

Sztuczna Inteligencja

3.2 Szukanie heurystyczne I

Włodzisław Duch
Katedra Informatyki Stosowanej UMK
Google: Włodzisław Duch

Było

- Szukanie jako uniwersalna metoda rozwiązywania problemów.
- Podstawy: grafy i drzewa
- Przykłady: przesuwanka, zagadki logiczne, kryptoartytymetyka
- Znaczenie reprezentacji
- Metody szukania na ślepo
- Szukanie w głąb
- Szukanie wszerek
- Szukanie iteracyjnie pogłębiane
- Szukanie dwukierunkowe
- PATHDEMO

Szukanie heurystyczne

- Metody szukania heurystycznego
- Definicja funkcji heurystycznej
- Najpierw najlepszy
 - Przesuwanka
 - Szukanie zachłanne
 - Szukanie A*
- Algorytmy iteracyjne
 - Wspinaczka
 - Monte Carlo
 - Symulowane wyżarzanie
- Klasyfikacja algorytmów szukania
- Szukanie z więzami

Szukanie heurystyczne



- Wiele rzeczywistych problemów wymaga szukania najlepszych kombinacji uwzględniających różne ograniczenia: połączeń, kosztów podróży, upakowania elementów, sekwencji DNA ...
- Największym problemem w takich zastosowaniach jest eksplozja kombinatoryczna liczby dróg w **możliwych sekwencjach**.
- Ślepe szukanie **nie używa informacji** o możliwej strukturze drzewa lub ocen podobieństwa danego stanu do pożądanego celu.
- Szukanie heurystyczne wykorzystuje informacje, które mogą poprawić efektywność procesu szukania, chociaż nie gwarantują znalezienia rozwiązań optymalnych.
- Taka informacja może przyczynić się do optymalizacji procesu szukania.
- Duże modele językowe dostarczają bardzo dobrych heurystyk w procesach szukania.

Ocena procesów szukania

Jak oceniać algorytmy szukania? Najważniejsze aspekty to:

- ✓ **Zupełność** – czy jest gwarancja dotarcia do celu, jeśli istnieje.
- ✓ **Optymalność** – czy gwarantuje znalezienie optymalnego rozwiązania, czyli najtańszego rozwiązania, najkrótszej drogi.
- ✓ **Złożoność czasowa.**
- ✓ **Złożoność pamięciowa.**

Podstawowe oznaczenia:

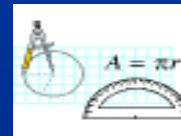
- b – czynnik rozgałęzień (ang. *branching factor*),
- d – minimalna głębokość drzewa, na której jest cel,
- m – maksymalna długość uwzględnianej ścieżki.

Heurystyczne Metody Szukania

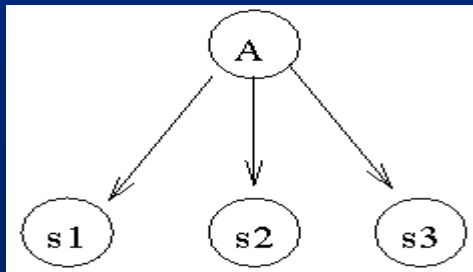


- Używają heurystyk, „reguły kciuka” by określić, która część drzewa decyzji rozwijać najpierw.
 - Heurystyki to reguły lub metody, które **często, choć nie zawsze**, gwarantują podjęcie lepszej decyzji.
 - Np. w sklepie z wieloma kasami dobrą regułą jest:
stać przy kasie z najkrótszą kolejką. Ale uwaga na wyjątki!
- 1) jeśli stoi przy niej osobnik z furą zakupów;
 - 2) lub nie ma przy niej kasjera;
 - 3) lub przyjmują tylko gotówkę, a chcesz na kartę;
-
to nie jest najlepsza decyzja.

Funkcja Heurystyczna



- Funkcja $h : \Psi \rightarrow \mathbb{R}$, gdzie Ψ to zbiór dozwolonych stanów, $\mathbb{R}$ to liczby rzeczywiste, odwzorowuje stany s ze zbioru Ψ na wartości $h(s)$ służące do oceny względnych kosztów lub zysków rozwijania dalszej drogi przez węzeł odpowiadający s .



Węzeł A ma 3 potomków.

$$h(s1)=0.8, h(s2)=2.0, h(s3)=1.6$$

Wartości = koszty utworzenia węzła;
najtaniej jest utworzyć węzeł s1 i ten
z punktu widzenia danej heurystyki
jest najlepszym kandydatem.

Przykładowy graf

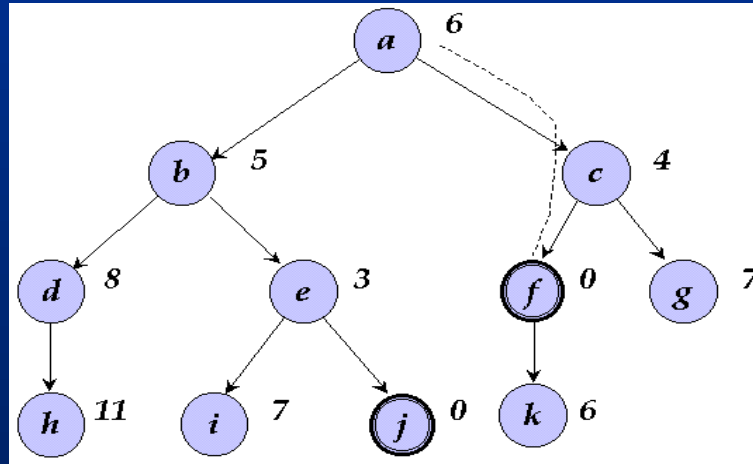
[Przykłady z Javapoint AI.](#)

