# Sztuczna inteligencja. Przeszukiwanie (bez informacji)

Paweł Rychlikowski

Instytut Informatyki UWr

6 marca 2018

# Literatura

- Stuart Russel, Peter Norvig, Artifficial Intelligence. A Modern Approach. 3rd edition (w Internecie leży pdf)
- Fajna, ale 1100 stron.
- Będą pojawiać się inne podręczniki...

**Wykład na Stanfordzie**

CS221: Artificial Intelligence: Principles and Techniques

Są też książki po polsku, ale one nie do końca odpowiadają naszemu wykladowi. O nich jeszcze powiemy.

# Zadania przeszukiwania na pierwszej liście

- Zadanie szachowe
- Rozwiązywanie obrazków logicznych

W mniejszym stopniu spacjowanie

# Masz jakiś problem?

## Definicja

Problem ma następujące pięć komponentów

1. Stan początkowy (i zbiór stanów, ale być może dany implicite)
2. Zbiór akcji (co agent może robić)
3. Model przejścia (stan + akcja = nowy_stan)
4. Test określający, czy stan jest końcowy (i znaleźliśmy rozwiązanie)
5. Sposób obliczania kosztu ścieżki (najczęściej podawany jako koszt akcji w stanie)

# Akcje i model przejścia

## Uwaga

Akcje oraz model przejścia są ze sobą powiązane. Jest kilka wariantów:

1. Zbiór akcji wspólny, funkcja przejścia pilnuje, żeby niemożliwe akcje nie zmieniały stanu
   Przykład: ludzik idzie w labiryncie na ścianę
2. Akcje to funkcja, która dla stanu zwraca zbiór czynności, które agent może zrobić w stanie
3. Akcja i model przejścia to jedna funkcja, która dla stanu zwraca listę par postaci: (akcja, nowy_stan)

# Poziomy abstrakcji.

Zadania można modelować na wiele różnych sposobów. Dla podróżowania MPK mamy następujące akcje/sekwencje akcji

- Przejedź z przystanku A do przystanku B
- Wsiądź do tramwaju na przystanku A, skasuj bilet, zajmij wygodne miejsce, obserwuj tablicę, podejdź do drzwi, gdy...
- Wykonuj naprzemienne ruchy lewą i prawą nogą, aż ...
- Napnij głowę większą mięśnia dwugłowego lewego uda, ...

**Uwaga**

Akcje muszą być zrozumiałe dla agenta, im wyższy poziom, tym łatwiejsze zadanie przeszukiwania.

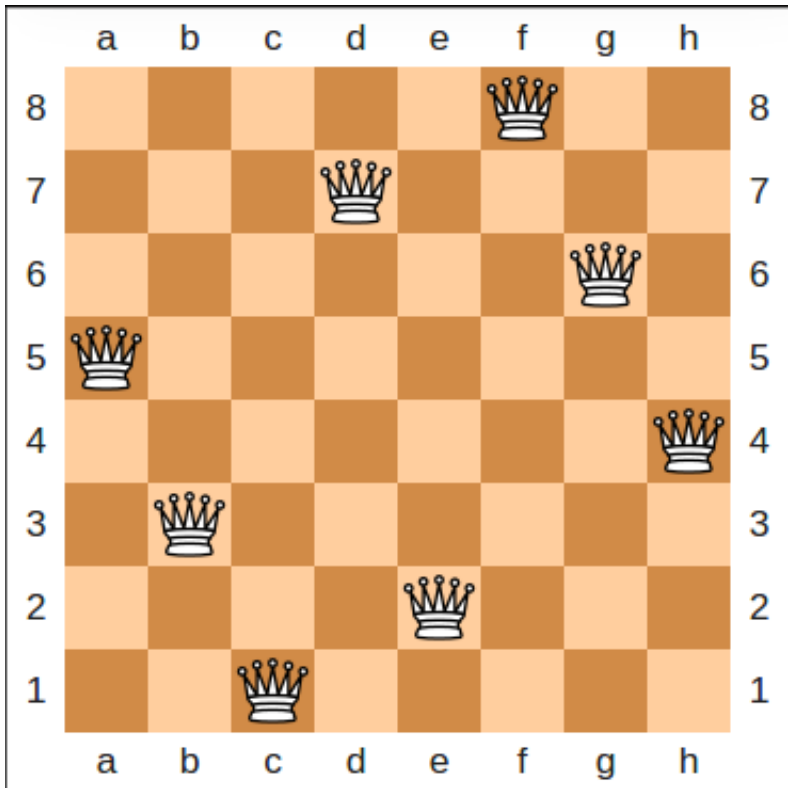
# Przykłady

- Przeanalizujemy dla kilku przykładów jak można pewne zadania przedstawiać jako problemy przeszukiwania

