



YZM 3217 - YAPAY ZEKA

DERS#4: -UNINFORMED

SEARCH METHODS-

BİLGİSİZ ARAMA

YÖNTEMLERİ

Bilgisiz Arama Stratejisi- Uninformed Search Methods

- Sadece problem formülasyonundaki mevcut bilgiyi kullanır
 - Durum bilgisinden yararlanmazlar
 - Çözüme ulaşmak için hiçbir bilgi verilmmez
- Aramanın herhangi bir adımında
 - çözüme ne kadar yakın (veya uzak) olduğu veya
 - çözümün bulunabileceği hakkında fikir söylemek mümkün değildir

Bilgisiz Arama Stratejisi- Uninformed Search Methods

- Uses only existing knowledge in problem formulation
 - They don't use status information
 - No information is given to reach the solution
- Any step of the search
 - how close (or far) it is to the solution, or
 - It is not possible to give an opinion on whether a solution can be found.

Bilgisiz Arama Yöntemleri- Uninformed Search Methods

- Genişlik-öncelikli (Breadth-first)
- Eşit-maliyetli (Uniform-cost)
- Derinlik-öncelikli (Depth-first)
- Derinlik-sınırlı (Depth-limited)
- Yinelemeli Derinleşen (Iterative deepening)
- İki Yönlü (Bi-directional)

Hedefler

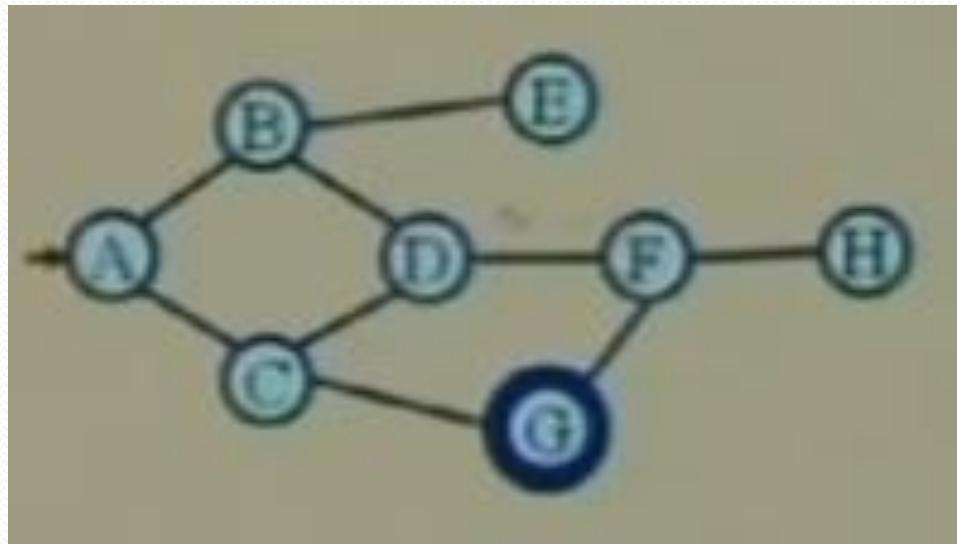
- Bu dersin sonunda beklenen kazanımlar
 - Verilen problemi analiz ederek problem için en uygun arama stratejisini tanımlayabilme
 - Verilen probleme çözüm bulmak amacıyla bu stratejilerden birini uygulayabilme

Goals

- Expected outcomes at the end of this course
 - Ability to identify the most appropriate search strategy for the problem by analyzing the given problem
 - Ability to apply one of these strategies to find a solution to a given problem

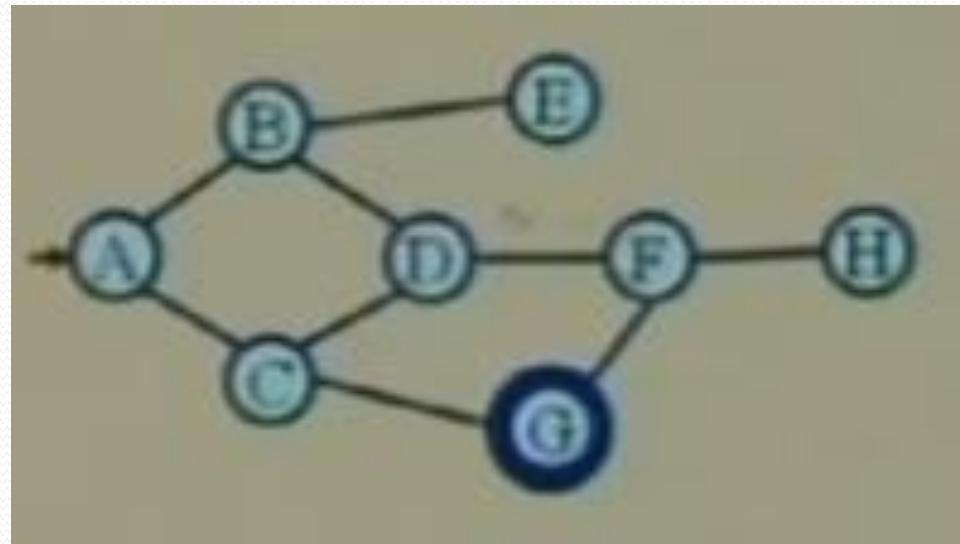
Yol Bulma

- En kısa yol
- Herhangi bir yol
- Kör Arama
 - BFS
 - DFS



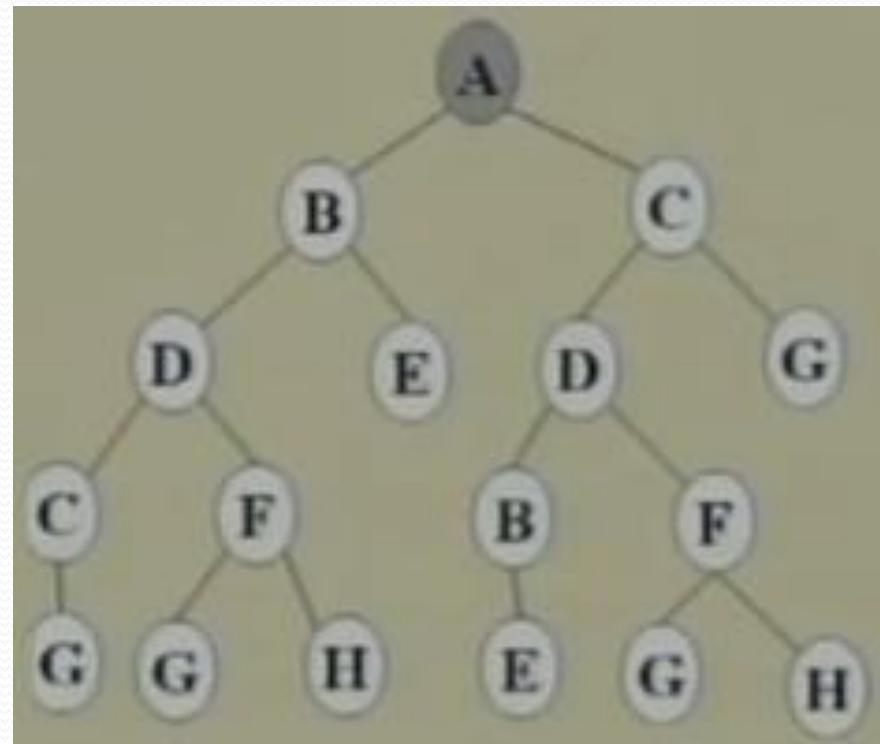
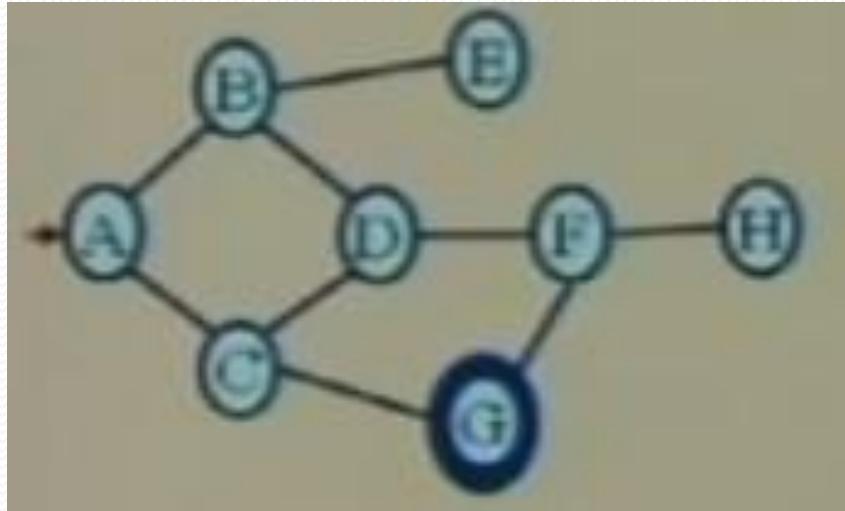
Road/Path Finding

- shortest route
- any route
- Blind Search
 - BFS
 - DFS



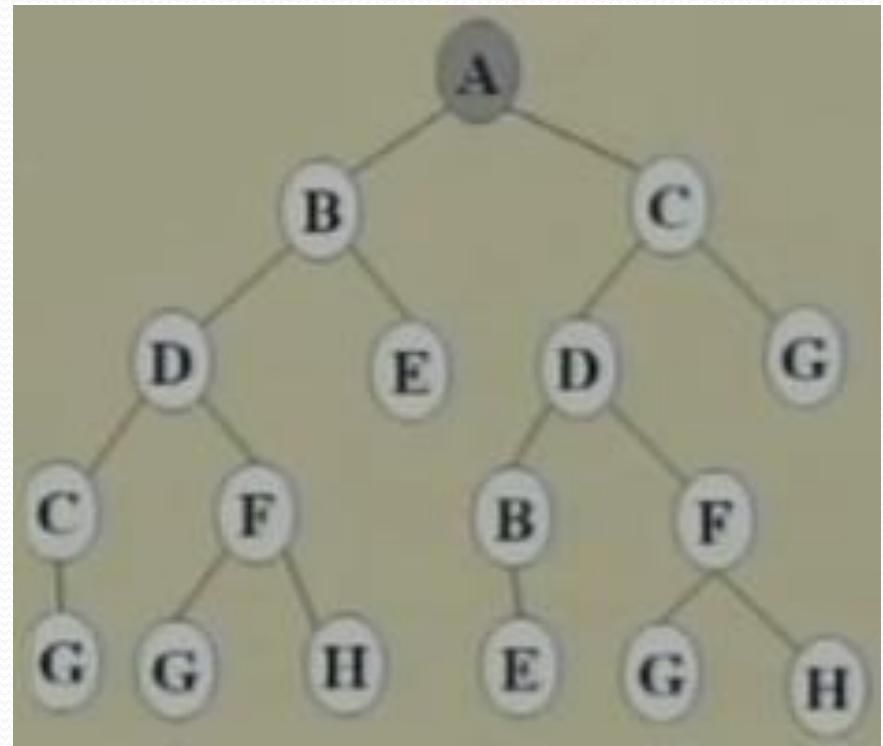
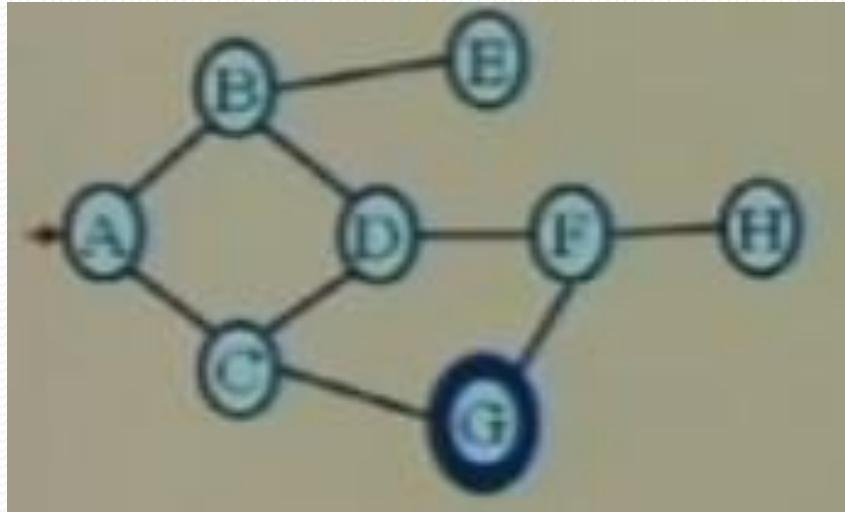
Arama Ağacı

- Tüm mümkün yolları listele
- Döngüleri çıkar
- Sonuç: Arama ağacı



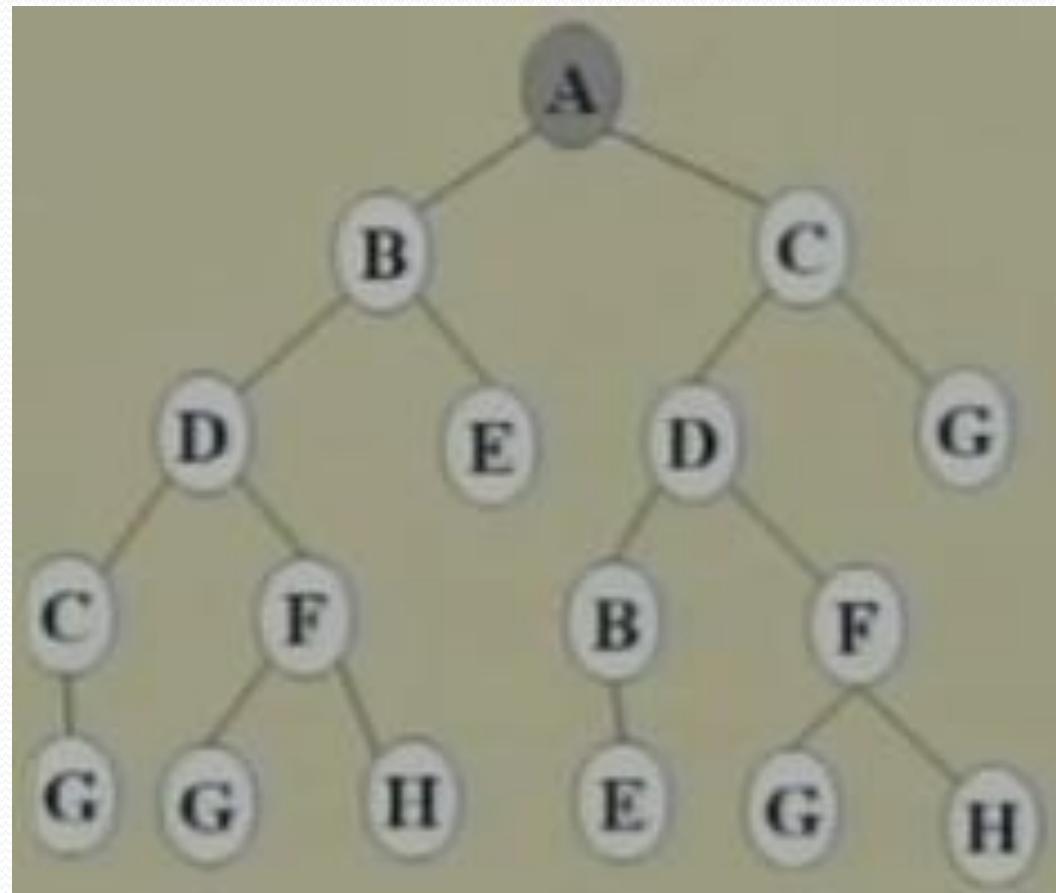
Search Tree

- List all possible ways
- Subtract the loops
- Result: Search tree



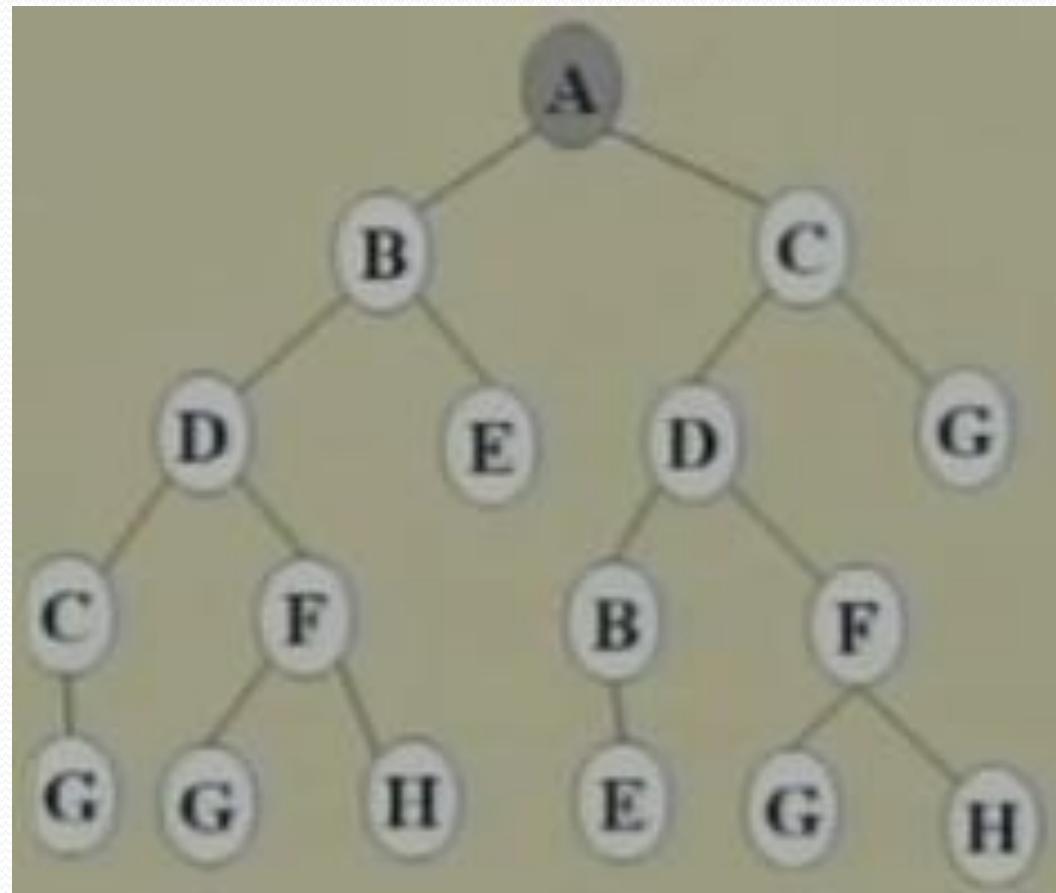
Arama Ağacı - Terminoloji

- Kök düğüm
- Yaprak Düğüm
- Ata / Torun
- Dallanma Faktörü
- Tam yol / Kısmi yol
- Açık düğümleri açmak kapalı düşüme sebep olur



Search Tree- Terminology

- Root node
- Leaf node
- Ancestor / Grandchild
- Branching Factor
- Full path / Partial path
- Opening open nodes causes a closed node



Temel Arama Algoritması

- Fringe, başlangıç durumunu tutan liste olsun
- Döngü
 - Eğer fringe boş ise return hata
 - Node $\leftarrow \text{remove_first}(\text{fringe})$
 - Eğer Node hedef düğüm ise
 - Node düğümüne kadar olan yolу döndür
 - Değilse Node düğümünün tüm successor 'larını oluştur ve yeni oluşturulan düğümleri Fringe'e ekle
- Döngüyü Bitir

Basic Search Algorithm

- Let fringe be the list holding the initial state
- Loop
 - If fringe is empty return error
 - Node \leftarrow remove_first (fringe)
 - If Node is the target node
 - Return the path to the node
 - If not, create all successors of Node node and add newly created nodes to Fringe
- End Loop

Arama stratejisi

- Problem çözme performansının ölçülmesi:
 - **Completeness:** Arama stratejisi çözümün olduğu durumda bir çözüm bulmayı garanti ediyor mu?
 - **Optimality:** Algoritma en az maliyetli çözümü bulmayı garnati ediyor mu?
 - **Zaman Karmaşıklığı:** Çözüm bulmak amacıyla en iyi veya ortalama bir durumda algoritma ne kadar zaman harcıyor (açılan düğüm sayısı)
 - **Uzay Karmaşıklığı:** Algoritmanın ihtiyaç duyduğu hafıza miktarı (fringe'in maksimum boyutu ile ölçülür)
 - **Fringe:** Algoritmanın takip etmek zorunda olduğu bir listedir.
 - Fringe'in boyutu en kötü veya ortalama bir durumda ne olabilir sorusunun cevabı ile algoritmanın verimliliğini ölçübiliriz.

Completeness: The search strategy finds a solution where there is a solution. Does it guarantee finding?

Optimality: The algorithm aims to find the least costly solution. does it?

