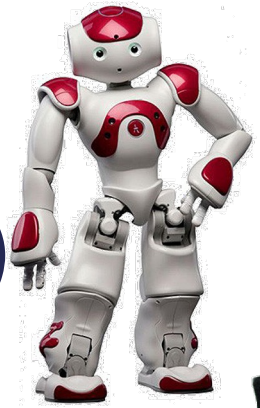
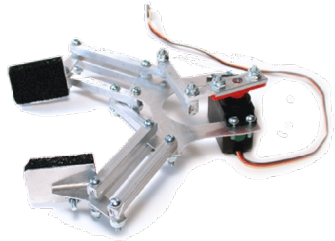
PROGRESIVNÍ TECHNOLOGIE V INFORMATICE II

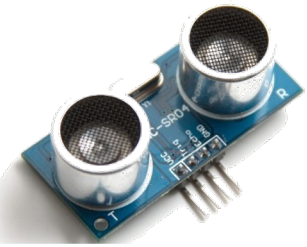
UMĚLÁ INTELIGENCE
ROBOTIKA A PLÁNOVÁNÍ

Financováno Evropskou unií
NextGenerationEU
Národní plán obnovy
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

ROBOTI (NEBO ROBOTY)



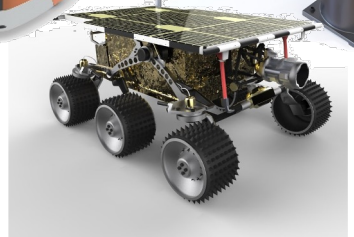




- **Fyzičtí agenti**
  - plní úkoly manipulací s fyzickým světem
    - pomocí **efektorů**
      - uplatňují **fyzickou sílu** na prostředí
      - ruce, nohy, kola, klouby, kleště, vrtáky, pily
    - mají také **senzory**
      - **vnímají** jimi prostředí
      - kamery, sonary, gyroskopy, akcelerometry

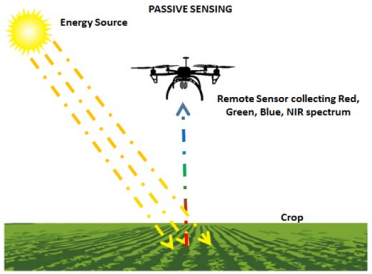
- **Typičtí roboti**
  - **robotická ruka** (manipulátor)
    - ukotvený na místě, mnoho kloubů
      - výrobní linky, operační roboti

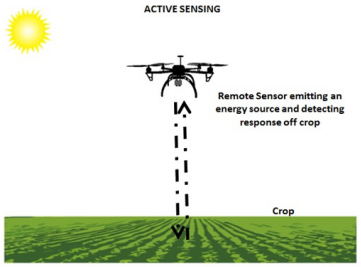
  mobilní roboti
    - ULV – vozidlo (unmanned land vehicle)
    - UAV – letoun (unmanned aerial vehicle)
    - AUV – ponorka (unmanned underwater vehicle)
  - **humanoidní/hybridní** roboti
    - neukotvený, různé efektory, mobilita

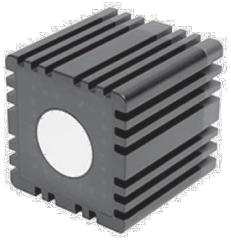


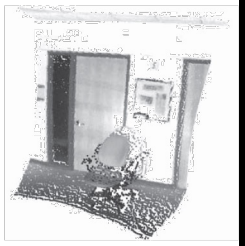
ROBOTICKÝ HARDWARE SENZORY

- **Senzory**
  - **pasivní**
    - pozorují signály z prostředí
      - kamera, mikrofon
  - **aktivní**
    - vysílají energii do prostředí a pozorují její odraz
      - sonar (aktivní)
    - vyšší spotřeba, nebezpečí interference
  - dělení podle **předmětu měření**
    - prostředí, umístění robota, vnitřní stav robota
- **Konkrétní hardware**
  - **prostředí**
    - radar, laser, sonar, lidar, kamera (stereo)
    - taktilní senzory hmatové vousy, nárazníky
  - **umístění**
    - GPS, majáčky
  - **vnitřní**
    - otáčky hřídelí (odometrie), inerciální senzory (gyroskopy)

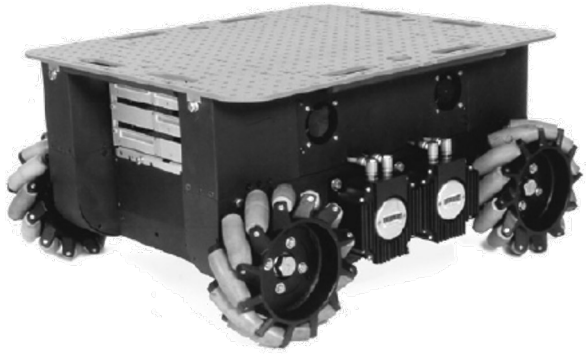




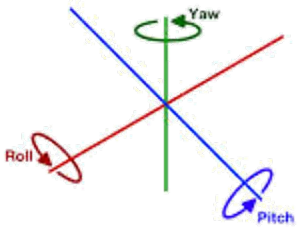


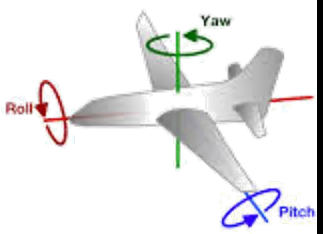


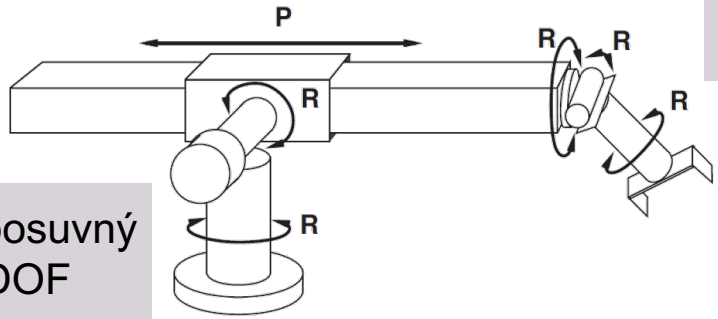
ROBOTICKÝ HARDWARE
EFEKTORY



- **Efektory pro pohyb**
  - zajišťují pohyb robota a změny jeho tvaru
- **Stupně volnosti** (degree of freedom - DOF)
  - **nezávislé směry**, ve kterých se robot nebo některý z jeho efektorů může pohybovat
  - UAV má 6 stupňů volnosti
    - 3 pro pozici (x, y, z), 3 pro orientaci
  - definují **kinematický stav** (postavení)
  - více DOF = lepší kontrola, složitější výpočty
- **Dynamický stav**
  - ke každé kinematické dimenzi přidává **rychlost změny**

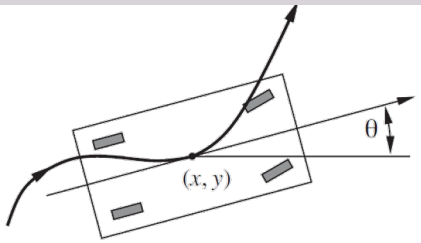




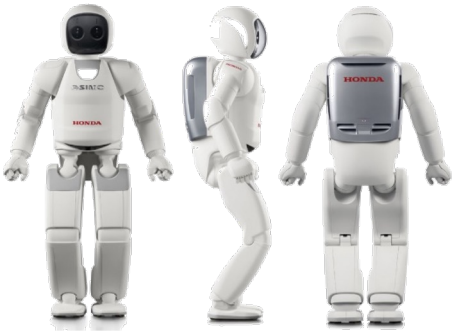


5 rotačních kloubů (R), 1 posuvný kloub (prismatic - P) → 6 DOF

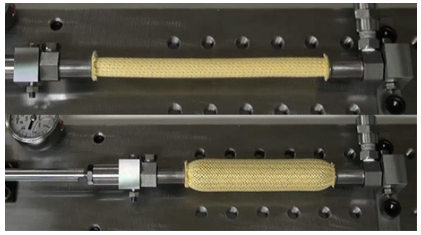
3 efektivní DOF, 2 ovládané DOF,
3 > 2 → neholonomický robot

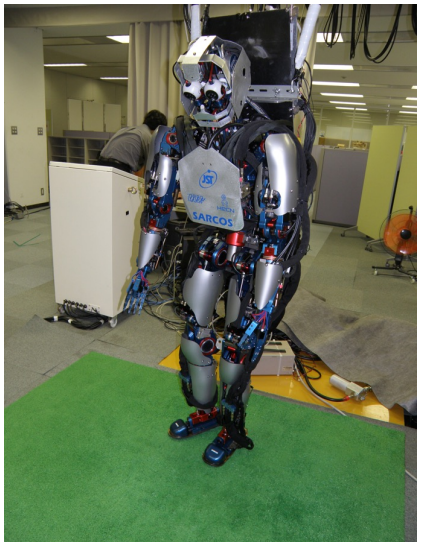


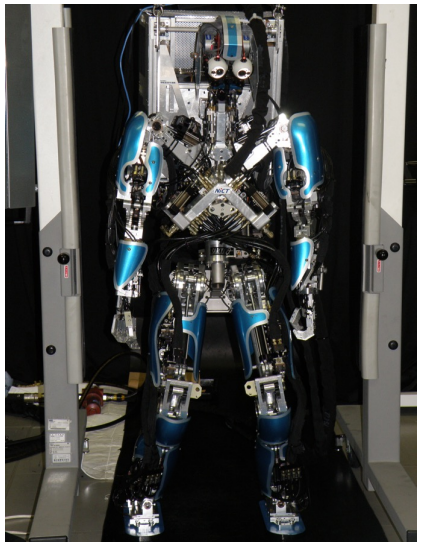
POHON



- **Elektrické motory + baterie, zásuvka**
  - těžké baterie, těžké motory, přesná kontrola
- **Pneumatický**
  - stlačený plyn (vzduch), kompresor (často externí)
  - hadice, písty, **umělé svaly**
    - plynulé pohyby (park **Disney**), horší kontrola
    - pozor na kondenzaci
- **Hydraulický**
  - vyšší tlak, menší velikost
- **Bezpečnost**
  - moc **silní** a **těžcí** roboti
    - zavěšení na konstrukci
    - provoz v kleci
    - **nouzový vypínač**
  - riziko výbuchu (pneumatický)
- **CPU, senzory**
  - nezanedbatelná spotřeba