# Piętnastka

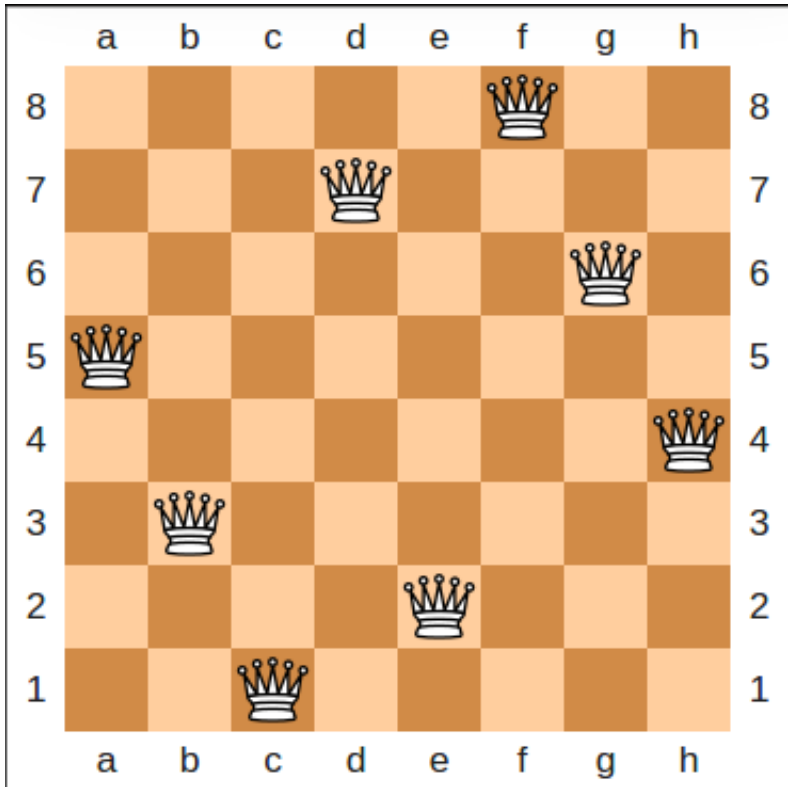

- Stan początkowy: jakieś ułożenie, na przykład powyższe
- Stan końcowy: liczby po kolei
- Koszt: jednostkowy
- Model: dowolna zamiana pustego kwadracika z sąsiadem

# Problem ośmiu hetmanów




- W przeciwieństwie do poprzedniego przykładu istnieje tu wiele sformułowań.
- Jakie są dwie najważniejsze opcje?

# Problem ośmiu hetmanów




Dwa sposoby opisywania zadania jako problemu przeszukiwania:

1. Stan kompletny (rozważamy sytuacje z 8 hetmanami na planszy, ruch to przestawienie hetmana)
2. Stan niepełny – zaczynamy od pustej planszy, ruchem jest postawienie hetmana.

# Szczypta hetmańskiej arytmetyki

- $8! = 40320$
- $8^8 = 16777216$
- $(8!)^2 = 1625702400$
- $8! \times 8^8 = 676457349120$
- $64 \times 63 \times \cdots \times 57 = 178462987637760$

**Pytanie**

Co oznaczają te liczby?

**Uwaga**

Wybierając reprezentację rzeczywistości mamy wielki wpływ na efektywność (i realistyczność) poszukiwań!

# Szczypta hetmańskiej arytmetyki (wyjaśnienie)

**Uwaga**

Rozmieszczamy hetmany wg określonego schematu, rozpoczynając od pustej planszy. Liczymy liczbę stanów.

