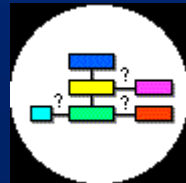


# Sztuczna Inteligencja

## 3.1 Metody szukania na ślepo



Włodzisław Duch

Katedra Informatyki Stosowanej UMK

Google: Włodzisław Duch

# Szukanie - metoda uniwersalna

Zakładamy, że problem jest zdefiniowany, tzn. jest:

1. Baza danych: fakty, stany, możliwości, opis sytuacji.
2. Możliwe operacje: zmieniają stan bazy danych.

Potrzebna jest strategia kontrolna poszukiwania rozwiązania.

Proces przeszukiwania wygodnie jest przedstawiać za pomocą drzew i grafów.

Droga na grafie od sytuacji startowej do rozwiązania

$\Leftrightarrow$  znalezieniu sekwencji operacji prowadzących do rozwiązania

$\Leftrightarrow$  rozumowaniu.

Ograniczenie: tylko problemy dyskretne, kombinatoryczne.

Ograniczone zastosowanie w przypadku percepcji sygnałów ciągłych, ale czasami możliwa jest dyskretyzacja.

# Procedury szukania

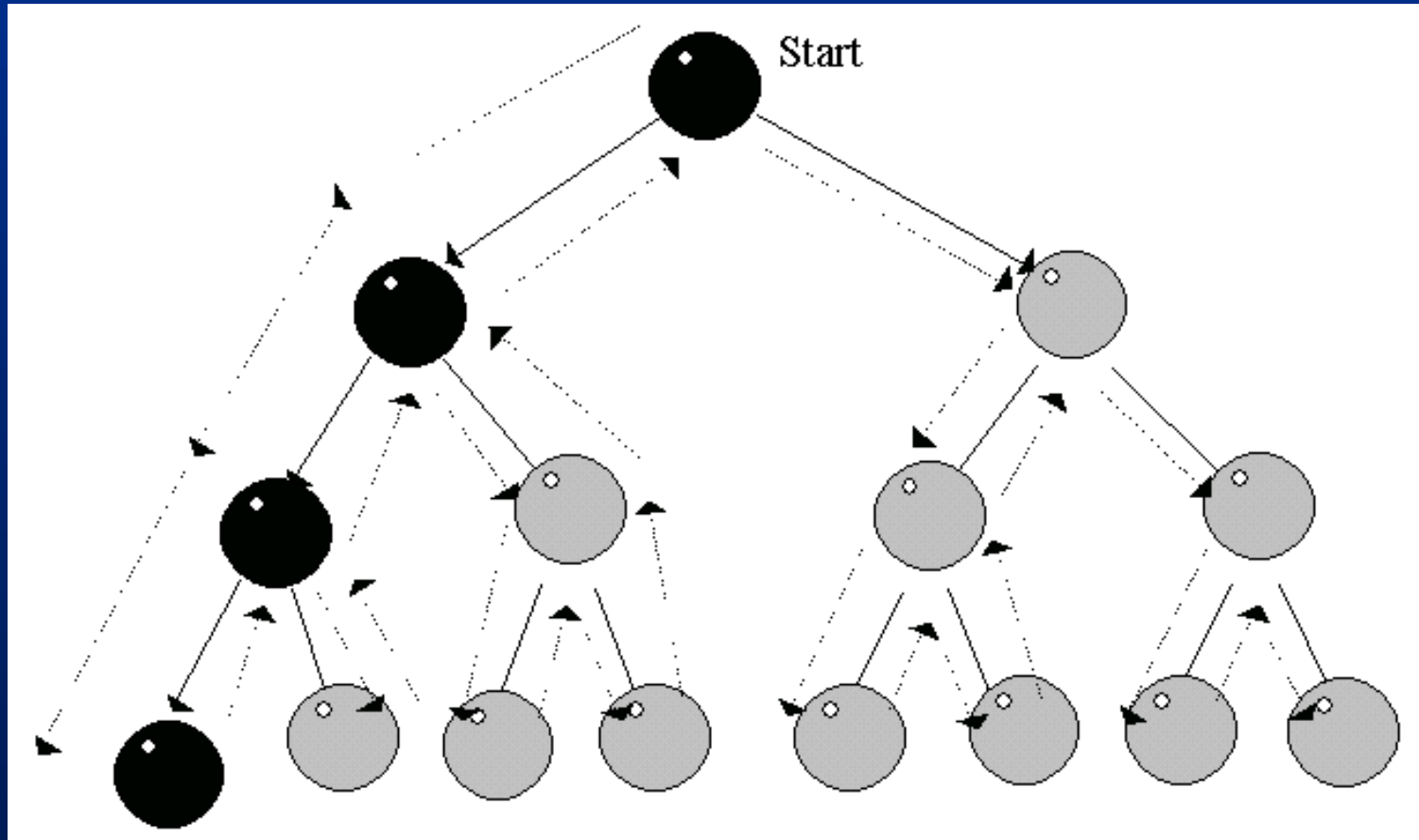
- Szukanie na ślepo - nie mamy żadnej informacji.
- Szukanie heurystyczne - potrafimy ocenić postępy.

