Plánování ve 2D prostředí

Úloha: Nalezení přípustné cesty mezi danými body (start → cíl) prostředí

- Počáteční a koncové body cesty reprezentují v jednoduchém případě **stavy robotu** (v obecnějším případě spolu s dalšími veličinami robotu – angulární a tangenciální rychlostí a zrychlením podvozku)
- Kinematické vlastnosti robotu - fyzický rozměr, omezení na řízení atd. reprezentují **omezující podmínky**
- Řešení úlohy plánování trajktorie lze dosáhnout **prohledáváním stavového prostoru** takového robotu:

  - **Neinformované postupy** prochází všechny přípustné alternativy a poskytuje první nalezené řešení (neefektivní, řešení je zřídkakdy optimální)

  - **Informované postupy** upřednostňují alternativy splňující přídavné kritérium (kvantitativní ohodnocení situace *cenovou/penalizační funkcí* reprezentující použitou *heuristiku*)
    - Minimalizace délky trajektorie
    - Minimalizace vynaložené energie (potlačením tangenciálních a radiálních akcelerací, výběrem rovinné cesty...)
    - Minimalizace času k dosažení cíle (minimalizace délky trajektorie a zatáčení, se zahrnutím omezujících podmínek podvozku – stabilita a maximální akcelerace, atd.)

Od plánování k realizovatelnému plánu (postup v mobilní robotice) I

- Metodami prohledávání stavového prostoru (v jediném kroku) lze dosáhnout přímo použitelného plánu pro činnost robotu jen v situacích, kdy je popis jeho vlastností a omezení plynoucích z prostředí úplný (tj. zahrnuje i geometrickou informaci, nezbytnou pro pozdější vykonání plánu).

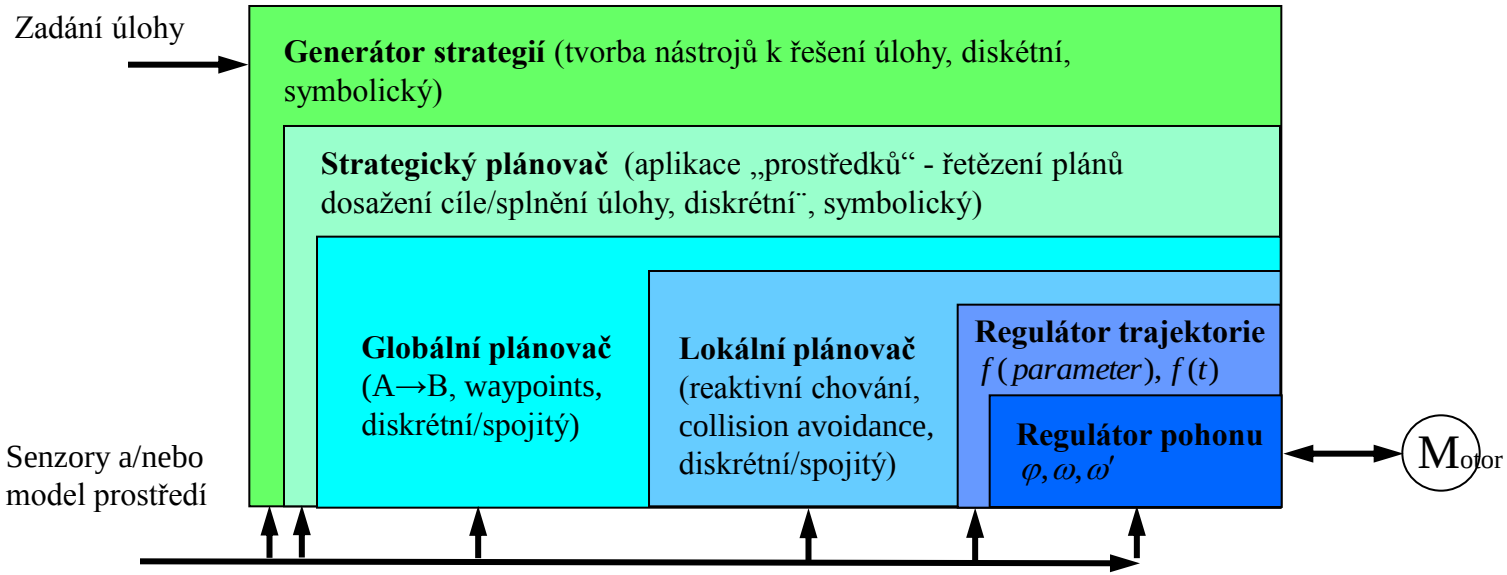
  - Přímá tvorba „spustitelného“ plánu je nemožná v případě použití pouhé symbolické reprezentace dat/znalosti (např. relační modely prostředí)

  - Řešení v jediném kroku při znalosti geometrické informace je nevýhodné pro:
    - **Vysokou složitost popisu** → vysoká výpočetní náročnost nalezení plánu,
    - Problematický **mechanismus přeplánování**, který vede k nutnosti přepracování celého plánu, atd.

- V ostatních případech je nutné (vhodné) plán vytvářet postupně, počínaje plánem přibližným, jenž se postupně zpřesňuje

Od plánování k realizovatelnému plánu (postup v mobilní robotice) II

K potlačení problému složitosti a zachování flexibility možné změny/úpravy/zpřesnění plánu se obvykle plánovací proces rozkládá do několika úrovní, např.:



Od plánování k realizovatelnému plánu (postup v mob. robotice) III

– Řešení se opírá o využití zjednodušeného popisu prostředí (popis se sníženou „rozlišovací schopností“) k vytvoření plánu plánovačem, nicméně:

- **Zjednodušený popis prostředí** → snížení výpočetní náročnosti nalezení plánu, možnost hierarchického nalézání plánů, zjednodušené přeplánování na lokální úrovni, atd.

