Wstęp do sztucznej inteligencji

Wojciech Typer

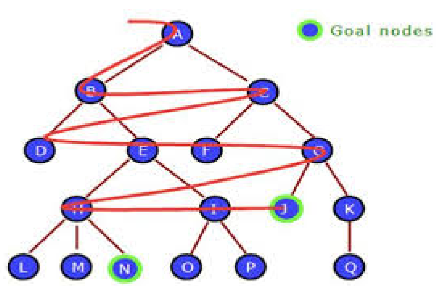
Przestrzeń stanów

Przestrzeń stanów to uporządkowana czwórka (V, E, S, F), gdzie:

- V → zbiór wierzchołków reprezentujących stany powstałe w trakcie rozwiązywania problemów
- E → zbiór krawędzi reprezentujących możliwe przejścia między stanami
- S → niepusty podzbiór V, zawierający stany początkowe problemu
- F → niepusty podzbiór V, zawierający stany docelowe problemu (mogą być zdefiniowane wprost lub przez własności które chcemy osiągnąć)
- Rozwiązaniem będziemy nazywać ścieżkę w grafie od stanu początkowego do stanu docelowego

Podstawowe problemy przy przeszukiwaniu przestrzeni stanów

- Czy algorytm gwarantuje znalezienie rozwiązania?
- Czy algorytm zawsze się kończy?
- Czy algorytm znajduje optymalne rozwiązanie?
- Jaka jest złożoność czasowa i pamięciowa algorytmu?
- Jak można poprawić złożoność czasową i pamięciową?

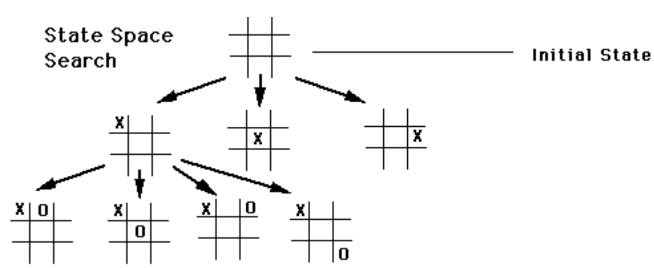


Sformułowanie zadania dla algorytmów przeszukiwania

- Precyzyjna definicja przestrzeni stanów
- określenie stanu początkowego
- określenie reguł przejścia między stanami (operatory akcji lub funkcja następnika)
- zbiór stanów docelowych lub funkcja weryfikacji osiągnięcia celu
- funkcja kosztu ścieżki

Kierunki przeszukiwania

- w przód (od stanu początkowego do celu)
- w tył (od stanu celu do początku)
- przeszukiwanie dwukierunkowe



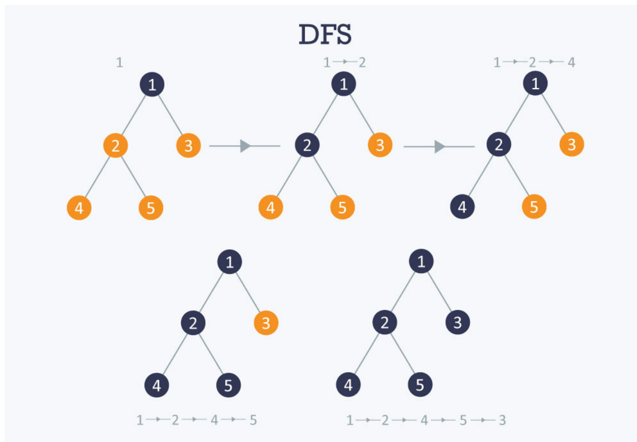
Strategie przeszukiwania

- Przeszukiwanie w głąb
  - zaczynamy w wierzchołu początkowym
  - dla aktualnego wierzchołka v:
    - oznacz v jako zbadany
    - jeśli v jest celem zakończ procedurę
    - jeśli niezbadany jest wierzchołek do którego można przejść to przejdź do niego

  * jeśli nie ma już niezbadanych sąsiadów to wróć do wierzchołka z którego przyszedłeś

  Zalety: łatwość implementacji, małe wymagania pamięciowe

  Wady: znalezione rozwiązanie nie musi być optymalne



- Przeszukiwanie w głąb ze stosem
  - zaczynamy w wierzchołku początkowym
  - dla aktualnego wierzchołka v:
    * oznacz v jako zbadany
    * jeśli v jest celem zakończ procedurę
    * dla każdego sąsiada v, który jest niezbadany, dodaj go na stos i oznacz jako odwiedzony
    * jako następny, weź wierzchołek ze szczytu stosu

