Politechnika
Warszawska

MSI

6. Przeszukiwanie poinformowane (z heurystyką)

Włodzimierz Kasprzak

Układ

1. Strategia „zachłanna" z heurystyką
2. Przeszukiwanie A*
3. Generowanie heurystyki kosztu
4. IDA*
5. SMA*
6. Strategie suboptymalne
7. „Real-time" A* (RTA*)

1. **Strategia zachłanna** z heurystyką („najbliższy celowi najpierw”)

- W tej strategii funkcja oceny przyjmuje postać:

  $f(n) = h(n)$ (czyli składa się wyłącznie z heurystyki, oszacowanie kosztów resztkowych, kosztów przejścia z węzła n do celu).

  Np. $h(n)$: „odległość w linii prostej z miasta n do Krakowa”.

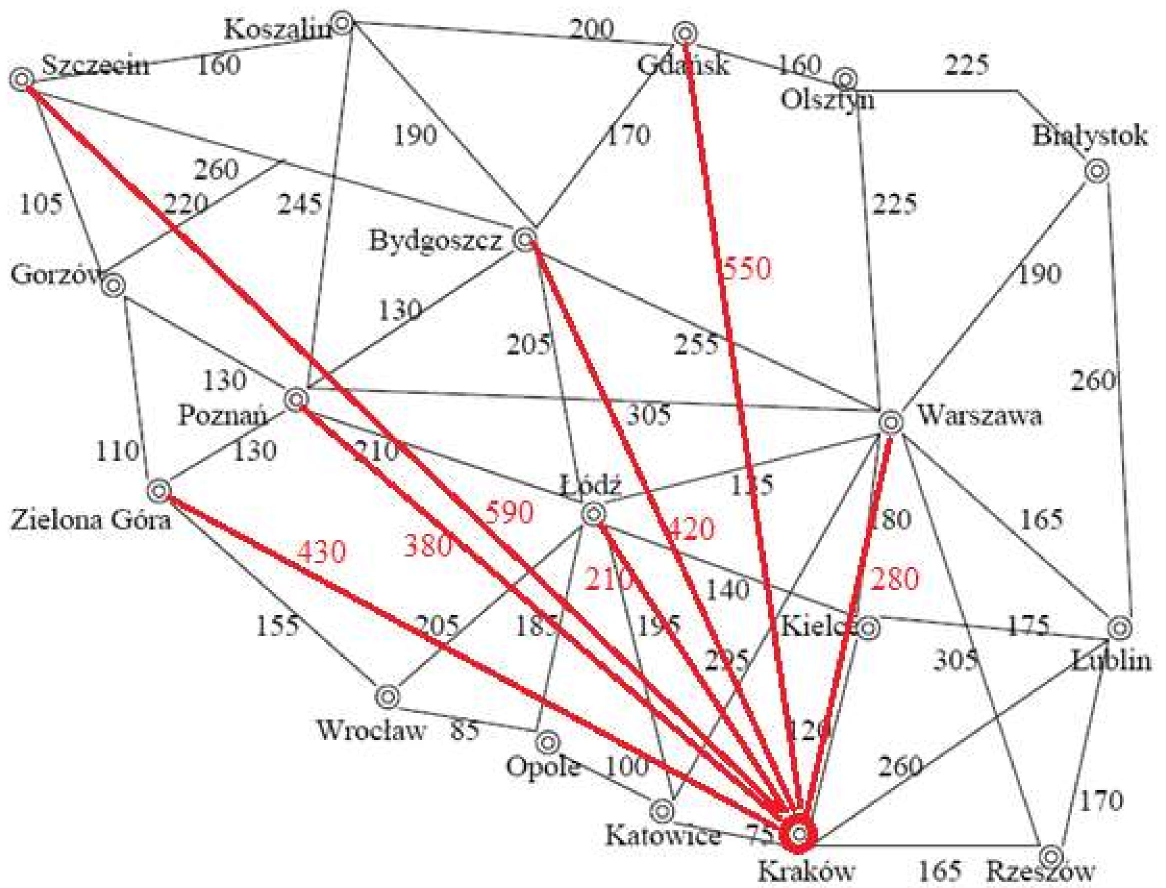
- Strategia przeszukiwania oceniająca, że „najlepszy węzeł to najbliższy celowi”, wybiera i rozwija ten węzeł, który wydaje się być *najbliższy* węzłowi docelowi. Dotychczas poniesione koszty dojścia do aktualnego węzła nie odgrywają żadnej roli.

„Strategia zachłanna z heurystyką”

1	INIT: Pobierz węzeł startowy s i umieść go w zbiorze OPEN. Ustaw $f(s) = h(s)$
2	Pobierz z OPEN najlepszy węzeł n (o najmniejszym koszcie $f(n)$) i przenieś go do CLOSED
3	JEŚLI (n jest węzłem końcowym) TO zakończ i zwróć g(n) oraz całą ścieżkę od s do n. 1
4	Znajdź węzły następców n - niech będą nimi: $n_1' \ldots n_k'$.
5	Dla każdego z następców $n_1' \ldots. n_k'$ wyznacz jego koszt $f_i = h(n'_i)$
6	Dla każdego z węzłów $n_1' \ldots. n_k'$:
	JEŚLI (n_i' nie należy do zbioru OPEN ani do CLOSED) TO dodaj go do zbioru OPEN i ustaw: $f(n_i') = f_i$.
7	Powtórz od kroku 2.

Przykład heurystyki: odległość w linii prostej od celu

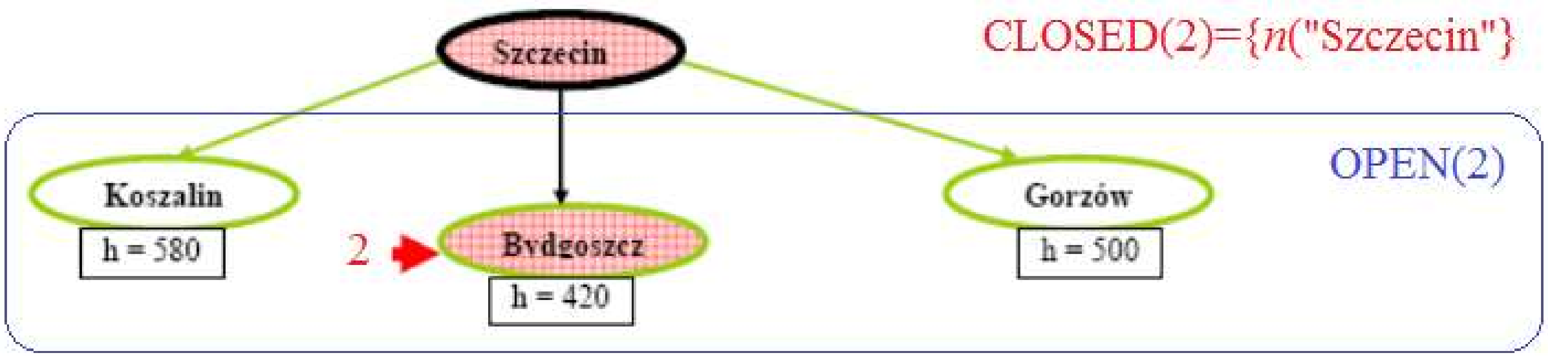
Mapa rzeczywistych odległości cząstkowych o szacowanych kosztów resztkowych względem Krakowa:



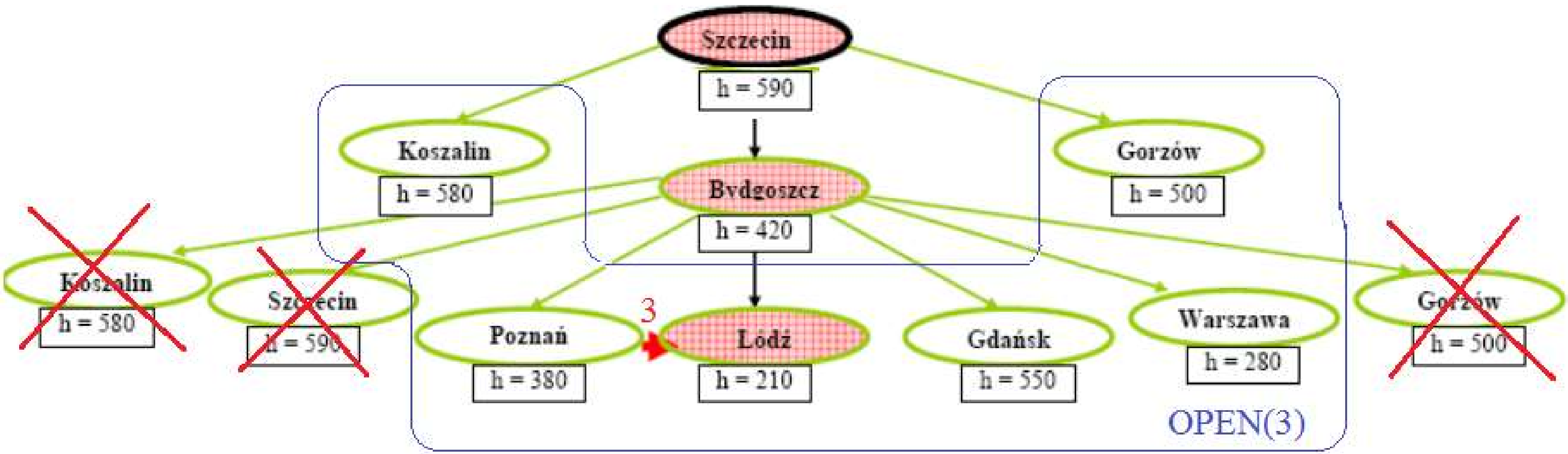
Oszacowanie $h(n)$: odległość w prostej linii do celu (z n do Krakowa)

Białystok	440
Bydgoszcz	420
Gdańsk	550
Gorzów	500
Katowice	70
Kielce	110
Koszalin	580
Kraków	0
Lublin	230
Łódź	210
Opole	150
Poznań	380
Rzeszów	150
Olsztyn	460
Szczecin	590
Warszawa	280
Wrocław	240
Zielona Góra	430