- **Plánování nad zjednodušeným modelem** prostředí poskytuje pouze rámcové plány (např. „posloupnost lokalit, jimiž je nutné projet k dosažení cíle“) bez podrobného postupu, jak takové plány realizovat.

- **Fyzická realizace plánů** pak vyžaduje zohlednit další (lokální) omezující podmínky k převedení rámcového plánu do posloupnosti skutečných povelů pro motorický systém robotu (realizátor plánů)

Vlastnosti stavového prostoru pro řešenou úlohu (jednoduchá situace bez zahrnutí dynamiky robotu)

- Stav robotu odpovídá okamžitým souřadnicím robotu ve 2D prostředí (x,y,φ?)
- Prostředí robotu je reprezentováno **geometrickým modelem** (mapou/znalostí o prostředí) prostředí s jednoznačným určením atributu místa – jisté „volné“ a „obsazené“ oblasti (binární mřížka)
- Nalezení plánu je deterministickou úlohou v případě, kdy předpokládáme dokonalou znalost mapy prostředí

Reprezentace nejistoty:

- Nejistota modelu spočívá výhradně v jeho možné „neznalosti“ (tj. **nejistotě existence jistě určeného objektu** a nikoliv **nejistotě určení hranice jistě existujícího objektu**)
- Nalezení plánu je pak úlohou nedeterministickou z hlediska možnosti výskytu neznámého objektu

Vhodné postupy (jednotlivé podúlohy):

Prohledávání stavového prostoru:

- Algoritmus Dijkstra
- Algoritmus A*

Zahrnutí geometrických omezujících podmínek:

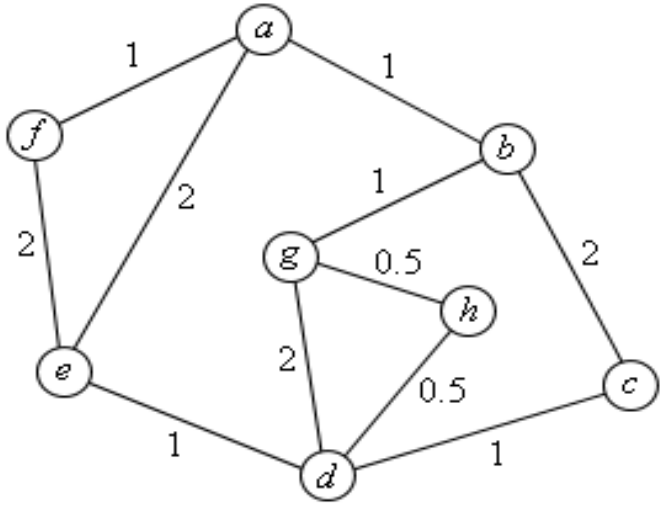
- Minkowského suma (dilatace)

Příprava realizace plánu:

- Vyhlazování, aproximace (lineární v celočíselné aritmetice)
  (Bresenhamův algoritmus, Digital Differential Analyzer - DDA algoritmus)

Dijkstra algoritmus

- Mějme ohodnocený (alternativně i orientovaný) graf
- Pro daný počáteční vrchol grafu algoritmus vyhledává cesty s nejnižší cenou (nejkratší), jimiž lze dosáhnout ostatní vrcholy zpracovávaného grafu (obecná situace).
- Výstupem je strom nejlevnějších cest do všech uzlů zpracovávaného grafu.
- Lze modifikovat pro vyhledávání „nejlevnější“ cesty do jediného cílového uzlu, přidáním testu na jeho dosažení.



Př. Vrcholy grafu reprezentují „města“ a hrany grafu a jejich ohodnocení mají význam existujících komunikací, resp. jejich vzdálenosti.

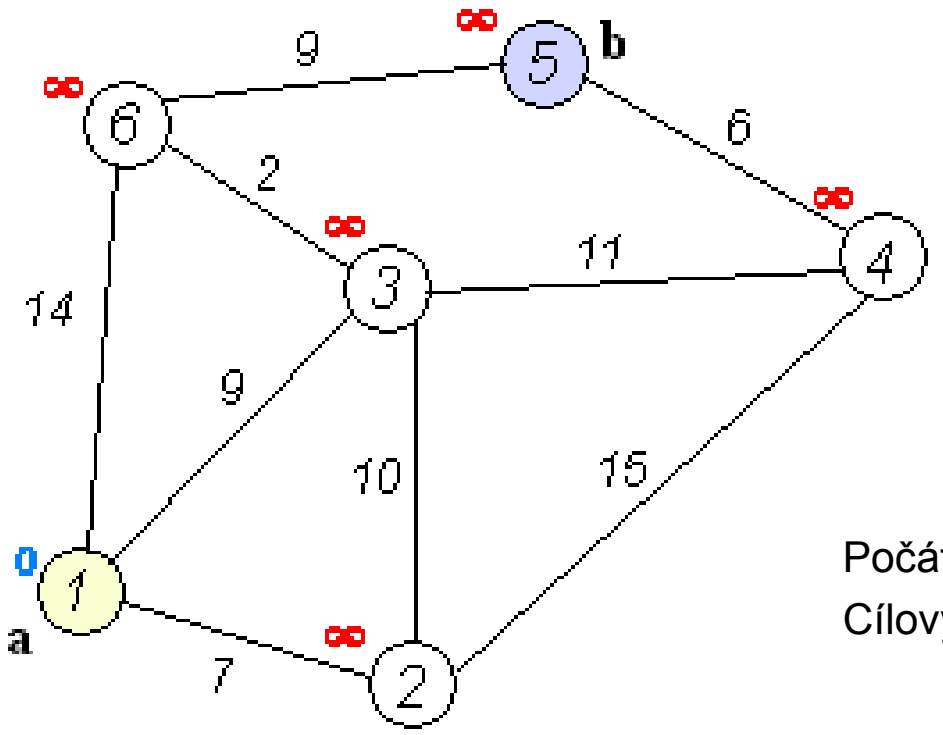
Dijkstra - popis:

- Zvolme počáteční uzel, z něhož bude prohledávání probíhat.
- Každému uzlu grafu přiřaďme číselné ohodnocení, reprezetující cenu jeho dosažení z počátečního uzlu.
- Algoritmus přiřazuje těmto vzdálenostem vhodné počáteční hodnoty, které v průběhu své činnosti zpřesňuje.

