STAVOVÁ REPREZENTACE PROSTŘEDÍ A PROHLEDÁVÁNÍ SP

doc. RNDr. Petr Tučník, Ph.D.

petr.tucnik@fpf.slu.cz

OBSAH PŘEDNÁŠKY

- Základní pojmy
- Neinformované prohledávání
  - Prohledávání do šířky
  - Prohledávání do hloubky
- Informované prohledávání
  - Uspořádané prohledávání
  - A* algoritmus

ZÁKLADNÍ POJMY

INTELIGENTNÍ AGENTY

- O inteligenci uvažujeme zpravidla v souvislostech s tím jak efektivně agent v interakci s prostředím koná.
- Abychom mohli mluvit o smysluplné interakci, je zapotřebí vytyčit **cíl**, který agent bude sledovat.
- **Vymezení cíle** je jedním z prvních kroků při řešení problému.

Nastínění problému cestovatele (1)

- Uvažujme situaci, kdy se náš agent nalézá v Rumunském městě Arad na dovolené.
- Hodnocení agentovy úspěšnosti je závislé na mnoha faktorech, jmenovitě na tom, do jaké míry se mu podařilo opálit se, zdokonalit rumunštinu, navštěvovat památky, užívat si noční život…

Nastínění problému cestovatele (2)

- Úkol tedy může být i poměrně složitý a je třeba zvažovat velké množství pro a proti.
- Agent má navíc zpáteční letenku na let z Bukurešťi na zítřejší den.
- Je tedy rozumné, že si agent určí za cíl stihnout tento let. Všechna řešení, která by nedovolila let stihnout, jsou z rozhodování vyloučena, čímž se rozhodovací problém výrazně zjednoduší.

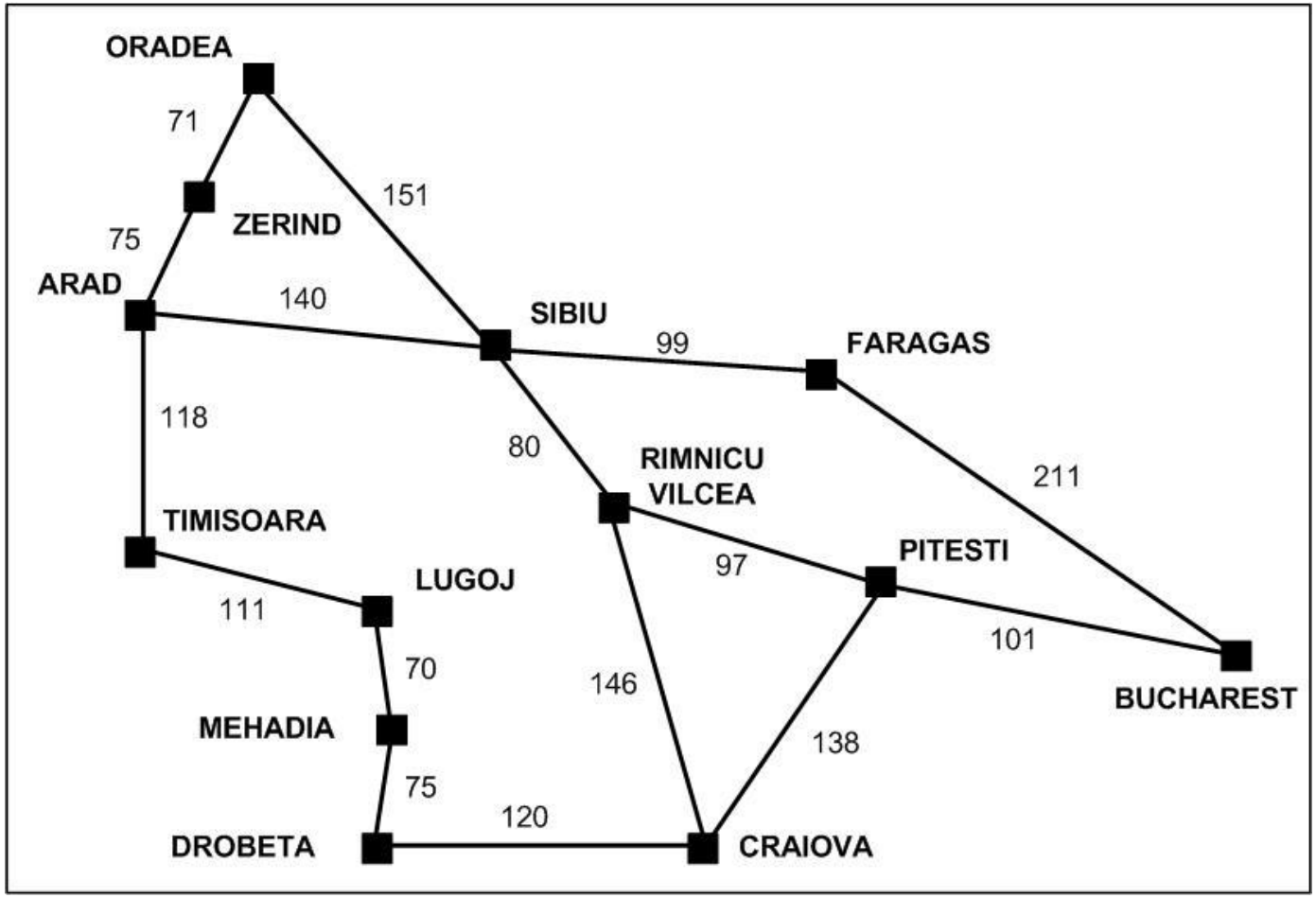
FORMULACE PROBLÉMU (1)

- Uvažujme o **cíli** jako o takové **množině stavů** světa, ve kterých je náš cíl naplněn.
- Úkolem agenta je nalézt takovou posloupnost akcí, která jej dostane do cílového stavu.
- Je třeba vymezit množinu akcí, které má agent k dispozici a zvážit úroveň detailu, kterou použijeme.

FORMULACE PROBLÉMU (2)

- Příliš detailní množina akcí (např. „pohnout levou nohou o tři centimetry dopředu“) by naši situaci neřešila, jelikož by řešení obsahovalo příliš mnoho kroků a nejistoty.
- **Formulace problému** je proces rozhodování o tom, jaké akce a stavy budeme zvažovat při řešení daného cíle.
- Pro naše potřeby nyní uvažujme akce jako řízení auta z jednoho města do druhého a množina stavů bude odpovídat pobytu v jednotlivých městech.

CESTOVNÍ MAPA*



*Pozn. názvy jsou v angličtině

PROVEDENÍ AKCE - 2 SITUACE

- Uvažujme dvě možné situace:
  - Agent nemá žádné informace o tom, která z možných akcí je nejlepší, protože o stavu, který je důsledkem jeho akce nemá dostatek informací (v našem příkladu nemá mapu).
  - Agent má k dispozici informace o stavech, do kterých se může dostat a akcích, které v nich může učinit. To umožňuje zvažovat následné další kroky a vytvořit **sekvenci akcí**, následujících za sebou.

POZNÁMKA

- Obecně lze říci, že *agent s několika bezprostředními možnostmi neznámé hodnoty se může rozhodnout co udělá předběžným zkoumáním rozdílných možných sekvencí akcí, které vedou ke stavům se známou hodnotou, a výběrem nejlepší takové sekvence akcí*.

PROHLEDÁVÁNÍ

- Hledání takové sekvence akcí se nazývá **prohledávání** (stavového prostoru). Je to velmi důležitý postup vhodný pro řešení úloh, které nelze přímo řešit použitím předem známých výpočetních postupů.
- **Řešení** představuje sekvence akcí.
- Jakmile je nalezeno řešení, může agent přistoupit k provádění akcí. To se nazývá **prováděcí fáze**.
- Následuje algoritmus jednoduchého agenta řešícího problém.

ALGORITMUS AGENTA ŘEŠÍCÍHO JEDNODUCHÝ PROBLÉM (1)

function SIMPLE-PROBLEM-SOLVING AGENT(*percept*) **returns** an action

inputs: *percept*, a percept

static: *seq*, an action sequence, initially empty

state, some description of the current world state

goal, a goal, initially null

problem, a problem formulation

state ← UPDATE-STATE(*state, percept*)

if *seq* is empty **then do**

goal ← FORMULATE-GOAL(*state*)

problem ← FORMULATE-PROBLEM(*state, goal*)

seq ← SEARCH(*problem*)

action ← FIRST(seq)

seq ← REST(seq)

return action

ALGORITMUS AGENTA ŘEŠÍCÍHO JEDNODUCHÝ PROBLÉM (2)

- PSA zformuluje cíl a problém.
- Hledá sekvenci akcí, které jsou řešením problému.
- Provádí akce jednu po druhé.
- Po dokončení zformuluje další cíl a začíná znovu.
- Při provádění sekvence akcí ignoruje vjemy – předpokládá se, že nalezené řešení bude vždy fungovat.

POZNÁMKY K ALGORITMU

- Předpokládáme, že prostředí je **statické** (static) – formulace a řešení problému nevěnuje pozornost změnám, které by mohly nastat v prostředí.
- Předpokládáme také, že agent zná svůj **počáteční stav** (initial state).

VLASTNOSTI PROSTŘEDÍ

- Prostředí je **pozorovatelné**, je-li agent schopen rozpoznat v jakém stavu se nachází.
- Jestliže zvažujeme přístup přinášející různé alternativní cesty řešení, předpokládáme, že prostředí může být popsáno **diskrétně**.
- Velmi důležitým předpokladem je také to, že uvažované prostředí bereme jako **deterministické**.

DOBŘE DEFINOVANÉ PROBLÉMY A JEJICH ŘEŠENÍ (1)

- Problém může být formálně definován čtyřmi komponentami:

1. **Počáteční stav** – stav, ve kterém se agent nachází na začátku řešení úlohy. V našem příkladě by takový počáteční stav mohl být popsán *In(Arad)*.

DOBŘE DEFINOVANÉ PROBLÉMY A JEJICH ŘEŠENÍ (2)

2. Popisem **možných akcí**, které má agent k dispozici. Obvyklá formulace používá termínu **následnická funkce** (successor function). Mějme dán určitý stav *x*, pak SUCCESSOR-FN(*x*) vrací množinu uspořádaných dvojic <*akce, následník*>, kde akce je jedna z akcí přípustných ve stavu *x* a každý následník je stavem, kterého může být dosaženo z *x* aplikací akce.

- Např. pro stav In(Arad) by následnická funkce vrátila hodnoty:

  {<Go(Sibiu), In(Sibiu)>, <Go(Timisoara), In(Timisoara)>, <Go(Zerind), In(Zerind)>}

DOBŘE DEFINOVANÉ PROBLÉMY A JEJICH ŘEŠENÍ (3)

- Počáteční stav a následnická funkce implicitně definují **stavový prostor** problému – množinu všech stavů dosažitelných z počátečního stavu.
- Stavový prostor může být reprezentován grafem, ve kterém jeho vrcholy představují stavy a hrany mezi vrcholy odpovídají akcím.
- **Cesta** ve stavovém prostoru je sekvence stavů propojených sekvencí akcí.

DOBŘE DEFINOVANÉ PROBLÉMY A JEJICH ŘEŠENÍ (4)

3. **Testování cílového stavu**, určuje, zda je daný stav cílový. Takových stavů může být více. V našem případě je množina cílových stavů jednoprvková {In(Bucharest)}. Cílový stav může být vymezen i abstraktní vlastností – např. šachmat v šachu.

DOBŘE DEFINOVANÉ PROBLÉMY A JEJICH ŘEŠENÍ (5)

4. **Cena cesty** (ve stavovém prostoru) je funkce, která každé cestě přiřazuje numerickou hodnotu. Agent volí takovou funkci ceny, která odpovídá jeho měření výkonnosti. **Cena kroku** provedení akce (předpokládejme nezápornou hodnotu) *a* pro přechod ze stavu *x* do stavu *y* se značí *c(x,a,y).* V našem příkladu je cenou kroku vzdálenost mezi jednotlivými městy.

DOBŘE DEFINOVANÉ PROBLÉMY A JEJICH ŘEŠENÍ (6)

- **Řešení problému** je cesta mezi počátečním stavem a koncovým stavem. Kvalita řešení se hodnotí podle funkce ceny, který každé řešení přiřazuje hodnotu. **Optimální řešení** je takové řešení, které má ze všech řešení nejnižší cenu.