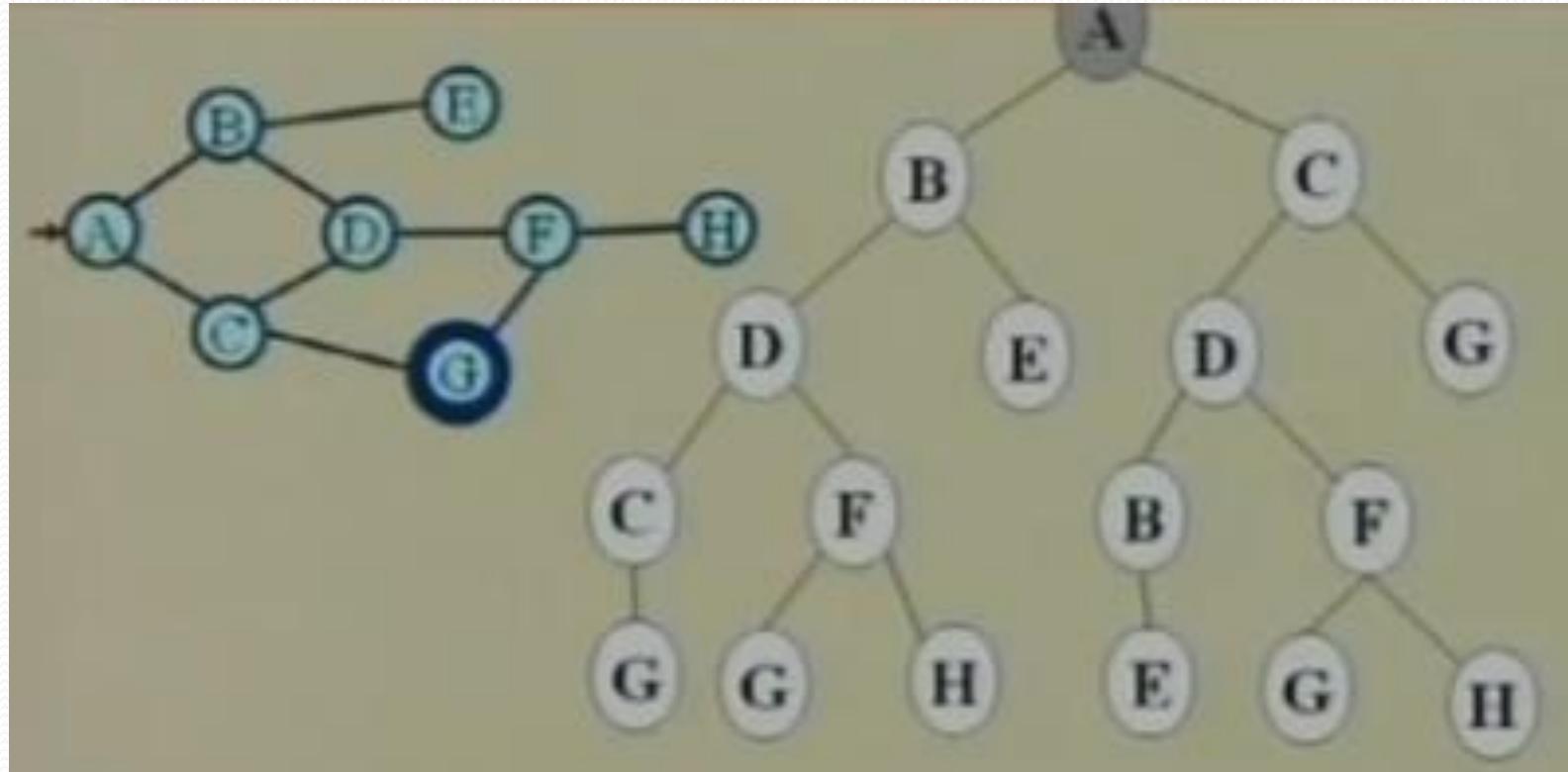
Time Complexity: Best or average time to find solution how much time does the algorithm spend in a situation (the opened node number)

Space Complexity: Amount of memory required by the algorithm(measured by the maximum size of the fringe)

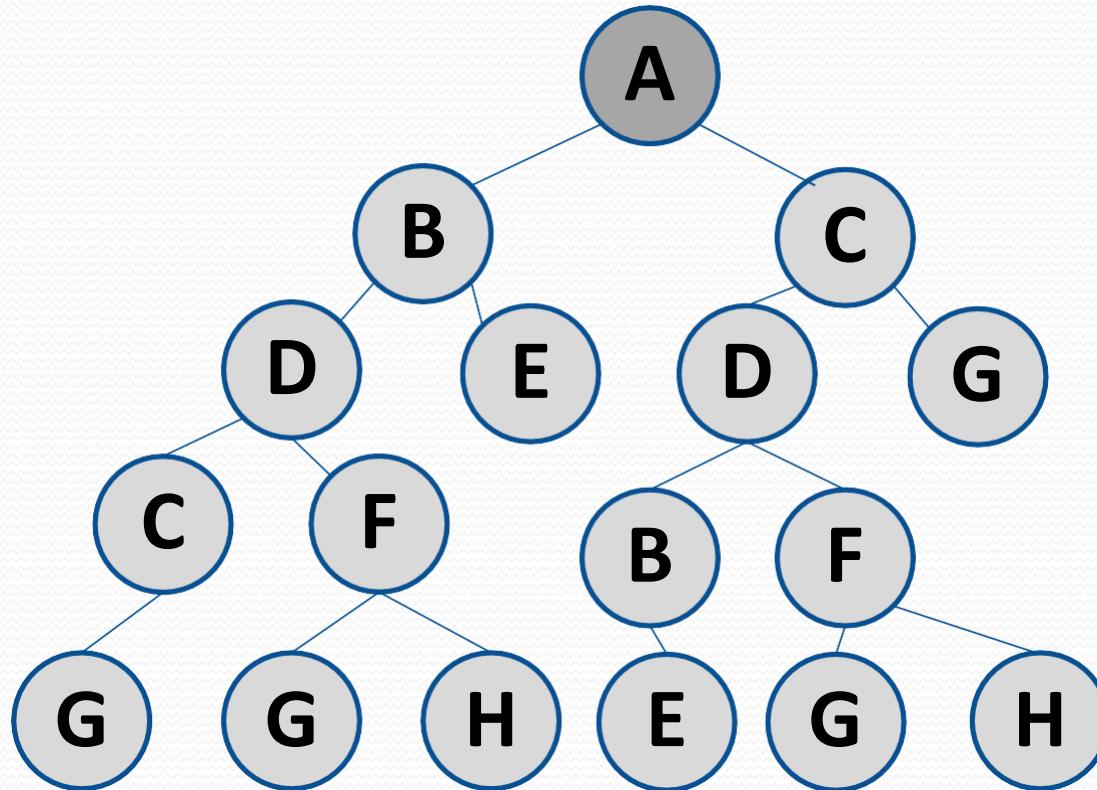
Breadth First Search

- Fringe, başlangıç durumunu tutan liste olsun
- Döngü
 - Eğer fringe boş ise return hata
 - Node $\leftarrow \text{remove_first}(\text{fringe})$
 - Eğer Node hedef düğüm ise
 - Node düğümüne kadar olan yolu döndür
 - Değilse Node düğümünün tüm successor 'larını oluştur ve
**seviye olarak en yakındaki düğümü önce aç
yeni oluşturulan düğümleri Fringe'in sonuna ekle**
- Döngüyü Bitir

