# هوش مصنوعی

## فصل دوم

## حل مسئله با جستجو

# فهرست

- عاملهای حل مسئله
- مسئله
- اندازه گیری کارایی حل مسئله
- جستجوی ناآگاهانه
- اجتناب از حالتهای تکراری
- جستجو با اطلاعات ناقص

# عاملهای حل مسئله

## چهار گام اساسی برای حل مسائل

- **فرموله کردن هدف:** وضعیتهای مطلوب نهایی کدامند؟
- **فرموله کردن مسئله:** چه فعالیتها و وضعیتهایی برای رسیدن به هدف موجود است؟
- **جستجو:** انتخاب بهترین دنباله از فعالیتهایی که منجر به حالاتی با مقدار شناخته شده میشود.
- **اجرا:** وقتی دنباله فعالیت مطلوب پیدا شد، فعالیتهای پیشنهادی آن میتواند اجرا شود.

# عامل حل مسئله

**function** SIMPLE-PROBLEM-SOLVING-AGENT (*percept*)
**returns** an action

**inputs**: *percept*, a percept

**static:** *seq*, an action sequence, initially empty

*state*, some description of the current world state

*goal*, a goal, initially null

*problem*, a problem formulation

*state* ← UPDATE-STATE (*state*, *percept*)

is empty *seq*

*goal* ← FORMULATE-GOAL (*state*)

*problem* ← FORMULAE-PROBLEM (*state*, *goal*)

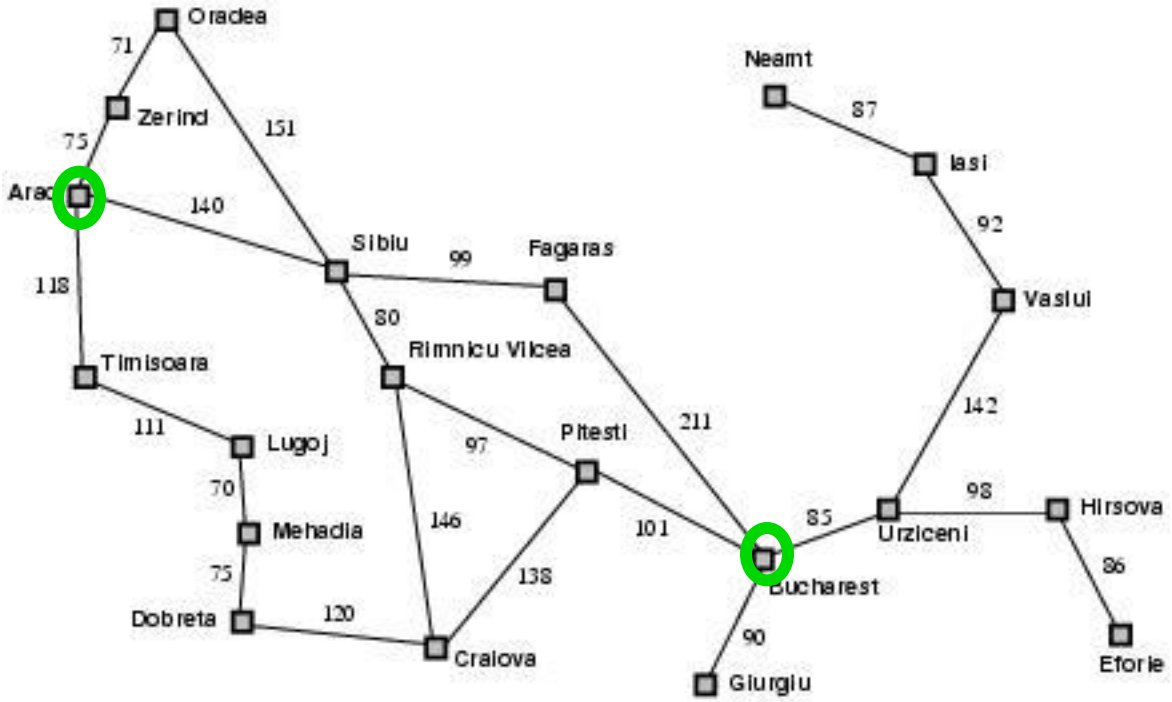
*seq* ←SEARCH (*problem*)

*action* ←FIRST (*seq*)

*seq* ←REST (*seq*)

**return** action

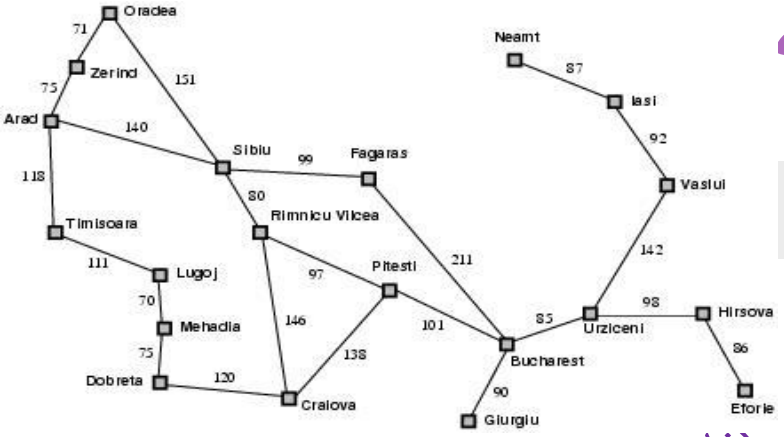
# مثال: نقشه رومانی



# مثال: نقشه رومانی

- صورت مسأله: رفتن از آراد به بخارست
- فرموله کردن هدف: رسیدن به بخارست
- فرموله کردن مسئله:
  - وضعیتها: شهرهای مختلف
  - فعالیتها: حرکت بین شهرها
- جستجو: دنباله ای از شهرها مثل:آراد، سیبیو، فاگارس، بخارست
  - این جستجو با توجه به کم هزینه ترین مسیر انتخاب میشود

# مسئله



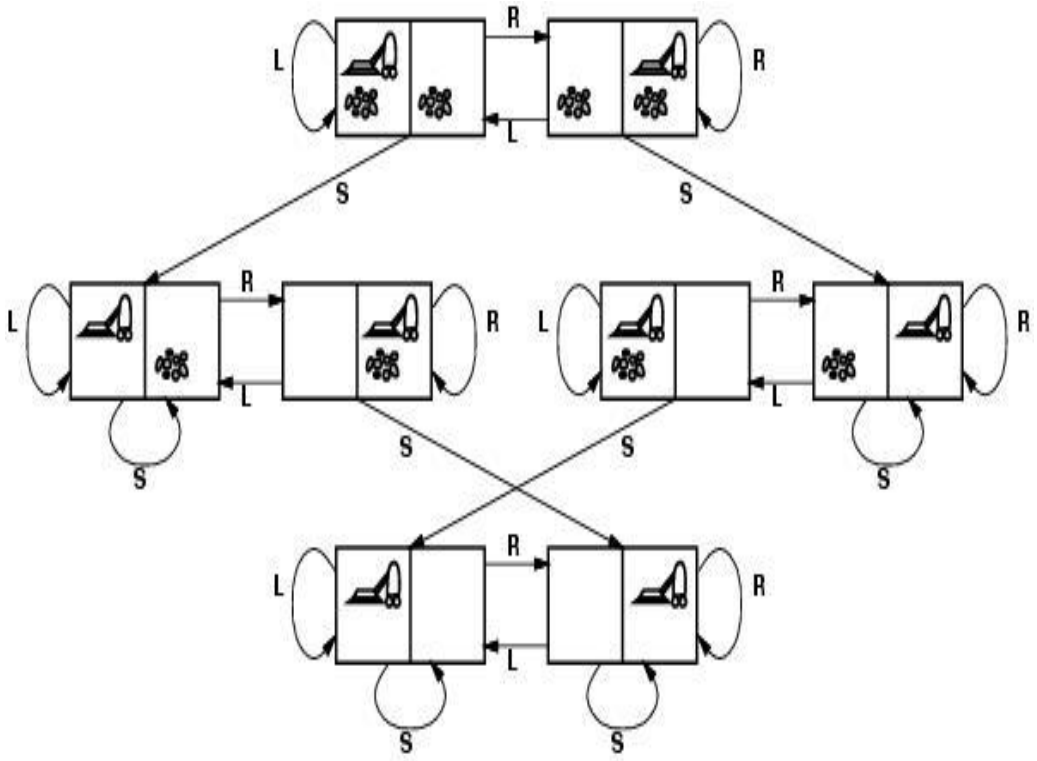

- حالت اولیه: حالتی که عامل از آن شروع میکند.
  - در مثال رومانی: شهر آراد In(Arad)
- *تابع جانشین: توصیفی از فعالیتهای ممکن که برای عامل مهیا است.*
  - *در مثال رومانی: S(Arad)={ Zerind,Sibui,Timisoara}*
- فضای حالت: مجموعه ای از حالتها که از حالت اولیه میتوان به آنها رسید.
  - در مثال رومانی: کلیه شهرها که با شروع از آراد میتوان به آنها رسید
- **تابع جانشین + حالت اولیه = فضای حالت**

# حل مسئله با جستجو

- آزمون هدف: تعیین میکند که آیا حالت خاصی، حالت هدف است یا خیر
  - هدف صریح: در مثال رومانی، رسیدن به بخارست
  - هدف انتزاعی: در مثال شطرنج، رسیدن به حالت کیش و مات
- مسیر: دنباله ای از حالتها که دنباله ای از فعالیتها را به هم متصل میکند.
  - در مثال رومانی: Arad, Sibiu, Fagaras یک مسیر است
- هزینه مسیر: برای هر مسیر یک هزینه عددی در نظر میگیرد.
  - در مثال رومانی: طول مسیر بین شهرها بر حسب کیلومتر
- **راه حل مسئله مسیری از حالت اولیه به حالت هدف است**
- **راه حل بهینه کمترین هزینه مسیر را دارد**

# مثال: دنیای جارو برقی

حالتها:



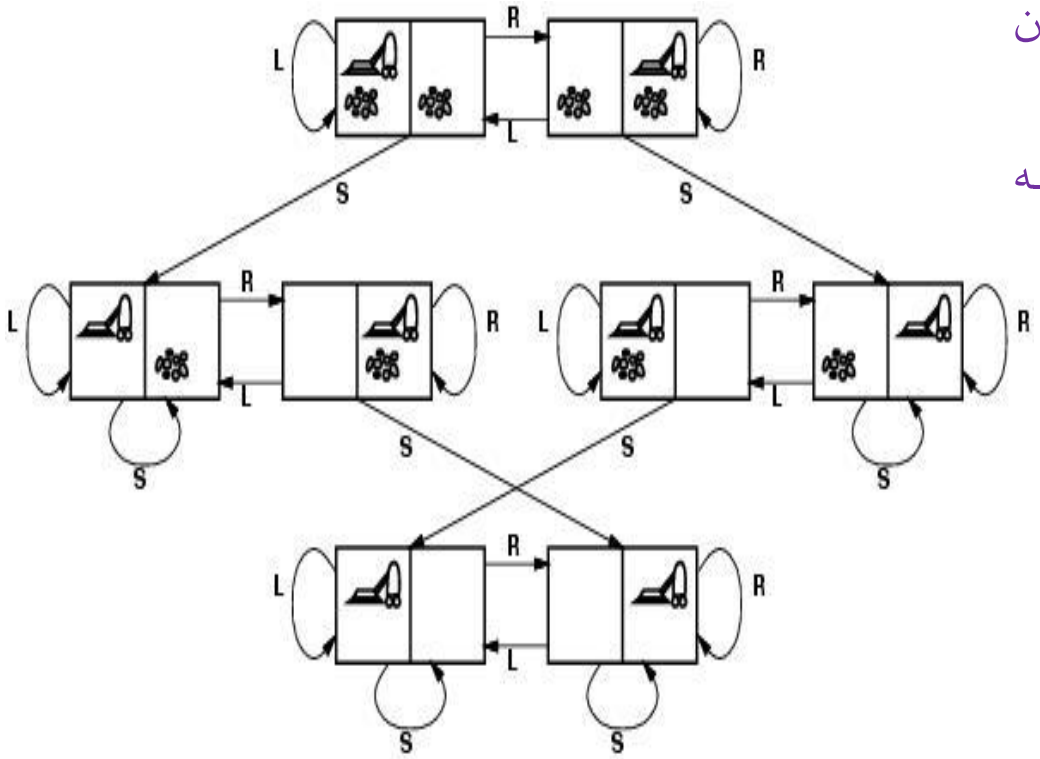

حالت اولیه:

تابع جانشین:

آزمون هدف:

هزینه مسیر:

# مثال: دنیای جارو برقی




حالتها: دو مکان که هر یک ممکن است کثیف یا تمیز باشند.لذا ۸ = ۲^۲*۲حالت در این جهان وجود دارد

حالت اولیه: هر حالتی میتواند به عنوان حالت اولیه طراحی شود

تابع جانشین: حالتهای معتبر از سه عملیات: راست، چپ، مکش

آزمون هدف: تمیزی تمام مربعها

هزینه مسیر: تعداد مراحل در مسیر

# مثال: معمای 8

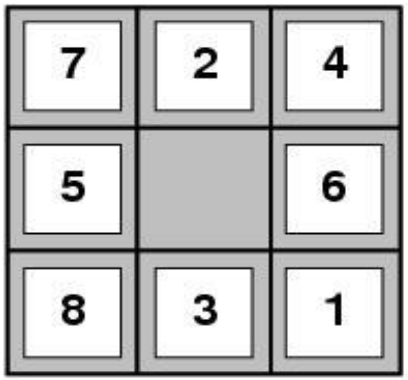
حالتها:

حالت اولیه:

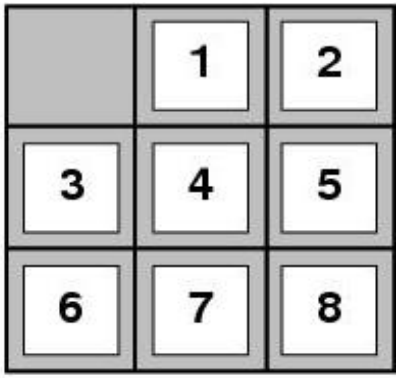
تابع جانشین:

آزمون هدف:

هزینه مسیر:




Start State




Goal State

# مثال: معمای 8

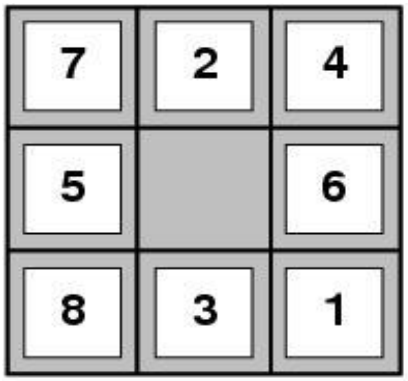
حالتها: مکان هر هشت خانه شماره دار و خانه خالی در یکی از ۹ خانه

حالت اولیه: هر حالتی را میتوان به عنوان حالت اولیه در نظر گرفت

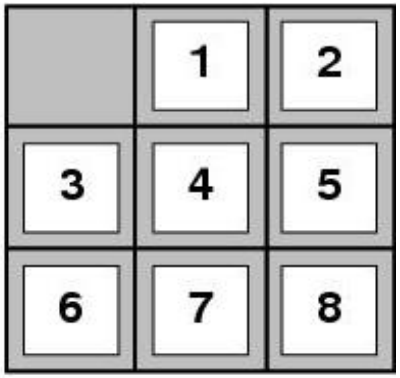
تابع جانشین: حالتهای معتبر از چهار عمل، انتقال خانه خالی به چپ، راست، بالا یا پایین

آزمون هدف: بررسی میکند که حالتی که اعداد به ترتیب چیده شده اند(طبق شکل روبرو) رخ داده یا نه

هزینه مسیر: برابر با تعداد مراحل در مسیر

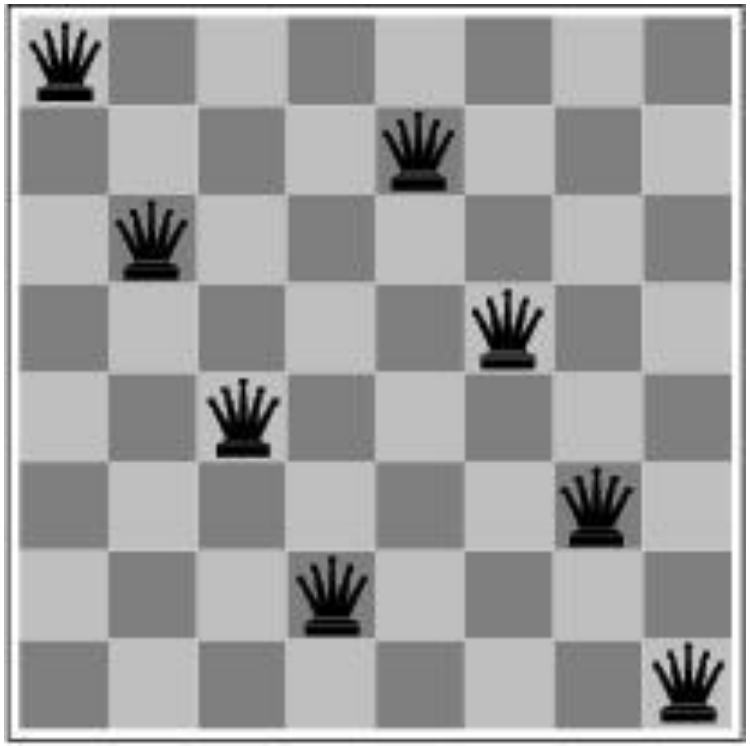



Start State




Goal State

# مثال: مسئله 8 وزیر



فرمول بندی افزایشی

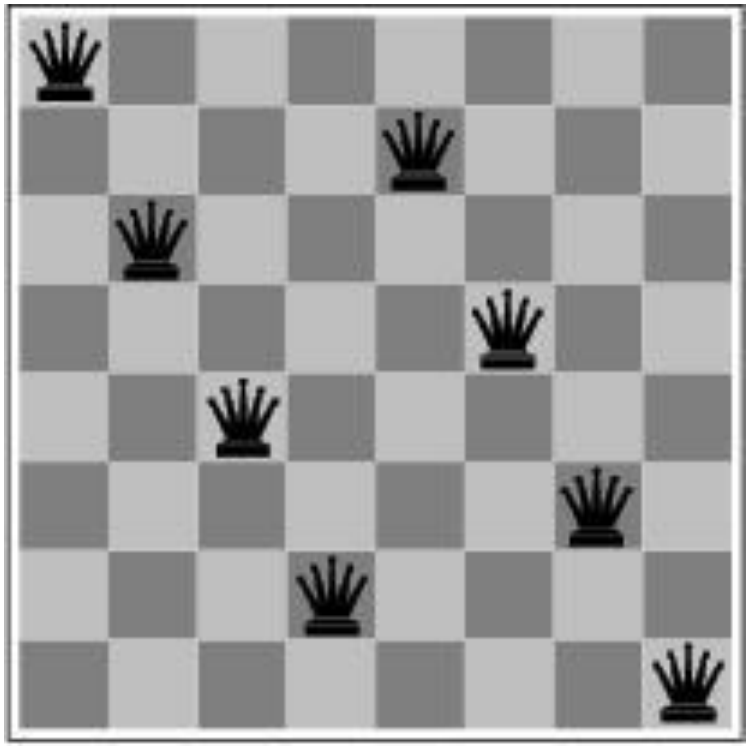
حالتها:

حالت اولیه:

تابع جانشین:

آزمون هدف:

# مثال: مسئله 8 وزیر



## فرمول بندی افزایشی

حالتها: هر ترتیبی از ۰ تا ۸ وزیر در صفحه، یک حالت است

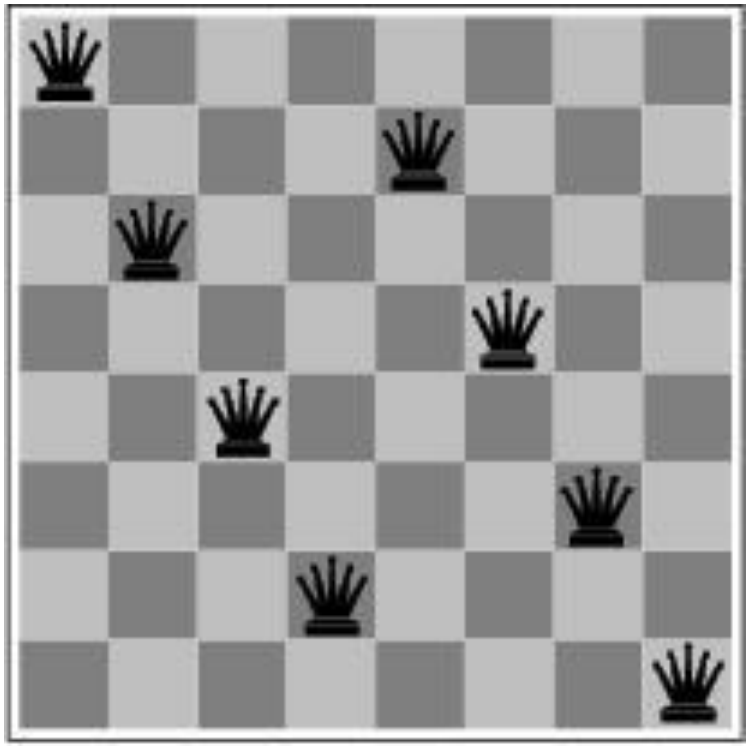
حالت اولیه: هیچ وزیری در صفحه نیست

تابع جانشین: وزیری را به خانه خالی اضافه میکند

آزمون هدف: ۸وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمیگیرند

**در این فرمول بندی باید ۱۰^۱۴*۳ دنباله ممکن بررسی میشود**

# مثال: مسئله 8 وزیر



فرمول بندی حالت کامل

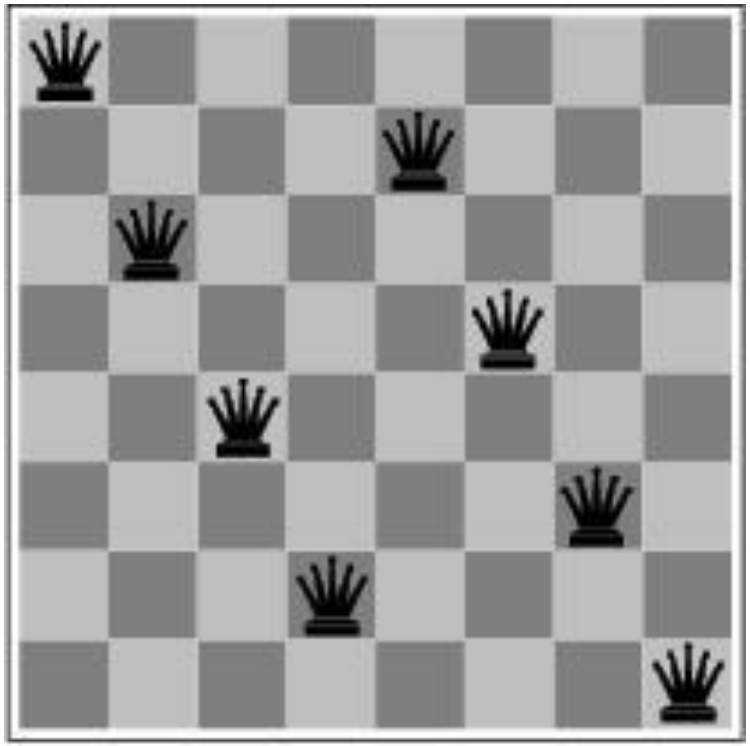
حالتها:

حالت اولیه:

تابع جانشین:

آزمون هدف:

# مثال: مسئله 8 وزیر



## فرمول بندی حالت کامل

حالتها: چیدمان n وزیر $(0 \le n \le 8)$، بطوریکه در هر ستون از n ستون سمت چپ، یک وزیر قرار گیرد و هیچ دو وزیری بهم گارد نگیرند

حالت اولیه: با ۸ وزیر در صفحه شروع میشود

تابع جانشین: وزیری را در سمت چپ ترین ستون خالی قرار میدهد، بطوری که هیچ وزیری آن را گارد ندهد

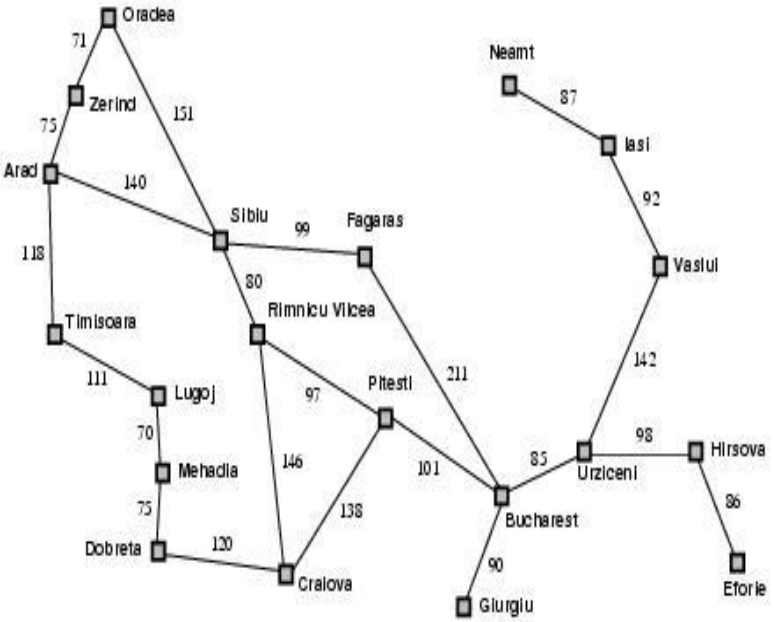
آزمون هدف: ۸وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمیگیرند

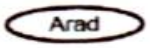
**این فرمول بندی فضای حالت را از ۳*۱۰^۱۴ به ۲۰۵۷ کاهش میدهد**

# جستجو برای راه حل

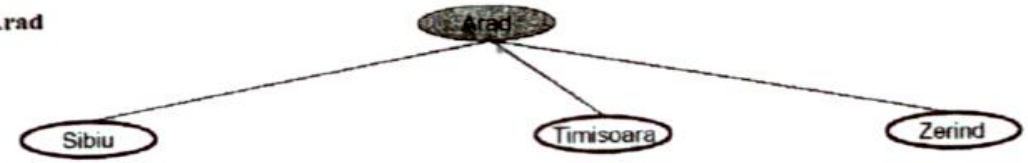
- بعد از فرموله سازی  نیاز به حل مسائل داریم.
- این عمل از طریق جستجو در گراف فضای حالت انجام خواهد شد.
- در مثال کشور رومانی :
  - ریشهٔ درخت جستجو یک گره جستجو است که به حالت آغازین In(Arad) وابسته است.
  - اولین مرحلهٔ آزمون آن است که آیا این گره هدف است؟
  - از آنجا که این حالت هدف نیست، توسعهٔ حالت جاری انجام می شود.
  - یعنی تابع مابعد را به حالت جاری اعمال نموده و مجموعهٔ جدیدی از حالات تولید خواهد شد.