Na ślepo:

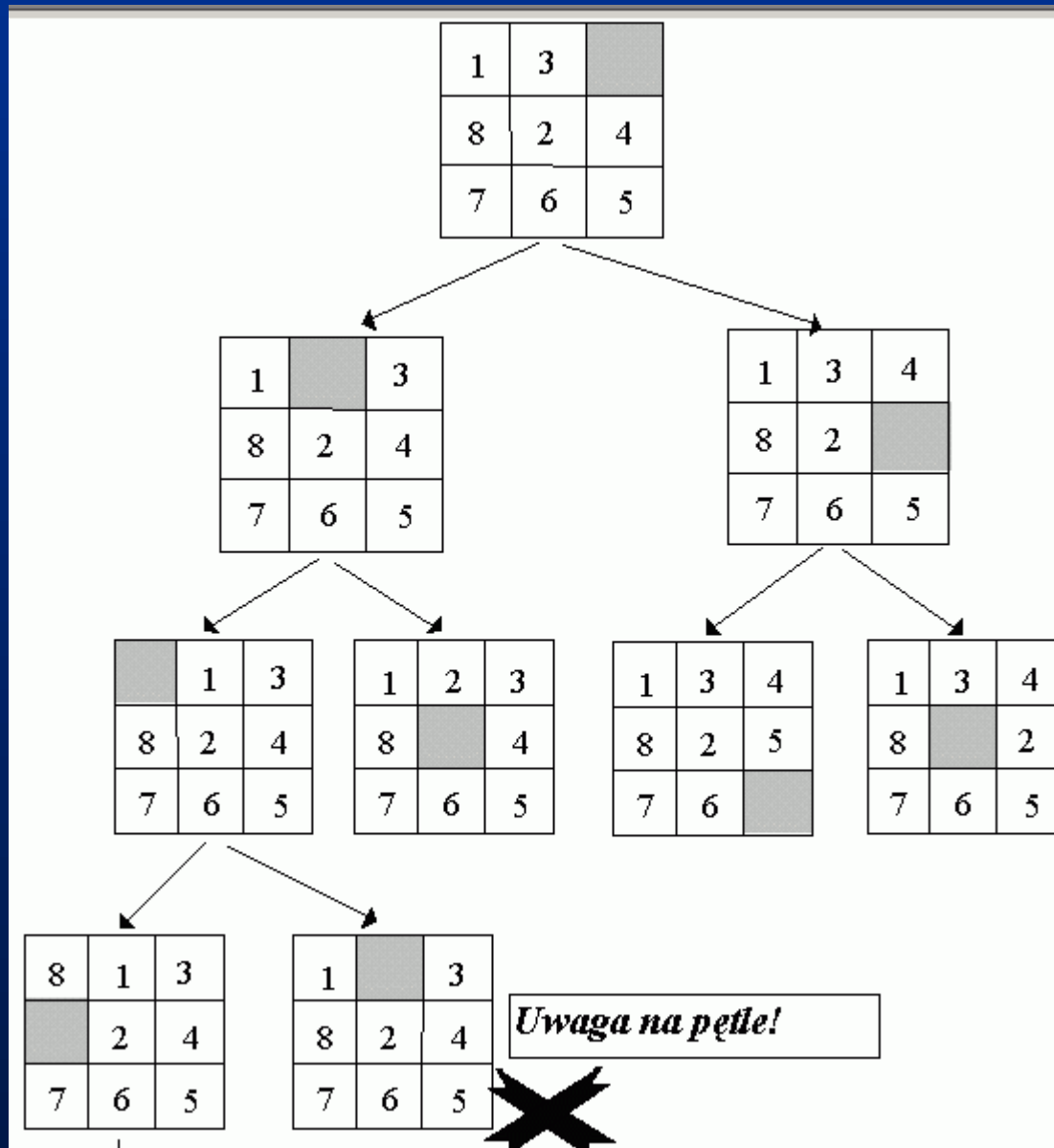
- Monte Carlo, czyli procedura Brytyjskiego Muzeum.  
Dla większych problemów jeśli istnieje wiele rozwiązań może coś znaleźć ... i ślepej kurze ...
- Szukanie w głąb.
- Szukanie w szerz, sprawdzając wszystkie możliwości.

Ilustracje: Program **PathFinding**, PathDemo

# W głąb, DFS



# Przykład algorytmu DFS dla przesuwanek



# Modyfikacja DFS

Zalety: wystarczy niewielka pamięć  $O(d)$

dla problemów, które mają wiele rozwiązań może być szybkie.

Wady: nieskończona głębokość, możliwe pętle w grafie,  
nie gwarantuje prostoty znalezionej rozwiązania.

Modyfikacja metody DFS:

ograniczaj głębokość szukania do poziomu  $d$ .

Złożoność: czas  $b^d$ , pamięć  $b^d$ .

Wada: jeśli  $d$  za małe nie znajdzie rozwiązania.

# Modyfikacja DFS - przykład

Zalety: wystarczy niewielka pamięć  $O(d)$

dla problemów, które mają wiele rozwiązań może być szybkie

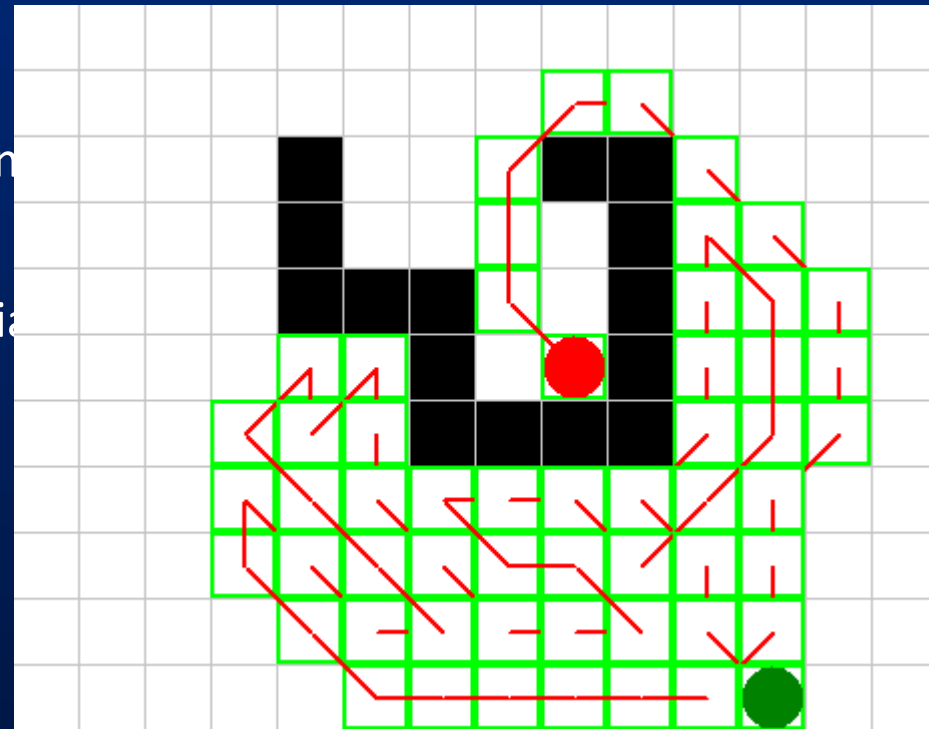
Wady: nieskończona głębokość, możliwe pętle w grafie,  
nie gwarantuje prostoty znalezionego rozwiązania.