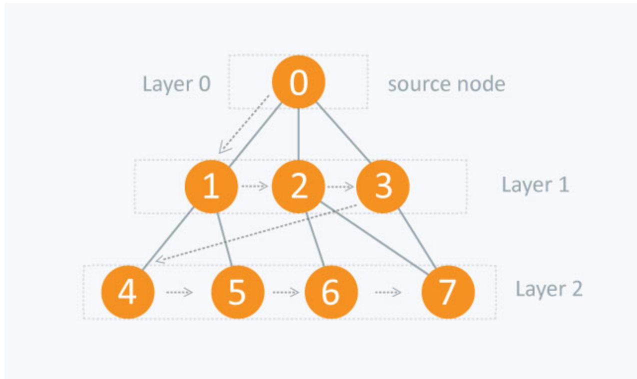
  Zalety: łatwość implementacji, łatwe struktury pamięciowe

  Wady: znalezione rozwiązanie nie musi być optymalne

- Przeszukiwanie wszerz
  - zaczynamy w wierzchołku początkowym
  - dla aktualnego wierzchołka v:
    * oznacz v jako zbadany
    * jeśli v jest celem zakończ procedurę
    * dla każdego sąsiada v, który jest niezbadany, dodaj go do kolejki i oznacz jako odwiedzony
    * jako następny, weź wierzchołek z początku kolejki

  Zalety: znalezione rozwiązanie jest optymalne

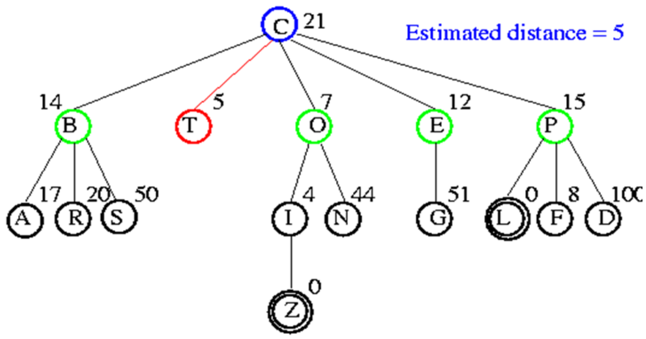
  Wady: duże wymagania pamięciowe



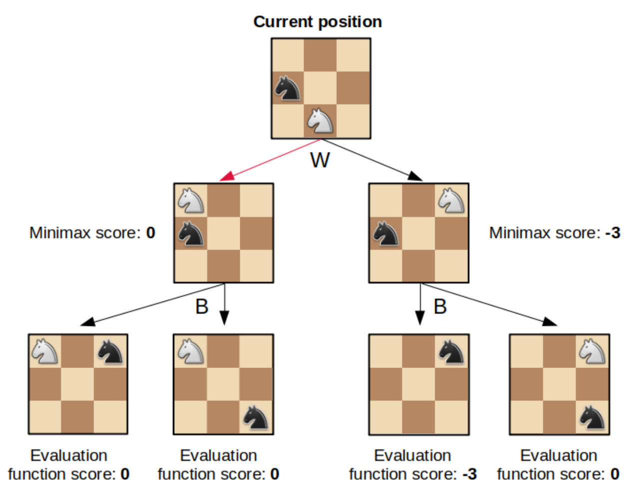
- Best-First-Search
  - Rozszerzenie przeszukiwania wszerz po dodaniu funkcji kosztu, która eksploruje graf poprzez rozwinięcie najbardziej obiecującego węzła wybranego zgodnie z określoną regułą.
    - zaczymay w wierzchołku początkowym, którego koszt ustwiamy na 0
    - dla akutalnego wierzchołka v:
      - oznacz v jako zbadany
      - jeśli v jest celem zakończ procedurę
      - dla każdego sąsiada v który jest odwiedzony, jeśli koszt v plus koszt przejścia są mniejsze niż jego dotychczasowy koszt to zmodyfikuj go w kolejce priorytetowej nadając nowy mniejszy koszt
      - dla każdego sąsiada v, który jest niezbadany, dodaj go do kolejki priorytetowej z kosztem v plus koszt przejścia oraz oznacz jako odwiedzony
      - jako następny, weź wierzchołek z kolejki priorytetowej o najmniejszym koszcie