Kroky:

1. Přiřaď každému uzlu vzdálenost. Pro počáteční uzel nechť je 0, pro ostatní uzly nekonečno.
2. Označ všechy uzly jako dosud „nezpracované“.
3. Pro zvolený aktuální uzel vypočti jeho složenou vzdálenost od počátečního uzlu. Je-li hodnota vypočtené vzdálenosti nižší než hodnota aktuálně přiřazená tomuto uzlu, přepiš ji nově vypočtenou hodnotou
4. Předchozí krok proveď pro všechny uzly, sousedící s aktuálním uzlem a následně jej označ jako „již zpracovaný“ uzel. Aktuální uzel již nebude více zpracováván a jeho vzdálenost je minimální a fixovaná.
5. Jesliže byly zpracovány všechny uzly grafu, ukonči činnost. Jinak zvol za nový aktuální uzel takový, jenž má nejmenší hodnotu vzdálenosti (od počátečního uzlu) a pokračuj krokem 3.

Dijkstra - demo:



Počáteční uzel **„a“**

Cílový uzel **„b“**

```
function Dijkstra(Graph, source):
2       for each vertex v in Graph:             // Initializations
3         dist[v] := infinity ; // Unknown dist. Funct. from source to v
4         previous[v] := undefined ;            // Previous node in optimal path from source
5       end for ;
6       dist[source] := 0 ;                     // Distance from source to source
7       Q := the set of all nodes in Graph ; // All nodes in the graph are unoptimized - thus are in Q
8       while Q is not empty:                   // The main loop
9           u := vertex in Q with smallest dist[] ;
10            if dist[u] = infinity:
11                break ;                       // all remaining vertices are inaccessible from source
12            fi ;
13          remove u from Q ;
14          for each neighbor v of u:           // where v has not yet been removed from Q.
15            alt := dist[u] + dist_between(u, v) ;
16            if alt < dist[v]:                 // Relax (u,v,a)
17                dist[v] := alt ;
18                previous[v] := u ;
19            fi  ;
20          end for ;
21      end while ;
22      return dist[] ;
  end Dijkstra
```

Dijkstra – vlastnosti

Nejlevnější cesta mezi dvěma uzly:

- Pro nalezení pouze nejlevnější (nejkratší) cesty mezi počátečním a jediným zvoleným cílovým uzlem lze výpočet zastavit na *kroku 11* podmínkou *u = cílový uzel (target)*
- Nalezená nejlevnější cesta je pak rekonstruhovatelná prostým algoritmem:

```
1   S := empty sequence
2   u := target
3   while previous[u] is defined:
4       insert u at the beginning of S
5       u := previous[u]
```

Složitost výpočtu:

- Prostá implementace (při ukládání vrcholů *Q* ve struktuře seznamu nebo pole je výběr minima lineárním prohledáváním před všechny vrcholy *Q*).
  - Horní odhad složitosti výpočtu v pak je: $O(V^2 + E) \sim O(V^2)$
- V případě „řídkých grafů“, kdy $E << O(V^2)$, *lze v implementaci graf* reprezentovat popisem sousedností a s užitím mechanismu prioritních front (binárního a/nebo Fibonacciho.. zásobníku) lze dosáhnou horní meze složitosti: $O((E + V)\log V)$, *resp.* $O(E + V\log V)$

Algoritmus A*

- A* je zobecněnou variantou Dijkstra algoritmu a obdobně též *úplným algoritmem* (tj. nalézá řešení vždy, když existuje)
- Informovaný prohledávací algoritmus (tj. jednotlivé kroky prohledávání vyhodcují, jak optimálně postupovat)
- Snižuje složitost prohledávaných podgrafů př(xi výpočtu za podmínky, že je k dispozici informace (heuristika) o dolním odhadu ceny cesty do cílového uzlu grafu.
- Realizuje strategii “nejlepší nejdříve“ (best-first), poskytuje nejlevnější cestu od počátečního do (jediného) cílového uzlu grafu.
- Užitá heuristika (funkce) $f(x) = g(x) + h(x)$ má složky:
  - $g(x)$ je cena dosud vykonané cesty z počátečního do aktuálního uzlu grafu
  - $h(x)$ je přípustný (dolní) odhad ceny (vzdálenosti) cesty do cílového stavu; přípustnost odhadu je garantována nepřekročením skutečné ceny cesty do cíle (např. přímá vzdálenost do cíle)
  - Pokud $h(x)$ splňuje $h(x) \leq d(x,y) + h(y)$ pro každou hranu *s délkou* $d(x,y)$ mezi uzly x, y, je $h(x)$ tzv. monotónní → žádný uzel grafu nemusí být zpracován více než 1x (efektivní implementace)