Modyfikacja metody DF:

ograniczaj głębokość szukania do poziomu  $d$

Złożoność: czas  $b^d$ , pamięć  $b^d$ .

Wada: jeśli  $d$  za małe nie znajdzie rozwiązania



# IDDF, iteracyjne pogłębianie.

- IDDF lub IDS, iteracyjnie pogłębiane szukanie w głąb.
- Próbuj szukać w głąb stopniowo zwiększając głębokość szukania - tani sposób na realizację szukania w głąb.

Zalety:

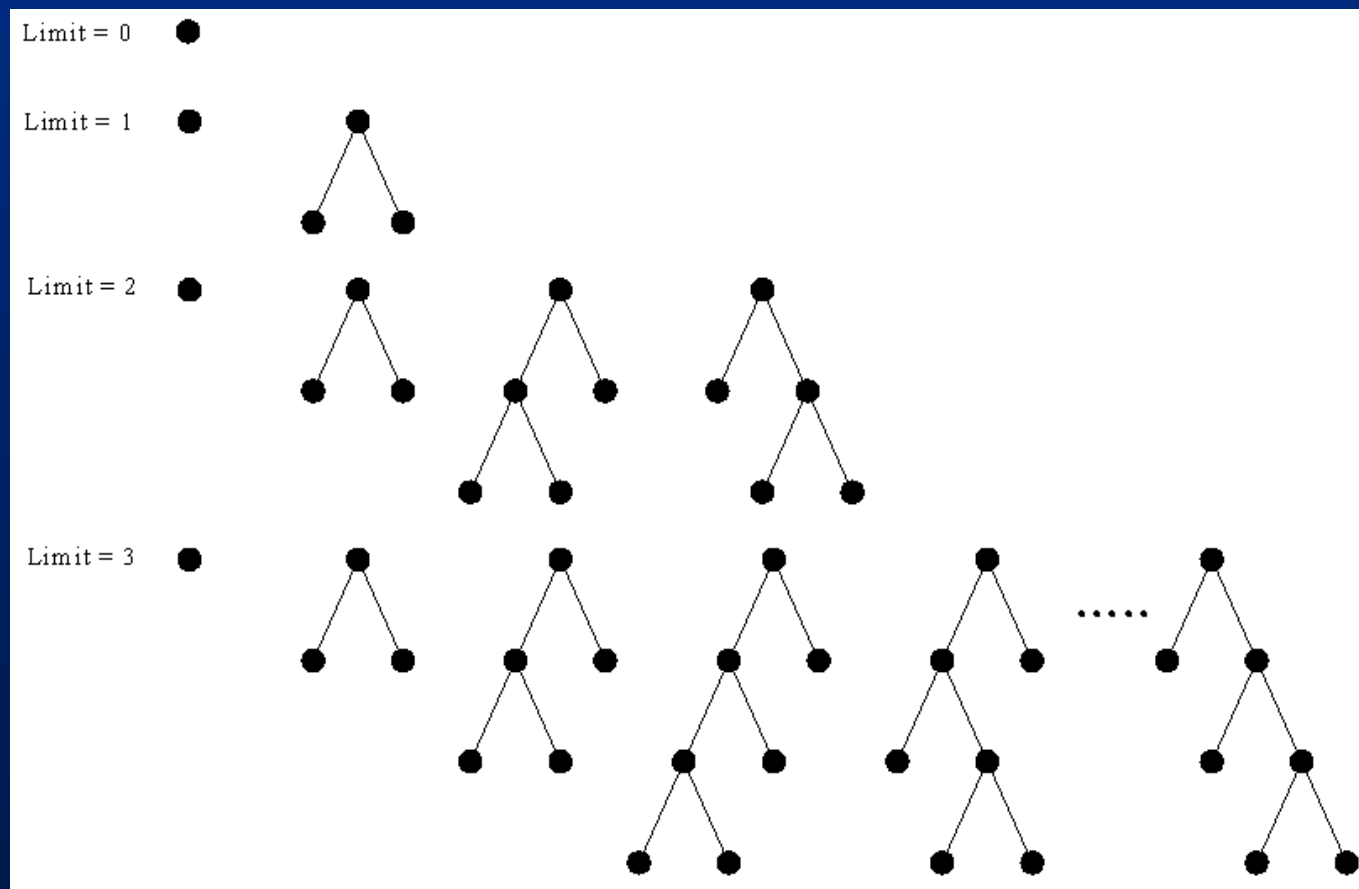
1. Zupełność – zawsze znajduje rozwiązanie.
2. Rozwiązanie jest optymalne (jeśli zwiększamy głębokość o 1)
3. Zużywa tylko  $b^d$  elementów pamięci (szukanie w głąb).

Wady: niewielkie zwiększenie kosztów na powtarzanie – ale ostatecznie szukanie jest bardziej kosztowne niż wszystkie poprzednie, więc złożoność  $\sim O(b^d)$ .