Najpierw najlepszy (BestFS)



- BestFS to klasa metod łącząca szukanie w głąb (**DFS**) i szukanie wszerz (**BFS**).
- DFS może znaleźć dobre rozwiązania taniej niż BFS, ale BFS nie wpada w zamknięte pętle ani w głębokie ślepe zaułki.
- Metoda „najpierw najlepszy” (BestFS) pozwala połączyć korzyści z obu metod.
- Jest kilka wariantów tej metody, posługujących się heurystycznymi miarami oceny odległości od celu.
- Najprostszym wariantem jest metoda zachłanna, w której rozwijany jest tylko pierwszy najlepszy węzeł: marsz prosto do celu z pomijaniem innych możliwości.

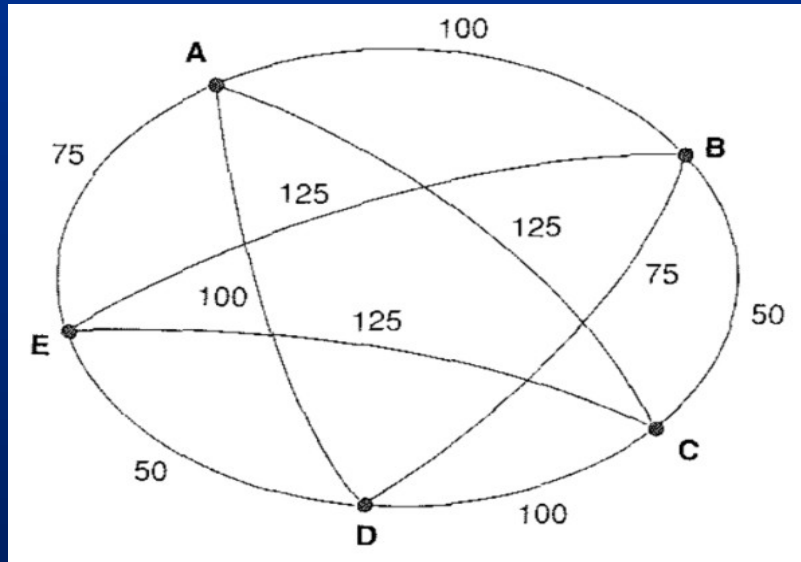
Szukanie zachłanne - ilustracja



Jeśli szukamy drogi z miejsca *a* do miejsca *j* mając tylko informacje o najbliższych miejscach to BFS odwiedzi *c*, *f* i *k*, cofnie się do *c*, odwiedzi *g*, cofnie się do *a*, odwiedzi *b*, *e*, *j*.

Potrzebna jest lepsza funkcja heurystyczna – np. ocena odległości od celu.

Problem wędrującego sprzedawcy



To „klasyczny problem”, którego złożoność rośnie jak $N!$ Dla N miejsc. Zaczynamy od A, trzeba obejść wszystkie miasta. Mamy $4!=24$ możliwości. Jaka jest najkrótsza droga? Tu odległość od celu jest nieznana. Możemy zastosować BFS bez cofania. Jaką drogę wybierze heurystyka zachłanna?

BFS1: szukanie zachłanne



- Szukanie zachłanne (Greedy Search, GS), jedna z najprostszych strategii BestFS.
- Funkcja heurystyczna $h(n)$ – ocenia pozostałe koszty dotarcia do celu, np. odległość od celu. Wówczas:
- GS = *minimalizacja ocenianych kosztów dotarcia do celu*.
- Generowane są wszystkie węzły i rozwijany węzeł najbliższy celu.
- Idziemy w głąb ile się da. Lepszy wariant:
jeśli rosną koszty wracamy i wybieramy kolejny najbliższy węzeł.
- W problemach szukania drogi możliwe są różne metryki:
(1) Najkrótsza odległość Euklidesowa między punktami;
(2) Odległość Manhattan, czyli poruszanie się tylko po prostych poziomych i pionowych (dozwolonych drogach).
- W innych problemach mogą to być oceny podobieństwa.

Algorytm zachłanny



Dane: start S , cel G . Operatory uporządkowane O_i .