Breadth First Search

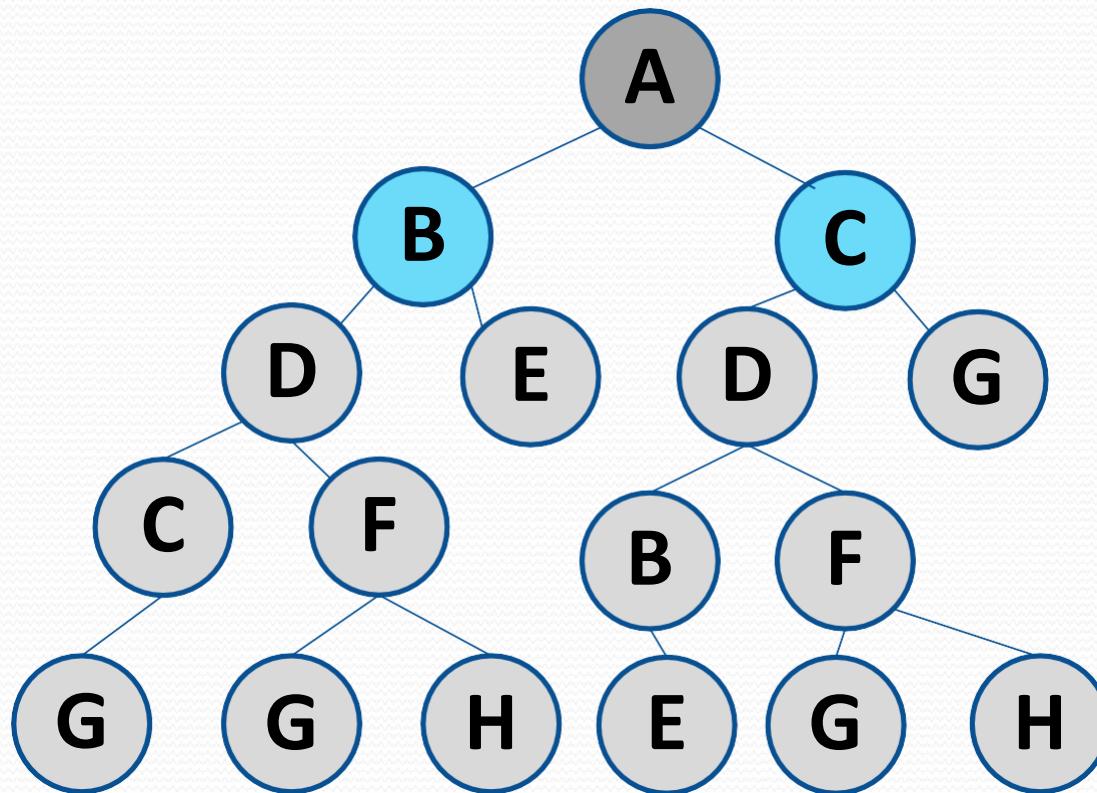


Breadth First Search



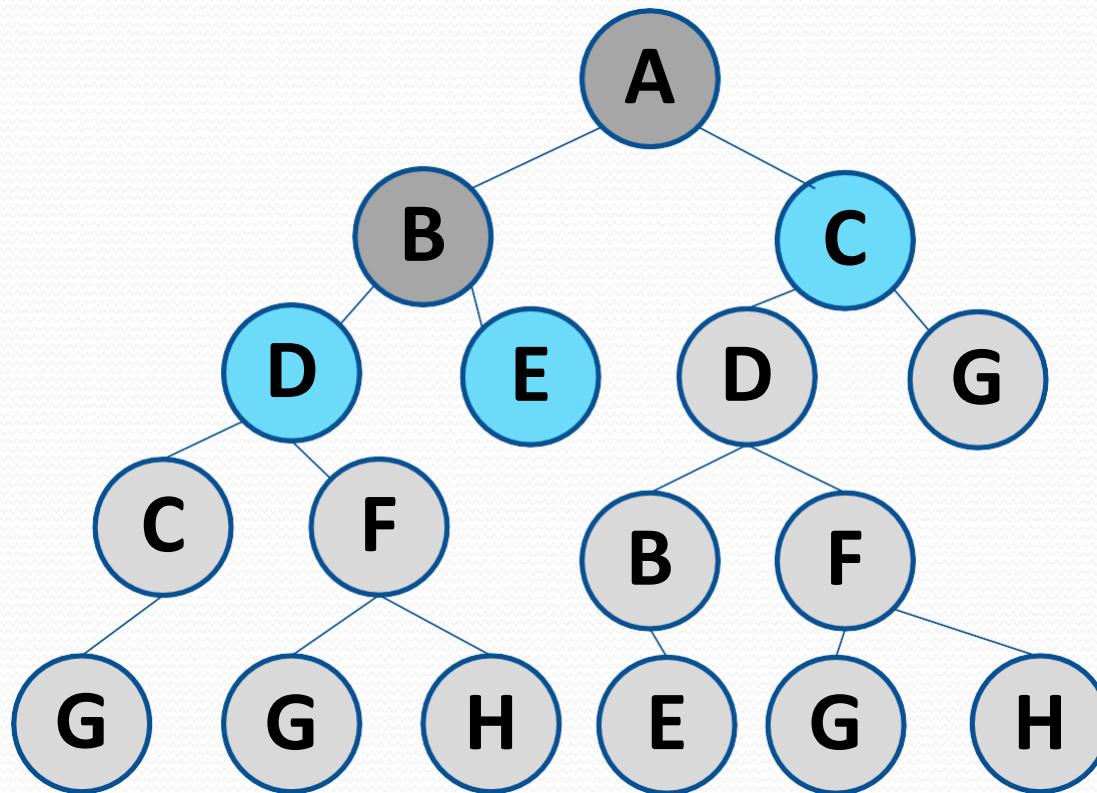
FRINGE : A

Breadth First Search



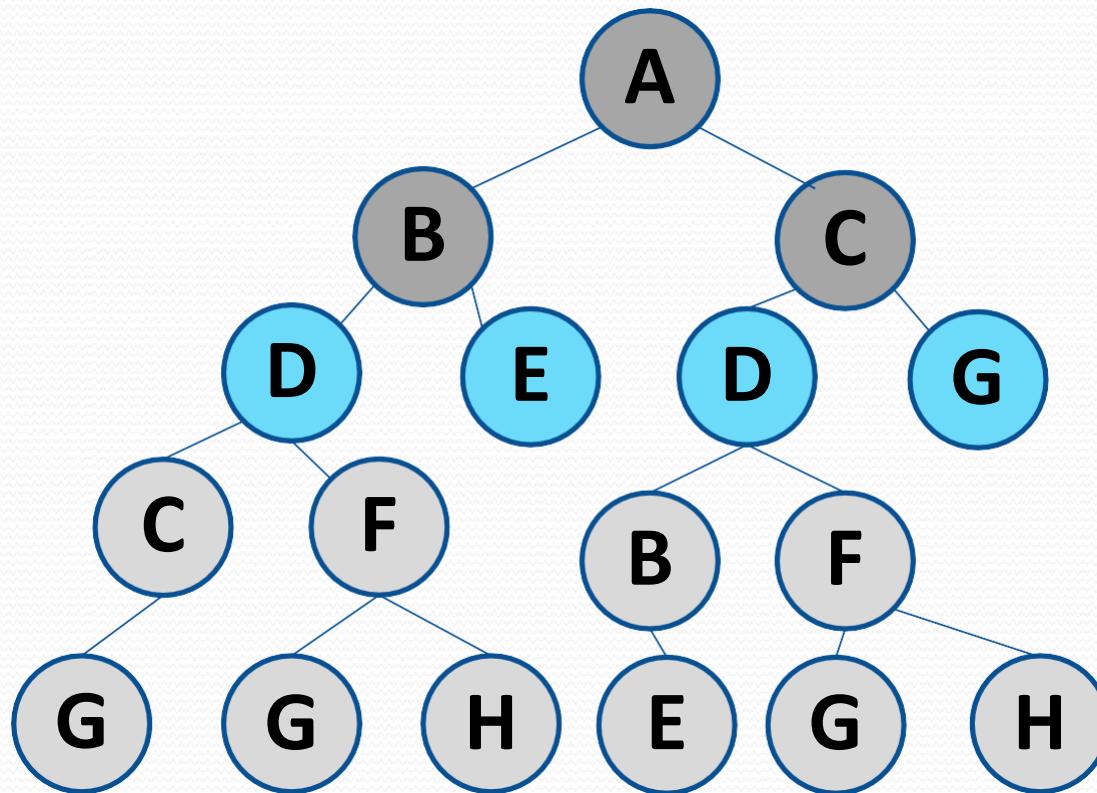
FRINGE : B C

Breadth First Search



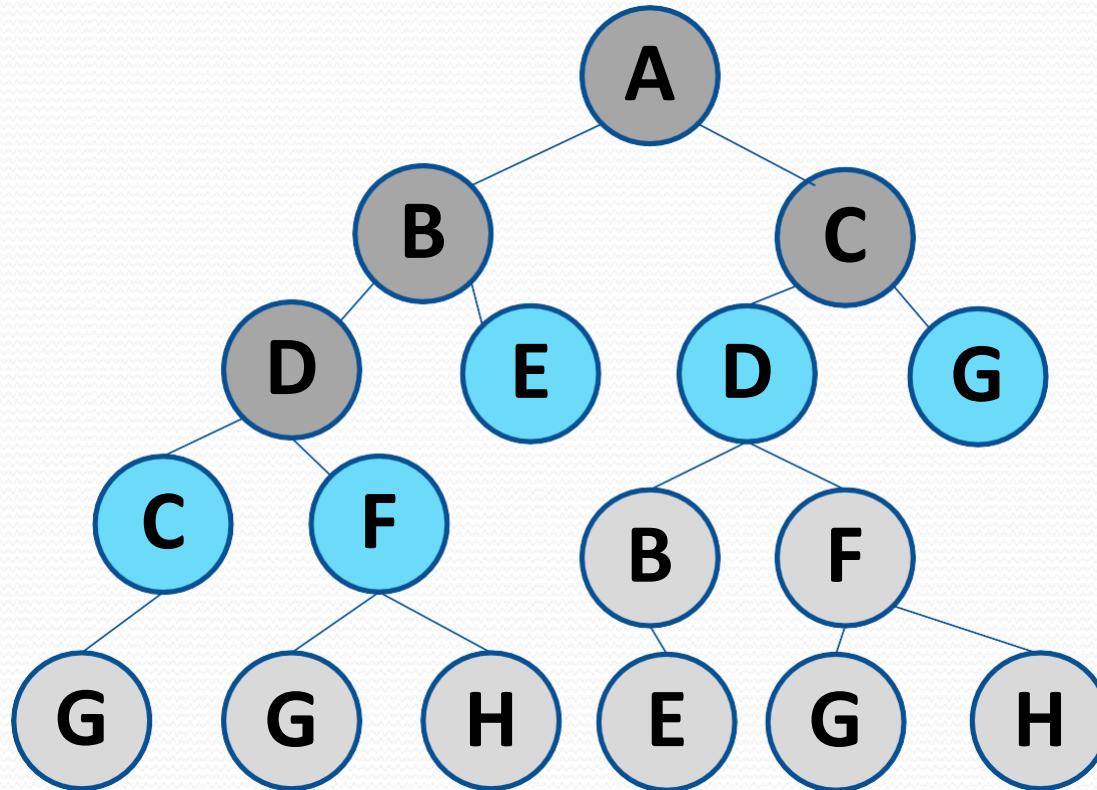
FRINGE : C D E

Breadth First Search



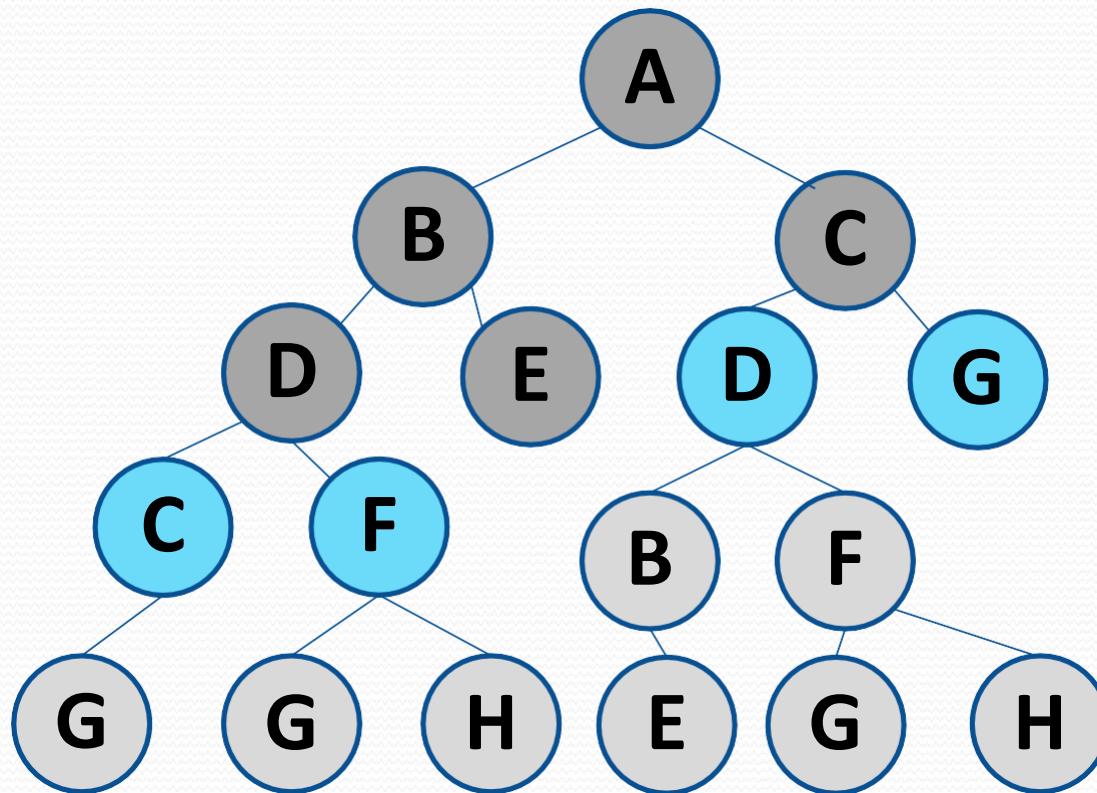
FRINGE : D E D G

Breadth First Search



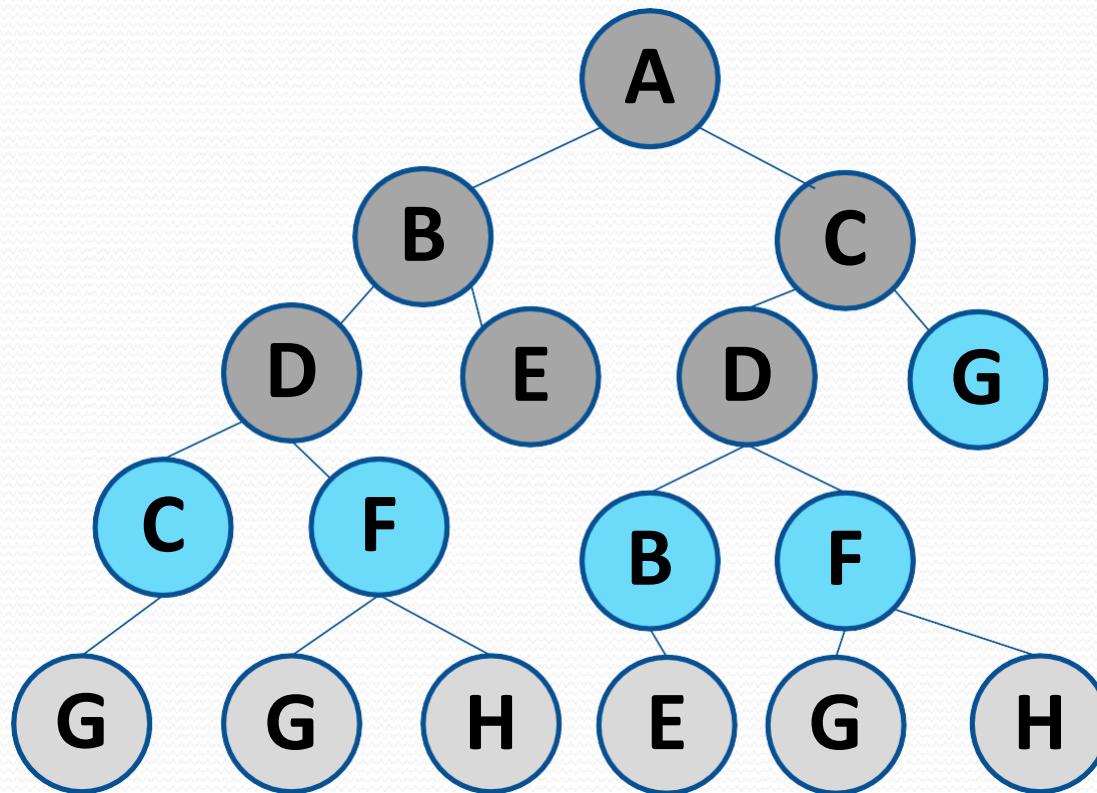
FRINGE : E D G C F

Breadth First Search



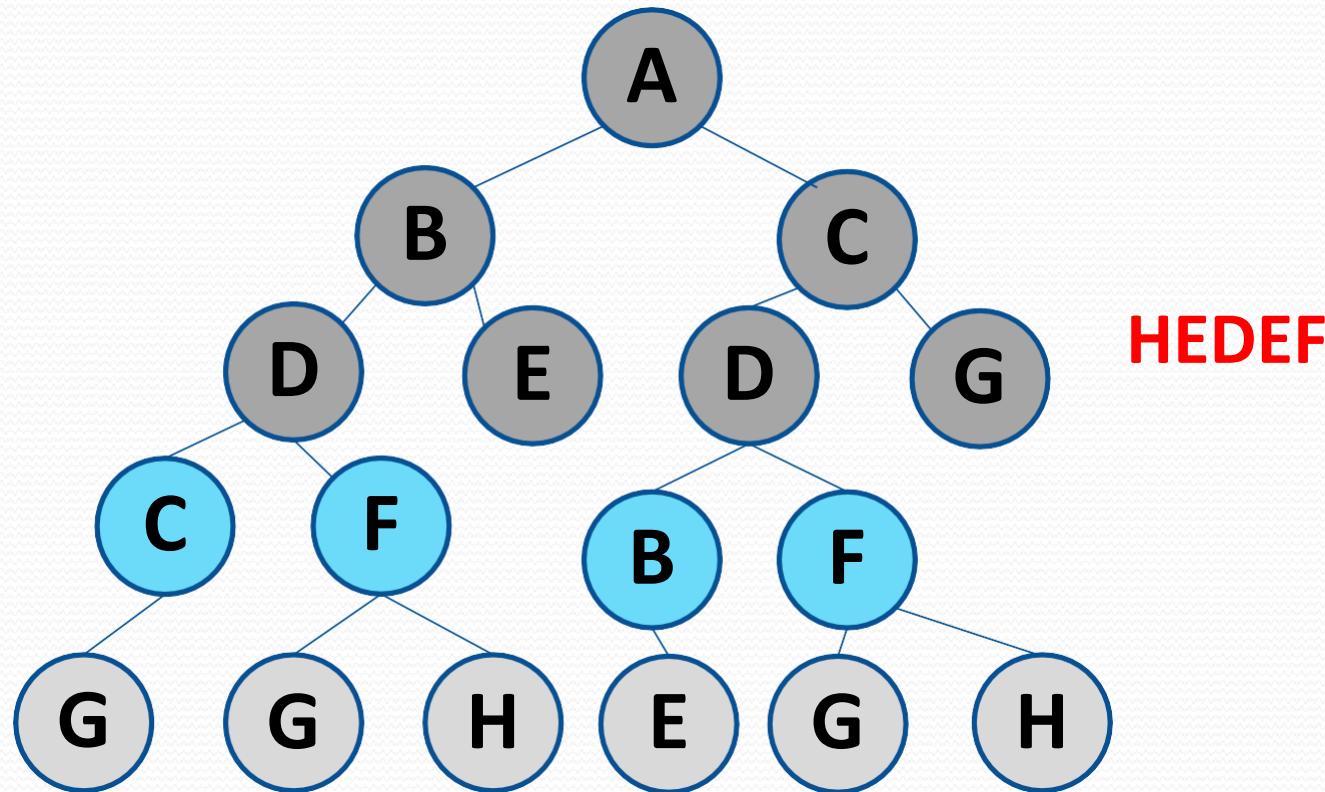
FRINGE : D G C F

Breadth First Search



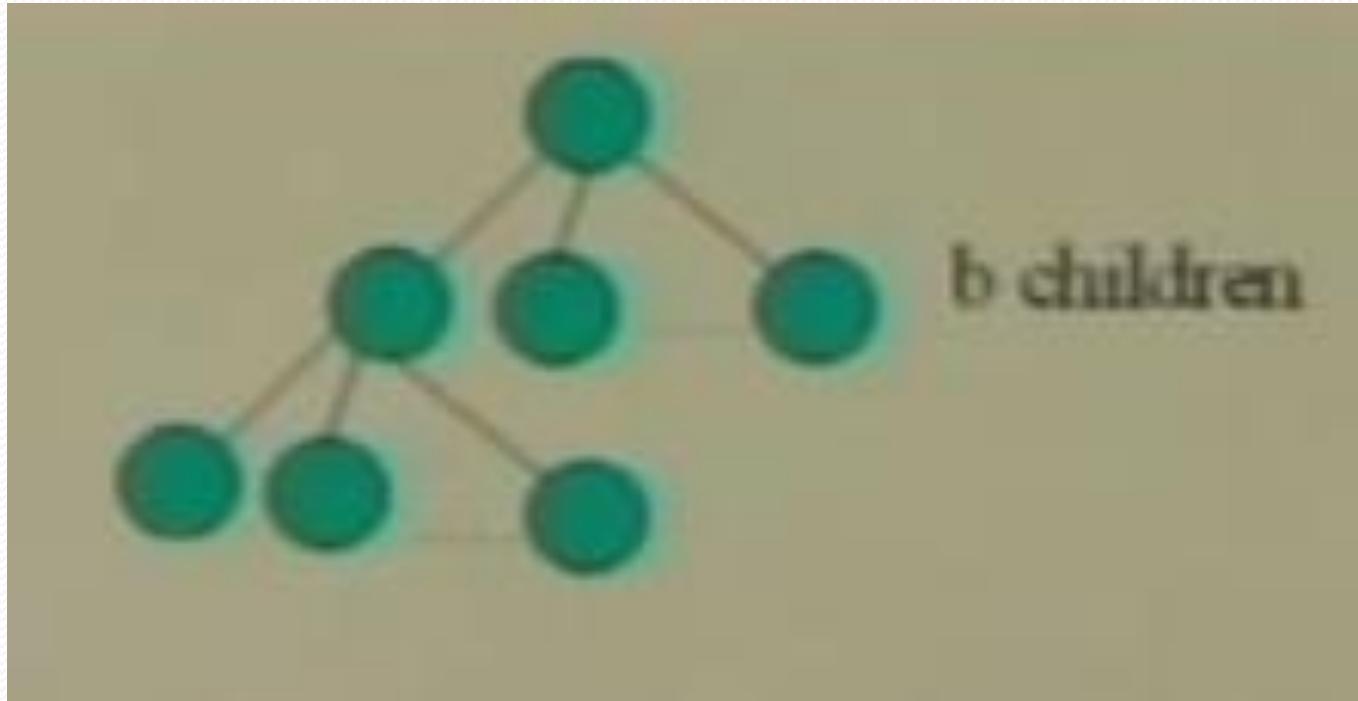
FRINGE : G C F B F

Breadth First Search



FRINGE : C F B F

Breadth First Search



b: en yakındaki hedef düğümün derinliği

m: arama ağacının maksimum derinliği

Breadth First Search

- Fringe'teki düğümler FIFO sırasıyla listeden çıkarılır.
Nodes in the Fringe are excluded from the list in FIFO order

- Complete

- Optimal : eğer tüm operatörler aynı maliyete sahip ise olabilir. Diğer durumda en kısa yol uzunluğuna sahip çözümü bulur.

It can happen if all operators have the same cost. In the other case, it finds the solution with the shortest path length.

- Zaman ve uzay karmaşıklığı: $O(b^{d+1})$, d: çözümün derinliği, b ise her düğümdeki dallanma faktörü (her düğümün max successor sayısı)

the depth of the solution, and b is the branching factor at each node (The maximum number of successors of each node)

Breadth First Search

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

- Dallanma faktörü $b=10$, saniyede 10.000 düğüm açılıyor, ve bir düğüm 1000 byte yer kaplıyor ise.
- 8. seviyede bir çözüm için 31 saat beklenebilir ancak terabayt seviyesinde hafızaya sahip çok az bilgisayar var.
- 12. seviyede bir çözüm düşünüldüğünde yaklaşık 35 yıl sürer. Ayrıca 10 perabyte hafıza gereklidir.
- Genel olarak denebilir ki üssel karmaşıklığa sahip problemler bilgisiz arama yöntemleri ile küçük durum uzayları dışında çözülemez.

Breadth First Search

● Avantajları

- Hedefe olan en kısa yolu bulur.

It finds the shortest path to the destination.

● Dezavantajları

- En yakındaki hedef düğümün derinliğine göre üssel olarak değişen boyutta bir arama ağacının üretilmesi ve saklanması gerektirir.

It requires the production and storage of a search tree of exponentially varying size according to the depth of the nearest target node.

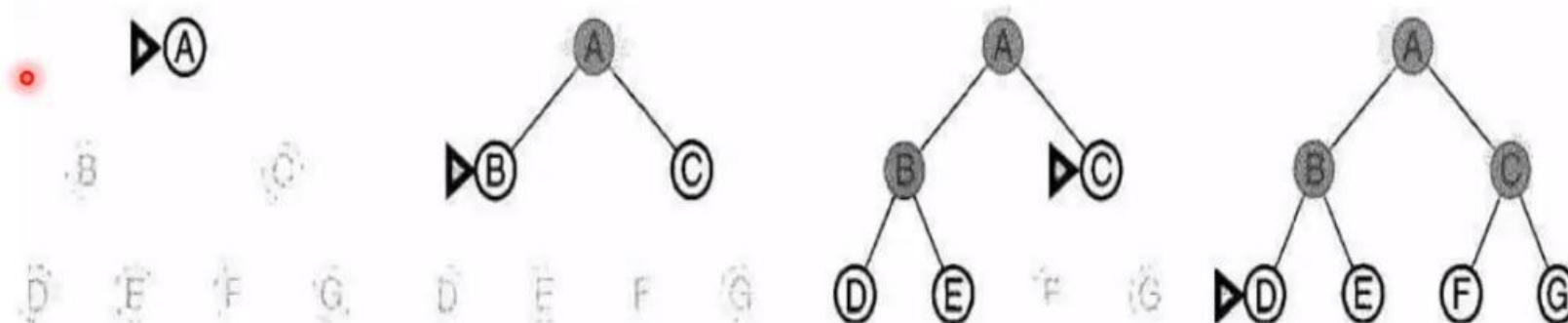
- Fringe'in saklanması için üsseş artan bir alana ve üssel olarak artan zamana ihtiyaç duyar.

For the storage of the Fringe, the base needs an increasing space and exponentially increasing time.

Breadth First Search

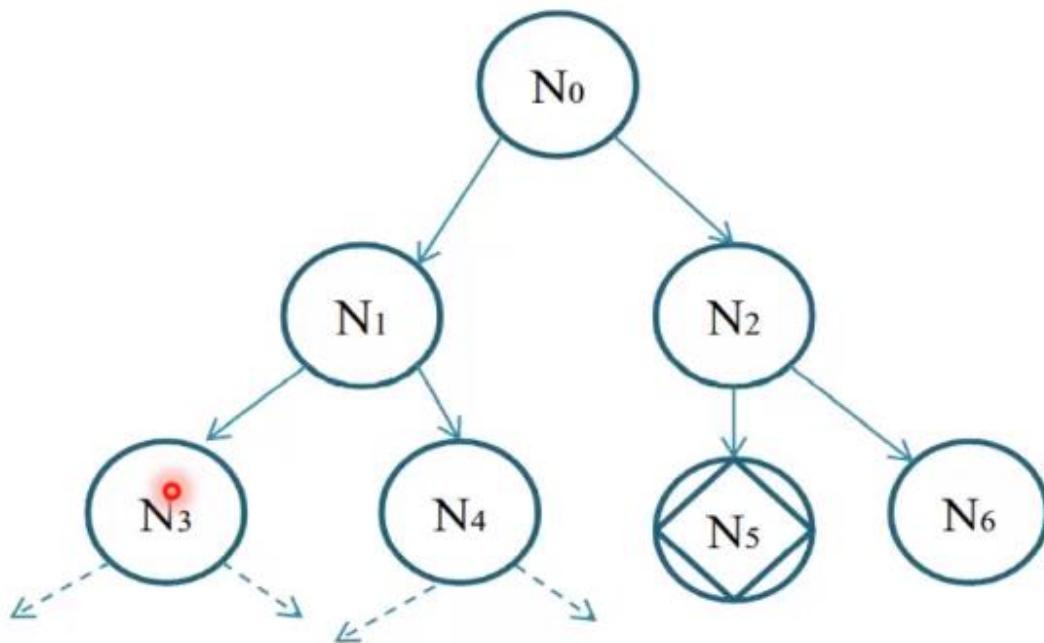
YAYILIM ÖNCELİKLİ ARAMA (BFS)

- Önce **kök düğüm** genişletilir. Genişletilen saçaktaki tüm düğümlere gidilir.
- **Yayılım öncelikli arama** ile gidilen düğüm en alt derinliğe kadar iner. Bu işlemler gerçekleştirilirken diğer saçaklarla **eşgündümlü** gidilir. Böylece bütün yollardaki düğümler derinlikleri eşit olarak açılır.



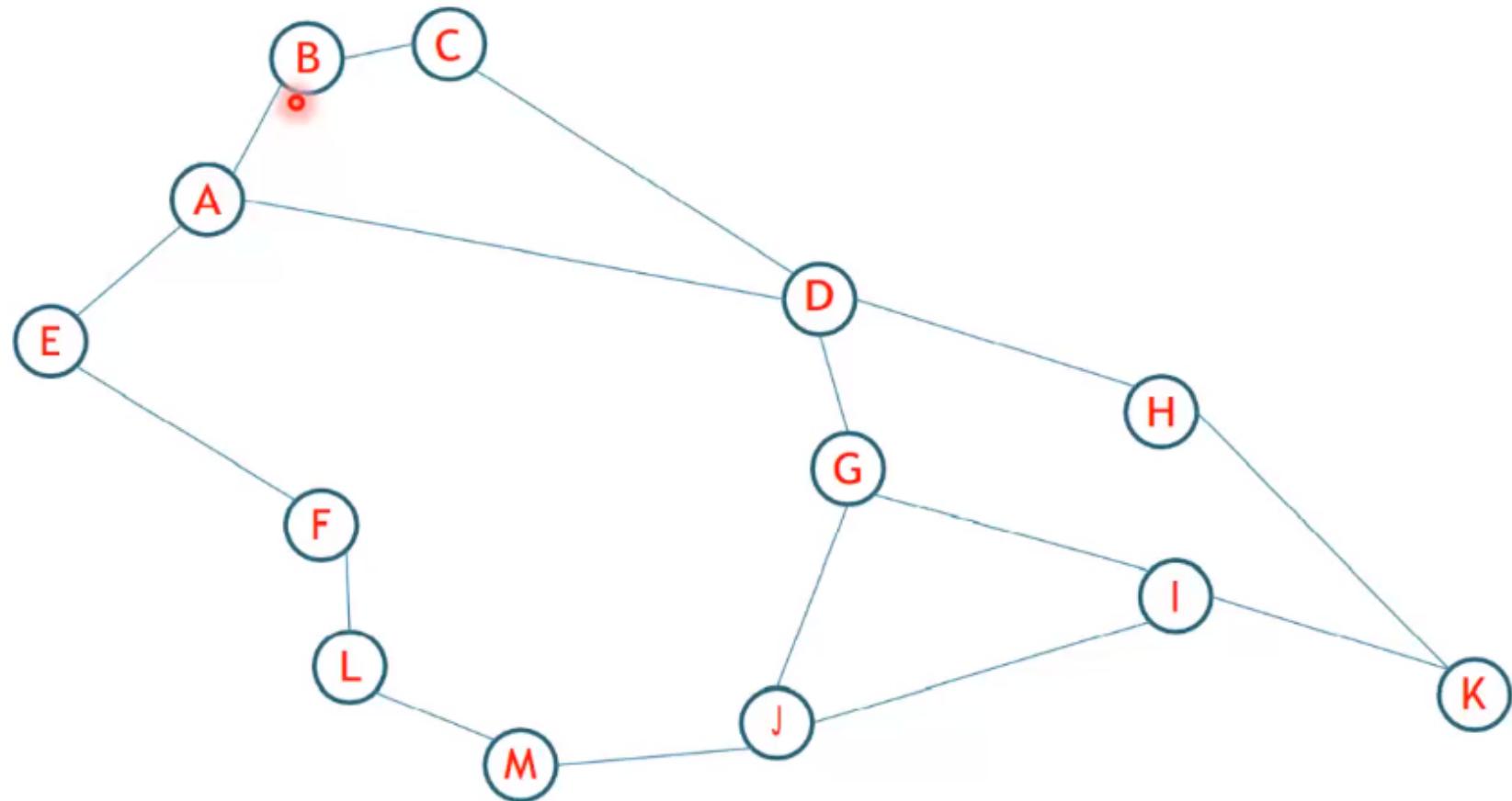
YAYILIM ÖNCELİKLİ ARAMA (BFS)

- N₀ durumundan N₅ durumuna gitmek için;
 - > Arama yolu: N₀-N₁-N₂-N₃-N₄-N₅
 - > Çözüm yolu: N₀-N₂-N₅

