Wprowadzenie do Sztucznej Inteligencji

Wykład 3

Informatyka Studia Inżynierskie

Przeszukiwanie przestrzeni stanów

Przestrzeń stanów jest to czwórka uporządkowana [*N*, *A*, *S*, *GD*], gdzie:

N jest zbiorem wierzchołków odpowiadających *stanom* w procesie rozwiązywania problemu

A jest zbiorem krawędzi, odpowiadających *krokom* w procesie rozwiązywania problemu

S jest niepustym podzbiorem ***N***, zawierającym *stany początkowe* problemu

GD jest niepustym podzbiorem ***N***, zawierającym *stany docelowe* problemu.

Stany ***GD*** są opisane:

- przez podanie *własności stanów* występujących w przeszukiwaniu
- przez podanie *własności ścieżki* tworzonej podczas przeszukiwania

Ścieżką rozwiązania nazywamy ścieżkę wiodącą przez ten graf z wierzchołka należącego do ***S*** do wierzchołka należącego do ***GD***.

Podstawowe problemy teorii przeszukiwania przestrzeni stanów

- Czy metoda gwarantuje *znalezienie rozwiązania*?
- Czy algorytm *zakończy się* w każdym przypadku, czy może wpaść w *pętlę nieskończoną*?
- Czy jeśli rozwiązanie zostanie znalezione, to mamy gwarancję, że będzie to *rozwiązanie optymalne*?
- Jaka jest *czasowa* i *pamięciowa* ("przestrzenna") *złożoność obliczeniowa* procesu przeszukiwania?
- Czy i w jaki sposób można *zredukować złożoność* obliczeniową?

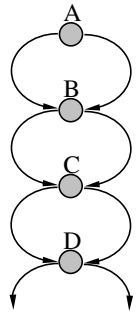
Sformułowanie zadania dla algorytmów przeszukiwania

- Definicja *stanu* przestrzeni
- *Stan początkowy* problemu (ang. initial state)
- Zbiór dopuszczalnych *operatorów/akcji* (ang. operator/action set) lub *funkcja następnika* (ang. successor function)
- Zbiór *stanów docelowych* (ang. goal states) lub *funkcja weryfikacji celu* (ang. goal test)
- *Funkcja kosztu ścieżki* (ang. path cost) - z reguły jest to głębokość przeszukiwania

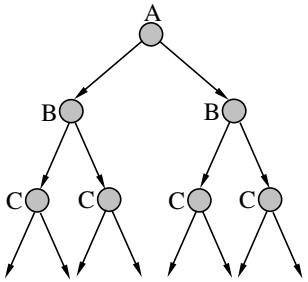
Istotne cechy przestrzeni stanów w reprezentacji grafowej

- Powtarzające się stany (cykle)
- Identyfikacja typu grafu reprezentującego przestrzeń przeszukiwania: drzewa, acykliczne grafy skierowane (DAG)
- Rozmiar przestrzeni stanów (ograniczanie złożoności)

Powtarzające się stany: *efektywność przeszukiwania*



Przestrzeń stanów



Graf przeszukiwania przestrzeni

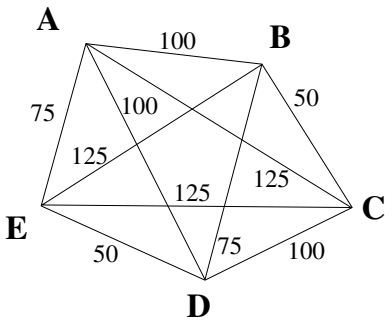
Powtarzające się stany: *charakterystyka*

- Konieczność wykrywania ze względu na efektywność przeszukiwania
- Nieuniknione w niektórych zadaniach (np. z odwracalnymi operatorami) - warunek zatrzymania
- Uwaga! Kompromis między kosztami przeszukiwania a kosztami wykrywania powtórzeń stanów

Powtarzające się stany: *metody przeciwdziałania*

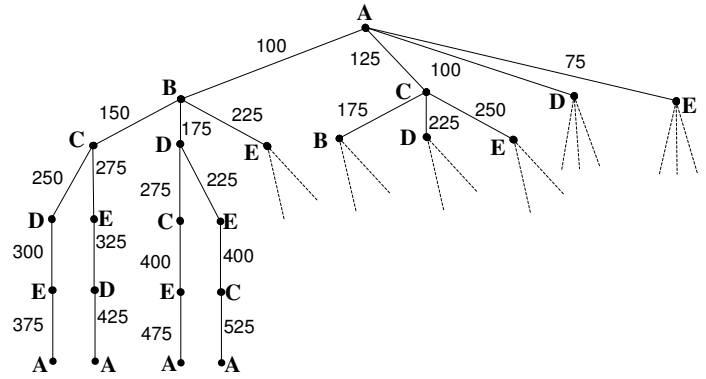
- Zakaz powrotu do bezpośrednio poprzedzającego stanu
- Wykrywanie cykli w ścieżce - zakaz generowania jakiegokolwiek stanu poprzedzającego (pośrednio)
- Wykrywanie dowolnego powtarzającego się stanu w całym grafie przeszukiwania (duże wymagania pamięciowe!)

Przykład przeszukiwania przestrzeni stanów: *problem komiwojażera*



|ACDBEA| = 450

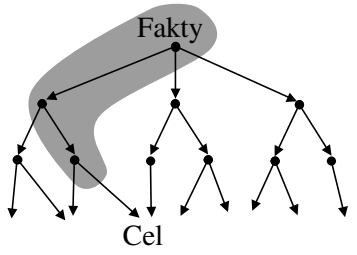
Przykład przeszukiwania przestrzeni stanów: *problem komiwojażera*



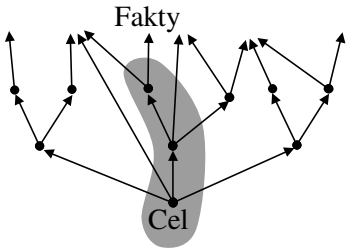
Kierunki przeszukiwania przestrzeni stanów

- Przeszukiwanie w przód (ang. forward chaining)
- Przeszukiwanie w tył (ang. backward chaining)
- Przeszukiwanie dwukierunkowe (ang. bidirectional search)

Przeszukiwanie w przód

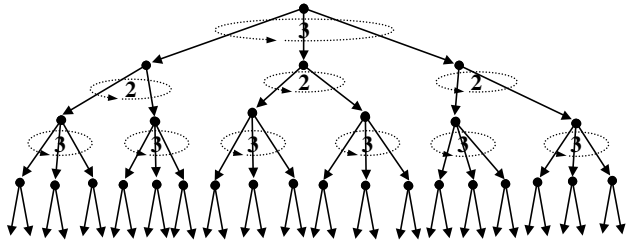


Przeszukiwanie w tył



©AM

Branching factor - co to takiego?



©AM

Kiedy przeszukiwanie w przód?