BAYESOVO PRAVIDLO

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- **Pravděpodobnostní odvozování**
  - využijeme jej k odvozování znalosti stavu prostředí (robota) na základě pravděpodobnostní informace ze senzorů
  - **P(A ∧ B) = P(A,B) = P(A|B) * P(B) = P(B|A) * P(A)**
  - **P(A|B) = P(B|A) * P(A) / P(B)**
- **Senzor intenzity světla ukazuje hodnotu z**
  - víme, že **P(z | rozsvíceno) = 0.7** a **P(z | není rozsvíceno) = 0.3**
  - dále víme jisté nepodmíněné pravděpodobnosti
    - **P(rozsvíceno) = 0.1** a **P(není rozsvíceno) = 0.9**
  - **P(rozsvíceno | z) = ?**

Kauzalita

Diagnóza

  - P(rozsvíceno | z) = P(z | rozsvíceno) * P(rozsvíceno) / P(z)
  - P(R|z) = α[P(z|rozsvíceno) * P(rozsvíceno), P(z|není) * P(není)] = = α[0.7 * 0.1, 0.3 * 0.9] = α[0.07, 0.27] = [0.206, 0.794], pro α=2.94
- **P(B|A) = α P(A|B) * P(B)**
  - P(rozsvíceno | z) = 2.94 * 0.7 * 0.1 = 0.206

VNÍMÁNÍ

- **Proces vytváření vnitřní reprezentace prostředí**
  - na základě měření senzory
- **Vlastnosti reprezentace**
  - dostatek informací pro další rozhodování robota
  - umožňuje snadné změny
  - přirozeně odpovídá prostředí
    - proměnné vyjadřují stavy prostředí
- **Značení**
  - stav prostředí X_t (včetně stavu robota)
    - vektor proměnných popisující **stav prostředí** v čase t
    - x_t vektor hodnot, kterých X_t nabývají v čase t
  - pozorování Z_t
    - vektor proměnných popisujících **pozorování** čase t
    - z_t vektor hodnot, kterých Z_t nabývají v čase t
  - akce A_t
    - vektor proměnných popisující robotovu **akci** v čase t
    - a_t vektor hodnot, kterých A_t nabývají v čase t

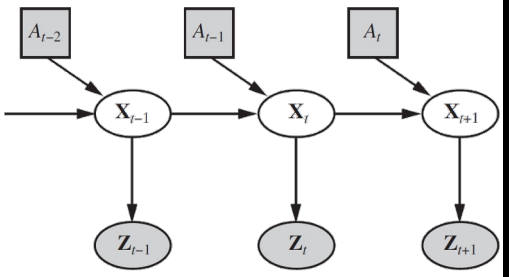
Pozice robota
$X_t = (x_t, y_t, z_t)$
$x_t = (1.0, 0.5, 1.5)$

Pozorovaná pozice
$Z_t = (zx_t, zy_t, zz_t)$
$z_t = (1.2, 0.6, 1.4)$

Akce 'kupředu'
$A_t = (dx_t, dy_t, dz_t)$
$a_t = (0.2, 0.2, 0.3)$

AKTUALIZACE PŘESVĚDČENÍ

- **Přesvědčení robota** (belief state) **$P(X_t \mid z_{1:t}, a_{1:t-1})$**
  - na základě měření senzory a provedených akcí se přesvědčení aktualizuje (Bayesovsky)

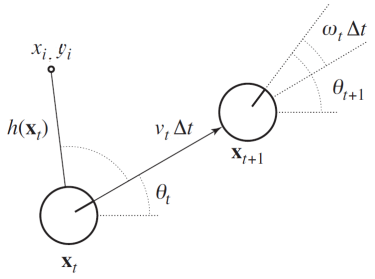


$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t})$$
$$= \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)\, P(\mathbf{x}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})\, d\mathbf{x}_t$$

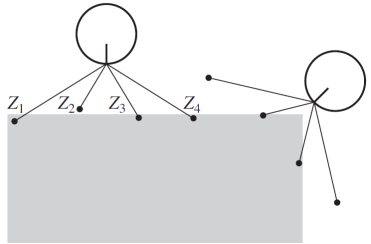
- **$P(x_t \mid z_{1:t}, a_{1:t-1})$**
  - pravděpodobnost stavu určeného x_t za předpokladu znalosti předchozích měření a akcí
    - reprezentováno distribuční funkcí
- **$P(X_{t+1} \mid x_t, a_t)$**
  - pravděpodobnost, že po provedení akce určené a_t ve stavu určeném x_t nastane stav X_{t+1}
    - určeno přechodovým modelem - <u>známe</u>
- **$P(z_{t+1} \mid X_{t+1})$**
  - pravděpodobnost, že pozorujeme z_{t+1} ve stavu X_{t+1}
    - určeno senzorovým modelem - <u>známe</u>

LOKALIZACE

- **Vědět kde se co nachází je pro robota klíčové, jedná se o nejběžnější příklad procesu vnímání**
- **Počáteční znalost**
  - známe počáteční polohu
    - **stopování** (tracking)
  - počáteční poloha neznámá
    - **globální lokalizace** – změní se na stopování, jakmile je poloha (objektu / robota) odhalena
- **Přechodový model**
  - robot se pohybuje v rovině, známe přesnou mapu, $X_t = (x_t, y_t, \theta_t)$
  - posuvná rychlost v_t, rotační ω_t
    - deterministická stavová predikce



$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t \Delta t \cos\theta_t \\ v_t \Delta t \sin\theta_t \\ \omega_t \Delta t \end{pmatrix}$$



LOKALIZACE POKRAČOVÁNÍ

- **Skuteční roboti jsou poněkud méně předvídatelní (nejsou determinističtí)**
  - použijeme Gaussovskou distribuci N, kde f(X_t, v_t, ω_t) bude střední hodnota a kovariance Σ_x

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, v_t, \omega_t) = N(\hat{\mathbf{X}}_{t+1}, \mathbf{\Sigma}_x)$$

- **Senzorový model**
  - budeme se vztahovat k orientačnímu bodu (x_i,y_i), senzory detekují orientační bod
    - oznámí vzdálenost a natočení vůči orientačnímu bodu
    - deterministicky by senzor hlásil

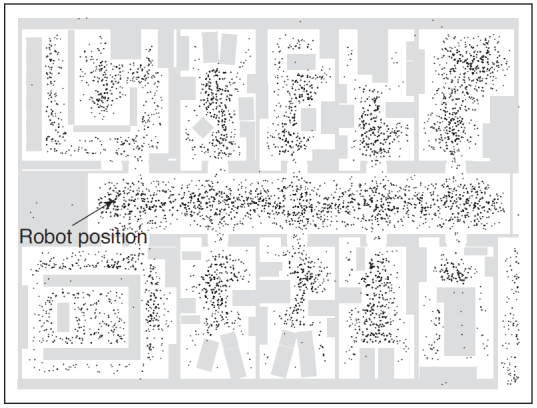
$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

    - ale opět musíme počítat s nespolehlivostí senzorů, použijeme Gaussovskou distribuci

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = N(\hat{\mathbf{z}}_t, \mathbf{\Sigma}_z)$$

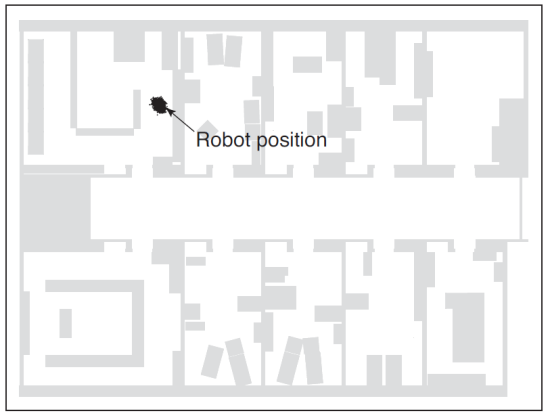
LOKALIZACE MONTE CARLO (MCL)

- **Založeno na částicové filtraci**
  - reprezentuje přesvědčení jako mračno částic (vzorků), které odpovídají stavům



(a)

(b)



(c)

function MONTE-CARLO-LOCALIZATION(a, z, N, $P(X'|X,\ v,\ \omega)$, $P(z|z^*)$, m) **returns** a set of samples for the next time step

inputs: a, robot velocities v and ω
z, range scan $z_1, \ldots, z_M$
$P(X'|X,\ v,\ \omega)$, motion model
$P(z|z^*)$, range sensor noise model
m, 2D map of the environment

