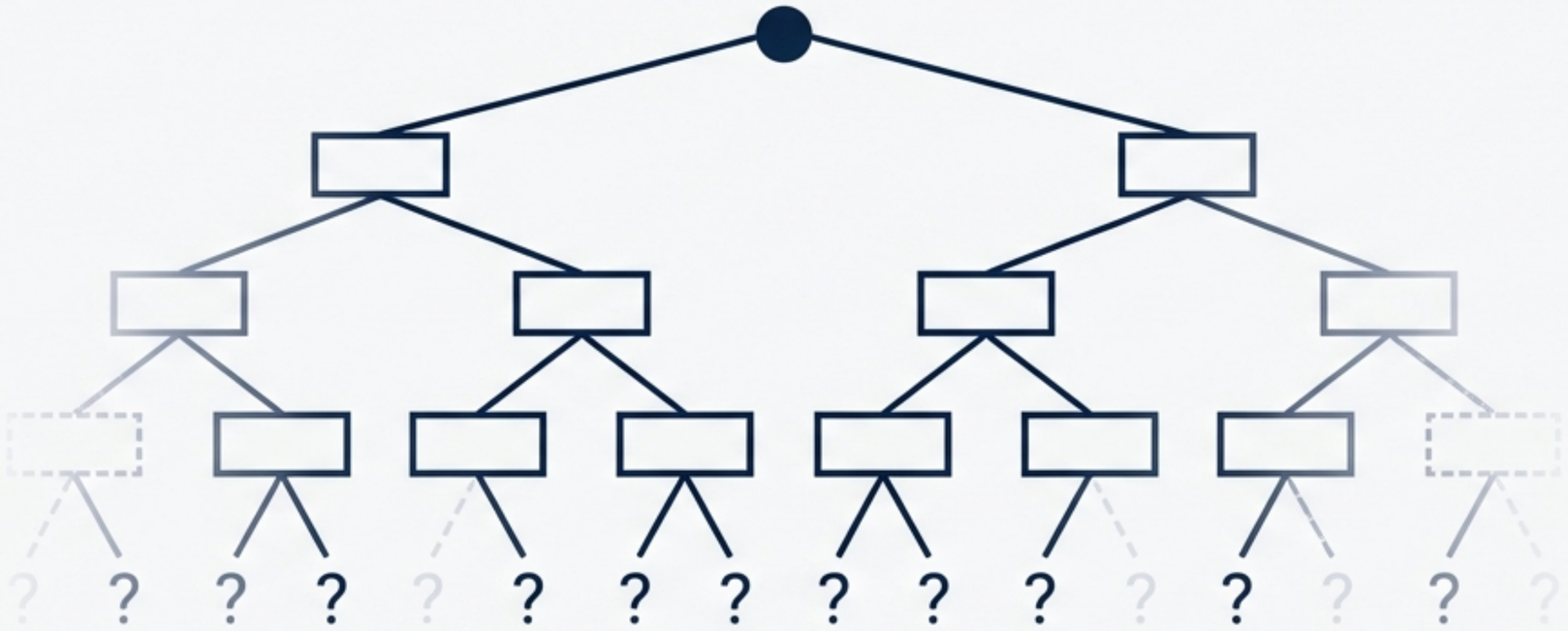
Sztuczna Inteligencja: Metody szukania na ślepo

Strategie przeszukiwania przestrzeni stanów



Wykład: Włodzisław Duch
Katedra Informatyki Stosowanej UMK

„Szukanie – metoda uniwersalna.”