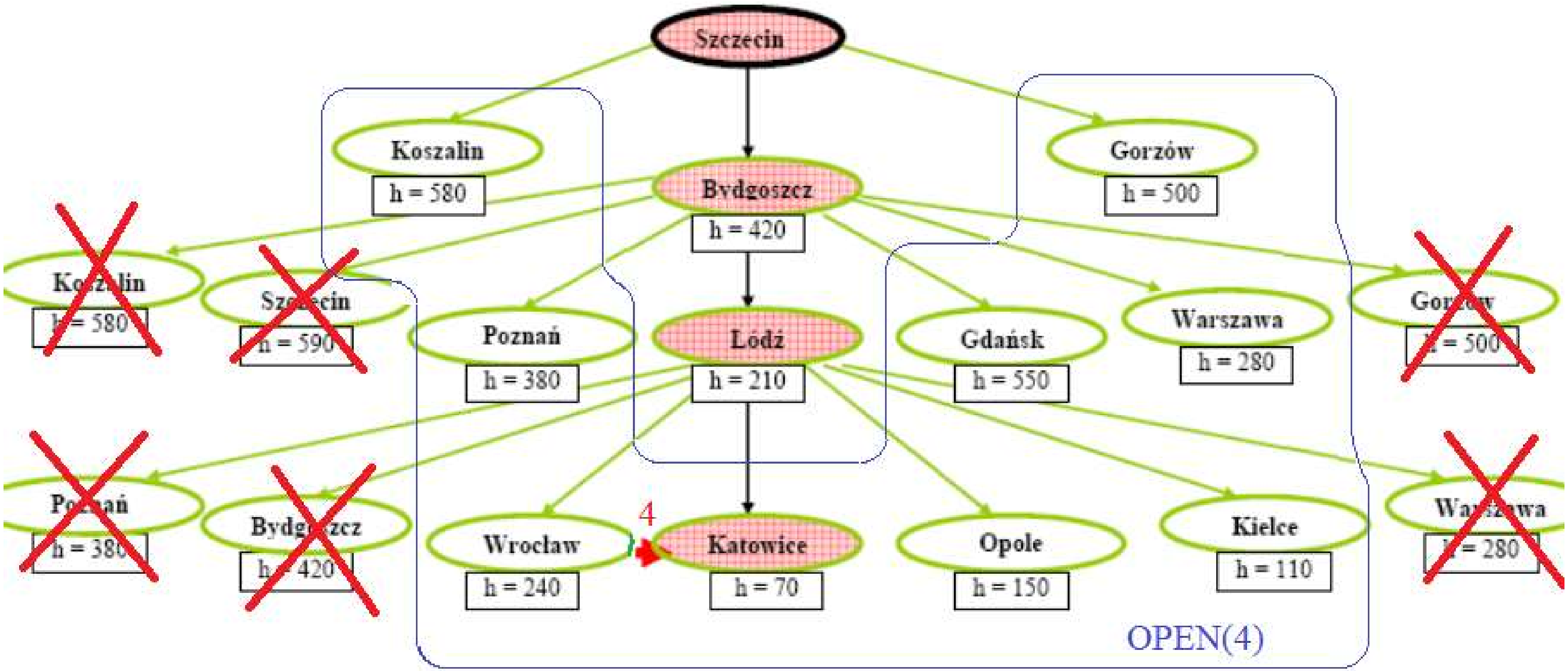
Przykład: strategia „zachłanna”



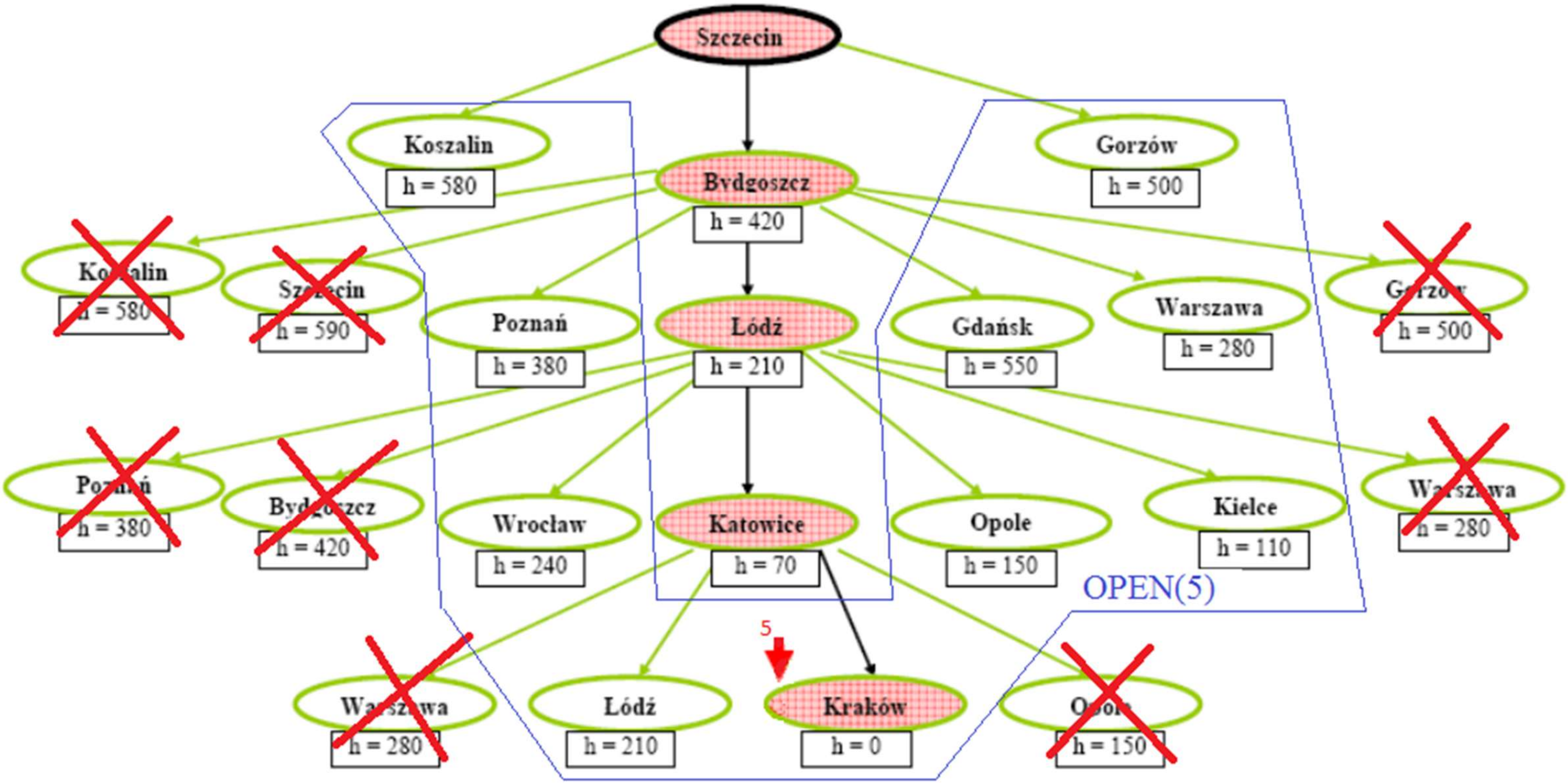
Przykład: strategia „zachłanna” (2)



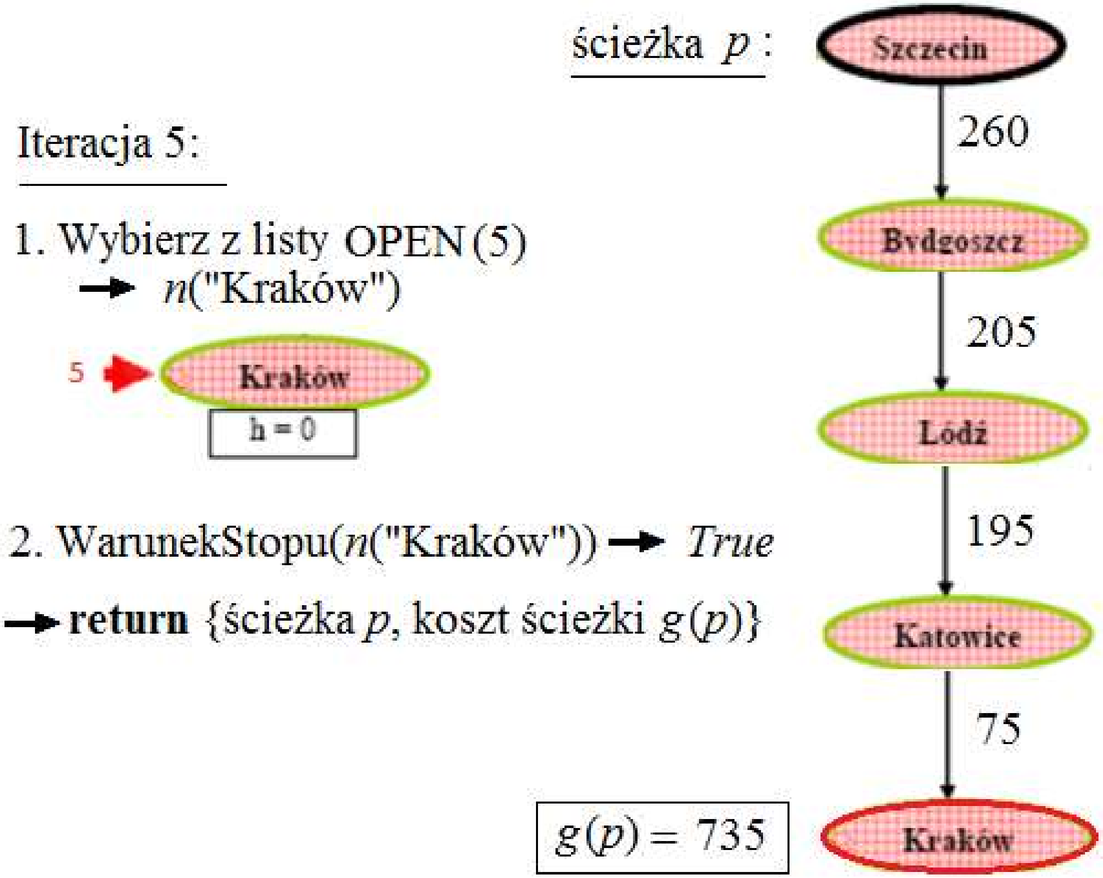
Przykład: strategia „zachłanna” (3)



Przykład: strategia „zachłanna” (4)



Przykład: strategia „zachłanna” (5)



Cechy strategii „zachłannej”

- Zupełność? Nie – może utknąć w pętli (w wersji bez sprawdzania ze zbiorem CLOSED).
- Czas? $O(b^m)$, ale dobra heurystyka może dać znaczącą poprawę.
- Pamięć? $O(b^m)$ – utrzymuje wszystkie węzły w pamięci.
- Optymalność? **Nie** (np. wybrano drogę przez *Bydgoszcz* zamiast drogi optymalnej przez *Gorzów*).

Zasadniczą wadą tej strategii jest brak gwarancji uzyskania optymalnego rozwiązania (w sensie minimalizacji sumarycznego kosztu ścieżki względnie maksymalizacji jakości).