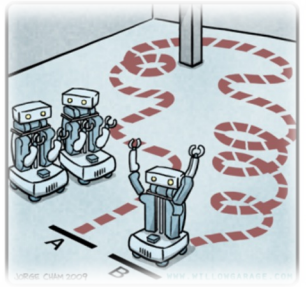
persistent: S, a vector of samples of size N

local variables: W, a vector of weights of size N
S', a temporary vector of particles of size N
W', a vector of weights of size N

if S is empty **then** /* initialization phase */
  for $i = 1$ to N **do**
    $S[i] \leftarrow$ sample from $P(X_0)$
for $i = 1$ to N **do** /* update cycle */
  $S'[i] \leftarrow$ sample from $P(X'|X = S[i], v, \omega)$
  $W'[i] \leftarrow 1$
  for $j = 1$ to M **do**
    $z^* \leftarrow$ RAYCAST($j, X = S'[i], m$)
    $W'[i] \leftarrow W'[i] \cdot P(z_j|\ z^*)$
$S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S', W')
return S

Přechodový a senzorový model na vstupu

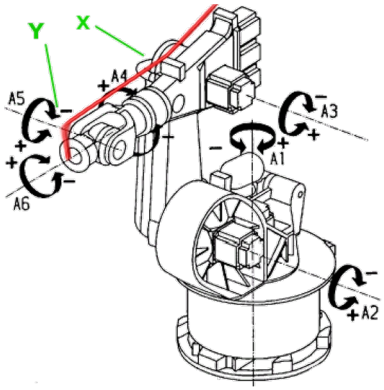
Jak odpovídá pozorování senzorem simulaci pozorování ve vzorku



PLÁNOVÁNÍ POHYBU

- **Typy pohybů**
  - **z bodu do bodu** (point-to-point)
    - celý robot, nebo jen efektor
  - **souladný** (compliant)
    - robot v kontaktu s překážkou

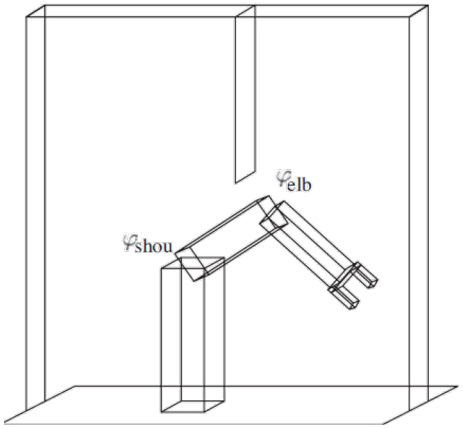
- **Konfigurační prostor**
  - poloha, orientace, natočení kloubů
  - **hledání cesty** (v konfigurační prostoru)
    - cesta spojující počáteční a cílovou konfigurace
    - v robotice se pohybujeme ve **spojitém prostoru**

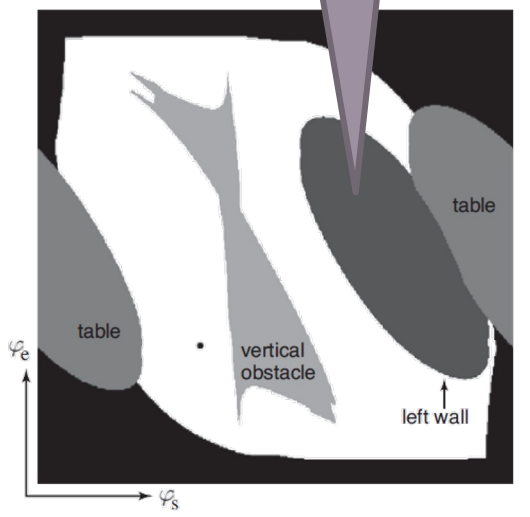


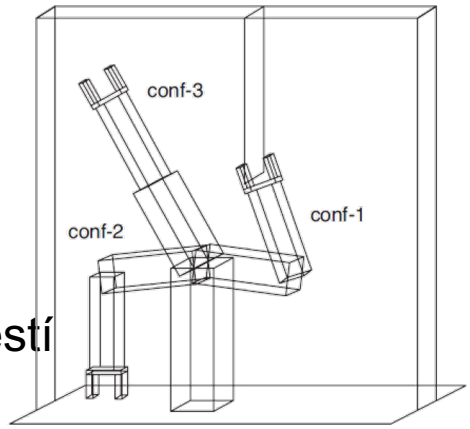
- **Jak zvládnout spojitost**
  - rozklad na **buňky**
  - **skeletonizace**
    - spojitý problém hledání cesty se převede na hledání cesty v **diskrétním prostoru (grafu)**
- Pro zjednodušení **nehledíme na neurčitost, vše deterministické**

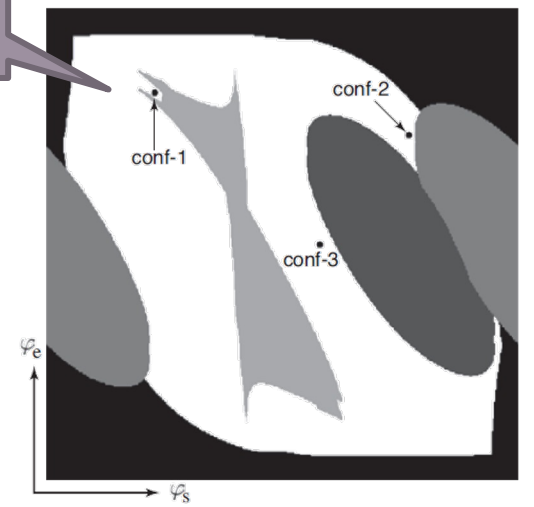
KONFIGURAČNÍ PROSTOR

- **Reprezentace pomocí pracovních souřadnic**
  - pozice prvků robota
    - zápěstí
      - (x_e, y_e)
    - kleští
      - (x_g, y_g)
  - plně popisuje pozici
  - vhodná na detekci kolizí
  - ne všechny souřadnice jsou možné
    - zápěstí a kleště jsou pevně spojeny
- **Reprezentace pomocí konfigurací**
  - natočení kloubů atd.
    - kloub ramena a zápěstí
      - $(\varphi_{shou}, \varphi_{elb})$

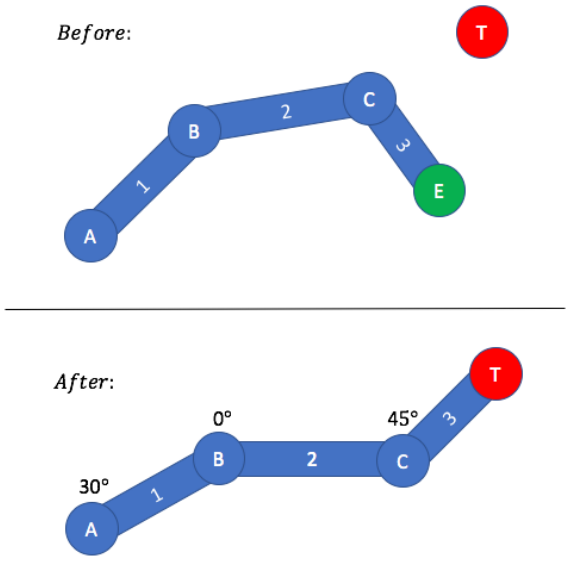








KINEMATIKA



- **Dopředná kinematika**
  - známe otočení kloubů, tj. konfiguraci
    - chceme určit polohu efektoru

konfigurace (φ_{shou},φ_{elb}) → pracovní souřadnice (x_g,y_g)

lineární transformace

- **Inverzní kinematika**
  - známe pozici efektoru reprezentovanou pracovními souřadnicemi
    - chceme určit konfiguraci

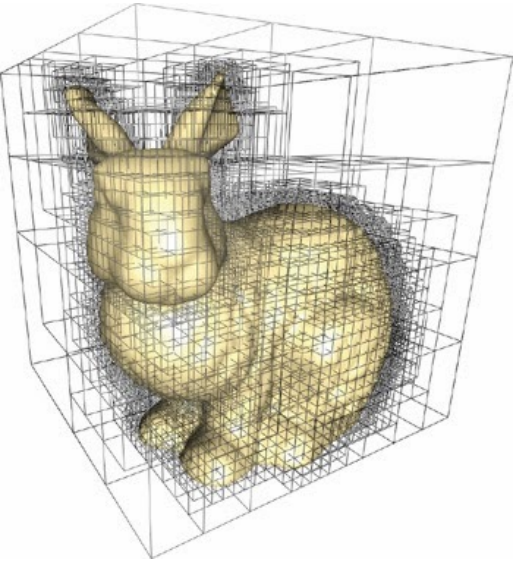
pracovní souřadnice (x_g,y_g) → konfigurace (φ_{shou},φ_{elb})

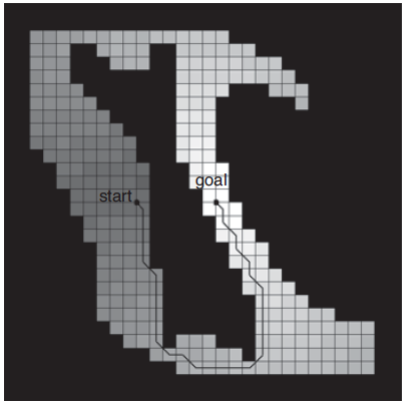
složitá transformace

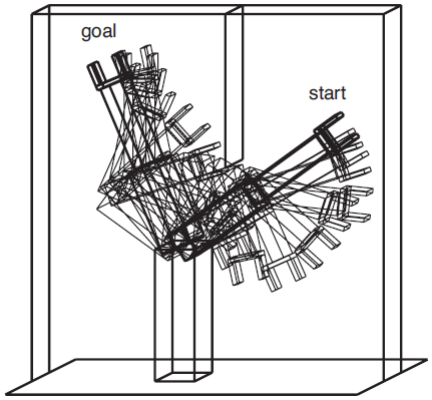
  - řešení rovnic, více DOF → více řešení (nekonečně mnoho)
    - hledáme řešení minimalizující např. energii, opotřebení