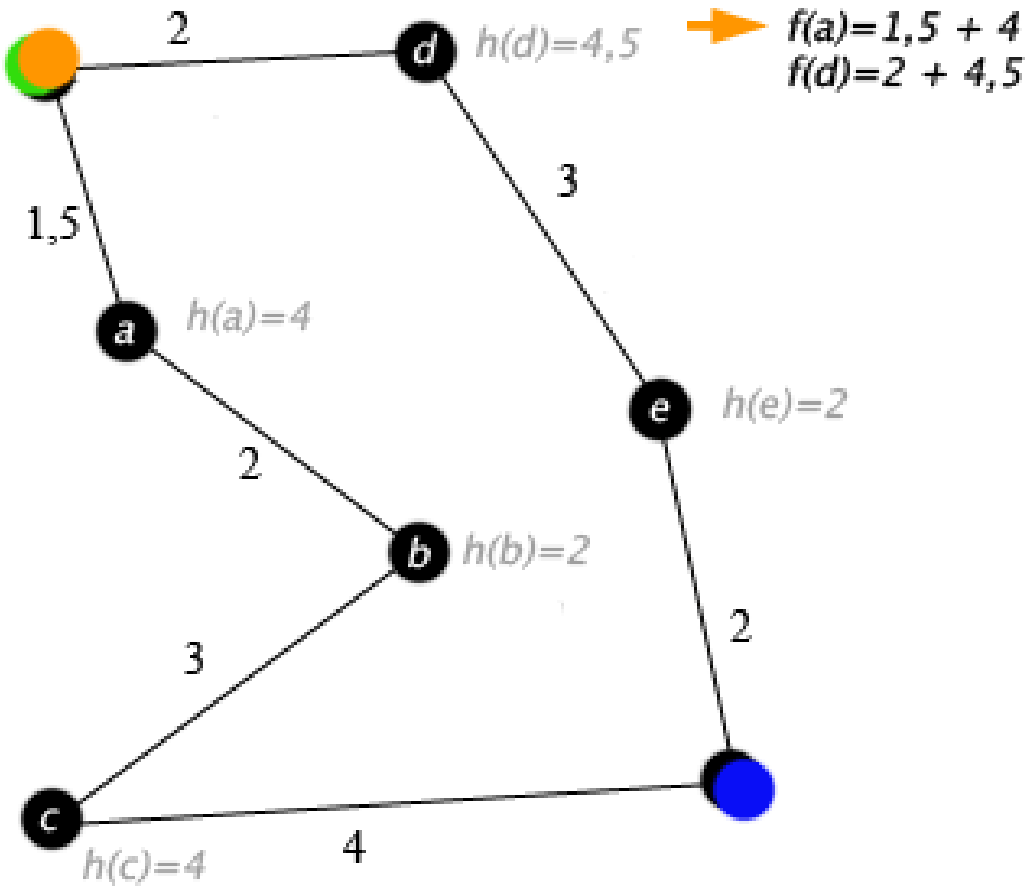
Koncepce A*

- A* prochází zpracovávaný graf podél nejlevnější (v geometrickém případě nejkratši) známé trajektorie.
- Během výpočtu je udržována struktura - uspořádaná prioritní fronta – alternativních úseků cesty podél určovaného řešení.
- Jestliže je v jakémkoliv kroku výpočtu zjištěno, že právě procházený úsek grafu směrem k cílovému řešení má vyšší cenu než jiný (z existujícího seznamu), je dražší úsek zamítnut a nahrazen levnějším.
- Předchozí proces je opakován do dosažení cíle.

- Odlišnost od ostatních informovaných algoritmů (informovaných, hladových) spočívá v zahrnutí již započtené ceny cesty z počátku do aktuálního uzlu, t.j. optimalizace není jen lokální (1-kroková) nýbrž globální (od počátku prohledávání).

Ilustrace postupu A*:

Počáteční uzel (zelená)
Cílový uzel (modrá)



Kroky A*

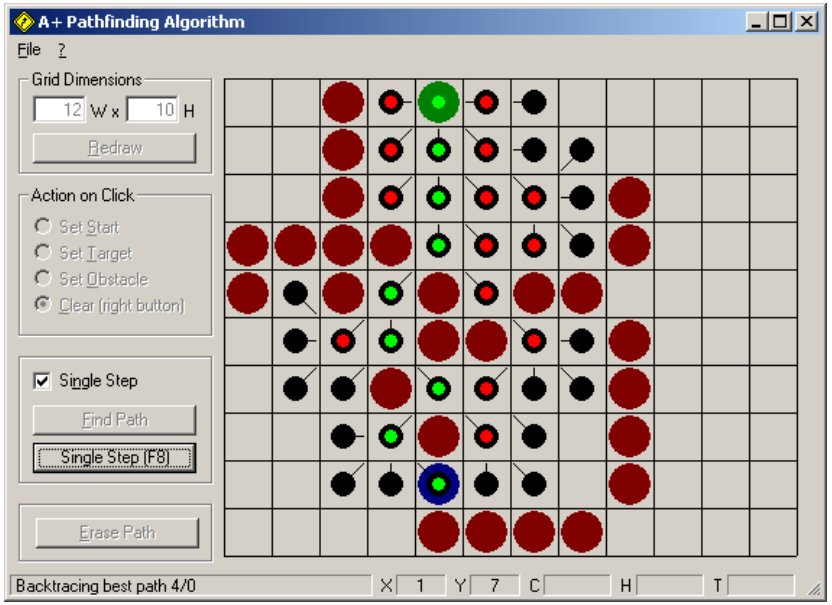
1. Definice počátečního uzlu prohledávání jako aktuální uzel grafu
2. Stanovení sousedních uzlů, jež jsou kandidáty na prohledávání z aktuálního uzlu (vytvoření ohodnocené fronty uzlů OPEN). Kritérium výběru (míry vhodnosti uzlu k expanzi) je hodnota heuristické funkce *f(x)* – nižší hodnota značí vyšší prioritu pro postup algoritmu do daného uzlu.
3. Je zvolen uzel s nejnižší hodnotou *f(x)* z OPEN, jako kandidát na odstranění.
4. Uzly sousední k uzlu k odstranění jsou přidány do fronty OPEN a přepočítány jim příslušné hodnoty *f(x)* a *h(x).* V předchozím označený uzel je odstraněn.
5. Pokračování do bodu 2 dokud cílový uzel nemá menší hodnotu *f(x)* než jakýkoliv jiný uzel z fronty OPEN, popř. OPEN je prázdná. (cílový uzel může být zpracován i vícekrát, pokud se vyskytují uzly s nižší hodnotou *h(x),* které by mohly vést k levnější cestě)
6. *Hodnota f(x) pro cílový uzel značí cenu výsledné cesty (h(x)=0 pro cílový uzel a přípustnou heuristiku)*
7. Rekonstrukce posloupnosti uzlů nalezené cesty se provede na základě předchozího zápisu bezprostředních předchůdců k jednotlivým uzlům při chodu algoritmu a následným zpětným postupem od cílového uzlu.

```
function A*(start,goal)
  closedset := the empty set                 // The set of nodes already evaluated.
  openset := set containing the initial node // The set of tentative nodes to be evaluated.
  came_from := the empty map                 // The map of navigated nodes.
  g_score[start] := 0                        // Distance from start along optimal path.
  h_score[start] := heuristic_estimate_of_distance(start, goal)
  f_score[start] := h_score[start]           // Estimated total distance from start to goal through y.
  while openset is not empty
     x := the node in openset having the lowest f_score[] value
     if x = goal
         return reconstruct_path(came_from, came_from[goal])
     remove x from openset
     add x to closedset
     foreach y in neighbor_nodes(x)
         if y in closedset
            continue
         tentative_g_score := g_score[x] + dist_between(x,y)

       if y not in openset
             add y to openset
             tentative_is_better := true
       elseif tentative_g_score < g_score[y]
             tentative_is_better := true
               else
             tentative_is_better := false
       if tentative_is_better = true
             came_from[y] := x
             g_score[y] := tentative_g_score
             h_score[y] := heuristic_estimate_of_distance(y, goal)
             f_score[y] := g_score[y] + h_score[y]
    return failure
```

