podobne ako vzdialenosť môžeme deliť aj čas. Ak sa niečo pohybuje stále rovnakou rýchlosťou, tak to do polovice cesty dospeje za polovicu času, do štvrtiny cesty to dospeje za štvrtinu času, a tak ďalej. Čím kratší úsek, tým kratší čas potrebný na jeho prekonanie*.
- * = časopis .týždeň, 3/2008.

záležitosť s časovým ohraničením

- Ak problém nemá riešenie, A* sa vykonáva donekonečna, ak stavový priestor je nekonečný alebo ak sa stavy môžu znova navštievoať ľubovoľný počet ráz.
- Ak aj problém má riešenie, jeho nájdenie si môže vyžadovať obrovský čas. Pričom my nevieme odhadnúť vopred, koľko času bude treba, nevieme dokonca vopred, či vôbec má riešenie.
- V praxi treba čas vykonávania vopred ohraničiť. Pokiaľ A* nenájde riešenie do stanoveného limitu, hľadanie sa skončí. V takom prípade ale nevieme nič: či problém nemá riešenie, alebo má riešenie, lebo bolo treba viac trpezností.
- Pri malých problémoch to nemusí vadiť. pri väčších je meta-problém: ako nastaviť limit?

dominujúca heuristika

- Nech h_1 a h_2 sú prípustné heuristiky.
Ak $h_2(n) \geq h_1(n)$ pre všetky n
• tak h_2 **dominuje** nad h_1 ,
- dôsledok: h_2 je lepšia na hľadanie
-
- Príklad. Typické náklady hľadania (priemerný počet rozvitych uzlov):
-
- $d=12$ IDS = 3,644,035 uzlov
 $A^*(h_1) = 227$ uzlov
 $A^*(h_2) = 73$ uzlov
- $d=24$ IDS = príliš veľa uzlov
 $A^*(h_1) = 39,135$ uzlov
 $A^*(h_2) = 1,641$ uzlov
-

tvrdenie o A*

Nech h je konzistentná heuristika. Vždy keď A* rozvinie uzol, tak už našiel optimálnu cestu do stavu reprezentovaného týmto uzlom

dôkaz (1/2)

- 1) uvažujme uzol N a jeho potomka N'
kedže h je konzistentná: $h(N) \leq c(N,N') + h(N')$

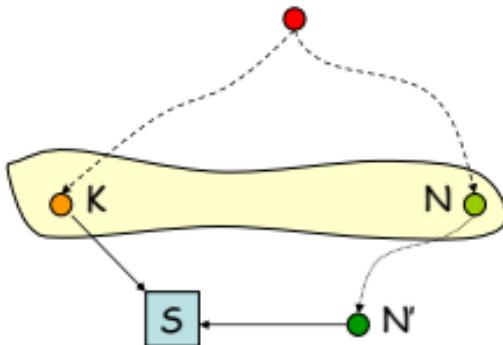
$$f(N) = g(N) + h(N) \leq g(N) + c(N,N') + h(N') = f(N')$$

takže f je neklesajúca pozdĺž každej cesty



dôkaz (2/2)

- 2) ak sa uzol K vyberie na rozvinutie, tak pre ľubovoľný iný uzol N na OKRAJi platí $f(N) \geq f(K)$



ak uzol N leží na inej ceste do stavu S reprezentovaného uzlom K, tak cena tejto inej cesty nie je menšia než cena cesty do K:

$$f(N') \geq f(N) \geq f(K)$$

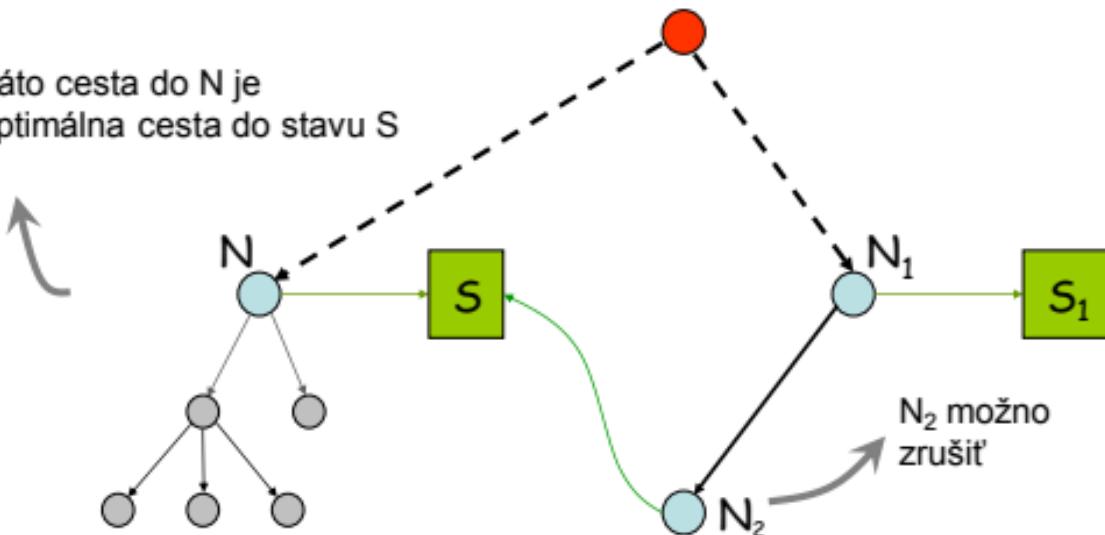
a platí $h(N') = h(K)$ lebo sa ohodnocuje ten istý stav

$$\text{takže } g(N') \geq g(K)$$

(t.j. ak sa vybral na rozvitie K, tak cena cesty do neho nie je väčšia, než ľubovoľná iná cesta)

dôsledok tvrdenia

Táto cesta do N je
optimálna cesta do stavu S



znovunavštívené stavy pri hľadaní s konzistentnou heuristikou

- keď sa uzol rozvije, ulož jeho stav do UZAVRETÉ
- keď sa vygeneruje nový uzol N:
 - ak $\text{STAV}(N)$ je v UZAVRETÉ, zruš N
 - ak existuje uzol N' na okraji taký, že $\text{STAV}(N') = \text{STAV}(N)$, zruš ten z nich – N alebo N' – s vyšším ohodnotením funkciou f

Je A* s nejakou konzistentnou heuristikou
to jediné, čo potrebujeme?

Nie !

Jestvujú veľmi hlúpe ale konzistentné heuristické funkcie

napr: $h \equiv 0$

- je konzistentná (a teda prípustná) !
- A^* s $h \equiv 0$ je hľadanie s rovnomernou cenou
- hľadanie do šírky a s rovnomernou cenou sú zvláštne prípady A^*

Zvláštne prípady

- Ak nás zaujíma iba to, aby sme sa do cieľa dostali hocjakou cestou, trebárs aj nie najlepšou, tak môžeme definovať g tak, že vždy bude jej hodnotou 0. Vtedy sa vyberie vždy uzol, ktorý sa javí byť najbližšie k cieľu (ide o lačné hľadanie).
- Ak chceme nájsť cestu, ktorá obsahuje najmenší počet uzlov, tak definujeme cenu prechodu z uzla do jeho nasledovníka ako konštantu, najčastejšie 1.
- Ak heuristiku vôbec nepoužijeme (h sa definuje tak, že jej hodnota je vždy 0), tak sa hľadanie riadi stratégou rovnomernej ceny.
- Ak navyše definujeme cenu prechodu z uzla do jeho nasledovníka ako 1, tak ide o hľadanie do šírky.

presnosť heuristiky

nech h_1 a h_2 sú dve konzistentné heuristiky také, že h_2 dominuje nad h_1 (t.j. pre všetky uzly N platí:

$$h_1(N) \leq h_2(N))$$

hovoríme aj, že h_2 je presnejšia (informovanejšia) než h_1

5		8
4	2	1
7	3	6

STAV(N)

1	2	3
4	5	6
7	8	

cieľový stav

- $h_1(N) =$ počet zle položených doštičiek
- $h_2(N) =$ súčet vzdialenosí každej doštičky do jej cieľového miesta
- h_2 je presnejšia než h_1

tvrdenie o A*

- nech h_2 je presnejšia než h_1
- nech A_1^* je A^* s použitím h_1
a nech A_2^* je A^* s použitím h_2
- vždy ak existuje riešenie, tak všetky uzly rozvinuté A_2^* , možno s výnimkou niektorých uzlov takých, že
$$f_1(N) = f_2(N) = C^* \text{ (cena optimálneho riešenia)}$$
sú rozvinuté aj A_1^*

Cyklicky sa prehlbujúce hľadanie algoritmom A* (IDA*)

- IDA* je úplný a prípustný.
- Tak ako hľadanie do hĺbky, vyžaduje pamäť v rozsahu úmernom dĺžke najdlhšej cesty, ktorú prezerá.

```
function IDA*(problém) returns riešenie alebo neúspech
    static: f-hranica, súčasná hranica určená funkciou f-CENA
            koreň, uzol

    koreň ← VYTvor-UZOL(ZAČIATOČNÝ-STAV[problém])
    f-hranica ← f-CENA(koreň)

    loop do
        riešenie, f-hranica ← HDH-OBRYS(koreň, f-hranica)
        if riešenie nie je null then return riešenie
        if f-hranica = ∞ then return neúspech
    end
```

Cyklicky sa prehľbujúce hľadanie algoritmom A* (IDA*)

```
function HDH-OBRYS(uzol, f-hranica)
```

returns riešenie a nová hranica určená funkciou f-CENA

static: d'alšia-f, hranica určená funkciou f-CENA pre ďalší obrys oblasti hľadania, na začiatku ∞

if f-CENA(uzol) > f-hranica **then return** null, f-CENA(uzol)

if CIEĽOVÝ-TEST[problém] aplikovaný na STAV(uzol) je úspešný
then return VYBER-RIEŠENIE (uzol), f-hranica

for each uzol u **in** NASLEDOVNÍKY(uzol) **do**

 riešenie, nová-f \leftarrow HDH-OBRYS(u , f-hranica)

if riešenie nie je null **then return** riešenie, f-hranica

 d'alšia-f \leftarrow MIN(d'alšia-f, nová-f)

end

return null, d'alšia-f

Algoritmus IDA*:

1. Inicializuj f-hranica na $f\text{-cena(koreň)}$
2. Opakuj:
 - a. Prehľadaj do hĺbky všetky uzly splňajúce podmienku $f\text{-cena}(N) \leq f\text{-hranica}$
 - b. Prirad f-hranica najnižšiu hodnotu zo všetkých nerozvítých uzlov (listov)

IDA* - príklad

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

$h(N)$ = počet nesprávne umiestnených poličok



4

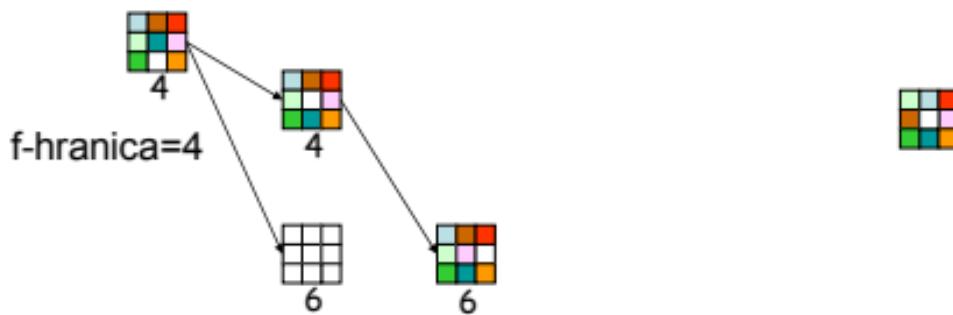
f -hranica=4



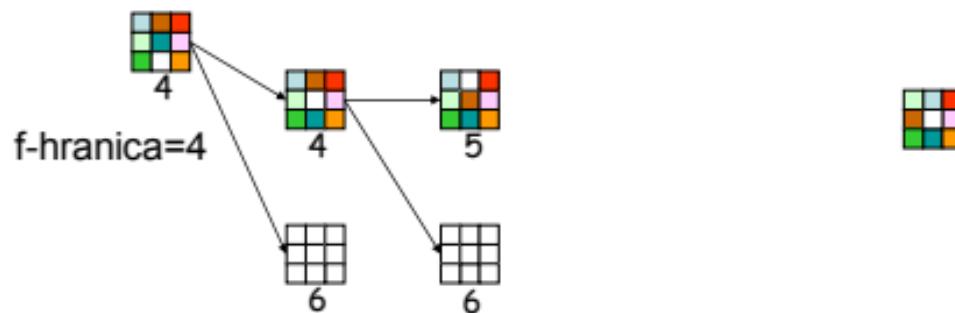
6



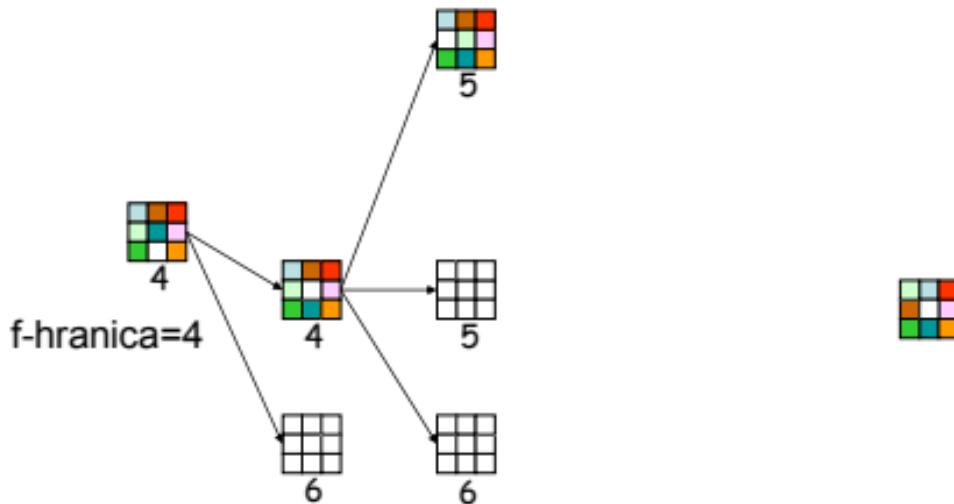
IDA* - príklad



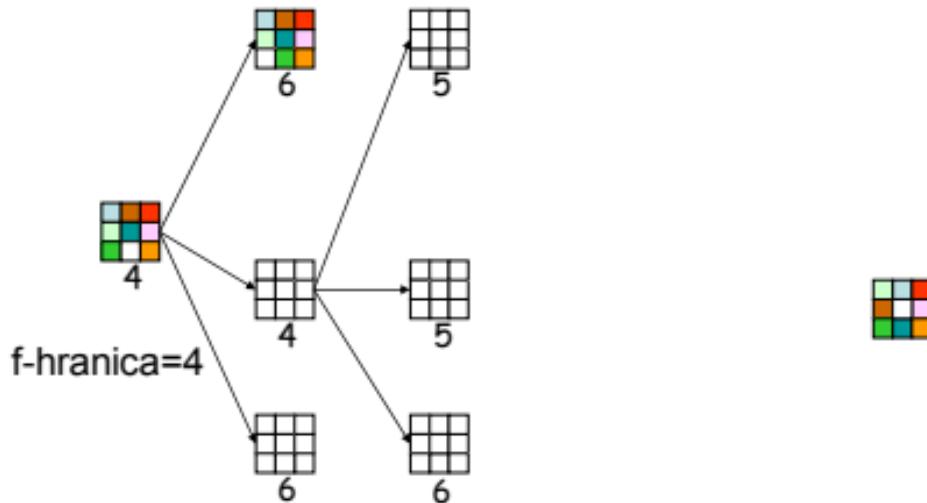
IDA* - príklad



IDA* - príklad



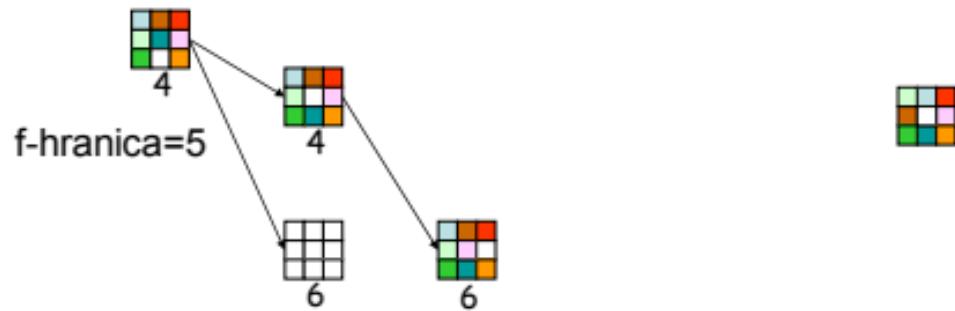
IDA* - príklad



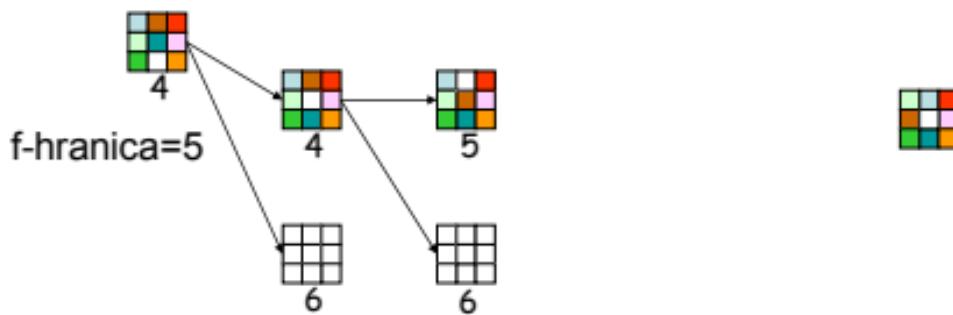
IDA* - príklad



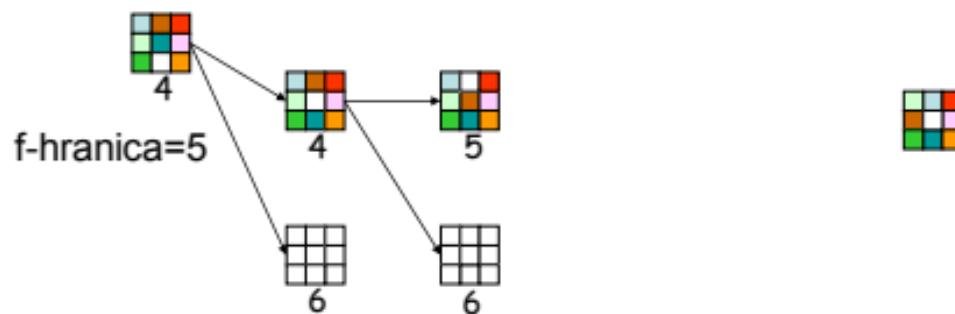
IDA* - príklad



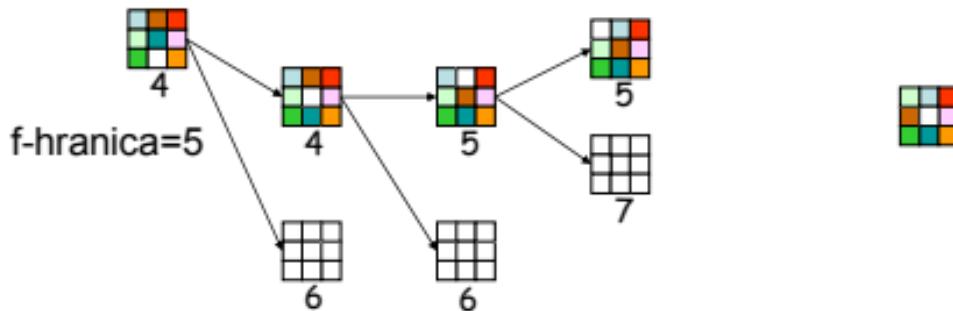
IDA* - príklad



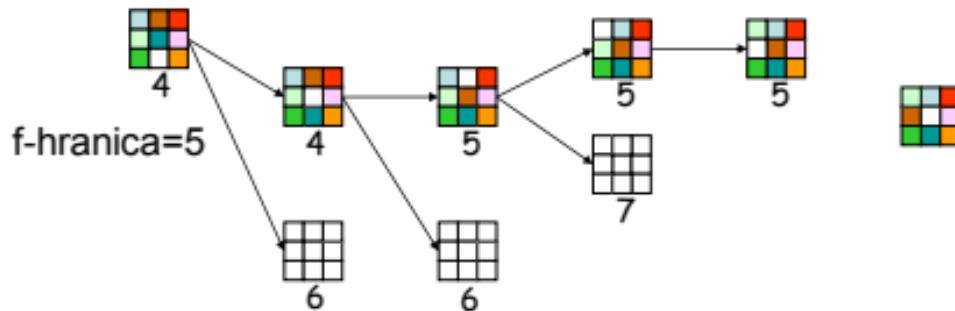
IDA* - príklad



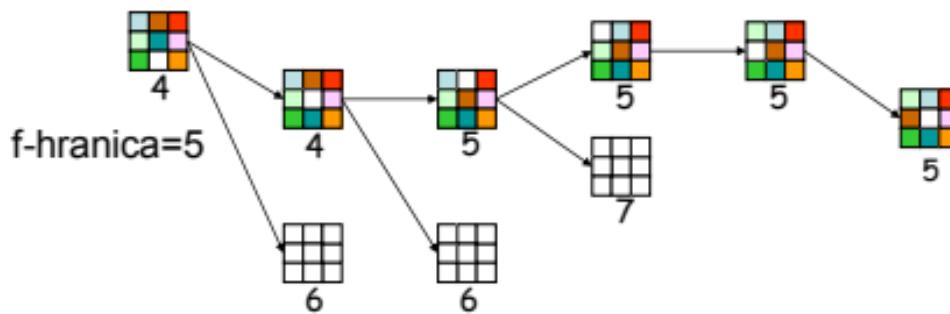
IDA* - príklad



IDA* - príklad



IDA* - príklad



účinný faktor vetvenia

- používa sa meranie účinnosti heuristiky
- nech n je celkový počet uzlov rozvítých stratégiou A* pre daný problém a nech d je hĺbka riešenia
- účinný faktor vetvenia b^* sa definuje
$$n = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

výsledky experimentov

(pozri R&N)

- 8-hlavolam s:
 - h_1 = počet zle položených doštičiek
 - h_2 = súčet vzdialenosí doštičiek od ich cieľovej polohy
- náhodné generovanie mnohých inštancií problému
- priemerné účinné faktory vetvenia (počet rozvinutých uzlov):

d	IDS	A_1^*	A_2^*
2	2.45	1.79	1.79
6	2.73	1.34	1.30
12	2.78 (3,644,035)	1.42 (227)	1.24 (73)
16	--	1.45	1.25
20	--	1.47	1.27
24	--	1.48 (39,135)	1.26 (1,641)

výhody/nevýhody IDA*

- výhody:
 - úplný a optimálny
 - potrebuje menej pamäti než A*
 - nestráca čas usporadúvaním okraja
- nevýhody:
 - nedá sa vyhnúť znovunavštíveniu stavov, ktoré nie sú na súčasnej ceste
 - slabé využívanie pamäti

kedy použiť hľadanie?

1) stavový priestor je malý a

- iný spôsob riešenia problému nepoznáme alebo
- vyvíjať efektívnejší spôsob nestojí za to

2) stavový priestor je veľký a

- iný spôsob riešenia problému nepoznáme ad
- existujú dobré heuristiky

Riešenie problémov rozkladom



A-ALEBO graf

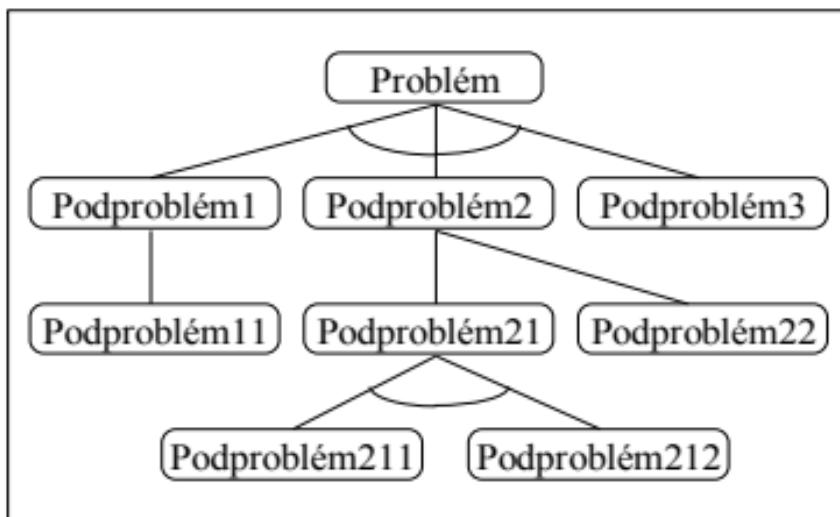
*hanba!

Riešenie problémov rozkladom

Rozložiteľný problém

- reprezentácia problémov a ich riešení:
 - rozklad problému na podproblémy
 - priestor rozkladov: konjunkcia podproblémov

Riešenie problémov rozkladom – A / ALEBO graf



- **hrany ALEBO** - spájajú uzol s alternatívami riešenia.
- **hrany A** - spájajú uzol s uzlami, ktoré spolu predstavujú riešenie

Riešenie problémov rozkladom – A / ALEBO graf

- Problém je v podstate daný
 - začiatočným stavom a
 - množinou rozkladajúcich operátorov.
- Riešenie v A/ALEBO grafe
 - jeho podgraf a nie cesta

Riešenie problémov rozkladom

```
function RIEŠENIE-ROZKLADOM(problém)
    returns graf_riešenia alebo neúspech
    if problém je elementárny then
        if problém je riešiteľný then
            return VYTVOR-GRAF-RIEŠENIA(problém)
        else return neúspech
    loop do
        rozlož problém na podproblémy
        if RIEŠENIE-ROZKLADOM je pre všetky podproblémy úspešné then
            return VYTVOR-GRAF-RIEŠENIA(problém,
                                         množina grafov riešenia pre podproblémy)
    end
```

Proces značkovania uzlov v grafe priestoru rozkladov

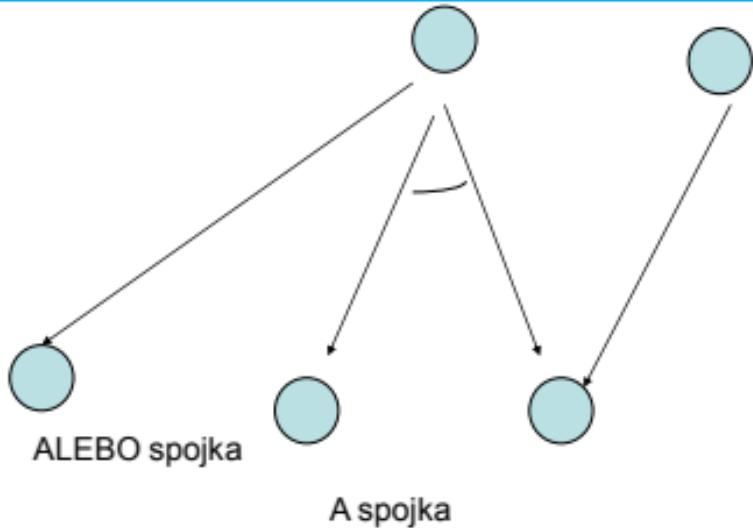
- Uzly značkujeme ako "**vyriešené**" takto:
 - každý koncový uzol, predstavujúci elementárny problém sa označí za "**vyriešený**";
 - každý nie koncový uzol, z ktorého vychádzajú hrany ALEBO a aspoň jeden z jeho nasledovníkov je označený ako "**vyriešený**" sa označí za "**vyriešený**";
 - každý nie koncový uzol, z ktorého vychádzajú hrany A a všetky jeho nasledovníky sú označené ako "**vyriešený**" sa označí za "**vyriešený**".

Proces značkovania uzlov v grafe priestoru rozkladov

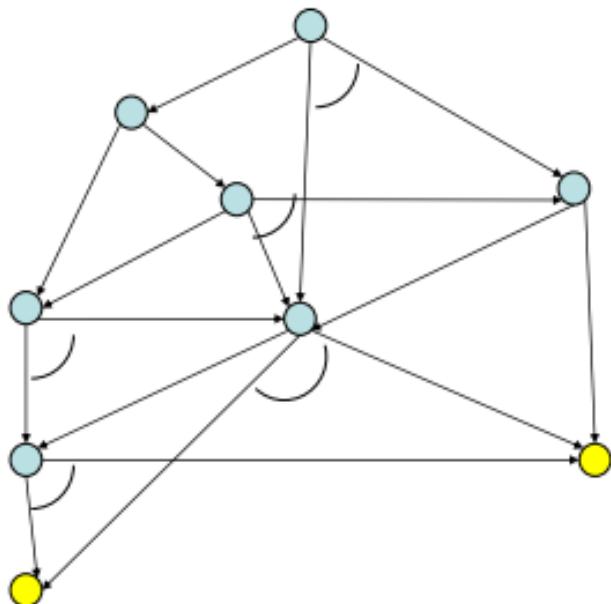
- Uzly sa značkujú ako “**neriešiteľné**” takto:
 - úlohy hrán A a ALEBO sa vymenia.
 - Tiež každý uzol, ktorý nereprezentuje elementárny problém a nemá žiadnych nasledovníkov sa označí za “neriešiteľný”.
- **Cieľový test**
 - testovanie, či je začiatočný stav “vyriešený” alebo “neriešiteľný”.

A/O* algoritmus

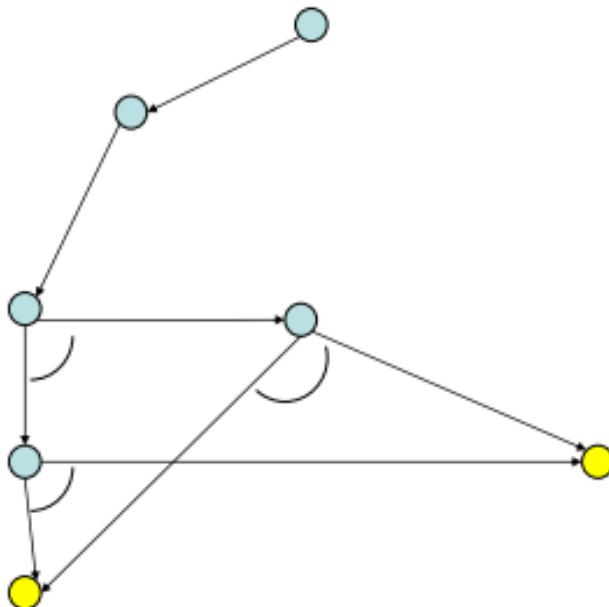
- dátová štruktúra
 - graf
 - označkované spojenia (hrany alebo dvojice hrán)
(orientované značky; nie ako A*)
 - ceny $q()$ sa udržiavajú na uzloch
 - označenie vyriešené
 - obmedzíme sa na acyklické grafy



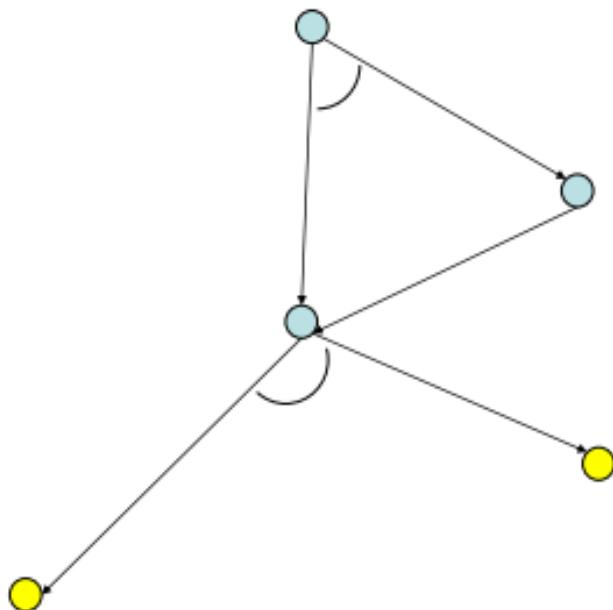
A/ALEBO graf



podgraf riešenia



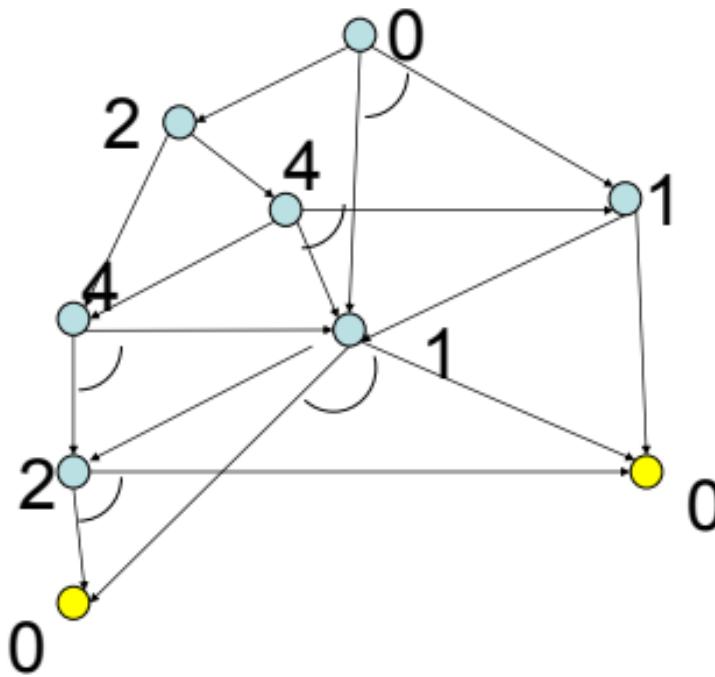
iný podgraf riešenia



podgraf riešenia $G' A/ALEBO$ grafu G

- z uzla n do koncových (vyriešených) uzlov T :
- ak n je v T , G' je len uzol n .
- inak n má jednu spojku do množiny uzlov n_1, n_2, \dots, n_k .
 - pre každý uzol n_i jestvuje graf riešenia z neho do T .
 - G' je n , tá spojka, uzly n_1, n_2, \dots, n_k
 - a k tomu grafy riešenia z každého uzla n_i .

heuristické hodnoty: odhad ceny do množiny vyriešených



obmedzenie monotónnosti

- $h(n) \leq c + h(n_1) + h(n_2) + \dots + h(n_k)$

kde c je cena spojky medzi n a množinou n_1, n_2, \dots, n_k .

toto obmedzenie zaručuje, že $h(n) \leq h^*(n)$.

hodnoty ceny ($q(n)$)

- ak n nemá nasledovníky, tak $q(n) = h(n)$
- inak počítajúc hodnoty od spodu,
 - $q(n) = \text{cena spojky} + \text{súčet ohodnotení } q(\text{nasledovníkov})$

z ohodnotení $q(n)$ uzla n vyber najmenšie a označ smer spojenia.

ak v označenom smere sú všetky nasledovníky vyriešené tak označ n vyriešený.

základná myšlienka A/O*

- najprv sa prechádza graf zhora nadol tak, že sa vyberajú najlepšie čiastočné podgrafy riešenia.
- rozvinie sa jeden listový uzol tohto grafu
- potom sa prechádza zdola nahor, upravujú sa ohodnotenia, podľa toho smerovania a značkuje sa *vyriešené*.

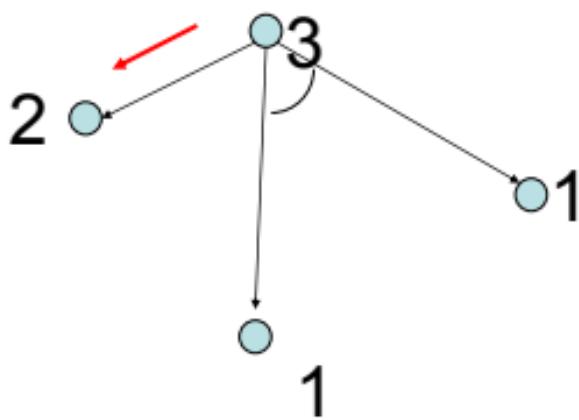
AO* algoritmus

1. vytvor $G = \langle s \rangle$; $q(s) = h(s)$
ak $s \in \text{TERM}$ tak označ s *vyriešený*
2. kým nie je s *vyriešený*:
 1. vypočítaj G' podgraf čiastočného riešenia sledovaním označených spojok v G z s .
 2. vyber n v G' , n nie je v TERM , n je list.
 3. rozvíň n , pridaj jeho nasledovníky do G a pre každý nasledovník, ktorý nie je ešte v G , priraď $q(\text{nasledovník}) = h(\text{nasledovník})$. Označ *vyriešené* všetky nasledovníky v TERM . (ak nie sú nasledovníky tak nastav $q(n) := \infty$).

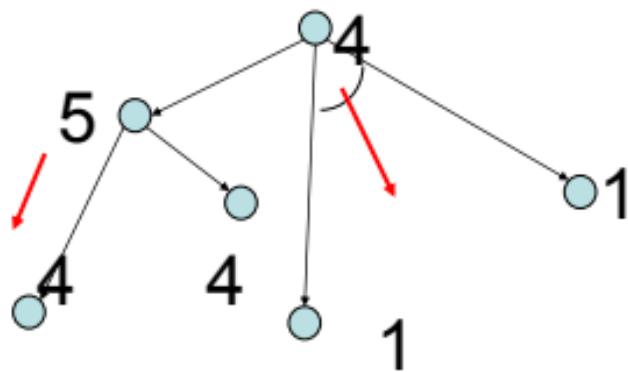
AO* algoritmus

4. priraď $S := \{n\}$.
5. pokial nie je $S = \emptyset$:
 1. odstráň uzol m z množiny S taký, že žiadny z jeho potomkov (nielen priamych nasledovníkov) v grafe G nie je aj v množine S . ak taký uzol nejestvuje v S , odstráň z S ľubovoľný uzol.
 2. vypočítaj cenu každej spojky vychádzajúcej z m . oprav cenu uzla m :
$$q(m) = \min [c + q(n_1) + \dots + q(n_k)]$$
označ zvolenú spojku.
ak všetky nasledovníky spojené práve označenou spojkou sú vyriešené tak označ m vyriešený.
 3. ak bol práve označený vyriešený alebo sa práve menilo ohodnotenie $q(m)$ tak pridaj do S všetkých "spojených" predchodcov uzla m .
4. koniec
6. koniec

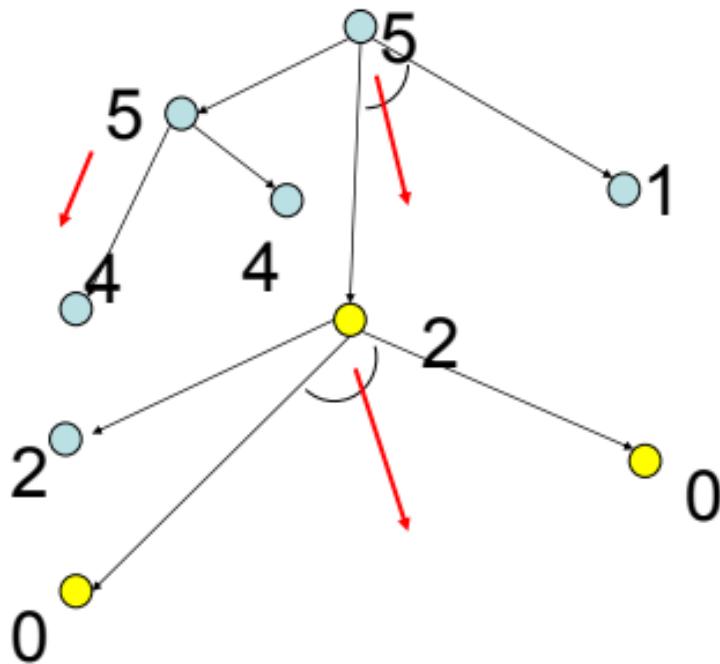
stopa algoritmu



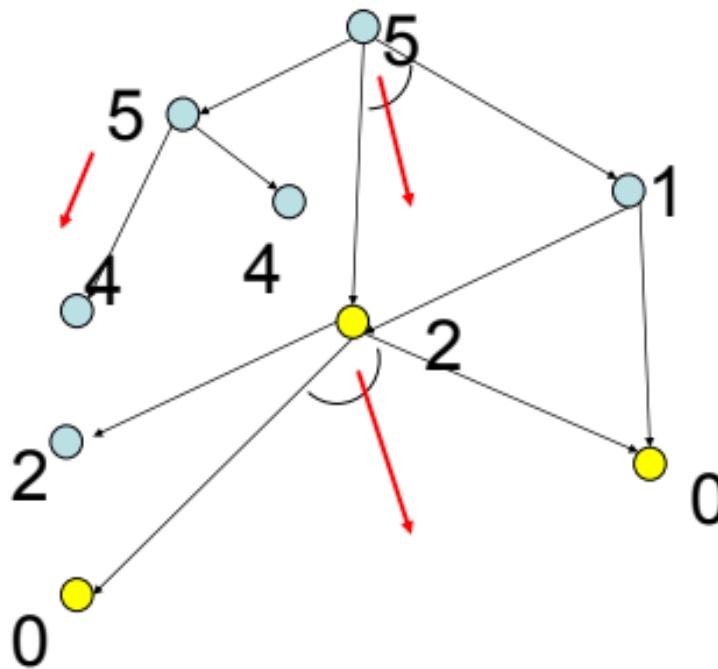
stopa algoritmu



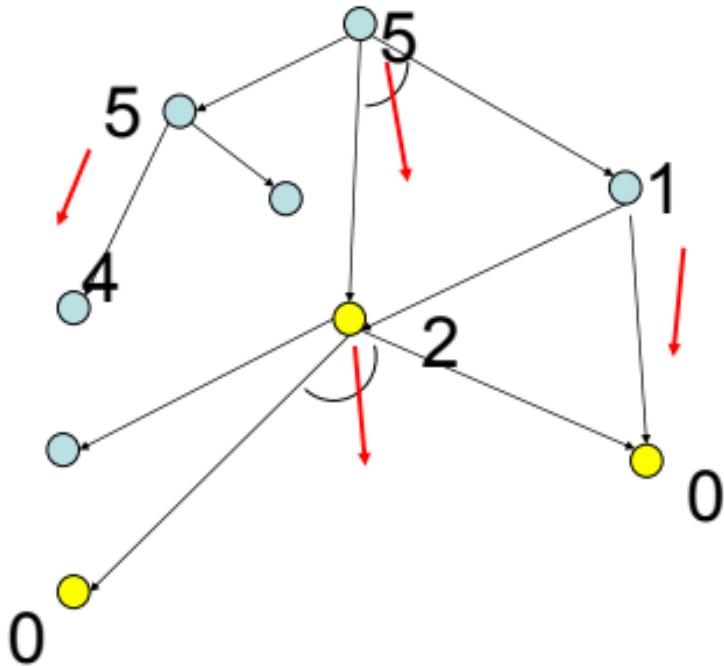
stopa algoritmu



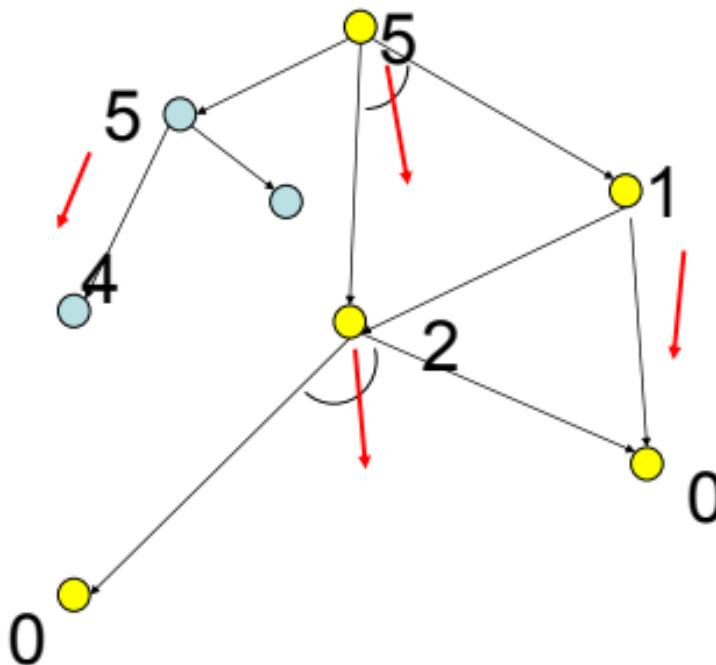
stopa algoritmu



stopa algoritmu



stopa algoritmu



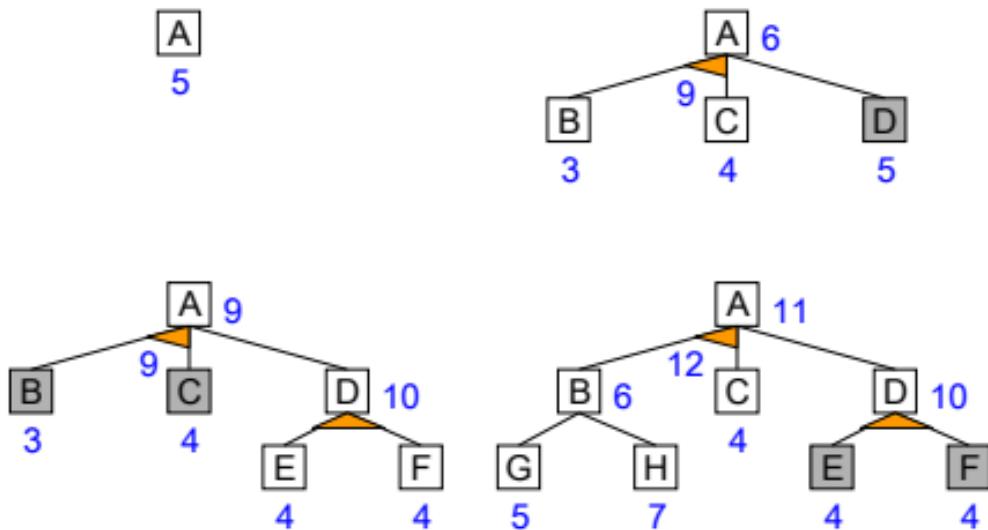
optimálnosť of A/O*?

- ak $h(n) \leq c + h(n_1) + \dots + h(n_k)$
pre všetky uzly n a spojky
tak cena riešenia je optimálna.

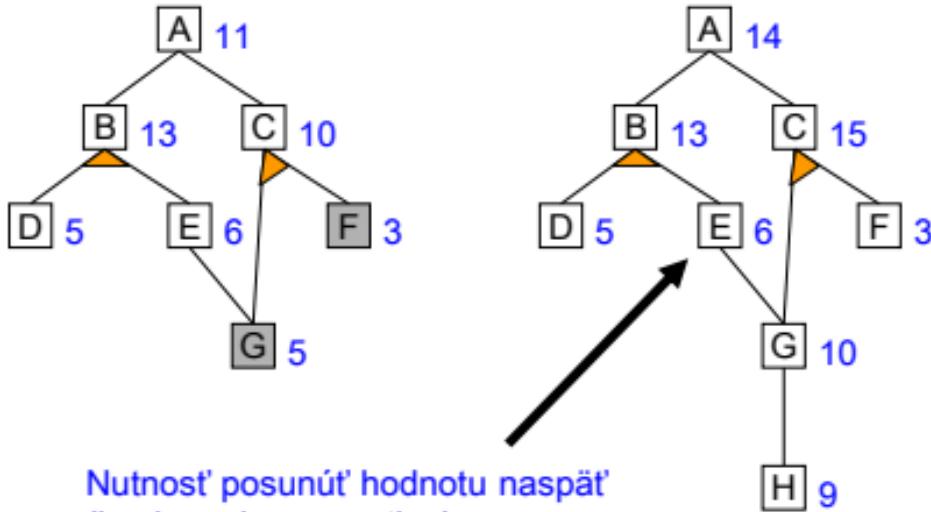
poznámky

- Acykličnosť grafu znamená, že množina S sa nakoniec vyprázdní.
- Ako vybrať ne-elementárny uzol v grafe čiastočného riešenia na rozvitie?
 - najlepšia voľba je uzol s najvyšším ohodnotením h^* (aj tak sa bude musieť rozvinúť)

Riešenie problémov rozkladom : AO*



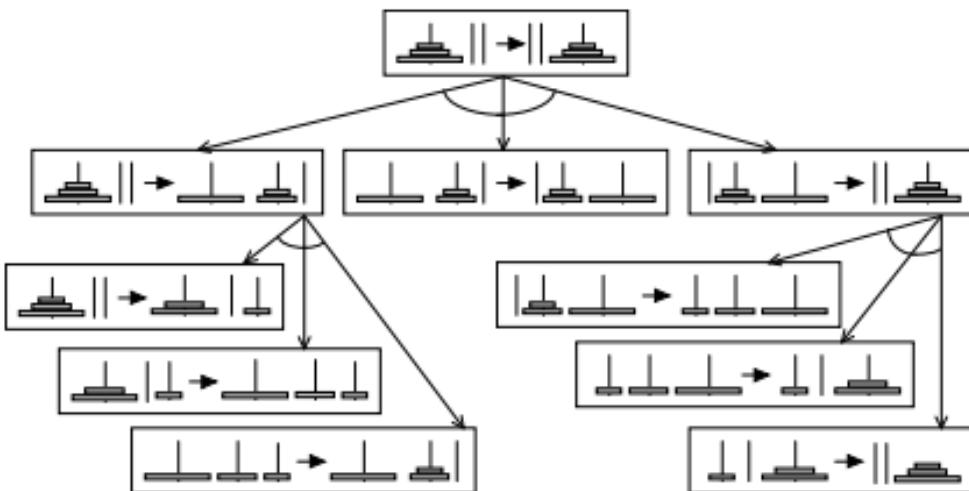
Riešenie problémov rozkladom: AO*



Problém Hanojskej veže

- Problém Hanojskej veže možno opísť takto:
- **stavy**: umiestnenie troch diskov na troch stojanoch tak, že disk s menším priemerom leží vždy na disku s väčším priemerom;
- **začiatočný stav**: všetky disky na prvom (ľavom) stojane tak, že disk s menším priemerom leží vždy na disku s väčším priemerom;
- **operátory**: presun disku na iný stojan, pričom nesmie nikdy vzniknúť situácia, aby na nie-ktorom stojane ležal väčší disk na menšom;
- **cieľový test**: všetky disky na poslednom (pravom) stojane tak, že disk s menším priemerom leží vždy na disku s väčším priemerom;
- **cena cesty**: každá aplikácia operátora má cenu 1.

Problém Hanojskej veže



Algoritmy cyklického vylepšovania

Algoritmus cyklického vylepšovania vychádza z ľubovoľného začiatočného stavu opisujúceho úplnú konfiguráciu a postupne mení stav tak, aby sa konfigurácia vylepšovala z hľadiska približovania sa k cieľovému stavu.

```
function GENEROVANIE-A-TESTOVANIE(problém, GENERUJ-STAV)
returns stav riešenia
    static: možné-riešenie, stav (ak riešením je stav)
            alebo posledný stav na ceste (ak riešením je cesta)

    loop do
        možné-riešenie ← GENERUJ-STAV(STAVY[problém])
        if CIEĽOVÝ-TEST[problém] aplikovaný na možné-riešenie je úspešný
            then return možné-riešenie
    end
```

Generovanie stavov

- **systematické**
 - exhaustívne (vyčerpávajúce, úplné) prehľadanie priestoru problému.
- **náhodné**
 - Nie je záruka, že sa riešenie nájde, aj ak riešenie existuje
(algoritmus Britského múzea)
hypotéza o opiciach a písacích strojoch

hypotéza o opiciach a písacích strojoch

- stačí posadiť dostatočný (nekonečný) počet opíc za dostatočný (nekonečný) počet písacích strojov a ich ťukaním skôr či neskôr vzniknú Shakespearove zoobrané spisy.
- ale (v skutočnosti, aspoň podľa pokusu výskumníkov Plymouthskej univerzity v Anglicku, 2003)
- 6 opíc druhu makak dostalo jeden počítač a 4 týždne času. výsledok?
 - jeden chytí kameň a mlátil do klávesnice
 - ďalšie vykonali nad klávesnicou rôzne kombinácie malej a veľkej potreby. opakovane.
 - po čase sa dostali k tomu, že začali písat písmeno S.
 - celkovo napísali 5 strán, okrem písmena S sa im do výsledného textu vkradlo aj zapár písmen A, J, L a M.
- projekt stál 2000 £.
- záver: opice nemožno redukovať na náhodné stroje. počítač ich nudí.
- napriek tomu, ak by bolo nie 6, ale 10^{813} opíc, nie 4 týždne, ale 5 rokov a každá mala svoj počítač, tak by vznikol sonet č. 3. samozrejme, od Shakespeara.



lokálne hľadanie

- metóda hľadania s nízkymi požiadavkami na pamäť
- hľadanie negeneruje strom hľadania; vždy sa pracuje len so zápisom súčasného stavu!
- dá sa použiť len na problémy, kde riešením nie je cesta (napr. 8 dám) ale stav – iba ak by sa cesta nejako kódovala do stavu
- podobnosti s metódami optimalizácie

lokálne hľadanie

- V mnohých optimalizačných problémoch nie je dôležitá cesta do cieľa – riešením je cieľový stav
- Stavový priestor = množina „úplných“ konfigurácií
- Treba nájsť konfiguráciu, ktorá splňa ohraničenia
- V takých prípadoch možno použiť lokálne hľadanie
- Udržiava sa „súčasný“ stav, je snaha vylepšovať ho

Stratégia lokálneho vylepšovania

Podstatou stratégie lokálneho vylepšovania je všeobecne používaná heuristika: pri hľadaní riešenia postupovať v každom kroku v smere, ktorý spôsobí lokálne zlepšenie z daného stavu.

Algoritmus neudržiava strom hľadania (nevybraté uzly sa nepamätajú, ale okamžite sa zabúdajú). Uzol obsahuje iba opis stavu a jeho ohodnotenie.

```
function LOKÁLNE-VYLEPŠOVANIE(problém) returns stav riešenia
    static: súčasný, uzol

    súčasný ← VYTvor-UZOL(ZAČIATOČNÝ-STAV[problém])
    loop do
        if nejaký nasledovník uzla súčasný má lepšie ohodnotenie
            then súčasný ← lepšie ohodnotený nasledovník uzla súčasný
        else return STAV(súčasný)
    end
```

Stratégia lokálneho vylepšovania

- "cyklus hľadania, ktoré sa nepretržite pohybuje v smere zvyšujúcej sa hodnoty"
 - Skončí keď sa dosiahne vrchol
 - Stratégia známa tiež ako ľačné lokálne hľadanie
 - Stratégia známa tiež ako horolezecký algoritmus
 - Výstup na Mt Everest v úplnej hmele
- Hodnota (ohodnocovacej funkcie) je buď
 - Hodnota cieľovej funkcie
 - Hodnota heuristickej funkcie
- Nepozerá sa dopredu pred bezprostredných susedov súčasného stavu.
- Volí náhodne z množiny nasledovníkov, ak viac je hodnotených lepšie

Stratégia lokálneho vylepšovania a lokálne extrémy

- ak existujú lokálne extrémy, hľadanie nie je optimálne
- jednoduchý spôsob nájdenia riešenia (a často efektívny)
 - viac pokusov hľadania – náhodné začiatky

Stratégia lokálneho vylepšovania - príklad

- Problém 8-dám, opis stavu zahŕňa úplnú konfiguráciu
 - Všetkých 8 dám na doske vnejakej konfigurácii
- Funkcia nasledovníka:
 - Presuň dámku na iné poličko v tom istom stĺpci.
- Príklad heuristickej funkcie $h(u)$:
 - Počet dvojíc dám, ktoré sa napádajú
 - (čiže toto chceme minimalizovať)

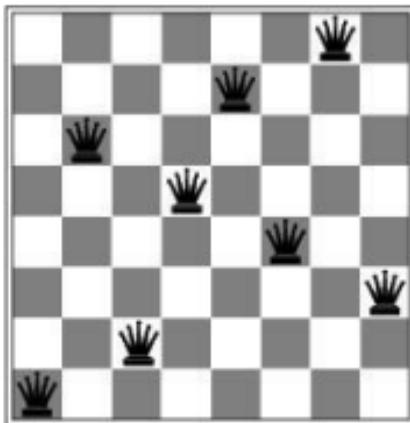
Stratégia lokálneho vylepšovania - príklad

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	14	13	16	13
14	14	17	15	15	14	16	16
17	15	16	18	15	15	15	17
18	14	14	15	15	14	15	16
14	14	13	17	12	14	12	18

Ohodnenie súčasného stavu: $h=17$

Znázornené sú h -hodnoty každého možného nasledovníka v každom stĺpci

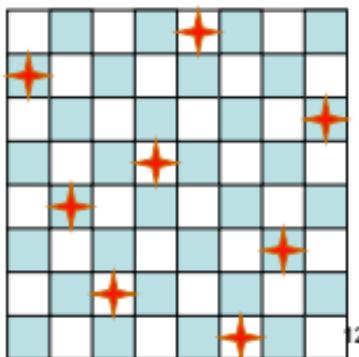
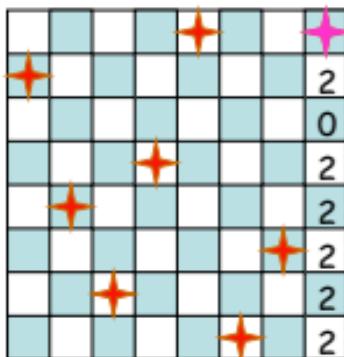
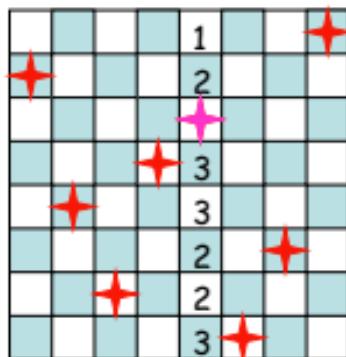
Lokálne minimum pre 8-dám



Lokálne minimum v stavovom priestore problému 8-dám ($h=1$)
mimochodom: prečo je tento stav lokálne minimum?

8-dám, trochu iná heuristika

- 1) zvoľ začiatočný stav S náhodne tak, že v každom stĺpci je práve 1 dáma
- 2) opakuj k razy:
 - a) If GOAL?(S) then return S
 - b) zvoľ náhodne dámu Q , ktorá je napadnutá
 - c) presuň Q v jej stĺpci na poličko, kde ju bude napádať čo najmenej dám
→ nový stav S
- 3) Return neúspech

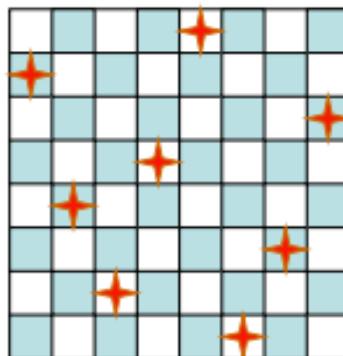
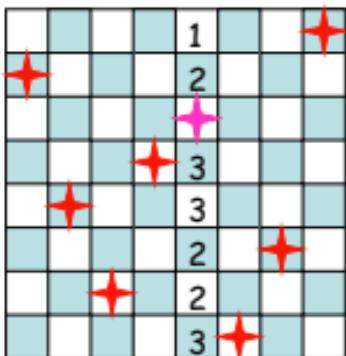


8-dám, trochu iná heuristika

opakujúci sa rozloženie

prečo to funguje??

- 1) jestvuje veľa cieľových stavov, ktoré sú dobre rozdelené v stavovom priestore
- 2) ak sa nenájde riešenie po niekoľkých krokoch, treba radšej skončiť a začať odznova. hľadanie by bolo kvôli vysokému faktoru vetvenia neefektívne
- 3) čas hľadania je skoro nezávislý od počtu dám



Ako funguje stratégia lokálneho vylepšovania na problém 8-dám

- Začiatočné stavy sa generujú náhodne...
- v 14% prípadov vyrieši problém
- v 86% prípadov zapadne v lokálnom minime
- avšak...
 - V prípade úspechu potrebuje len 4 kroky na nájdenie cieľového stavu
 - Len 3 kroky v priemere na zapadnutie v lokálnom minime
 - (v stavovom priestore s ~17 miliónmi stavov)

Možné riešenie...bočné úkroky

- Ak nie sú možné kroky nahor (nadol), treba povoliť bočné úkroky v nádeji, že hľadanie sa vyhne lokálnemu extrému
 - Treba stanoviť hranicu na možný počet bočných úkrokov, aby nemohlo dôjsť k nekonečnému cyklu
- pre 8-dám
 - Nech je povolených 100 bočných úkrokov
 - Toto zvýši podiel úspešne vyriešených inštancií problémov z 14 na 94%
 - avšak....
 - 21 krokov do každého úspešného riešenia
 - 64 krokov do každého neúspechu

Stratégia lokálneho vylepšovania - variácie

- stochastická stratégia lokálneho vylepšovania (stochastic hill-climbing)
 - náhodný výber spomedzi krokov smerujúcich nahor
 - pravdepodobnosť výberu ďalšieho kroku môže byť daná strmostou pohybu v smere príslušného kroku
- stratégia lokálneho vylepšovania prvý berie (first-choice hill-climbing)
 - stochastická stratégia lokálneho vylepšovania generovaním nasledovníkov náhodne dovtedy, kým sa najde lepší
 - užitočné, ak je veľmi veľa nasledovníkov
- stratégia lokálneho vylepšovania s náhodným reštartom
 - pokúša sa vyhnúť uviaznutiu v lokálnom maxime

stratégia lokálneho vylepšovania s náhodným reštartom

- rôzne obmeny
 - pre každý opakovaný začiatok hľadania (reštart):
 - hľadá sa, kým neskončí
 - hľadá sa vopred stanovenú dobu
 - opakovanie hľadania:
 - vopred stanovený počet opakovania (reštartov):
 - hľadá sa „donekonečna“
- ako odhadnúť počet opakovania?
- ako odhadnúť počet krokov do nájdenia riešenia?

stratégia lokálneho vylepšovania v lúči (local beam search)

- udržiava si nie 1 ale k súčasných stavov
 - na začiatku: vyberie sa náhodne k stavov
 - ďalší krok: určia sa všetky nasledovníky všetkých k stavov
 - ak je hodnoty z nich riešením – return
 - inak vyberie sa k najlepších nasledovníkov, ďalší krok

stratégia lokálneho vylepšovania v lúči (local beam search)

- zdá sa, že je to k paralelných hľadaní stratégou lokálneho vylepšovania
- nie, lebo informácie o hľadaní sú spoločné pre všetkých k vlákien
- ak jeden stav generuje viacero dobrých nasledovníkov, môžu sa dostať (aj všetky) do ďalšieho kola
- stavy, ktoré generujú zlé nasledovníky, sa odstránia

stratégia lokálneho vylepšovania v lúči (local beam search)

- spôsob výberu nasledovníkov je silná aj slabá stránka stratégie
- silná:
 - neproduktívne stavy (smery hľadania) sa rýchlo zanechajú
 - stavy sľubujúce najväčší pokrok sa uprednostňujú
- slabá:
 - nedostatočná rôznorodosť (hľadá sa v malom výseku stavového priestoru)
 - náprava: vybrať k nasledovníkov náhodne, s predispozíciou pre lepšie stavy, (tj s pravdepodobnosťou danou ohodnotením nasledovníka)

stratégia lokálnej optimalizácie

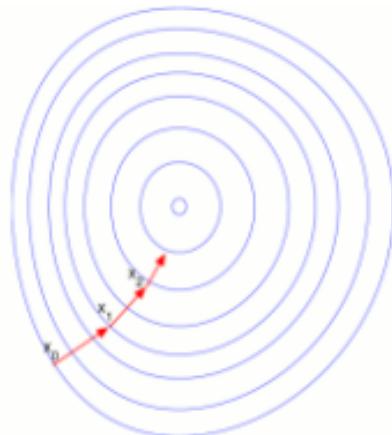
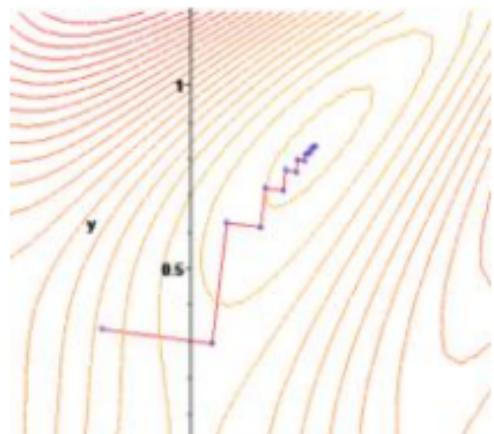
Ako ďalší uzol sa nevyberá hodiktorý lepší než súčasný, ale najsľubnejší z jeho nasledovníkov.

```
function LOKÁLNA-OPTIMALIZÁCIA(problém) returns stav riešenia
    static: súčasný, uzol
            ďalší, uzol

    súčasný ← VYTvor-UZOL(ZAČIATOČNÝ-STAV[problém])
    loop do
        ďalší ← najlepšie ohodnotený nasledovník uzla súčasný
        if HODNOTA[ďalší] < HODNOTA[súčasný]
            then return STAV(súčasný)
        súčasný ← ďalší
    end
```

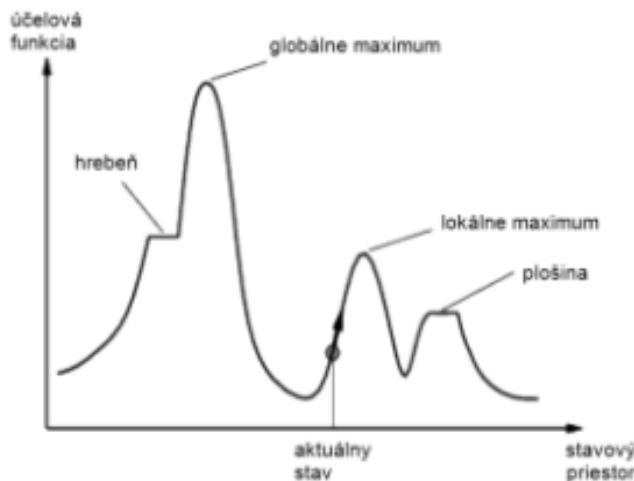
stratégia lokálnej optimalizácie

Ako ďalší uzol sa nevyberá hodiktorý lepší než súčasný, ale najsľubnejší z jeho nasledovníkov.



stratégie lokálneho vylepšovania / optimalizácie

Stratégie lokálneho vylepšovania skrývajú v sebe aspoň tri úskalia:



- hrebeň = postupnosť stavov ohodnotených ako lokálne maximá. Lačné lokálne hľadanie len ľahko naviguje po hrebeni
- plošina = oblasť stavového priestoru, v ktorej sú hodnoty vyhodnocovacej funkcie ploché (rovnaké).