ROZKLAD NA BUŇKY

- **Konečný počet spojitých buněk**
  - hledání cesty je v rámci buňky „snadné“
    - například jej lze řešit lineární interpolací (pohyb podél úsečky)
- **Vlastnosti**
  - ✓ jednoduché na implementaci
  - ✗ komplikuje se s rostoucí dimenzí
  - ✗ jak naložit s polo-obsazenými buňkami
    - rozdělení na 2^d menších buněk, d je dimenze
      ve 2D/3D grafice známé jako kvadrantový/oktantový strom



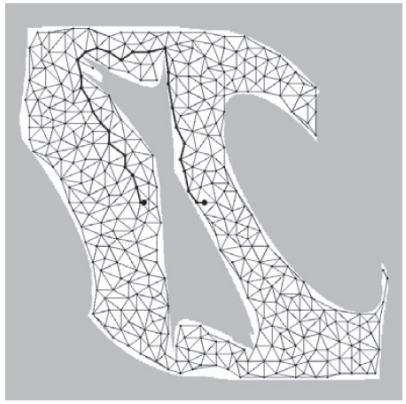




Stupně šedé určují vzdálenost do cíle, lze určit **Bellman-Fordovým algoritmem**

SKELETONIZACE

- **Převod hledání cesty na úlohu v jedné dimenzi**
  - **Voroného** diagram
    - množina bodů, které mají stejnou vzdálenost ke dvěma a více překážkám
    - tvoří graf, v němž je problém hledání cesty jednodimenzionální
      - na začátku vstup v počáteční konfigurace do Voroného diagramu
      - na konci výstup do cílové konfigurace
        - vstup i výstup **po úsečce**
    - **obtížná** konstrukce
  - **pravděpodobnostní mapa**
    - generujeme náhodné konfigurace
    - spojíme ty, mezi nimiž vede „snadná“ cesta
    - generovat více
      - v obtížných místech
      - v součinnosti s prohledáváním, prohledávání určí slibné místo



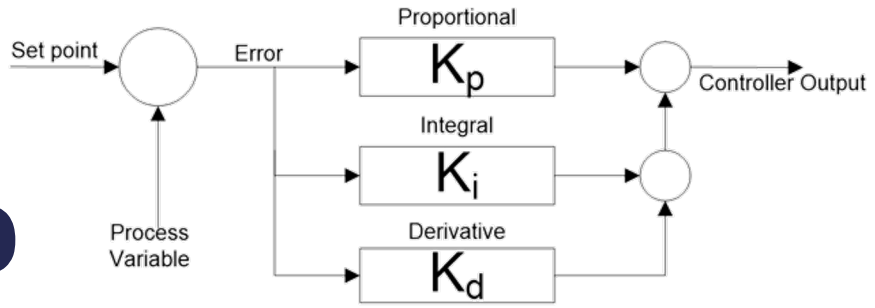
DYNAMIKA A ŘÍZENÍ

- **Nutno uvažovat nejen kinematický stav (konfiguraci), ale i <u>rychlost</u>** → dynamický stav
  - robot má nějakou hmotnost, nerozjede se a nezastaví okamžitě
- **Plánování s dynamickými stavy**
  - příliš obtížné, řešení diferenciálních rovnic
    - použitelné jen pro velmi jednoduché roboty
- **Alternativa – metoda kontroleru**
  - je-li hodnota nějaké stavové proměnné x_t odchýlena od požadované y(t), robot ji kompenzuje působením a_t (síla efektoru)
  - kontroler P - **proporcionální**
    - může oscilovat

$$a_t = K_P(y(t) - x_t)$$

  - kontroler PD – proporcionálně derivační
    - derivace působí jako tlumič
    - na náhlou velkou změnu reaguje pomalu
    - na stálé odchylování reagovat přestane (opotřebení)

$$a_t = K_P(y(t) - x_t) + K_D\frac{\partial(y(t) - x_t)}{\partial t}$$

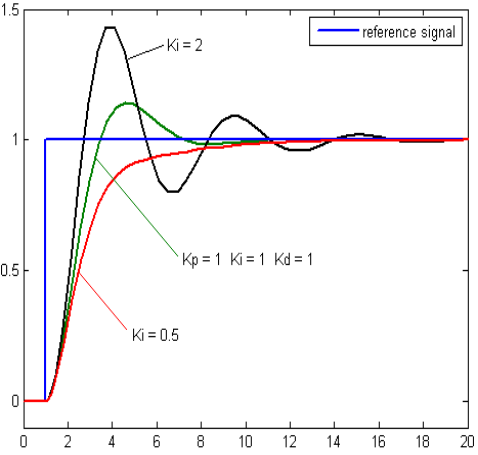


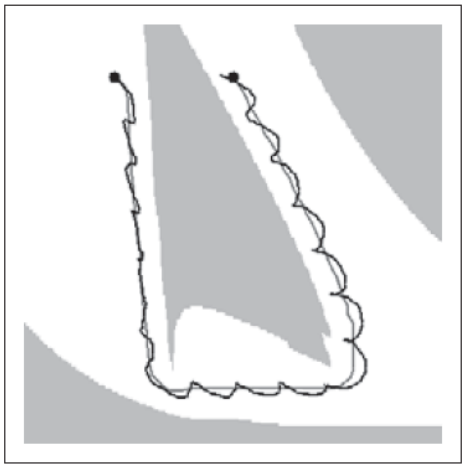
KONTROLER PID

- **PID – proporcionálně derivačně integrační**

$$a_t = K_P(y(t) - x_t) + K_I \int (y(t) - x_t)dt + K_D \frac{\partial(y(t) - x_t)}{\partial t}$$

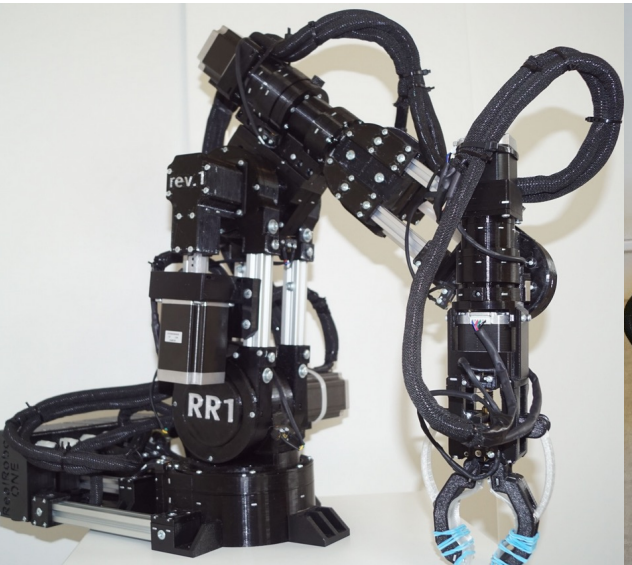
  - bere v potaz systematické externí působení
    - zahrnuto v integračním faktoru
  - nejčastěji používaný typ kontroleru v robotice (i v průmyslu)

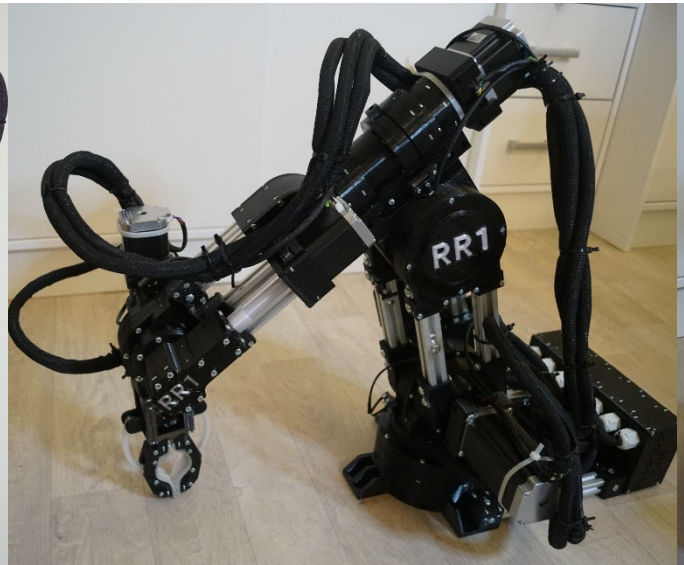


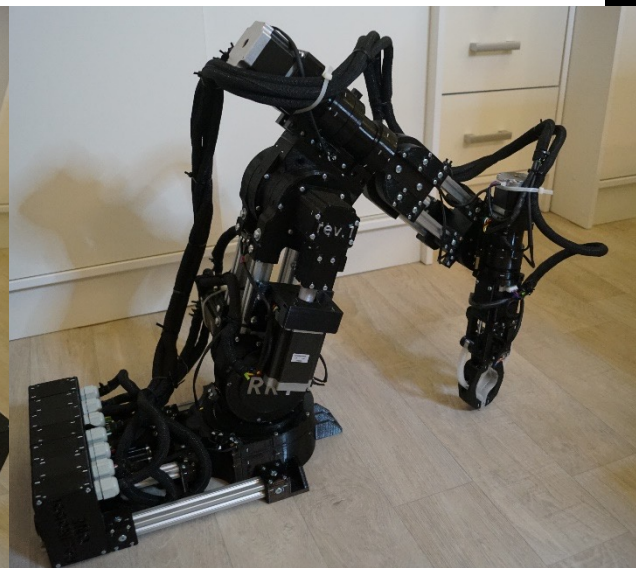


KONSTRUKCE ROBOTA

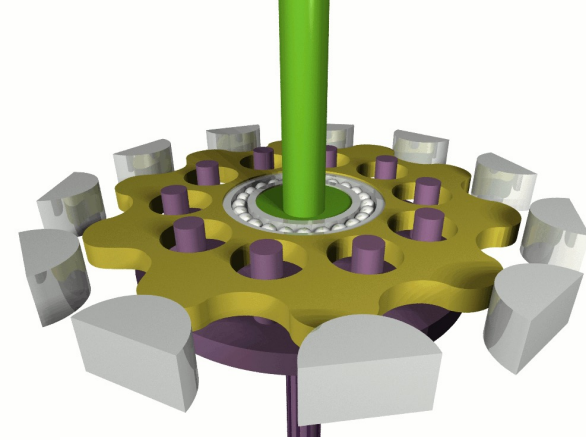
- Robotická paže RR1: Real Robot One
  - 6 kloubů + 1 kloub v kleštích, uzavřená smyčka
  - 50 cm výška, dosah cca. 80cm
  - 14 kg robotické rameno + 8 kg řídící jednotka
  - nosnost = cca. 2kg, opakovatelnost = teprve se měří
  - výroba: CAD, 3D tisk, šroubovák, páječka, trpělivost