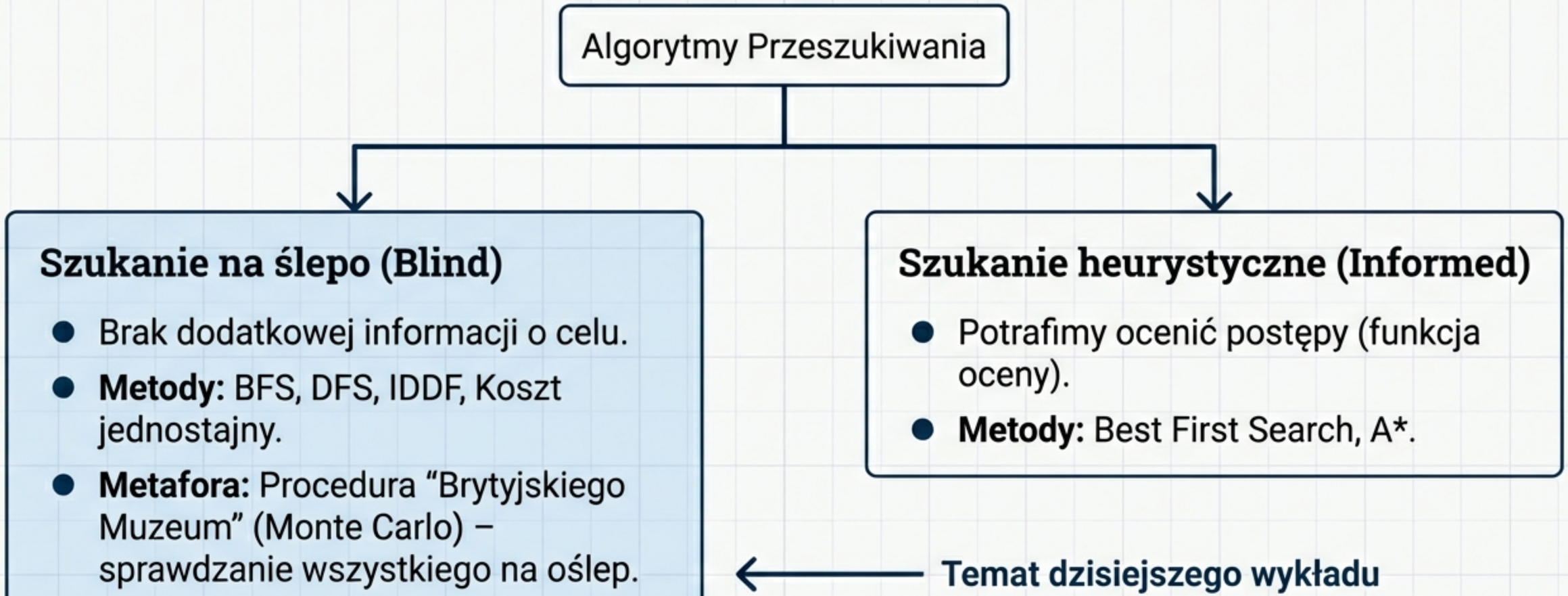
Szukanie jako mechanizm rozumowania

Definicja problemu:

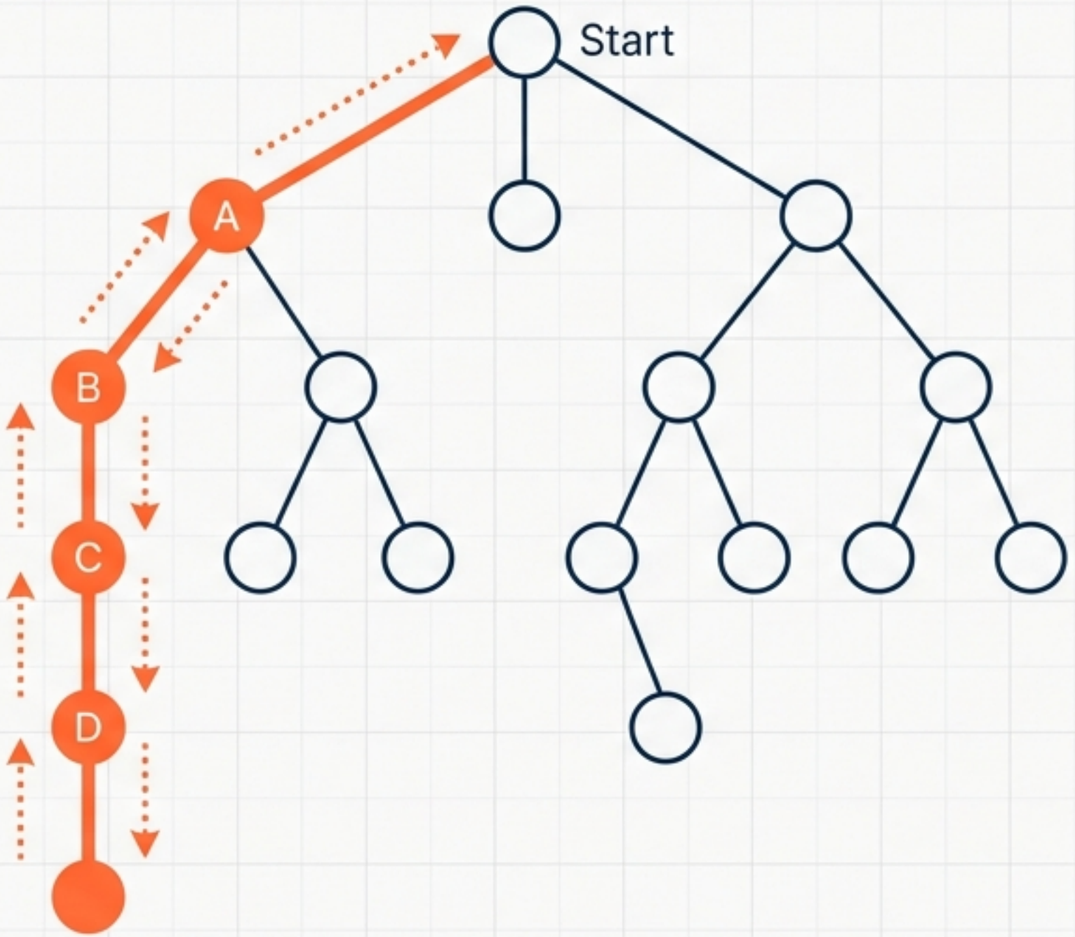
- 1. "**Baza danych**" – Fakty, stany, opis sytuacji.
- 2. "**Operacje**" – Ruchy zmieniające stan bazy.

