A* algoritmus

Vypracovala: **Nikola Jirešová**

OBSAH

Historie

V roce 1964 **Nils Nilsson** zvýšil rychlost Dijkstrova algoritmu pomocí heuristiky. Tento algoritmus nazval A1

- Dijkstrův algoritmus - nalezení nejkratší cesty. Prochízí všechny možné varianty, tudíž je pomalý

V roce 1967 **Bertram Raphael** vylepšil A1 na A2, ale ten se nepsosadil.

V roce 1968 **Peter E. Hart** doladil algoritmus A2 za pomocí heuristiky a malých změn a vznikl A* (A star) algoritmus.

A* algoritmus

A* je algoritmus je search algoritmus - má za úkol nalezení nejkratší cesty v grafu/poli. Je kombinací dvou metod prohledávání grafu – BFS a DFS a heuristické funkce.

Byl vynalezen za účelu zefektivnění hledání cesty pro robota Shakeyho v prostoru s přemístitelnými překážkami, na kterém pracovali vědci ze Standfordské univerzity.

BFS (Breadth-First Search)

Breadth-First Search je metoda prohledávání grafu přeloženo jako “Prohledávání do šířky”

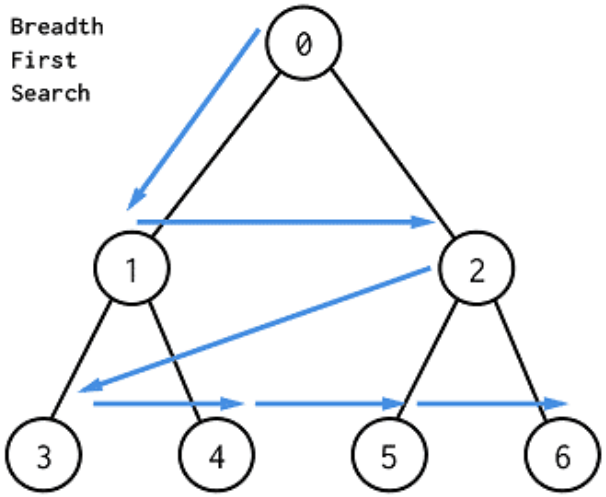
Pomocí BFS je graf prohledáván po po vrstvách. Nejdříve “projede” a zapíše uzly na stejné úrovni (všechny sousedy zdrojového uzlu), a teprve poté se pohybujeme do následující úrovně.

K prohledání používá tak zvanou queue - frontu, kam postupně zapíše za sebe nalezené uzly a postupně je odebírá ze začátku a nové zapisuje na konec.

Příklad BFS

Představme si, že si čteme článek na Wikipedii. Jakmile narazíme na slovo odkazující na další stránku, zapíšeme si ho a pokračujeme ve čtení. Narazíme na další odkaz a zapíšeme ho za první již zapsaný odkaz. Tímto způsobem projedeme celou stránku.

Po dočtení si rozklikneme první odkaz na našem seznamu (frontě) a celý proces zopakujeme. Jakmile se dostaneme na konec, rozklikneme si druhý odkaz z prvního článku, který je teď ve frontě první. Tento proces opakujeme než prohledáme všechny odkazy.



DFS (Depth-First Search)

Depth-First Search je metoda prohledávání grafu přeloženo jako “Prohledávání do hloubky”