Rekonstrukce posloupnosti uzlů (cesty):

```
function reconstruct_path(came_from, current_node)
    if came_from[current_node] is set
        p = reconstruct_path(came_from, came_from[current_node])
        return (p + current_node)
    else
        return current_node
```



Poznámky:

- V případě nevyužití sezamu CLOSED, redukuje se mechanismus na prohledávíní stromové struktury za podmínek, že:
  - Řešení jistě existuje
  - Jestliže je algoritmus upraven tak, že nové uzly jsou do OPEN přidávány pouze, mají-li nižší hodnotu *f(x)* než v jakékoliv jiné předcházející iteraci

A* složitost

- Výpočetní složitost A* závisí na použité heuristice
  - Nejhorší případem je exponenciální expanze uzlů podél řešení (hledání nejkratší cesty)
  - Složitost je polynomiální v případě, že:
    - prostor hledání má stromovou strukturu a
    - hledání probíhá s jediným cílovým uzlem a
    - užitá heuristická fce splňuje podmínku:

$$h(x) - h^*(x) = O(\log h^*(x))$$

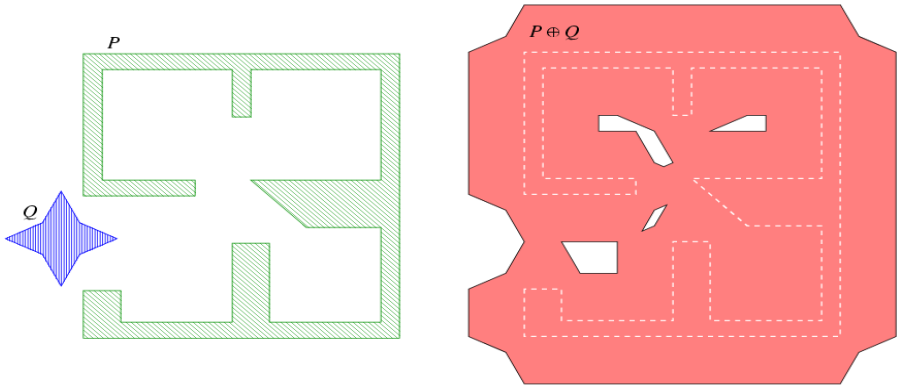
kde $h^*(x)$ značí optimální heuristiku - přesná cena cesty z uzlu x do cíle (t.j. chyba skutečně užité heuristiky $h(x)$ neroste rychleji než logaritmus optimální heuristiky $h^*(x)$

Slučování objektů, Minkowského suma, dilatace

- Operace sloužící ke geometrickéhu slučování objektů
- Minkowského suma je binární operací (binární dilatace P pomocí Q)
- V Euklidovkém prostoru označuje sdružení/množinové sečtení každého prvku (bodu) jedné množiny (objektu) s prvky druhé množiny: $P \oplus Q = \{p + q : p \in P, q \in Q\}$

Minkowského suma - použití

- Aplikace v oblasti plánování pohybu robotu mezi překážkami – umožňuje určit robotu dostupný prostor v okolí překážek (obecně při výpočtu konfiguračního prostoru robotu ~ množiny přípustných pozic objektu vzhledem k omezením implikovaným úlohou - robotem a překážkami)



Příklad výpočtu dilatace hranice prostředí pohyblivým objektem. Výsledek určuje hranice maximálního možného přiblížení objektu ke hranici prostředí.

- Základní nástroj k realizaci tzv. matematické morfologie.