Ograniczenie: Dotyczy problemów dyskretnych i kombinatorycznych. (Percepcja sygnałów ciągłych wymaga dyskretyzacji).

Dwie kategorie procedur szukania



Strategia „W głąb” (DFS – Depth First Search)



Filozofia:

Idź tak głęboko, jak się da, zanim się cofniesz.

Logika działania:

1. Generuj stan $L(k+1) = O_j L(k)$
2. Jeśli to rozwiązanie (S_f) → **Stop** ✅
3. Jeśli pętla ($L(k+1) = L(m)$) → **Cofnij się** ($k = k-1$)

Struktura danych:

Stos (LIFO – Last In, First Out)

Implementacja DFS: Rekurencja vs. Iteracja

Wersja Rekurencyjna

```python
def dfs_recursive(graph, vertex, visited=None):
    if visited is None:
        visited = set()

    visited.add(vertex)
    print(vertex, end=' ')

    for neighbor in graph[vertex]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbor, visited)
```

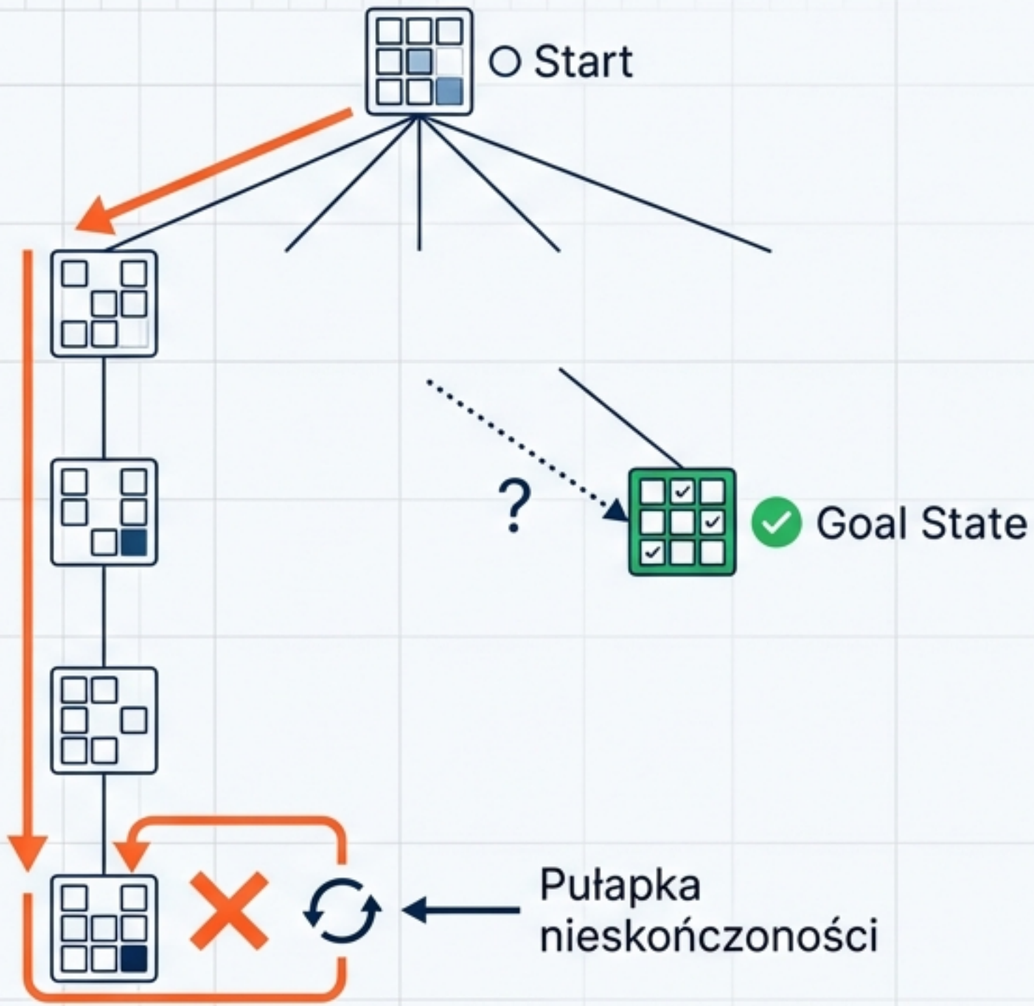
Stos ukryty w wywołaniach funkcji

Wersja Iteracyjna (ze Stosem)

```python
def dfs_iterative(graph, start):
