Derinlik Sınırlı Arama

Derinlik öncelikli aramanın derinlik sınırı 1 olan hali.
Yani, derinlik 1'deki düğümler haleflere sahip değildir.
Problem bilgisi kullanılabilir.

Sonsuz yol problemine çözüm getirir.

Sorunlar: Eğer l < d ise (gerekli derinlikten küçükse), sonuçlar eksik olur.Eğer l > d ise (gerekli derinlikten büyükse), optimal olmaz.

Karmaşıklıklar:

Zaman karmaşıklığı: $O(b^l)$

Alan karmaşıklığı: O(bl)

Yinelemeli Derinleşme Araması

```
function ITERATIVE_DEEPENING_SEARCH(problem)
  return a solution or failure

  inputs: problem

  for depth ← 0 to ∞ do
      result ← DEPTH-LIMITED_SEARCH(problem, depth)
      if result ≠ cuttoff then return result
```

IDS: Example

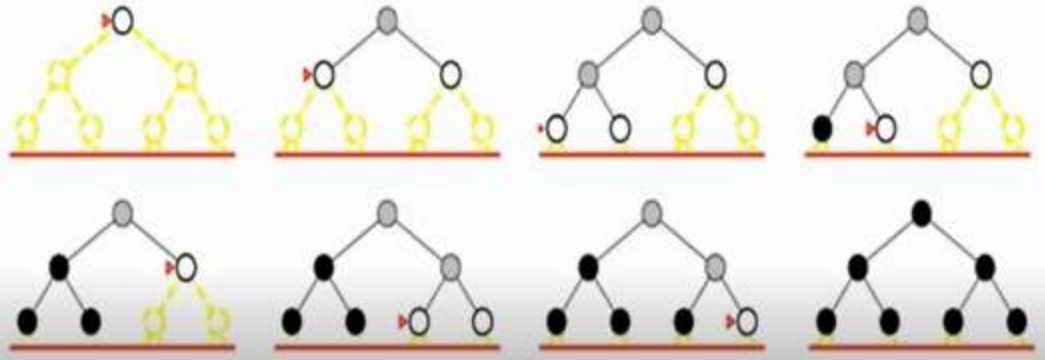
- Limit=0

IDS: Example

- Limit=1

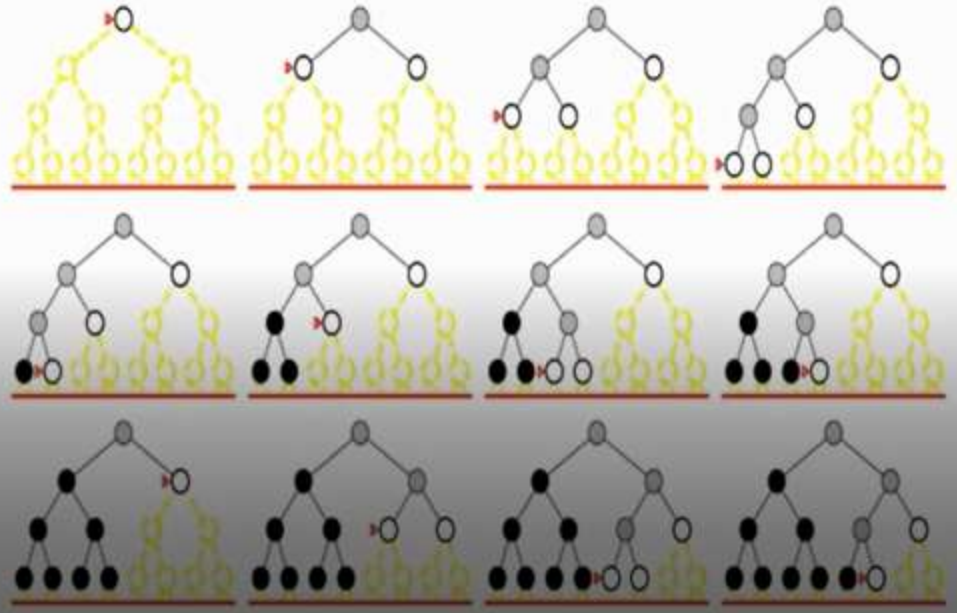
IDS: Example

- Limit=2



IDS: Example

- Limit=3



IDS: Değerlendirme

• Tamlık: EVET (sonsuz yol yok)

• Zaman karmaşıklığı: z Algoritması, belirli durumların tekrar tekrar üretilmesi nedeniyle maliyetli görünüyor.

Düğüm üretimi:

• seviye d: bir kez
• seviye d-1: 2
• seviye d-2: 3
• seviye 2: d-1
• seviye 1: d

$N(IDS) = (d)b + (1-d)\, b^2 + \ldots + (1)b^d$

$N(BFS) = b + b^2 \ldots + b^d + (b^{d+1} - b)$

Num. Karşılaştırma için b=10 , d=5

N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123450

N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100

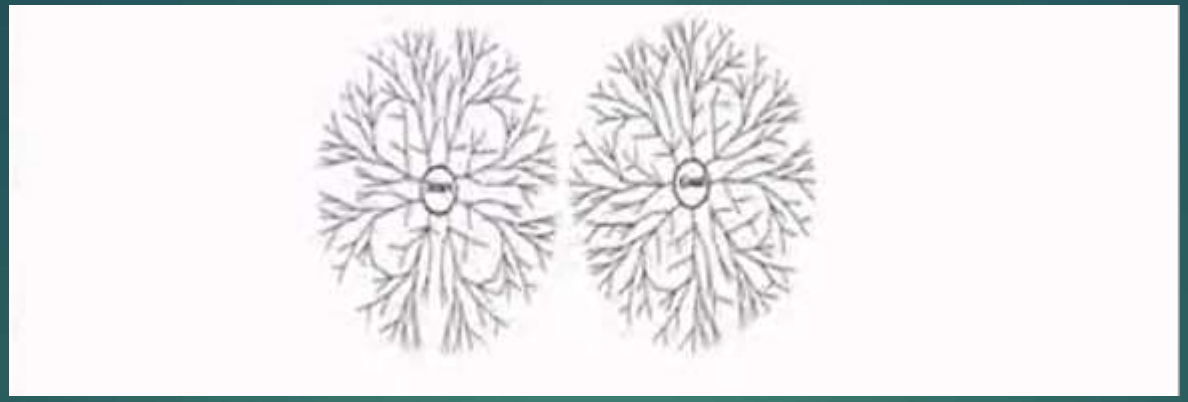
İteratif Derinlik Arama (IDS): Değerlendirme

•Tamlık:
- EVET (Sonsuz yol yok)

•Zaman Karmaşıklığı:$O(b^d)$
•Alan Karmaşıklığı:O(bd)
•Optimalite:
- EVET eğer adım maliyeti sabitse (örneğin, maliyet = 1).

Birim Maliyet Araması ile Karşılaştırma
•IDS, birim maliyet aramasının aynı ilkelerini kullanarak yinelemeli uzunluk artırma araması olarak genişletilebilir.

İki Yönlü Arama (Bidirectional Search)

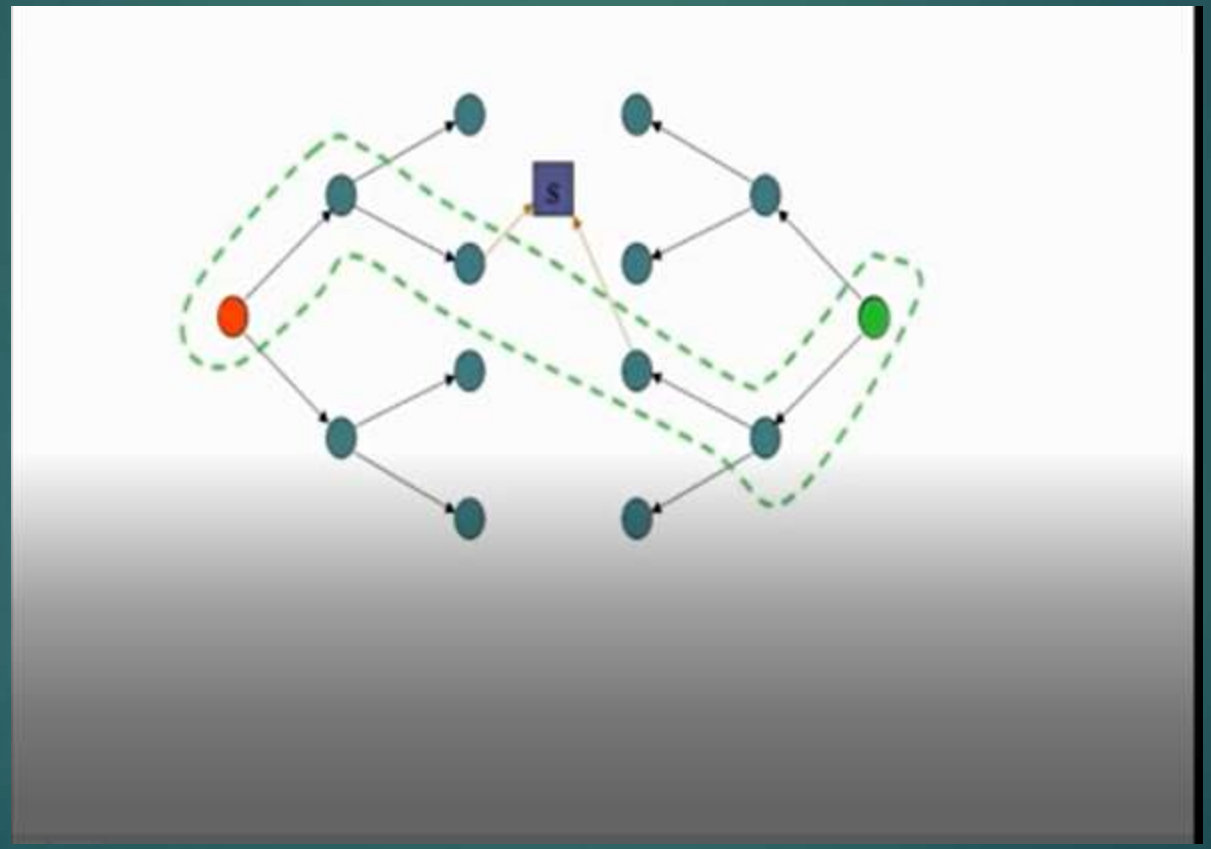


•Başlangıç ve hedeften iki eş zamanlı arama yapılır. Motivasyon:

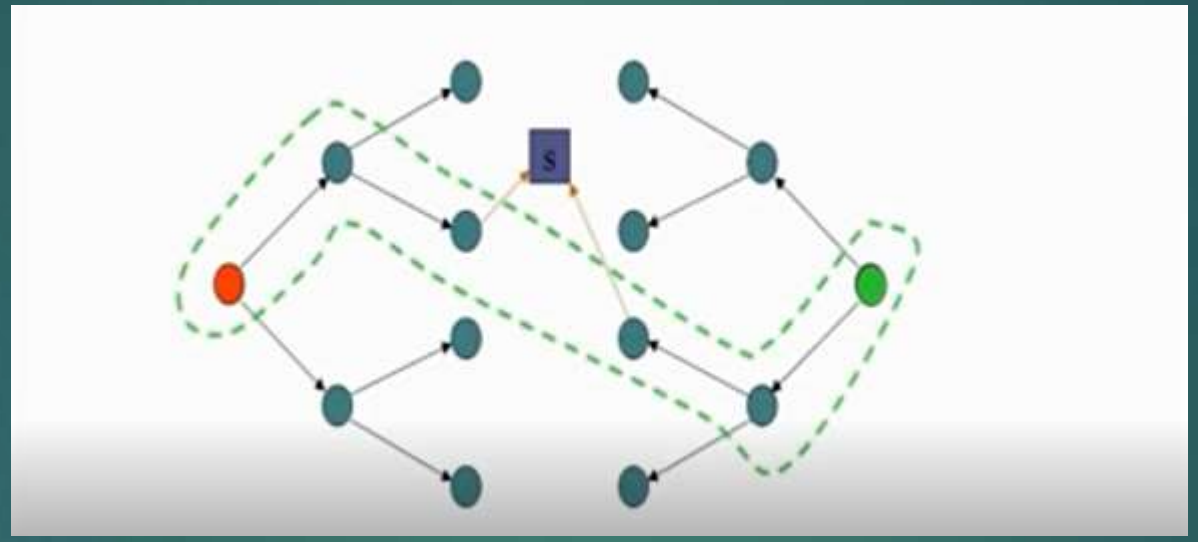
Motivation: $b^{d/2} + b^{d/2} \neq b^{d}$

- Düğüm genişlemeden önce diğer Fring e ait olup olmadığını kontrol edin.
- Alan karmaşıklığı bir dezavantajdır.
- Her iki arama da tamamlayıcı ve optimaldir eğer her ikisi de BFS ise.