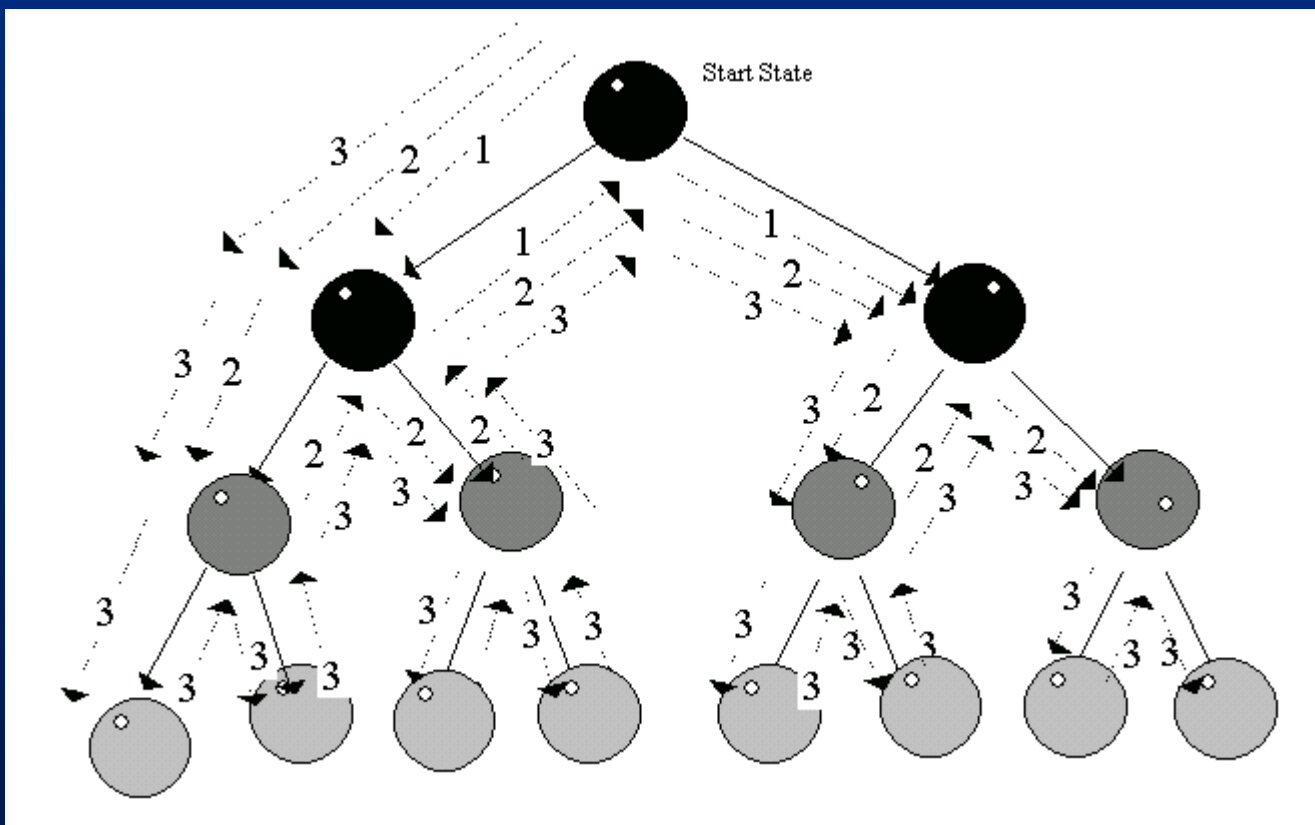
# Ilustracja IDDF

- IDDF lub IDS, iteracyjnie pogłębiane szukanie w głąb.
- Próbuj szukać w głąb stopniowo zwiększając głębokość szukania - tani sposób na realizację szukania w głąb.



# IDDF, iteracyjne pogłębianie.

- IDDF lub IDS, iteracyjnie pogłębiane szukanie w głąb.
- Próbuje szukać w głąb stopniowo zwiększając głębokość szukania - taki sposób na realizację szukania w głąb.



# Wszerz, BFS

1. lista początkowa  $\mathcal{L}(1) = S_w$  (stan wyjściowy)
2. wygeneruj stany  $S_{kj} = O_k \mathcal{L}(j)$  dla wszystkich  $\mathbf{j}$
3. if  $S_{kj} = S_f$  zakończ;
4. else skopiuj  $S_{kj} \rightarrow \mathcal{L}()$  i przejdź do 2

Zalety: znajduje najkrótszą listę operatorów jeśli szuka do końca.