Minkowského suma – 2D

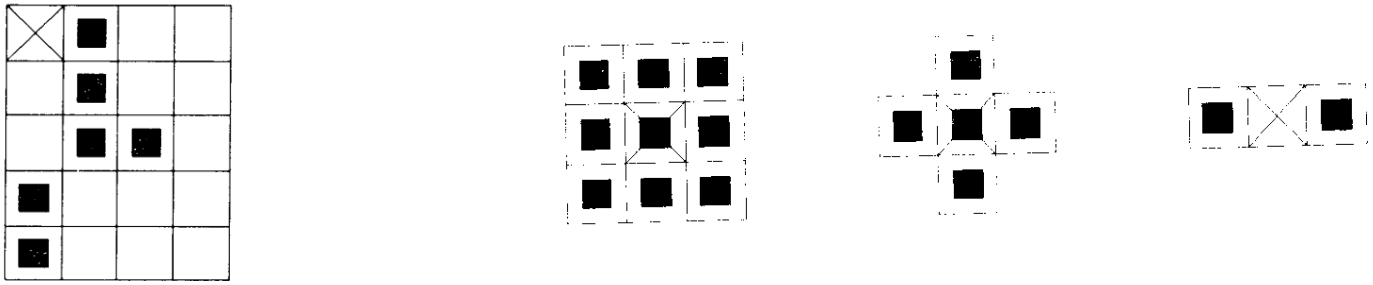
- Mějme konvexní polygony *P* a *Q* mající *m* a *n* vrcholů.
- Jejich Minkowského suma je konvexní polygon s nejvýše *m+n* vrcholy, který lze určit výpočtem v čase *O(m+n)*:
  - Předpokládejme, že hrany jednotlivých polygonů jsou orientované v sekvencích S_p a S_q (uspořádané v jednom zvoleném/polárním směru podle úhlu, t.j. Každý počátek jednotlivé hrany má přiřazený jedinečnou úhlovou souřadnici)
  - V důsledku předchozího existuje uspořádaná posloupnost *S,* jenž je tvořena sjednocením elementů S_p a S_q a jejíž konstrukce je přímočará (nové uspořádání množiny $S_p \bigcup S_q$ (Sjednocení obou posloupností zachováva původní směry všech hran zpracovávaných polygonů)
  - V předchozím uspořádaná posloupnoust tvoří Minkowského součet *P* a *Q.*

Složitost výpočtu Minkowského sumy:

- Dvojice konvexních objektů *O(m+n)*
- Konvexní a nekonvexní objekt *O(m.n)*
- Dvojice nekonvexních objektů $O((mn)^2)$

Minkowského suma - morfologie

- Speciálním případem je realizace Minkowského sumy jako dilatace vstupního objektu ve formě bodové množiny vhodně zvolenou (typicky menší) bodovou množinou, tzv. *strukturním elementem*

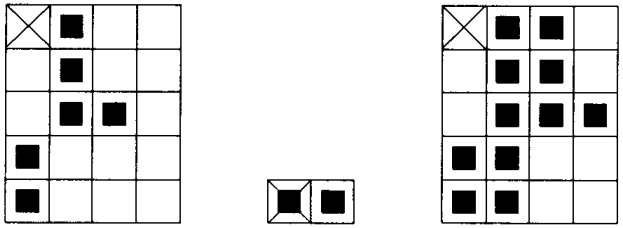


Příklad reprezentace bodové množiny v rastru (vlevo) a nejčastěji používané izotropické *strukturní elementy* (vpravo)

Princip výpočtu (dilatace strukturním elementem)

- Strukturní element se při výpočtu systematicky (2-jice vnořených cyklů) pohybuje v rastrové reprezentaci vstupní množiny; jeho střed/počátek souřadnic definuje tzv. aktuální bod.
- Výsledek relace Minkovského součtu $\oplus$ mezi vstupní množinou a strukturním elementem je zapisován do aktuální pozice (binární hodnoty 0,1)

Příklad (dilatace strukturním elementem 1x2 a vliv dilatace na hranici objektu):



Vstupní objekt (vlevo), použitý strukturní element a výsledný dilatovaný objekt (vpravo)

Situaci lze popsat jako: $P \oplus Q = \{(0,1),(1,1),(2,1),(2,2),(3,0),(4,0),(0,2),(1,2),(2,2),(2,3),(3,1),(4,1)\}$

kde: $P = \{(0,1),(1,1),(2,1),(2,2),(3,0),(4,0)\}$

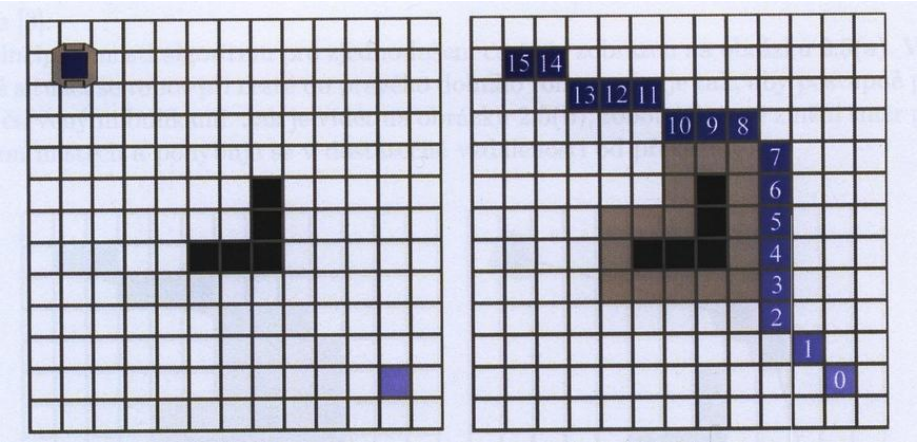
$Q = \{(0,0),(0,1)\}$ (strukturní element)

Chování na hranici objektů (a v dírách) - ve hranici se zaplní „propady“ do velikosti použitého strukturního elementu a dle jeho „směru“/tvaru.