İki Yönlü Arama (Bidirectional Search)



İki Yönlü Arama (Bidirectional Search)



Her düğümün öncülü verimli bir şekilde hesaplanabilir olmalıdır. Eylemler kolayca tersine çevrilebiliyorsa..

Tersine çevrilebilir problemlere basit örnekler:

8-Puzzle (Sekizli Bulmaca)
•“Yukarı” hareketinin tersi “Aşağı”dır.
•“Sol” hareketinin tersi “Sağ”dır.
Bu nedenle bu bulmaca tersine çevrilebilir bir problemdir.

Harita veya şehir ağı üzerinde hareket
•Eğer A şehrinden B şehrine bir yol varsa ve aynı yol B’den A’ya dönüşte de kullanılabiliyorsa,
bu yol tersine çevrilebilir bir yoldur.

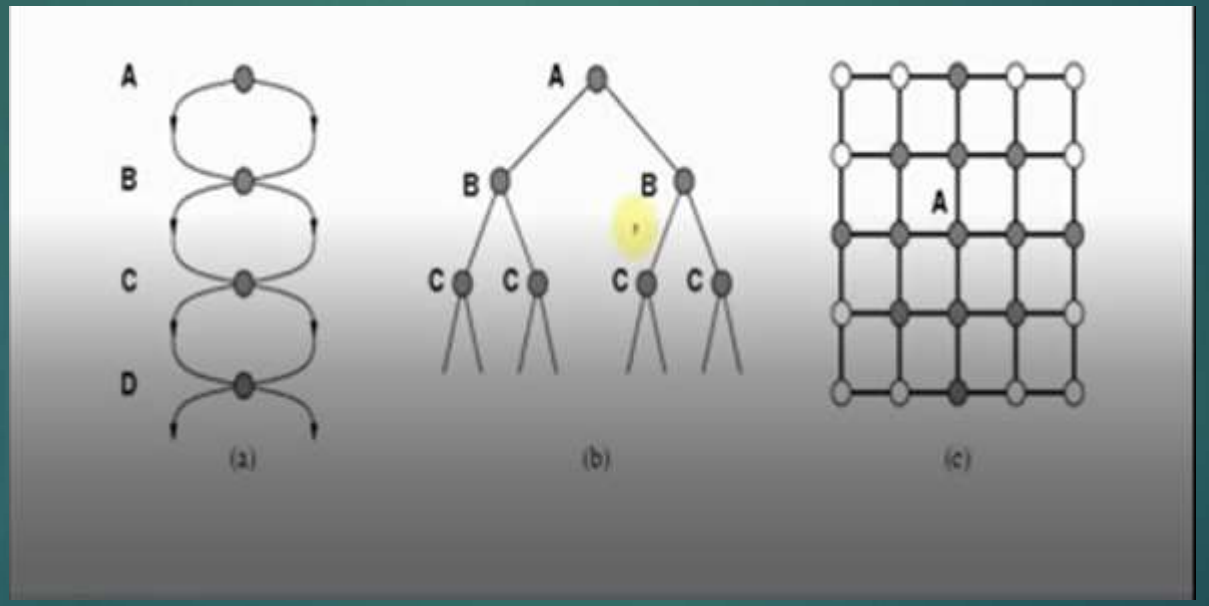
Engelsiz bir ortamda robotun hareketi
•Eğer robot bir adım ileri gidebiliyorsa ve aynı yolu kullanarak geri dönebiliyorsa,
bu problem tersine çevrilebilir bir problemdir.

Summary of algorithms

Criterion	Breadth-First	Uniform-cost	Depth-First	Depth-limited	Iterative deepening	Bidirectional search
Complete?	YES*	YES*	NO	YES, if $l \geq d$	YES	YES*
Time	b^{d+1}	$b^{C^*/\epsilon}$	b^m	b^l	b^d	$b^{d/2}$
Space	b^{d+1}	$b^{C^*/\epsilon}$	bm	bl	bd	$b^{d/2}$
Optimal?	YES*	YES*	NO	NO	YES	YES

Tekrar Eden Durumlar

•Tekrar eden durumların tespit edilmemesi, çözülebilir problemleri çözülemez hale getirebilir.



- Tekrar eden durumlar, arama algoritmalarında önemli bir sorundur çünkü bu durumların tespit edilememesi, çözülebilir problemlerin çözülemez hale gelmesine neden olabilir.

- Örneğin, bir labirentte kaybolduğunuzu düşünün; eğer algoritma, gittiğiniz yolları kaydetmiyorsa, sürekli olarak aynı yollara geri dönerek çıkışı bulamaz.

- Bu durum, gereksiz hesaplamalara yol açarak zaman kaybına neden olur. Tekrar eden durumların önlenmesi için daha önce ziyaret edilen durumları saklamak amacıyla durum kaydı gibi yöntemler kullanılabilir.

- Böylece, arama algoritmalarının etkinliği artırılır ve daha hızlı sonuçlar elde edilir, özellikle büyük arama alanlarında ve karmaşık problemlerde bu yaklaşım büyük önem taşır.

Graf Arama Algoritması

•Kapalı liste, tüm genişletilmiş düğümleri saklar.

```
function GRAPH-SEARCH(problem,fringe) return a solution or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if GOAL-TEST[problem] applied to STATE[node] succeeds
            then return SOLUTION(node)
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

Graf Arama Algoritması: Değerlendirme

•Optimalite: GRAF ARAMA, yeni keşfedilen yolları reddeder. Bu durum, alt-optimal bir çözüme yol açabilir.

Yine de, sabit adım maliyeti ile birim maliyet araması veya genişleme (BF-search) kullanıldığında bu geçerli olabilir.

•Zaman ve alan karmaşıklığı, durum alanının boyutuna orantılıdır ($O(b^d)$dan çok daha küçük olabilir).

•DFS ve IDS arama yöntemlerinde, kapalı liste artık doğrusal alan gereksinimlerine sahip değildir, çünkü tüm düğümler kapalı listede saklanır.

Ağaç Arama Algoritması

```
function TREE-SEARCH(problem,fringe) return a solution or
  failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
      if EMPTY?(fringe) then return failure
      node ← REMOVE-FIRST(fringe)
      if GOAL-TEST[problem] applied to STATE[node] succeeds
          then return SOLUTION(node)
      fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

Fringe, her zaman bizim stratejilerimizi belirler.

Özellik	Ağaç Arama (Tree Search)	Grafik Arama (Graph Search)
Yapı	Döngü veya tekrar yokmuş gibi varsayar	Döngü ve tekrarlar olabilir
Bellek	Genellikle sadece mevcut yolu tutar	Daha önce ziyaret edilen düğümleri kontrol etmek için **kapalı liste (closed list)** gerekir
Zaman	Daha hızlı ve basittir	Daha uzun sürebilir çünkü her düğümün daha önce açılıp açılmadığını kontrol eder
Kullanım	Küçük ve döngüsüz durum alanları	Gerçek grafikler, tekrar eden yollar veya döngüler

En iyi-Öncelikli Arama

Bilgili aramanın genel yaklaşımı:

En iyi-öncelikli arama: bir düğüm, bir değerlendirme işlevine f(n) dayalı olarak genişletilmek üzere seçilir

Değerlendirme işlevi: uygunluğun tahmini

Genellikle tahmini maliyet veya mesafedir

En iyi görünen düğümü seç

Uygulama: kuyruk, azalan uygunluk sırasına göre sıralanır

Özel durumlar: açgözlü arama, A* araması

Açgözlü En İyi-Öncelikli Arama

• Değerlendirme işlevi f(n) = h(n)