Simulované žíhanie

- Upustenie od podmienky, že pri hľadaní sa nemôže ani o krok zostúpiť.
- Po uviaznutí v lokálnom maxime sa vykoná niekoľko krokov dolu kopcom, aby sme unikli z okolia lokálneho maxima, namiesto toho aby hľadanie pokračovalo z nejakého náhodne zvoleného miesta.
- Vnútorný cyklus simulovaného žíhania sa podobá lokálnej optimalizácii. Nevyberá sa však najlepší krok, ale náhodný krok.
- Ak sa pritrafí, že vybratý krok naozaj zlepšuje situáciu, tak sa aj vykoná. Inak vykonanie kroku určuje nejaká pravdepodobnosť, ktorá je menšia než 1.

Simulované žíhanie

```
function SIMULOVANÉ-ŽÍHANIE(problém, ROZVRH) returns stav riešenia
    inputs: problém
            rozvrh, zobrazenie času to "teploty"
    static: súčasný, uzol
            ďalší, uzol
            T, "teplota", ktorá riadi pravdepodobnosť krokov nadol

    súčasný ← VYTvor-UZOL(ZAČIATOČNÝ-STAV[problém])
    for k ← 1 to ∞ do
        T ← ROZVRH[k]
        if T = 0 then return STAV(súčasný)
        ďalší ← náhodne vybratý nasledovník uzla súčasný
        ΔE ← HODNOTA[ďalší] – HODNOTA[súčasný]
        if ΔE > 0 then súčasný ← ďalší
                    else súčasný ← ďalší iba s pravdepodobnosťou  $e^{\Delta E / T}$ 
    end
```

Analógia so žíhaním

- funkcia HODNOTA zodpovedá celkovej energii atómov v materiáli
- T zodpovedá teplote.
- ROZVRH určuje, ako sa znižuje teplota.
- Jednotlivé kroky v stavovom priestore zodpovedajú náhodným fluktuáciám spôsobeným teplotným šumom.
- Ak sa teplota znižuje dostatočne pomaly, materiál prijme konfiguráciu s najnižšou energiou (dokonalé usporiadanie).
- V kontexte hľadania riešenia to zodpovedá tvrdeniu, že ak ROZVRH definuje znižovanie T dostatočne pomaly, algoritmus nájde globálne optimum.

Simulované žíhanie

- určenie počtu „krokov“ – rozvrh
- v každom kroku:
 - náhodne zavedť zmenu do súčasného stavu
 - nový stav prijmi vždy, ak zlepšuje hodnotu
 - (inak) náhodne prijmi aj stav, ktorý zhoršuje hodnotu
- pomaly znižuj pravdepodobnosť prijatia stavu, zhoršujúceho hodnotu

Simulované žíhanie

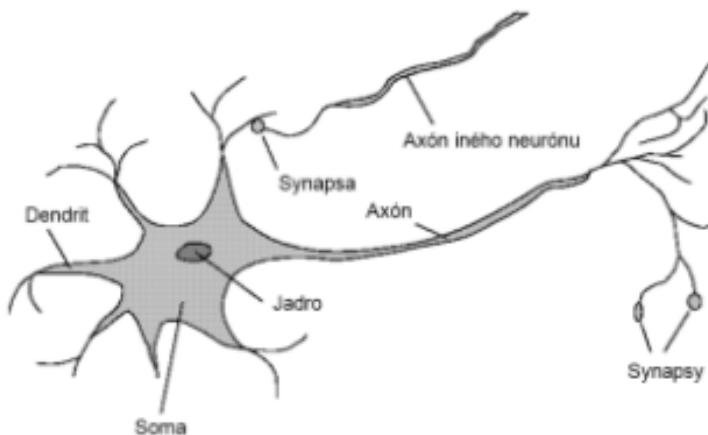
- znáhodňujúci algoritmus
- umožňuje rýchlo prehľadať veľké časti stavového priestoru
- priatím zhoršujúcich zmien sa otvára možnosť vyhnutia sa lokálnym extrémom
- prijatie zhoršujúcej zmeny môže byť zlé
 - pravdepodobnosť ich prijatia sa postupne zmenšuje

Umelé neurónové siete



- Napodobenie mozgu
- Emergentné správanie
 - Učenie sa z príkladov a paralelné spracovanie signálov mnohými prvkami
- Neurobiológia
 - Pri narodení pozostáva ľudský mozog z približne 10^{12} neurónov
 - Každý neurón môže byť spojený s ďalšími cca. 10^5 neurónmi
 - Bežný počítač: 10^9 operácií za sekundu
 - Mucha domáca: 10^{11} operácií za sekundu

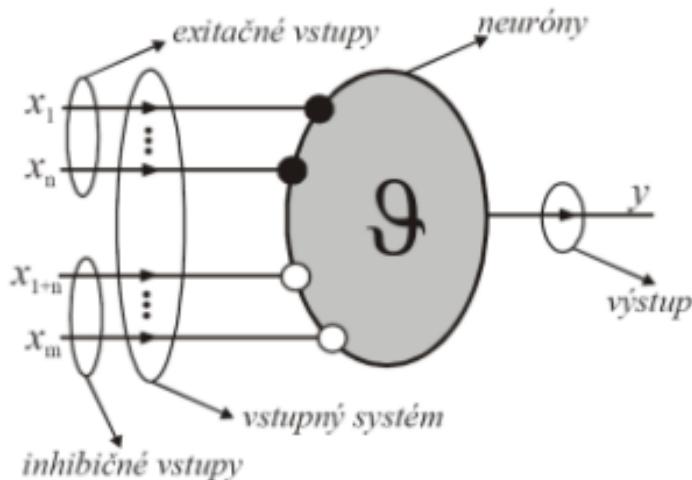
Umelé neurónové siete



Základným stavebným kameňom ľudského mozgu je neurón:

- neurón pomocou dendritov **prijíma signály** z okolia od ostatných neurónov,
- neurón **spracováva** (integruje) prijaté signály,
- neurón pomocou axónu **posiela spracované vstupné signály iným neurónom zo svojho okolia.**

Umelé neurónové siete



- excitačné a inhibičné synapsy
- prah excitácie
- výstupný impulz (cez axón)

Umelé neurónové siete

- učenie - zmeny váh synáps
- kódovanie a reprezentácia
 - Redundancia

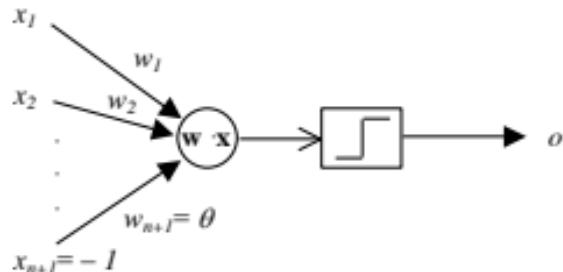
Informácia sa prenáša, prijíma a spracúva nadbytočným počtom neurónov a synáps, aby sa v prípade poškodenia siete nestratila úplne.

- Distribuovaná reprezentácia

Každý objekt sa reprezentuje celou danou sietou neurónov.

Perceptrón

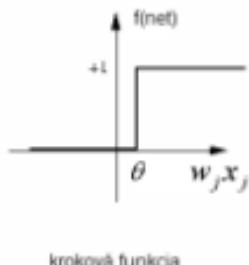
- vstupný vektor a vektor váh



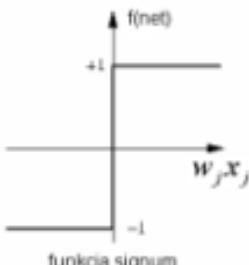
- výstup perceptrónu (f – aktivačná funkcia)

$$o = f(\text{net}) = f(\overline{w \cdot x}) = f\left(\sum_{j=1}^{n+1} w_j x_j\right) = f\left(\sum_{j=1}^n w_j x_j - \theta\right)$$

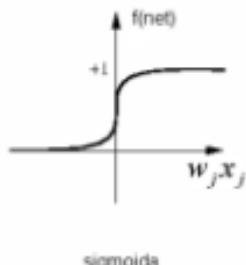
Perceptrón



kroková funkcia



funkcia signum



sigmoída

Kroková funkcia: $f(\text{net}) = \text{step}(\text{net}) = \begin{cases} 1 & \text{net} \geq 0 \Leftrightarrow \sum_{j=1}^n w_j x_j \geq \theta \\ 0 & \text{net} < 0 \Leftrightarrow \sum_{j=1}^n w_j x_j < \theta \end{cases}$

Funkcia Signum: $f(\text{net}) = \text{sign}(\text{net}) = \begin{cases} +1 & \text{net} \geq 0 \Leftrightarrow \sum_{j=1}^n w_j x_j \geq \theta \\ -1 & \text{net} < 0 \Leftrightarrow \sum_{j=1}^n w_j x_j < \theta \end{cases}$

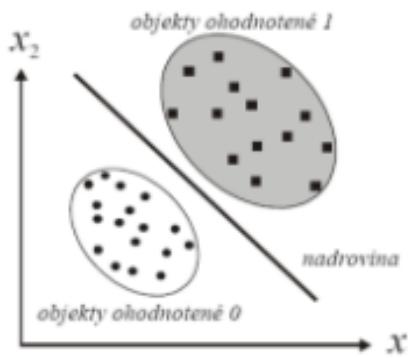
Sigmoída: $f(\text{net}) = \text{sigm}(\text{net}) = \frac{1}{1+e^{-\lambda \text{net}}}$

Perceptrón

- klasifikácia:

$$\sum_{j=1}^n w_j x_j - \theta = 0$$

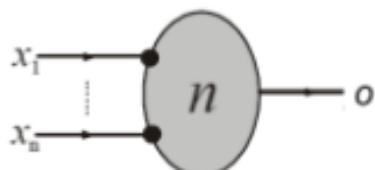
- lineárne separovateľné problémy



Boolova funkcia $f(x_1, x_2, \dots, x_n)$ je *lineárne separovateľná*, ak existuje taká rovina $w_1x_1 + w_2x_2 + \dots + w_nx_n - \theta = 0$, ktorá separuje priestor vstupných aktivít tak, že v jednej časti priestoru sú vrcholy ohodnotené 0, zatiaľ čo v druhej časti priestoru sú vrcholy ohodnotené 1.

Príklad – Perceptrón (funkcia AND)

$$o_{\text{AND}}(x_1, x_2) = f(x_1 + x_2 - 2)$$

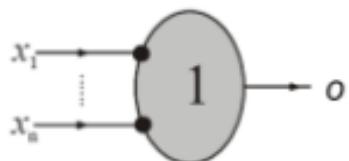


Boolova funkcia konjunkcie
 $o = x_1 \wedge \dots \wedge x_n$

x_1	x_2	$o_{\text{AND}}(x_1, x_2)$	$x_1 \wedge x_2$
0	0	$f(-2)$	0
0	1	$f(-1)$	0
1	0	$f(-1)$	0
1	1	$f(0)$	1

Príklad – Perceptrón (funkcia OR)

$$o_{\text{OR}}(x_1, x_2) = f(x_1 + x_2 - 1)$$

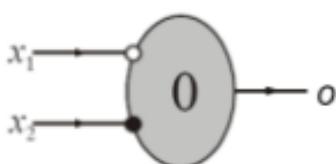


Boolova funkcia disjunkcie
 $o = x_1 \vee \dots \vee x_n$

x_1	x_2	$o_{\text{OR}}(x_1, x_2)$	$x_1 \vee x_2$
0	0	$f(-1)$	0
0	1	$f(0)$	1
1	0	$f(0)$	1
1	1	$f(1)$	1

Príklad – Perceptrón (funkcia IF-THEN)

$$o_{\text{IF-THEN}}(x_1, x_2) = f(-x_1 + x_2)$$

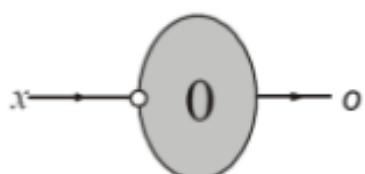


Boolova funkcia implikácie
 $o = x_1 \Rightarrow x_2$

x_1	x_2	$o_{\text{IF-THEN}}(x_1, x_2)$	$x_1 \wedge x_2$
0	0	$f(0)$	1
0	1	$f(1)$	1
1	0	$f(-1)$	0
1	1	$f(0)$	1

Príklad – Perceptrón (funkcia NOT)

$$o_{\text{NOT}}(x) = f(-x + 0)$$



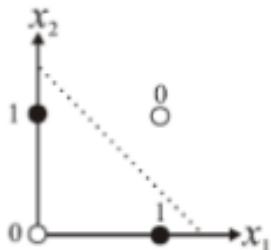
Boolova funkcia negácie
 $o = \neg x$

X	$o_{\text{NOT}}(x)$	$\neg x$
0	$f(0)$	1
1	$f(-1)$	0

Príklad – Perceptrón (funkcia XOR)

FUNCKIA XOR NIE JE LINEÁRNE SEPAROVATEĽNÁ !!!

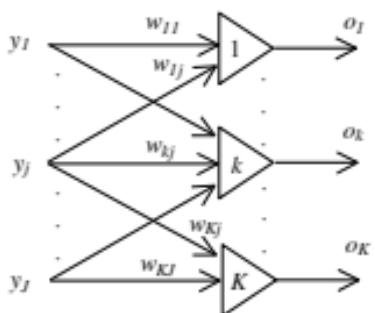
$$\varphi_{XOR}(x, y) = x \oplus y$$



#	x	y	$\varphi_{XOR}(x, y)$
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

**JE JU VŠAK MOŽNÉ REPREZENTOVAŤ POMOCOU
VIACVRSTVOVEJ DOPREDNEJ NEURÓNOVEJ SIETE**

Jednovrstvová siet'

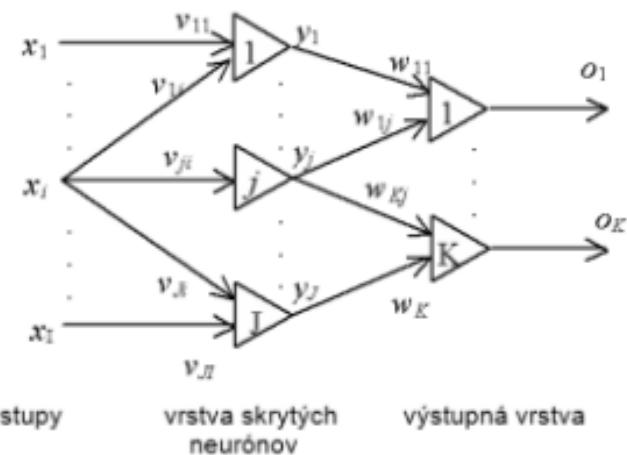


- V dopredných neurónových sietiach existujú iba dopredné spojenia medzi neurónmi
- Každý neurón jednej vrstvy vysiela signály na každý neurón nasledujúcej vrstvy.
- Chybová funkcia:

$$E_p = \frac{1}{2} \sum_{k=1}^K (d_{pk} - o_{pk})^2$$

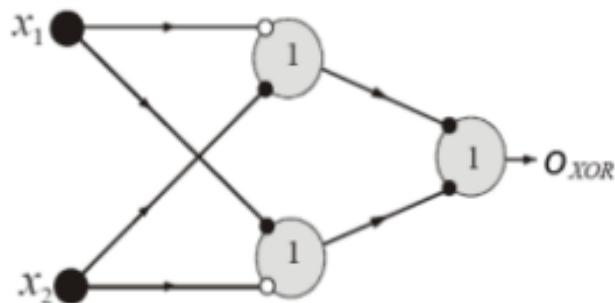
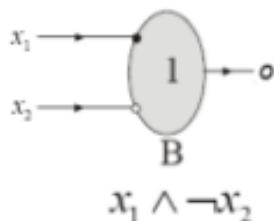
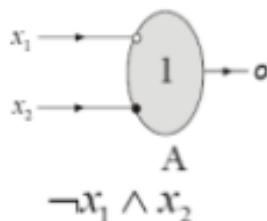
Viacvrstvové dopredné siete

- Pridanie ďalšej vrstvy neurónov – vrstva skrytých neurónov
- Neurón vysiela signály každému na nasledujúcej vrstve



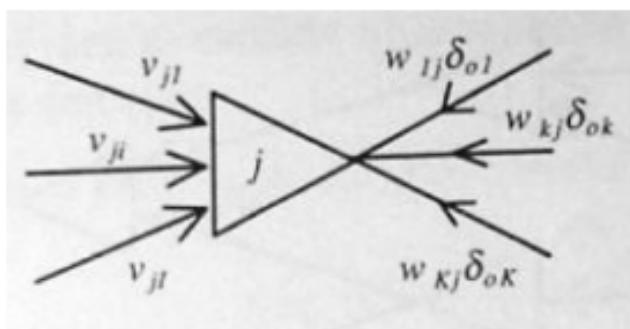
Príklad - Viacvrstvová siet' (funkcia XOR)

$$\Phi_{XOR}(x_1, x_2) = (\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$$



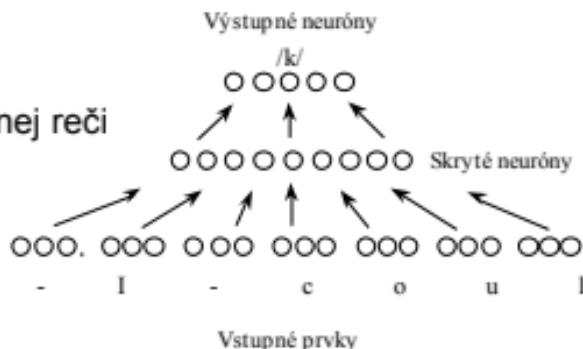
Viacvrstvové dopredné siete

- riešenie nelineárnych problémov
 - učenie spätným šírením chýb



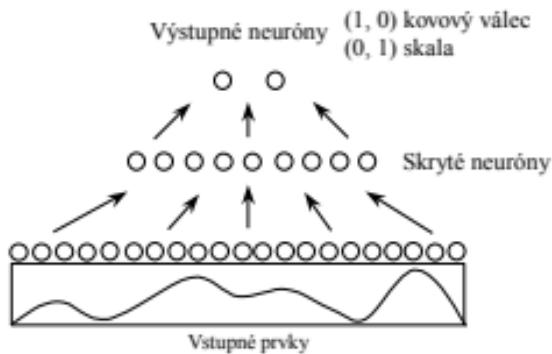
Praktické aplikácie

- NETTalk (Sejnowski, Rosenberg 1987)
 - hlasné čítanie anglického textu
 - výslovnosť v kontexte, výnimky
 - 7 vstupných skupín (26 one-hot neur.), 80, resp. 120 na skrytej
 - výstup: fonéma, artikulácia, dôraz, ...
 - úspešnosť: 95% / 78% (98% / 90%)
 - Modifikácie pre:
 - skúmanie dyslexie
 - odšumenie hovorenej reči



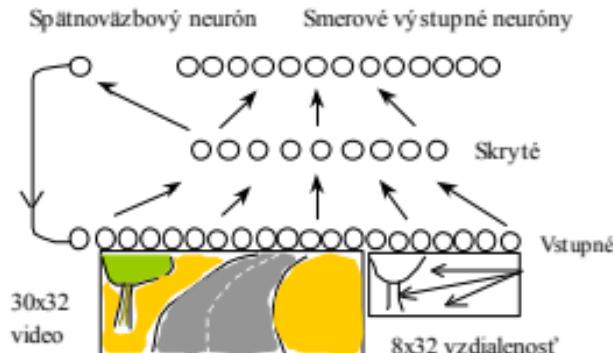
Praktické aplikácie

- podmorské sonarové signály (Gorman, Sejnowski 1988)
 - kovové objekty, skaly
 - predspracovanie signálov
 - vstup 60 čísiel, výstup 2 triedy,
24 skrytá vrstva
 - úspešnosť: 100% / 89% (človek 82%)



Praktické aplikácie

- ALVINN - autonómne pozemné vozidlo
(Pomerleau 1989)
 - vstup video a vzdialenosť, výstup smer (45 uhlov), 29 skrytá vrstva
 - 1/2 hod. tréning, schopné samostatnej jazdy po neznámom teréne (5 km/h)



Logická reprezentácia a usudzovanie

Znalostný agent

- rozumný agent, ktorého konanie sa bude zakladať na znalostiach
- znalostný agent získa schopnosť riešiť nový druh problémov naučením alebo získaním nových znalostí o prostredí
- jazyk pre znalostný agent: zápis poznatkov + odvodzovanie
- v matematickej (formálnej) logike sa skúmajú spôsoby *odvodzovania* dôsledkov zo známych axióm, t.j. logické usudzovanie
- *báza poznatkov*
 - množina zápisov faktov o svete vo forme *viet* v jazyku pre reprezentáciu poznatkov

Znalostný agent

- operácie nad bázou poznatkov:
 - pridanie novej vety
 - pýtanie sa na niečo známe
- určovanie toho, čo vyplýva z bázy poznatkov, zabezpečuje **odvodzovací stroj**

```
function ZNALOSTNÝ-AGENT(vnem) returns akcia
    static: BP, báza poznatkov
            t, počítadlo, ktoré indikuje čas; na začiatku 0

    PRIDAJ (BP, VYTVOR-VETU-O-VNEME (vnem, t))
    akcia ← ODPOVEDAJ(BP, VYTVOR-DOPYT-NA-AKCIU (t))
    PRIDAJ (BP, VYTVOR-VETU-O-AKCII (akcia, t))
    t ← t + 1
return akcia
```

Znalostný agent - úrovne abstrakcie opisu

- **znalostná úroveň** (čo vie)
 - agent zabezpečujúci riadenie vozidla môže vedieť, že do časti Bratislava Petržalka sa možno dostať z časti Staré Mesto po Starom moste
- **logická úroveň** (zápis vo vetách)
 - spája (*Starý most, Staré Mesto, Petržalka*).
- **implementačná úroveň** (technické vybavenie agenta)
 - "spaja(stary_most, stare_mesto, petrzalka)"
 - reťazec "stary_most" v dvojrozmernej tabuľke prepojení častí Bratislavы

Logika a reprezentácia poznatkov

- jazyk je prostriedok na vyjadrenie faktov okolitého sveta. To, ktorá *syntaktická jednotka jazyka* (ktorá veta) vyjadruje ktorý fakt, určuje *sémantika jazyka*.
- *proces usudzovania* musí byť taký, že nové fakty vyplývajú z doteraz známych faktov.
 - z danej množiny faktov **vyplýva** nový fakt
 - z množiny viet (reprezentujúcich danú množinu faktov) **logicky vyplýva** nová veta (reprezentujúca nový fakt).

- **odvodzovanie** (v logike) - spôsob skúmania, či z množiny faktov vyplýva iný fakt
 - musí zachovávať *pravdivosť*
 - *úplné*, ak nájde *dôkaz* každej vety logicky vyplývajúcej z množiny formúl
- **dôkaz**
 - odvodzovacími pravidlami vytvorená postupnosť formúl nad množinou pôvodných a v predchádzajúcich krokoch odvodených formúl

Logika a reprezentácia poznatkov

- báza poznatkov i jednotlivé fakty sú súčasťou okolitého sveta
- množina formúl i jednotlivé formuly sú súčasťou reprezentácie sveta
- vzťah medzi nimi:
 - **syntax** jazyka pre reprezentáciu poznatkov
 - **sémantika** jazyka pre reprezentáciu poznatkov

- **interpretácia** je zobrazenie, ktoré priraduje formulám fakty
- interpretovaním formuly sa z nej stáva **výrok** (ne- / pravdivý)
- stav sveta, pre ktorý je nejaká formula pravdivá pri danej interpretácii, predstavuje **model** (tej formuly)
- formula je:
 - **splniteľná**, ak existuje interpretácia, pri ktorej je pravdivá (má model)
 - **nesplniteľná**

Model formuly - príklad

Nech je daná formula $\forall x \exists y P(x, y)$ a interpretácia I predikátového symbolu P je takáto:

$P \in N^2$ je relácia menší ($<$), kde N je množina prirodzených čísel.

Potom interpretácia I je modelom tejto formuly, lebo v I formula vyjadruje pravdivé tvrdenie:

“pre každé prirodzené číslo existuje prirodzené číslo, ktoré je od neho väčšie”.

Na druhej strane I nie je modelom formuly $\exists y \forall x P(x, y)$, ktorej slovenská parafráza je takáto:

“existuje prirodzené číslo také, že všetky prirodzené čísla sú od neho menšie”.

- **tautológia** - formula pravdivá pri ľubovoľnej interpretácii
- **model množiny formúl** - interpretácia, ktorá je modelom každej formuly z tejto množiny
- postup určenia, či formula logicky vyplýva z nejakej množiny formúl:
 - preskúšať všetky interpretácie
 - zúžiť interpretácie na používané v báze poznatkov
 - *znalostný agent rozhoduje nie na základe sémantiky formúl, ale len na základe syntaxe*

- **odvodzovanie**
 - formálny a mechanický proces
 - neobmedzené rozsahom použitých poznatkov (počtom a zložitosťou formúl)
- jazyk *logiky* (syntax aj sémantika) a teória dôkazu
 - študuje pravidlá odvodzovania dôsledkov vyplývajúcich z množiny viet

- logika v UI
 - výroková
 - predikátová
 - temporálna (čas)
 - modálna („možno platí, že“)
 - viachodnotová
 - fuzzy (neostrá, rozmazaná)

- syntax (BNF - Backusov Naurov tvar)

formula → jednoduchá_formula | zložená_formula

jednoduchá_formula → True | False | P | Q | ...

zložená_formula → (formula) | formula spojka formula | \neg formula

spojka → \wedge | \vee | \Leftrightarrow | \Rightarrow

- terminálne symboly
 - Logické konštanty True a False,
 - výrokové premenné P, Q, R a pod.,
 - spojky \wedge , \vee , \Leftrightarrow , \Rightarrow (dvojmiestne funktry), \neg (jednomiestny funktor)
 - zátvorky (,)
- neterminálny symbol *formula*
- literál

Výroková logika

- sémantika
 - interpretácia - ohodnenie pravdivostnou hodnotou
 - pravdivostné hodnoty *Pravda, Lož*
 - (logické konštanty True, False)
- pravdivostné tabuľky

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
Lož	Lož	Pravda	Lož	Lož	Pravda	Pravda
Lož	Pravda	Pravda	Lož	Pravda	Pravda	Lož
Pravda	Lož	Lož	Lož	Pravda	Lož	Lož
Pravda	Pravda	Lož	Pravda	Pravda	Pravda	Pravda

Výroková logika – príklad

- $P \Rightarrow (Q \Rightarrow P)$

P	Q	$(Q \Rightarrow P)$	$P \Rightarrow (Q \Rightarrow P)$
Lož	Lož	Pravda	Pravda
Lož	Pravda	Lož	Pravda
Pravda	Lož	Pravda	Pravda
Pravda	Pravda	Pravda	Pravda

- Tautológia

Výroková logika - odvodzovacie pravidlá

▪ pravidlo odlúčenia (modus ponens) $\frac{\alpha, \alpha \Rightarrow \beta}{\beta}$	▪ pravidlo vovedenia dizjunkcie $\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \dots \vee \alpha_n}$
▪ pravidlo odstránenia konjunkcie $\frac{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}{\alpha_i}$	▪ pravidlo odstránenia dvojitej negácie $\frac{}{\neg \neg \alpha} \alpha$
▪ pravidlo vovedenia konjunkcie $\frac{\alpha_1, \alpha_2, \dots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_n}$	▪ pravidlo jednotkovej rezolvencie $\frac{\alpha \vee \beta, \neg \beta}{\alpha}$

Výroková logika - odvodzovacie pravidlá

- Pravidlo rezolvencie

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

α	β		$\alpha \vee \beta$		
Lož	Lož		Lož		
Lož	Pravda		Lož		
Lož	Pravda		Pravda		
Pravda	Lož		Pravda		
Pravda	Lož		Pravda		
Pravda	Pravda		Pravda		
Pravda	Pravda		Pravda		

Výroková logika - odvodzovacie pravidlá

- Pravidlo rezolvencie

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

	β	γ		$\neg\beta \vee \gamma$	
	<i>Lož</i>	<i>Lož</i>		<i>Pravda</i>	
	<i>Lož</i>	<i>Pravda</i>		<i>Pravda</i>	
	<i>Pravda</i>	<i>Lož</i>		<i>Lož</i>	
	<i>Pravda</i>	<i>Pravda</i>		<i>Pravda</i>	
	<i>Lož</i>	<i>Lož</i>		<i>Pravda</i>	
	<i>Lož</i>	<i>Pravda</i>		<i>Pravda</i>	
	<i>Pravda</i>	<i>Lož</i>		<i>Lož</i>	
	<i>Pravda</i>	<i>Pravda</i>		<i>Pravda</i>	

Výroková logika - odvodzovacie pravidlá

- Pravidlo rezolvencie

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

			$\alpha \vee \beta$	$\neg\beta \vee \gamma$	$\alpha \vee \gamma$
			<i>Lož</i>	<i>Pravda</i>	<i>Lož</i>
			<i>Lož</i>	<i>Pravda</i>	<i>Pravda</i>
			<i>Pravda</i>	<i>Lož</i>	<i>Lož</i>
			<i>Pravda</i>	<i>Pravda</i>	<i>Pravda</i>
			<i>Pravda</i>	<i>Pravda</i>	<i>Pravda</i>
			<i>Pravda</i>	<i>Pravda</i>	<i>Pravda</i>
			<i>Pravda</i>	<i>Lož</i>	<i>Pravda</i>
			<i>Pravda</i>	<i>Pravda</i>	<i>Pravda</i>

Výroková logika - odvodzovacie pravidlá

- Pravidlo rezolvencie

$$\frac{\alpha \vee \beta, \neg \beta \vee \gamma}{\alpha \vee \gamma}$$

α	β	γ	$\alpha \vee \beta$	$\neg \beta \vee \gamma$	$\alpha \vee \gamma$
Lož	Lož	Lož	Lož	Pravda	Lož
Lož	Lož	Pravda	Lož	Pravda	Pravda
Lož	Pravda	Lož	Pravda	Lož	Lož
Lož	Pravda	Pravda	Pravda	Pravda	Pravda
Pravda	Lož	Lož	Pravda	Pravda	Pravda
Pravda	Lož	Pravda	Pravda	Pravda	Pravda
Pravda	Pravda	Lož	Pravda	Lož	Pravda
Pravda	Pravda	Pravda	Pravda	Pravda	Pravda

Výroková logika - odvodzovanie

- použitie odvodzovacích pravidiel
- dôkaz formuly

```
function VÝROKOVÝ-ZNALOSTNÝ-AGENT(vnem) returns akcia
    static: BP, báza poznatkov
            t, počítadlo, ktoré indikuje čas; na začiatku 0

    PRIDAJ (BP, VYTVOR-VETU-O-VNEME (vnem, t))
    for each akcia in zoznam možných akcií do
        if ODPOVEDAJ (BP, VYTVOR-DOPYT-NA-AKCIU (t, akcia)) then
            t ← t + 1
            return akcia
    end
```

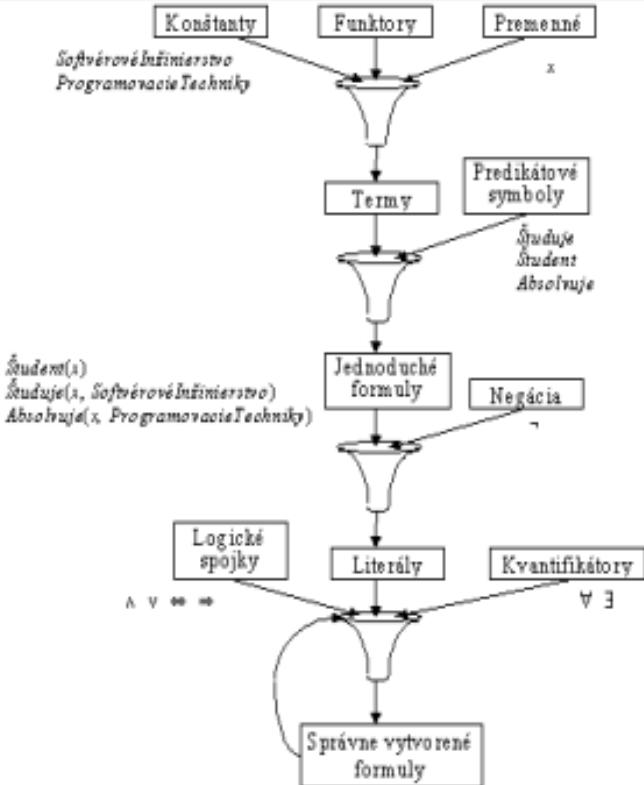
Predikátová logika

- syntax (BNF - Backusov Naurov tvar)

<i>formula</i>	\rightarrow jednoduchá_formula formula spojka formula kvantifikátor premenná, ... formula \neg formula (formula)
<i>jednoduchá_formula</i>	\rightarrow predikátový_symbol(term,...) term = term
<i>term</i>	\rightarrow funktor(term,...) konštanta premenná
<i>spojka</i>	\rightarrow \wedge \vee \Rightarrow \Leftrightarrow
<i>kvantifikátor</i>	\rightarrow \forall \exists
<i>konštanta</i>	\rightarrow Identifikátor
<i>premenná</i>	\rightarrow Identifikátor
<i>predikátový_symbol</i>	\rightarrow identifikátor
<i>funktor</i>	\rightarrow identifikátor

- sémantika:
 - interpretácia konštanty jej priraduje objekt okolitého sveta
 - interpretáciou predikátového symbolu je relácia medzi objektmi okolitého sveta
 - interpretáciou funkторa je funkcia nad objektmi okolitého sveta
- term
- jednoduchá formula

Predikátová logika



$$\forall x (\text{Študent}(x) \wedge \text{Študuje}(x, \text{Softwarevélníkarsvo})) \Rightarrow \text{Absolvuje}(x, \text{ProgramacieTechniky})$$

- **univerzálny kvantifikátor \forall**
 - „Všetky cesty vedú do Ríma.“
 - “pre ľubovoľný objekt x , ak x je cesta, tak x vedie do Ríma”.
 - $\forall x \ Cesta(x) \Rightarrow VedieDoRíma(x)$
 $Cesta(ViaAppia) \Rightarrow VedieDoRíma(ViaAppia) \wedge$
 $Cesta(ViaSalaria) \Rightarrow VedieDoRíma(ViaSalaria) \wedge$
 $Cesta(ViaCassia) \Rightarrow VedieDoRíma(ViaCassia) \wedge$
 $Cesta(ViaAurelia) \Rightarrow VedieDoRíma(ViaAurelia) \wedge \dots$
- ak je formula pravdivá, vypovedá to, čo tvorí pravú stranu implikácie, iba o objektoch, ktoré splňujú ľavú stranu

- **existenčný kvantifikátor \exists**

- „Do Ríma sa dá dôjsť po ceste.“
- “existuje cesta x taká, že x vedie do Ríma”.
- $\exists x \ Cesta(x) \wedge VedieDoRíma(x)$

$Cesta(ViaAppia) \wedge VedieDoRíma(ViaAppia) \vee$

$Cesta(ViaSalaria) \wedge VedieDoRíma(ViaSalaria) \vee$

$Cesta(ViaAurelia) \wedge VedieDoRíma(ViaAurelia) \vee \dots$

- **vnorené kvantifikátory**

- ak dva kvantifikátory v jednej formule vovádzajú tú istú premennú, „patri“ príslušný výskyt bližšiemu

$\forall x [Cesta(x) \vee \exists x \ VedieDo(Rím, x)]$

alebo

$\forall x [Cesta(x) \vee \exists y \ VedieDo(Rím, y)]$

Predikátová logika – kvantifikátory

- vzťah medzi existenčným a všeobecným kvantifikátorom
 - "všetci ľudia nemajú krídla"
 - "neexistuje taký človek, ktorý má krídla"
 - $\forall \text{ č } \neg \text{Má}(\text{č}, \text{Krídla})$
 - $\neg \exists \text{ č } \text{Má}(\text{č}, \text{Krídla})$
- pravidlá pre kvantifikátory

$$\forall x \neg F \equiv \neg \exists x F \quad \neg F \wedge \neg G \equiv \neg(F \vee G)$$

$$\neg \forall x F \equiv \exists x \neg F \quad \neg(F \wedge G) \equiv \neg F \vee \neg G$$

$$\forall x F \equiv \neg \exists x \neg F \quad F \wedge G \equiv \neg(\neg F \vee \neg G)$$

$$\exists x F \equiv \neg \forall x \neg F \quad F \vee G \equiv \neg(\neg F \wedge \neg G)$$

Predikátová logika

- reprezentácia poznatkov
 - formuly (axiómy, definície)
 $\text{PRIDAJ}(BP, \forall s \forall v \text{StarýRodič}(s, v) \Leftrightarrow \exists r \text{Rodič}(s, r) \wedge \text{Rodič}(r, v))$
 - dopyty
 $\text{ODPOVEDAJ}(BP, \exists s \text{StarýRodič}(s, Lenny))$
- reprezentácia zvláštnych druhov poznatkov
 - vyjadrenie zmien stavu sveta

Príklad – reprezentácia poznatkov v predikátovej logike

- Každý, kto je spôsobilý na jazdu a má auto, môže jazdiť. Na jazdu je spôsobilý ten, kto má vodičský preukaz. Peter má vodičský preukaz a požičal si auto. Tí, čo jazdia, nemajú problémy s dopravou. Dá sa nájsť niekto, kto nemá problémy s dopravou?
- **Opis problému:**

$$\forall x \text{ spôsobilý}(x) \wedge \text{má_auto}(x) \Rightarrow \text{jazdí}(x)$$
$$\forall x \text{ má_vodičský_preukaz}(x) \Rightarrow \text{spôsobilý}(x)$$
$$\exists x = \text{Peter} \text{ má_vodičský_preukaz}(x) \wedge \text{má_auto}(x)$$
$$\forall x \text{ jazdí}(x) \Rightarrow \text{nemá_problémy_s_dopravou}(x)$$

Otázka:

$$\exists x \text{ nemá_problémy_s_dopravou}(x)$$

- situačný počet

- vlastnosť/vzťah meniaci sa v čase sa vyjadruje predikátom s argumentom navyše

$JeNa(RobotArnold, Pitvor, S_1)$

- dôsledok:

$JeNa(RobotArnold, Pitvor, S_1) \wedge \neg JeNa(RobotArnold, Pitvor, S_2)$

- zmena stavu sveta:

$\neg JeNa(RobotArnold, Pitvor, S_1) \Rightarrow$

$JeNa(RobotArnold, Pitvor, \text{\v{D}al\v{s}iaSitu\'acia} (Vst\'up, S_1))$

- riešenie problémov - dôkaz existenčnej cieľovej formuly a postupnosť akcií

- **fakty (začiatočný stav S_0)**

"robot Arnold sa nachádza v pitvore, debna B leží na debne A,
debny B a C sú voľné."

$\text{JeNa}(\text{RobotArnold}, \text{Pitvor}, S_0)$

$\text{LežíNa}(B, A, S_0)$

$\text{VoľnýVrch}(B, S_0) \wedge \text{VoľnýVrch}(C, S_0)$

- **všeobecné tvrdenia**

$\forall x \forall y \forall s [\text{LežíNa}(x, y, s) \Rightarrow \neg \text{LežíNa}(y, x, s)]$

- **akcie a ich účinky**

$\forall x \forall s [\text{VoľnýVrch}(x, s) \Rightarrow \text{Zdvihnutý}(x, \text{ĎalšiaSituácia}(\text{Zdvihni}(x), s))]$

- **rámcové axiómy**

$\forall x \forall y \forall s [[\text{VoľnýVrch}(x, s) \wedge \text{VoľnýVrch}(y, s) \wedge x \neq y] \Rightarrow \text{VoľnýVrch}(y, \text{ĎalšiaSituácia}(\text{Zdvihni}(x), s))]$

- **ciel'**

$\exists s [\text{LežíNa}(B, A, s) \wedge \text{LežíNa}(C, B, s)]$

- **Substitúcia**

- SUBST ($\{x/Cesta61, y/Západ\}$, $Smeruje(x, y)$) =
 $Smeruje(Cesta61, Západ)$

- **Odstránenie univerzálneho kvantifikátora**

- Pre ľubovoľnú formulu α , premennú v a konštantný výraz g

$$\forall v \alpha$$

$$\text{SUBST}(\{v/g\}, \alpha)$$

- **Odstránenie existenčného kvantifikátora**

- pre ľubovoľnú formulu α , premennú v a konštantu k takú, že sa nenachádza inde v báze poznatkov

$$\exists v \alpha$$

$$\text{SUBST}(\{v/k\}, \alpha)$$

- **Vvedenie existenčného kvantifikátora**

- pre ľubovoľnú formulu α , premennú v takú, že sa nenachádza v α a konštantný výraz g

$$\alpha$$

$$\exists v \text{ SUBST}(\{g/v\}, \alpha)$$

- **Zovšeobecnené pravidlo odlúčenia**

- Nech p_i, p'_i, q sú jednoduché formuly také, že existuje substitúcia θ taká, že $\text{SUBST}(\theta, p'_i) = \text{SUBST}(\theta, p_i)$ pre všetky i

$$p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)$$

$$\text{SUBST}(\theta, q)$$

- **Hornove formuly**

- formula v báze poznatkov je buď jednoduchá formula alebo implikácia, ktorá má na ľavej strane konjunkciu jednoduchých formúl a jedinú jednoduchú formulu na pravej strane

Príklad odvodenia (dôkazu)

Predpokladajme, že platia axiómy

- (A1) Každá manželka obdivuje svojho manžela
- (A2) Dorota je manželka

treba dokázať, že Dorota obdivuje svojho manžela

Manželka(x), Obdivuje(x, y), Manžel(x), Dorota

cieľ: Obdivuje(Dorota, Manžel(Dorota))

axiómy:

- (1) $\forall x (\text{Manželka}(x) \Rightarrow \text{Obdivuje}(x, \text{Manžel}(x)))$
- (2) $\text{Manželka}(\text{Dorota})$

odstránime univerzálny kvantifikátor na (1)

- (3) $\text{Manželka}(\text{Dorota}) \Rightarrow \text{Obdivuje}(\text{Dorota}, \text{Manžel}(\text{Dorota}))$

pravidlo odlúčenia na (2) a (3)

- (4) $\text{Obdivuje}(\text{Dorota}, \text{Manžel}(\text{Dorota}))$

Dorota vo zvolenom axiomatickom systéme obdivuje svojho manžela

- **unifikačný algoritmus**

$\text{UNIFY}(p, q) = \theta$ taká,
že $\text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$

- **unifikátor θ**

$\text{UNIFY}(\text{Vedie}(x, \text{Západ}), \text{Vedie}(y, z))$
 $= \{y/x, z/\text{Západ}\}$
 $= \{y/x, z/\text{Západ}, u/\text{Cesta61}\}$
 $= \{y/\text{Cesta61}, z/\text{Západ}, x/\text{Cesta61}\}$
 $= \dots$

- **najvšeobecnejší unifikátor**

– Najmenej zužuje možnosti náhrady premenných

Unifikačný algoritmus

function UNIFY(x, y) returns substitúcia, ktorou sa stanú x a y totožné,
ak existuje VLASTNÝ-UNIFY($x, y, \{\}$)

function VLASTNÝ-UNIFY(x, y, θ) returns substitúcia,
ktorou sa stanú x a y totožné, ak existuje (pre dané θ)

inputs: x , premenná / konštantá / zložená formula
 y , premenná / konštantá / zložená formula
 θ , dosiaľ vytvorená substitúcia

if $\theta = \text{neúspech}$ **then return** neúspech
else if $x = y$ **then return** θ
else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
else if ZLOŽENÁ?(x) **and** ZLOŽENÁ?(y) **then**
return VLASTNÝ-UNIFY(ARGUMENTY[x], ARGUMENTY[y],
 VLASTNÝ-
 UNIFY(OPERÁTOR[x], OPERÁTOR[y], θ))
else if ZOZNAM?(x) **and** ZOZNAM?(y) **then**
return VLASTNÝ-UNIFY (ZVÝŠOK[x], ZVÝŠOK [y],
 VLASTNÝ-UNIFY (PRVÝ[x], PRVÝ [y], θ))
else return neúspech

Unifikačný algoritmus

```
function UNIFY-VAR(var, x,  $\theta$ ) returns substitúcia  
    inputs: var, premenná  
            x, ľubovoľná formula  
             $\theta$ , dosiaľ vytvorená substitúcia  
  
    if {var/val}  $\in \theta$  then return VLASTNÝ-UNIFY(val, x,  $\theta$ )  
    else if {x/val}  $\in \theta$  then return VLASTNÝ-UNIFY(var, val,  $\theta$ )  
    else if var sa vyskytuje hocikde v x then return neúspech  
    else return { x/var }  $\cup \theta$ 
```

- **dopredné zret'azenie**

- vychádza sa z formúl v báze poznatkov a odvodzujú sa nové dôsledky, ktoré môžu poslúžiť na odvodzovanie ešte ďalších dôsledkov

- **spätné zret'azenie**

- vychádza sa z formuly, ktorá sa má dokázať, hľadajú sa implikácie, ktoré by ju umožnili odvodiť a pre nájdené implikácie sa pokračuje pokusmi dokázať ich predpoklady

Príklad - Odvodzovanie v predikátovej logike 1.rádu

- Zákon v USA hovorí, že ak Američan predáva zbrane znepríateľeným národom, je to zločin. Krajina Nono, nepriateľ USA vlastní rakety, ktoré jej predal plukovník West – americký občan.
- Dokážte, že plukovník West je zločinec.

Príklad - Odvodzovanie v predikátovej logike 1.rádu

- ... Američan, ktorý predáva zbrane znepriateľeným národom je zločinec:
$$\text{Američan}(x) \wedge \text{Zbraň}(y) \wedge \text{Predáva}(x,y,z) \wedge \text{Znepriateľený}(z) \Rightarrow \text{Zločinec}(x)$$
- Nono ... vlastní rakety, t.j., $\exists x \text{ Vlastní}(Nono,x) \wedge \text{Raketa}(x)$:
$$\text{Vlastní}(Nono,M_1) \wedge \text{Raketa}(M_1)$$
- ... všetky rakety boli predané plukovníkom Westom
$$\text{Raketa}(x) \wedge \text{Vlastní}(Nono,x) \Rightarrow \text{Predáva}(West,x,Nono)$$
- Rakety sú zbrane:
$$\text{Raketa}(x) \Rightarrow \text{Zbraň}(x)$$
- Nepriateľ Ameriky je znepriateľený národ:
$$\text{Nepriateľ}(x,America) \Rightarrow \text{Znepriateľený}(x)$$
- West je Američan ...
$$\text{Američan}(West)$$
- Krajina Nono je nepriateľom Ameriky ...
$$\text{Nepriateľ}(Nono,America)$$

Dopredné zrečazenie - príklad

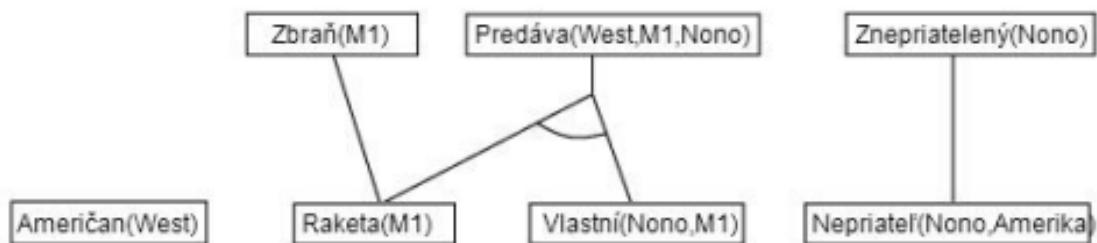
Američan(West)

Raketa(M1)

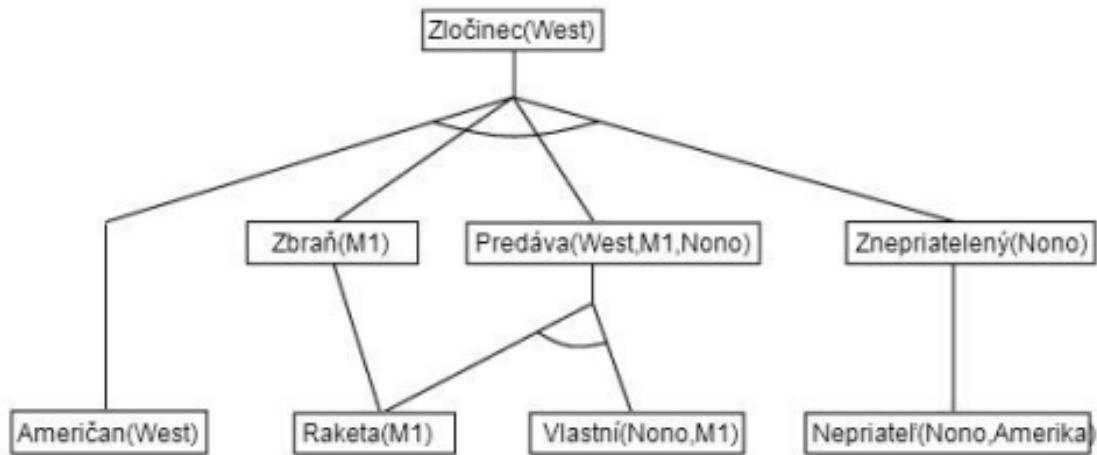
Vlastní(Nono,M1)

Nepriateľ(Nono,Amerika)

Dopredné zreťazenie - príklad



Dopredné zreťazenie - príklad



Odvodzovanie v predikátovej logike 1.rádu

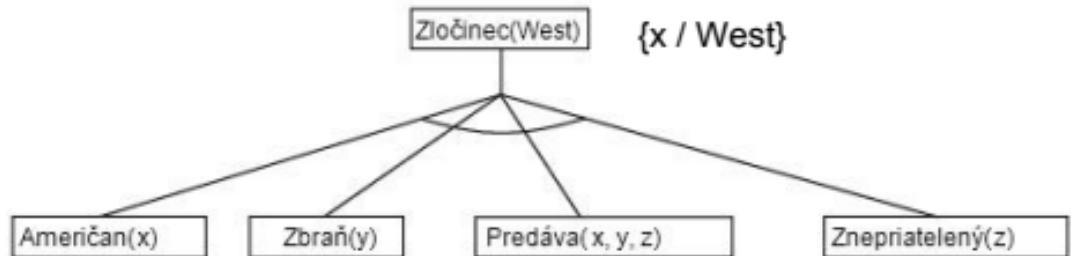
```
procedure DOPREDNÉ-ZREŽAZENIE (BP, p)
    if existuje v BP formula, ktorá je premenovaním p then return
        pridaj p do BP
    for each (p1  $\wedge$  ...  $\wedge$  pn  $\Rightarrow$  q) in BP
        také, že pre nejaké i UNIFY(pi, p) =  $\theta$  je úspešné do
            NÁJDI-A-ODVOĎ (BP, [p1, ..., pi-1, pi+1, ..., pn], q,  $\theta$ )
    end
```

```
procedure NÁJDI-A-ODVOĎ (BP, predpoklady, dôsledok,  $\theta$ )
    if predpoklady = [] then
        DOPREDNÉ-ZREŽAZENIE (BP, SUBST( $\theta$ , dôsledok))
    else for each p' in BP také, že
        UNIFY (p', SUBST( $\theta$ , PRVÝ(predpoklady))) =  $\theta_2$  do
            NÁJDI-A-ODVOĎ (BP, ZVYŠOK(predpoklady), dôsledok, ZLOŽ( $\theta$ ,  $\theta_2$ ))
    end
```

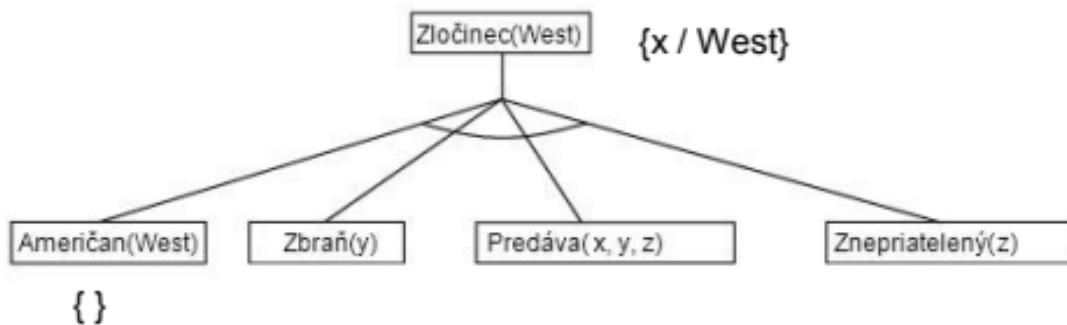
Spätné zrešťazenie - príklad

Zločinec(West)

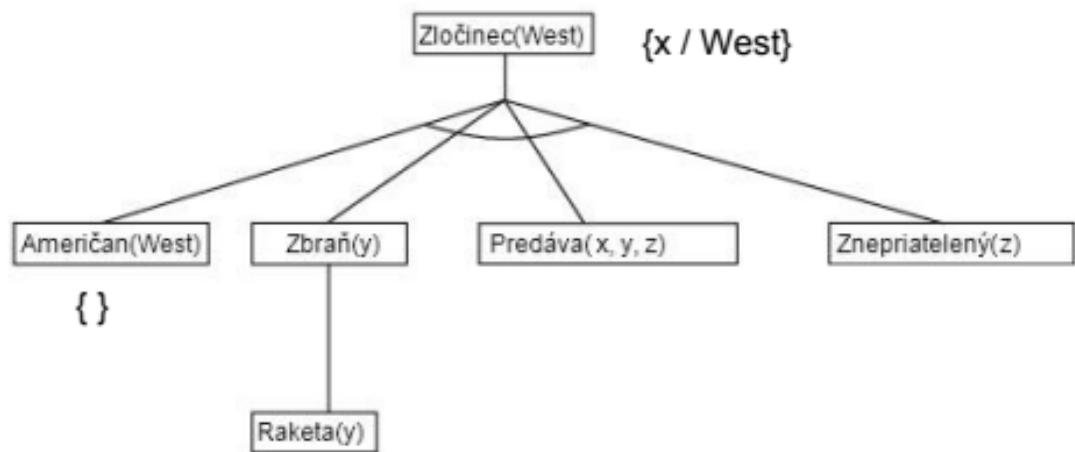
Spätné zreťazenie - príklad



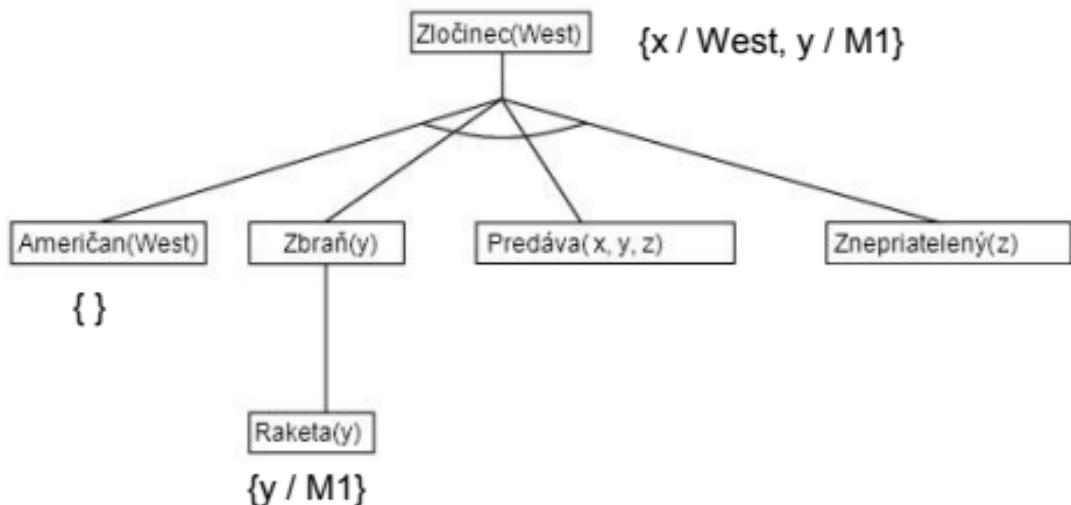
Spätné zreťazenie - príklad



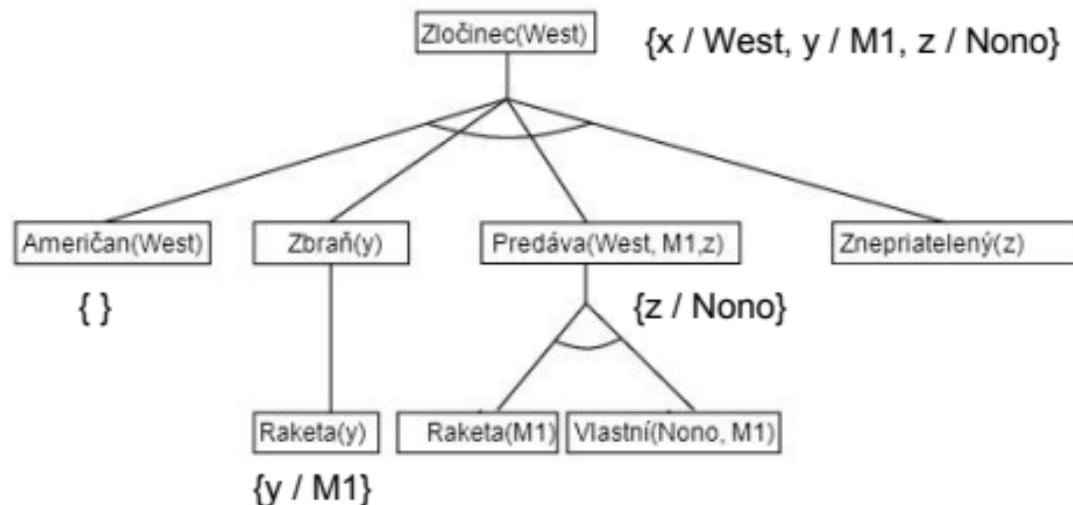
Spätné zreťazenie - príklad



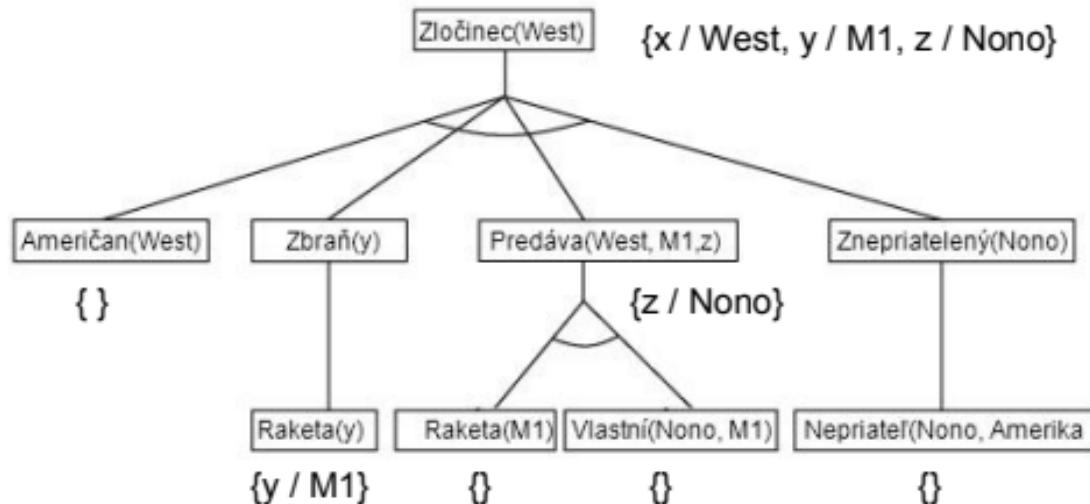
Spätné zreťazenie - príklad



Spätné zreťazenie - príklad



Spätné zreťazenie - príklad



Odvodzovanie v predikátovej logike 1.rádu

```
function SPÄTNÉ-ZREŽAZENIE(BP, q)  returns množina substitúcií  
SPÄTNÉ-ZREŽAZENIE-ZOZNAMU(BP, [q], {})
```

```
function SPÄTNÉ-ZREŽAZENIE-ZOZNAMU(BP, qzoznam, θ)
```

inputs: BP , báza poznatkov

$qzoznam$, konjunktov tvoriacich dopyt (po aplikácii θ)

θ , súčasná substitúcia

static: *odpovede*, množina substitúcií, na začiatku prázdna

if $qzoznam$ je prázdny **then return** θ

$q \leftarrow \text{PRVÝ}(qzoznam)$

for each qi' **in** BP také, že $\theta i \leftarrow \text{UNIFY}(q, qi')$ úspešné **do**

pridaj $ZLOŽ(\theta, \theta i)$ do *odpovede*

end

for each formula $(p_1 \wedge \dots \wedge p_n \Rightarrow q_i')$ **in** BP

taká, že $\theta i \leftarrow \text{UNIFY}(q, qi')$ úspešné **do**

$odpovede \leftarrow \text{SPÄTNÉ-ZREŽAZENIE-ZOZNAMU}(BP,$
 $\text{SUBST}(\theta i, [p_1, \dots, p_n], ZLOŽ(\theta, \theta i)) \cup odpovede$

end

return zjednotenie $\text{SPÄTNÉ-ZREŽAZENIE-ZOZNAMU}(BP, \text{ZVYŠOK}(qzoznam), \theta)$
pre každé $\theta \in \text{odpovede}$

pravidlá pre reprezentáciu znalostí

- AK ... TAK (IF... THEN) pravidlá možno použiť na reprezentovanie znalostí, napr:
 - ak prší tak zmokneš
- pravidlá môžu byť aj odporúčaniami, napr:
 - ak prší tak by si mal mať pršiplášť

pravidlové odvodzovacie systémy

spôsob, akým vyjadri expert nejakú znalosť, nesie so sebou dôležitú informáciu, napr:

ak osoba má horúčku a cíti bolesť v žalúdku tak môže mať infekciu.

v predikátovej logike sa to dá vyjadriť ako:

$$\forall x (\text{má_horúčku}(x) \ \& \ \text{bolest_v_žalúdku}(x) \rightarrow \text{má_infekciu}(x))$$

ak sa takáto formula skonvertuje do klauzulárneho tvaru (pozri ďalej), stratíme časť obsahu, lebo rovnakú reprezentáciu majú aj iné ekvivalentné formuly napr:

- (i) $\text{má_horúčku}(x) \ \& \ \sim \text{má_infekciu}(x) \rightarrow \sim \text{bolest_v_žalúdku}(x)$
- (ii) $\sim \text{má_infekciu}(x) \ \& \ \text{bolest_v_žalúdku}(x) \rightarrow \sim \text{má_horúčku}(x)$

všimnime si, že:

- (i) a (ii) sú logicky ekvivalentné s pôvodnou vetou
- stratili hlavnú informáciu v nej obsiahnutú.

produkčný systém

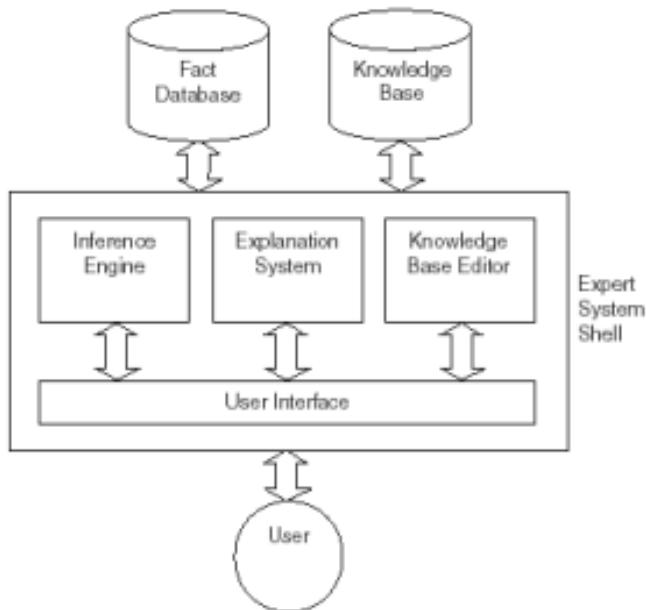
- hlavná myšlienka za dopredným/spätným produkčným systémom je:
 - využiť výhodu tvaru implikácie, v akom formuluje expert pravidlá
 - použiť túto informáciu, aby pomohla dosiahnuť cieľ.
- typicky v produkčných systémoch majú formuly tvar:
 - pravidlá
 - fakty
- produkčné pravidlá sú vyjadrenia v tvare implikácie.
 - vyjadrujú špecifické znalosti o probléme.
- fakty sú tvrdenia, ktoré nie sú vyjadrené implikáciou
 - vyjadrujú sa formulou s použitím negácie, súčinu a súčtu.

pravidlový produkčný systém

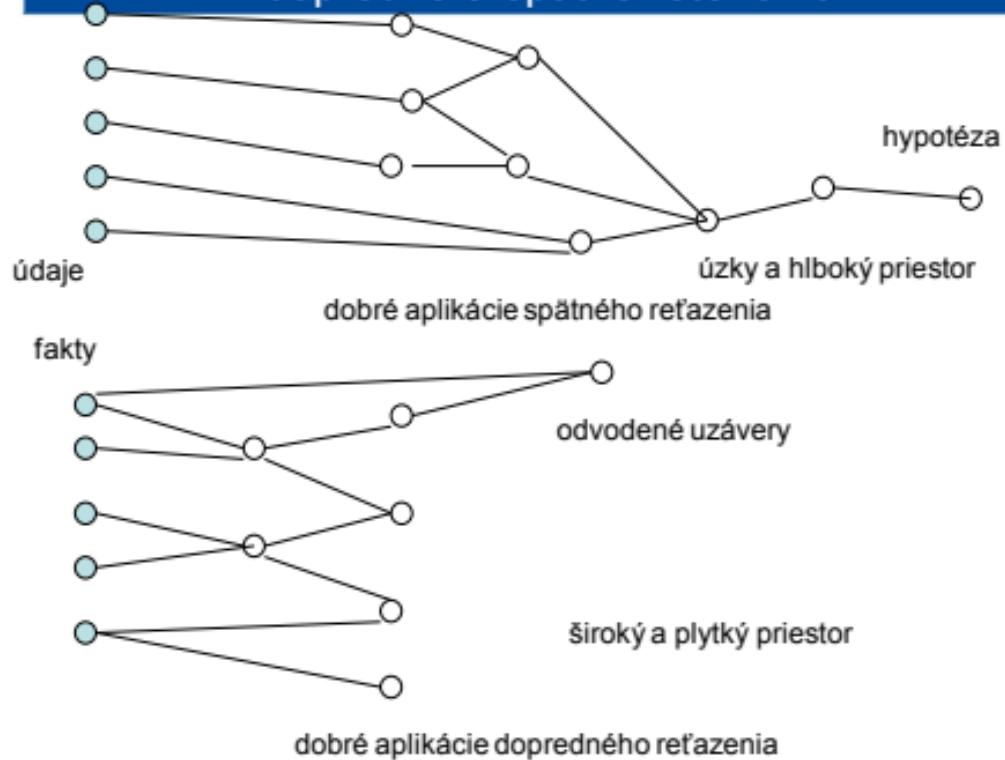
- produkčný systém používa znalosti v tvare pravidiel na to, aby poskytoval diagnózu, radu, odporúčanie na základe vstupných údajov.
- pozostáva z databázy pravidiel (báza znalostí), databázy faktov a odvodzovacieho stroja (mechanizmu), ktorý robí odvodenia nad faktami pomocou pravidiel.

architektúra expertného systému

- báza znalostí (Knowledge base): databáza pravidiel (doménové znalosti).
- vysvetľovací systém (explanation system): vysvetluje rozhodnutia, ktoré systém robí.
- rozhranie na používateľa (user interface): pomocou neho interaguje používateľ s expertným systémom.
- editor bázy znalostí (knowledge base editor): umožňuje používateľovi editovať znalosti v báze znalostí.



dopredné a spätné reťazenie



produkčný systém

dopredne reťazený

všetka komunikácia prostredníctvom pracovnej pamäti

1. [zisťovanie podobnosti - matching] nájdi všetky pravidlá, ktorých podmienková časť je splnená (pri súčasnom stave pracovnej pamäti)
2. [skončenie] ak nie je ani jedno pravidlo aplikovateľné, tak skonči
3. [rozriešenie konfliktov – conflict resolution] ak sú viac než jedno pravidlo aplikovateľné, vyber jedno (s najvyššou prioritou)
 - metapravidlá
4. [vykonanie- execution] vykonaj (odpál) vybrané pravidlo. vykonanie zmení obsah pracovnej pamäti.
 - pridaj, vymaž údaj, vykonaj v/v, ...
5. opakuj od 1.

