

ARAMA SEARCH

Prof. Dr. Aybars UĞUR

BÖLÜM I

ARAMA ile PROBLEM ÇÖZME (Solving Problems by Searching)

Aramanın Önemi

Problemlerin Çözümünde Aramanın Önemi

Bir problemde hedef belirtilir.

Hedefe ulaşmak için arama yapılır.

Terminoloji

Initial State

Goal Test

Successor Function

Path Cost

State Space

Solution

Path

Örnek Problem 1) 8-puzzle Problemi

Başlangıç, Bitiş Durumu, Durum Uzayı

Durum Uzayı (State Space) Boyutu : $9!/2 = 181440$

8-puzzle için 181440, 15-puzzle=?, 24-puzzle=?

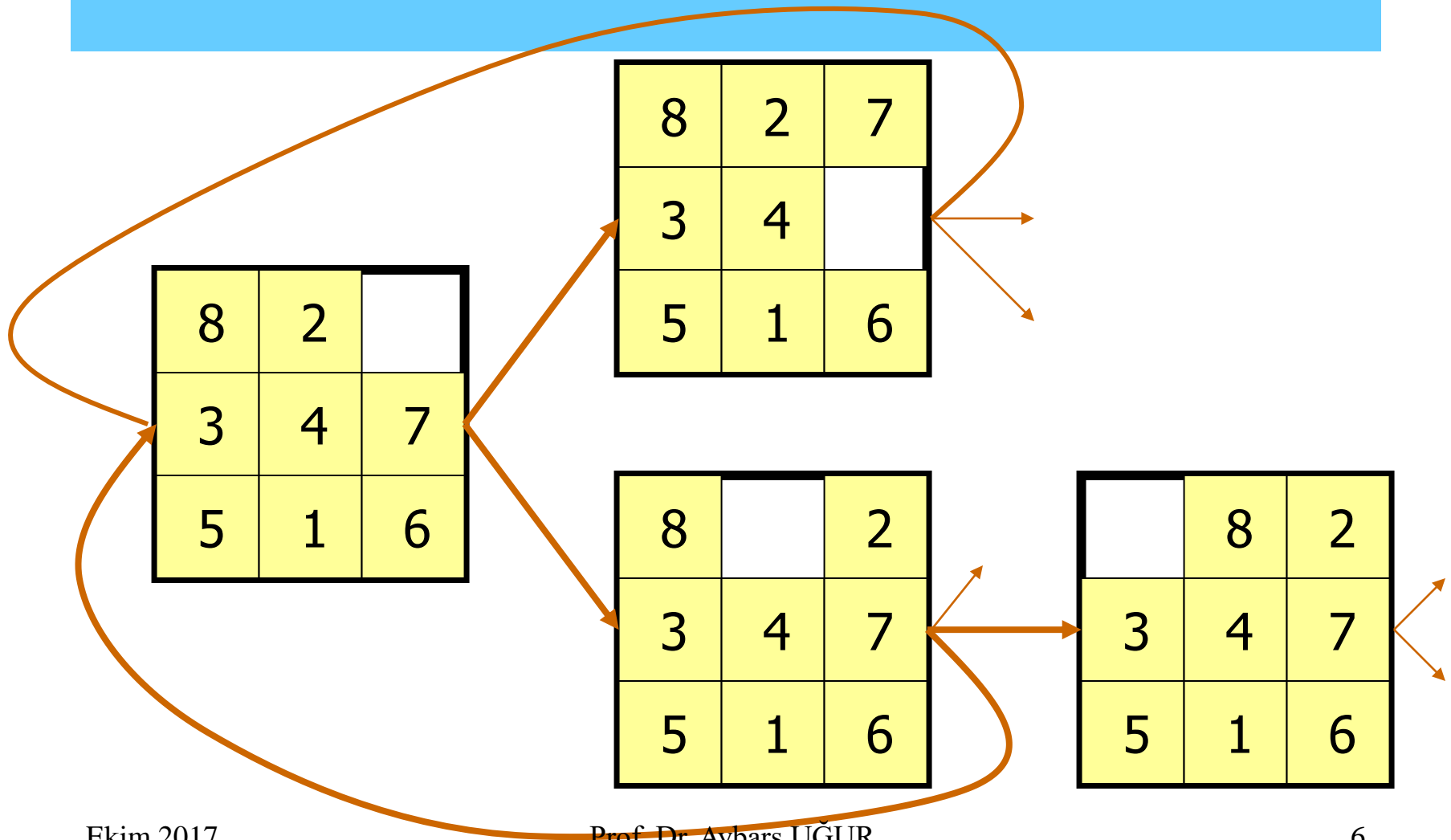
8	2	
3	4	7
5	1	6

Initial state

1	2	3
4	5	6
7	8	

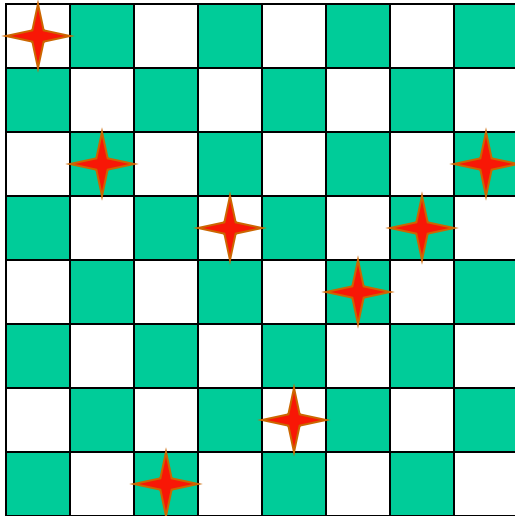
Goal state

8-puzzle : Kurallar ve Olası Hareketler



Örnek Problem 2) 8-queens Problemi

64⁸ farklı durum var.

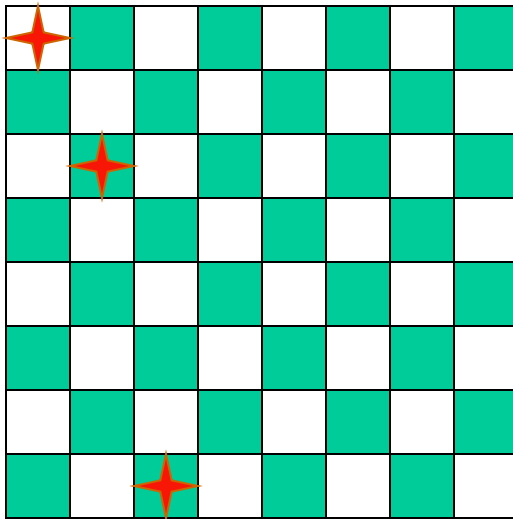


1. Yöntem

- Durumlar (States) : 8 vezirin oyun tahtasına herhangi bir şekilde yerleştirilmesi.
- İlk Durum (Initial State) : Tahtada vezir olmadığı durum.
- Sonraki Fonksiyonu (Successor function) : Veziri herhangi bir kareye ekle.
- Hedef Durum (Goal State) : 8 vezirin, tahtaya birbirini alamayacak şekilde yerleştirilmesi

8-queens Problemi

2067 farklı durum var.



2. Yöntem

- Durumlar : $k=1..8$, k vezirin, olabilen en sol sütuna, birbirini alamayacak şekilde yerleştirilmesi.
- İlk Durum : Tahtada vezir olmadığı durum.
- Sonraki Fonksiyonu : Bir vezirin, boş olan en soldaki sütundaki bir kareye, diğer vezirler tarafından alınamayacak şekilde yerleştirilmesi.
- Hedef Durum : 8 vezirin tahtada olması.

Arama Problemleri

- Bir veya daha fazla başlangıç durumu.
- Bir veya daha fazla bitiş durumu (çözüm).
- Çözüm, yol veya hedef düğümdür :
 - 8-puzzle’da yol.
 - 8-queen’de hedef düğüm.

Gerçek Hayat Problemleri :

Robot Navigation, Route Finding, ...

Önemli Parametreler

- Durum uzayındaki durum sayısı.
- Durumları tutmak için gereken bellek miktarı.

Arama Yöntemleri

Search Strategies

Her aşamada hangi düğümün açılacağını belirler.

Bilgisiz (Kör) Arama (Uninformed Search)

Durum bilgisinden yararlanmaz.

Bilgiye Dayalı (Sezgisel) Arama (Informed Search)

Durum bilgisinden yararlanır. Daha umut verici hareketi tercih eder.

Sezgisel Yöntemlerin Avantajı

8	2	
3	4	7
5	1	6

STATE



N1

Sezgisel yöntemde, yerinde olan kareler sayıldığında N2, N1'den daha umut vericidir.

1	2	3
4	5	
7	8	6

STATE



N2

1	2	3
4	5	6
7	8	

Goal state

8-puzzle Probleminin Arama Ağacı

8	2	
3	4	7
5	1	6

8	2	7
3	4	
5	1	6

8		2
3	4	7
5	1	6

	8	2
3	4	7
5	1	6

8	4	2
3		7
5	1	6

8	2	
3	4	7
5	1	6

Search Nodes \neq States

Durum Uzayı sonlu olsa bile
Arama Ağacı sonsuz olabilir.

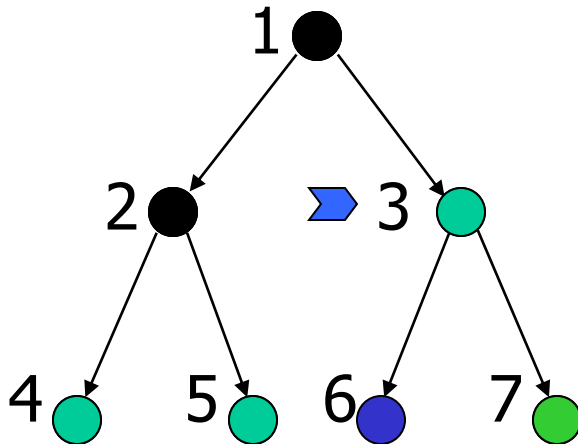
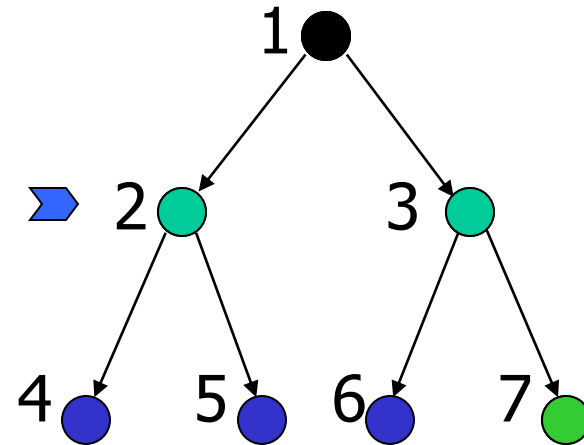
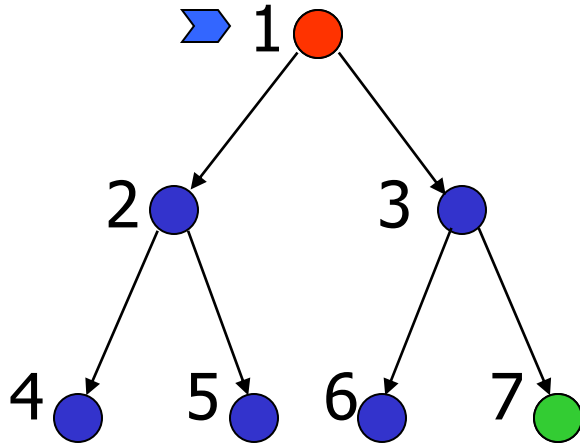
Measuring Problem Solving Performance

- Strategies are evaluated along the following dimensions:
 - **completeness**: does it always find a solution if one exists?
 - **time complexity**: number of nodes generated
 - **space complexity**: maximum number of nodes in memory
 - **optimality**: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
 - b : maximum branching factor of the search tree
 - d : depth of the least-cost solution
 - m : maximum depth of the state space (may be ∞)

Sezgisel Olmayan Arama Yöntemleri

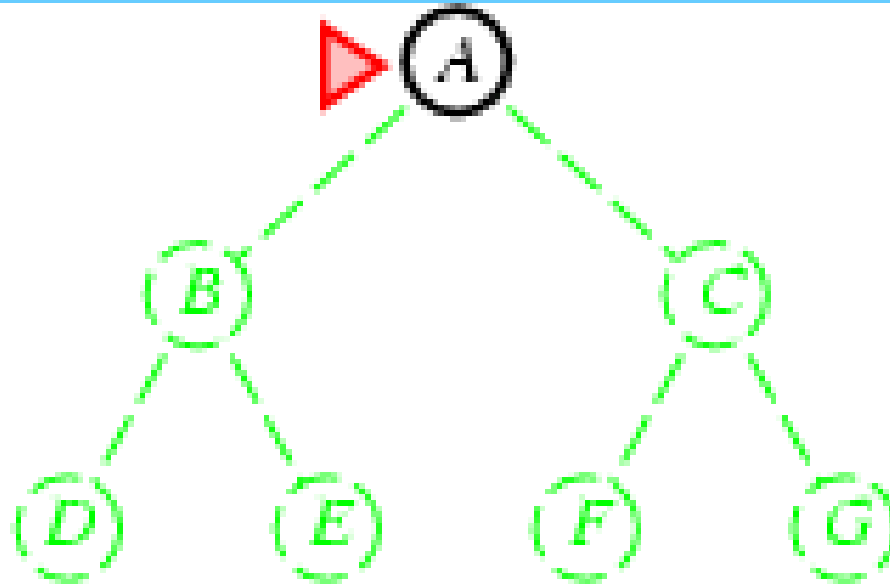
- Breadth-First
 - Bidirectional
- Depth-First
 - Depth-limited
 - Iterative deepening
- Uniform-Cost

Önce Genişliğine Arama BFS (Breadth-First Search)

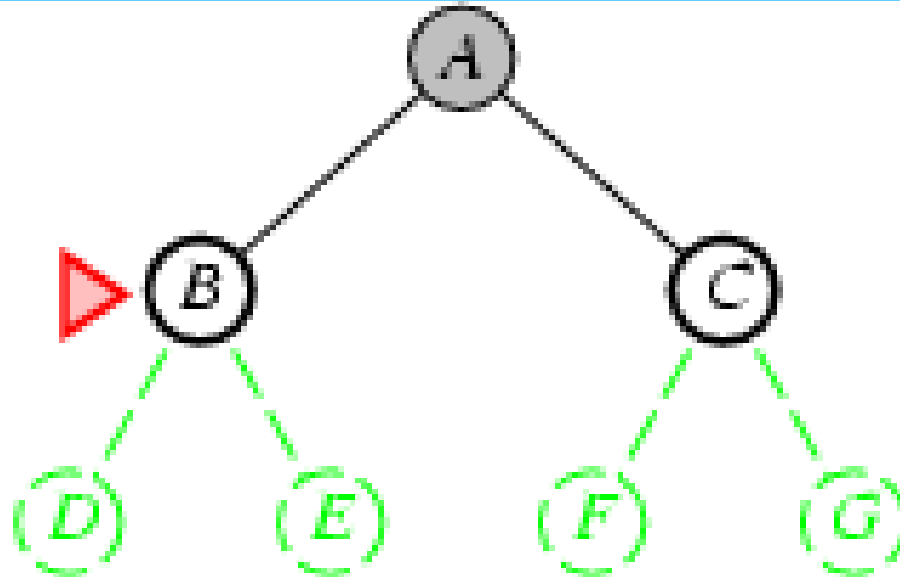


- Düğümleri, kuyruğun sonuna ekler.
- i . Düzeydeki tüm düğümler, $(i+1)$. Düzeydeki tüm düğümlerden önce açılır. BFS, hedefe ulaştıran en kısa yolu bulmayı garantiler.