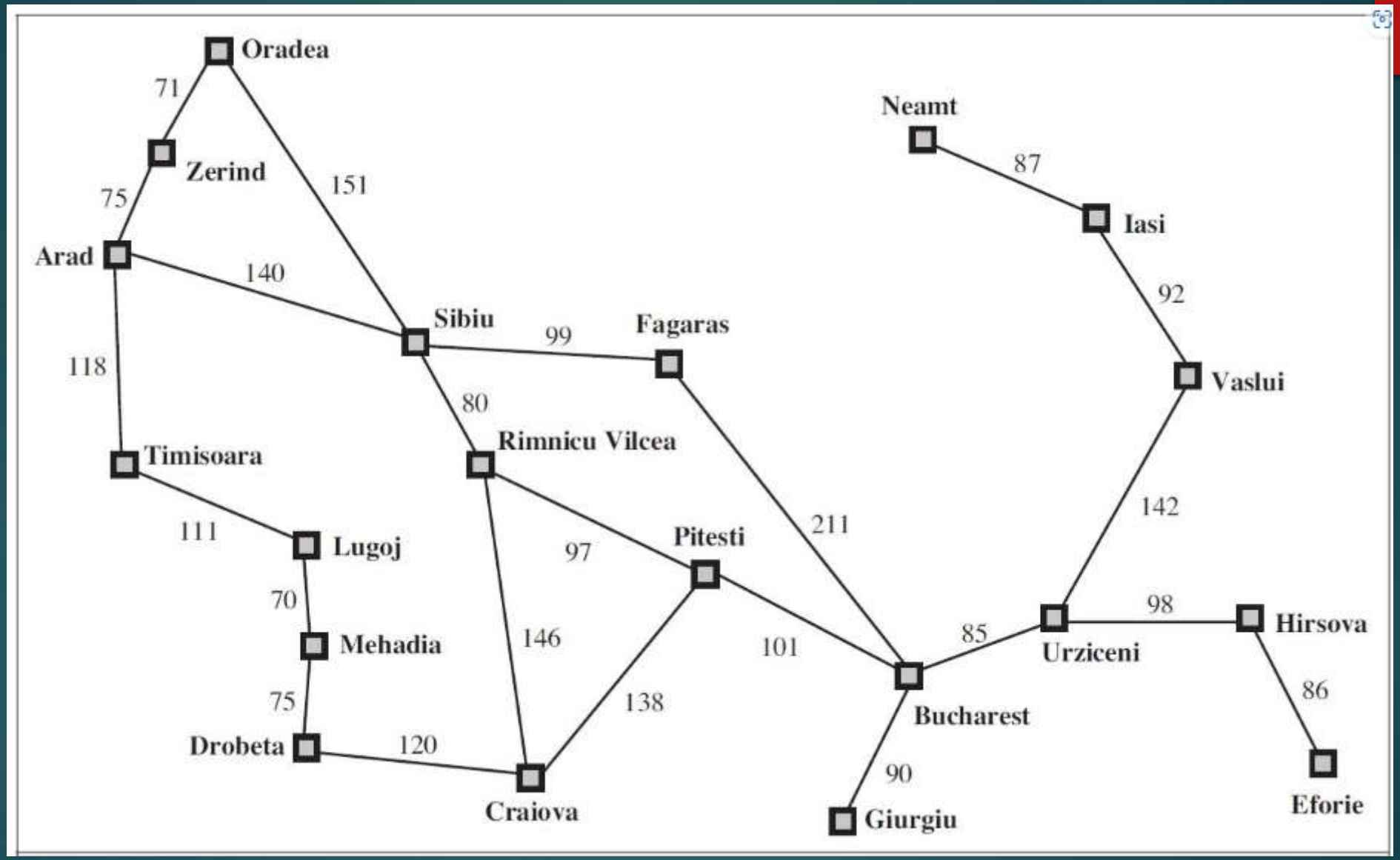
h(n): sezgisel işlev

h(n): düğüm n'den hedef düğüme en ucuz yolun tahmini maliyeti

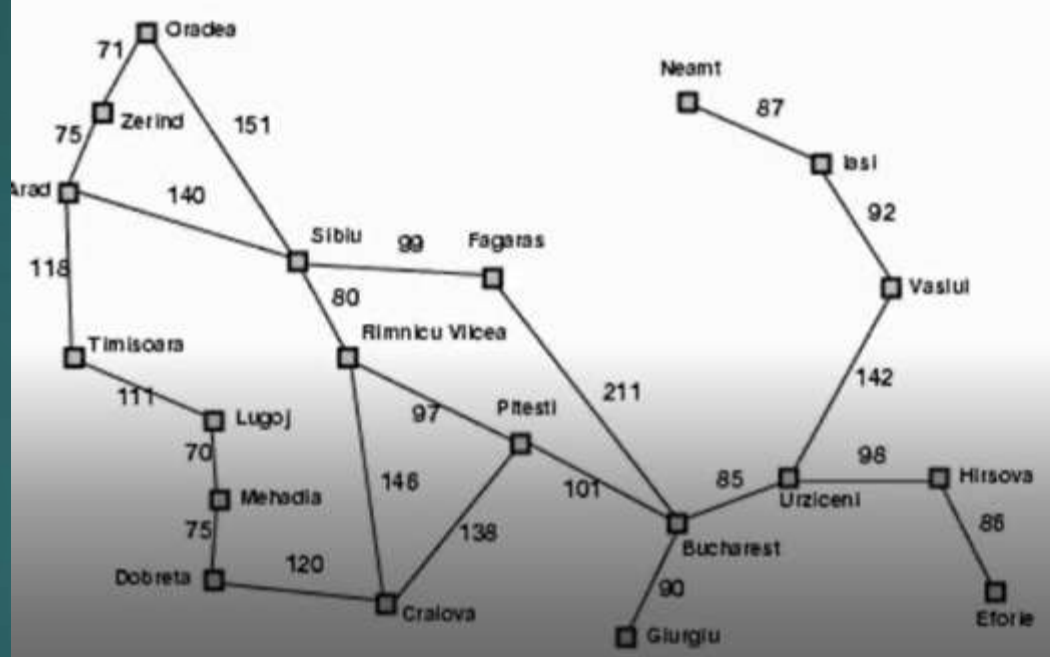
Eğer n hedefse, h(n) = 0

Açgözlü en iyi-öncelikli arama, hedefe en yakın görünen düğümü genişletir.

Örnek: h_{sld}(n) = n'den Bükreş'e olan düz çizgi mesafesi


Oradea
71
Zerind
75
151
Arad
140
Sibiu
99
Fagaras
118
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Drobeta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
Hirsova
85
Urziceni
86
Bucharest
90
Giurgiu
Eforie

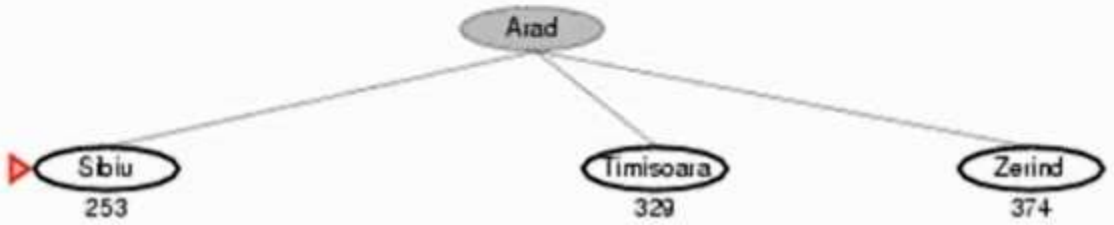
Example: Romania



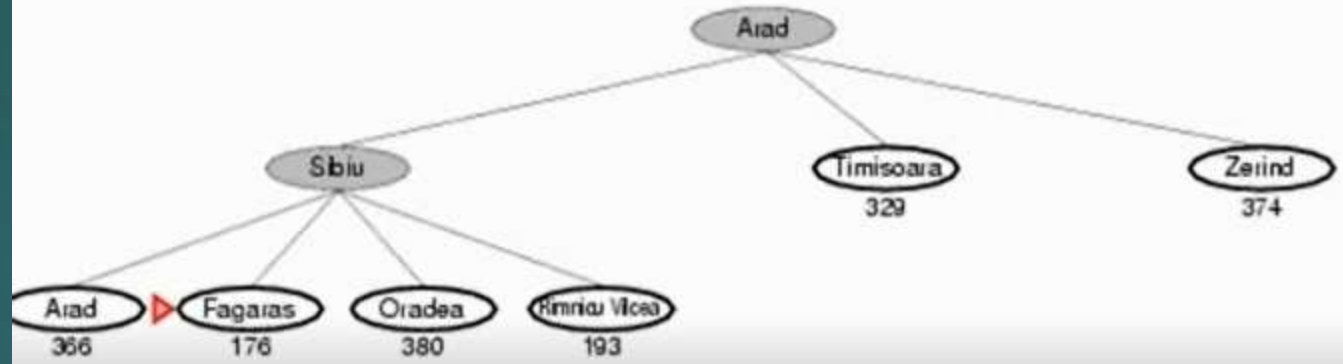
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search: Example

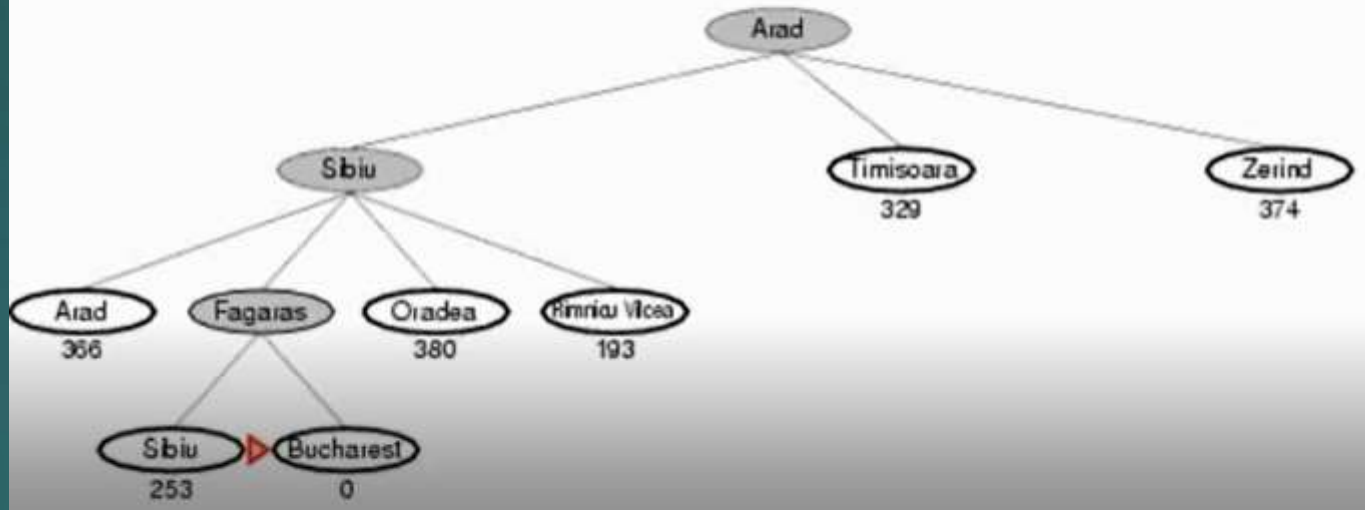
Greedy Search: Example



Greedy Search: Example

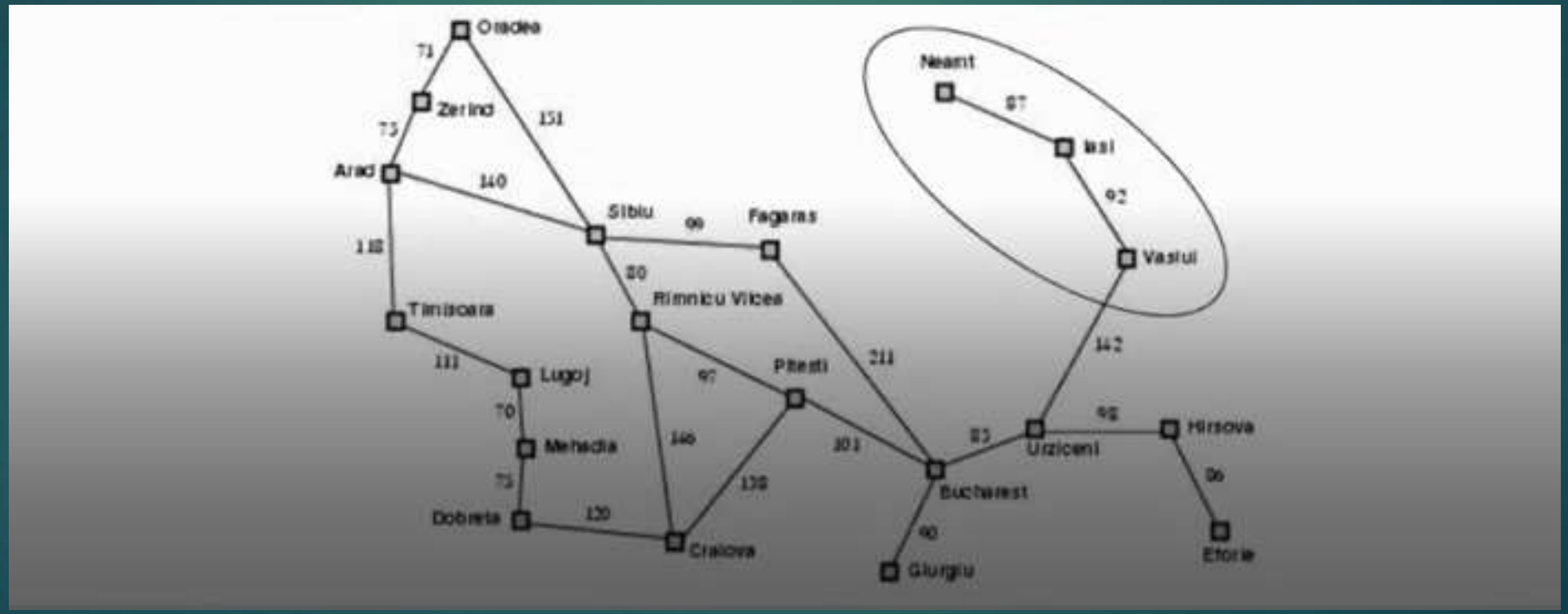


Greedy Search: Example



Açgözlü Arama: Değerlendirme

• Tamlık: HAYIR (bkz. Derinlik-Öncelikli Arama)
• Tekrarlanan durumlar üzerinde kontrol
• h(n)'yi minimize etmek yanlış başlangıçlara neden olabilir, örneğin Iasi'den Fagaras'a.