ABSTRAKCE

- Popis světa může obsahovat velké množství detailů. Informace, které jsou pro řešení problému irelevantní, je vhodné z reprezentace problému vypustit. Proces vynechání detailů z reprezentace problému se nazývá **abstrakce**.
- Abstrakce je **validní**, jestliže můžeme expandovat libovolné abstraktní řešení na řešení v detailnějším světě.

KRITERIA HODNOCENÍ VÝKONU PRO STRATEGIE PROHLEDÁVÁNÍ

- **Úplnost** – nalezne algoritmus vždy řešení, pokud řešení existuje?
- **Optimálnost** – nalezne strategie vždy optimální řešení (takové, kde neexistuje jiné řešení s menšími náklady)?
- **Časová složitost** – kolik času zabere najít řešení?
- **Prostorová složitost** – kolik paměti zabere provedení prohledávání?

NEINFORMOVANÉ PROHLEDÁVÁNÍ

NEINFORMOVANÉ PROHLEDÁVÁNÍ

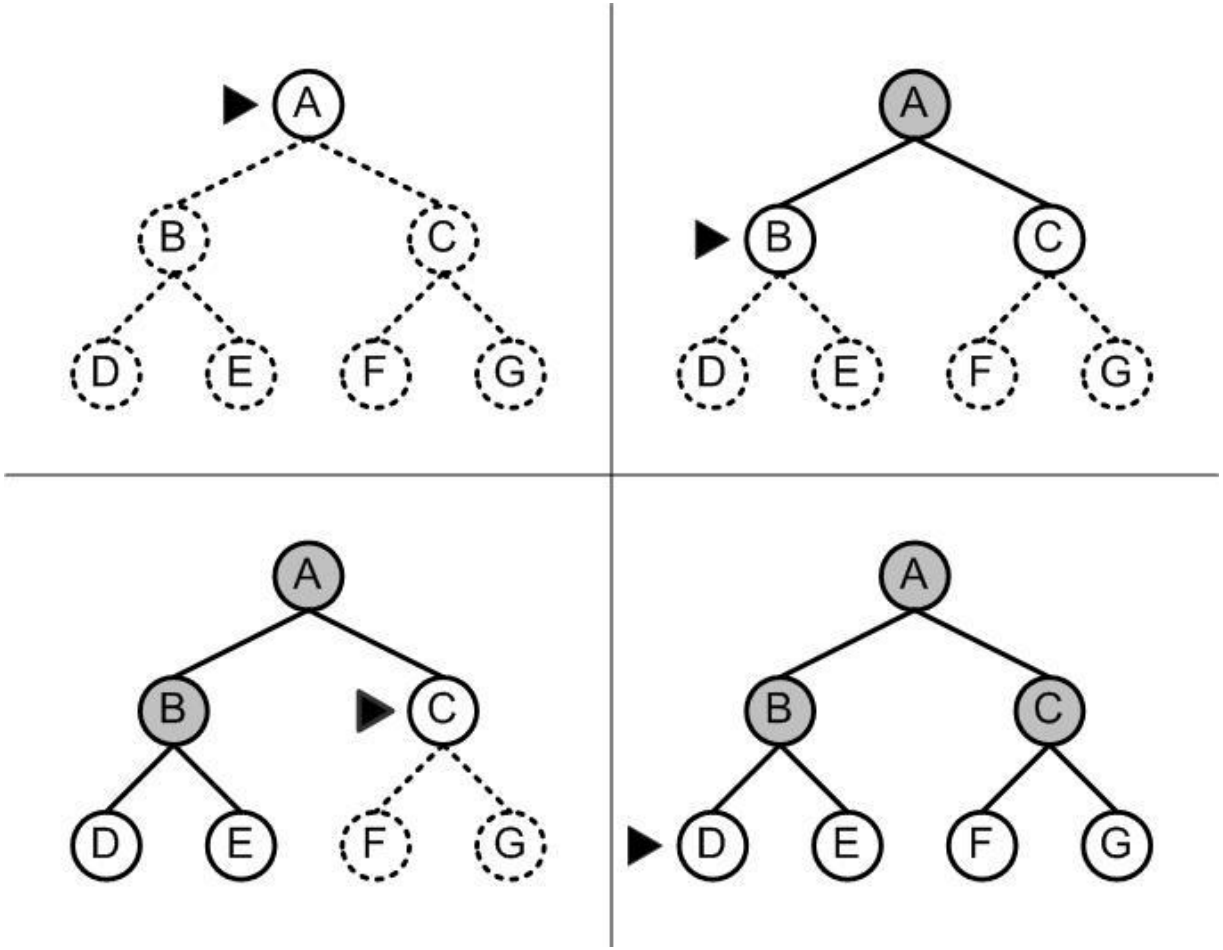
- Někdy také nazývané slepé prohledávání (*blind search*).
- Pro prohledávání stavového prostoru nedisponujeme žádnými dodatečnými informacemi.
- Tyto algoritmy dokáží pouze generovat následný uzel a rozlišit cílový a necílový stav.
- Dělíme je dle pořadí, v němž jsou expandovány uzly grafu stavového prostoru na:
  - Prohledávání do šířky
  - Prohledávání do hloubky.

PROHLEDÁVÁNÍ DO ŠÍŘKY*

- Jednoduchá prohledávací strategie expanduje kořenový uzel, pak postupně všechny jeho následníky a pak jejich následníky atd.
- Cílový stav, který je v nejmenší hloubce stromu reprezentujícího stavový prostor je vybrán jako řešení.
- To neznamená, že je vždy optimální (záleží na ohodnocení jednotlivých hran grafu). Můžeme vybírat uzly, které jsou ne v nejmenší hloubce, ale s nejlevnějšími náklady na dosažení – pak je nalezen optimální stav.

*Pozn.: angl. "Breadth-first search"

PROHLEDÁVÁNÍ DO ŠÍŘKY – DEMONSTRACE NA BINÁRNÍM STROMU



ALGORITMUS PROHLEDÁVÁNÍ DO ŠÍŘKY (1)

1. Zapiš počáteční stav do seznamu OPEN (seznam neexpandovaných stavů), seznam CLOSED (seznam již expandovaných stavů) je prázdný. Je-li počáteční stav stavem cílovým, ukonči prohledávání.
2. Pokud je seznam OPEN prázdný, řešení neexistuje, ukonči prohledávání.
3. Vymaž první stav *j v seznamu OPEN a zapiš tento stav do* seznamu CLOSED.
4. Expanduj stav *j. Pokud tento stav nemá následovníky nebo* všichni následovníci byli již expandováni (jsou v seznamu CLOSED), pokračuj krokem algoritmu č. 2.

Dle: Mařík a kol., Umělá inteligence 1, Academia, 1993

ALGORITMUS PROHLEDÁVÁNÍ DO ŠÍŘKY (2)

5. Napiš všechny následovníky stavu *j, kteří nejsou v seznamu* CLOSED, na konec seznamu OPEN.
6. Pokud některý z následovníků *j je cílovým stavem, řešení bylo* nalezeno, ukonči prohledávání. Jinak pokračuj krokem č. 2.

VLASTNOSTI PROHLEDÁVÁNÍ DO ŠÍŘKY (1)

- Algoritmus funguje dobře, je-li cena cesty neklesající funkcí hloubky uzlu (např. mají-li všechny akce stejnou cenu).
- Prohledávání je úplné (za předpokladu, že je faktor větvení **b** konečný).
- **Faktor větvení b** – maximální počet následovníků libovolného uzlu.

VLASTNOSTI PROHLEDÁVÁNÍ DO ŠÍŘKY (2)

- Před aplikací je třeba zvážit časové a paměťové nároky algoritmu.
- Uvažme situaci – máme stavový prostor, ve kterém každý uzel má *b* následníků. Tj. kořenový uzel má *b* uzlů, z každý z nich má dalších *b* uzlů atd. Tj. na první úrovni generuje náš prostor b uzlů, na druhé b^2, na třetí b^3 atd.

Vlastnosti prohledávání do šířky (3)

- Je-li řešení v hloubce *d*, budeme muset v nejhorším případě prohledat

  $b+b^2+b^3+\ldots+b^d+(b^{d+1}-b) = O(b^{d+1})$

  uzlů (cílový uzel již další uzly na úrovni d+1 generovat nebude, proto b^{d+1}-b na konci).

- Každý uzel musí být uchován v paměti, protože je buď listem nebo nějakým jeho předchůdcem.

Náročnost prohledávání do šířky

Hloubka	Uzly	Čas	Paměť
2	1 100	0.11 sec	1 megabyte
4	111 100	11 sec	106 megabyte
6	10^7	19 min	10 gigabyte
8	10^9	31 hod	1 terabyte
10	10^{11}	129 dnů	10 terabyte
12	10^{13}	35 let	1 petabyte
14	10^{15}	3 523 let	1 exabyte

Předpoklady: faktor větvení (branching) b=10; 10 000 uzlů/sec; 1000 bytů/uzel.

DŮSLEDKY

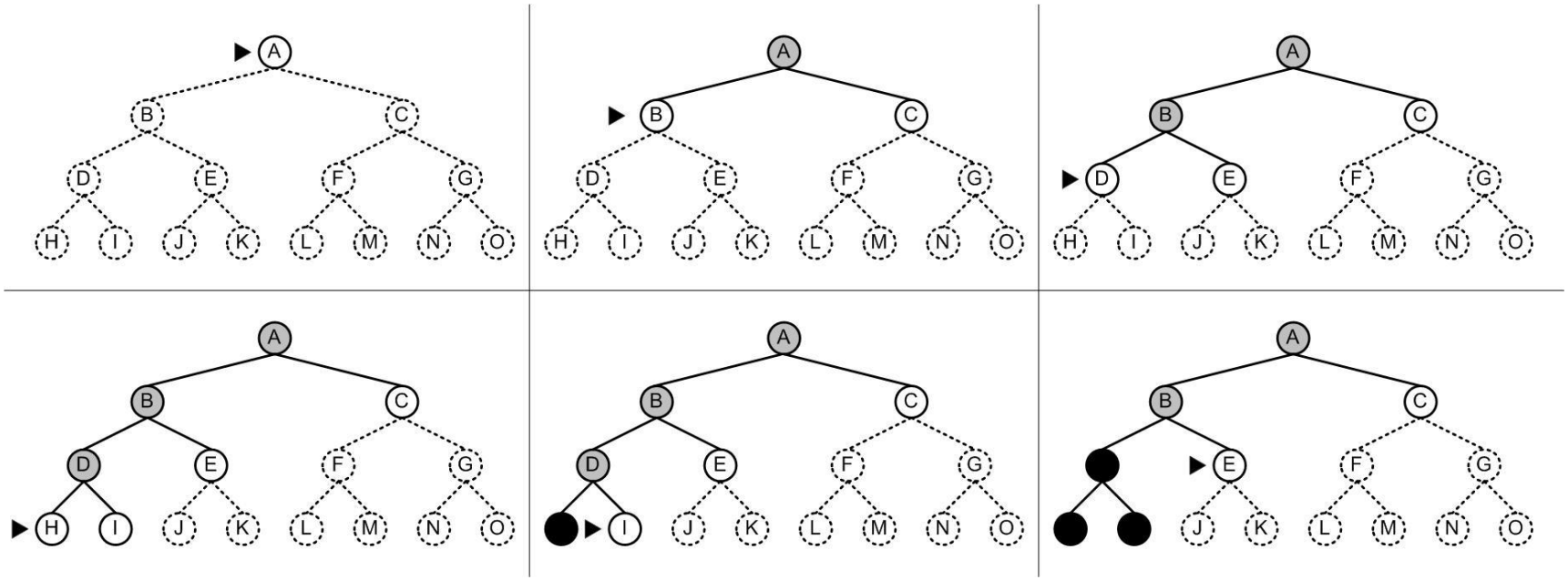
- Nároky na paměť jsou u prohledávání do šířky větším problémem než je čas provádění vyhledávání.
- Obecně lze říci, že hledání u problémů s exponenciální složitostí nemůže být řešeno neinformovanými metodami v jiných než minimálních případech.

PROHLEDÁVÁNÍ DO HLOUBKY*

- Vždy je expandován uzel s největší hloubkou.
- Hledání se rychle dostává na nejnižší úroveň grafu. Není-li koncový uzel cílový, vrací se algoritmus zpět k nejbližšímu uzlu, který má neprozkoumané následovníky.
- Části prohledávaného prostoru, kde nebylo nalezeno řešení, se mohou průběžně uvolňovat z paměti.