- Wszystkie lub większość danych zawarta jest w początkowym sformułowaniu problemu - np. interpretacja i analiza dużych zbiorów danych
- Występuje duża liczba potencjalnych celów, ale jest tylko kilka możliwości zastosowania faktów i informacji wejściowych dla konkretnej instancji problemu
- Trudno sformułować hipotezę docelową - np. określanie struktury związków chemicznych

Przeszukiwanie w przód wykorzystuje wiedzę i ograniczenia zawarte w danych i opisie stanu początkowego problemu, aby pokierować przeszukiwaniem zgodnie z zasadami opisanymi przez operatory zmiany stanów.

©AM

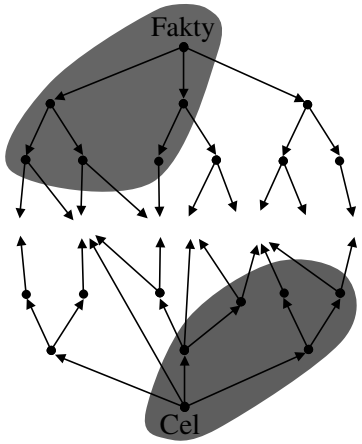
Kiedy przeszukiwanie w tył?

- Cel lub hipoteza jest dana w sformułowaniu problemu albo można ją łatwo sformułować – np. dowodzenie twierdzeń matematycznych, systemy diagnostyczne
- Liczba reguł możliwych do zastosowania rośnie szybko i powoduje, że liczba odwiedzanych stanów jest bardzo duża; wczesna sekcja celów może wyeliminować większość gałęzi tak, że przeszukiwanie będzie bardziej efektywne – np. dowodzenie twierdzeń
- Dane o problemie nie są znane *explicite*, tylko muszą być pozyskane przez rozwiązującego; przeszukiwanie w tył może pomóc ukierunkować proces pozyskiwania danych - np. diagnostyka medyczna

Przeszukiwanie wstecz wykorzystuje informacje o żądanym celu i kieruje procesem przeszukiwania poprzez dobór odpowiednich operatorów zmiany stanów oraz eliminację pewnych gałęzi z przestrzeni stanów.

©AM

Przeszukiwanie dwukierunkowe



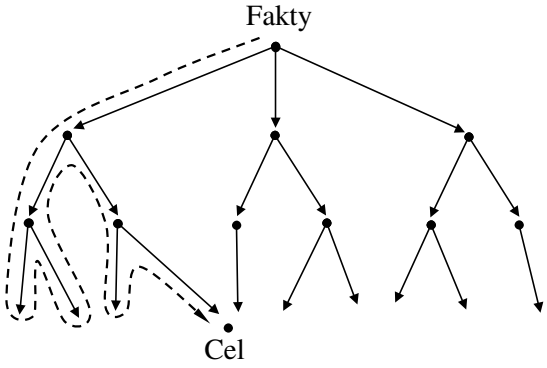
©AM

Przeszukiwanie dwukierunkowe

- Jednoczesne przeszukiwanie w przód i w tył
- Ograniczenie złożoności czasowej
- Problem generowania poprzedników stanu
- Problem wielu stanów docelowych (np. stany poprzedzające mata w szachach)
- Efektywna metoda sprawdzania występowania stanu w grafie przeszukiwania przeciwnego kierunku (problem wyminięcia!)
- Dobór odpowiedniej strategii przeszukiwania w każdym kierunku

©AM

Praktyczna realizacja przeszukiwania - *mechanizm nawrotów*



Nawroty gwarantują realizację przeszukiwania *wszystkich* ścieżek w sposób *systematyczny*.

©AM

Strategie przeszukiwania przestrzeni stanów

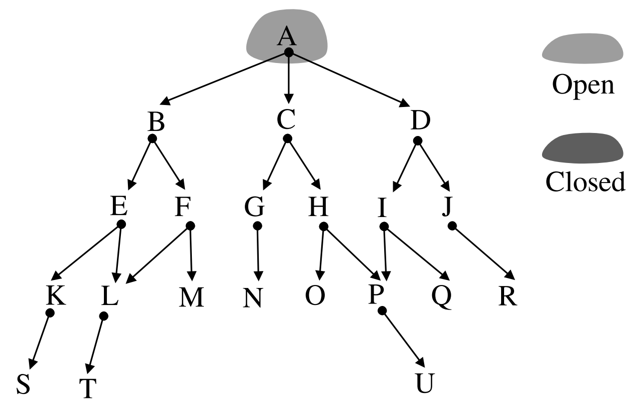
- Przeszukiwanie wszerz (ang. breadth-first search)
- Przeszukiwanie w głąb (ang. depth-first search)
  - Przeszukiwanie w głąb z nawrotami (ang. depth-first search with backtracking)
- Przeszukiwanie z iteracyjnym pogłębianiem (ang. iterative deepening search)
- Przeszukiwanie metodą jednolitego (równomiernego) kosztu (ang. uniform-cost search)

©AM

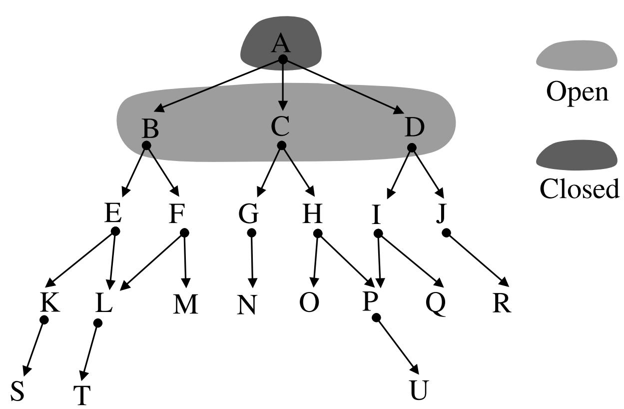
Algorytm przeszukiwania w głąb

```
procedure depth_first_search(initial_state)
begin
    open = [initial_state];
    closed = [];
    while open ≠ [] do
    begin
        remove the leftmost state from open, call it X;
        if X is goal state then return(success);
        generate all children of X;
        put X on closed;
        eliminate any children of X already on either open or closed,
            as this will cause loops in the search;
        put the remaining descendants, in order of discovery,
            on the LEFT end of open;
    end
end.
```