Wady: duża złożoność, przy liczbie operatorów  $b$  i kroków  $d$

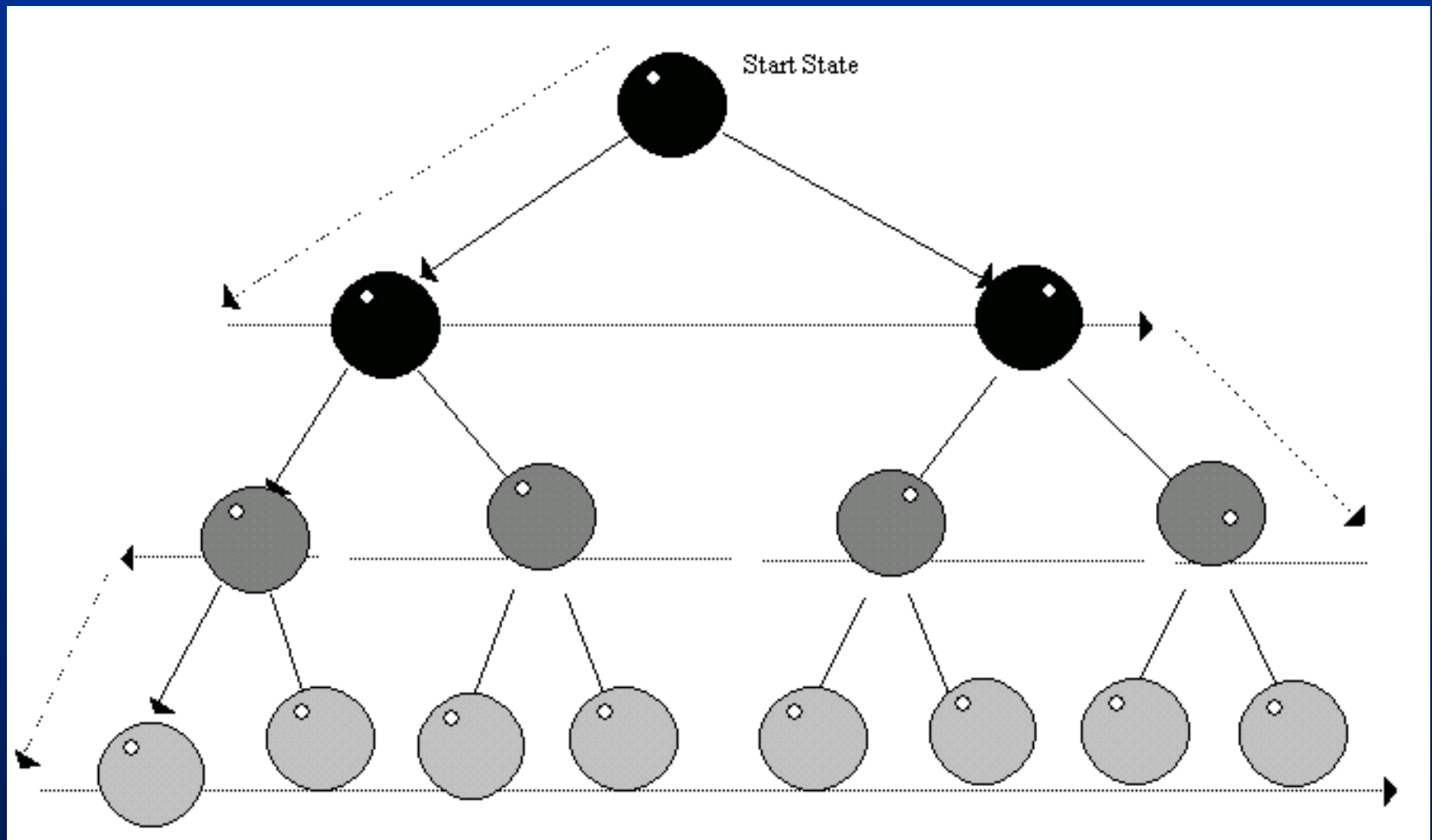
czas  $\sim O(b^d)$ ,

pamięć  $\sim O(b^d)$

Nawet dla 8-ki trzeba pamiętać aż  $(7/3)^{20} \approx 23$  mln stanów.

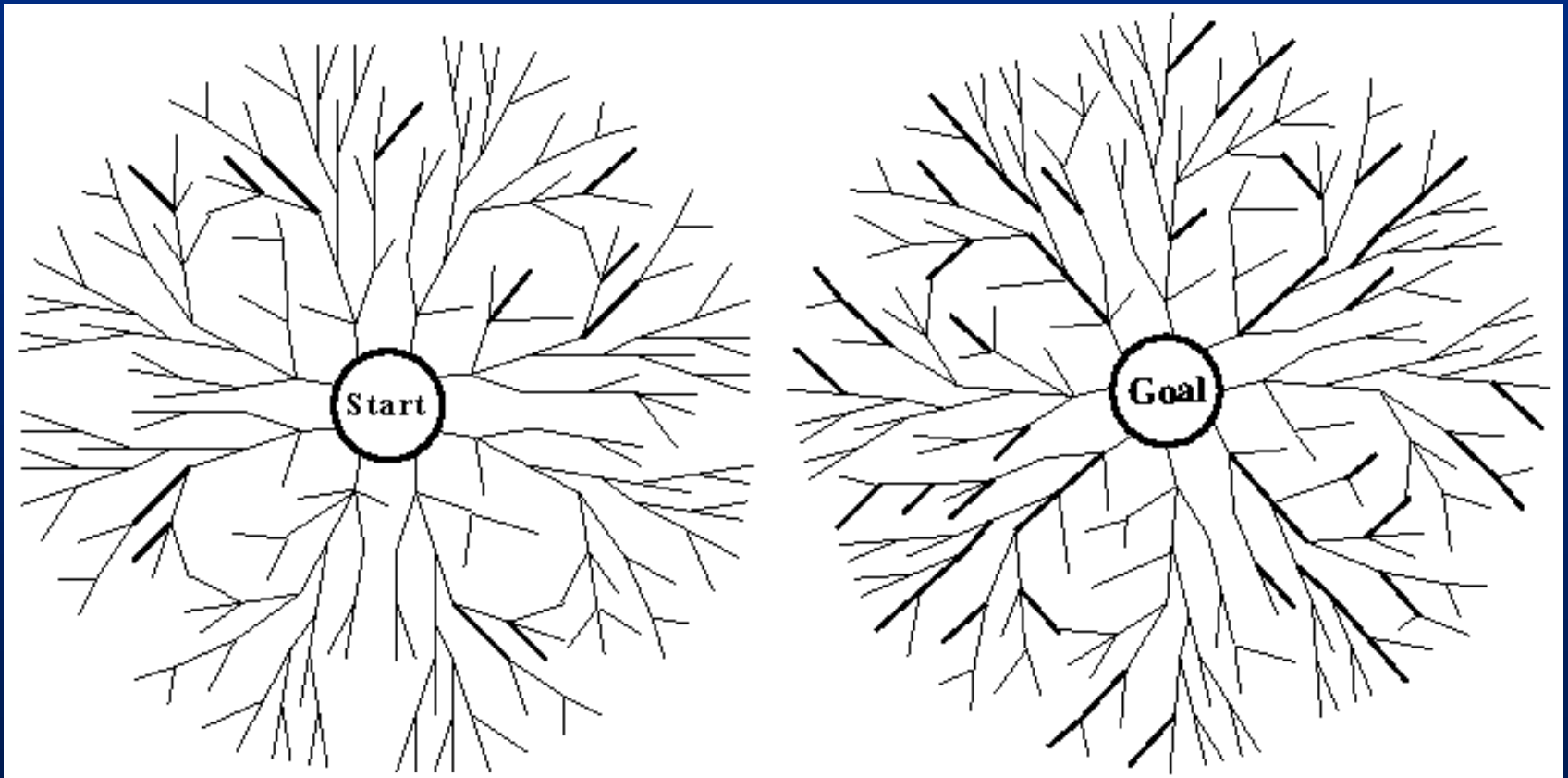
Co-pilot napisze nam ten algorytm w Python.