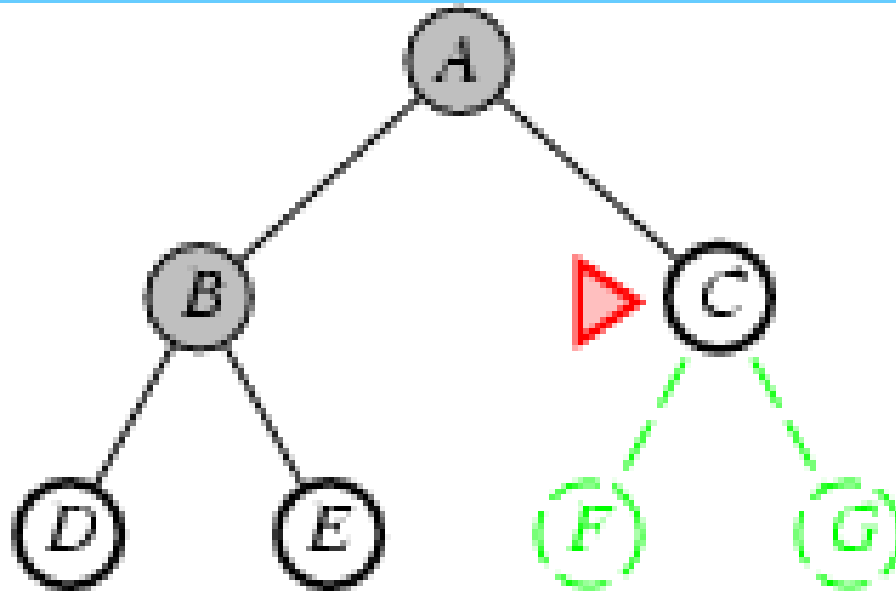
BFS



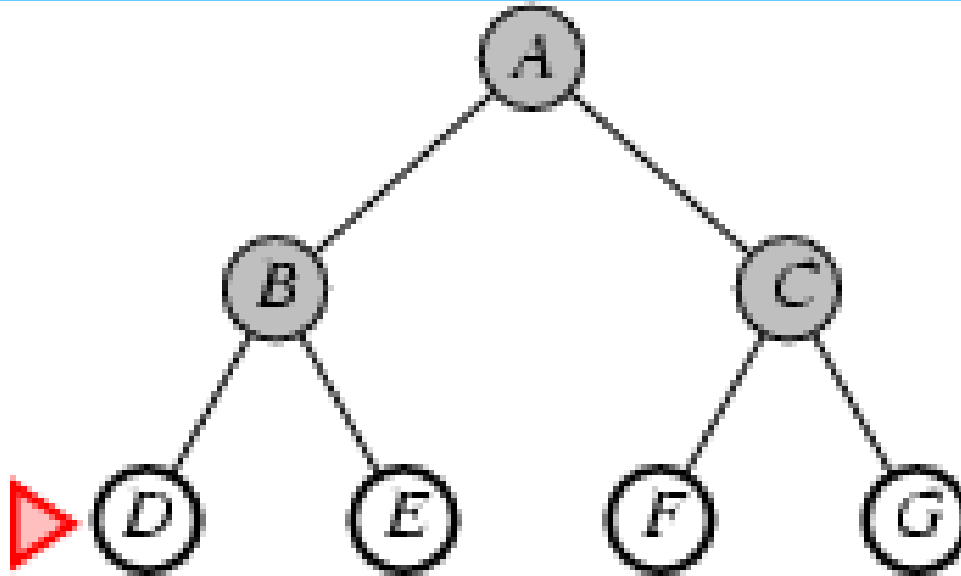
BFS



BFS



BFS



BFS: Algoritma Performansı

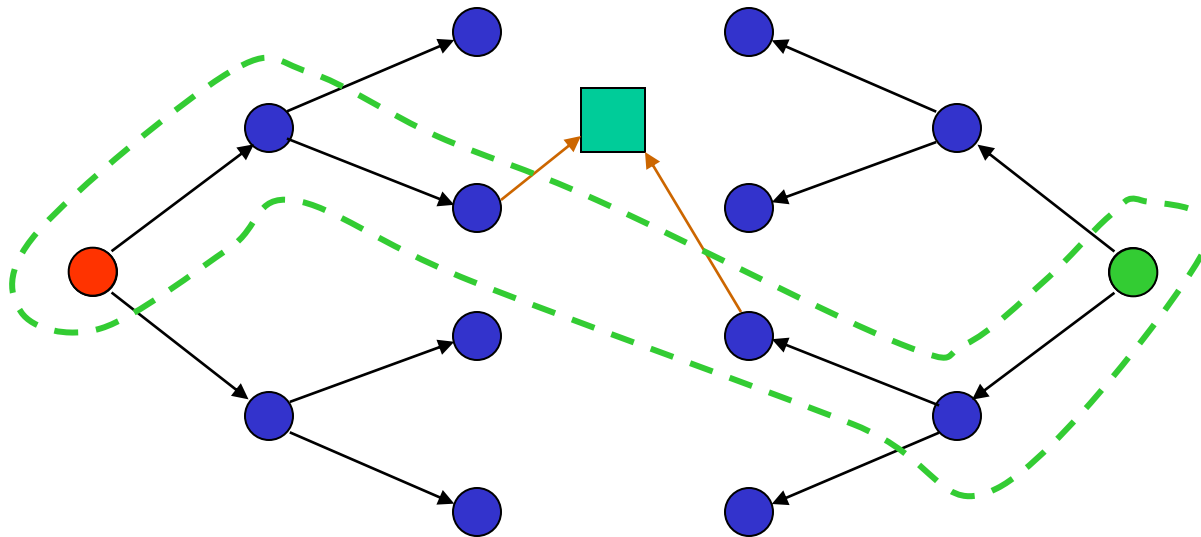
- Tamlık? Yes (if b is finite)
- Zaman Karmaşıklığı?
 $1 + b + b^2 + b^3 + \dots + b^d = O(b^d)$
- Yer Karmaşıklığı?
 $O(b^{d+1})$ (keeps every node in memory)
- Optimal? Yes (if cost = 1 per step)

Space is the bigger problem (more than time)

Çift Yönlü Arama Bidirectional (BF) Search

Start ●

● Goal

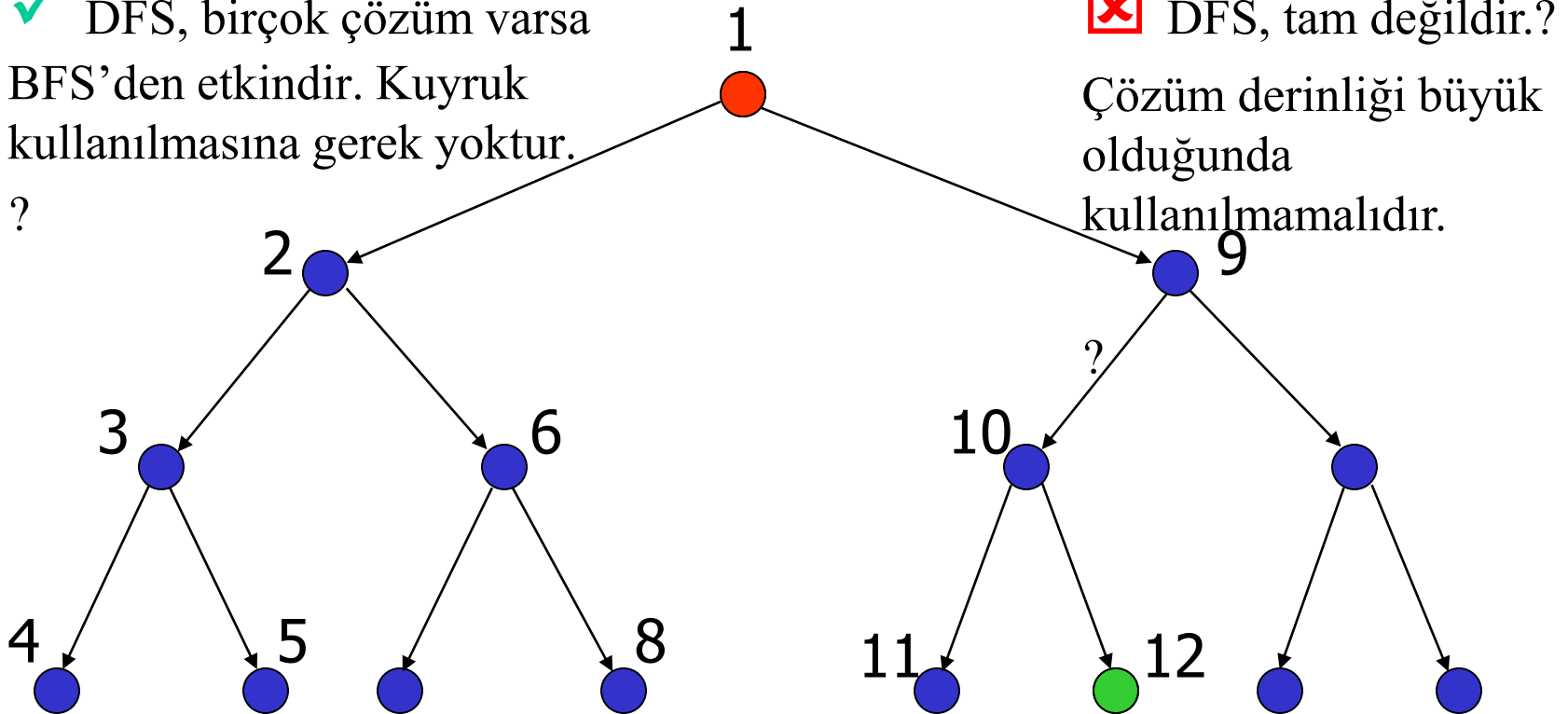


Önce Derinliğine Arama

DFS (Depth-First Search)

✓ DFS, birçok çözüm varsa
BFS'den etkindir. Kuyruk
kullanılmasına gerek yoktur.

?

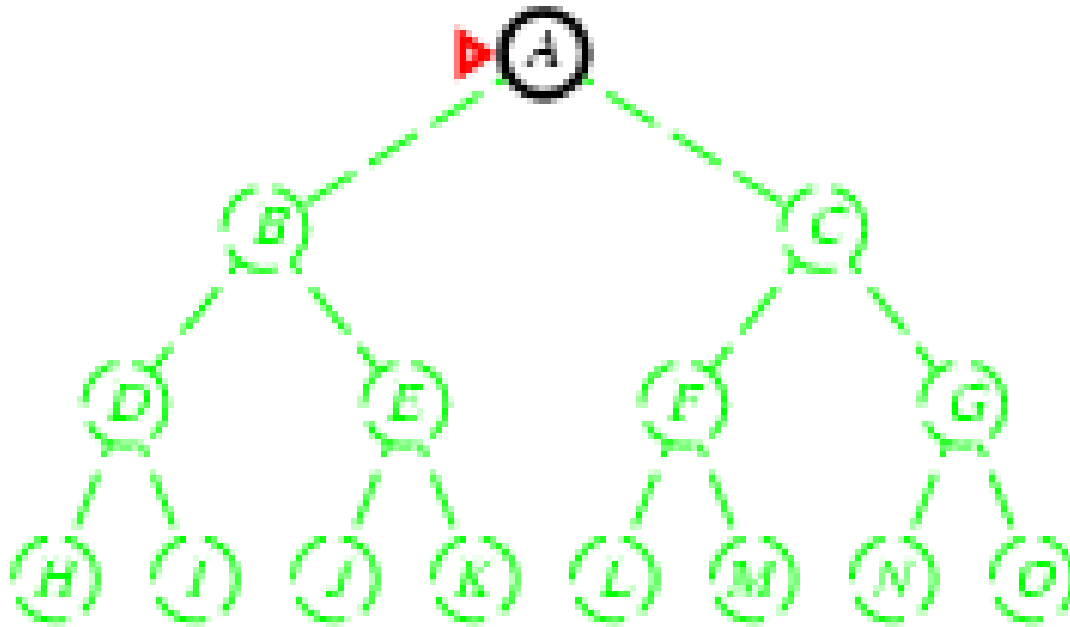


✗ DFS, tam değildir.?