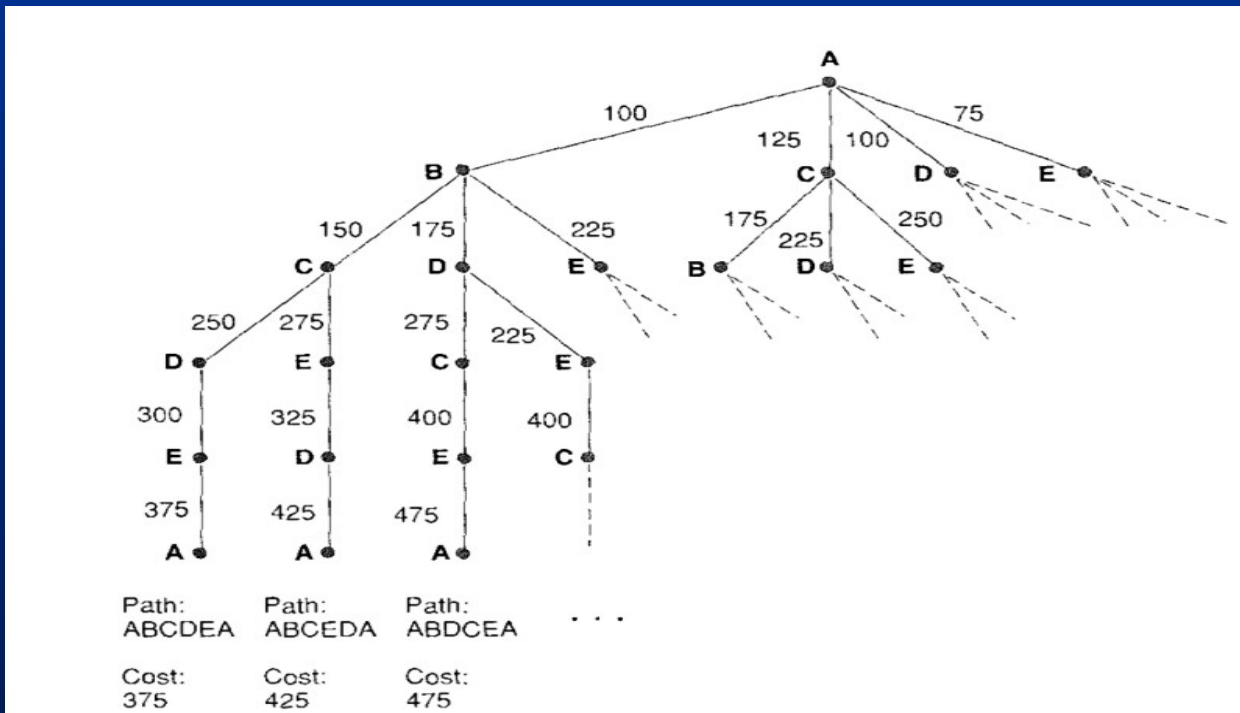
1. Inicjalizacja: bieżący węzeł $C=S$; koszt całkowity $F=0$; poziom $k=1$; droga $V(k)=C$.
2. Powtarzaj 3-5 jeśli $k < m+1$ (max głębokość)
3. Zastosuj wszystkie operatory O_i do C , oceń koszty $f_c(O_i)$ węzłów.
4. Jeśli nie można zastosować żadnego operatora zwróć węzeł C i k -wymiarowy wektor rozwiązań $V(\cdot)$; stop.
5. Przejdź do węzła $C=\arg \min f(O_i)$; $k=k+1$; $V(k)=C$

Problemy: kilka węzłów ma ten sam koszt. Impas - wybieramy pierwszy, albo przypadkowy.

Bez cofania algorytm może trafić w ślepy zaułek.

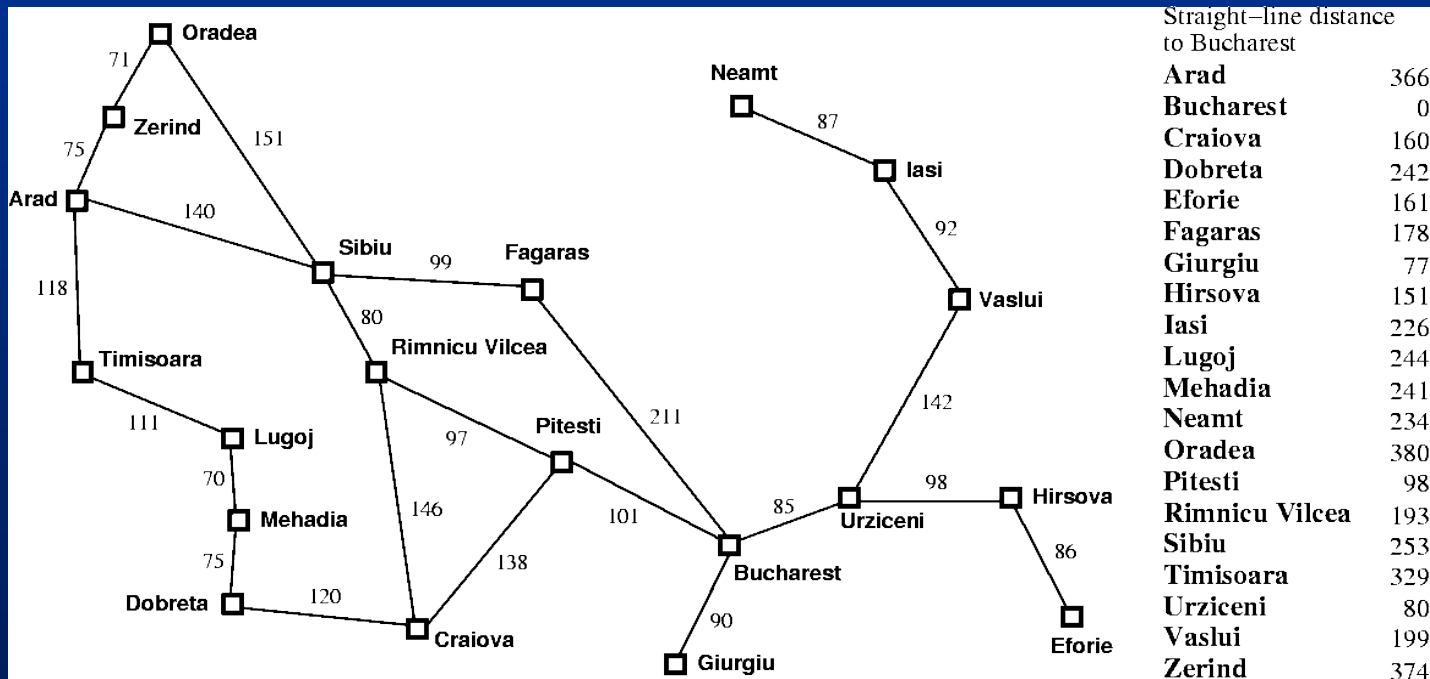
Złożoność pamięciowa i czasowa **$O(bd)$** ;
metoda nie jest ani optymalna ani zupełna, ale jest tania.