Vstupní objekt (vlevo), objekt po dilataci (vpravo) bez otvorů a propadů

Příklad (aplikace dilatace v robotice k určení přístupné oblasti v okolí překážky):

- Metoda zajišťuje/zjišťuje prostupnost daného místa v prostředí pro objekt, jako výsledek dilatace vstupních dat strukturním elementem, korespondujícím s rozměry a tvarem objektu (robotu s nenulovými rozměry)



Rastrová mapa prostředí (vlevo) s překážkou (černá), robotem (tm.modrá) a cílovou pozicí (fialová)

Mapa s dilatovanou překážkou (šedá), robot při realizaci naplánované trajektorie (modrá-fialová) nekoliduje s překážkou

Příprava realizace cesty – proč je třeba činit?

- Plánování trajektorie probíhá obvykle na diskrétní struktuře (modelu prostředí robotu)
  - Grafová reprezentace prostředí (geometricky „řidké“ a diskrétní)
  - Mřížkový (rastrový) model prostředí (může být geometricky „hustou“ reprezentací, nicméně se zvyšujícím se rozlišením mřížky roste paměťová náročnost takového modelu)

- → proto je často voleno rozlišení modelu (mřížkový model) jako kompromis mezi paměťovou náročností a přesností modelu (nese geometrickou infomaci)
- Topologická infomace je nesena uspořádáním modelu (mřížky), resp. grafem sousednosti jednotlivých elementů (pixelů); *uzel* ~jednotlivý element mřížky, *hrana* ~ relace sousednosti s okolními elementy mřížky
- Plán činnosti robotu (cesta) je generován prohledávacím algoritmem (viz. výše A*, Dijkstra) jako diskrétní posloupnost uzlů s přiřazenou geometrickou infomací (polohami) a které je třeba projít

Vyhlazování cesty – související úlohy k řešení:

- Skutečně realizovaná trajektorie musí být spojitá (tzv. hard-real time kdy výpočet řízení robotu končí dříve než jsou k dispozici další data (pixel) – regulace rychlosti a polohy)

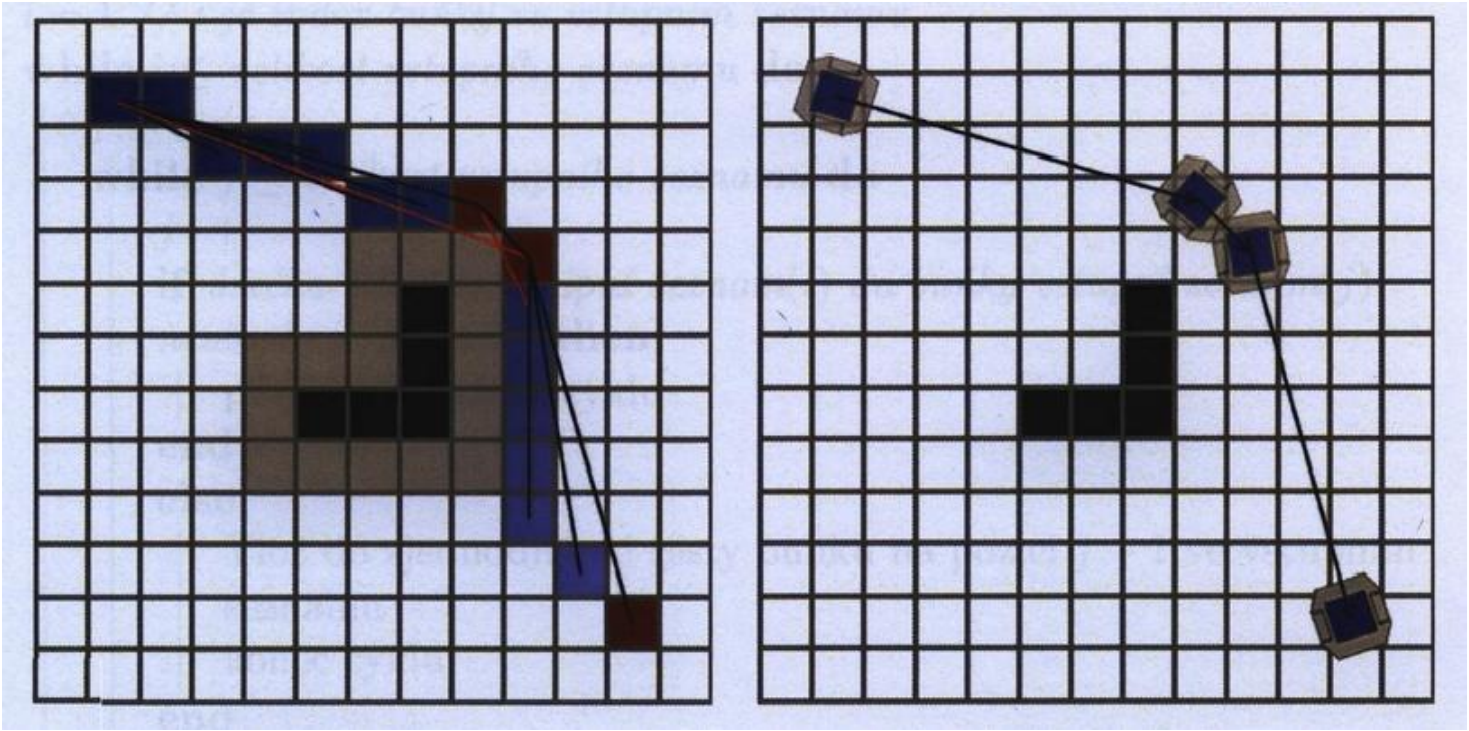
- Řešeni spočívá ve:
  1. Vyhlazování naplánované trajektorie do formy s minimální křivostí (odstranění zbytečných zatáček, optimalizuje spotřebu energie a rychlost vykonávání trajektorie)
  2. Optimálním umístění vyhlazené trajektorie na diskrétní rastr modelu prostředí (interpolace trajektorie vhodnými křivkami (spline), popř. lineárními úseky (zde) a
  3. Realizacï spojitého řízení přechodu mezi sousedními elementy modelu prostředí