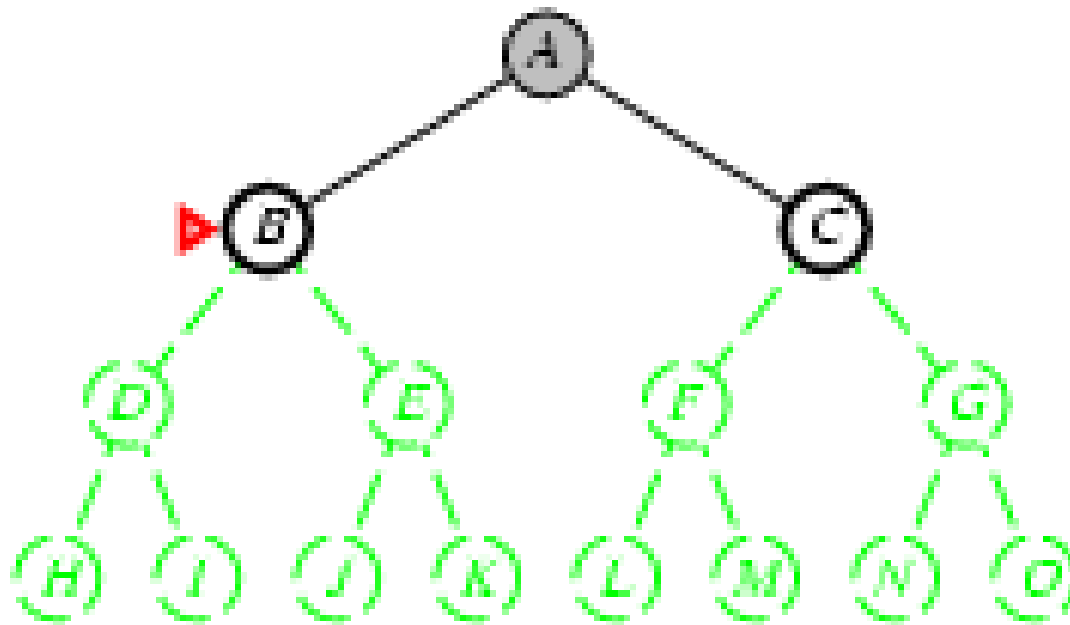
Çözüm derinliği büyük
olduğunda
kullanılmamalıdır.

?