Drzewo DFS dla wędrującego sprzedawcy



Zakładając $N+1$ miejsc i powrót do początku jest tu $N!$ dróg dla wszystkich permutacji. Np. plany odwiedzin 21 miast i powrót do domu to $20! = 2.4 \times 10^{18}$ czyli 2.4 mld mld możliwości.

BestFS1: przykład GS z szukaniem trasy

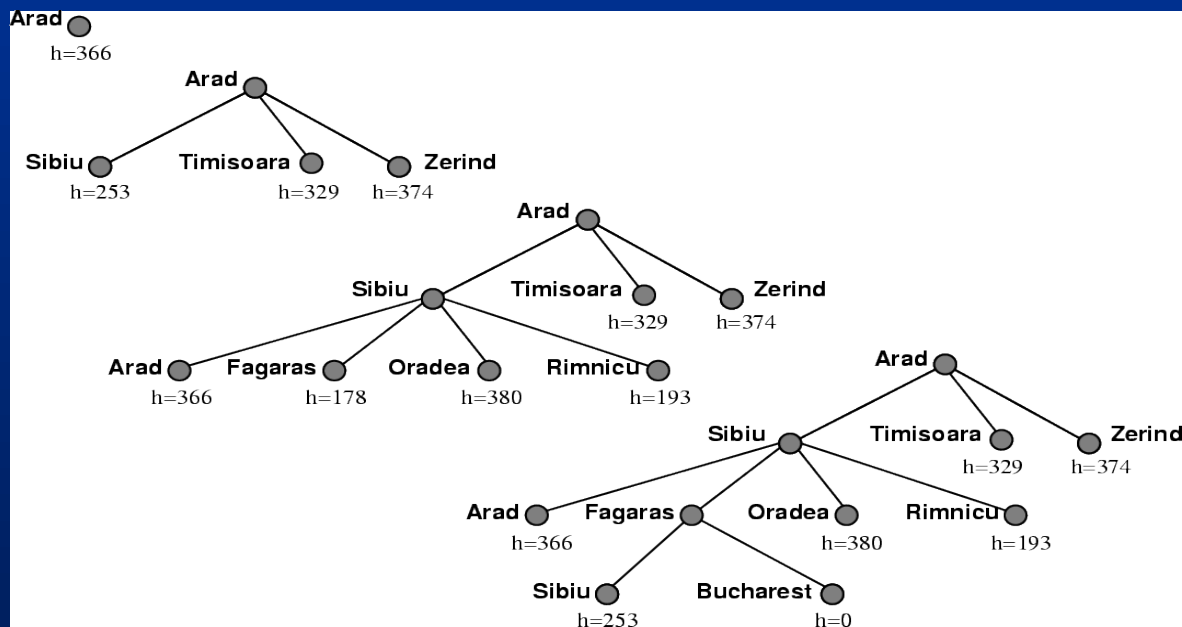


Odległości od Bukaresztu miast na mapie Rumunii;

$h_{SLD}(n)$ = odległości w linii powietrznej do miasta n .

Przykład z Russell/Norvig, Sztuczna inteligencja. Nowe spojrzenie, 2023.

BestFS1: szukanie trasy - graf



Szukanie zachłanne najkrótszej drogi do Bukaresztu.

Wartość funkcji heurystycznej $h(n)$ = odległości mierzonej w linii prostej od Bukaresztu, podanej obok węzłów grafu.

Znaleziona droga $A \rightarrow S \rightarrow F \rightarrow B$. Czy jest optymalna?

BestFS zachłanne

- Znalezione rozwiązanie $A \rightarrow S \rightarrow F \rightarrow B$ **nie jest optymalne**, jest o 32 km dłuższe niż optymalna droga $A \rightarrow S \rightarrow R \rightarrow P \rightarrow B$.
- Strategia zachłanna GS maksymalnie zmniejsza różnicę redukując koszt następnego kroku dotarcia do celu, ale bez oceny czy znalezione rozwiązanie jest „globalnie” optymalne.
- Chciwość to jeden z 7 grzechów głównych, ale **zwykle się opłaca**, chociaż nie zawsze rozwiązanie jest najlepsze.
- Szukanie zachłanne może utknąć w ślepej uliczce.
Np. jadąc z *Iasi* do *Fagaras* zacznie od *Neamt*, gdzie droga się od razu kończy; algorytm Best FS musi uwzględnić cofanie się (backtracing) jeśli nie dotarł do celu.

Zadanie: zmodyfikuj algorytm zachłanny dodając cofanie.
Jak komplikuje to obliczenia?

BestFS-backtracing

- Szukanie Best FS przypomina **DFS** rozwijając tylko jedną ścieżkę, wycofując się, kiedy trafi się na ślepy zaułek.
- Algorytm ma te same problemy co DFS – nie jest to algorytm optymalny ani zupełny, ale jest szybki.
- Złożoność w najgorszym razie (wiele ślepych dróg) wynosi $O(b^m)$, dla m kroków w głąb i średnio b możliwości, ale w praktyce bywa bliższa $O(m)$.
- Dobra funkcja heurystyczna powinna zredukować znacznie złożoność procesu szukania, ale zależy to od konkretnego problemu i od samej funkcji heurystycznej, nie można więc podać dokładniejszych oszacowań.