*Pozn.: angl. "Depth-first search"

PROHLEDÁVÁNÍ DO HLOUBKY – DEMONSTRACE NA BINÁRNÍM STROMU



ALGORITMUS PROHLEDÁVÁNÍ DO HLOUBKY (1)

1. Zapiš počáteční stav do seznamu OPEN (seznam neexpandovaných stavů), seznam CLOSED (seznam již expandovaných stavů) je prázdný. Je-li počáteční stav stavem cílovým, ukonči prohledávání.
2. Pokud je seznam OPEN prázdný, řešení neexistuje, ukonči prohledávání.
3. Vymaž první stav *j v seznamu OPEN a zapiš tento stav do* seznamu CLOSED.
4. Pokud je hloubka uzlu *j rovna maximální přípustné hloubce,* pokračuj krokem č. 2.

Dle: Mařík a kol., Umělá inteligence 1, Academia, 1993

ALGORITMUS PROHLEDÁVÁNÍ DO HLOUBKY (2)

5. Expanduj stav *j. Pokud tento stav nemá následovníky nebo všichni* následovníci byli již expandováni (jsou v seznamu CLOSED), pokračuj krokem algoritmu č. 2.
6. Napiš všechny následovníky stavu *j, kteří nejsou v seznamu* CLOSED, na začátek seznamu OPEN.
7. Pokud některý z následovník *j je cílovým stavem, řešení bylo* nalezeno, ukonči prohledávání. Jinak pokračuj krokem č. 2.

VLASTNOSTI PROHLEDÁVÁNÍ DO HLOUBKY (1)

- Algoritmus má skromné požadavky na paměť, jelikož je zapotřebí uchovávat pouze jednu cestu z kořene k listovému uzlu spolu se zbylými neexpandovanými potomky každého uzlu na cestě.
- Stavový prostor s faktorem větvení *b* a maximální hloubkou *m* vyžaduje prostor pouze pro $bm+1$ uzlů.

VLASTNOSTI PROHLEDÁVÁNÍ DO HLOUBKY (2)

- Uvážíme-li stejné předpoklady jako u prohledávání do šířky, byla by náročnost prohledávání 118 kilobytů (namísto 10 petabytů) v hloubce 12.
- Nevýhodou je, že na začátku může být velmi dlouho (i nekonečně) expandován špatný uzel a řešení přitom může ležet hned u kořenového uzlu.
- Algoritmus proto může být modifikován na prohledávání omezené hloubky.

INFORMOVANÉ PROHLEDÁVÁNÍ

INFORMOVANÉ (HEURISTICKÉ) PROHLEDÁVACÍ STRATEGIE

- Kromě samotné definice problému využívají i specifické znalosti o problému, které umožňují hledat řešení efektivněji.
- Uzly určené pro expandování se vybírají na základě **vyhodnocovací funkce** *f(n)*.
- Uzel s nejnižším ohodnocením je vybrán k expanzi. Vyhodnocovací funkce uzlu měří vzdálenost uzlu od cíle.

ZNAČENÍ FUNKCÍ (1)

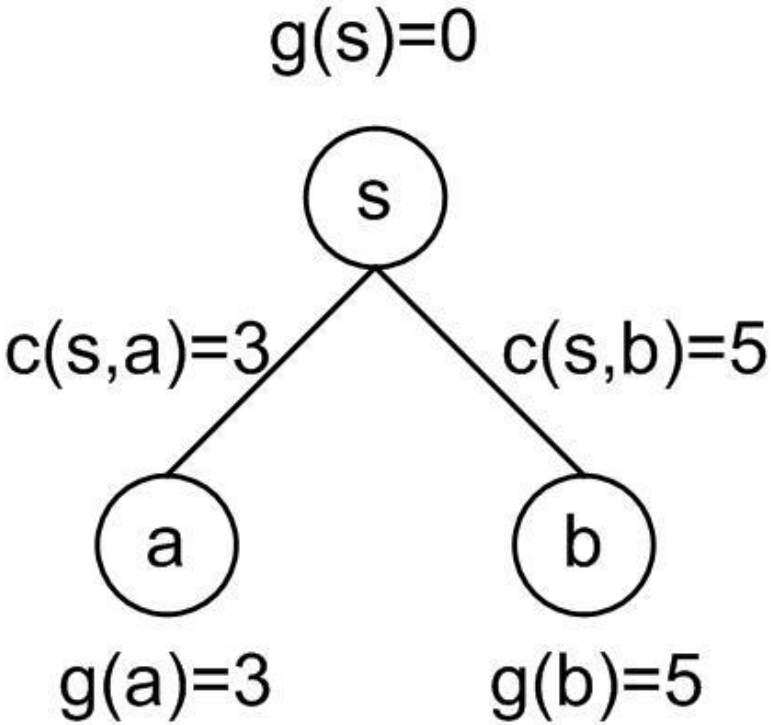
- Nákladová funkce

  g(s):=0,

  g(m):=g(n)+c(n,m),

  kde *m* je následovníkem *n*
- Cena hrany

  c(s,m) – cena cesty z *s* do *m*



ZNAČENÍ FUNKCÍ (2)

- *h**(*n*) označuje nejmenší náklady na cestu z uzlu *n* do některého z cílových uzlů (náklady na optimální cestu z uzlu *n).*
- *h**(*n*) předem neznáme, pouze odhadujeme.
- *h**(*n*) je odhad *h*(*n*). „h“ značí **heuristickou funkci.**
- *h(n)* vyjadřuje naši *heuristickou znalost* o tom, jaké jsou šance nalézt (optimální) řešení, kdybychom pokračovali expanzí uzlu *n.*

ZNAČENÍ FUNKCÍ - SHRNUTÍ

- *f(n)* – vyhodnocovací funkce.
- *h(n)* – heuristická funkce, vyjadřuje náklady na cestu z n do některého z cílových uzlů.
- *h*(n)* – odhadované náklady na optimální cestu z *n* do některého z cílových uzlů.
- *g(n)* – nákladová funkce, cena za dosažení uzlu.
- *c(s,a)* ohodnocení hrany mezi uzly *s* a *a*, tj. náklady spojené s krokem z *s* do *a*.
- f(n), h(n), h*(n), g(n), c(s, a) >= 0.

USPOŘÁDANÉ PROHLEDÁVÁNÍ* (1)

- Nejobecnější přístup k prohledávání stavového prostoru informovaným (heuristickým) přístupem.
- K expanzi volíme uzel, o kterém předpokládáme, že je podle vyhodnocovací funkce nejlepším kandidátem.
- V reálných situacích nemusí být vyhodnocovací funkce dostatečně výstižná.

*Pozn.: angl. "Best-first search"

USPOŘÁDANÉ PROHLEDÁVÁNÍ (2)

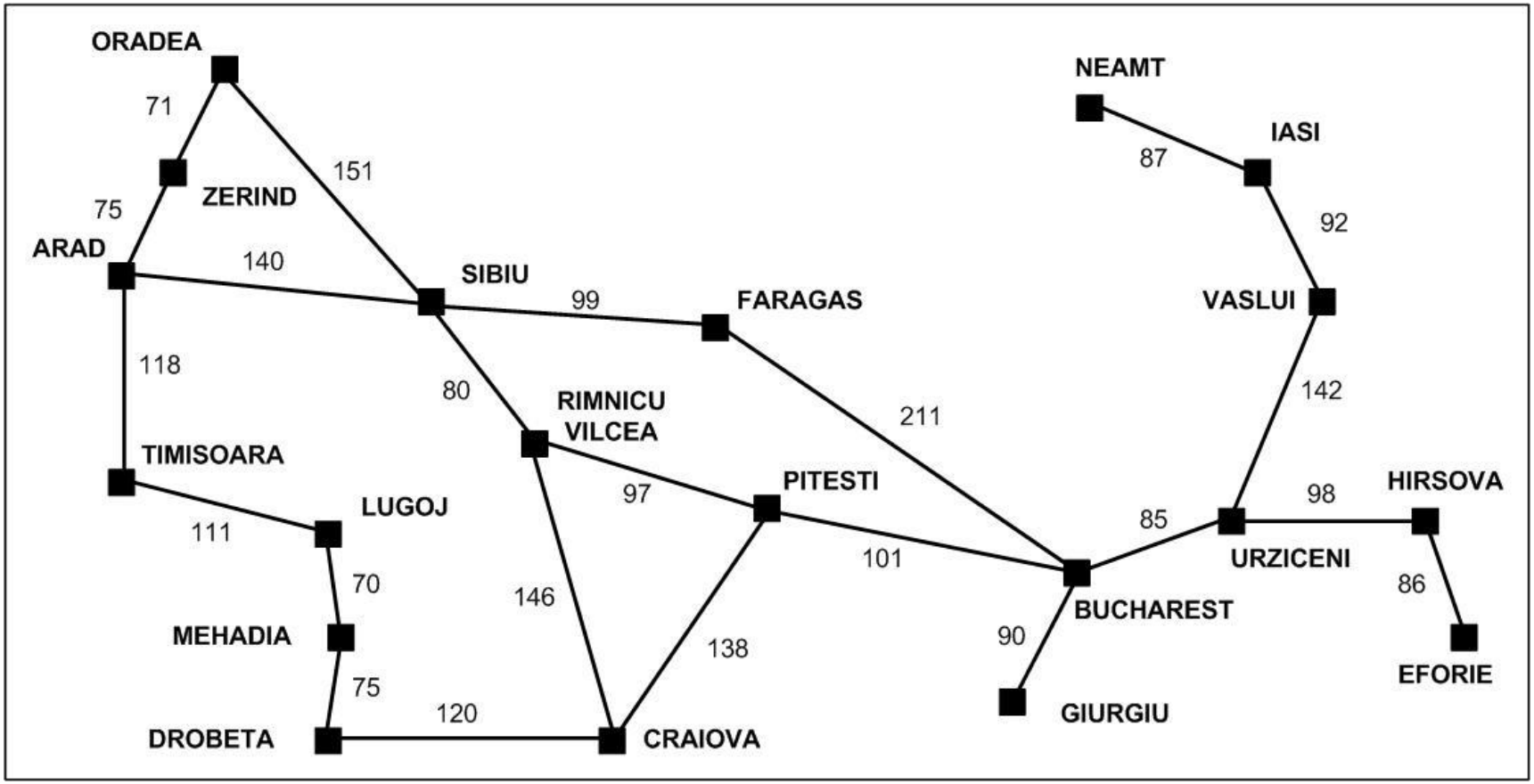
- Klíčovou komponentou informovaného prohledávání je **heuristická funkce**, ozn. $h(n)$.
- Algoritmus volí takové uzly, které se jeví být cíli co nejblíže (dle ohodnocení). Ohodnocení uzlů je tedy provedeno pouze na základě vztahu:

$$f(n) = h(n)$$

Ukázka řešení problému cestovatele

- Vraťme se k příkladu uvedenému dříve – cestování v Rumunsku. Úkolem je dostat se co nekratší cestou do Bukurešti.
- h_{SLD} – heuristika Straight Line Distance, která označuje přímou vzdálenost do Bukurešti.
- Hodnoty h_{SLD} nemohou být vypočteny ze samotné definice problému. Proto se hodnoty na původní mapě a v tabulce liší.