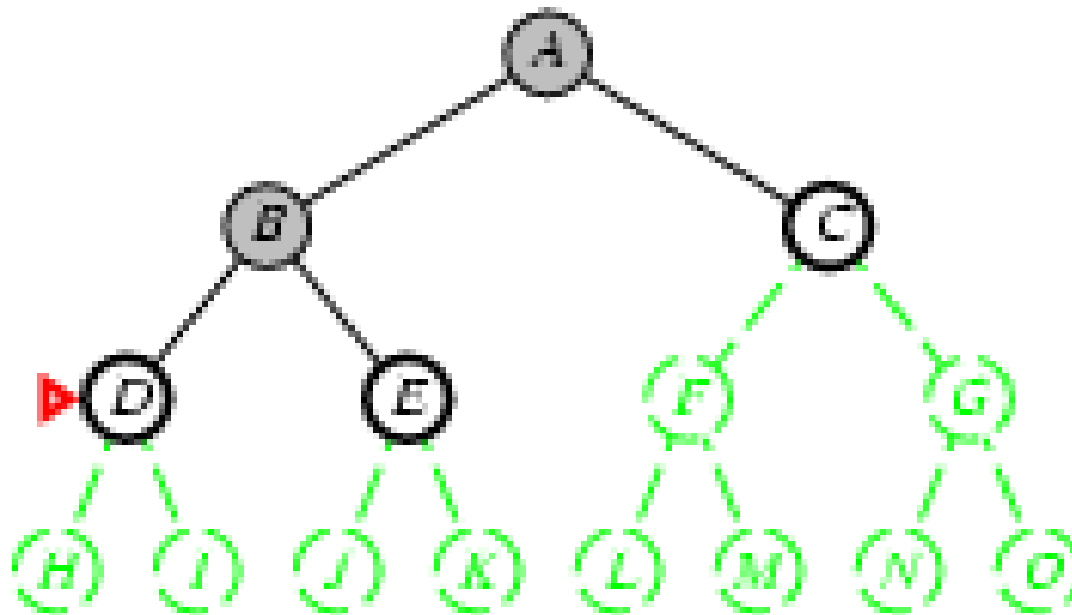
DFS



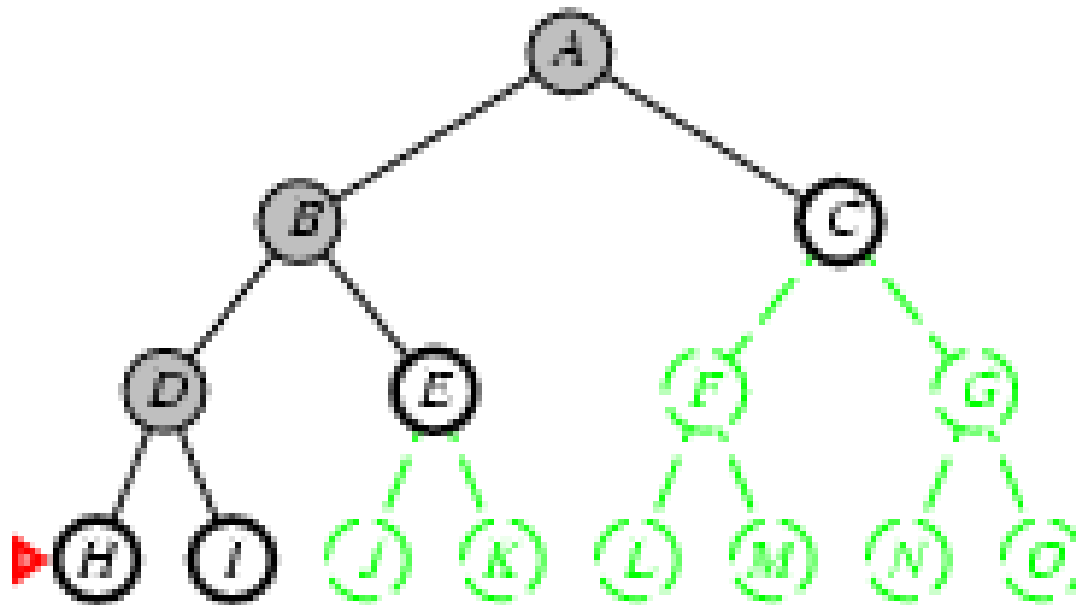
DFS



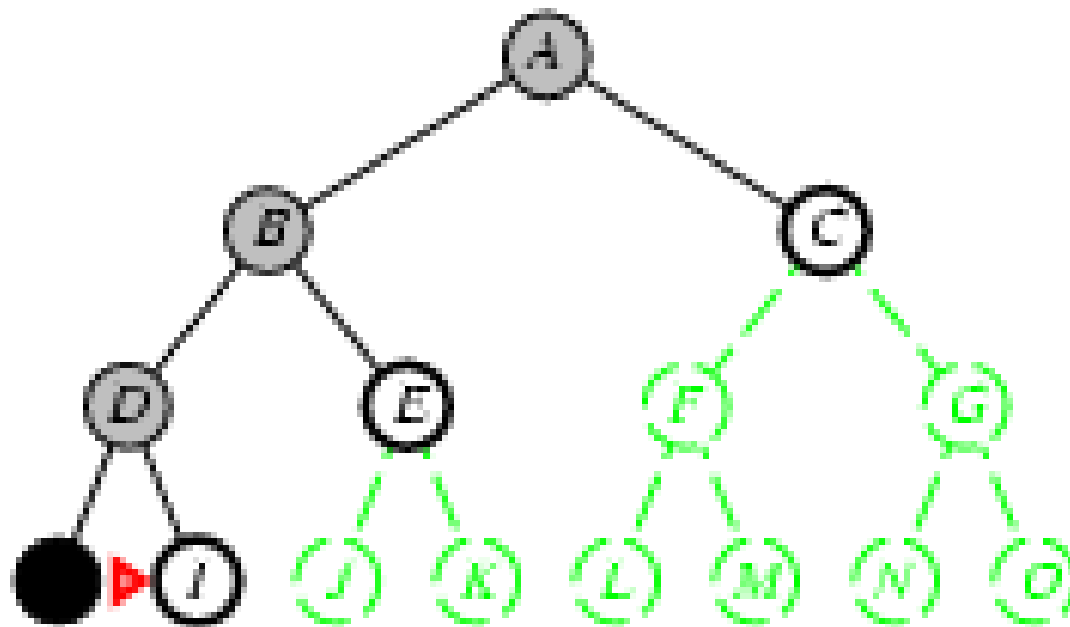
DFS



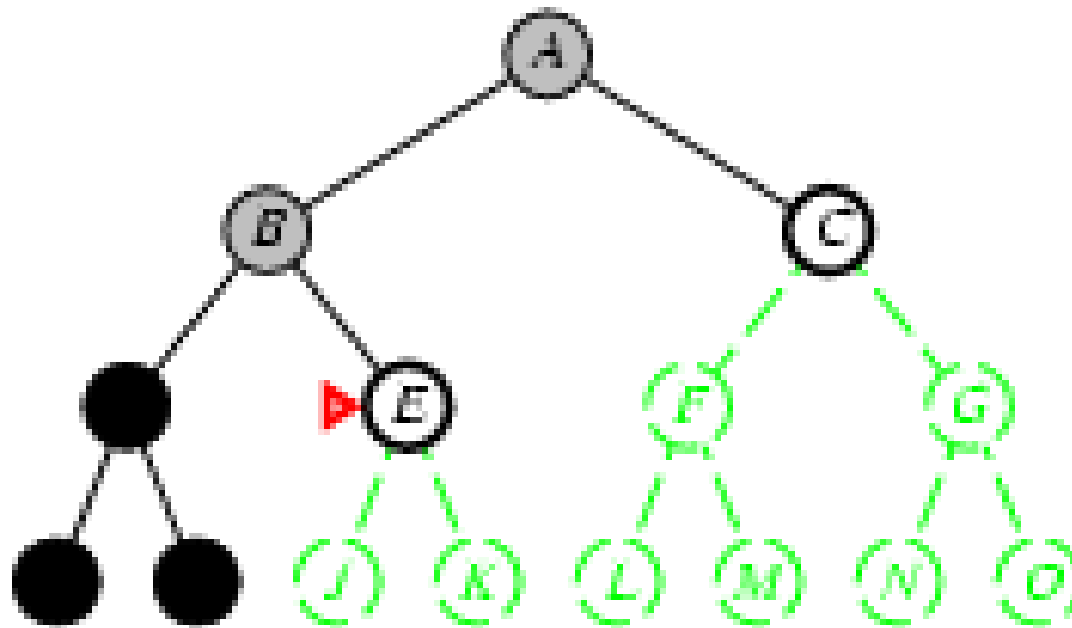
DFS



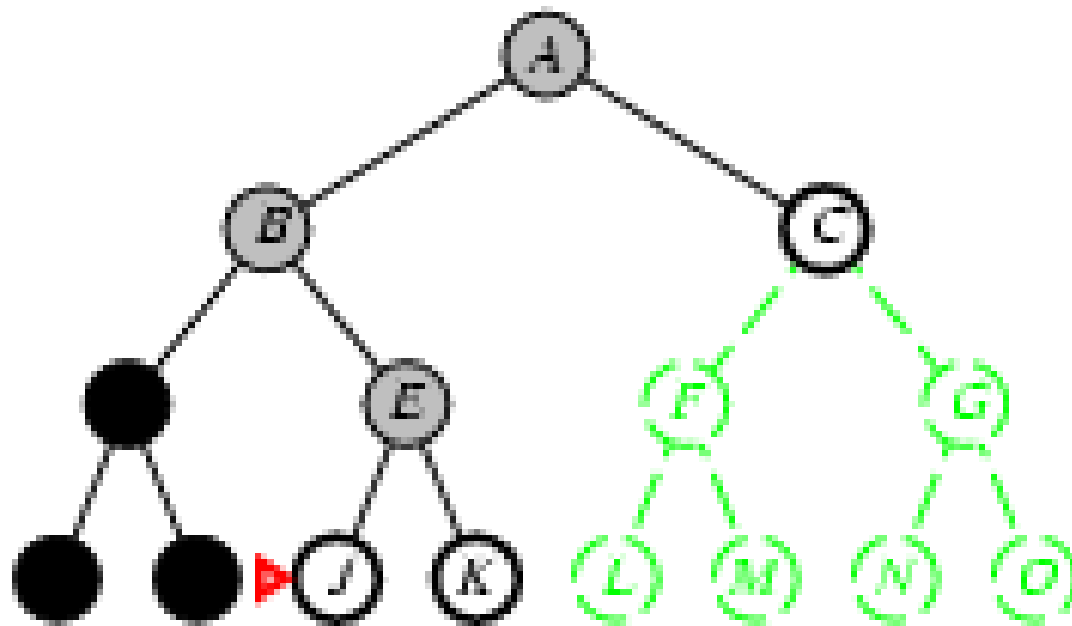
DFS



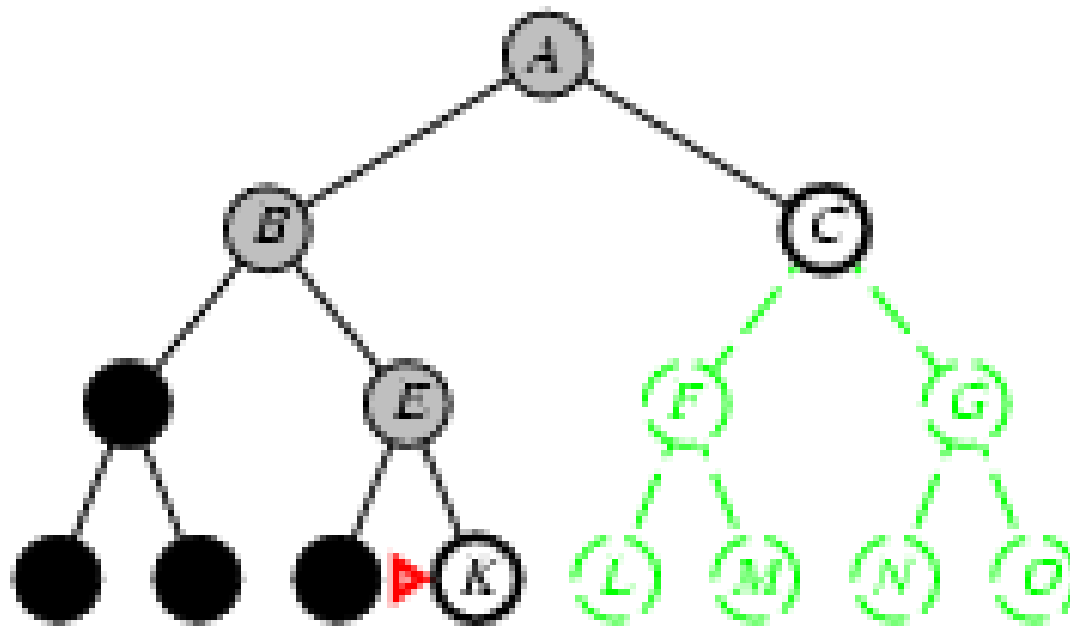
DFS



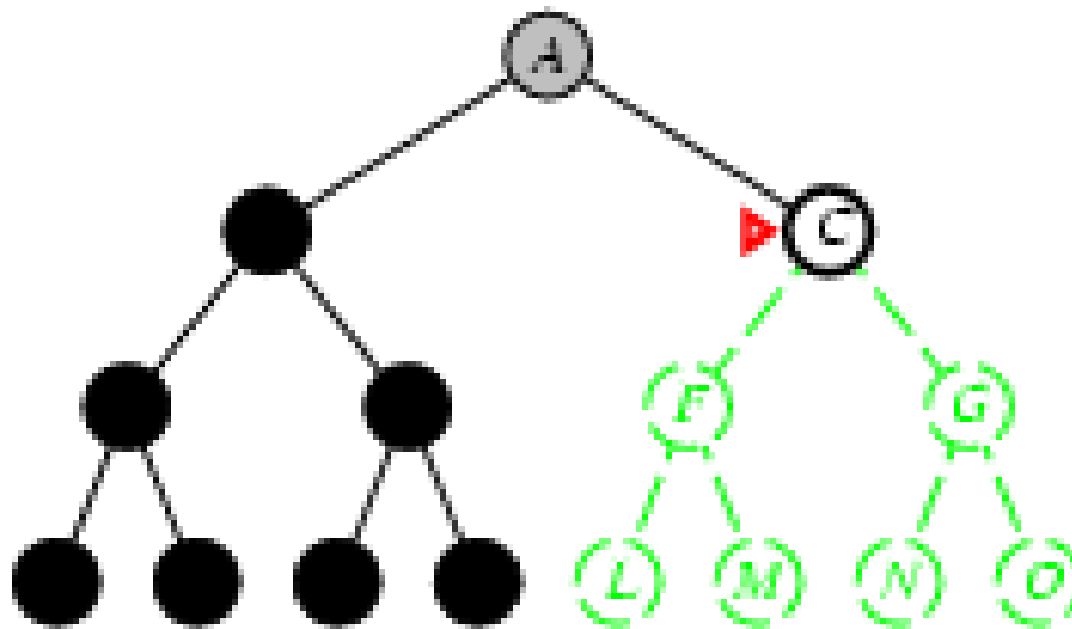
DFS



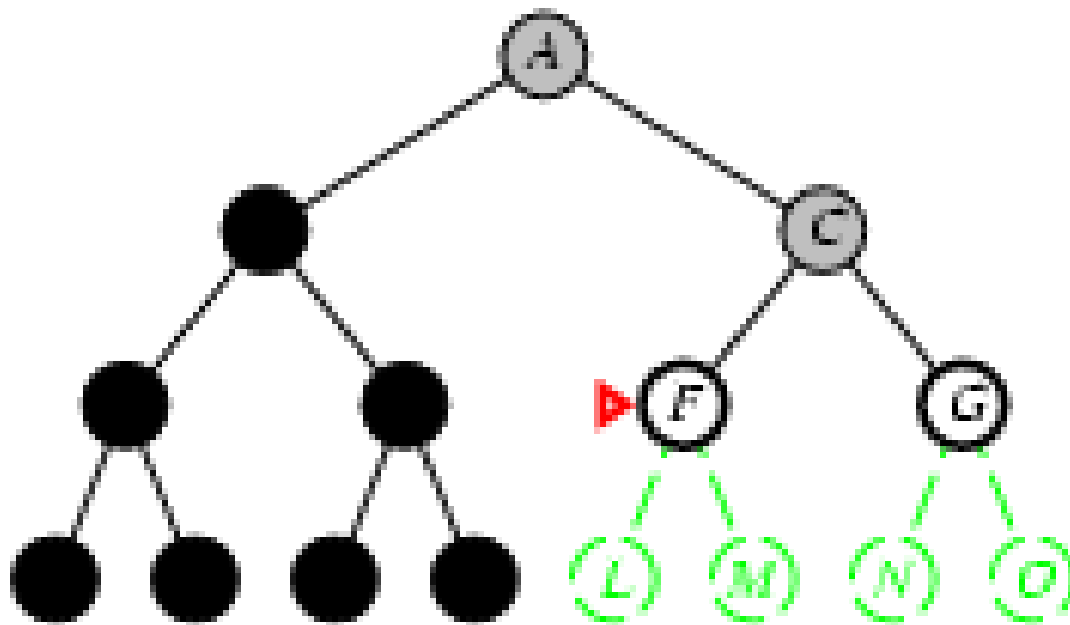
DFS



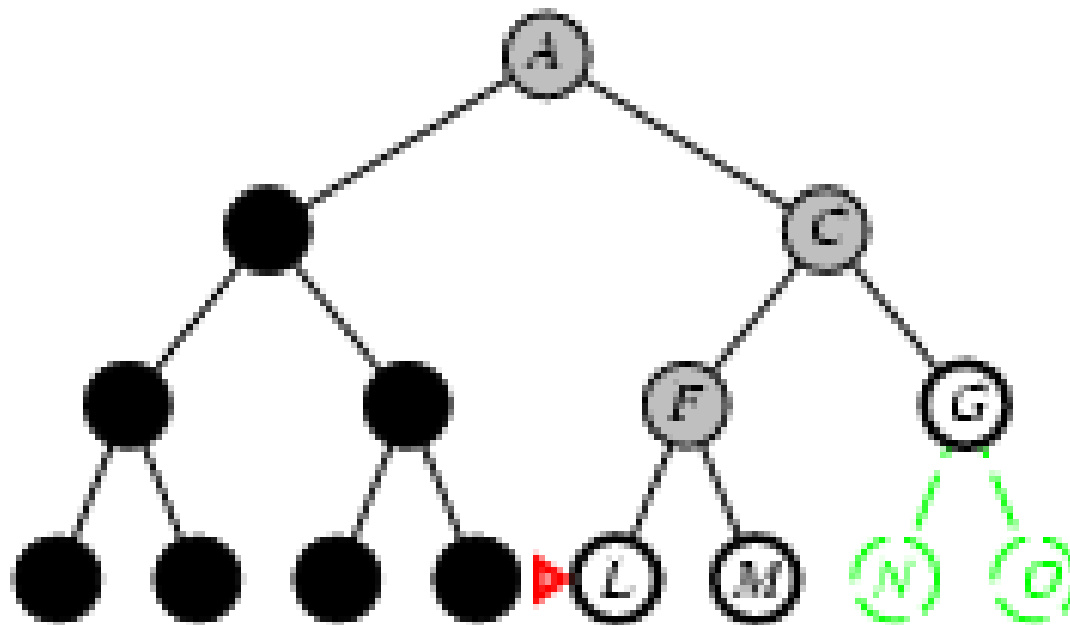
DFS



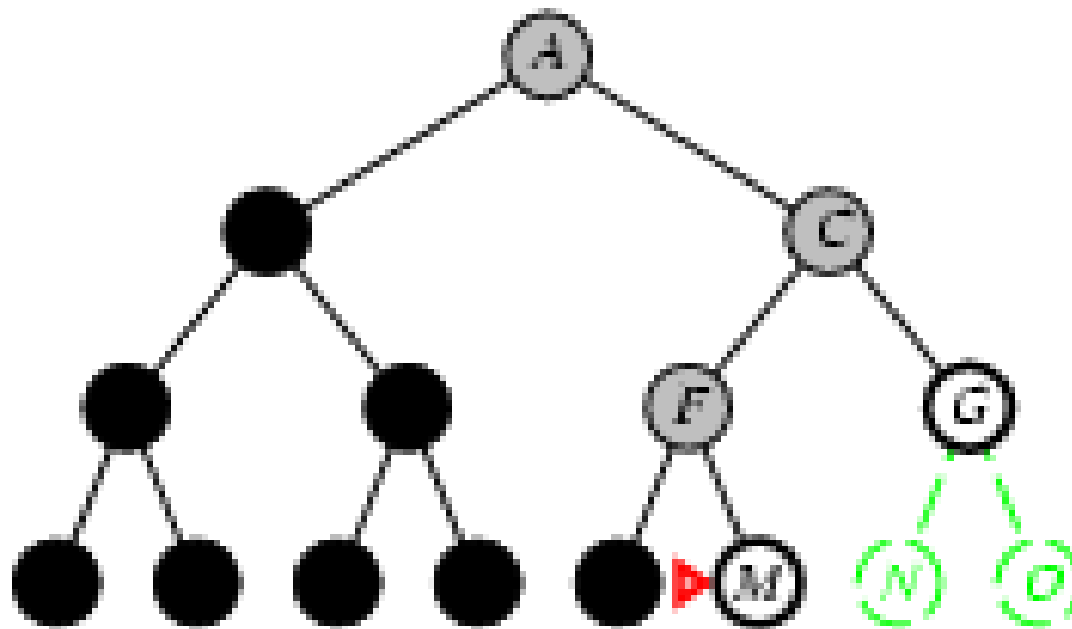
DFS



DFS



DFS



DFS: Algoritma Performansı

- Tamlık? No: fails in infinite-depth spaces, spaces with loops
 - Modify to avoid repeated states along path
 - complete in finite spaces
- Zaman Karmaşıklığı?
 $O(b^m)$: terrible if m is much larger than d
 - but if solutions are dense, may be much faster than breadth-first
- Yer Karmaşıklığı? $O(bm)$, i.e., linear space!
- Optimal? No

Derinlik Sınırlandırılmalı Arama

Depth-Limited Search

k derinliğinde kesilen (cutoff) DFS'dir.

k, altındaki düğümlerin açılmayacağı maximum derinliktir.

Üç olası sonuç vardır :

- Çözüme ulaşılır (Solution)
- Çözüme ulaşılamaz (Failure)
- Cutoff (derinlik sınırları içinde sonuca ulaşılamaz)

Derinlik Sınırlandırılmalı Arama Algoritması

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred?  $\leftarrow$  false
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result  $\leftarrow$  RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred?  $\leftarrow$  true
        else if result  $\neq$  failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Yineli Derinleştirmeli Arama

Iterative Deepening Search

- For $k=0,1,2,\dots$:
 - K derinlik sınırı ile DFS yöntemini uygula

DFS ve BFS yöntemlerinin iyi yönlerini birleştirir.