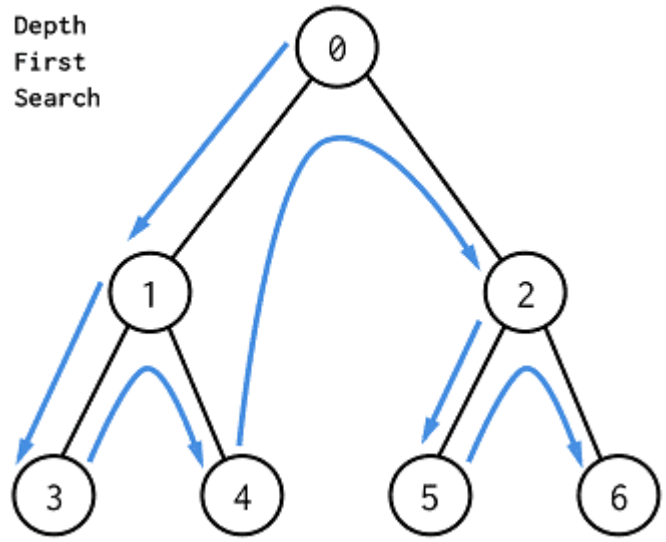
DFS se postupně ponořuje do nejhlubších částí grafu, dokud nenarazí na konec cesty, a poté se vrací a prozkoumává další možnosti.

K prohledávání používá tak zvaný Stack - zásobník, kam přidává nalezené uzly na začátek a postupně je po naražení na konec odebírá neboli přejde až na konec první cesty a postupně se vrací a doprohledává další uzle.

Příklad DFS

Opět si ředstavme Wikipedia článek jako u BFS. Tentokrát ale rozklikneme odkaz hned, jakmile na něj narazíme.

Čteme si článek například o zvířatech. Uvidíme zde odkaz na podrobnější vysvětlení savců. Začneme tedy číst o savcích, kde nalezneme odkaz pro kočky. Ve článku o kočkách už není žádný odkaz, vrátíme se tedy ke článku o savcích. Rozklikneme si psy, kde také není žádný odkaz a znovu se vrátíme. Takto postupujeme, dokud neprohledáme všechny odkazy – uzle.



Heusitika

Původ slova je z řečtiny - nalézt, objevit

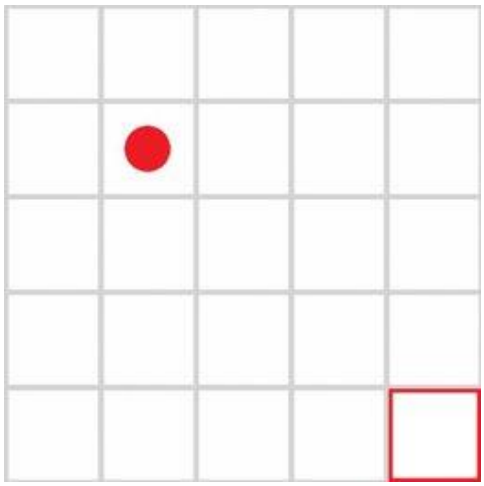
Heuristika je obecně teorie řešení problémů, který slouží k odhadu řešení problému, aniž by byl využit algoritmický přístup. Spoléhá se na intuici a zkušenosti.

V A* algoritmu je funkce, která poskytuje odhadovanou hodnotu nákladů od aktuálního uzlu k cílovému uzlu. Slouží k urychlení celého procesu nasměrováním algoritmu k cíli. Díky heuristické funkci v rovnici A* nemusíme prohledávat všechny možné varianty.

Využíva **Manhattanskou vzdálenost,** která sčítá absolutní hodnoty rozdílů souřadnic x a y.

Příklad Manhattanské vzdálenosti

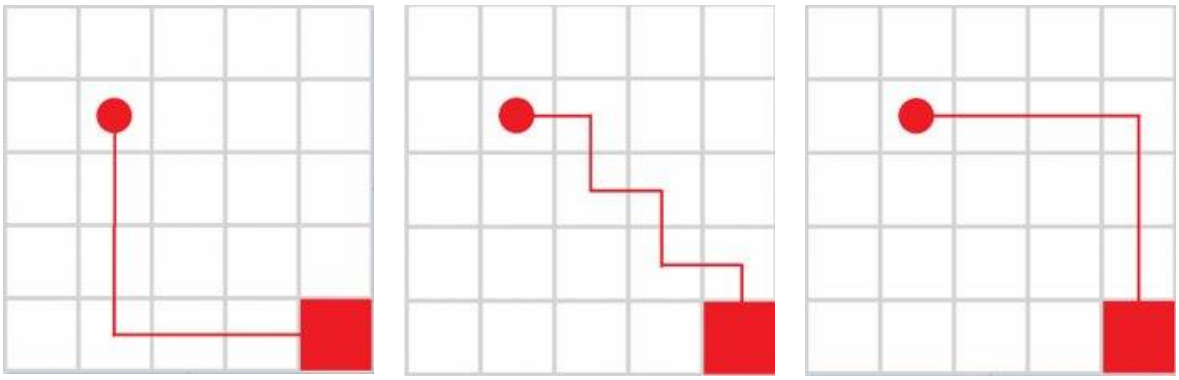
V 5x5 poly máme označený start [2, 2] a cíl [5, 5].