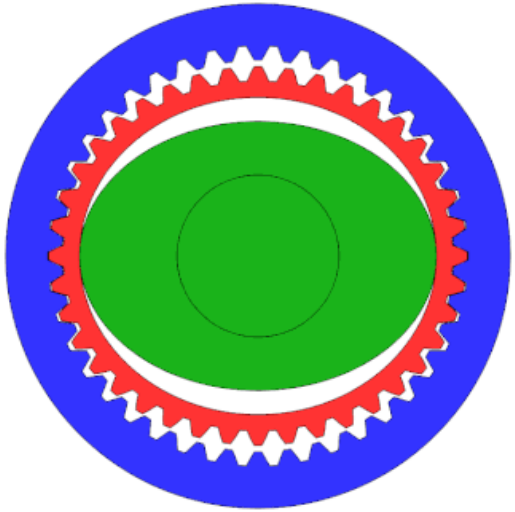


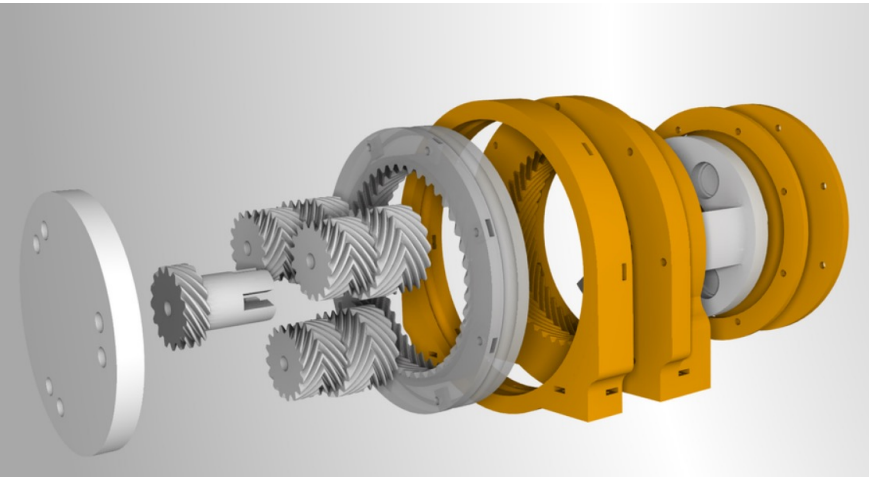


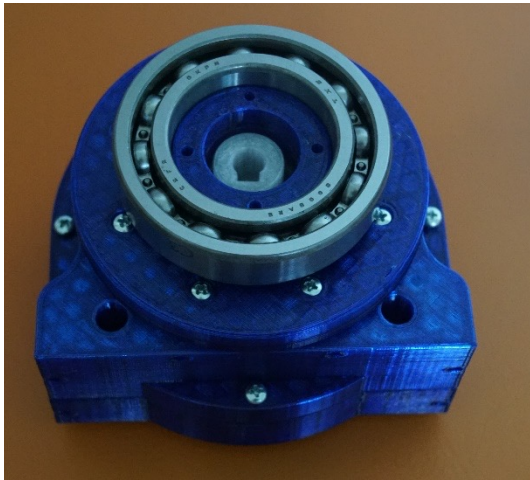
PŘEVODOVKY



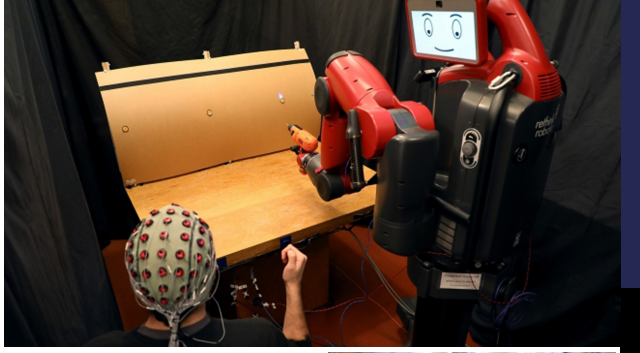
- **Planetová**
  - používá se v RR1, jednoduchá konstrukce, určitá vůle (backlash)
- **Cykloidní**
  - malá vůle, náročnější na konstrukci
- **Harmonická**
  - téměř žádná vůle, velmi náročná na konstrukci



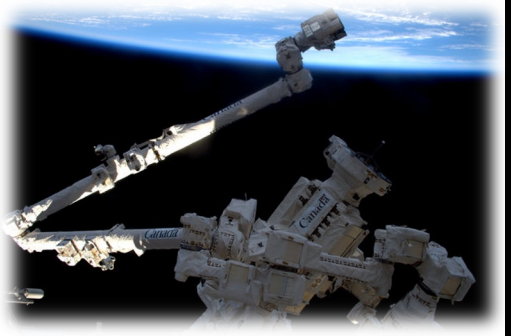




APLIKAČNÍ DOMÉNY

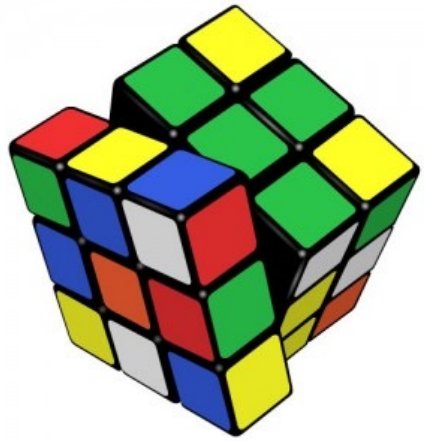


- **Humanoidní, nehumanoidní (animální či jiný)**
- **Pomocník člověka**
  - doprava, logistika, průzkum, zdravotnictví – operační robot/rehabilitační robot, osobní služby (Pepper)
- **Vylepšování člověka**
  - bio-feedback, neurální propojení, komunikace, exo-skelet
- **Nahrazování člověka**
  - robot recepční, robot společník (Japonsko)
- **Vojenství**
  - drony, bezpilotní letouny (vysoká G), autonomní tanky
- **Zábava**
  - hračky (Sony Aibo, Nao), robotický fotbal
- **Každý den nové aplikace**



PROHLEDÁVÁNÍ STAVOVÉHO PROSTORU

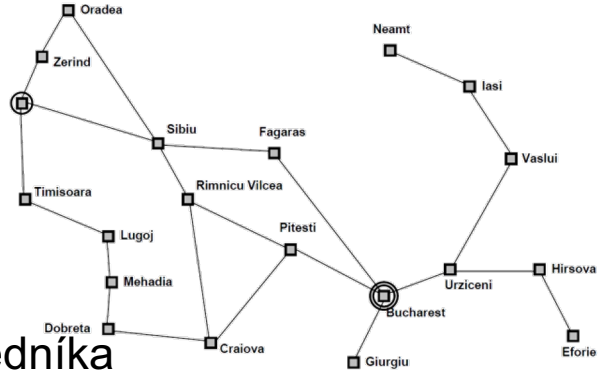
- **Několik kroků vedoucích k řešení úlohy**
  - Formulace **cíle**
    - požadavky → žádoucí stav světa
  - Formulace **úlohy**
    - jak vypadají stavy světa a akce, které jej mění
  - **Vyřešení** úlohy
    - určení posloupnosti stavů/akcí vedoucích k cíli
  - **Provedení** řešení
    - vykonání nalezené posloupnosti akcí
    - může dojít k chybám při vykonávání



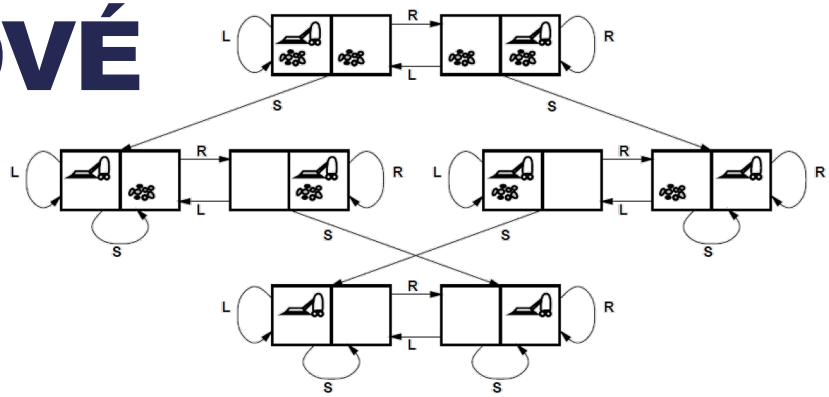
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