# جستجو برای راه حل

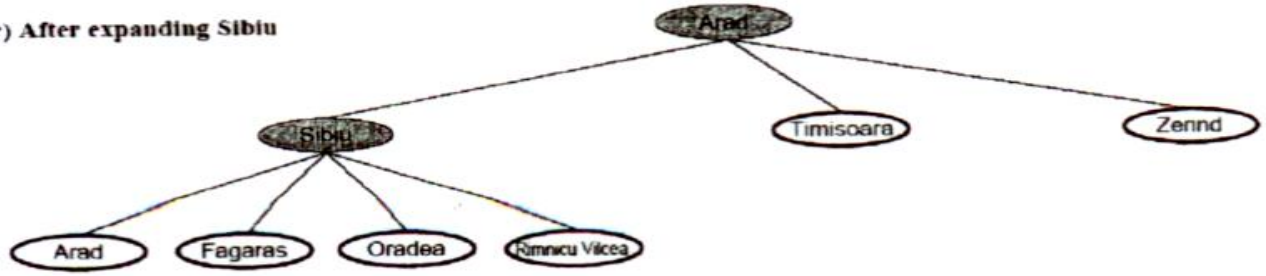


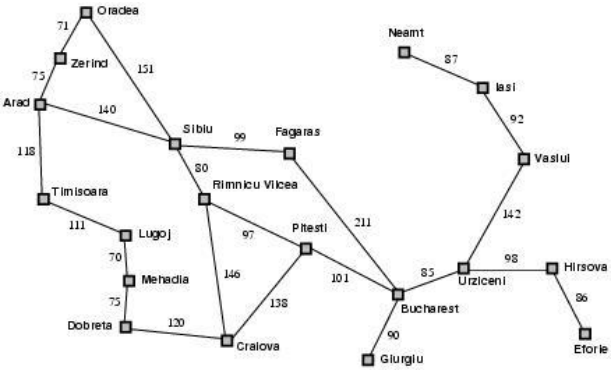




(b) After expanding Arad




(c) After expanding Sibiu






# جستجو برای راه حل

فرض می کنیم که یک گره دارای ساختار داده ای با پنج مولفه است:

- حالت: حالت در فضای حالت که گره به آن وابسته است.
- گره والد: گره ای در درخت جستجو که این گره را تولید کرده است.
- عمل: عملی که به والد اعمال شده تا گره را تولید کند.
- هزینه مسیر: هزینه معمولاً با g(n) بیان می شود که شامل مسیر از گره آغازین تا گره جاری است.
- عمق: تعداد مراحل در طول مسیر از حالت آغازین

# جستجو برای راه حل

- مجموعه گره ها را به صورت یک صف پیاده سازی می کنیم. عملیات روی صف به قرار زیرند:
- Make-Queue(element,...) صفی با عناصر داده شده می سازد.
- Empty(queue) درست بر می گرداند اگر عنصر دیگری در صف نباشد.
- First(queue) اولین عنصر صف را بر می گرداند.
- Remove-First(queue) ابتدا First(queue) را بر می گرداند و سپس آن را از صف حذف می کند.
- Insert(element, queue) عنصر در صف درج می کند و صف حاصل را بر می گرداند
- Insert-All(elements, queue) مجموعه عناصری را داخل صف درج کرده و صف را بر می گرداند.

# الگوریتم عمومی جستجوی درخت

**function** TREE-SEARCH (*problem*, *fringe*) **returns** 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*)

# الگوریتم عمومی جستجوی درخت

**function** EXPAND (*node*, *problem*) **returns** a set of nodes

*successors* ← the empty set

**For each** (*action*, *result*) in SUCCESSOR-FN [*problem*] )STATE[*node*] **do**

*s* ← a new NODE

STATE [*s*] ← *result*

PARENT-NODE[*s*] ← *node*

ACTION[*s*] ← *action*

PATH-Cost[*s*] ← PATH-Cost [*node*] +STEP-Cost (*node*, *action*, *s*)

DEPTH[*s*] ← DEPTH [*node*]+1

add s to *successors*

**return** *successors*

# اندازه گیری کارایی حل مسئله

- کامل بودن: آیا الگوریتم تضمین میکند که در صورت وجود راه حل، آن را بیابد؟
- بهینگی: آیا این راهبرد، راه حل بهینه ای را ارائه میکند.
- پیچیدگی زمانی: چقدر طول میکشد تا راه حل را پیدا کند؟
  - تعداد گره های تولید شده در اثنای جستجو
- پیچیدگی فضا: برای جستجو چقدر حافظه نیاز دارد؟
  - حداکثر تعداد گره های ذخیره شده در حافظه

**پیچیدگی به صورت سه کمیت بیان می شود:**

- **b** فاکتور انشعاب یا حداکثر تعداد مابعدهای یک گره،
- **d** عمق کم سطح ترین گره هدف،
- **m** طول حداکثر هر مسیری در فضای حالت.

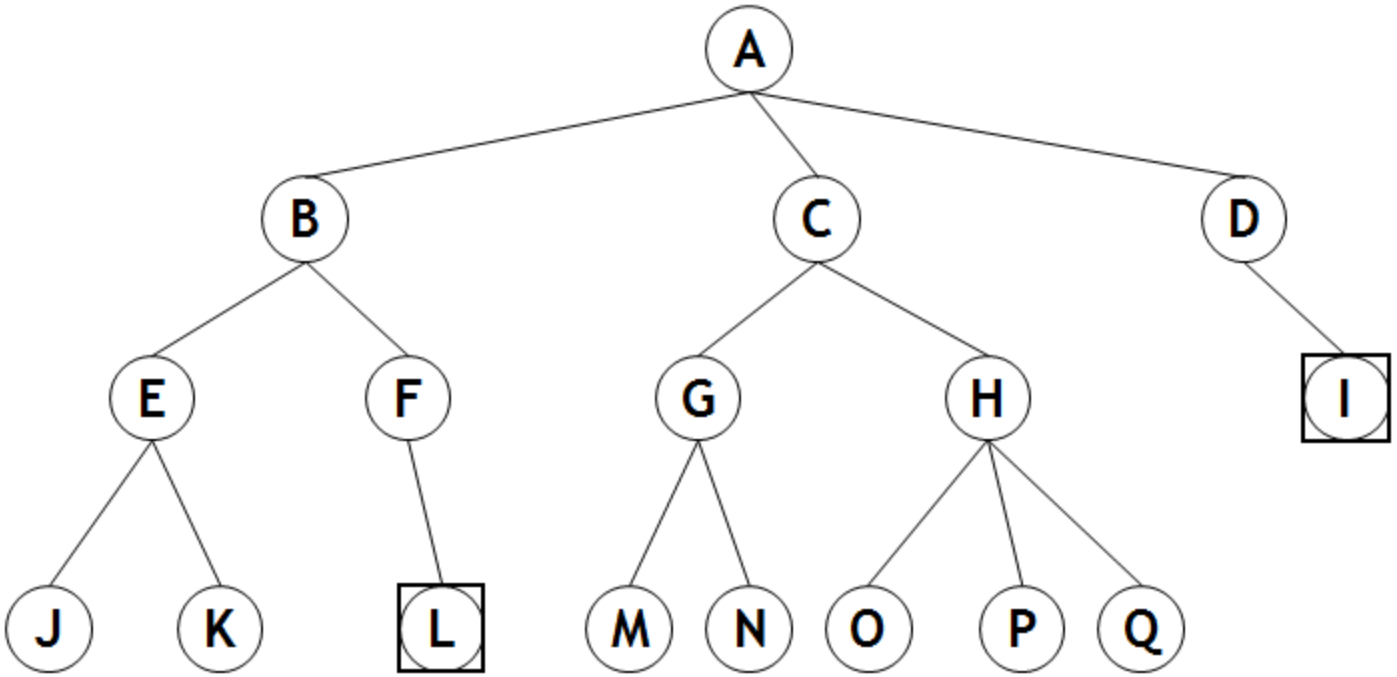
# انواع جستجو

- جستجوی ناآگاهانه : الگوریتم هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارد این الگوریتمها فقط میتوانند جانشینهایی را تولید و هدف را از غیر هدف تشخیص دهند
- جستجوی آگاهانه : راهبردهایی که تشخیص میدهد یک حالت غیر هدف نسبت به گره غیر هدف دیگر، امید بخش تر است، جست و جوی آگاهانه یا جست و جوی اکتشافی نامیده میشود.

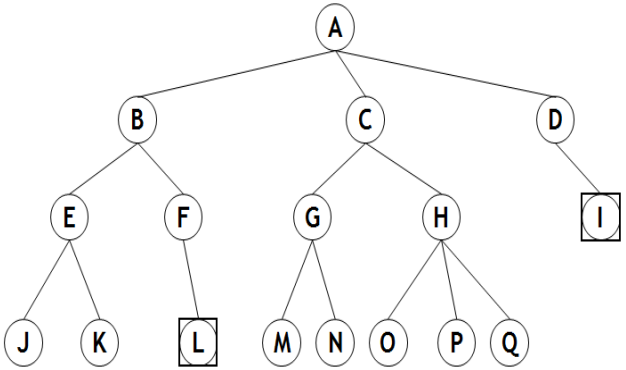
## راهبردهای ناآگاهانه:

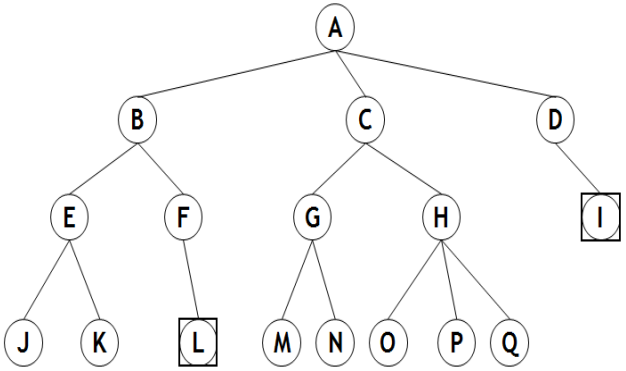
- جست و جوی عرضی
- جست و جوی عمقی
- جست و جوی عمیق کننده تکراری
- جست و جوی هزینه یکنواخت
- جست و جوی عمقی محدود
- جست و جوی دو طرفه

# BREADTH-FIRST SEARCH جستجوی سطحی(عرضی)

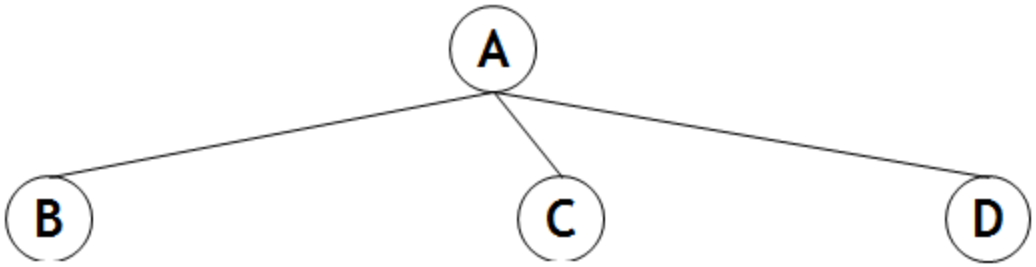


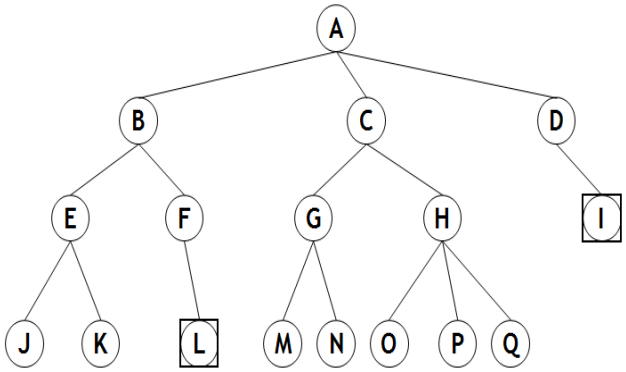
# BREADTH-FIRST SEARCH(جستجوی سطحی(عرضی



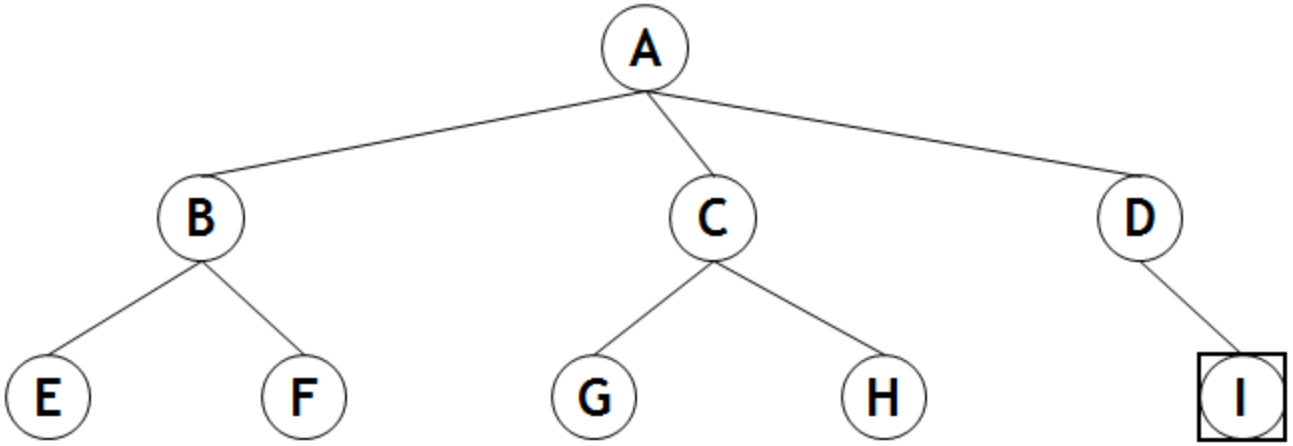



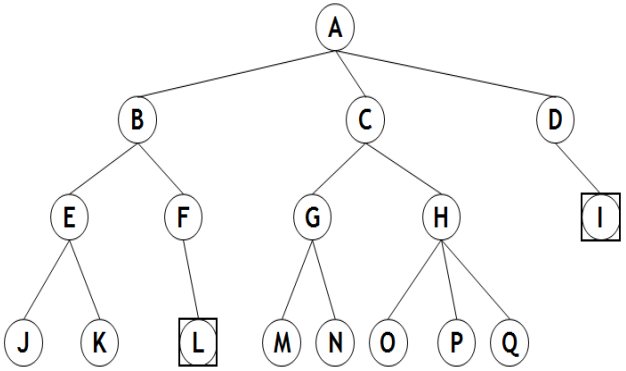

# جستجوی سطحی(عرضی)



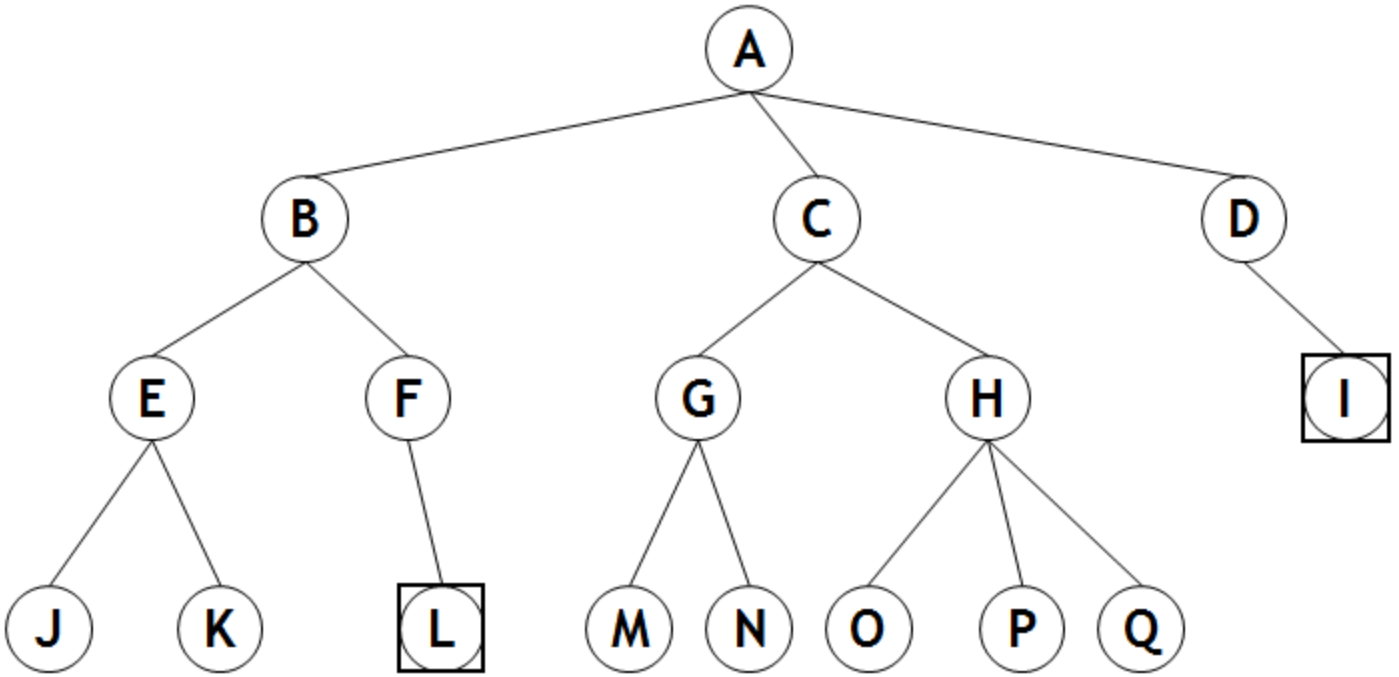



# جستجوی سطحی(عرضی)

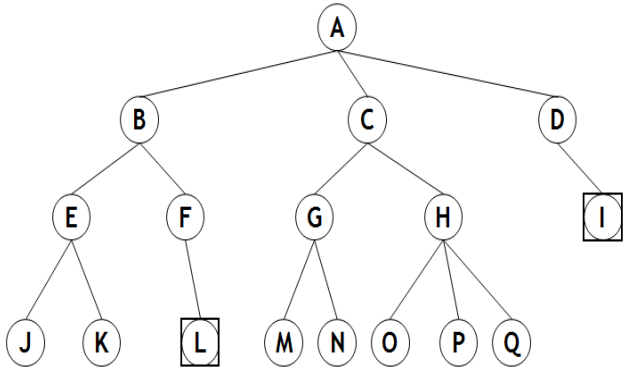





# جستجوی سطحی(عرضی)



# Breadth-First Search جستجوی سطحی(عرضی)




کامل بودن: بله

بهینگی: بله (مشروط)

در صورتی بهینه است که هزینه مسیر، تابعی غیر نزولی از عمق گره باشد.(مثل وقتی که فعالیتها هزینه یکسانی دارند)