Açgözlü Arama: Değerlendirme

Tamlık: HAYIR (bkz. Derinlik-Öncelikli Arama)

Zaman karmaşıklığı: $O(b^m)$

Derinlik-Öncelikli Arama'nın en kötü durumu ile karşılaştırabilirz (burada m, arama alanındaki maksimum derinliktir)

Alan karmaşıklığı: $O(b^m)$
Tüm düğümleri bellekte tutar.

Optimalite: HAYIR
Derinlik-Öncelikli Arama ile aynı.

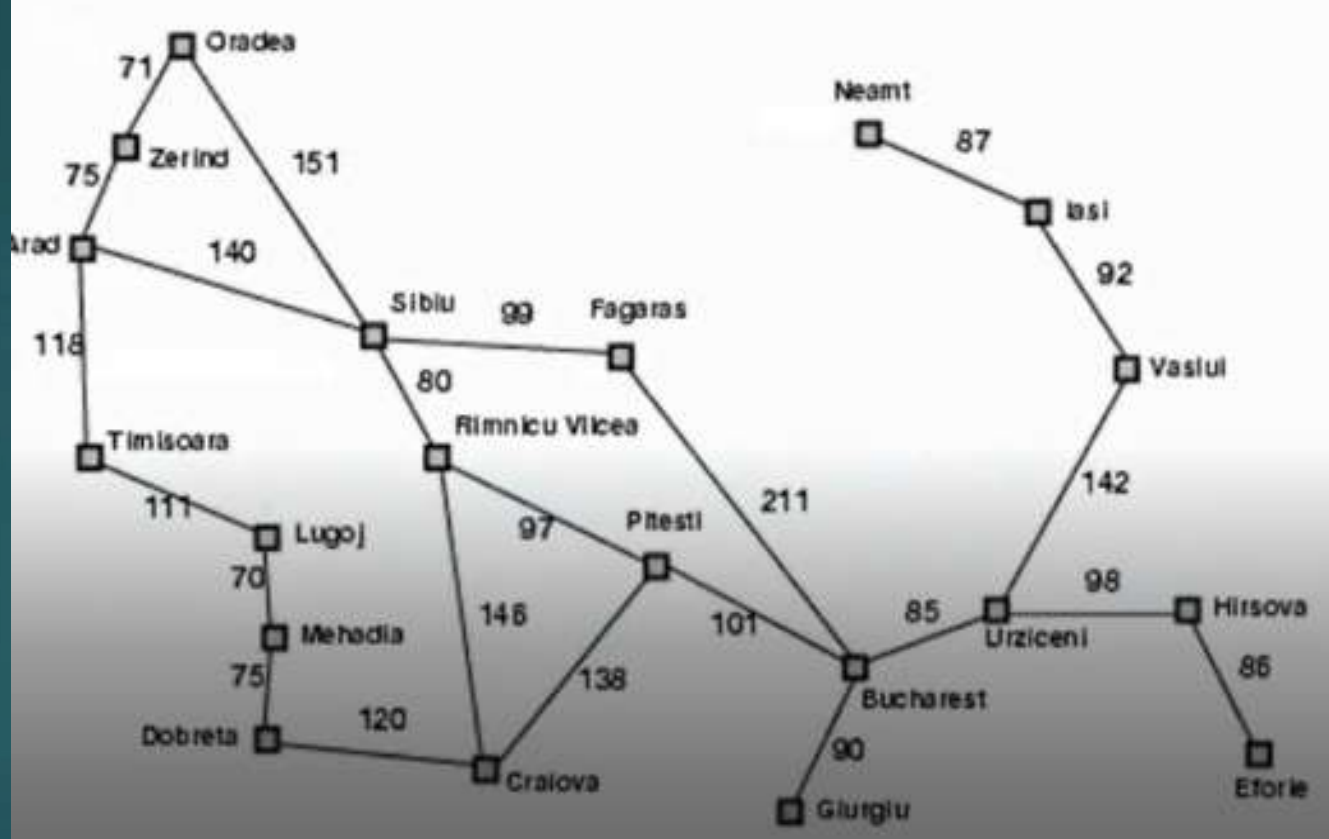
A* Arama Algoritması

- En iyi bilinen en iyi-öncelikli arama biçimi.
- Fikir: Pahalı olan yolları genişletmekten kaçınmak.

Değerlendirme işlevi: f(n) = g(n) + h(n)

- g(n): Düğüme ulaşma maliyeti (şu ana kadar).
- h(n): Düğümden hedefe gitmenin tahmini maliyeti.
- f(n): Düğümden geçerek hedefe ulaşmanın tahmini toplam maliyeti.

Example: Romania



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

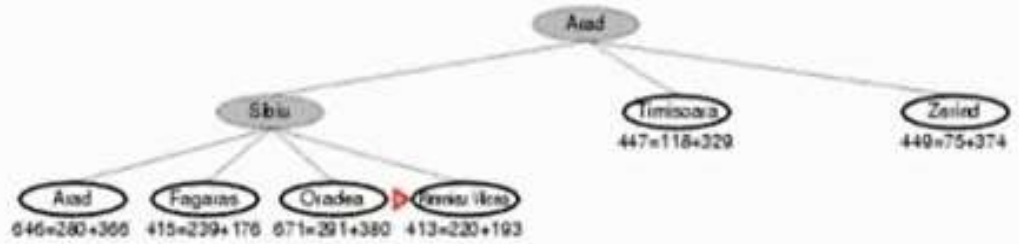
A* search: Example

- Find Bucharest starting at Arad
  - f(Arad) = c(??,Arad)+h(Arad)=0+366=366

A* search: Example

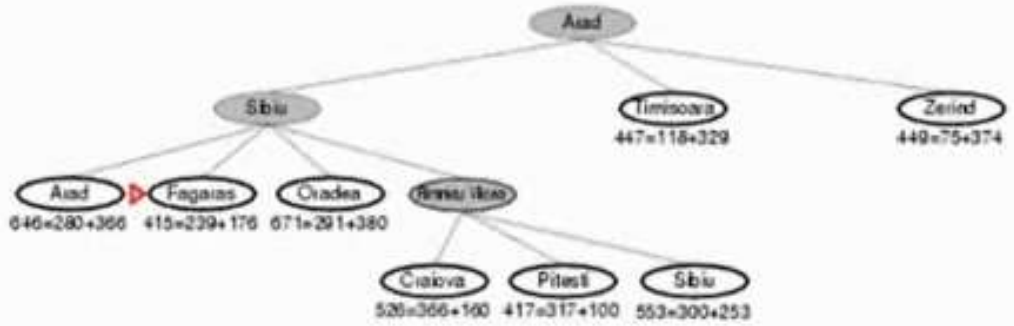
- Expand Arrad and determine *f(n)* for each node
  - f(Sibiu)=c(Arad,Sibiu)+h(Sibiu)=140+253=393
  - f(Timisoara)=c(Arad,Timisoara)+h(Timisoara)=118+329=447
  - f(Zerind)=c(Arad,Zerind)+h(Zerind)=75+374=449
- Best choice is Sibiu

A* search: Example



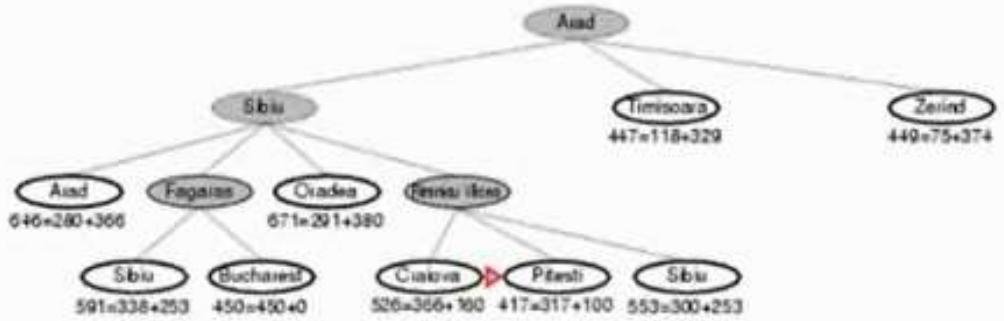
- Expand Sibiu and determine *f(n)* for each node
  - f(Arad)=c(Sibiu,Arad)+h(Arad)=280+366=646
  - f(Fagaras)=c(Sibiu,Fagaras)+h(Fagaras)=239+176=415
  - f(Oradea)=c(Sibiu,Oradea)+h(Oradea)=291+380=671
  - f(Rimnicu Vilcea)=c(Sibiu,Rimnicu Vilcea)+h(Rimnicu Vilcea)=220+192=413
- Best choice is Rimnicu Vilcea

A* search: Example



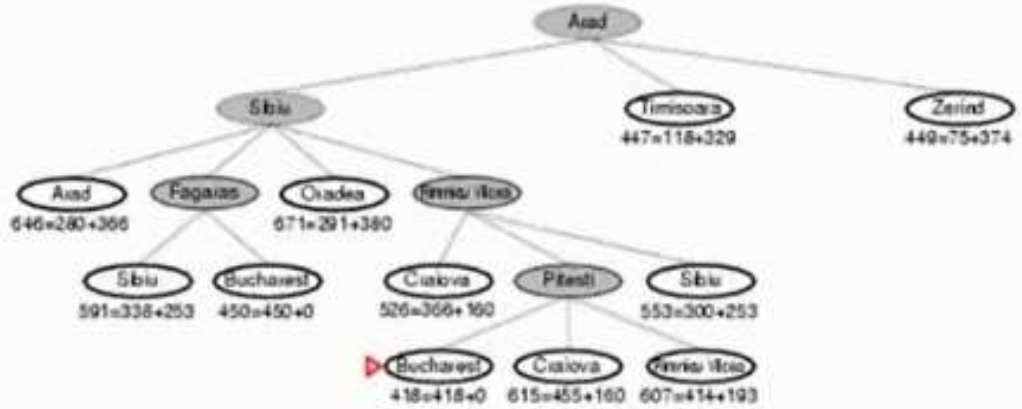
- Expand Rimnicu Vilcea and determine *f(n)* for each node
  - f(Craiova)=c(Rimnicu Vilcea, Craiova)+h(Craiova)=360+160=526
  - f(Pitesti)=c(Rimnicu Vilcea, Pitesti)+h(Pitesti)=317+100=417
  - f(Sibiu)=c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553
- Best choice is Fagaras

A* search: Example



- Expand Fagaras and determine *f(n)* for each node
  - f(Sibiu)=c(Fagaras, Sibiu)+h(Sibiu)=338+253=591
  - *f(Bucharest)=c(Fagaras,Bucharest)+h(Bucharest)=450+0=450*
- Best choice is Pitesti !!!