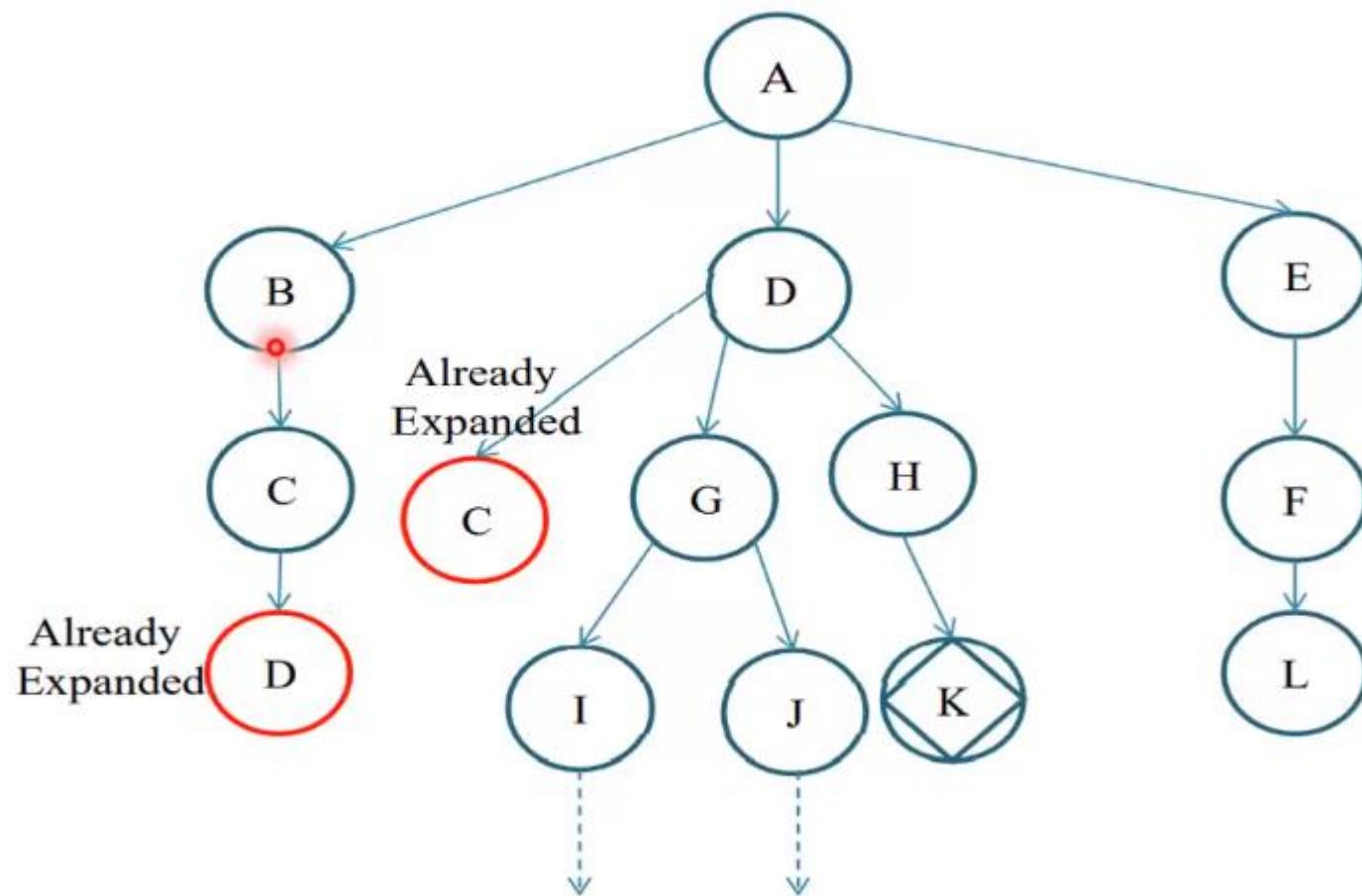


YAYILIM ÖNCELİKLİ ARAMA (BFS)

- A başlangıç durumundan K hedefine ulaşmak için;



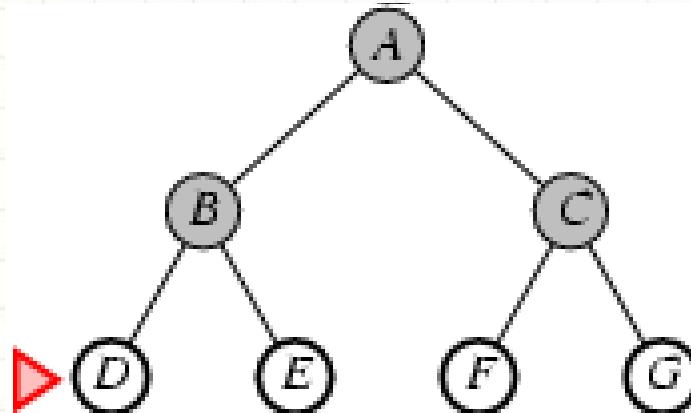
YAYILIM ÖNCELİKLİ ARAMA (BFS)



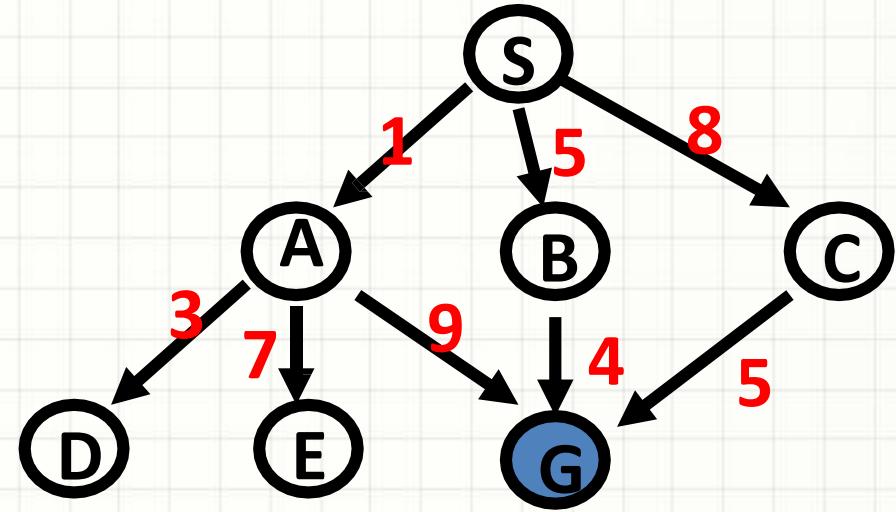
- Arama Yolu: {A, B,D,E,C,G,H,F,I,J,K}
- Çözüm Yolu: {A,D,H,K}

Genişlik(Yayılım) Öncelikli Arama

- Her bir seviyedeki düğümlerin tamamı arandıktan sonra, bir sonraki seviyenin düğümlerine ilerlenir
 - Aynı seviyede arama: soldan sağa
- En sıç (üst seviye) genişletilmemiş düğüm açılır



Örnek



Yol	OPEN list	CLOSED list
S	{ S }	{ }
S,A	{ A B C }	{ S }
S,B	{ B C D E G }	{ S A }
S,C	{ C D E G G' }	{ S A B }
S,A,D	{ D E G G' G" }	{ S A B C }
S,A,E	{ E G G' G" }	{ S A B C D }
S,A,G	{ G G' G" }	{ S A B C D E }
	{ G' G" }	{ S A B C D E }

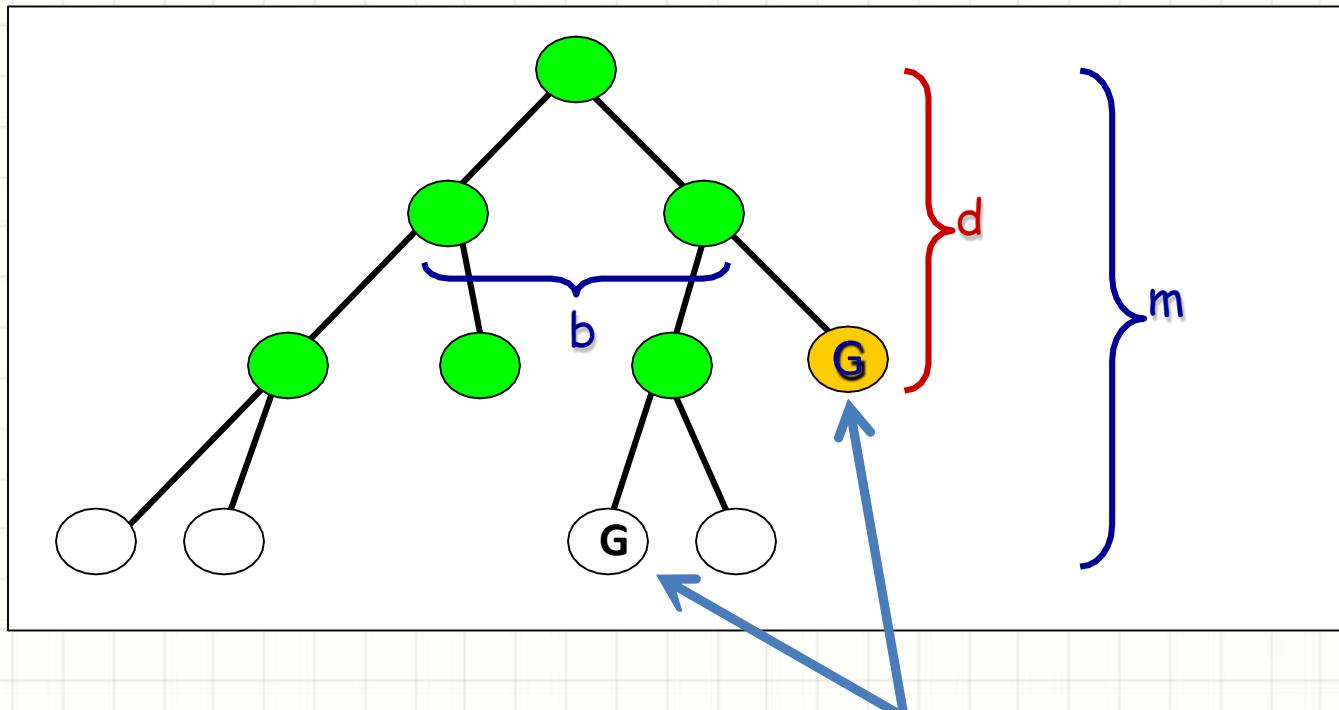
Bulunan çözüm yolu **S A G** --> maliyeti = **10**

Genişletilen düğüm sayısı (hedef düğüm dahil) = **7**

Zaman karmaşıklığı-Time Complexity

If the target node is located at depth d of the tree, all nodes up to that depth are created

- Eğer hedef düğüm ağacın d derinliğinde bulunursa bu derinliğe kadarki tüm düğümler oluşturulur.



- Böylece: $O(b^d)$

Optimal çözüm:

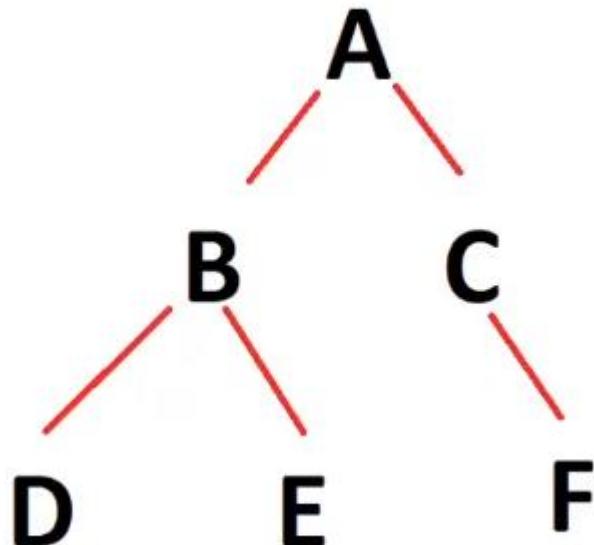
Köke en yakın (az maliyetli-adımlı)

Özellikleri

If each node has b number of sub-branches, the total number of branches that need to be examined:

- Her bir düğümün b sayıda alt dal varsa, incelenmesi gereken toplam dal sayısı
 - $M = 1 + b + b^2 + b^3 + \dots + b^d$
 - Bütünlük: b sonsuz değil ise evet – completeness: yes if b is not infinite
 - Zaman karmaşıklığı: $O(b^d)$
 - Bellek karmaşıklığı: $O(b^d)$
 - En iyi çözüm bulunabilir mi (optimality): Evet
 - Çünkü köke en yakın olanı bulur (her adım maliyeti = 1 varsayıarak)
- Çok fazla bellek ihtiyacı var
 - Tüm düğümleri bellekte tutmaya gerek var mı?
- Ağaçta çok fazla dallanma varsa ve çözüm köke uzak ise uygun değildir

Örnek- Yayılım Öncelikli--BFS

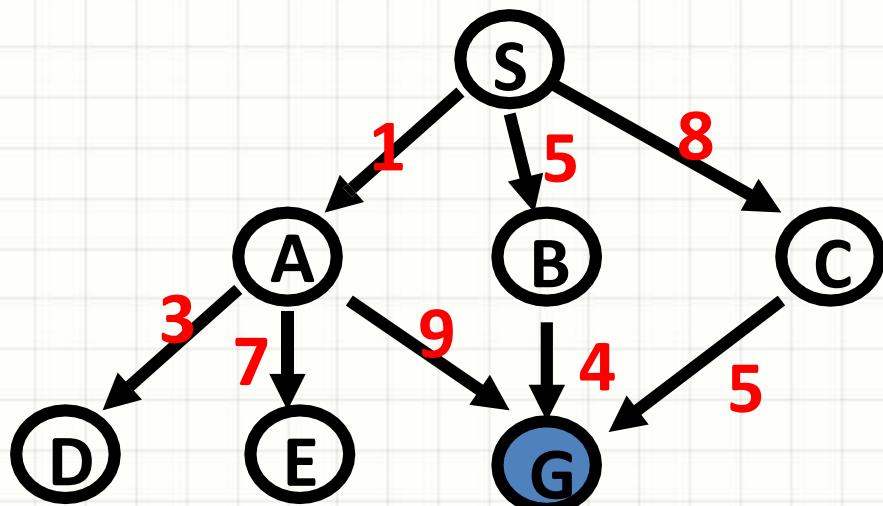


```
1
2
3
4
5     grafik={
6         'A':['B','C'],
7         'B':['D','E'],
8         'C':['F'],
9         'D':[],
10        'E':[],
11        'F':[],
12    }
13
14    ziyaret=[]
15    yigın=[]
16
17    def bfs(ziyaret,grafik,düğüm):
18        ziyaret.append(düğüm)
19        yigın.append(düğüm)
20
21        while yigın:
22            s=yigın.pop(0)
23            print(s,end=" ")
24
25            for komşu in grafik[s]:
26                if komşu not in ziyaret:
27                    yigın.append(komşu)
28
29
30
31    bfs(ziyaret,grafik,'A')
```

```
In [25]: runfile('C:\Users\HP\PycharmProjects\untitled3.py')
A B C D E F
In [26]:
```

Eşit Maliyetli (Uniform Cost) Arama

- Genişlik öncelikli aramaya benzer
- Kökten itibaren toplam maliyeti en az olan düğüm seçilir ve genişletilir
 - **Sıralama:** Maliyeti en azdan en çoga



Genişlik-Öncelikli çözüm:
S->A->G

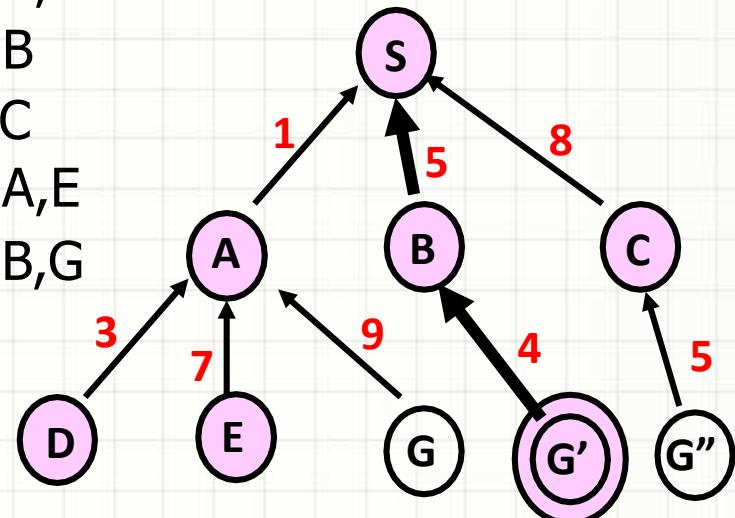
- Width is similar to Breadthsearch.
- From the root, the node with the least total cost is selected and expanded.
 - Sequencing: Cost from least to maximum.

Örnek

G.Düğüm D.listesi

	$\{S(0)\}$	Yol
S	$\{A(1) B(5) C(8)\}$	S
A	$\{D(4) B(5) C(8) E(8) G(10)\}$	S,A
D	$\{B(5) C(8) E(8) G(10)\}$	S,A,D
B	$\{C(8) E(8) G'(9) G(10)\}$	S,B
C	$\{E(8) G'(9) G(10) G''(13)\}$	S,C
E	$\{G'(9) G(10) G''(13)\}$	S,A,E
G'	$\{G(10) G''(13)\}$	S,B,G

Yol
S
S,A
S,A,D
S,B
S,C
S,A,E
S,B,G



Bulunan çözüm yol: **S B G** --> maliyeti = **9** (10 değil)
 Genişletilen düğüm sayısı (hedef düğüm dahil) = **7** (aynı)

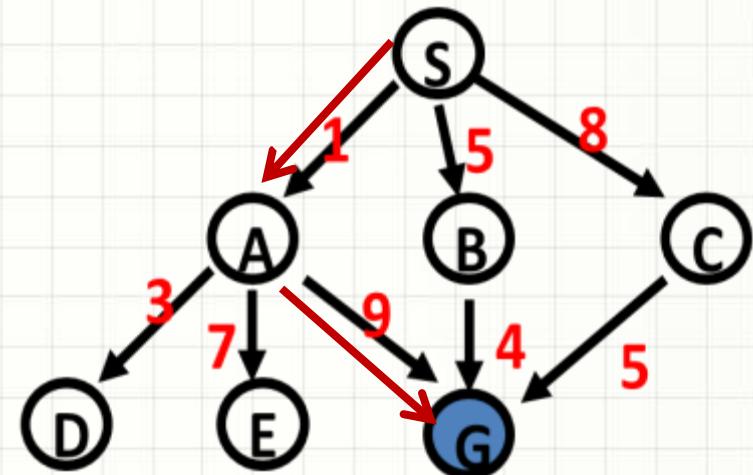
Önemli!

The search process would not be optimal if any node in the queue (in the ordered node list) was terminated when the target node was the target node.