Odvodzovanie v predikátovej logike 1.rádu

- ? úplnosť
 - odvodzovanie, založené iba na zovšeobecnenom pravidle odlúčenia, je *neúplné*
- pravidlo zovšeobecnenej rezolvencie

Pre literály p_i a q_j , pričom platí $\text{UNIFY}(p_j, \neg q_k) = \theta$:

$$\frac{\begin{array}{c} p_1 \vee \dots \vee p_j \vee \dots \vee p_m \\ q_1 \vee \dots \vee q_k \vee \dots \vee q_n \end{array}}{\text{SUBST}(\theta, (p_1 \vee \dots \vee p_{j-1} \vee p_{j+1} \vee \dots \vee p_m \vee q_1 \vee \dots \vee q_{k-1} \vee q_{k+1} \vee \dots \vee q_n))}$$

Pre atómy p_i , q_j , r_i , s_i , pričom platí $\text{UNIFY}(p_j, q_k) = \theta$:

$$\frac{\begin{array}{c} p_1 \wedge \dots \wedge p_j \wedge \dots \wedge p_{n_1} \Rightarrow r_1 \vee \dots \vee r_{n_2} \\ s_1 \wedge \dots \wedge s_{n_3} \Rightarrow q_1 \vee \dots \vee q_k \vee \dots \vee q_{n_4} \end{array}}{\text{SUBST}(\theta, (p_1 \wedge \dots \wedge p_{j-1} \wedge p_{j+1} \wedge \dots \wedge p_{n_1} \wedge s_1 \wedge \dots \wedge s_{n_3} \Rightarrow \\ r_1 \vee \dots \vee r_{n_2} \vee q_1 \vee \dots \vee q_{k-1} \vee q_{k+1} \vee \dots \vee q_{n_4}))}$$

- rezolvenčný dôkaz

- sporom

$$(BP \wedge \neg P \Rightarrow Lož) \Leftrightarrow (BP \Rightarrow P)$$

- klauzulárny tvar

- klauzula je dizjunkcia niekoľkých literálov (nemusí byť ani jeden)
 - literál je atóm alebo negácia atómu
 - všetky premenné v klauzule sú implicitne univerzálnie kvantifikované
 - medzi všetkými klauzulami je implicitná konjunkcia

Prepis formuly do klauzulárneho tvaru

1. odstránenie ekvivalencie

$$\frac{F \Leftrightarrow G}{(\neg F \vee G) \wedge (\neg G \vee F)}$$

2. odstránenie implikácie

$$\frac{F \Rightarrow G}{\neg F \vee G}$$

3. zmenšenie rozsahu negácie

$$\frac{\neg(F \vee G)}{\neg F \wedge \neg G} \quad \frac{\neg \forall x F}{\exists x \neg F} \quad \frac{\neg \neg F}{F}$$

$$\frac{\neg(F \wedge G)}{\neg F \vee \neg G} \quad \frac{\neg \exists x F}{\forall x \neg F}$$

Prepis formuly do klauzulárneho tvaru

4. premenovanie premenných
5. odstránenie existenčných kvatifikátorov

Skolemov výraz (konštanta / funkcia, presnejšie funktor)

$\exists u \forall v \forall w \exists x \forall y \exists z F(u, v, w, x, y, z)$, po náhrade

$\forall v \forall w \forall y F(b, v, w, f(v, w), y, g(v, w, y))$

6. presun kvantifikátorov doľava (pozn.: $H\{x\}$ - formula H neobsahuje prem. X)

$$\frac{}{\forall x F(x) \vee \forall y G(y)} \quad \frac{\forall x F(x) \vee H\{x\}}{\forall x \forall y (F(x) \vee G(y))} \quad \frac{\forall x (F(x) \vee H\{x\})}{\forall x (F(x) \vee H\{x\})}$$

Prepis formuly do klauzulárneho tvaru

7. odstránenie prefixu
8. prepis do konjuktívneho tvaru

$$\frac{F \vee (G \wedge H) \qquad F \wedge (G \vee H)}{(F \vee G) \wedge (F \vee H) \quad (F \wedge G) \vee (F \wedge H)}$$

9. zápis konjukcie klauzúl ako množiny
10. normalizácia premenných v klauzulách

$$\frac{\forall x (F(x) \wedge G(x))}{\forall x \forall y (F(x) \wedge G(y))}$$

Príklad - Prepis formuly do klauzulárneho tvaru

Každého, kto má rád všetky zvieratá, má niekto rád.

$$\forall x [\forall y \text{Zviera}(y) \Rightarrow \text{Má_rád}(x,y)] \Rightarrow [\exists y \text{Má_rád}(y,x)]$$

1. Odstránenie implikácie

$$\forall x [\neg \forall y \neg \text{Zviera}(y) \vee \text{Má_rád}(x,y)] \vee [\exists y \text{Má_rád}(y,x)]$$

2. Zmenšenie rozsahu negácie:

$$\forall x [\exists y \neg (\neg \text{Zviera}(y) \vee \text{Má_rád}(x,y))] \vee [\exists y \text{Má_rád}(y,x)]$$

$$\forall x [\exists y \neg \neg \text{Zviera}(y) \wedge \neg \text{Má_rád}(x,y)] \vee [\exists y \text{Má_rád}(y,x)]$$

$$\forall x [\exists y \text{Zviera}(y) \wedge \neg \text{Má_rád}(x,y)] \vee [\exists y \text{Má_rád}(y,x)]$$

Príklad - Prepis formuly do klauzulárneho tvaru

3. Premenovanie premenných

$$\forall x [\exists y \text{Zviera}(y) \wedge \neg \text{Má_rád}(x,y)] \vee [\exists z \text{Má_rád}(z,x)]$$

4. Odstránenie existenčných kvantifikátorov

$$\forall x [\text{Zviera}(F(x)) \wedge \neg \text{Má_rád}(x,F(x))] \vee \text{Má_rád}(G(x),x)$$

5. Odstránenie prefixu:

$$[\text{Zviera}(F(x)) \wedge \neg \text{Má_rád}(x,F(x))] \vee \text{Má_rád}(G(x),x)$$

6. Prepis do konjuktívneho tvaru :

$$[\text{Zviera}(F(x)) \vee \text{Má_rád}(G(x),x)] \\ \wedge [\neg \text{Má_rád}(x,F(x)) \vee \text{Má_rád}(G(x),x)]$$

P - dokazovaná formula

- prepis F_1, F_2, \dots, F_n a $\neg P$ do klauzulárneho tvaru vznikne vstupná množina U
- opakujúce sa rezolvovanie
 - výber dvoch klauzúl z U , ktoré sa dajú rezolvovať
 - ak treba, standardizácia týchto klauzúl
 - odvodenie rezolventy
 - pridanie rezolventy do U
- *pokial* (t.j. skončí ak):
 - rezolventa je prázdna klauzula (P je teoréma)
 - neexistujú dve klauzuly v U , ktoré by sa dali rezolvovať, alebo by sa dala odvodiť nová rezolventa (P nie je teoréma)
 - vyčerpali sa vopred určené výpočtové zdroje

Príklad rezolvenčného dôkazu

Každý, kto je spôsobilý na jazdu a má auto, môže jazdiť. Na jazdu je spôsobilý ten, kto má vodičský preukaz. Peter má vodičský preukaz a požičal si auto. Tí, čo jazdia, nemajú problémy s dopravou. Dá sa nájsť niekto, kto nemá problémy s dopravou?

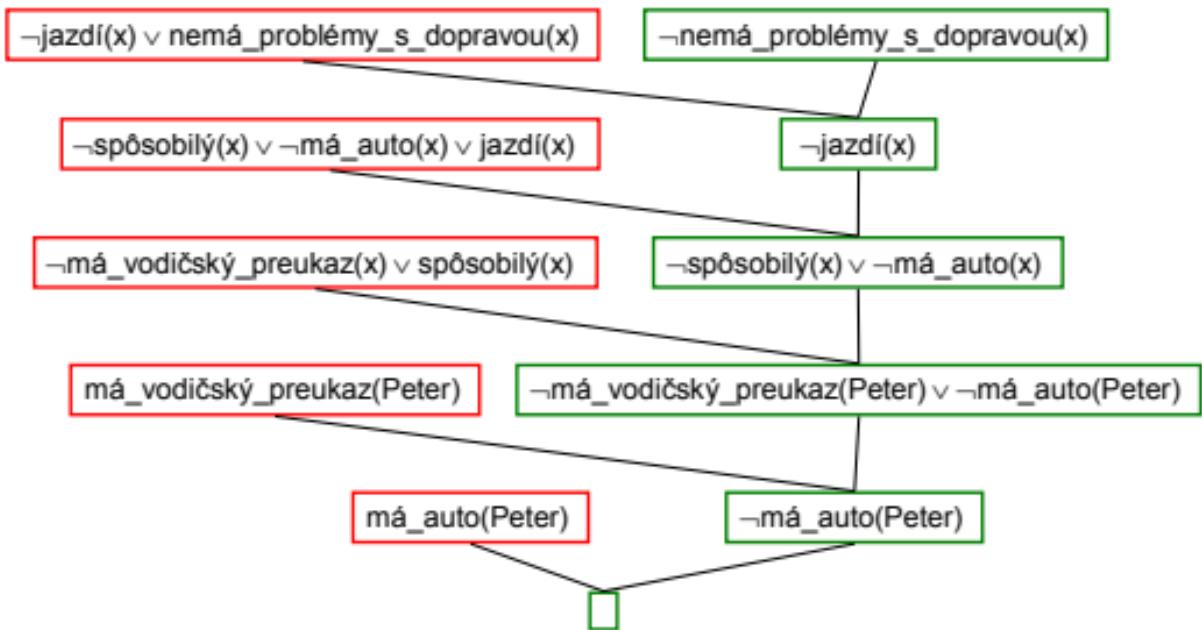
Formuly jazyka predikátovej logiky prvého rádu:

- $\forall x \text{ spôsobilý}(x) \wedge \text{má_auto}(x) \Rightarrow \text{jazdí}(x)$
- $\forall x \text{ má_vodičský_preukaz}(x) \Rightarrow \text{spôsobilý}(x)$
- $\exists x = \text{Peter} \text{ má_vodičský_preukaz}(x) \wedge \text{má_auto}(x)$
- $\forall x \text{ jazdí}(x) \Rightarrow \text{nemá_problémy_s_dopravou}(x)$
- $\exists x \text{ nemá_problémy_s_dopravou}(x)$

Úprava do klauzulárneho tvaru (pre jednoduchosť nasledujúceho obrázku tu ešte pred posledným premenovaním premenných) :

- $\neg \text{spôsobilý}(x) \vee \neg \text{má_auto}(x) \vee \text{jazdí}(x) \quad (K1)$
- $\neg \text{má_vodičský_preukaz}(x) \vee \text{spôsobilý}(x) \quad (K2) \quad (\text{napr. } x \text{ na } x1)$
- $\text{má_vodičský_preukaz}(\text{Peter}) \quad (K3)$
- $\text{má_auto}(\text{Peter}) \quad (K4)$
- $\neg \text{jazdí}(x) \vee \text{nemá_problémy_s_dopravou}(x) \quad (K5) \quad (\text{napr. } x \text{ na } x2)$
- $\neg \text{nemá_problémy_s_dopravou}(x) \quad (K6) \quad (\text{napr. } x \text{ na } x3)$

Príklad rezolvenčného dôkazu



Procedúra odvodzovania s pravidlom rezolvencie

- zefektívnenie:
rezolvenčné stratégie
 - usporadúvajúce
 - híbkové saturovanie, preferencia najmenšieho počtu literálov, jednotková preferencia
 - odsekávajúce
 - vylúčenie jalových klauzúl, vylúčenie tautológií, zahrnutie
 - obmedzujúce
 - podporná množina, vstupná rezolvencia, lineárna rezolvencia
- úplnosť stratégie definícia – otázka na jednotlivé stratégie

- **Usporadúvajúce stratégie**

- **Stratégia hĺbkového saturovania**

Predpisuje najprv odvodzovať všetky možné rezolventy v hĺbke n a až potom v hĺbke (n+1).

- **Stratégia preferencie najmenšieho počtu literálov**

Uprednostňuje spomedzi dvojíc rezolvovateľných klauzúl tú, ktorej súčet dĺžok je najmenší.

- **Stratégia jednotkovej preferencie**

Stratégia uprednostňuje použitie rezolvenčného pravidla tak, že jedna z klauzúl je len jeden literál (jednotková klauzula).

- **Odsekávajúce stratégie**

- **Stratégia vylúčenia jalových klauzúl**

Literál L je jalový, ak sa vyskytuje v množine klauzúl, ale komplementárny unifikovateľný literál $\neg L$ sa v nej nevyskytuje. Stratégia predpisuje vylúčiť všetky jalové literály zo vstupnej množiny.

- **Stratégia vylúčenia tautológií**

Predpisuje vylúčiť zo vstupnej množiny klauzuly, ktoré sú tautológie, napr. $F(x) \vee G(x,y) \vee \neg F(x)$.

- **Stratégia zahrnutia**

Klauzula J je zahrnutá v klauzule I, ak existuje taká substitúcia θ , že všetky literály v klauzule $SUBST(\theta, I)$ sa vyskytujú v klauzule J. Napr. $F(A)$ je špecifickejšia ako $F(x)$ a preto je v nej zahrnutá.

Na začiatku sa vylúčia všetky zahrnuté klauzuly zo vstupnej množiny, ak sa neskôr odvodí rezolventa, zahrnutá v nejakej klauzule, vylúči sa tiež.

- **Obmedzujúce stratégie**

- **Stratégia podpornej množiny**

Najprv sa identifikuje podporná množina (klauzuly, ktoré vznikli z negovanej dokazovanej formuly). Každé použitie rezolvenčného pravidla vezme jednu klauzulu z podpornej množiny a rezolventu potom zase pridá do podpornej množiny.

- **Stratégia vstupnej rezolvencie**

Predpisuje vziať vždy ako jednu z klauzúl na rezolvovanie klauzulu, ktorá vznikla zo vstupných formúl.

- **Stratégia lineárnej rezolvencie**

Predpisuje rezolvovať dve klauzuly nielen v prípade, ak je jedna zo vstupnej množiny klauzúl, ale aj v prípade, ak je jedna predchodkyňou druhej v strome dôkazu.

problém = hra

- nájsť riešenie problému = nájsť postup, ako (vždy) vyhrať
- situácia sa môže meniť počas hľadania riešenia

- skúmali oddávna
- Charles Babbage zamýšľal zestrojiť hráča piškvoriek (tic-tac-toe) na svojom analytickom stroji (1834), aby tak získal finančnú podporu pre svoj výskum
- Dvaja hráči hrajú na štvorčekovom hracom poli. Každý hráč má jeden druh symbolu - krúžok alebo križik. Hráči sa postupne striedajú vo vpisovaní symbolov do políčok. Cieľom je vytvoriť rad piatich* symbolov za sebou - vodorovne, zvisle alebo šikmo. Tento rad musí tvoriť priamku a nesmie byť nikde prerušený súperovým symbolom. Kto takýto rad vytvorí ako prvý, vyhral. Symbol je možné napísanie iba na voľné polička.



* my budeme v ďalšom uvažovať zjednodušenú verziu s troma symbolmi a hracom poli 3x3.

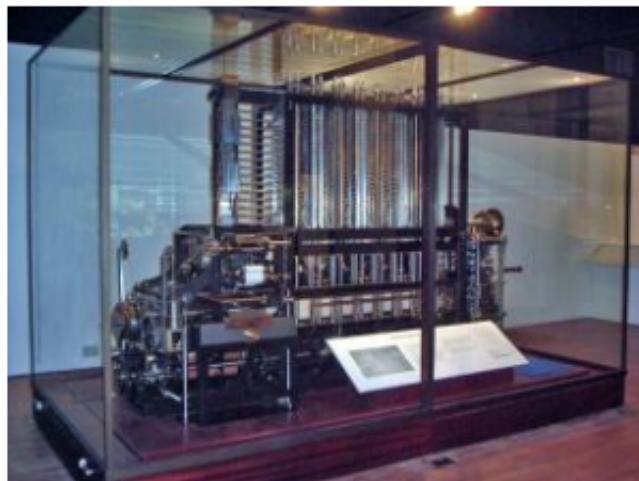
Charles Babbage

- 26 december 1791 Londýn – 18 október 1871 Londýn
- 1810 – Trinity College, Cambridge
- (Klub duchov, Klub vyhadzovačov)
- anglický matematik, filozof, vynálezca, strojní inžinier
- pôvodca myšlienky programovateľného počítača
- rozdielový stroj
- analytický stroj



difference engine

- rozdielový stroj
- výpočet hodnôt polynomiálnych funkcií
- metódou konečných rozdielov
- 1820 - 1. prototyp nedokončil
- neskôr navrhol verziu č. 2
- 1989 – 91 Londýnske múzeum vedy dalo postaviť 2 kusy



analytical engine

- analytický stroj
- programovateľný počítač
- dierne štítky (vtedy používané v tkacích stavoch)
- program na diernych štítkoch
- mechanická kalkulačka
- predvídal potrebu sekvencie, selekcie, iterácie
- barónka Ada Lovelace napísala program na výpočet postupnosti Bernoulliho čísel



šach

- šach
- Claude Shannon: prvý program, publikoval 1950

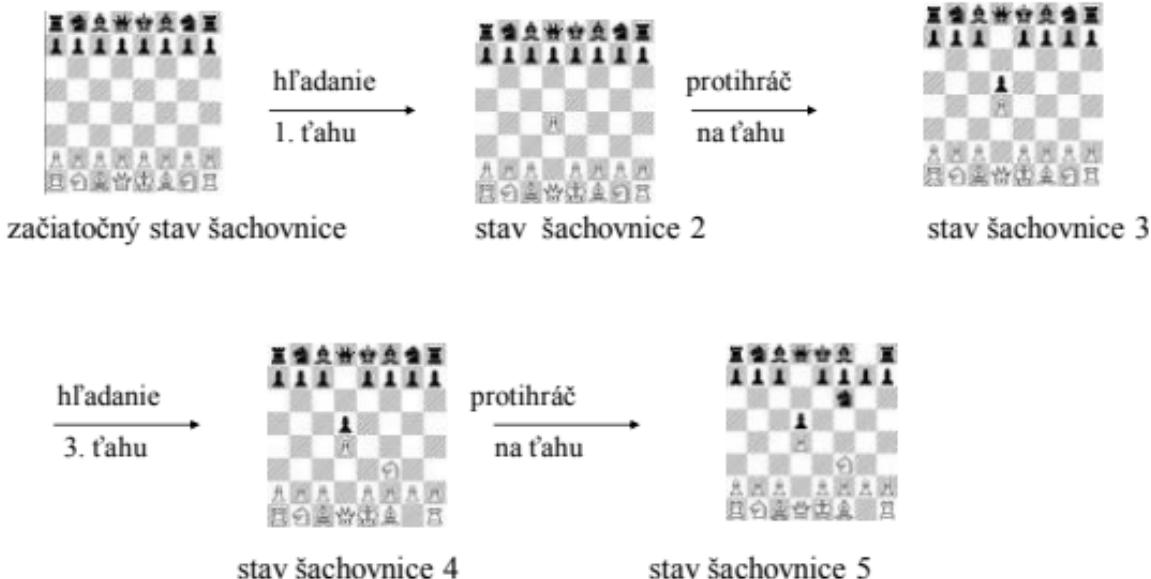


Claude Shannon

- April 30, 1916, Petoskey, Michigan – February 24, 2001
- 1936 – Bc elektrotechnika a Bc matematika, U Michigan
- 1937 – MS, MIT, diplomová práca *Symbolická analýza preklápacích obvodov*, označená za najvýznamnejšiu diplomovku storočia
- 1940 – PhD, MIT, Algebra pre teoretickú genetiku (viedol Vannevar Bush)
- 1940 – výskumník Princeton
- americký matematik, elektroinžinier, kryptograf
- cez 2. svetovú vojnu - Bellove laboratóriá
- 1948 - matematická teória komunikácie
- teória informácie, entropia



šach – hľadanie najlepších ťahov



- Claude Shannon: prvý program, publikoval 1950
- 1957 – Newell a Simon predpovedali, že počítač sa stane majstrom sveta do 10 rokov
- 1958 – IBM 704 prvý počítač, hrajúci šach
- 1967 – Mac Hack dokázal sa zúčastniť turnaja
- 1983 – Belle získalo status šachového majstra (2250 bodov), Bell Labs, počítač PDP 11
- ½ 80' – CMU začal vývoj toho (Chip Test, Hlboká myšlienka), čo sa neskôr preslávilo ako Hlboká modrá.
- 1989 – projekt prešiel k IBM (veľká modrá firma)
- 1997.5.11 – Garry Kasparov prehral 2.5:3.5

- $\sim 5 \times 10^{20}$ možných pozícií
- 1952 Arthur Samuel program pre IBM 701
- 1954 Arthur Samuel program pre IBM 704
 - (10k hlavnej pamäti)
 - mechanizmus učenia sa (učí sa vlastnú vyhodnocovaciu funkciu)
- dlho nič – poklona Samuelovi

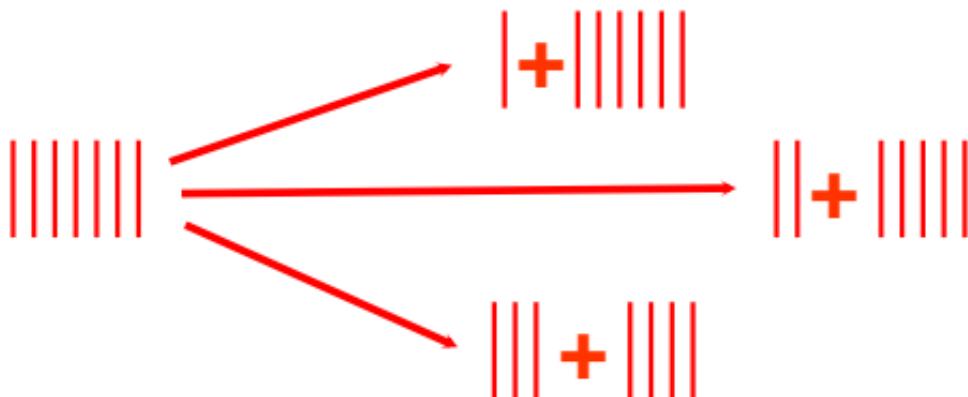
- 1989 - Jonathon Schaeffer: Chinook
 - databáza otvorení
 - hľadanie v strednej hre
 - databáza koncoviek
- 1992 – Chinook vyhral turnaj
- 1994 – aj proti majstrovi sveta
- 2007 - Jonathon Schaeffer et al. v Science:
 - Dáma je vyriešená!
 - Perfektná hra oboch hráčov zaručuje remízu.

- hranie hry:
- súper sa neustále snaží zmaríť každý ťah súpera
- 1944 – John (János) von Neumann navrhol metódu hľadania riešenia:
 - maximalizuje pozíciu hráča a minimalizuje pozíciu protihráča (odtiaľ **minimax**)



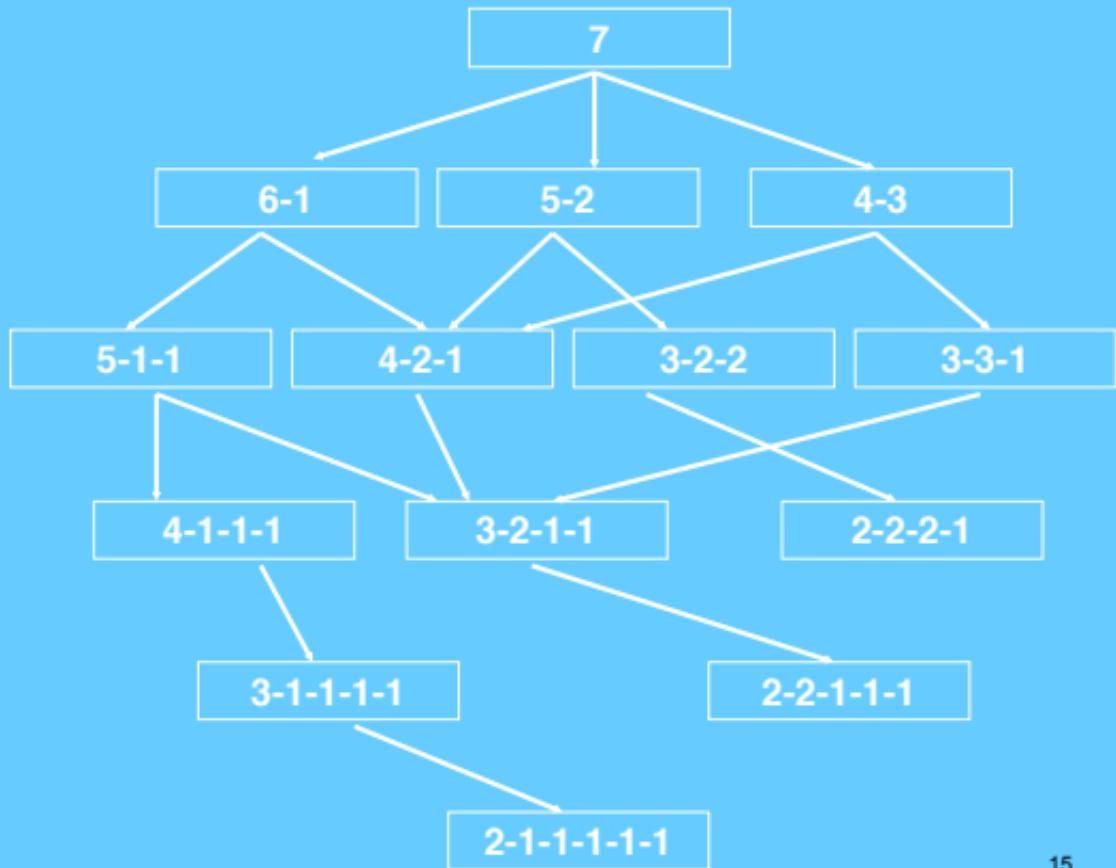
príklad ním

- jednoduchá hra (pochádza zo starovekej Číny, fan-tan, zápalky)
- začína s jednou kôpkou paličiek
- v každom kroku hráč musí vybrať kôpku tak, že rozdelí paličky do dvoch neprázdných nerovnakých kôpok



príklad nim

- ak hráme hru len so 7 zápalkami/paličkami, celý priestor je dosť malý na to, že môžeme nakresiť úplný strom hry
- kvôli úspore nakreslíme graf hry (jeden stav sa reprezentuje len raz)



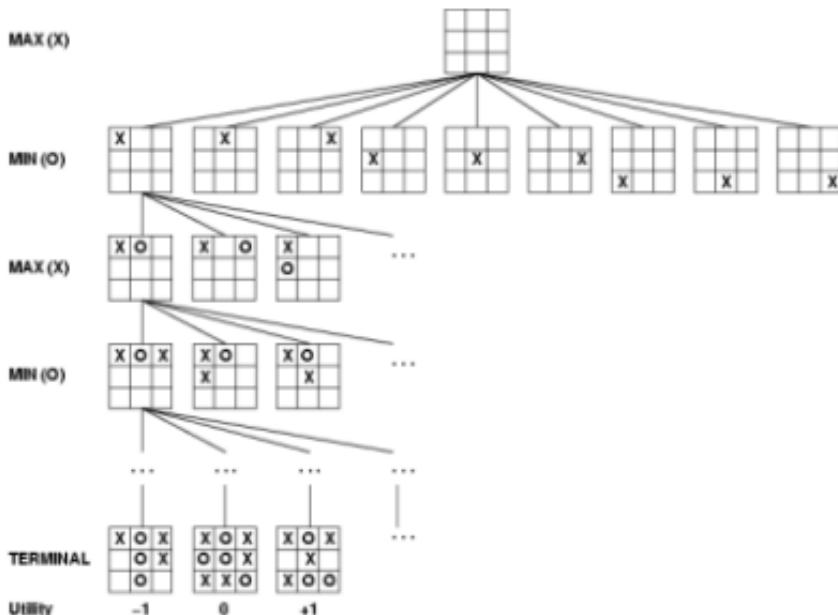
graf hry – čo s ním?

- Čo s grafom hry? Ako nám pomôže pri rozhodovaní, ako hrať (a vyhrať)?
- podieme na minimax
- potrebujeme ohodnotenie pozície:
 - funkcia užitočnosti, výhodnocovacia funkcia, atď.
- pomenujme hráčov Min a Max

vyhodnocovacia funkcia f()

- vyhodnocovacia (bodovacia) funkcia $f()$ dáva vyššie hodnoty pre lepšie pozície z pohľadu Maxa
- predpokladajme:
 - 1 – výhra pre Maxa
 - 0 – výhra pre Mina
- ide iba o porovnávanie hodnôt
- pri inej hre môže byť
 - 1 – výhra
 - 0 – remíza
 - -1 – prehra

zjednodušené piškvorky: strom hry



minimax – základná myšlienka

- hráč Max si vyberá najlepší dostupný ťah
 - čiže vyberie ťah vedúci do nasledujúceho stavu s najvyššou užitočnosťou
 - čiže hodnota uzla **Maxa** je **maximum** spomedzi hodnôt nasledujúcich možných stavov
 - čiže maximum nasledovníkov uzla v strome hľadania

minimax – základná myšlienka

- hráč Min si tiež vyberá najlepší dostupný ťah
 - čiže najhorší dostupný z pohľadu Maxa
 - čiže vyberie ťah vedúci do nasledujúceho stavu s najnižšou užitočnosťou
 - čiže hodnota uzla **Mina je minimum** spomedzi hodnôt nasledujúcich možných stavov
 - čiže minimum nasledovníkov uzla v strome hľadania

minimax – základná myšlienka

- priebeh hry:
- hráči Max a Min sa striedajú,
 - začína Min
- je to jedno, kto začína, podstata zostáva
- Treba vygenerovať úplný Min/Max strom
- predpokladajme:
 - 1 – výhra pre Maxa
 - 0 – výhra pre Mina

MIN



MAX



MIN



MAX



MIN



MAX



0 (prehra MAXa)

zhodnotenie vyhodnotenia stromu hry

- začiatočný uzol Min má hodnotu +1
- všetky ťahy Mina vedú do stavov s hodnotou +1 pre Maxa
- Min nedokáže zabrániť prehre
- z ohodnotení uzlov v strome sa dá odčítať najlepší ťah v každom kroku

algoritmus minimax

```
function MiniMax-Rozhodnutie(stav) returns operátor
    v ← Max-Hodnota(stav)
    return op v Nasledovníky(stav) s hodnotou v
function Max-Hodnota(stav) returns bodová-hodnota
    if cielový-Test(stav) then return Bodovacia-Funkcia(stav)
    v ← -∞
    for each s in Nasledovníky(stav) do
        v ← Max(v, Min-Hodnota(s))
    end
function Min-Hodnota(stav) returns bodová-hodnota
    if cielový-Test(stav) then return Bodovacia-Funkcia(stav)
    v ← +∞
    for each s in Nasledovníky(stav) do
        v ← Min(v, Max-Hodnota(s))
    end
    return v
```

vlastnosti algoritmu minimax

- Úplný? Áno (ak je strom konečný)
-
- Optimálny? Áno (proti optimálnemu protihráčovi)
-
- Časová zložitosť? $O(b^h)$
-
- Pamäťová zložitosť? $O(bh)$ (hľadá sa do hĺbky)
-
-
- Šach: $b \approx 35$, $h \approx 100$ pri "rozumných" hráčach
→ presné riešenie je úplne neschodné
-

zhodnotenie minimaxu

- pekné, ale...
- skutočné hry majú stromy hľadania omnoho širšie a hlbšie než nim
- nie je možné ohodnotiť celý strom
- čo s tým?
 - stanoviť hranicu, do akej hĺbky sa hľadá
 - koncový stav nie je koncový (kde skončí hra výhrou/prehrou), je koncový (kde došlo k useknutiu)

ohraničený minimax

- koncové stavy už nebudú istá výhra/prehra
 - v skutočnosti budú, len to nevieme s dostupnými výpočtovými zdrojmi rozhodnúť
- musí sa heuristicky/približne ohodnotiť kvalitu koncových stavov
- vyhodnotenie môže byť náročné (na vymyslenie a potom aj na čas výpočtu), najmä pre prípady nie jasnej výhry/prehry
- nasleduje umelý príklad ohraničeného minimaxu

heuristická vyhodnocovacia funkcia

- Funkcia e : stav $s \rightarrow$ číslo $e(s)$
- $e(s)$ je heuristika, ktorá odhaduje, ako nádejnejší je stav s pre Maxa
 - $e(s) > 0$ znamená, že stav s je nádejnejší pre Maxa (čím vyššia hodnota, tým nádejnejší/lepší)
 - $e(s) < 0$ znamená, že stav s je nádejnejší pre Mina
 - $e(s) = 0$ znamená, že stav s je neutrálny

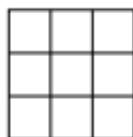
príklad e_{nim}

$e_{nim}(s) =$

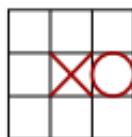
počet riadkov, stĺpcov a uhlopriečok voľných pre Maxa

- (mínus)

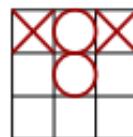
počet riadkov, stĺpcov a uhlopriečok voľných pre Mina



$$8 - 8 = 0$$



$$6 - 4 = 2$$



$$3 - 3 = 0$$

ako navrhnuť vyhodnocovaciu funkciu?

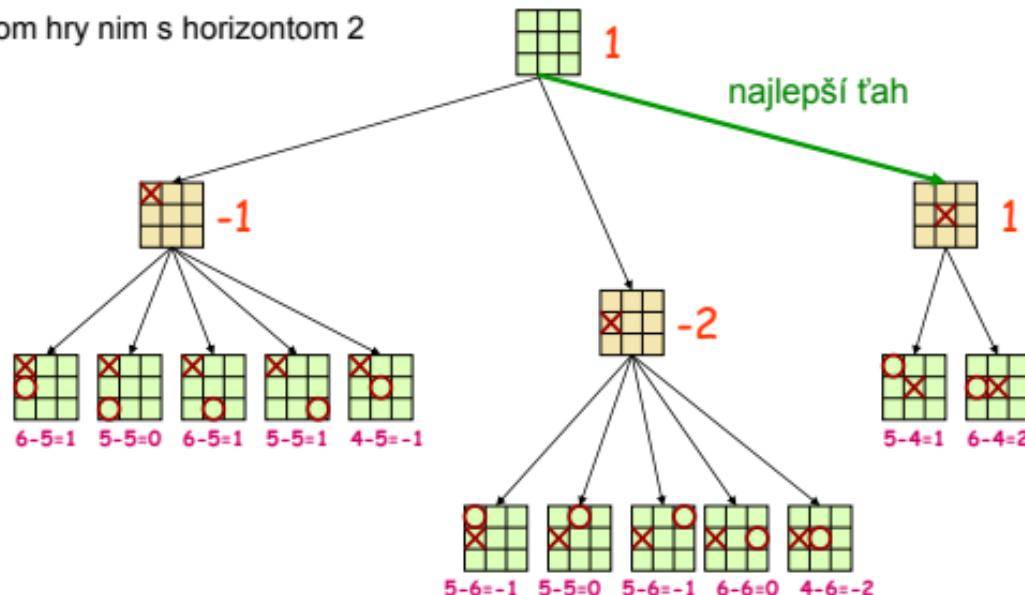
- zvyčajne ako vážený súčet "črt":

$$e(s) = \sum_{i=1}^n w_i f_i(s)$$

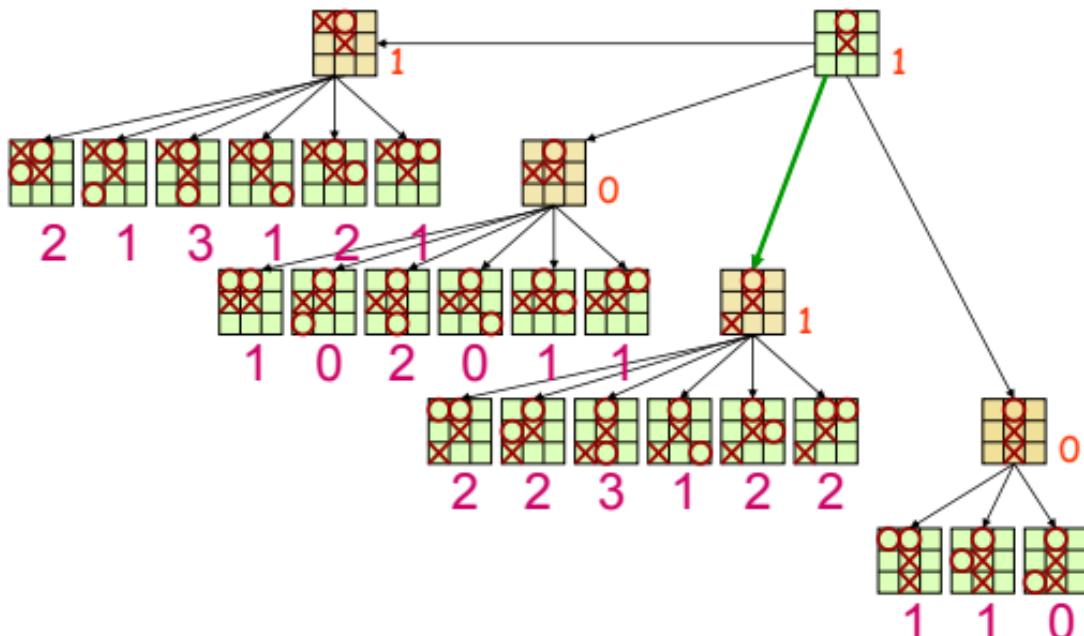
- Črty f_i môžu zahŕňať
 - počet kameňov každého možného druhu
 - počet možných ťahov
 - počet ovládaných políčok

príklad e_{nim}

strom hry nim s horizontom 2



príklad e_{nim} pokračovanie



príklad šachovej heuristickej funkcie



- čierny má:
 - 5 sedliakov, 1 strelca, 2 veže
- skóre = $1*(5)+3*(1)+5*(2)$
 $= 5+3+10 = 18$

biely má:

- 5 sedliakov, 1 vežu
- skóre = $1*(5)+5*(1)$
 $= 5 + 5 = 10$

celkové ohodnotenie pozície:

$$\text{čierny} = 18 - 10 = 8$$

$$\text{biely} = 10 - 18 = -8$$

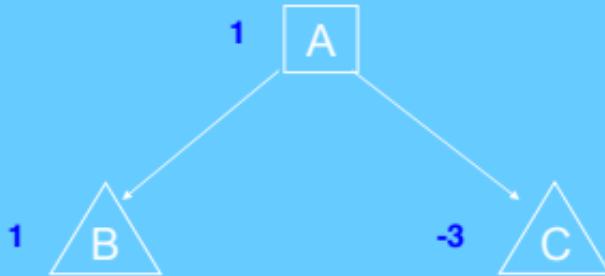
$$e(\text{pozícia}) = -8$$

Zapamätávanie hodnôt zdola nahor. Prečo?

- pri každom vnútornom uzle N sa zapamäta hodnota najlepšieho stavu, ktorý môže Max dosiahnuť v hĺbke h , ak Min hrá svoju najlepšiu hru (podľa rovnakého kritéria ako Max)
- takáto zapamätaná hodnota je lepším odhadom nádejnosti stav(N) než $e(\text{stav}(N))$

MAX

MIN



hodnoty "koncových" stavov dá
vyhodnocovacia funkcia



= koncová pozícia



= hráč



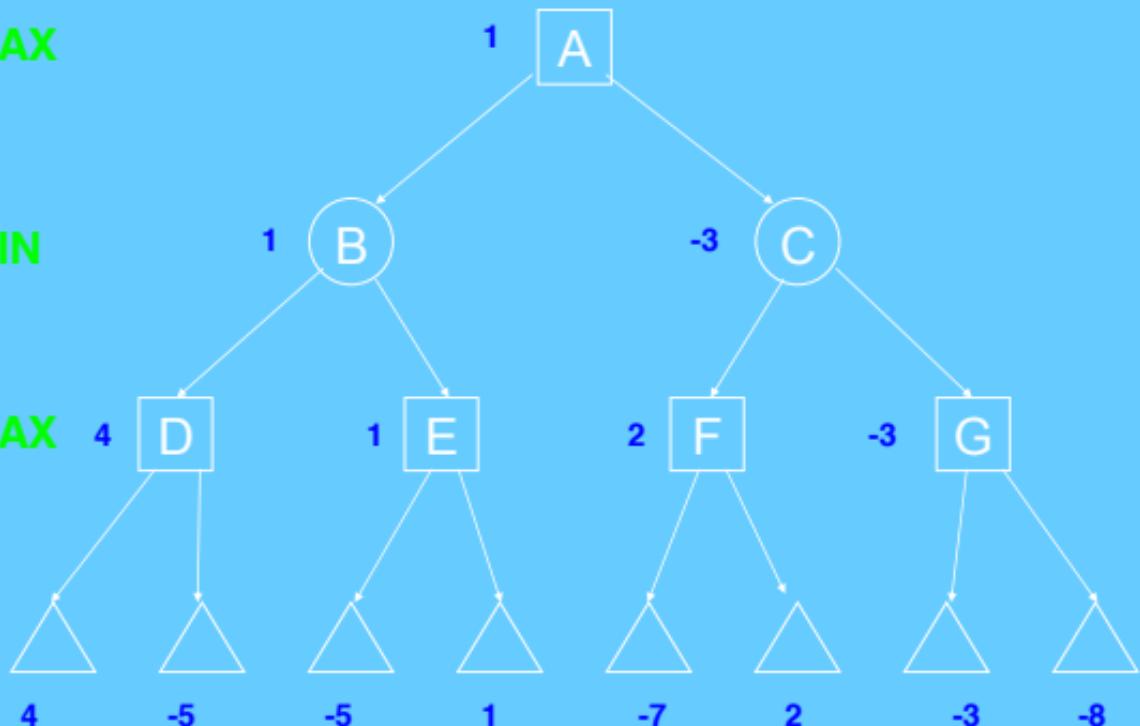
= protihráč

35

MAX

MIN

MAX



△ = koncová pozícia

□ = hráč

○ = protihráč

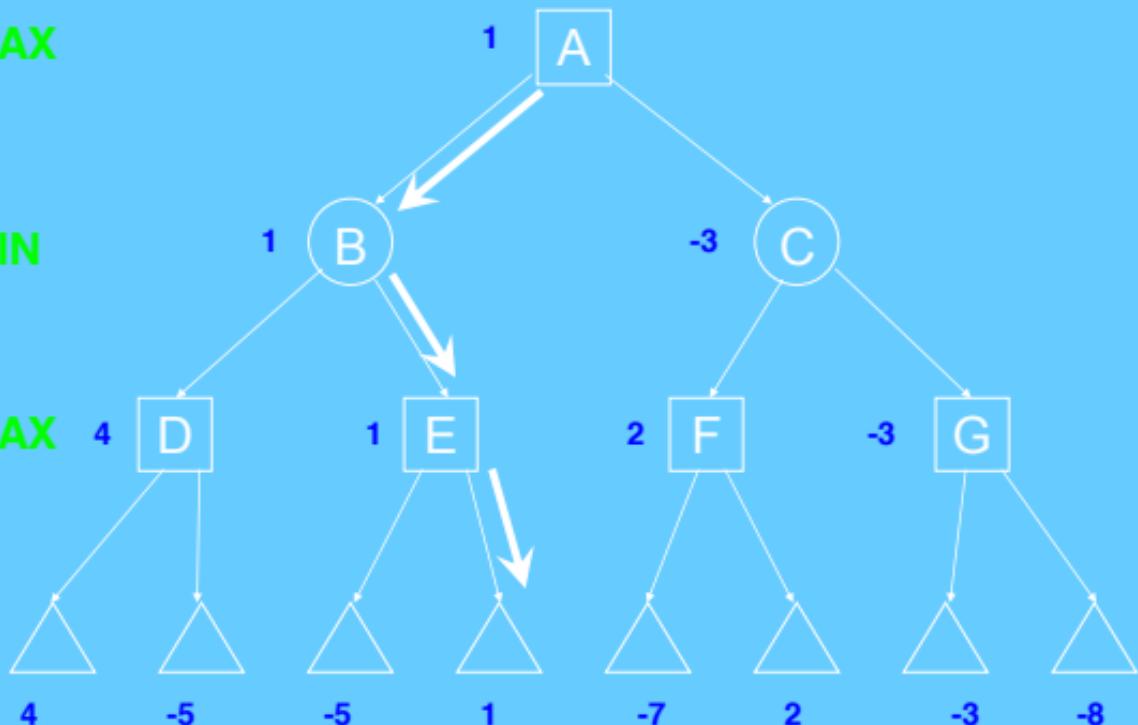
príklad: ohraničený minimax

- ak hrajú oba hráči svoje najlepšie ťahy, ako bude prebiehať hra?

MAX

MIN

MAX



△ = koncová pozícia

□ = hráč

○ = protihráč

príklad: ohraničený minimax

- dokonalá hra viedie do koncového uzla s rovnakým ohodnotením, ako má začiatočný uzol
- aj všetky uzly po ceste hry majú také isté ohodnotenie
- v podstate, to je význam hodnoty koreňa

ohraničený minimax

- minimax do stanovenej hĺbky:
- treba vytvoriť strom hry a potom popridávať minimaxové hodnoty, aby sme zistili, ako sú ohodnotené tahi zo súčasnej pozície.

prečo minimax nestačí?

- efektívnosť hľadania
 - stromy hry sú veľmi veľké
 - vyhodnocovanie pozícií je náročné a dlho trvá
- ako sa dá zmenšiť počet vyhodnocovaných uzlov?
- ohraničovanie hľadania do hĺbky má slabiny
 - prečo?
- ako ohraničovať rozrastanie prehľadávanej časti stromu do šírky?
 - α/β hľadanie

prečo minimax nestací?

- hodnoty sme pridávali zdola nahor od listov po prehľadaní do šírky
- nevýhoda: vyžaduje mnoho pamäti
 - spomeňme si na rozdiel v pamäťovej náročnosti hľadania do šírky a do hĺbky
- minimax pri prehľadávaní stromu hry do hĺbky by potreboval o mnoho menej pamäti
- minimax sa naozaj dá urobiť aj pri prehľadávaní do hĺbky
- ale: zníženie potreby pamäti nie je jediná výhoda:

MAX

MIN

MAX

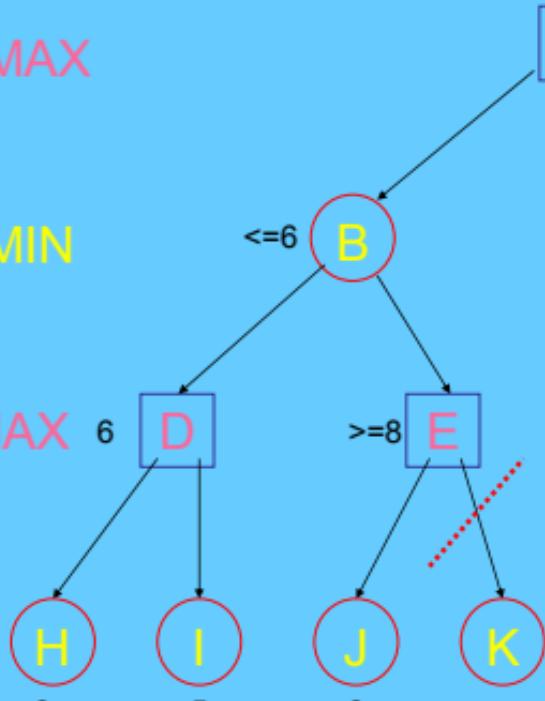
6

5

8

□ = hráč

Smeričnosť
či uvedomovanie?



keď odhalíme, že $e(D) = 6$

usúdime, že $e(B) \leq 6$

keď odhalíme, že $e(J) = 8$

usúdime, že $e(E) \geq 8$

rozvíjanie E sa môže skončiť,
pretože najlepšia hra nepôjde cez E

hodnota K je irrelevantná – odsekni
ho!

○ = protihráč

ako zlepšiť minimax?

- ak prehliadame strom do hĺbky, nemá zmysel vyhodnocovať uzol K.
- nech by bola hodnota K hocijaká, žiadna hra cezeň nepôjde
- uzol K možno odseknúť, t.j. nebude sa rozvíjať
- v (stave reprezentovanom) uzle B, lebo Min nikdy nezvolí E; pretože J je lepší než D pre Maxa, Min mu nesmie umožniť takú príležitosť
- zdá sa, že ide o úsporu len jedného uzla. Ale K môže nachádzať ešte vysoko nad stanovenou hranicou hĺbky hľadania. Vtedy sa usekne celý podstrom s počtom uzlov exponenciálne závislým od jeho hĺbky

zlepšenie minimaxu

- Predpokladajme, že by sme robili hľadanie do šírky. Mohli by sme takto osekávať?
- Nie! Osekávanie sa opiera o to, že sme už vyhodnotili uzol D na základe vyhodnotenia podstromu od ním.
- Takýto spôsob osekávania je príkladom alfa-beta osekávania. Opiera sa o hľadanie do hĺbky so stanovenou obmedzenou hĺbkou

zlepšenie minimaxu

- predpokladajme, že
 - uzly K a J by sa vyhodnocovali v opačnom poradí
 - dalo by sa podobne usekávať?
- závisí od hodnoty K
- predpokladajme, že K dostane hodnotu 2 a rozvíja sa skôr:

MAX

MIN

MAX

6

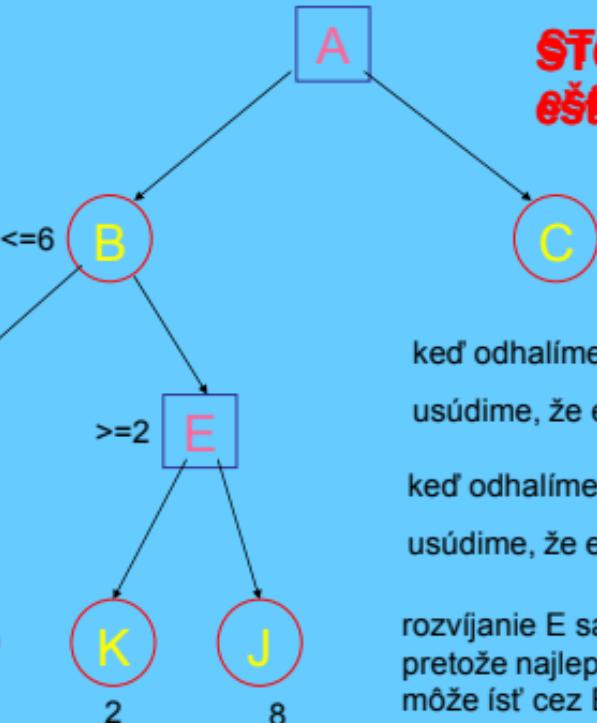
5

2

8

 = hráč

 = protihráč



STOP! Čo sa dá ešte usudzovať!?

keď odhalíme, že $e(D) = 6$

usúdime, že $e(B) \leq 6$

keď odhalíme, že $e(K) = 2$

usúdime, že $e(E) \geq 2$

rozvíjanie E sa nemôže skončiť,
pretože najlepšia hra stále ešte
môže ísiť cez E

Hodnota J je relevantná – nesekáme

zlepšenie minimaxu

- Ak K dostalo hodnotu 2 a rozvíjalo sa skôr (ako J), nenastali podmienky na useknutie nasledovníka E.
- Aby sa dalo čo najviac usekávať, treba začať rozvíjať tie nasledovníky, ktoré sú najlepšie pre ich predchodcu
 - to však nevieme
 - zišla by sa heuristika usporadúvajúca nasledovníky
- ak by sme to vedeli, hľadanie by mohlo ísiť do väčšej (až dvojnásobnej) hĺbky
 - napr pri šachu by to znížilo faktor vetvenia z 30 na 8
 - čo je významné zlepšenie!

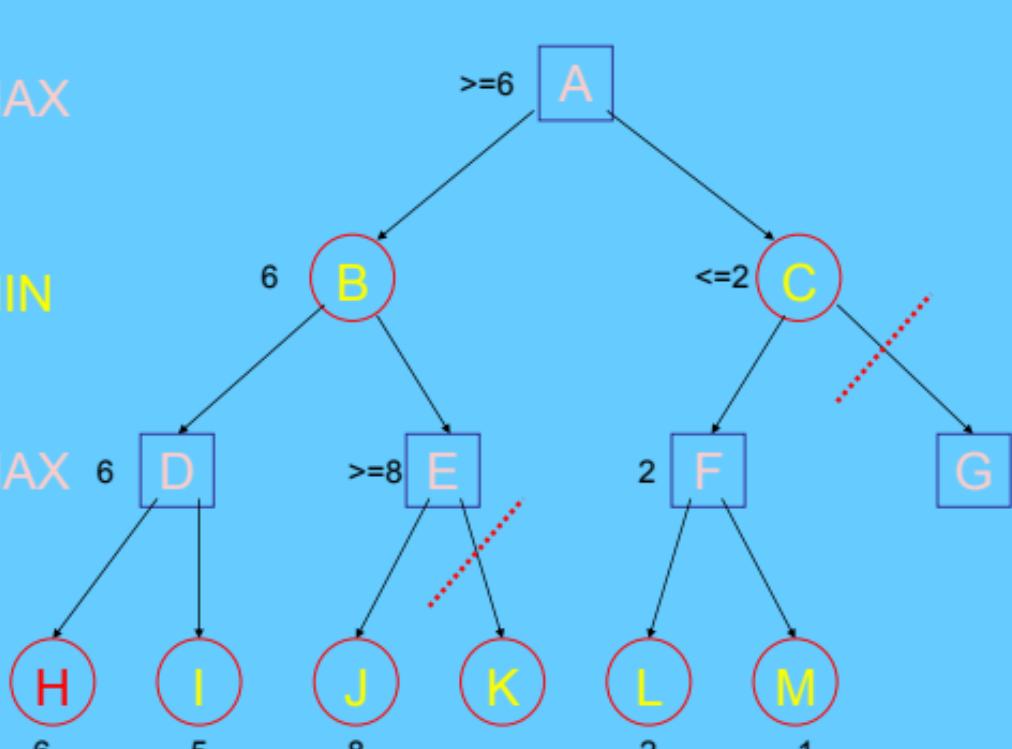
zlepšenie minimaxu

- Uvažované hry sú symetrické. Čo sme uvažovali usekávať z pohľadu Maxa, môžeme podobne aj z pohľadu Mina.
- zdôvodnenie rovnaké, len roly Maxa a Mina sa vymenia.
- Odhalíme ďalšie uzly, ktoré nemôžu byť súčasťou najlepšej hry (a možno ich useknúť)

MAX

MIN

MAX



□ = hráč

○ = protihráč

Genetické algoritmy

- Evolúcia k optimálnym riešeniam prebieha v mnohých generáciach populácií riešení pomocou prirodzeného výberu
- neúspešné riešenia vymierajú
- úspešné riešenia prezívajú a množia sa
- hybnou silou evolúcie je kríženie (výmena genetickej informácie medzi riešeniami) a mutovanie

Evolúcia

- veľmi zjednodušená inšpirácia tým, ako evolúcia prebieha v prírode (a študuje sa v biológii):
- živé organizmy (zvieratá alebo rastliny) vytvárajú potomkov, ktorí sú skoro (ale nie celkom) ako oni.
 - variáciu môže spôsobiť mutácia (náhodné zmeny)
 - variácia môže byť výsledkom pohlavného rozmnožovania (potomkovia majú nejaké vlastnosti od každého rodiča)
- niektorí z týchto potomkov prežijú a vytvoria vlastných potomkov, niektorí neprežijú
 - lepšie prispôsobení potomkovia prežijú s väčšou pravdepodobnosťou
 - ako evolúcia pokračuje v čase, neskoršie generácie sa stávajú lepšie a lepšie prispôsobenými
- genetické algoritmy používajú ten istý proces, aby vyvinuli lepšie programy

gén a chromozóm

- Gén je časťou molekuly DNA, je jednotkou dedičnosti, v ktorej je zakódovaná informácia pre vznik určitého znaku. Znak je určitá jednotlivá vlastnosť organizmu, v ktorej sa odlišuje jeden organizmus od druhého.
- Gény sú základné inštrukcie „budovania“ živého organizmu.
- Chromozóm obsahuje viacero génov. V každom chromozóme sú gény, zoradené v presnom poradí za sebou.
- Chromozóm je postupnosť génov.

genotyp a fenotyp

- Genotyp je súbor dedičných vlôh, ktoré má organizmus.
- Na tento súbor vlôh však pôsobí prostredie, ktoré vyvíjajúceho sa jedinca obklopuje. Jeho genotyp a vplyvy prostredia vyvolávajú určité znaky a vlastnosti.
- Súbor týchto znakov a vlastností, ktoré sa prejavujú už navonok, sa nazýva fenotyp. Fenotyp je výsledkom vzájomného pôsobenia medzi genotypom a prostredím.
- Genotyp je viac alebo menej nemenný, zatiaľ čo fenotyp sa vplyvom prostredia môže meniť. Pravda, aj genotyp sa môže meniť, a to jednak krížením – kombináciou vlôh obidvoch rodičov a jednak mutáciami. Mutácie sú náhle vzniknuté a trvalé dedičné zmeny.
- genotyp organizmu (gény a chromozómy) – jeho fenotyp (aký je v skutočnosti)

základný genetický algoritmus

- začni s veľkou „populáciou“ náhodne vygenerovaných „pokusných riešení“ daného problému
- opakuj:
 - ohodnot' každé z „pokusných riešení“
 - podrž si podmožinu z nich (tie najlepšie ohodnotené)
 - použi tieto riešenia na generovanie novej populácie
- skonči keď máš uspokojivé riešenie alebo nemáš viac času

naozaj jednoduchý príklad

- predpokladajme, že naše organizmy sú 32-bitové počítačové slová
- riešením je slovo so samými jednotkami
- ako to dosiahnuť:
 - vytvor 100 náhodne vygenerovaných počítačových slov
 - opakuj:
 - spočítaj jednotky v každom slove
 - skonči ak má nejaké slovo 32 jednotiek
 - podrž si 10 slov, ktoré majú najviac jednotiek (odhad' zvyšné)
 - z každého slova vygeneruj 9 nových slov takto:
 - zvoľ náhodne jeden bit v slove a zmeň ho
- všimnime si, že táto procedúra nezaručuje, že v ďalšej generácii budú slová s väčším počtom jednotiek, hoci je to pravdepodobné

ozajstnejší príklad I

- predpokladajme, že máme veľký počet dvojíc čísel (bodov v dvojrozmernom priestore) (x, y)
 - napr (1.0, 4.1), (3.1, 9.5), (-5.2, 8.6), ...
- treba preložiť týmito bodmi polynóm (najviac stupňa 5)
 - tj hľadáme výraz $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$ taký, ktorý čo najlepšie vyhovuje daným údajom
 - bežný postup ako určiť „vyhovenie“:
 - vypočítať súčet (dané y - vypočítané y)² pre všetky body
 - najnižší súčet predstavuje najlepšie vyhovenie
- sú známe bežné metódy approximácie, povedzme, že ich nepoznáme
- dá sa použiť genetický algoritmus na nájdenie „celkom dobrého“ riešenia

ozajstnejší príklad II

- náš výraz je $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$
- naše „gény“ sú $a, b, c, d, e, a f$
- náš “chromozóm” je pole $[a, b, c, d, e, f]$
- naša vyhodnocovacia funkcia pre jedno pole je:
 - pre každý bod (x, y) :
 - vypočítaj $\hat{y} = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$
 - vypočítaj súčet $(y - \hat{y})^2$ cez všetky x
 - súčet je mierou „nedobrosti“ (suma štvorcov rozdielov má byť čo najmenšia, čím je väčšia, tým je horšie)
 - napr: pre pole $[0, 0, 0, 2, 3, 5]$ a body $(1, 12)$ and $(2, 22)$:
 - $\hat{y} = 0x^5 + 0x^4 + 0x^3 + 2x^2 + 3x + 5$ je $2 + 3 + 5 = 10$ pre x je 1
 - $\hat{y} = 0x^5 + 0x^4 + 0x^3 + 2x^2 + 3x + 5$ je $8 + 6 + 5 = 19$ pre x je 2
 - $(12 - 10)^2 + (22 - 19)^2 = 2^2 + 3^2 = 13$
 - ak by boli len tieto dva body, „nedobrost“ poľa $[0, 0, 0, 2, 3, 5]$ je 13

ozajstnejší príklad III

- samotný algoritmus by bol:
 - vytvor 100 6-prvkových polí náhodných čísel
 - opakuj 500 razy (alebo hocijaký iný počet opakovania):
 - pre každé z týchto 100 polí vypočítaj jeho nedobrosť (použijúc všetky dátové body)
 - ponechaj 10 najlepších polí (zahod' ostatných 90)
 - z každého poľa, ktoré ponechávaš, vygeneruj 9 nových polí takto:
 - zvoľ ľubovoľný prvok poľa (spomedzi jeho 6 prvkov)
 - zvoľ náhodné reálne číslo medzi 0.0 a 2.0
 - vynásob náhodne zvolený prvok poľa s náhodne zvoleným číslom
 - po 500 pokusoch vráť pole s najlepším ohodnotením ako riešenie

bezpohlavné vs. pohlavné rozmnožovanie

- v doterajších príkladoch:
 - každý jedinec - “organizmus” (alebo “riešenie”) mal len jedného rodiča
 - rozmnožovanie bolo bezpohlavné
 - jediný spôsob, ako vniestť variáciu, bol mutáciou (náhodnými zmenami)
- pri pohlavnom rozmnožovaní:
 - každý “organizmus” (alebo “riešenie”) má dvoch rodičov (okrem členov počiatočnej populácie)
 - za predpokladu, že každý organizmus má iba jeden chromozóm, noví potomkovia vznikajú vytváraním nového chromozómu z častí chromozómov každého rodiča

naozaj jednoduchý príklad

- predpokladajme, že naše organizmy sú 32-bitové počítačové slová
- riešením je slovo so samými jednotkami
- ako to dosiahnuť:
 - vytvor 100 náhodne vygenerovaných počítačových slov
 - opakuj:
 - spočítaj jednotky v každom slove
 - skonči ak má nejaké slovo 32 jednotiek
 - podrž si 10 slov, ktoré majú najviac jednotiek (odhod' zvyšné)
 - z každého slova vygeneruj 9 nových slov takto:
 - zvoľ jedno z ďalších slov
 - vezmi prvú polovicu z tohto slova a skombinuj ju s druhou polovicou toho ďalšieho slova

naozaj jednoduchý príklad - kríženie

- polovica z jedného, polovica z druhého:

0110 1001 0100 1110 1010 1101 1011 0101
1101 0100 0101 1010 1011 0100 1010 0101
0110 1001 0100 1110 1011 0100 1010 0101

- alebo zvolíme „gény“ (bity) náhodne:

0110 1001 0100 1110 1010 1101 1011 0101
1101 0100 0101 1010 1011 0100 1010 0101
0100 0101 0100 1010 1010 1100 1011 0101

- alebo považujeme za “gén” väčšie zoskupenia bitov:

0110 1001 0100 1110 1010 1101 1011 0101
1101 0100 0101 1010 1011 0100 1010 0101
1101 1001 0101 1010 1010 1101 1010 0101

porovnanie jednoduchých príkladov

- v naozaj jednoduchom príklade (dostať všetky 1):
 - pohlavný (dvaja rodičia, bez mutácie) prístup, ak uspeje, tak omnoho rýchlejšie
 - lebo sa v každom kroku mení až polovica bitov, nie iba jeden
 - pravda, bez mutácie nemusí vôbec uspieť
 - čírou nešťastnou náhodou sa môže napr. stať, že žiadne z pôvodne generovaných slov nemá napr. 27. bit nastavený na 1
 - vtedy nejestvuje žiadny spôsob, aby sa môže 1 dostať na 27. miesto
 - ďalší problém je nedostatok genetickej rôznorodosti
 - možno aj malo niekoľko pôvodne generovaných slov na 27. mieste 1, ale ani jedno z nich sa nevybral do druhej generácie
- najlepší prístup vo všeobecnosti sa javí byť pohlavné rozmnožovanie s malou pravdepodobnosťou mutácií

aproximácia pomocou pohlavného rozmnožovania

- náš výraz je $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$
- naše „gény“ sú $a, b, c, d, e, a f$
- náš “chromozóm” je pole $[a, b, c, d, e, f]$
- ako najlepšie kombinovať dva chromozómy do jedného?
 - môžeme vziať prvú polovicu z jedného a druhú polovicu z druhého: $[a, b, c, d, e, f]$
 - môžeme voliť gény náhodne: $[a, b, c, d, e, f]$
 - môžeme počítať “priemerné gény:”
 $[(a+a)/2, (b+b)/2, (c+c)/2, (d+d)/2, (e+e)/2, (f+f)/2]$
 - domnienka: posledný nápad by mohol byť najlepší

usmerňovaná evolúcia

- v doterajších príkladoch sa potomkovia vytvárali náhodne
 - nepokúšali sme sa vyberať „najlepšie“ gény z každého rodiča
 - takto funguje prirodzená (biologická) evolúcia
 - evolúcia v prírode nie je / sa nejaví byť usmerňovaná —nemá "cieľ"
- genetické algoritmy sa inšpirujú prírodou, ale neberú ju ako množinu pravidiel, ktoré treba otrocky dodržiavať
 - pre úlohu dostať slovo so samými 1 jestvuje evidentná miera toho, čo je „dobrý“ gén
 - lenže to vlastne vôbec nie je dobrý príklad problému
 - pre problém approximácie je omnoho ľažšie rozpoznať „dobrý“ gén
 - ľažké pre praktický každý „skutočný“ problém riešený genetickým algoritmom

stochastický výber

- v doterajších príkladoch sa vyberalo N „najlepších“ organizmov za rodičov v ďalšej generácii
- bežnejší prístup je vyberať rodičov náhodne, s prihliadnutím na ich mieru „dobrosti“, odhadovanej úspešnosti
 - čiže organizmus, ktorý je hodnotený ako dvakrát lepší než iný bude mať pravdepodobne dvakrát toľko potomkov
- výhody:
 - do ďalšej generácie najviac prispievajú najlepšie organizmy
 - každý organizmus má stále aspoň nejakú nádej, že bude vybraný ako rodič, preto sa nestráca genetická rôznorodosť

GA trochu podrobnejšie

- každé riešenie úlohy (každý stav) označuje pojmom **chromozóm**.
- populácia P obsahujúcou M chromozómov.
- chromozóm - *binárny reťazec* x dĺžky N
- Napríklad pre problém 8 dám by chromozóm zodpovedal usporiadanej množine poličok šachovnice, pričom jednotka by znamenala, že dané poličko je obsadené dámou a nula, že je prázdné.