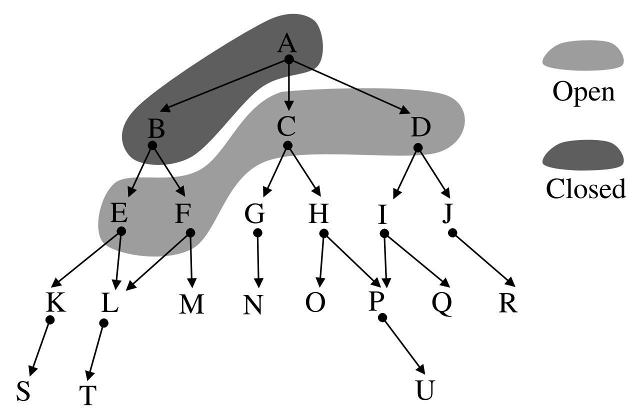

Przeszukiwanie w głąb - przykład



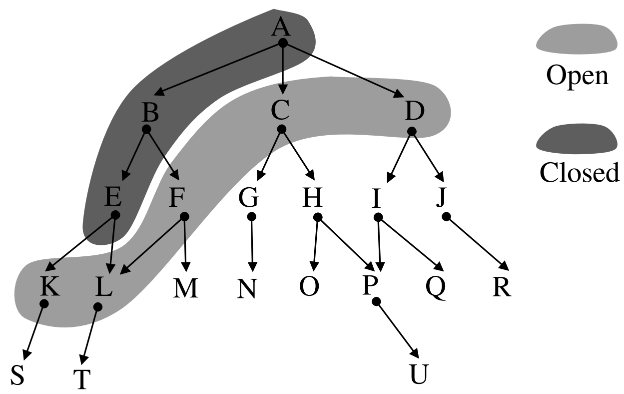
Przeszukiwanie w głąb - przykład



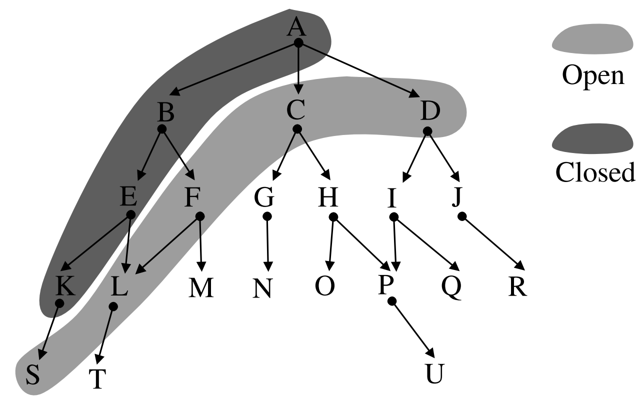
Przeszukiwanie w głąb - przykład



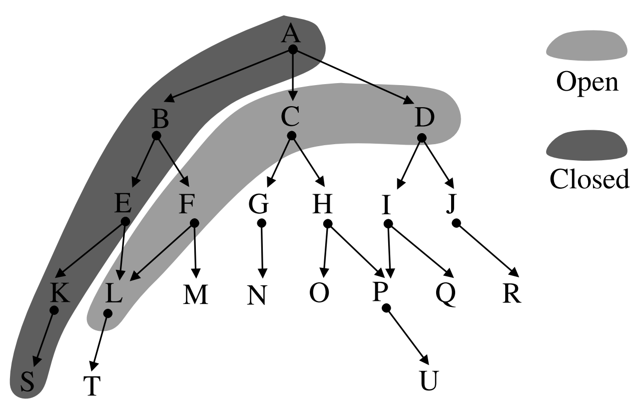
Przeszukiwanie w głąb - przykład



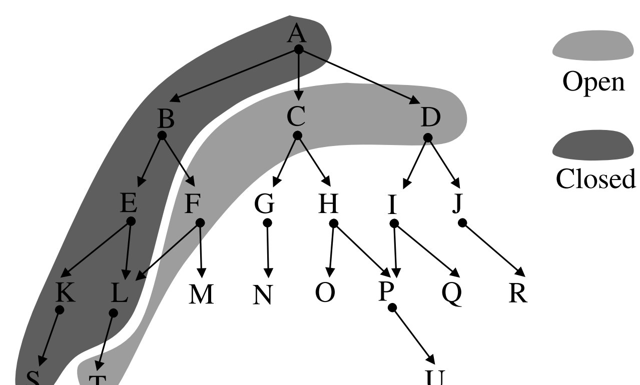
Przeszukiwanie w głąb - przykład



Przeszukiwanie w głąb - przykład



Przeszukiwanie w głąb - przykład

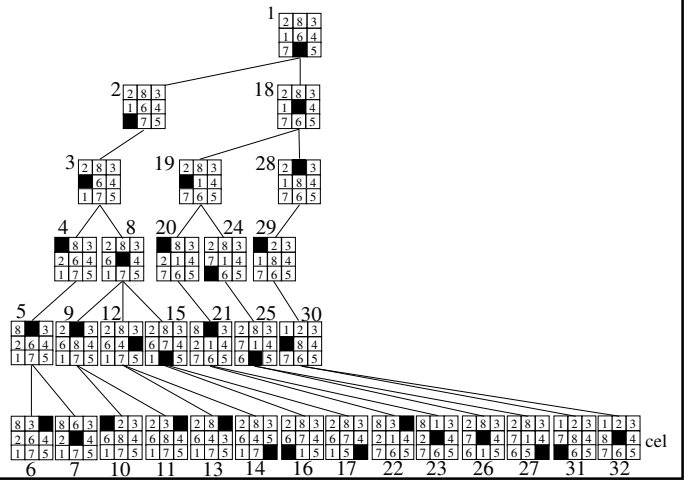


itd.

Algorytm przeszukiwania w głąb (rekurencja)

```
function depth_first_search(current_state)
begin
    if current_state is a goal then return(success)
    else
    begin
        add current_state to closed;
        while current_state has unexamined children do
        begin
            child := next unexamined child of current_state
            if child is not a member of closed then     {check for loops}
                if depth_first_search(child) = success then
                    return(success)
        end
    end;
    return(fail)
end.
```

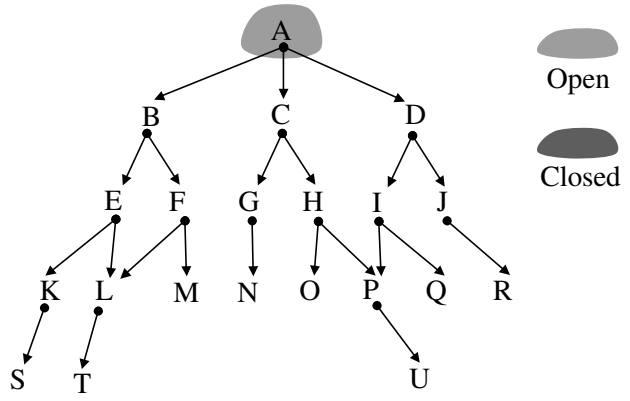

Przeszukiwanie w głąb - przykład



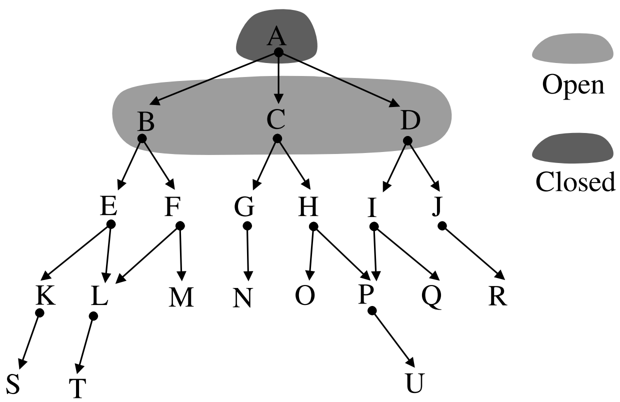
Algorytm przeszukiwania wszerz

```
procedure breadth_first_search(initial_state)
begin
    open = [initial_state];
    closed = [];
    while open ≠ [] do
    begin
        remove the leftmost state from open, call it X;
        if X is goal state then return(success);
        generate all children of X;
        put X on closed;
        eliminate any children of X already on either open or closed,
            as this will cause loops in the search;
        put the remaining descendants, in order of discovery,
            on the RIGHT end of open;
    end
end.
```