Manhattanskou vzdálenost spočítáme za pomocí rovnice: $(x_s - x_c) + (y_s - y_c)$

$(5 - 2) + (5 - 2) = 3 + 3 = \underline{6}$

Manhattanská vzdálenost je tedy 6. Výsledná nejkratší cesta by mohla vypadat například takto:



Proč se používá heuristika?

- Zrychlení vyhledávání
- Omezení počtu prohledaných uzlů
- Nalezení optimálního řešení

Jak A* funguje?

Řídí se hlavní rovnicí:

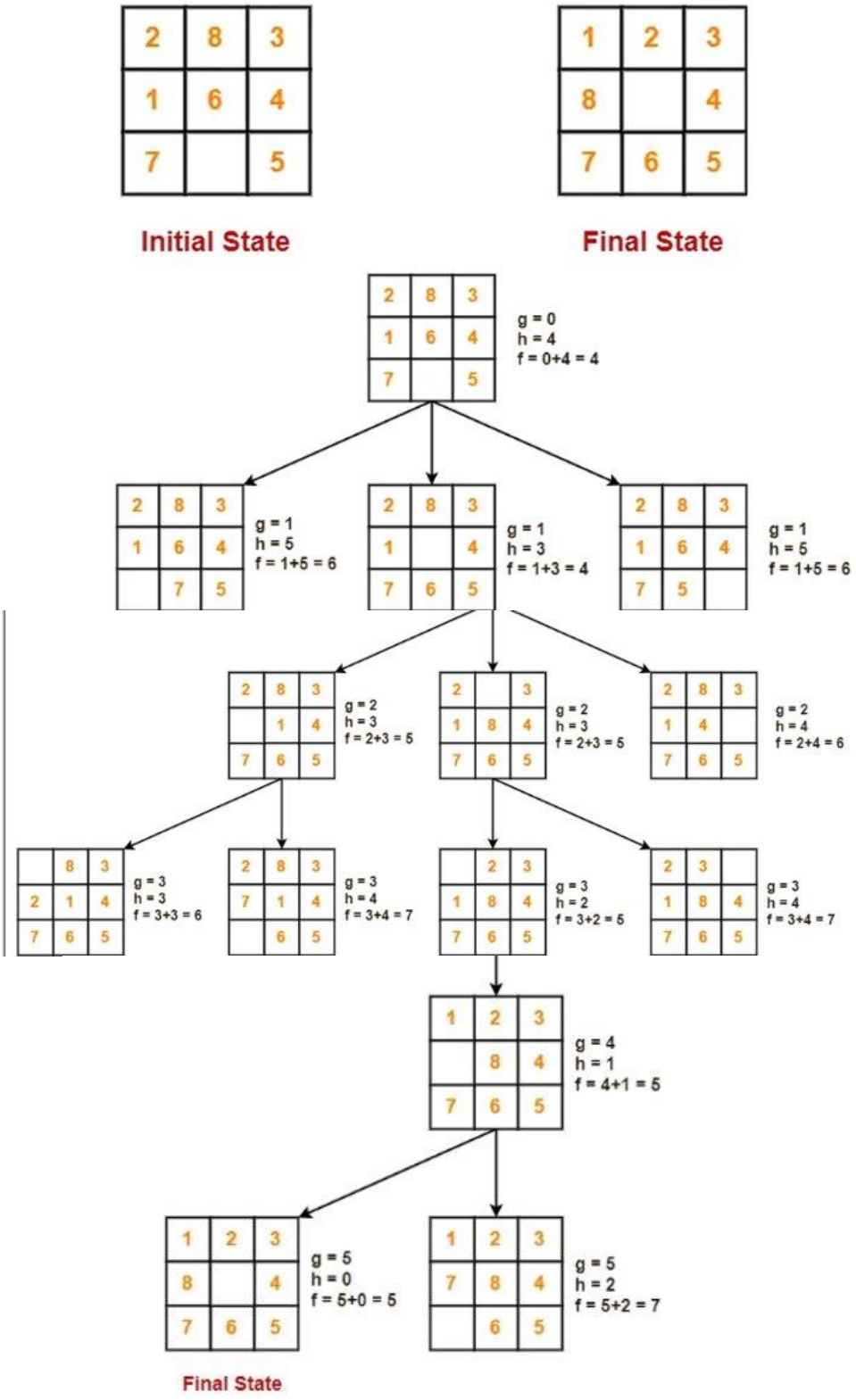
$f(x) = g(x) + h(x)$

- g(x) je celková vzdálenost z počáteční pozice do aktuální pozice.
- h(x) je heuristická funkce

Když A* prochází grafem, sleduje cestu s nejnižšími známými náklady. Pokud se v libovolném bodě cesty objeví segment cesty, který má vyšší náklady než jiný segment cesty, na který narazil, opustí segment s vyššími náklady a místo toho projde segment cesty s nižšími náklady. Tento proces pokračuje, dokud je dosaženo cíle.

Příklad

Nalezení nejvýhodnější cesty k dosažení konečného stavu z počátečního stavu pomocí algoritmu A*.



Pseudokód

1. Zapíšeme počáteční stav do seznamu OPEN, seznam CLOSED je prázdný. Jestli je počáteční stav stavem cílovým – konec prohledávání.

2. Jestli je OPEN prázdný – řešení neexistuje – konec prohledávání.

3. Vybereme stavy s nejmenší hodnotou f(*x*) ze seznamu OPEN.