    visited = set()
    stack = [start]  # Jawny stos (LIFO)

    while stack:
        vertex = stack.pop()

        if vertex not in visited:
            visited.add(vertex)
            print(vertex, end=' ')

            # Dodaj sąsiadów w odwrotnej kolejności
            for neighbor in reversed(graph[vertex]):
                if neighbor not in visited:
                    stack.append(neighbor)
```

Kluczowy mechanizm LIFO

Wady i Zalety DFS



Zalety (+):

- Niska złożoność pamięciowa: $O(d)$ (liniowa względem głębokości)
- Szybkie, jeśli istnieje wiele rozwiązań głęboko w drzewie.

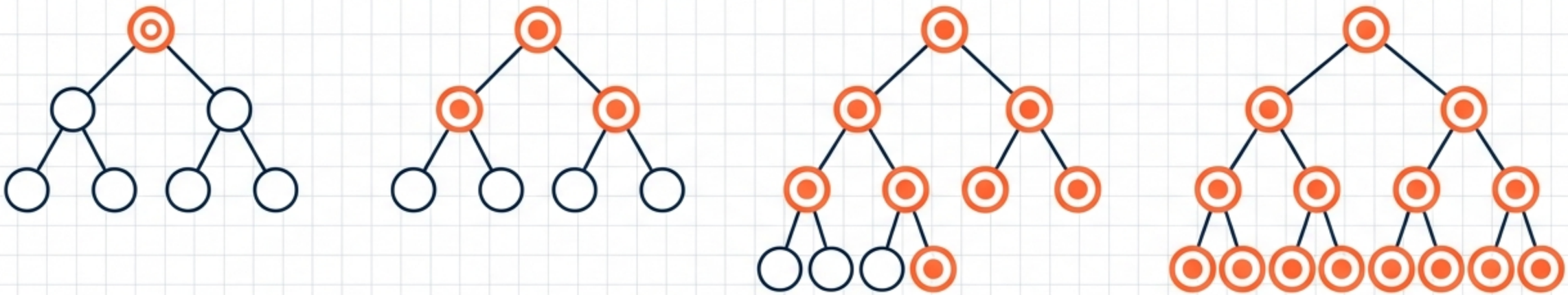
Wady (-):

- Nieskończona głębokość (ryzyko braku stopu).
- Możliwe pętle (wymaga pamiętania historii).
- **Brak optymalności**: Nie gwarantuje znalezienia najkrótszej drogi.

IDDF – Iteracyjne Pogłębianie

Iterative Deepening Depth First Search

Kompromis: Tani sposób na realizację szukania w głąb z zaletami szukania wszerz.