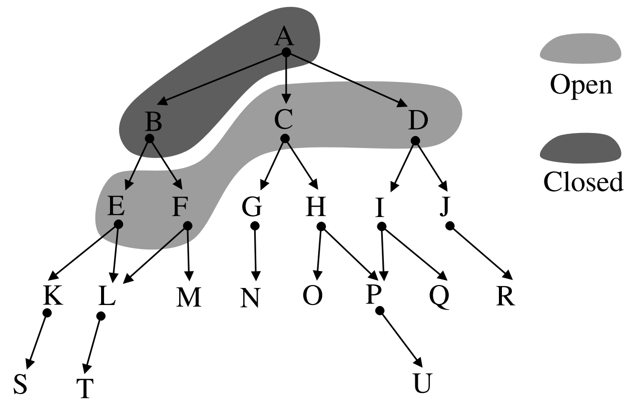

Przeszukiwanie wszerz - przykład



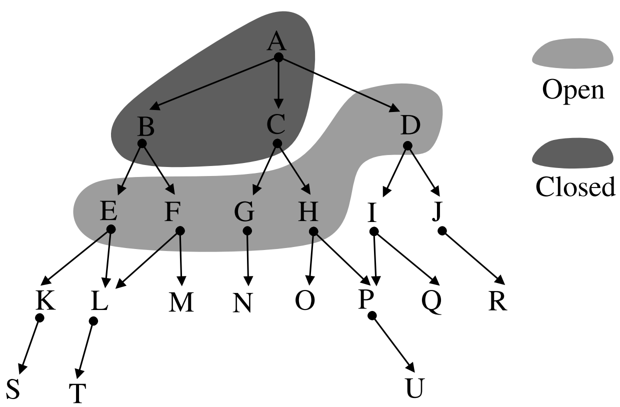
Przeszukiwanie wszerz - przykład



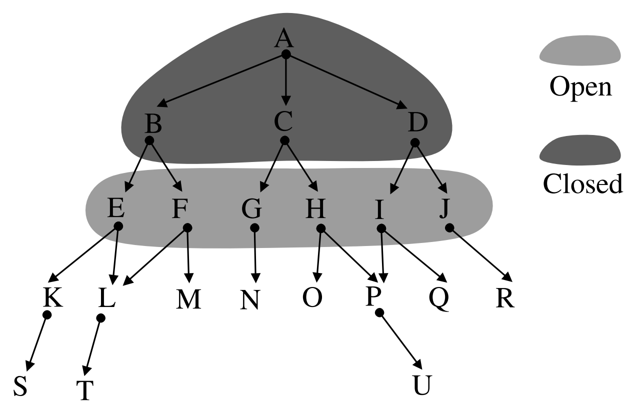
Przeszukiwanie wszerz - przykład



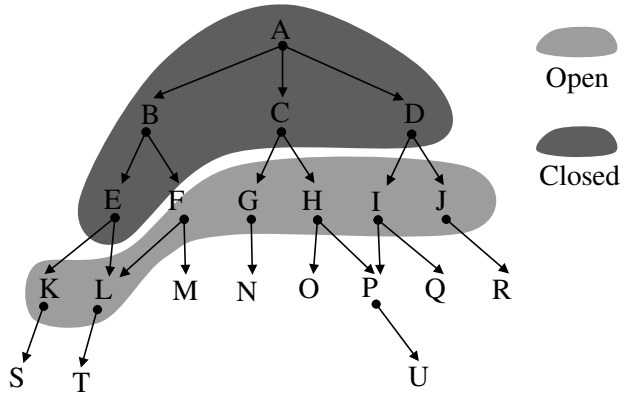
Przeszukiwanie wszerz - przykład



Przeszukiwanie wszerz - przykład

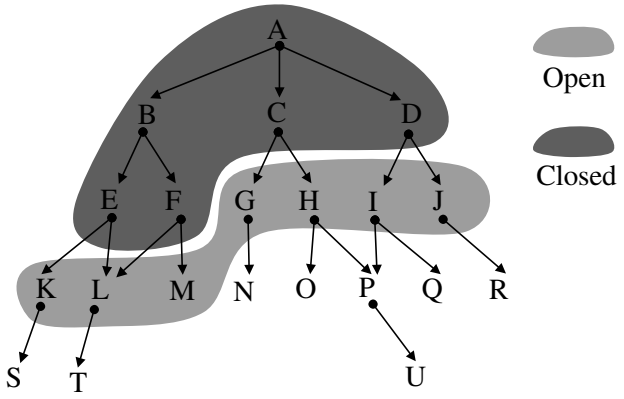


Przeszukiwanie wszerz - przykład



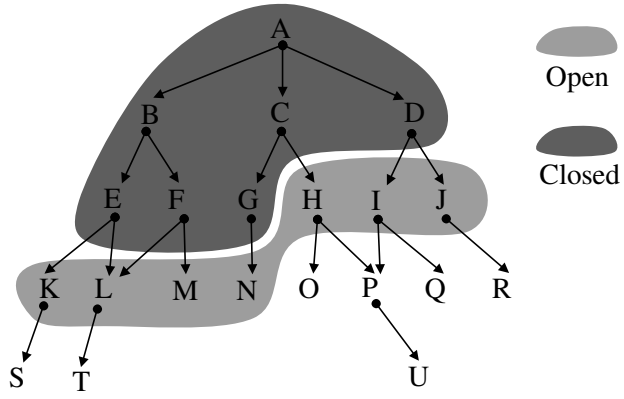
©AM

Przeszukiwanie wszerz - przykład



©AM

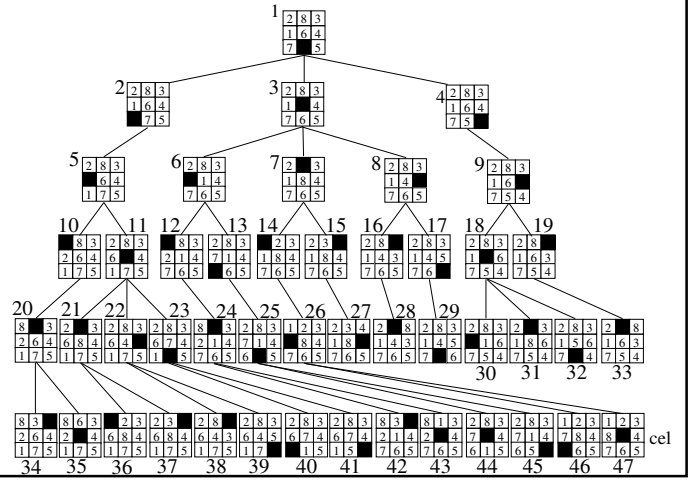
Przeszukiwanie wszerz - przykład



itd.