A* search: Example



- Expand Pitesti and determine *f(n)* for each node
  - *f(Bucharest)=c(Pitesti,Bucharest)+h(Bucharest)=418+0=418*
- Best choice is Bucharest !!!
  - Optimal solution?
  - Yes, only if *h(n)* is admissible

Kabul Edilebilir Sezgiler

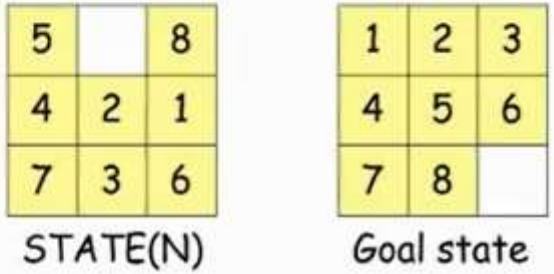
•Bir sezgi **h(n)**, her düğüm **n** için $h(n) \leq h^*(n)$ koşulunu sağlıyorsa kabul edilebilir (admissible) olarak tanımlanır; burada h^*(n), düğüm **n**'den hedef duruma ulaşmanın gerçek maliyetidir.

•Kabul edilebilir bir sezgi, hedefe ulaşmanın maliyetini asla abartmaz; yani, iyimserdir.

•Örnek: $\mathbf{h_{sld}(n)}$, gerçek yol mesafesini asla abartmayan bir sezgidir.

•Teorem: Eğer h(n) kabul edilebilir (admissible) bir sezgi ise, A* algoritması ağaç araması (tree-search) kullanarak optimaldir.

Another Example: 8 puzzle



- $h_1(N)$ = number of misplaced numbered tiles = 6
- $h_2(N)$ = sum of the (Manhattan) distance of every numbered tile to its goal position

  = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13

1. Manhattan Mesafesi,

her taşın mevcut konumundan hedef konumuna ulaşmak için gereken en kısa yolun toplam uzunluğunu ölçer. Her taş için, yatay ve dikey olarak hedef konumuna olan mesafe hesaplanır ve bu mesafelerin toplamı, Manhattan mesafesi olarak tanımlanır.

$$h(n) = \sum_{i=1}^{8} (|x_i - x_i'| + |y_i - y_i'|)$$

Burada:

(x_i, y_i): Taşın mevcut konumu

(x'_i, y'_i): Taşın hedef konumu

2. Yerinde Taş Sayısı (Misplaced Tiles)

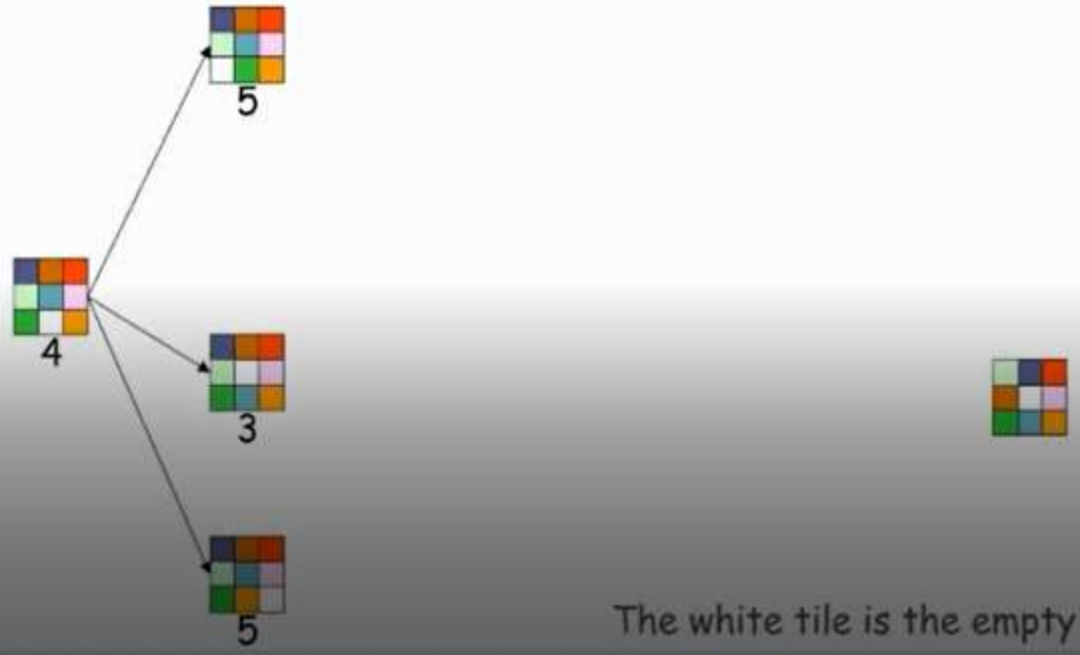
Yerinde taş sayısı, mevcut durumdaki taşların hedef durumdaki konumları ile eşleşmeyen sayısını sayar. Bu sezgisel, taşların yanlış yerleştirildiği durumları doğrudan sayarak bulmacanın çözümüne ne kadar uzak olduğuna dair bir tahmin verir.

h(n)=(Yanlıs¸ yerles¸tirilmis¸ tas¸ sayısı)

İki sezgisel yöntem de, A* arama algoritması gibi durum arama algoritmalarında kullanılabilir ve bulmacayı çözmek için etkili bir yol sağlar; ancak birisi diğerinden daha iyi etkilidir.

Another Example: 8 puzzle

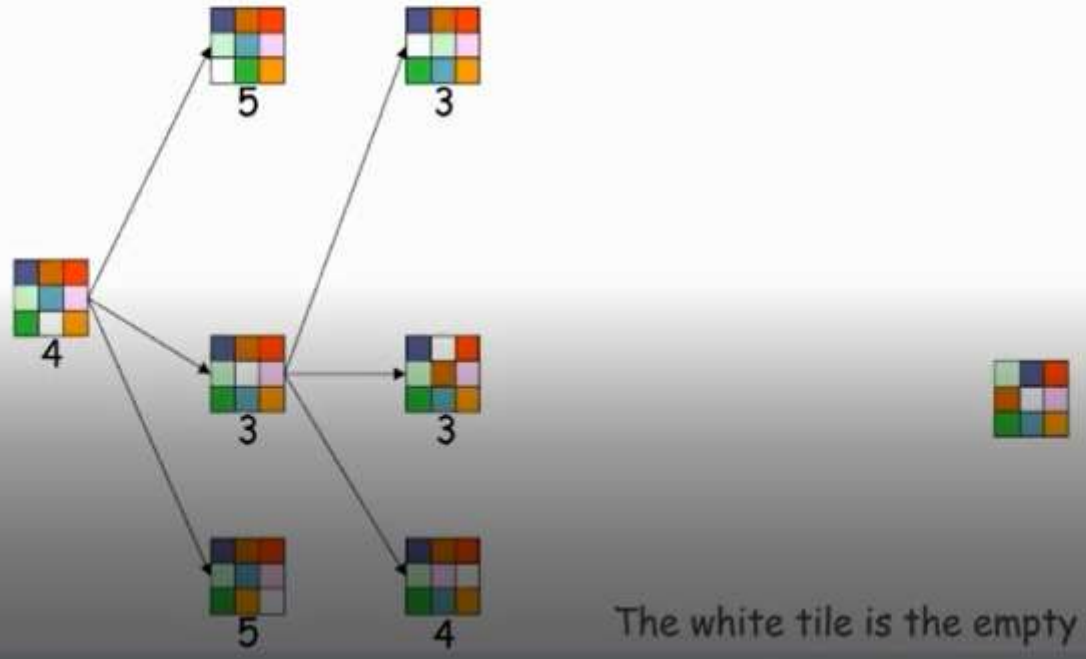
f(n) = h(n) = number of misplaced numbered tiles



The white tile is the empty tile

Another Example: 8 puzzle

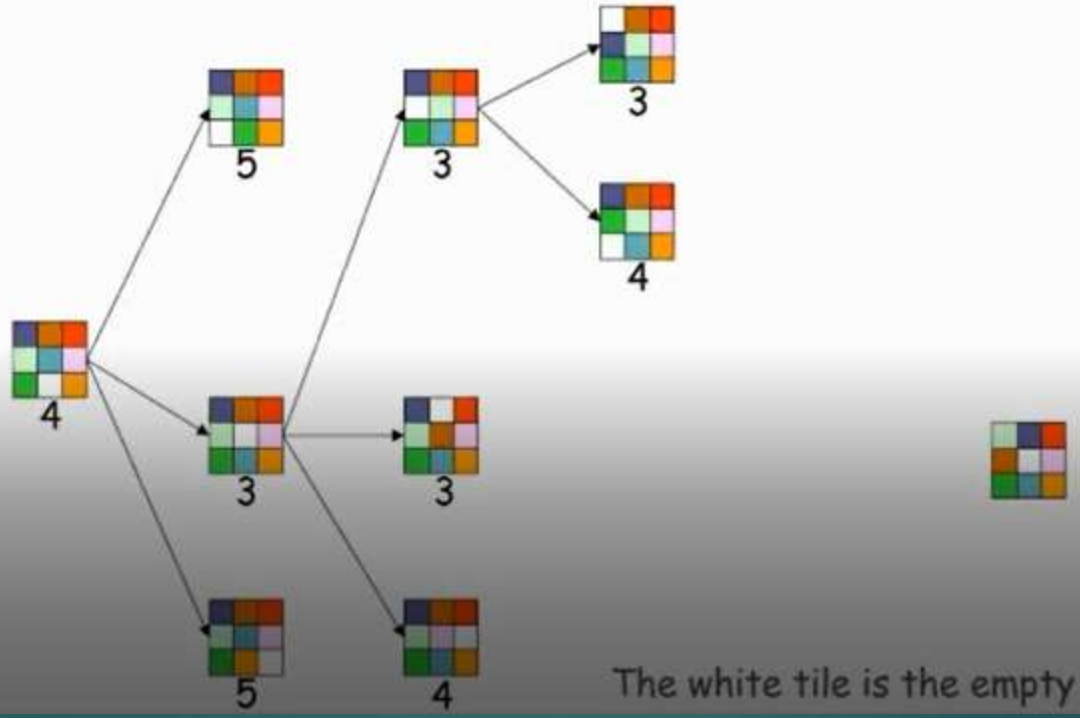
f(n) = h(n) = number of misplaced numbered tiles