CESTOVNÍ MAPA*

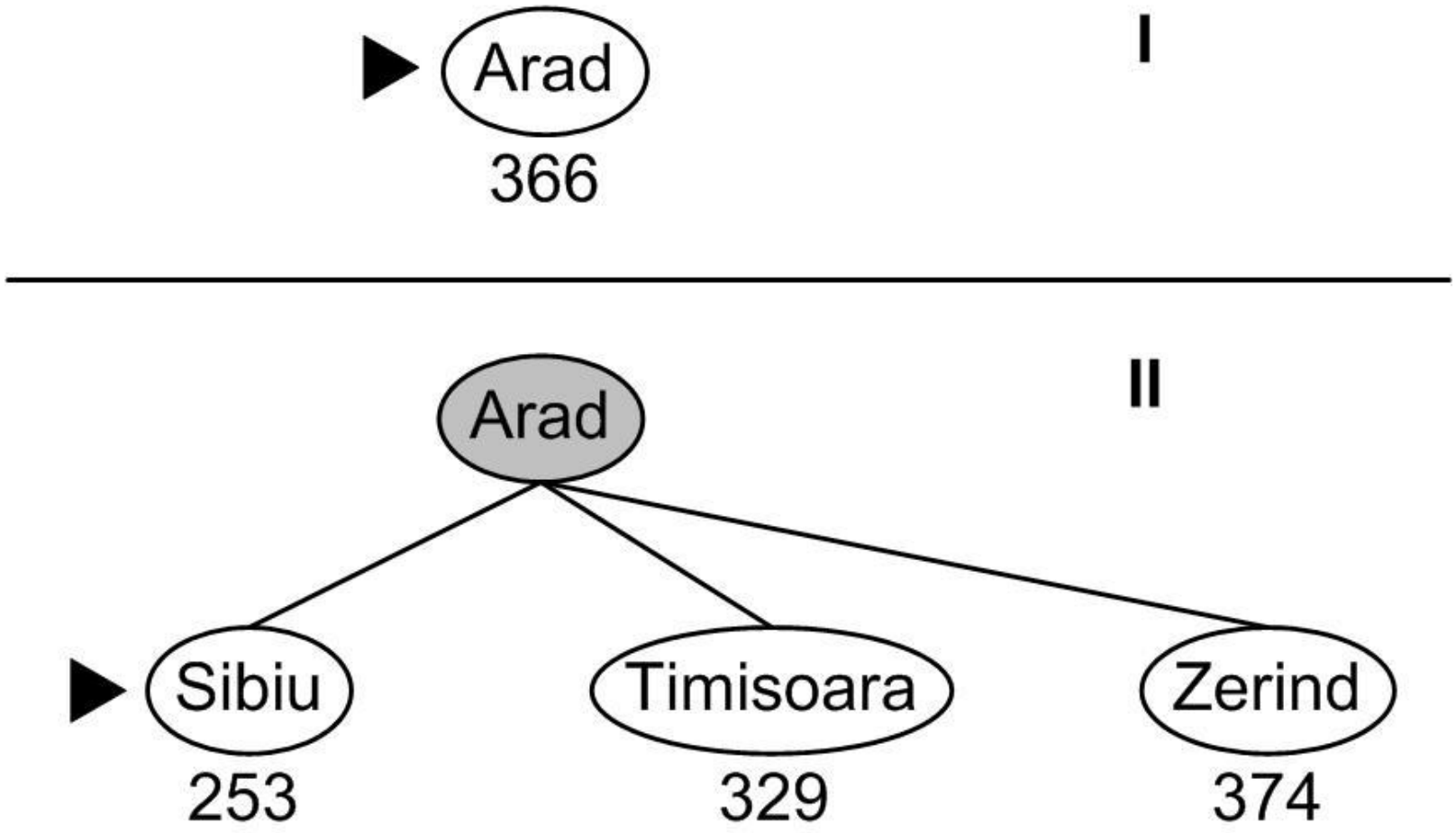


*Pozn. názvy jsou v angličtině

H_{SLD} – Straight Line Distance

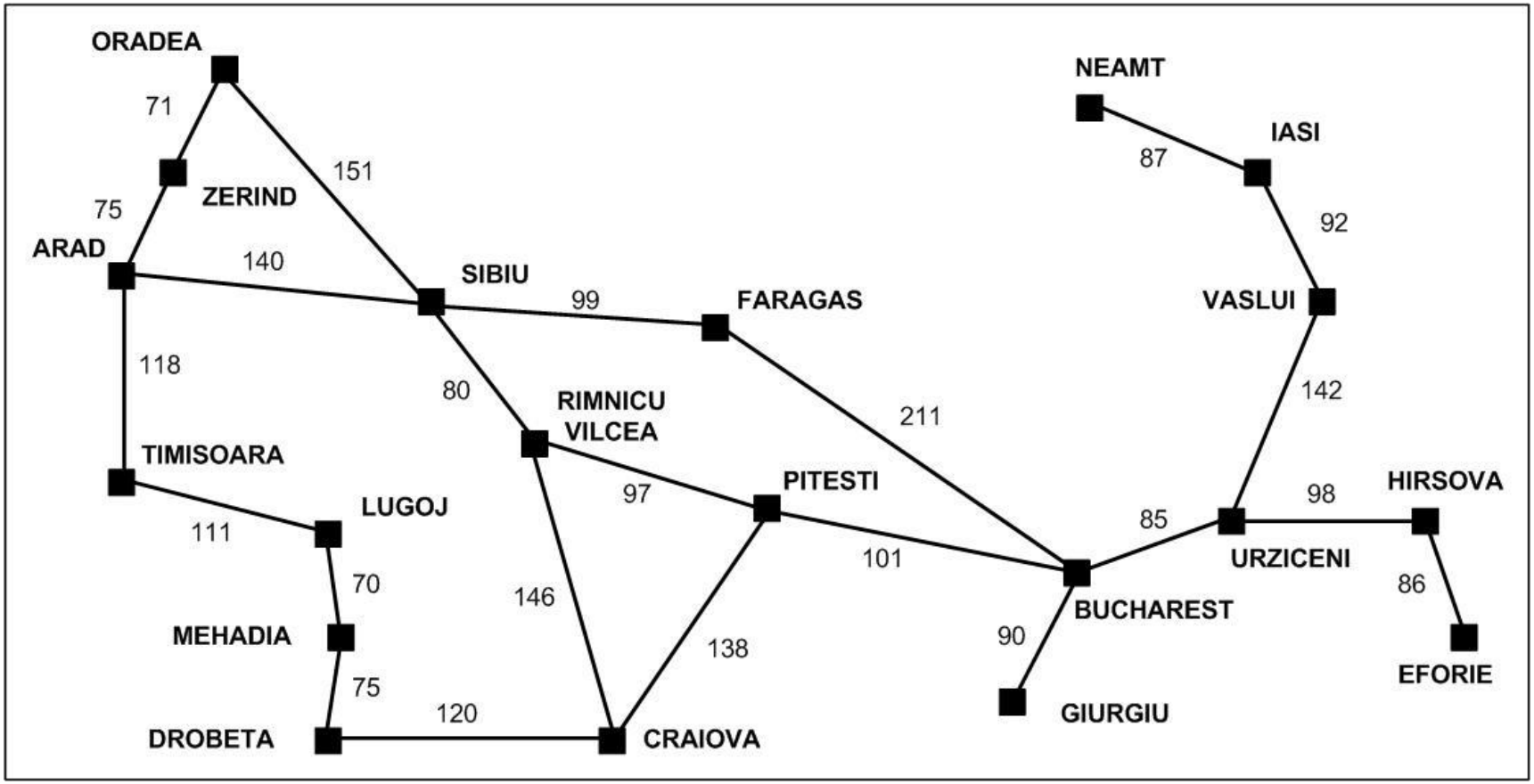
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Počáteční stav řešení a stav po expanzi uzlu Arad



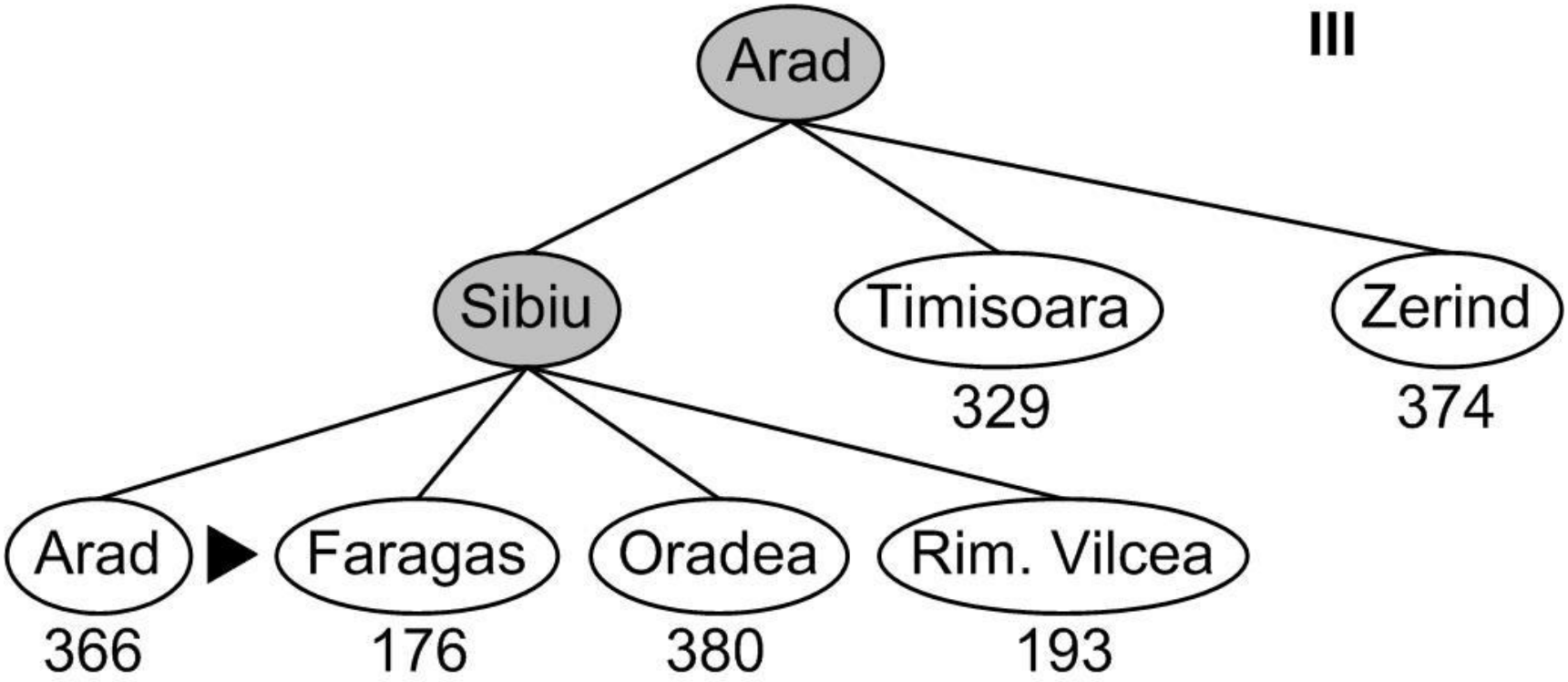
Pozn. uzly označeny hodnotou h(n)