FORMULACE ÚLOHY

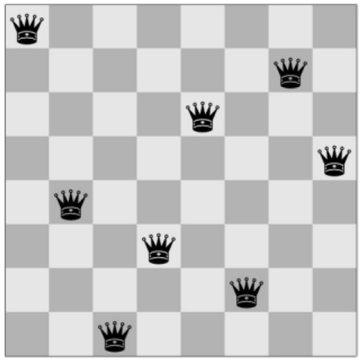
- **Situace**
  - „agent se nachází v **Rumunsku** ve městě **Arad** a chce si užít dovolenou, tj. opálit se, učit se Rumunsky, navštívit památky, okusit noční život, atd.“
- **Zjednodušení**
  - Přijme cíl = jet do Bukurešti
- **Dobře formulovaná úloha a řešení**
  - Počáteční stav
    - `in(Arad)`
  - Popis akcí, zpravidla určené pomocí funkce následníka
    - pro daný stav *x* vrací dvojici *(y,a)* následného stavu *y* a akce *a*, která nás převede z *x* do *y*
    - `{(go(Sibiu),in(Sibiu));(go(Timisoara),in(Timisoara);go(Zerind),in(Zerind)}`
  - Test, zda daný stav je cílový
    - například výčtem: `{in(Bucharest)}`
  - **Řešení**
    - Posloupnost akcí a stavů - cesta
      - **Vrcholy** na cestě – odpovídají stavům, **hrany** – odpovídají akcím

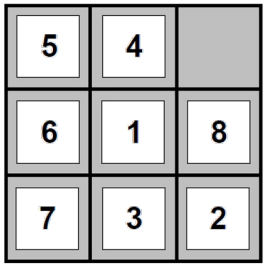


PŘÍKLADY **HRAČKOVÉ**

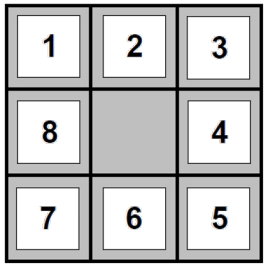


- **Původně zkoumány v rámci mikro-světů**
  - důležité pro vývoj, testování a porovnávání algoritmů
- **Vysavač**
  - úkolem je vysát smetí
    - **stavy:** pozice vysavače a smetí
    - **akce:** vysát smetí, přesun
- **8 královen (obecně N královen)**
  - umístit královny na šachovnici bez vzájemného ohrožení
    - **stavy:** rozmístění královen
    - **akce:** přidání královny
- **Lloydova osmička (Lišák)**
  - srovnat kameny na hracím poli
    - **stavy:** rozložení kamenů
    - **akce**: pohyb kamenem na volné místo





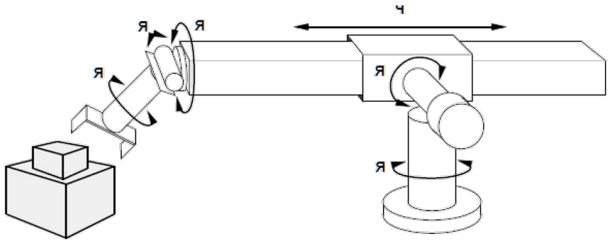
Start State

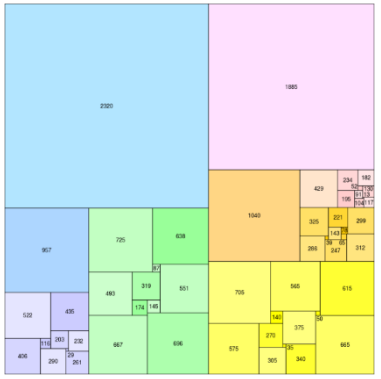


Goal State

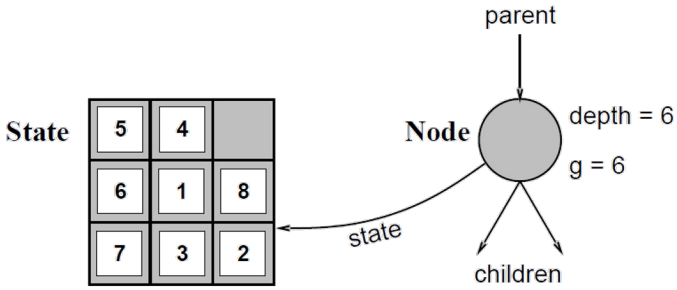
PŘÍKLADY **REÁLNÉ**

- **Těžké pro původní algoritmy UI**
  - obrovský stavový prostor
    - kombinatorická exploze
  - důležité pro praxi
    - průmysl, doprava, obchod, věda
- **Multi-agentní hledání cest, multi-robotické plánování pohybu**
  - mnoho robotů, každý má svůj počátek a cíl
    - **Stavy:** rozložení robotů, konfigurace
    - **Akce:** pohyby robotů
- **Montáž výrobků**
  - Robot má za úkol sestavit výrobek
    - **Stavy:** konfigurace robota a dílů
    - **Akce:** vše, co robot může udělat
- **Perfektní čtverec**
  - UI v roli servisní vědy pro matematiku

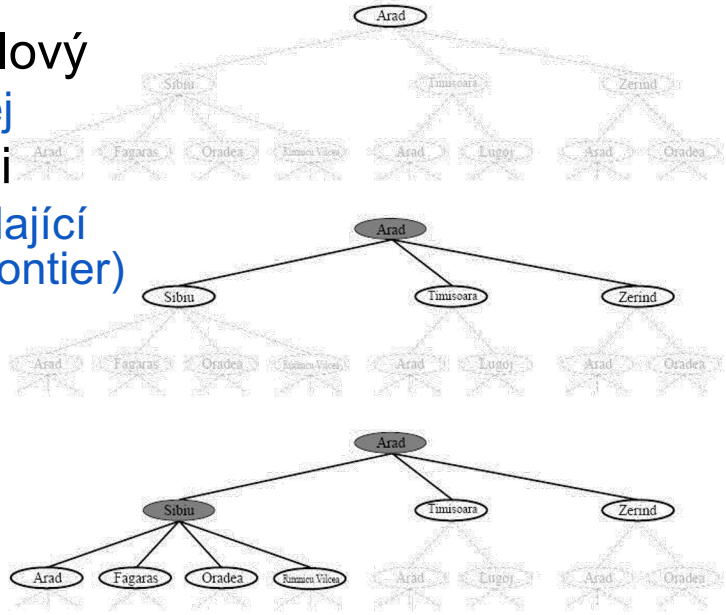




NEINFORMOVANÉ PROHLEDÁVÁNÍ



- **Předběžnosti**
  - stav × uzel
    - **stav** – stav světa
    - **uzel** – datová struktura obsahující stav a další informace (rodič, cena, …)
- **Prohledáváme stavový prostor**
  - Začneme v počátečním stavu
  - Otestujeme, zda počáteční stav není cílový
    - je-li cílový, máme řešení a vrátíme jej
  - Není-li cílový, **vybereme** uzel k expanzi
    - expanzí vzniknou nové uzly, odpovídající následníkům, přidáme do „okraje“ (frontier)



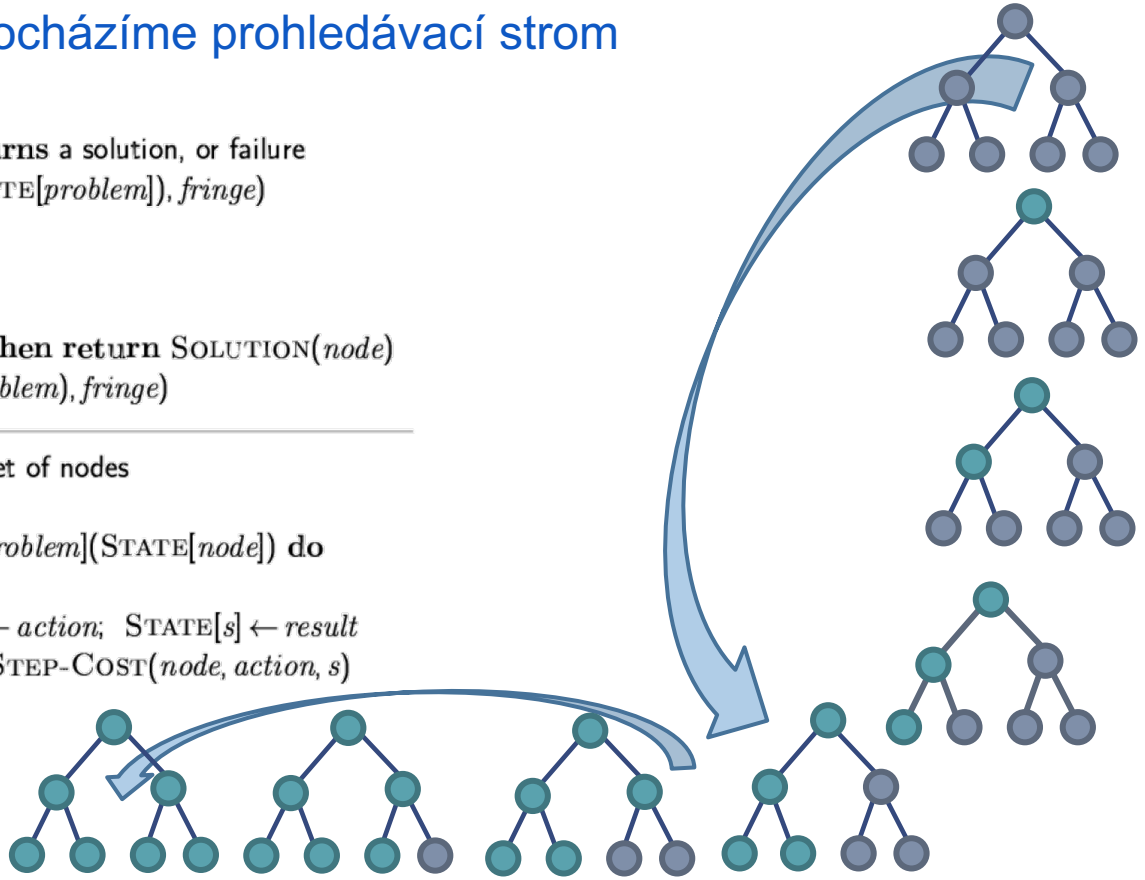
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