- $64 \times 63 \times \cdots \times 57 = 178462987637760$ – rozmieszczamy po kolei hetmany, na jednym polu jest 1 hetman.
- $8^8 = 16777216$ – ustalona kolejność, w każdej kolumnie 1 hetman
- $8! = 40320$ – ustalona kolejność, w każdej kolumnie 1 hetman, nie szachują się w wierszach
- $(8!)^2 = 1625702400$ – nieustalona kolejność, w każdej kolumnie 1 hetman, nie szachują się w wierszach (to wielkość stanu, gdy uwzględniamy historię)
- $8! \times 8^8 = 676457349120$ – nieustalona kolejność, w każdej kolumnie 1 hetman (to wielkość stanu, gdy uwzględniamy historię)

# Problem zabawkowy: hipoteza Knutha

## Hipoteza

Zaczynając od 4 możemy dojść do dowolnej liczby wykonując operacje: silni, pierwiastka i podłogi (części całkowitej, `int`).

Przykładowo:

$$\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \rfloor = 5$$

Sformułowanie dość oczywiste (stany to liczby)

## Uwaga

Przestrzeń jest nieograniczona (nie znamy żadnego twierdzenia, które ograniczałoby przestrzeń prostą funkcją).

# Modyfikacje zadania znajdowania marszruty

Przesuwamy się w stronę rzeczywistych zadań.

## Wariant 1

W co $K$-tej miejscowości na trasie znajduje się dobra knajpa.

## Wariant 2

Powinniśmy zaliczyć co najmniej $K$ dodatkowych atrakcji turystycznych.

Zastanówmy się, co jest stanem w powyższych zadaniach?

# Inne zadania z rzeczywistego świata

- Projektowanie układów scalonych (jak umieścić funkcję logiczną w ograniczonej przestrzeni takiego układu)
- Nawigacja robotów (musimy uwzględniać stawy itd)
- Składanie produktów przez robota (organizacja procesu produkcji, również związków chemicznych)
- Ogólnie logistyka

# Oczekiwania wobec rozwiązania

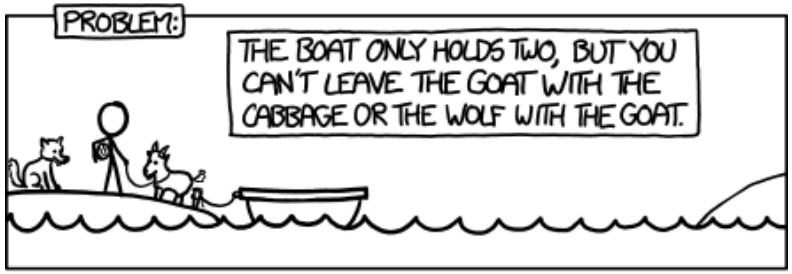
- Zupełność: czy program znajdzie drogę do rozwiązania, jeżeli takowa istnieje?
  Czy to jest konieczny warunek użyteczności algorytmu?
- Optymalność: czy będzie ona najkrótsza
- Złożoność czasowa: jak długo będzie trwało szukanie
- Złożoność pamięciowa: ile zużyjemy pamięci