Yineli Derinleştirmeli Arama Algoritması

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or fail-  
ure  
    inputs: problem, a problem  
    for depth  $\leftarrow$  0 to  $\infty$  do  
        result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)  
        if result  $\neq$  cutoff then return result
```

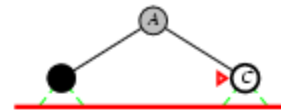

Iterative deepening search $l = 0$

Limit = 0



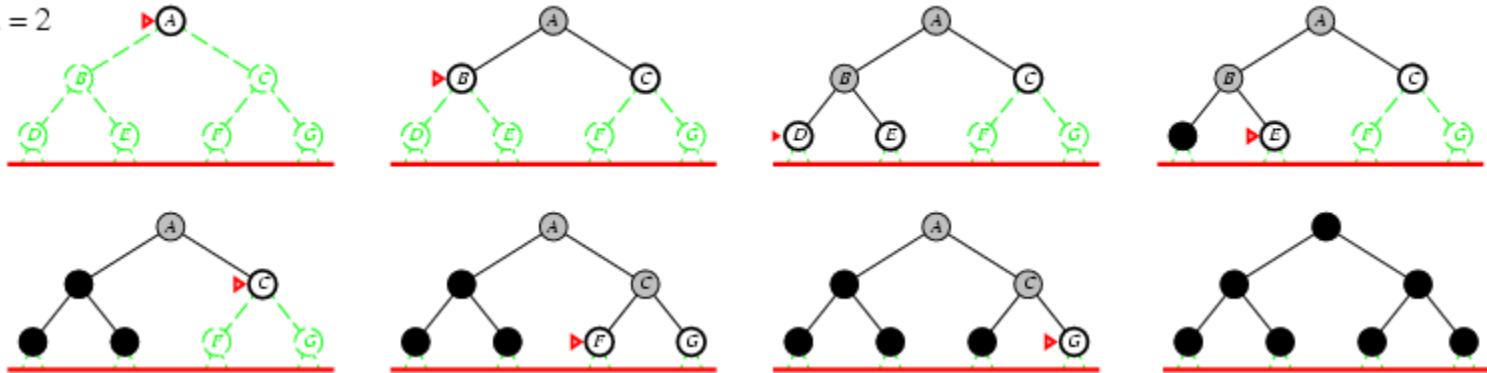
Iterative deepening search $l = 1$

Limit = 1



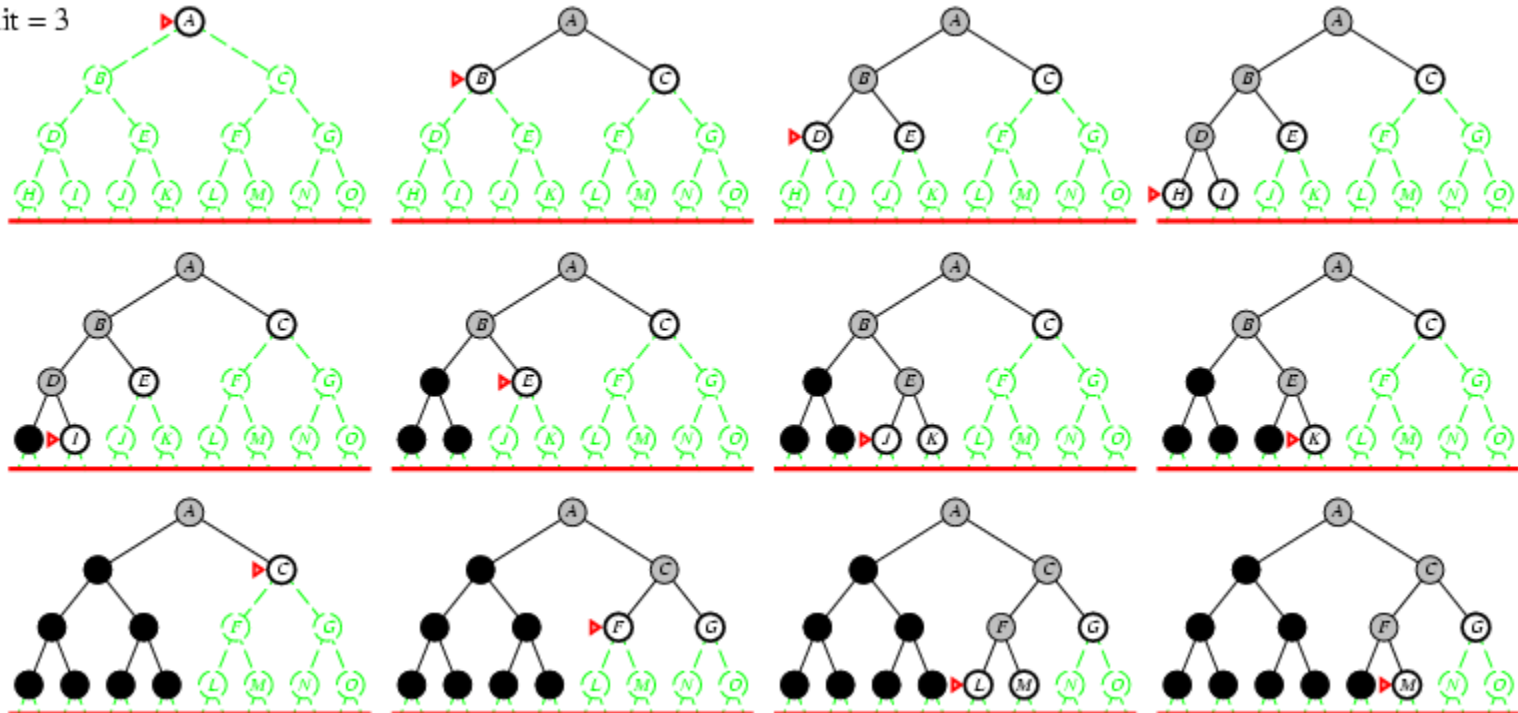
Iterative deepening search $l = 2$

Limit = 2



Iterative deepening search $l = 3$

Limit = 3



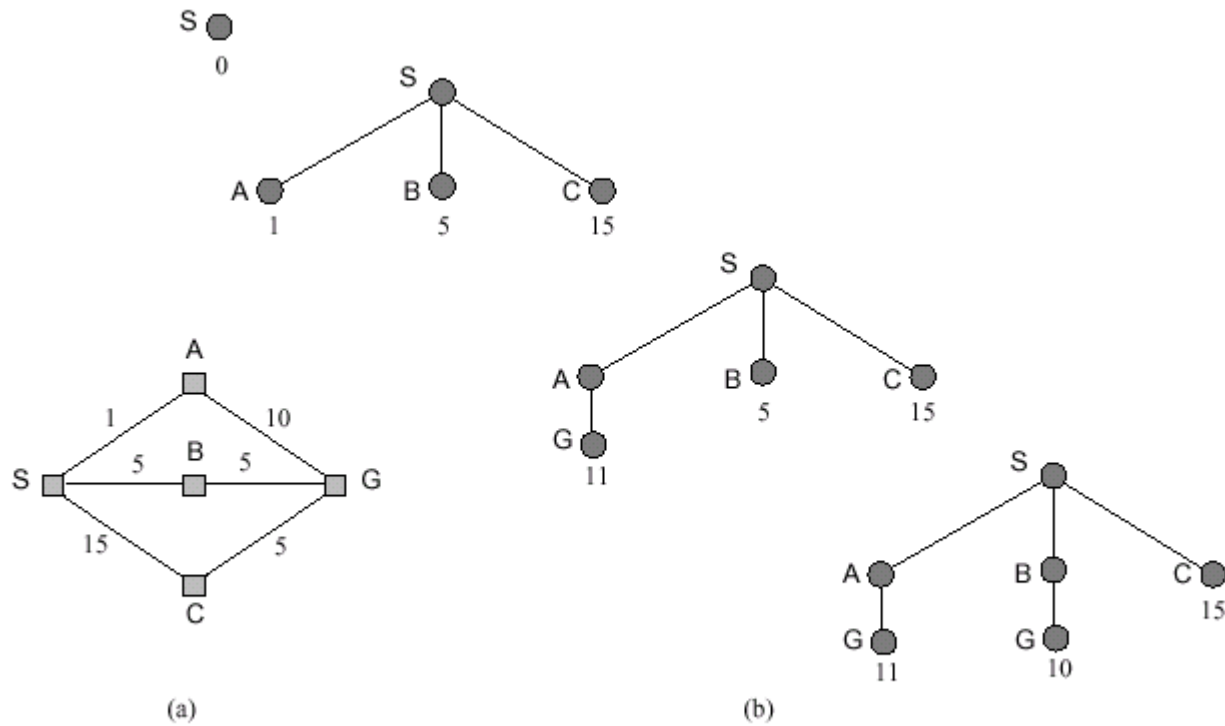
Özet: Sezgisel Olmayan Arama Yöntemleri Karşılaştırma

- BFS, tamdır, optimaldir ancak yer karmaşıklığı yüksektir.
- DFS, yer karmaşıklığı etkindir, ancak tam da değildir, optimal de değildir.
- Yineli derinleştirme, asimptotik olarak optimaldir.

Uniform Cost Search

- BFS'nin benzeridir.
- Basit kuyruk yerine öncelik kuyruğu kullanır.
- Düğümler, (o ana kadar gelenler içerisinde) artan sırada kuyruğa yerleştirilirler.
- Optimal çözümü bulmayı garantiler.

Uniform Cost Search Örneği



BÖLÜM II

SEZGİSEL ARAMA ve DOLAŞMA (Informed Search and Exploration)

Heuristic Search (Sezgisel Arama)

- Önceki arama yöntemlerinde düğümlerin açılmasında kullanılan yöntem, başlangıç düğümünden uzaklık bilgisine dayanıyordu.
- Birçok problemde de bilinen budur!
- Hedeften uzaklık tahminlenirse ne olur?
- Hedef biliniyorsa, bu arama olmaz. Sadece gereken düğümler açılır.
- Ama gerçek uzaklık tam olarak bilinmese de tahminlenebilir. Bu tahmine, Heuristic (Sezgi) yani $h(n)$ denilir.

Örnek : Rota Planlama

- Bir karayolu sisteminde, bir noktadan diğer noktaya düz hat uzaklığı, uygun bir sezgi ölçüsüdür.
- Her zaman doğru mudur?
- Hayır. Bazı yollar dolambaçlıdır.

Sezgisel Arama Yöntemleri

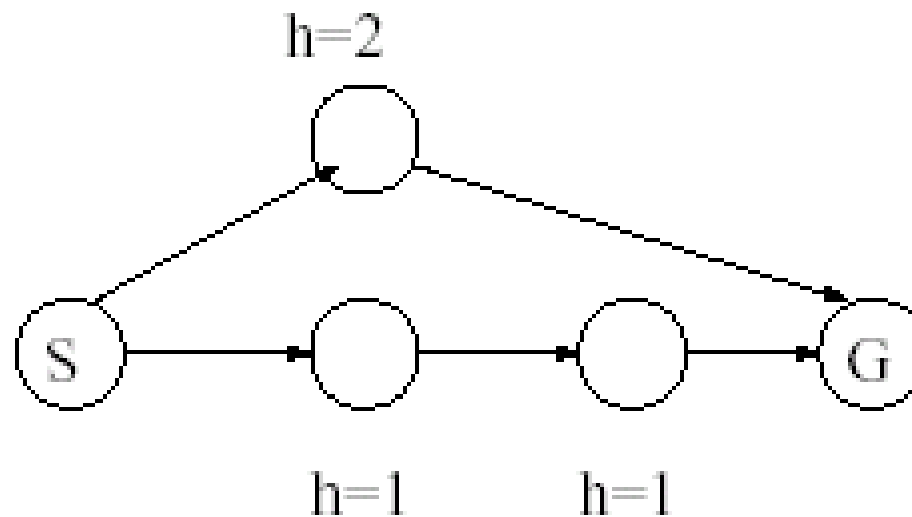
- Best-First Search
- A* Search
- IDA* (Iterative Deepening A*) Search
- SMA* (Simplified Memory-Bounded A*) Search

Best-First Search

- Herhangi bir anda, sezgiye göre en umut verici düğüm açılır.
- Aşağı yukarı uniform-cost search'ün tersidir.
- DFS'ye benzer, ancak sezgi içerir.
- Sezgi, her zaman 0 ise, BFS'ye döner.
- Best-first Search, bir Greedy Yöntemidir.
- Greedy Yöntemleri, uzun vadeli sonuçları önemsemeden, kısa vadeli avantajları en iyi duruma getirir.

Greedy Yöntemleri

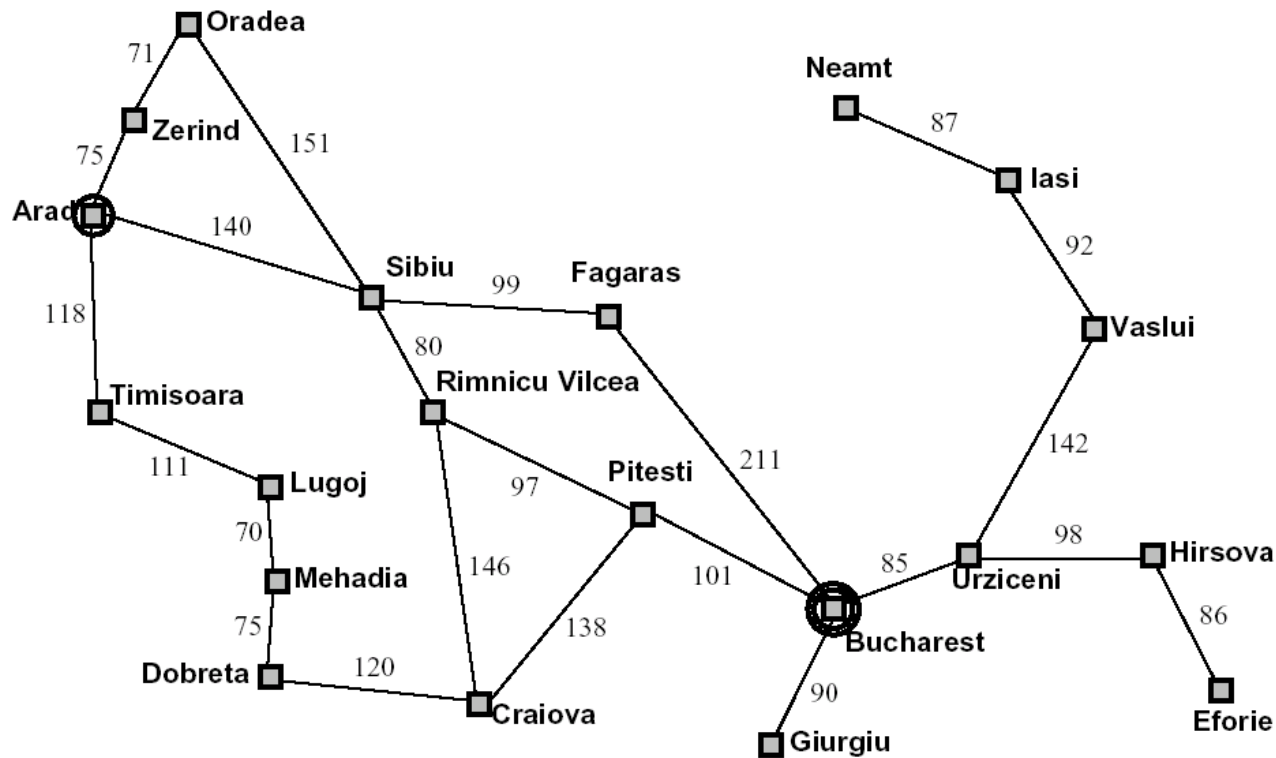
- Tam değildir
- Optimal değildir.
- Zaman ve Yer Karmaşıklıkları kötüdür (üstel)