2. Przeszukiwanie A*

- <u>Idea:</u> unikać rozwijania sekwencji, które już dotąd są kosztowne a poza tym niezbyt „obiecujące” pod względem możliwości szybkiego dojścia do celu.
- <u>Stosowana jest funkcja oceny odnosząca się do kosztu</u>:

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

  – $g(n)$ = koszt dotarcia do n ;

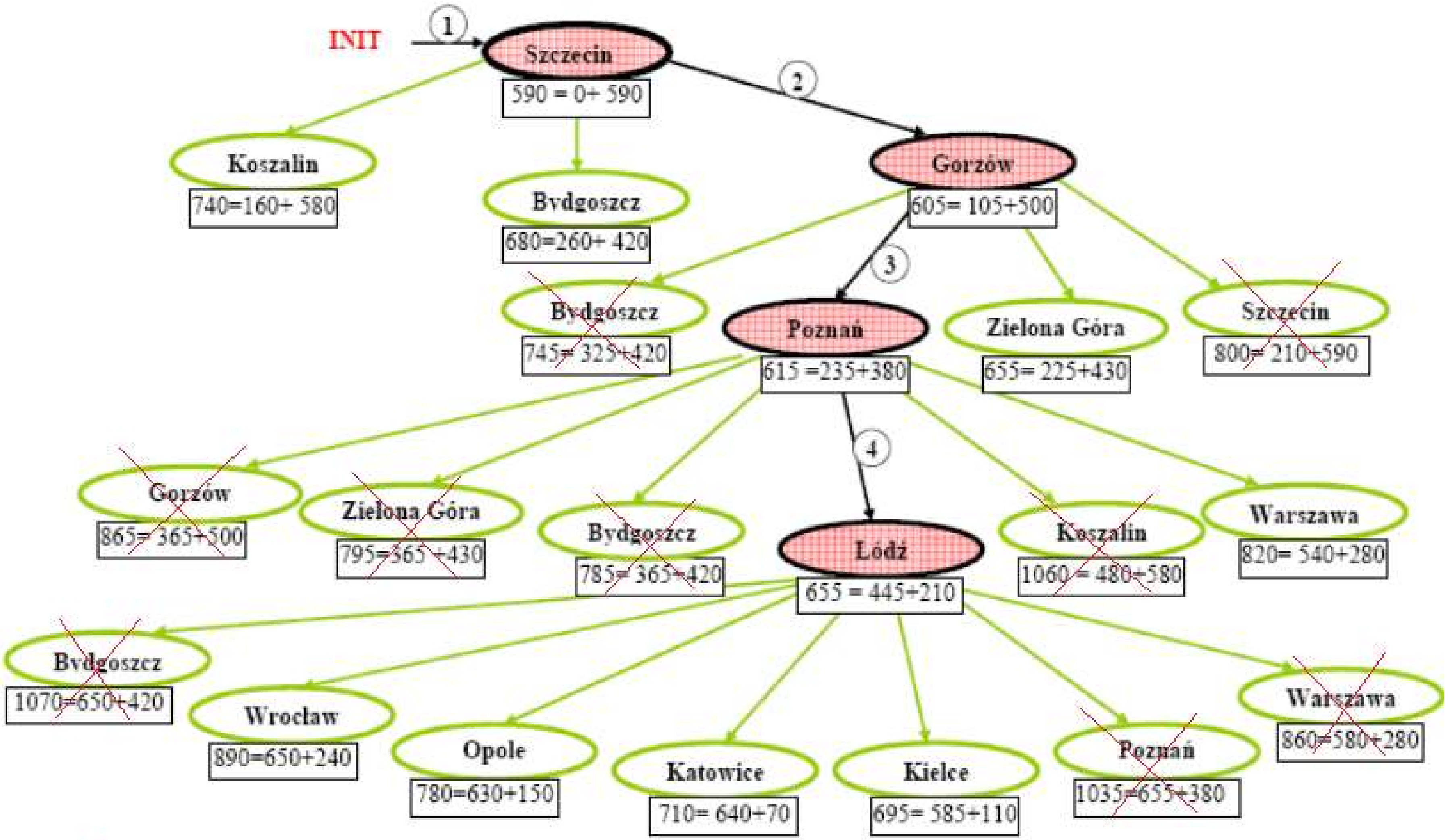
  – $h(n)$ = przewidywany koszt z n do celu ;
- Tym samym $f(n)$ reprezentuje przewidywany całkowity koszt ścieżki prowadzącej od węzła startowego przez węzeł n do celu.
- Uwaga: w *dalszym ciągu przy omawianiu strategii A* funkcja oceny będzie wyrażać koszt.*

Algorytm „przeszukiwania A* ”

1	INIT: Pobierz węzeł startowy *s* i umieść go w zbiorze OPEN. Ustaw $f(s)=0$, $g(s)=0$.	
2	Pobierz z OPEN węzeł *n* o najmniejszej wartości funkcji $f(n)$ i umieść go w zbiorze CLOSED.	
3	JEŚLI (*n* jest węzłem końcowym) TO zakończ i zwróć g(n) oraz całą ścieżkę od *s* do *n*.	
4	Znajdź węzły następców *n* - niech będą nimi: $n_1' \dots n_k'$.	
5	Dla każdego z następców $n_1' \dots. n_k'$ oblicz koszt dojścia do niego: $g_i' = g(n) + c(n, n_i')$.	
6	Dla każdego z węzłów $n_1' \dots. n_k'$:	
	a	JEŚLI (n_i' nie należy do zbioru OPEN ani do CLOSED) TO dodaj go do zbioru OPEN i ustaw: $g(n_i') = g_i'$, $f(n_i') = g_i' + h(n_i')$.
	b	JEŚLI (istnieje $m \equiv n_i'$, należy do zbioru OPEN lub CLOSED i $g(m) > g_i'$) TO ustaw $g(n_i') = g_i'$, $f(n_i') = g_i' + h(n_i')$, usuń m, usuń ścieżkę od *s* do *m* dodaj n_i' do zbioru OPEN
7	Powtórz od kroku 2.	

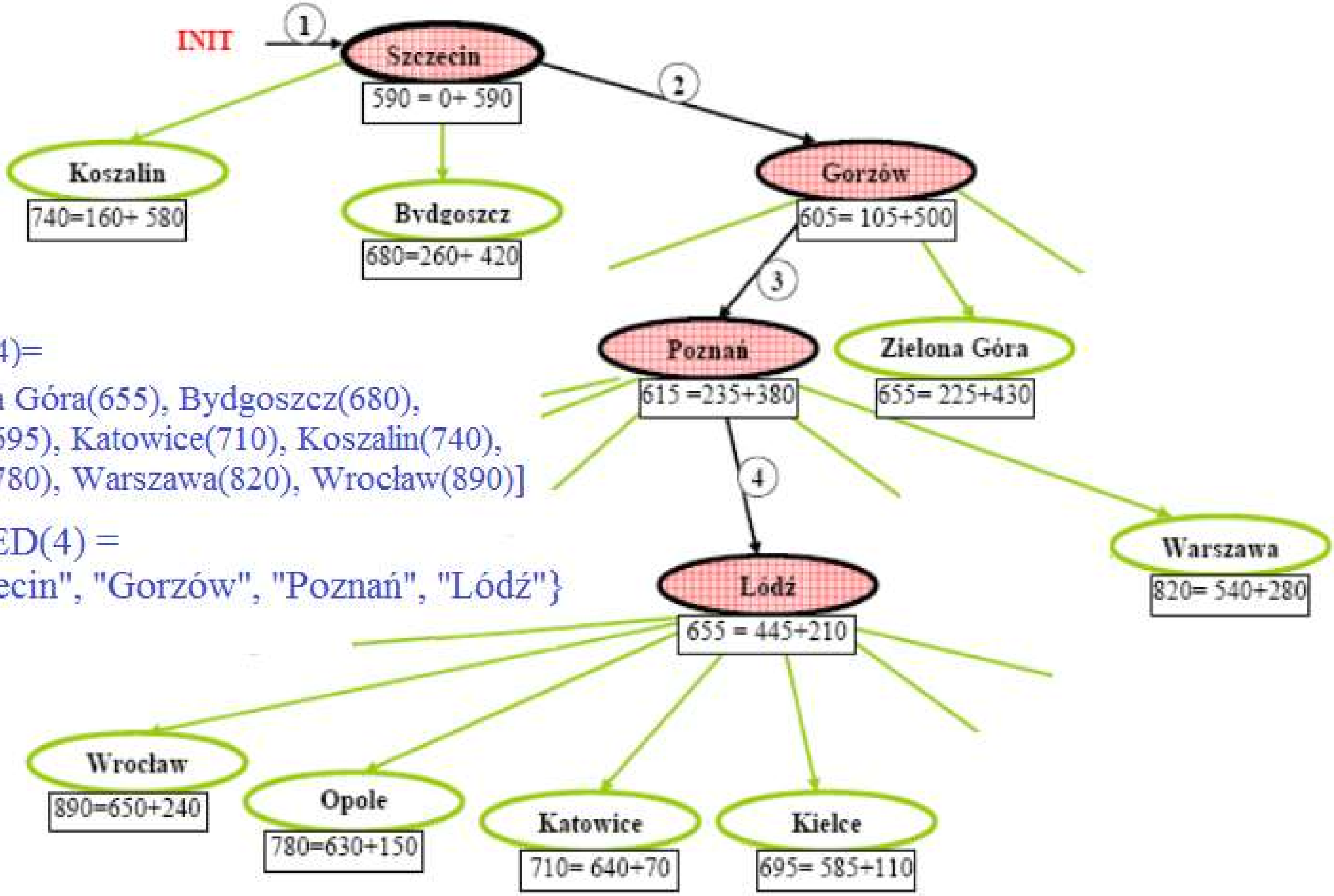
Przykład przeszukiwania A* (1)

Po czterech iteracjach:



Przykład przeszukiwania A* (2)

Z każdych dwóch równoważnych węzłów, „gorszy” węzeł jest usuwany. <u>Rzeczywiste drzewo przeszukiwania po 4 iteracjach</u>:

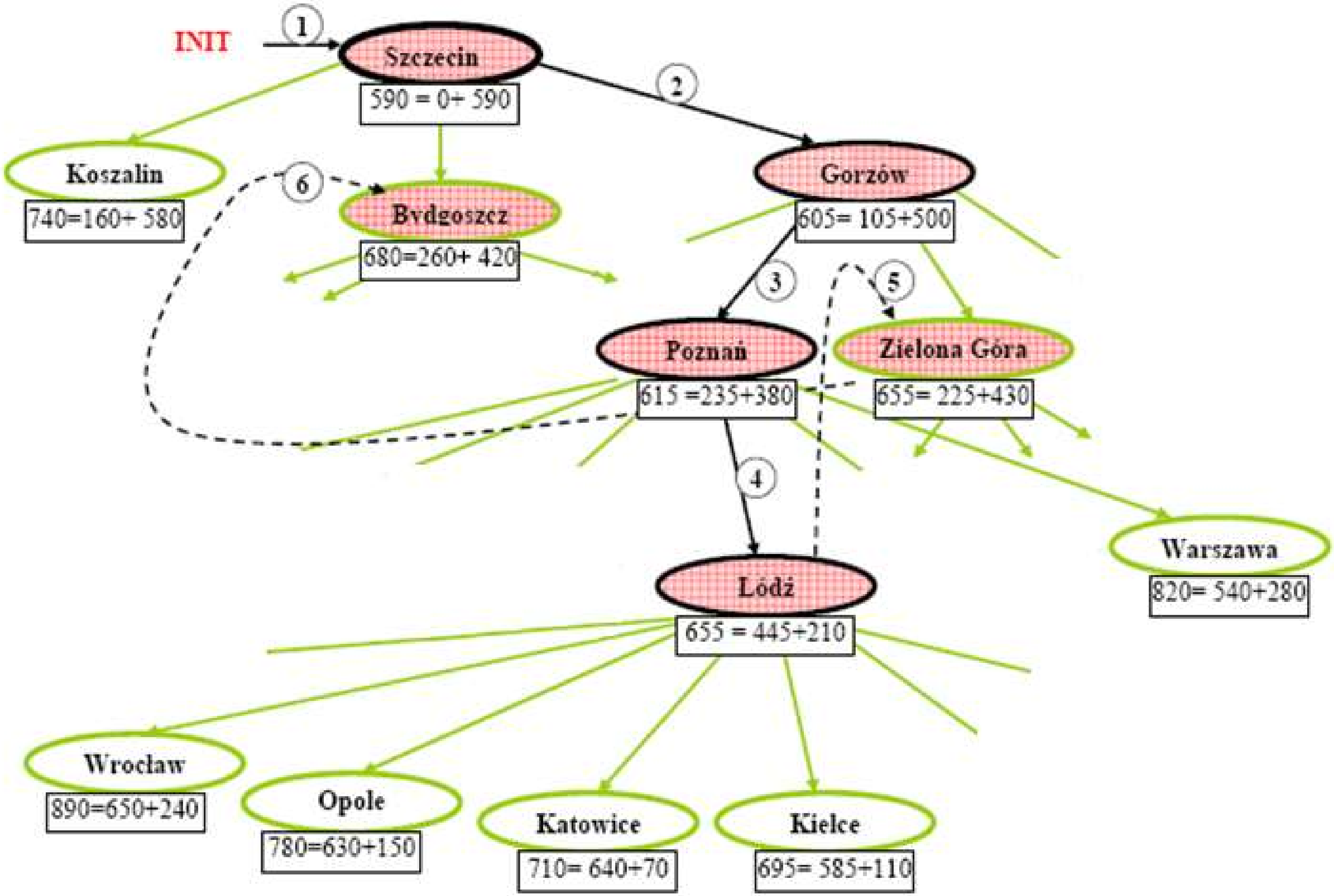


OPEN(4)=
[Zielona Góra(655), Bydgoszcz(680), Kielce(695), Katowice(710), Koszalin(740), Opole(780), Warszawa(820), Wrocław(890)]

CLOSED(4) =
{"Szczecin", "Gorzów", "Poznań", "Łódź"}

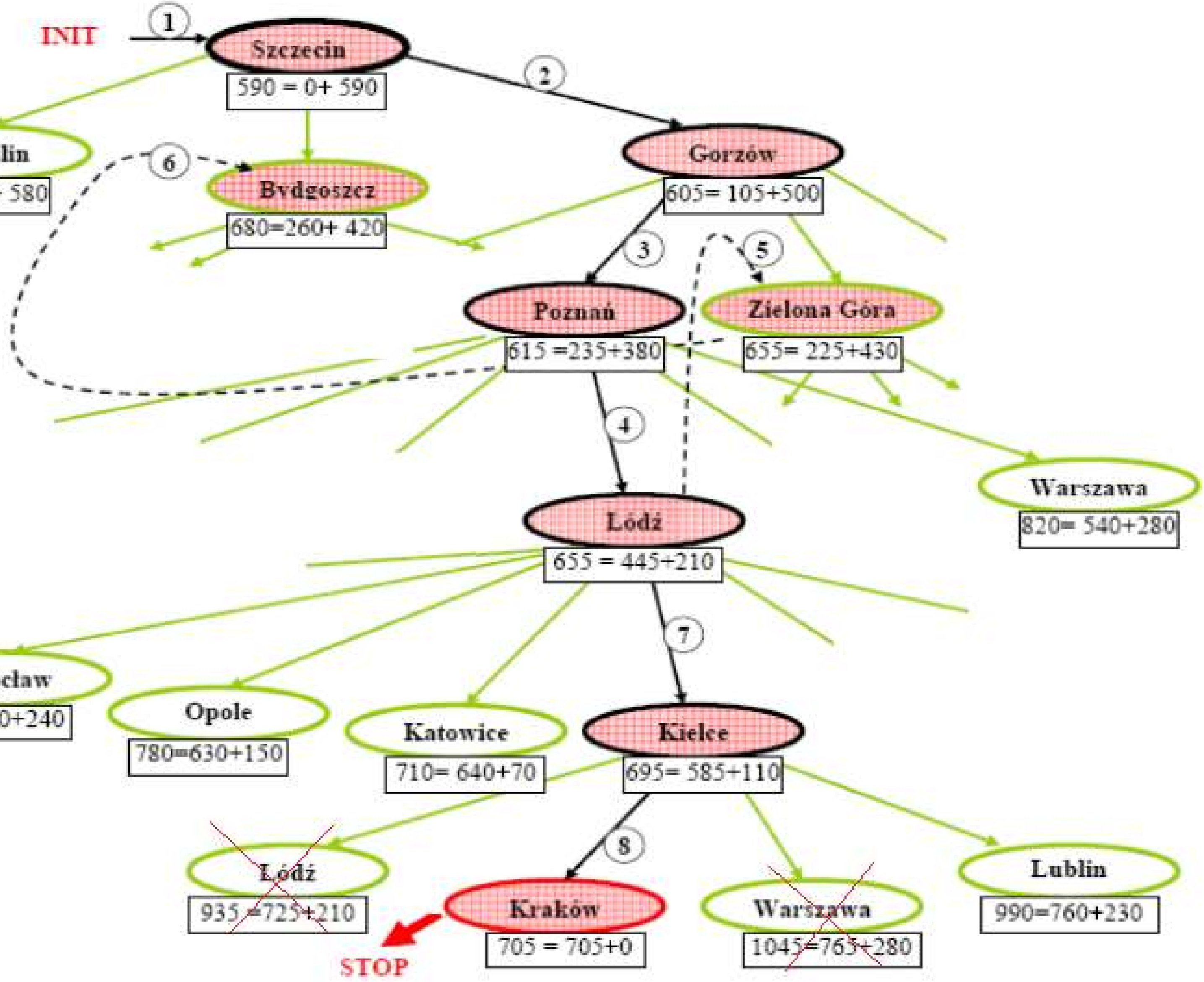
Przykład przeszukiwania A* (3)

Po sześciu iteracjach:



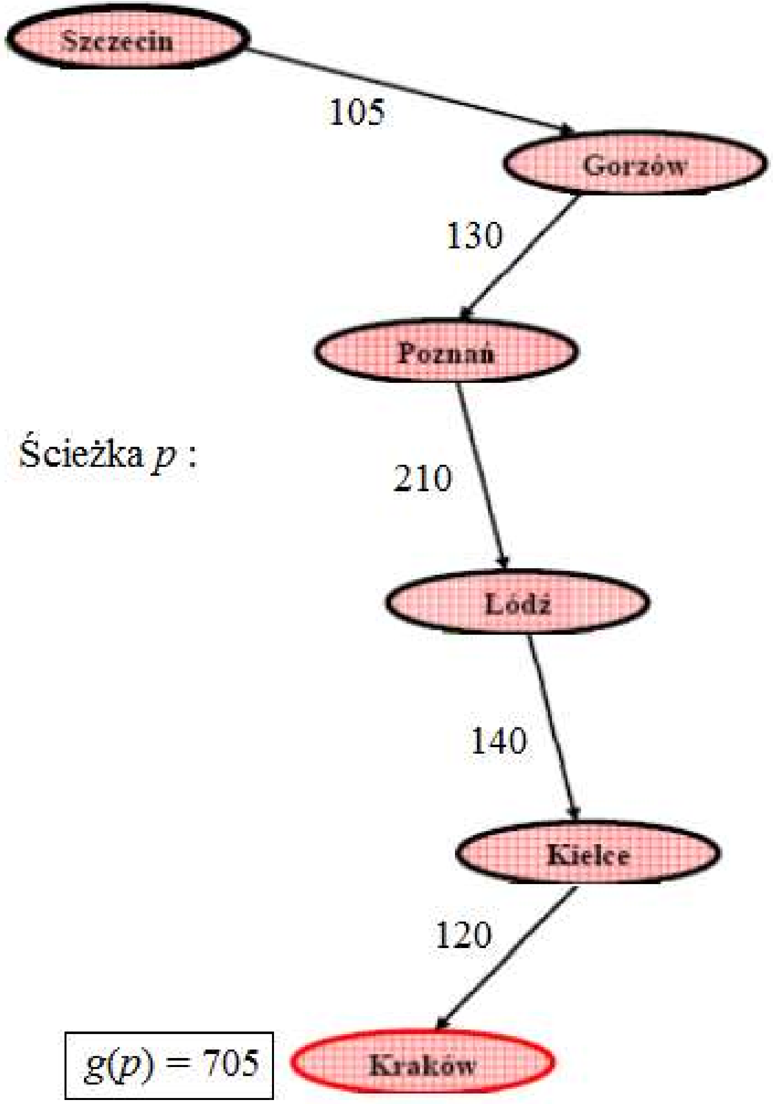
Przykład przeszukiwania A* (4)

Po ośmiu iteracjach - cel osiągnięty:



Przykład przeszukiwania A* (5)

Zwracany wynik: [ścieżka p, koszt ścieżki $g(p) = 705$]:



Cechy algorytmu A*

- Zupełność? Tak (chyba, że istnieje nieskończenie wiele węzłów *n*, dla których $f(n) \leq f(G)$, gdzie *G* jest optymalnym celem).
- Czas? Potencjalnie wykładniczy – zmniejszenie czasu zależy od istnienia dobrej heurystyki.
- Pamięć? Wszystkie rozwijane węzły są pamiętane.
- Optymalność? Tak, jeśli heurystyka jest *dopuszczalna* to A* zawsze znajduje najlepsze rozwiązanie.

Przeszukiwanie A* dysponuje jeszcze jedną ciekawą cechą:

- Optymalna efektywność: przy *spójnej* heurystyce, żaden inny algorytm przeszukiwania grafu nie rozwija mniej węzłów niż algorytm A* dla optymalnego dotarcia do celu.

Dopuszczalna heurystyka

- Heurystyka *h*(*n*) jest **dopuszczalna** (ang. *admissible*), jeżeli dla każdego węzła *n* zachodzi

$$h(n) \le h^*(n),$$

gdzie $h^*(n)$ jest prawdziwym kosztem osiągnięcia celu z węzła n.

- Dopuszczalna heurystyka nigdy nie przecenia kosztu osiągnięcia celu – jest optymistyczna .
- Przykład: $h(n)$ *dla problemu dotarcia do Krakowa* nigdy nie przecenia faktycznej odległości liczonej wzdłuż drogi.
- Twierdzenie

Jeżeli $h(n)$ *jest dopuszczalna*, to algorytm A^* jest optymalnym algorytmem **przeszukiwania grafu**.

Optymalność A*

- Strategia A* rozwija węzły w kolejności nie malejących wartości funkcji oceny (kosztu) f :
- $f_i \leq f_{i+1}$
- Strategia A* nie może wybrać węzła o określonym koszcie f_i zanim „nie sprawdzi” (nie wybierze) uprzednio wszystkich węzłów o niższym koszcie.
- Gwarantuje to osiągnięcie optymalnego rozwiązania (jeśli heurystyka jest dopuszczalna).