GA trochu podrobnejšie

- Každému chromozómu $x \in P$ sa priradí hodnota funkcie úspešnosti **fitnes** $f(x)$.
- vyjadruje úspešnosť daného riešenia.
- hľadáme globálne maximum fitnes.

GA trochu podrobnejšie

```
function Genetický-Algoritmus(populácia, Vyhodnocovacia-Funkcia)
    returns riešenie
    inputs: populácia, množina riešení
        Vyhodnocovacia-Funkcia, funkcia, ktorá vyjadruje
        úspešnosť daného riešenia
    repeat
        rodičia ← Vyber(populácia, Vyhodnocovacia-Funkcia)
        populácia ← Reprodukcia(rodičia)
    until nejaké riešenie je dostatočne úspešné
    return najlepšie riešenie v populácii podľa Vyhodnocovacej-
        Funkcie
    end
```

GA trochu podrobnejšie

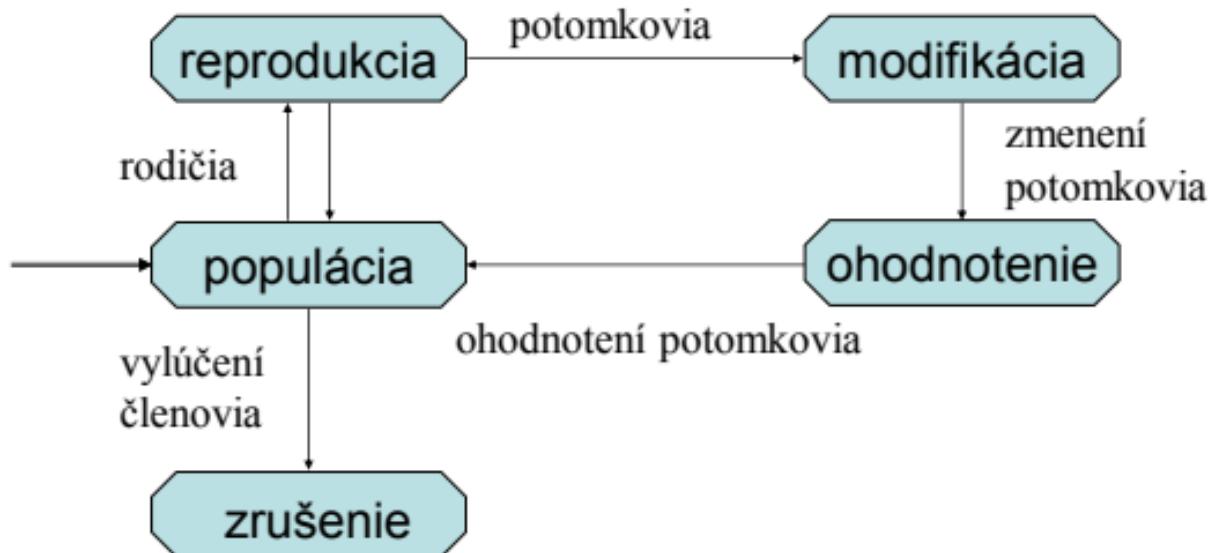
- 0 ŠTART : vytvor náhodnú populáciu n chromozómov
- 1 FITNES : vyhodnot fitnes $f(x)$ každého chromozómu v populácii
- 2 REPRODUKCIA
 - 0 VÝBER : založený na $f(x)$
 - 1 REKOMBINÁCIA : križenie chromozómov
 - 2 MUTÁCIA : zmutuj chromozómy
 - 3 PRIJATIE : odstráň alebo prijmi nový chromozóm
- 3 NÁHRADA : nahrad starú populáciu novou
- 4 TEST : otestuj, či je v populácii riešenie problému
- 5 CYKLUS : opakuj kroky 1 - 4 dovtedy, až nejaký chromozóm prejde úspešne testom

zložky genetického algoritmu

problém, ktorý treba vyriešiť a ...

- metóda kódovania (gén, chromozóm)
 - procedúra inicializácie (vytvorenie)
 - vyhodnocovacia funkcia (prostredie)
 - výber rodičov (reprodukcia)
 - genetické operátory (mutácia, kríženie)
 - nastavenia parametrov (skúsenosť a šikovnosť)

cyklus genetického algoritmu evolúcie



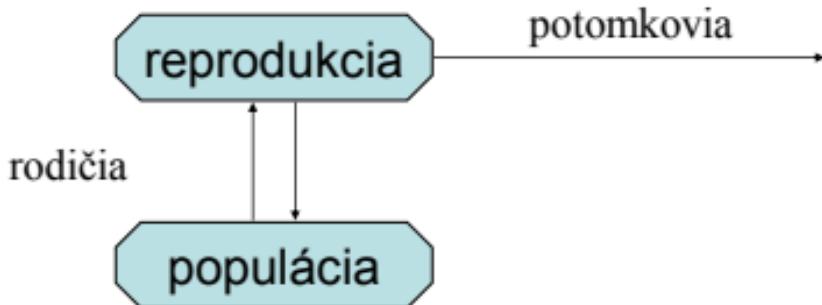
populácia



chromozómy môžu byť:

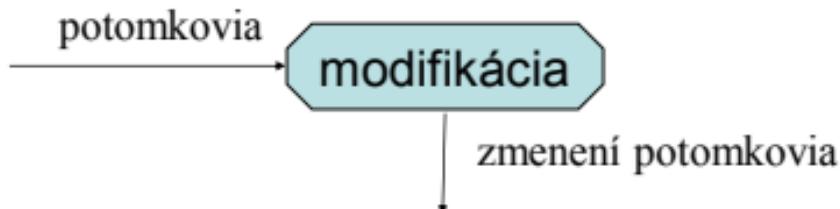
- bitové reťazce (0101 ... 1100)
- reálne čísla (43.2 -33.1 ... 0.0 89.2)
- permutácie prvku (E11 E3 E7 ... E1 E15)
- zoznam pravidiel (R1 R2 R3 ... R22 R23)
- prvky programu (genetické programovanie)
- ... ľubovoľná údajová štruktúra ...

reprodukcia



Rodičia sa vyberajú náhodne, s pravdepodobnosťou úmernou ohodnoteniu chromozómu.

modifikácia



- modifikácie sa spúšťajú stochasticky
- typy operátorov:
 - mutácia
 - kríženie (rekombinácia)

Mutovanie

- mení náhodne potomkov
- prečo?
aby sa predišlo prechodu populácie do lokálneho optima
- ako?
niekoľko náhodne zvolených bitov sa invertuje
- príklad

pôvodný potomok 1	1001010000001111
pôvodný potomok 2	0101101100111010
zmutovaný potomok 1	1011010000001111
zmutovaný potomok 2	0101101110111010

mutovanie – lokálna modifikácia

pred: (1 0 1 1 0 1 1 0)

po: (1 0 1 0 0 1 1 0)

pred: (1.38 -69.4 326.44 0.1)

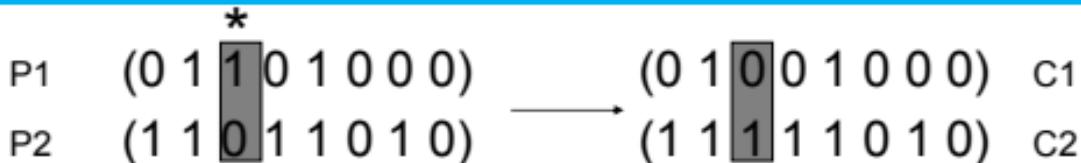
po: (1.38 -67.5 326.44 0.1)

- spôsobuje pohyb v priestore hľadania (lokálne alebo globálne)
- obnovuje v populácii stratenú informáciu

Pravdepodobnosť mutovania

- ako často sa budú mutovať časti chromozómu
 - 0 – potomok prechádza do novej generácie skrížený alebo nezmenený
 - (0 , 1) – časť chromozómu sa zmení
 - 1 – celý chromozóm sa zmení
- prečo mutovať?
 - aby GA neupadol do lokálneho extrému
- ak je mutovanie príliš časté, hľadanie riešenia sa blíži náhodnému

kríženie - rekombinácia



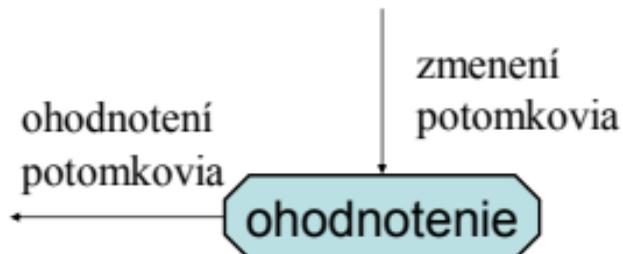
kríženie je kritická črta genetických algoritmov:

- silne urýchľuje hľadanie v začiatkoch evolúcie populácie
- vedie k účinnému kombinovaniu čiastočných riešení na rôznych chromozómoch

pravdepodobnosť kríženia

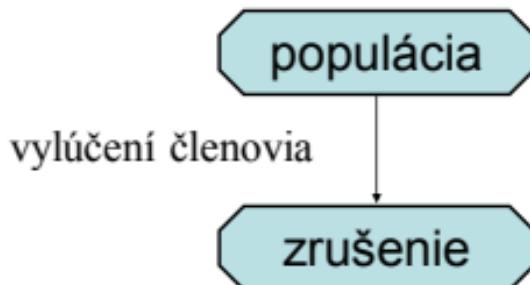
- ako často sa bude krížiť
 - 0 – celá nová generácia vzniká z presných kópií chromozómov starej generácie
 - (0 , 1) – potomkovia vznikajú z častí rodičov
 - 1 – všetci potomkovia vzniknú krížením
- prečo krížiť?
 - nové chromozómy snáď zdedia dobré časti (vlastnosti) a budú lepšie
- odporúča sa nechať časť populácie prežiť do ďalšej generácie

ohodnotenie



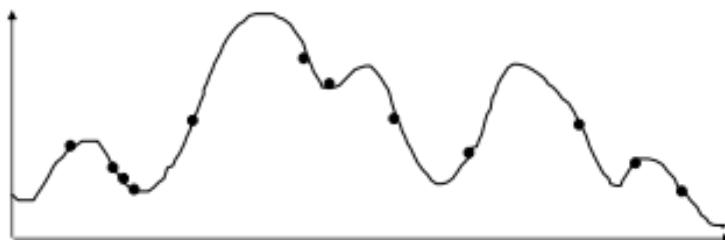
- ohodnocovač dekóduje chromozóm a priradí mu hodnotu fitnes
- ohodnocovač je jediným spojením medzi GA a problémom, ktorý riešia

zrušenie

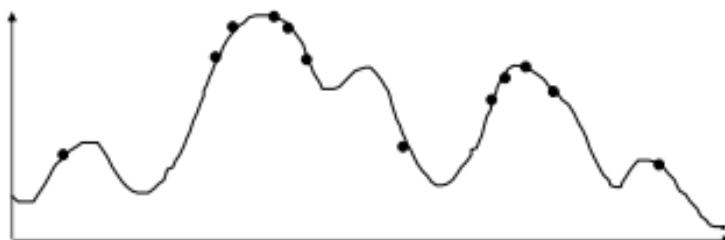


- celé populácie sa nahradzajú v každom kroku iterácie
- len niekoľko členov sa nahradza v každom kroku

abstraktný príklad



rozloženie jedincov v generácii 0



rozloženie jedincov v generácii N

opäť naozaj jednoduchý príklad

- pre jednoduchosť, l -bitové slová, začiatočná populácia má n chromozómov.
- nech $l = 10$ a $n = 6$

naozaj jednoduchý príklad - inicializácia

hodíme mincu 60 razy, dostaneme začiatočnú populáciu:

$$s_1 = 1111010101 \quad f(s_1) = 7$$

$$s_2 = 0111000101 \quad f(s_2) = 5$$

$$s_3 = 1110110101 \quad f(s_3) = 7$$

$$s_4 = 0100010011 \quad f(s_4) = 4$$

$$s_5 = 1110111101 \quad f(s_5) = 8$$

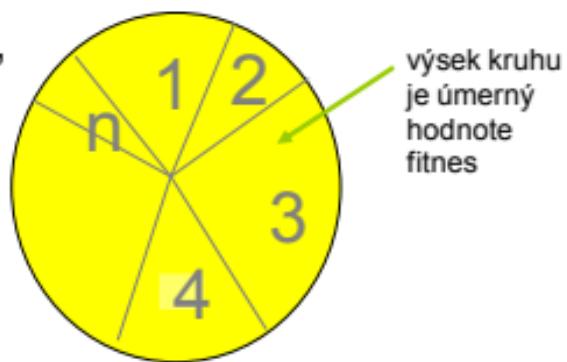
$$s_6 = 0100110000 \quad f(s_6) = 3$$

naozaj jednoduchý príklad - výber

výber v závislosti od ohodnotenia užitočnosti fitnes
metódou rulety:

jedinec i bude mať
pravdepodobnosť výberu $\frac{f(i)}{\sum_i f(i)}$

výber sa opakuje toľkokrát,
koľko potrebujeme
jedincov do populácie
rodičov – rovnaký počet
ako je veľkosť populácie
(6 v tomto príklade)



naozaj jednoduchý príklad - výber

predpokladajme, že po výberoch dostaneme takúto populáciu:

$$s_1' = 1111010101 \quad (s_1)$$

$$s_2' = 1110110101 \quad (s_3)$$

$$s_3' = 1110111101 \quad (s_5)$$

$$s_4' = 0111000101 \quad (s_2)$$

$$s_5' = 0100010011 \quad (s_4)$$

$$s_6' = 1110111101 \quad (s_5)$$

naozaj jednoduchý príklad - kríženie

nastupuje kríženie. pre každý rodičovský pár (dvojicu chromozómov) sa rozhodne na základe pravdepodobnosti kríženia (nech je určená napr 0.6), či sa vôbec skrížia.

nech sa rozhodlo, že skrížia sa iba páry ($s_1`$, $s_2`$) a ($s_5`$, $s_6`$). Pre každý pár sa náhodne určí bod kríženia, napr. 2 pre prvý a 5 pre druhý

naozaj jednoduchý príklad - kríženie

pred skrížením:

$$s_1' = 11\textcolor{yellow}{11010101}$$

$$s_2' = 11\textcolor{teal}{10110101}$$

$$s_5' = 01000\textcolor{yellow}{10011}$$

$$s_6' = 11101\textcolor{teal}{11101}$$

po skrížení:

$$s_1'' = 11\textcolor{teal}{10110101}$$

$$s_2'' = 11\textcolor{yellow}{11010101}$$

$$s_5'' = 01000\textcolor{teal}{11101}$$

$$s_6'' = 11101\textcolor{yellow}{10011}$$

naozaj jednoduchý príklad - mutácia

posledný krok je zmutovať nové jedince: pre každý bit nového jedinca jestvuje nenulová pravdepodobnosť (napr 0.1), že sa zmutuje

pred aplikáciou mutácie:

$$s_1^{\prime\prime} = 11101\textcolor{teal}{1}0101$$

$$s_2^{\prime\prime} = 111101010\textcolor{teal}{1}$$

$$s_3^{\prime\prime} = 11101\textcolor{teal}{1}11101$$

$$s_4^{\prime\prime} = 0111000101$$

$$s_5^{\prime\prime} = 0100011101$$

$$s_6^{\prime\prime} = 11101100\textcolor{teal}{1}1$$

naozaj jednoduchý príklad - mutácia

po aplikovaní mutácie:

$$s_1''' = 11101\textcolor{teal}{0}0101 \quad f(s_1''') = 6$$

$$s_2''' = 1111\textcolor{teal}{1}10100 \quad f(s_2''') = 7$$

$$s_3''' = 11101\textcolor{teal}{0}1111 \quad f(s_3''') = 8$$

$$s_4''' = 0111000101 \quad f(s_4''') = 5$$

$$s_5''' = 0100011101 \quad f(s_5''') = 5$$

$$s_6''' = 11101100\textcolor{teal}{0}1 \quad f(s_6''') = 6$$

naozaj jednoduchý príklad - záver

v jednej generácii sa celková užitočnosť
(súčet fitnes všetkých jedincov) zmenila z 34
na 37, zlepšenie o ~9%

a tak ďalej sa pokračuje od začiatku, pokiaľ
nie splnený test riešenia

dľalší príklad: problém obchodného cestujúceho

obchodný cestujúci musí navštíviť každé mesto v jemu pridelenej oblasti práve raz a potom sa vrátiť do miesta, kde začal. Dané sú ceny prepravy medzi všetkými dvojicami miest. Aký má byť plán jeho okružnej cesty, aby bola celková cena najmenšia?

TSP \in NP-úplné problémy

pozn: začneme jedným možným prístupom k hľadaniu približného riešenia problému obchodného cestujúceho pomocou GA

TSP (reprezentácia, vyhodnotenie, inicializácia a výber)

vektor $v = (i_1 \ i_2 \dots \ i_n)$ reprezentuje cestu (v je permutácia $\{1,2,\dots,n\}$)

Fitnes f riešenia je prevrátená hodnota ceny zodpovedajúcej cesty

Inicializácia: použiť nejakú heuristiku alebo náhodný výber permutácií $\{1,2,\dots,n\}$

Použijeme náhodný výber s pravdepodobnosťou úmernou fitnes

TSP (kríženie1)

Potomok vznikne zvolením podpostupnosti cesty z jedného rodiča a zachovaním relatívneho poradia miest z druhého rodiča

napr:

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \text{ a}$$

$$p_2 = (4 \ 5 \ 2 \ 1 \ 8 \ 7 \ 6 \ 9 \ 3)$$

Najprv sa podpostupnosti medzi bodmi kríženia skopírajú do potomkov

$$o_1 = (x \ x \ x \ 4 \ 5 \ 6 \ 7 \ x \ x) \text{ a}$$

$$o_2 = (x \ x \ x \ 1 \ 8 \ 7 \ 6 \ x \ x)$$

TSP (kríženie2)

Ďalej, počínajúc od druhého bodu kríženia jedného rodiča, mestá z druhého rodiča sa kopírujú pri zachovaní poradia postupnosti miest v druhom rodičovi je

$$9 - 3 - 4 - 5 - 2 - 1 - 8 - 7 - 6$$

po odstránení miest prevzatých z prvého potomka dostaneme
9 – 3 – 2 – 1 – 8

Táto postupnosť sa vloží do prvého potomka

$$o_1 = (2 \ 1 \ 8 \ 4 \ 5 \ 6 \ 7 \ 9 \ 3), \text{ a podobne do druhého}$$

$$o_2 = (3 \ 4 \ 5 \ 1 \ 8 \ 7 \ 6 \ 9 \ 2)$$

TSP (inverzia)

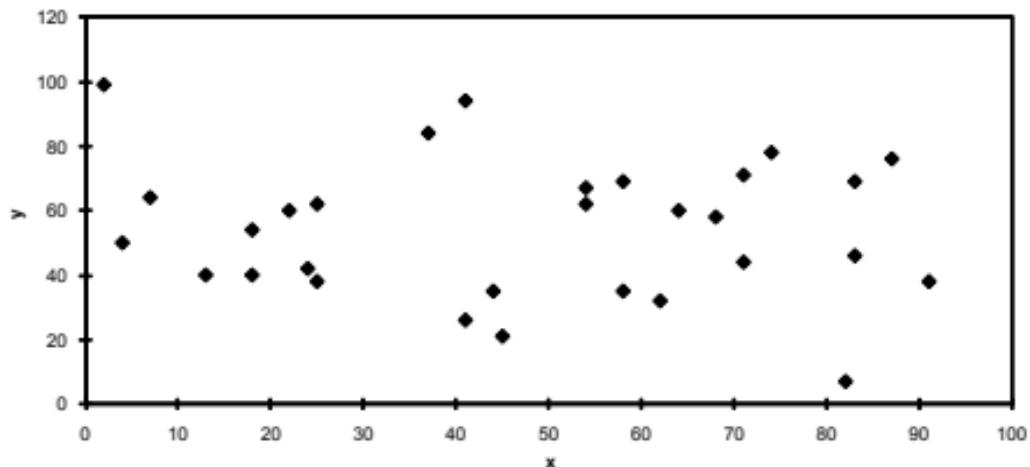
Podpostupnosť medzi dvoma náhodne zvolenými bodmi sa obráti

napr:

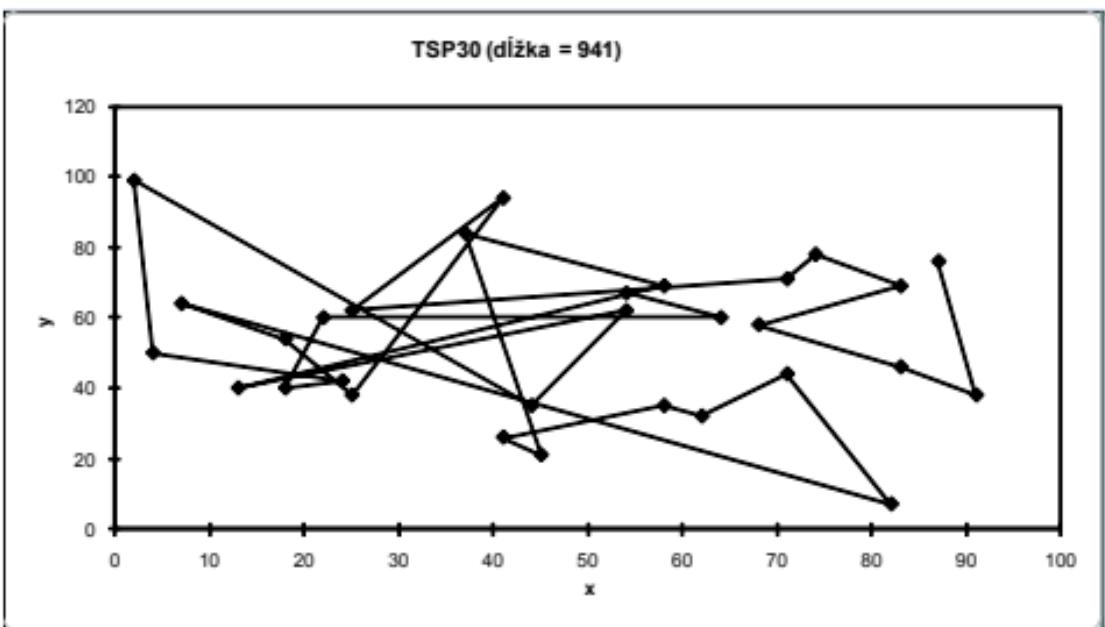
(1 2 3 4 5 6 7 8 9) sa zmení na (1 2 7 6 5 4 3 8 9)

Také jednoduché obrátenie poradia génov (miest) zaručuje, že výsledný potomok je prípustná cesta

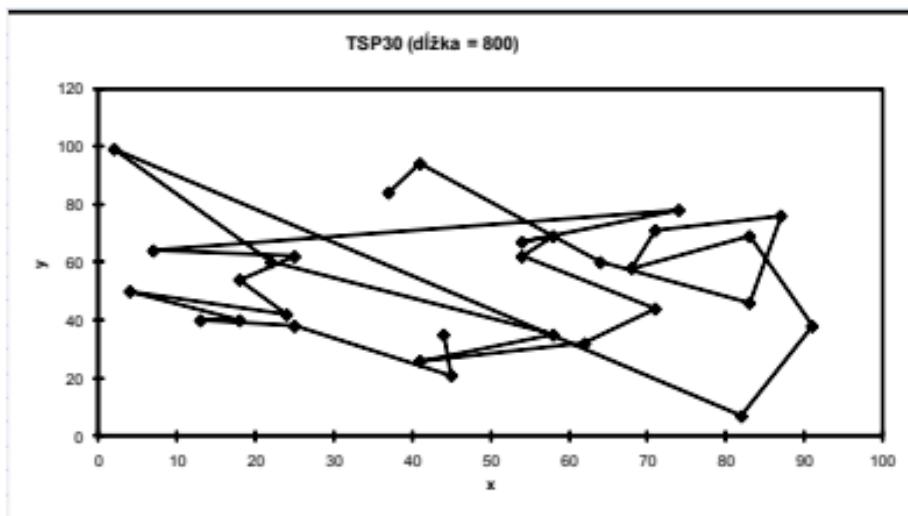
TSP príklad – 30 miest



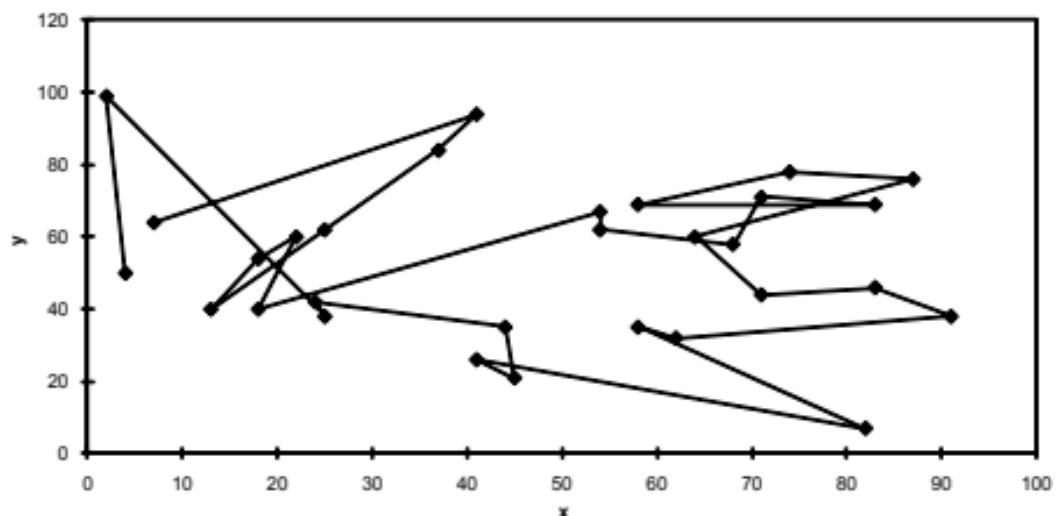
TSP príklad – 30 miest



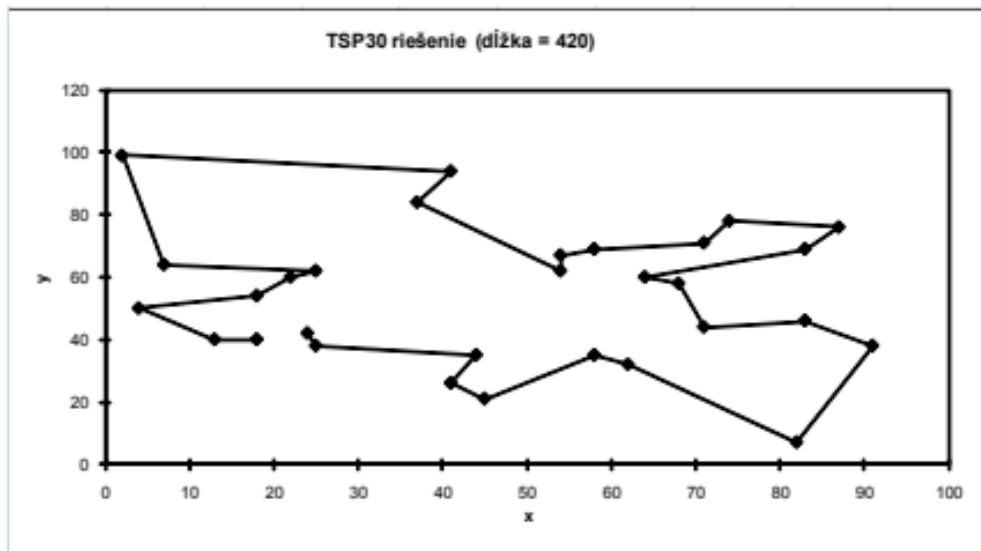
riešenie_j (dĺžka 800)



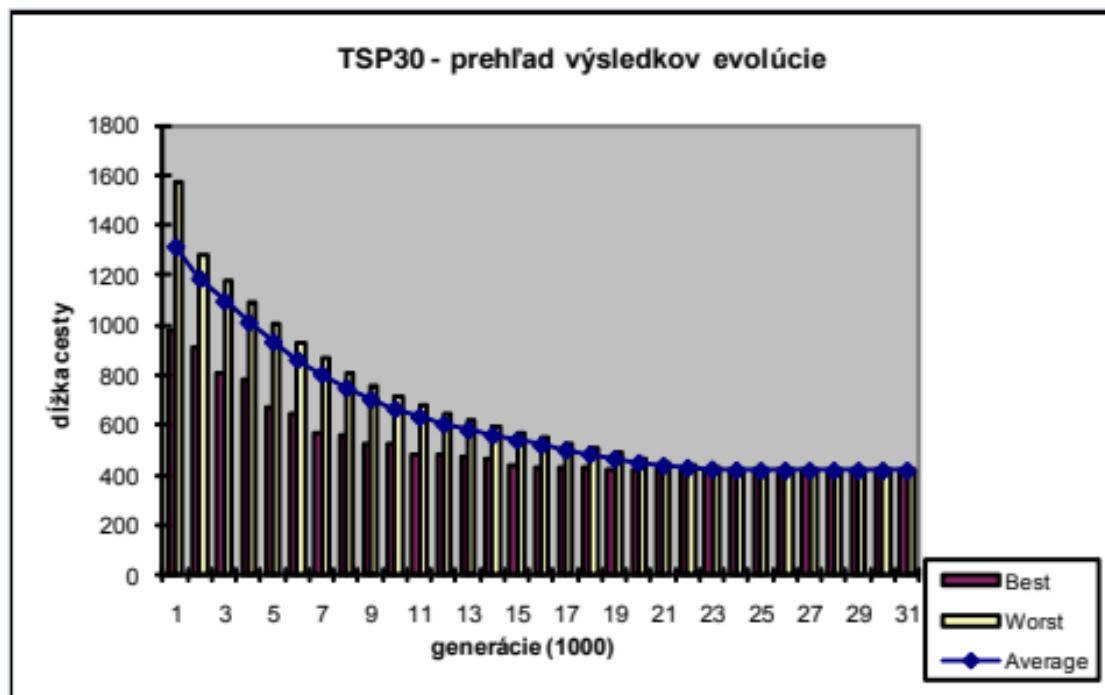
riešenie_k (dĺžka 652)



najlepšie riešenie (dĺžka = 420)



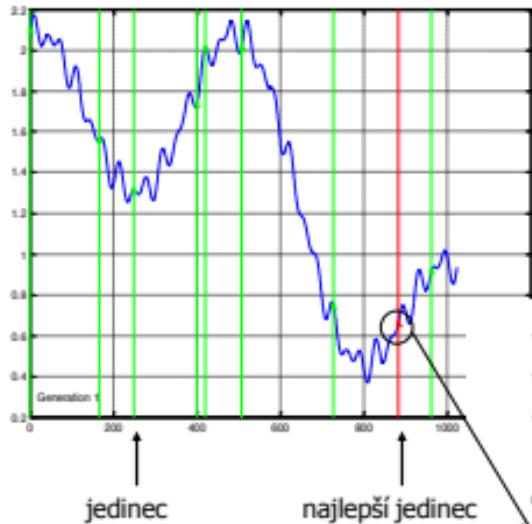
prehľad výsledkov evolúcie



príklad: hľadanie minima funkcie

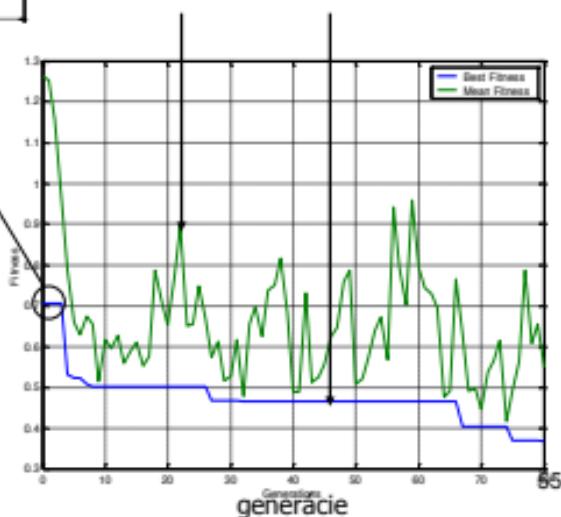


príklad: hľadanie minima funkcie



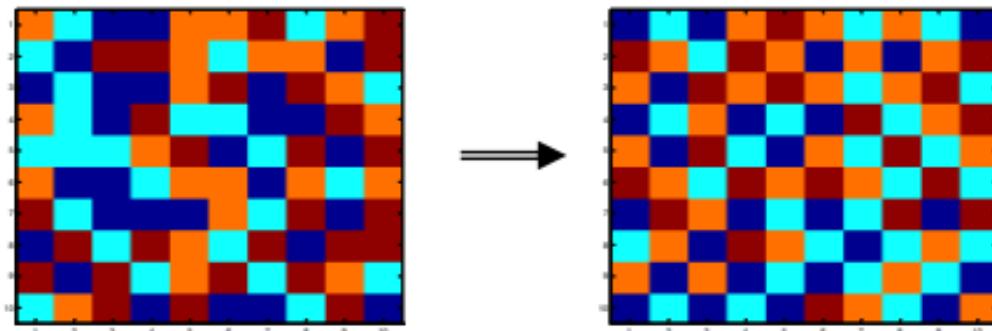
stredná
hodnota
fitnes

najlepšia
hodnota
fitnes



príklad: hracia doska

- daná je hracia doska $n \times n$, na ktorej každé poličko môže mať svoju farbu, jednu z množiny 4 farieb.
- cieľ je dosiahnuť konfiguráciu hracej dosky takú, že žiadne susediace polička nemajú rovnakú farbu (polička na uhlopriečke nesusedia)

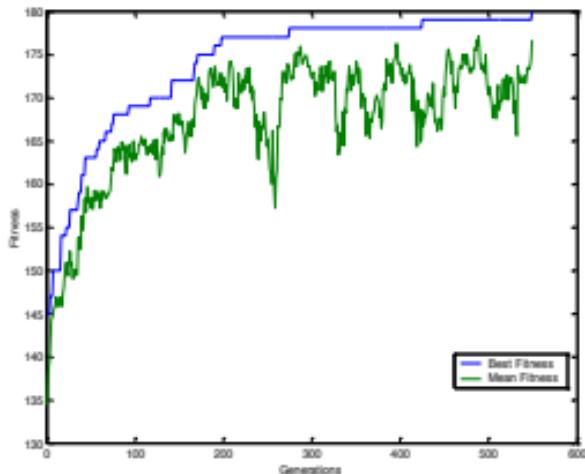


príklad: hracia doska

- chromozómy reprezentujú spôsob, ako je hracia doska ofarbená.
- chromozómy sa nereprezentujú reťazcami bitov, ale 4-bitovými maticami
- 4-bity v matici môžu mať jednu z hodnôt 0, 1, 2 alebo 3, v závislosti od farby
- kríženie zahŕňa manipuláciu matice namiesto operácií nad 4-bitmi. Kríženie môže kombinovať rodičovské matice vodorovne, zvisle, do trojuholníka alebo do štvorca
- mutácia zostáva menením bitov

príklad: hracia doska

priebeh funkcie fitnes

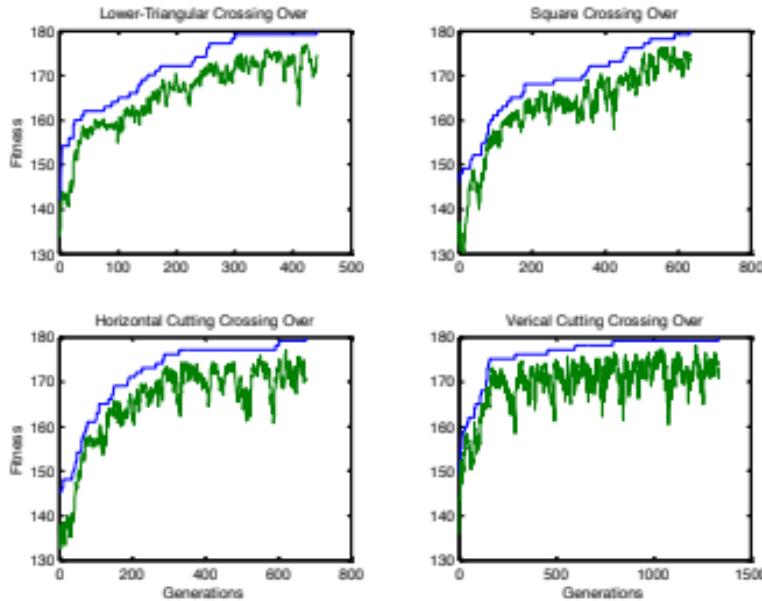


tento problém sa dá opísat' grafom s n uzlami a $(n-1)$ hranami,
takže funkcia fitnes $f(x)$ sa dá ľahko definovať ako:

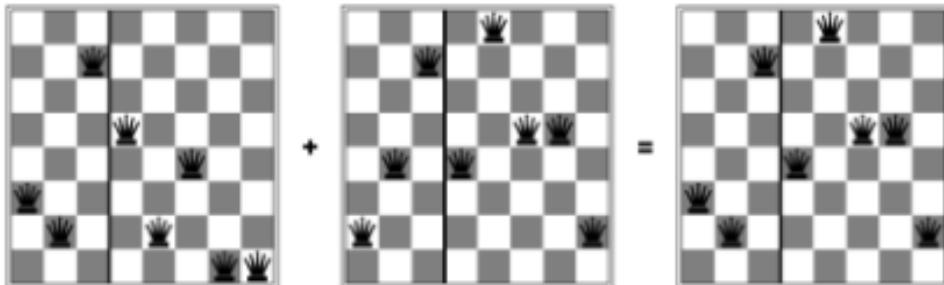
$$f(x) = 2 \cdot (n-1) \cdot n$$

príklad: hracia doska

priebehy funkcie fitnes pre rôzne pravidlá kríženia



príklad: 8 dám



32752411

24748552

32748552

príklad: 8 dám



- Funkcia fitnes: počet nenapádajúcich sa dvojíc dám (min = 0, max = $8 \times 7/2 = 28$)
- $\text{fitnes}(\text{chromozóm}[24748552]) = 24/(24+23+20+11) = 31\%$
- $\text{fitnes}(\text{chromozóm}[32752411]) = 23/(24+23+20+11) = 29\%$
- atď.
-

implementačné aspekty

- parametre
 - reprezentácia kandidátov riešení
 - veľkosť populácie, pravdepodobnosť mutácie, ...
 - pravidlá výberu, vylučovania
 - operátory kríženia, mutácie
- kritérium ukončenia
- výkon, škálovateľnosť
- riešenie je len tak dobré, ako je dobrá vyhodnocovacia funkcia (navrhnutú ju je často najťažšie)

počet chromozómov

- kolko chromozómov je v populácii (v jednej generácii)
 - príliš málo
 - málo možností na kríženie -> malá časť stavového priestoru sa dá prehľadať
 - príliš veľa
 - GA sa spomalí
 - zvyšovanie nad nejakú hranicu (závislú najmä od spôsobu kódovania problému) neprináša užitok (zrýchlenie)

výber rodičov

- ako vybrať chromozómy, ktoré sa stanú rodičmi?
- (Darwin:) prežiť (a mať potomkov) majú najlepší
- ako vybrať najlepšie chromozómy?
 - ruleta
 - poradie
 - ďalšie

ruleta

- výber sa robí na základe fitnes
 - čím má chromozóm lepšiu fitnes, tým dostane väčšiu šancu
- na kolese rulety má každý chromozóm vyhradený segment úmerný jeho fitnes
 - náhodný hod, guľka skončí v niektorom segmente
 - chromozóm s lepšou fitnes sa častejšie vyberie
- algoritmus výberu ruletou:

S:= súčet všetkých ohodnotení fitnes v celej populácii
r:= generuj náhodné číslo v intervale (0, S)
s:= preber jedného člena populácie za druhým a spočítavaj ich fitnes začnúc od 0. keď s > r, zastav a vráť chromozóm, pri ktorom si.

výber podľa poradia

- výber ruletou môže príliš potlačiť šancu ostatných chromozómov, ak fitnes jedného priveľmi dominuje
- výber sa robí na základe fitnes
 - chromozóm s lepšou fitnes dostane väčšiu šancu
- na kolese má každý chromozóm vyhradený segment úmerný jeho poradiu v usporiadaní podľa fitnes
 - chromozóm s najmenšou fitnes – 1
 - chromozóm s druhou najmenšou fitnes – 2
 - ...
 - chromozóm s najlepšou fitnes – N (počet chromozómov v populácii)
- všetky chromozómy dostanú lepšiu šancu byť vybrané
- najlepšie sa nie veľmi líšia od ostatných -> pomalšie konverguje k riešeniu

dľalšie možnosti

- stabilita
 - vybrať zopár chromozómov s vysokou fitnes ako rodičov
 - vybrať niekoľko chromozómov so slabou fitnes, vylúčiť ich z populácie a nahradíť potomkami rodičov
- elitárstvo
 - najlepší alebo niekoľko najlepších chromozómov sa skopírujú do novej populácie
 - ďalej štandardne

prínosy GA

- základný pojem je ľahko pochopiteľný
- môže byť samostatný modul, nie súčasť aplikácie
- podporuje aj multikriterálnu optimalizáciu
- vhodný pre „zašumené“ prostredia
- vždy dáva nejakú odpoved; odpoved sa zlepšuje s časom
- z podstaty paralelný; ľahko distribuovaným spracovaním

prínosy GA

- veľa možností, ako sa pokúsiť o zrýchlenie, najmä ak pribúdajú znalosti o doméne problému
- ľahko sa dá využiť nejaké predchádzajúce alebo alternatívne riešenia
- pružná softvérová súčiastka vhodná na tvorbu hybridných aplikácií
- významný záznam doterajších prípadov použitia, široký rozsah

kedy použiť GA

- alternatívne riešenia sú príliš pomalé alebo príliš zložité
- treba nástroj na prieskum nových prístupov
- problém sa podobá inému, ktorý sa už úspešne vyriešil pomocou GA

niektoré oblasti použitia

Doména	Druh aplikácie
riadenie	plynovod, udržiavanie rovnováhy, uhýbanie pred streľami
navrhovanie	polovodičová doska, lietadlo, klávesnica, komunikačné siete
rozvrhovanie	výroba, pridelenie zdrojov
robotika	plánovanie trasy
strojové učenie	navrhovanie neurónových sietí, vylepšovanie klasifikačných algoritmov, klasifikácia
spracovanie signálov	navrhovanie filtrov
hranie hier	poker, dáma, väzňova dilema
optimalizácia	TSP, prepojovanie, balenie plecniakov, ofarbovanie grafov

UMELÁ INTELIGENCIA

Plánovanie

Plánovanie a rozvrhovanie

- **Plánovanie:** je daný jeden alebo viacero cieľov, vytvor postupnosť akcií vedúcich k jeho/ich splneniu
- **Rozvrhovanie (Scheduling):** je daná množina akcií a ohraničení, prideľ zdroje a prirad časy akciám tak, aby neboli porušené ohraničenia
- Tradične, plánuje sa špecializovanými logickými metódami
- Tradične, rozvrhuje sa splňaním ohraničení, lineárnym programovaním alebo metódami operačného výskumu
- Avšak plánovanie a rozvrhovanie sú natoľko blízke úlohy, že sa nie vždy dajú oddeliť.

Plánovanie

- plánovanie v kontexte umelej inteligencie
- vyjadrenie problému plánovania
- potreba odhaliť spojitosti priamo medzi stavmi a akciami
 - (pod)cieľ: $Mám(Mlieko)$
 - akcia $Kúp(x)$ s dôsledkom $Mám(x)$
 - plánovaná akcia: $Kúp(Mlieko)$
 - (ale napr. netreba uvažovať o
 $Kúp(\text{Šampón-a-Vymývač-Hláv-v-Jednom})$
 $Odstrihni-sa)$
- stratégia „rozdeľuj a panuj“

Plánovanie

```
function JEDNODUCHÝ-PLÁNOVACÍ-AGENT(vnem) returns akcia
    static: BP, báza poznatkov (sú v nej aj opisy akcií)
            plán, plán, na začiatku NoPlan
            t, počítadlo, na začiatku 0, udáva čas
    local variables: cieľ, cieľ
                      stav, opis súčasného stavu

    PRIDAJ(BP, VYTvor-VETU-O-VNEME(vnem, t))
    stav ← OPIS-STAVU(BP, t)
    if plán = NoPlan then
        cieľ ← ODPOVEDAJ(BP, VYTvor-DOPYT-NA-CIEL(t))
        plán ← IDEÁLNY-PLÁNOVAČ(stav, cieľ, BP)
    if plán = NoPlan or plán je prázdny then akcia ← NoOp
    else
        akcia ← VYBER-PRVÚ(plán)
        plán ← ZVÝŠOK-AKCIÍ(plán)
    PRIDAJ(BP, VYTvor-VETU-O-AKCII(akcia, t)
    t ← t + 1
return akcia
```

Reprezentácia plánovacieho problému

- reprezentácia stavov a cieľov

stav: $NachádzamSa(Doma) \wedge \neg Mám(Mlieko) \wedge \neg Mám(Rožky) \wedge \neg Mám(Noviny) \wedge \dots$

cieľ: $NachádzamSa(Doma) \wedge Mám(Mlieko) \wedge Mám(Rožky) \wedge Mám(Noviny)$

$NachádzamSa(x) \wedge PredávaSa(x, Mlieko)$

- reprezentácia akcií

- operátor (jazyk STRIPS)
 - opis akcie
 - predpodomienka
 - účinky

$Op(AKCIA: Chod' (tam),$

PREDPOD: $NachádzamSa(tu) \wedge Cesta(tu, tam),$

ÚČINKY: $NachádzamSa(tam) \wedge \neg NachádzamSa(tu))$

- aplikovateľnosť operátora

rozhodovanie sa o tom, ako reprezentovať znalosti

- vyjadrovacia schopnosť vs výpočtová efektívnosť
- STRIPS: jednoduchý, ale rozumne vyjadrovací plánovací jazyk založený na výrokovom počte
- STRIPS = Stanford Research Institute Problem Solver
- Stanford Research Institute, výskum 1966-1972, behal tam legendárny robot Šejkí



Shakey - jednoduché predikáty modelu sveta

Predicative Predicates	Literal Form	Example Literal
None		
type	type(<i>Term</i> , "Term")	type(t, hand)
name	name(<i>Term</i> , <i>Name</i>)	name(t, hand)
hasValue	hasValue(<i>Term</i> , <i>Value</i>)	hasValue(t, 0.22)
grasp	grasp(<i>Term</i> , <i>Tool</i>)	grasp(t, g)
isHeld	isHeld(<i>Term</i> , <i>Tool</i> , <i>Object</i>)	isHeld(t, g, h)
(None)		
type	type(<i>Object</i> , "Object")	type(h, hand)
name	name(<i>Object</i> , <i>Name</i>)	name(h, hand)
hasValue	hasValue(<i>Object</i> , <i>Number</i> , <i>Number</i>)	hasValue(h, 0.1, 0.0)
isInFront	isInFront(<i>Object</i> , <i>Tool</i> , <i>Tool</i>)	isInFront(h, t1, t2)
isBetween	isBetween(<i>Object</i> , <i>Tool</i> , <i>Tool</i>)	isBetween(h, t1, t2)
isOverhead	isOverhead(<i>Object</i> , <i>Object</i>)	isOverhead(h, "open")
(None)		
type	type(<i>Object</i> , "Object")	type(t, tool)
name	name(<i>Object</i> , <i>Name</i>)	name(t, screwdriver)
grasp	grasp(<i>Object</i> , <i>Tool</i>)	grasp(t, g)
(None)		
type	type(<i>Object</i> , "Object")	type(s, object)
name	name(<i>Object</i> , <i>Name</i>)	name(s, screw)
at	at(<i>Object</i> , <i>Number</i> , <i>Number</i>)	at(s, 3.0, 3.2)
isUnder	isUnder(<i>Object</i> , <i>Object</i>)	isUnder(s, t2)
shape	shape(<i>Object</i> , <i>Shape</i>)	shape(s, wedge)
rotation	rotation(<i>Object</i> , <i>Number</i>)	rotation(s, 3.3)
(None)		
type	type("robot", "robot")	type(robot, robot)
name	name("robot", "name")	name(robot, shakey)
at	at("robot", <i>Number</i> , <i>Number</i>)	at(robot, 0, 0.2)
Theta	Theta("robot", <i>Number</i>)	Theta(robot, 0.1)
size	size("robot", <i>Number</i>)	size(robot, 0.2)
pos	pos("robot", <i>Number</i>)	pos(robot, 0.1)
isUnder	isUnder("robot", "shape")	isUnder(robot, s)
isOverhead	isOverhead("robot", "shape")	isOverhead(robot, h)
range	range("robot", <i>Number</i>)	range(robot, 0.4)
volume	volume("robot", "shape")	volume(robot, s)
force	force("robot", <i>Number</i>)	force(robot, 0.1)

Table 1: PRIMITIVE PREDICATES FOR THE ROBOT'S WORLD MODEL*

jazyk STRIPS

Each STRIPS operator must be described in some convenient way. We characterize each operator in the repertoire by three entities: an *add function*, a *delete function*, and a *precondition wff*. The meanings of these entities are straightforward. An operator is



SHAKY THE ROBOT

Technical Note 800

April 1984

Edited by: **Mike J. Moises, Director**
Artificial Intelligence Center
Computer Science and Technology Division

Approved:

Donald L. Knott, Acting Director
Computer Science and Technology Division

2000 Prairieview Lane, P.O. Box 800000
415-265-6000 • 7000 415-265-6000 • Telex: 214-600

PRECONDITION
OPENED
OPENED(X) & OPENED(Y)
& OPENED(Z) & OPENED(W)
& OPENED(V)
CLOSED
CLOSED(X) &
CLOSED(Y) &
CLOSED(Z) &
CLOSED(W) &
CLOSED(V)
WALL
WALL(X) &
WALL(Y) &
WALL(Z) &
WALL(W) &
WALL(V)
MOVE
MOVE(X,Y,Z,W,V)
MOVE(X,Y,Z,W,V) & OPENED(X)
MOVE(X,Y,Z,W,V) & OPENED(Y)
MOVE(X,Y,Z,W,V) & OPENED(Z)
MOVE(X,Y,Z,W,V) & OPENED(W)
MOVE(X,Y,Z,W,V) & OPENED(V)
MOVE(X,Y,Z,W,V) & OPENED(X) &
OPENED(Y) &
OPENED(Z) &
OPENED(W) &
OPENED(V)
OPENED
OPENED(X) &
OPENED(Y) &
OPENED(Z) &
OPENED(W) &
OPENED(V)

Table 1: STRIPS OPERATORS

Reprezentácia plánovacieho problému

- priestor situácií a priestor plánov
 - progresívny a regresívny plánovač
 - least commitment
- reprezentácia plánov
 - množina krokov
 - množina ohraničení, určujúcich usporiadanie krokov $S_i \prec S_j$
 - množina ohraničení na priradenia premenným $v = x$
 - množina príčinných spojení $S_i \rightarrow^c S_j$

Reprezentácia plánovacieho problému

- reprezentácia plánov (pokr.)

Začni  Skonči

*Op(AKCIA:ZačniNákup,
ÚČINKY:NachádzamSa(Doma) \wedge
PredávaSa(Trafika, Noviny) \wedge
PredávaSa(PredajňaPotravín, Mlieko) \wedge
PredávaSa(PredajňaPotravín, Rožky))*

*Op(AKCIA: SkončiNákup,
PREDPOD: NachádzamSa(Doma) \wedge
Mám(Noviny) \wedge Mám(Mlieko) \wedge Mám(Rožky))*

*Plán(KROKY: { S_1 ; Op(AKCIA: ZačniNákup),
 S_2 ; Op(AKCIA: SkončiNákup)}},*

USPORIADANIA:{ $S_1 \rightarrow S_2$ }

PRIRADENIA: {}

SPOJENIA: {}})

Reprezentácia plánovacieho problému

- reprezentácia plánov (pokr.)
 - linearizácia plánu

- riešenia

- úplný neprotirečivý plán
 - úplnosť

Krok S_i dosiahne predpomienku c kroku S_j ak:

- $S_i \prec S_j$ a $c \in \text{Účinky}(S_i)$
- neexistuje krok S_k taký, že $(\neg c) \in \text{Účinky}(S_k)$ a $S_i \prec S_k \prec S_j$ vnejakej linearizácii plánu.

- neprotirečivosť

$S_i \prec S_j$ aj $S_j \prec S_i$

$v = A$ aj $v = B$ (A a B sú rozdielne konštanty)

Plánovanie v situačnom počte

- Reprezentácia plánovacieho problému:

- začiatočný stav

$$\begin{aligned} & \text{NachádzamSa}(\text{Doma}, S_0) \wedge \neg \text{Mám}(\text{Mlieko}, S_0) \\ & \wedge \neg \text{Mám}(\text{Rožky}, S_0) \wedge \neg \text{Mám}(\text{Noviny}, S_0) \end{aligned}$$

- cieľový stav

$$\begin{aligned} & \exists s \text{ NachádzamSa}(\text{Doma}, s) \wedge \text{Mám}(\text{Mlieko}, s) \\ & \wedge \text{Mám}(\text{Rožky}, s) \wedge \text{Mám}(\text{Noviny}, s) \end{aligned}$$

- operátory

$$\begin{aligned} & \forall a \forall s \text{ Mám}(\text{Mlieko}, \text{Výsledok}(a, s)) \Leftrightarrow \\ & [(a = \text{Kúp}(\text{Mlieko}) \\ & \wedge \text{NachádzamSa}(\text{PredajňaPotravín}, s) \\ & \vee (\text{Mám}(\text{Mlieko}, s) \wedge a \neq \text{Rozlejem}(\text{Mlieko})))] \end{aligned}$$
$$\forall s \text{ Výsledok}'([], s) = s$$
$$\forall s \text{ Výsledok}'([al p], s) = \text{Výsledok}'(p, \text{Výsledok}(a, s))$$

Plánovanie v situačnom počte

- potom cieľový test:

NachádzamSa(Doma, Výsledok'(p, S₀))

\wedge *Mám(Mlieko, Výsledok'(p, S₀))*

\wedge *Mám(Rožky, Výsledok'(p, S₀))*

\wedge *Mám(Noviny, Výsledok'(p, S₀))*

- a plán:

*p = [Chod' (PredajňaPotravín), Kúp(Mlieko), Kúp(Rožky),
Chod'(Trafika), Kúp(Noviny), Chod'(Doma)]*

Algoritmus čiastočne usporadúvajúceho plánovania

```
function ČUP(stav, cieľ, operátory) returns plán  
  
plán  $\leftarrow$  VYTvor-MINIMÁLNY-PLÁN(stav, cieľ)  
loop do  
    if RIEŠENIE?(plán) then return plán  
    Streba, cieľ  $\leftarrow$  ZVOL'-PODCIEL'(plán)  
    VYBER-OPERÁTOR(plán, operátory, Streba, cieľ)  
    VYRIEŠ-OHROZENIA(plán)  
end
```

```
function ZVOL'-PODCIEL'(plán) returns Streba, cieľ  
    vezmi krok plánu Streba z KROKY(plán)  
        s predpomienkou c, ktorá sa ešte nedosiahla  
    return Streba, c
```

Algoritmus čiastočne usporadúvajúceho plánovania

```
procedure VYBER-OPERÁTOR(plán, operátory,  $S_{treba}$ , c)
    choose krok  $S_{pridaj}$  z operátory alebo z KROKY(plán)
        taký, ktorý má za účinok c

    if taký krok neexistuje then fail
    pridaj príčinné spojenie  $S_{pridaj} \rightarrow^c S_{treba}$  do SPOJENIA(plán)
    pridaj usporadúvajúce spojenie  $S_{pridaj} \leftarrow S_{treba}$  do USPORIADANIA(plán)

    if  $S_{pridaj}$  je krok, ktorý sa teraz pridal z operátory then
        pridaj  $S_{pridaj}$  do KROKY(plán)
        pridaj Začni  $\leftarrow S_{pridaj} \leftarrow$  Skonči do USPORIADANIA(plán)
```

Algoritmus čiastočne usporadúvajúceho plánovania

```
procedure VYRIEŠ-OHROZENIA(plán)
    for each  $S_{hrozí}$  ohrozujúci spojenie  $S_i \rightarrow^c S_j$  z SPOJENIA(plán)
    do
        choose bud'
            Predsunutie: pridaj  $S_{hrozí} \leftarrow S_i$  do USPORIADANIA(plán)
            Odsunutie: pridaj  $S_j \leftarrow S_{hrozí}$  do USPORIADANIA(plán)
        if not ZLUČITEĽNÝ(plán) then fail
    end
```

Algoritmus čiastočne usporadúvajúceho plánovania

- regresívny plánovač
- ohrozenia množiny na priradenia premenným:

```
procedure VYBER-OPERÁTOR(plán, operátory,  $S_{treba}$ , c)
    choose krok  $S_{pridaj}$  z operátory alebo z KROKY(plán)
        taký, ktorý má za účinok  $c_{pridaj}$ 
        taký, že  $u = \text{Unify}(c, c_{pridaj}, \text{PRIRADENIA}(plán))$ 

    if taký krok neexistuje then fail
    pridaj  $u$  do PRIRADENIA(plán))
    pridaj príčinné spojenie  $S_{pridaj} \rightarrow^c S_{treba}$  do SPOJENIA(plán)
    pridaj usporadúvajúce spojenie  $S_{pridaj} \leftarrow S_{treba}$  do USPORIADANIA(plán)

    if  $S_{pridaj}$  je krok, ktorý sa teraz pridal z operátory then
        pridaj  $S_{pridaj}$  do KROKY(plán)
        pridaj Začni  $\leftarrow S_{pridaj} \leftarrow Skonči$  do USPORIADANIA(plán)
```

Algoritmus čiastočne usporadúvajúceho plánovania

```
procedure VYRIEŠ-OHROZENIA(plán)
    for each  $S_i \rightarrow^c S_j$  in SPOJENIA(plán) do
        for each  $S_{hrozí}$  in KROKY(plán) do
            for each  $c'$  in =ÚČINKY( $S_{hrozí}$ ) do

                if SUBST(PRIRADENIA(plán, c)) =
                    SUBST(PRIRADENIA(plán),  $\neg c'$ ), then
                    choose bud'
                        Predsunutie:
                            pridaj  $S_{hrozí} \leftarrow S_i$  do USPORIADANIA(plán)
                        Odsunutie:
                            pridaj  $S_j \leftarrow S_{hrozí}$  do USPORIADANIA(plán)
                    if not ZLUČITEĽNÝ(plán) then fail

            end
        end
    end
```

Algoritmus čiastočne usporadúvajúceho plánovania

- rozšírenie definície dosiahnutia podmienky krokom v pláne

Krok S_i dosiahne predpodmienku c kroku S_j ak:

- $S_i \prec S_j$ a $\text{UNIFY}(c, u)$ pre nejaký účinok $u \in \text{ÚČINKY}(S_i)$
- neexistuje krok S_k taký, že $\text{UNIFY}(\neg c, u)$ pre nejaký účinok $u \in \text{ÚČINKY}(S_k)$ a $S_i \prec S_k \prec S_j$ vnejakej linearizácii plánu

niektoré aplikácie plánovania pomocou umelej inteligencie

- vojenské operácie
- operácie v kontajnerových prístavoch
- konštrukčné úlohy
- autonómne riadenie družíc a iných vesmírnych lodí



plánovacie domény skutočného sveta

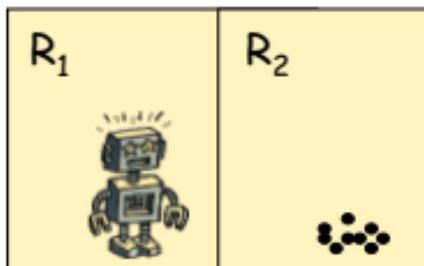
- domény skutočného sveta sú zložité a nemusia splňať predpoklady jazyka ako je STRIPS alebo metód ako je čup
- v skutočnom svete sa možno budeme musieť bojiť s takýmito charakteristikami:
 - Modelovanie a usudzovanie o **zdrojoch**
 - Reprezentovanie a usudzovanie o **čase**
 - Plánovanie na rôznych úrovniach **abstrakcie**
 - **Podmienené** výsledky akcií
 - **Neurčité** výsledky akcií
 - **Vonkajšie** (zvonku pôsobiace, exogénne) udalosti
 - **Inkrementálne navrhovanie plánu**
 - **Dynamické preplánovanie v reálnom čase**

}

a.k.a.