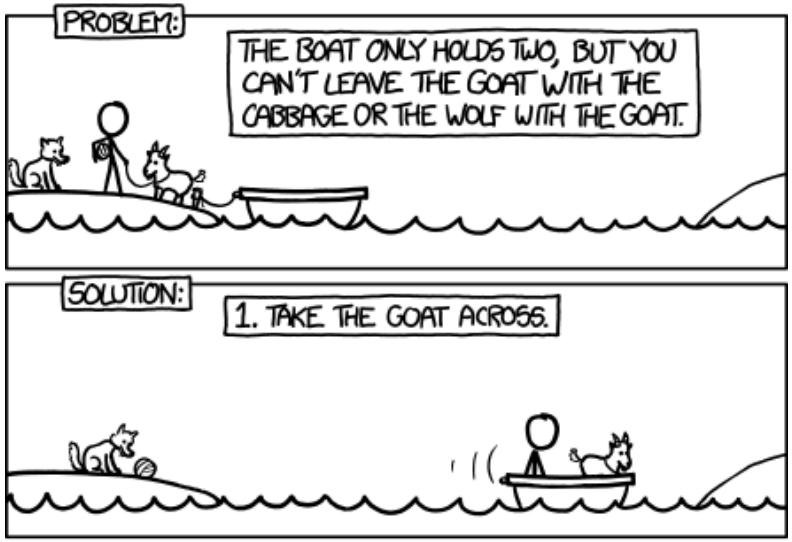
**Pytanie**

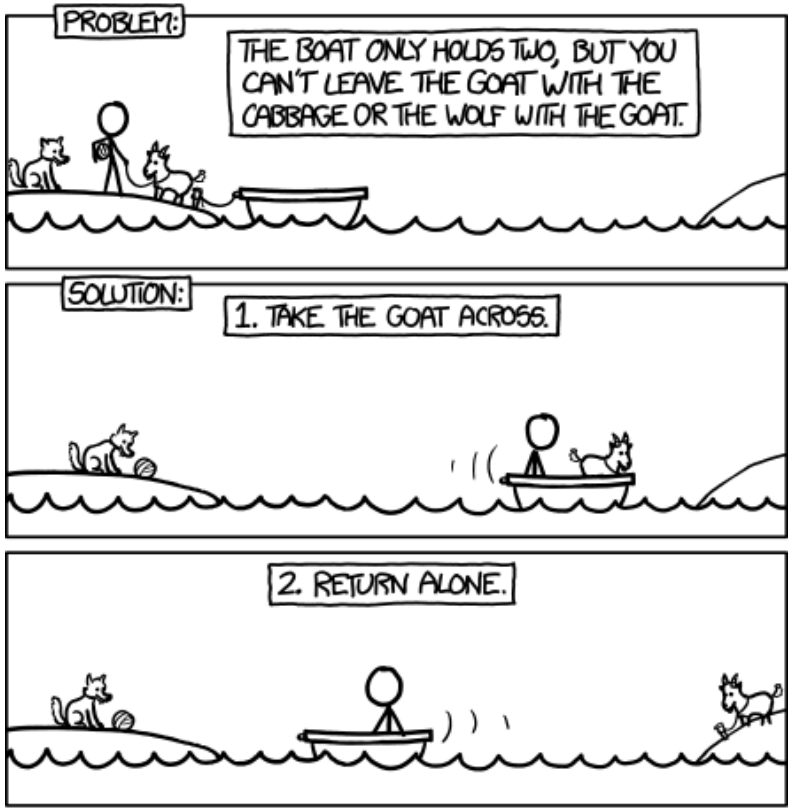
Dlaczego tak ważna jest złożoność pamięciowa?

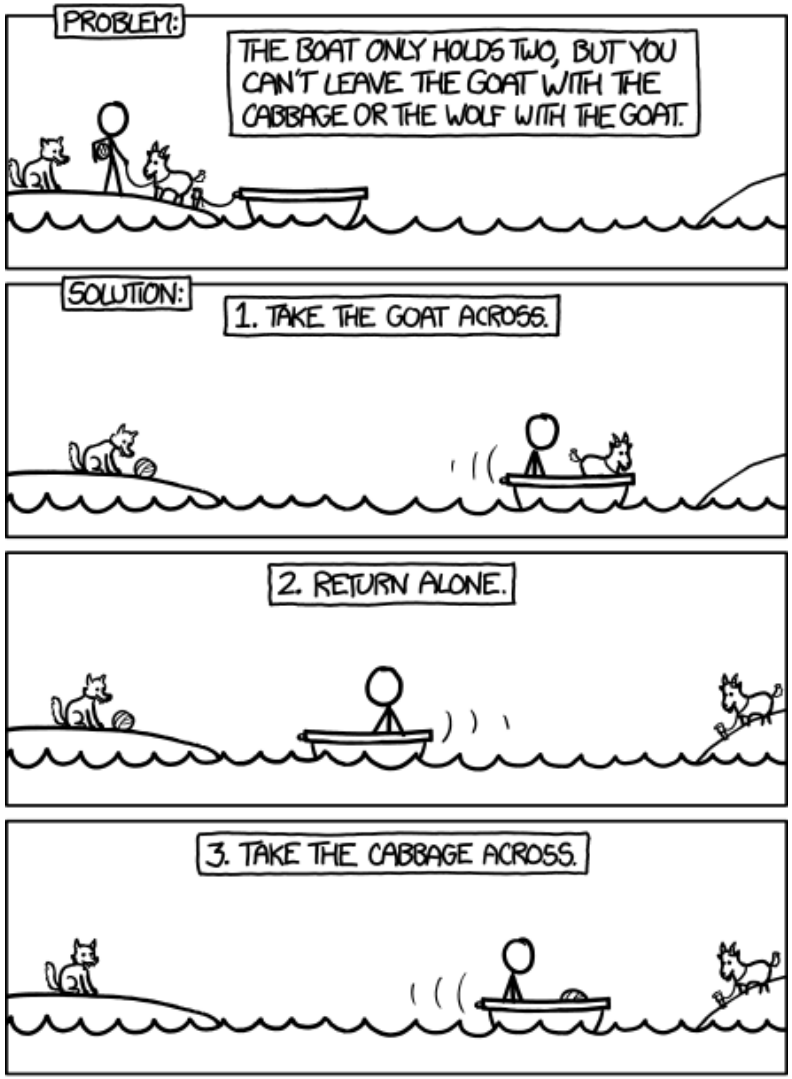
# Drzewo poszukiwań. Wilk, koza i kapusta