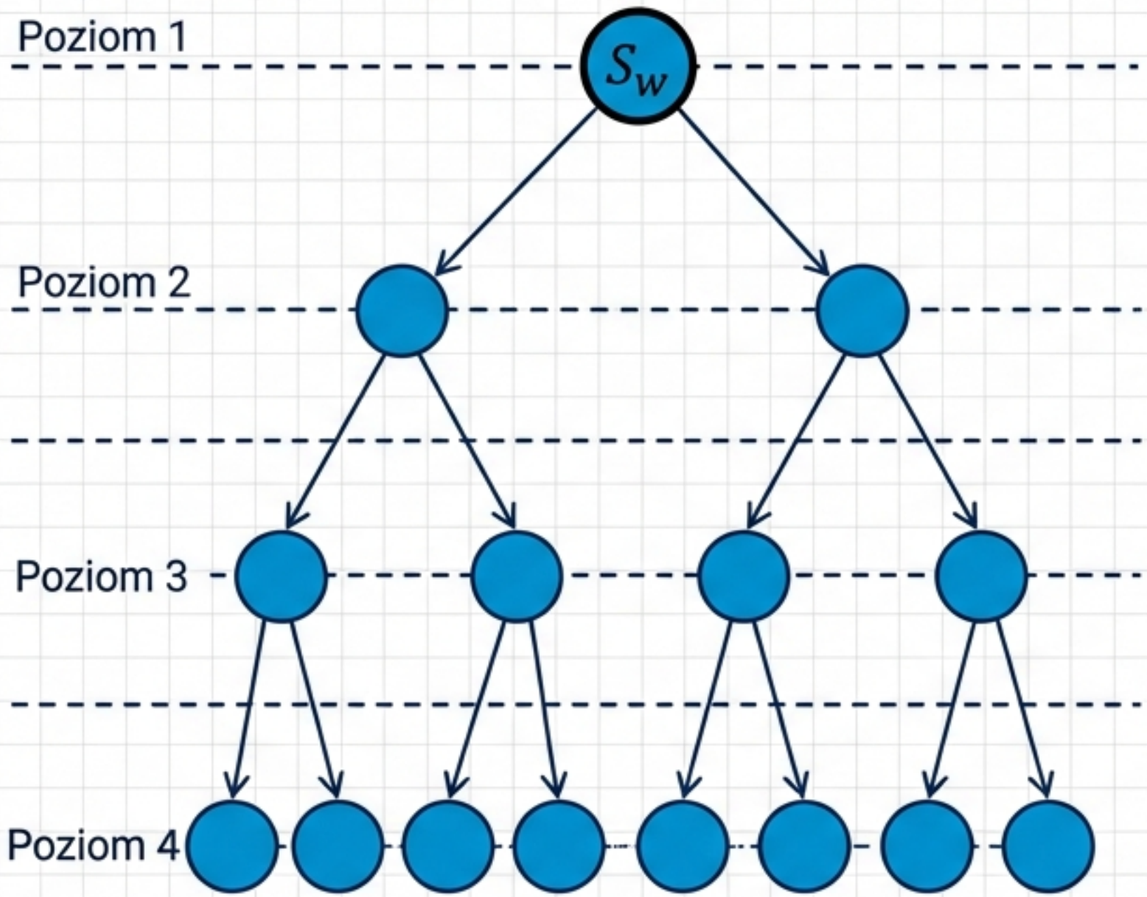
Limit = 0 | Limit = 1 | Limit = 2 | Limit = 3

Zupełność:
TAK (zawsze znajdzie, jeśli istnieje)

Optymalność:
TAK (znajdzie najpłytsze rozwiązanie)

Pamięć:
$O(bd)$ (Niska!)

Strategia „Wszerz” (BFS – Breadth First Search)



Filozofia:

Sprawdź wszystko blisko, zanim pójdziesz dalej.