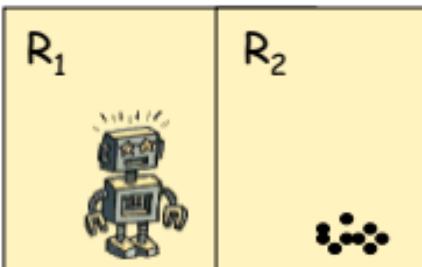
rozvrhovanie!

príklad: robot vysávač



- dve miestnosti: R_1 and R_2
- robot vysávač
- prach

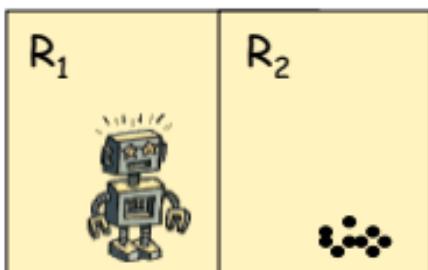
reprezentovanie stavu


$$\text{In(Robot, R}_1\text{)} \wedge \text{Clean(R}_1\text{)}$$

výroky/tvrdenia, ktoré
"platia"
(t.j. sú pravdivé)
v súčasnom stave

logická spojka
"a"

reprezentovanie stavu



$\text{In}(\text{Robot}, \text{R}_1) \wedge \text{Clean}(\text{R}_1)$

- konjunkcia tvrdení
- nie je možné negované tvrdenie napr. $\neg \text{Clean}(\text{R}_2)$
- **predpoklad uzavretého sveta:** Každé tvrdenie, ktoré sa v danom stave neuvádza, je v tom stave nepravdivé
- nie je logická spojka "alebo" $\text{In}(\text{Robot}, \text{R}_1) \vee \text{In}(\text{Robot}, \text{R}_2)$
- nie sú premenné, napr. $\exists x \text{ Clean}(x)$

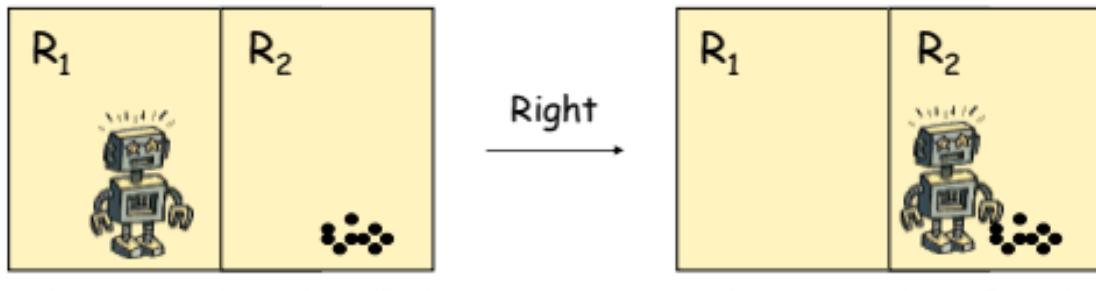
príklad: $\text{Clean}(R_1) \wedge \text{Clean}(R_2)$

- konjunkcia tvrdení
- neobsahuje negované tvrdenie
- neobsahuje spojku “alebo”
- neobsahuje premennú

Cieľ G sa dosiahne v stave S, ak všetky tvrdenia v G (nazývané podciele) sú tiež v S

Right

- Precondition = $\text{In}(\text{Robot}, R_1)$
- Delete-list = $\text{In}(\text{Robot}, R_1)$
- Add-list = $\text{In}(\text{Robot}, R_2)$



$\text{In}(\text{Robot}, R_1) \wedge \text{Clean}(R_1)$

$\text{In}(\text{Robot}, R_2) \wedge \text{Clean}(R_1)$

Right

- ■ Precondition = $\text{In}(\text{Robot}, R_1)$
- ■ Delete-list = $\text{In}(\text{Robot}, R_1)$
- ■ Add-list = $\text{In}(\text{Robot}, R_2)$

množiny tvrdení

rovnaký tvar ako ciele: konjunkcia tvrdení

Right

- Precondition = $In(Robot, R_1)$
 - Delete-list = $In(Robot, R_1)$
 - Add-list = $In(Robot, R_2)$
-
- Akcia A je aplikovateľná v stave S, ak tvrdenia v jej predpodmienke sú všetky v S
 - Aplikovaním akcie A na stav S vzniká nový stav taký, že sa vymažú z S tvrdenia uvedené v zozname na zrušenie (delete-list) a pridajú sa tvrdenia uvedené v zozname na pridanie (add-list)

Left

- $P = In(Robot, R_2)$
- $D = In(Robot, R_2)$
- $A = In(Robot, R_1)$

$Suck(R_1)$

- $P = In(Robot, R_1)$
- $D = \emptyset$ [empty list]
- $A = Clean(R_1)$

$Suck(R_2)$

- $P = In(Robot, R_2)$
- $D = \emptyset$ [empty list]
- $A = Clean(R_2)$

Left

- $P = \text{In}(\text{Robot}, R_2)$
- $D = \text{In}(\text{Robot}, R_2)$
- $A = \text{In}(\text{Robot}, R_1)$

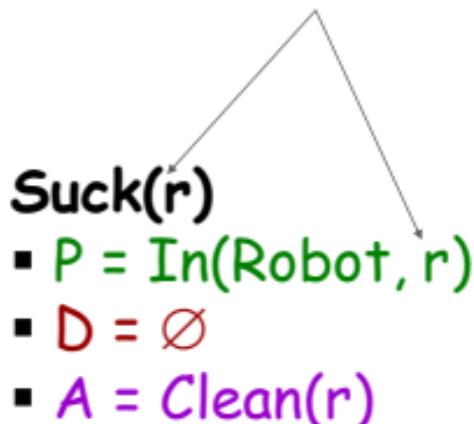
Suck(r)

- $P = \text{In}(\text{Robot}, r)$
- $D = \emptyset$ [prázdny zoznam]
- $A = \text{Clean}(r)$

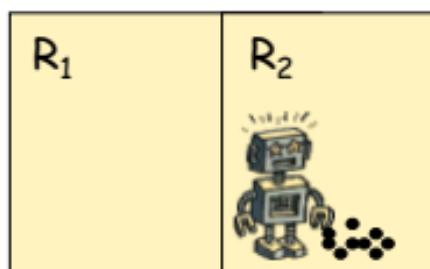
jedna schéma opisuje viacero akcií, v tomto príklade:

$\text{Suck}(R_1)$ and $\text{Suck}(R_2)$

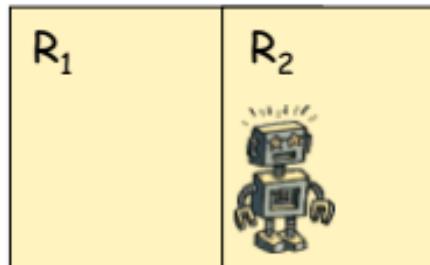
Parameter, ktorý sa nainštaluje prirovnáním
(matching) predpodmienky a stavu



akciová schéma



$\xrightarrow{\text{Suck}(R_2)}$

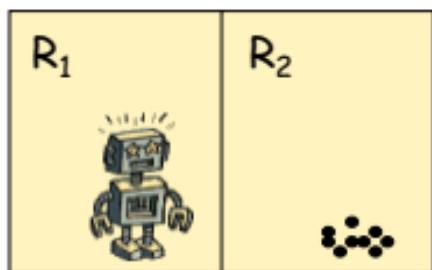


$\text{In}(\text{Robot}, R_2) \wedge \text{Clean}(R_1)$

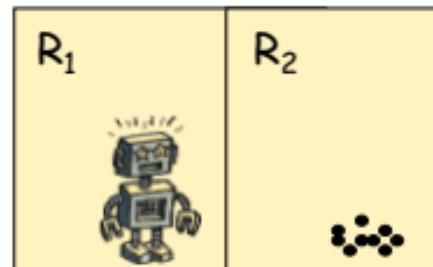
$\text{In}(\text{Robot}, R_2) \wedge \text{Clean}(R_1)$
 $\wedge \text{Clean}(R_2)$

$r \leftarrow R_2$ { **Suck(r)**
■ $P = \text{In}(\text{Robot}, r)$
■ $D = \emptyset$
■ $A = \text{Clean}(r)$

akciová schéma



$\text{In}(\text{Robot}, R_1) \wedge \text{Clean}(R_1)$

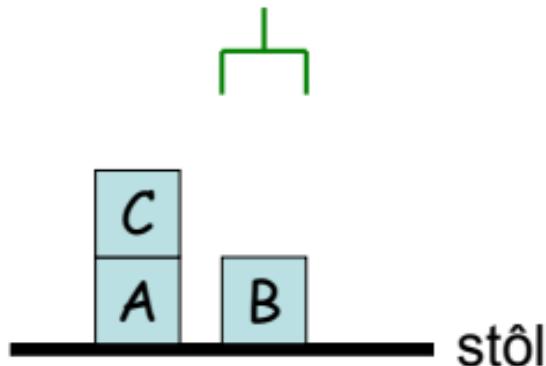


$\text{In}(\text{Robot}, R_1) \wedge \text{Clean}(R_1)$

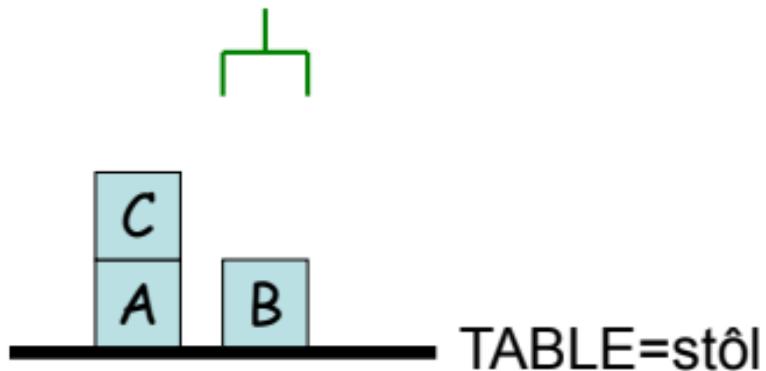
$r \leftarrow R_1$ **Suck(r)**
 {

- $P = \text{In}(\text{Robot}, r)$
- $D = \emptyset$
- $A = \text{Clean}(r)$

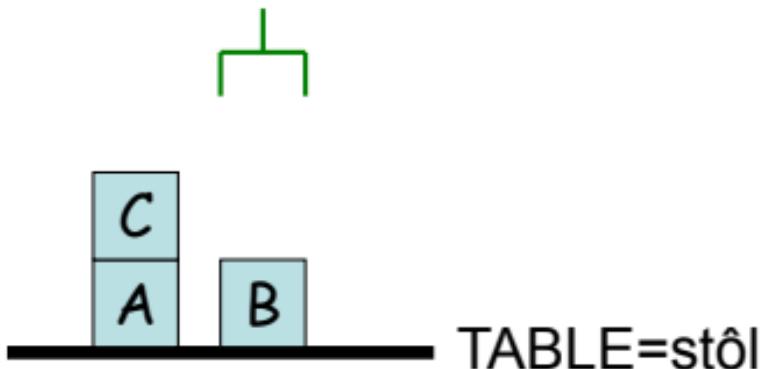
príklad: svet kociek



- robot dokáže ramenom pohybovať kocky na stole
- rameno nemôže držať viac než jednu kocku v jednom okamihu
- nijaké dve kocky sa nezmestia priamo na tú istú kocku
- stôl je ľubovoľne veľký

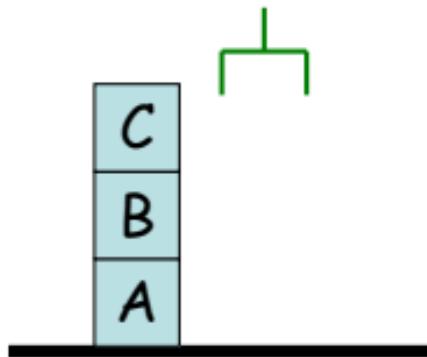


$\text{Block}(A) \wedge \text{Block}(B) \wedge \text{Block}(C) \wedge$
 $\text{On}(A,\text{stôl}) \wedge \text{On}(B,\text{stôl}) \wedge \text{On}(C,A) \wedge$
 $\text{Clear}(B) \wedge \text{Clear}(C) \wedge \text{Handempty}$



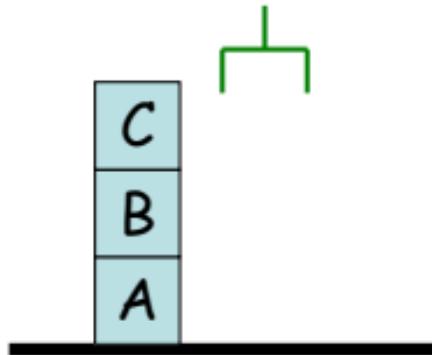
kocka(A) \wedge kocka(B) \wedge kocka(C) \wedge
Na(A,stôl) \wedge Na(B,stôl) \wedge Na(C,A) \wedge
čistá(B) \wedge čistá(C) \wedge prázdneRameno

ciel



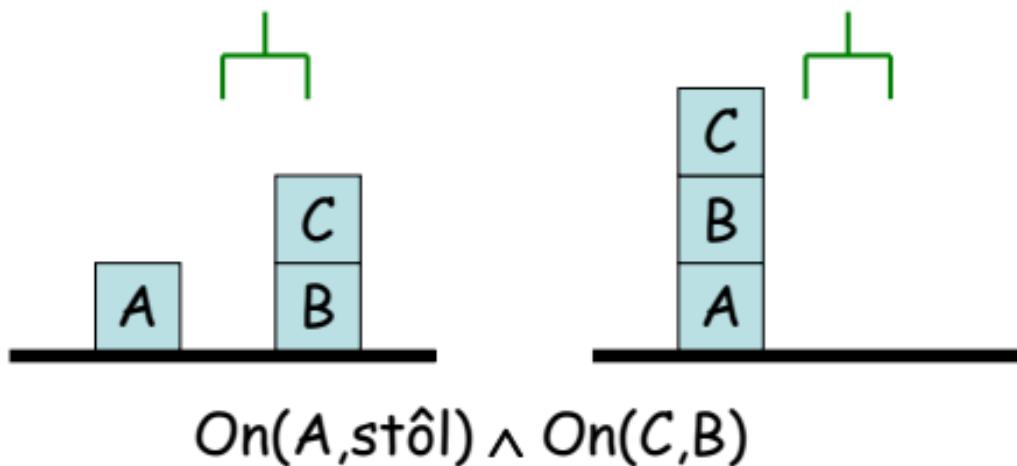
$\text{On}(A, \text{stôl}) \wedge \text{On}(B, A) \wedge \text{On}(C, B) \wedge \text{Clear}(C)$

ciel'



$\text{On}(A, \text{stôl}) \wedge \text{On}(B, A) \wedge \text{On}(C, B) \wedge \text{Clear}(C)$

ciel



Unstack(x,y)

P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge
On(x,y)

D = Handempty, Clear(x), On(x,y)

A = Holding(x), Clear(y)

Unstack(x,y)

P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge On(x,y)

D = Handempty, **Clear(x), On(x,y)**

A = Holding(x), Clear(y)

vlastne netreba, iba ak má
robot viac ramien

Unstack(x,y)

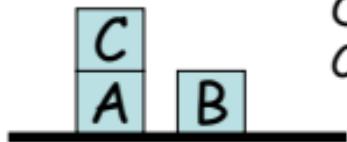
P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge On(x,y)

D = Handempty, Clear(x), On(x,y)

A = Holding(x), Clear(y)



Block(A) \wedge Block(B) \wedge Block(C) \wedge
 On(A , stôl) \wedge On(B , stôl) \wedge On(C , A) \wedge
 Clear(B) \wedge Clear(C) \wedge Handempty

**Unstack(C,A)**

P = Handempty \wedge Block(C) \wedge Block(A) \wedge Clear(C) \wedge On(C,A)

D = Handempty, Clear(C), On(C,A)

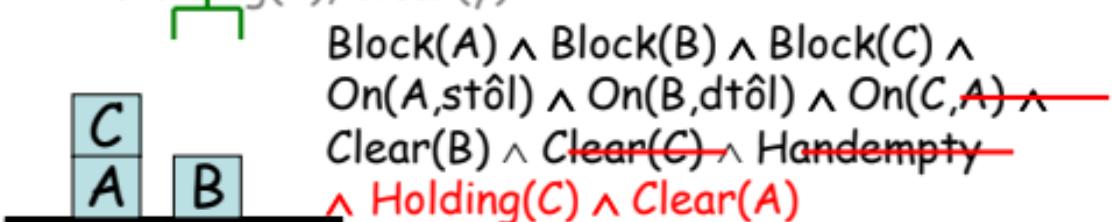
A = Holding(C), Clear(A)

Unstack(x,y)

P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge On(x,y)

D = Handempty, Clear(x), On(x,y)

A = Holding(x), Clear(y)



Unstack(C,A)

P = Handempty \wedge Block(C) \wedge Block(A) \wedge Clear(C) \wedge On(C,A)

D = Handempty, Clear(C), On(C,A)

A = Holding(C), Clear(A)

Unstack(x,y)

P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge On(x,y)

D = Handempty, Clear(x), On(x,y)

A = Holding(x), Clear(y)

Stack(x,y)

P = Holding(x) \wedge Block(x) \wedge Block(y) \wedge Clear(y)

D = Clear(y), Holding(x)

A = On(x,y), Clear(x), Handempty

Pickup(x)

P = Handempty \wedge Block(x) \wedge Clear(x) \wedge On(x,Table)

D = Handempty, Clear(x), On(x,Table)

A = Holding(x)

Putdown(x)

P = Holding(x), \wedge Block(x)

D = Holding(x)

A = On(x,Table), Clear(x), Handempty

všetky akcie

Unstack(x,y)

P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge On(x,y)

D = Handempty, Clear(x), On(x,y)

A = Holding(x), Clear(y)

Stack(x,y)

P = Holding(x) \wedge Block(x) \wedge Block(y) \wedge Clear(y)

D = Clear(y), Holding(x),

A = On(x,y), Clear(x), Handempty

vlastne
netreba,
iba ak
má
robot
viac
ramien

Pickup(x)

P = Handempty \wedge Block(x) \wedge Clear(x) \wedge On(x,Table)

D = Handempty, Clear(x), \neg On(x,Table)

A = Holding(x)

Putdown(x)

P = Holding(x), \wedge Block(x)

D = Holding(x)

A = On(x,Table), Clear(x), Handempty

všetky akcie

Unstack(x,y)

P = Handempty \wedge Block(x) \wedge Block(y) \wedge Clear(x) \wedge On(x,y)

D = Handempty, Clear(x), On(x,y)

A = Holding(x), **Clear(y)**

Stack(x,y)

P = Holding(x) \wedge Block(x) \wedge Block(y) \wedge Clear(y)

D = **Clear(y)**, Holding(x),

A = On(x,y), Clear(x), Handempty

Pickup(x)

P = Handempty \wedge Block(x) \wedge Clear(x) \wedge On(x,Table)

D = Handempty, Clear(x), On(x, TABLE)

A = Holding(x)

Putdown(x)

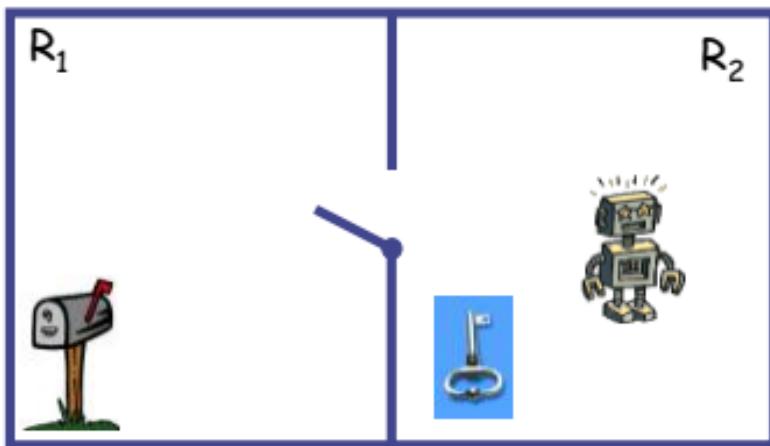
P = Holding(x), \wedge Block(x)

D = Holding(x)

A = On(x, TABLE), Clear(x), Handempty

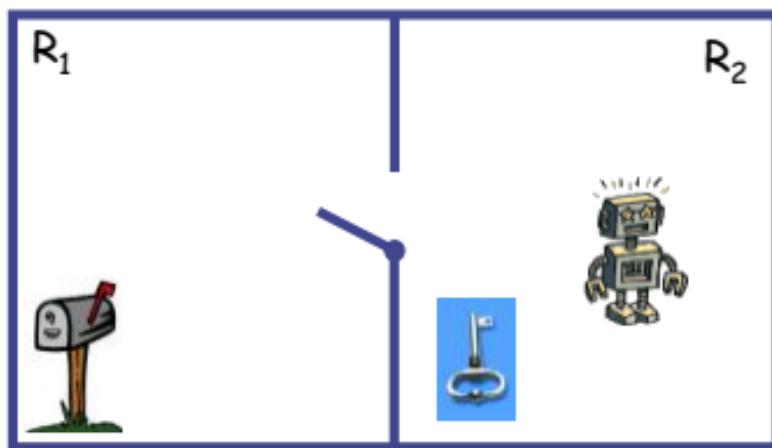
kocka sa vždy zmestí na
stôl

príklad: kľúč v schránke

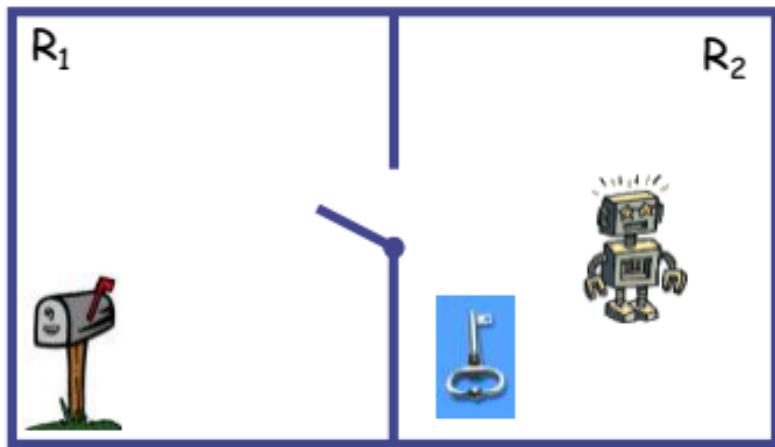


- robot musí zamknúť dvere a vložiť kľúč do schránky
- kľúč treba na zamknutie a odomknutie dverí
- keď sa raz kľúč dostane do schránky, robot ho už nedokáže dostať späť

príklad: kľúč v schránke: začiatočný stav

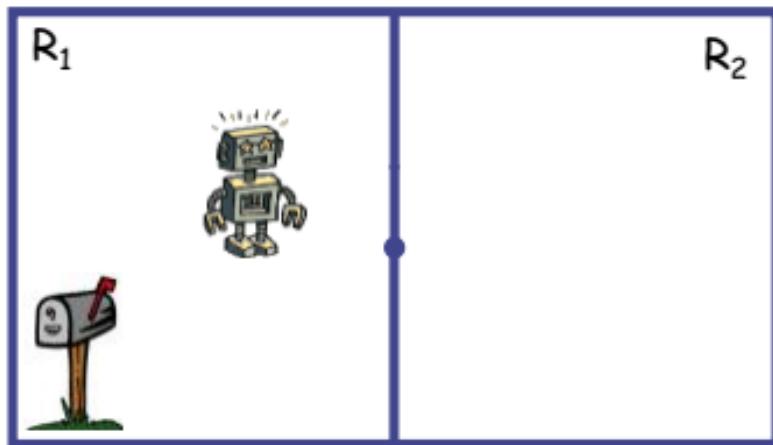

$$\text{In}(\text{Robot}, \mathcal{R}_2) \wedge \text{In}(\text{Kľúč}, \mathcal{R}_2) \wedge \text{Unlocked}(\text{Dvere})$$

príklad: kľúč v schránke: začiatočný stav



$v(\text{Robot}, R_2) \wedge v(\text{Kľúč}, R_2) \wedge \text{odomknuté}(Dvere)$

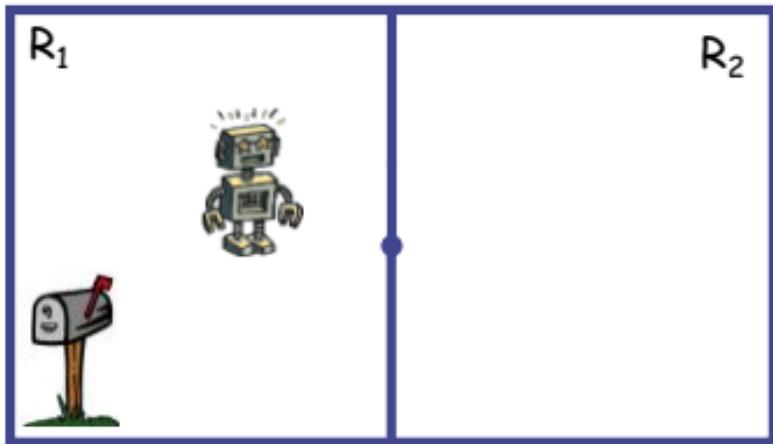
príklad: kľúč v schránke: cieľ



$\text{Locked}(\text{Dvere}) \wedge \text{In}(\text{Kl}\acute{\text{u}}\text{č}, \text{Box})$

[miesto robota sa v cieli nešpecifikuje]

príklad: kľúč v schránke: cieľ



zamknuté(Dvere) \wedge v(Kľúč,Schránka)

[miesto robota sa v cieli nešpecifikuje]

Grasp-Key-in-R₂

P = In(Robot,R₂) \wedge In(Key,R₂)

D = \emptyset

A = Holding(Key)

Lock-Door

P = Holding(Key)

D = \emptyset

A = Locked(Door)

Move-Key-from-R₂-into-R₁

P = In(Robot,R₂) \wedge Holding(Key) \wedge Unlocked(Door)

D = In(Robot,R₂), In(Key,R₂)

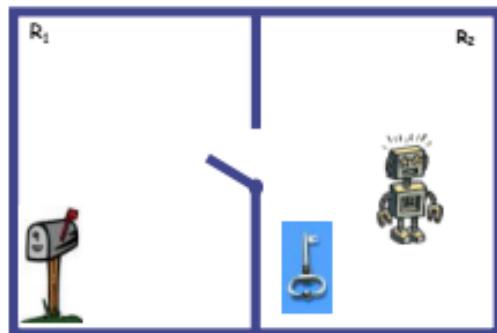
A = In(Robot,R₁), In(Key,R₁)

Put-Key-Into-Box

P = In(Robot,R₁) \wedge Holding(Key)

D = Holding(Key), In(Key,R₁)

A = In(Key,Box)



Uchop-Kľúč-v-R₂

P = v(Robot, R₂) \wedge v(Key, R₂)

D = \emptyset

A = Drží(Kľúč)

Zamkní-Dvere

P = Drží(Kľúč)

D = \emptyset

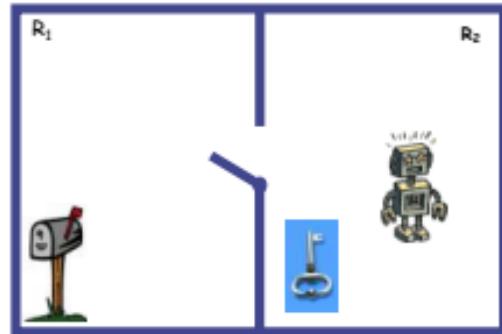
A = Zamknuté(Dvere)

Presuň-Kľúč-z-R₂-do-R₁

P = v(Robot, R₂) \wedge Drží(Kľúč) \wedge Odomknuté(Dvere)

D = v(Robot, R₂), v(Kľúč, R₂)

A = v(Robot, R₁), v(Kľúč, R₁)



Vlož-Kľúč-do-Schránky

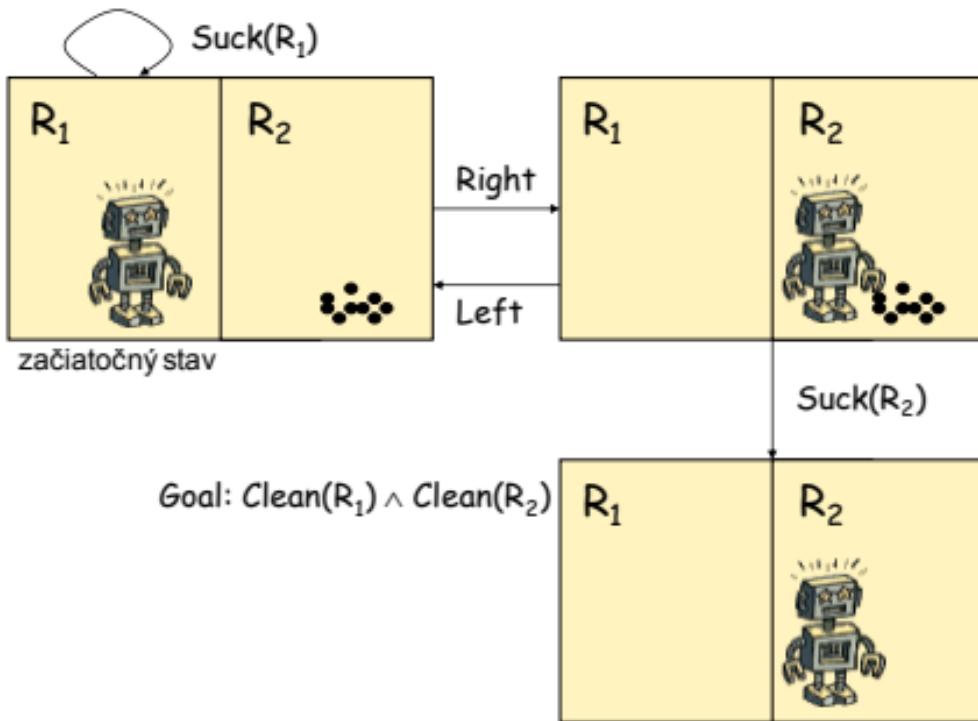
P = v(Robot, R₁) \wedge Drží(Kľúč)

D = Drží(Kľúč), v(Kľúč, R₁)

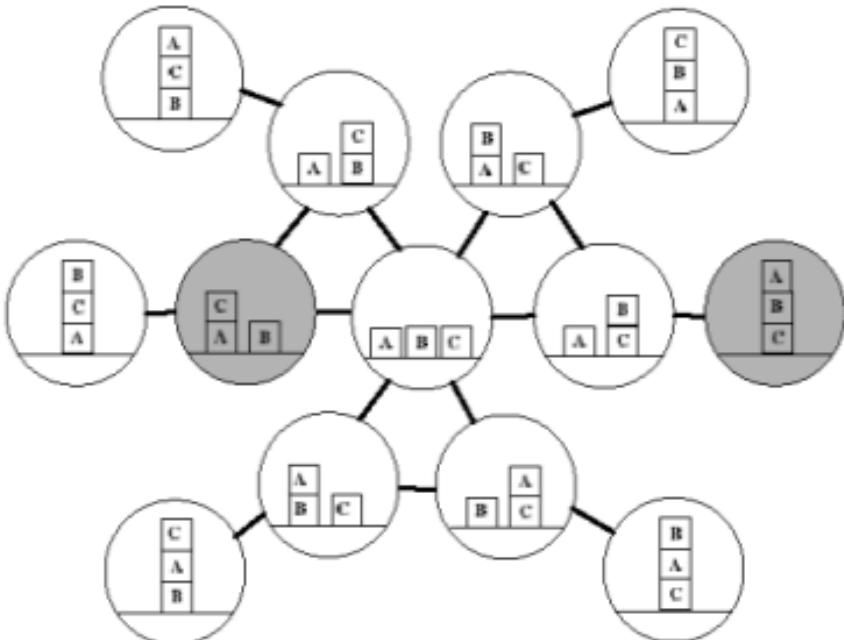
A = v(Key, Box)

Metódy plánovania

dopredné plánovanie

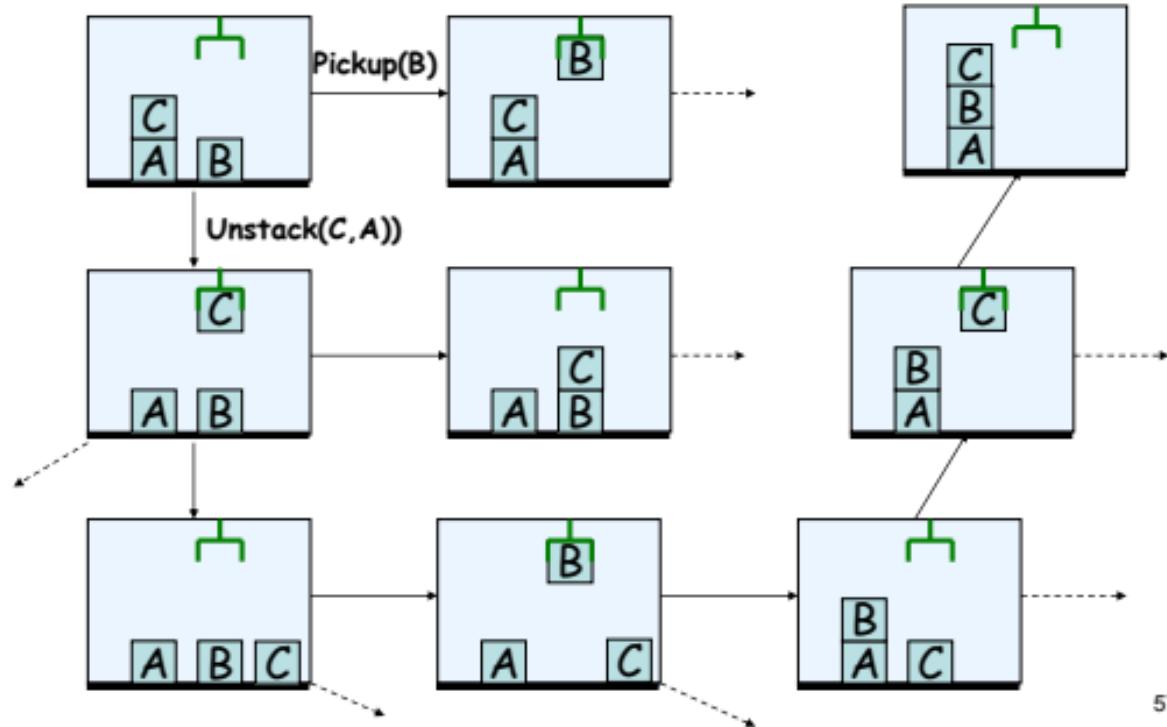


priestor hľadania: svet kociek



dopredné plánovanie

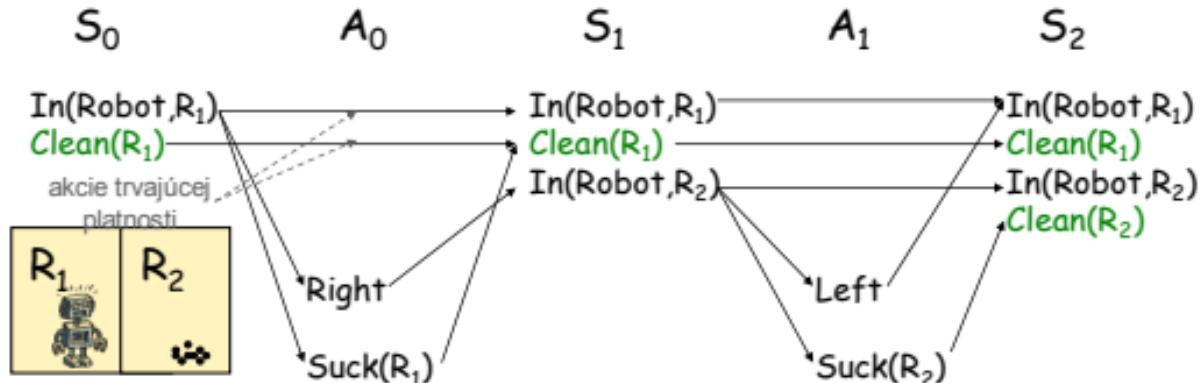
Goal: $On(B,A) \wedge On(C,B)$



potreba presnej heuristiky

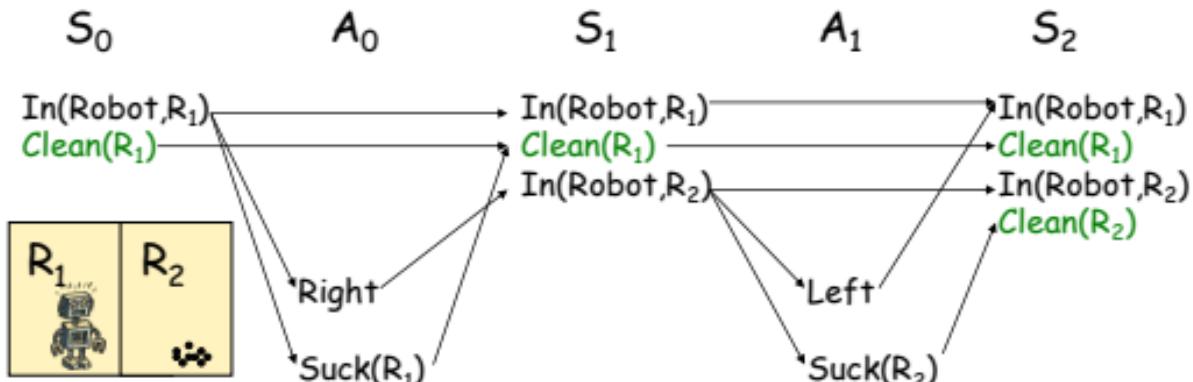
- Dopredné plánovanie jednoducho prehľadáva priestor stavov sveta od začiatočného k cieľovému stavu
- Predstavme si agenta s veľkou knižnicou akcií, ktorých cieľom je G , napr., $G = \text{Mať}(Mlieko)$
- Vo všeobecnosti sa dá v každom stave aplikovať mnoho akcií, takže faktor vetvenia je obrovský
- V ľubovoľnom stave je väčšina akcií irrelevantná z pohľadu dosiahnutia cieľa $\text{Mať}(Mlieko)$
- Našťastie, dá sa počítať konzistentná heuristika použitím plánovacích grafov

plánovací graf pre stav robota vysávača



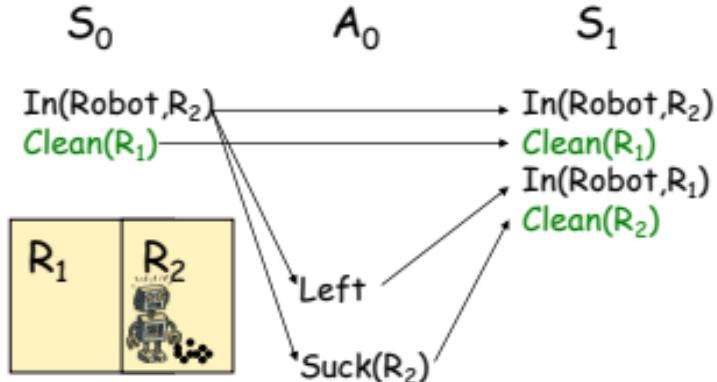
- S_0 obsahuje tvrdenia o stave (tu, o začiatočnom stave)
- A_0 obsahuje všetky akcie, ktorých predpodmienky sa vyskytujú v S_0
- S_1 obsahuje všetky tvrdenia, ktoré boli v S_0 alebo sú v zoznamoch na pridanie akcií v A_0
- V dôsledku toho S_1 obsahuje všetky tvrdenia, ktoré by mohli byť pravdivé v stave dosiahnutom po prvej akcii
- A_1 obsahuje všetky akcie, ktoré už nie sú v A_0 , ktorých predpodmienky sú v S_1 , čiže môžu sa vykonať v stave dosiahnutom po vykonaní prvej akcie. Atd...

plánovací graf pre stav robota vysávača



- Hodnota i taká, že S_i obsahuje všetky cieľové tvrdenia, sa nazýva úrovňová cena (**level cost**) cieľa (tu $i=2$)
- Tak, ako sa zostavuje plánovací graf, je to dolná hranica počtu akcií potrebných na dosiahnutie cieľa
- V tomto príklade je 2 aj skutočná dĺžka najkratšej cesty do cieľa

plánovací graf pre iný stav robota vysávača

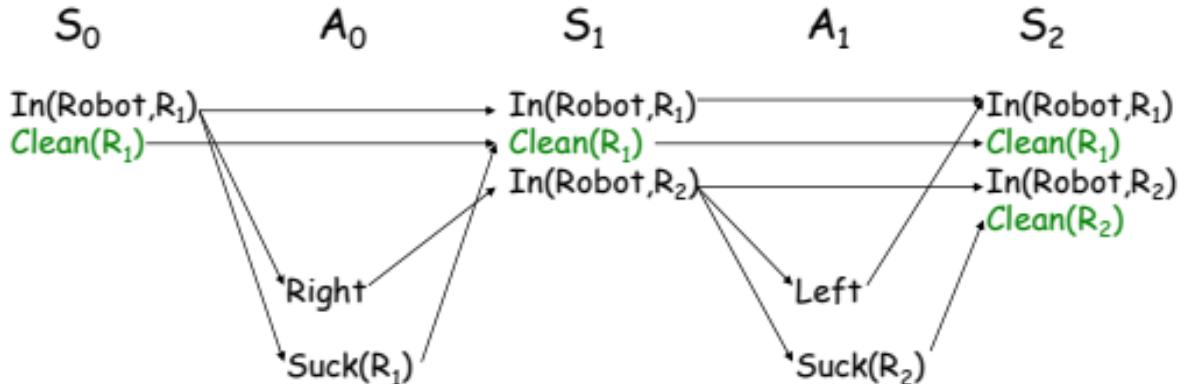


- úrovňová cena cieľa je 1 a je to opäť aj skutočná dĺžka najkratšej cesty do cieľa

použitie plánovacích grafov pri doprednom plánovaní

- Kedykoľvek sa vygeneruje nový uzol, vypočítaj plánovací graf jeho stavu [oprav/update plánovací graf pri rodičovskom uzle]
- Skonči počítanie plánovacieho grafu ak:
 - buď cieľové tvrdenia sú v množine S_i
[vtedy i je úrovňová cena cieľa]
 - alebo ak $S_{i+1} = S_i$
[vtedy vygenerovaný uzol **nie** je na ceste riešenia]
- Nastav hodnotu heuristickej funkcie $h(N)$ pre uzol N na úrovňovú cenu cieľa pre stav reprezentovaný uzlom N
- **h je konzistentná heuristika pre akcie s jednotkovou cenou**
- čiže A* s použitím takejto h bude nachádzať riešenia s minimálnym počtom akcií

veľkosť plánovacieho grafu



- Nejaká akcia sa vyskytuje najviac raz
- Tvrdenie sa pridáva najviac raz a každý S_k ($k \neq i$) je ostrou nadmnožinou S_{k-1}
- Takže počet úrovní je ohraničený
 $\text{Min}\{\text{počet akcií}, \text{počet tvrdení}\}$
- Naproti tomu stavový priestor rastie exponenciálne s počtom tvrdení
- Počítanie plánovacích grafov môže ušetriť mnoho nepotrebného prehľadávania

zlepšenie plánovacieho grafu: vzájomné vylúčenia

- **Ciel (čo chceme dosiahnuť):** zjednime vyjadrenie úrovňovej ceny cieľa, aby bola presnejším odhadom počtu akcií, ktoré treba na dosiahnutie cieľa
- **Metóda:** rozpoznať zjavné vylúčenia medzi tvrdeniami na rovnakej úrovni
- Zvyčajne to vedie k presnejším heuristikám, ale plánovacie grafy budú väčšie a ich počítanie bude drahšie (bude dlhšie trvať)

dopredné plánovanie a jeho nedostatky

- napriek všetkým snaham, dopredné plánovanie je spojené s veľkým faktorom vetvenia
- vo všeobecnosti je omnoho menej akcií, ktoré môžu prispieť k naplneniu cieľa (sú relevantné) než akcií, ktoré sa môžu v nejakom stave vykonať
- **lepšie je potom začať od cieľa**
- Ako určiť, ktoré akcie sú relevantné? Ako ich použiť?

- → spätné plánovanie

akcia relevantná k cieľu

- akcia je **relevantná** k dosiahnutiu cieľa ak tvrdenie v jej opise.add.list sa pripodobí (match) nejakému podcieľu (čo je tiež nejaké tvrdenie)
- napr.:

Stack(B,A)

P = Holding(B) \wedge Block(B) \wedge Block(A) \wedge Clear(A)

D = Clear(A), Holding(B),

A = **On(B,A)**, Clear(B), Handempty

je relevantná k dosiahnutiu

On(B,A) \wedge On(C,B)

Regresia cieľa

regresia cieľa G cez akciu A je najmenej ohraničujúca predpomienka $R[G,A]$ taká, že:

Ak stav S dosiahne $R[G,A]$, tak:

1. predpomienka akcie A je splnená v S
2. aplikovaním akcie A v stave S sa dosiahne stav, v ktorom je splnený G

- $G = \text{On}(B, A) \wedge \text{On}(C, B)$
- **Stack(C, B)**
 - P = Holding(C) \wedge Block(C) \wedge Block(B) \wedge Clear(B)
 - D = Clear(B), Holding(C)
 - A = On(C, B), Clear(C), Handempty
- $R[G, \text{Stack}(C, B)] =$
 - On(B, A) \wedge
 - Holding(C) \wedge Block(C) \wedge Block(B) \wedge Clear(B)

- $G = \text{On}(B,A) \wedge \text{On}(C,B)$
- $\text{Stack}(C,B)$

P = Holding(C) \wedge Block(C) \wedge Block(B) \wedge Clear(B)

D = Clear(B), Holding(C)

A = On(C,B), Clear(C), Handempty

- $R[G, \text{Stack}(C,B)] =$

On(B,A) \wedge

Holding(C) \wedge Block(C) \wedge Block(B) \wedge Clear(B)

ďalší príklad

- $G = \text{In}(\text{key}, \text{Box}) \wedge \text{Holding}(\text{Key})$

- **Put-Key-Into-Box**

$$P = \text{In}(\text{Robot}, R_1) \wedge \text{Holding}(\text{Key})$$

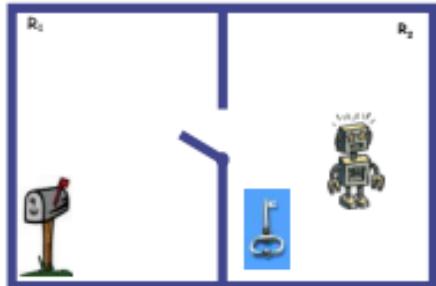
$$D = \text{Holding}(\text{Key}), \text{In}(\text{Key}, \text{Box})$$

$$A = \text{In}(\text{Key}, \text{Box})$$

- $R[G, \text{Put-Key-Into-Box}] = \text{False}$

kde False je nedosiahnuteľný cieľ

- To ale znamená, že $\text{In}(\text{key}, \text{Box}) \wedge \text{Holding}(\text{Key})$ sa nedá dosiahnuť vykonaním akcie **Put-Key-Into-Box**

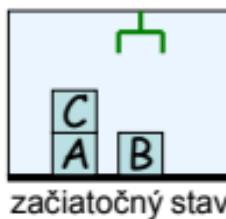


postup počítania R[G,A]

1. ak je ľubovoľný podiel ciela G v zozname na zrušenie v opise akcie A (delete list)
tak return False
2. inak
 - a. $G' \leftarrow$ predpredmienka akcie A
 - b. pre každý podiel SG ciela G vykonaj
ak SG nie je v zozname na pridanie v opise akcie A (add list) tak pridaj SG do G'
3. Return G'

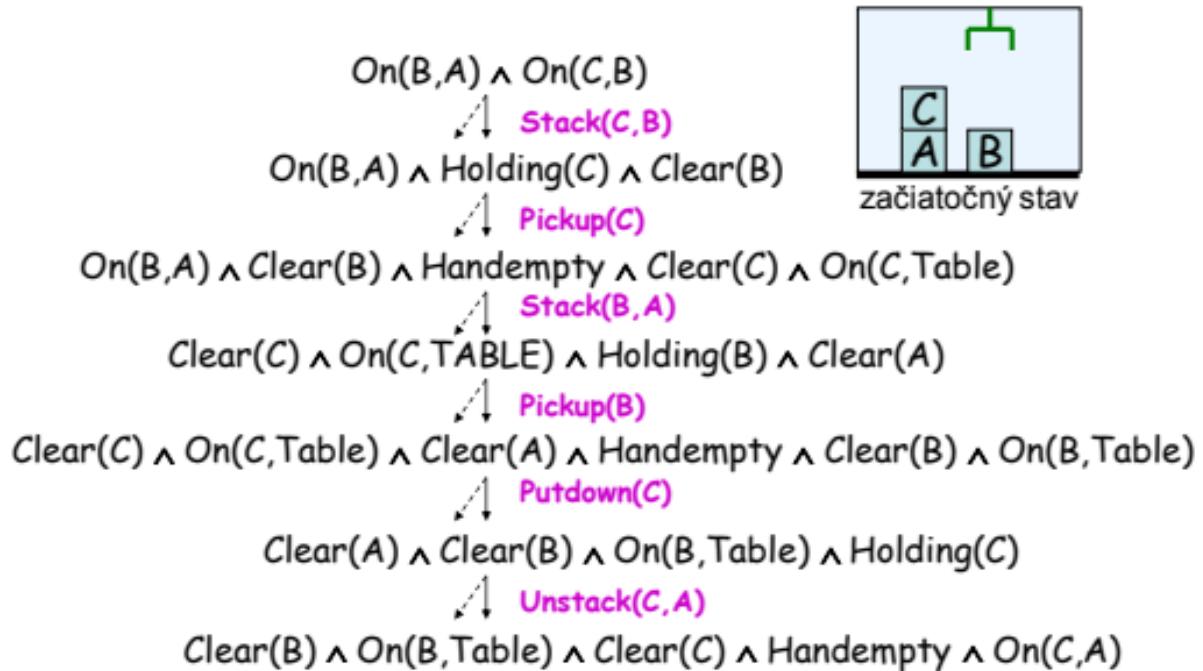
spätné plánovanie (backward planning)

$On(B, A) \wedge On(C, B)$

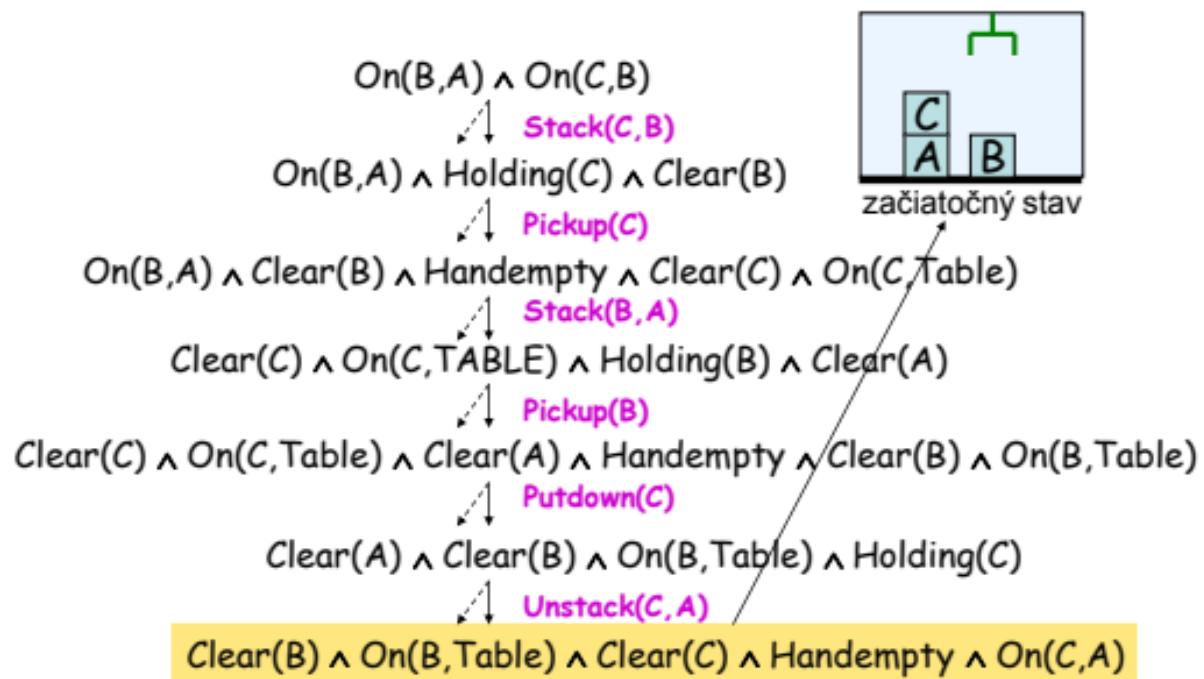


začiatočný stav

spätné plánovanie



spätné plánovanie



strom hľadania

- Spätné plánovanie prehľadáva priestor cieľov od pôvodného cieľa problému k cieľu, ktorý je splnený v začiatočnom stave
- Zvyčajne je omnoho menej akcií relevantných k nejakému cieľu, než je akcií, aplikovateľných v nejakom stave → menší faktor vetvenia než pri doprednom plánovaní
- Dĺžky ciest riešenia sú rovnaké

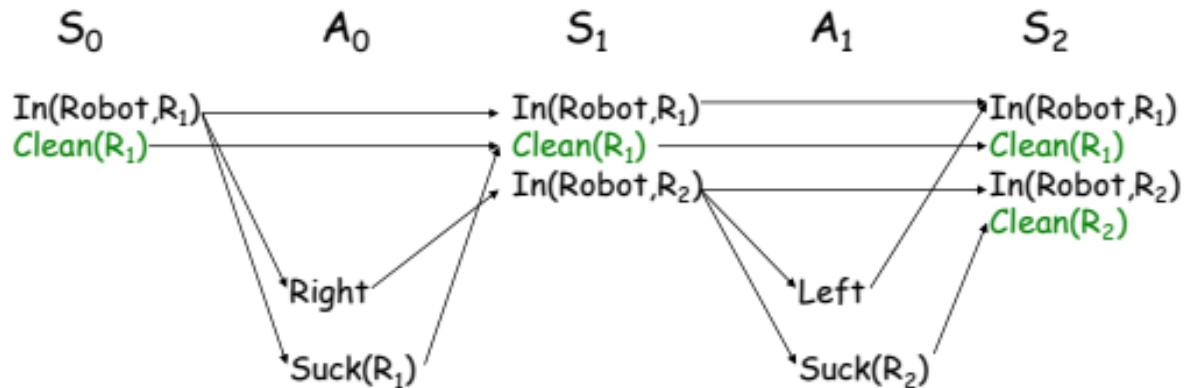
konzistentná heuristika pre spätné plánovanie

postup:

1. Pred-vypočítať plánovací graf začiatočného stavu až pokial' neklesne úroveň na nulu
2. pre každý uzol N , ktorý sa pridal do stromu hľadania, stanoviť hodnotu heuristickej funkcie $h(N)$ ako hodnotu úrovňovej ceny cieľa združeného s uzlom N

Ak sa cieľ združený s uzlom N nedá splniť v nijakej množine S_k plánovacieho grafu, nedá sa vôbec dosiahnuť – useknúť ho!

počíta sa len jeden plánovací graf



Ako spätné plánovanie rozpozná slepé vetvy?

$On(B,A) \wedge On(C,B)$

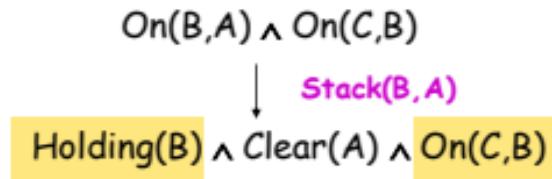
A diagram illustrating a stack operation. It consists of two parts: a mathematical expression $On(B,A) \wedge On(C,B)$ at the top, and below it, a downward-pointing arrow with a bracket underneath it. To the right of the arrow, the word "Stack" is followed by (C,B) .

Stack(C, B)

Ako spätné plánovanie rozpozná slepé vetvy?



Ako spätné plánovanie rozpozná slepé vetvy?



stavové ohraničenie napr.

$$\text{Holding}(x) \rightarrow \neg(\exists y)\text{On}(y,x)$$

by umožnilo useknúť túto vetvu skôr

Progresia: dopredné hľadanie (I -> G)

ProgWS(stav, ciele, akcie, cesta)

If stav splňa ciele, then return cesta

else a = **choose**(akcie), takú, že

predpodomienky(a) sú splnené v stave

if nie je taká a, then return neúspech

else return

ProgWS(apply(a, stav), ciele, akcie,
pridaj(cesta, a))

prvé volanie: ProgWS(*IS*, *G*, *Akcie*, ())

Príklad dopredného hľadania

I: (on-table A) (on C A) (on-table B) (clear B) (clear C)

G: (on A B) (on B C)

- P(I, G, AkcieSvetaKociek, ())
- P(S1, G, ASK, (Unstack&Putdown(C, A))
- P(S2, G, ASK, (Unstack&Putdown(C, A),
Stack(B, C))
- P(S3, G, ASK, (Unstack&Putdown(C, A),
Stack(B, C),
Stack(A, B))

nedeterministický
výber!

$G \subseteq S3 \Rightarrow \text{úspech!}$

Regresia: spätné hľadanie ($I \leftarrow G$)

RegWS(*init-stav*, *súčasné-ciele*, *akcie*, *cesta*)

If *init-stav* splňa *súčasné-ciele*, then return *cesta*
else a = **choose**(*akcie*), takú, že nejaký účinok *akcie* a splňa
jeden zo *súčasných-cieľov*
If nie je taká akcia a, then return neúspech
[nedosiahnuteľný*]

If nejaký účinok a je v rozpore s niektorým *súčasným-cieľom*,
then return neúspech [nezlučiteľný stav]

$CG' = súčasné-ciele - účinky(a) + predpodmienky(a)$

If *súčasné-ciele* $\subset CG'$, then return neúspech [zbytočné*]

RegWS(*init-stav*, CG' , *akcie*, *pridaj(a, cesta)*)

prvé volanie: RegWS(*IS*, *G*, *Akcie*, ())

príklad spätného hľadania

I: (on-table A) (on C A) (on-table B) (clear B) (clear C)

G: (on A B) (on B C)

- R(I, G, AkcieSvetaKociek, ())
- R(I, ((clear A) (on-table A) (clear B) (on B C)), ASK,
(Stack(A, B)))
- R(I, ((clear A) (on-table A) (clear B) (clear C), (on-table B)),
ASK, (Stack(B, C), Stack(A, B)))
- R(I, ((on-table A) (clear B) (clear C) (on-table B) (on C A)),
ASK, (Unstack&Pushdown(C, A), Stack(B, C),
Stack(A, B)))

nedeterministický
výber!



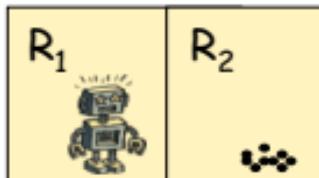
súčasné-ciele \subseteq I \Rightarrow úspech!

Progresia vs. Regresia

- oba algoritmy sú:
 - správne: výsledný plán je platný
 - úplné: ak existuje platný plán, nájdu ho
- nedeterministický výber => hľadanie!
 - neinformované: DFS, BFS, Iterative Deepening, ...
 - Heuristické: A*, IDA*, ...
- zložitosť: $O(b^n)$ v najhoršom prípade
 b = faktor vetvenia, n = |“choose”|
- Regresia: často menší faktor vetvenia b

Niekteré rozšírenia jazyka STRIPS

1. Negované tvrdenia v opise stavu



$\text{In}(\text{Robot}, \mathcal{R}_1) \wedge \neg \text{In}(\text{Robot}, \mathcal{R}_2) \wedge \text{Clean}(\mathcal{R}_1) \wedge \neg \text{Clean}(\mathcal{R}_2)$

Dump-Dirt(r)

$$\begin{aligned} P &= \text{In}(\text{Robot}, r) \wedge \text{Clean}(r) \\ E &= \neg \text{Clean}(r) \end{aligned}$$

Suck(r)

$$\begin{aligned} P &= \text{In}(\text{Robot}, r) \wedge \neg \text{Clean}(r) \\ E &= \text{Clean}(r) \end{aligned}$$

- $Q \vee E$ znamená zrušiť $\neg Q$ a pridať Q do stavu
- $\neg Q \vee E$ znamená zrušiť Q a pridať $\neg Q$

Predpoklad otvoreného sveta: Tvrdenie v stave je pravdivé ak nie je negované, inak je nepravdivé. Pravdivosť tvrdenia, ktoré nie je v stave uvedené, je neznáma.

Metódy plánovania sa dajú relativne ľahko rozšíriť tak, aby pracovali aj s negáciou. Len opisy stavov veľmi narastú (predstavme si, že by sa robot pohyboval v budove s 50 miestnosťami)

2. predikáty rovnosti/rôznosti

svet kociek:

Move(x,y,z)

P = Block(x) \wedge Block(y) \wedge Block(z) \wedge On(x,y) \wedge Clear(x)
 \wedge Clear(z) \wedge (x \neq z)

D = On(x,y), Clear(z)

A = On(x,z), Clear(y)

Move(x,Table,z)

P = Block(x) \wedge Block(z) \wedge On(x,Table) \wedge Clear(x)
 \wedge Clear(z) \wedge (x \neq z)

D = On(x,y), Clear(z)

A = On(x,z)

Move(x,y,Table)

P = Block(x) \wedge Block(y) \wedge On(x,y) \wedge Clear(x)

D = On(x,y)

A = On(x,Table), Clear(y)

2. predikáty rovnosti/rôznosti

svet kociek:

Move(x,y,z)

P = Block(x) \wedge Block(y) \wedge Block(z) \wedge On(x,y) \wedge Clear(x)
 \wedge Clear(z) \wedge ($x \neq z$)

D = On(x,y), Clear(z)

A = On(x,z), Clear(y)

Move(x,Table,z)

P = Block(x) \wedge Block(z) \wedge O
 \wedge Clear(z) \wedge ($x \neq z$)

D = On(x,y), Clear(z)

A = On(x,z)

Move(x,y,Table)

P = Block(x) \wedge Block(y) \wedge O
D = On(x,y)

A = On(x,Table), Clear(y)

Plánovacie metódy jednoducho vyhodnotia ($x \neq z$) keď sú obe premenné nainštalované

To je ekvivalentné tomu, ako by sa uvažovalo, že tvrdenia ($A \neq B$), ($A \neq C$), ... sú implicitne pravdivé v každom stave

3. Algebraické výrazy

Dve flaše F_1 a F_2 majú objemy 30 a 50

F_1 obsahuje 20 nejakej tekutiny

F_2 obsahuje 15 tej istej tekutiny

Stav:

$\text{Cap}(F_1, 30) \wedge \text{Cont}(F_1, 20) \wedge \text{Cap}(F_2, 50) \wedge \text{Cont}(F_2, 15)$

Akcia prelivania (pour) obsahu jednej flašky do druhej:

Pour(f, f')

$P = \text{Cont}(f, x) \wedge \text{Cap}(f', c') \wedge \text{Cont}(f', y)$

$D = \text{Cont}(f, x), \text{Cont}(f', y),$

$A = \text{Cont}(f, \max\{x+y-c', 0\}), \text{Cont}(f', \min\{x+y, c'\})$

3. Algebraické výrazy

Dve fľaše F_1 a F_2 majú objemy 30 a 50

F_1 obsahuje 20 nejakej tekutiny

F_2 obsahuje 15 tej sitej tekutiny

Stav:

$\text{Cap}(F_1)$ plánovacie metódy „vedeli“ aj algebraické manipulácie

Akcia preli

Pour(f, f')

$$P = \text{Cont}(f, x) \wedge \text{Cap}(f', c') \wedge \text{Cont}(f', y)$$

$$D = \text{Cont}(f, x), \text{Cont}(f', y),$$

$$A = \text{Cont}(f, \max\{x+y-c', 0\}), \text{Cont}(f', \min\{x+y, c'\})$$

4. Stavové ohraničenia

h	b	
c	d	g
e	a	f

Stav:

$$\begin{aligned} & \text{Adj}(1,2) \wedge \text{Adj}(2,1) \wedge \dots \wedge \text{Adj}(8,9) \wedge \text{Adj}(9,8) \wedge \\ & \text{At}(h,1) \wedge \text{At}(b,2) \wedge \text{At}(c,4) \wedge \dots \wedge \text{At}(f,9) \wedge \text{Empty}(3) \end{aligned}$$

Move(x,y)

$$P = \text{At}(x,y) \wedge \text{Empty}(z) \wedge \text{Adj}(y,z)$$

$$D = \text{At}(x,y), \text{Empty}(z)$$

$$A = \text{At}(x,z), \text{Empty}(y)$$

4. Stavové ohraničenia

h	b	
c	d	g
e	a	f

Stav:

$$\text{Adj}(1,2) \wedge \cancel{\text{Adj}(2,1)} \wedge \dots \wedge \text{Adj}(8,9) \wedge \cancel{\text{Adj}(9,8)} \wedge \\ \text{At}(h,1) \wedge \text{At}(b,2) \wedge \text{At}(c,4) \wedge \dots \wedge \text{At}(f,9) \wedge \text{Empty}(3)$$

Stavové ohraničenie:

$$\text{Adj}(x,y) \rightarrow \text{Adj}(y,x)$$

Move(x,y)

$$P = \text{At}(x,y) \wedge \text{Empty}(z) \wedge \text{Adj}(y,z)$$

$$D = \text{At}(x,y), \text{Empty}(z)$$

$$A = \text{At}(x,z), \text{Empty}(y)$$

zložitejšie stavové ohraničenia v FOL

svet kociek:

$$(\forall x)[\text{Block}(x) \wedge \neg(\exists y)\text{On}(y,x) \wedge \neg\text{Holding}(x)] \rightarrow \text{Clear}(x)$$

$$(\forall x)[\text{Block}(x) \wedge \text{Clear}(x)] \rightarrow \neg(\exists y)\text{On}(y,x) \wedge \neg\text{Holding}(x)$$

$$\text{Handempty} \leftrightarrow \neg(\exists x)\text{Holding}(x)$$

takéto tvrdenia by veľmi zjednodušili opisy akcií

Stavové ohraničenia si vyžadujú plánovacie metódy, ktoré „vedia“ usudzovať, aby rozhodli, či sú ciele dosiahnuté a ohraničenia splnené

UMELÁ INTELIGENCIA

Strojové učenie sa

Strojové učenie sa

- Simon [1983] navrhol, aby sa pod pojmom učenie sa rozumeli:

*„... zmeny v systéme, ktoré sú adaptívne v tom zmysle,
že umožňujú, aby systém splnil tú istú úlohu alebo úlohy z
tej istej triedy úloh napodruhé efektívnejšie a účinnejšie.“*

- Bifľovanie
 - Samuel (1963): zapamätanie si ohodnotení z predchádzajúcej hry
- Učenie sa prispôsobovaním parametrov
 - $v_1 a_1 + v_2 a_2 + \dots + v_{16} a_{16}$
 - hra dvoch programov proti sebe
- Učenie sa s makro-operátormi

Učiaci sa znalostný agent

- výkonná časť
 - výkonný prvak
 - snímače
 - efektory
- učiaca sa časť
 - učiaci sa prvak
 - hodnotiteľ konania
 - generátor problémov

- učenie sa s učiteľom (supervised learning, s dohľadom)
 - okamžite sú dostupné vedomosti o vstupoch aj výstupoch
- učenie sa s odmenou a trestom (reinforcement learning, s posilňovaním)
 - agent dostane informáciu o hodnotení jeho akcie, ale nie o tom, aká je správna akcia
- učenie sa bez učiteľa (unsupervised learning, bez dohľadu)
 - agent nedostáva nijakú informáciu o tom, aké by mali byť správne akcie

Induktívne učenie sa

```
global priklady ← {}  
function VÝKONNÝ-PRVOK-S-ODRAZOM(vnem) returns akcia  
    if (vnem, a) in priklady then return a  
    else  
        h ← INDUKUJ(priklady)  
        return h(vnem)
```

```
function UČIACI-SA-PRVOK-S-ODRAZOM(vnem, akcia)  
    inputs: vnem, spätnoväzobný vnem  
            akcia, spätnoväzobná akcia  
    priklady ← priklady ∪ {(vnem, akcia)}
```

Učenie sa s učiteľom

Dané: Trénovacie príklady $(\mathbf{x}; f(\mathbf{x}))$, t.j. trénovacia množina

$$\{(\mathbf{x}_1, \mathbf{f}(\mathbf{x}_1)), (\mathbf{x}_2, \mathbf{f}(\mathbf{x}_2)), \dots, (\mathbf{x}_P, \mathbf{f}(\mathbf{x}_P))\}$$

pre nejakú neznámu funkciu f

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

Nájdi:

$$\mathbf{f}(\mathbf{x})$$

dobré priblíženie (aproximáciu) funkcie f , t.j. predpovedaj

$$\mathbf{y}' = \mathbf{f}(\mathbf{x}')$$

kde \mathbf{x}' nie je v trénovacej množine

Príklady použitia

- **rozpoznanie rukopisu**
 - x : údaje o pohybe pera
 - $f(x)$: písmeno abecedy
- **diagnóza choroby**
 - x : vlastnosti pacienta (symptómy, výsledky laboratórnych testov)
 - $f(x)$: choroba (alebo možno dokonca odporúčaná liečba)
- **opoznanie osoby**
 - x : bitmapový obraz tváre osoby
 - $f(x)$: meno osoby
- **zistenie spamu**
 - x : e-správa
 - $f(x)$: spam or nie-spam.