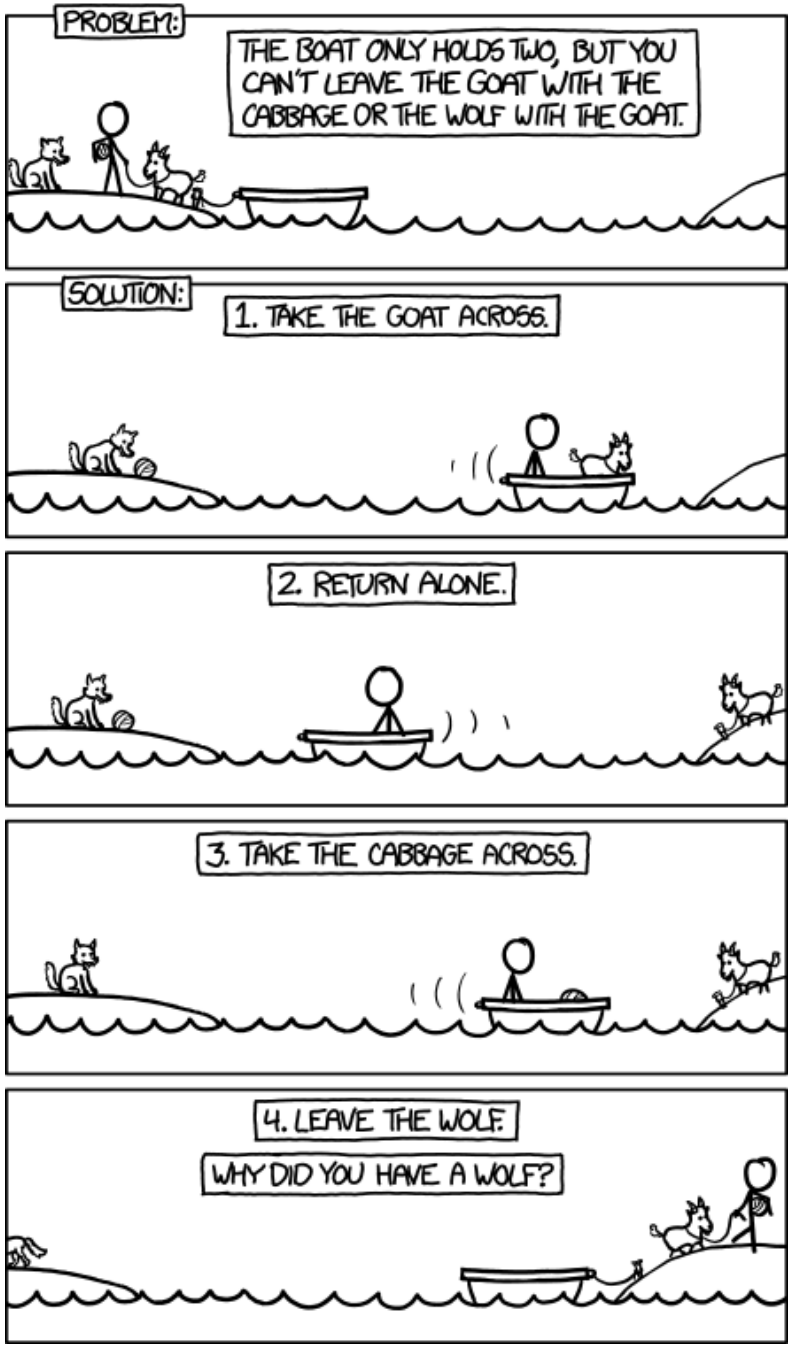
- Znacie zadanie o Wilku, kozie i kapuście?
- Jak nie, to popatrzcie (źródło, oczywiście, xkcd.com)