پیچیدگی زمانی: $O(b^{d+1})$

$$1+b+b^2+b^3+\ldots+b^d+\left(b^{d+1}-b\right)$$

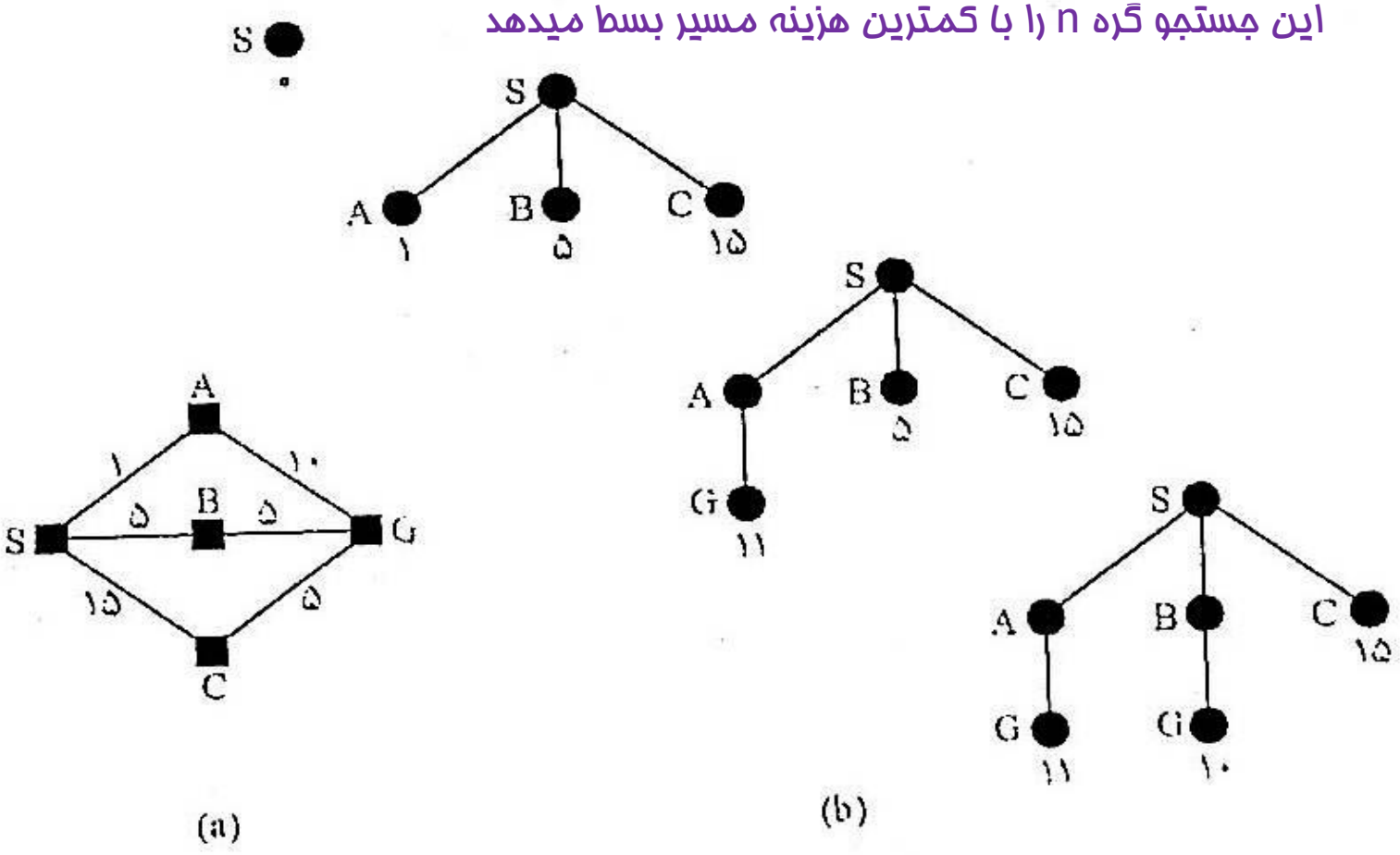
پیچیدگی فضا: $O(b^{d+1})$

# نیازهای زمانی و فضایی جستجوی اول- سطح

| Depth | Nodes | Time | Memory |
|---|---|---|---|
| 2 | 1100 | 11 seconds | 1 megabyte |
| 4 | 111,100 | 11 seconds | 106 megabytes |
| 6 | $10^7$ | 19minutes | 10 gigabytes |
| 8 | $10^9$ | 31 hours | 1 terabytes |
| 10 | $10^{11}$ | 129 days | 101 terabytes |
| 12 | $10^{13}$ | 35years | 10 petabytes |
| 14 | $10^{15}$ | 5,523 years | 1exabyte |

# جستجوی هزینه یکنواخت

این جستجو گره n را با کمترین هزینه مسیر بسط میدهد



# جستجوی هزینه یکنواخت

**کامل بودن:** بله

هزینه هر مرحله بزرگتر یا مساوی یک مقدار ثابت و مثبت ε باشد.(هزینه مسیر با حرکت در مسیر افزایش می یابد)

**بهینگی:** بله

هزینه هر مرحله بزرگتر یا مساوی **ε** باشد

**پیچیدگی زمانی:** $O(b^{[C^*/\varepsilon]})$

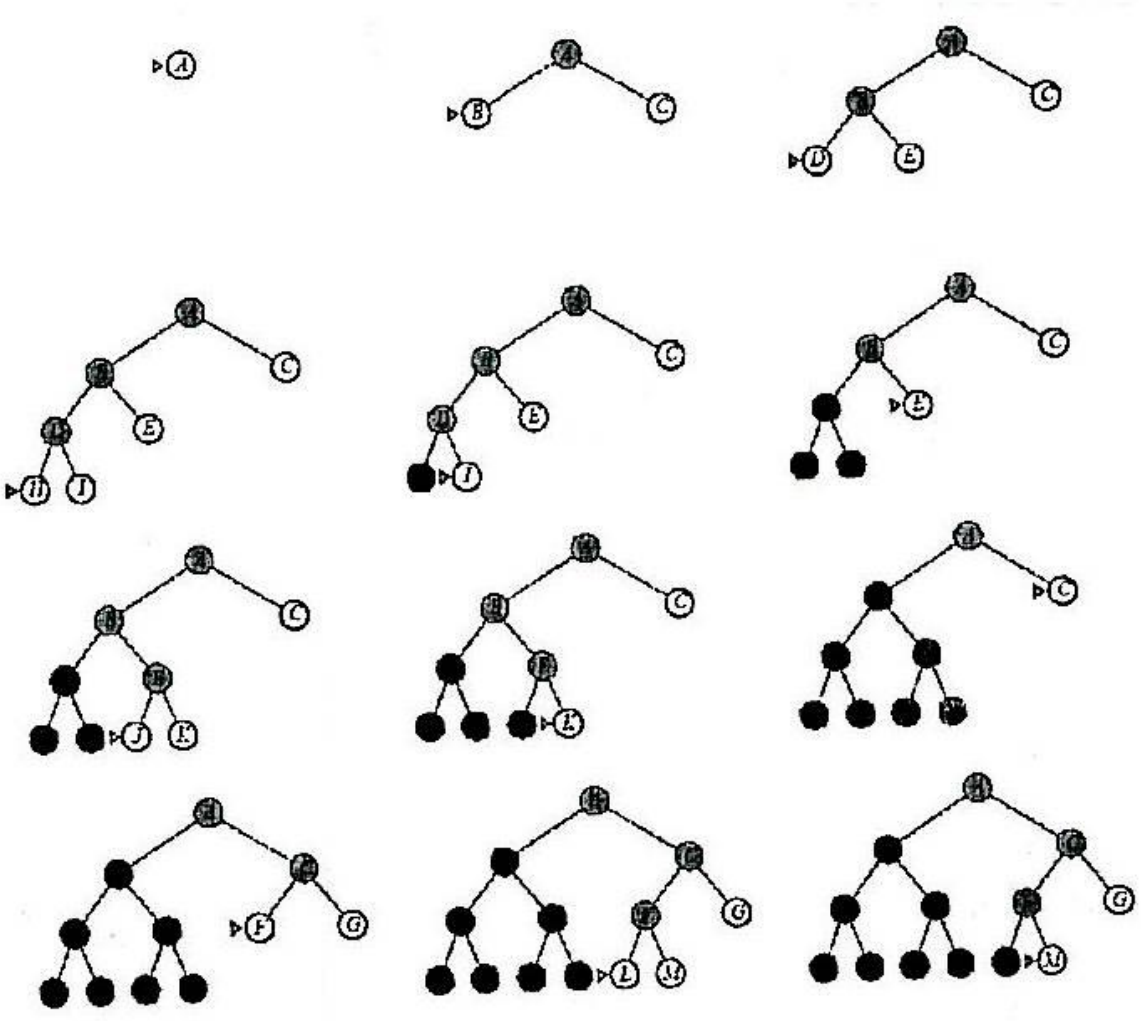
**پیچیدگی فضا:** $O(b^{[C^*/\varepsilon]})$

# جستجوی عمقی DEPTH-FIRST-SEARCH

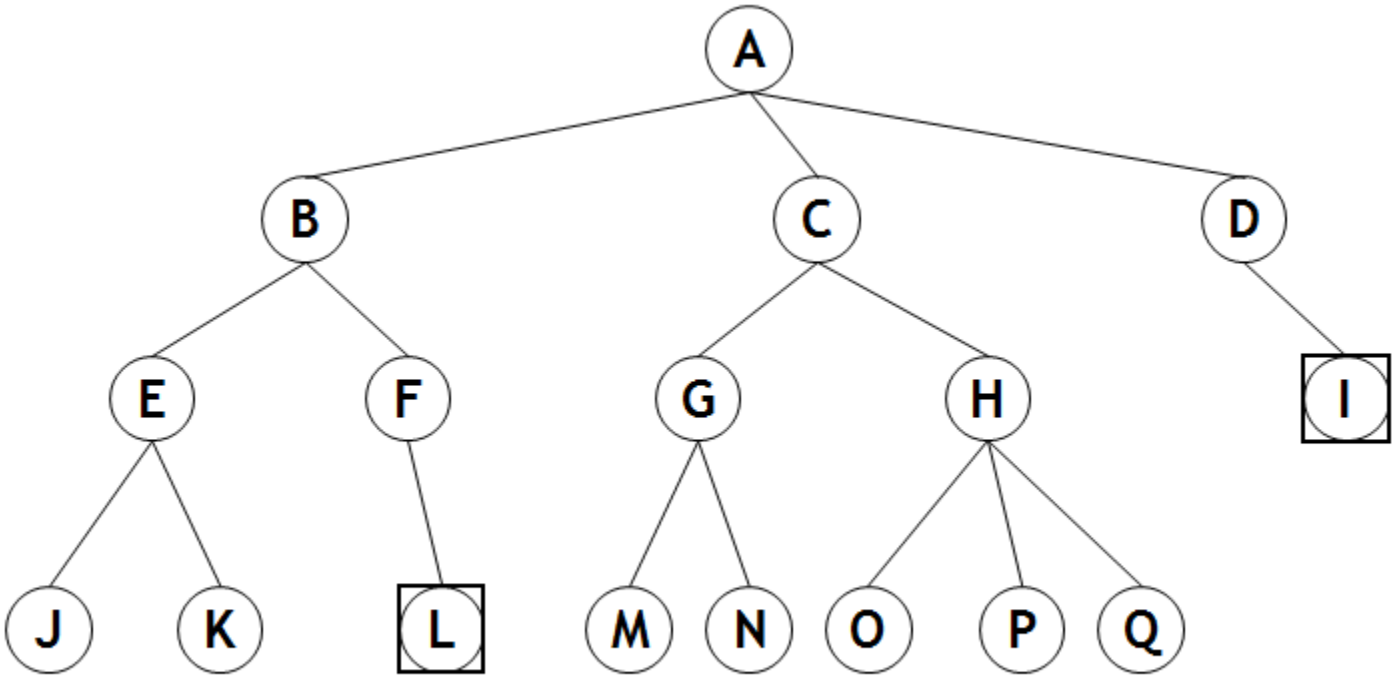
- این استراتژی می تواند بوسیله tree-search توسط پشته (صف آخر- ورود- اول- خروج (LIFO)) پیاده سازی شود.

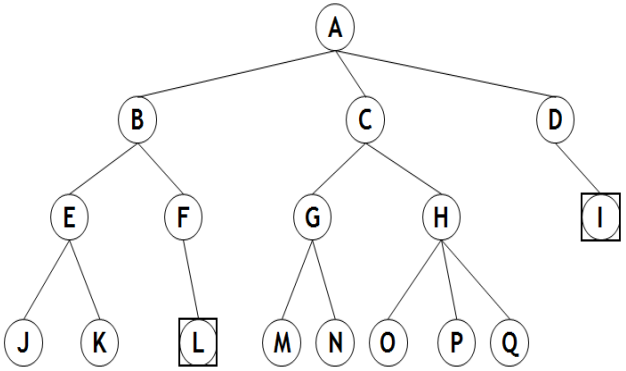
```
function Depth-First-Search (problem) return solution
 {
    Return TREE-SEARCH( problem , Enque-at-front)
 }
```