©AM

Przeszukiwanie wszerz - przykład



©AM

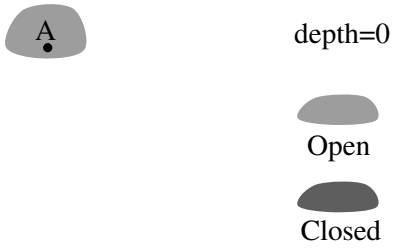
Algorytm przeszukiwania z iteracyjnym pogłębianiem (1)

```
procedure iterative_deepening_search(initial_state)
begin
    for depth ← 0 to ∞ do
    begin
        if depth_limited_search(initial_state, depth) = success then
            return(its result);
    end;
    return(failure);
end.
```

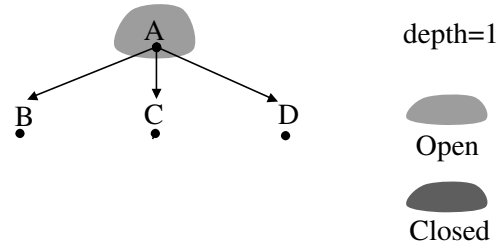

Algorytm przeszukiwania z iteracyjnym pogłębianiem (2)

```
function depth_limited_search(current_state, depth)
begin
    if current_state is a goal then return(success)
    else
    if depth = 0 then return(failure)
    else
    begin
        add current_state to closed;
        while current_state has unexamined children do
        begin
            child := next unexamined child of current_state
            if child is not a member of closed then          {check for loops}
                if depth_limited_search(child, depth-1) = success then
                    return(success)
        end
    end;
    return(failure)
end.
```