- Arama süreci kuyrukta (sıralı düğüm listesinde) ***herhangi*** bir düğüm hedef düğüm olduğunda sonlandırılırsa optimal olmaz

G.Dgm. D.listesi

		Yol
	{S(0)}	-
S	{A(1) B(5) C(8)}	S
A	{D(4) B(5) C(8) E(8) G(10)}	S,A
D	{B(5) C(8) E(8) G(10)}	S,A,D
B	{C(8) E(8) G'(9) G(10)}	S,B
C	{E(8) G'(9) G(10) G''(13)}	S,C
E	{G'(9) G(10) G''(13)}	S,A,E
G'	{G(10) G''(13)}	S,B,G



Özellikleri

- Completeness: Her adım maliyeti $\geq \varepsilon > 0$ ise evet
- Time complexity: $O(b^{C^*/\varepsilon})$
 - C^* : optimal çözümün maliyeti
 - ε : en düşük adım maliyeti
 - Her adımın maliyeti aynı ise $O(b^d)$, **Neden?**

Genişlik-öncelikliye dönüşür

- Space complexity: $O(b^{C^*/\varepsilon})$
- Optimality: Her adım maliyeti $\geq \varepsilon > 0$ ise evet

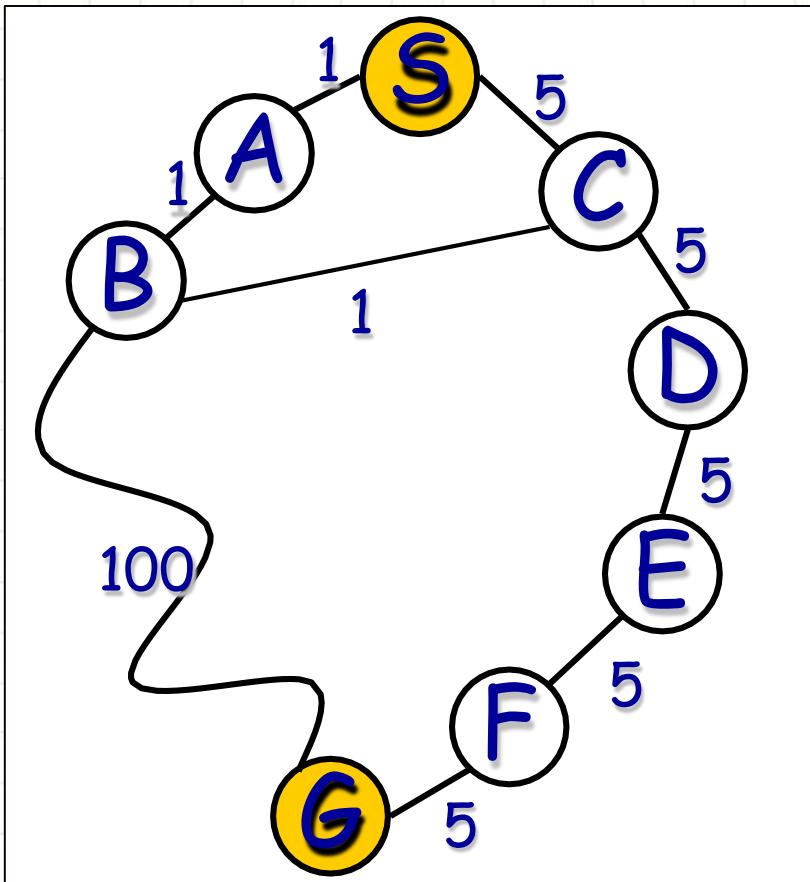
b: dallanma faktörü

d: en düşük maliyetli çözümün derinliği

İyileştirme

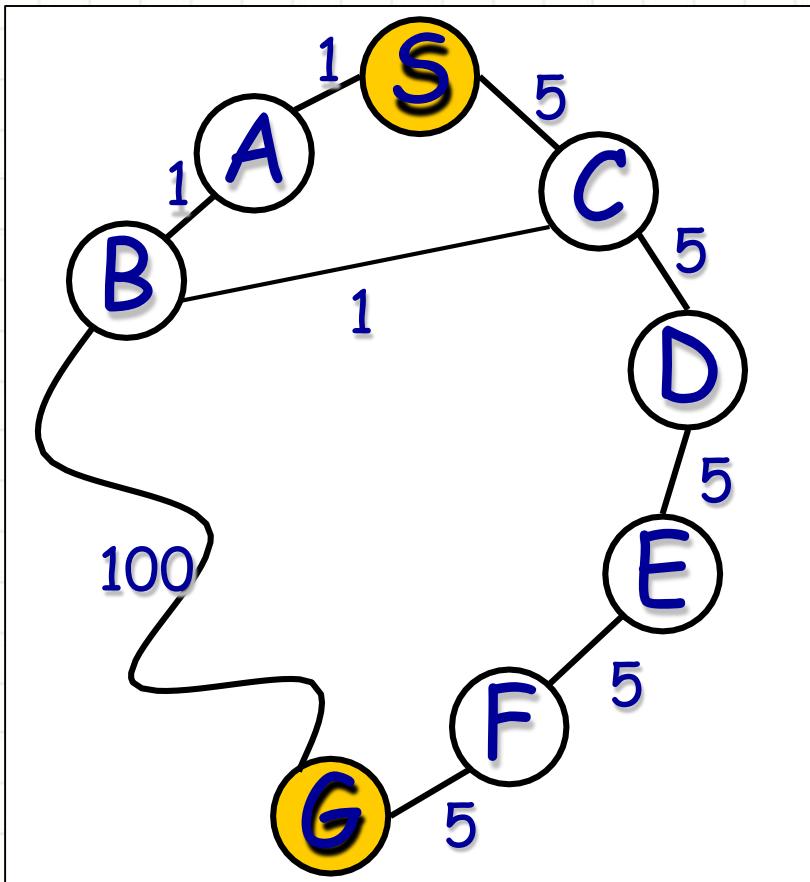
- Kuyruktaki (Düğüm listesi) düğümlerden birine başka bir yoldan tekrar ulaşılırsa
 - Önceki maliyetle karşılaştır
 - Maliyeti küçük olanı tut
 - Yolu ve yeni maliyeti güncelle
- If one of the nodes in the queue (Node list) is reached again in another way:
 - Compare with the previous cost.
 - Keep the one that costs less.
 - Update the path and the new cost.

Örnek



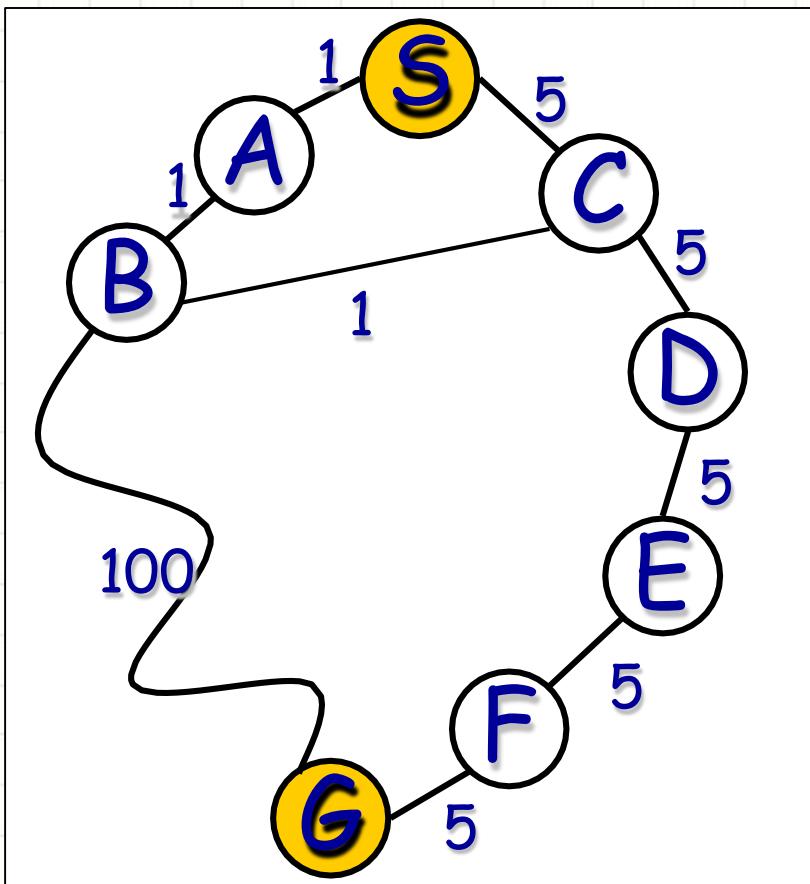
#	Durum	Maliyet	Ata
1	S	0	-

Örnek



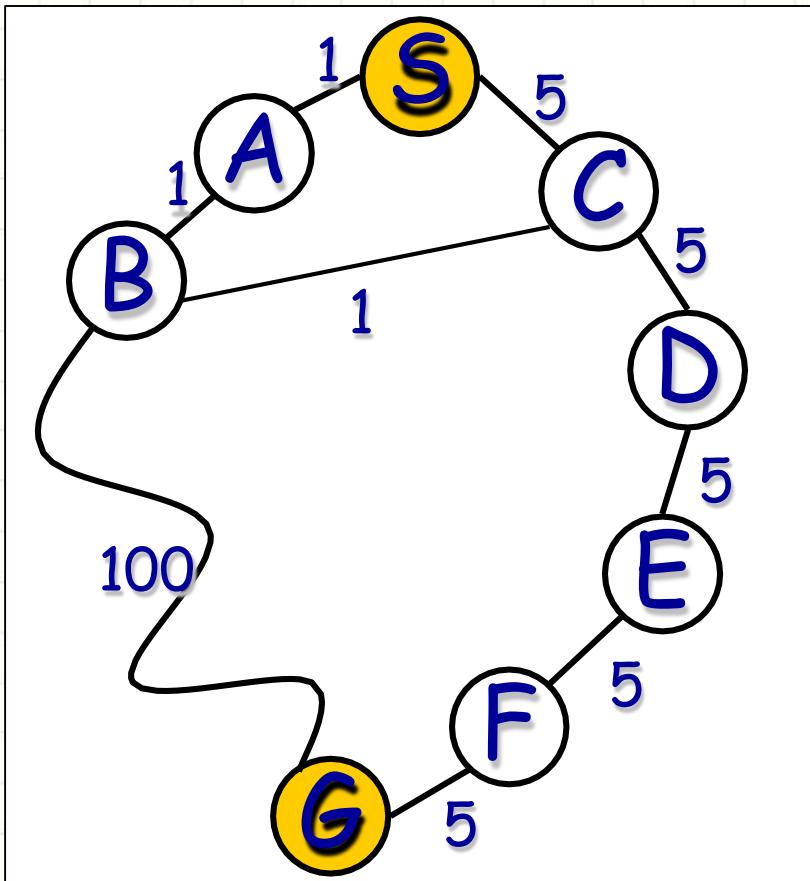
#	Durum	Maliyet	Ata
1	S	0	-
2	A	1	S
3	C	5	S

Örnek



#	Durum	Maliyet	Ata
1	S	0	-
2	A	1	S
4	B	2	A
3	C	5	B

Örnek



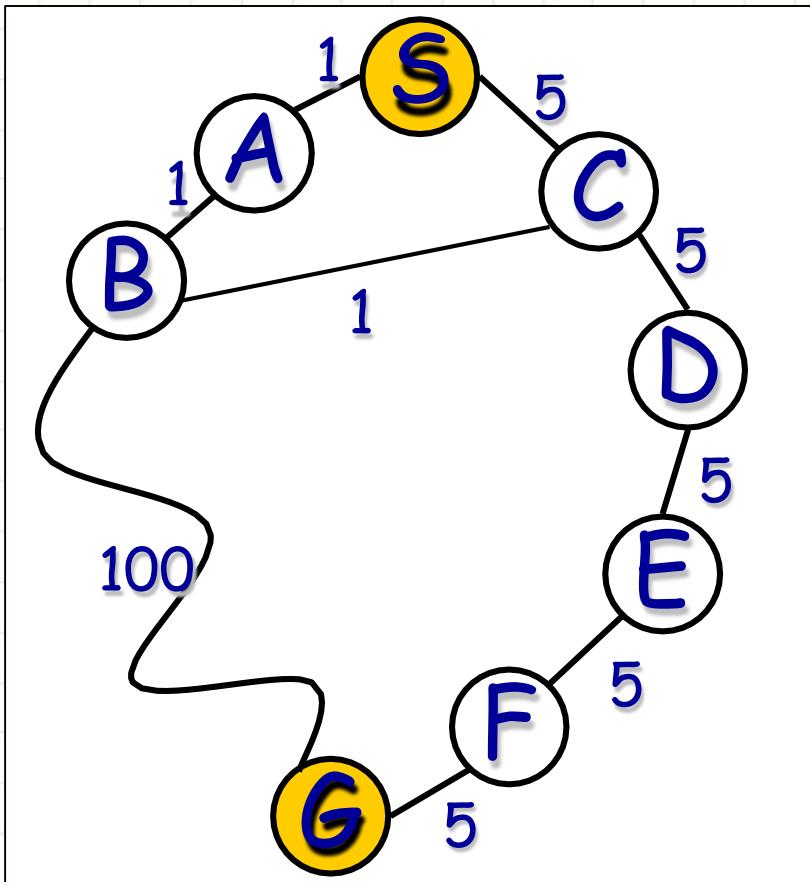
#	Durum	Maliyet	Ata
1	S	0	-
2	A	1	S
4	B	2	A
5	C	3	B
6	G	102	B

C'nin maliyeti ve atası güncellenir:

Ata: S -> B

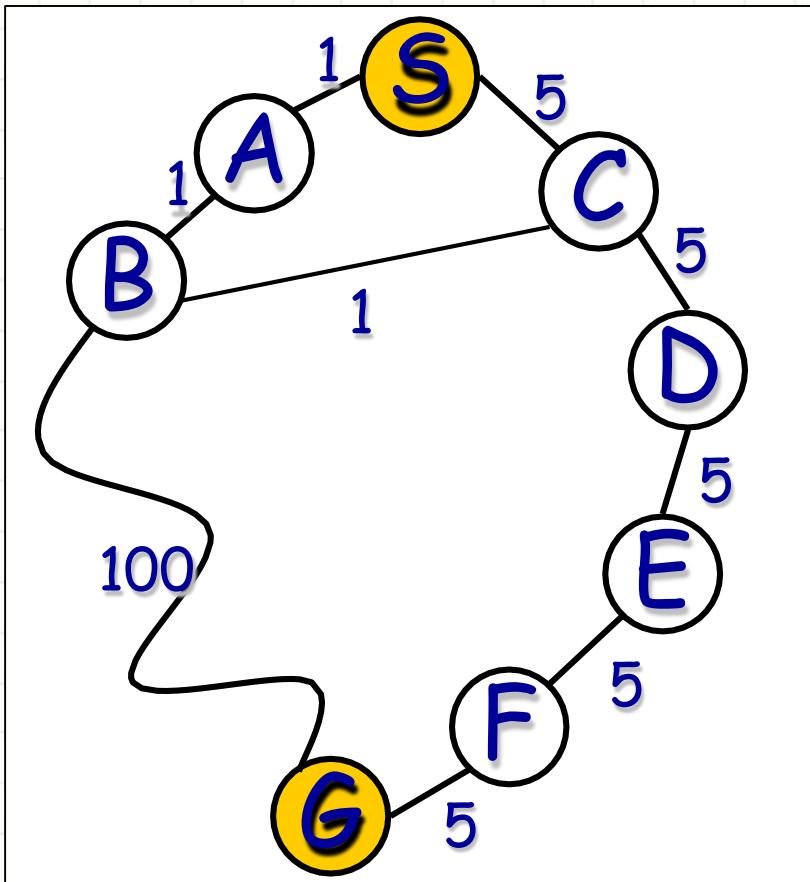
Maliyet: 5 -> 3

Örnek



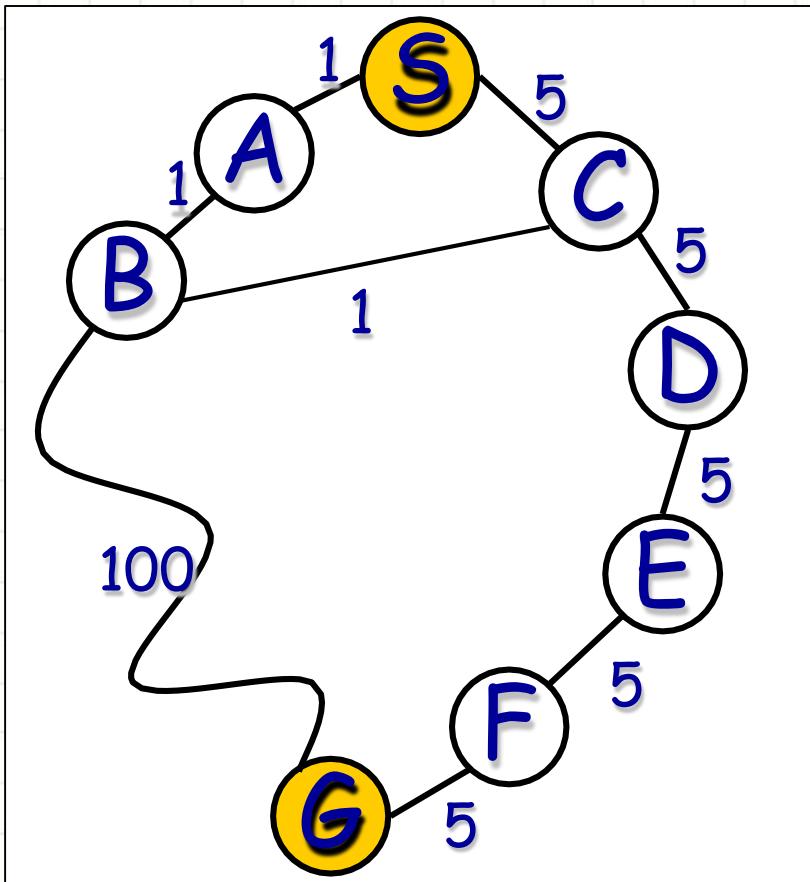
#	Durum	Maliyet	Ata
1	S	0	-
2	A	1	S
4	B	2	A
5	C	3	B
7	D	8	C
6	G	102	D

Örnek



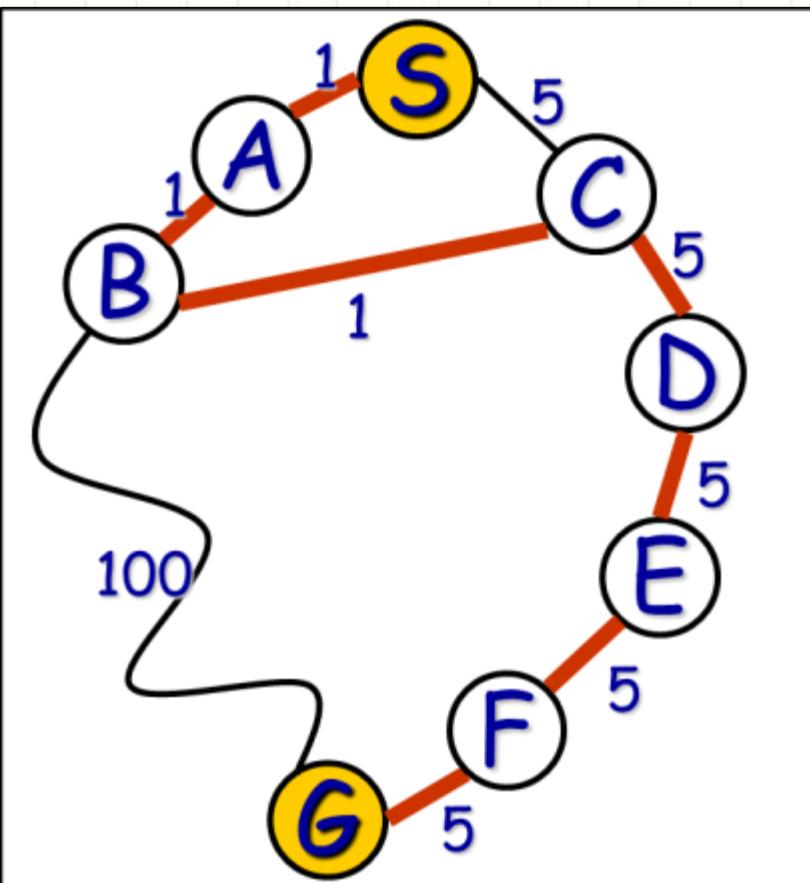
#	Durum	Maliyet	Ata
1	S	0	-
2	A	1	S
4	B	2	A
5	C	3	B
7	D	8	C
8	E	13	D
6	G	102	B

Örnek



#	Durum	Maliyet	Ata
1	S	0	-
2	S A	1	S
4	S A B	2	A
5	S A B C	3	B
7	S A B C D	8	C
8	S A B C D E	13	D
9	S A B C D E F	18	E
6	S A B C D E F G	102	B

Örnek



#	Durum	Maliyet	Ata
1	S	0	-
2	A	1	S
4	B	2	A
5	C	3	B
7	D	8	C
8	E	13	D
9	F	18	E
10	G	23	F
6	G	102	G

Hedefe ulaşıldı

DERİN ÖNCELİKLİ ARAMA (DFS)

- Düğümde **en derine** inerek ilerler. Eğer hedefini o düğümde bulamazsa diğer düğümlerde de en derine inerek arama yapar.
- **Derin Öncelikli Arama;**
 - > Son giren ilk çıkar (LIFO) olarak çalışır.
 - > Riskli aramadır.
 - > Döngüye girerek sonsuzluğa gidebilir.
 - > Her zaman sonuca ulaşamaz.

It progresses by going deepest in the knot. If it cannot find its target in that node, it searches the other nodes by going deeper.