# DEPTH-FIRST-SEARCH جستجوی عمقی

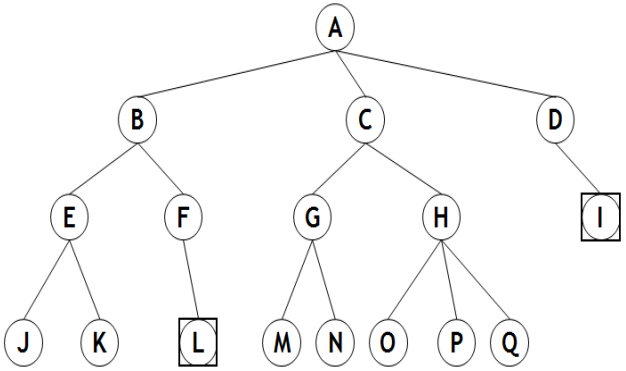


# DEPTH-FIRST-SEARCH جستجوی عمقی

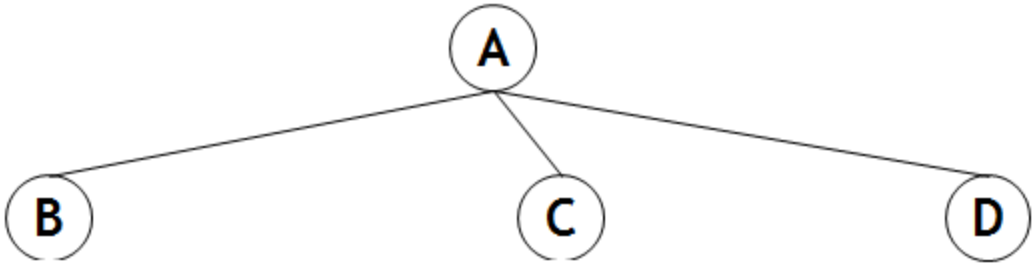


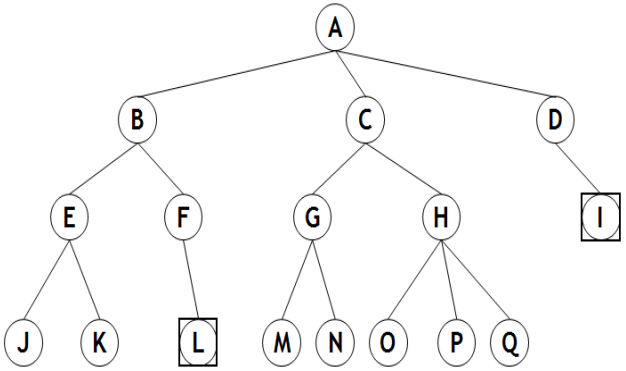



# جستجوی عمقی

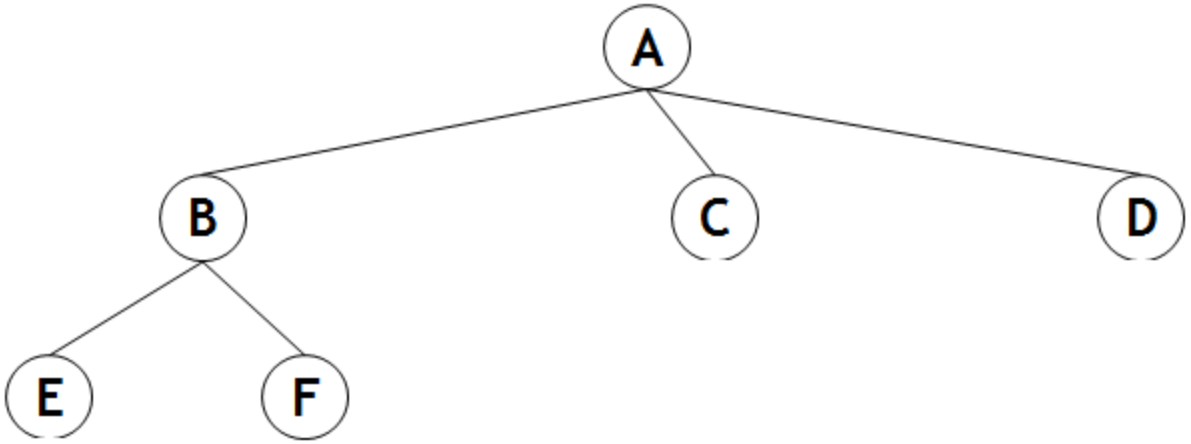


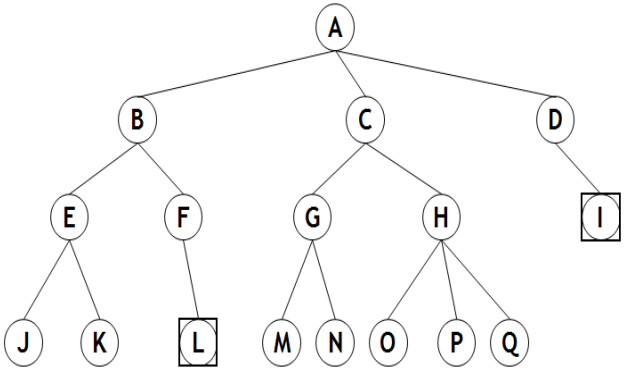

# جستجوی عمقی



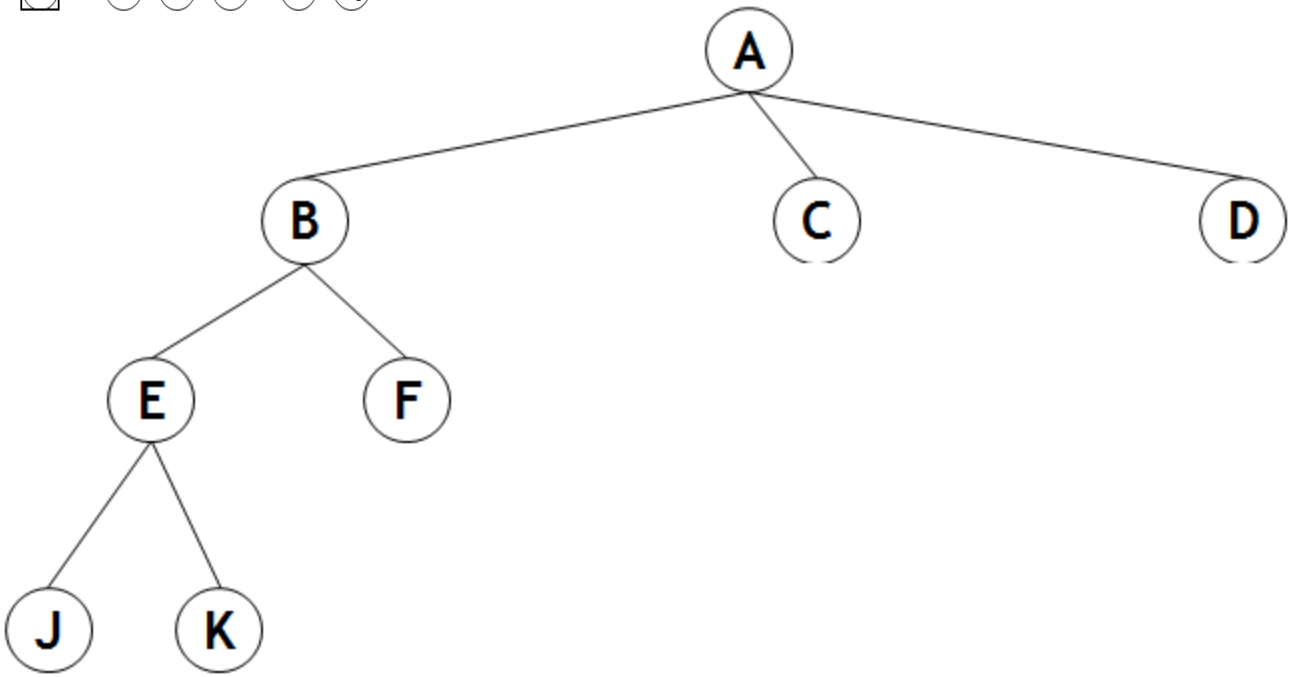


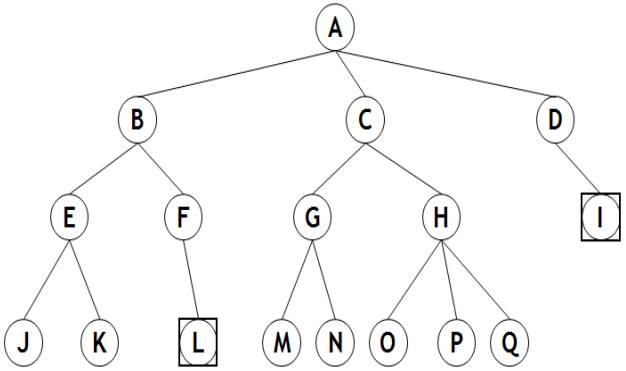

# جستجوی عمقی



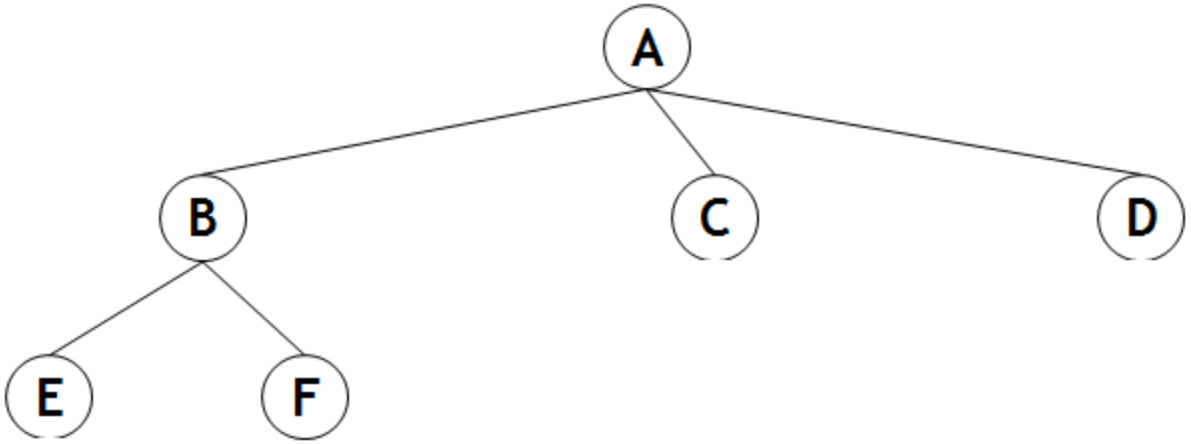


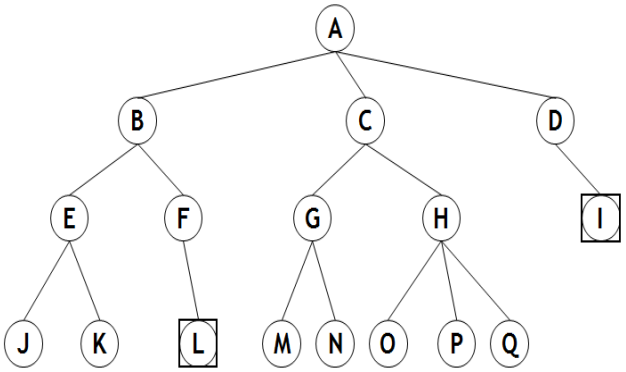

# جستجوی عمقی



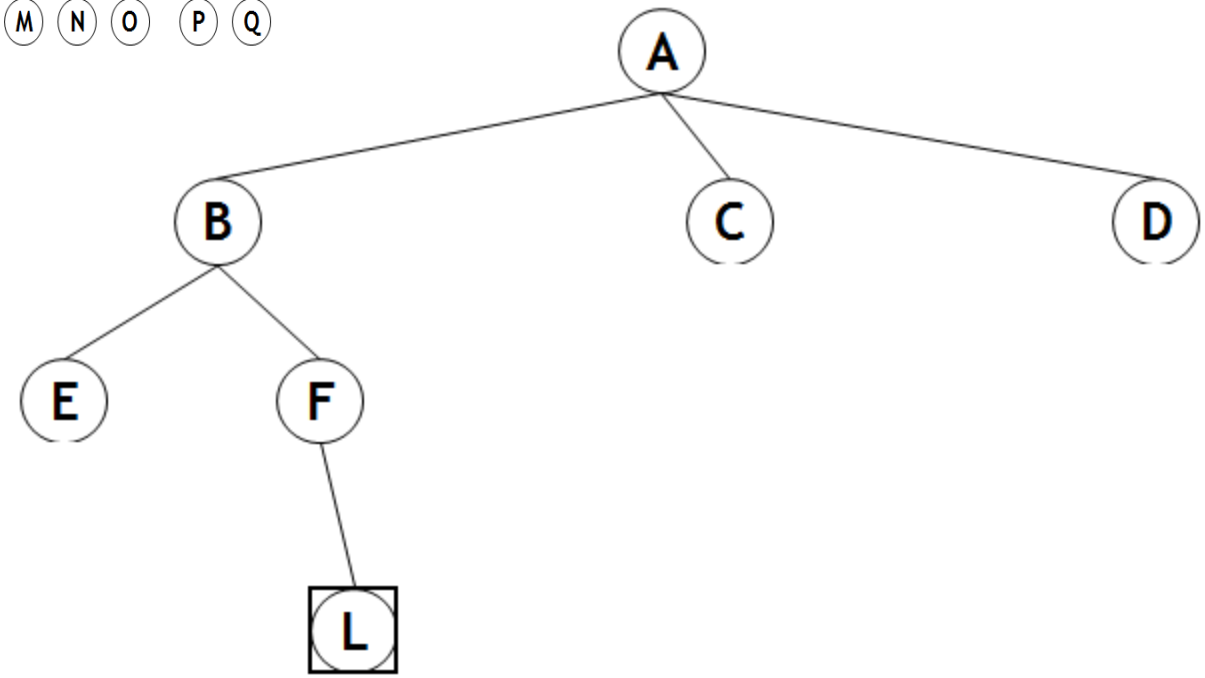


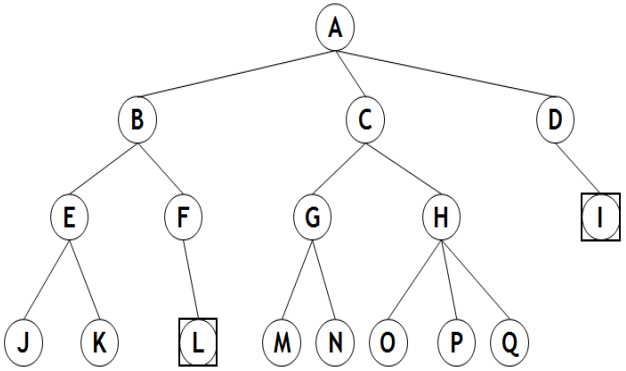

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



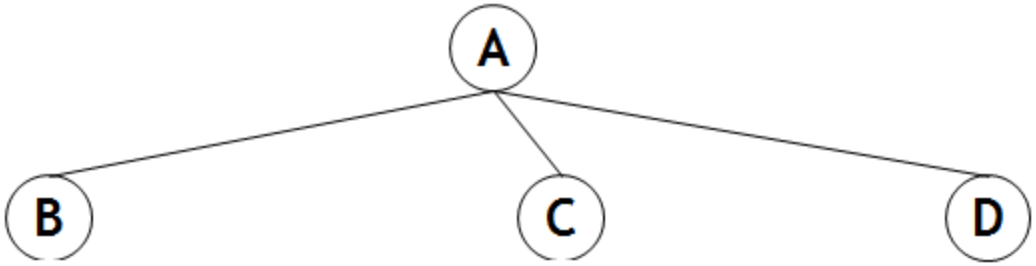


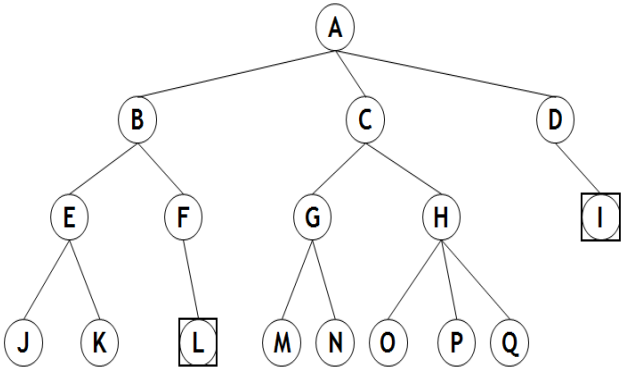

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



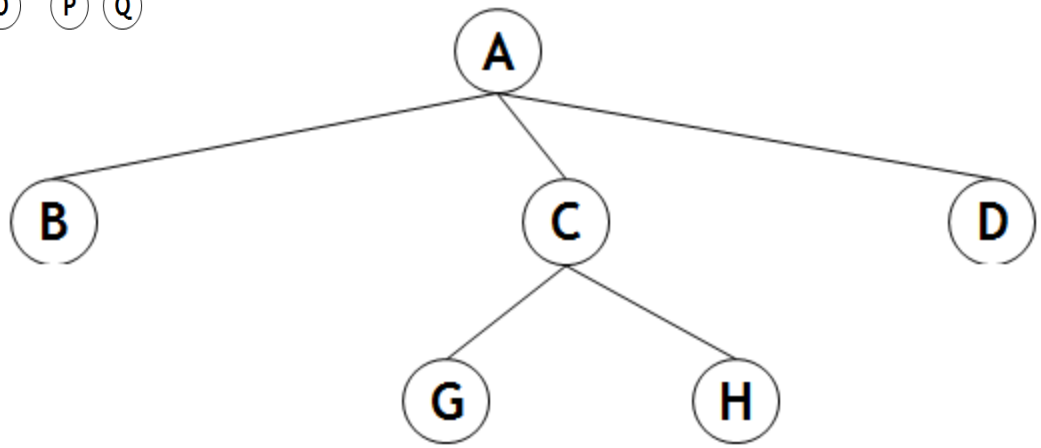


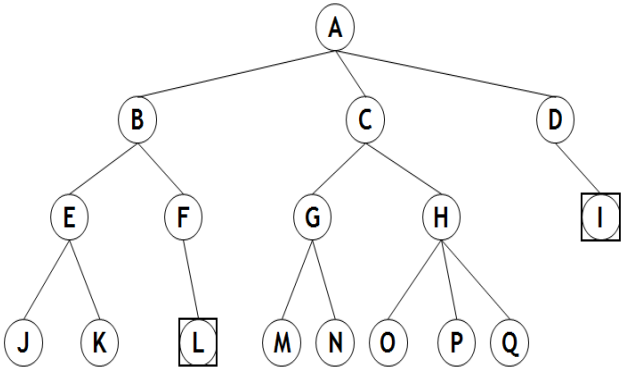

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



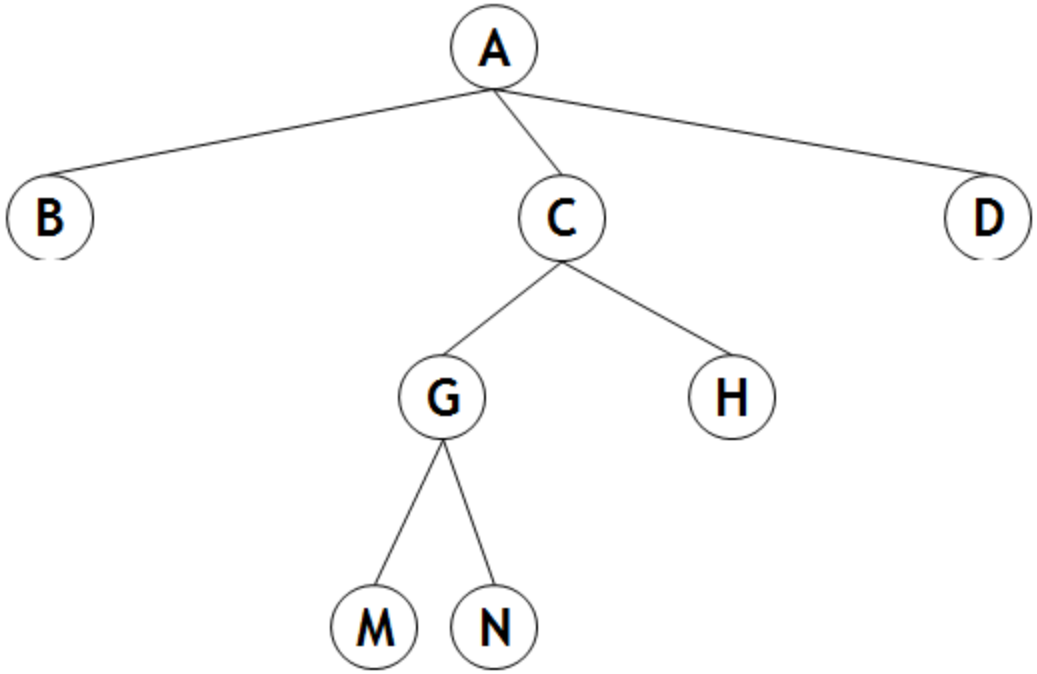


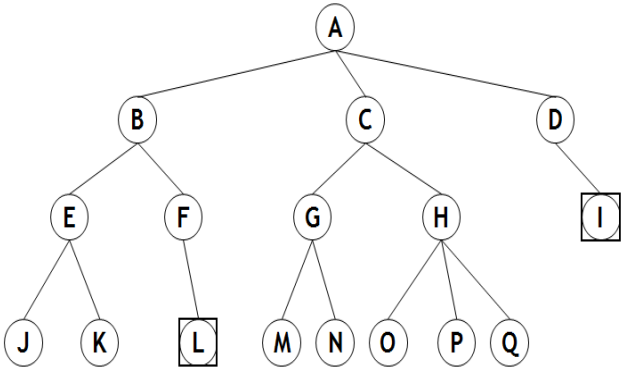

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



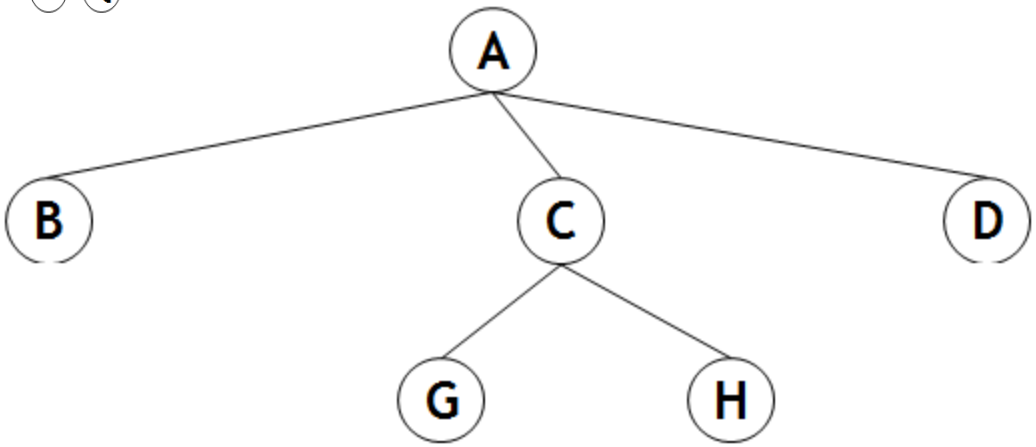


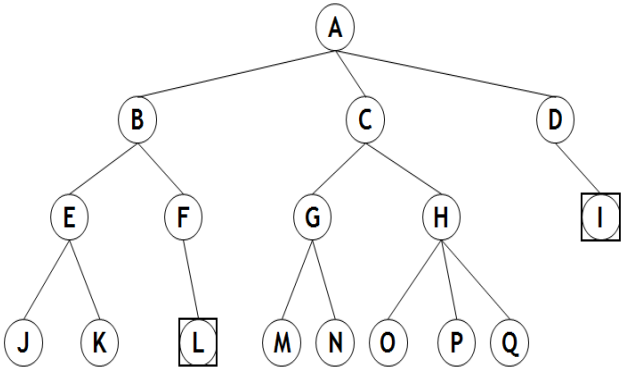

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



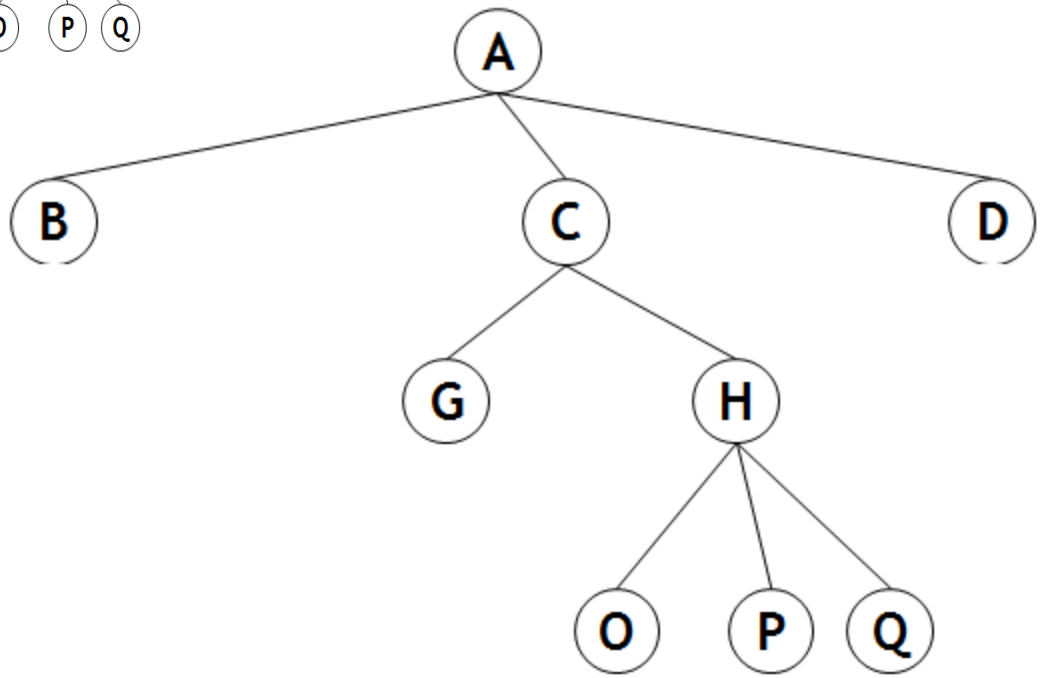


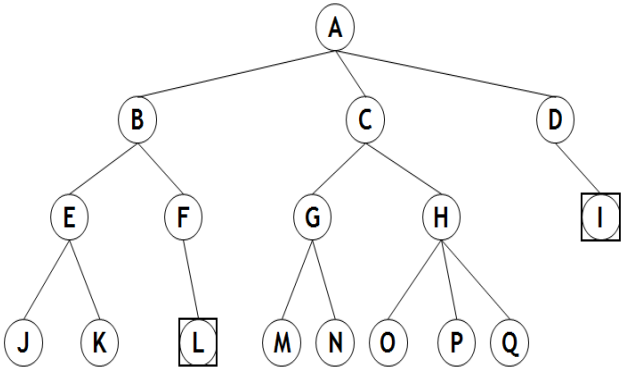

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



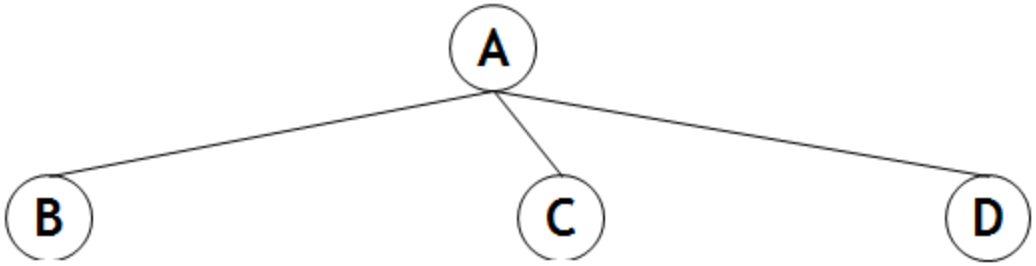


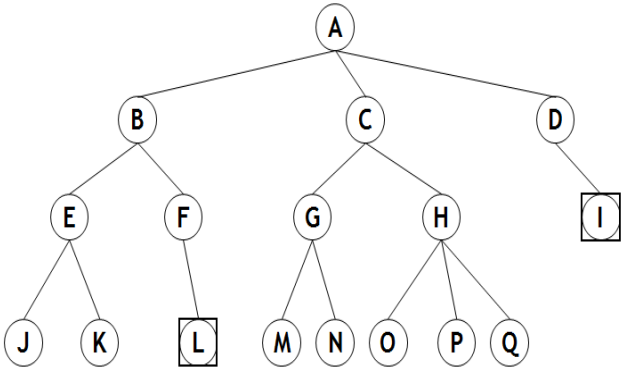

# جستجوی عمقی
# DEPTH-FIRST-SEARCH



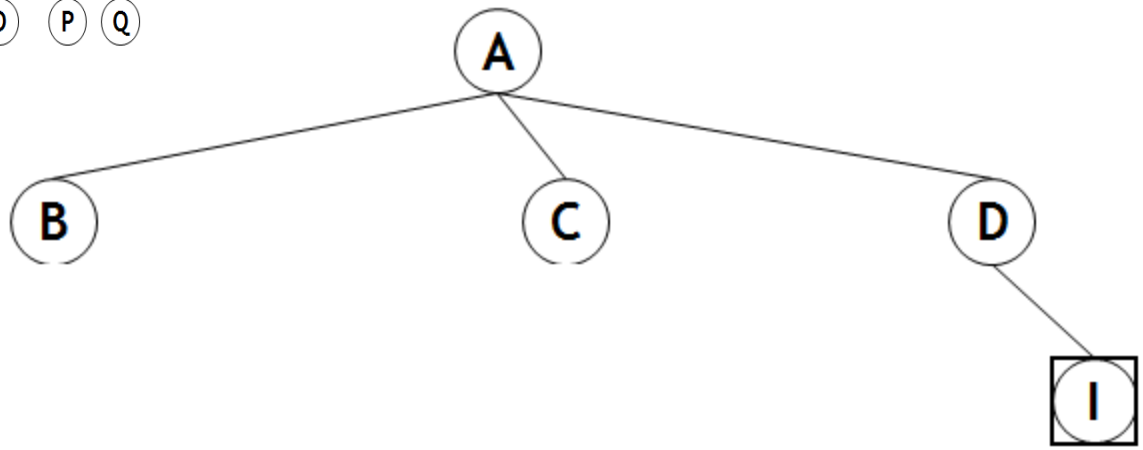



# جستجوی عمقی
# DEPTH-FIRST-SEARCH






# جستجوی عمقی
# DEPTH-FIRST-SEARCH



# جستجوی عمقی

**کامل بودن:** خیر

اگر زیر درخت چپ عمق نامحدود داشت و فاقد هر گونه راه حل باشد، جستجو هرگز خاتمه نمی یابد.

**بهینگی:** خیر

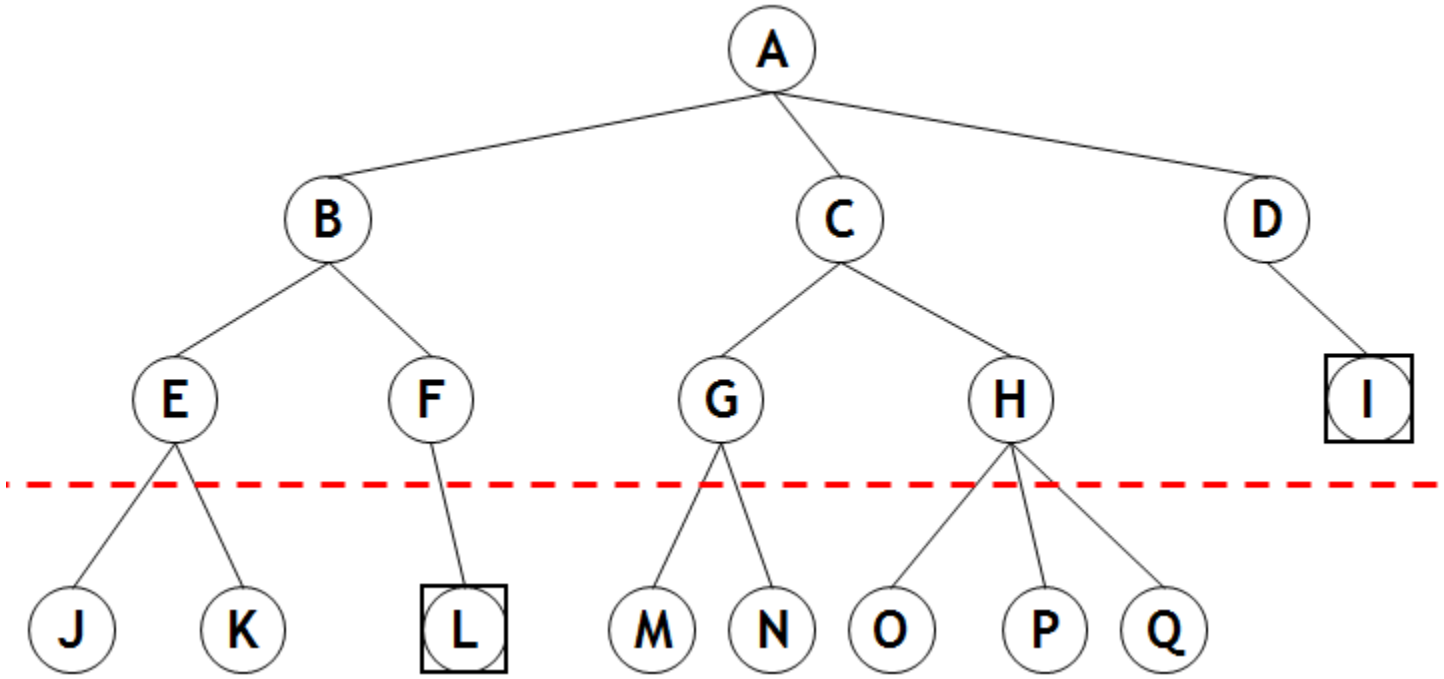
**پیچیدگی زمانی:** $O(b^m)$

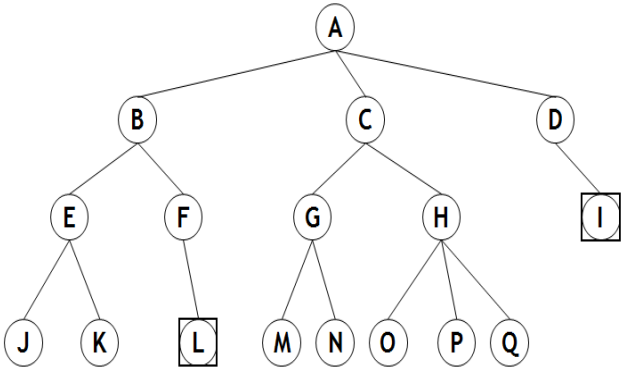
$O(b^m)$

**پیچیدگی فضا:** $O(bm)$

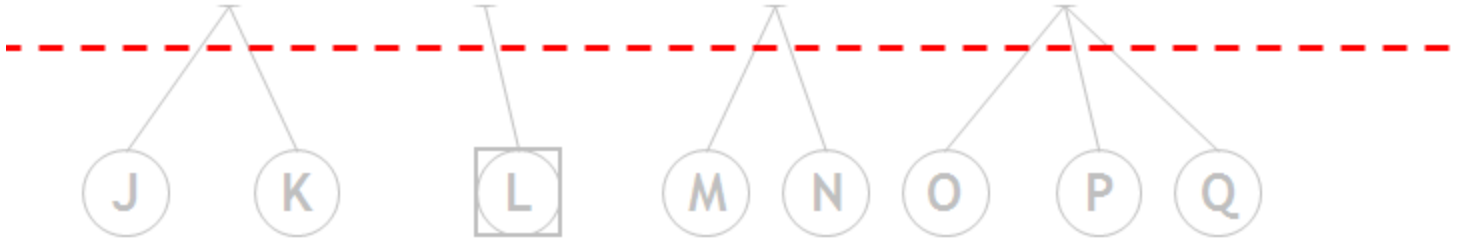
# جستجوی عمقی محدود

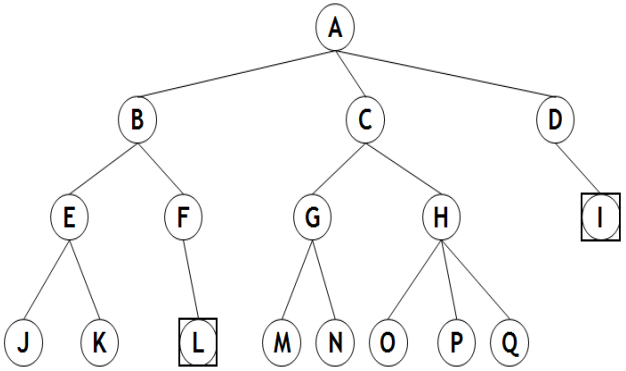
مسئله درختهای نامحدود میتواند به وسیله جست و جوی عمقی با عمق محدود L بهبود یابد