A* Search

- Best-first Search'ün en büyük dezavantajı, uzak görüşlü olmamasıdır.
- g : şu anki duruma kadar maliyet (c) fonksiyonu
- h : uygun sezgi fonksiyonu olsun.
- Uygun olması, iyimser olması yani hedefe götüren maliyeti hiçbir zaman gerçek değerinden daha fazla tahminlememesi demektir.
- Best-first search, greedy yerine $f=g+h$ kullanılarak yapıldığında, A*'a ulaşılır.

Romanya Haritası (Yol Uzaklıkları ile)



Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Problem Tanımı

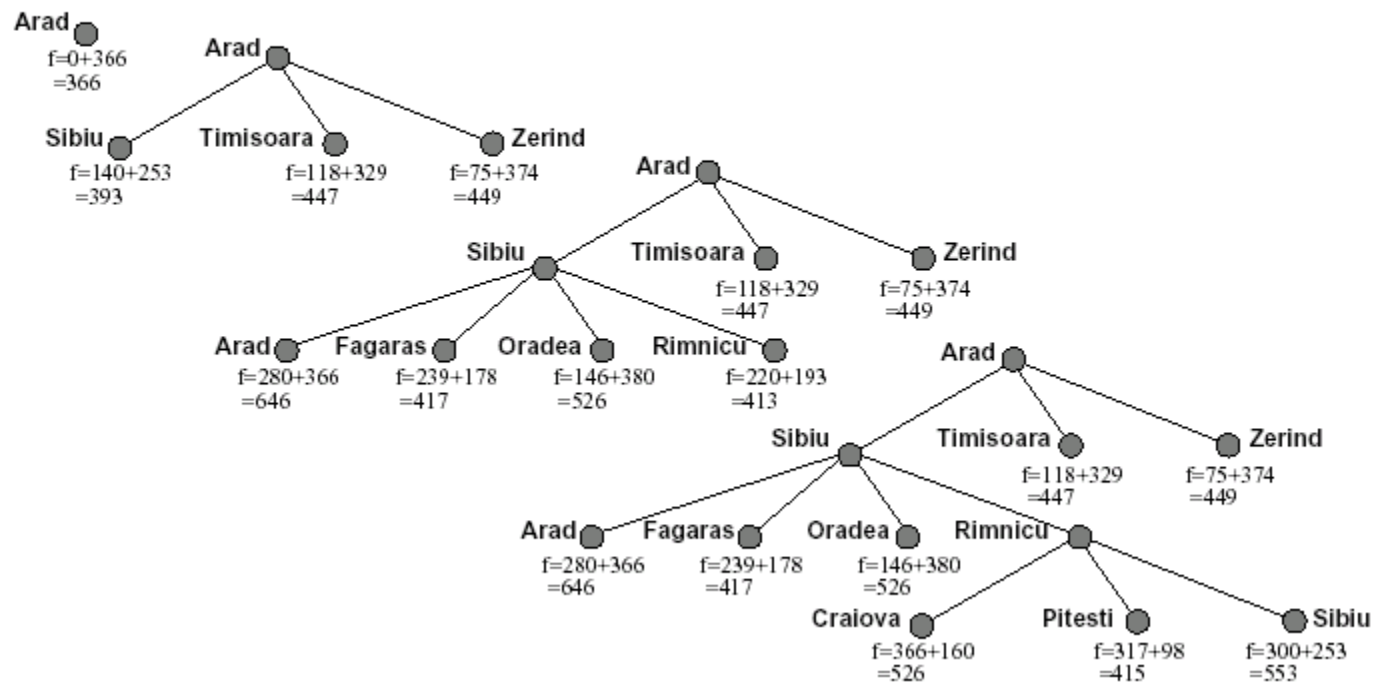
Problemi 4 bileşeni ile tanımlayalım:

1. **initial state**: "at Arad"
2. **actions** or **successor function** $S(x)$ = set of action–state pairs
 $S(Arad) = \{ \langle Arad \rightarrow Zerind, Zerind \rangle, \dots \}$
3. **goal test**, can be
 $x = \text{"at Bucharest"}$
4. **path cost** (additive)
sum of distances

Solution is a sequence of actions leading from the initial state to a goal state

A* Aramasının Aşamaları

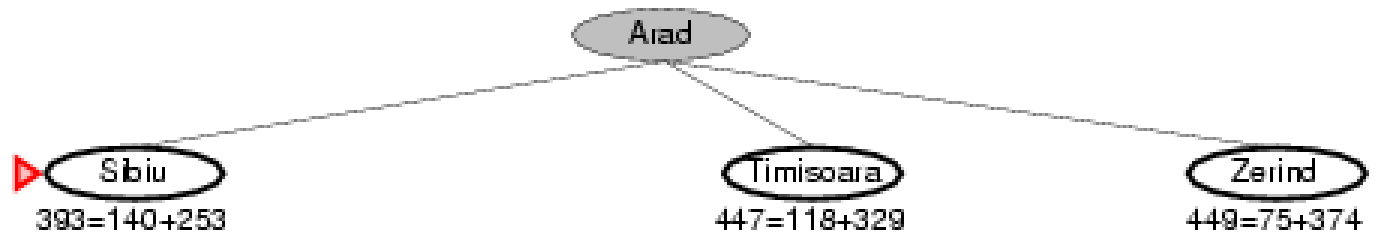
Example: Behavior of A^* search



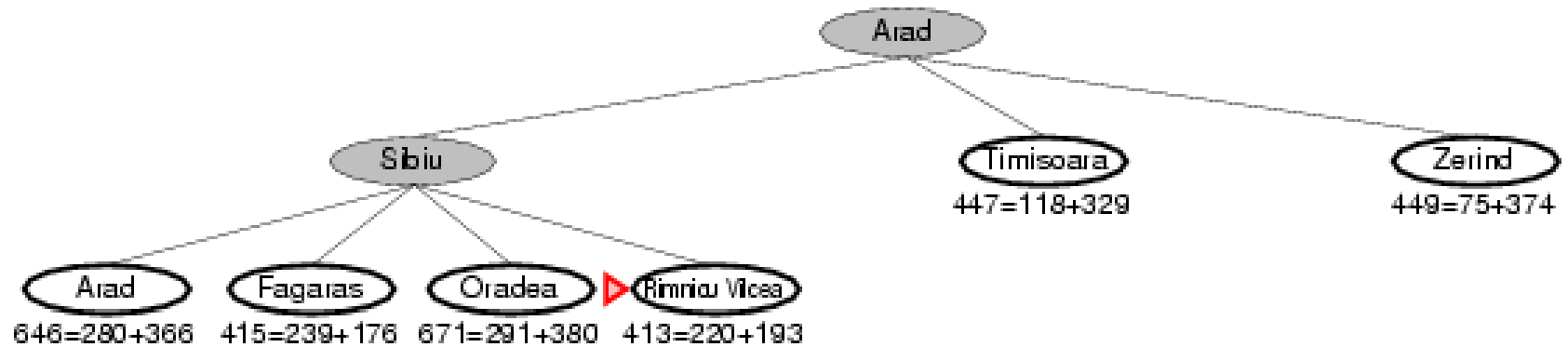
A* search example

▶ A_{rad}
366=0+366

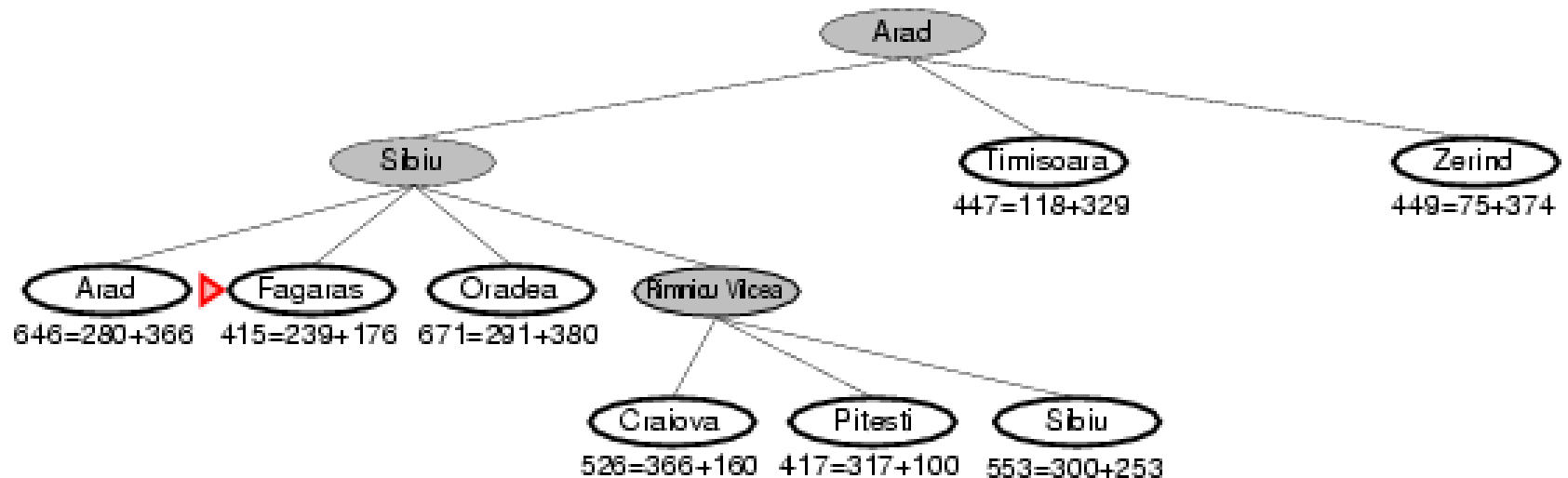
A* search example



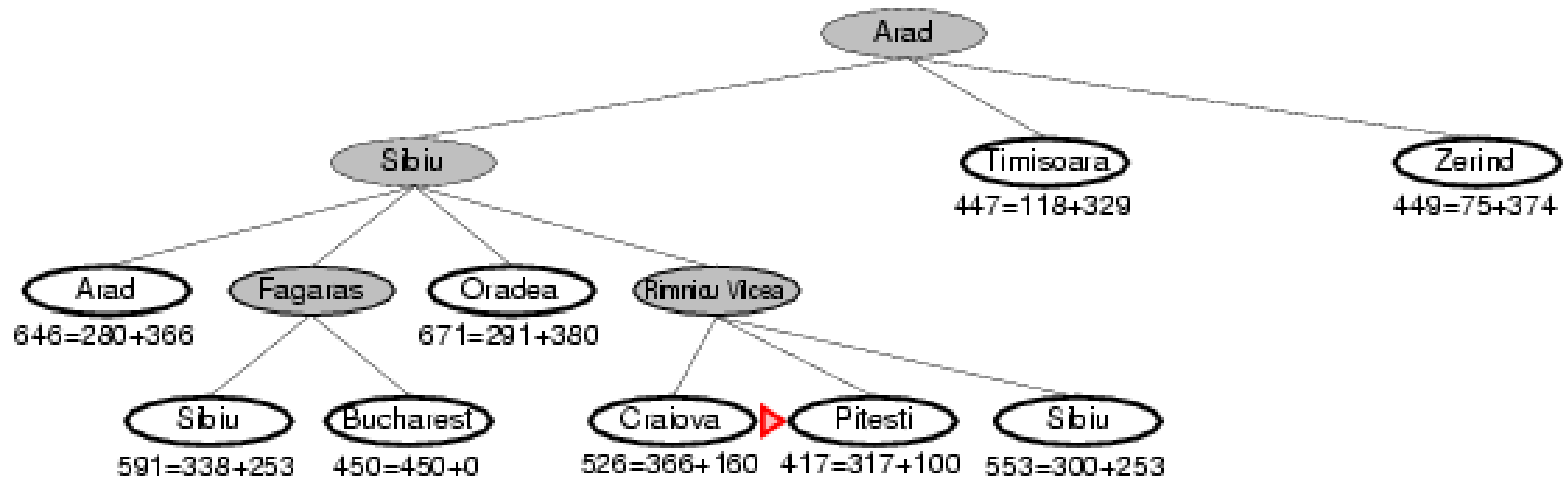
A* search example



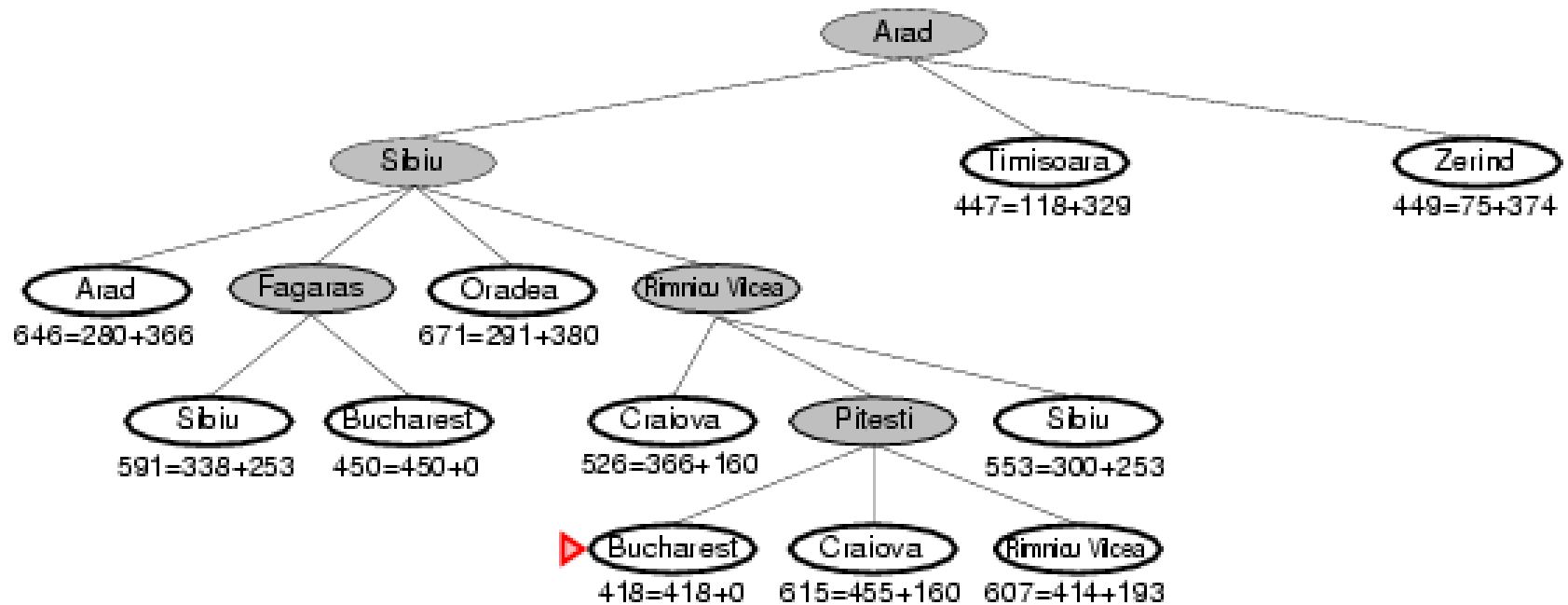
A* search example



A* search example



A* search example

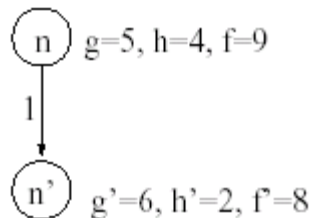


A* Aramasının Özellikleri

- Tamdır. Sonuçta, çözüm varsa bulunur. Kökten başlanarak gidilen herhangi bir yolda f tahmini her zaman artar (h , monoton olsa bile). Zaman karmaşıklığı Optimaldir.
- Zaman ve Yer karmaşıklıkları kötü olsa da iyi bir sezgi ile, düzelecektir.

Non-monotone Heuristics

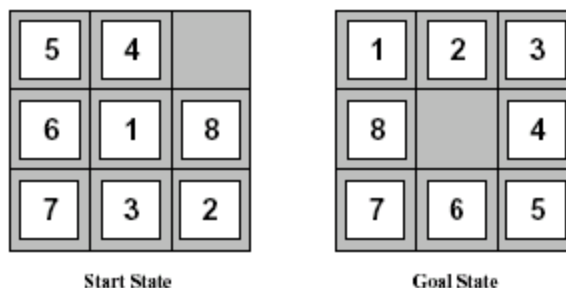
- Bazı sezgilerde f , yol boyunca düşebilir. n' , n den sonraki düğüm olmak üzere :



- $f(n)=9$ 'un anlamı, n aracılığı ile gidilebilen yolun maliyeti ≥ 9 , ve n' için de. Basit bir değişiklik, bu durumun önüne geçilerek, F' 'in yol boyunca azalmaması sağlanır.

$f(n')=g(n')+h(n')$ yerine $f(n')=\max(g(n')+h(n'), f(n))$
kullanılır

Example of admissible heuristics: 8-puzzle



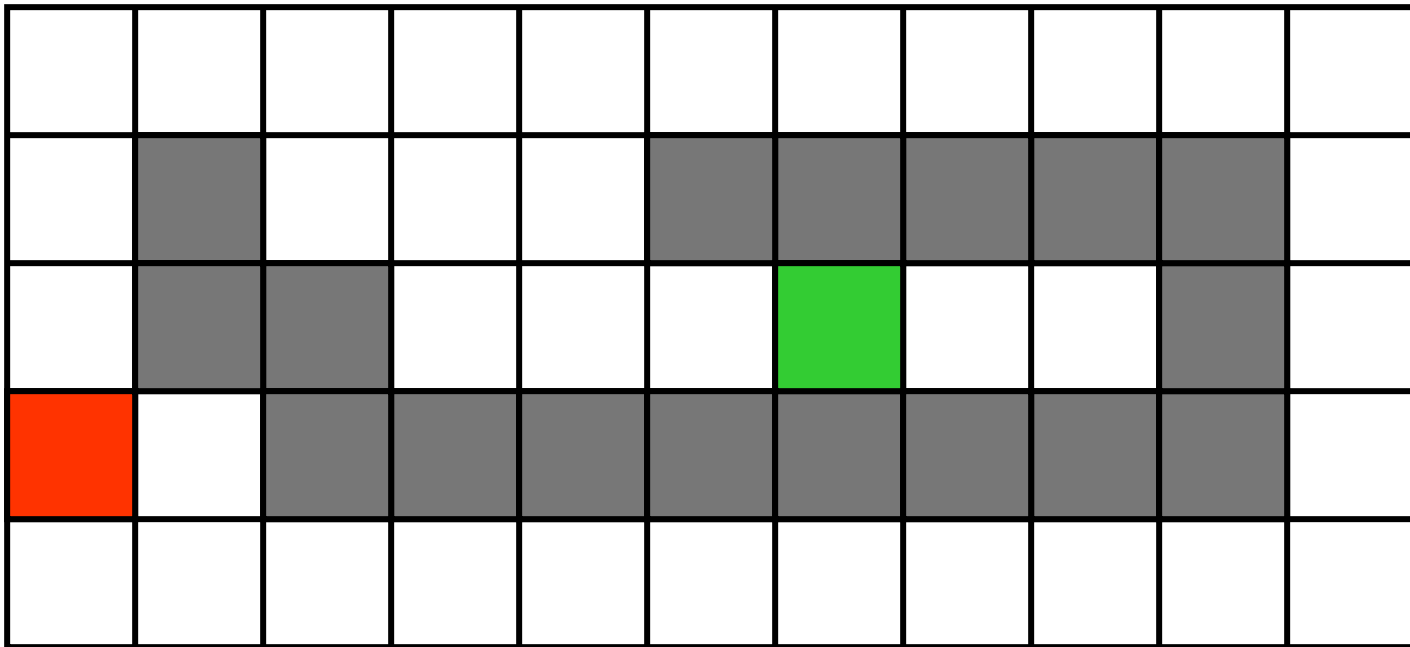
Two possible heuristics:

$h_1(n)$ = number of misplaced tiles = 7

$h_2(n)$ = total Manhattan distance (i.e., no. of squares from desired location of each tile) = $2+3+3+2+4+2+0+2 = 18$

Intuitively, h_2 seems better: it varies more across the state space, and its estimate is closer to the true cost.

Robot Navigation



Robot Navigation

$f(N) = h(N)$, with $h(N)$ = Manhattan distance to the goal