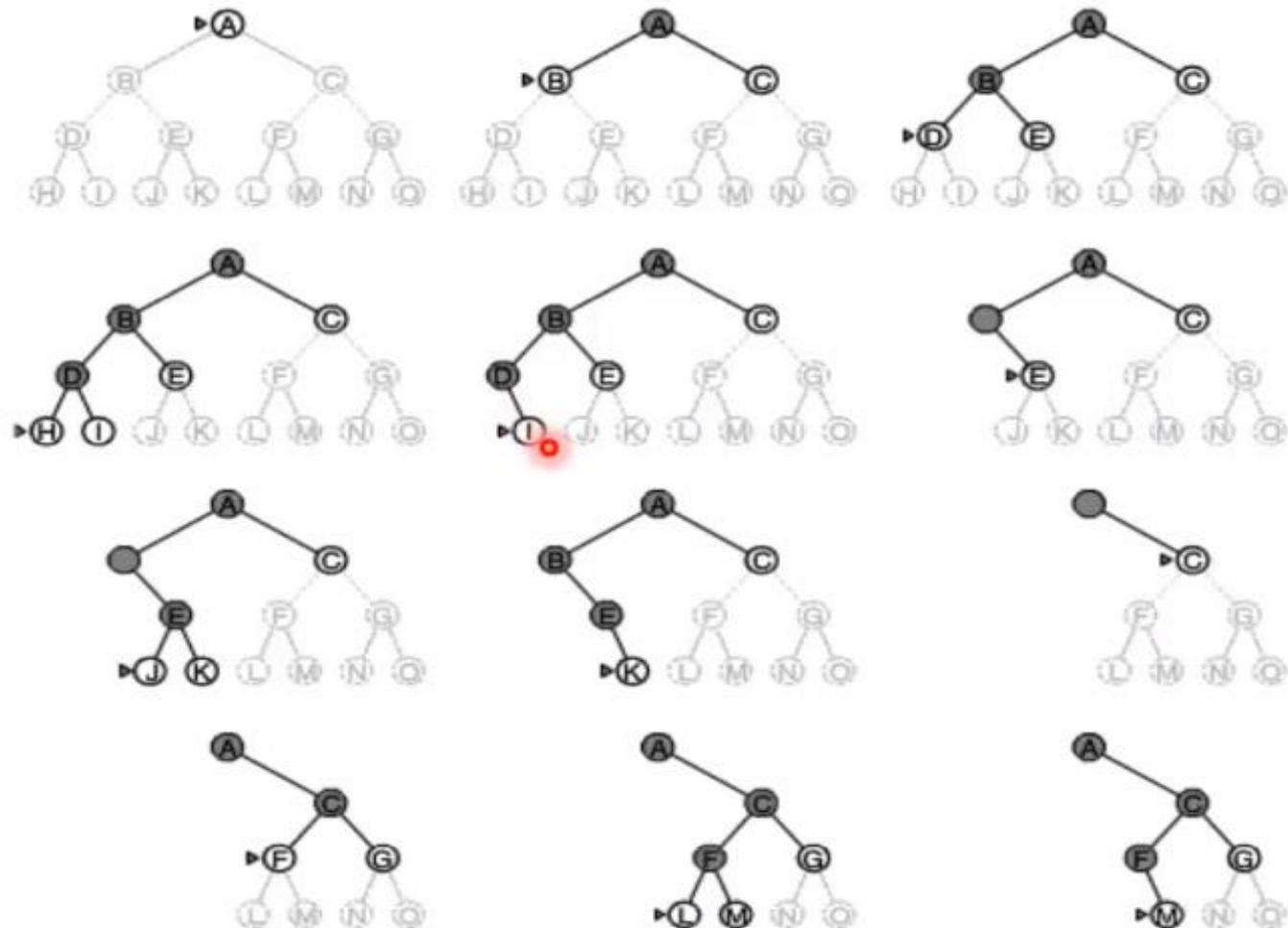
Deep first Search;
--> Operates as a last-in, first-out (LIFO).
... > It is a risky search.
--> It can go to infinity by looping.
> It doesn't always get results.

Depth First Search

Let Fringe be the list that holds the initial state

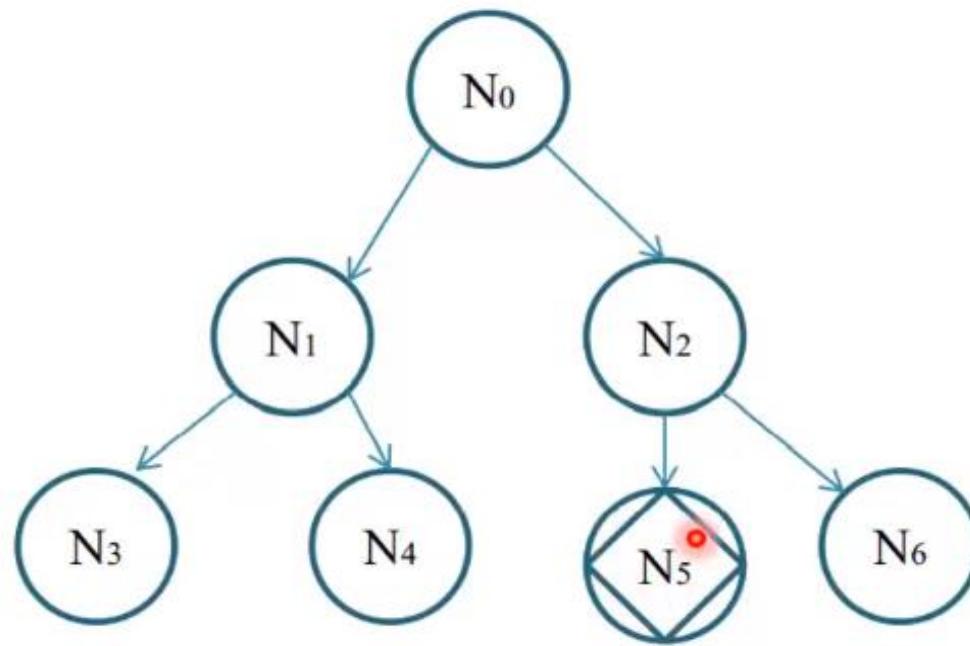
- Fringe, başlangıç durumunu tutan liste olsun
- Döngü Loop
 - Eğer fringe boş ise return hata
 - Node $\leftarrow \text{remove_first}(\text{fringe})$
 - Eğer Node hedef düğüm ise
 - Node düğümüne kadar olan yolu döndür Rotate the path up to the node node
 - Değilse Node düğümünün tüm successor 'larını oluştur ve **En derindeki düğümü önce aç** Untie the deepest knot first **yeni oluşturulan düğümleri Fringe'in başına ekle**
- Döngüyü Bitir

DERİN ÖNCELİKLİ ARAMA (DFS)

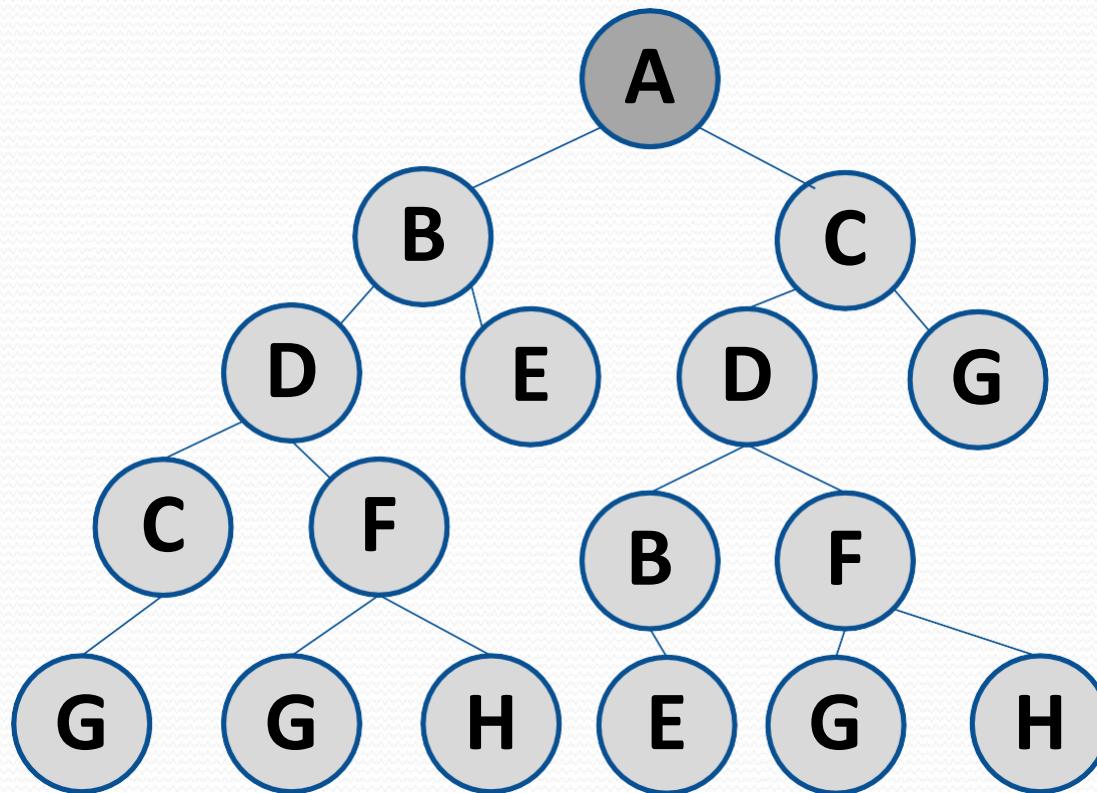


DERİN ÖNCELİKLİ ARAMA (DFS)

- N₀ başlangıç durumundan N₅ hedefine ulaşmak için;
- Arama Yolu: N₀-N₁-N₃-N₄-N₂-N₅
- Çözüm Yolu: N₀-N₂-N₅

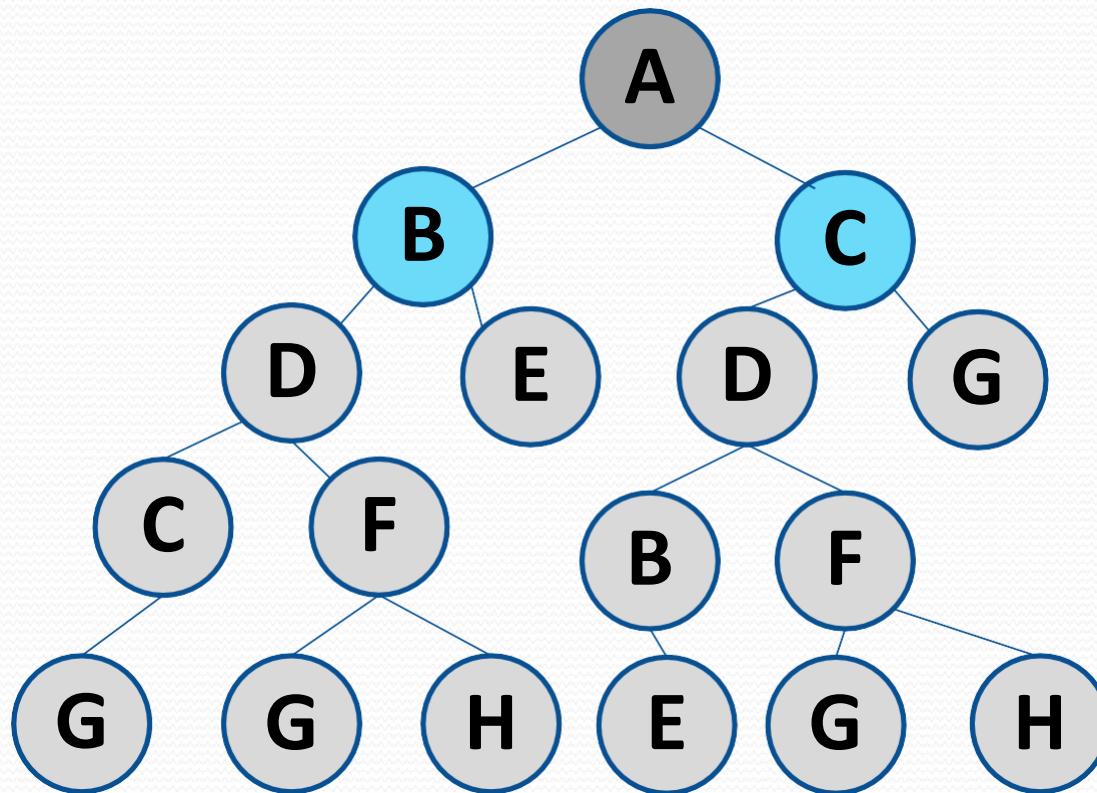


Depth First Search



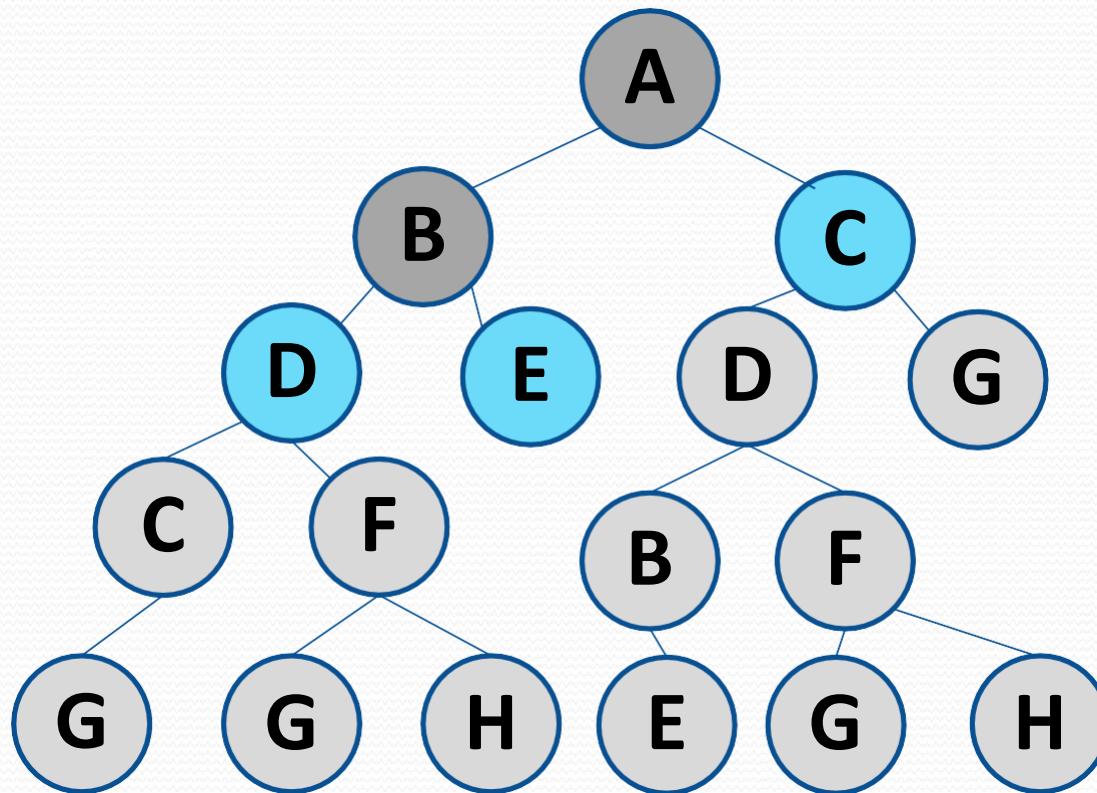
FRINGE : A

Depth First Search



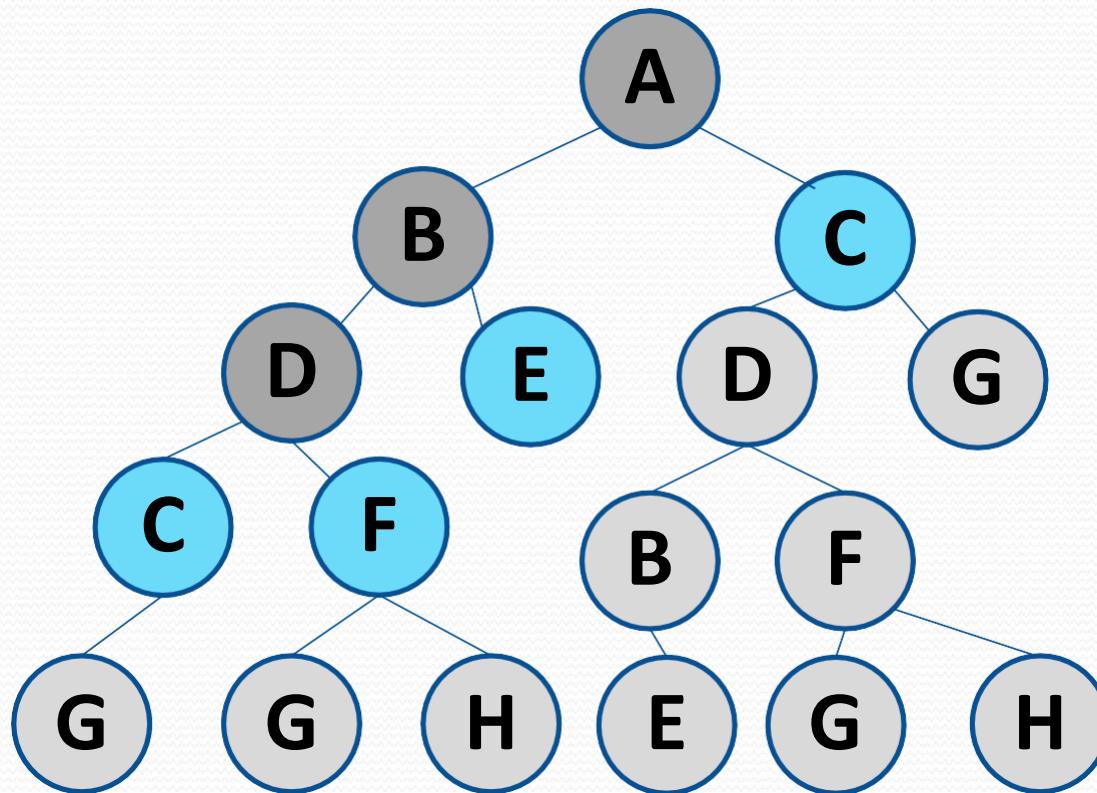
FRINGE : B C

Depth First Search



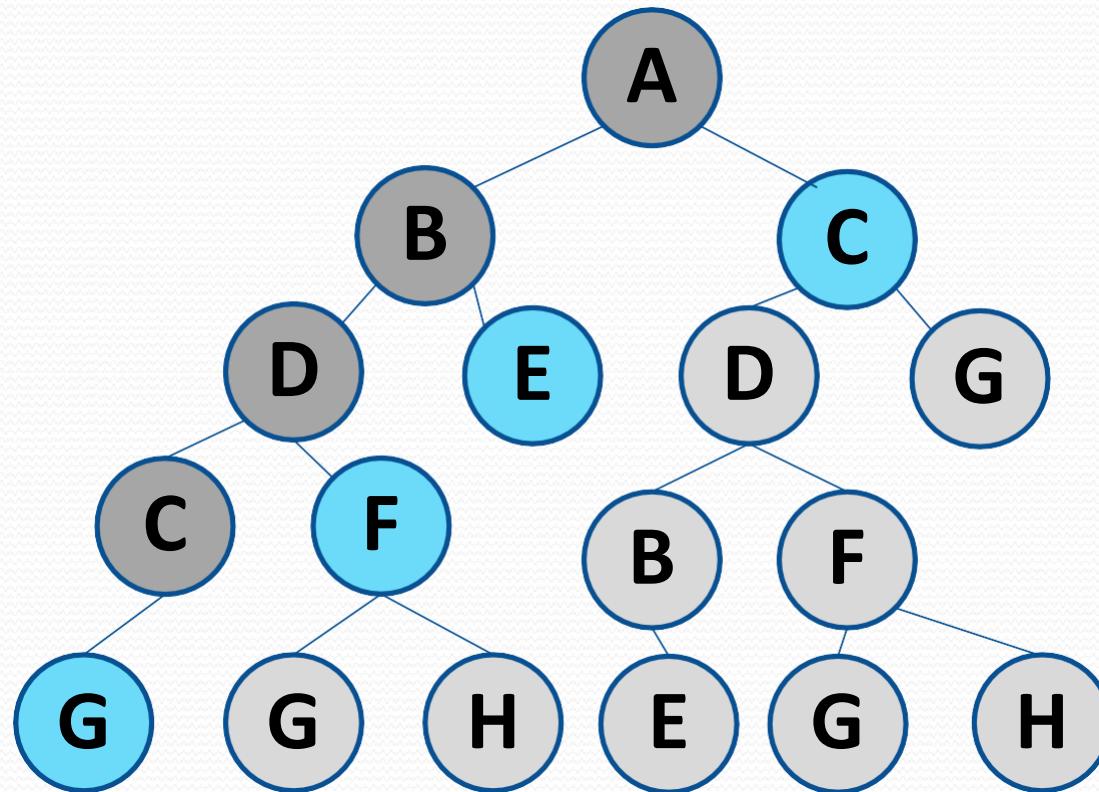
FRINGE : D E C

Depth First Search



FRINGE : C F E C

Depth First Search



HEDEF

FRINGE : G F E C

Depth First Search

- Stack yapısı kullanır. Fringe'teki düğümler LIFO mantığıyla listeden çıkarılır. Son giren ilk çıkar.
- Üssel artan zaman $O(b^d)$ ama sadece doğrual uzay $O(bd)$
- Derinlik sınırlaması olmadan sonlanmayabilir (Depth-limited-search)
- Complete değil.