Učenie sa s učiteľom – kedy je vhodné použiť?

- **Situácie, keď nemáme ľudského experta**
- **Situácie, keď ľudia vedia riešiť problém, ale nevedia opísat', ako to robia**
 - x : bitmapový obraz znaku napísaného rukou
 - $f(x)$: ascii kód toho znaku
- **Situácie, keď sa požadovaná funkcia často mení**
 - x : opis cien akcií a obchodov za posledných 10 dní
 - $f(x)$: odporúčané obchody
- **Situácie, keď každý zákazník potrebuje personalizovanú funkciu f**
 - x : prichádzajúca e-správa
 - $f(x)$: dôležitosť správy (skôr udávajúce naliehavosť, s akou správu zákazníkovi prezentovať alebo naopak zrušiť bez prezentovania)

klasifikovanie

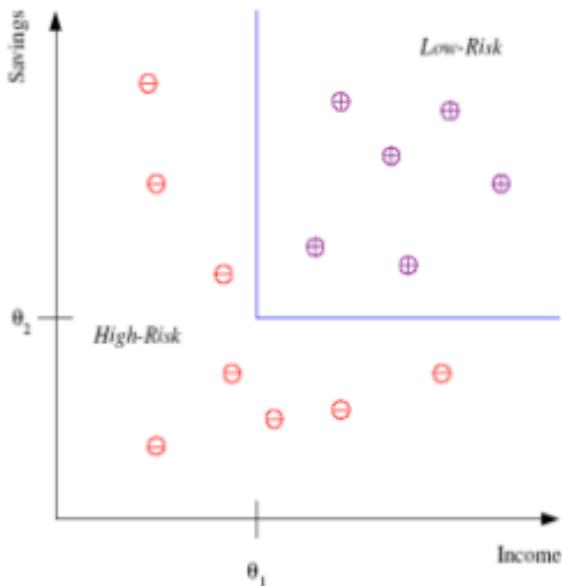
- príklad: hodnotenie úverov
- Rozlišovanie medzi **nízko-rizikovými** a **vysoko-rizikovými** zákazníkmi na základe ich príjmu a úspor

Diskriminant:

IF $\text{príjem} > \theta_1 \text{ A } \text{úspory} > \theta_2$

THEN **nízke-riziko**

ELSE **vysoké-riziko**



opoznanie osoby (tváre)

Trénovacie príklady obrazu tváre osoby



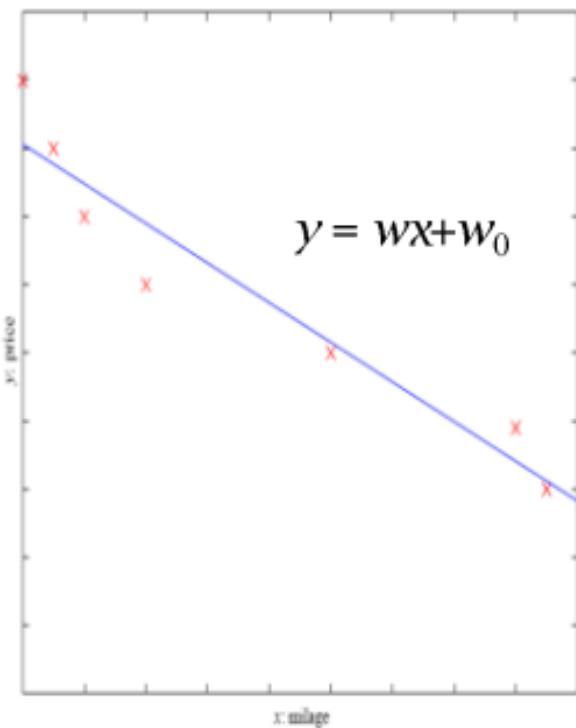
Testovacie obrazy



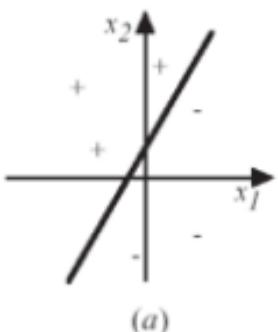
AT&T Laboratories, Cambridge UK
<http://www.uk.research.att.com/facedatabase.html>

Regresia

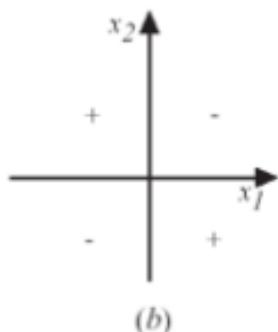
- príklad: cena jazdeného auta
 - x : atribúty auta
 y : cena
- $$y = g(x | \theta)$$
- $g(\cdot)$ model,
 θ parametre



Lineárna regresia



lineárne separovateľný



nie je lineárne separovateľný

Rozhodovacie stromy

Učenie sa pomocou rozhodovacích stromov je všeobecne najvhodnejšia metóda na riešenie problémov s nasledujúcou charakteristikou:

- Inštancie sú reprezentované dvojicou atribút - hodnota.
- Cieľová funkcia má diskrétnu výstupnú hodnotu.
- Problémy vyžadujúce disjunktívny opis
- Rozhodovacie stromy prirodzene reprezentujú disjunktívne výrazy.
- Údaje na trénovanie obsahujú chyby
- Tréningové údaje môžu obsahovať chýbajúce hodnoty atribútov

Učiace funkcie sú buď reprezentované rozhodovacími stromami alebo reprezentované množinou if - then pravidiel na zlepšenie čitateľnosti.

Príklad - Rozhodovacie stromy

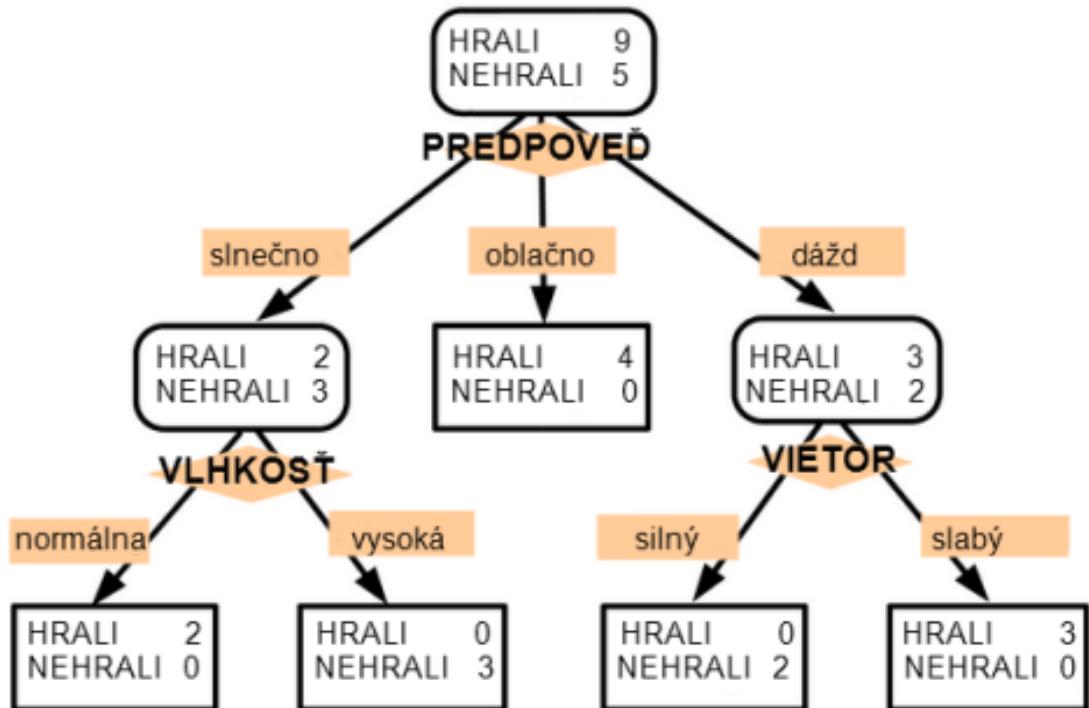
Dávid je manažérom golfového klubu. Sú dni, keď chce hrať golf každý a personál ihriska je preťažený, inokedy však golf nehrá nik a personál klubu má príliš veľa voľného času.

Dávidovým cieľom je optimalizovať dostupnosť personálu podľa predpovede, koľko ľudí bude v ten ktorý deň hrať golf. Počas 14 dní si zaznamenal, aké bolo počasie, vlhkosť vzduchu, rýchlosť vetra a či ľudia hrali alebo nehrali golf.

Príklad - Rozhodovacie stromy

DEŇ	PREDPOVEĎ	VLHKOSŤ	VIETOR	HRALI
1	Slnečno	Vysoká	Slabý	Nie
2	Slnečno	Vysoká	Silný	Nie
3	Oblačno	Vysoká	Slabý	Áno
4	Dážď	Vysoká	Slabý	Áno
5	Dážď	Normálna	Slabý	Áno
6	Dážď	Normálna	Silný	Nie
7	Oblačno	Normálna	Silný	Áno
8	Slnečno	Vysoká	Slabý	Nie
9	Slnečno	Normálna	Slabý	Áno
10	Dážď	Normálna	Slabý	Áno
11	Slnečno	Normálna	Silný	Áno
12	Oblačno	Vysoká	Silný	Áno
13	Oblačno	Normálna	Slabý	Áno
14	Dážď	Vysoká	Silný	Nie

Príklad - Rozhodovacie stromy



Príklad - Rozhodovacie stromy

IF (Predpoved = slnečno) \wedge (Vlhkosť = normálna)

THEN Hrali = **ÁNO**

IF (Predpoved = slnečno) \wedge (Vlhkosť = vysoká)

THEN Hrali = **NIE**

IF (Predpoved = oblačno) **THEN** hrali = **ÁNO**

IF (Predpoved = dážď) \wedge (Vietor = silný)

THEN Hrali = **NIE**

....

Induktívne vytváranie rozhodovacích stromov

```
function UČENIE-SA-ROZHODOVACIEHO-STROMU(priklady, atribúty, štandard)
    returns rozhodovací strom
    inputs: priklady, množina príkladov
            atribúty, množina atribútov
            štandard, štandardná hodnota cieľovej hypotézy

    if priklady sú prázdne then return štandard
    else if všetky priklady majú rovnakú klasifikáciu then return klasifikácia
    else if atribúty sú prázdne then return HODNOTA-VÄČŠINY(priklady)
    else
        najlepší ← VYBER-ATRIBÚT(atribúty, priklady)
        strom ← nový rozhodovací strom s koreňom s testom na najlepší
        for each hodnotu  $v_i$  atribútu najlepší do
            priklady ← {prvky z priklady s hodnotou atribútu najlepší =  $v_i$ }
            podstrom ← UČENIE-SA-ROZHODOVACIEHO-STROMU
                (priklady, atribúty - najlepší, HODNOTA-VÄČŠINY(štandard))
            pridaj hranu do stromu s ohodnením  $v_i$  a podstromom podstrom
        end
    return strom
```

Posudzovanie kvality algoritmu učenia sa

- klasifikácia príkladu hypotézou
- testovacia množina
 1. Nazhromaždiť veľkú množinu príkladov.
 2. Rozdeliť ich do dvoch disjunktných množín: trénovacej a testovacej.
 3. Použiť učiaci sa algoritmus s trénovacou množinou ako množinou príkladov na vytvorenie hypotézy H .
 4. Zmerať podiel príkladov v testovacej množine, ktoré sa správne klasifikovali podľa hypotézy H .
 5. Opakovať kroky 1 až 4 pre rôzne veľkosti trénovacích množín a pre rôzne náhodne zvolené testovacie množiny pre každú veľkosť.

Učenie sa všeobecných logických opisov

- priestor hypotéz

$$H_1 \vee H_2 \vee \dots \vee H_n$$

- zlučiteľnosť hypotézy

- falošný negatívny príklad
 - falošný pozitívny príklad

Učenie sa hľadaním pomocou najlepšej súčasnej hypotézy

```
function UČENIE-SA-SÚČASNEJ-NAJLEPŠEJ-HYPOTÉZY(priklady)
    returns hypotéza
    inputs: priklady, množina príkladov

    H ← ľubovoľná hypotéza zlučiteľná s prvým príkladom v priklady
    for each zvyšujúci príklad p v priklady do
        if p je falošný pozitívny pre H then
            H ← choose špecializácia hypotézy H zlučiteľná s priklady
        else if p je falošný negatívny pre H then
            H ← choose zovšeobecnenie hypotézy H zlučiteľné s priklady
        if nedá sa nájsť zlučiteľné zovšeobecnenie ani špecializácia then fail
    end
    return H
```

Učenie sa hľadaním v priestore verzií

```
function UČENIE-SA-V-PRIESTORE-VERZIÍ(priklady)
    returns priestor verzií
    inputs:          priklady, množina príkladov
    local variables: V, priestor verzií: množina všetkých hypotéz

    V ← množina všetkých hypotéz
    for each príklad p v priklady do
        if V nie je prázdny then
            V ← OBNOVA-PRIESTORU-VERZIÍ(V, p)
        end
    return V
```

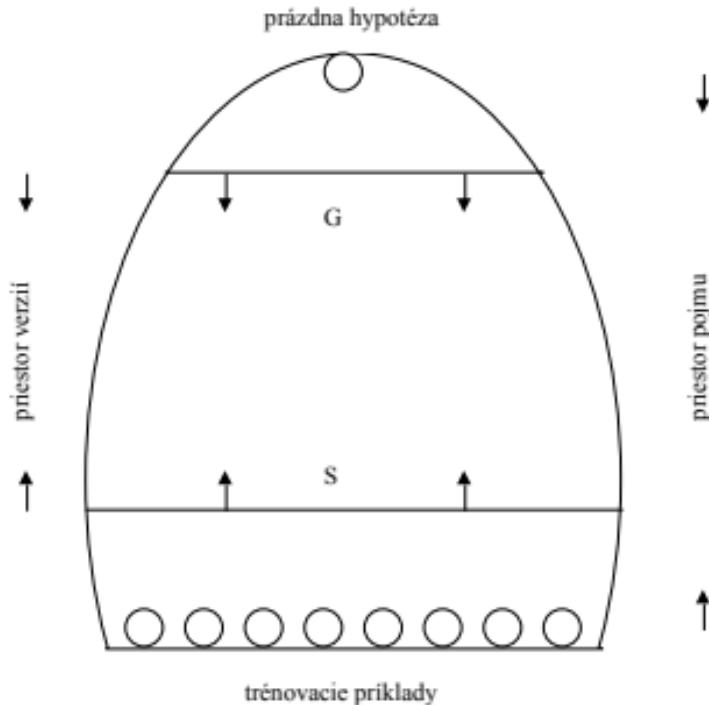
```
function OBNOVA-PRIESTORU-VERZIÍ(V, p)
    returns obnovený priestor verzií

    V ← {h ∈ V : h je zlučiteľná s p}
```

Učenie sa hľadaním v priestore verzií

- priestor verzií
 - G-množina, S-množina
- obnova priestoru verzií
 - Falošný pozitívny príklad (FPP) pre S_i ; S_i je príliš všeobecná, vylúčime ju z S -množiny
 - FNP pre S_i ; S_i je príliš zvláštna, nahradíme ju všetkými jej možnými bezprostrednými zovšeobecneniami
 - FPP pre G_i ; G_i je príliš všeobecná, nahradíme ju všetkými jej možnými bezprostrednými špecializáciami
 - FNP pre G_i ; G_i je príliš zvláštna, vylúčime ju z G -množiny

Priestor pojmu a priestor verzíí



Algoritmus odstraňovania kandidátov

Výstupom je opis pojmu, ktorý je zlučiteľný so všetkými pozitívnymi príkladmi a s nijakým negatívnym príkladom.

1. Inicializuj G tak, aby obsahovala práve jeden prvok: prázdny opis
2. Inicializuj S tak, aby obsahovala práve jeden prvok: prvý pozitívny príklad.
3. Vezmi ďalší trénovací príklad.

Ak je to pozitívny príklad,

tak zovšeobecni prvky v S čo najmenej tak, aby boli zlučiteľné s novým trénovacím príkladom.

Ak je to negatívny príklad,

tak špecializuj prvky v G čo najmenej tak, aby nový trénovací príklad už neboli zlučiteľní s hocjakým prvkom v G

4. **Ak** sú S aj G obe jednoprvkové množiny,

tak ak sú rovnaké, **tak** vráť ten prvok a zastav,

inak oznám, že množina trénovacích príkladov bola nezlučiteľná a zastav,

inak pokračuj krokom 3.

Príklad – Algoritmus odstraňovania kandidátov

Uvažujme problémovú oblasť historických stavebných pamiatok.

Predpokladajme ďalej, že v každej položke sa môžu vyskytovať len tieto hodnoty:

krajina $\in \{$ Slovensko, Francúzsko, Škótsko, Česko $\}$

druh $\in \{$ kostol, hrad, zámok, palác $\}$

sloh $\in \{$ románsky, gotika, renesancia, barok, neoklasicizmus $\}$

stav $\in \{$ zachovalý, poškodený, zrúcanina $\}$

Ako prebehne učenie sa pojmu „slovenská gotická pamiatka“ ??

pamiatka_slovenská_gotická

krajina : Slovensko

druh : x1

sloh : gotika

stav : x2

Príklad – Algoritmus odstraňovania kandidátov

Predpokladajme, že v trénovacej množine budú tieto príklady:

(1:pozitívny)

krajina: Slovensko
druh: hrad
sloh: gotika
stav: poškodený

(2:negatívny)

krajina: Slovensko
druh: kostol
sloh: renesancia
stav: zachovalý

(3:pozitívny)

krajina: Slovensko
druh: kostol
sloh: gotika
stav: poškodený

(4:negatívny)

krajina: Francúzsko
druh: kostol
sloh: gotika
stav: zachovalý

(5:pozitívny)

krajina: Slovensko
druh: kostol
sloh: gotika
stav: zachovalý

Príklad – Algoritmus odstraňovania kandidátov

- Na začiatku budú G aj S jednoprvkové množiny opisov
$$G = \{(x_1, x_2, x_3, x_4)\}$$
$$S = \{(Slovensko, hrad, gotika, poškodený)\}$$
- Druhý príklad je negatívny. Množinu G treba špecializovať tak, aby tento negatívny príklad už nepatril do priestoru verzií (premenná sa nahradí konštantou).
$$G = \{(x_1, hrad, x_3, x_4), (x_1, x_2, gotika, x_4), (x_1, x_2, x_3, poškodený)\}$$
Množina S sa spracovaním druhého príkladu nezmení.
- Tretí príklad je opäť pozitívny. Najprv sa z G odstránia všetky opisy, ktoré sú s týmto príkladom nezlučiteľné. S sa zovšeobecní tak, aby zahŕňala aj tento príklad (nahradením konštanty premennou).
$$G = \{(x_1, x_2, gotika, x_4), (x_1, x_2, x_3, poškodený)\}$$
$$S = \{(Slovensko, x_2, gotika, poškodený)\}$$

Príklad – Algoritmus odstraňovania kandidátov

- Ďalší príklad je negatívny. Je to pamiatka, ktorá leží vo Francúzsku. Množina S sa nezmení. Množinu G treba špecializovať, aby nepokryla tento príklad:

$$G = \{(Slovensko, x2, gotika, x4), (Slovensko, x2, x3, poškodený)\}$$

$$S = \{(Slovensko, x2, gotika, poškodený)\}$$

- Posledný príklad je pozitívny. Aby s ním zostala množina G zlučiteľná, treba z nej odstrániť druhý prvok a množinu S zovšeobecniť tak, aby zahŕňala aj nový príklad :

$$G = \{(Slovensko, x2, gotika, x4)\}$$

$$S = \{(Slovensko, x2, gotika, x4)\}$$

- Teraz sú obe množiny S aj G jednoprvkové. Sú rovnaké, takže sa podarilo nájsť opis pojmu, ktorý bolo cieľom naučiť sa.

Príklad – Učenie sa pre hľadanie na webe



Príklad – Učenie sa pre hľadanie na webe

- Ako získať dátá pre zdokonalenie vyhodnocovacej funkcie ?
 - Explicitná vs. implicitná spätná väzba
 - Absolútna vs. relatívna spätná väzba
 - Štúdia sledovania pohybu očí

Príklad – Učenie sa pre hľadanie na webe

Explicitná spätná väzba

- Obťažuje používateľov
- Nízka odozva
- > málo reprezentujúca



Implicitná spätná väzba

- Dotazy, kliknutia, čas strávený na stránke, skrolovanie, atď
- Neobťažuje používateľov
- Horšia interpretácia

Príklad – Učenie sa pre hľadanie na webe

Relatívna spätná väzba:

Kliknutie znamená uprednostnenie odkazu pred ostatnými zobrazenými.

Absolútна spätná väzba:

Odkazy, na ktoré používateľ klikol sú relevantné k vyhľadávaciemu dopytu.

($3 < 2$),
($7 < 2$),
($7 < 4$),
($7 < 5$),
($7 < 6$)



1. Kernel Machines
<http://svm.first.gmd.de/>
2. Support Vector Machine
<http://jboltvar.freeservers.com/>
3. SVM-Light Support Vector Machine
http://ais.gmd.de/~thorsten/svm_light/
4. An Introduction to Support Vector Machines
<http://www.support-vector.net/>
5. Support Vector Machine and Kernel ... References
<http://svm.research.bell-labs.com/SVMrefs.html>
6. Archives of SUPPORT-VECTOR-MACHINES ...
<http://www.jiscmail.ac.uk/lists/SUPPORT...>
7. Lucent Technologies: SVM demo applet
<http://svm.research.bell-labs.com/SVT/SVMsvt.html>
8. Royal Holloway Support Vector Machine
<http://svm.dcs.rhbnc.ac.uk>

Je implicitná spätná väzba spoľahlivá ?

Podľa čoho sa používateľ rozhoduje, kde kliknúť ?

- Koľko výsledkov si používateľ prezrie predtým, ako na nejaký klikne ?
- Prezerá používateľ výsledky vyhľadávania odhora nadol?
- Prezrie si používateľ všetky výsledky vyhľadávania nad odkazom, na ktorý klikol ?
- Sú pre používateľa zaujímavé aj výsledky vyhľadávania zobrazené pod odkazom, na ktorý klikol ?

Ako súvisí kliknutie na odkaz s jeho relevantnosťou ?

- Absolútна spätná väzba:

Sú odkazy, na ktoré používateľ klikol relevantné?

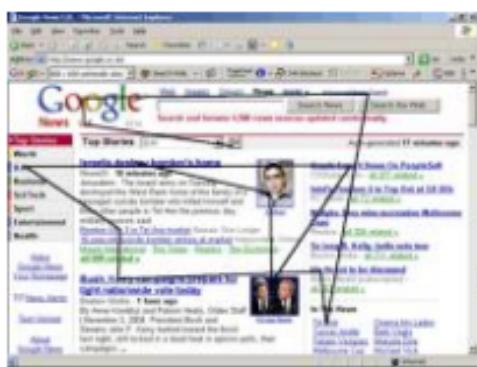
Sú odkazy, na ktoré používateľ neklikol nerelevantné ?

- Relatívna spätná väzba:

Sú odkazy na ktoré používateľ klikol viac relevantné ako odkazy, na ktoré používateľ neklikol ?

1. Kernel Machines
<http://www.kernel-machines.org/>
2. Support Vector Machine
<http://jbolivar.freeservers.com/>
3. SVM-Light Support Vector Machine
http://ais.gmd.de/~thorsten/svm_light/
4. An Introduction to SVMs
<http://www.support-vector.net/>
5. Support Vector Machine and ...
<http://svm.bell-labs.com/SVMrefs.html>
6. Archives of SUPPORT-VECTOR...
<http://www.jisic.ac.mk/lists/SUPPORT...>
7. Lucent Technologies: SVM demo applet
<http://svm.bell-labs.com/SVMsvt.html>
8. Royal Holloway SVM
<http://svm.dcs.rhbuc.ac.uk>
9. SVM World
<http://www.svmworld.com>
10. Fraunhofer FIRST SVM page
<http://svm.first.gmd.de>

Príklad – Učenie sa pre hľadanie na webe



Štúdia sledovania pohybu očí

- Fixácia:
~200 – 300ms, získanie informácie
- Kmitanie oka:
extrémne rýchle pohyby medzi jednotlivými fixáciami
- Rozšírenie zreničiek:
veľkosť zreničky súvisí so záujmom používateľa o prezeranú tému

Príklad – Učenie sa pre hľadanie na webe

- Koľko výsledkov vyhľadávania používateľ prezrie?



Príklad – Učenie sa pre hľadanie na webe

- Používatelia najčastejšie prezrú 2 výsledky vyhľadávania.
- Používatelia zvyčajne prezerajú výsledky vyhľadávania odhora nadol.
- Používatelia najčastejšie kliknú na prvý odkaz na stránke s výsledkami vyhľadávania.
- Používatelia sa zvyčajne nepozerajú na odkazy, ktoré sú zobrazené nižšie ako ten, na ktorý klikli.

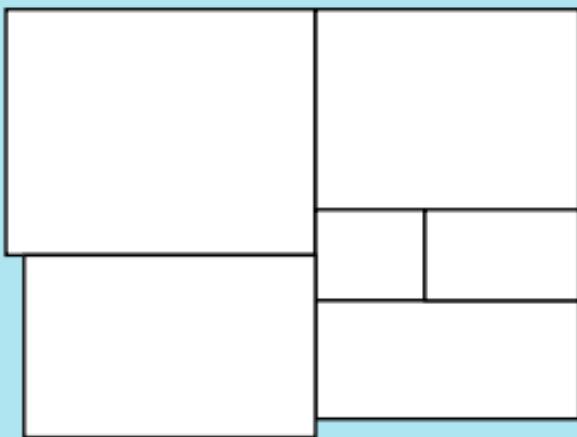
Spĺňanie ohraničení

Farbenie mapy

- Úloha:
 - Majme mapu krajín, priradťte každej krajine farbu, tak že:
 - AK majú dve krajiny spoločnú hranicu,
 - TAK musia mať krajiny rôznu farbu
- Je možné použiť iba obmedzené množstvo farieb, snahou je použiť pre úspešné vyriešenie problému minimálne množstvo farieb

Príklad - Farbenie mapy

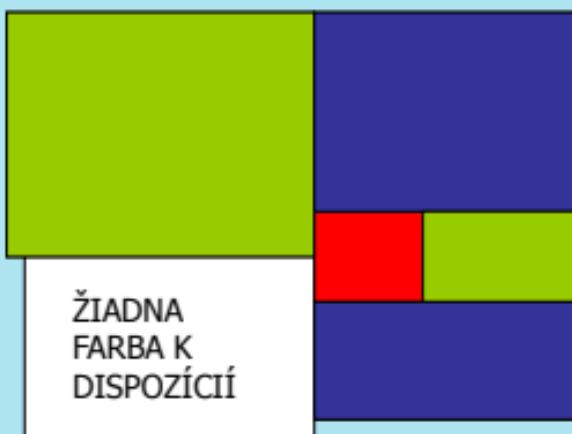
- Pre zjednodušenie uvažujme, že každá krajina má tvar štvoruholníka:



- Koľko farieb je potrebných pre úspešné vyriešenie problému ?

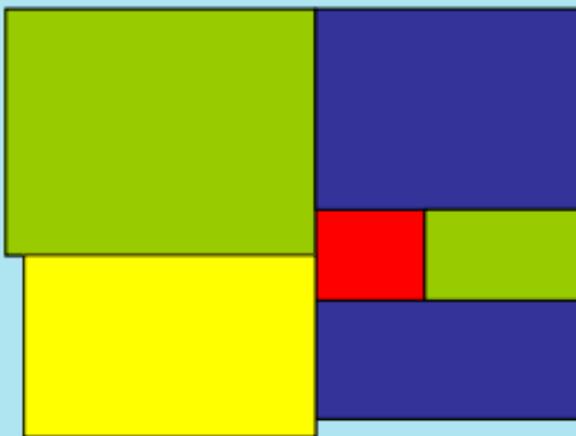
Príklad – Farbenie mapy: 3 farby ?

- Pokúsme sa použiť pre vyriešenie problému 3 farby: červenú, modrú a zelenú



- Môžeme si byť istí, že pri použití troch farieb sa problém nedá vyriešiť ?
- Nemôžeme, je potrebné použiť jednu z úplných metód prehľadávania!

Príklad – Farbenie mapy: 4 farby



- PRI POUŽITÍ
ĎALŠEJ, 4. FARBY
SA DÁ PROBLÉM
VYRIEŠIŤ

Motivácia

- Farbenie mapy je špecifickým problémom
 - Prečo sa ním teda zaoberať ?
- Farbenie mapy je typickým príkladom typu problému s ohraničeniami (problém spĺňania ohraničení: CSP - Constraint Satisfaction Problem)
- CSP sa vyskytujú v mnohých oblastiach
 - Plánovanie
 - Cestovné poriadky
 - Optimalizačné problémy v oblasti priemyslu

príklad: hlavolam sudoku

- <http://www.websudoku.com/>
- každé Sudoku má jediné riešenie dosiahnuteľné len logickým uvažovaním bez hádania.
- Prázdne miesta treba vyplniť číslami 1 až 9.
- V každom riadku musí byť každé číslo a práve raz.
- V každom stĺpci tiež.
- V každom štvorci 3×3 tiež.

príklad: hlavolam sudoku

				9		3		
	2		6		1	5	9	
3			5		8		2	4
	9					4	8	
6			2		9			1
	5	2					3	
9	7		1		5			3
	3	8	7		4		1	
		5		8				

“Programovanie ohraničení”

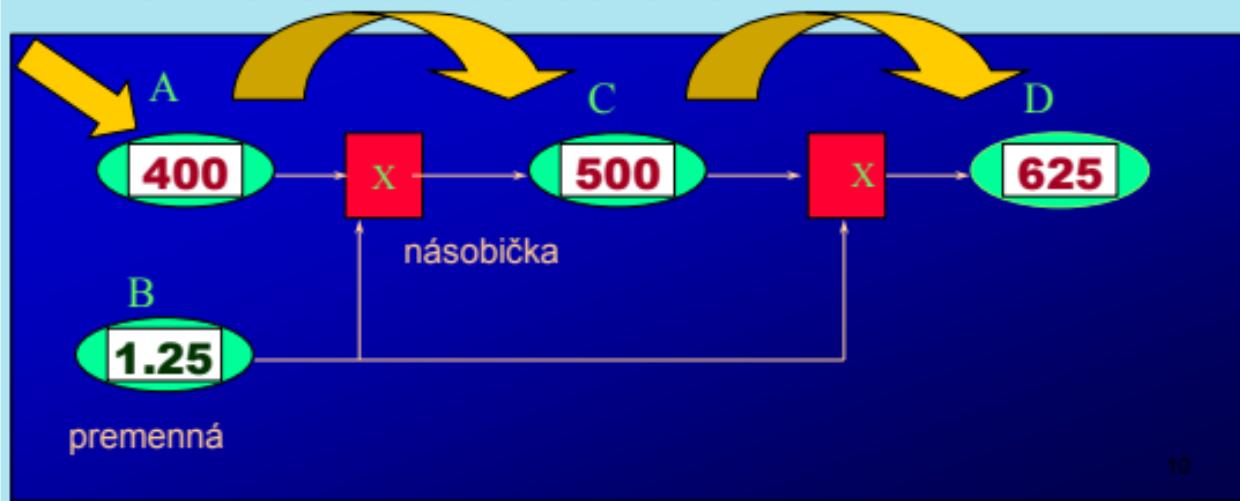
- Odlišná programovacia paradigma
 - “stačí stanoviť”, čo je potrebné vyriešiť a systém sám rozhodne, ako daný problém vyriešiť”
 - Kód programu na vyriešenie Sudoku hlavolamu môže mať iba 20 riadkov
 - Napr. ohraničenie pre riadok, že všetky čísla v riadku sú rôzne
 - prevšetky(j in 1..9) všetky \neq zne(prevšetky(i in 1..9) pos[i,j]);
- Veľmi odlišné od tradičných paradigm:
 - Procedurálne programovanie (Fortran, Pascal, C)
 - Objektovo-orientované programovanie (C++, Java, C#)
 - Funkcionálne programovanie (Lisp, ML, Haskell)

príklad: sústava číselných ohraničení

nech je daná množina rovníc:

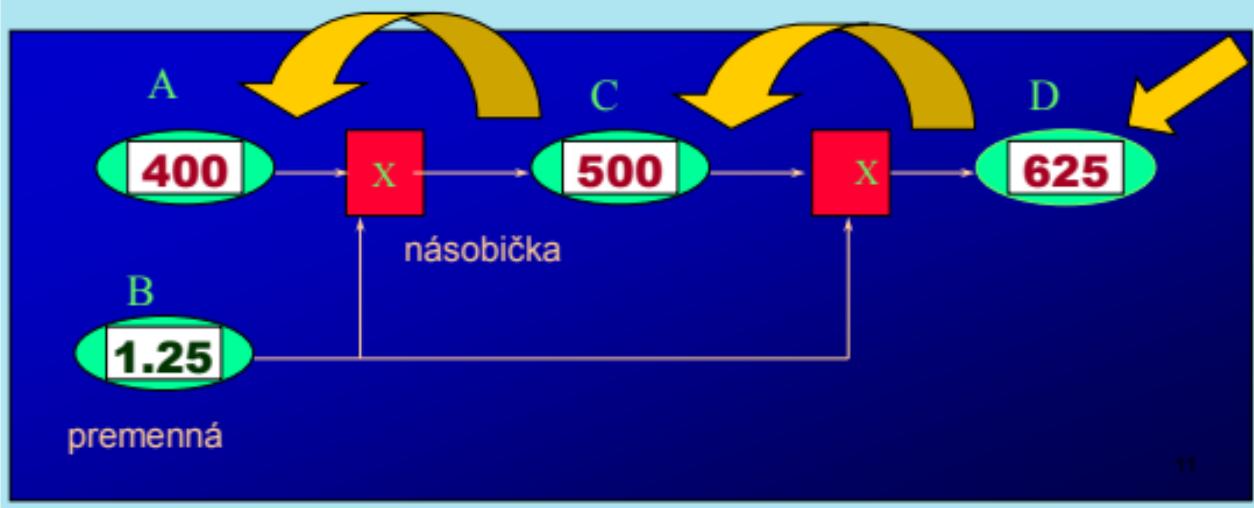
$$\begin{cases} C = A * B \\ D = C * B \\ B = 1.25 \end{cases}$$

možno s ňou združiť sieť ohraničení:



príklad: sústava číselných ohraničení 2

možno s ňou združiť siet' ohraničení:



príklad: kalkulačná tabuľka (spreadsheet)

	A	B	C	D
1	pomer X	1.2		
2	pomer Y	1.1		
4		1. rok	2. rok	3. rok
5	príjem X	3000	= B1 * B5	= B1 * C5
6	príjem Y	5000	= B2 * B6	= B2 * C6
7	výdaje	9000	= B7	= C7
8	celkovo	= B5+B6 -B7	= C5+C6 - C7	=D5+D6 -D7

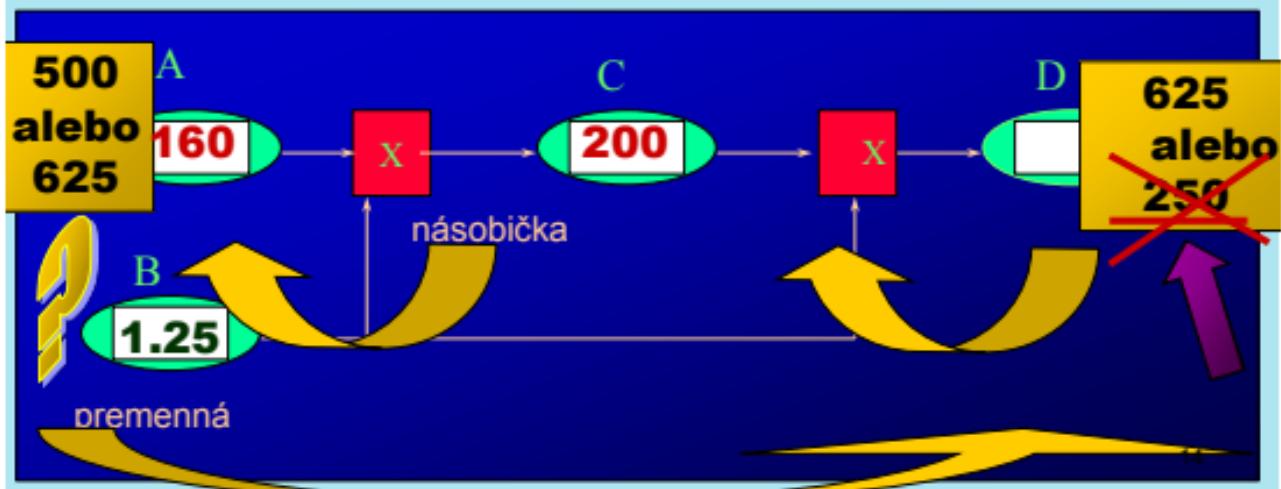
modus:čo ak

príklad kalkulačná tabuľka (spreadsheet) 2

	A	B	C	D
1	pomer X	1.2		
2	pomer Y	1.1		
4		1. rok	2. rok	3. rok
5	príjem X	3000	=B13600	=B43205
6	príjem Y	5000	=B5500	=B60506
7	výdaje	9000	=9000	9000
8	celkovo	=B1000 -B7	=C51006 - C7	=D51370 -D7

príklad: sústava číselných ohraničení 3

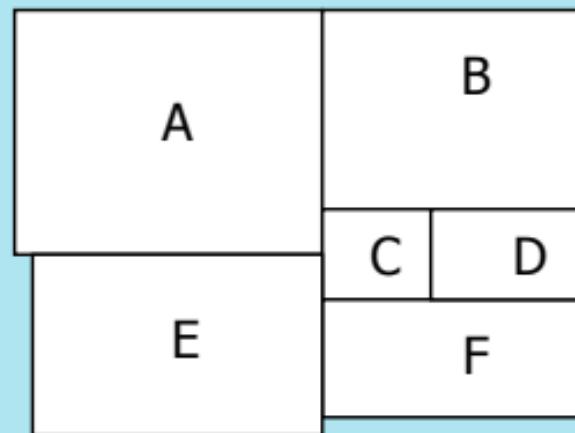
možno s ňou združiť sieť ohraničení:



Od máp k spĺňaniu ohraničení

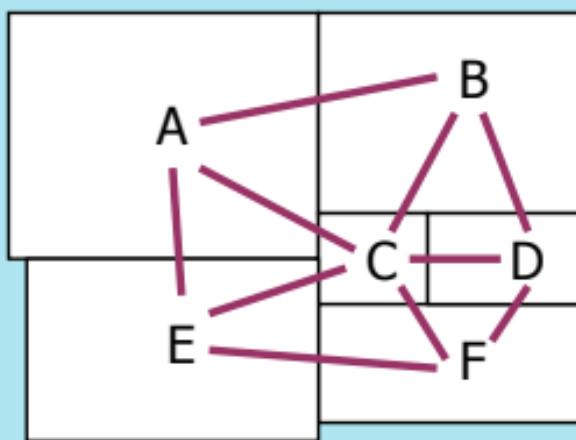
- Upravme problém farbenia máp podľa všeobecných predstáv používaných pri riešení problémov s ohraničeniami:
 - Priradené hodnoty musia spĺňať niektoré z ohraničení

Farbenie mapy – priradenie hodnôt



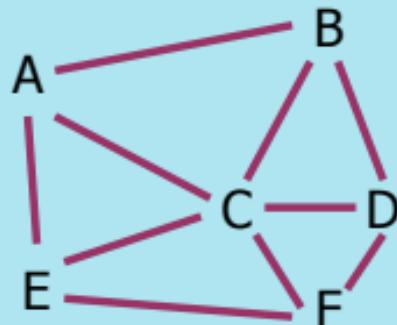
Farbenie mapy - ohraničenia

- Grafy opisujú problém všeobecnejšie ako mapy – konvertujme teda mapu na graf
- Hrana vyjadruje skutočnosť, že krajiny majú spoločnú hranicu, čiže musia byť zafarbené odlišnou farbou



Farbenie grafu 3 farbami

- Každému uzlu musí byť priradená hodnota z množiny farieb { r, g , b }
- Napr.: B := b
- Každej hrane medzi uzlami je možné priradiť iba dvojicu z množiny { (r,g), (r,b), (g, r), (g, b), (b, r), (b, g) }



Reprezentácia problému pomocou grafu

- Zovšeobecníme teda problém farbenia troma farbami na problém s ohraničeniami (CSP)
- Každému uzlu musí byť priradená hodnota z pevne danej množiny prípustných hodnôt zvyčajne nazývanej "doména" D
- Každej hrane medzi dvoma uzlami je možné priradiť iba prípustnú dvojicu z množiny prípustných dvojíc hodnôt
 - Hrana sa nazýva ohraničenie
 - Taktiež ide o synonymum pre množinu prípustných dvojíc

Domény CSP

- Domény je možné považovať za
 - diskrétné konečné
 - zvláštny prípad: boľovské hodnoty
 - diskrétné nekonečné
 - spojité

Ohraničenia: Explicitné vs. implicitné

- Ohraničenie sa vyjadruje (predpokladajme, že x a y majú doménu $D=\{u,v,w\}$)
 - explicitne: množina prípustných dvojíc hodnôt
napr $(x,y) \in \{ (u,u), (u,v), (w,u), (v,w) \}$
 - implicitne: napr.:
 $y \neq w$
ale aj v matematike napr.
 $x + y = 2$
 $x + 2 < y$

Ohraničenia: n-miestne

- 1-miestne (unárne)
 - $x \neq R$
- 2-miestne (binárne)
 - $z_i < z_j \wedge z_i + i \neq z_j - j$
problém výlučne s binárnymi ohraničeniami sa dá reprezentovať pomocou grafu.
Hrana medzi dvoma uzlami reprezentuje ohraničenie
- n-miestne, $n \geq 3$
 - všetky rozdielne (x_1, \dots, x_n)
n-miestne ohraničenie sa dá previesť na množinu 2-miestnych ohraničení
 $\{x_1 \neq x_2, x_1 \neq x_3, \dots\}$

Diskrétna konečná doména

- Predpokladajme, že máme n uzlov a každému uzlu je priradená hodnota z množiny D a že veľkosť množiny D je d
- Počet možných priradení je potom:

$$d * d * d * \dots * d = d^n$$

- Exponenciálna veľkosť problému $O(d^n)$

Boolovská doména

- Predpokladajme, že máme n uzlov a každému uzlu je priradená hodnota z množiny $\{T, F\}$
- Počet možných priradení je potom 2^n

príklad problému:

- splniteľnosť formuly v konjuktívnom normálnom tvare, kde každá klauzula má najviac 3 literály – 3SAT
- $(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge (x_{31} \vee x_{32} \vee x_{33}) \wedge \dots$
- kde x_{ij} je boolovská premenná alebo negácia premennej a každá premenná sa môže vyskytovať viackrát vo výraze
- splniteľnosť boolovskej formuly – či existuje priradenie boolovských hodnôt premenným vo formule také, že formula sa vyhodnotí na T

Boolovská doména: 3SAT

- inštancia príkladu problému:
 - $E = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4)$
 - premenné: (x_1, x_2, x_3, x_4)
 - riešenie: napr $\{x_1 = T, x_2 = T, x_3 = T, x_4 = T\}$
 - existuje mnoho riešení, napr všetky priradenia s $x_1 = T$

diskrétna nekonečná doména

- diskrétnie premenné
 - nekonečné domény (celé čísla, reťazce, atď.)
 - napr. rozvrhovanie úloh: premenné sú začiatočné/koncové dni pre každú úlohu
 - potreba jazyka na vyjadrenie ohraničení, napr. $StartJob_1 + 5 \leq StartJob_3$.
 - nekonečne veľa riešení
 - lineárne ohraničenia: riešiteľné
 - nelineárne: neexistuje vešobecný algoritmus

spojitá doména

- spojité premenné
 - napr. zostavenie letového poriadku alebo rozvrhu fakulty.
 - lineárne ohraničenia sú riešiteľné v polynomiálnom čase metódami lineárneho programovania.

jemné ohraničenia

- preferencie (jemné ohraničenia)
 - napr. *modrá* je lepšia než *červená* sa často dajú reprezentovať cenou pre každé priradenie premennej
 - kombinácia optimalizácie s CSP

Riešenie problému spĺňania ohraničení

- Domény pokladajme za konečné
- Predpokladajme, že máme n uzlov a každému uzlu je priradená hodnota z množiny D a že veľkosť množiny D je d
- Počet možných priradení je potom:

$$d * d * d * \dots * d = d^n$$

- Exponenciálna veľkosť problému
- pre domény veľkosti $d \Rightarrow O(d^n)$ úplných priradení.

formulovanie problému pomocou stavového priestoru

- Čo je stavový priestor?
- Čo je začiatočný stav?
- Aká je množina operátorov (funkcia generujúca nasledovníkov)?
- Aký je cieľový test?
- Aká je cena cesty?

zvláštnosti

- veľký priestor hľadania
- komutatívnosť
- čo tak použiť hľadanie do šírky alebo hĺbky?
 - pevná hĺbka

Čo ešte treba?

- nastačí len funkcia nasledovníka a cieľový test
- ale aj spôsob, ako šíriť ohraničenia na ďalšie kroky, vzniknúvšie v dôsledku nejakého kroku a tiež včasný test zlyhania
- → Explicitné reprezentovanie ohraničení a algoritmy na manipulovanie s ohraničeniami

problém spĺňania ohraničení

- množina premenných $V = \{X_1, X_2, \dots, X_n\}$
- každá premenná X_i má doménu D_i možných hodnôt
 - zvyčajne D_i je diskrétna a konečná
- množina ohraničení $\{C_1, C_2, \dots, C_p\}$
 - každé ohraničenie C_k obsahuje podmnožinu premenných a určuje prípustné kombinácie hodnôt týchto premenných
- cieľ: priradiť hodnotu každej premennej tak, aby boli všetky ohraničenia splnené
 - priradenie je prvok z $D_1 \times D_2 \times \dots \times D_n$

Spĺňanie ohraničení ako problém hľadania riešenia

- Máme množinu n premenných (uzlov) $V = \{X_1, \dots, X_n\}$
- Stavy: platné priradenia
 - Platné priradenie : $\{X_{i1} \leftarrow v_{i1}, \dots, X_{ik} \leftarrow v_{ik}\}, \quad 0 \leq k \leq n,$
také, že hodnoty spĺňajú ohraničenia vzťahujúce sa k uzlom X_{i1}, \dots, X_{ik}
 - Úplné priradenie: $k = n$
- Začiatočný stav: prázdne priradenie $\{\}$, $k = 0$
- Operátory:
 - $\{X_{i1} \leftarrow v_{i1}, \dots, X_{ik} \leftarrow v_{ik}\} \rightarrow \{X_{i1} \leftarrow v_{i1}, \dots, X_{ik} \leftarrow v_{ik}, X_{ik+1} \leftarrow v_{ik+1}\}$
priradenie hodnoty premennej, ktorá ju ešte nemá priradenú a takej, ktorá neprotirečí doteraz priradeným premenným
- Cieľový test: $k = n$
- cena riešenia: konštantná rovnaká cena každého kroku (napr 1). v princípe irrelevantná

Druhy priradení

- **Úplné priradenie:**
 - Všetkým uzlom je priradená hodnota
- **Neúplné priradenie:**
 - Hodnota je priradená všetkým, niektorým alebo žiadnemu uzlu

CSP ako problém hľadania - zhrnutie

- začiatočný stav: prázdne priradenie
- stavy: konzistentné priradenia (úplné aj neúplné)
- funkcia nasledovníka: priradenie hodnoty premennej, ktorá ju ešte nemá priradenú a takej, ktorá neprotirečí doteraz priradeným premenným
- cieľový test: priradenie je úplné (všetky premenné majú priradenú hodnotu)
- cena cesty: irrelevantná
- faktor vetvenia b na vrchnej úrovni je nd .
- $b=(n-l)d$ v hĺbke l , takže je n/d^n listov (ale len d^n úplných priradení).

dá sa aj lepšie.....

problém spĺňania ohraničení

- $(\exists x_1) \dots (\exists x_n) (D_1(x_1) \wedge \dots \wedge D_n(x_n) \Rightarrow C_1(x_1, \dots, x_n) \wedge \dots \wedge C_m(x_1, \dots, x_n))$

riešiť problém splňania ohraničení

- preukázať, či ne/jestuje riešenie
- zistiť počet riešení
- nájsť jedno riešenie
- nájsť všetky riešenia
- nájsť optimálne riešenie (ktoré minimalizuje nejakú oceňovaciu funkciu), ak je taká funkcia definovaná

Definovanie problému s ohraničeniami

◎ problém s ohraničeniami (alebo problém splnenia ohraničení) je:

- konečná množina premenných: z_1, z_2, \dots, z_n
- pre každú premennú s ňou združená konečná množina (doména) jej možných hodnôt
 $d_i = \{a_{i1}, a_{i2}, \dots, a_{in_i}\}$
- pre každú dvojicu premenných $z_i, z_j, i < j$,
ohraničenie $c(z_i, z_j)$

◆ príklad: $z_i < z_j \wedge z_i + i \neq z_j - j$

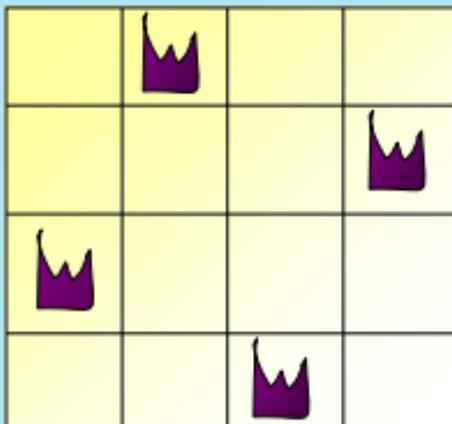
◆ obmedzíme sa na binárne ohraničenia

◎ riešenie: pre každú premennú z_i , hodnota a_{ij} z jej domény d_i taká, že všetky ohraničenia $c(z_i, z_j)$ sú splnené.

príklad: N-dám:

- ◎ dané: šachovnica $N \times N$
- ◎ Problém: nájsť (všetky možné) spôsoby rozmiestnenia N dám na šachovniči tak, že sa žiadne 2 dámy nenapádajú.

◎ 4-dámy:

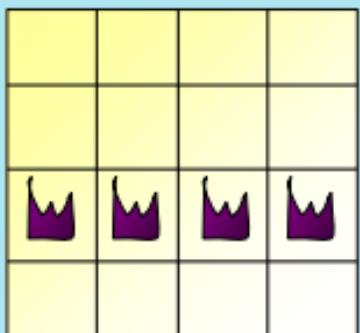
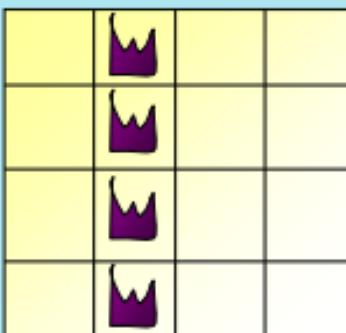
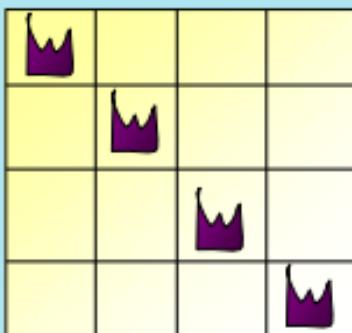


... je riešenie

popletených N-dám:

= rovnaká špecifikácia problému až na to, že ľubovoľné 2 dámy sa **musia** napádať.

◎ p-4-dámy:



ako formulovať ohraničenia?

- ◎ N-dám a p-N-dám sa dajú formulovať ako problémy s ohraničeniami:

→ záleží na voľbe domén a ohraničení

- ◎ jedna možnosť:

→ pre každú dámu: premenná z_i

→ pre každú z_i , doména d_i je množina všetkých možných pozícií na šachovniči:

◆ $d_i = \{(1,1), (1,2), \dots, (1,N), (2,1), (2,2), \dots\}$

→ $c(z_i, z_j) = ria(z_i) \neq ria(z_j)$
 $\wedge stl(z_i) \neq stl(z_j)$
 $\wedge |ria(z_i) - ria(z_j)| \neq |stl(z_i) - stl(z_j)|$

'lepšia' formulácia:

- prijmememe dohodu, že každá dáma bude na zvláštnom riadku:

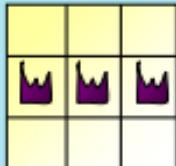
	1	2	3	4
z1				
z2				
z3				
z4				

④ reprezentácia:

- pre dámu na riadku i : premenná z_i
- pre každú z_i , doména $d_i = \{1, 2, \dots, N\}$
- $c(z_i, z_j) = z_i \neq z_j \wedge |z_i - z_j| \neq |i - j|$

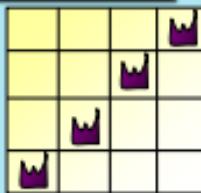
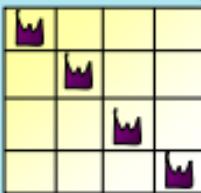
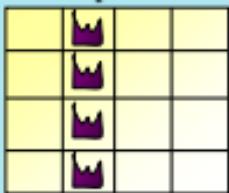
poznámky k p-N-dám:

- druhý spôsob reprezentácie definuje trochu iný problém:



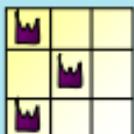
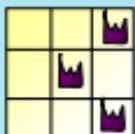
... toto už nie je riešenie

⌚ problém má teraz vždy N+2 riešení !



→ výhoda pre analýzu

⌚ výnimka: N = 3: zložitosti



sú navyše riešenia

príklad: 8-dám

- 64 premenných Z_{ij} , $i = 1$ to 8, $j = 1$ to 8
- doména pre každú premennú $\{0,1\}$
- ohraničenia majú tvar:
- riadky a stĺpce
 - $Z_{ij} = 1 \rightarrow Z_{ik} = 0$ for all $k = 1$ to 8, $k \neq j$
 - $Z_{ij} = 1 \rightarrow Z_{kj} = 0$ for all $k = 1$ to 8, $k \neq i$
- uhlopriečky
 - $Z_{ij} = 1 \rightarrow Z_{i+l, k+l} = 0$ $l = 1$ to 8, $i+l \leq 8$; $k+l \leq 8$ (doprava nahor)
 - $Z_{ij} = 1 \rightarrow Z_{i-l, k+l} = 0$ $l = 1$ to 8, $i-l \geq 1$; $k+l \leq 8$ (doprava nadol)
 - $Z_{ij} = 1 \rightarrow Z_{i-l, k-l} = 0$ $l = 1$ to 8, $i-l \geq 1$; $k-l \geq 1$ (doľava a nadol)
 - $Z_{ij} = 1 \rightarrow Z_{i+l, k-l} = 0$ $l = 1$ to 8, $i+l \leq 8$; $k-l \geq 1$ (doľava a nahor)

príklad: 8-dám

- 8 premenných Z_i , $i = 1$ to 8
- doména pre každú premennú $\{1,2,\dots,8\}$
- ohraničenia majú tvar:
 - $Z_i \neq Z_j$ ak $j \neq i$
 - dámy na tej istej uhlopriečke
 - $Z_i - Z_j \neq i - j$ a
 - $Z_i - Z_j \neq j - i$

krypto-aritmetický hlavolam

$$\begin{array}{r} \text{SEND} & 9567 \\ + \text{MORE} & + 1085 \\ \hline \text{-----} & \text{-----} \\ \text{MONEY} & 10652 \end{array}$$

premenné: S E N D M O R Y

domény:

[1..9] pre S and M

[0..9] pre E N D O R Y

krypto-aritmetický hlavolam ohraničenia 1

- 1 jednoduché ohraničenie

$$1000 S + 100 E + 10 N + D +$$

$$1000 M + 100 O + 10 R + E$$

=

$$10000 M + 1000 O + 100 N + 10 E + Y$$

alebo

- 5 ohraničujúcich rovností používajúcich prenosy - premenné $C_1, \dots, C_4 \in [0..1]$

$$\begin{array}{rl} D + E = 10 C_1 + Y; & \text{SEND} \\ C_1 + N + R = 10 C_2 + E; & +\text{MORE} \\ C_2 + E + O = 10 C_3 + N; & \hline \\ C_3 + S + M = 10 C_4 + O; & \text{MONEY} \\ C_4 = M & \end{array}$$

krypto-aritmetický hlavolam ohraničenia 2

- 28 ohraničujúcich nerovností

- $X \neq Y, X, Y \in \{S, E, N, D, M, O, R, Y\}$

- alebo

- jediné ohraničenie

- všetky rôzne(S, E, N, D, M, O, R, Y)

všetky rôzne(X₁, … , X_n) → kompaktné tvrdenie, že premenné

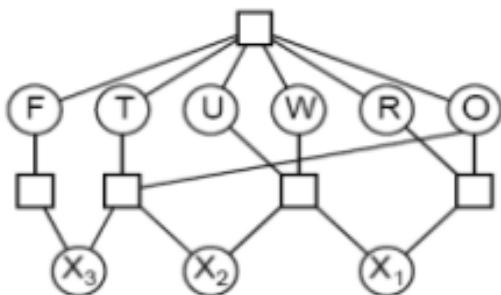
X₁, … , X_n všetky majú priradené navzájom rôzne hodnoty

→ globálne ohraničenie (obsahuje n-árne ohraničenie)

→ zvláštne procedúry na prácu s takým ohraničením

krypto-aritmetický hlavolam príklad

$$\begin{array}{r} \text{T} \ \text{W} \ \text{O} \\ + \ \text{T} \ \text{W} \ \text{O} \\ \hline \text{F} \ \text{O} \ \text{U} \ \text{R} \end{array}$$



Variables: $F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

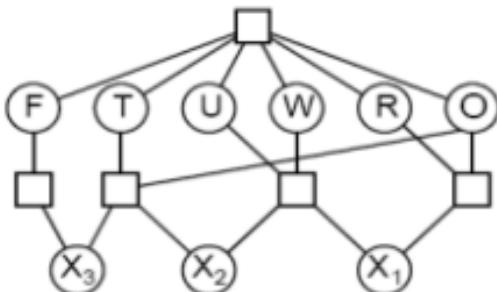
Constraints

$\text{alldiff}(F, T, U, W, R, O)$

$O + O = R + 10 \cdot X_1$, etc.

krypto-aritmetický hlavolam príklad

$$\begin{array}{r} \text{T} \ \text{W} \ \text{O} \\ + \ \text{T} \ \text{W} \ \text{O} \\ \hline \text{F} \ \text{O} \ \text{U} \ \text{R} \end{array}$$



Variables: $F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints

$\text{alldiff}(F, T, U, W, R, O)$

$O + O = R + 10 \cdot X_1$, etc.

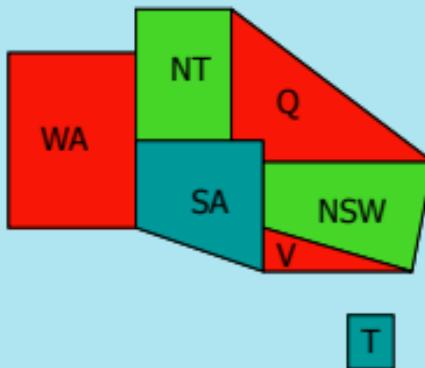
príklad: hlavolam sudoku

- premenné:
- domény:
- ohraničenia:
- cieľ: prirad hodnotu každej premennej tak, aby boli všetky ohraničenia splnené

príklad: hlavolam sudoku

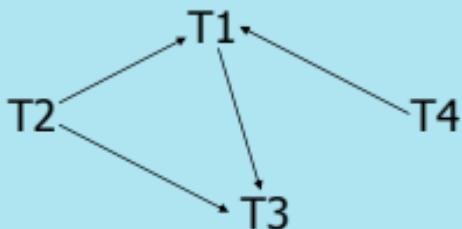
- premenné: X_{11}, \dots, X_{99}
- domény: $\{1, \dots, 9\}$
- ohraničenia:
 - riadkové ohraničenie: $X_{11} \neq X_{12}, \dots, X_{11} \neq X_{19}$
 - stĺpcové ohraničenie: $X_{11} \neq X_{12}, \dots, X_{11} \neq X_{19}$
 - blokové ohraničenie: $X_{11} \neq X_{12}, \dots, X_{11} \neq X_{33}$
- cieľ: prirad hodnotu každej premennej tak, aby boli všetky ohraničenia splnené

príklad: ofarbenie mapy



- 7 premenných $\{WA, NT, SA, Q, NSW, V, T\}$
- všetky premenné majú tú istú doménu $\{\text{červená, zelená, modrá}\}$
 - nijaké 2 susediace premenné nemajú rovnakú hodnotu:
 $WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, SA \neq Q, SA \neq NSW, SA \neq V, Q \neq NSW, NSW \neq V$

príklad: rozvrhovanie úloh



T1 sa musí urobiť počas T3

T2 sa musí dokončiť predtým, než začne T1

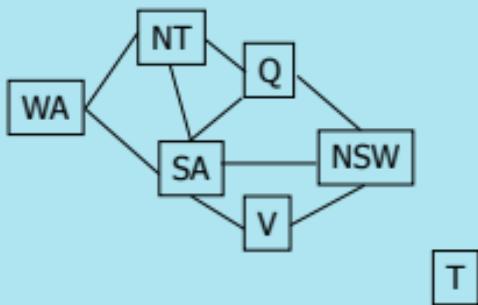
T2 sa musí prekrývať s T3

T4 sa musí začať po skončení T1

- Sú tieto ohraničenia vôbec zlučiteľné?
- nájst' časový vzťah medzi každými dvoma úlohami

graf ohraničení

binárne ohraničenia



Aký by bol graf
ohraničení pre sudoku?

dve premenné susedia ak sú spojené hranou

Generuj a testuj

- Generuj všetky možné priradenia
- Otestuj každé priradenie, či splňa všetky ohraničenia
- Veľmi neefektívny postup !