The white tile is the empty tile

Another Example: 8 puzzle

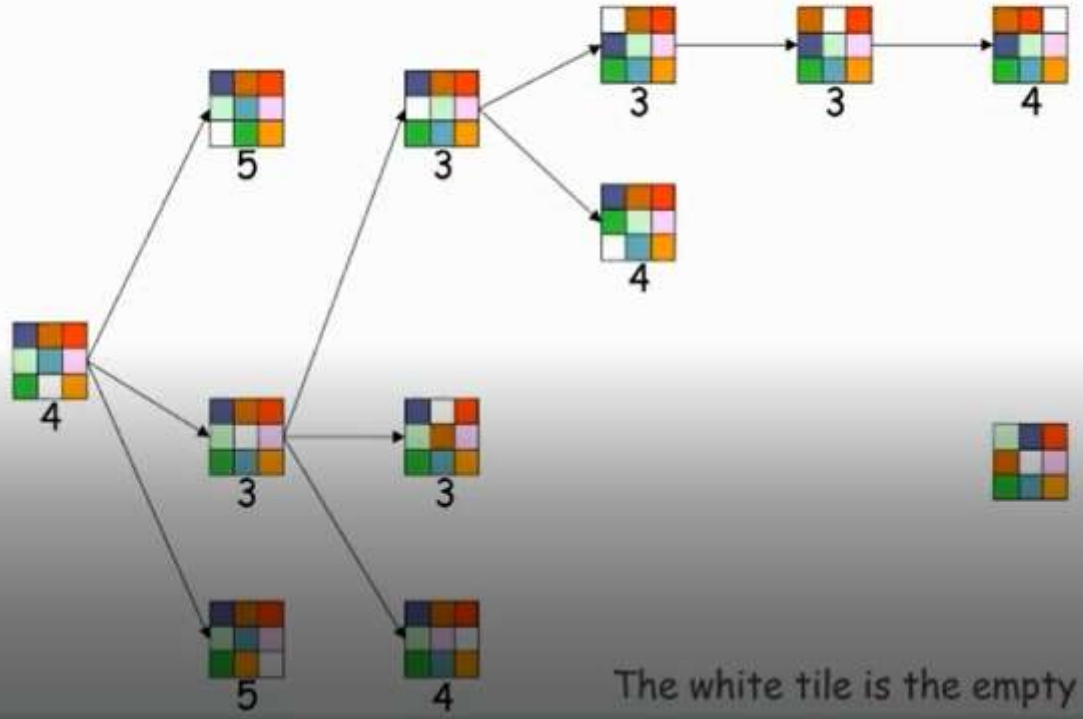
f(n) = h(n) = number of misplaced numbered tiles