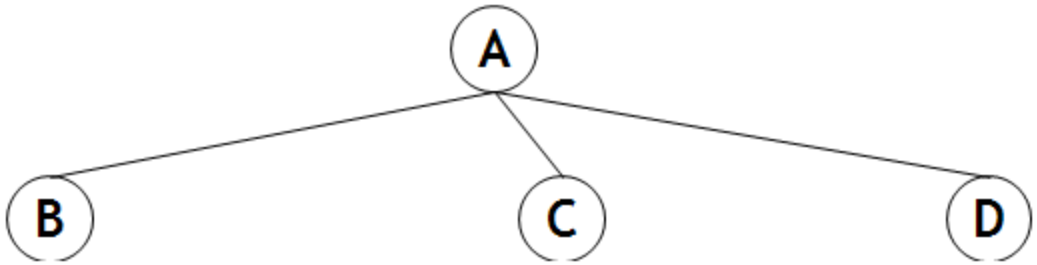


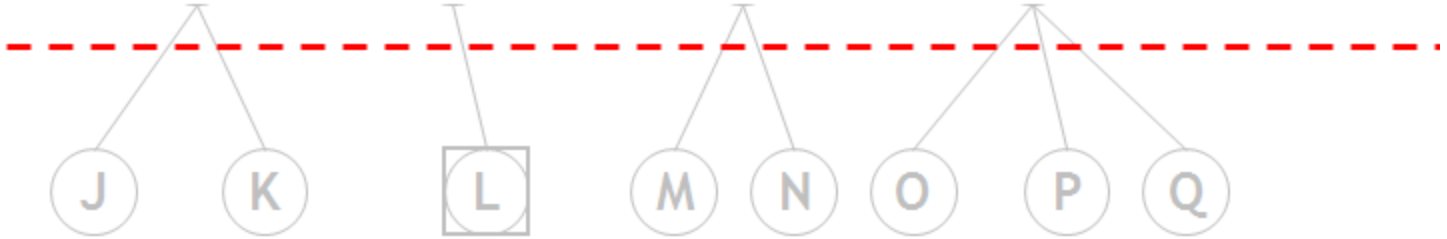

# جستجوی عمقی محدود

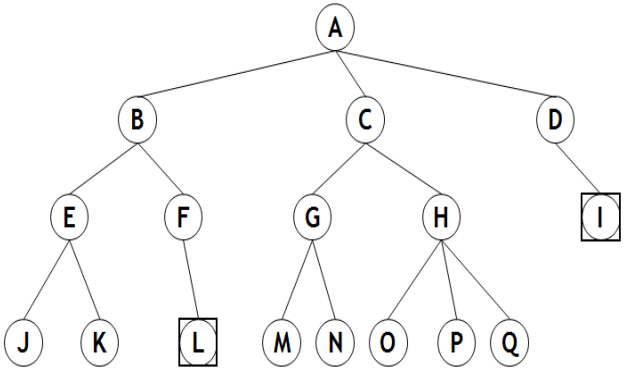





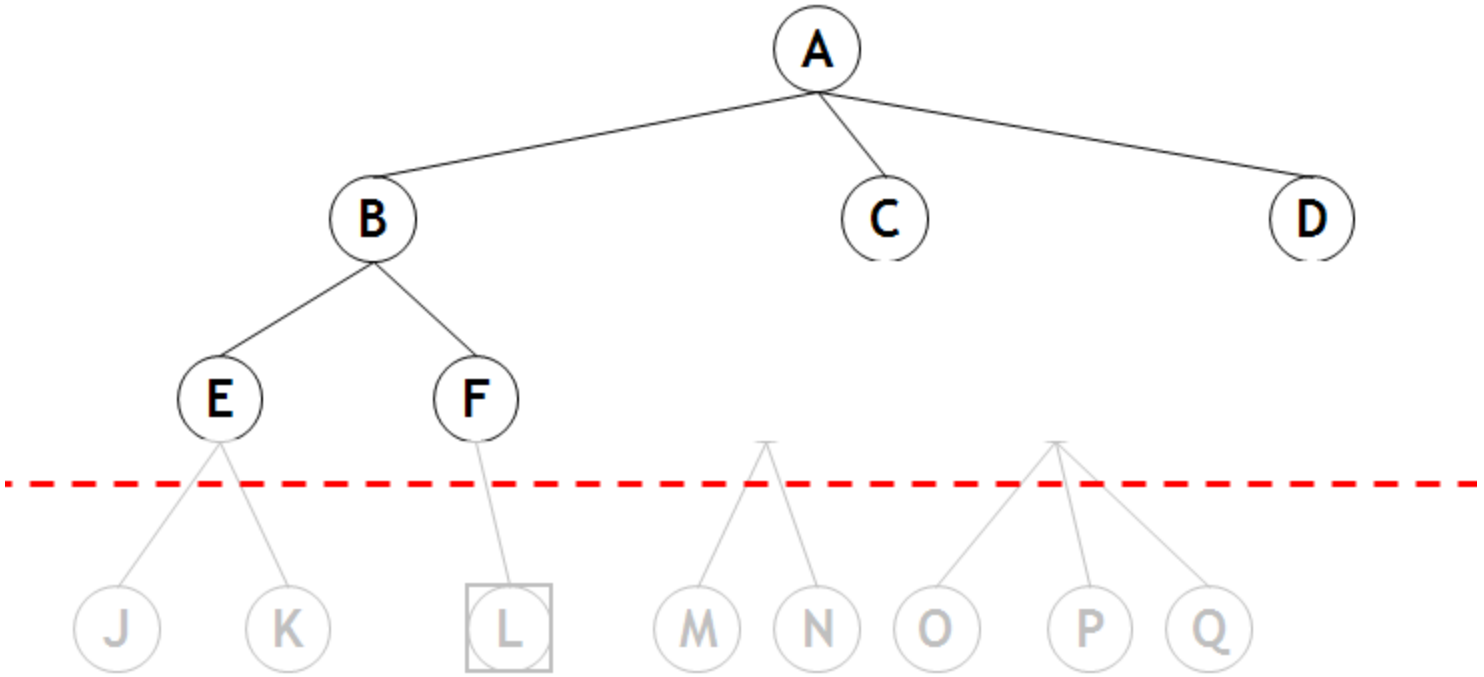

# جستجوی عمقی محدود

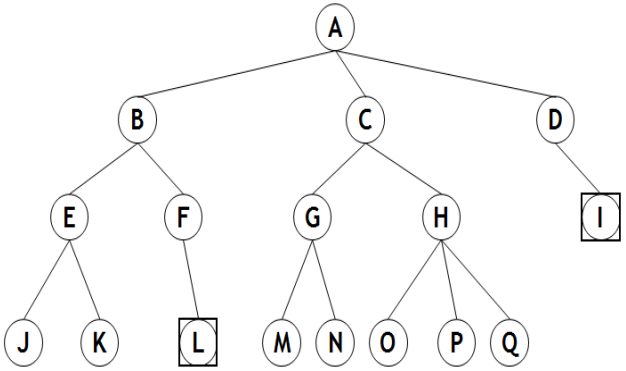


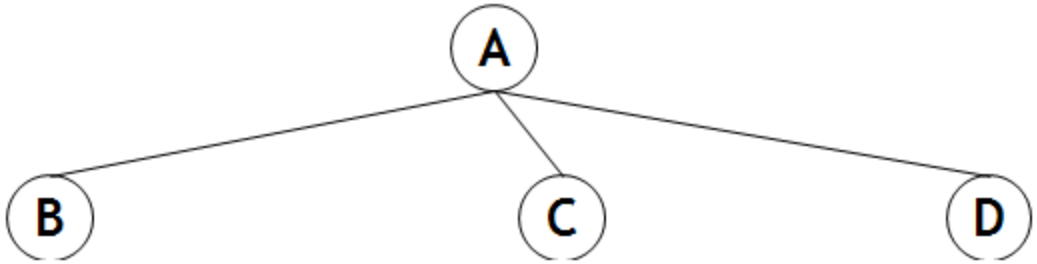


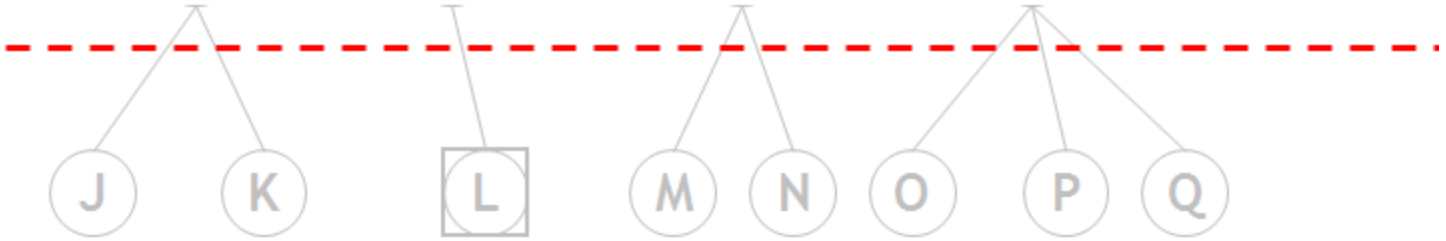

# جستجوی عمقی محدود

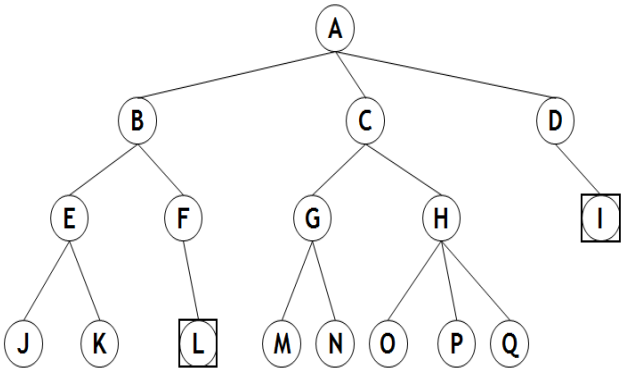




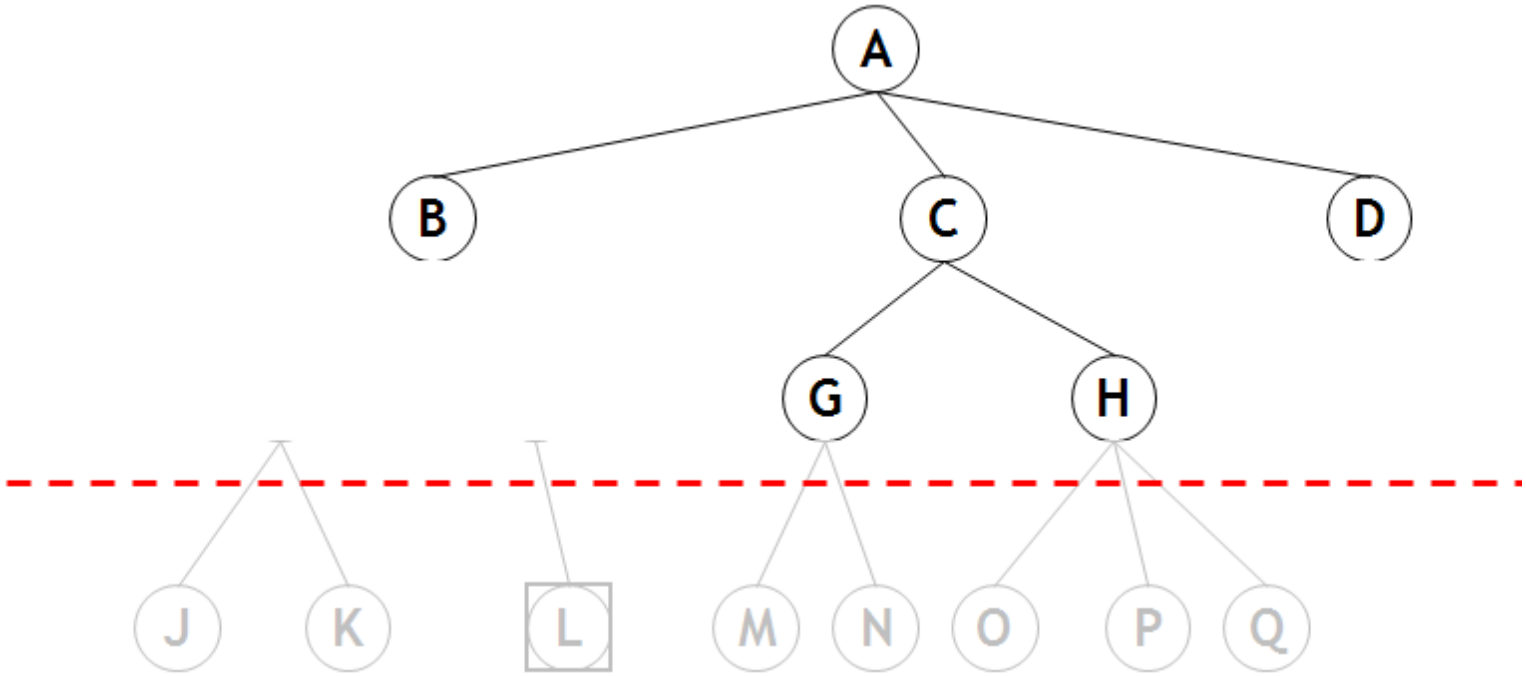

# جستجوی عمقی محدود

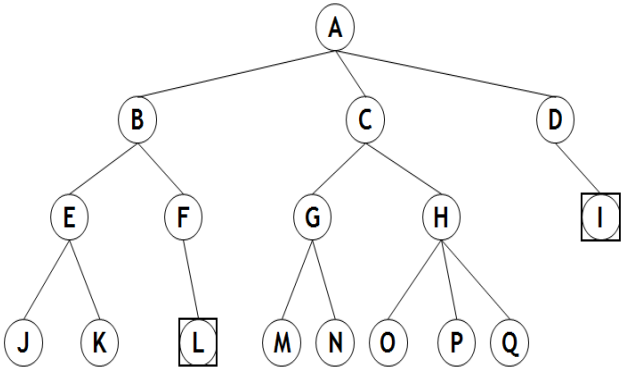


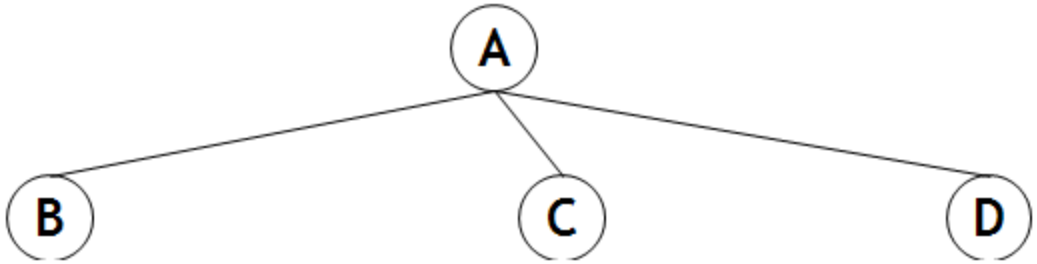


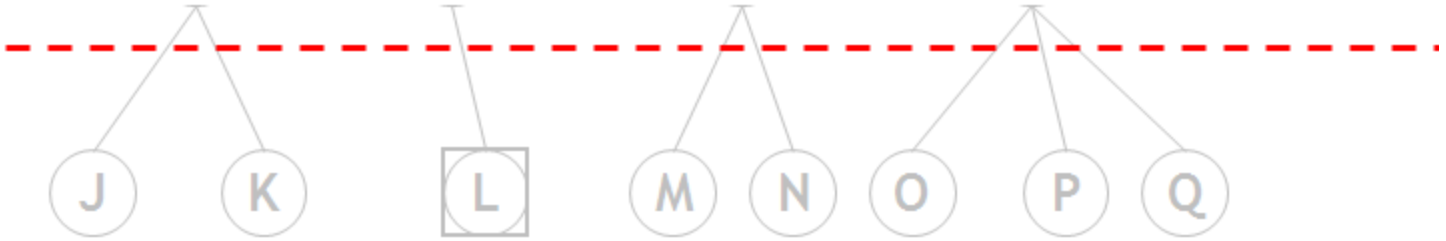

# جستجوی عمقی محدود

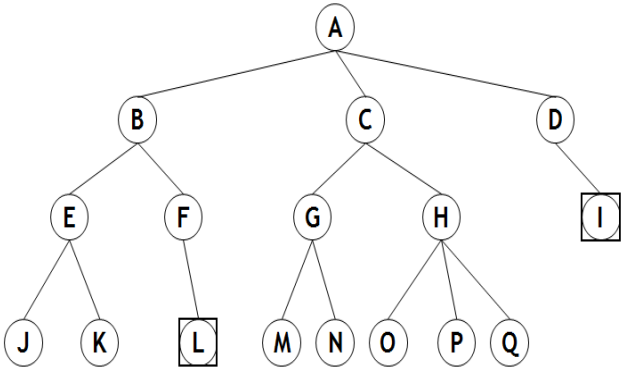




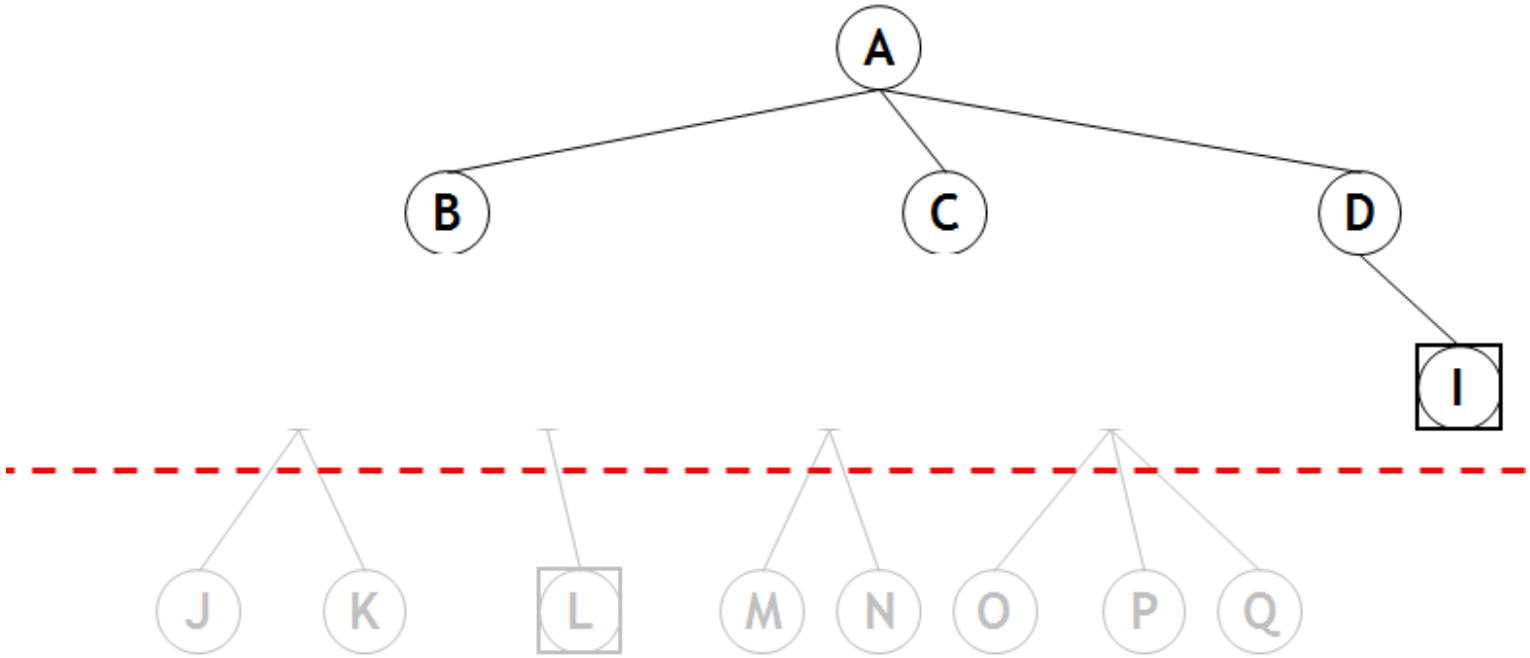

# جستجوی عمقی محدود






# جستجوی عمقی محدود



# جستجوی عمقی محدود

**function** DEPTH-LIMTTED-SEARCH (*problem, limit*) **returns** a solution, or failure/cutoff

**return** RECURSIVE-DLS (MAKE-NODE(INITIAL-STATE [*problem*]) , *problem, limit*)

**function** RECURSIVE-DLS (*node, problem, limit*) **returns** a solution, or failure/ cutoff

*cutoff- occurred* ← 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* ← RECURSIVE_DLS (*successor, problem, limit*)

**if** *result* = *cutoff* **then** *cutoff -occurred* true

**else if** *result* <> *failure* **then return** *result*

**if** *cutoff*-occurred **then return** *cutoff*

**else return** *failure*

# جستجوی عمقی محدود

## کامل بودن: خیر

اگر L<d و سطحی ترین هدف در خارج از عمق محدود قرار داشته باشد، این راهبرد کامل نخواهد بود.

## بهینگی: خیر

اگر L>d انتخاب شود، این راهبرد بهینه نخواهد بود.