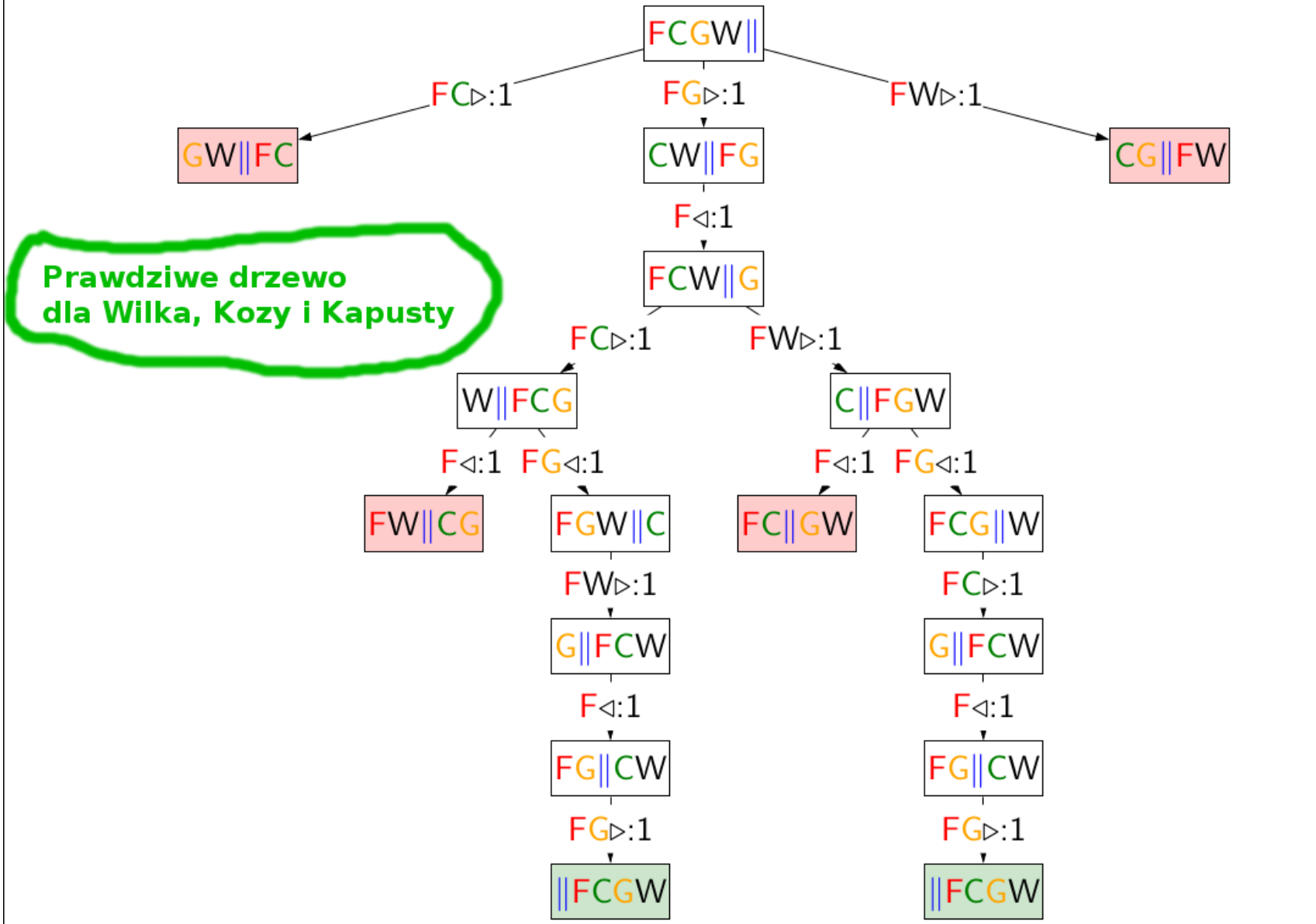

PROBLEM:
THE BOAT ONLY HOLDS TWO, BUT YOU CAN'T LEAVE THE GOAT WITH THE CABBAGE OR THE WOLF WITH THE GOAT.


PROBLEM:
THE BOAT ONLY HOLDS TWO, BUT YOU CAN'T LEAVE THE GOAT WITH THE CABBAGE OR THE WOLF WITH THE GOAT.
SOLUTION:
1. TAKE THE GOAT ACROSS.


PROBLEM:
THE BOAT ONLY HOLDS TWO, BUT YOU CAN'T LEAVE THE GOAT WITH THE CABBAGE OR THE WOLF WITH THE GOAT.
SOLUTION:
1. TAKE THE GOAT ACROSS.
2. RETURN ALONE.


PROBLEM:
THE BOAT ONLY HOLDS TWO, BUT YOU CAN'T LEAVE THE GOAT WITH THE CABBAGE OR THE WOLF WITH THE GOAT.
SOLUTION:
1. TAKE THE GOAT ACROSS.
2. RETURN ALONE.
3. TAKE THE CABBAGE ACROSS.


PROBLEM:
THE BOAT ONLY HOLDS TWO, BUT YOU CAN'T LEAVE THE GOAT WITH THE CABBAGE OR THE WOLF WITH THE GOAT.
SOLUTION:
1. TAKE THE GOAT ACROSS.