“Programovanie ohraničení”

- Problém:

$X::\{1,2\}$, $Y::\{1,2\}$, $Z::\{1,2\}$

$X = Y$, $X \neq Z$, $Y > Z$

generovanie a testovanie

X	Y	Z	test
1	1	1	nie
1	1	2	nie
1	2	1	nie
1	2	2	nie
2	1	1	nie
2	1	2	nie
2	2	1	áno

hľadanie s ustupovaním

X	Y	Z	test
1	1	1	nie
		2	nie
	2		nie
2	1		nie
	2	1	áno

Prístupy k riešeniu problémov spĺňania ohraničení

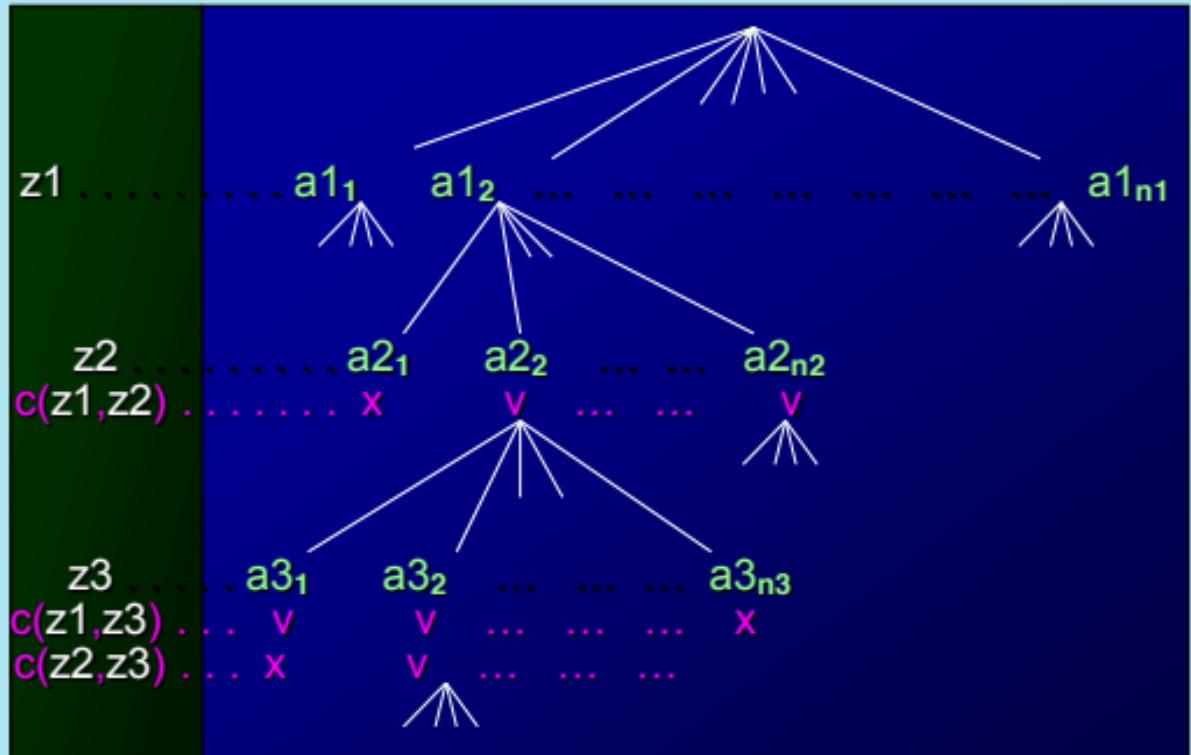
- Redukčné metódy
 - sa snažia zredukovať priestor problému
 - domény premenných a/alebo ohraničenia
 - aby nájdenie riešenia bolo čo najjednoduchšie
 - kľúčový pojem: silná k-konzistencia
 - zabezpečiť ju je výpočtovo náročné,
 - ale potom nájdenie riešenia je veľmi jednoduché
- Prehľadávacie algoritmy
 - prehľadávajú priestor problému, aby našli riešenie
 - jednoduché algoritmy, ale často musia prehľadávať
 - obrovské časti priestoru problému
- Kombinované algoritmy
 - kombinujú redukovanie a prehľadávanie
 - pred každým krokom prehľadávania sa robí redukcia

Metódy prehľadávania

- V prípade nutnosti použitia niektornej z úplných metód prehľadávania je bežným postupom použiť prehľadávanie do hĺbky s neúplným priradením
 - Na začiatku neexistujú žiadne priradenia
 - Generovanie potomkov priradením hodnoty pre ďalší uzol
- Analógia: hľadanie cesty – na začiatku cestu nepoznáme, v každom časovom okamihu k ceste pridávame ďalší segment

reprezentovanie ALEBO stromom:

zvoliť usporiadanie premenných: z_1, z_2, \dots, z_n



ALEBO strom

- pre každú úroveň (hĺbku) (i) :

→ vytvor hranu pre každé možné priradenie premennej z_i

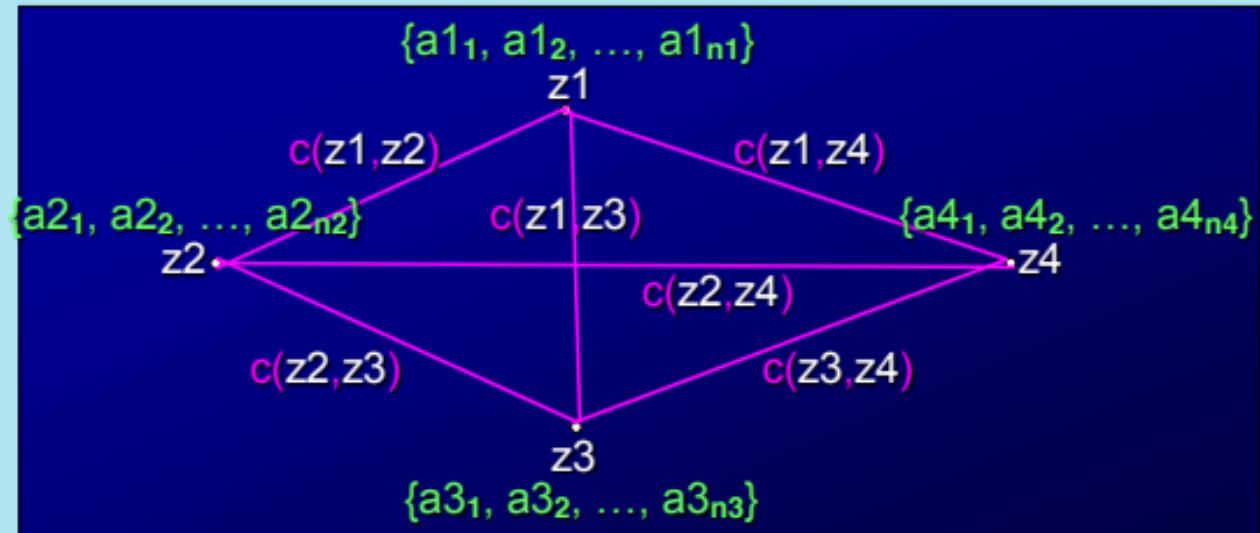
→ prever splnenie všetkých ohraničení $c(z_j, z_i)$, $j < i$

→ ak sú všetky ohraničenia splnené, pokračuj na úroveň $i+1$

◎ poznámka: toto je opis reprezentácie problému pre hľadanie riešenia

→ pri samotnom hľadaní sa buduje iba (malá) časť tohto stromu

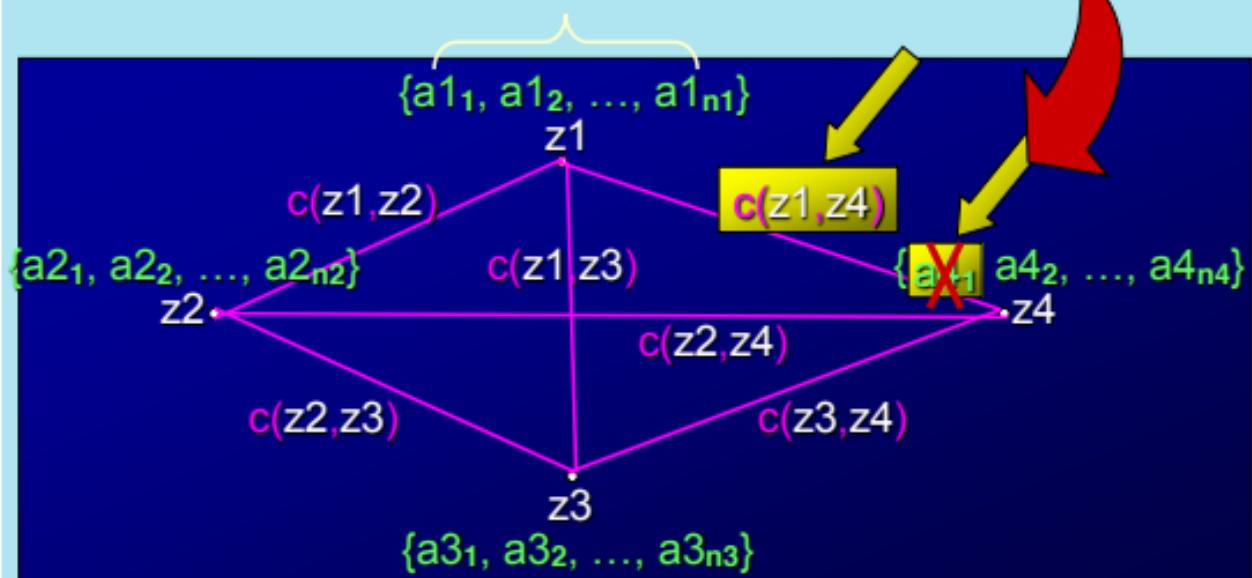
siete ohraničení



+ relaxácia = vyber hranu (ohraničenie) a odstráň nezlučiteľné hodnoty domény

Relaxácia:

c(a₁₁, a₄₁) je nikdy splnené!



Prehľadávanie s ustupovaním (backtracking, spätný chod)

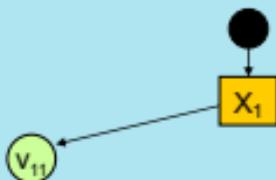
- Pri prehľadávaní do hĺbky dochádza ku generovaniu všetkých potomkov
 - Vysoké pamäťové nároky
- Prehľadávanie s ustupovaním: počkaj s vytvorením potomka až do doby, kedy ho budeš potrebovať
 - Zapamätaj si iba to, že ho v budúcnosti bude potrebné vytvoriť
- Chronologické vracanie sa

Príklad – Prehľadávanie s ustupovaním (3 uzly)



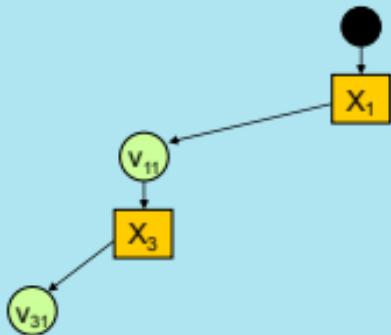
Priradenie = {}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



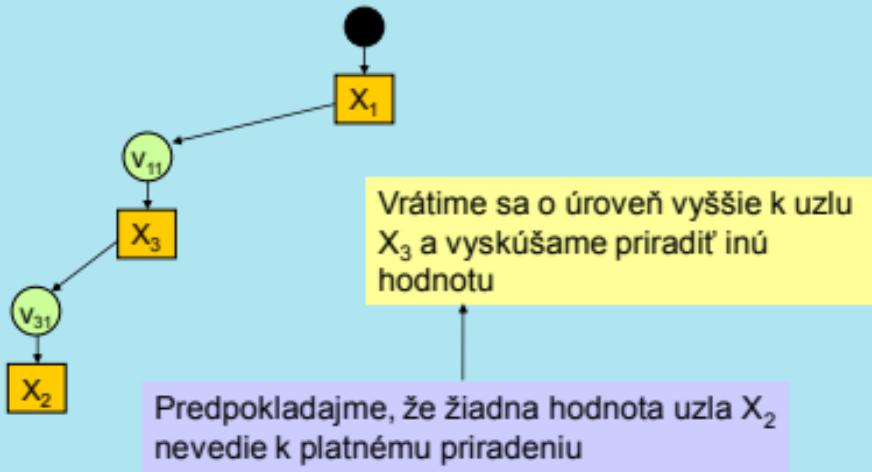
Priradenie = $\{(X_1, v_{11})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



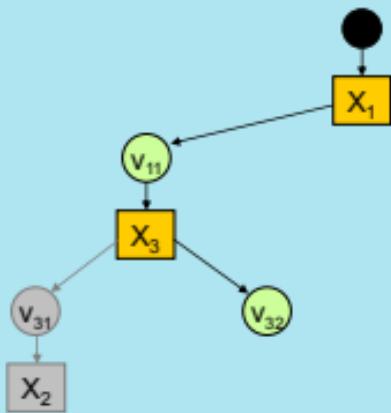
Priradenie = $\{(X_1, v_{11}), (X_3, v_{31})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



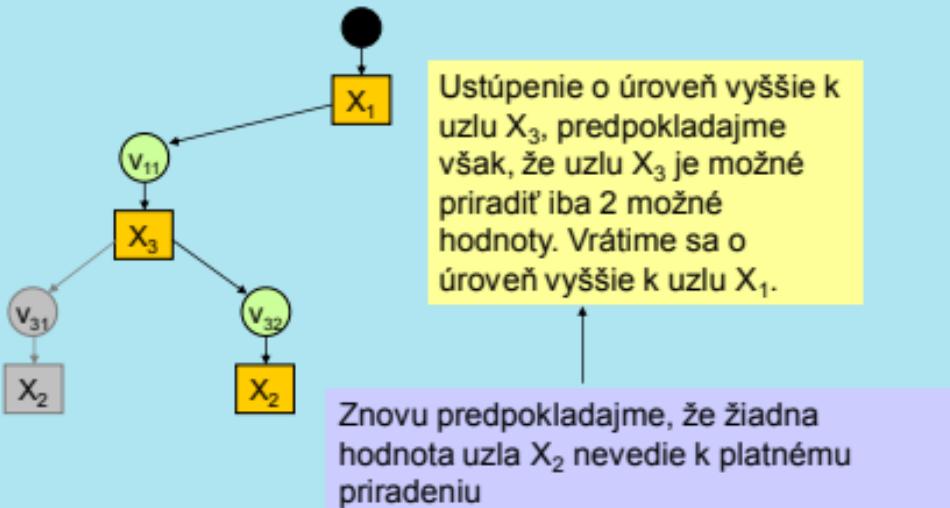
$$\text{Priradenie} = \{(X_1, v_{11}), (X_3, v_{31})\}$$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



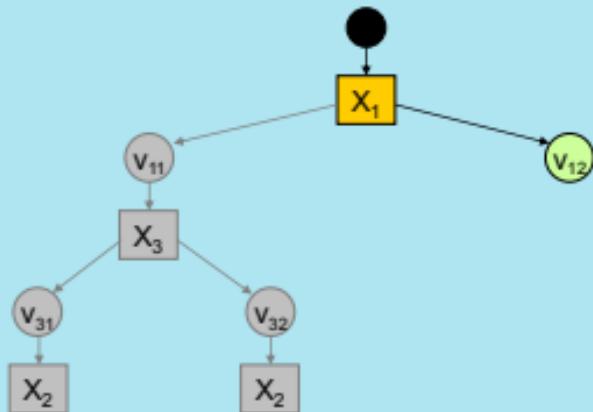
Priradenie = $\{(X_1, v_{11}), (X_3, v_{32})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



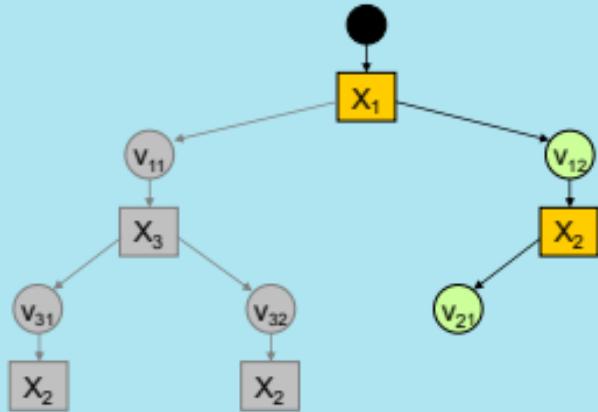
$$\text{Priradenie} = \{(X_1, v_{11}), (X_3, v_{32})\}$$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



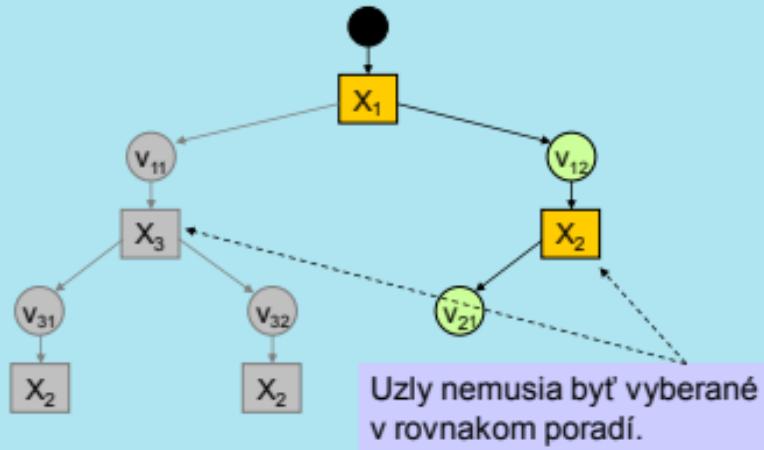
Priradenie = $\{(X_1, v_{12})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



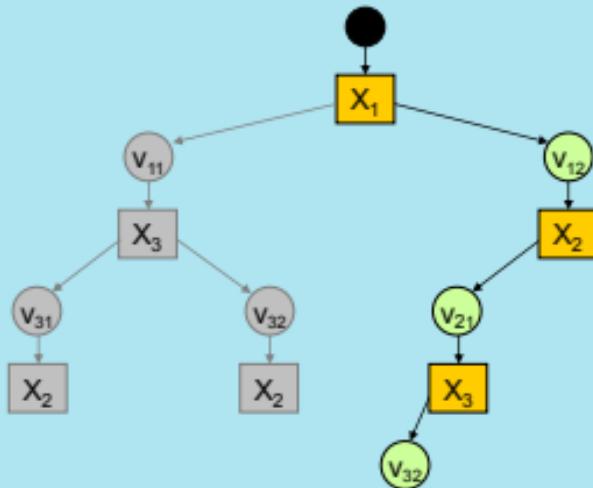
Priradenie = $\{(X_1, v_{12}), (X_2, v_{21})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



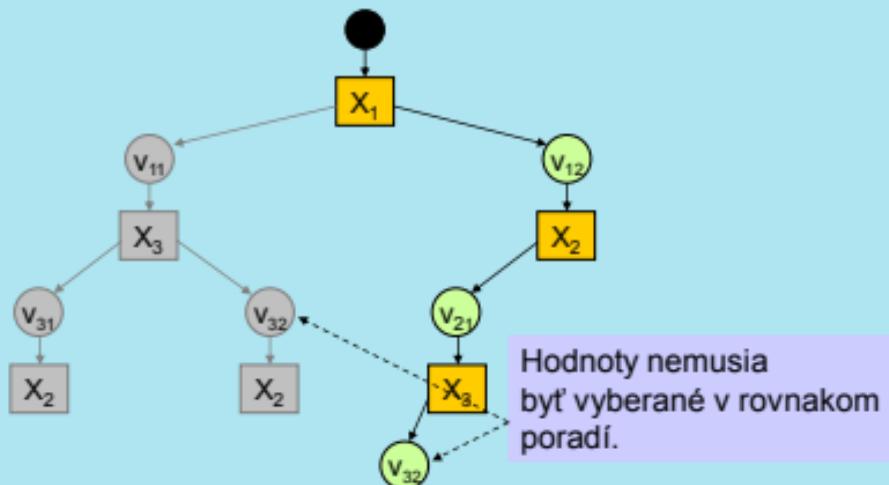
Priradenie = $\{(X_1, v_{12}), (X_2, v_{21})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



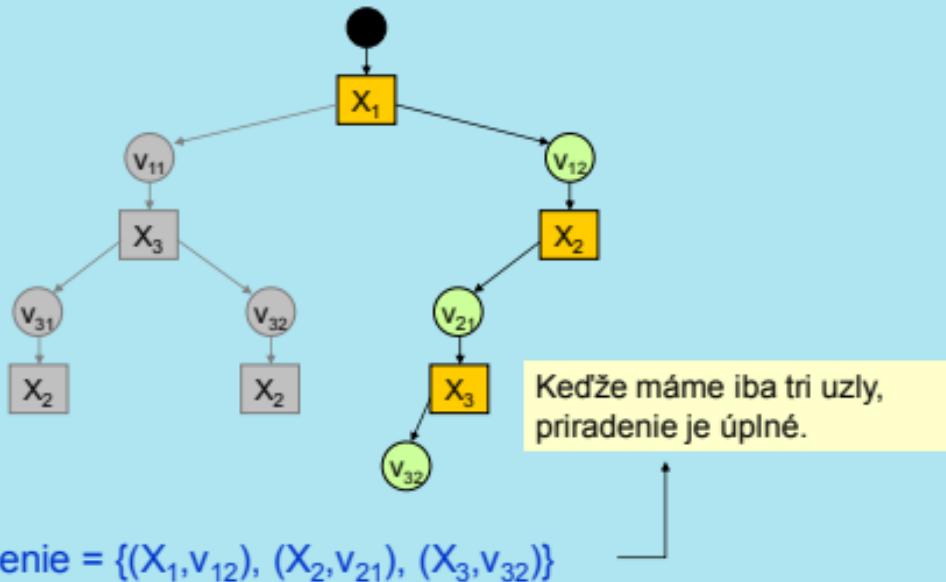
Priradenie = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



Priradenie = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

Príklad – Prehľadávanie s ustupovaním (3 uzly)



Algoritmus - prehľadávanie s ustupovaním

CSP-BACKTRACKING(A)

if priradenie A je úplné **then return** A

X \leftarrow vyber uzol, ktorý nie je v A, tj premennú s nepriradenou hodnotou

D \leftarrow vyber usporiadanie na doméne X

for each hodnota v in D **do**

if $X \leftarrow v$ je konzistentné s A **then**

 pridaj ($X \leftarrow v$) do A

 riešenie \leftarrow CSP-BACKTRACKING(A)

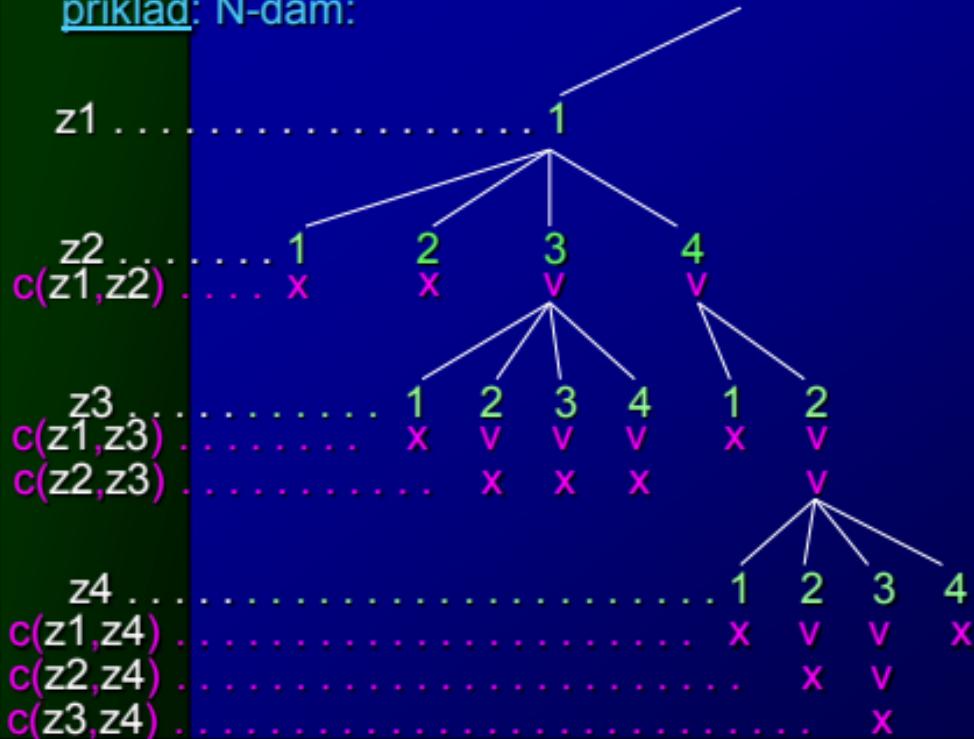
if riešenie \neq neúspech **then return** riešenie

return neúspech

prehľadávanie s ustupovaním

prezeraj ALEBO strom do hĺbky, zľava doprava.

príklad: N-dám:



Algoritmus - prehľadávanie s ustupovaním

Backtr(híbka)

For $k = 1$ to $n_{\text{híbka}}$ do

$Z_{\text{híbka}} := a_{\text{híbka},k};$

preveruj všetky ohraničenia $c(z_1, Z_{\text{híbka}})$

$s 1 \leq i < \text{híbka}$ pokial nejaké nesplnené

If žiadne nesplnené then

If $\text{híbka} = n$ then

return(z_1, z_2, \dots, z_n)

Else

$\text{híbka} := \text{híbka} + 1;$

Backtr(híbka);

$\text{híbka} := \text{híbka} - 1;$

End-For

$Z_{\text{híbka}}, \dots, a_{\text{híbka},k}$

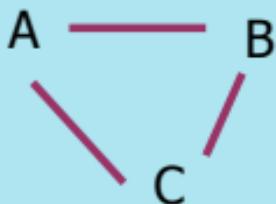
$C(z_1, Z_{\text{híbka}}) \dots v$
 $C(z_2, Z_{\text{híbka}}) \dots v$

$C(Z_{\text{híbka}-1}, Z_{\text{híbka}}) v$

$Z_{\text{híbka}+1}, \dots, a_{\text{híbka}+1,1}$

Je možné zafarbiť trojuholník dvoma farbami ?

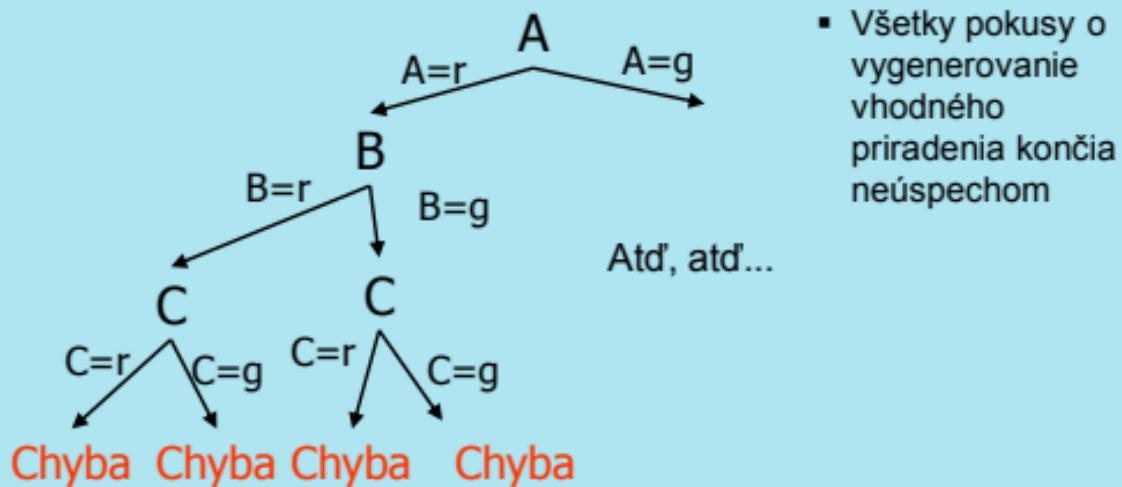
- Je možné priradiť daným uzlom a ohraničeniam hodnoty z množiny $\{r, g\}$?



- Je zrejmé, že nie!
- Ako to však dokázať pomocou metód prehľadávania ?

Je možné zafarbiť trojuholník dvoma farbami ?

- Generuj a Testuj": Zafarbi uzly v poradí A, B, C

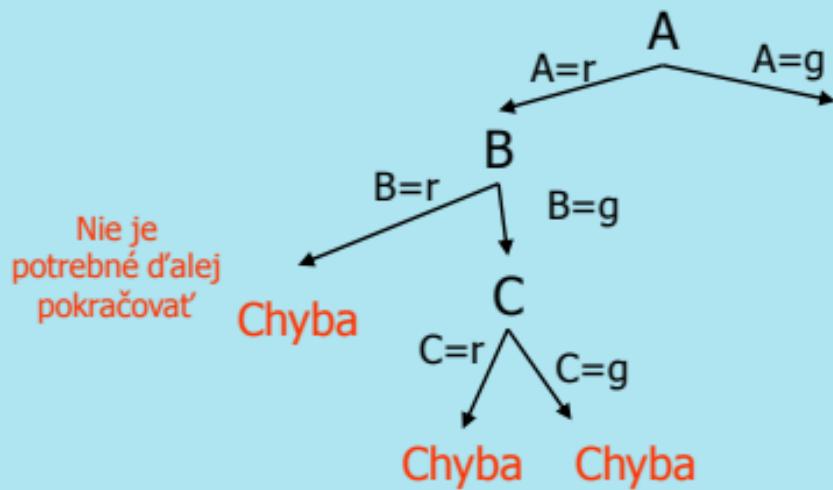


Predčasný neúspech

- Predpokladajme, že neúplné priradovanie zlyhá, napr. dôjde k porušeniu ohraničenia
- Akékoľvek priradenia hodnôt uzlom s doposiaľ nepriradenou hodnotou, neovplyvnia fakt, že došlo k porušeniu ohraničenia – hľadanie skončí neúspechom
- V prehľadávaní s ustupovaním:
 - len čo dôjde k porušeniu ohraničenia neúplným priradením, je možné orezať prehľadávanie

Je možné zafarbiť trojuholník dvoma farbami ?

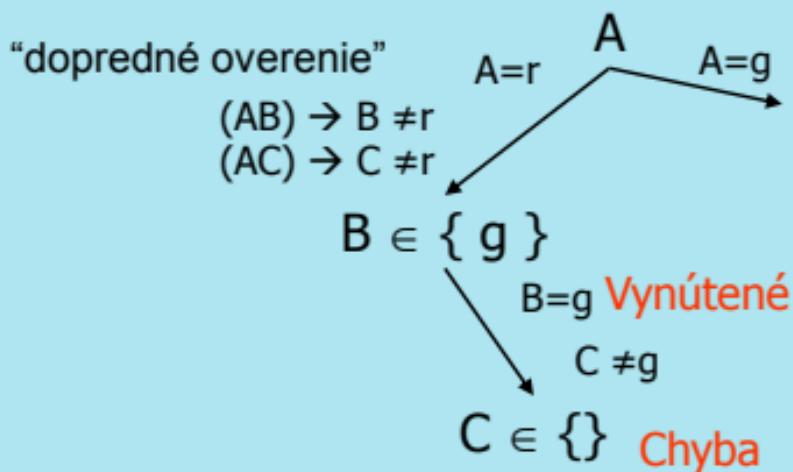
- Naivné prehľadávanie s ustupovaním:
 - Orezanie nesprávnych neúplných priradení



Prehľadávanie s ustupovaním a farbenie grafu

- Naivné prehľadávanie s ustupovaním je niekedy očividne neefektívne
- Po vyfarbení uzla nejakou farbou, nemôžeme už danú farbu priradiť jeho susedom
 - je teda zbytočné vytvárať potomkov tohto uzla

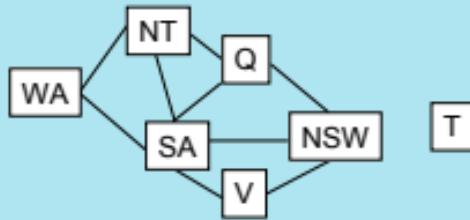
Je možné zafarbiť trojuholník dvoma farbami ?



Dopredné overenie a problém farbenia grafu

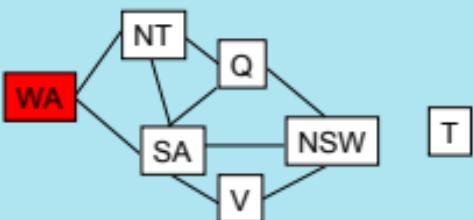
- Po vyfarbení uzla nejakou farbou, nemôžeme už danú farbu priradiť jeho susedom
- Pred priradením hodnoty danému uzlu sa overí, ktoré hodnoty je možné danému uzlu priradiť
- Tým sa niektoré priradenia stávajú vynútenými
- Redukcia faktoru vetvenia a veľkosti stromu

Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB						

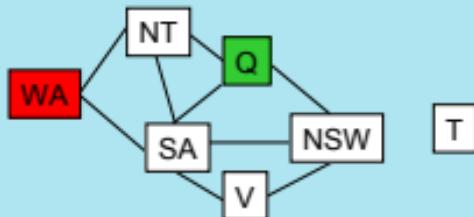
Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB						
R	X	RGB	RGB	RGB	X	RGB

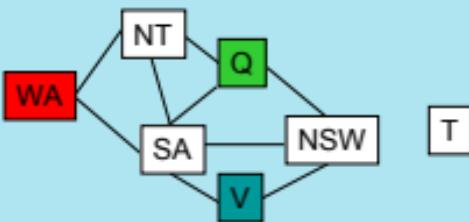
dopredné overenie odstráni hodnotu R uzlov NT a SA

Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB						
R	GB	RGB	RGB	RGB	GB	RGB
R	GB	G	RGB	RGB	GB	RGB

Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB						
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	RB	B	B	RGB

Príklad - Dopredné overenie a farbenie grafu

Prázdna množina:

Priradenie $\{(WA \leftarrow R), (Q \leftarrow G), (V \leftarrow B)\}$ nevedie k riešeniu



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	RB	B	B	RGB

Dopredné overenie

- Po vynútení hodnoty pre daný uzol je potrebné skontrolovať všetkých jeho susedov.
- Všetky hodnoty, ktoré sú s práve vynútenou hodnotou nekonzistentné, sa odstránia z domény.
- Pri odvodzovaní budú použité už iba nové hodnoty, čím sa redukuje počet uzlov, ktoré je potrebné prezrieť.

Dopredné overenie

- Znižuje veľkosť domény pre uzly
 - Počet možných alternatív klesá
 - Faktor vetvenia sa znižuje
- Zvláštne prípady:
 - $|D| = 1$: uzlu je možné priradiť iba jednu hodnotu
 - Vynútená hodnota
 - $|D| = 0$: uzlu nie je možné priradiť žiadnu hodnotu
 - Neúplné priradenie končí neúspechom
 - orezanie riešenia a spätný chod

Algoritmus – Modifikované prehľadávanie s ustupovaním

CSP-BACKTRACKING(A , doména)

if priradenie A je úplné **then return** A

$X \leftarrow$ vyber uzol, ktorý nie je v A

$D \leftarrow$ vyber usporiadanie na doméne X

for each hodnota v in D **do**

 Pridaj $(X \leftarrow v)$ do A

 doména \leftarrow DOPREDNÉ-OVERENIE (doména, X , v , A)

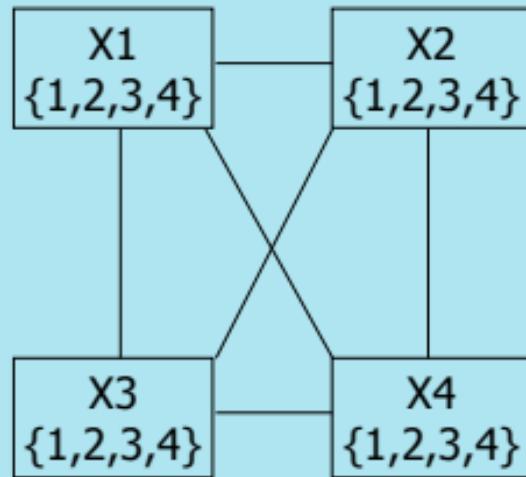
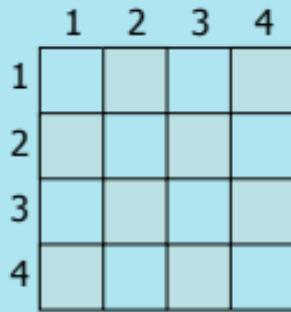
if doména pre uzol je prázdna **then return** neúspech

 riešenie \leftarrow CSP-BACKTRACKING(A , doména)

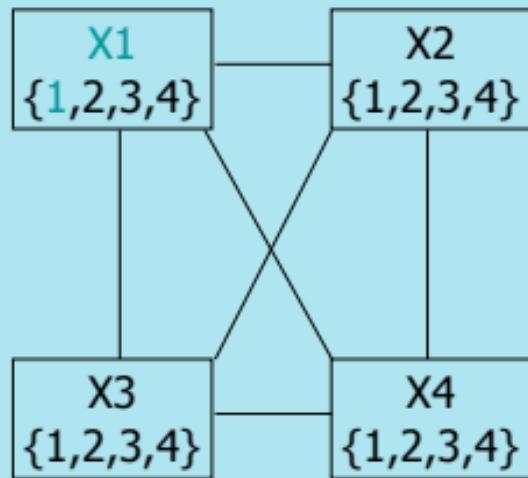
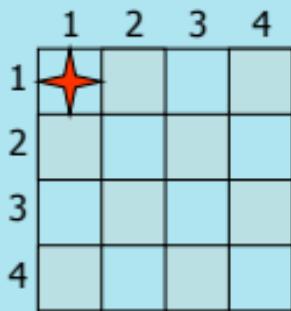
if riešenie \neq neúspech **then return** riešenie

return neúspech

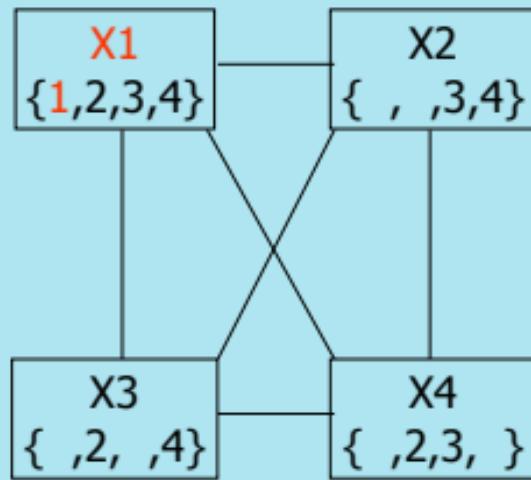
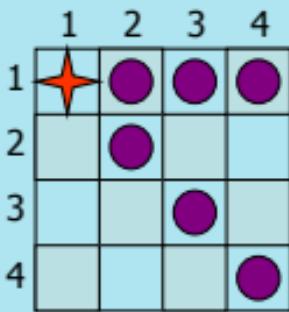
príklad: 4 dámy



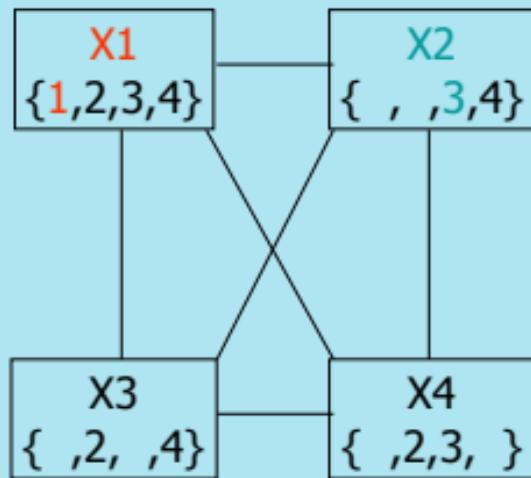
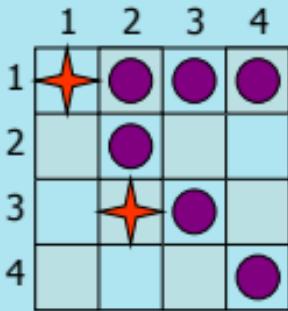
príklad: 4 dámy



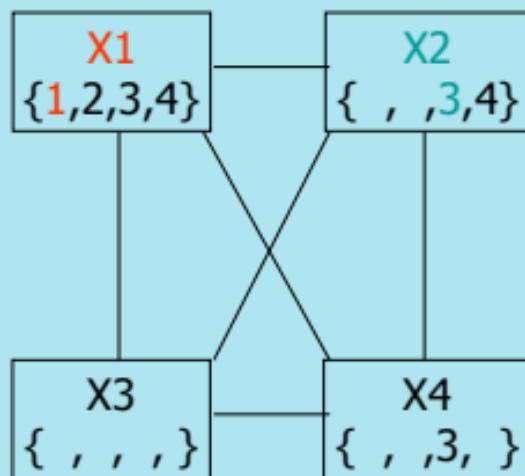
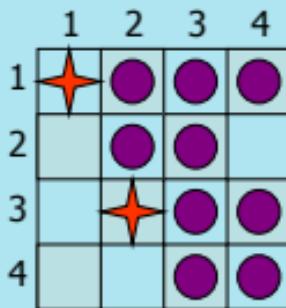
príklad: 4 dámy



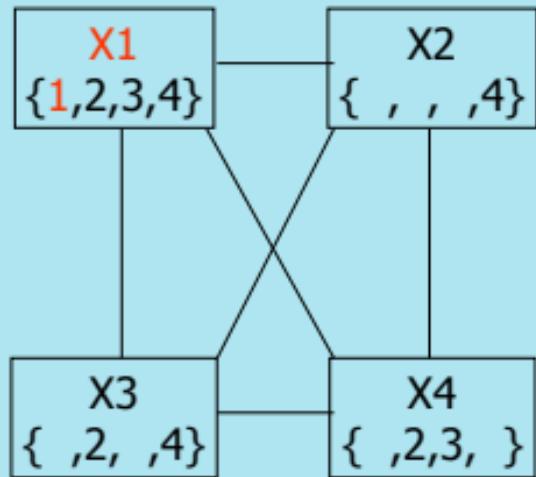
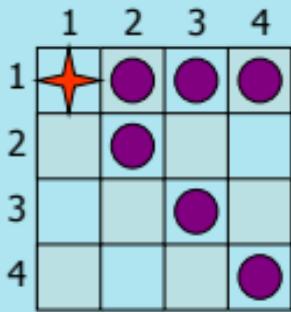
príklad: 4 dámy



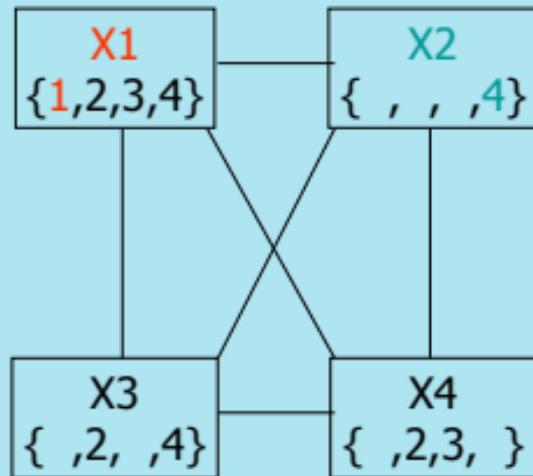
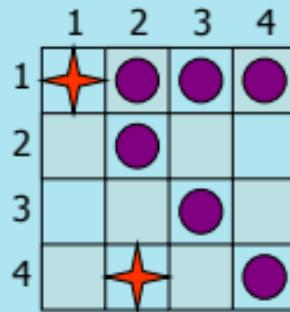
príklad: 4 dámy



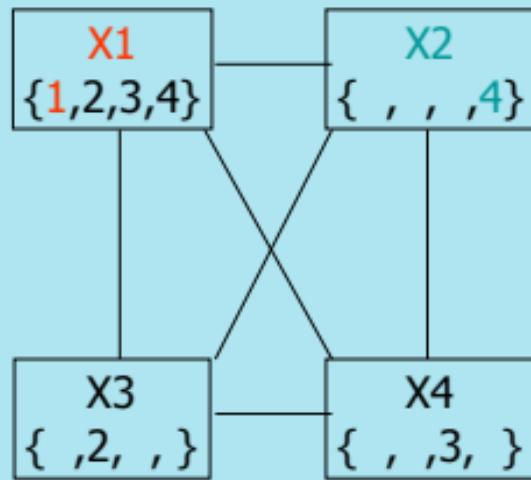
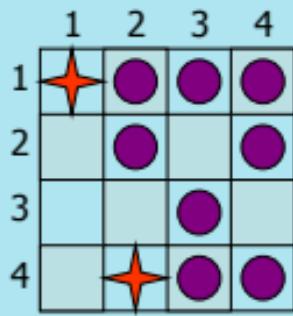
príklad: 4 dámy



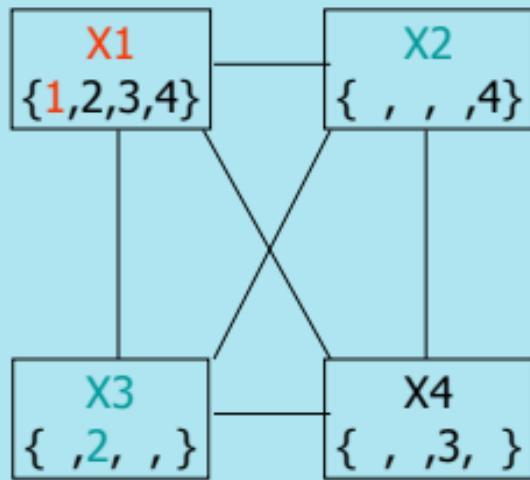
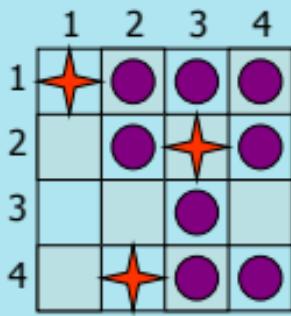
príklad: 4 dámy



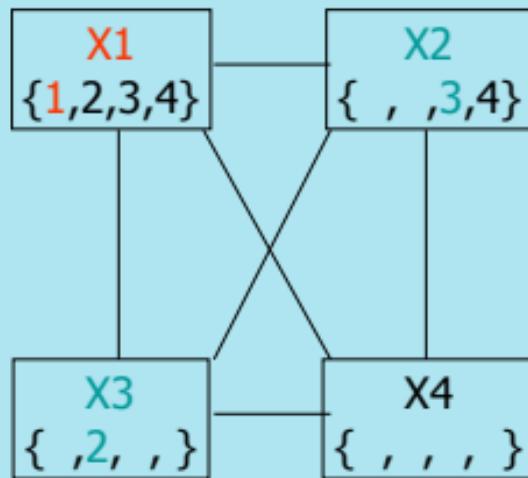
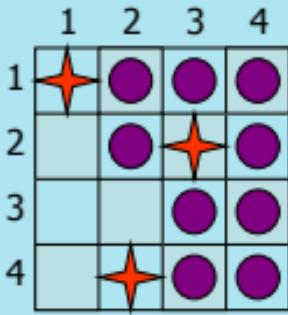
príklad: 4 dámy



príklad: 4 dámy



príklad: 4 dámy



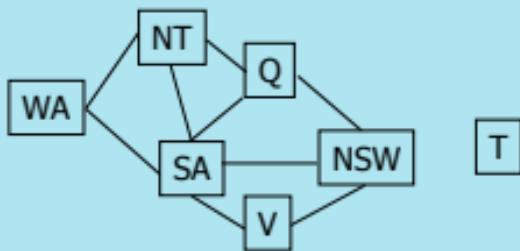
šírenie ohraničení (constraint propagation) ...

... proces určovania, ako možné hodnoty jednej premennej ovplyvnia možné hodnoty iných premenných

dopredné preverovanie (forward checking)

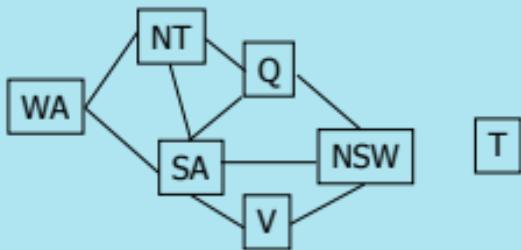
po tom, ako sa premennej X priradí hodnota v ,
pozri sa na každú nepriradenú premennú Y
spojenú s X nejakým ohraničením a zruš z domény
premennej Y všetky hodnoty, ktoré nie sú
zlučiteľné s v

ofarbenie mapy: dopredné preverovanie



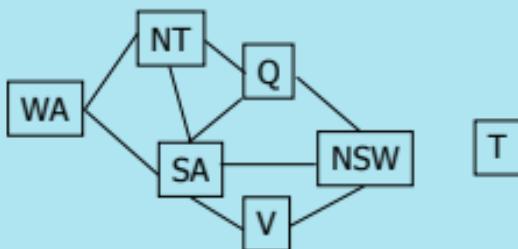
WA	NT	Q	NSW	V	SA	T
RGB						

ofarbenie mapy: dopredné preverovanie



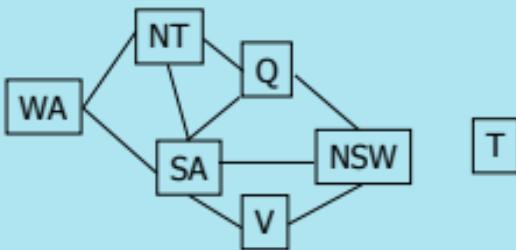
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	RGB	RGB	RGB	RGB	RGB	RGB

ofarbenie mapy: dopredné preverovanie



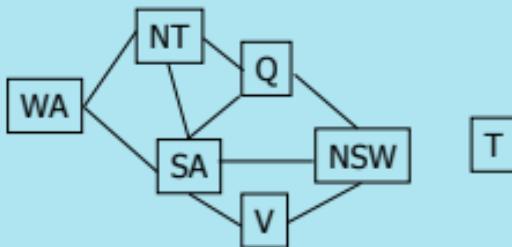
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	X	RGB	RGB	RGB	X	RGB

ofarbenie mapy: dopredné preverovanie



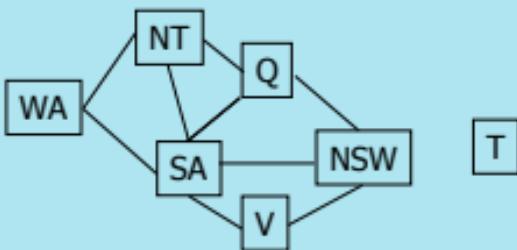
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	X	RGB	RGB	RGB	X	RGB
R	X	2: G	RGB	RGB	X	RGB

ofarbenie mapy: dopredné preverovanie



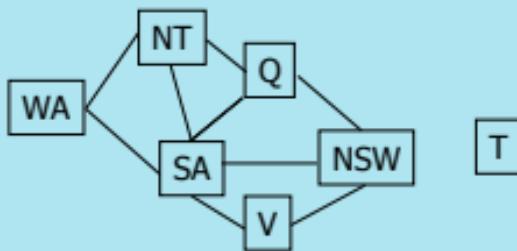
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	XGB	RGB	RGB	RGB	XGB	RGB
R	XXB	2: G	RX	RGB	XXB	RGB

ofarbenie mapy: dopredné preverovanie



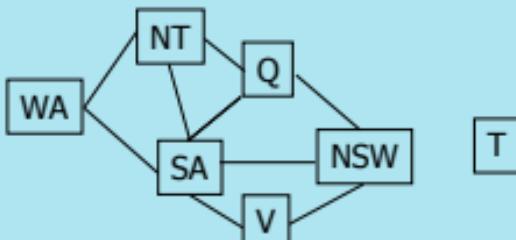
WA	NT	Q	NSW	V	SA	T
RGB						
R	XGB	RGB	RGB	RGB	XGB	RGB
R	XXB	G	RGB	RGB	XXB	RGB
R	XGB	G	RXB	3:B	XXB	RGB

ofarbenie mapy: dopredné preverovanie



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	XRGB	RGB	RGB	RGB	XRGB	RGB
R	XRGB	G	RX	RGB	XRGB	RGB
R	XRGB	G	RX	3:B	XRGB	RGB

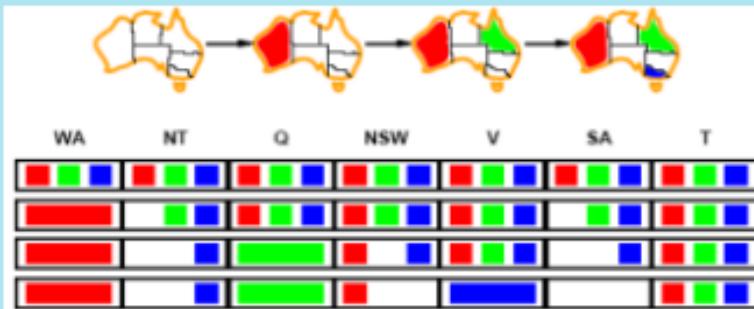
Iné nekonzistentnosti?



Nemožné priradenia, na ktoré dopredné preverovanie nepríde.

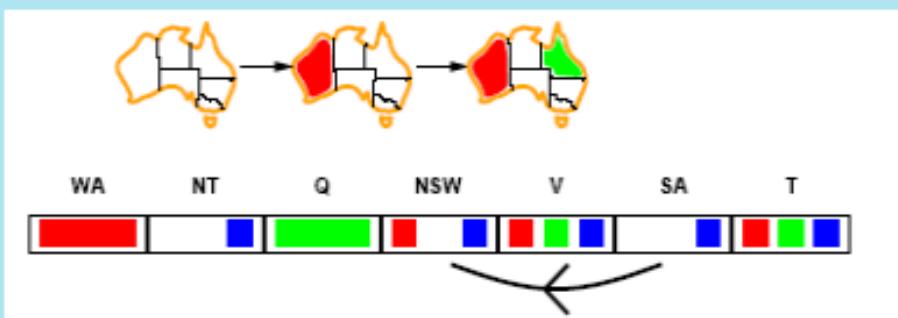
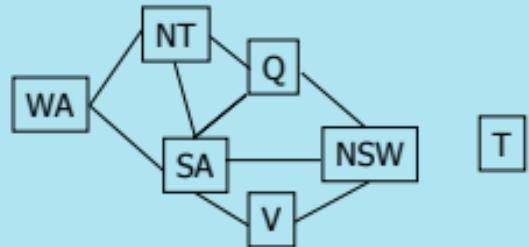
WA	NT	Q	SA	NSW	V	T
RGB	RGB	RGB		RGB	RGB	RGB
R	XGB	RGB		RGB	XGB	RGB
R	XXB	G	XGB	RGB	XXB	RGB
R	XXB	G	RXX	3:B	XXX	RGB

šírenie ohraničení



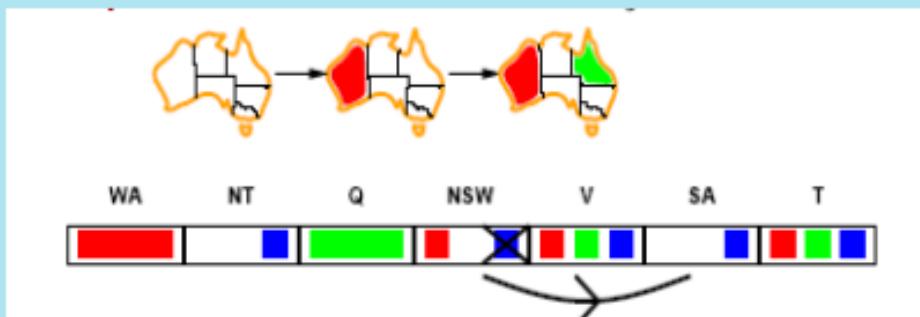
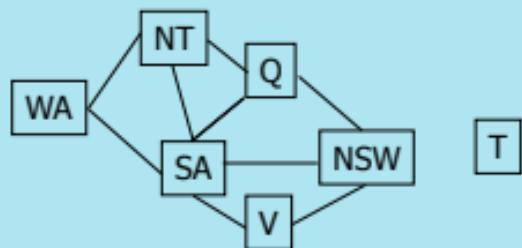
- Riešenie CSP s kombináciou heuristik a dopredného preverovania je efektívnejšie než ľubovoľný z prístupov osamote.
- Dopredné preverovanie šíri informáciu od priradených ku nepriradeným premenným, nevie však odhaliť všetky zlyhania.
 - Šírenie ohraničení opakovane vynucuje dôsledky ohraničení.

hranová konzistentnosť



- $X \rightarrow Y$ je konzistentná vtedy a len vtedy ak pre každú hodnotu x premennej X existuje nejaká prípustná hodnota y premennej Y
- $SA \rightarrow NSW$ je konzistentná vtedy a len vtedy ak $SA = \text{modrá}$ a $NSW = \text{červená}$

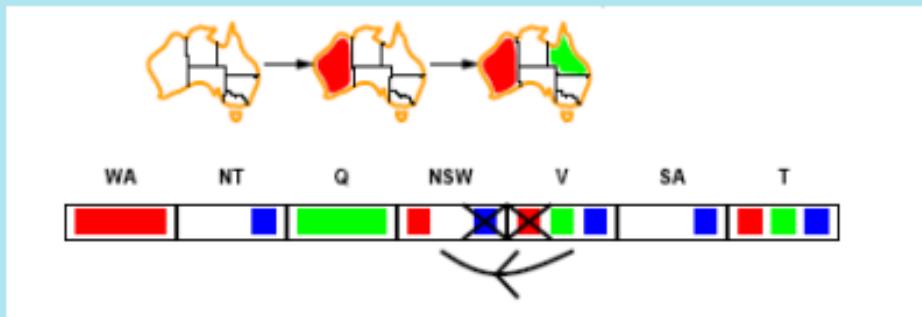
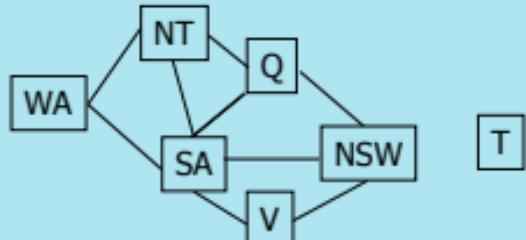
hranová konzistentnosť



- $X \rightarrow Y$ je konzistentná vtedy a len vtedy ak
pre každú hodnotu x premennej X existuje nejaká prípustná hodnota y premennej Y
- $NSW \rightarrow SA$ je konzistentná vtedy a len vtedy ak
 $NSW = \text{červená}$ a $SA = \text{modrá}$
 $NSW = \text{modrá}$ a $SA = ???$

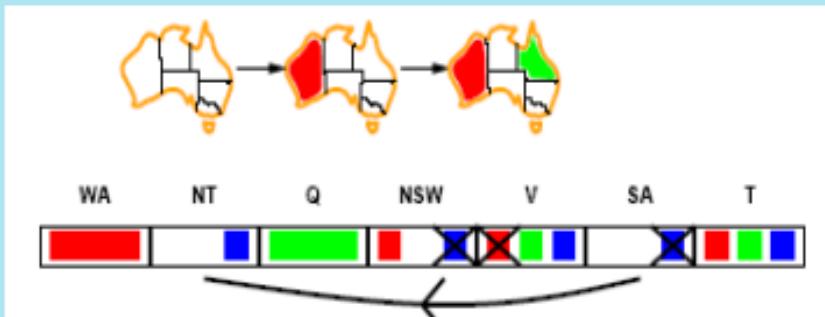
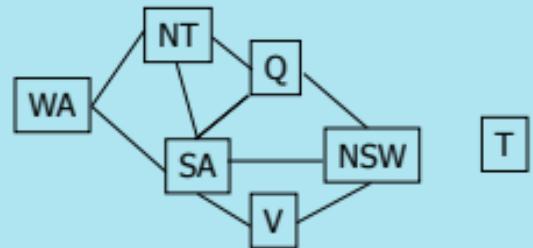
Hrana sa stane konzistentnou odstránením modrej z NSW

hranová konzistentnosť



- Znovuprever susedov *NSW*!
- $V \rightarrow NSW$ je konzistentná vtedy a len vtedy ak
 - $V = \text{modrá}$ a $NSW = \text{červená}$
 - $V = \text{zelená}$ a $NSW = \text{červená}$
 - $V = \text{červená}$ a $NSW = ??$
 - Odstráň červenú z V

hranová konzistentnosť



- $SA \rightarrow NT$ nie je konzistentná hrana
 - a nedá sa skonzistentniť
- preverovanie hranovej konzistentnosti odhalí neúspech skôr než dopredné preverovanie

Šírenie ohraničení

- Predpokladajme, že pomocou dopredného overovania (DO) nájdeme pre niektorý z uzlov vynútenú hodnotu
 - Je zbytočné čakať, pokial' sa k danému uzlu dopracujeme pomocou prehľadávania s ustupovaním, hodnotu priradíme danému uzlu okamžite.
- Keďže priradením vynútenej hodnoty došlo k zmene, je možné vykonať znova DO
- Proces je príkladom šírenia ohraničení
 - Priradenie hodnoty jednému uzlu sa rozšíri na ostatné uzly cez ohraničenia
 - Šírenie končí v okamihu, keď už nie je možné nič odvodiť

šírenie ohraničení

- Skombinovaním hodnôt pre uzly a množiny ohraničení je možné eliminovať ostatné možné hodnoty
- Zvláštne prípady
 - Všetky hodnoty pre uzol sú eliminované
 - Orezanie uzla a ústup späť
 - Všetky hodnoty, okrem jednej boli pre uzol eliminované
 - Ide o vynútenú hodnotu
 - Potrebné znova vykonať rozšírenie ohraničení
- Zníženie faktoru vetvenia, hĺbky a veľkosti stromu hľadania

šírenie ohraničení

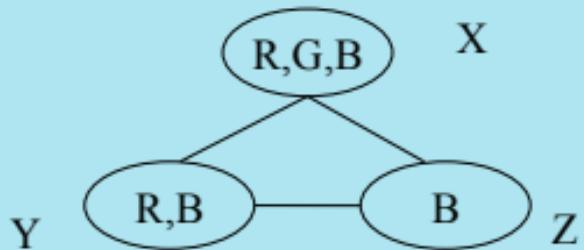
- Väčšina práce v CSP a programovaní ohraničení pozostáva z:
 - hľadaní výkonnejších foriem šírenia – metód redukujúcich veľkosť domény
 - efektívnejšej implementácií šírenia

Mechanizmus šírenia ohraničení

- hranová konzistentnosť
 - hraha $V_1 \rightarrow V_2$ je hranovo konzistentná ak pre všetky $x \in D_1$ existuje $y \in D_2$ také, že (x, y) je ohodnotenie konzistentné s ohraničeniami
- na čo je to dobré?
 - Vymaž spomedzi možných hodnôt pre vrcholy/premenné V_1 a V_2 tie, ktoré kazia (hranovú) konzistentnosť

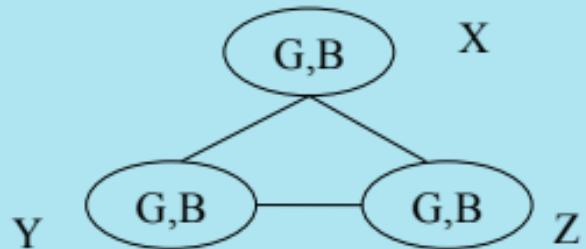
príklad: hranová konzistentnosť

- ofarbenie grafu:
 - premenné: vrcholy X, Y, Z
 - doména: farby (R,G,B)
 - ohraničenie: If hrana(a,b), then farba(a) ≠ farba(b)
- začiatočné priradenia:

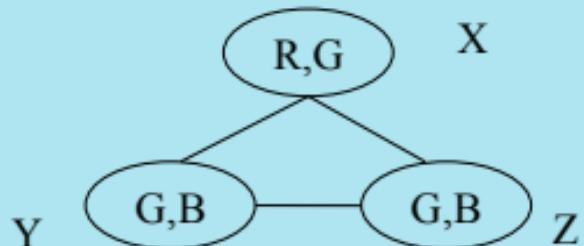


medze metódy hranovej konzistentnosti

- neexistuje konzistentné priradenie



- viac konzistentných priradení



k-konzistentnosť

- Graf (siet) ohraničení je k-konzistentný ak pre každú množinu $k-1$ premenných a každé konzistentné priradenie hodnôt týmto premenným existuje konzistentné priradenie ľubovoľnej k -tej premennej.
- inými slovami: *Nech ľubovoľným $k-1$ premenným sú priradené také hodnoty z ich domén, aby všetky ohraničenia definované nad touto $(k-1)$ -ticou premenných boli splnené. Pre ľubovoľnú ďalšiu k -tu premennú je možné vybrať takú hodnotu z jej domény, že všetky ohraničenia definované nad vzniknutou k -ticou premenných sú splnené.*

k-konzistentnosť

- k-konzistentnosť je zovšeobecnením hranovej konzistentnosti:
 - $k = 2$ – hranová konzistentnosť
 - $k = 1$ – vrcholová konzistentnosť
 - $k = 3$ – konzistentnosť po ceste: ľubovoľnú dvojicu susediacich premenných možno rozšíriť na tretiu susediacu premennú

ako zabezpečiť k-konzistentnosť?