# Drzewo BFS



# BDS, szukanie dwukierunkowe.

Szukaj wszerek startując od stanu wyjściowego i od stanu końcowego.  
Złożoność: czas  $\sim O(b^{d/2})$ , pamięć  $\sim O(b^{d/2})$



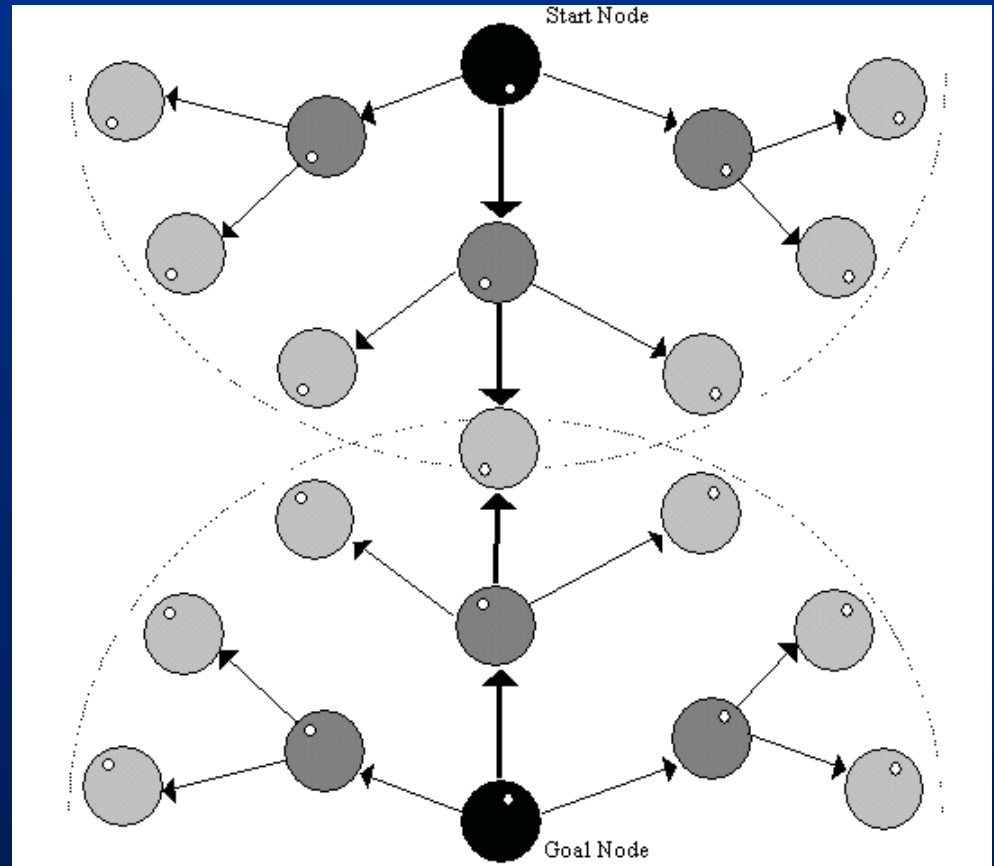
# BDS, szukanie dwukierunkowe.

Szukaj wszerek startując od stanu wyjściowego i od stanu końcowego.

Złożoność:

czas  $\sim O(b^{d/2})$ ,

pamięć  $\sim O(b^{d/2})$



# PathDemo

W programie używane są funkcje:

- Random Bounce. Po dojściu do przeszkody wykonaj jeden krok w przypadkowym kierunku i kontynuuj swój algorytm.
- Simple Trace. Obejdź przeszkodę aż będziesz mógł kontynuować w tym samym kierunku.
- Robust Trace. Oblicz kierunek pomiędzy blokującym klockiem a klockiem końcowym i obchodź przeszkodę aż dojdiesz do kwadratu leżącego w tym kierunku.
- <http://aispace.org/search/> przeszukiwanie grafów w Java
- [PathFind](#) alternatywa wersja na Github