CESTOVNÍ MAPA*



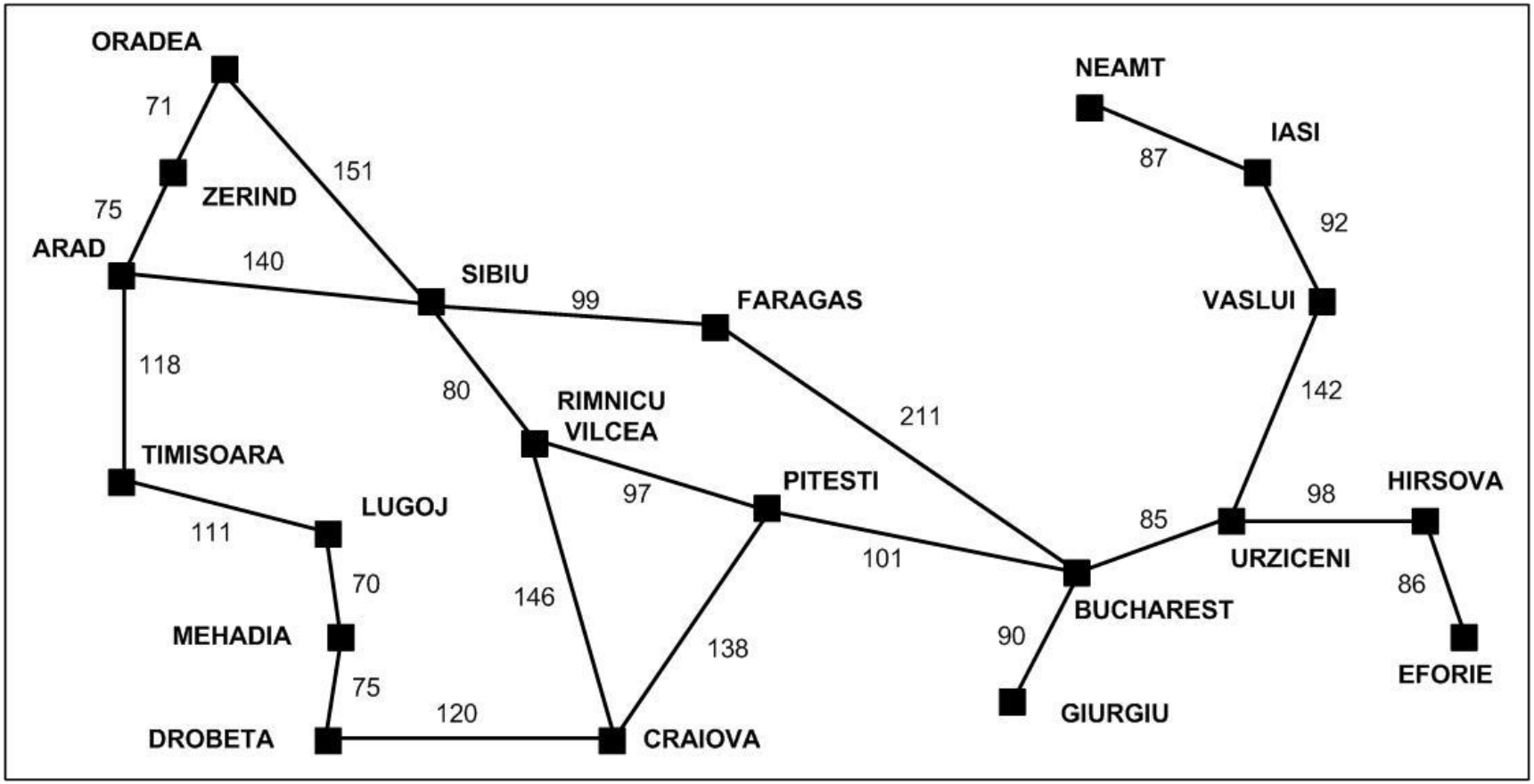
*Pozn. názvy jsou v angličtině

STAV PO EXPANZI UZLU SIBIU



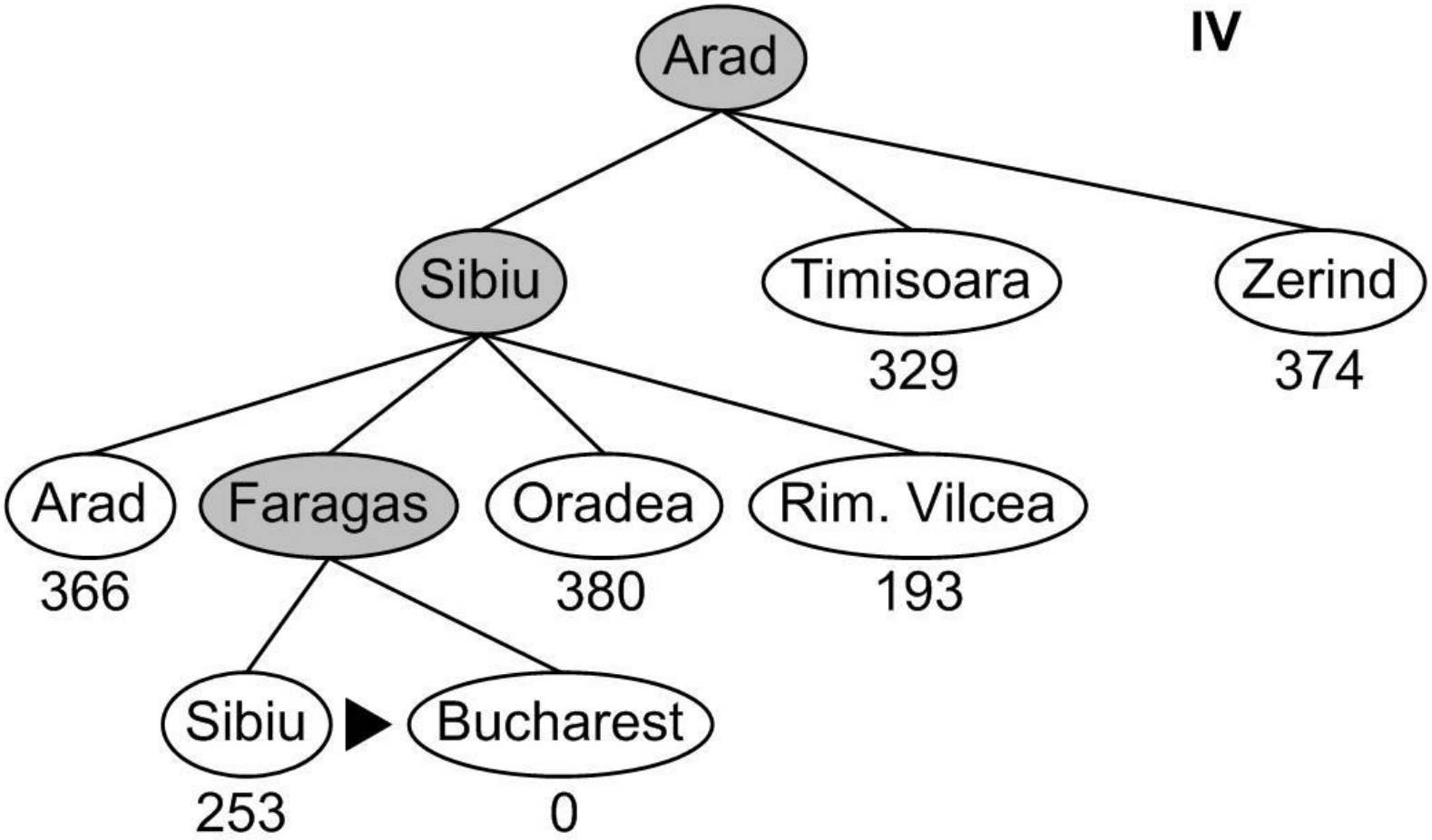
Pozn. uzly označeny hodnotou h(n)

CESTOVNÍ MAPA*



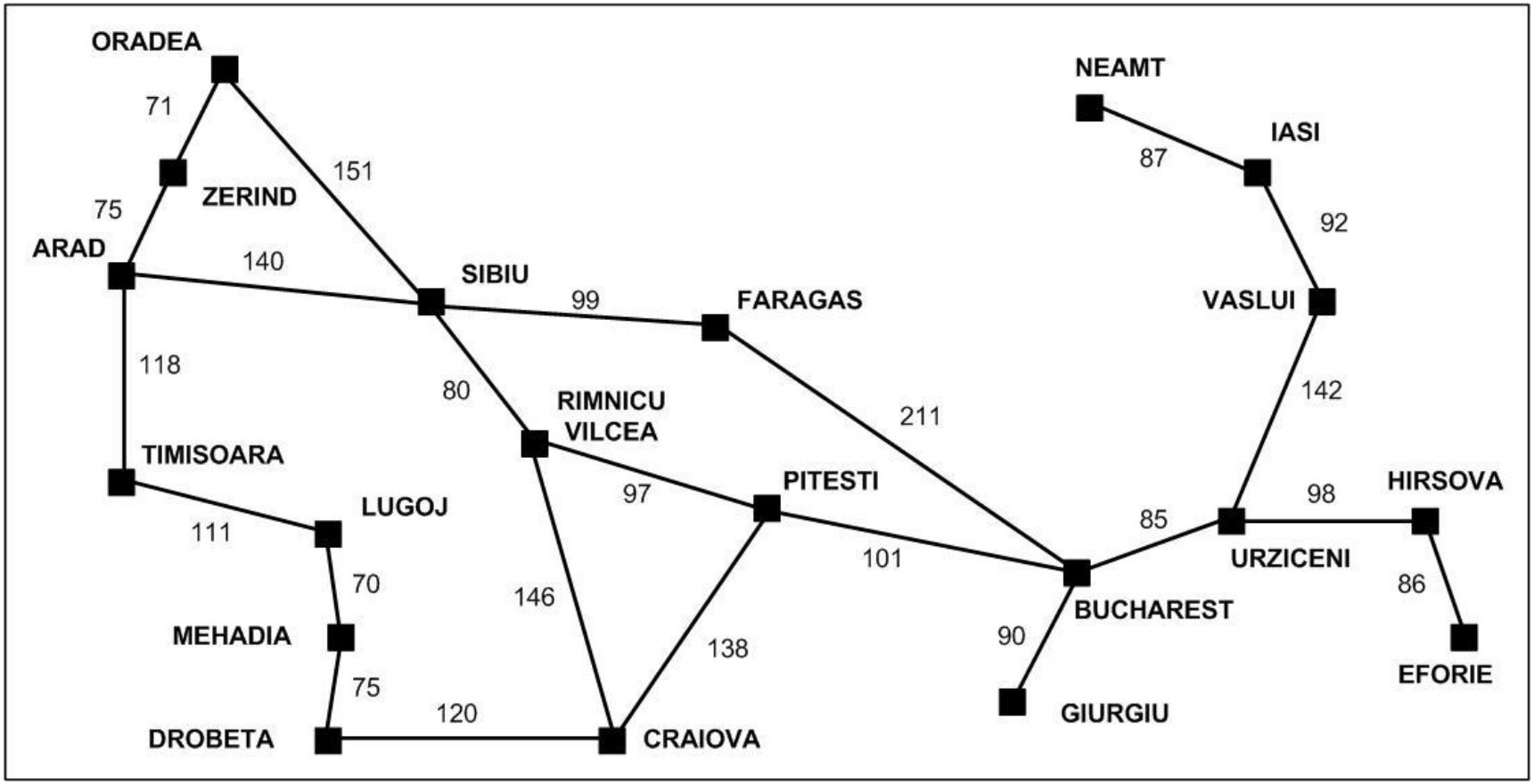
*Pozn. názvy jsou v angličtině

STAV PO EXPANZI UZLU FARAGAS



Pozn. uzly označeny hodnotou h(n)

CESTOVNÍ MAPA*



*Pozn. názvy jsou v angličtině

POPIS ŘEŠENÍ

1. K expandování je vybrán uzel Sibiu, jelikož je k Bukurešti blíže než Timisoara nebo Zerind.
2. Expandován je Faragas, je nejblíže.
3. Faragas expanduje do Bukurešti, což je cíl.

- Při řešení tohoto problému algoritmus neexpandoval žádný uzel, který by nebyl součástí řešení. Prohledávací cena tedy byla minimální.

POZNÁMKY K ŘEŠENÍ

- Algoritmus nenalezl optimální cestu – cesta Sibiu→Faragas →Bukurešť je o 32 km delší než trasa přes Rimnicu Vilcea a Pitesti.

  140+99+211 = 450 km

  140+80+97+101 = 418 km

- Algoritmus se v každém kroku snaží maximalizovat přiblížení k cíli.

A* ALGORITMUS (1)

- Je to modifikace uspořádaného prohledávání (best-first algoritmu).
- Ohodnocuje uzly kombinací *g(n)* a *h(n)*, výsledná cena je tedy:

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

- Jelikož *g(n)* vrací cenu cesty z počátečního uzlu do uzlu *n* a *h(n)* je odhadovanou cenou nejlevnější cesty z *n* k cíli, máme

  f(n) = odhadovaná cena nejlevnějšího řešení přes *n*

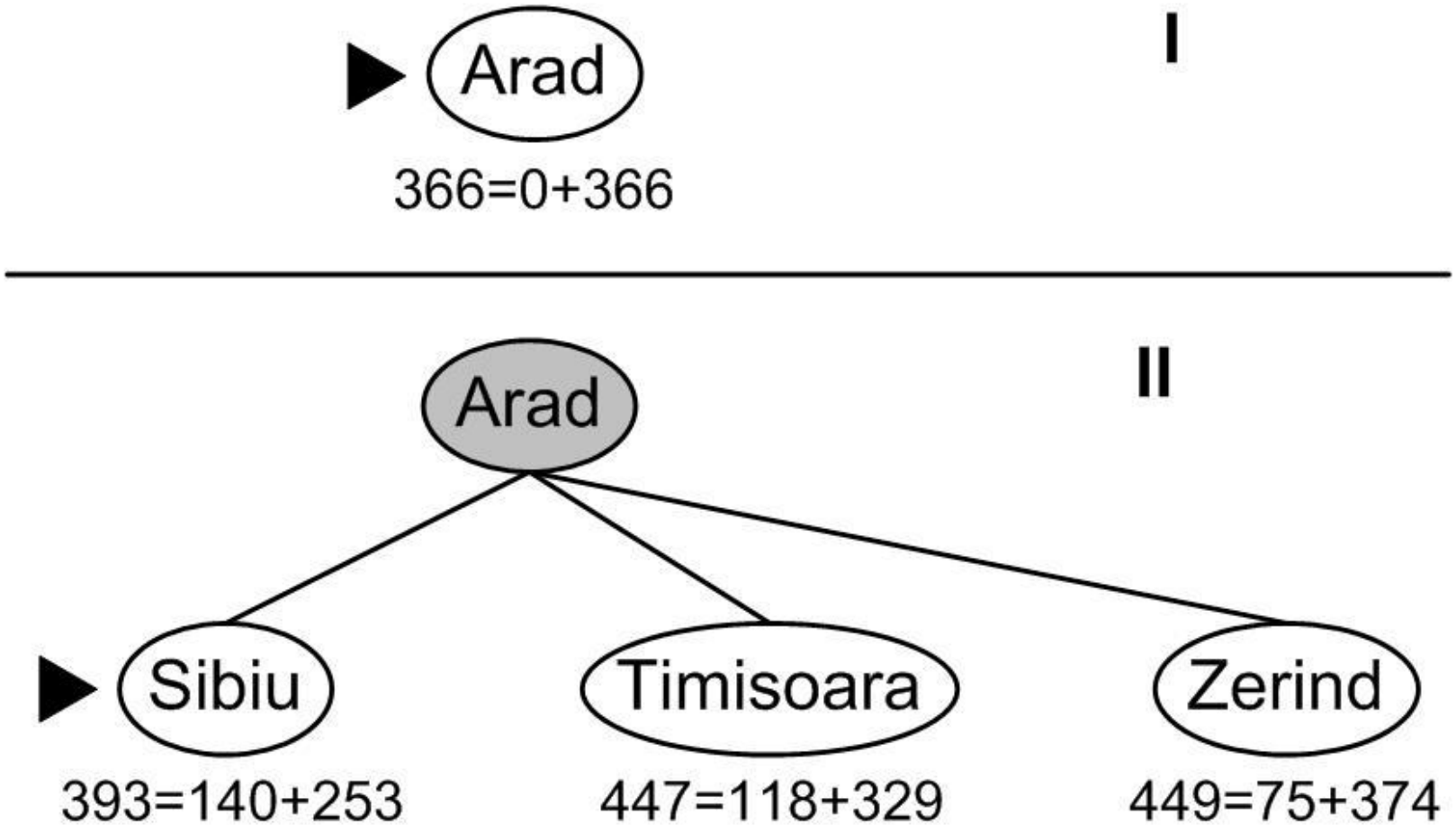
Pozn. čte se [a: star]