Spójna heurystyka

- Heurystyka jest spójna (*consistent*), jeżeli jest dopuszczalna i dla każdego węzła n i dla każdego następnika n', osiąganego przez akcję a, spełniony jest „warunek trójkąta”:

  $$h(n) \leq c(n,a,n') + h(n'),$$

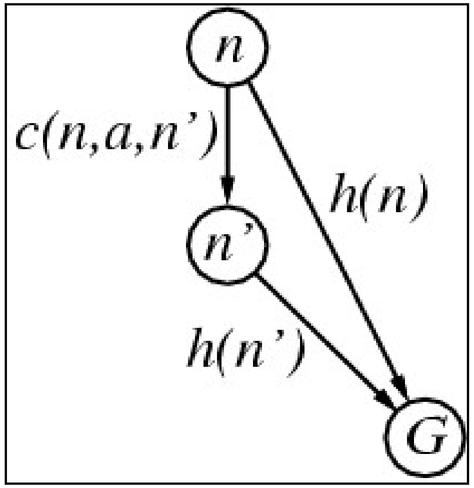
  gdzie $c(n,a,n')$ - koszt przejścia z n do n' za pomocą akcji a.
- O koszcie każdej akcji $c(n,a,n')$ z góry zakładamy, że jest nieujemny. Wtedy spójność h oznacza, że

  1) $h(n) \leq h^*(n)$, czyli heurystyka jest dopuszczalna;

  2) $f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n')$

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

  $f(n') \geq f(n)$, czyli ocena $f(n)$ jest niemalejąca (**monotoniczna**) wzdłuż dowolnej ścieżki.



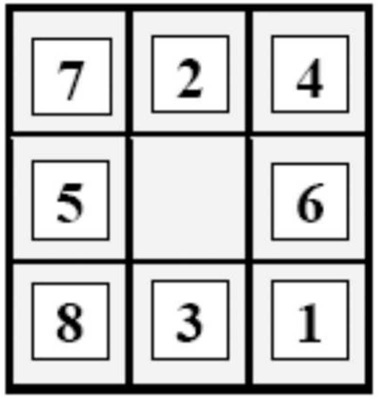
Efektywnościowa optymalność

- Jeżeli heurystyka $h(n)$ jest spójna, to algorytm A* jest **efektywnościowo optymalnym** przeszukiwaniem grafu.
- Oznacza to, że żadna inna strategia bazująca na funkcji kosztu **nie** rozwija (wizytuje) **mniej węzłów** niż strategia A*.

3. Generowanie heurystyki kosztu

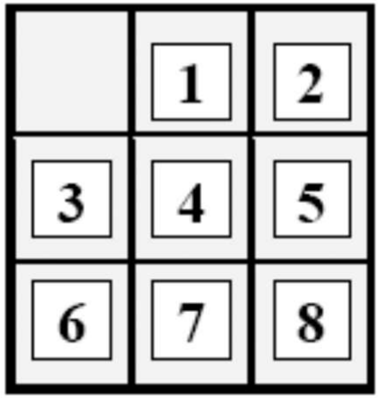
Przykłady heurystyk dla problemu 8 puzzli:

- $h_1(n)$: liczba kafelków nie na swoim (docelowym) miejscu,
- $h_2(n)$: całkowita odległość kafelków od swoich (docelowych) miejsc wyrażona w metryce Manhattan



7	2	4
5		6
8	3	1

Stan początkowy



	1	2
3	4	5
6	7	8

Stan końcowy

Np.

- $h_1(\text{Start}) = 8$
- $h_2(\text{Start}) = 3+1+2+2+2+3+3+2 = 18$

Dominacja heurystyki

- *Jeżeli* $h_2(n) \geq h_1(n)$ dla każdego węzła *n* (i obie heurystyki są dopuszczalne) to h_2 dominuje nad h_1
- Wtedy h_2 *jest lepszą heurystyką niż* h_1 dla poinformowanej strategii przeszukiwania.
- Przykładowe koszty przeszukiwania dla problemu *D* puzzli:

D = 12

Przeszukiwanie ślepe IDS → ok. 3,644,000 rozwijanych węzłów
Strategia A^* z heurystyką h_1 → 227 węzłów
Strategia A^* z heurystyką h_2 → 73 węzły

D = 24

Przeszukiwanie ślepe IDS → brak wyniku – zbyt dużo węzłów
Strategia A^* z heurystyką h_1 → 39,135 węzłów
Strategia A^* z heurystyką h_2 → 1,641 węzłów

Generowanie heurystyk metodą „złagodzonego problemu”

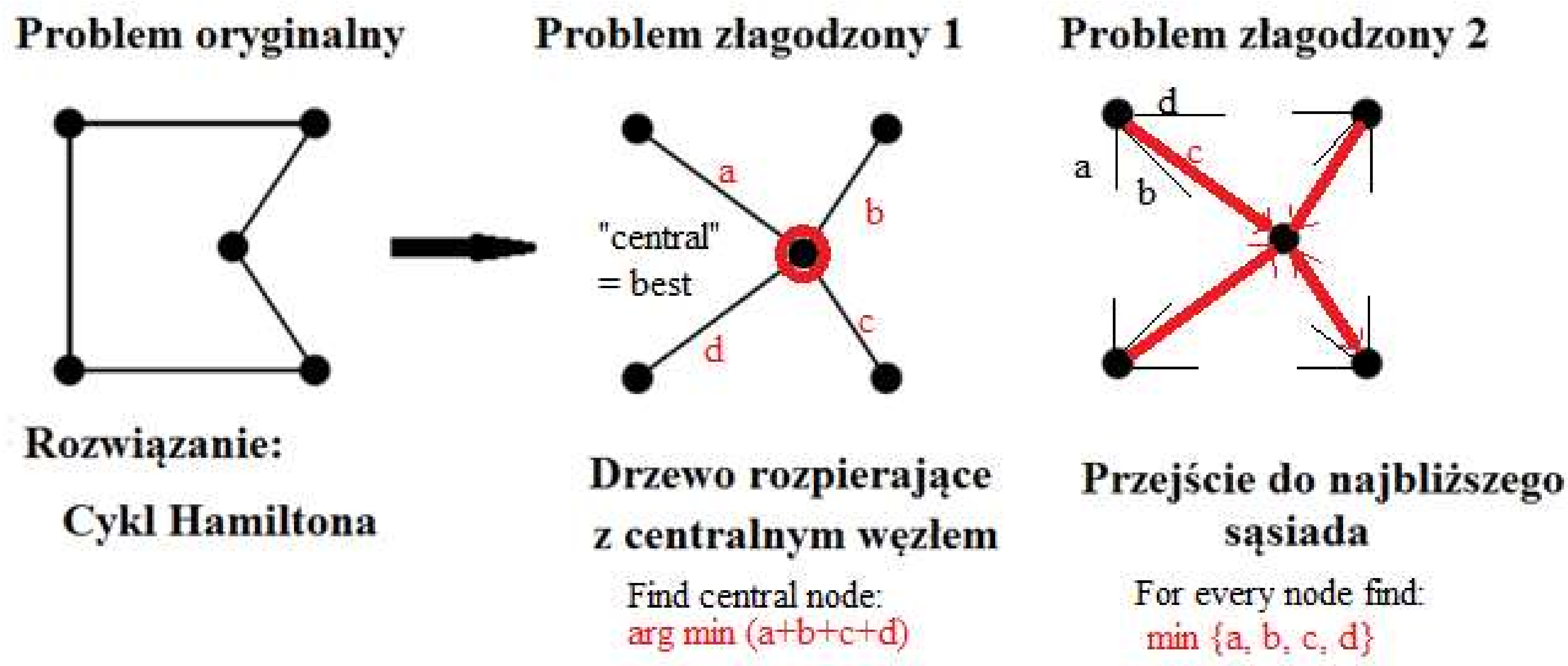
- Problem „uproszczony” o zmniejszonych (w porównaniu z oryginalnym) ograniczeniach nakładanych na przejścia pomiędzy stanami nazywamy problemem złagodzonym.
- Optymalne rozwiązanie złagodzonego problemu (od stanu startowego *n*) stanowi jednocześnie dobrą, dopuszczalną heurystykę dla stanu *n* w oryginalnym problemie.
- Np. jeśli złagodzimy problem „8-puzzle” tak, że kafelek może zostać przesunięty w dowolne miejsce, to heurystyka $h_1(n)$ dla oryginalnego problemu odpowiada w problemie złagodzonym kosztowi optymalnej ścieżki (rozwiązania) rozpoczynanej w stanie *n*.
- Podobnie, jeśli złagodzimy problem „8-puzzle” tak, że kafelek może zostać przesunięty w dowolną sąsiednią pozycję (tzn. nawet zajętą), to heurystyka $h_2(n)$ podaje koszt optymalnej ścieżki (rozwiązania) w tym problemie.

Generowanie heurystyk metodą „złagodzonego problemu” (2)

- Problem komiwojażera – znaleźć najkrótszą trasę łączącą n miast, odwiedzając każde miasto jedynie raz (jest to problem o wysokiej złożoności $O(n!)$).
- Problem złagodzony 1 – wyznaczyć minimalne drzewo rozpierające dla (pod-)zbioru węzłów pozostałych do odwiedzenia (jest to problem o złożoności jedynie $O(n^2)$). Problem złagodzony 2 – wyznaczyć sumę przejść do najbliższego sąsiada dla (pod-)zbioru pozostałych miast. Znalezione rozwiązania obu problemów są optymistycznym oszacowaniem kosztu resztkowego w problemie oryginalnym, gdy pozostaje jeszcze wizytowanie zadanego (pod-)zbioru miast.

Generowanie heurystyk (3)

Problem komiwojażera (c.d.)



4. IDA* - Iterative deepening A*

A* wymaga list OPEN i CLOSE dla pamiętania węzłów → może to prowadzić do dużych wymagań wobec pamięci.