2. RETURN ALONE.
3. TAKE THE CABBAGE ACROSS.
4. LEAVE THE WOLF.
WHY DID YOU HAVE A WOLF?


FCGW||
FC▷:1
FG▷:1
FW▷:1
GW||FC
CW||FG
CG||FW
F◁:1
FCW||G
Prawdziwe drzewo
dla Wilka, Kozy i Kapusty
FC▷:1
FW▷:1
W||FCG
C||FGW
F◁:1
FG◁:1
F◁:1
FG◁:1
FW||CG
FGW||C
FC||GW
FCG||W
FW▷:1
FC▷:1
G||FCW
G||FCW
F◁:1
F◁:1
FG||CW
FG||CW
FG▷:1
FG▷:1
||FCGW
||FCGW

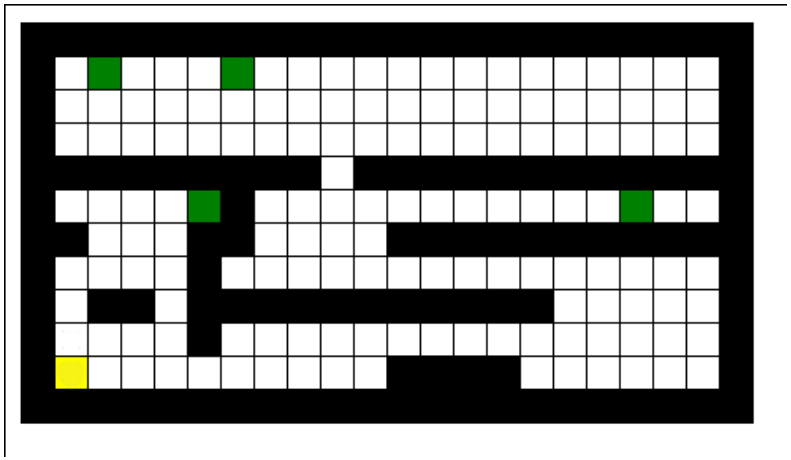
# Drzewa i grafy

- Agent bez pamięci z konieczności operuje drzewem przeszukiwań (bo nie potrafi stwierdzić, że w jakimś stanie już był)
- Najczęściej lepiej modelować świat za pomocą grafu

**Uwaga**

Drzewo pamięta całą ścieżkę dojścia. Rzadko jest to potrzebne, częściej wystarcza jakaś abstrakcja ścieżki: na przykład liczba zużytych biletów vs. pełna historia dojazdu do przystanku.