Depth First Search

- It uses a stack structure. Nodes in the fringe are delisted using LIFO logic. Last in, first out.
- Exponentially increasing time $O(b^d)$ but only linear space $O(bd)$
- May not terminate without depth limitation (Depth-limited-search)
- It's not complete.

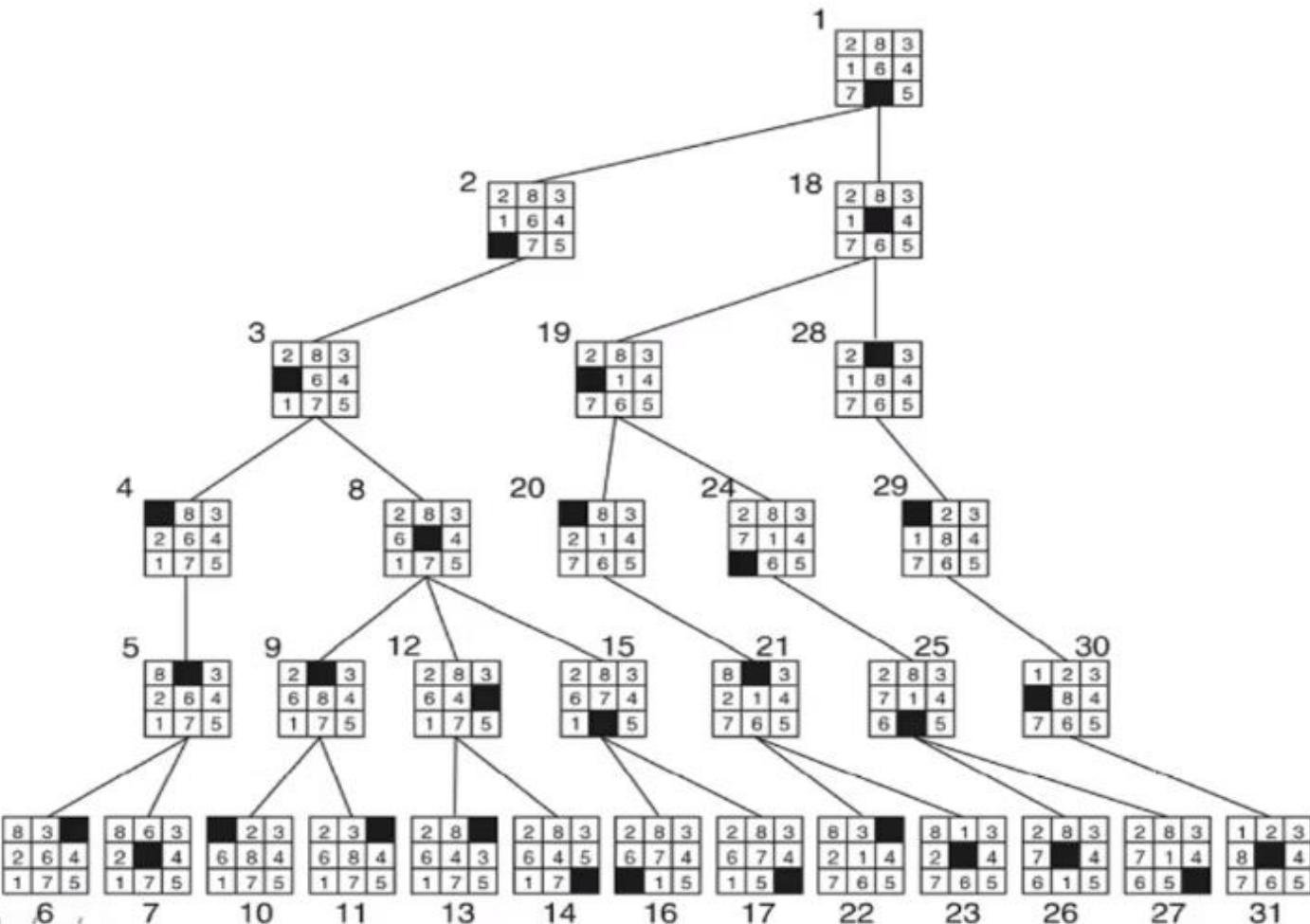
Arama Formülasyonu

- Bir durum uzayı grafır (V, E)
 - V , düğümler kümesi
 - E , kenar kümesi

-Graphs a state space (V, E)
--- V , set of nodes
--- E , edge set
- Her kenar sabit ve pozitif maliyete sahip
Each edge has a fixed and positive cost
- Her düğüm bir veri yapısı
 - Durum tanımlaması
 - Düğümün ebeveyni
 - Düğümü derinliği
 - Bu düğümü oluşturan operatör
 - Yolun maliyeti (operatör maliyetleri toplamı)

Each node is a data structure
-Status definition
-The parent of the node
-Node depth
-The operator that created this node
-Cost of the road (sum of operator costs)

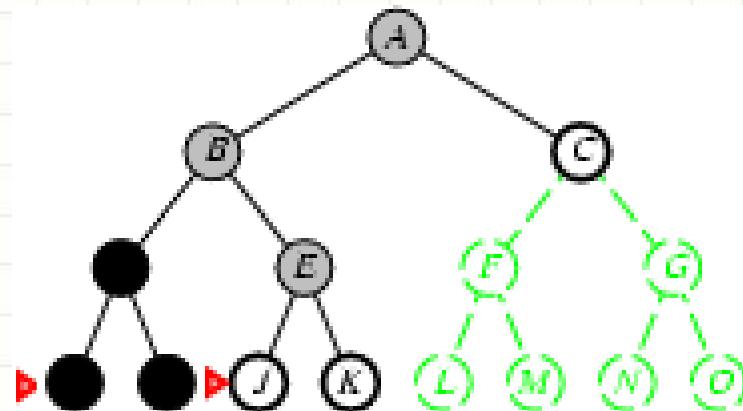
ÖRNEK:8 PUZZLE



Döngüye girer ve aynı sonuçları tekrar tekrar döndürür.

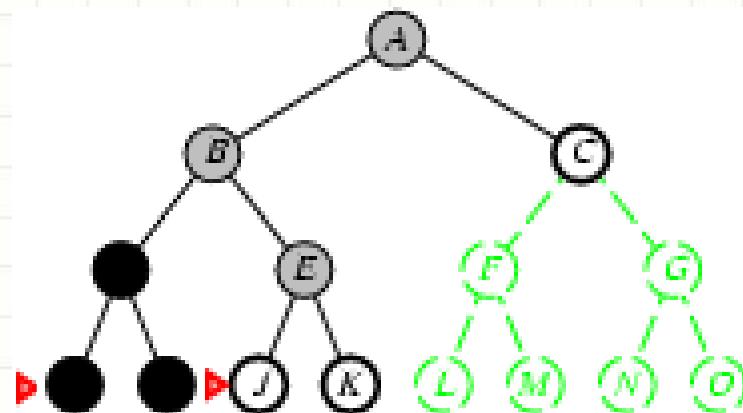
Derinlik Öncelikli Arama-Depth First Search

- Bir düğümün öncelikle tüm alt düğümlerine bakıldıktan sonra aynı seviyedeki diğer düğümlere geçilir



Derinlik Öncelikli Arama-Depth First Search

- After looking at all the child nodes of a node, other nodes at the same level are passed.



Örnek

G.Düğüm

S

A

D

E

G

D. Listesi

{ S }

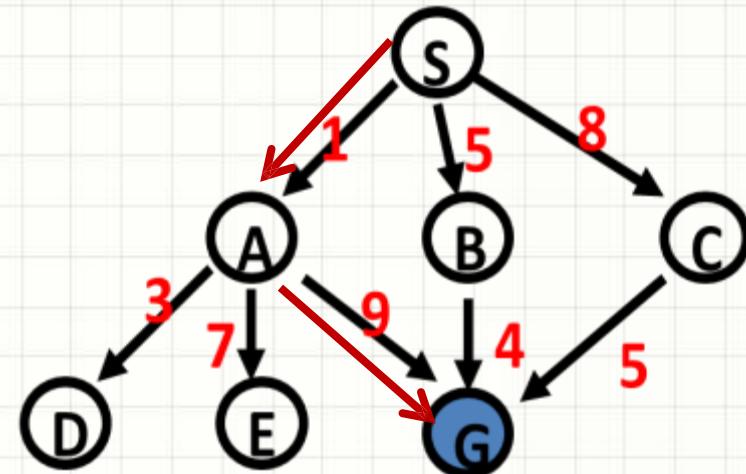
{ A B C }

{ D E G B C }

{ E G B C }

{ G B C }

{ B C }



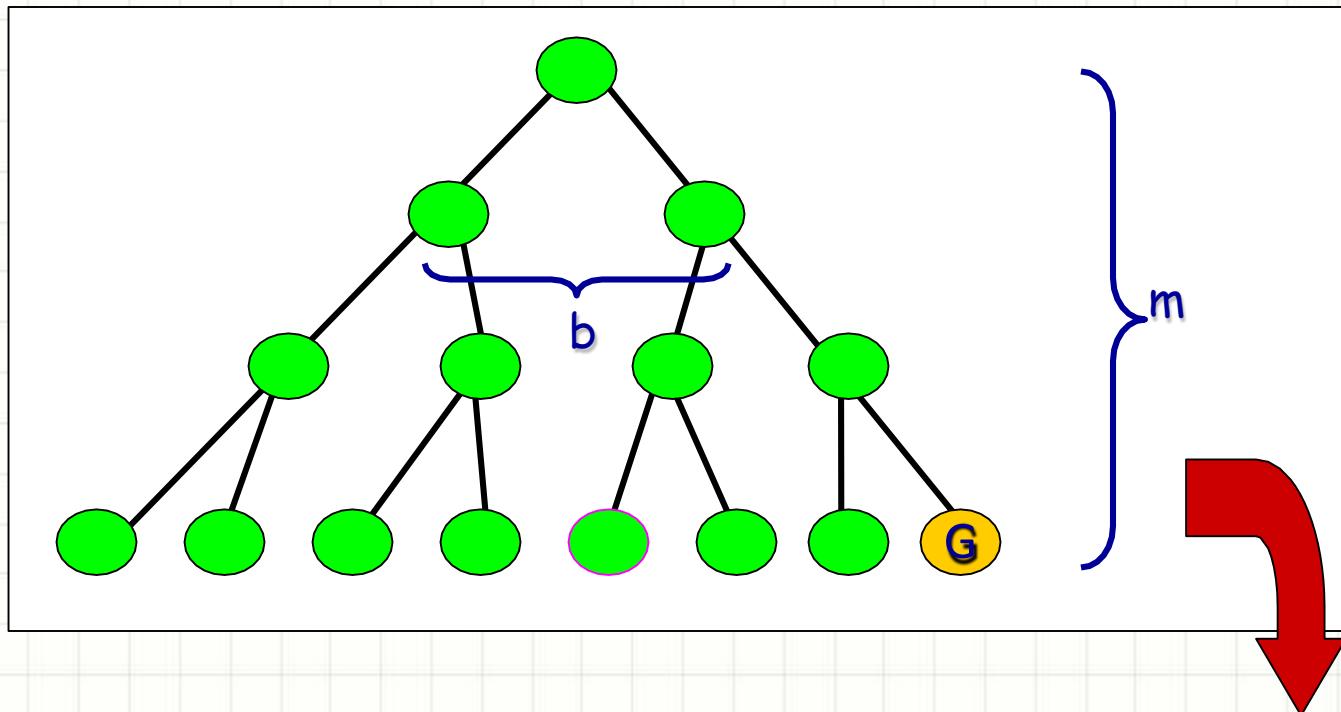
Düğüm kontrol sırası: D, E, G (3. kontrolde hedef bulunur)

Bulunan çözüm yolu: **S A G** --> maliyeti = **10**

Genişletilen düğüm sayısı (hedef düğüm dahil) = **5** (daha iyi)

Zaman karmaşıklığı- Time Complexity

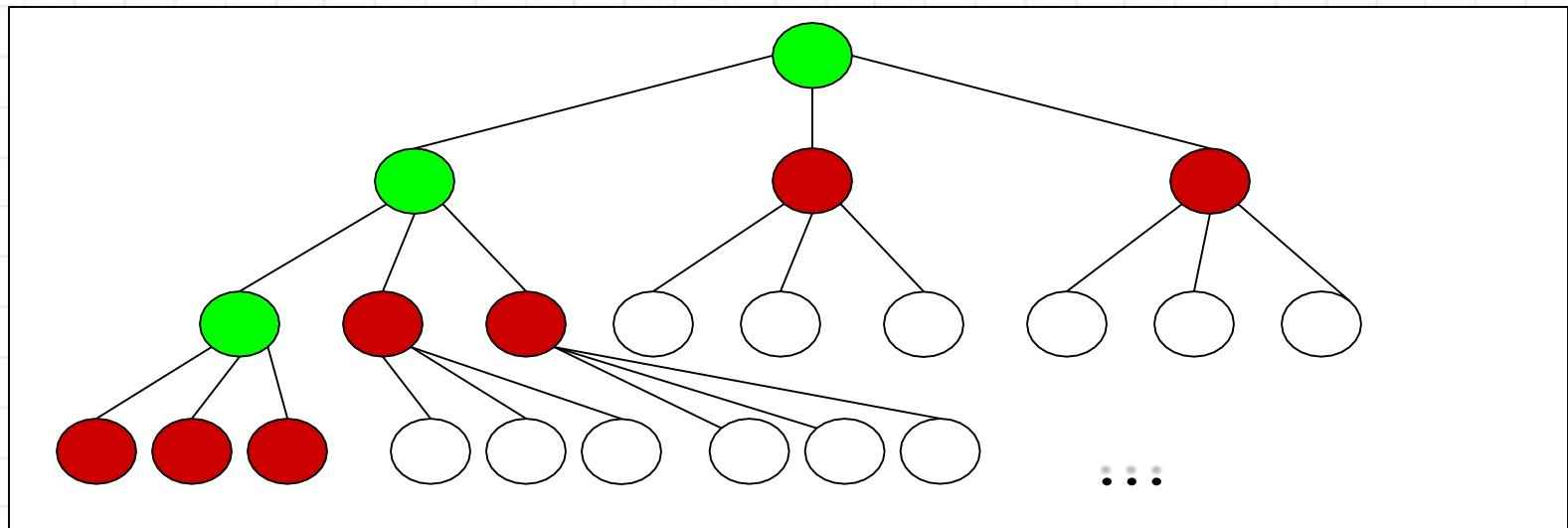
- En kötü durumda:
 - hedef (sadece) dalın en sağında olabilir,



- Zaman karmaşıklığı = $b^m + b^{m-1} + \dots + 1 = \frac{b^{m+1}-1}{b-1}$
- Böylece: $O(b^m)$

Bellek karmaşıklığı-Space Complexity

- Kuyrukta (Düğ. Listesi) en fazla düğüme altta en solda ulaşılır.
- Örnek: $m = 3$, $b = 3$:



- Kuyruk, tüm  düğümleri içerir --> 7
- Genelde: $((b-1) * m) + 1$
- Karmaşıklık: $O(m*b)$

Özellikleri

- Completeness: Hayır It cannot reach the result in trees of infinite depth or repeated state (loop).
 - Sonsuz derinlikteki ya da tekrarlanan durumlu (döngülü) ağaçlarda sonuca ulaşamaz
- Time complexity: $O(b^m)$
- Space complexity: $O(b m)$
 - Genişlik-öncelikliye göre daha az – Less than bfs
- Optimality: Hayır
 - Kökten mümkün olduğunca uzaklaştığı içinSince it moves as far away from the root as possible.

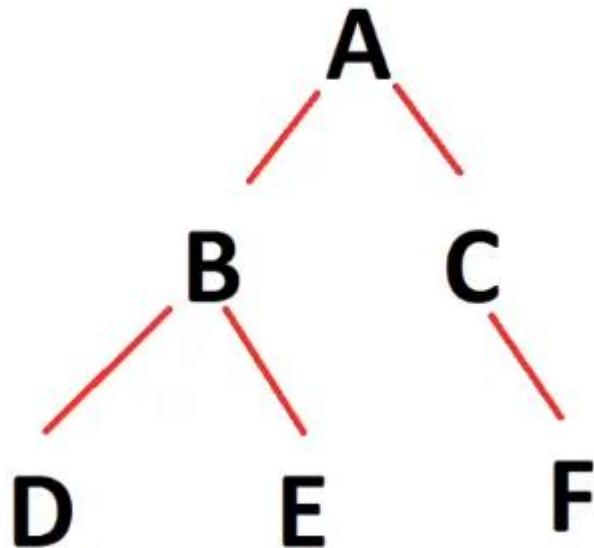
b: dallanma faktörü

: branching factor

m: arama ağacının max derinliği

Max depth of the search tree

Örnek- Derin Öncelikli--DFS



```
untitled7.py*
1
2
3
4
5  - grafik={
6      'A':["B","C"],
7      'B':["D","E"],
8      'C':["F"],
9      'D':[],
10     'E':[],
11     'F':[],}
12
13 ziyaret=set()
14
15 - def dfs(ziyaret,grafik,dügüm):
16 -     if dügüm not in ziyaret:
17         print(dügüm)
18         ziyaret.add(dügüm)
19         for komşu in grafik[dügüm]:
20             dfs(ziyaret,grafik,komşu)
21
22 dfs(ziyaret,grafik,'A')
```

```
In [29]:  
A  
B  
D  
E  
C  
F ]
```

```
In [30]:
```

Derinlik Sınırlı Arama- Depth-limited Searching

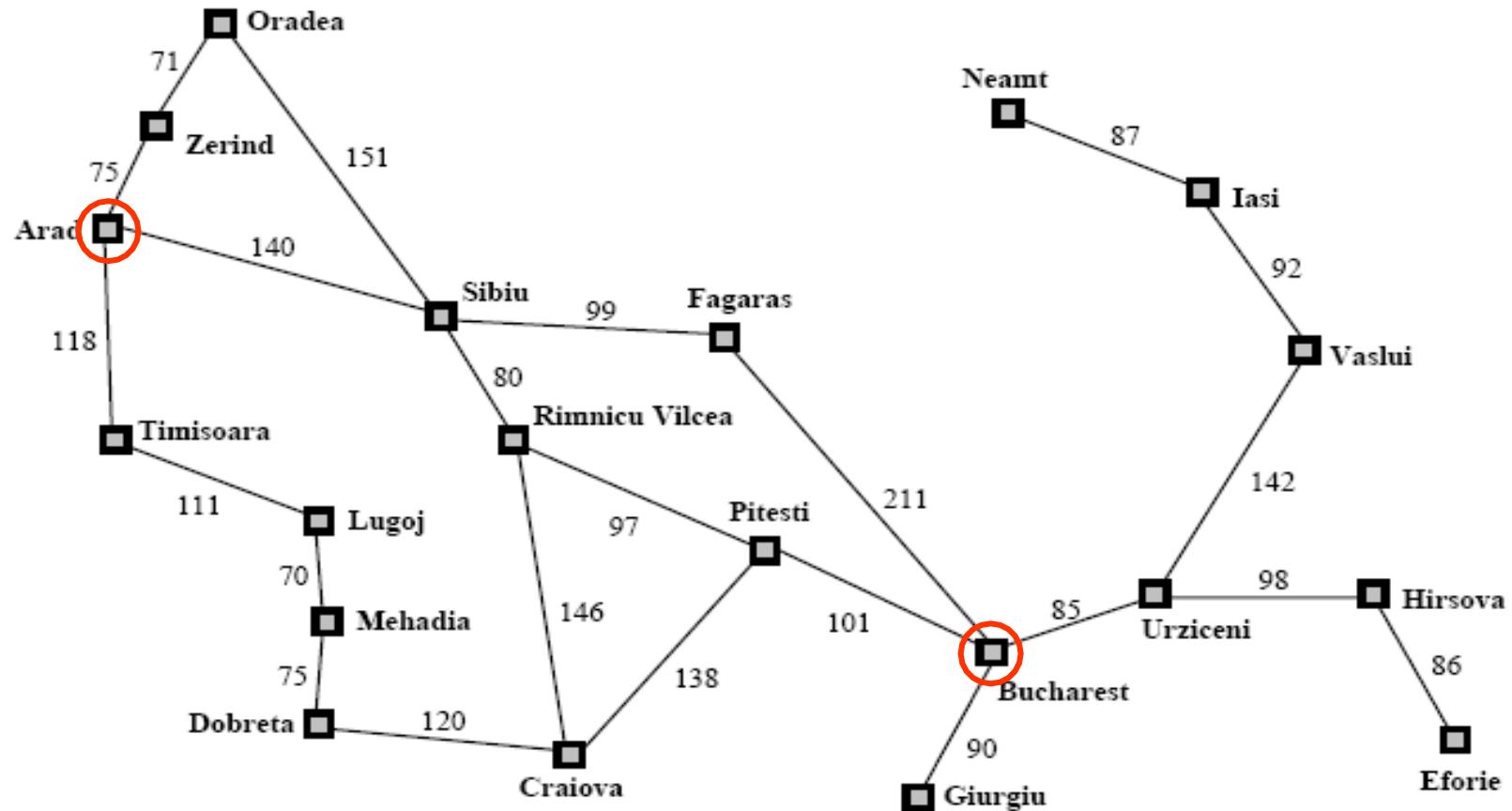
It is a type of depth-first search in which a maximum depth of L is allowed ...