The white tile is the empty tile

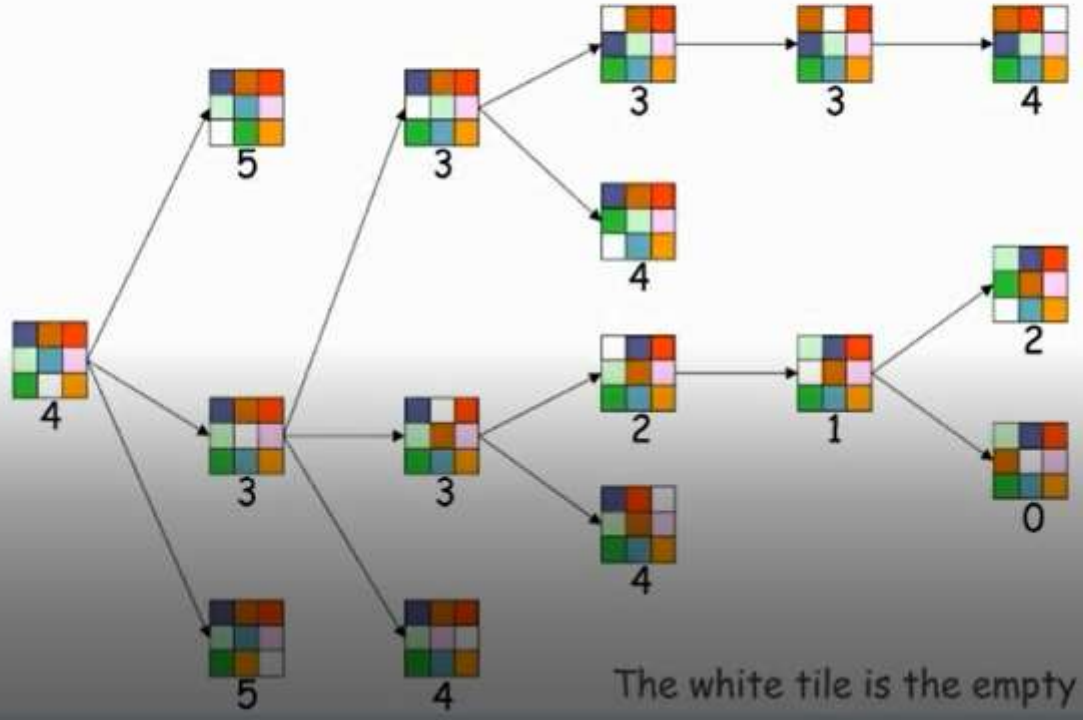
Another Example: 8 puzzle

f(n) = h(n) = number of misplaced numbered tiles



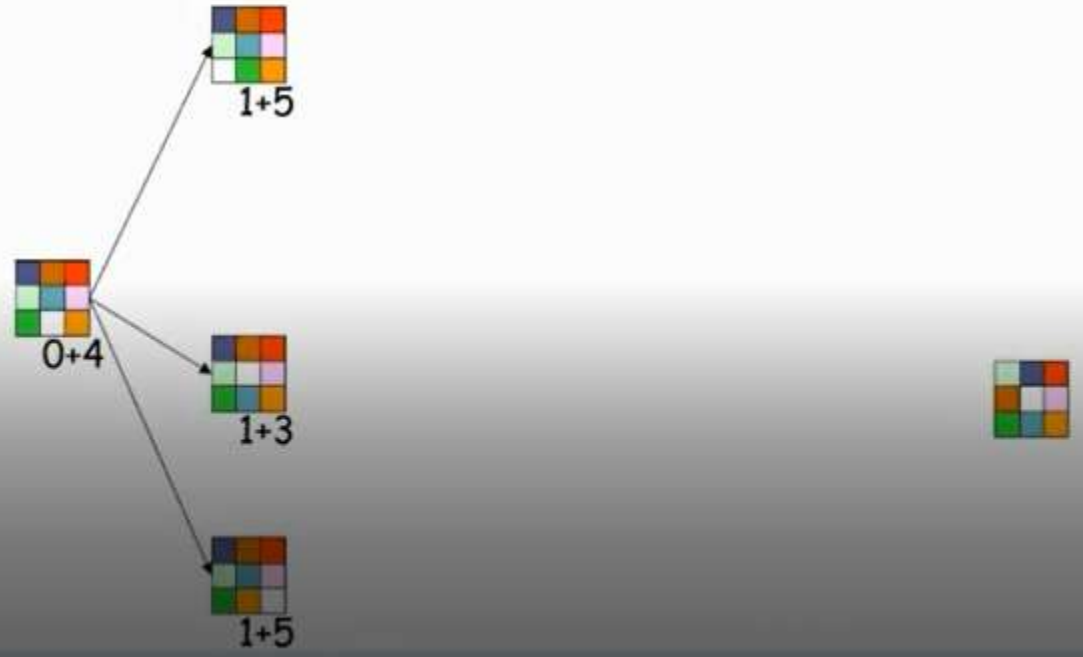
Another Example: 8 puzzle

f(n) = h(n) = number of misplaced numbered tiles



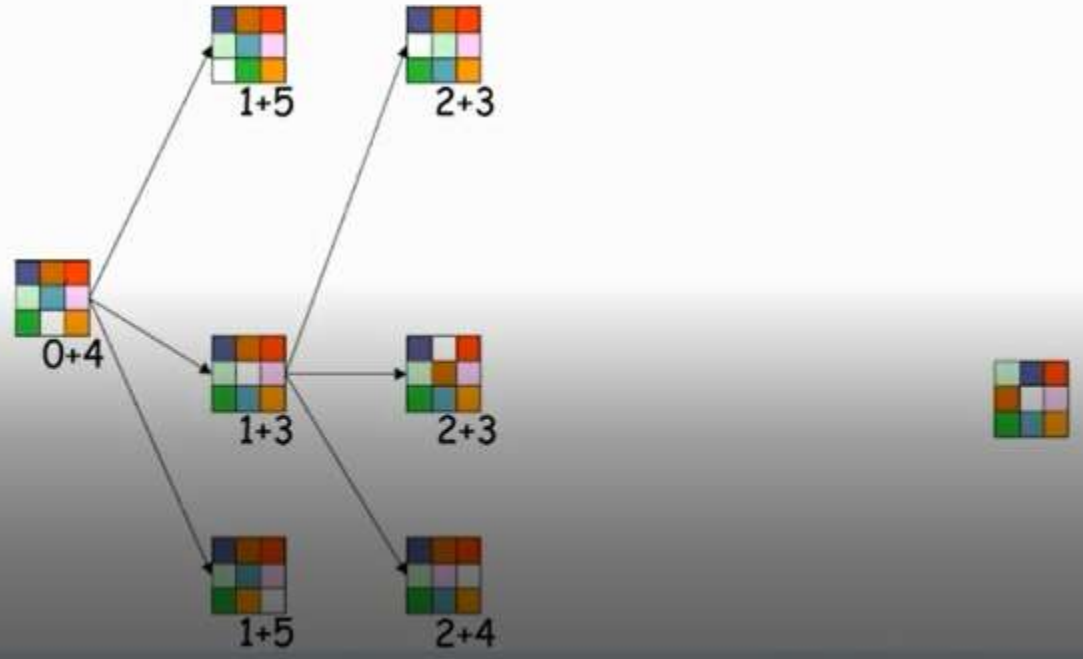
Another Example: 8 puzzle

f(n) = g(n) + h(n) with h(n) = number of misplaced numbered tiles



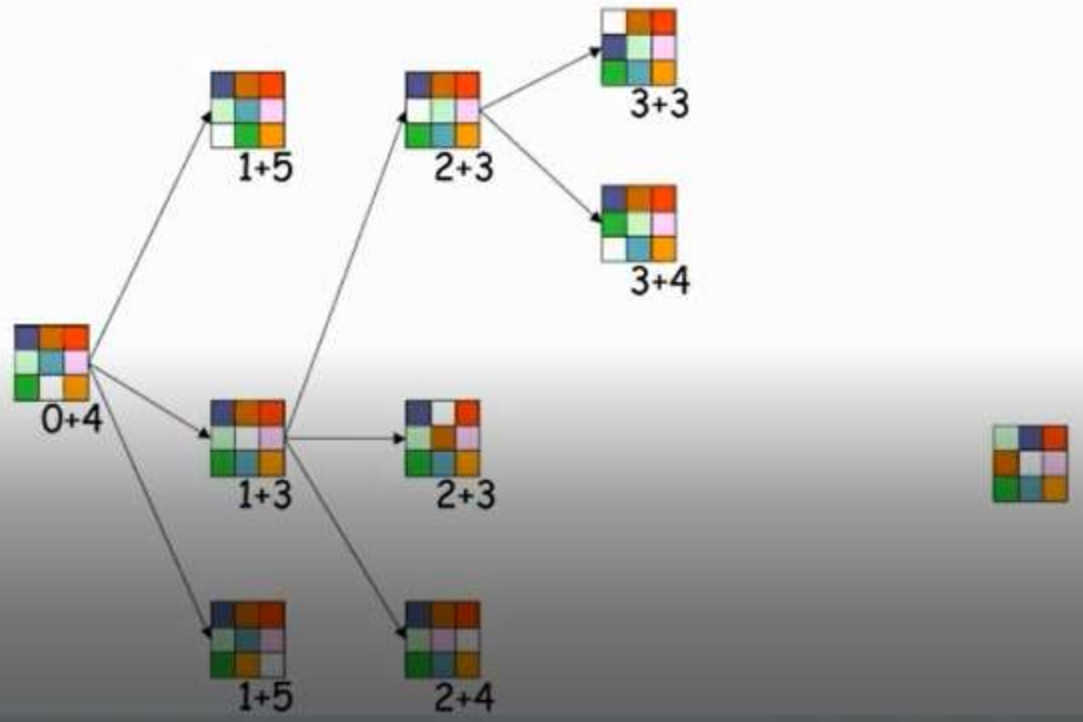
Another Example: 8 puzzle

f(n) = g(n) + h(n) with h(n) = number of misplaced numbered tiles



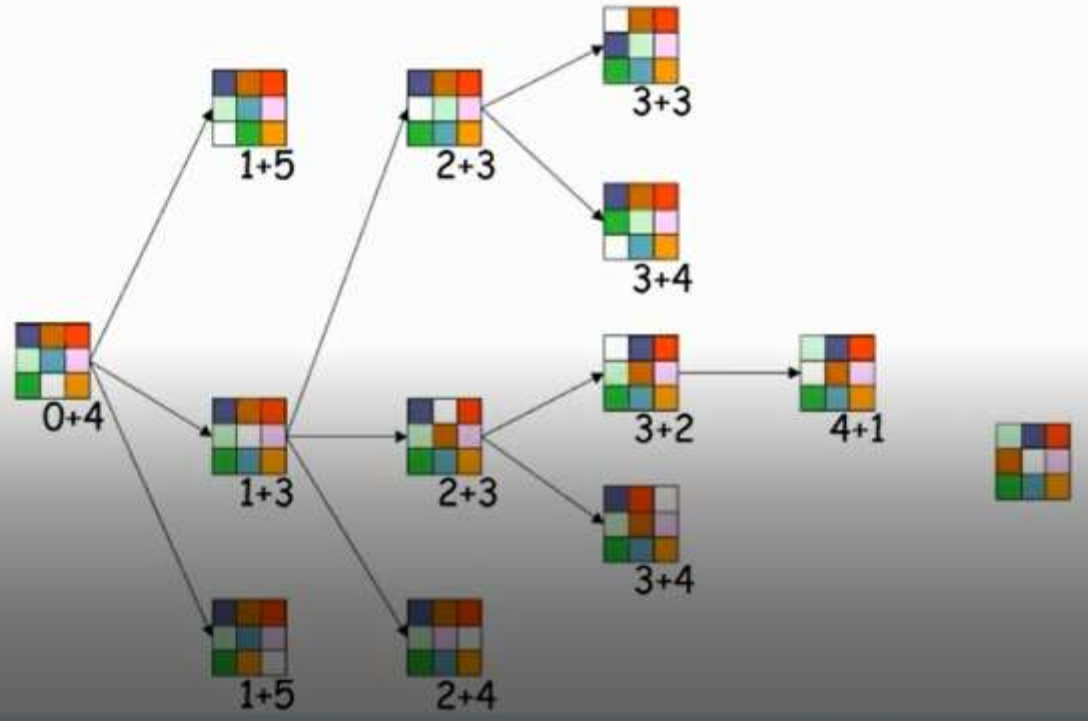
Another Example: 8 puzzle

f(n) = g(n) + h(n) with h(n) = number of misplaced numbered tiles



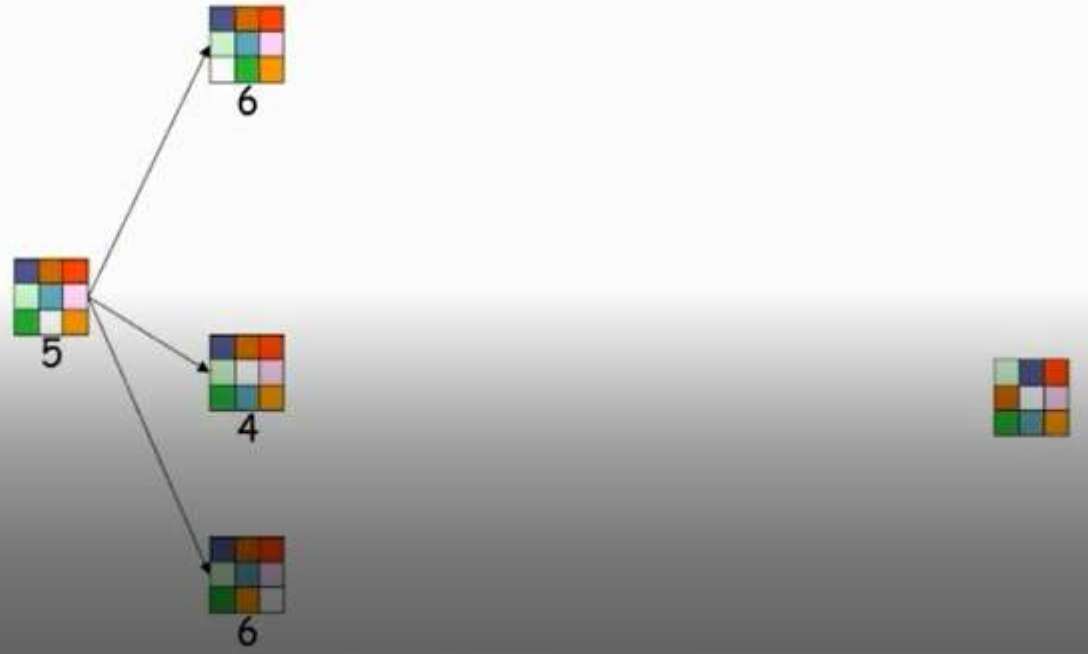
Another Example: 8 puzzle

f(n) = g(n) + h(n) with h(n) = number of misplaced numbered tiles



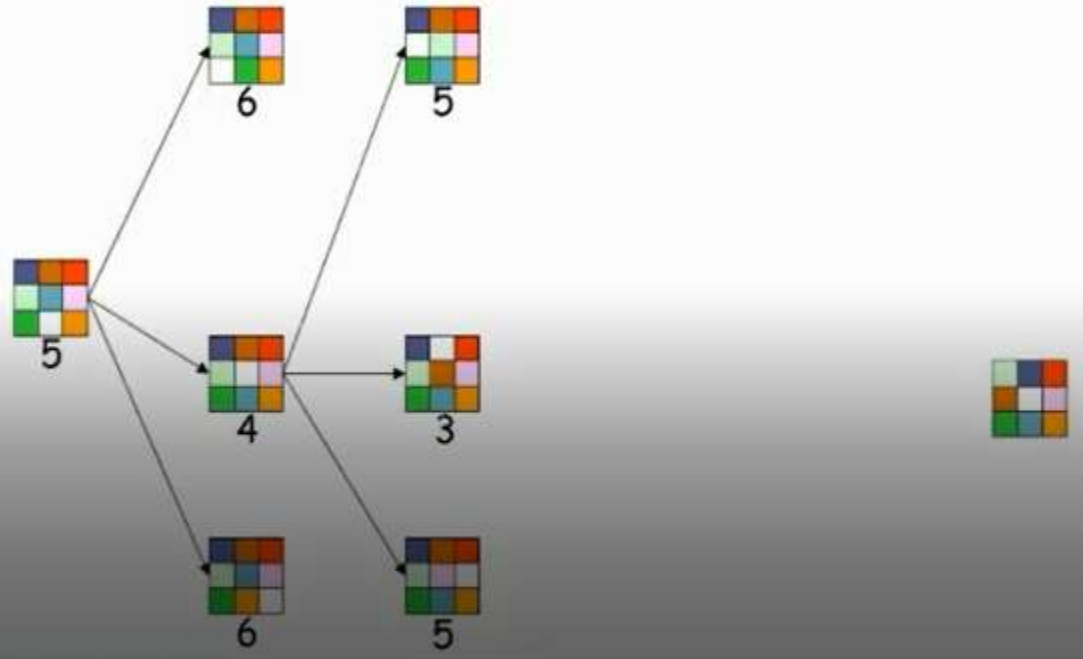
Another Example: 8 puzzle

$f(n) = h(n) = \sum$ distances of numbered tiles to their goals



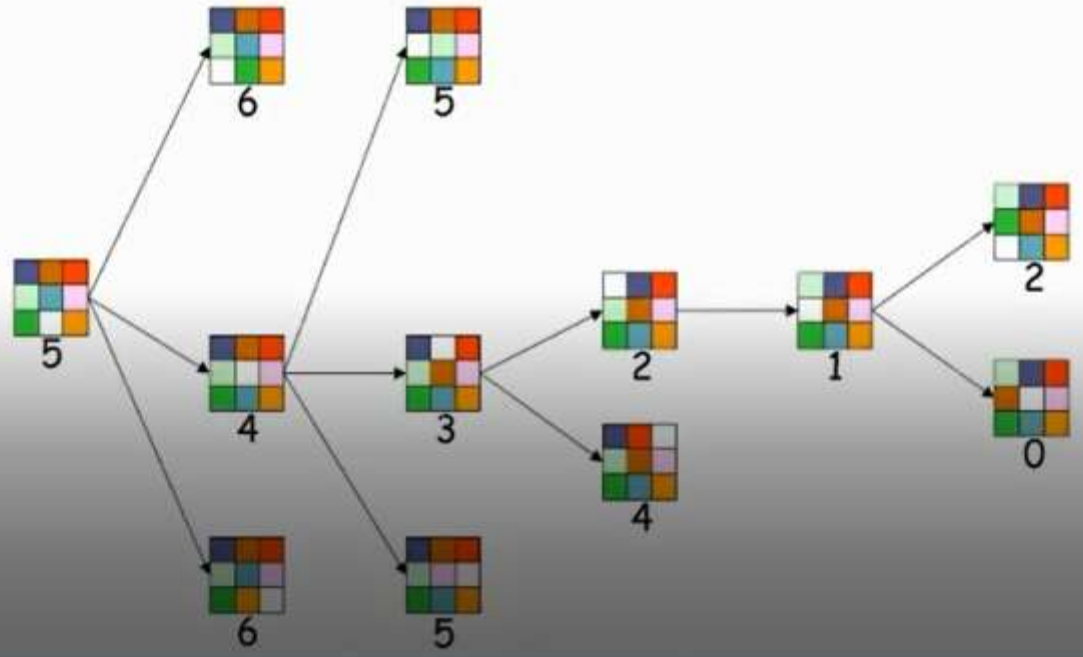
Another Example: 8 puzzle

f(n) = h(n) = ∑ distances of numbered tiles to their goals



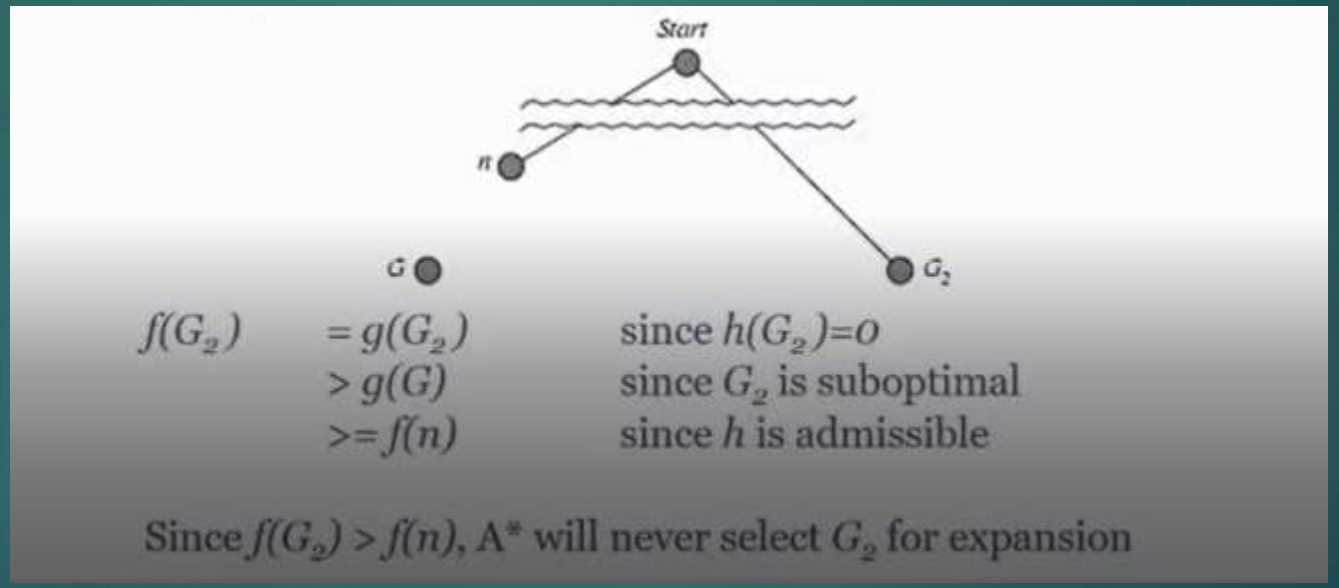
Another Example: 8 puzzle

f(n) = h(n) = ∑ distances of numbered tiles to their goals



Optimalite A*(proof)

•Varsayalım ki bazı suboptimal bir hedef G_2, oluşturulmuş ve kenarda bulunmaktadır. n, optimal bir hedef G'ye en kısa yol üzerinde bulunan ve henüz genişletilmemiş bir kenar düğümü olsun



Graf ile Çözüm:A* algoritması, graf yapısı üzerinde çalışır; düğümler arasındaki bağlantılar ve yollar değerlendirilir.Her düğüm, bağlı olduğu diğer düğümlerle birlikte incelenir ve en iyi (optimal) yol belirlenmeye çalışılır.