A* ALGORITMUS (2)

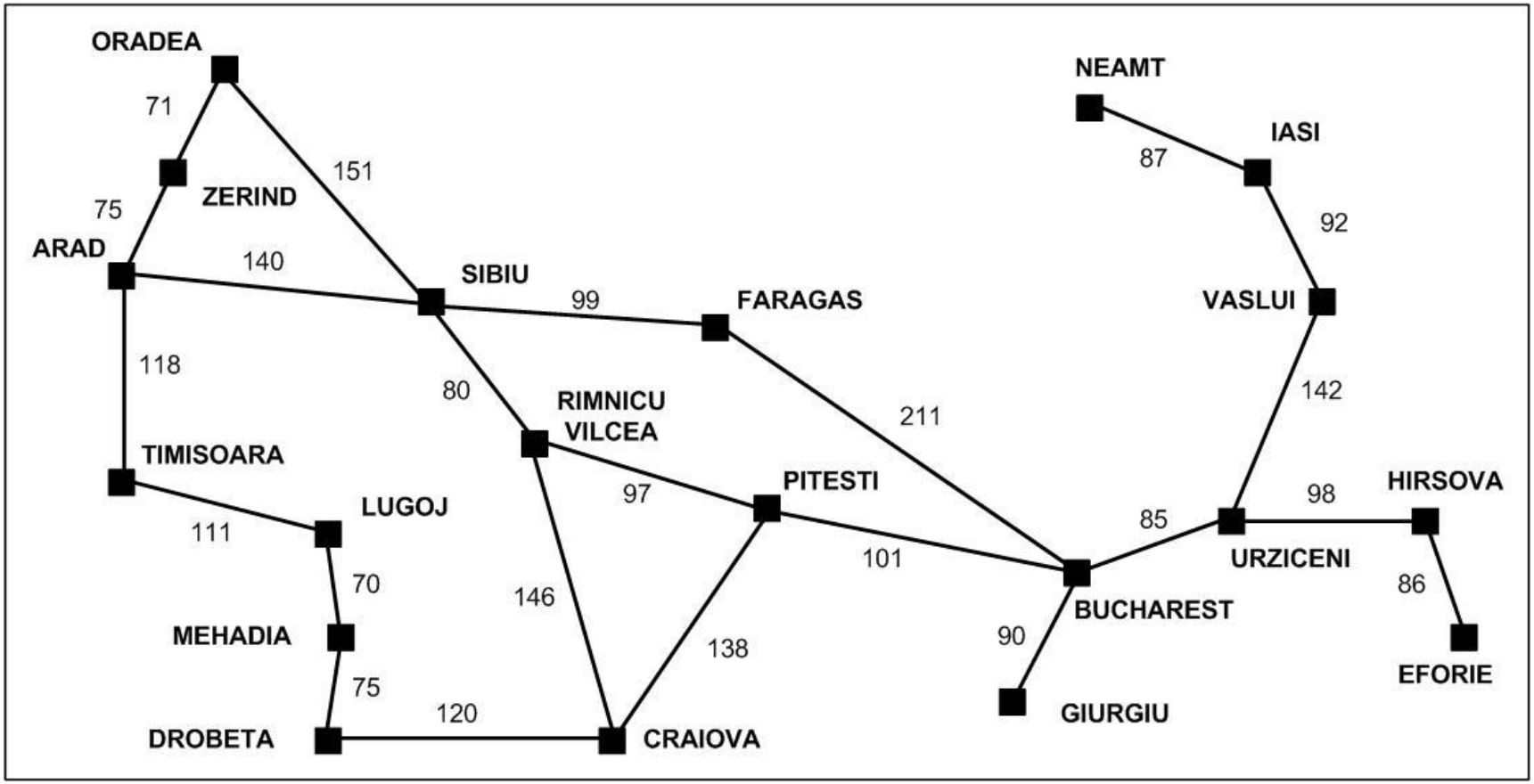
- Hledáme-li nejlevnější řešení, bývá nejvýhodnější začít uzlem s nejnižší hodnotou $g(n)+h(n)$.
- Je-li $h(n)$ přípustná heuristika (angl. admissible heuristic), tj. pokud $h(n)$ nikdy nepřeceňuje cenu k dosažení cíle, je A* optimální*.
- Přípustné heuristiky jsou z takové, že odhadují cenu řešení menší než ve skutečnosti je.
- Jelikož $g(n)$ dává přesnou cenu k dosažení n, f(n) v důsledku nikdy nepřecení skutečnou cenu řešení přes uzel n.