# Labirynt



- Będziemy teraz rozważać różne labirynty, na kwadratowej siatce.
- Labirynt jako problem wyszukiwania:
  - stan – współrzędne pola na którym można stanąć (nie ściany)
  - start – ustalona pozycja w labiryncie (żółta)
  - cel – ustalone pozycje w labiryncie (zielona)
  - model – 4-sąsiedztwo (modulo ściany), akcje to N, W, E, S.
  - koszt – jednostkowy

# Labirynt 2. Możliwe modyfikacje

**Pytanie**

Jak możemy sobie ten labirynt utrudnić. Jakieś propozycje?

a) Modyfikacja ruchów na planszy:
   - Dodać pary teleportów (dodatkowy ruch w niektórych polach)
   - Wprowadzać różne rodzaje terenu (piaski, bagna, drogi, ...)

b) Bogatszy stan:
   - Dodać drzwi i klucze (czym stanie się stan)?
   - Dodać poruszających się (deterministycznie) wrogów (stan?)
   - Dodać skrzynie z bronią, apteczki i punkty życia (stan?)

# BFS

BFS = Breadth First Search

## Opis

- Mamy 3 grupy stanów: do-zbadania, zbadane i pozostałe.
- Na początku mamy 1 stan do-zbadania: stan startowy
- Stany do zbadania przechowujemy w kolejce FIFO (first-in first out)
- Badanie stanu:
  - Sprawdzenie, czy jest stanem docelowym (jak tak, to **koniec!**)
  - Ustalenie, jakie akcje możemy zrobić w tym stanie, znalezienie nowy stanów do-zbadania

## Skrócony opis

Pobieraj stan z kolejki, przetwarzaj, jak kolejka się skończy (nic do-zbadania) to zakończ działanie, (możesz też zakończyć, jak znajdziesz stan docelowy).

# BFS. Labirynt

- Jako prostą demonstrację, spróbujmy sprawdzić, jak błądzenie losowe poradzi sobie w labiryncie.
- a następnie zapiszemy BFS korzystając z udostępnianych funkcji.

Popatrzmy na programy `search_intro.py` oraz `search_bfs.py` uzupełniając brakujący kod.

# DFS

DFS = Depth First Search

## Opis

- Stany przetwarzamy w innej kolejności: dzieci aktualnie rozwijanego mają priorytet
- Czyli zamiast FIFO używamy LIFO (List in First out), czyli po prostu stosu.
- Oprócz tego algorytm się nie zmienia.

# Analiza czasowo pamięciowa

- W algorytmach na grafach używa się takich parametrów jak $|V|$ oraz $|E|$ (liczba stanów, liczba krawędzi)
- Dobra złożoność to może być $O(|V| + |E|)$
- W sztucznej inteligencji, gdzie często nie znamy grafu (lub jest on zbyt duży, żeby traktować go jako daną do zadania), używamy innych parametrów

# Analiza czasowo pamięciowa (2)

## Parametry zadania wyszukiwania

- $b$ – maksymalne rozgałęzienie (branching factor)
- $d$ – głębokość najpłytszego węzła docelowego
- $m$ – maksymalna długość ścieżki w przestrzeni poszukiwań

## Uwaga

Mówiąc o czasie (pamięci) często używamy jako jednostki liczby węzłów (przetworzonych/pamiętanych).

# Czas i pamięć dla BFS i DFS

## BFS

Czas $= (O(b + b^2 + b^3 + \cdots + O(b^d)) = O(b^d))$
Pamięć = Czas
**Uwaga**: Może być też $O(b^{d+1})$ jak testujemy warunek sukcesu dopiero podczas rozwijania.

## DFS

Czas $= O(b^m)$ – łeeee...
Pamięć $= O(bm)$ – hura!