## پیچیدگی زمانی: $O(b^L)$

## پیچیدگی فضا: $O(bL)$

# جستجوی عمیق شونده تکراری

**function** ITERATIVE-DEEPENING-SEARCH (*problem*) **returns** a solution, or failure

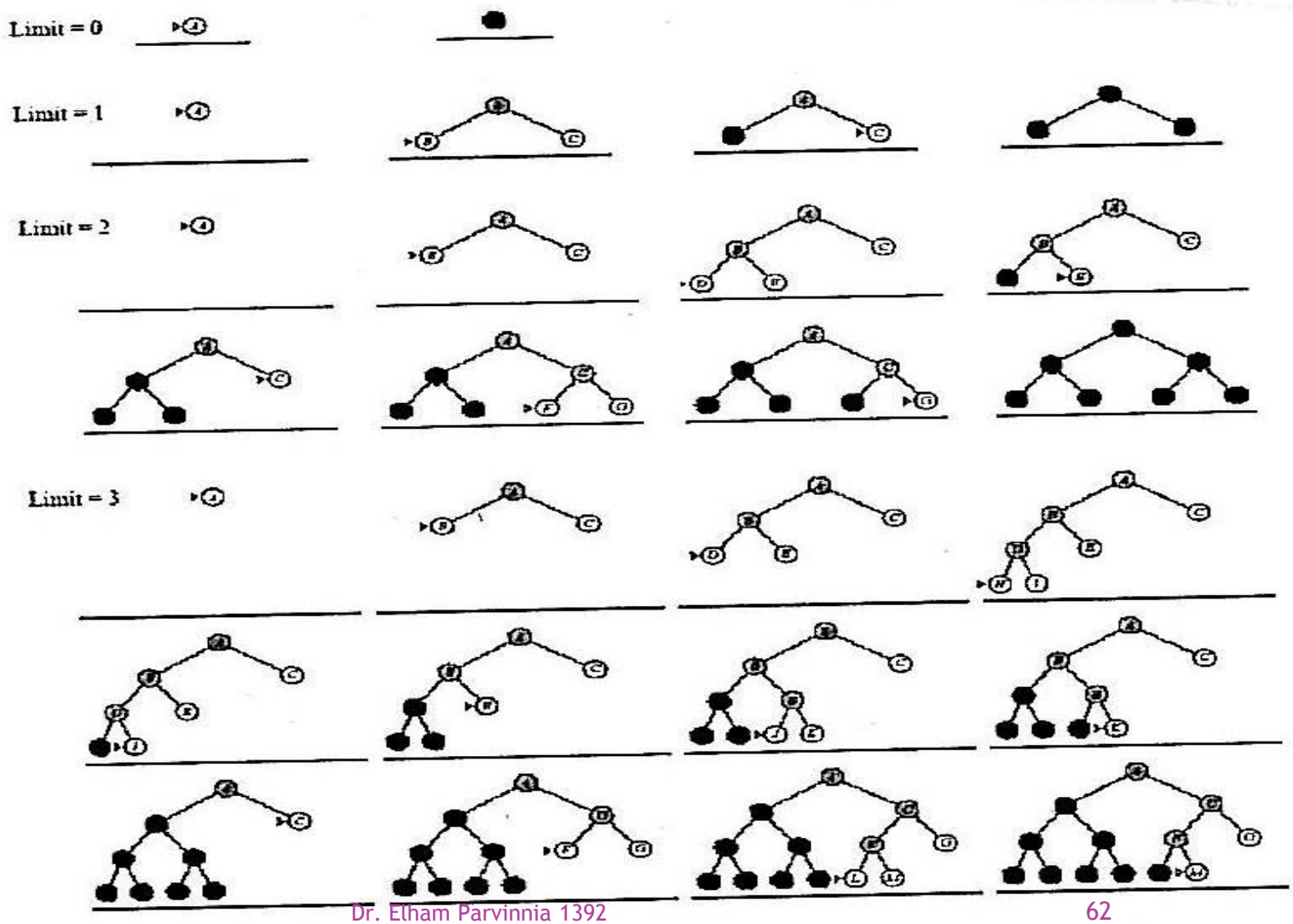
**input:** *problem*, a problem

**for** *dept* ← *0* **to** ∞

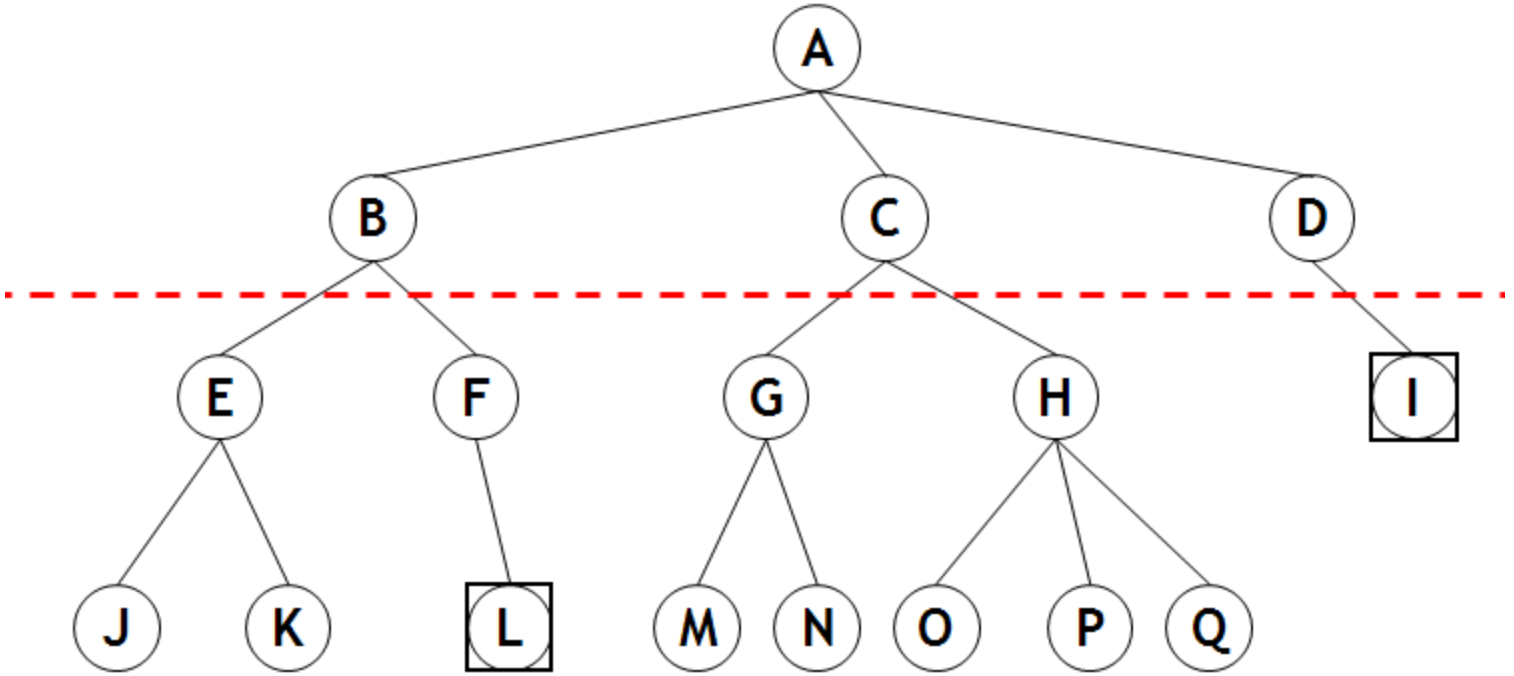
*result* ← DEPTH-LIMITED-SEARCH (*problem*. *depth*)