PROHLEDÁVÁNÍ **DO HLOUBKY**

- **DFS**: vybíráme uzel v **největší hloubce**
  - okraj implementujeme jako zásobník (**LIFO**)
    - ve výsledku procházíme prohledávací strom **zleva doprava**

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

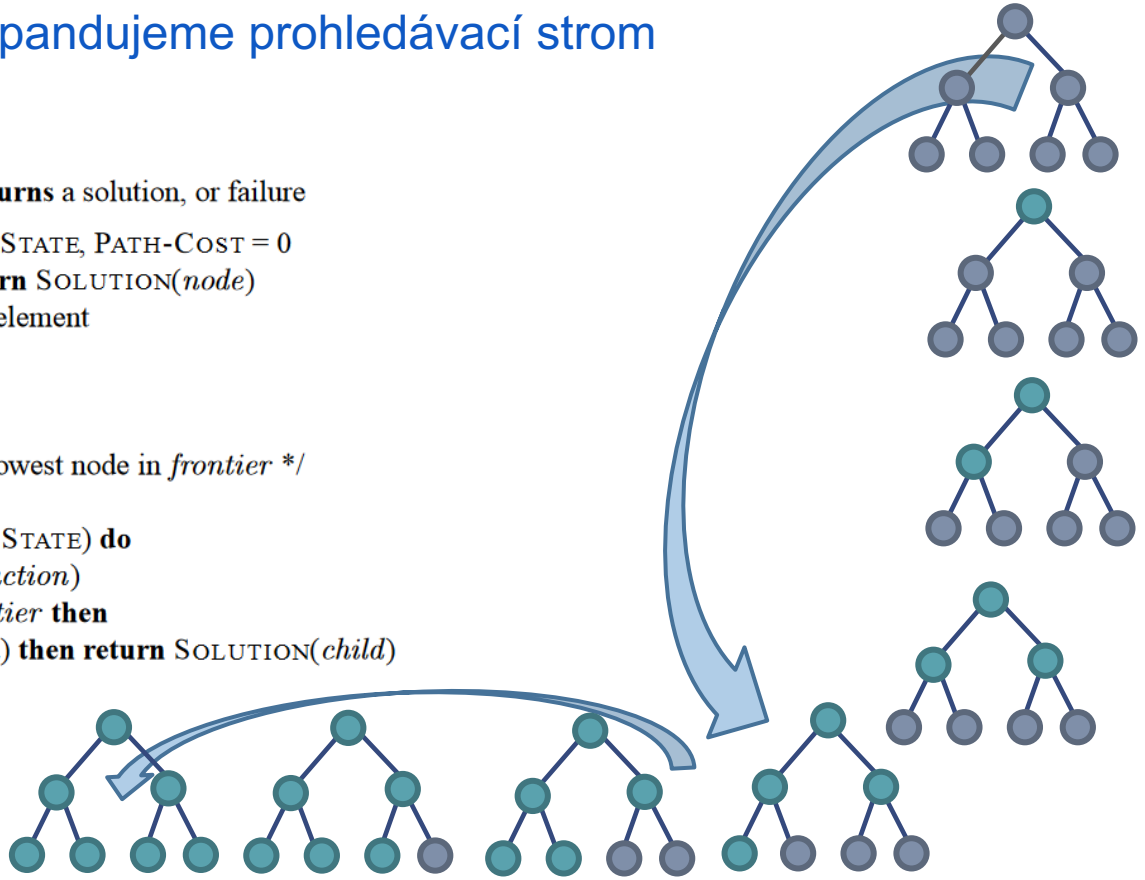
```
function EXPAND( node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;  ACTION[s] ← action;  STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```



PROHLEDÁVÁNÍ **DO ŠÍŘKY**

- **BFS**: vybíráme uzel v **nejmenší hloubce** (s nejmenší cenou)
  - okraj implementujeme jako frontu (**FIFO**)
    - ve výsledku expandujeme prohledávací strom **po vrstvách**

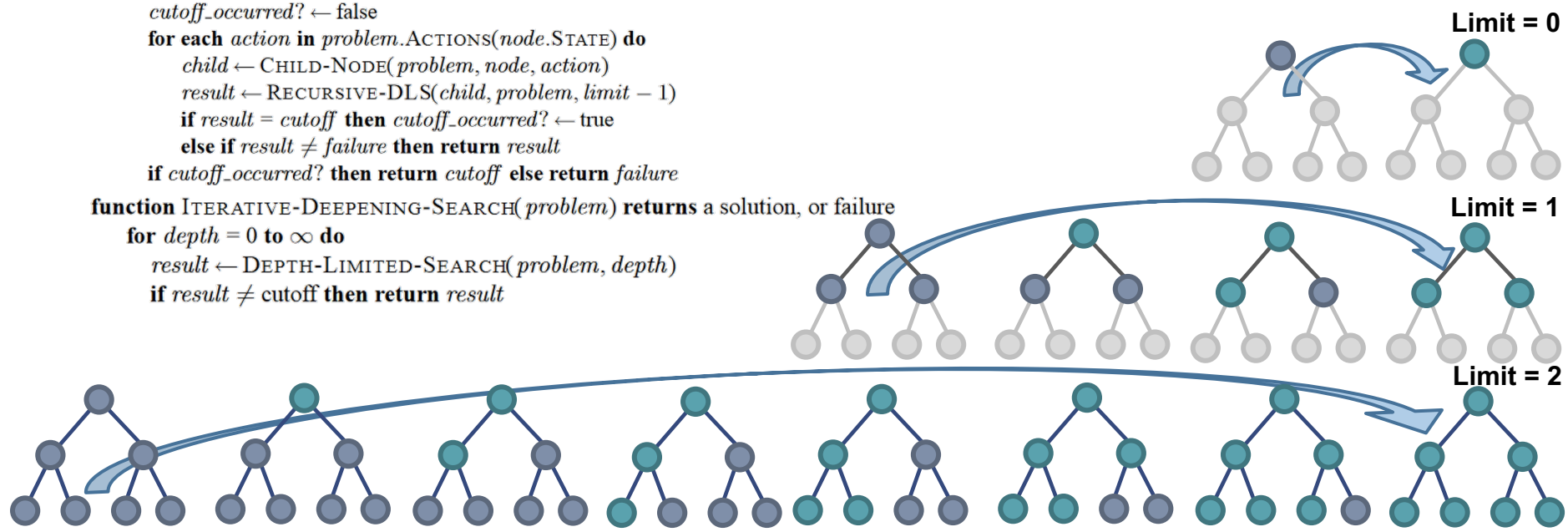
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```



ITERATIVNĚ DO HLOUBKY

- **IDS**: kombinuje výhody DFS a BFS
  - zaručuje **optimalitu** (za předpokladu monotonie) a **úplnost**
  - malé paměťové nároky (jako DFS)

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
      cutoff_occurred? ← false
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          result ← RECURSIVE-DLS(child, problem, limit − 1)
          if result = cutoff then cutoff_occurred? ← true
          else if result ≠ failure then return result
      if cutoff_occurred? then return cutoff else return failure

function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
    for depth = 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
      if result ≠ cutoff then return result
```



VLASTNOSTI DFS A BFS

- **Úplnost**
  - Algoritmus vždy skončí a vrátí správnou odpověď
- **Časová složitost, paměťová složitost**
  - b – větvící faktor, d – nejmenší hloubka obsahující řešení, m – maximální hloubka stromu, l – limit na hloubku

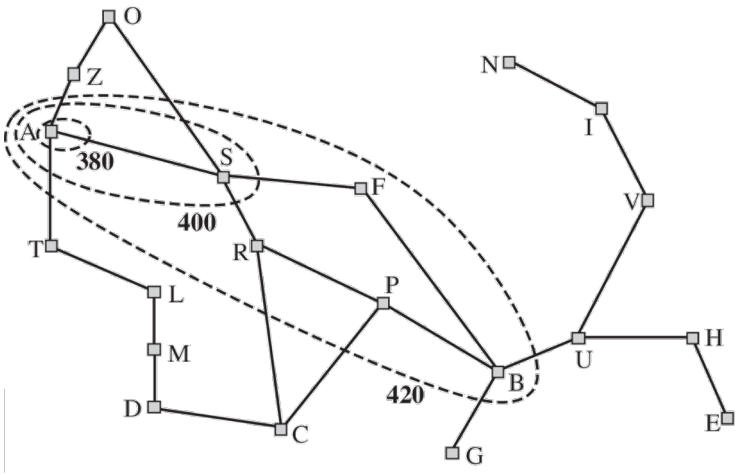
	DFS	BFS	DLS	IDS
Úplnost	NE	ANO	NE	ANO
Čas	$O(b^m)$	$O(b^{d+1})$	$O(b^l)$	$O(b^d)$
Paměť	$O(bm)$	$O(b^{d+1})$	$O(bl)$	$O(bd)$
Optimalita	NE	ANO*	NE	ANO*