4. Pokud některý z nich je cílovým stavem, zjisti cestu od počátečního stavu do cílového stavu a ukončíme prohledávání.

5. Jinak vybereme první stav ze seznamu stavů z předchozího bodu, které mají nejmenší hodnotu f(*x*). Vymažeme stav ze seznamu OPEN, zařadíme ho do seznamu CLOSED.

6. Expandujeme vybraný stav *x* a pro všechny jeho následníky *y* vyhodnocujeme hodnotu f(*x*). Do seznamu OPEN zapíšeme ty, kteří nejsou v seznamu OPEN ani v CLOSED. Jestliže některý z následníků *y* je už v seznamu OPEN (stav *k*), ale s větší hodnotou hodnotící funkce (f(*k*) > f(*y*)), tak odebereme existující stav *k* a přidáme následníka *y* do seznamu OPEN. Jinak pokračujeme krokem 3.

Výhody a nevýhody

Výhody

- Najde optimální řešení
- Heuristika ,,nasměruje" algoritmus k cíli - neprohledává tolik uzlů
- Různorodost využití
- Rozhodování v reálném čase

Nevýhody

- Náročnost na paměť (při větších grafech)
- Úspěšnost algoritmu závisí na způsobu implementace a struktury dat
- Neprovádí cestu zpět

Využití

- Navigace
  - GPS
  - Autonomní vozidla
- AI a hry
  - Hledání cest pro NPC
  - Tahové strategické hry
- Robotika
  - Navigace robotů (např.: robot Shakey díky kterému byl algoritmus vytvořen)
- Medicína
  - Trasování neurologických spojení
- Další
  - Trasovánání a doručování zásillek