# Algorytm Dijkstry

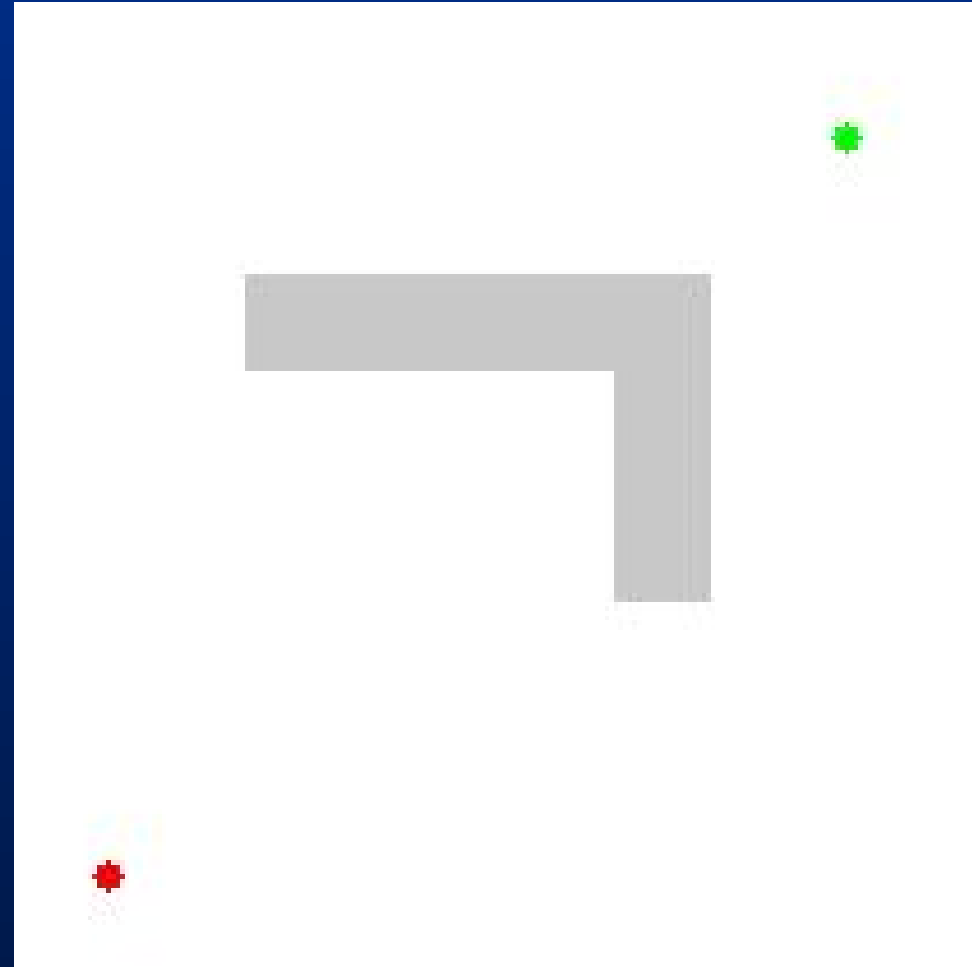


Pierwotnie algorytm najkrótszej drogi w grafie, buduje drzewo najkrótszych dróg metodą zachłanną.

Tworzy drogi o minimalnej długości z wierzchołka  $v$  do wszystkich pozostałych, tworzy drzewo rozpinające zawierające najkrótsze drogi z wierzchołka  $v$ .

Dla  $n$  wierzchołków złożoność czasowa jest rzędu  $O(n^2)$ .

Przykład [na geeksforgeeks](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm/)

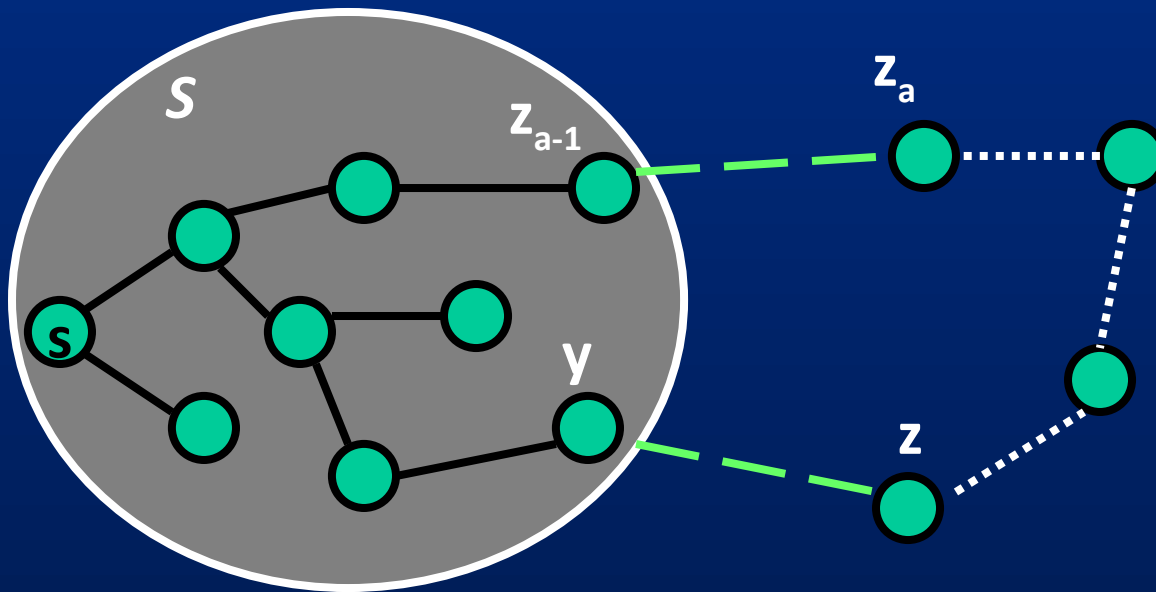




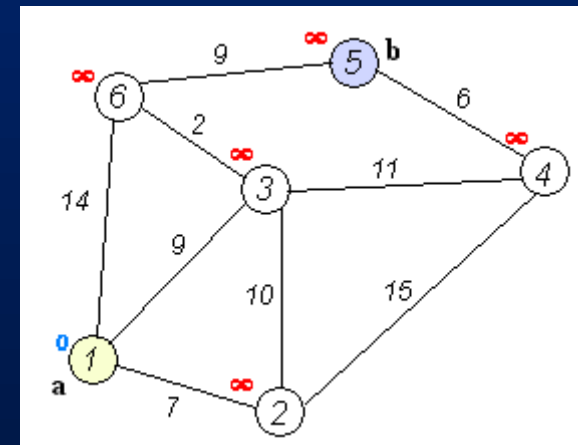
# Algorytm Dijkstry



Algorytm najkrótszej drogi w grafie buduje drzewo najkrótszych dróg do kolejnych węzłów metodą zachłanną i pamięta najkrótsze.