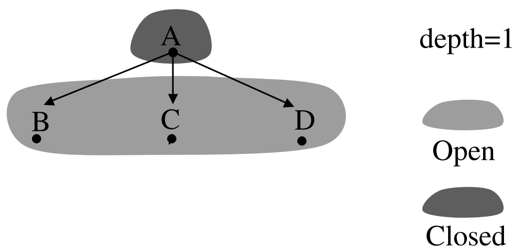

Iteracyjne pogłębianie - przykład



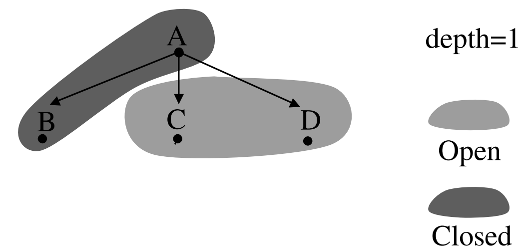
Iteracyjne pogłębianie - przykład



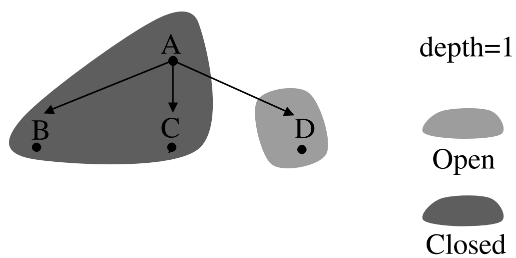
Iteracyjne pogłębianie - przykład



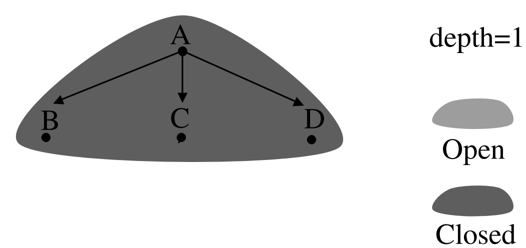
Iteracyjne pogłębianie - przykład



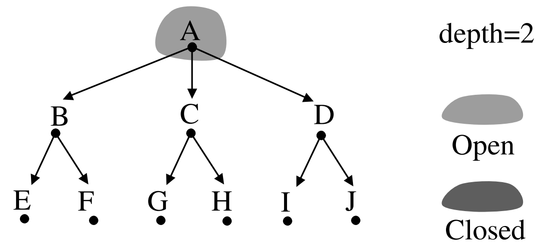
Iteracyjne pogłębianie - przykład



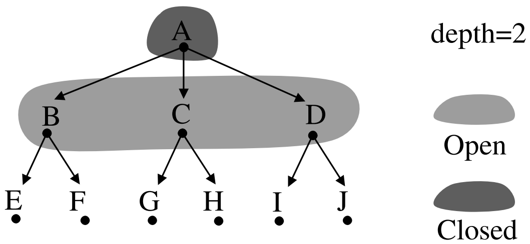
Iteracyjne pogłębianie - przykład



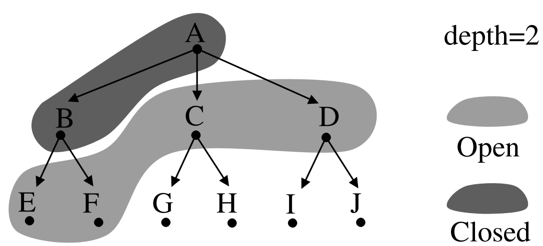
Iteracyjne pogłębianie - przykład



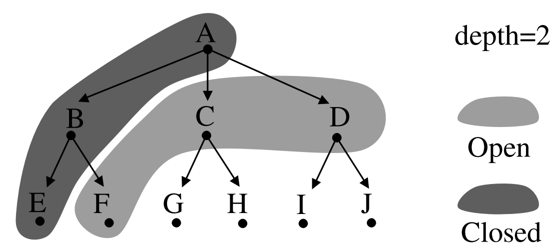
Iteracyjne pogłębianie - przykład



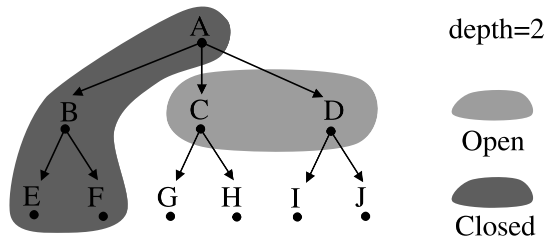
Iteracyjne pogłębianie - przykład



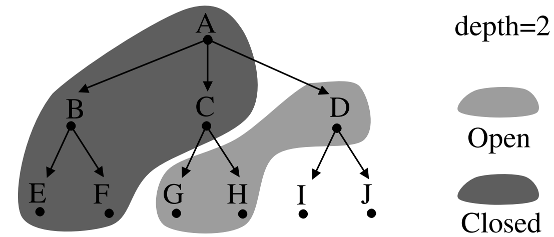
Iteracyjne pogłębianie - przykład



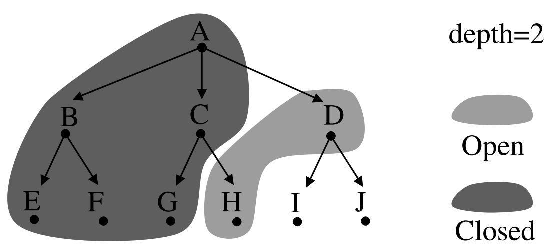
Iteracyjne pogłębianie - przykład



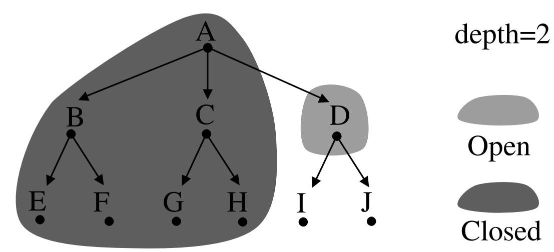
Iteracyjne pogłębianie - przykład



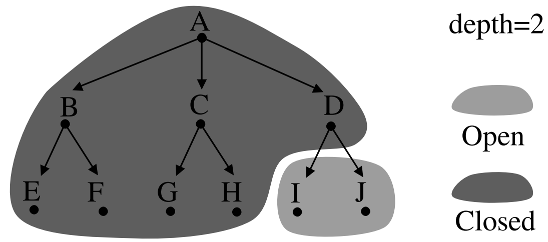
Iteracyjne pogłębianie - przykład



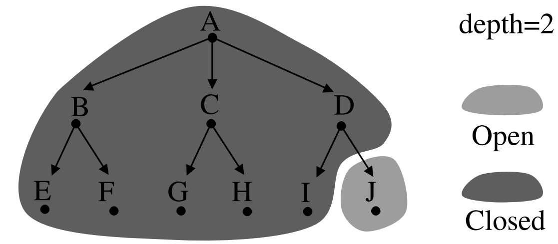
Iteracyjne pogłębianie - przykład



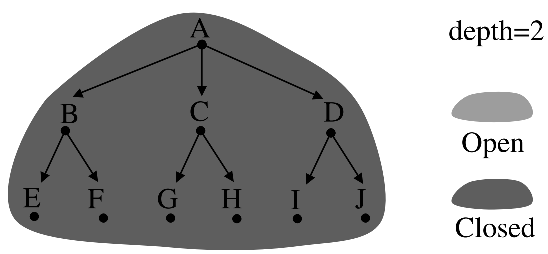
Iteracyjne pogłębianie - przykład



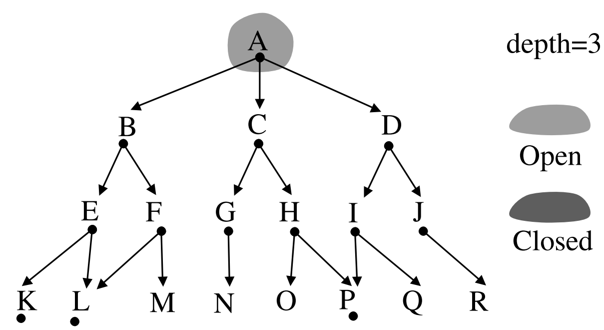
Iteracyjne pogłębianie - przykład



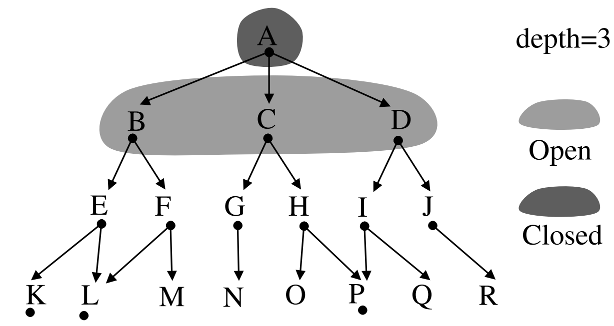
Iteracyjne pogłębianie - przykład



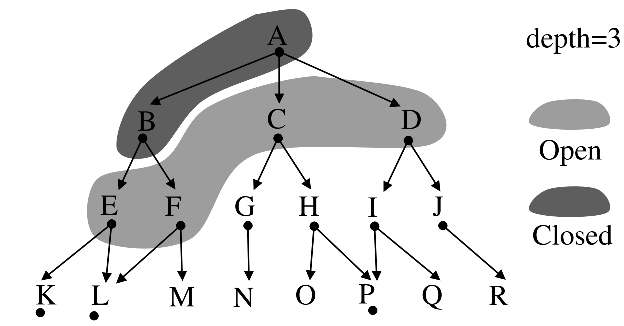
Iteracyjne pogłębianie - przykład



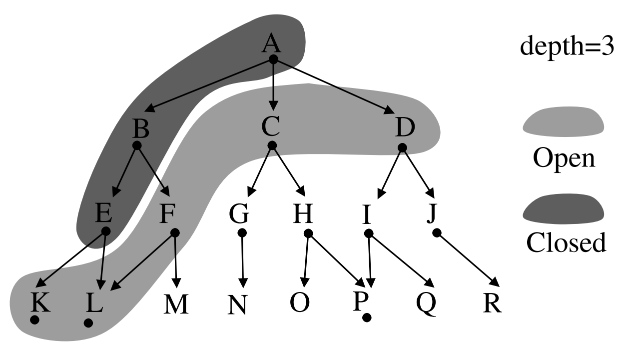
Iteracyjne pogłębianie - przykład



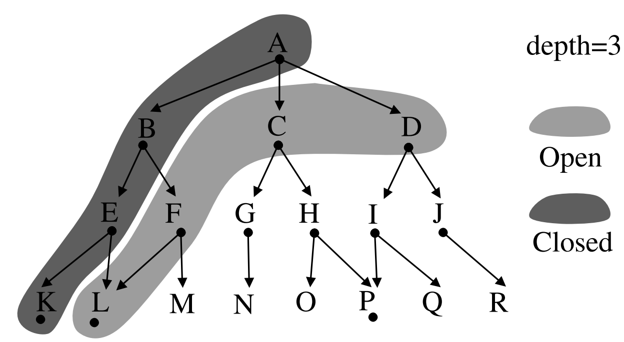
Iteracyjne pogłębianie - przykład



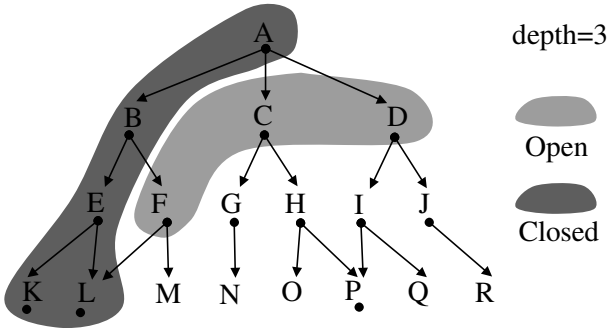
Iteracyjne pogłębianie - przykład



Iteracyjne pogłębianie - przykład

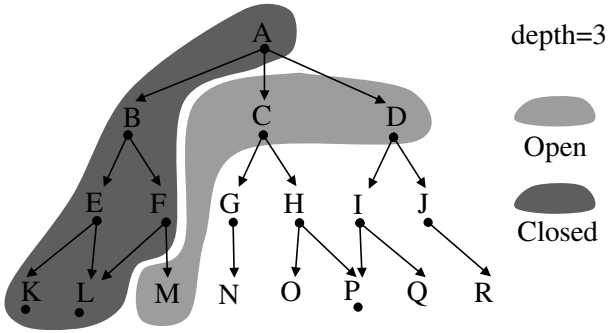


Iteracyjne pogłębianie - przykład



©AM

Iteracyjne pogłębianie - przykład



itd.