if *result* <> Cutoff **then return** *result*

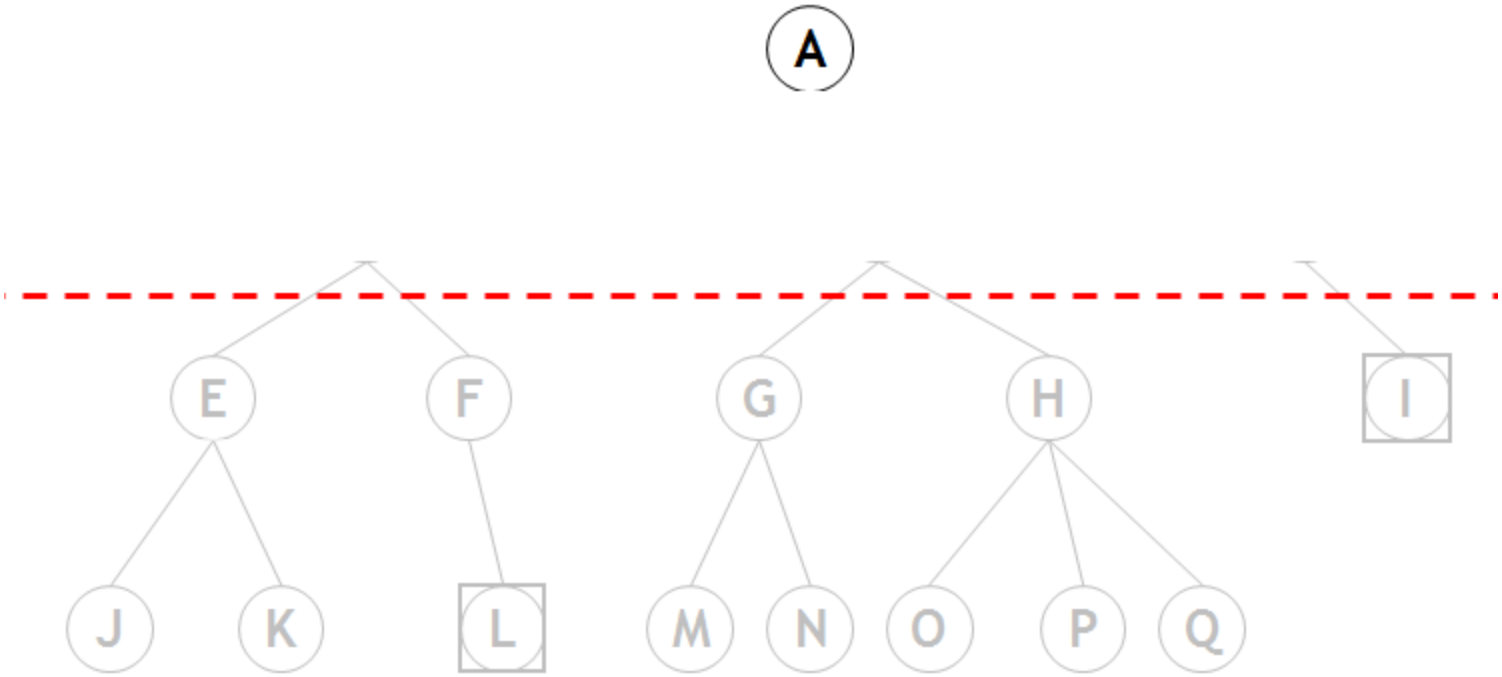
# جستجوی عمیق شونده تکراری



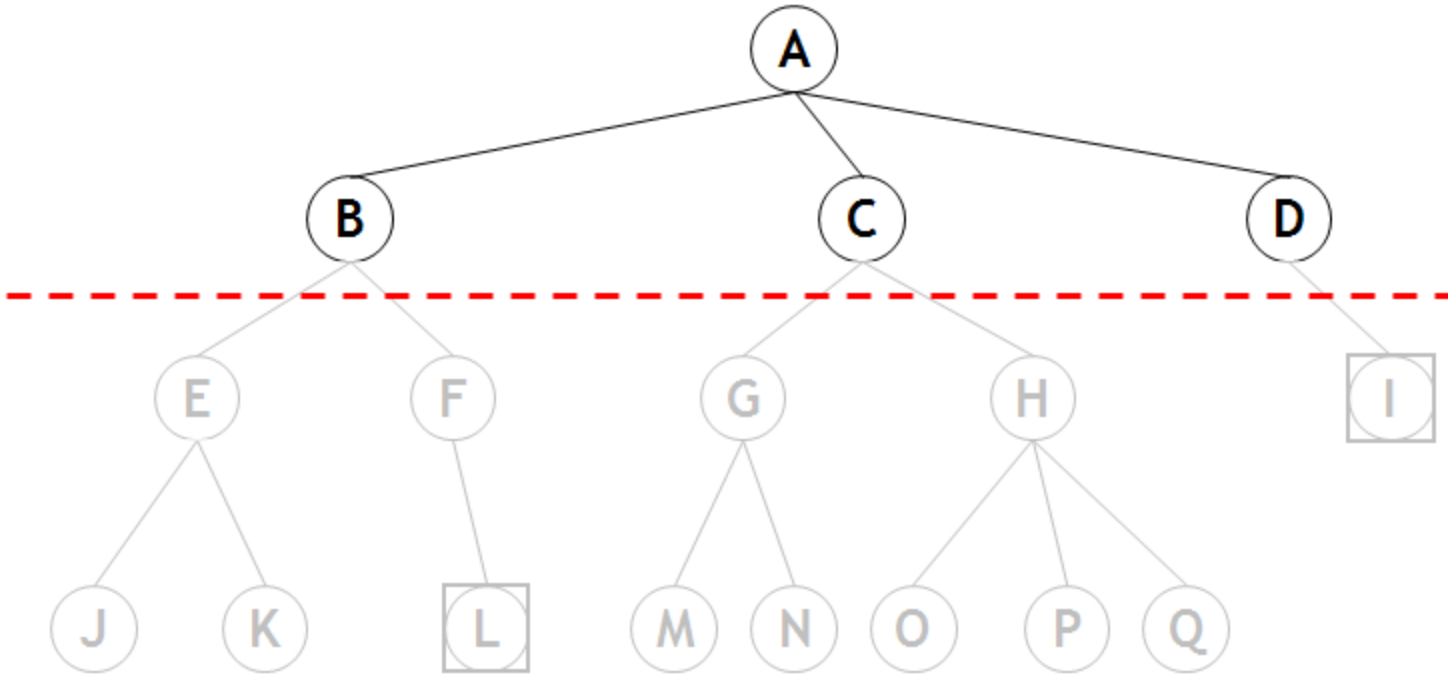
# جستجوی عمیق شونده تکراری



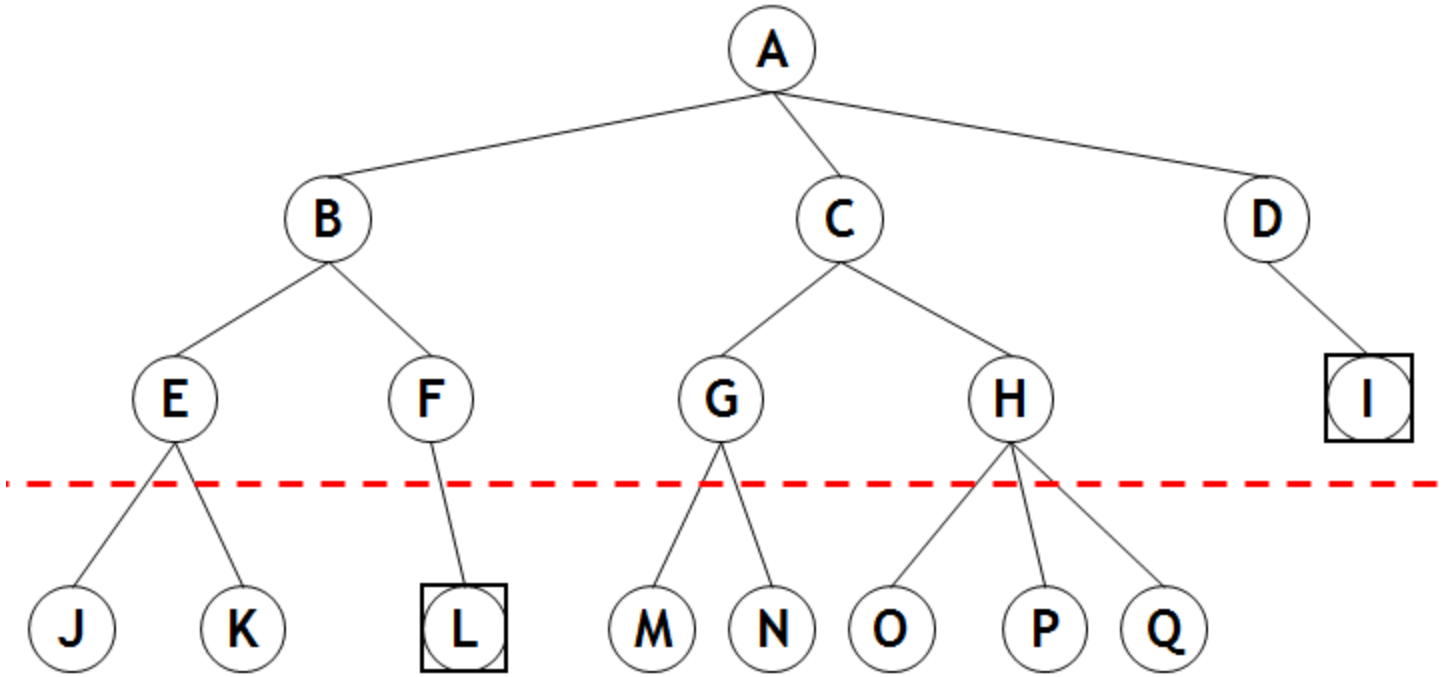
# جستجوی عمیق شونده تکراری



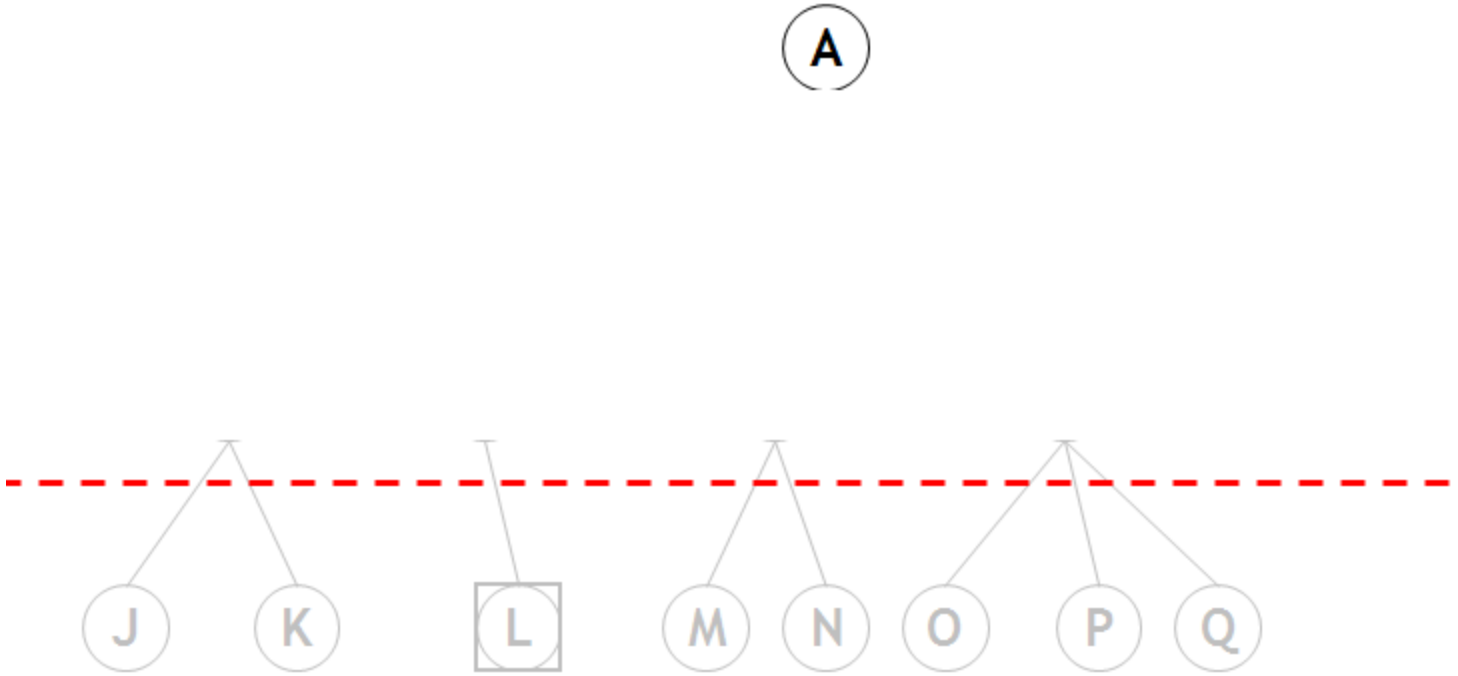
# جستجوی عمیق شونده تکراری



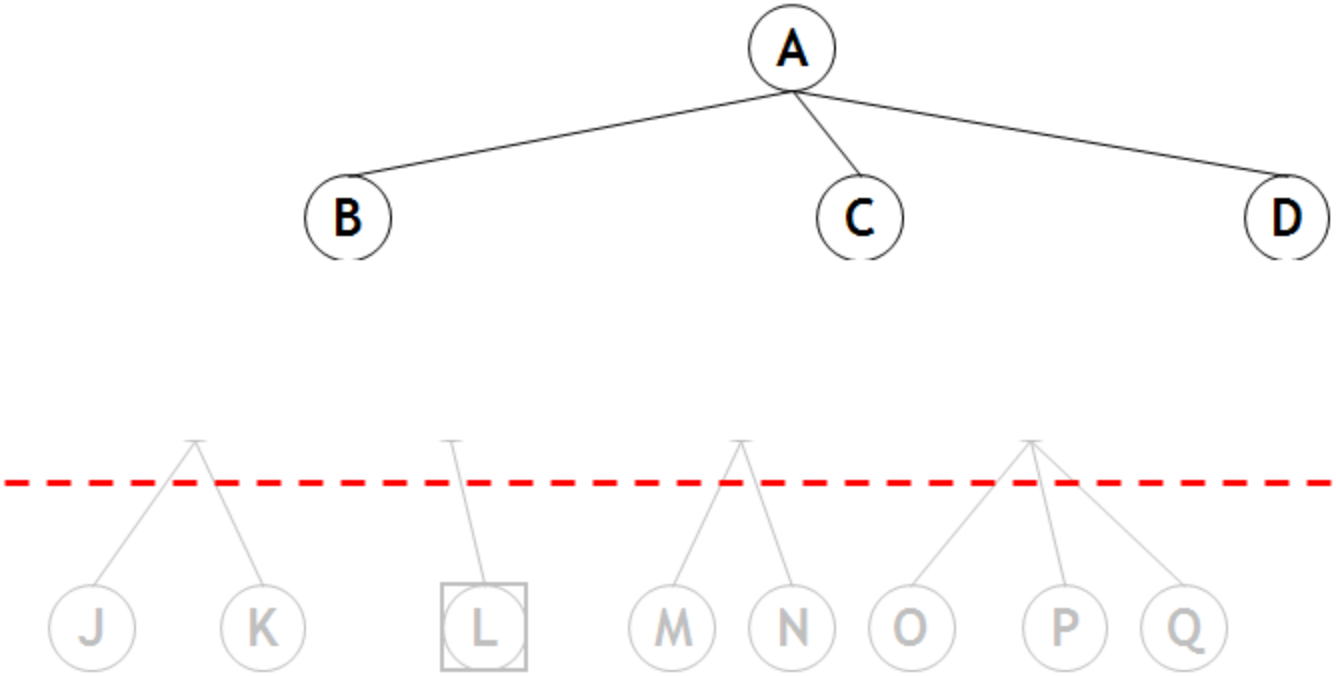
# جستجوی عمیق شونده تکراری



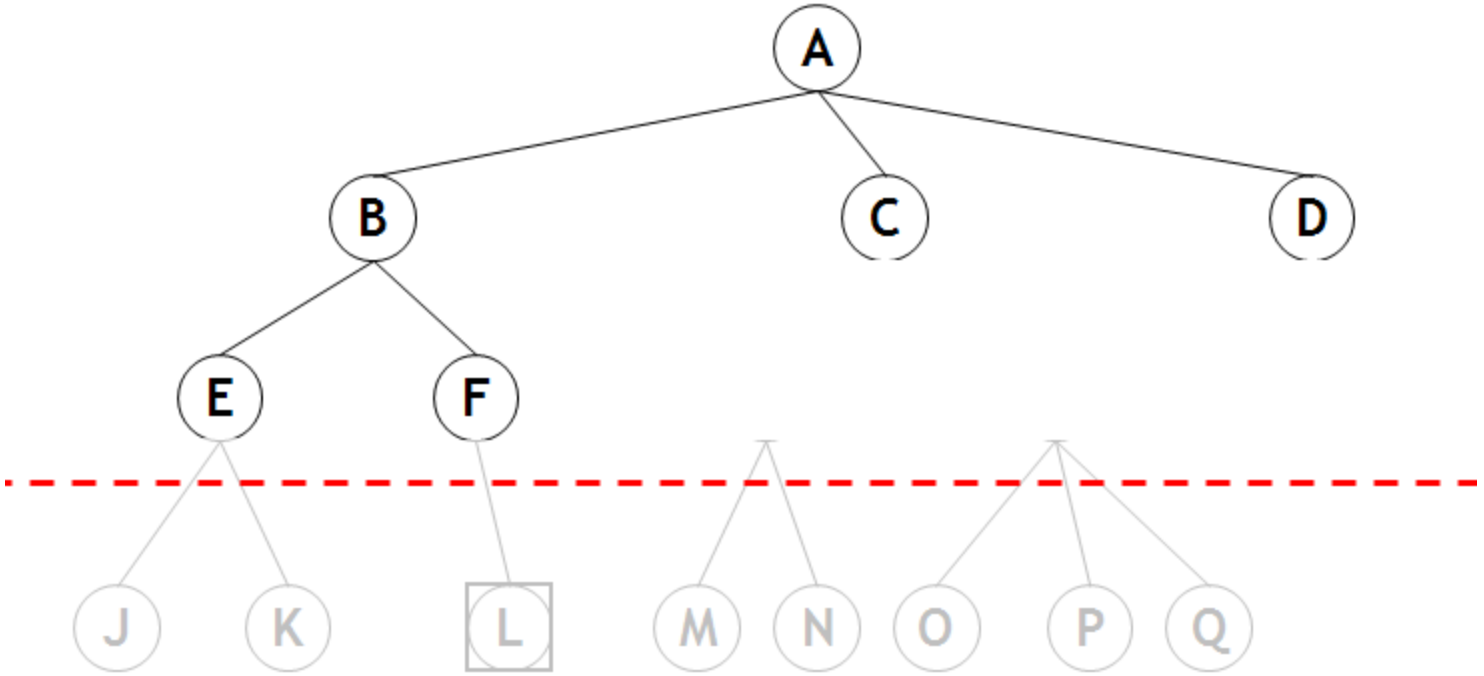
# جستجوی عمیق شونده تکراری



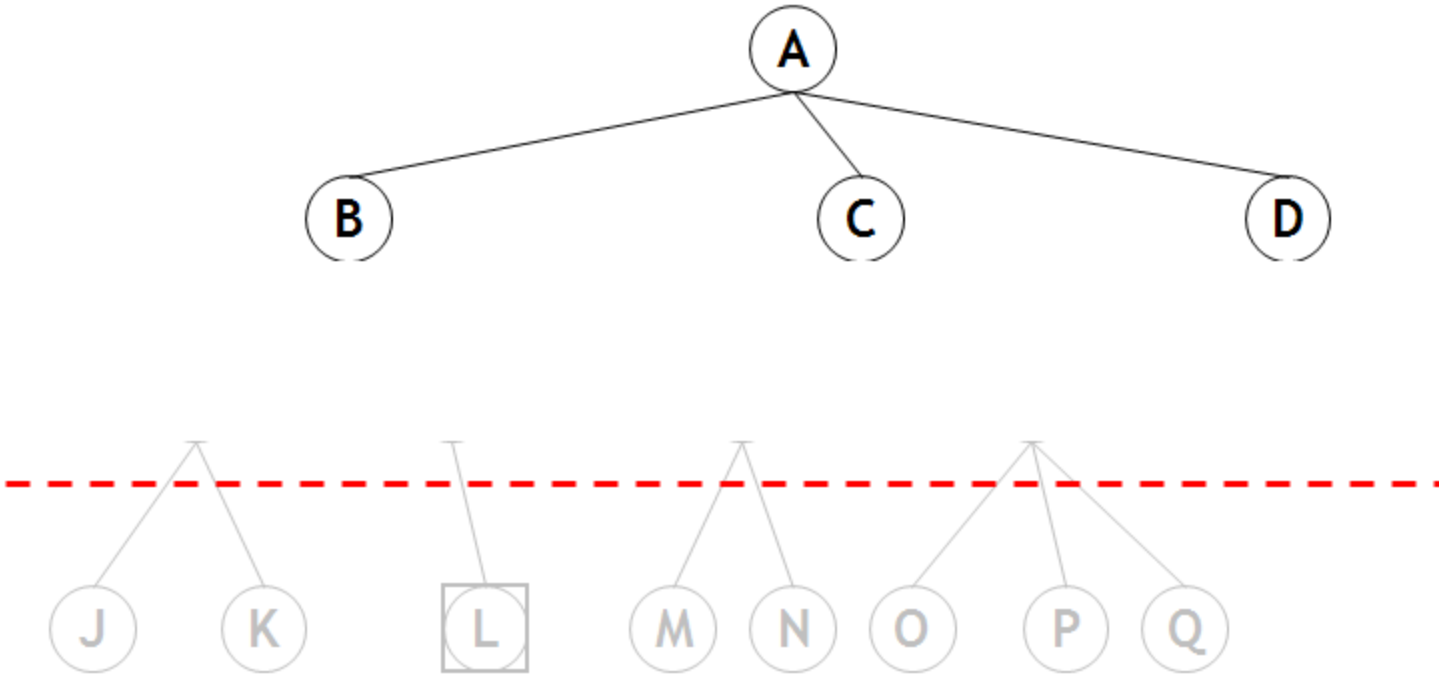
# جستجوی عمیق شونده تکراری



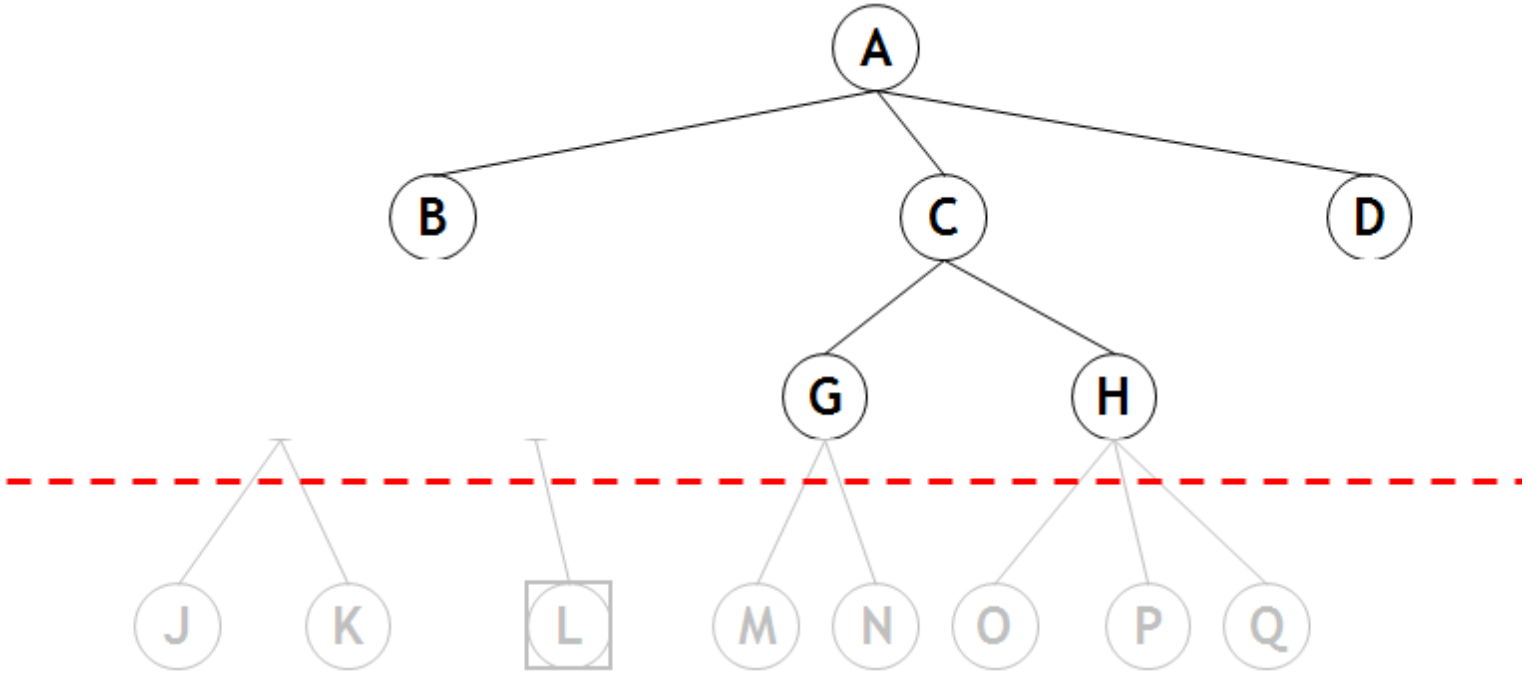
# جستجوی عمیق شونده تکراری



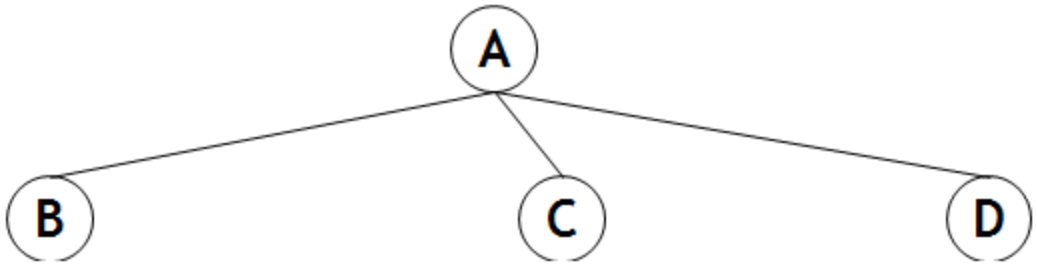
# جستجوی عمیق شونده تکراری

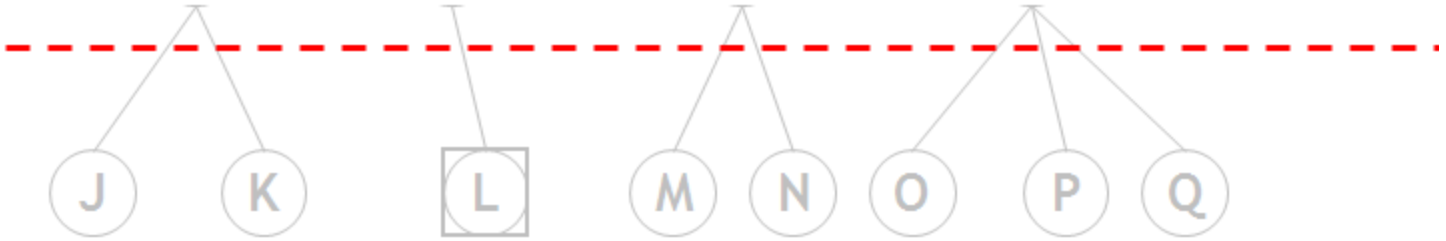


# جستجوی عمیق شونده تکراری

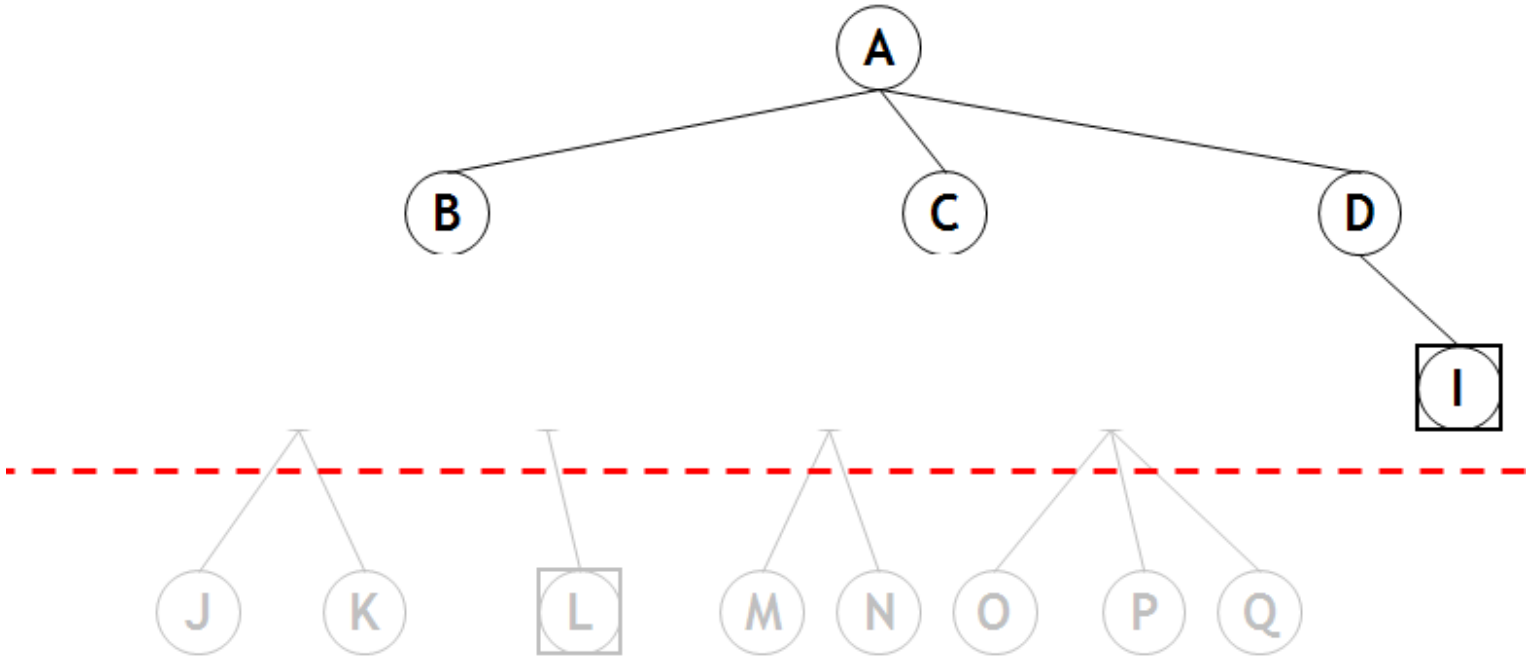


# جستجوی عمیق شونده تکراری

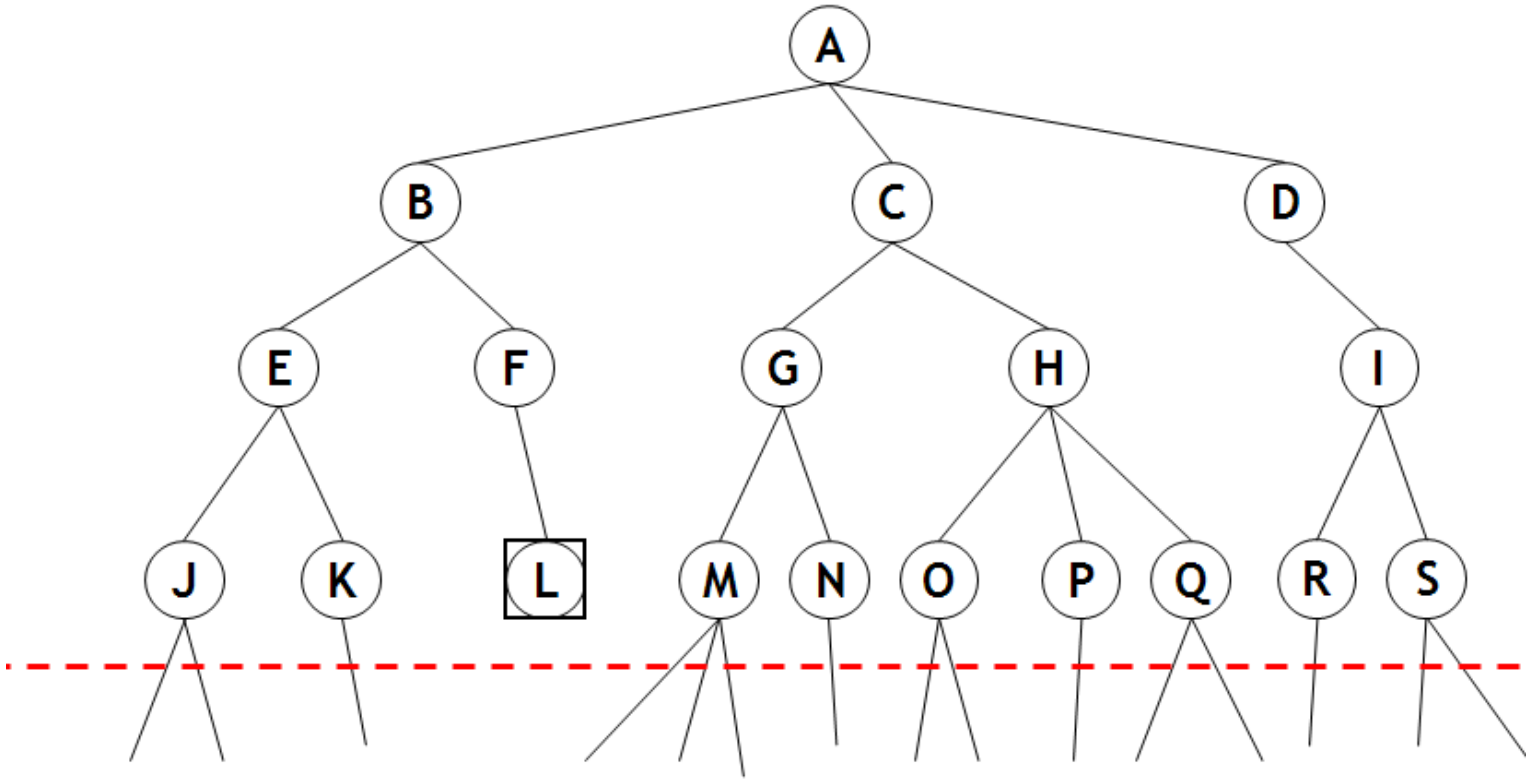


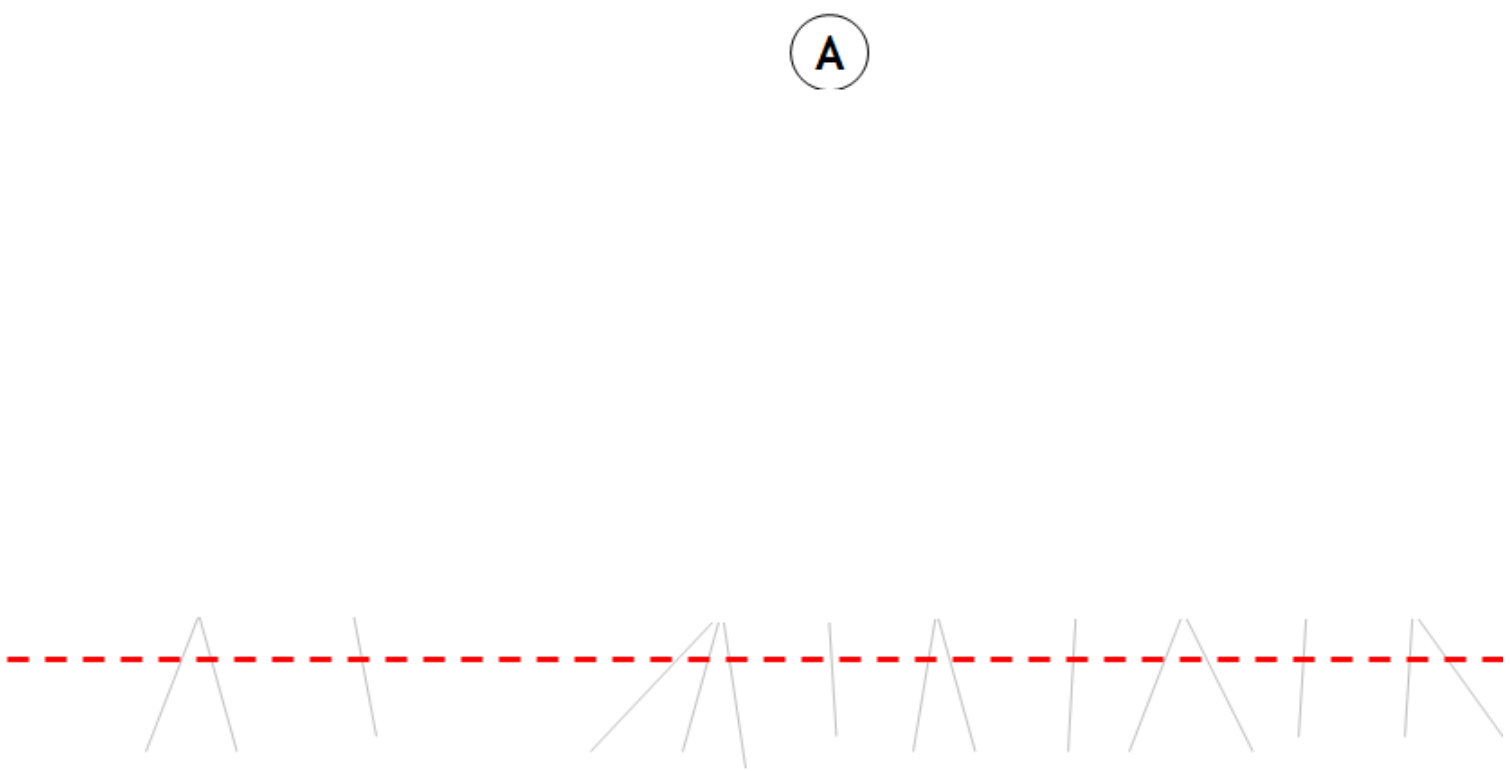
# جستجوی عمیق شونده تکراری



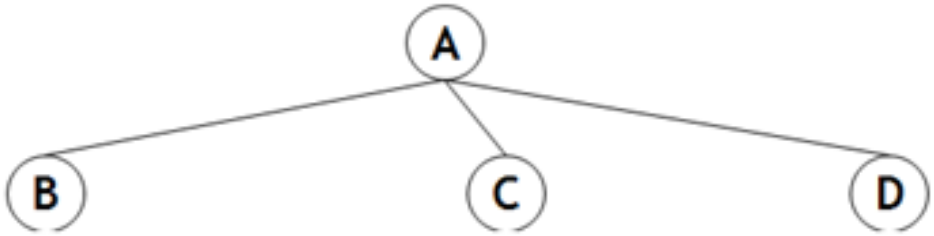
# جستجوی عمیق شونده تکراری(مثال دوم)