- En fazla ℓ derinliğe izin verilen derinlik öncelikli arama çeşididir
 - Completeness: Yes if the solution is $\leq \ell$ deep.
 - If the required solution is at a depth of $\ell+1$, then it cannot be found.
- Completeness: Çözümü $\leq \ell$ derinlikte ise evet
 - Eğer gereken çözüm $\ell+1$ derinlikte ise, o zaman bulunamaz
- Time complexity: $O(b^\ell)$
- Space complexity: $O(b \ell)$
- Optimality: Hayır
- Derinlik sınırı (ℓ) nası seçilmeli?
 - **Durum sayısı**: çözüm yolu uzunluğu en fazla durum sayısı kadar
- Arama uzayı büyük ve çözüm derinliği belli olmayan durumlarda **Yinelemeli Derinleşen Arama** tercih edilir
 - Başarıya ulaşana dek derinlik sınırı her defa 1 artırılıyor

How to choose the depth limit (ℓ)?

- Number of states: The length of the resolution path is the maximum number of states.
- In cases where the search space is large and the solution depth is uncertain, iterative Deepening Search is preferred.
 - The depth limit is increased by 1 each time until success is achieved.

Örnek: Romanya'ya ulaşmak



Haritada 20 şehir var. Maximum derinlik sınırı $|I| = 19$ seçilebilir

Depth First Iterative Deepening Search

● Avantajları

- Depth first ve breadth first aramanın olumlu yönlerini birleştirir.
- Depth first aramanın doğrusal büyüyen hafıza ihtiyacı
- En az derinliğe sahip hedef düğümün bulunması garantilenir.

● Yöntem

- Her birinin derinlik sınırlaması 1 artırılan Depth first aramalar birleştirilir. .

Depth First Iterative Deepening Search

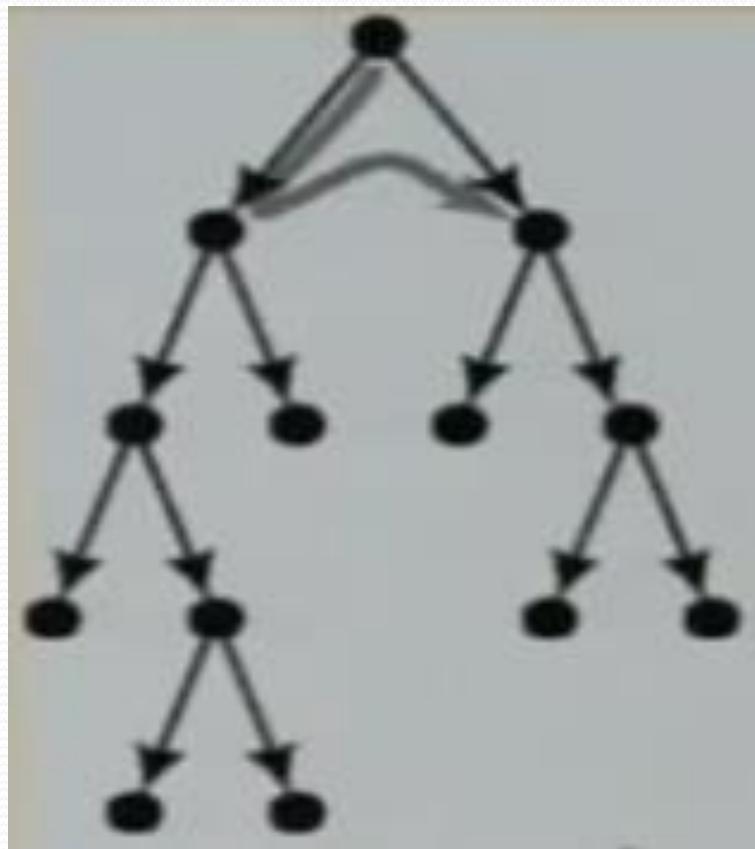
● Advantages

- ❖ It combines the positive aspects of depth first and breadth first search.
- ❖ Linear growing memory requirement of Depth first call
- ❖ The target node with the least depth is guaranteed to be found.

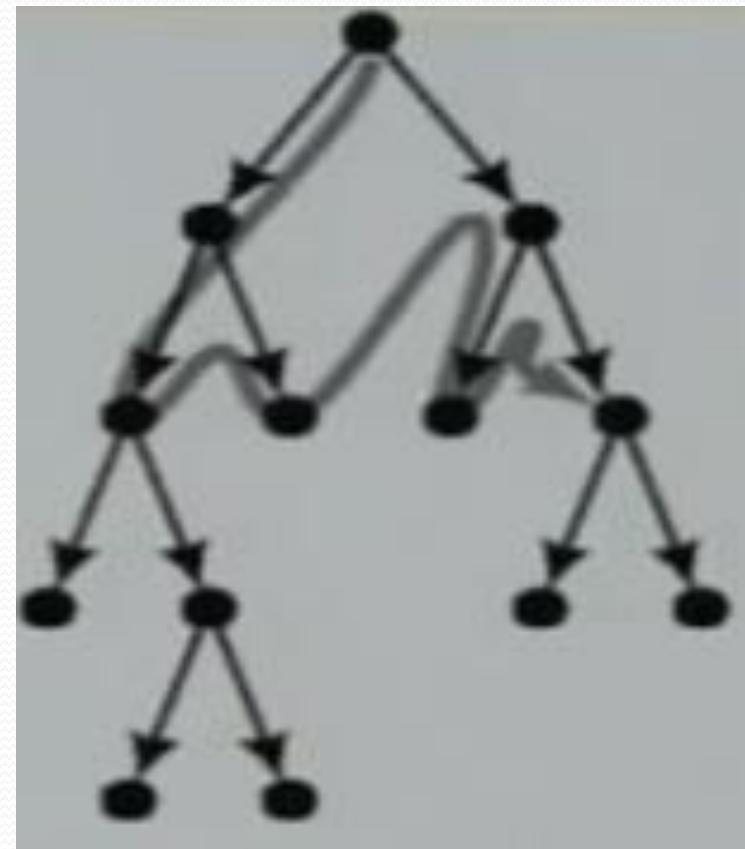
● Method

- ❖ Depth first calls, each with a depth limitation increased by 1, are combined. .

Depth First Iterative Deepening Search



Derinlik Sınırlaması = 1



Derinlik Sınırlaması = 2

Depth first iterative deepining arama

- Complete dir.
- Optimal: Eğer tüm operatörler aynı maliyete sahipse optimaldır. Diğer durumda optimal değil ancak en kısa yolu bulmayı garanti eder (BFS gibi)
- Zaman Karmaşıklığı BFS veya DFS den biraz daha kötü çünkü arama ağacının tepesindeki düğümler birçok defa oluşturuluyor, ancak neredeyse tüm düğümler ağacın altında olduğundan en kötü durumda karmaşıklığı hala $O(b^d)$

Depth first iterative deepining arama

- It is complete.
- Optimal: It is optimal if all operators have the same cost. Otherwise, it's not optimal but it guarantees to find the shortest path (like BFS)
- The Time Complexity is slightly worse than BFS or DFS because the nodes at the top of the search tree are created many times, but since almost all the nodes are at the bottom of the tree, the complexity is still $O(b d)$ in the worst case.

Yinelemeli derinleştirme araması / =0

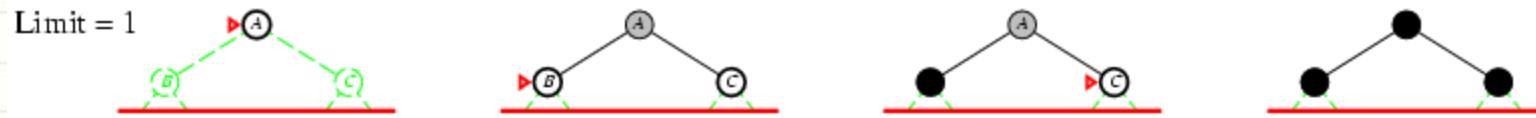
Depth first iterative deepining Searching

Limit = 0



Yinelemeli derinleştirme araması / =1

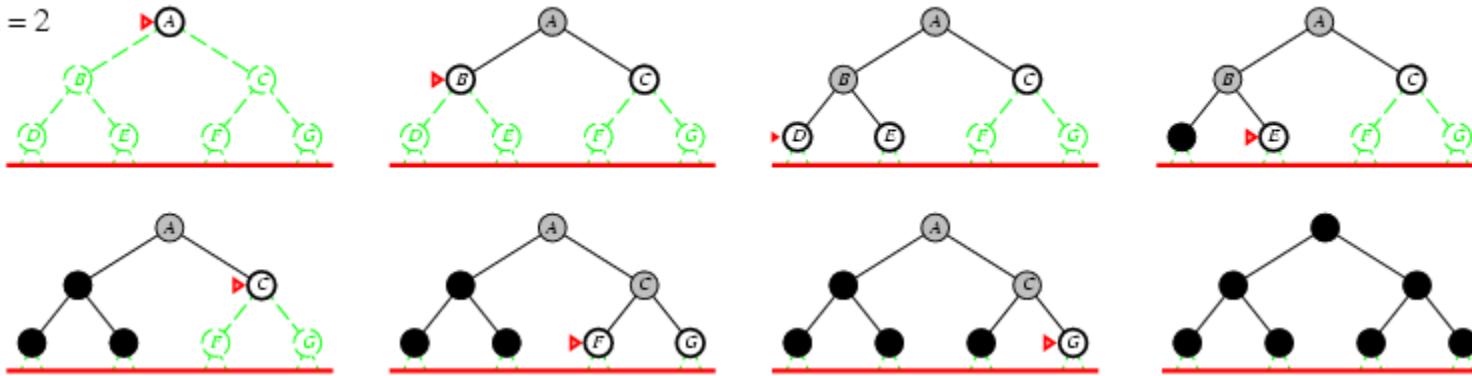
Depth first iterative deepining Searching



Yinelemeli derinleştirme araması / =2

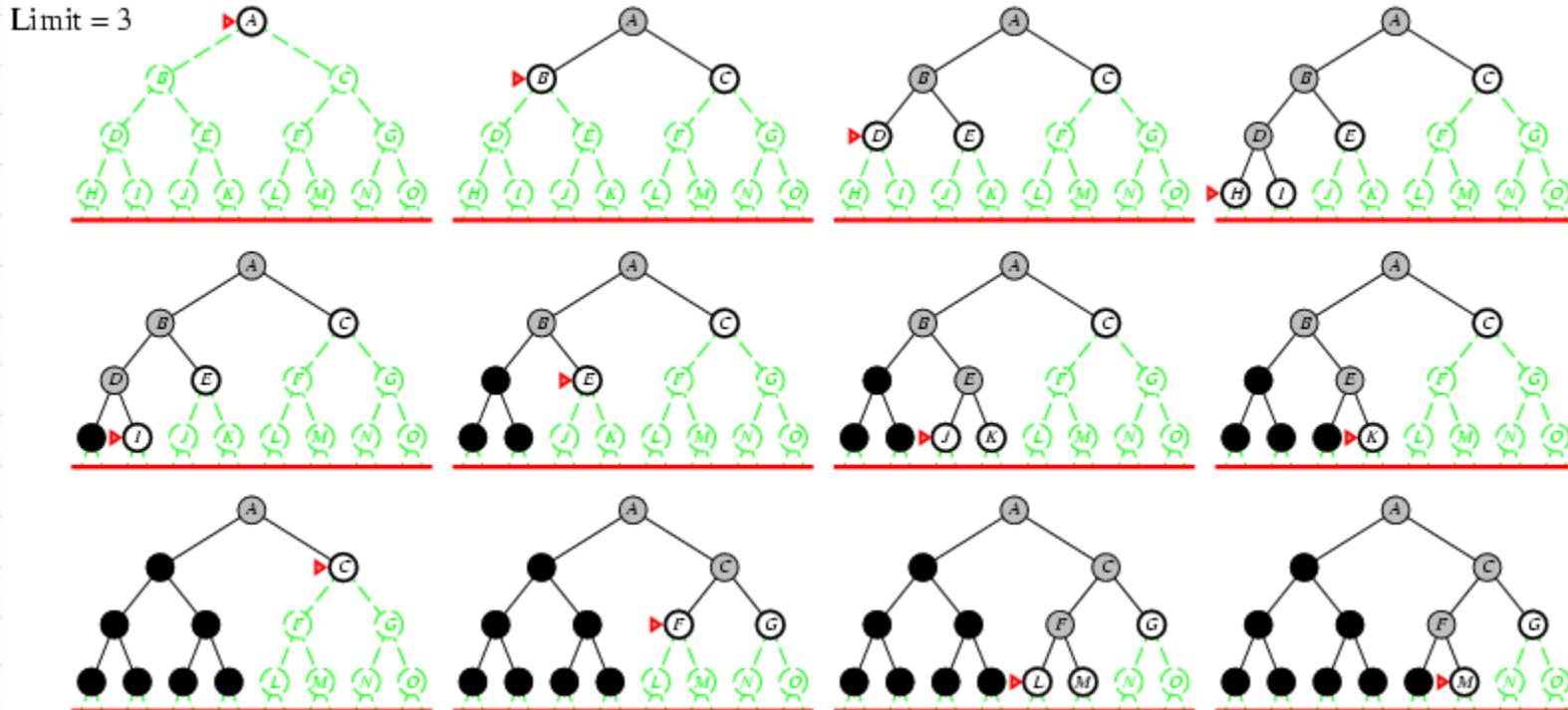
Depth first iterative deepining Searching

Limit = 2



Yinelemeli derinleştirme araması / =3

Depth first iterative deepining Searching



Depth first iterative deepening

Searching Özellikleri

- Completeness: Evet
 - Derinlik sınırlı da hedef düğüm sınır derinliğinin altında ise çözüm bulunamazdı
 - Yinelemeli derinleşende ise, her iterasyonda derinlik sınırını arttırarak çözümü bulmayı garantiler
- Time complexity: $O(b^d)$
- Space complexity: $O(b d)$
- Optimality: Evet
 - Çünkü kök düğüme en yakın çözümü bulur (eğer adım maliyeti = 1 ise)

b: dallanma faktörü

d: en düşük maliyetli çözümün derinliği

Depth first iterative deepening

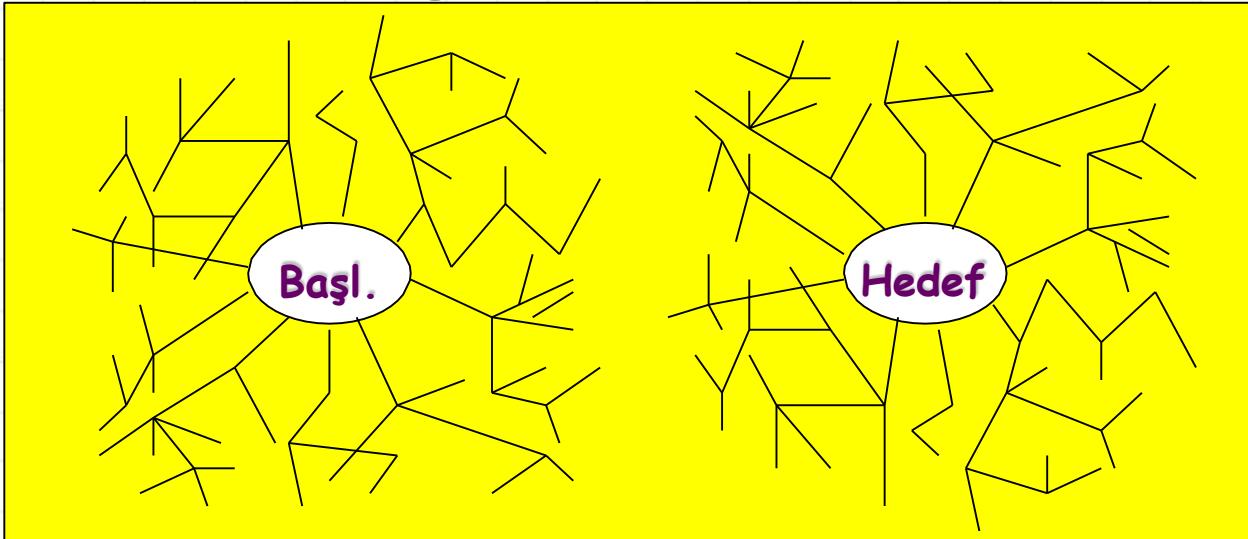
Searching Özellikleri

- Completeness: Evet/Yes
 - If the depth is limited but the target node is below the boundary depth, no solution could be found.
 - In iterative deepening, it ensures finding the solution by increasing the depth limit at each iteration.
- Time complexity: $O(b^d)$
- Space complexity: $O(b d)$
- Optimality: Evet/Yes
 - Because it finds the closest solution to the root node (if step cost = 1)

b: branching factor

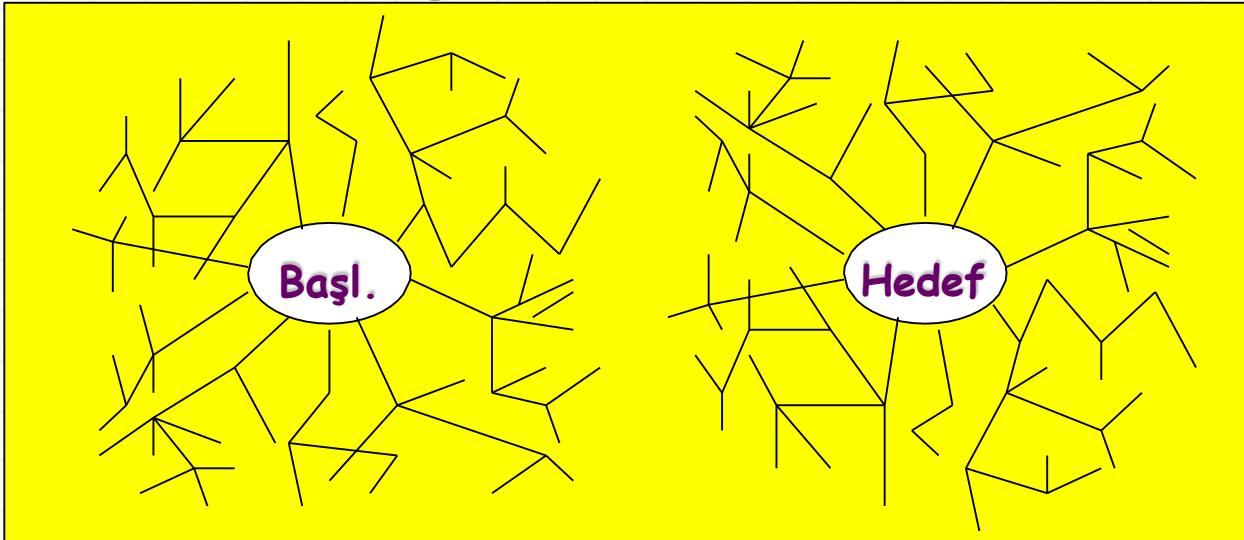
d: depth of least cost solution

İki Yönlü Arama - Bi-directional Searching



- Aramaya başlangıç ve hedef durumlarından aynı anda başlanır
- İki arama ortada karşılaştığı zaman biter
- Tek bir başlangıç ve amaç durumu olduğunda iyidir
- Çözüme daha hızlı ulaşmak mümkün olabilir

İki Yönlü Arama - Bi-directional Searching



- The search starts from the start and target states at the same time
- Ends when two calls meet in the middle
- It's good when there's a single start and purpose situation
- It may be possible to reach the solution faster

İki Yönlü Arama- Bi-directional Searching

- Hedef düğümden başlayarak önceki düğümler de (predecessor) sırayla üretilir
 - Bazı problemlerde öncekileri bulmak zor olabilir
- Birden fazla hedef durum var ise ne yapılabilir?
 - Hedef durum yerine hedef durum kümesine aynı işlemler uygulanabilir
 - Ancak, hedef durumların tespiti güç olabilir
 - **Örneğin** satrançta şah-mat amacını üretecek durumların testpiti?
- Yeni oluşturulacak bir düğümün arama ağacının diğer yanında yer alıp almadığını kontrol etmenin etkin bir yolu olmalıdır
- Her iki yanında ne çeşit aramanın yapılacağına karar vermek gereklidir (**Örn.**, genişlik-öncelikli?)