UCS - stałe koszty



Uniform Cost Search (UCS): szukanie przy stałych kosztach, ulepszona wersja algorytmu „najpierw najlepszy” BestFS.

1. Oceń koszty, uporządkuj kolejkę według kosztów.
2. Przejdź do węzła o najniższym koszcie.
3. Jeśli jest to stan docelowy G zakończ.
4. Jeśli koszt przekroczy zadany próg cofnij się o jeden poziom i wybierz kolejny węzeł z kolejki.

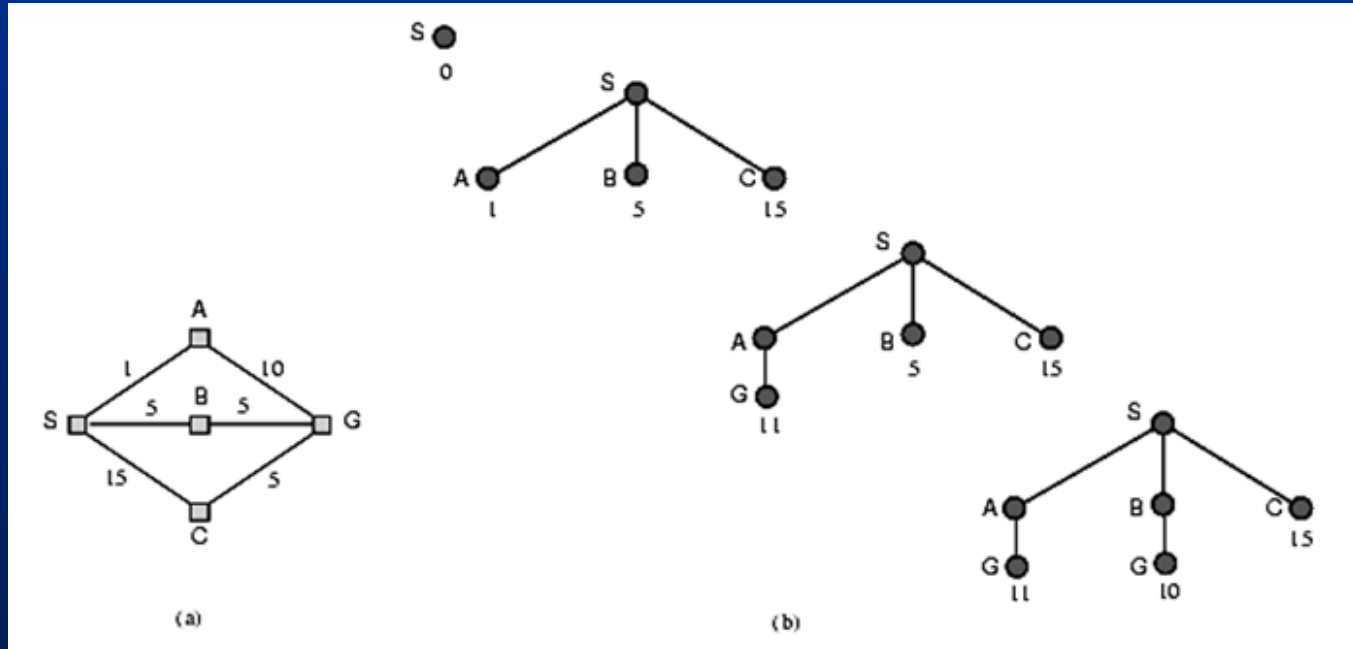
Jeśli koszt wszystkich węzłów jest jednakowy to UCS jest równoważne zwykłemu szukaniu w głąb.

Złożoność: C^* to długość najkrótszej ścieżki do celu,
 ϵ to minimalny koszt każdego kroku, b to maksymalna liczba rozgałęzień.
W najgorszym przypadku czas i pamięć jest $O(b^{1+\lceil C^* / \epsilon \rceil})$.

Dowód jest w podręczniku Russela i Norviga.

UCS - przykład

Przechodzimy z S do G. Koszty są po lewej stronie. Niech $F_{\max}=10$



Węzeł C nie wymaga sprawdzenia, droga do G ma koszt $> F_{\max}$.

Przykłady i warianty [UCS w Java](#).

Programowanie dynamiczne



Ogólna idea (Bellman, 1957):

1. Rozbij rekursywnie problem na coraz prostsze podproblemy.
2. Oceń koszty i rozwiąż najprostsze problemy.
3. Zapisz koszty rozwiązania w tablicy.
4. Poszukaj drogi w tablicy minimalizującej koszty rozwiązań całego problemu, złoż końcowe rozwiązanie z podproblemów zgodnie z minimalnymi kosztami.

Wielokrotne wykorzystanie powtarzających się prostych rozwiązań pozwala czasami zredukować złożoność eksponencjalną na wielomianową.

Intro to [Dynamic Programming](#).

PD-proste przykłady



Bardzo przydatna technika, szczególnie w bioinformatyce, np. szukania współliniowości ciągu aminokwasów w białkach lub par zasad DNA.