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

IDA* (Iterative-Deepening A*)

- DFS'ye dayanır. Bir düğümün çocuklarının açılma önceliğini belirlemede f değerinden yararlanır.
- f değeri sınırı vardır (derinlik limiti yerine)
- IDA*, A* ile aynı özellikleri taşır ama daha az bellek kullanır.

Özet

- Sezgisel arama yöntemleri, problem hakkındaki bilgiden yararlanırlar.
- Sezgi (Heuristic), hedefe ulaşmak için kalan maliyetin tahminidir.
- İyi bir sezgi, arama süresini, üstelden doğrusala indirir.
- Best-first Search tam da, optimal de değildir.
- A*, AI'da anahtar teknolojidir. $f=g+h$,
(g, harcanan miktar ve h, hedefe ulaşmada harcanması beklenen miktar)

Diğer Arama Teknikleri

- Gerçek-zamanlı Arama
- Optimizasyon Problemlerinde Arama
- CSP'de Arama

BÖLÜM III

YEREL ARAMA (Local Search)

Optimizasyon Problemleri

- Traveling Salesman Problem
- Optimizasyon Problemlerinde Arama
 - Constructive Methods
 - Iterative Improvement/Repair Methods

Yineli Geliştirme Algoritmaları (Iterative Improvement Algorithms)

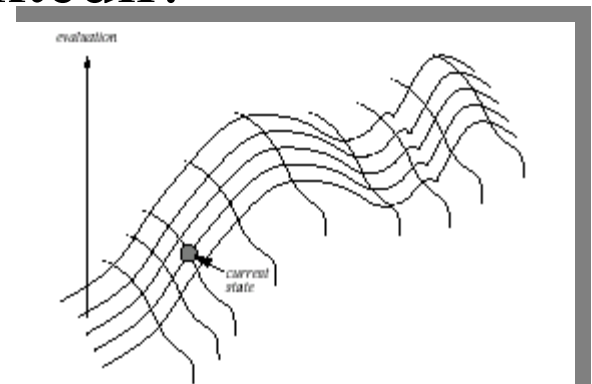
Local Search Algorithms (Yerel Arama Algoritmaları)

- Hill Climbing (Tepe Tırmanma) veya Gradient Descent (Eğim Düşümü)
- Tabu Search (Tabu Arama)
- Simulated Annealing
(Isıl İşlem, Tavlama Benzetimi)
- Local Beam Search
- Genetic Algorithms

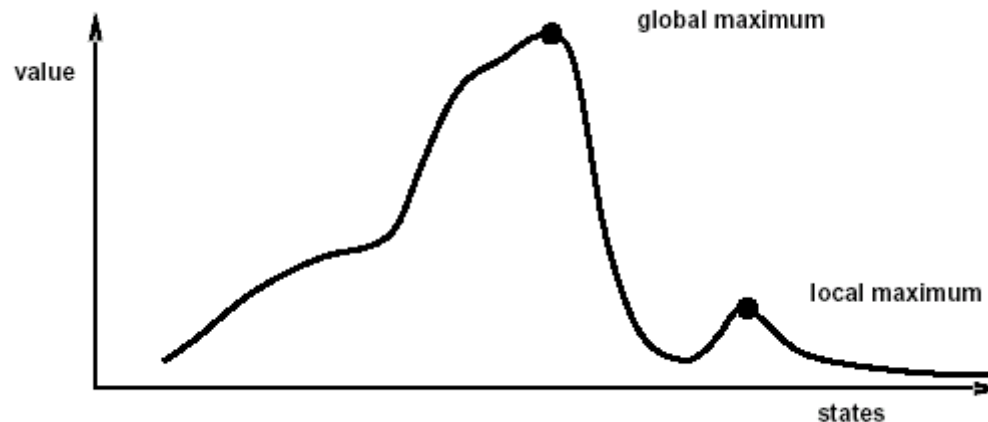
Hill Climbing

- Yoğun bir siste, Everest Dağına tırmanmaya benzer. Sadece etkin durumun bilgisini tutar.
- Ana düşünce : Her zaman, şimdiki değerlendirmeyi en fazla geliştiren yönde adım at.
- Best-first Search'e benzemektedir.

Öğrenme algoritmalarında popülerdir.



Hill Climbing : Problemleri

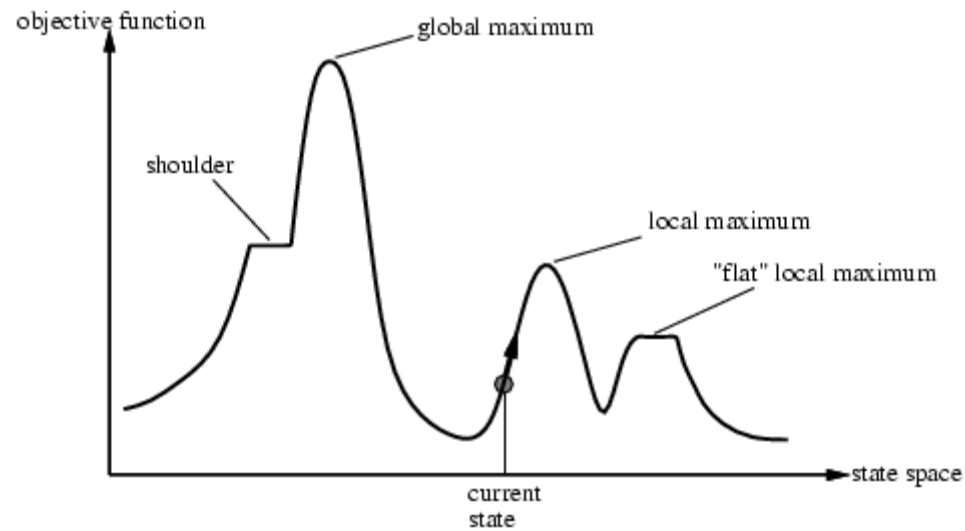


- Local Maximum'da şaşırabilir.
- Yaylada şaşabilir.
- Sıralı tepelerde şaşabilir.

Hızlı Çözüm

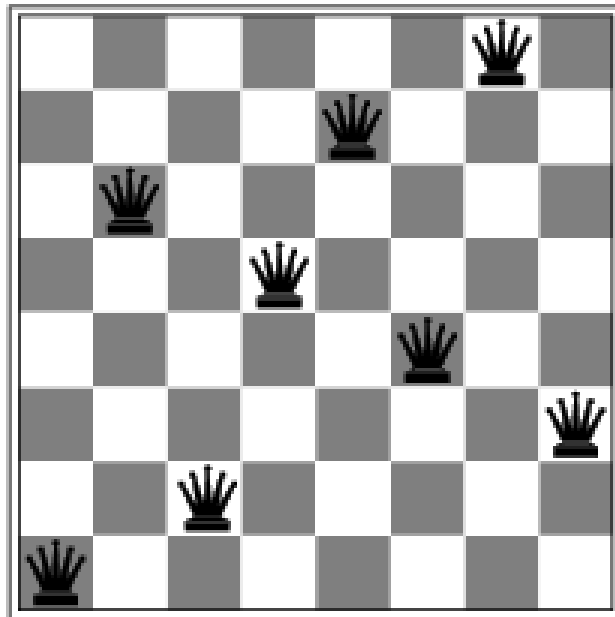
Random Restart

Hill Climbing : Problemleri



Hill-climbing search: 8-queens problem

A local minimum with $h = 1$



Stochastic Hill-climbing search: 8-queens problem

Stochastic hill climbing:
Pick next move **randomly**
from among uphill moves.
(many other
variations of hill climbing)

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♚	13	16	13	16
♚	14	17	15	♚	14	16	16
17	♚	16	18	15	♚	15	♚
18	14	♚	15	15	14	♚	16
14	14	13	17	12	14	12	18

h = number of pairs of queens that are attacking each other, either directly or indirectly

$h = 17$ for the above state

Operators: move queen in column - best move leads to $h=12$ (marked with square)

Hill Climbing Algorithm