Animacja algorytmu



# Algorytm Dijkstry

Oznaczenia:  $U$ =otwarte,  $Z$ =węzły,  $B$ =bieżący węzeł.

Drzewo  $S = \{\text{Start}\}$ ,  $D(y)=D(y,S)$ , koszt od  $S$ ;  $d(y)=d(y,B)$  koszt od  $B$

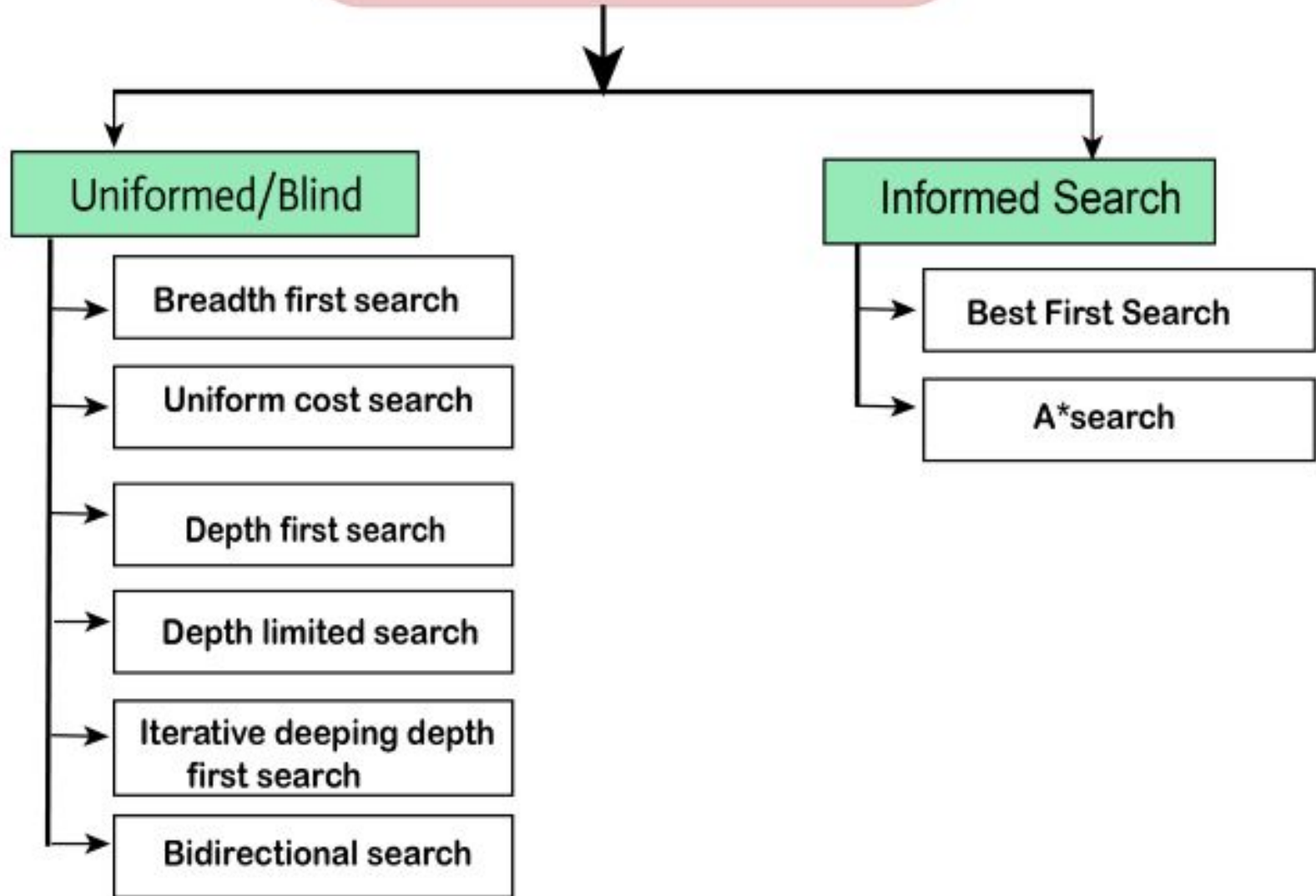
1. Budujemy drzewo  $S$ , bieżący węzeł  $D(B=S)=0$ , pozostałe  $D(y \in U)=\infty$
2. Utwórz wszystkie możliwe węzły  $y \in U$  połączone z bieżącym węzłem  $B$ .
3. Oblicz  $d(y)+d(B)$ , wybierz  $D(y) = \min(D(y), d(y)+d(B))$ , droga  $B \leftrightarrow y$  najkrótsza
4. Zamknij bieżący węzeł  $B$ , usuwając go z  $U$ .
5. Węzeł bieżący = następny otwarty węzeł o najniższym koszcie  $d(B)$ .
6. Powtarzaj 2-5 aż dodanym węzłem będzie Cel lub  $U=\emptyset$ .

- Drzewo szukania rośnie przez dokładanie węzłów.
- Algorytm gwarantuje znalezienie najkrótszej drogi.

Złożoność dla grafu o  $n$  wierzchołkach jest rzędu  $O(n^2)$

Mniejsza złożoność: implementacja z kolejką priorytetową z kopcem Fibonacciego  $\sim O(n \log n + K)$  dla  $K$  krawędzi drzewa.

# Search Algorithm



# Literatura

ChatGPT, Co-pilot i inne systemy potrafią napisać algorytmy i wyjaśnić dokładnie ich działania a także ocenić ich złożoność.

- Wiele repozytoriów zawiera opis i demonstracje algorytmów szukania, np..  
[geeksforgeeks.org](https://www.geeksforgeeks.org)