- 1-konzistentnosť (NC):
 - stačí porovnať každú doménu D_i s unárnym ohraničením C_i a odstrániť všetky nekonzistentné hodnoty z D_i
- 2-konzistentnosť (AC):
 - hodnoty ľubovoľnej premennej v_i v doméne D_i sa porovnávajú s hodnotami inej premennej v_j v doméne D_j a ak pre ľubovoľnú hodnotu $h_{m,i}$ z D_i neexistuje konzistentná hodnota z D_j (s ohľadom na $C_{i,j}$), táto hodnota $h_{m,i}$ sa odstráni z domény D_i
- k-konzistentnosť, $k > 2$: exponenciálna zložitosť

silná k-konzistentnosť

- Graf ohraničení je silne k-konzistentný ak je k-konzistentný a $(k-1)$ -konzistentný a ... 2-konzistentný a 1-konzistentný t.j. ak je j-konzistentný pre všetky j z intervalu $<1,k>$
- načo je to dobré?
 - ak máme úlohu s ohraničeniami s n vrcholmi a vieme ju opísť grafom ohraničení, ktorý je silne n -konzistentný, tak riešenie problému sa dá nájsť bez ustupovania
 - ako to?
 - hľadáme riešenie pre nejakú premennú v_1 . Len čo ho nájdeme, máme zaručené, že sa dá nájsť pre v_2 , pretože graf je 2-konzistentný atď. až po n .

silná k-konzistentnosť

- časová zložitosť je parádna - $O(nd)$ namiesto $O(n^2d^3)$ (zložitosť AC3)
- ale! nič nie je zadarmo. Ľubovoľný algoritmus na zistenie silnej k-konzistentnosti musí byť exponenciálny v čase.
- možno hľadať „optimum“ kolko nechať na hľadanie a kolko osekávať preverovaním konzistentnosti:
 - začať s preverovaním konzistentnosti, osekávať kým sa dá alebo kým je to ešte ako tak efektívne
 - pokračovať s prehľadávaním (nutne aj s ustupovaním, pokiaľ sme sa nedosekali do silnej n-konzistentnosti; o čo sa ale kvôli zložitosti často ani nepokúšame, najčastejšie hranová konzistentnosť lebo binárny problém)

preverovanie hranovej konzistentnosti

- predspracovanie pre začatím prehľadávania
- prípadne po každom priradení
- znamená viac počítania, ale môže skôr eliminovať nekonzistentné časti stavového priestoru (než prehľadávanie)
- hrany sa musia preverovať systematicky
 - ak premenná stratí jednu hodnotu, musia sa nanovo preveriť všetky hrany do nej smerujúce

Algoritmus AC-3

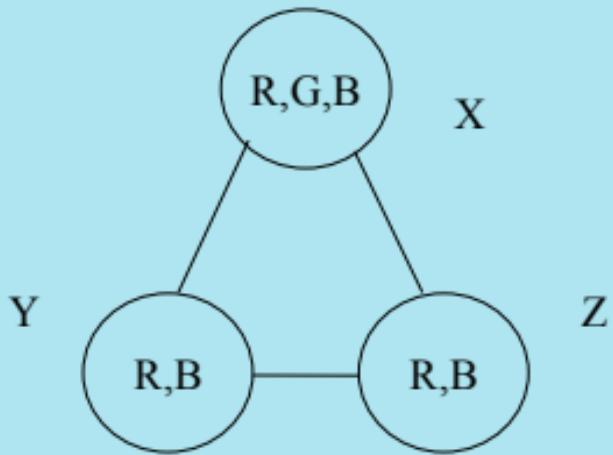
```
function AC-3(csp) returns CSP s redukovanou doménou
    inputs: csp, binárne CSP s uzlami  $\{X_1, X_2, \dots, X_N\}$ 
            front, front hrán
    for each  $X_i$  in  $\{X_1, X_2, \dots, X_N\}$  do
        for each  $X_j$  in  $\{X_1, X_2, \dots, X_N\}$  do
            if  $C(X_i, X_j)$  existuje then front  $\leftarrow (X_i, X_j)$ 
    while front nie je prázdný do
         $(X_i, X_j) \leftarrow$  ODSTRÁŇ-PRVÝ(front)
        if NEKONZISTENTNÉ-HODNOTY( $X_i, X_j$ ) then
            for each  $X_k$  in SUSEDIA[ $X_i$ ] do
                ZARAĎ-DO-FRONTU( $(X_k, X_j)$ , front)
```

```
function NEKONZISTENTNÉ-HODNOTY( $X_i, X_j$ )
    returns: true, v prípade ak dôjde k odstráneniu hodnoty
    odstránené  $\leftarrow$  false
    for each  $x$  in DOMÉNA[ $X_i$ ] do
        if nie je možné priradiť žiadnu hodnotu  $y$  z DOMÉNA[ $X_j$ ],
           tak aby bolo splnené ohraničenie  $C(X_i, X_j)$  then
            odstráň  $x$  z DOMÉNA[ $X_i$ ]
            odstránené  $\leftarrow$  true
    return odstránené
```

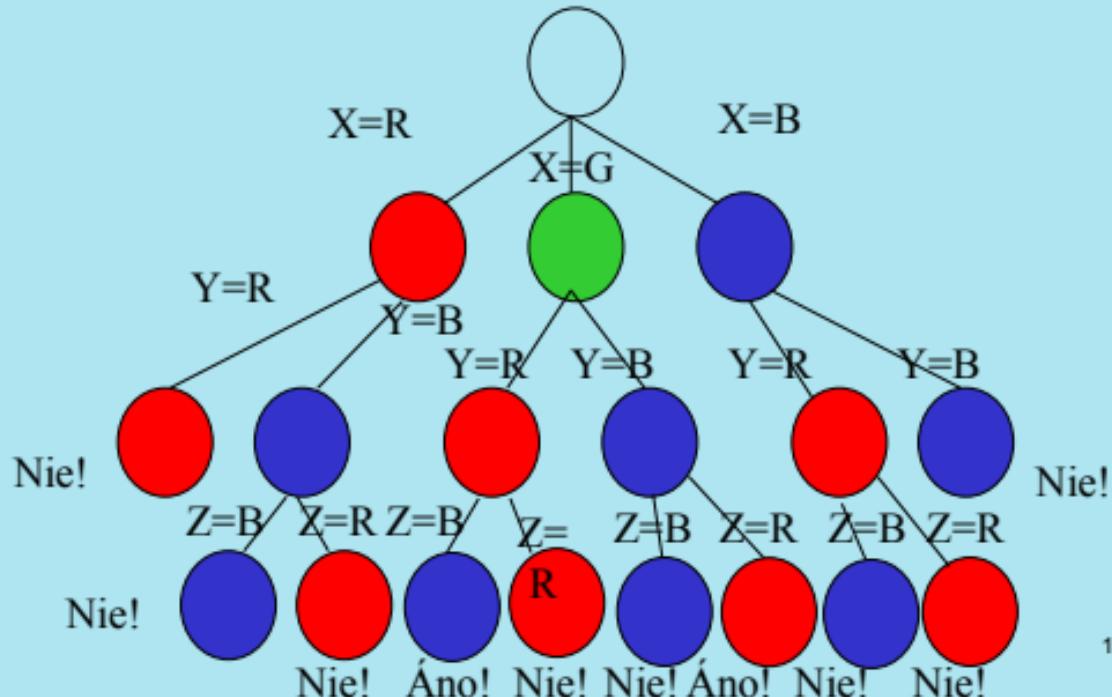
zložitosť C-3

- binárny problém s ohraničeniami má nanajvýš n^2 hrán
- každá hrana sa vloží do frontu najviac d razy (najhorší prípad)
 - (X, Y) : len d hodnôt vrchola X na zrušenie
- konzistentnosť hrany sa dá preveriť v čase $O(d^2)$
- časová zložitosť je $O(n^2 d^3)$

graf ohraničení



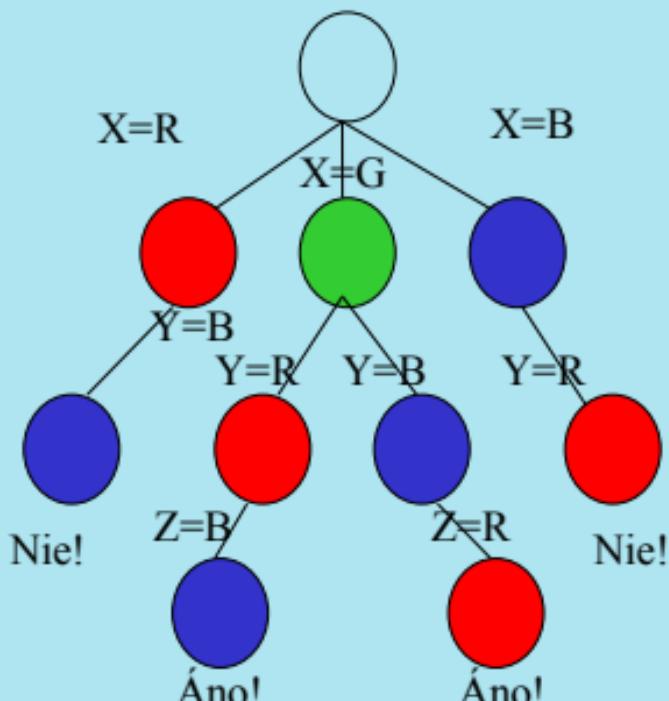
CSP ako hľadanie



súvislosti ustupovania

- CSP ako hľadanie
 - hľadanie do hĺbky
 - Maximálna hĺbka = # počet premenných
 - pokračuje dovtedy, kým nie je čiastočné alebo úplné priradenie
 - ak je konzistentné: return výsledok
 - ak nie je konzistentné (porušuje nejaké ohraničenie) -> vráť sa k predchádzajúcemu priradeniu
 - premárnené úsilie
 - prehľadávajú sa aj vetvy, kde neexistuje priradenie v súlade s ohraničeniami

ustupovanie s dopredným preverovaním



Heuristické prístupy k riešeniu CSP: dynamické usporadúvanie

- otázka: ktorý uzol rozvíjať ako ďalší?
- doterajší postup:
 - staticky: ďalší v pevne danom poradí
 - Lexikografické, zľava doprava..
 - náhodne
- Alternatívny postup:
 - dynamicky: vybrať “najlepší” v aktuálnom stave

otázky voľby

- Ako zvoliť premennú, ktorej sa bude priradovať?
- Ako zvoliť hodnotu, ktorú priradiť?

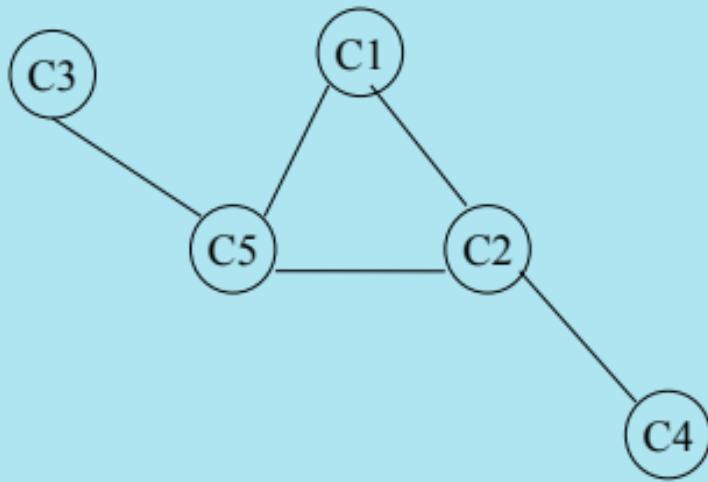
Dynamické usporadúvanie uzlov

- Doteraz sa vždy vyberal ďalší uzol vetvenia podľa presne stanoveného poradia, čo viedlo k zvýšeniu pamäťových nárokov.
- Výhodnejšie je vyberať ďalší uzol vetvenia heuristicky a dynamicky v závislosti od aktuálneho stavu.

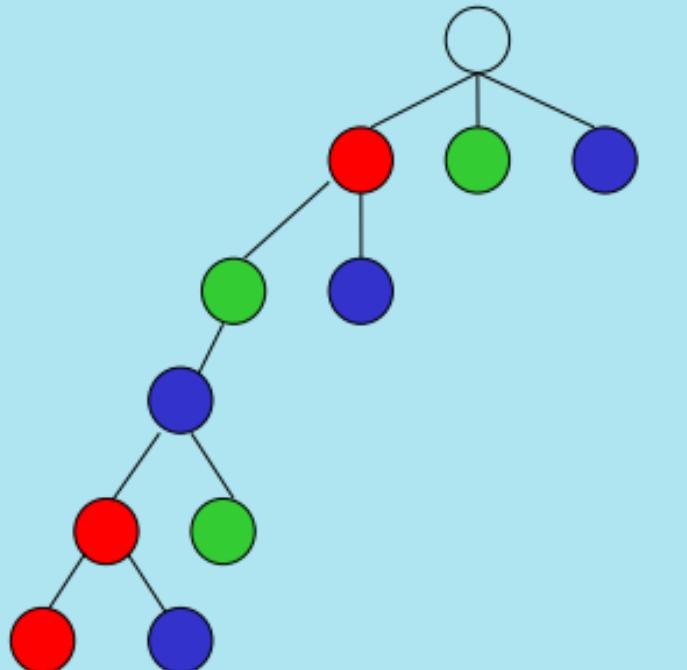
Dynamické usporadúvanie

- Heuristiky
 - najohraničenejšia premenná:
 - zvoľ premennú s najmenšou aktuálnou doménou
 - najmenej ohraničujúca hodnota:
 - zvoľ hodnotu, ktorá odstraňuje najmenej hodnôt z domén iných premenných

Dynamické usporadúvanie



Dynamické usporadúvanie



C1

C2

C5

C3

C4

inkrementálne opravovanie

- začať so začiatočným úplným priradením
 - hľadať ľačno
 - prvé priradenie je pravdepodobne nekonzistentné – t.j. porušuje niektoré ohraničenia
- inkrementálne premeň na platné riešenie
 - použiť heuristiku na zamenenie (iné priradenie) hodnoty ktorá spôsobuje porušenie ohraničenia
 - stratégia “min-konflikt”:
 - zmeň hodnotu tak, aby nastalo čo najmenej porušení ohraničení
 - ak sa nedá určiť inak, rozhoduj náhodne
 - zahrň do horolezeckého algoritmu

inkrementálne opravovanie

	Q2		Q4
Q1		Q3	



	Q2		Q4
Q1		Q3	

5 konfliktov

	Q2		Q4
Q1			
		Q3	



2 konflikty

0 konfliktov

Dôležité otázky pri prehľadávaní s ústupom

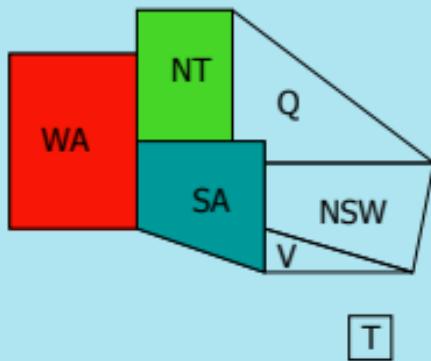
- Ktorý uzol budeme rozvíjať ako nasledujúci?
 - tzv. "Branch Variable" výber uzla na rozvíjanie
- Aké zvoliť usporiadanie potomkov jednotlivých uzlov?
 - tzv. "Branch Value" výber poradia hodnôt na priradenie
- Dobrá voľba môže významne znížiť množstvo potrebných prehľadávaní

Heuristika “Branch Variable“

- “Vyber najviac ohraničený uzol”
 - Vyber uzol s najmenšou doménou
 - Zamerané na zníženie faktoru vetvenia
- “Vyber najviac ohraničujúci uzol”
 - Vyber uzol, ktorého sa týka najviac ohraničení
 - Zamerané na generovanie rozšírení

vol'ba premennej

- ofarbenie mapy

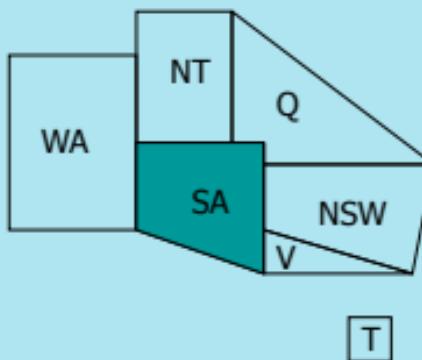


vol'ba premennej

#1: Minimum Remaining Values (aka Most-constrained-variable heuristic or Fail First):

- najohraničenejšia premenná:
 - zvoľ premennú s najmenšou aktuálnou doménou

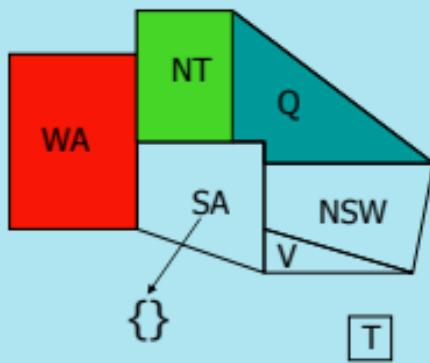
voľba premennej



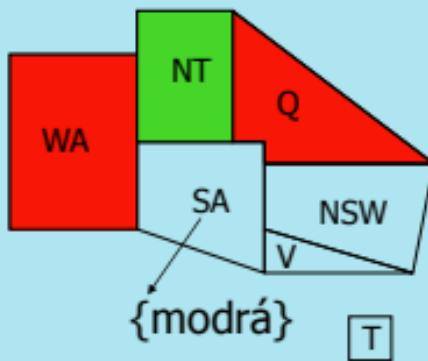
#2: heuristika najväčšieho stupňa vplyvu (degree heuristic):

zvoľ premennú účastnú v najväčšom počte
ohraničení iných ešte nepriradených premenných

vol'ba hodnoty



voľba hodnoty



#3: Least-constraining-value heuristic:

- najmenej ohraničujúca hodnota:
 - zvoľ hodnotu, ktorá odstraňuje najmenej hodnôt z domén iných premenných

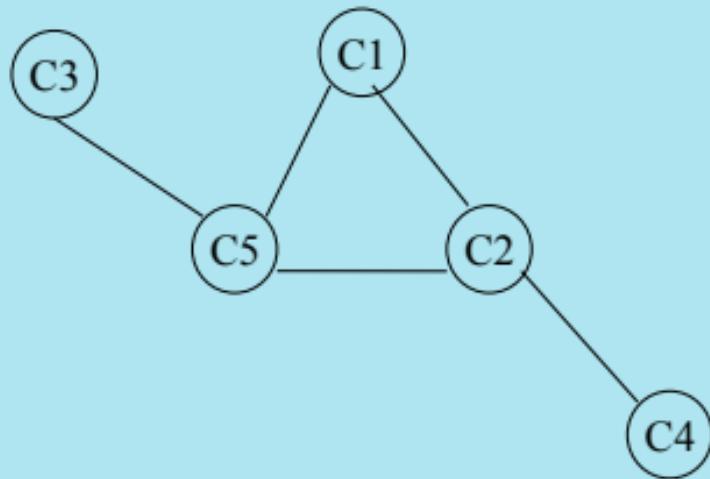
aké heuristiky pre sudoku?

- voľba premennej:
 - najohraničenejšia premenná?
 - heuristika najväčšieho stupňa vplyvu
- voľba hodnoty:
 - najmenej ohraničujúca hodnota?
- iné??

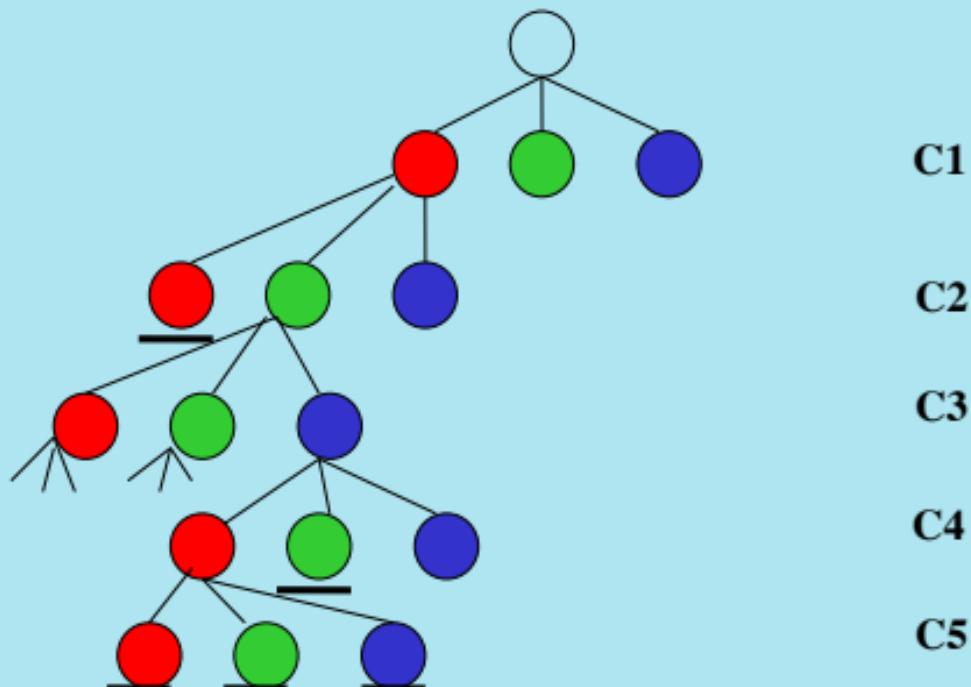
Heuristické prístupy k riešeniu CSP

- vylepšenie ustupovania
 - štandardné ustupovanie, spätný chod
 - vrátiť hľadanie do posledného priradenia
 - spätné skákanie back-jumping
 - vrátiť hľadanie do posledného priradenia, ktoré zmenšilo aktuálnu doménu
 - zmeniť priradenie, ktoré viedlo do slepého konca

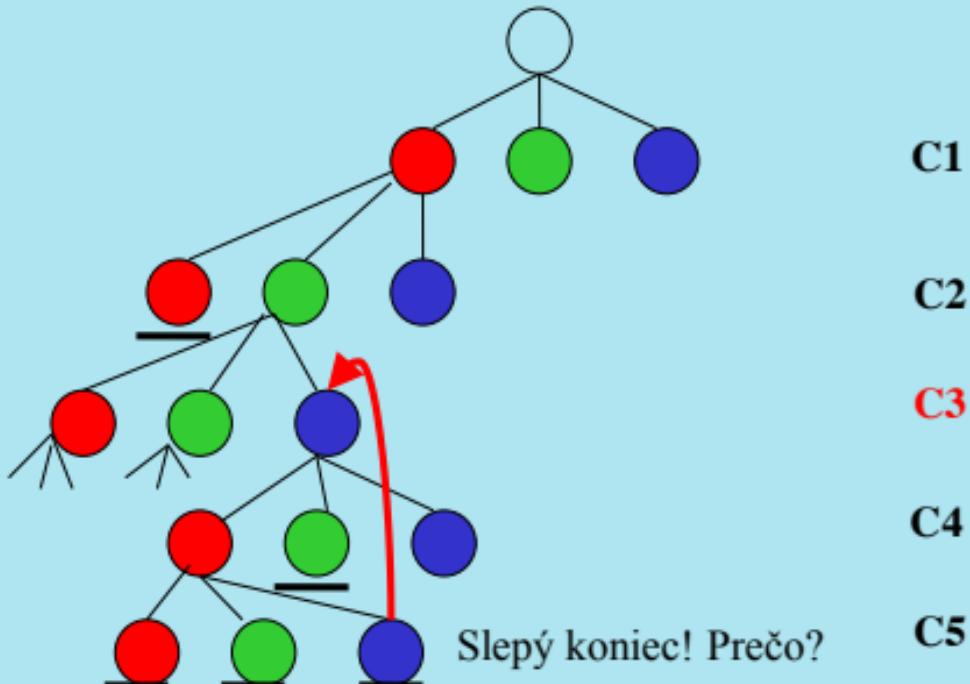
Heuristické ustupovanie: spätné skákanie



Heuristické ustupovanie: spätné skákanie



Heuristické ustupovanie: spätné skákanie

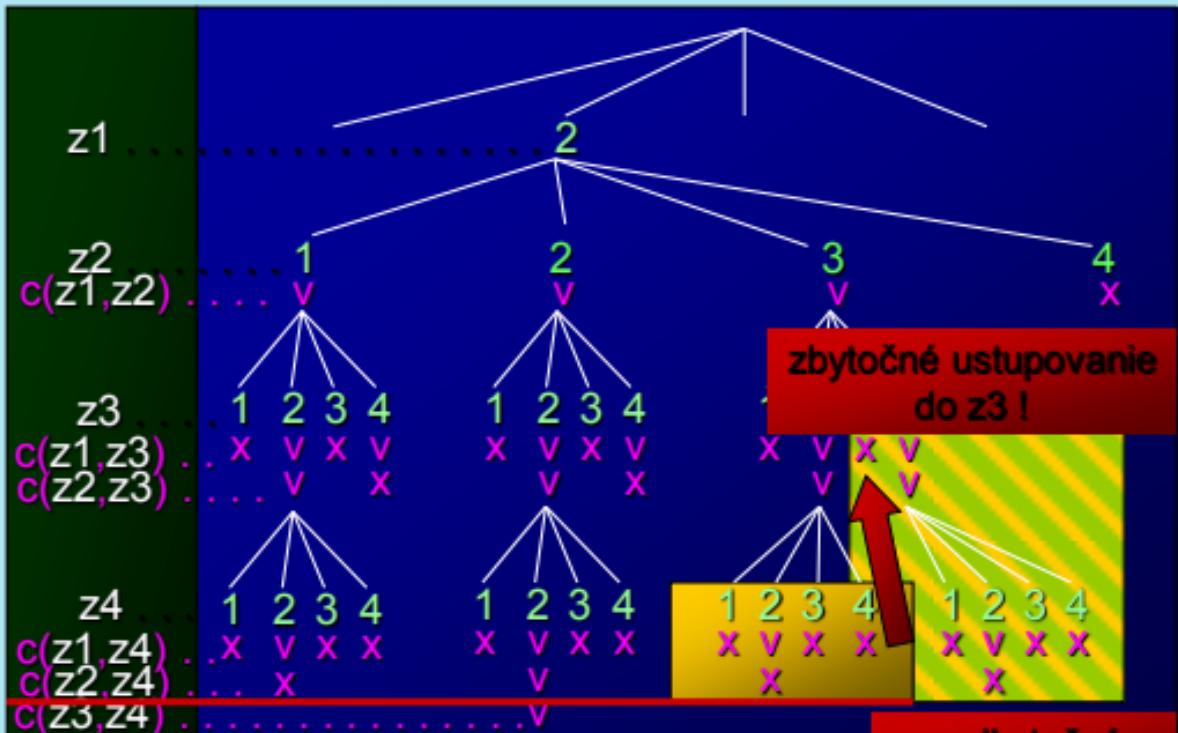


spätné skákanie

- Predchádzajúce priradenie zúžilo doménu
 - C3 = B
- zmeny v priradení medzi nemôžu ovplyvniť slepost konca
- vrátiť sa až na C3
 - vyhnúť sa márnej práci – alternatívam na C4
- vo všeobecnosti môže byť dopredné preverovanie efektívnejšie

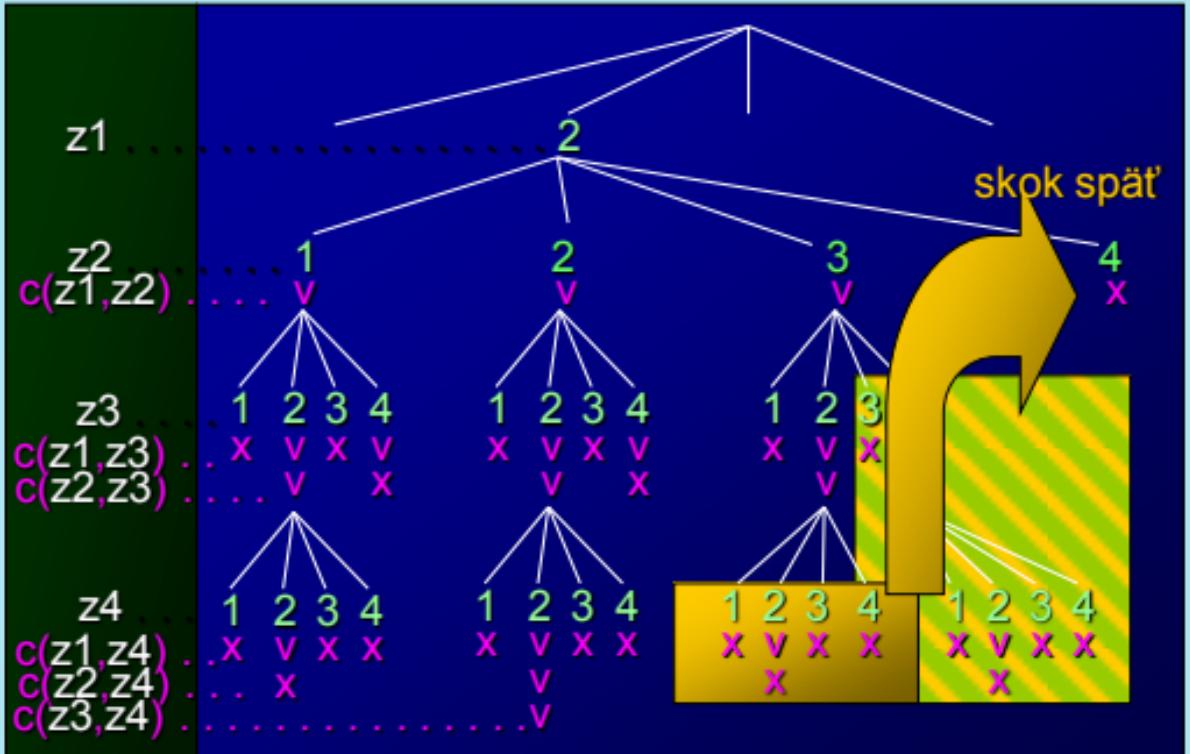
mlátenie (trashing):

príklad: p-4-dámy:



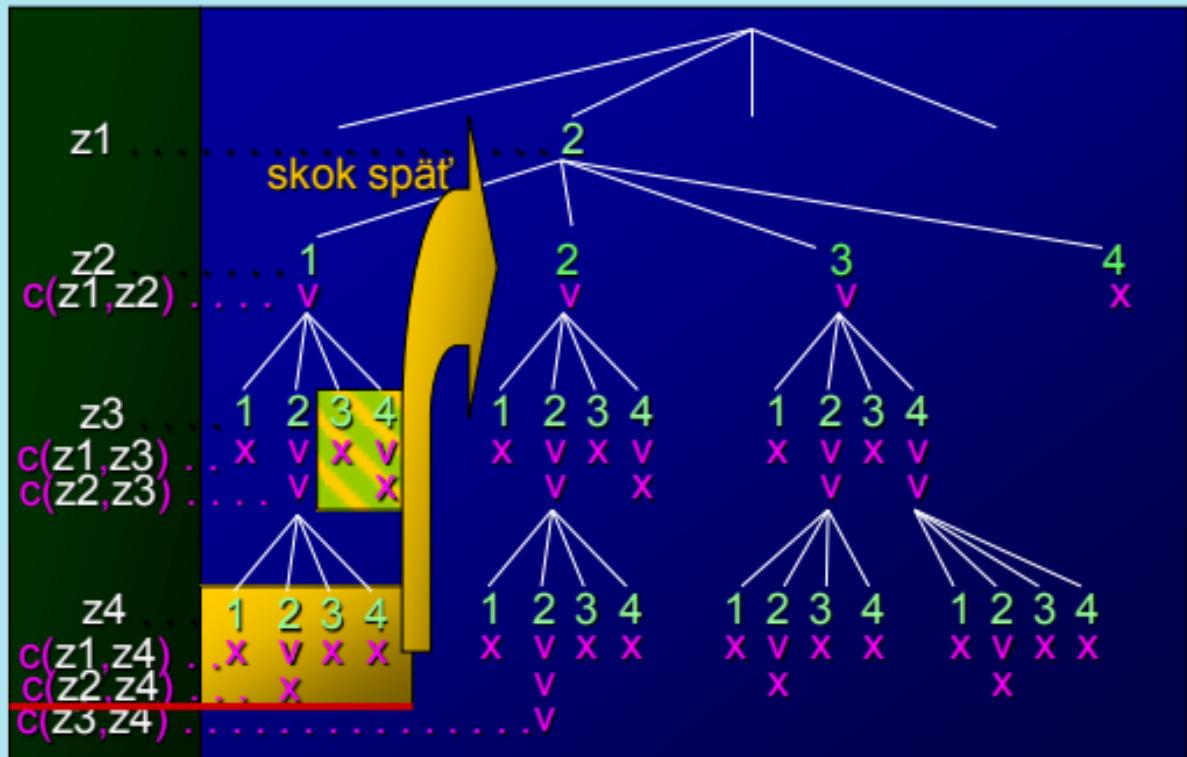
len z1 a z2 sa testujú voči hodnote
z4; z3 sa vôbec neuvažuje

mlátenie: riešenie

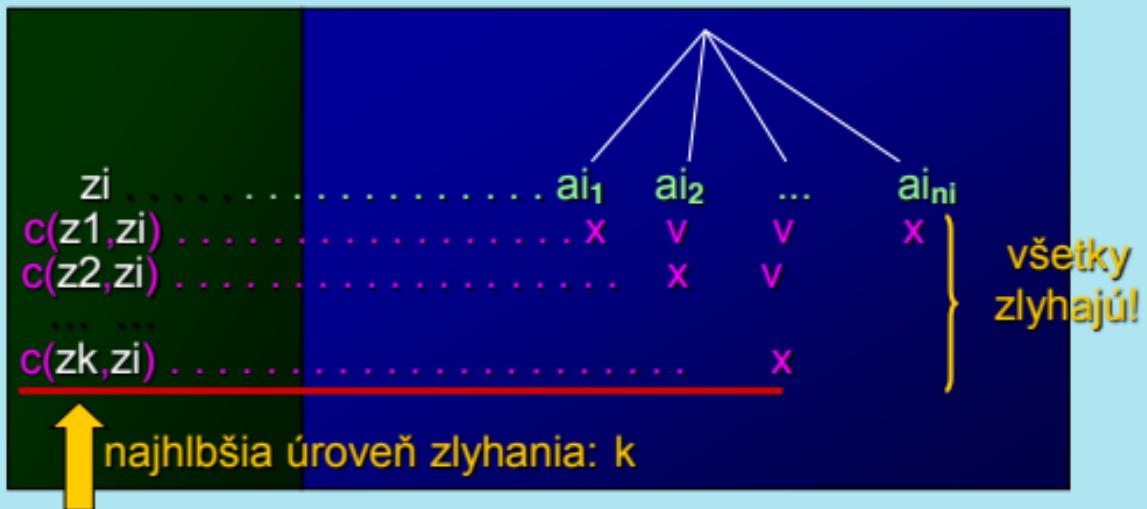


mlátenie:

iný výskyt



spätné skákanie (backjumping) (Gaschnig'78)



⌚ princíp:

If všetky priradenia premennej zi zlyhajú,
and $c(zk, zi)$ je najlbšie ohraničenie spôsobujúce zlyhanie
Then backjump kvôli zmeneniu priradenia zk

algoritmus spätného skákania

BackJ(híbka, out: spätnýskok)

For k = 1 to nhíbka do

Zhíbka := $a_{híbka,k}$;

preveruj všetky ohraničenia $c(z_i, z_{híbka})$

s $1 \leq i < híbka$ pokial nejaké nesplnené

$híbkapreverenia_k :=$ najhlbšia preverená úroveň i;

If žiadne nesplnené then

If híbka = n then

return(z1, z2, ..., zn)

Else

híbka := híbka + 1;

BackJ(híbka, spätnýskok);

híbka := híbka - 1;

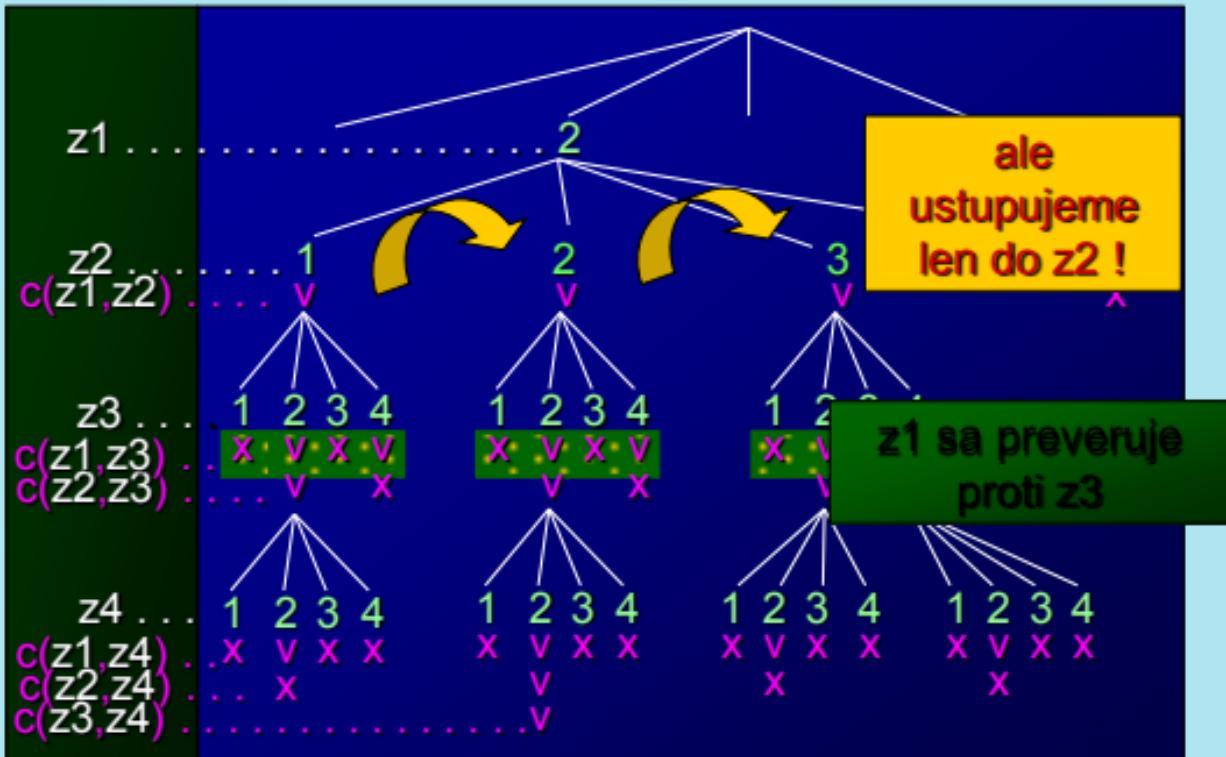
If spätnýskok < híbka then return;

End-For

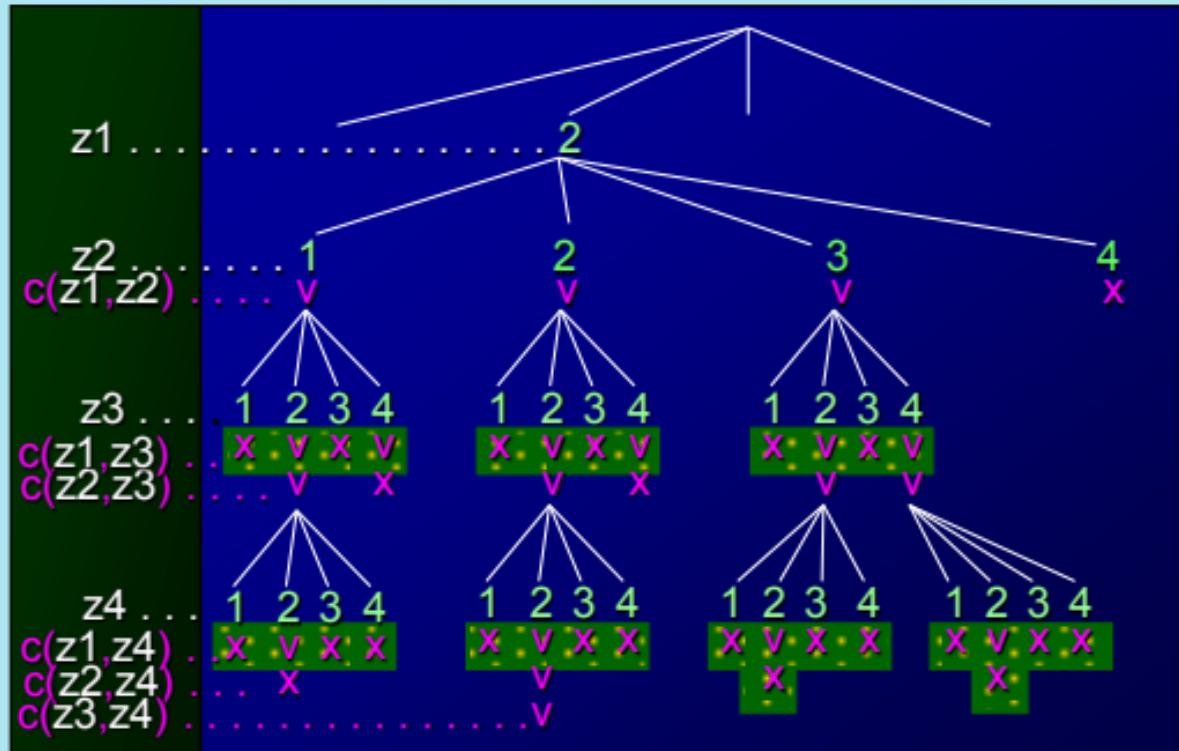
spätnýskok := max(híbkapreverenia_k)

Zhíbka . . . $a_{híbka,l}$
 $c(z_1, z_{híbka}) . . . v$
 $c(z_2, z_{híbka}) . . . v$
 $c(z_m, z_{híbka}) . . . x$
 $híbkapreverenia_l = m$

ďalšie nadbytočné preverovania:



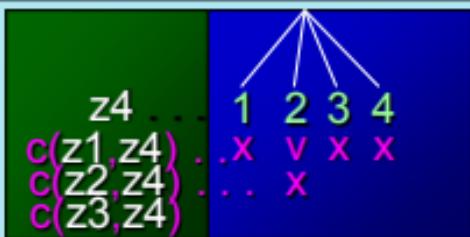
nastáva veľmi často:



zahodenie vs nadbytočné previerky:

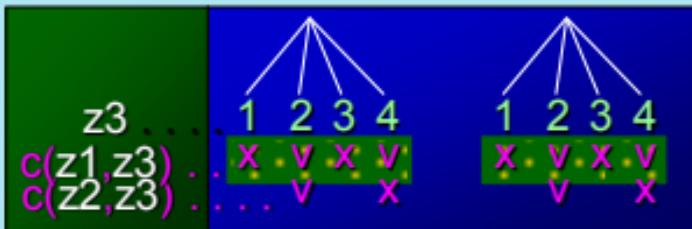
◎ zahodenie:

len keď zlyhá celý blok previerok



◎ nadbytočné previerky:

aj pre úspešné previerky a pre previerky len v jednom riadku



vyvarovanie sa nadbytočným previerkam:

◎ 2 prístupy:

1. Tabelácia:

- ➔ generovanie liem (ukladanie výsledkov previerok do tabuľky)
- + aplikovanie liem (vyhľadávanie v tabuľke pri nových previerkach)

◎ do istej miery zvyšuje rýchlosť

➔ ale nevyvaruje sa naozaj nadbytočným previerkam

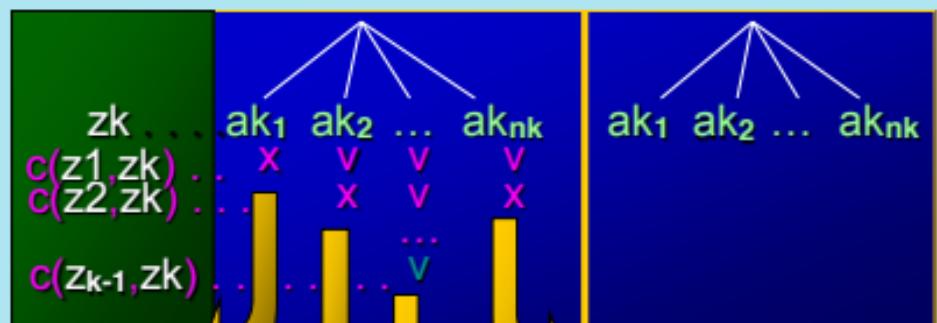
◎ zvyšuje nároky na pamäť

2. Spätné značkovanie (BACKMARKING):

- ◎ ~ úplná úspora času (nie sú nadbytočné previerky)
- ◎ len obmedzené zvýšenie pamäťovej rézie (2 polia)

spätné značkovanie (backmarking) (Gaschnig '77):

◎ 2 polia:



$\text{híbkaprevierky}_{z_k,1} = 1$

$\text{híbkaprevierky}_{z_k,n_k} = 2$

$\text{híbkaprevierky}_{z_k,2} = 2$

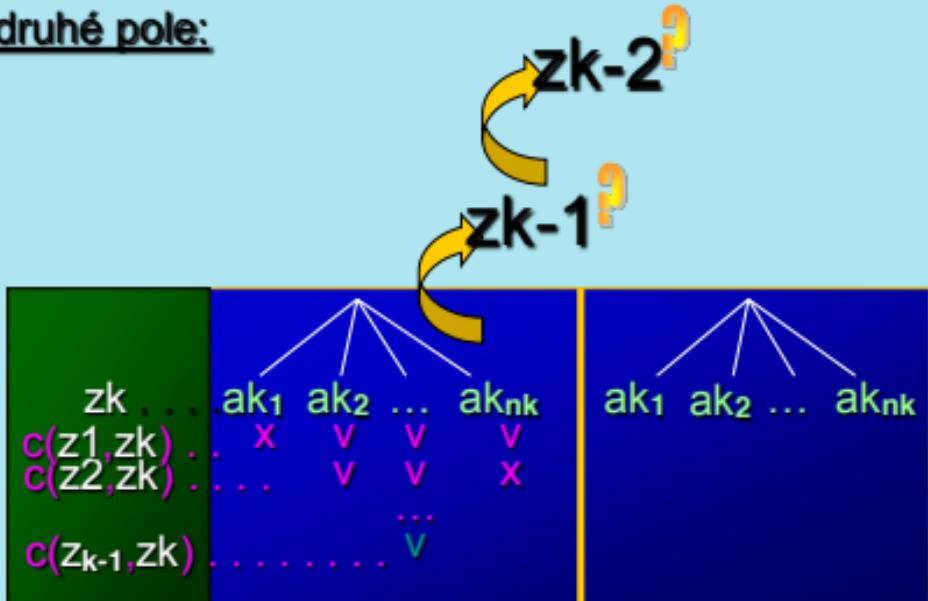
$\text{híbkaprevierky}_{z_k,\dots} = k-1$

◎ $\text{híbkaprevierky}(k,l):$

= $\text{híbkaprevierky}_{z_k, l}$.

spätné značkovanie

◎ druhé pole:



◎ Backup(k):

= najnižšia úroveň, ku ktorej sme sa vrátili medzi návštěvami týchto 2 blokov pre zk.

vlastnosť funkcie híbkaprevierky_{k,l}

- ◎ If $\text{híbkaprevierky}_{k,l} < k-1$:

$z_k \dots a_{k,l}$	↑
$c(z_1, z_k) \dots v$	
$c(z_2, z_k) \dots v$	
\dots	
$c(z_i, z_k) \dots x$	

výsledok kontroly platnosti poslednej podmienky $c(z_l, z_k)$ musel byť neúspech, inak by kontrolovanie pokračovalo!

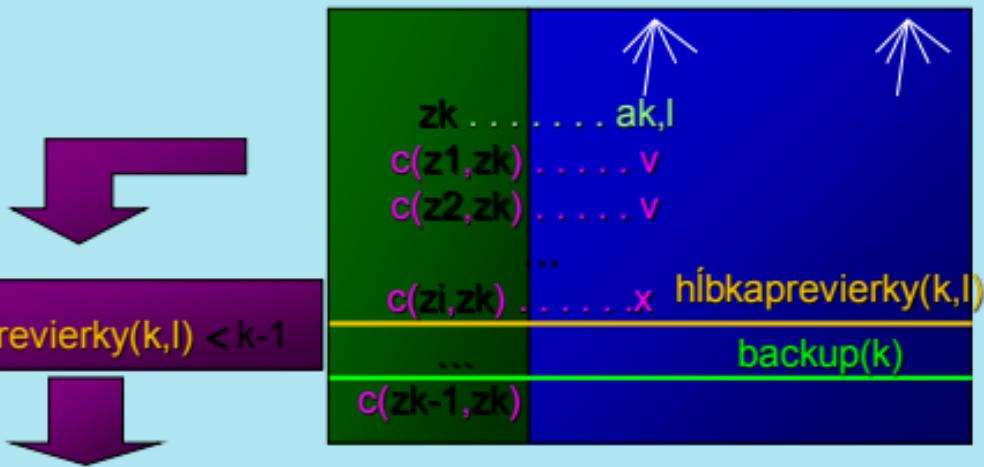
- ◎ If $\text{híbkaprevierky}_{k,l} = k-1$:

$z_k \dots a_{k,l}$	
$c(z_1, z_k) \dots v$	
$c(z_2, z_k) \dots v$	
\dots	
$c(z_{k-1}, z_k) \dots x/v$	

výsledok kontroly platnosti poslednej podmienky $c(z_{k-1}, z_k)$ mohol byť úspech alebo neúspech.

vlastnosti híbkaprevierky vs Backup (1):

- ◎ If $\text{híbkaprevierky}(k,l) < \text{backup}(k)$:



hodnota $a_{k,l}$ pre premennú z_k spôsobila neúspech minule a spôsobí neúspech aj teraz (v tej istej híbke)

vlastnosti híbkaprevierky versus Backup (2):

- ④ $\text{híbkaprevierky}(k,l) \geq \text{backup}(k)$:



zk	ak,l	↑
c(z1,zk)	v	
c(z2,zk)	v	
...		
c(z _i ,zk)	v	backup(k)
...		
c(z _j ,zk)	v/x	híbkaprevierky(k,l)

všetky preverovania premenných zm s m nižším ako $\text{backup}(k)$ uspeli minule a uspejú znova.

algoritmus spätného značkovania:

BackM(híbka, out: híbkaprevierky, backup)

For k= 1 to n_{híbka} do

Z_{híbka} := a_{híbka,k} ;

If híbkaprevierky(híbka,k) ≥ backup(híbka)

použila sa vlastnosť 1

preveruj všetky ohraničenia c(z_i, Z_{híbka}) s
backup(híbka) $\forall i < \text{híbka}$ pokial' nejaké nesplnené:

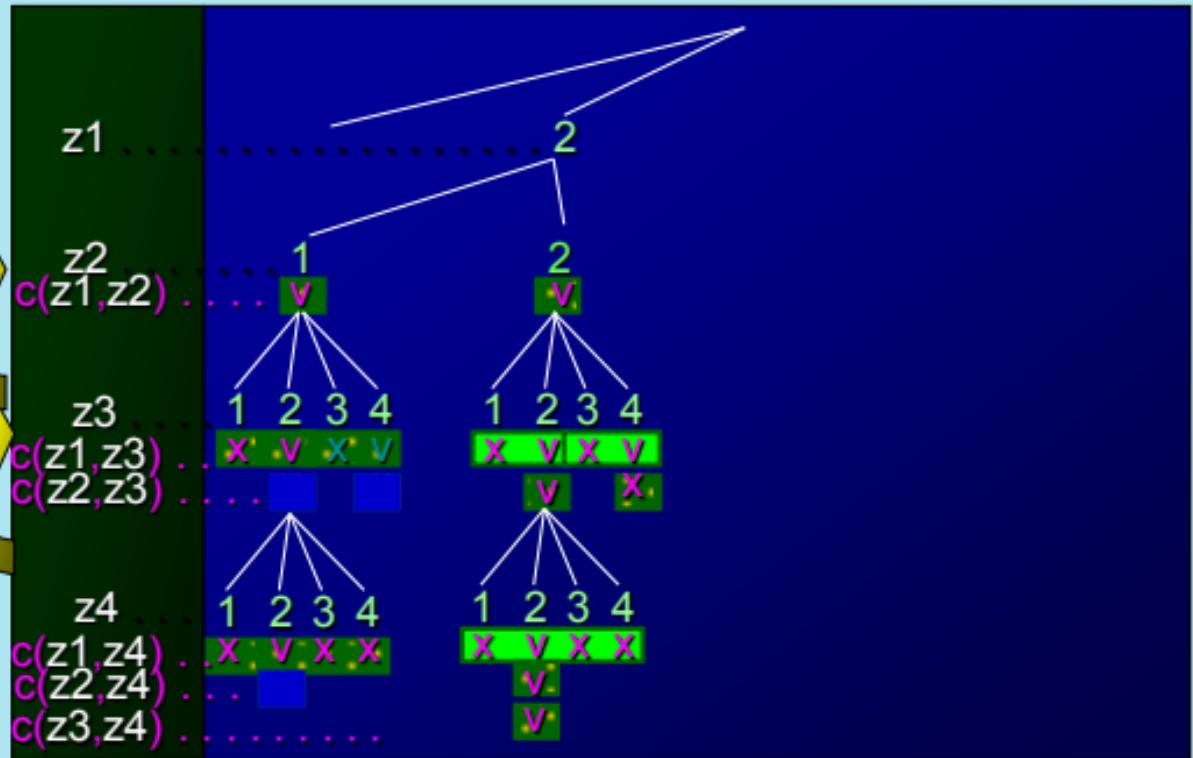
použila sa vlastnosť 2

If žiadne nesplnené then

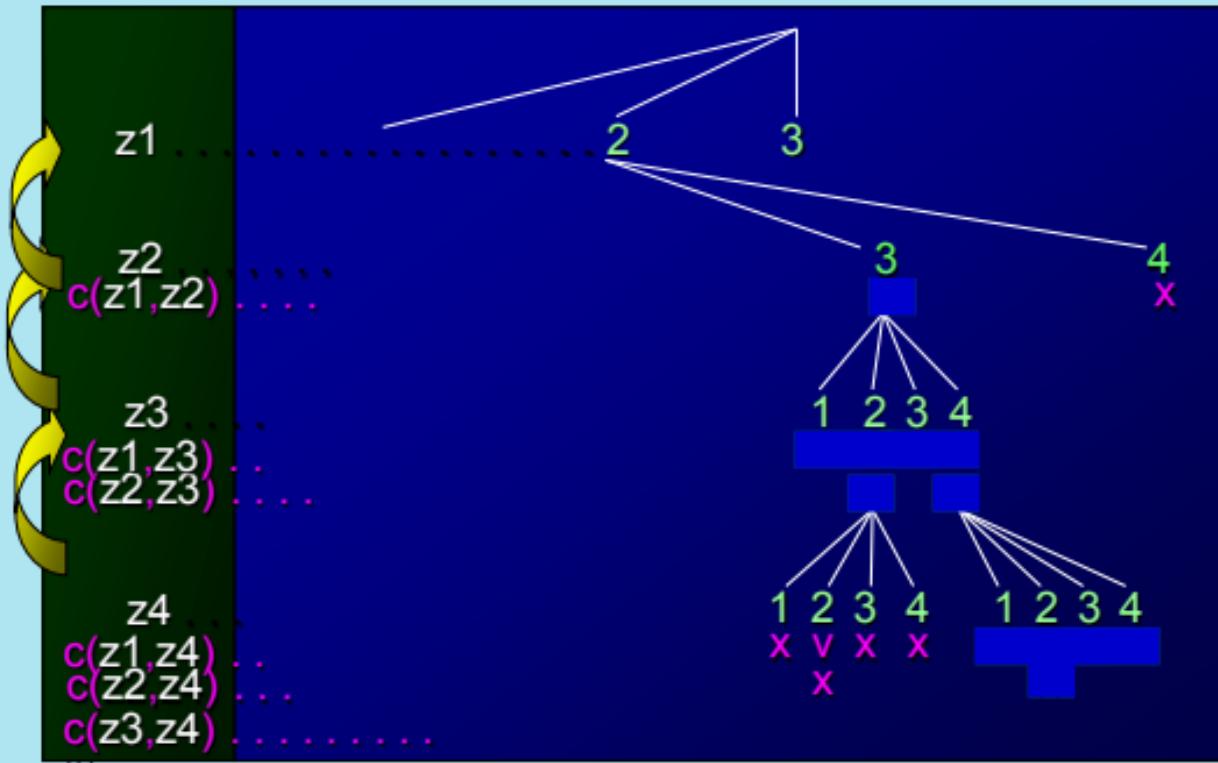
...

End-For

spätné značkovanie pracuje:



pri omnoho väčšom strome:



Diskusia a výsledky:

Poznámka: začiatočné hodnoty pre híbkaprevierky(k,l) a backup(k) sú 1

◎ Experimentálne výsledky:

→ pre p-4-dámy

	BT	BJ	BM
uzly	29	27	29
previerky	160	139	90

◎ celkovo:

→ spätné značkovanie = najlepší algoritmus (nie absolútne)

◎ kombinovať spätné skákanie a značkovanie? Áno!

→ ale vylepšenia sa neskombinujú.

Inteligentné ustupovanie (Bruynooghe - Pereira '80)

- ◎ =~ závislosťou riadené ustupovanie (Dependency-directed Backtracking) (Doyle)
 - ◎ ≠ jeden špecifický algoritmus

omnoho viac ide o: generická stratégia, je základom mnohých rôznych algoritmov

◎ základná myšlienka:

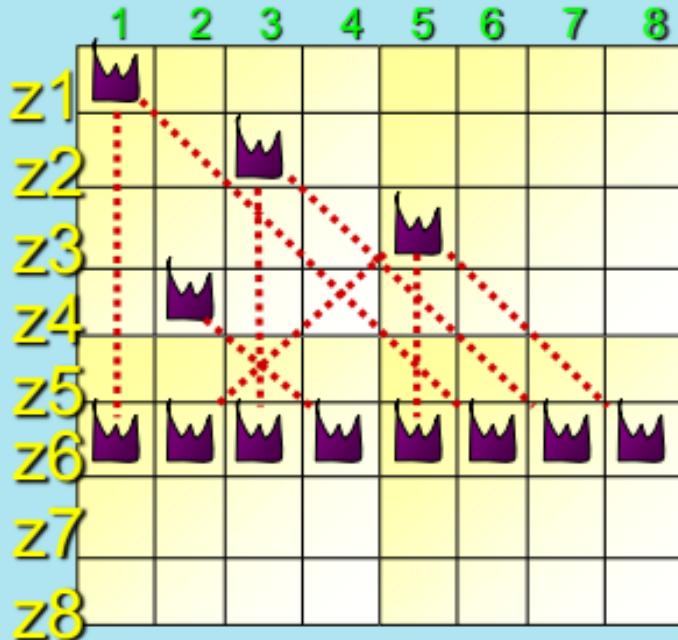
počas zostrojovania stromu odvoď a zapamätaj ne-dobroty “no-goods”,

použi ne-dobroty na zlepšenie priebehu spätného chodu.

◎ ne-dobrota no-good je:

množina priradení hodnôt premenným, ktoré nemôžu byť súčasne časťou nijakého riešenia.

príklad: 8-dám:

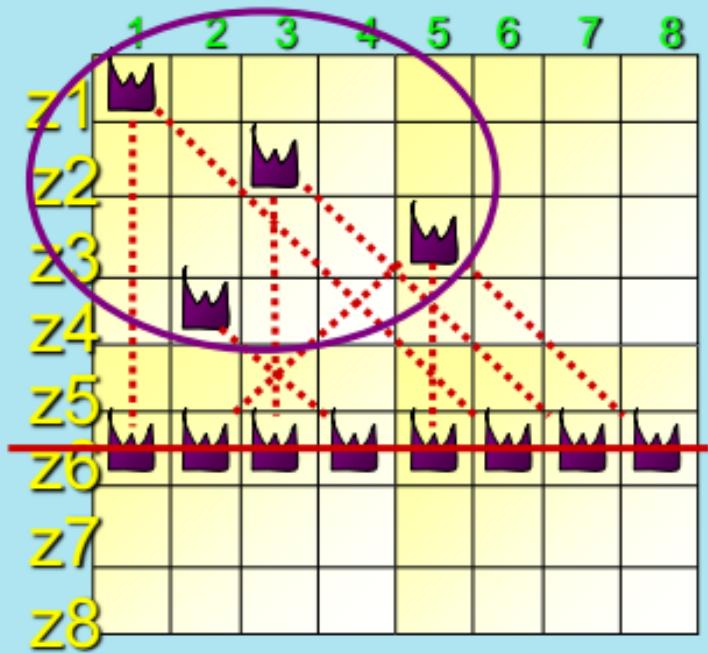


niekoľko jednoduchých
ne-dobrôt, porušujúcich
1 ohraničenie:

- {z1 = 1, z6 = 1}
- {z3 = 5, z6 = 2}
- {z2 = 3, z6 = 3}
- {z4 = 2, z6 = 4}
- {z3 = 5, z6 = 5}
- {z1 = 1, z6 = 6}
- {z2 = 3, z6 = 7}
- {z3 = 5, z6 = 8}

odvodená ne-
dobrota: {z1 = 1, z2 = 3, z3 = 5, z4 = 2 }

dôvod:

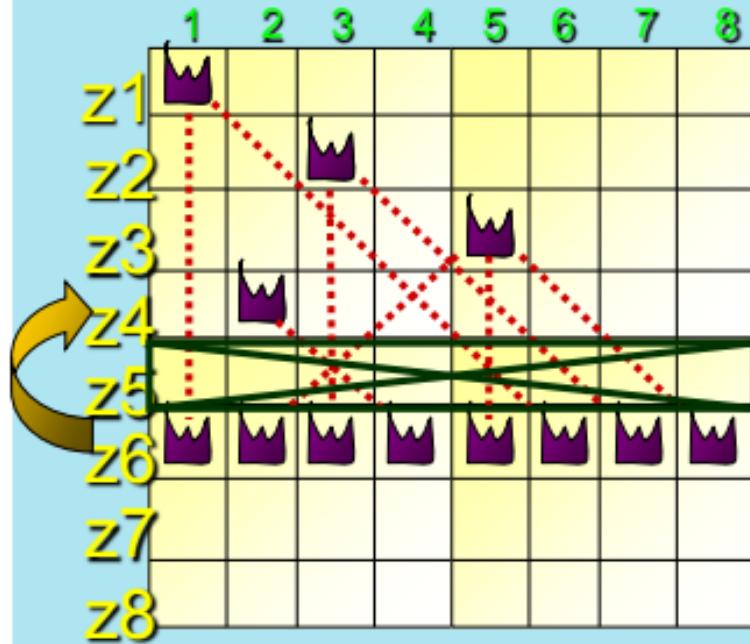


preto, lebo tieto 4
priradenia už
nenechávajú žiadnu
možnosť umiestnenia
z6 !

odvodená ne-
dobrota:

$$\{ z1 = 1, z2 = 3, z3 = 5, z4 = 2 \}$$

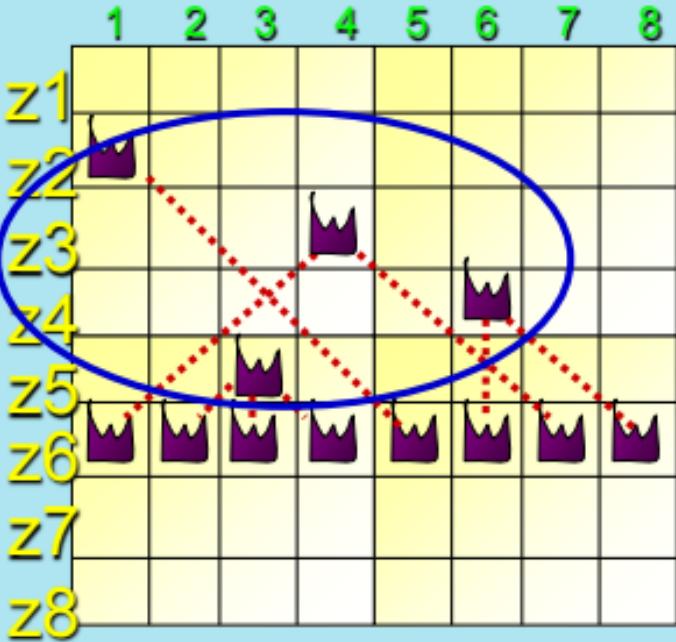
prípad spätného skoku:



nie je dôvod
ustupovania cez
hodnoty premennej **z5**,
protože dôvod zlyhania
(ne-dobrota) nezahŕňa
z5 !

Ustupuj cez najhlbšiu
premennú účastnú na
dôvode zlyhania (**z4**).

iný príklad:



niekoľko jednoduchých

ne-dobrôt

- { $z_3 = 4, z_6 = 1$ }
- { $z_5 = 3, z_6 = 2$ }
- { $z_5 = 3, z_6 = 3$ }
- { $z_5 = 3, z_6 = 4$ }
- { $z_2 = 1, z_6 = 5$ }
- { $z_4 = 6, z_6 = 6$ }
- { $z_3 = 4, z_6 = 7$ }
- { $z_4 = 6, z_6 = 8$ }

odvodená ne-dobrota:

$$\{z_2 = 1, z_3 = 4, z_4 = 6, z_5 = 3\}$$

ošetuje nadbytočnosť ako u spätného značenia:

	1	2	3	4	5	6	7	8
z1								
z2								
z3								
z4								
z5								
z6								
z7								
z8								

The diagram shows a 9x9 grid with columns labeled 1-8 and rows labeled z1-z8. Purple crowns are placed at (z2, 1), (z3, 5), (z4, 4), (z5, 2), (z6, 1-8), and (z7, 3). Red dotted lines indicate constraints: a diagonal from (z2, 1) to (z8, 8), a vertical line from (z3, 5) to (z7, 5), and a horizontal row from (z4, 4) to (z6, 4).

odvodená ne-dobrota:

{ $z_2=1$, $z_3=4$, $z_4=6$, $z_5=3$ }

vždy, keď sa táto kombinácia vynori opäť (pre iné hodnoty z_1): nepreveruj, ustupuj!

Dynamické preusporiadanie prehľadávania

- = ľubovoľný algoritmus ustupovania
- +
- dynamický výber poradia premenných v strome

◎ Účel:

Znížiť veľkosť stromu ako dôsledok “princípu najprv-neúspech”

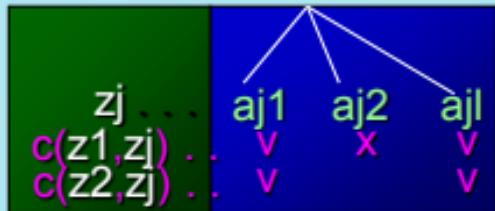
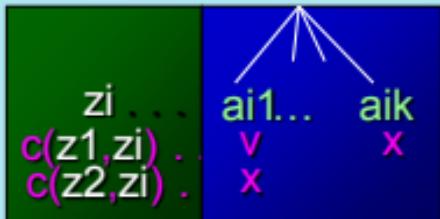
◎ princíp najprv-neúspech:

ak dôsledkom priradenia hodnoty premennej z_i je, že dôjde *pravdepodobnejšie* k zlyhaniu než priradením hodnoty premennej z_j :

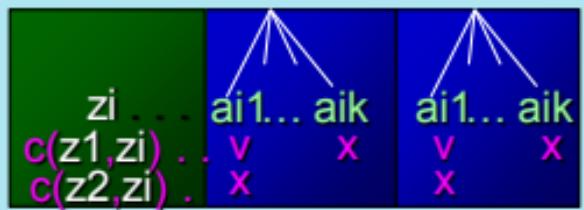
tak: najprv priraď premennej z_i .

Účinok?

- 1. predpokladajme, že náš odhad bol správny...



hotovo



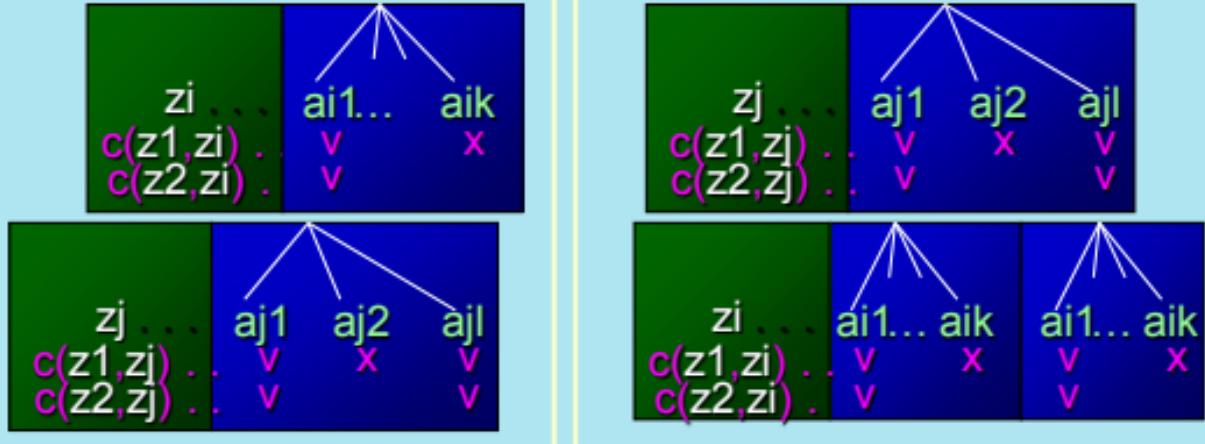
◎ ... takže získavame v každom prípade

menší strom hľadania !

Účinok (2)?

- 2. predpokladajme, že náš odhad bol iba čiastočne správny ...

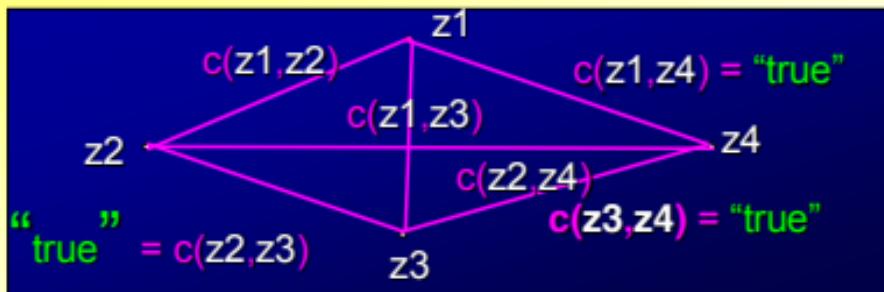
Priradenie premennej zi nespôsobuje
neúspech, ale má menej úspechov!



niekoľko všeobecných heuristik prístupu najprv zlyhanie:

→ Vyber najprv premennú s najmenšou doménou – znižuje faktor vetvenia (menej možností -> menej úspechov)

→ Zvoľ najprv premennú s najmenším počtom netriviálnych ohraničení:



⊕ Heuristika sa zakladá na danom probléme.