```
function hillclimbing(node)
  node = selectrandomnode;
  loop
```

```
    if (node is goal) then return(node);
    successors = generatesuccessors(node);
    bestsuccessor = selectbestnode(successors);
    if (value(bestsuccesor) < value(node))
      then return(node);
      node = bestsuccessor;
  end loop
```

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```


Simulated Annealing

- Ana fikir : Yerel Maksimum'dan kaçmak için, istenmeyen hareketlere izin vermesidir. Yavaş yavaş boyutu ve frekansı azaltılır.
- Random hareket olasılığını kontrol eden T (Temperature) parametresi yeterince yavaş azaltılırsa, en iyi çözüme ulaşılması garantilenir.

SA Algorithm

- **Function** SIMULATED-ANNEALING(*Problem*, *Schedule*) **returns** a solution state
- **Inputs:** *Problem*, a problem
Schedule, a mapping from time to temperature
Local Variables : *Current*, a node
Next, a node
T, a “temperature” controlling the probability of downward steps
- *Current* = MAKE-NODE(INITIAL-STATE[*Problem*])

SA Algorithm

For $t = 1$ **to** ∞ **do**

$T = \textit{Schedule}[t]$

If $T = 0$ **then return** *Current*

Next = a randomly selected successor of *Current*

$\Delta E = \text{VALUE}[\textit{Next}] - \text{VALUE}[\textit{Current}]$

if $\Delta E > 0$ **then** *Current* = *Next*

else *Current* = *Next* only with probability $\exp(-\Delta E/T)$

Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to "temperature"
  local variables: current, a node
                    next, a node
                    T, a "temperature" controlling the probability of downward steps


  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  schedule[t]
    if T = 0 then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow$  VALUE[next] - VALUE[current]
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```

BÖLÜM IV

KISIT SAĞLAMA PROBLEMLERİ (Constraint Satisfaction Problems)

CSP (Constraint Satisfaction Problems) Uygulamaları

- Gerçek CSP'ler
 - Zaman Çizelgesi Problemleri
 - Donanım Konfigürasyonu
 - Nakliye Planlama
 - Fabrika Planlama
 - Kat Planlama



Min-conflicts
Heuristic

Hill Climbing ve Simulated Annealing, tüm durumlarla çalışır. CSP'lere uygulamada, çığnenen kısıtlamaların sayısı ile Hill Climbing uygun olabilir.

BÖLÜM V

RAKİPLİ ORTAMLARDA ARAMA
(ADVERSARIAL SEARCH)

veya

OYUN OYNAMA
(GAME PLAYING)

5. Game Playing (Oyun Oynama)

- AI'nın en eski ve iyi çalışılmış alanlarındanıdır.
- Nedenleri
 - İnsanlar oyun oynamaktan hoşlanır.
 - Zeka göstergesi kabul edilir.
 - Ortamın net bir tanımı vardır.
 - Sonuçları kolay görülür.
- Oyun Türleri (Satranç, Dama, Tavla, Briç, ...)
 - Perfect vs Imperfect Info (ChessxBridge Hidden Cards)
 - Deterministic vs Stochastic (ChessxBackgammon) ?

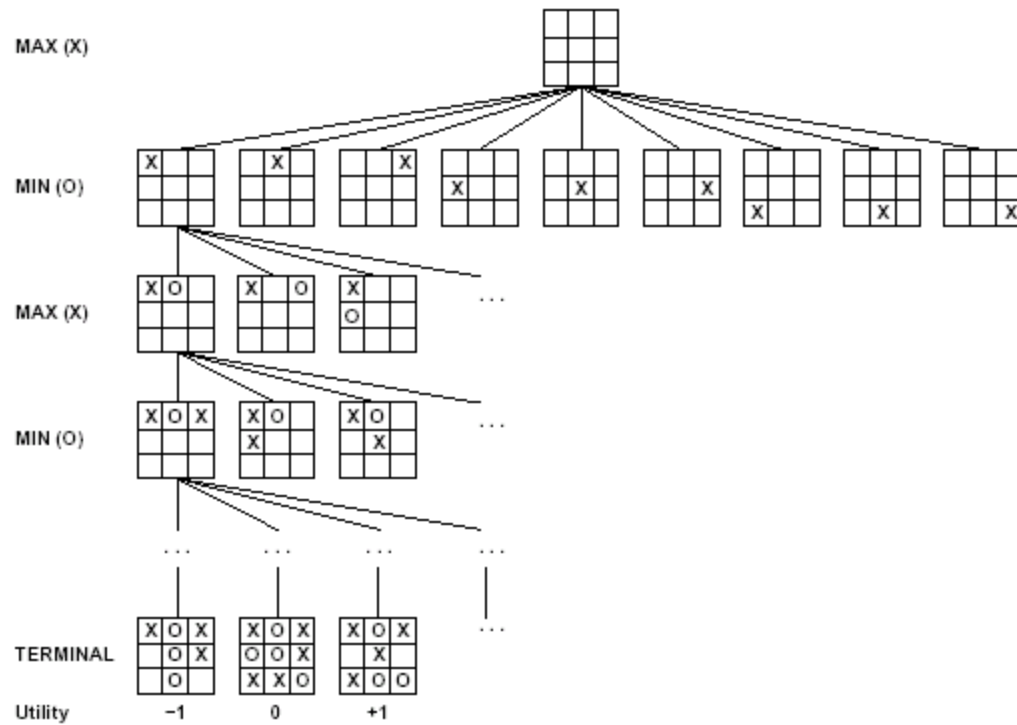
Arama Problemi Olarak Oyun Oynama

- Tahtanın Durumu
- Geçerli Hareketler
- Uç Durumlar
- Strateji

Rakibin Durumu da düşünülmeli!

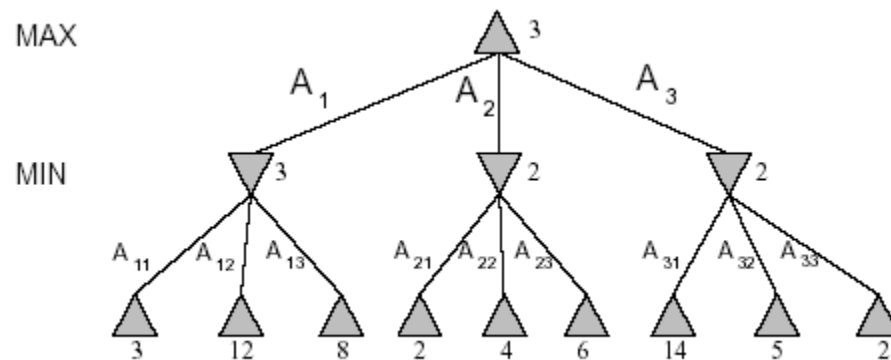
- Oyuncunun hareketlerini en iyiye, rakibin hareketlerini en kötüye getiren durumlar.

Example: Tic-Tac-Toe

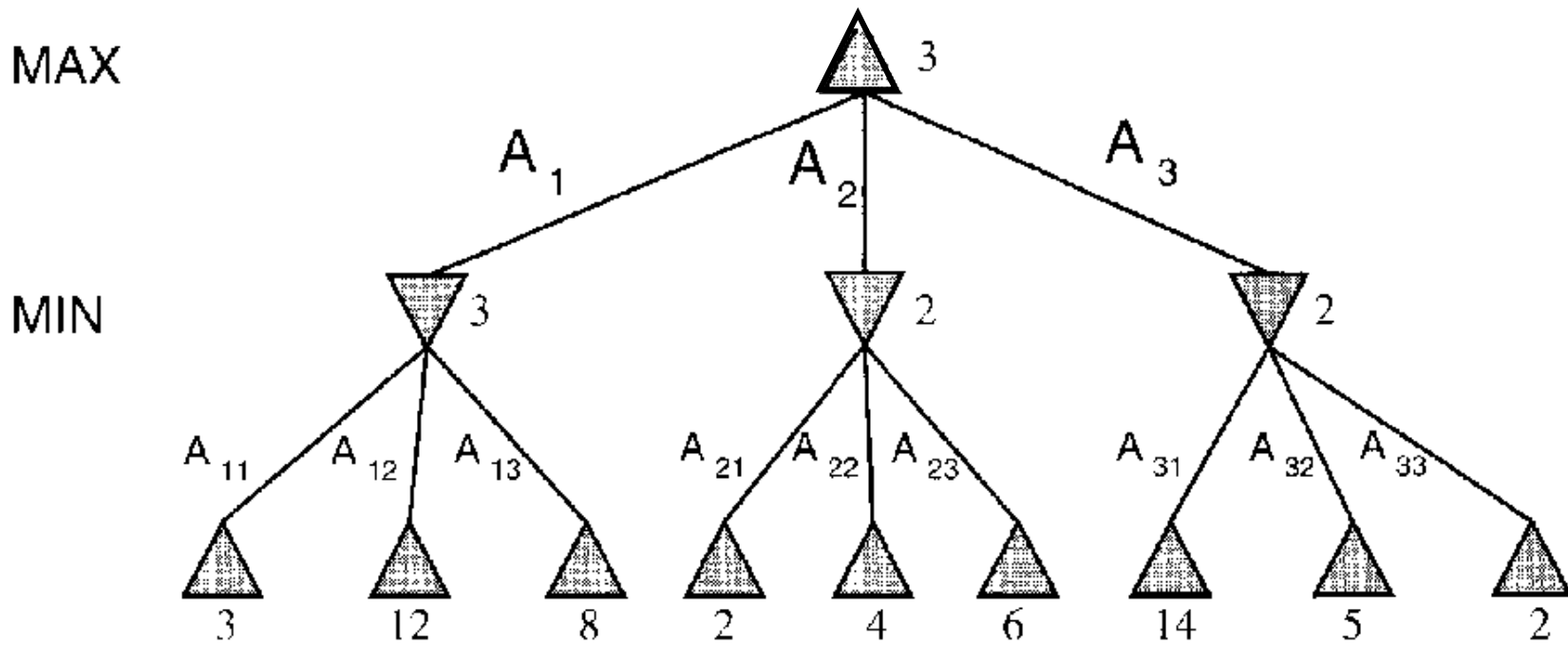


Minimax Search

- Uç durumlara ulaşılan kadar tüm arama ağacı açılır.
- Yapraklardan şimdiki duruma doğru geriye dönülür.
- Her min. Düğümünde, çocuk düğümlerin en kötüsü, max düğümün de en iyisi tutulur.
- Oyun ağacı sonlu ise tamdır. Satranç?



Minimax Search



Kaynak Kısıtlamaları ile Başa Çıkmak

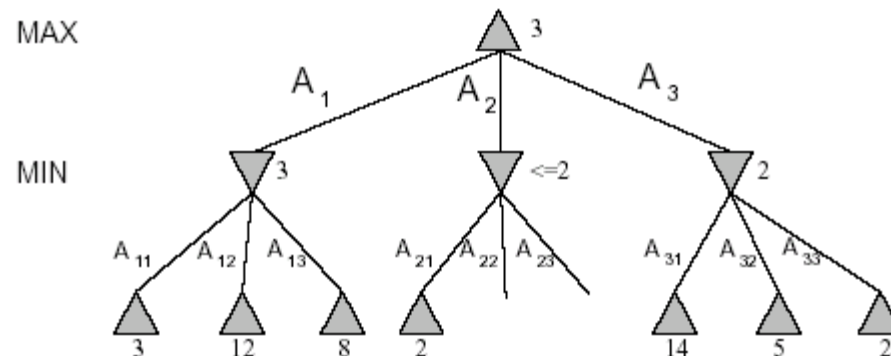
- Hareket için 100 saniyeniz varsa ve saniyede 10^4 düğüm dolaşabiliyorsanız, hareket etmeden önce 10^6 düğüm dolaşabilirsiniz.
- Standart Yaklaşım
 - Cutoff Test (derinlik sınırlandırmasına dayalı)
 - Evaluation Function (düğüm için, aramanın kesileceği)

Alpha-Beta Algorithm

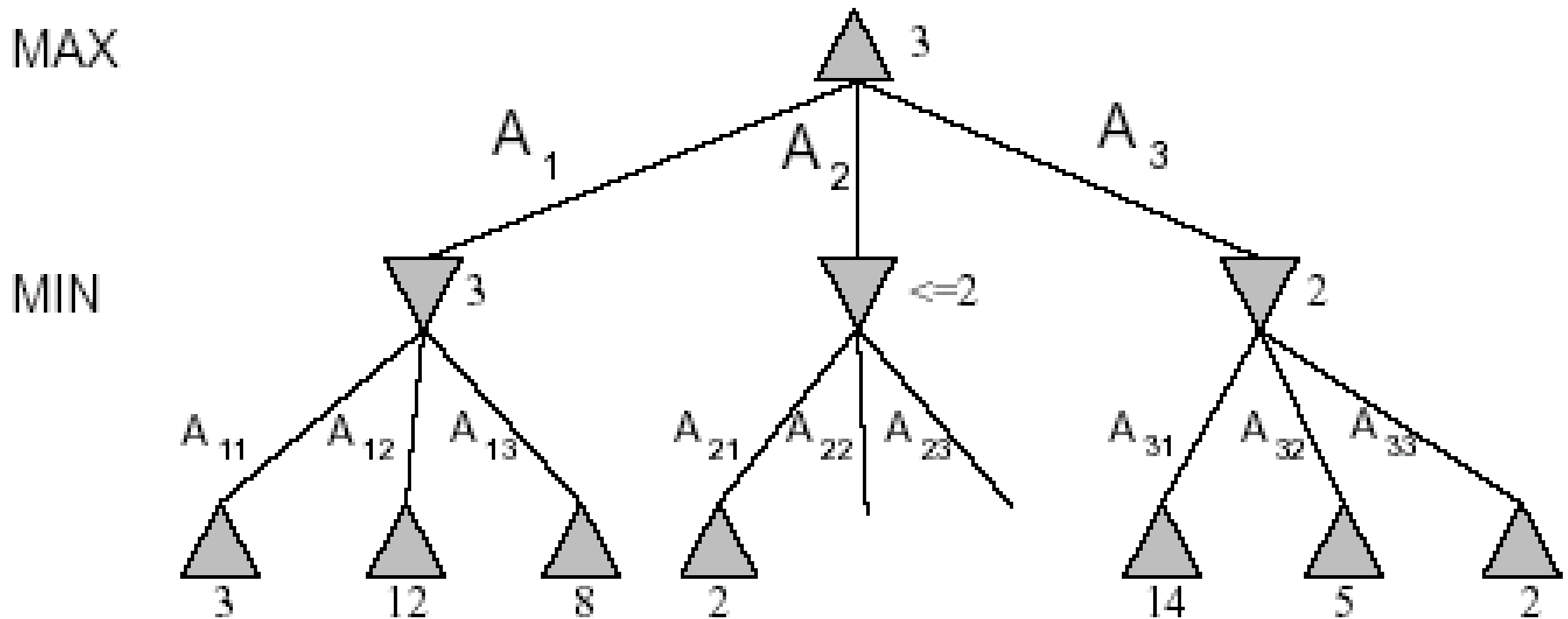
- Minimax'a benzer.
- Farklı olarak, arama ağacının sonuçla ilgili olmayan dallarını budar.
- Tüm oyun ağacını açmak (alpha-beta dahil) pek önerilmez. Bir noktada kesilip, ona göre değerlendirilmelidir.
- Şans oyunları da, minimax algoritmasına ek yapılarak gerçekleştirilir.

Alpha Beta Pruning (Budama)

- Deterministic, Perfect Information Oyunlar için kullanılan standart bir yöntemdir.
- Düşünce Alpha Pruning'e benzer : Bir yol, şimdiki durumdan daha kötüyse elenir.
- Minimax'a da benzer ancak, oyuncunun ve rakibin en iyi yaprak değerinin izini tutar. Örnek :



Alpha Beta Pruning (Budama)



Alpha-Beta Algorithm

Instead of MinimaxValue, now we have MaxValue and MinValue

double MaxValue(s, α, β)

1. if *cutoff*(s) return *evaluation*(s)
2. for each $s' \in \text{Successors}(s)$
 - (a) $\alpha \leftarrow \text{Max}(\alpha, \text{MinValue}(s', \alpha, \beta))$
 - (b) if $\alpha \geq \beta$ return β
3. return α

double MinValue(s, α, β)

1. if *cutoff*(s) return *evaluation*(s)
2. for each $s' \in \text{Successors}(s)$
 - (a) $\beta \leftarrow \text{Min}(\beta, \text{MaxValue}(s', \alpha, \beta))$
 - (b) if $\alpha \geq \beta$ return α
3. return β

Alfa Beta Budama'nın Özellikleri

- Budama, sonucu değiştirmez.
- Arama derinliğini, aynı kaynaklarla iki katına çıkarır.
- Satranç gibi oyunlarda,
 - Novice Player (Acemi Oyuncu)
 - Expert Player (Uzman Oyuncu)farkını belirler.

Özet

- Oyunlar eğlencelidir ve zaman alıcıdır.
- AI'ın birçok önemli yönünü ortaya koyar.
- Mükemmel duruma ulaşamaz \Rightarrow Yaklaşım
- Grand Prix, otomobil tasarımı için ne ise, Oyunlar da AI için odur.