Załóżmy, że funkcja kosztu f jest monotoniczna:

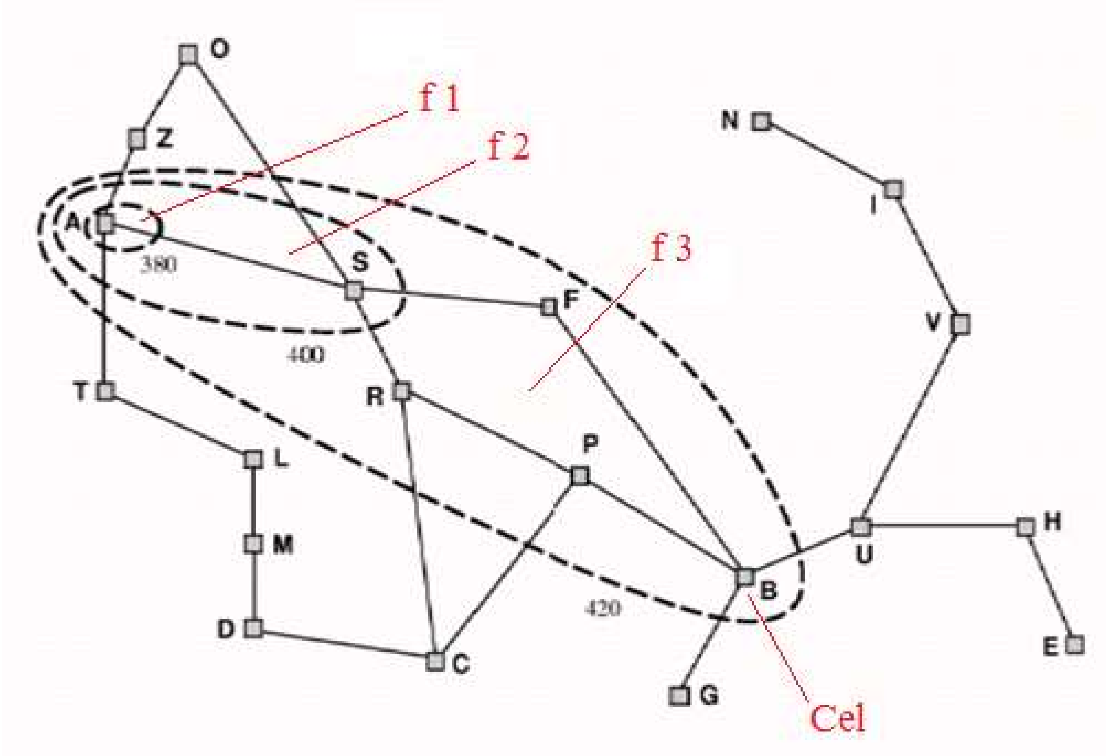
- $f = g + h \leq f^*$ (rzeczywisty koszt) i
- $f(n) \leq f(n')$, gdzie n' jest następnikiem n

Idea: tworzymy podprzestrzeń ograniczoną aktualnym $f(n)$ i przeszukujemy ją „w głąb" (depth-first).

- Wymaga znacznie mniej pamięci.
- Może wymagać więcej obliczeń – jeśli koszt jest znacząco niedoszacowany i wymaganych jest wiele kroków dla osiągnięcia $f = f^*$.
- W najgorszym przypadku: N obliczeń dla A* i N^2 dla IDA*

IDA*

- Monotoniczność funkcji oceny wzdłuż każdej ścieżki umożliwia wyróżnienie kolejnych podprzestrzeni o coraz większych wartościach f.
- Kolejne wartości f odpowiadają coraz lepszemu „poinformowaniu” agenta i „zbliżaniu” się podprzestrzeni do optymalnego celu.
- Przy „dobrej” heurystyce podprzestrzenie „rozciągają się” w kierunku celu.



Algorytm IDA*

Algorytm IDA* („Iterative-Deepening-A*”)

1) (Init) Ustaw f_B = oszacowanie wartości f dla węzła początkowego;

2) Wykonaj przeszukiwanie „w głąb” z węzła początkowego, przycinając gałęzie drzewa za liśćmi wtedy, gdy koszt wygenerowanego liścia: $(g + h) > f_B$.

3) Jeśli znaleziono rozwiązanie to zakończ

→ **return**(optymalne rozwiązanie)

4) Zwiększ f_B do najmniejszej wartości kosztu dla węzłów **liści** wygenerowanych (ale nie rozszerzonych) w aktualnej iteracji i kontynuuj od kroku (2).

Efektywność IDA*

- Złożoność czasowa IDA* asymptotycznie zdąża do złożoności A*.
- Złożoność pamięciowa IDA* wynosi tylko $O(d)$, a A* tymczasem - $O(b^d)$.
- W IDA* unika się sortowania kolejki węzłów.
- IDA* posiada prostszą implementację – brak listy CLOSED (mniejsza lista OPEN).
- IDA* może wykonywać się szybciej mimo, iż generuje więcej węzłów niż A*.
- IDA* może rozwiązać problem, którego A* z powodu braku pamięci nie rozwiąże (lub rozwiąże po znacznie dłuższym czasie).

5. SMA*

SMA* (Simplified Memory Bounded A*)

Parametry:

1.Definiujemy rozmiar dostępnej pamięci, tzn. jaka jest maksymalna liczba węzłów na liście rozwijanych węzłów.

2.Określamy limit długości ścieżki.

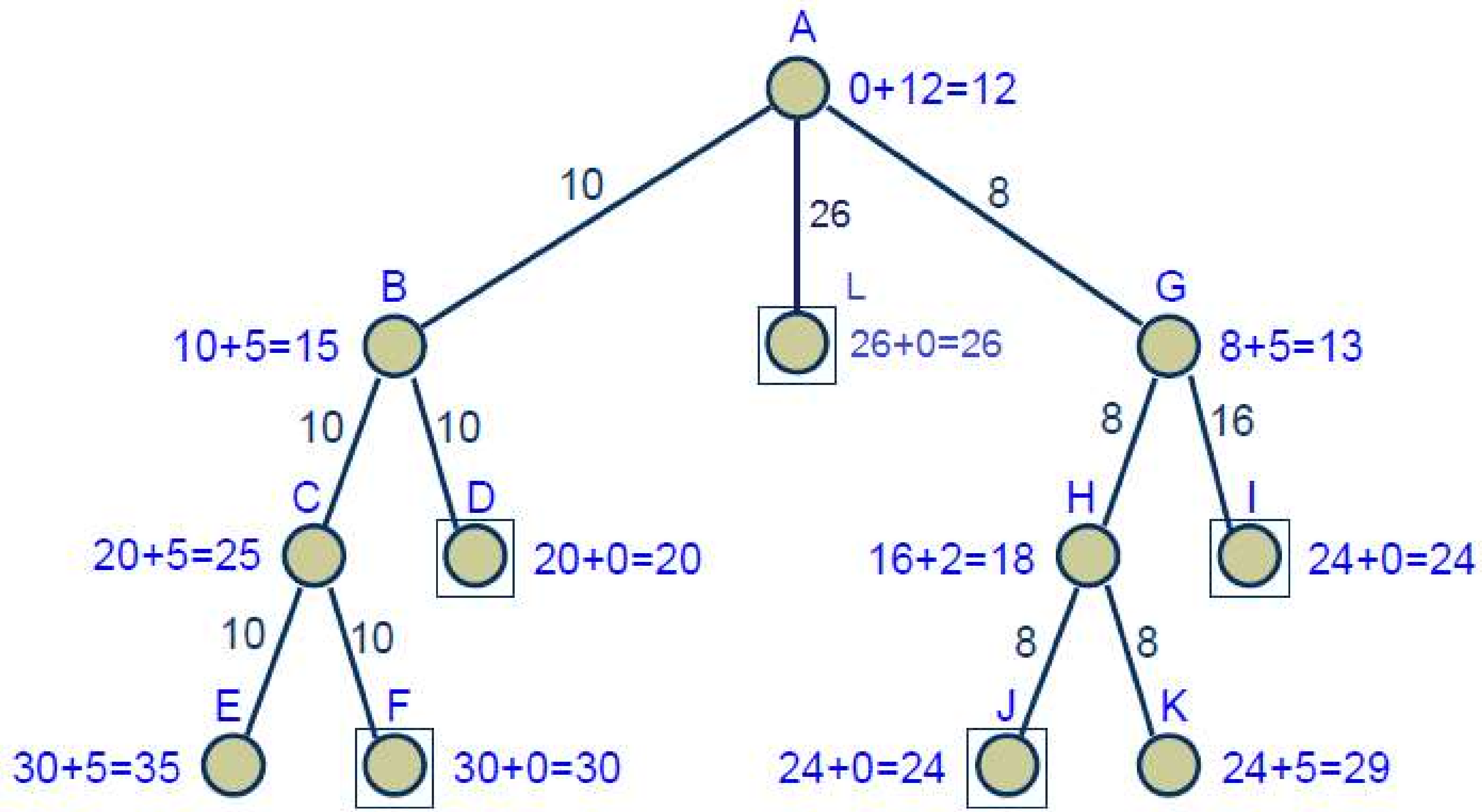
Sensowne jest założenie, że ograniczenie 1 jest większe od ograniczenia 2.

Strategia SMA*:

- Rozwija najgłębszy liść o najmniejszej wartości kosztu f.
- Modyfikuje koszt f węzła niekońcowego zastępując go najmniejszą wartością jego następników.
- Gdy wyczerpie się pamięć, obcina płytsze węzły o najwyższej wartości f – pamięta koszt najlepszego odrzuconego następnika w węźle rodzica.
- Ścieżka o dłuższa niż „limit długości ścieżki" uzyskuje koszt ∞ .

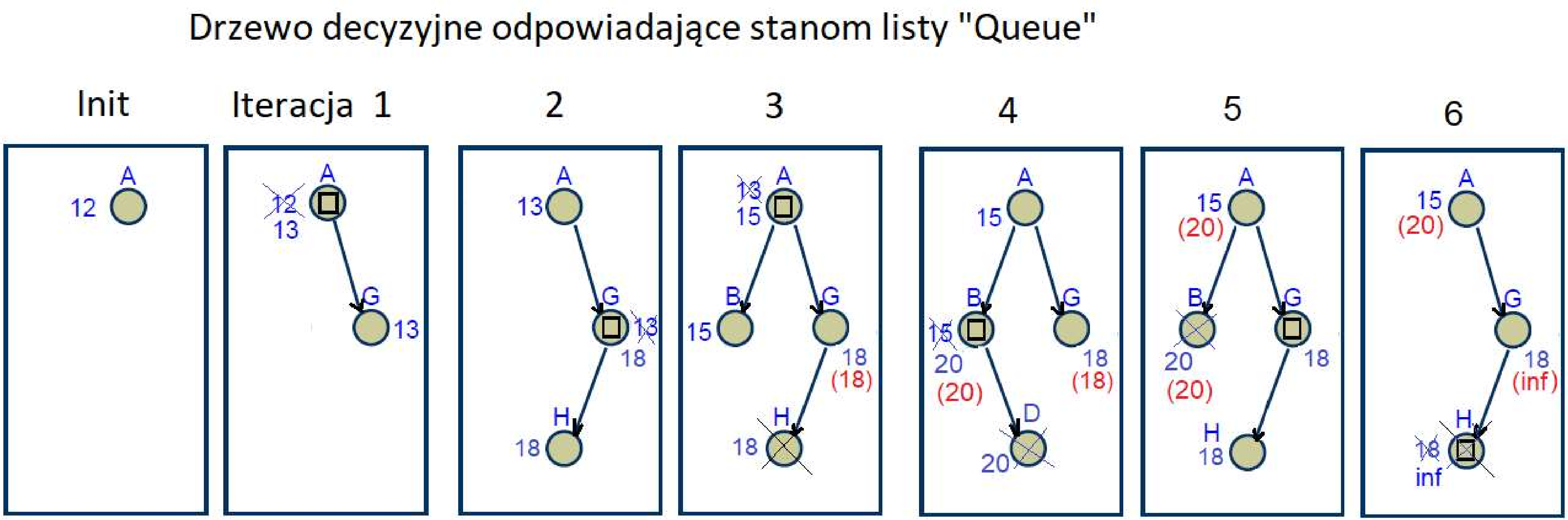
SMA*

- Przykład. Załóżmy (nieznane agentowi) pełne drzewo przeszukiwania o poniższej postaci.