Przykłady: ciąg Fibonacciego, oblicz $F(n)$ dla dużego n . [Animacja](#).

$$F(n) = F(n-1) + F(n-2); F(0) = 0; F(1) = 1$$

Jeśli robimy to rekursywnie za każdym razem powtarzamy n kroków, jaka jest wtedy złożoność? $O(2^n)$. Lepiej część zapamiętać.

Tabela: $F(i)=[1,1,2,3,5,8,13 \dots]$

Współczynniki dwumienne: oblicz $C(n,k)$

$$C(n,k) = C(n-1,k) + C(n-1,k-1)$$

for $n > k > 0$; $C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

Potrzebna tabela 2D dla $C(n,k)$ – trójkąt Pascala.

Częstkowe obliczenia drastycznie redukują złożoność problemu.

			1			
		1		1		
	1		2		1	
	1	3		3		1
1	4	6		4		1
1	5	10		10	5	1

Idea PD dla szukania



Zasada „programowania dynamicznego” dla algorytmów szukania:

- Jeśli najlepsza droga do celu G przechodzi przez pośredni węzeł P, to najlepsza droga od startu S do P połączona z najlepszą drogą z P do G daje optymalne rozwiązanie.

Wniosek: szukając najlepszej drogi do celu wystarczy rozpatrywać tylko najkrótszą drogę do P (ale trzeba znać P).

Trzeba więc znać długość różnych fragmentów – stąd hasło DP: „kto nie pamięta przeszłości, musi ją odtworzyć”.

Przykład: wędrówki po Australii ze strony „shortest path place”:

<http://www.ifors.ms.unimelb.edu.au/tutorial/path/>

Przykłady zastosowań i tutorial [Programowania Dynamicznego](#) z freecodecamp.org oraz [VisualGo Dynamic Programming](#) - problem komiwojażera i jego złożoność metodą PD.

Algorytm Floyda-Warshalla



Szukaj najbliższej drogi pomiędzy dowolną parą węzłów w grafie.

Oblicz tablicę $D(v_i, v_j)$ minimalnych odległości (kosztów) pomiędzy węzłami mając Graf $G = (V; E; d)$.

Dla każdej pary połączonych węzłów oblicz $d(v_1, v_2)$;
jeśli nie są połączone to wstaw $\text{inf } (\infty)$.

Algorytm:

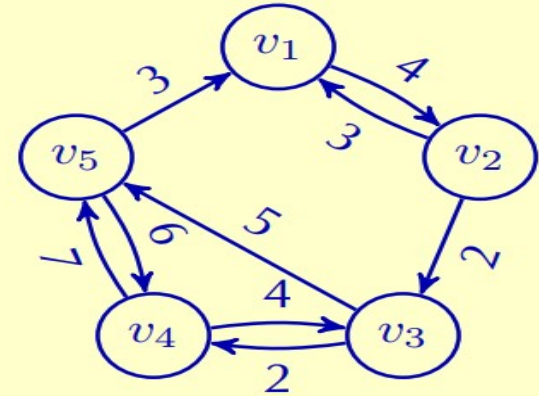
```
for (int k = 1; k<=n; k++)  
  for (int i = 1; i<=n; i++)  
    for (int j = 1; j<=n; j++)  
      D[i,j] := min (D[i,j], D[i,k] + D[k,j]);  
return D;
```

Złożoność czasowa $\sim O(n^3)$ dla n węzłów grafu.

Ilustracja algorytmu z Programiz

Przykład PD metodą Floyda

W k -tym kroku wartość $D[i;j]$ to długość najkrótszej drogi z v_i do v_j o wierzchołkach pośrednich ze zbioru $\{v_1 \dots v_k\}$.