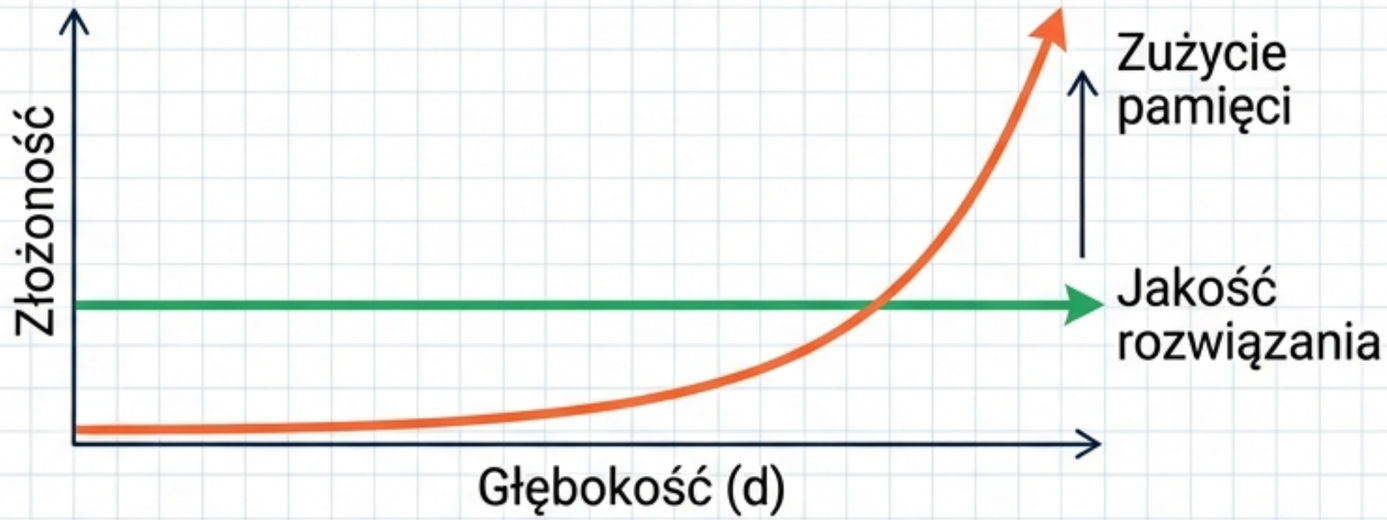
Logika działania:

1. Lista początkowa $L(1) = S_w$
2. Wygeneruj wszystkie stany potomne dla bieżącej warstwy.
3. Sprawdź czy cel został osiągnięty.
4. Jeśli nie, dodaj nowe stany do kolejki.

Struktura danych:

Kolejka (FIFO – First In, First Out)

Koszt BFS: Eksplozja Kombinatoryczna



Złożoność Matematyczna

Złożoność Czasowa: $O(b^d)$

Złożoność Pamięciowa: $O(b^d)$

(gdzie *b* = rozgałęzienia, *d* = głębokość)

Przykład Rzeczywisty: Układanka "Przesuwanka" (8-puzzle)

Dla głębokości 20 kroków trzeba zapamiętać:

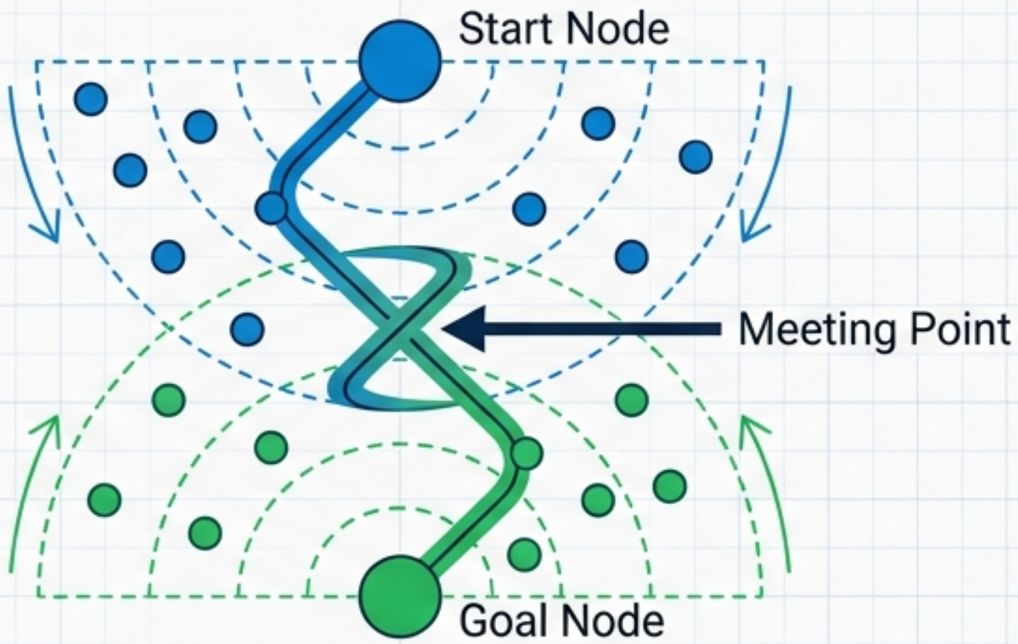
~23 miliony stanów

$(7/3)^{20}$

BFS jest optymalny, ale kosztowny pamięciowo.