SMA* przykład

- Przykład (c.d.) Wprowadzamy ograniczenie rozmiaru pamięci do 3 węzłów i limit długości ścieżki do 3.

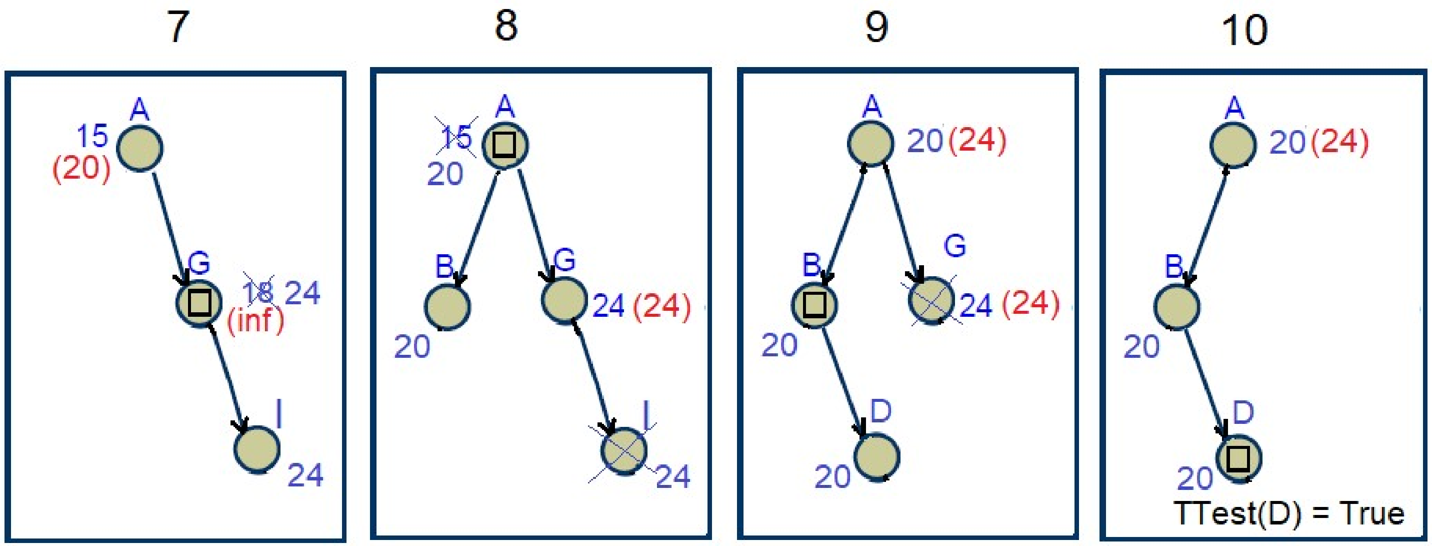


SMA* przykład (c.d.)

Inicjalizacja – węzeł początkowy A(koszt=12).

1) Wybierz A(12), A niekońcowe, dodaj następnik G(13), zmień ocenę A(12->13);
2) Wybierz G(13), G niekońcowe, dodaj następnik H(18), zmień ocenę G(12->18);
3) Wybierz A(13), A niekońcowe, dodaj następnik B(15), zmień ocenę A(13->15), limit liczby węzłów 3 przekroczony – usuń H(18), zapamiętaj G(18,(18)).
4) Wybierz B(15), B niekońcowe, dodaj następnik D(20), zmień ocenę B(15->20), limit liczby węzłów 3 przekroczony – usuń D(2), zapamiętaj B(20, (20)).
5) Wybierz G(18), G niekońcowe, dodaj następnik H(18), ocena B(18->18), limit węzłów 3 przekroczony – usuń B(20), zapamiętaj A(15, (20)).
6) Wybierz H(18), niekońcowe, limit długości ścieżki osiągnięty -> usuń H(inf), zapamiętaj G(18, (inf)).

SMA* przykład c.d.



7. Wybierz G(18), G niekońcowe, dodaj następnik I(24), zmień ocenę A(18->24);

8. Wybierz A(…,(20)), A niekońcowe, dodaj następnik B(20), zmien ocenę A(15->20), limit węzłów 3 przekroczony -> usuń I(24);

9. Wybierz B(20), B niekońcowe, dodaj następnik D(20), ocena D(20->20), limit węzłów 3 przekroczony -> usuń G(24);

10. Wybierz D(20), D spełnia warunek końca -> końcowy wynik: ścieżka A-B-D, koszt ścieżki 20.

Algorytm SMA*

```
function SMAstar(problem, Lm, Lp) returns [ścieżka, koszt ścieżki ]
{ static: Queue; // lista węzłów uporządkowana kosztem f
    Queue ← MAKEQUEUE(MAKENODE(INITIALSTATE[problem]));
    do { if Queue jest puste then return [∅, ∞]; // brak rozwiązania
          n ← najgłębszy węzeł o najmniejszym koszcie w Queue
          if GOALTEST(n) then return [ścieżka do n, jej koszt]; // ścieżka
          if n nie jest celem i jest na maksymalnej głębokości Lp
          then f(n) ← ∞
          else { s ← NEXTSUCCESSOR(n); // najlepszy następnik n
                    f(s) ← g(n) + c(n->s) + h(s)); // koszt węzła s
                    Jeśli nowy węzeł to wstaw s do Queue; // jeśli nowy węzeł
                    f(n) ← MAX(f(n), f(s)); // ewentualnie modyfikuj f(n)
```

(c.d. →)

Algorytm SMA* (c.d.)

(c.d.) **function** SMAstar(*problem*) **returns** *solution*

…

if wszystkie SUCCESSORS(*n*) są w pamięci **then** usuń *n* z *Queue* ;

// gdy wszystkie następniki n są w Queue

}

if (pamięć jest pełna) (tzn. liczba_węzłów > Lm) **then** {

Usuń „*m*”- najpłytszy z węzłów o najwyższym koszcie f z *Queue*;

Wstaw jego przodka „*p*” do *Queue*, jeśli nie występuje tam;

Pamiętaj koszt usuniętego węzła w węźle przodka: $f_{mem}(p)=f(m)$.

}

} // koniec pętli **do**

} // koniec funkcji

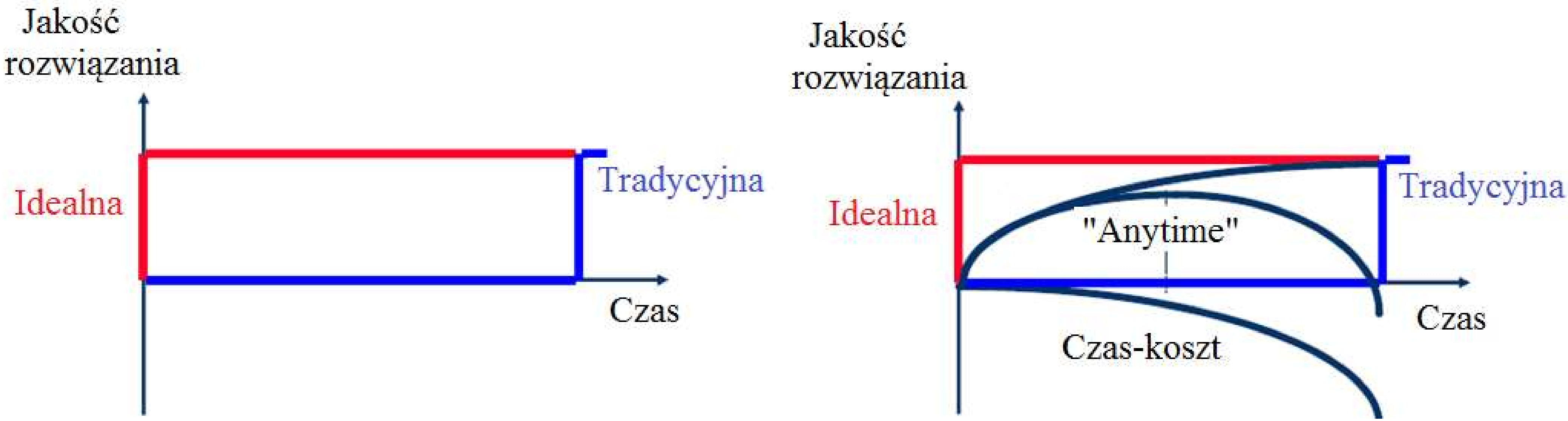
SMA*

Własności strategii SMA*:

- Jest zupełna, jeśli w pamięci zmieści się ścieżka do najpłytszego węzła końcowego.
- Jest optymalna, jeśli w pamięci zmieści się optymalna ścieżka.
- Jeśli w pamięci mieści się całe drzewo, SMA* zachowuje się identycznie jak A*.

6. Strategie suboptymalne

Algorytmy „anytime” („wynik w dowolnym czasie”)



- Idealnie (maksymalna jakość uzyskana zostaje natychmiast)
- Tradycyjnie (maksymalizacja jakości bez uwzględnienia czasu)
- „Anytime” (wartość uwzględnia kryterium jakości i czas jej uzyskania)

Anytime A*

Trzy zmiany w porównaniu do A*:

1. Zastosuj **niedopuszczalną** heurystykę, co umożliwi szybkie znalezienie sub-optymalnych rozwiązań.
2. Kontynuuj przeszukiwanie po znalezieniu pierwszego rozwiązania obcinając nią listę OPEN.
3. Jeśli lista OPEN staje się pusta znalezione rozwiązanie staje się optymalne.

Przykład niedopuszczalnej funkcji kosztu i heurystyki:

$$f'(n) = (1 - w)*g(n) + w*h(n)$$

Powyższa funkcja jest tylko wtedy dopuszczalną funkcją kosztu, gdy $h(n)$ jest dopuszczalne i $w \leq 0.5$.

7. „Real-time” A* (RTA*)

- A* jest stosowany „off-line” – najpierw znajdujemy pełną ścieżkę (tj. sekwencję akcji) od węzła startowego do końcowego, a dopiero potem ta sekwencja akcji jest wykonywana przez agenta.
- Algorytm czasu rzeczywistego (np. RTA*) - pozwala na wykonywanie akcji w trybie „on-line”, tzn. gdy brak jest jeszcze pełnego rozwiązania.

Real Time Heuristic Search (RTA*):

- Następnik aktualnego węzła wybierany jest spośród węzłów w jego lokalnym sąsiedztwie, z możliwością nawrotu do poprzedniego węzła.
- Przy spełnieniu warunku spójnej heurystyki gwarantuje osiągnięcie stanu końcowego, zwykle suboptymalnego w sensie długości ścieżki.

RTA* : oryginalna wersja (+)

Wymagane jest, aby heurystyka była dopuszczalna i funkcja kosztu była monotoniczna (czyli aby heurystyka była spójna).