D^0	1	2	3	4	5
1	0	4	∞	∞	∞
2	3	0	2	∞	∞
3	∞	∞	0	2	5
4	∞	∞	4	0	7
5	3	∞	∞	6	0

D^1	1	2	3	4	5
1	0	4	∞	∞	∞
2	3	0	2	∞	∞
3	∞	∞	0	2	5
4	∞	∞	4	0	7
5	3	7	∞	6	0

D^2	1	2	3	4	5
1	0	4	6	∞	∞
2	3	0	2	∞	∞
3	∞	∞	0	2	5
4	∞	∞	4	0	7
5	3	7	9	6	0

D^3	1	2	3	4	5
1	0	4	6	8	11
2	3	0	2	4	7
3	∞	∞	0	2	5
4	∞	∞	4	0	7
5	3	7	9	6	0

D^4	1	2	3	4	5
1	0	4	6	8	11
2	3	0	2	4	7
3	∞	∞	0	2	5
4	∞	∞	4	0	7
5	3	7	9	6	0

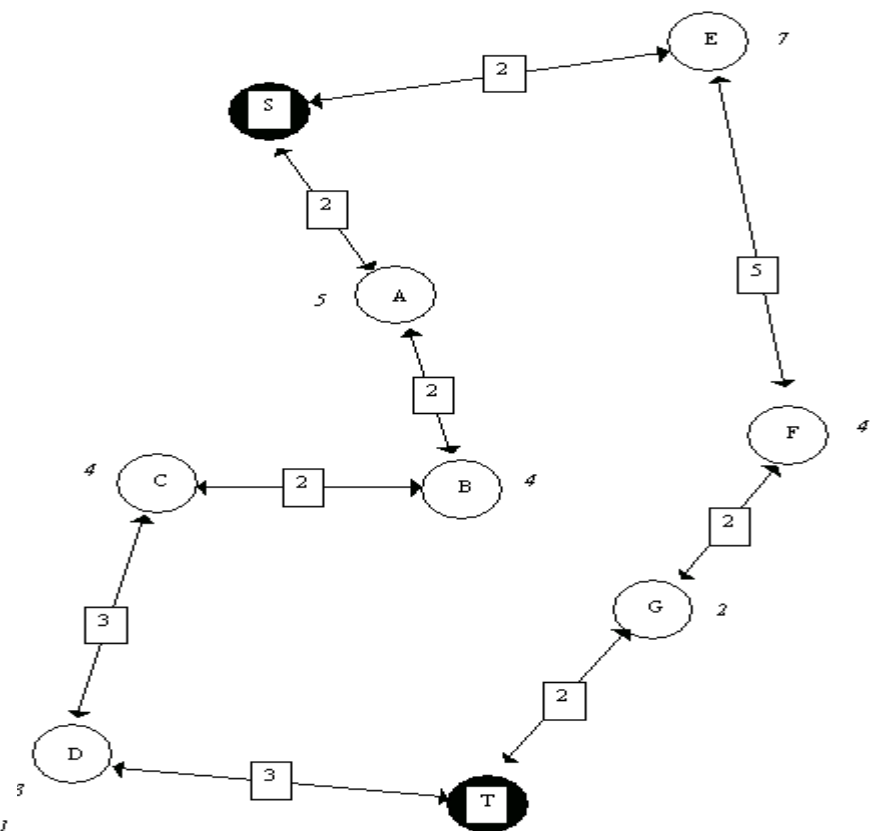
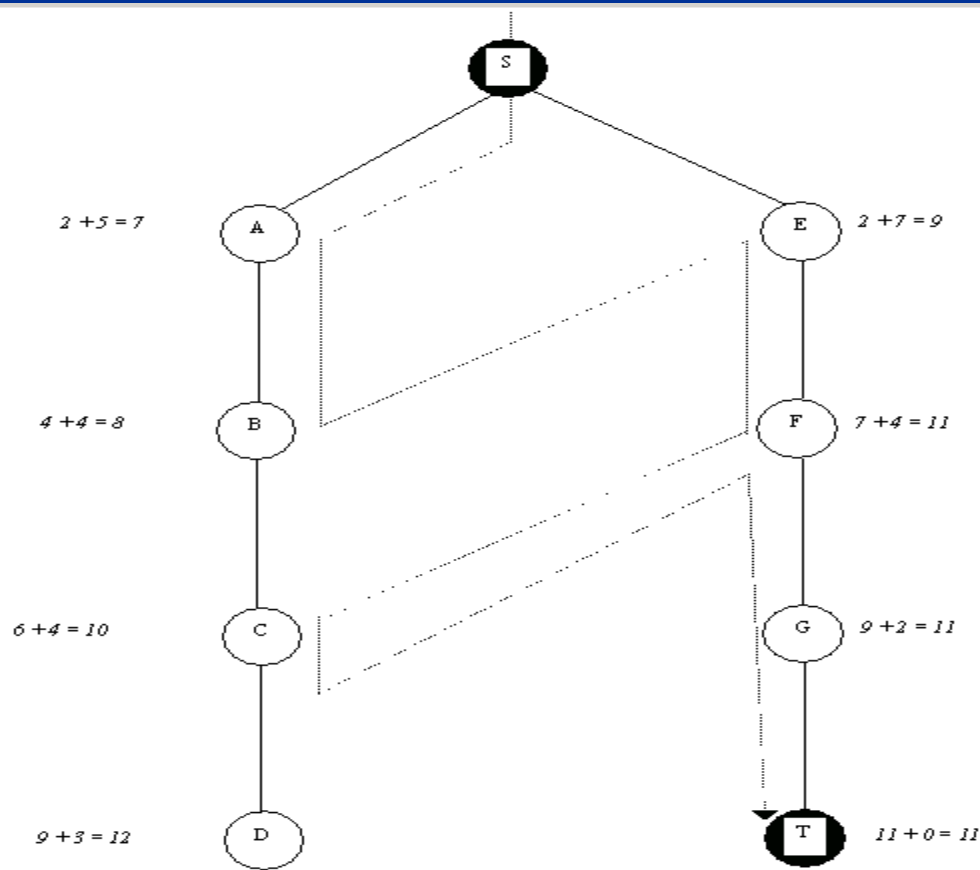
D^5	1	2	3	4	5
1	0	4	6	8	11
2	3	0	2	4	7
3	8	12	0	2	5
4	10	14	4	0	7
5	3	7	9	6	0

BestFS2: szukanie A^*



- BestFS1 minimalizuje tylko koszty dojścia do celu $h(n)$.
- **Alternatywą** jest minimalizacja kosztów dojścia do każdego wężła $g(n)$ jak w PD – jest to metoda kompletna, optymalna, ale dość kosztowna $\sim O(n^3)$.
- Metoda A^* łączy obydwie funkcje heurystyczne, $h(n)$ oraz $g(n)$ w jednej funkcji oceniającej koszty najtańszego rozwiązania przechodzącego przez węzeł n , tzn.
 $f(n) = g(n) + h(n)$.
- Jeśli $f(n) \leq$ rzeczywistych kosztów A^* jest optymalne!

Ilustracja A^* w [Javatpoint](#) i w [PathFinding](#).



BestFS2: A* cd.