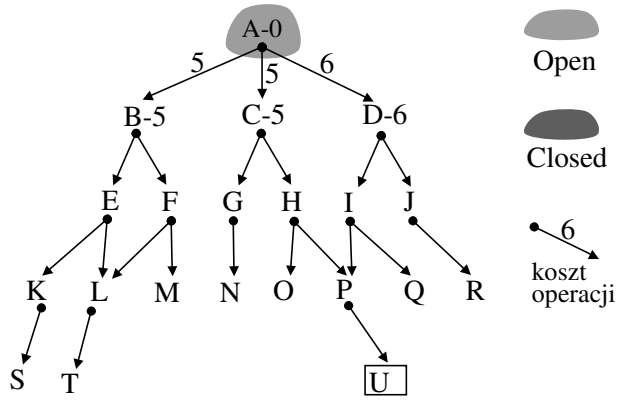
©AM

Algorytm *Uniform-cost search*

```
procedure uniform_cost_search(initial_state)
begin
    open = [initial_state];
    closed = [];
    while open ≠ [] do
    begin
        remove the first state from open, not already on closed, call it X;
        if X is a goal state then return(solution path that led to X);
        generate all children of X;
        put X on closed;
        for each child of X do
            assign path cost to the child state;
            add the child state to the open;
        end;
        re-order states on open according to path cost value (lower values first)
    end
end.
```

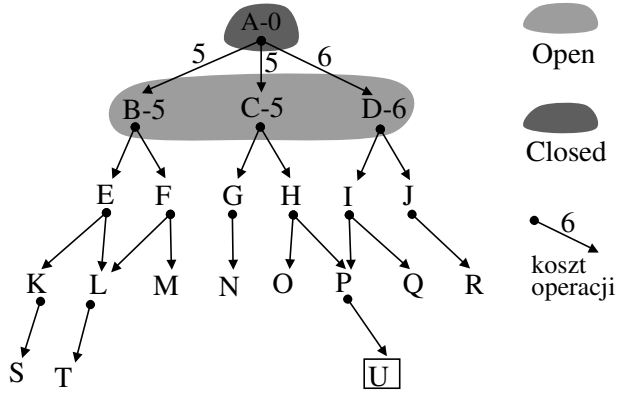
©AM

Algorytm UCS - przykład(1)

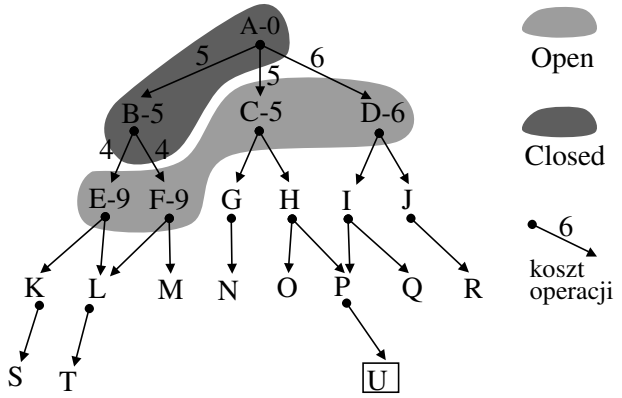


©AM

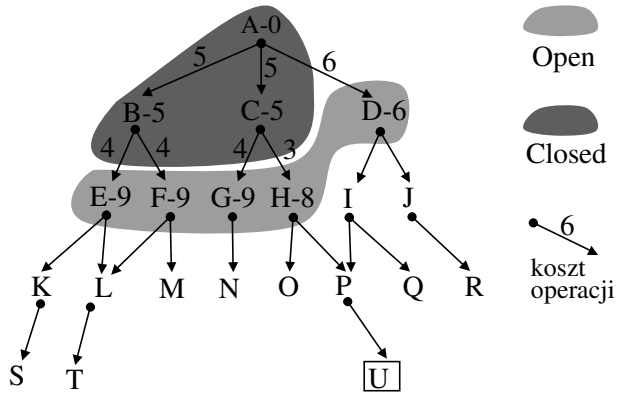
Algorytm UCS - przykład(2)



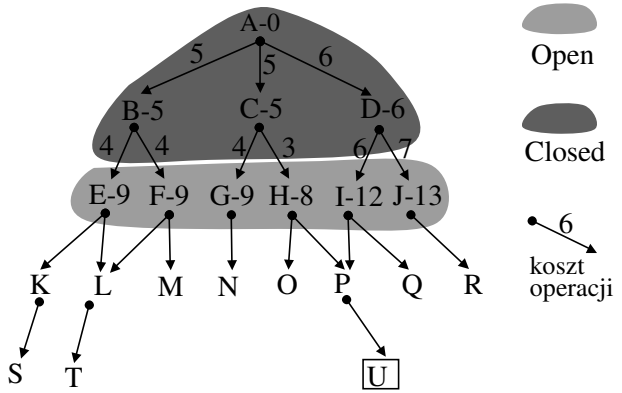
Algorytm UCS - przykład(3)



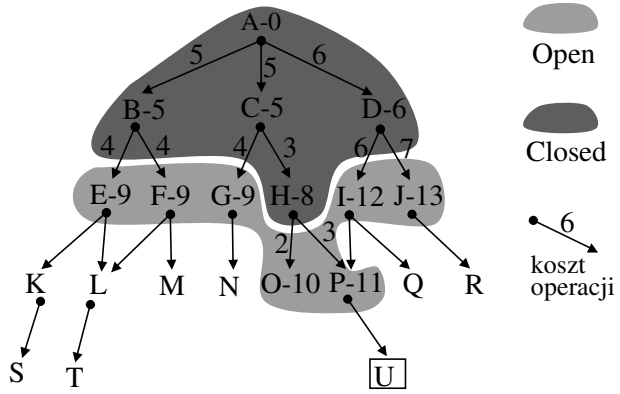
Algorytm UCS - przykład(4)



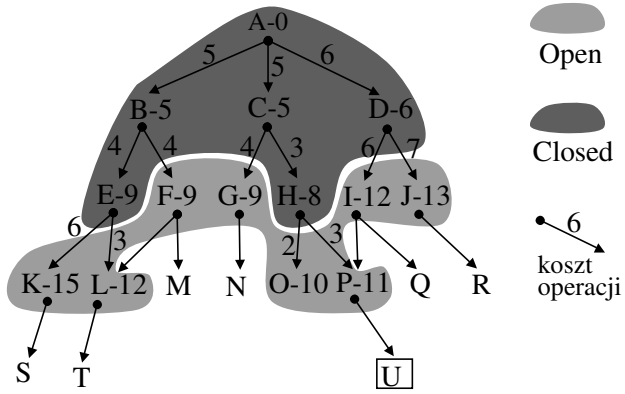
Algorytm UCS - przykład(5)