Vyhlazování cesty – příklad:

Naplánovaná diskrétní trajektorie a její vyhlazování aproximací lineárními

Úseky (vlevo)

Výsledný, po úsecích lineární, pohyb robotu po vyhlazené trajektorii (vpravo)

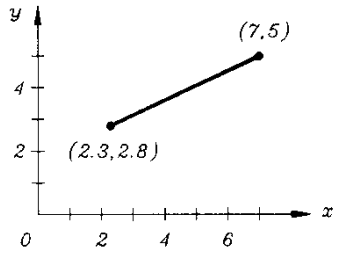


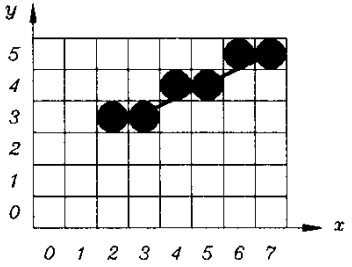
Algoritmus 2: Vyhlazení cesty

```
   Data: vstupní seznam buněk
   Result: zjednodušená cesta (posloupnost buněk)
 1 begin
 2     zjednodušená cesta := prázdný seznam buněk
 3     i = 1 // i je index buňky ve vstupním seznamu
 4     while i ≤ velikost vstupního seznamu do
 5         j = i
 6         while j ≤ velikost vstupního seznamu do
 7             j++
 8             if úsečka z buňky vstupní seznam(i) do buňky vstupní seznam(j)
               neobsahuje překážku then
 9                 přejdi na začátek cyklu
10             end
11             else
12                 Vlož do zjednodušené cesty buňku na pozici j − 1 ve vstupním
                   seznamu
13                 konec cyklu
14             end
15             if j = velikost vstupního seznamu then
16                 i = j
17                 vlož buňku vstupní seznam(j) do zjednodušené cesty break
18             end
19         end
20         i++
21     end
22 end
```

Umístění trajektorie v diskrétním rastru – aproximace lineárními úseky:

- Výpočet pozice bodů přímky/úsečky v obecné poloze (ve 2D) může reprezetovat potřebu operací (“*” a “/”) s reálnými čísly – výpočetně náročné.





- Realizace užitím Bresenhamova algoritmu (popř. DDA - Digital Differential Analyzeru) pracujícího výhradně v celočíselné aritmetice
  - Přímka (úsečka) $y = kx + q$ je zadána svým počátečním a koncovým bodem (x_1, y_1) a (x_2, y_2), kde $k = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta x}{\Delta y}; q = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}$
  - Využití principu přičítání relativního (racionálního) přírustku $\Delta y = k * \Delta x$ k souřadnicím (x, y) k výpočtu pozice bodů konstruhované úsečky

```
{LINEBRES.INC – POČÍTAČOVÁ GRAFIKA: kresba úsečky Bresenhamo-
vou metodou pro 1. a 5. kvadrant}
procedure Usecka_Bresenham (x1, y1, x2, y2: integer);
  var    deltax, deltay, konst1, konst2: integer;
         x, y, x_koncove, predikce: integer;
  begin
    if x1 > x2  {určení počátečního a koncového bodu}
      then begin
             x := x2;  y := y2;  x_koncove := x1;
           end
      else begin
             x := x1;  y := y1;  x_koncove := x2;
           end;
    deltax := x_koncove – x;
    deltay := abs(y1 – y2);
    konst1 := 2 * deltay;
    konst2 := 2 * (deltay – deltax);
    predikce := 2 * deltay – deltax;
    Put_pixel (x, y);
    while x < x_koncove do
      begin
        x:= x + 1;
        if predikce < 0                    { souřadnice y zůstává      }
          then predikce := predikce + konst1
          else                             { souřadnice y se zvyšuje }
            begin
              y := y + 1;
              predikce:= predikce + konst2
            end;
        Put_pixel(x, y)
      end {while x < x_koncové}
  end; {Úsečka_Bresenham}
```

Pozn.:

- Uvedený algoritmus postihuje případy se směrnicí $0 < k < 1$
- Ostatní případy pro $k > 1$ se řeší záměnou souřadnic x a y
- Obdobně, záporné směrnice k vyžadují změnu smyslu přírustků (znaménka)

```
{LINEDDA.INC – POČÍTAČOVÁ GRAFIKA: kresba úsečky metodou DDA}

procedure Usecka_DDA (x1, y1, x2, y2: integer);
   var   deltax, deltay, kolik_bodu, k: integer;
         x_prirustek, y_prirustek, x, y: real;
   begin
      deltax := x2 – x1;
      deltay := y2 – y1;
      if abs(deltax) > abs(deltay)
         then kolik_bodu := abs(deltax)
         else kolik_bodu := abs(deltay);
      x_prirustek := deltax / kolik_bodu;
      y_prirustek := deltay / kolik_bodu;
      x := x1; y := y1;
      Put_pixel (round(x), round(y));
      for k := 1 to kolik_bodu do
         begin
            x := x + x_prirustek;
            y := y + y_prirustek;
            Put_pixel (round(x), round(y))
         end {for k}
   end; {Úsečka_DDA}
```

Alternativní algoritmus DDA (Digital Differential Analyzer)

- **Přičítání konstantních přírustků k oběma souřadnicím**

Reference:

- Václav Hlaváč, Milan Šonka: Počítačové vidění, Grada a.s., Praha, 1992, 272 s., ISBN 80-85424-67-3
- Jiří Žára a kolektiv: Počítačová grafika, Grada a.s., Praha, 1992, 472 s., ISBN 80-85623-00-5
- Josef Štrunc: Multirobotická explorace prostředí, Bakalářská práce, Katedra kybernetiky, FEL ČVUT Praha, 2009, 31 s.
- Wikipedia...