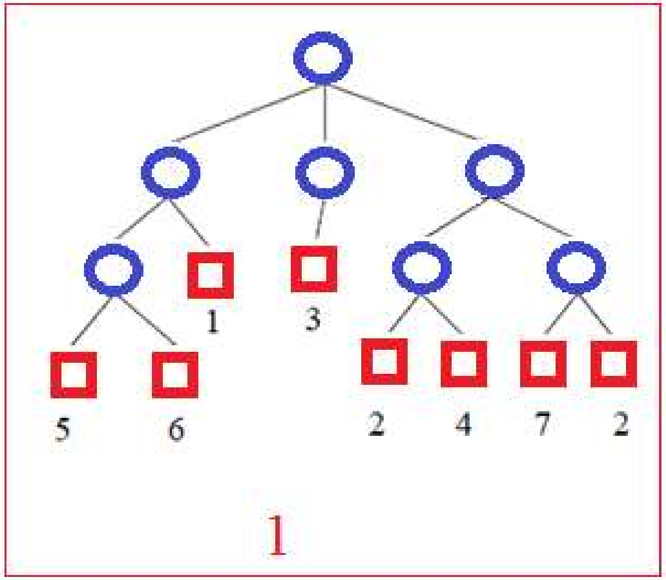
RTA* zawiera następujące modyfikacje A*:

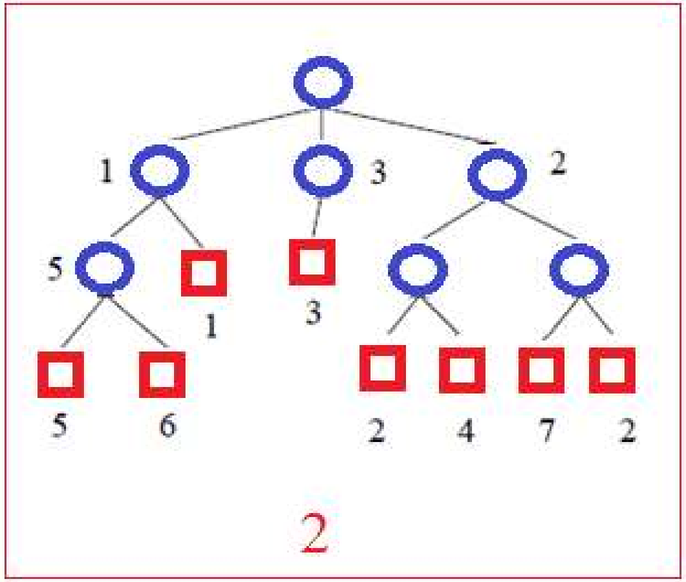
1. Podejmij decyzję o pojedynczej akcji:
   - Wykonaj podgląd w przód (drzewo „*mini-min*" z „cięciami *alfa*");
   - Wybierz najlepszy węzeł w lokalnym sąsiedztwie aktuwalnego węzła (w tym możliwy jest nawrót do węzła-rodzica). Ocena węzłów ustalana jest względem aktualnego węzła.
2. Pamięć związana z dotychczasową ścieżką:
   - W aktualnym węźle, dla unikania pętli, po wyznaczeniu najlepszego następnika, pamiętany jest też „drugi najlepszy" węzeł-następnik.

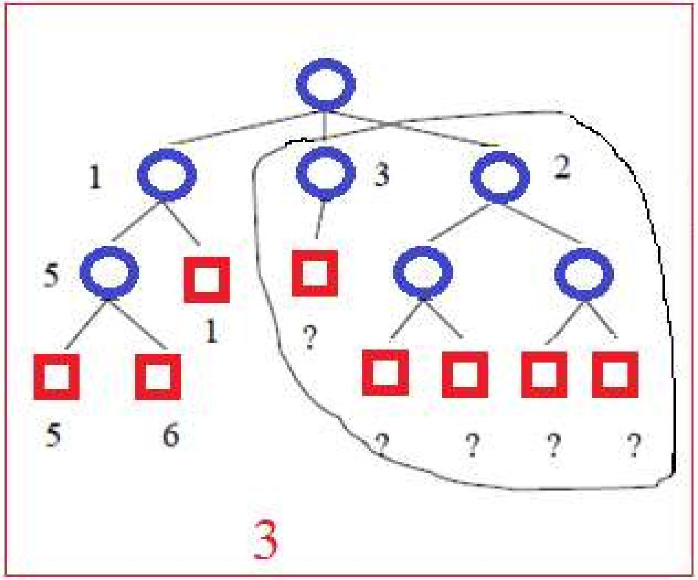
Podgląd w przód (Mini-min i cięcia „alfa”)

1: Przeszukuje liście drzewa i przekazuje zwrotnie do węzłów pośrednich najmniejszą wartość kosztu *f* dla jego następników.

2: Określa najmniejszą wartość „*alfa*” kosztu *f* dla liści aktualnego drzewa.



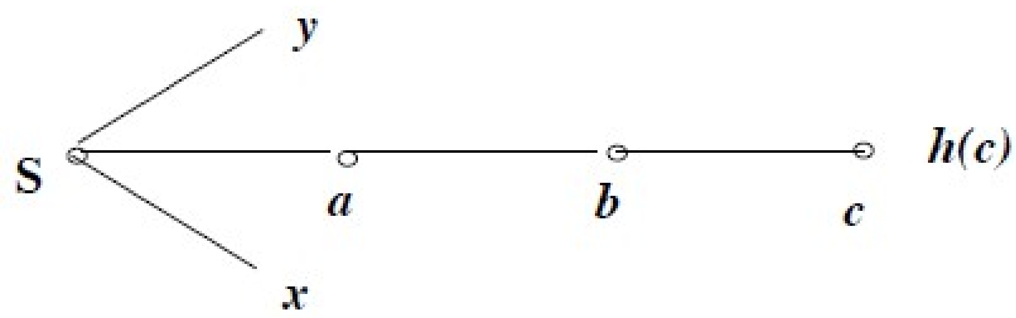




3: Usuwa gałęzie i ich liście o koszcie większym niż *alfa*.

Pojedyncza decyzja

Załóżmy, że agent jest w stanie S.



Po wykonaniu „podglądu ścieżki w przód” (od węzła „*a*” do węzła „c”) możliwa jest zwrotna informacja do węzła-poprzednika „*a*” i zastąpienie dotychczasowego oszacowania jego kosztów resztkowych przez „lepiej poinformowany” – niemniejszy koszt resztkowy, ale nadal dopuszczalny:

$$\mathrm{h}(a) \leq \mathrm{g}(a \rightarrow c) + \mathrm{h}(c) \leq \mathrm{h}^*(a).$$

Podobny podgląd „w przód” wykonywany jest dla pozostałych następników węzła S (tzn. x, y).

Teraz możliwa jest lepiej poinformowana decyzja: „jaki krok wykonać ze stanu S?” : do „y”, „a”, lub „x”.

Pamięć związana ze ścieżką

Zasada: należy wykonać **nawrót** do poprzedniego węzła wtedy, gdy szacowany koszt osiągnięcia celu z tego stanu plus koszt powrotu do niego jest mniejszy niż koszt pójścia „do przodu" z aktualnego stanu.

Zauważmy: funkcja kosztu jest nadal postaci: $\mathrm{f}(n) = \mathrm{c}(n) + \mathrm{h}(n)$, ale teraz koszt $\mathrm{c}(n)$ oznacza jedynie koszt akcji przejścia z aktualnego stanu do następnika (lub rodzica) n.

„Drugi najlepszy": dla uniknięcia pętli – ponownego wyboru już raz wybranego węzła – rozszerzany aktualny węzeł przyjmuje „drugą najlepszą" wartość f spośród swoich następników.

Wykaz: już wybierane węzły są zapisywane w wykazie i przyjmują drugą najlepszą wartość swoich następników. Dla węzłów znajdujących się w wykazie nie jest powtórnie wykonywany „podgląd w przód".

Oryginalna RTA* (wersja „plus”)

Oryginalna wersja RTA* (szybkie „poszukiwanie **rozwiązania**” w trybie „on-line”, optymalizacja **rzeczywiście** wykonywanej ścieżki):

function RTA*(*start*, *goal, limit*): **return** *path*

{ *current* = *start*; *parent* = ∅;

table(*start*) = Minimin_evaluate(*start, limit*);

WHILE *current* != *goal* DO {

best = ∝; *bnode* = *parent*;

FOR each neighbor *next* of *current* DO {

IF *next* == *parent* THEN f = table(*parent*)+**(PLUS)***c(current, parent*);

ELSE { h = Minimin_evaluate(*next, limit*);

f(*next*) = c(*current*, *next*) + h; }

IF f < best THEN { sec_best = best; *bnode* = *next*; best = f; }

}

table(*parent*) = sec_best; *current* = *bnode*;

} **return** Path(*current*);

}

Dyskusja RTA* vs. A*

Oryginalna RTA* nie jest wersją „on-line" strategii A*, gdyż w ogólności jest celem jest **znalezienie rozwiązania** (stanu końcowego w przestrzeni stanów zdefiniowanej dla rozwiązywanego problemu) przy **optymalizacji rzeczywiście** przebytej ścieżki. Tymczasem celem A* jest znalezienie teoretycznie optymalnej ścieżki w przestrzeni stanów.

Jeśli istnieje tylko jeden stan końcowy to w obu strategiach zostanie on znaleziony, ale rzeczywista ścieżka w RTA* nie musi pokrywać się z teoretycznie najlepszą ścieżką w A*.

Zmodyfikujemy strategię RTA* tak, aby działając w trybie on-line wyznaczyła teoretycznie najlepszą ścieżkę, czyli dawała ten sam wynik co A*. Tę nową strategię nazwiemy RTA*(minus), gdyż koszt powrotu do węzła-rodzica odejmujemy od aktualnego kosztu ścieżki.

Modyfikacja RTA* (wersja „minus”)

Modyfikacja RTA* (wersja on-line algorytmu A* - poszukiwanie teoretycznej **optymalnej ścieżki**):

```
function RTA*(start, goal, limit): return path
{ current = start; parent = ∅;  g(current) = 0;
  table(start) = Minimin_evaluate(start, limit);  f(current) = table(start);
  WHILE current != goal  DO {
    best = ∝;      bnode = parent;
    FOR each neighbor next of current  DO {
      IF next==parent THEN
                f = table(parent)+g(current) – (MINUS) c(current, parent);
      ELSE { h = Minimin_evaluate( next, limit );
             f(next) = g(current) + c(current, next) + h; }
       IF f < best THEN { sec_best = best; bnode = next;  best = f; }
    }
    table(parent) = sec_best;       current = bnode;
} return Path(current); }
```

Pytania

1. Przedstawić ideę **poinformowanego** przeszukiwania.
2. Przedstawić strategie przeszukiwania: **zachłanną** i **A*.** Która z nich jest **optymalna** i w jakich warunkach?
3. Co oznaczają**: „dominacja heurystyki”** i **„problem złagodzony”**?
4. Omówić podstawowe **modyfikacje** strategii A*:
   - IDA*,
   - SMA*,
   - Suboptymalne („wynik w dowolnym czasie”),
   - RTA*,
   - RTA* („minus”)