BDS – Szukanie Dwukierunkowe

Bidirectional Search



Metoda:

Szukaj wszerz (BFS) startując jednocześnie od stanu wyjściowego (Start) i od stanu końcowego (Goal).

Zysk obliczeniowy:

Zamiast przeszukiwać obszar o promieniu d (b^d)...

...przeszukujemy dwa obszary o promieniu $d/2$.

Złożoność:

$$O(b^{d/2})$$

Drastyczna redukcja przestrzeni poszukiwań.

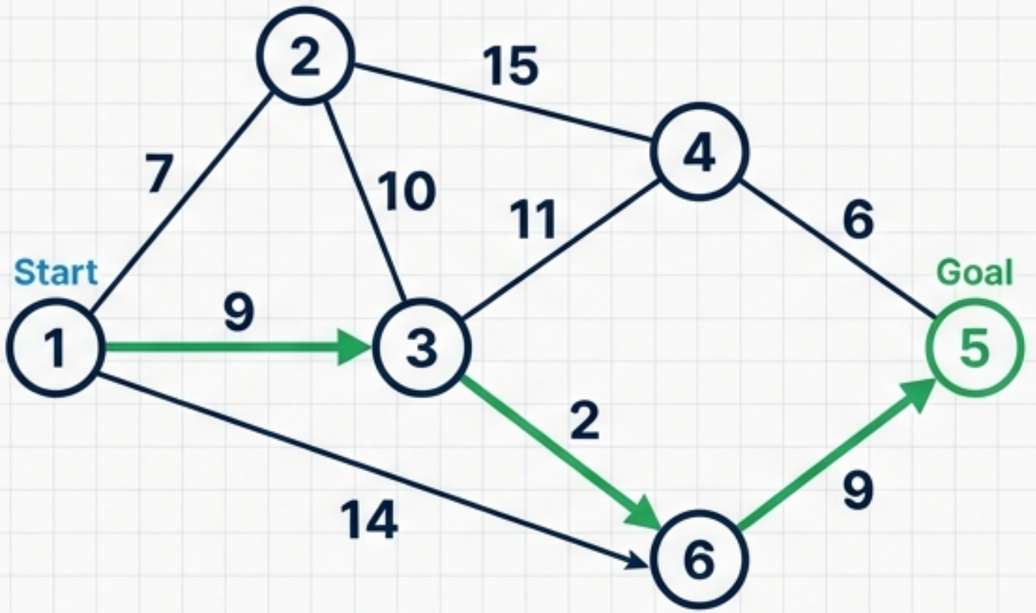
Algorytm Dijkstry: Wprowadzenie Kosztu

Nowe założenie:

Nie wszystkie kroki są równe. Krawędzie grafu posiadają wagi (koszty).

BFS: Szuka najmniejszej *liczby kroków*.

Dijkstra: Szuka drogi o *minimalnym sumarycznym koszcie*.



1 -> 3 (9) -> 6 (2) -> 5 (9) = Total Cost: 20

1 -> 2 (7) -> 4 (15) -> 5 (6) = Total Cost: 28

1 -> 6 (14) -> 5 (9) = Total Cost: 23

Metoda Zachłanna (Greedy) – zawsze wybieraj najtańszy dostępny węzeł.

Logika Dijkstry: Relaksacja Krawędzi

Pseudokod

1. **Inicjalizacja**: $D(Start) = 0$, reszta = ∞.
2. **Wybór**: Wybierz węzeł B o najniższym koszcie.
3. **Relaksacja**:

   Dla każdego sąsiada y węzła B:

   ```
   nowy_koszt = d(B) + waga(B, y)
   if nowy_koszt < D(y):
       D(y) = nowy_koszt
   ```
4. **Pętla**: Powtarzaj aż dotrzesz do Celu.

Wizualizacja

Relaksacja: Znaleziono krótszą ścieżkę do B przez A. Koszt *D*(B) zaktualizowany z ∞ na 8.