*Pozn. důkaz viz. Russell, Norvig: Artificial Intelligence, Ch. 4

POČÁTEČNÍ STAV ŘEŠENÍ A STAV PO EXPANZI UZLU ARAD

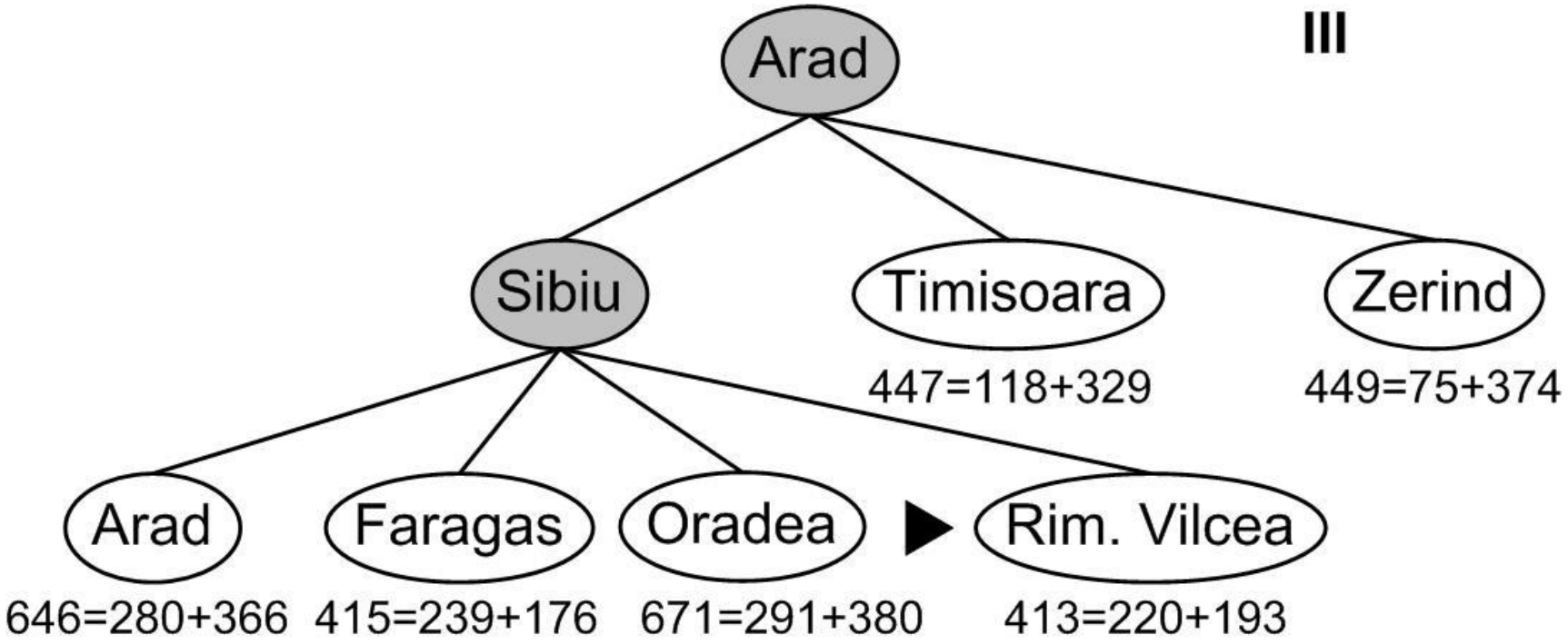


CESTOVNÍ MAPA*

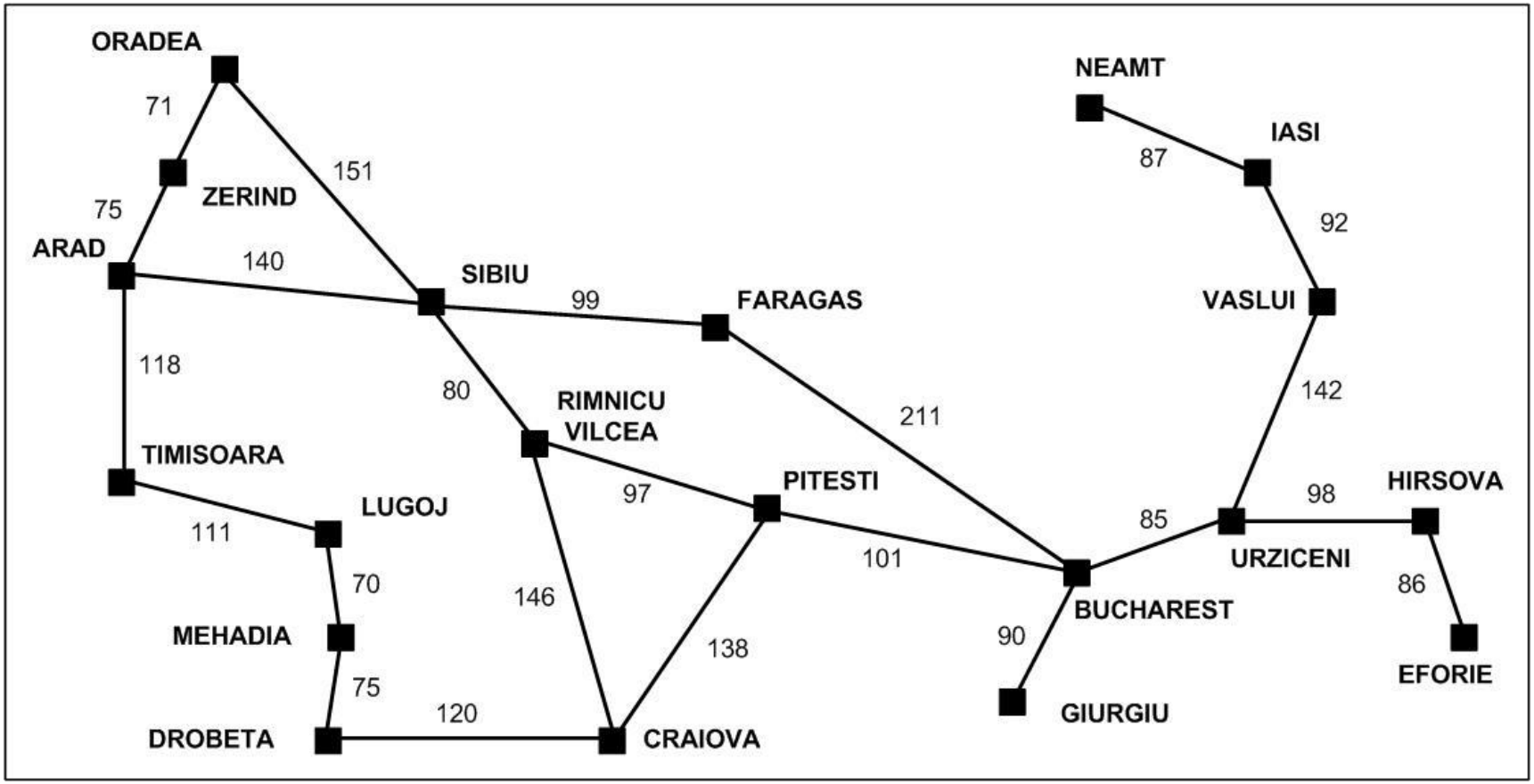


*Pozn. názvy jsou v angličtině

STAV PO EXPANZI UZLU SIBIU

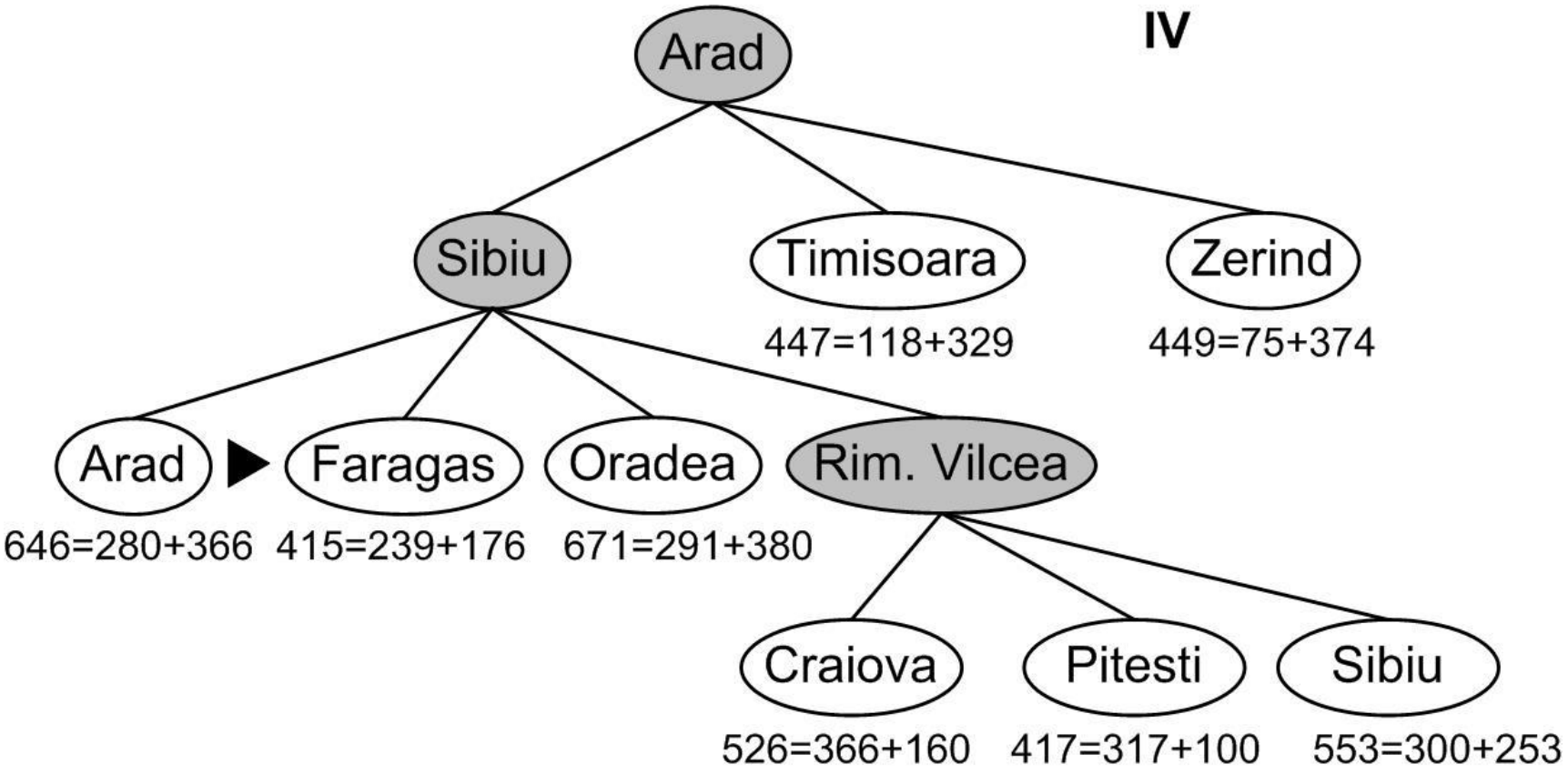


CESTOVNÍ MAPA*

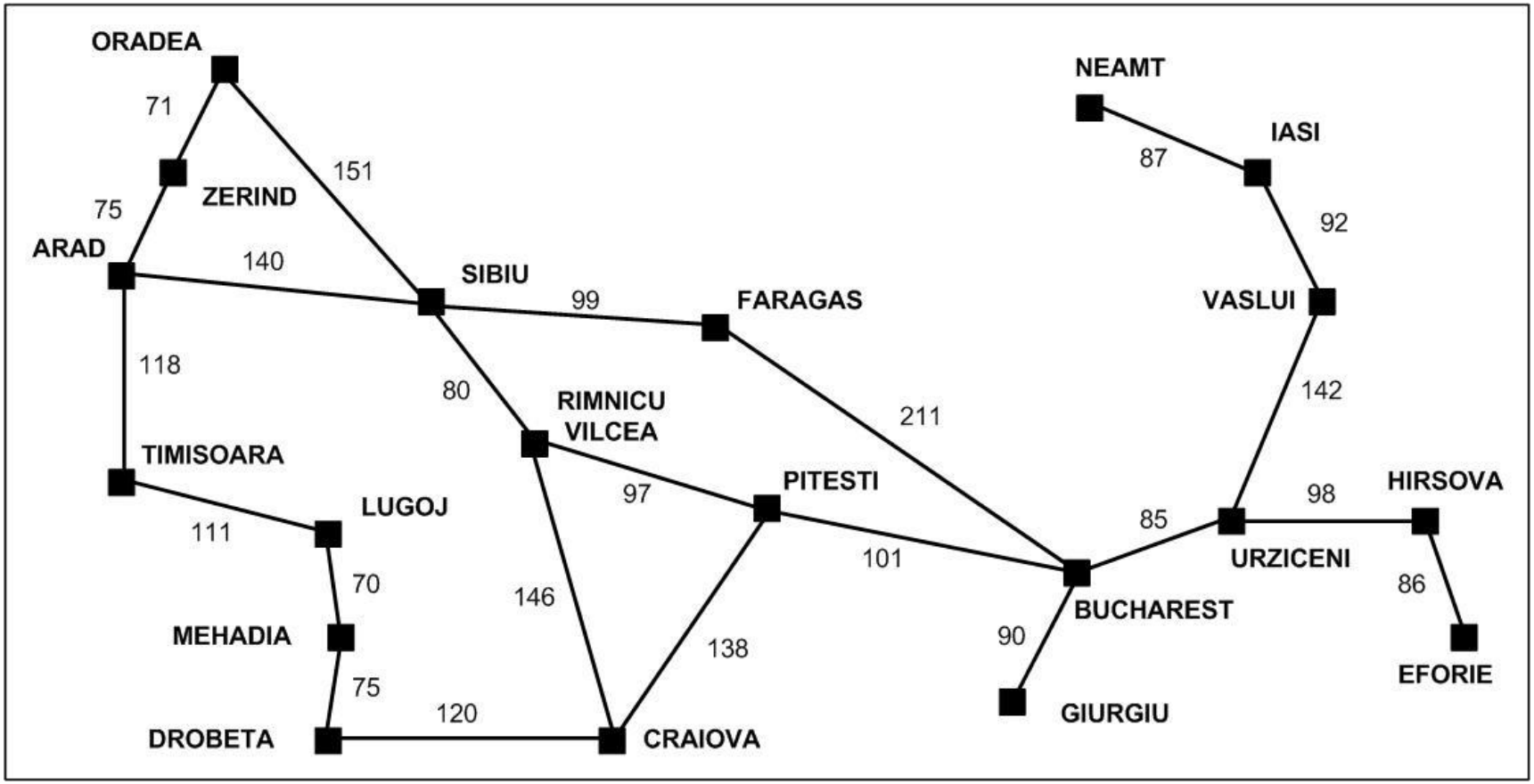


*Pozn. názvy jsou v angličtině

STAV PO EXPANZI UZLU RIMNICU VILCEA

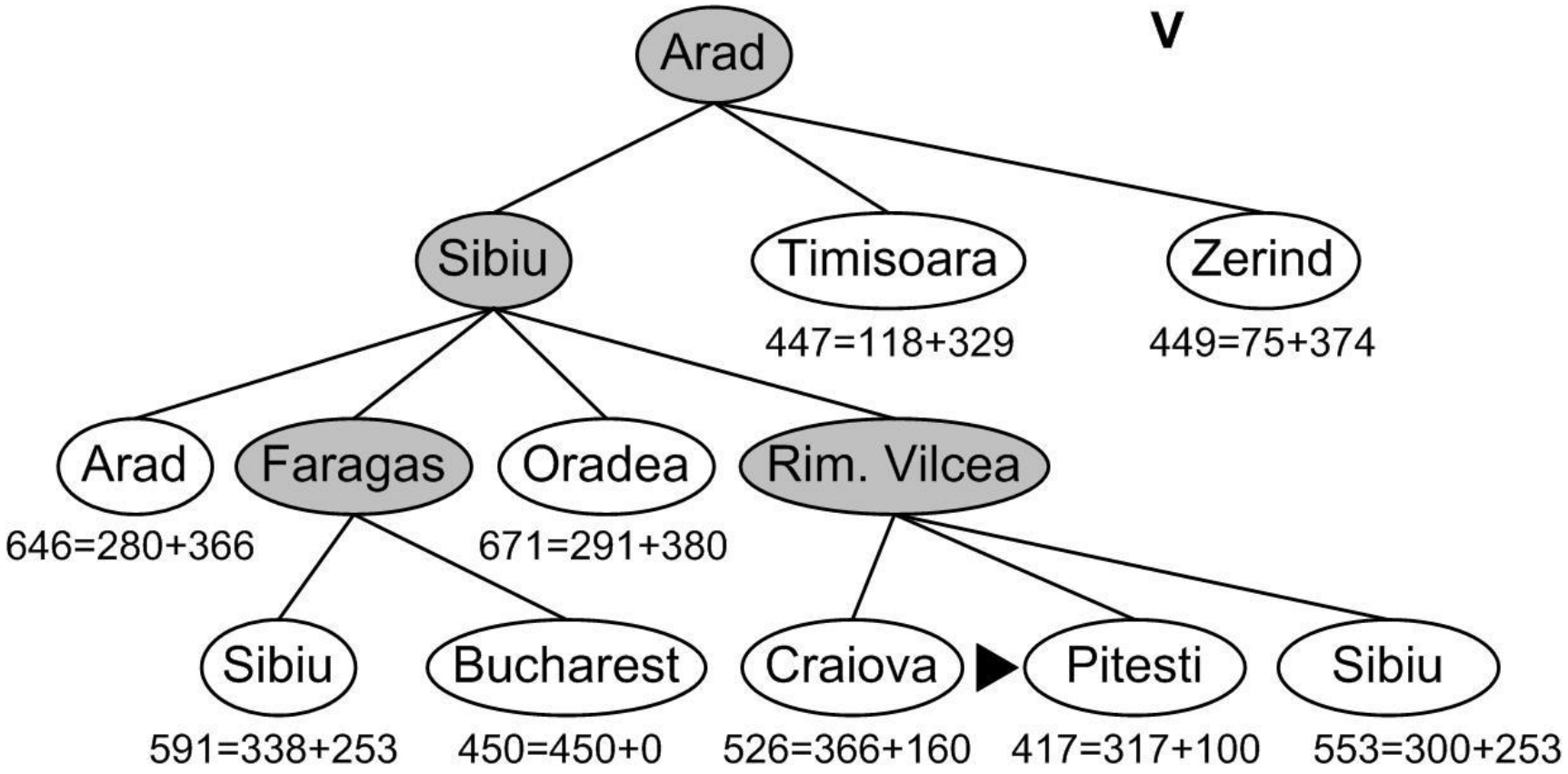


CESTOVNÍ MAPA*

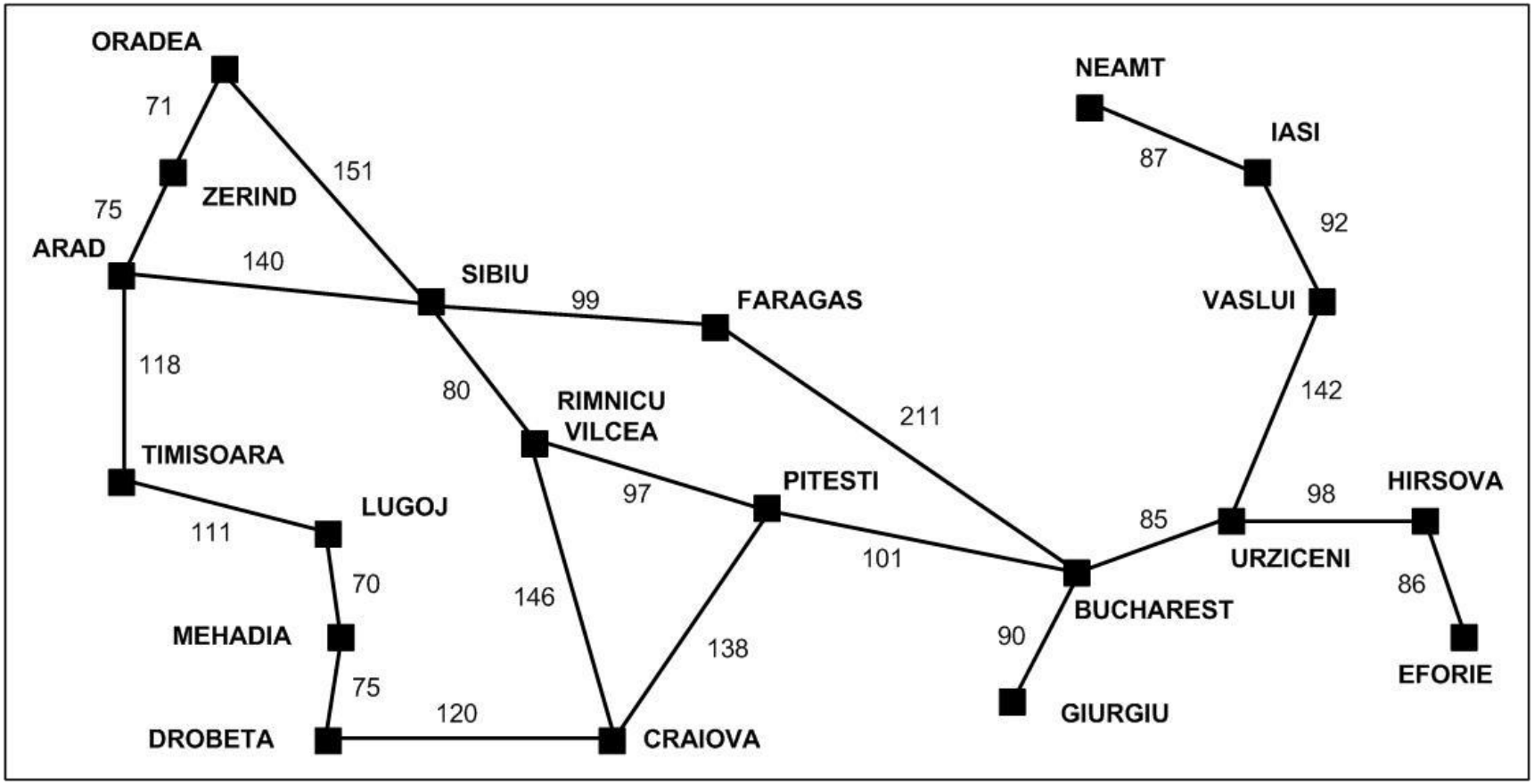


*Pozn. názvy jsou v angličtině

STAV PO EXPANZI UZLU FARAGAS

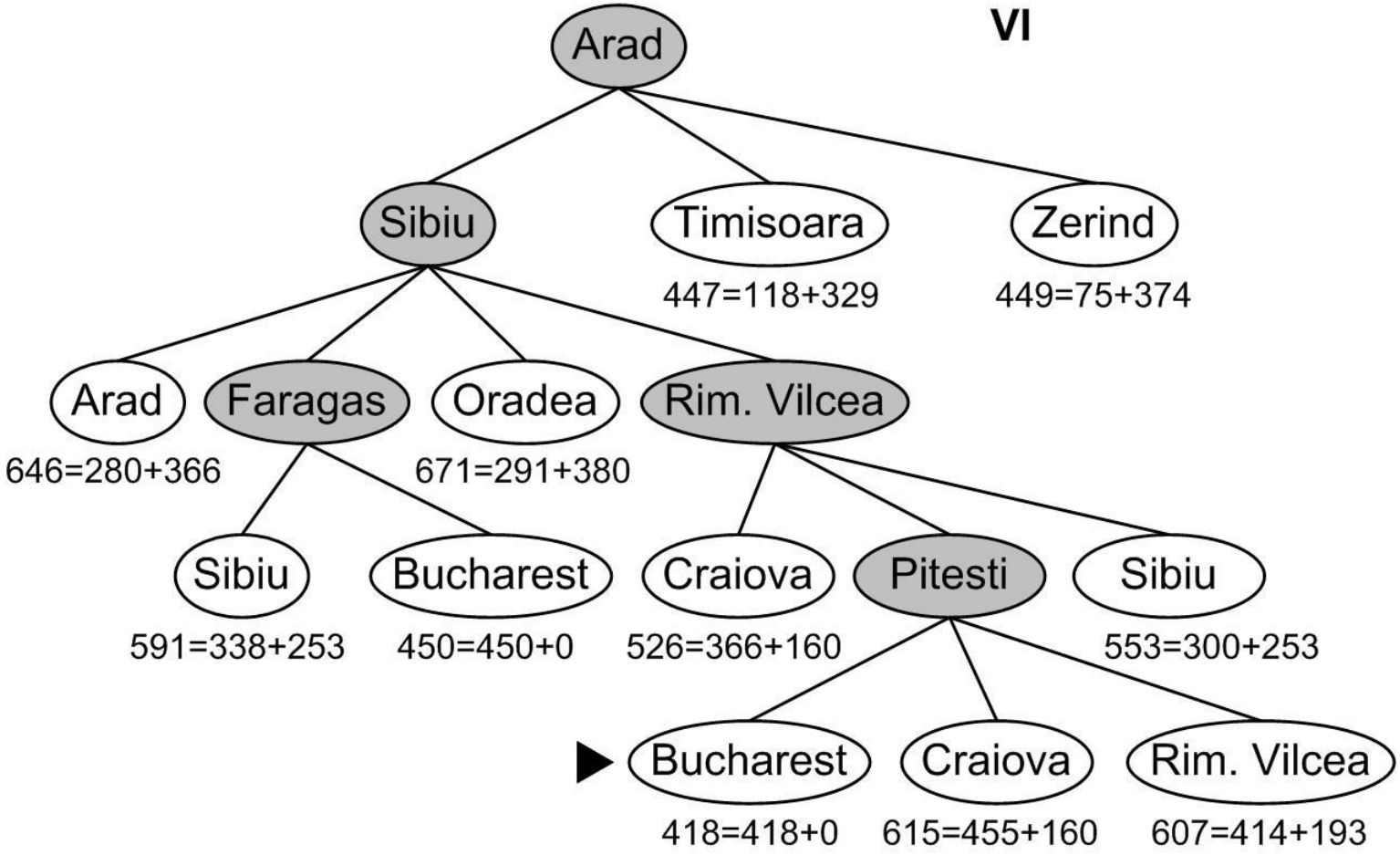


CESTOVNÍ MAPA*

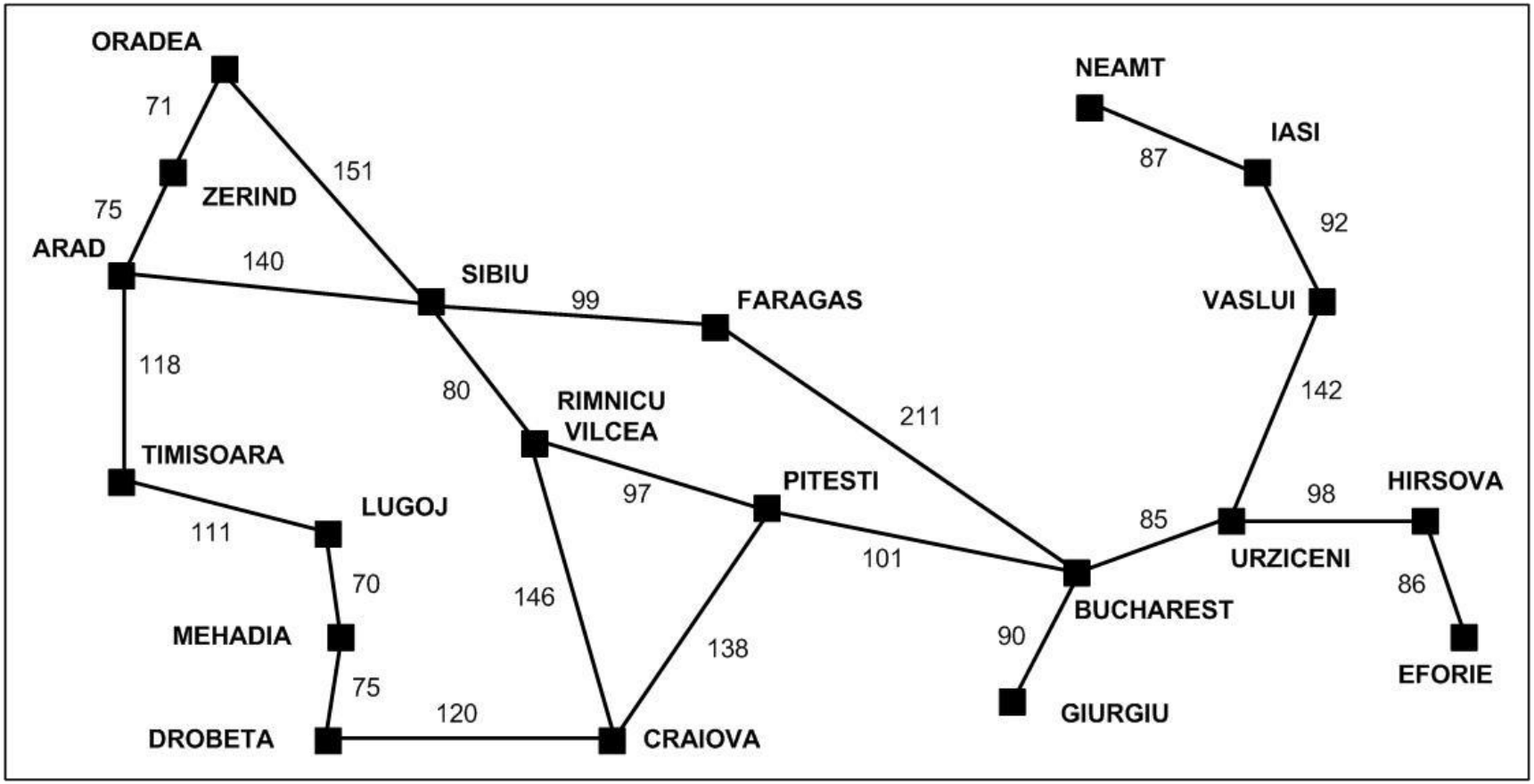


*Pozn. názvy jsou v angličtině

STAV PO EXPANZI UZLU PITESTI



CESTOVNÍ MAPA*



*Pozn. názvy jsou v angličtině

POZNÁMKY K A* ALGORITMU (1)

- Algoritmus A* vždy najde optimální cestu. Krom toho je schopen prohledávat celý stavový prostor a je optimálně efektivní.
- Při nedostatečně (málo informované) výstižné heuristické funkci se ale pro složitější řešení dostáváme do exponenciálního růstu počtu stavů a algoritmus je nepoužitelný.

POZNÁMKY K A* ALGORITMU (2)

- V případě, že nedisponujeme výstižnou heuristikou se proto často aplikuje pro rychlé získání suboptimálního řešení.
- V každém případě představuje velkou úsporu času i paměti v porovnání s neinformovaným prohledáváním.

DĚKUJI ZA POZORNOST

POUŽITÁ LITERATURA

- Russel S., Norvig P.: „Artificial Intelligence – A Modern Approach“. Prentice Hall, 2003. (2nd Edition)
- Mařík V., Štěpánková O., Lažanský J. a kol.: „Umělá inteligence 1“. Academia, 1993.