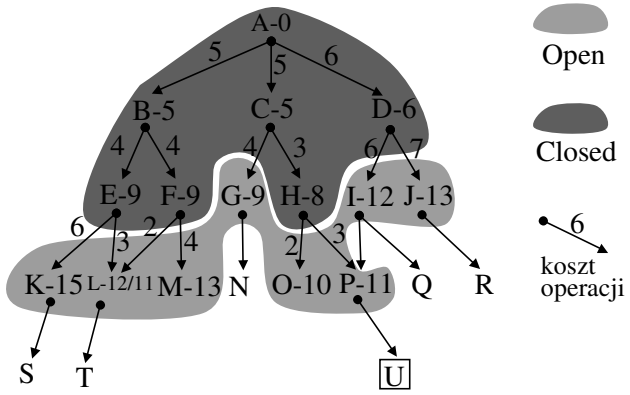
Algorytm UCS - przykład(6)



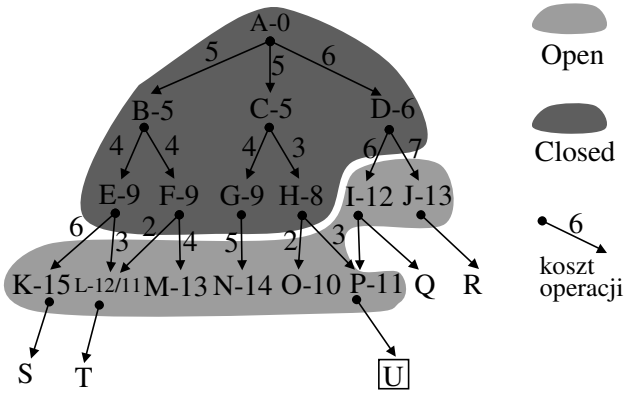
Algorytm UCS - przykład(7)



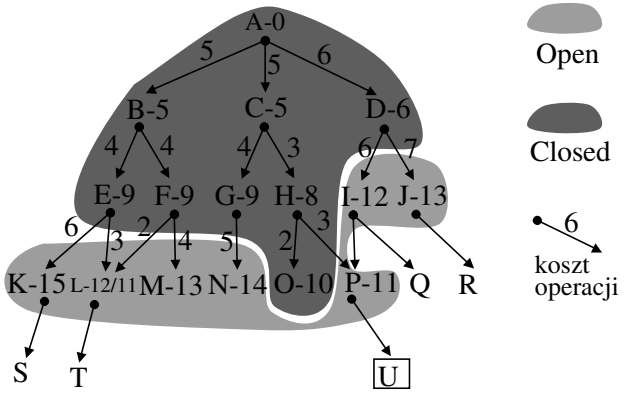
Algorytm UCS - przykład(8)



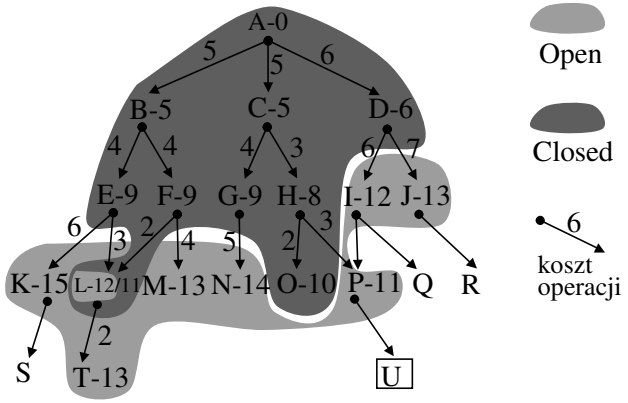
Algorytm UCS - przykład(9)



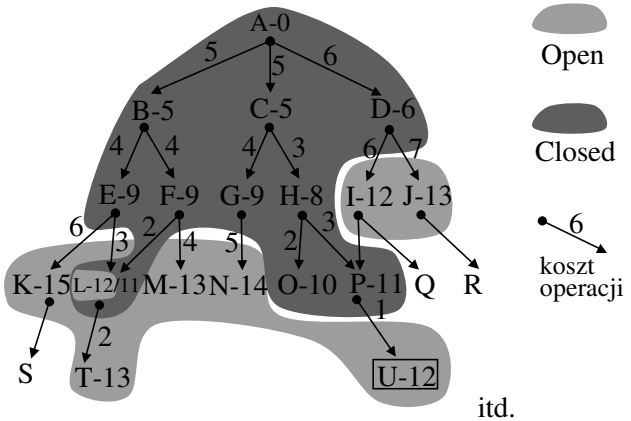
Algorytm UCS - przykład(10)



Algorytm UCS - przykład(11)



Algorytm UCS - przykład(12)



itd.

Algorytm UCS: charakterystyka

Cechy szczególne

- Ocena stanu to koszt dojścia do tego stanu (koszt pokonanej ścieżki od stanu początkowego) - koszt przejścia z dowolnego stanu do jego stanu potomnego musi być zawsze ≥ 0
- Stanowi uogólnienie algorytmu przeszukiwania wszerz - w algorytmie przesz. wszerz koszt, to głębokość na jakiej leży stan
- Optymalny (gwarantuje znalezienie najkrótszego rozwiązania)

Wady

- Duże wymagania czasowe i pamięciowe (rosnące wykładniczo!)

Porównanie strategii przeszukiwania

Kryterium	Wszerz	Uniform-cost	W głąb z nawrotami	Iteracyjne pogłębianie	Dwukie-runkowe (jeśli możliwe)
Czas	B^d	B^d	B^n	B^d	$B^{d/2}$
Pamięć	B^d	B^d	Bn	Bd	$B^{d/2}$
Optymalny?	Tak	Tak	Nie	Tak	Tak
Zupełny?	Tak	Tak	Tak	Tak	Tak

B - średnia liczba następników każdego stanu (ang. branching factor)
n - maksymalna głębokość przeszukiwania
d - głębokość rozwiązania

Zasady implementacji algorytmów przeszukiwania przestrzeni stanów

1. Reprezentacja rozwiązania problemu jako *ścieżki* od stanu początkowego do stanu docelowego.
2. Przeglądanie *systematyczne* wszystkich ścieżek w poszukiwaniu celu.
3. *Powrót do poprzedniego stanu* pozwalający na wznowienie przeszukiwania w sytuacji, gdy dotychczasowa ścieżka nie prowadzi do celu - mechanizm *nawrotów*.
4. Struktury listowe umożliwiające utrzymywanie w sposób jawny danych o aktualnie analizowanych stanach:
   - lista *open*, pozwalająca na powrót do nie odwiedzonych jeszcze stanów,
   - lista *closed* stanów już odwiedzonych, pozwalająca na wykrywanie pętli i uniknięcie powtarzania bezowocnych ścieżek.
5. Zastosowanie *stosu* w algorytmie przeszukiwania w głąb i *kolejki jednokierunkowej* w przeszukiwaniu wszerz.