Złożoność Algorytmu Dijkstry

Standardowa implementacja (Tablica/Lista)

Złożoność czasowa:

$$O(n^2)$$

(dla n wierzchołków)

Optymalizacja

Użycie kolejki priorytetowej (Kopiec Fibonacciego)

Złożoność czasowa:

$$O(n \log n + K)$$

(gdzie K to liczba krawędzi)

Gwarancja

Warunek konieczny:

Nieujemne wagi krawędzi.

Wynik:

Zawsze optymalna (najtańsza) droga.

Podsumowanie Metod „Na Ślepo”

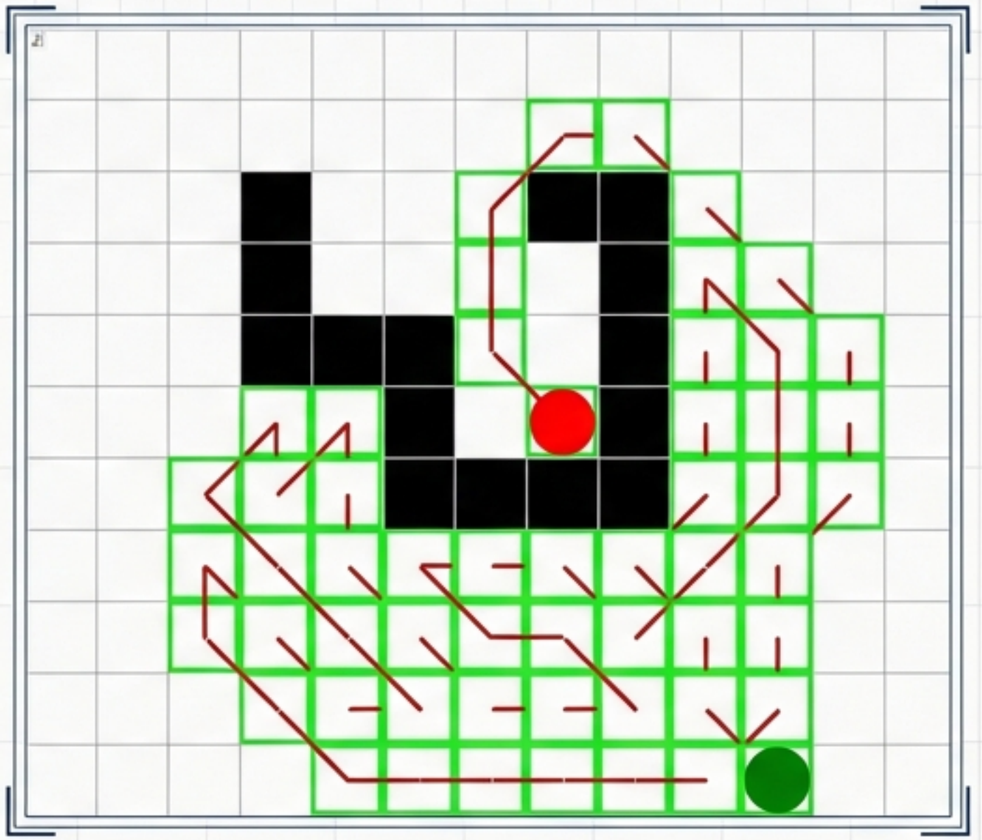
Algorytm	Czas	Pamięć	Optymalny?	Zupełny?
BFS	$O(b^d)$	$O(b^d)$ (Wysoka)	Tak (liczba kroków)	Tak
DFS	$O(b^d)$	$O(d)$ (Niska)	Nie	Nie (możliwe pętle)
IDDF	$O(b^d)$	$O(bd)$ (Niska)	Tak	Tak
BDS (Dwukierunkowe)	$O(b^{d/2})$	$O(b^{d/2})$	Tak	Tak

Wniosek: IDDF jest często najlepszym kompromisem dla grafów nieważonych. Dijkstra jest standardem dla grafów ważonych.

Materiały i Dalsze Kroki

Narzędzia do wizualizacji:

- PathDemo (Random Bounce, Simple Trace)
- GeeksForGeeks (Animacje)
- GitHub: Repozytoria “PathFind”



Metody na ślepo to fundament. Kolejny krok: Jak wykorzystać wiedzę, by szukać mądrzej? (A* i Heurystyki)