A* algoritmasında bir sezgisel fonksiyon $h(n)$ tutarlı (veya yerel tutarlı) olarak kabul edilir, eğer her iki düğüm n ve n' için aşağıdaki koşul sağlanıyorsa:

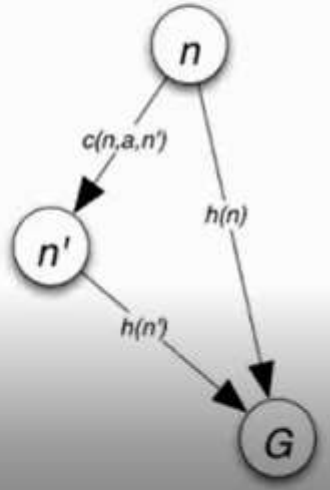
$$h(n) \leq c(n,a,n') + h(n')$$

Burada $c(n,n')$, düğüm n ile n' arasındaki gerçek maliyet (mesafe) olarak tanımlanır. Bu, bir düğümden diğerine geçerken hedefe olan tahmini maliyetin, aradaki maliyetle kıyaslandığında aşılmaması gerektiğini belirtir.

Consistent heuristics

- A heuristic is consistent if for every node n, every successor n' of n generated by any action a,

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



- If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

- i.e., $f(n)$ is non-decreasing along any path.
- Theorem: If $h(n)$ is consistent, A* using `GRAPH-SEARCH` is optimal

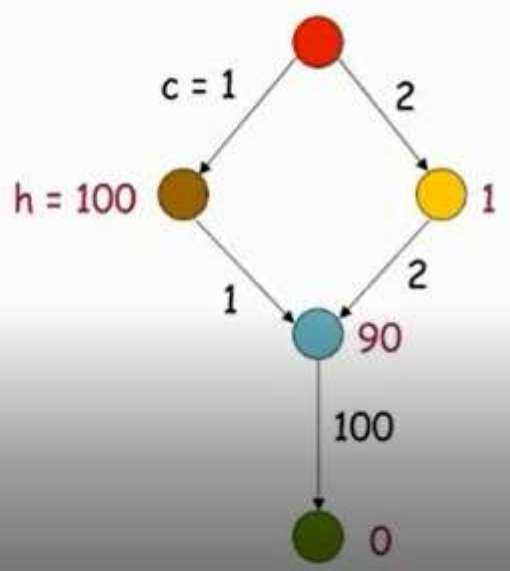
Geçerlilik, bir sezgisel fonksiyonun her durum için hedefe ulaşmanın gerçek maliyetini asla aşmaması durumudur; bu da algoritmanın optimal çözümler bulmasını garanti eder.

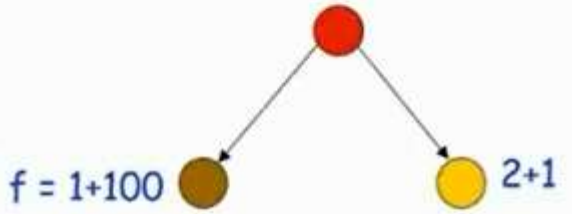
Bu iki özellik arasında sıkı bir ilişki vardır: her tutarlı fonksiyon geçerli olsa da, her geçerli fonksiyon tutarlı olmayabilir.

Yani, eğer bir sezgisel fonksiyon tutarlı ise, optimal çözümler bulma garantisi sağlanır.

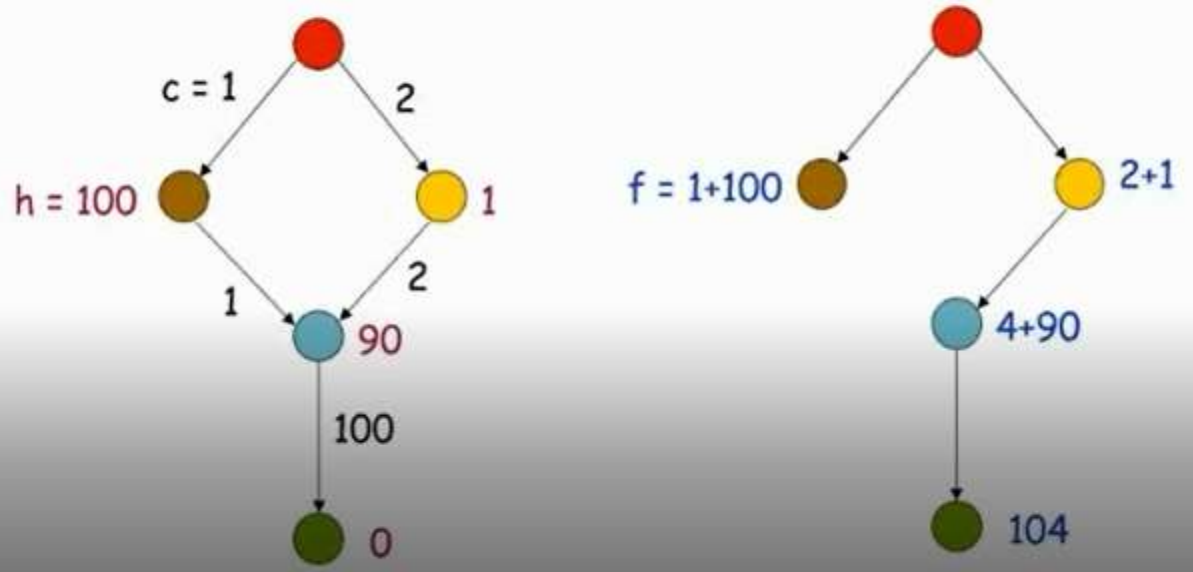
Ancak sadece geçerli bir fonksiyon kullanılması, tutarlılığın sağlanmadığı durumlarda algoritmanın optimal çözüm bulma yeteneğini tehlikeye atabilir. Bu nedenle, etkili bir arama algoritması için her iki özelliğin de göz önünde bulundurulması önemlidir.

Consistent heuristic: Example

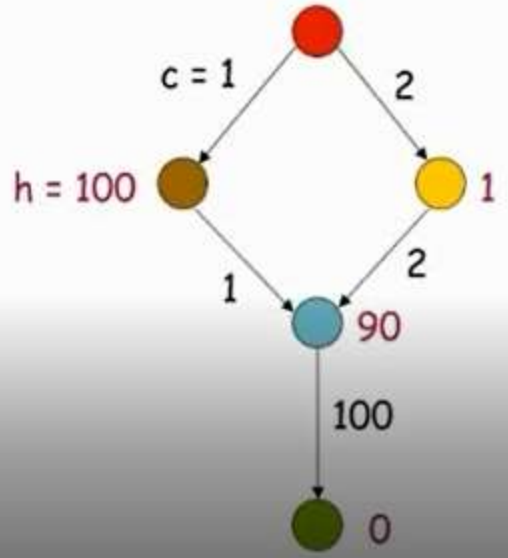


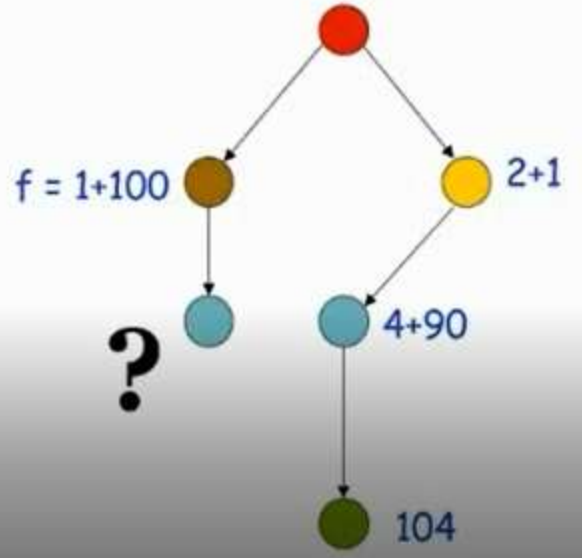


Consistent heuristic: Example



Consistent heuristic: Example





Consistent heuristic: Example