  Zalety: znalezienie rozwiązania jest optymalne, ze względu na koszt ścieżki

  Wady: kolejka priorytetowa jest trudniejsza w implementacji i ma większą złożoność czasową

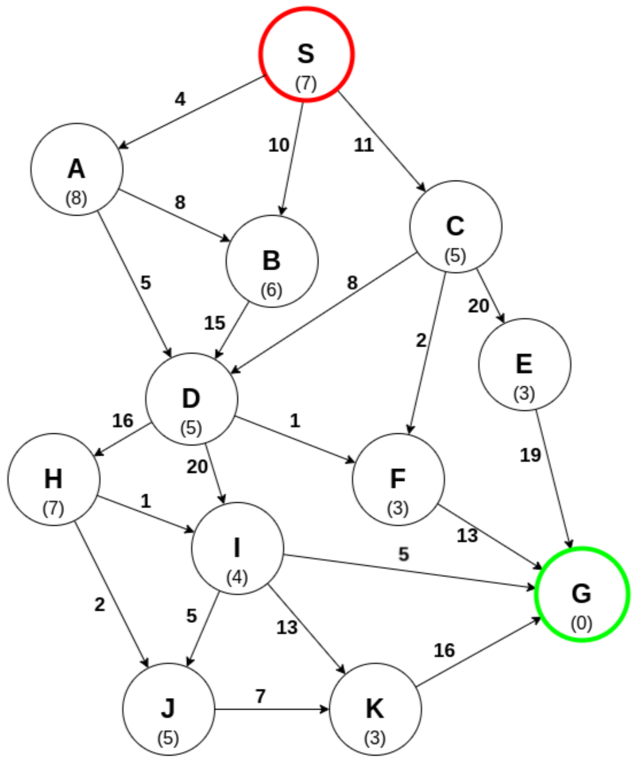


- Algorytm A
  - Funkcja oceny heurystycznej Funkcją oceny heurystycznej nazywamy funkcję kosztów określoną na stanach postaci:
    $f(v) = g(v) + h(v)$
    Gdzie g(v) jest aktualną odległością (kosztem) od stanu początkowego do stanu v
    a h(v) jest heurystycznym oszacowaniem odległości (kosztu) od stanu v do celu
  - Przykład → skoczki (chcemy uniknąć bicia)



Ponieważ bardziej szukamy rozwiązania niż liczby ruchów, możemy też zastosować algorytm zachłanny
(nie liczyć ruchów, czyli g(v) = 0)

– przykład → graf



Musimy przejść z punktu S do punktu G. Liczby w nawiasach to ocena heurystyki.
Po sprawdzeniu całej przestrzeni stanów, okazuje się, że najlepszę rozwiązanie to:
$S \to A \to D \to F \to G$

- Algorytm A*
  - Jeżeli algorytm A wykorzystuje funkcję oceny heurystycznej taką, że dla każdego v zachodzi h(v) ≤ h*(v), to otrzymujemy algorytm A*
  - **Twierdzenie:** Algorytm A* jest dopuszczalny
    D-d: Niedoszacowanie funkcji h powoduje, że zaniżamy rzeczywisty koszt ścieżki, czyli nie może zajść sytuacja, że ominiemy optymalny wierzchołek w drodze do celu. Gdyby oszacowanie było zawyżone, to zanim rozpatrzylibyśmy ten wierzchołek przez inny, moglibyśmy dojść do celu

- dla dwóch dopuszczalnych heurystyk $h_1 i h_2$, jeżeli dla dowolnego stanu v zachodzi: $h_1(v) \leq h_2(v)$, mówimy, że h_2 zawiera więcej informacji niż h_1 Jeżeli h_2 jest lepiej poinformowana niż h_1, to zbiór stanów odwiedzonych przez A* z heurystyką h_2 jest podzbiorem zbioru stanów odwiedzonych przez A* z heurystyką h_1
- więzy
  * Dla stanów definiujemy warunki dopuszczalności, które ograniczają nam ich liczbę
  * Musimy wówczas zmodyfikować operacje przejścia między stanami, aby przechodzić między dopuszczalnymi
  * Dzięki zmnijeszeniu liczby stanów, zmniejszamy zasoby potrzebne do znalezienia rozwiązania
  * Najczęsciej zapisujemy ograniczenia (więzy) jako formuły logiczne narzucone na na własności stanów