Własności:

- Ponieważ wybierana jest najtańsza droga do danego wężła n żadna inna droga nie może obniżyć całkowitego kosztu (**monotoniczność**).
- $h(n)$ powinno być wiarygodną oceną kosztów dojścia do celu – monotoniczne zaniżenie wszystkich kosztów nie przeszkadza.
- Algorytm A* **jest w tym przypadku optymalny**.

Ćwiczenie: udowodnić optymalność A*.

Ocena złożoności zależy od użytych heurystyk do obliczania oszacowania kosztów dotarcia do celu $h(n)$.

W najgorszym przypadku to $O(b^d)$ dla czasu i pamięci, co może być ograniczeniem, stąd wariant z iteracyjnym pogłębianiem.

IDA*, czyli A* iteracyjnie pogłębiane.

Podobny do IDDF

- Stosuj algorytm szukania w głąb.
- Oceniaj całkowite koszty $f(n) = g(n) + h(n)$ heurystyką A*.
- Jeśli $f(n) > T$ cofaj się; T jest tu zmiennym progiem.
- Jeśli nie znaleziono rozwiązania zwiększ T i powtarzaj.

Wady: powtarza część ścieżek, ale i tak końcowe szukanie zajmuje najwięcej czasu.

Zalety: znajduje optymalne rozwiązanie jak A*, ale nie trzyma w pamięci wszystkich węzłów, tylko węzły na rozwijanej ścieżce, więc złożoność pamięciowa jest liniowa $O(db)$.

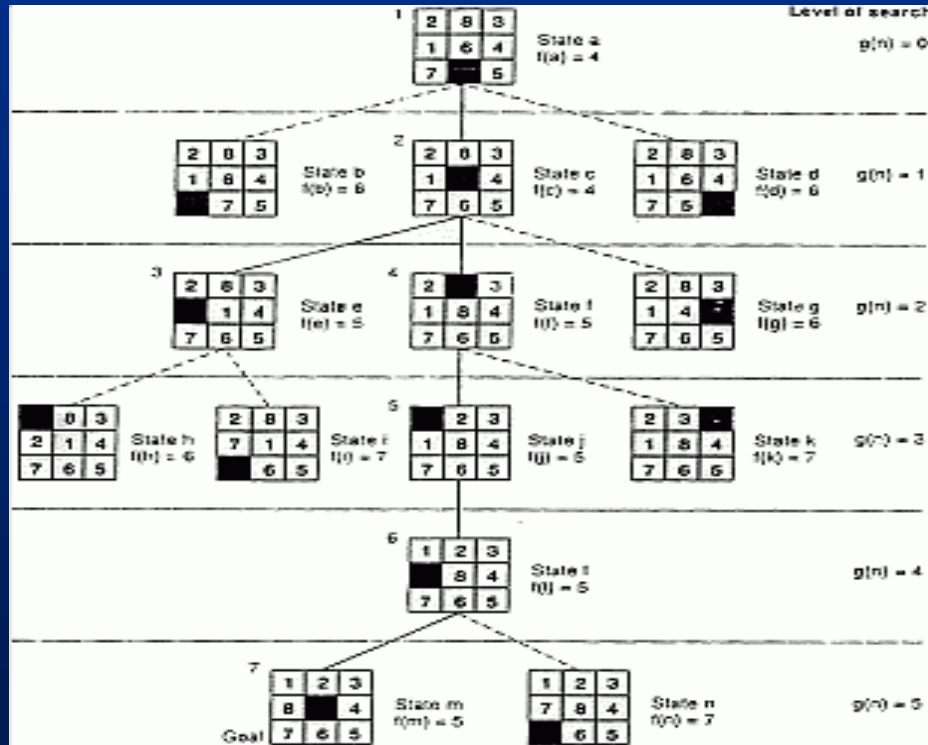
Heurystyki dla 8-ki



- Algorytmy szukania heurystycznego testuje się często na problemie przesuwanki.
- Dla 8-ki jest $9!/2$ or **181.440 możliwych stanów**, dla **15-ki** ok. **653 mld**.
- W procesie szukania dobra funkcja heurystyczna zmniejsza liczbę rozpatrywanych stanów **< 50**.
- Dwie funkcje, które nigdy nie przeceniają kosztów:
 1. h_1 = liczba płytek na złych pozycjach – każdą trzeba przesunąć przynajmniej raz.
 2. h_2 = suma odległości od celu, metryka Manhattan; każdy ruch zmniejsza odległość o 1; tu $h_1=8$, $h_2 = 18$

5	4			1	2	3
6	1	8		8		4
7	3	2		7	6	5

Przykład A* dla 8-ki



Przestrzeń stanów utworzona w czasie heurystycznego szukania 8-ki.

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

$g(n)$ = odległość od startu do stanu n .

$h(n)$ = liczba elementów na złym miejscu.

Animacja algorytmu A* dla labiryntu.

Porównanie efektów heurystyk

<i>d</i>	Search Cost			Effective Branching Factor		
	IDS	A*(h ₁)	A*(h ₂)	IDS	A*(h ₁)	A*(h ₂)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Średnia z 100 symulacji dla 8-ki i dwóch heurystyk.

Ilustracje algorytmów

- Animacje algorytmów: [VisualAlgo](#)
- Animacje algorytmów szukania [na PathFinding](#)
- Ilustracje [data structures and algorithms](#)
- Visualizations of [data structures and algorithms](#)
- [Algorithm-visualizer](#) z przykładami w Java/C++
- [Data Structures & Algorithms](#) (Google)
- [ERIC, Institute for Education Sciences](#) (publikacje)

Jest wiele animacji algorytmów w sieci.