- **Ukázka výpočtu časové složitosti pro IDS:**
  - $d \cdot b^1 + (d-1) \cdot b^2 + (d-2) \cdot b^3 + (d-3) \cdot b^4 + \ldots + 2 \cdot b^{d-1} + b^d = O(b^d)$

PROHLEDÁVÁNÍ **HEURISTICKY**

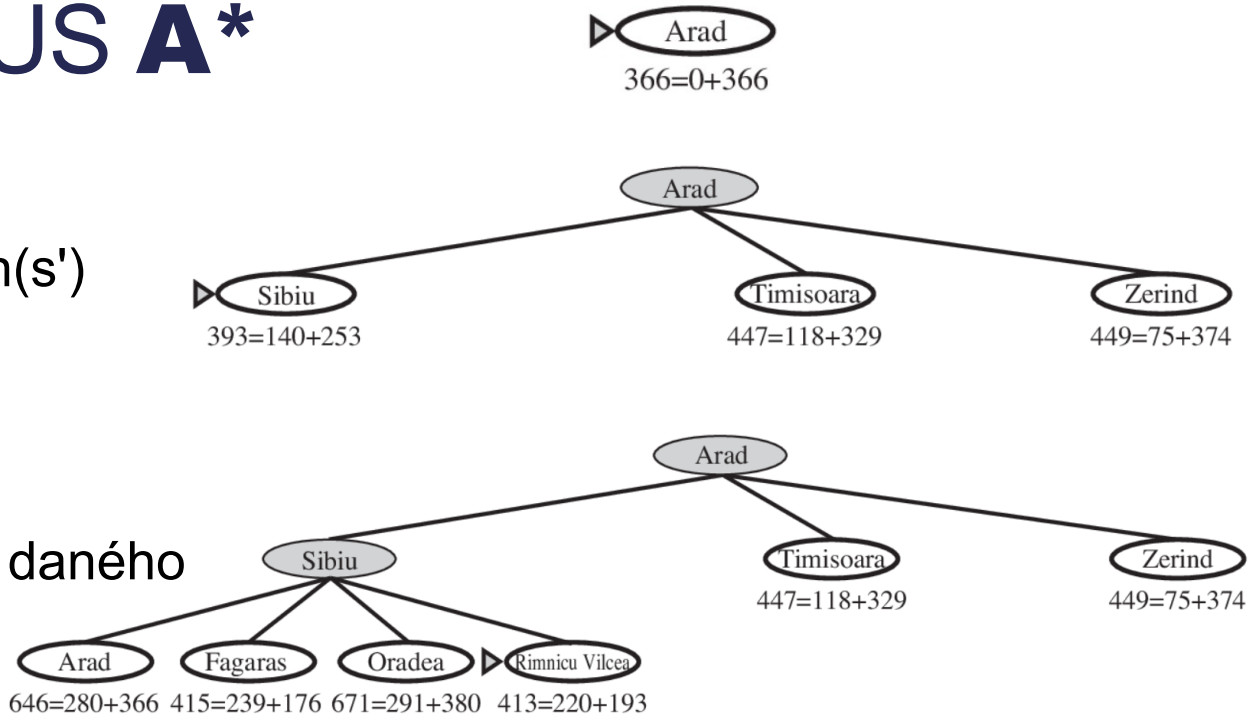
- **Algoritmus A*: okraj** implementujeme uspořádaně (pro snadné hledání minima/maxima)
  - stavy ohodnotíme pomocí *f* = *g* + *h*
    - *f* = odhad ceny cesty z **počátku do cíle**
    - *g* = cena cesty z **počátku do** daného **stavu**
    - *h* = *odhad ceny cesty z daného* ***stavu do cíle***
  - expandujeme stav s nejmenším *f*

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
    return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```



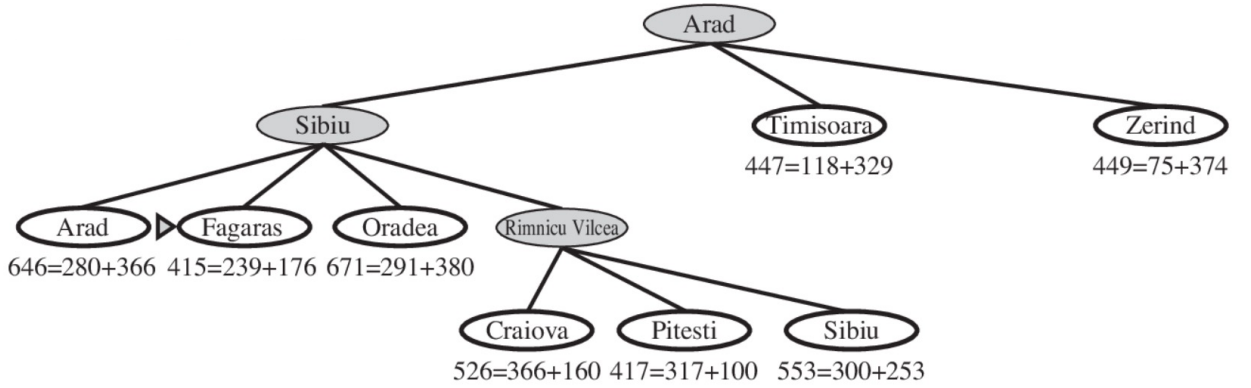
ALGORITMUS **A***

- **Vlastnosti**
  - implikuje
    - h(s) ≤ c(s,s') + h(s')
    - ⇓ **monotonie**
    - h(s) ≤ h*(s)
    - **přípustnost**
      - h* cena cesty z daného stavu do cíle
- ⇒ nalezení **optima**



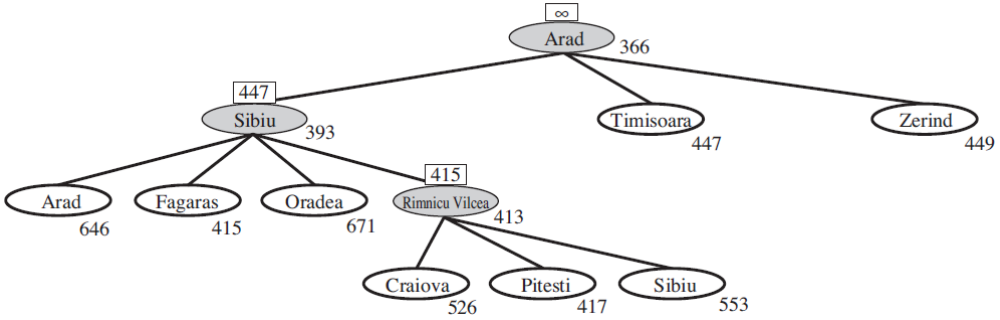
Vzdušná vzdálenost
monotonní h

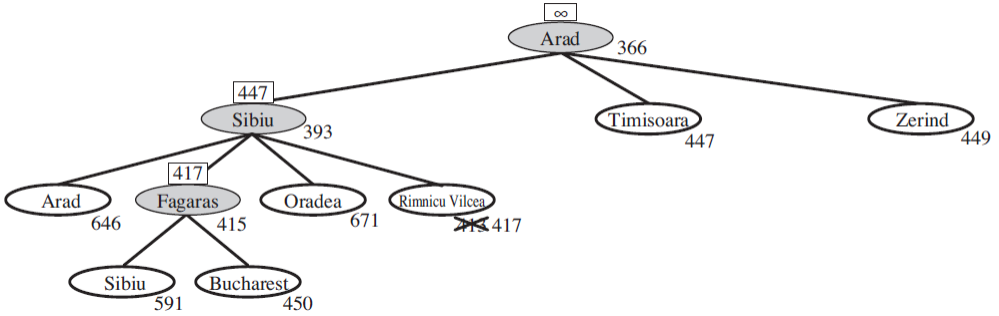
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

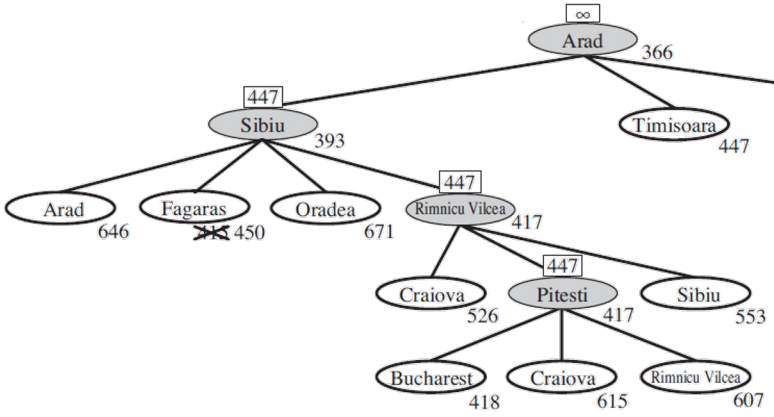


ALGORITMUS **RBFS**

- **Napodobuje A* podobně jako IDS napodobuje BFS**
  - šetří paměť, zachovává optimalitu







ZÁVĚREČNÉ POZNÁMKY

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

- **Pozor na opakující se stavy**
  - Zapamatovat
    - znovu neexpandovat
- **Rozšiřující témata**
  - Prohledávání s využitím **externí paměti** (disk, …)
  - **Téměř přípustná** heuristika
  - **Obousměrné** prohledávání (*bi-directional search*)
    - směrem od počátku i od cílového stavu najednou
    - v ideálním případě se prohledávací strom rozvine 2x do hloubky d/2, celková složitost $O(b^{d/2})$
  - Prohledávání s **částečnou informací**
    - neznámý počáteční stav
    - výsledek akce je nejistý
    - stavy a akce nejsou předem známé