# جستجوی عمیق شونده تکراری

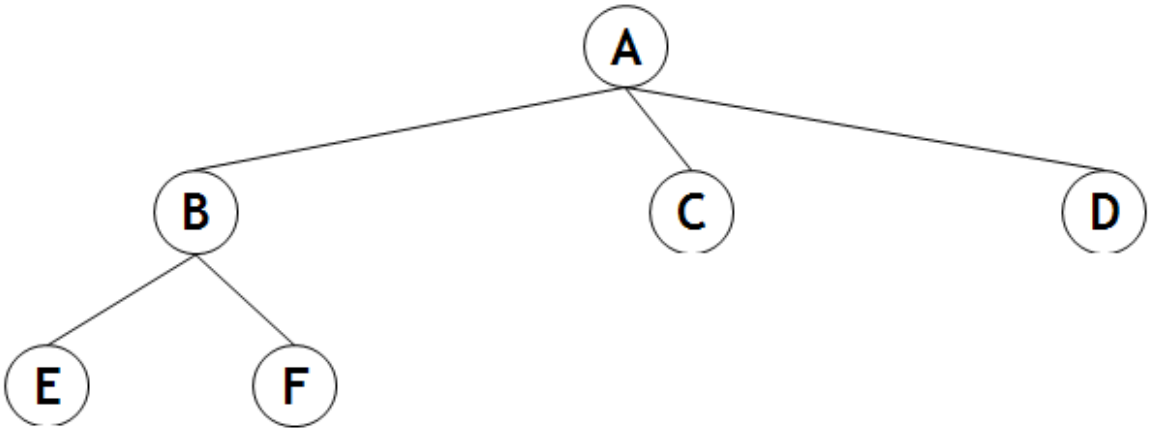


# جستجوی عمیق شونده تکراری

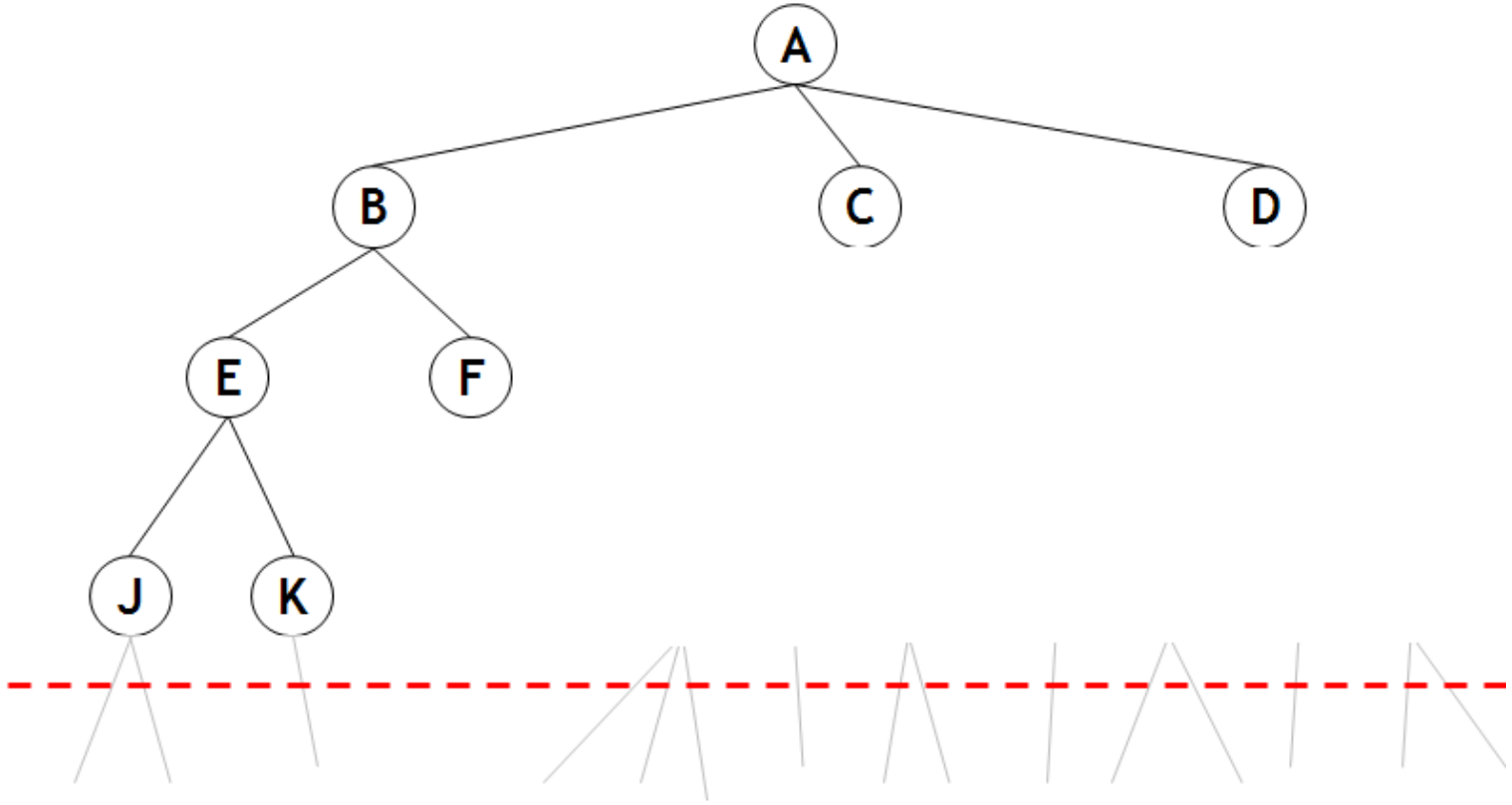



77

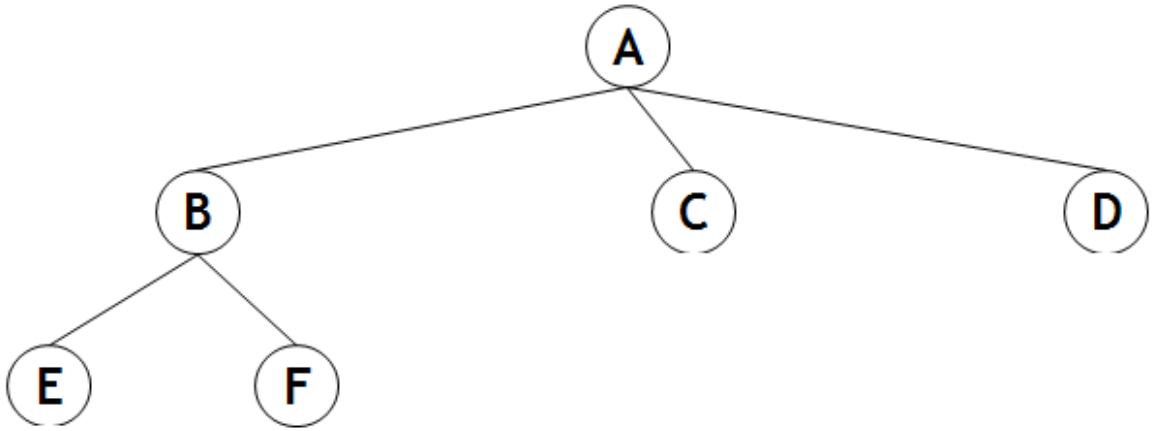
# جستجوی عمیق شونده تکراری



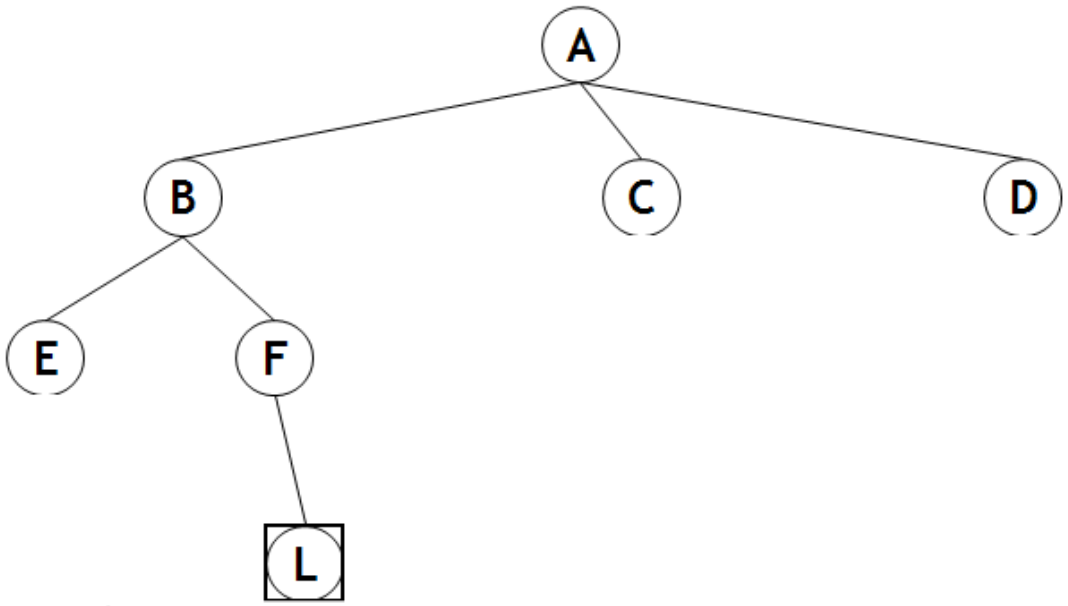
# جستجوی عمیق شونده تکراری



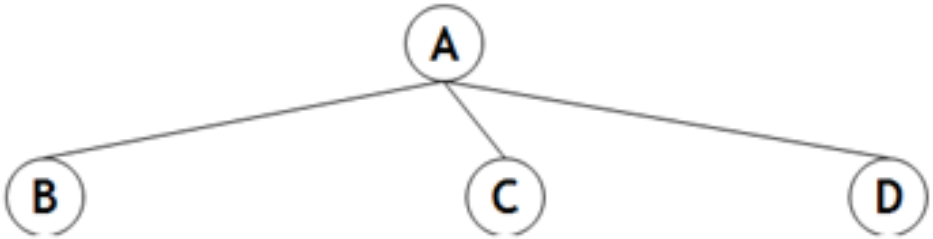
# جستجوی عمیق شونده تکراری



# جستجوی عمیق شونده تکراری

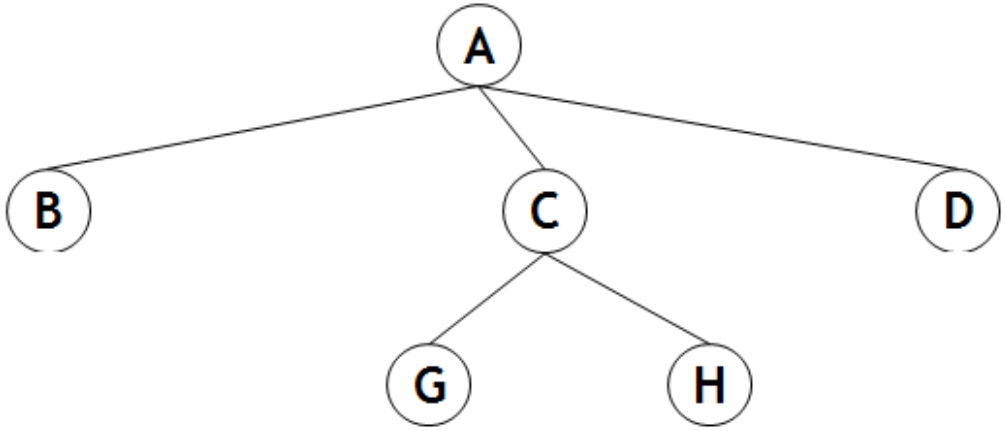


# جستجوی عمیق شونده تکراری

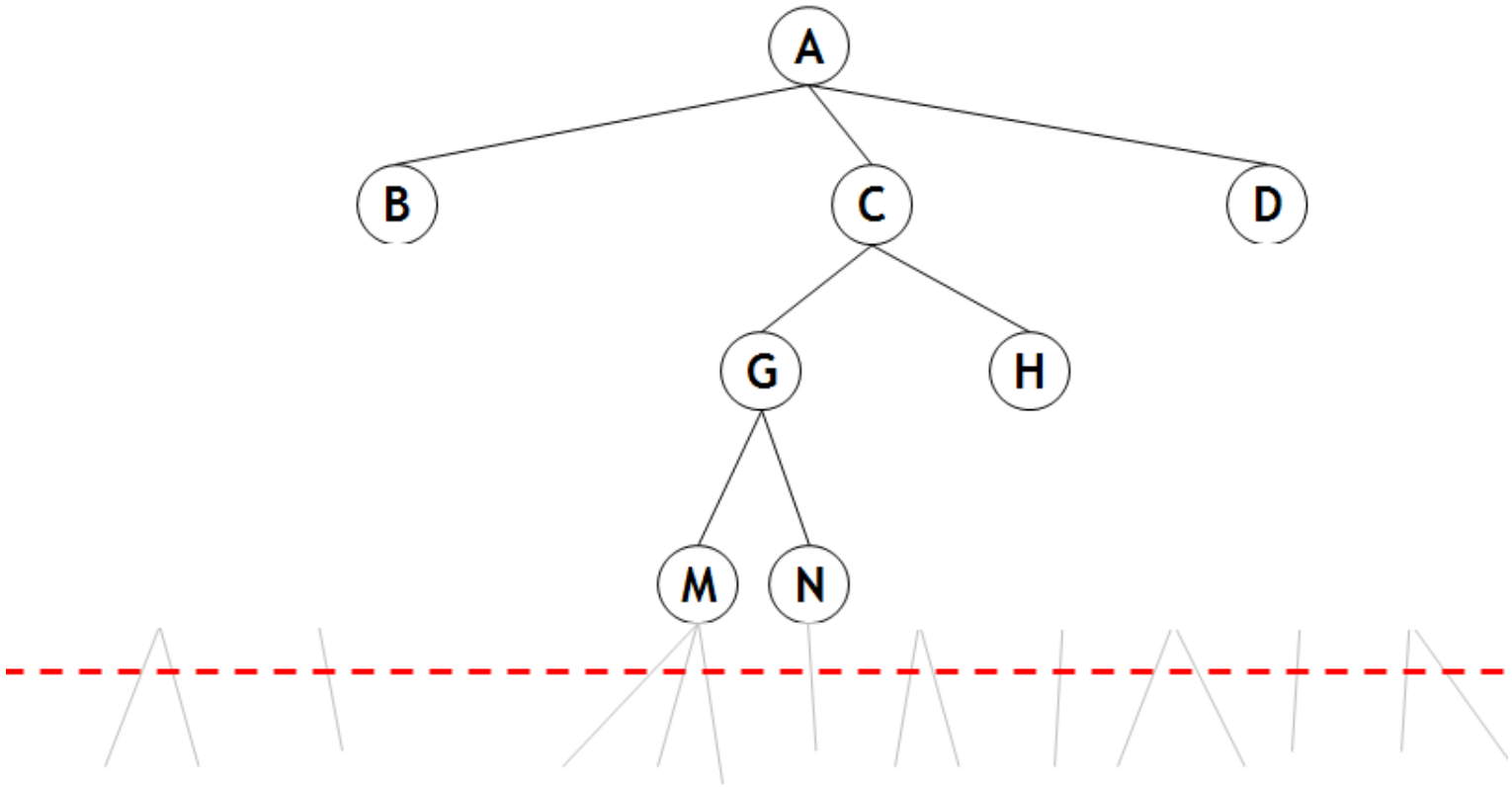



77

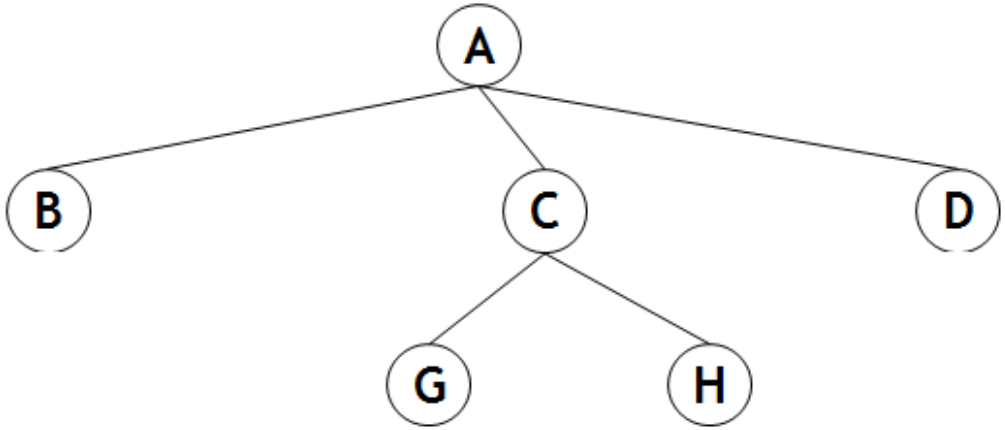
# جستجوی عمیق شونده تکراری

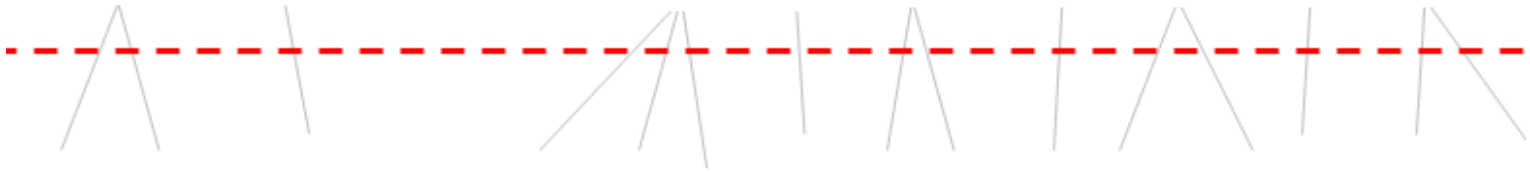


# جستجوی عمیق شونده تکراری



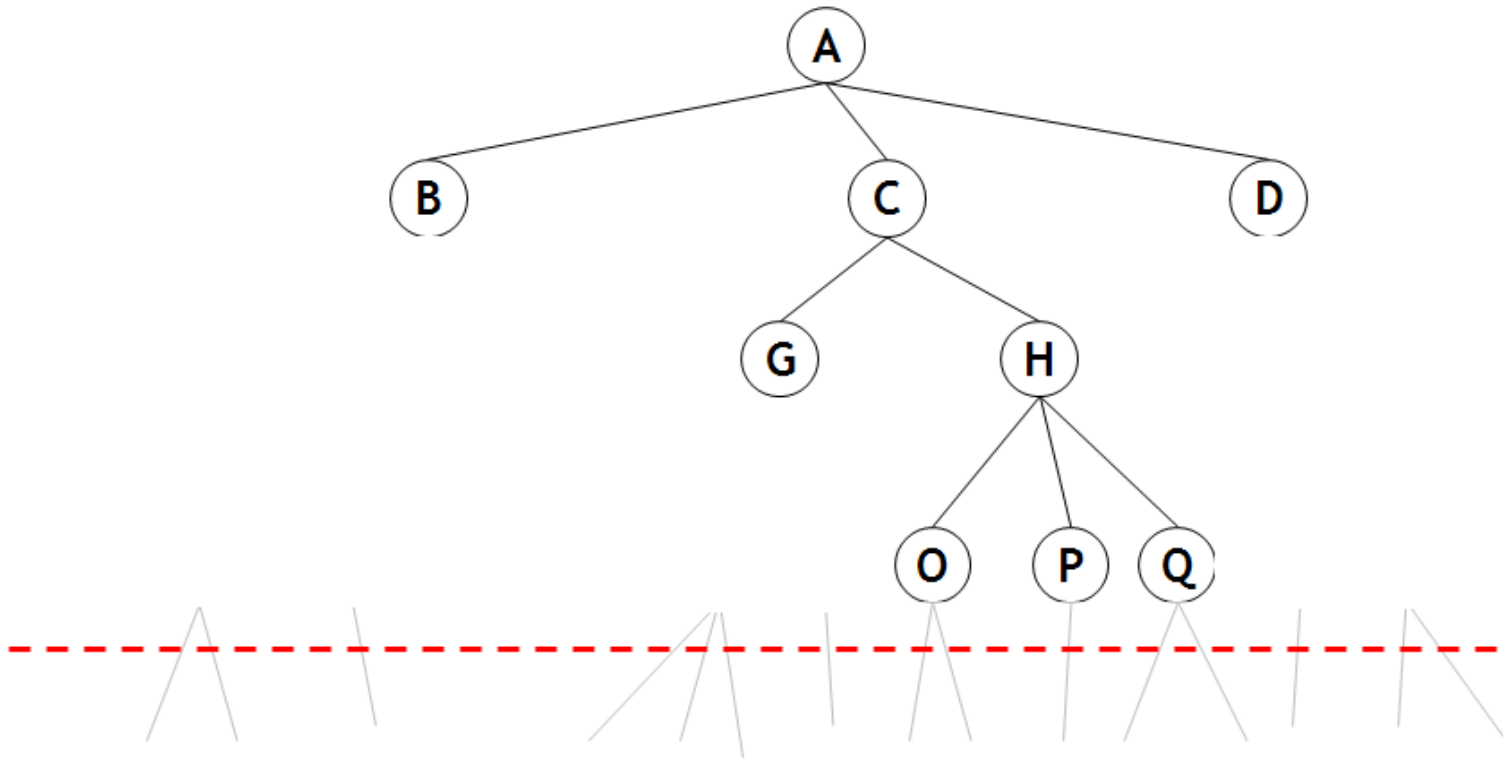
# جستجوی عمیق شونده تکراری



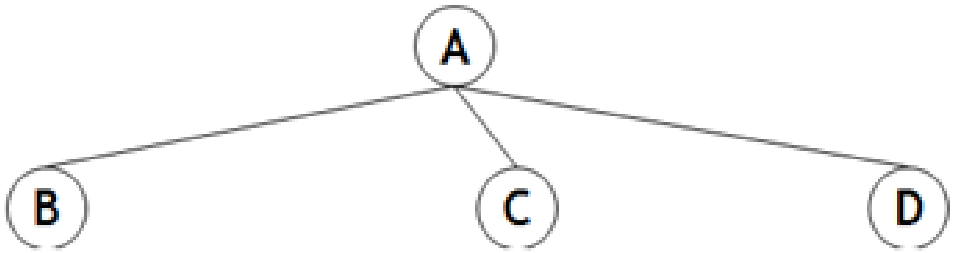



82

# جستجوی عمیق شونده تکراری

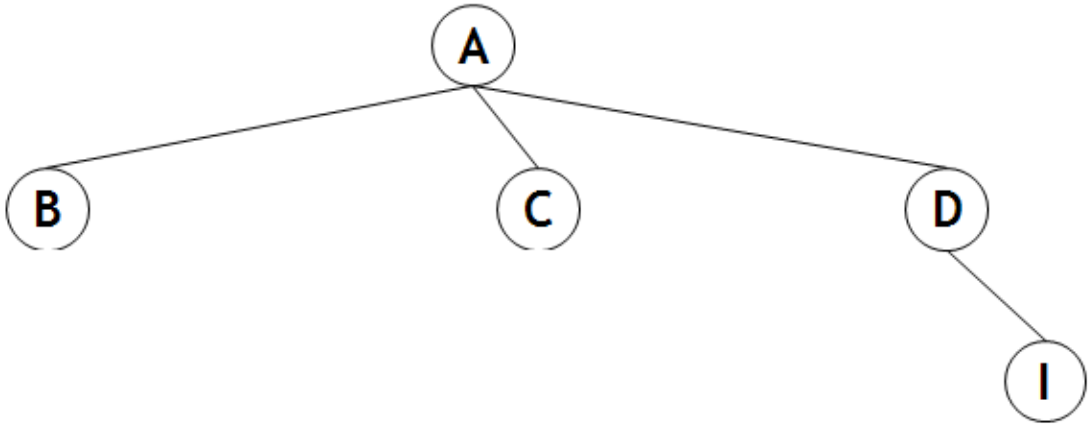


# جستجوی عمیق شونده تکراری

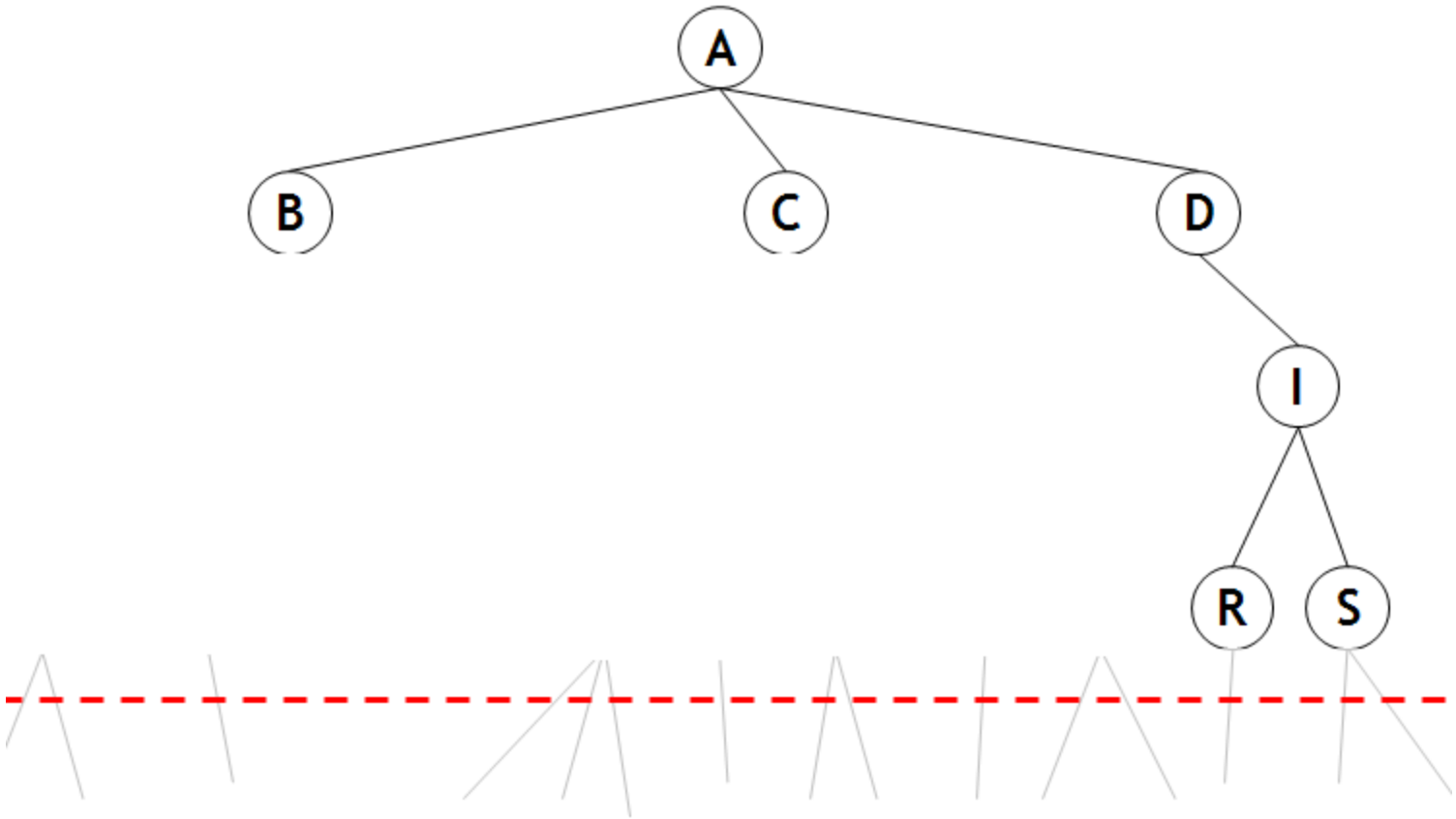



77

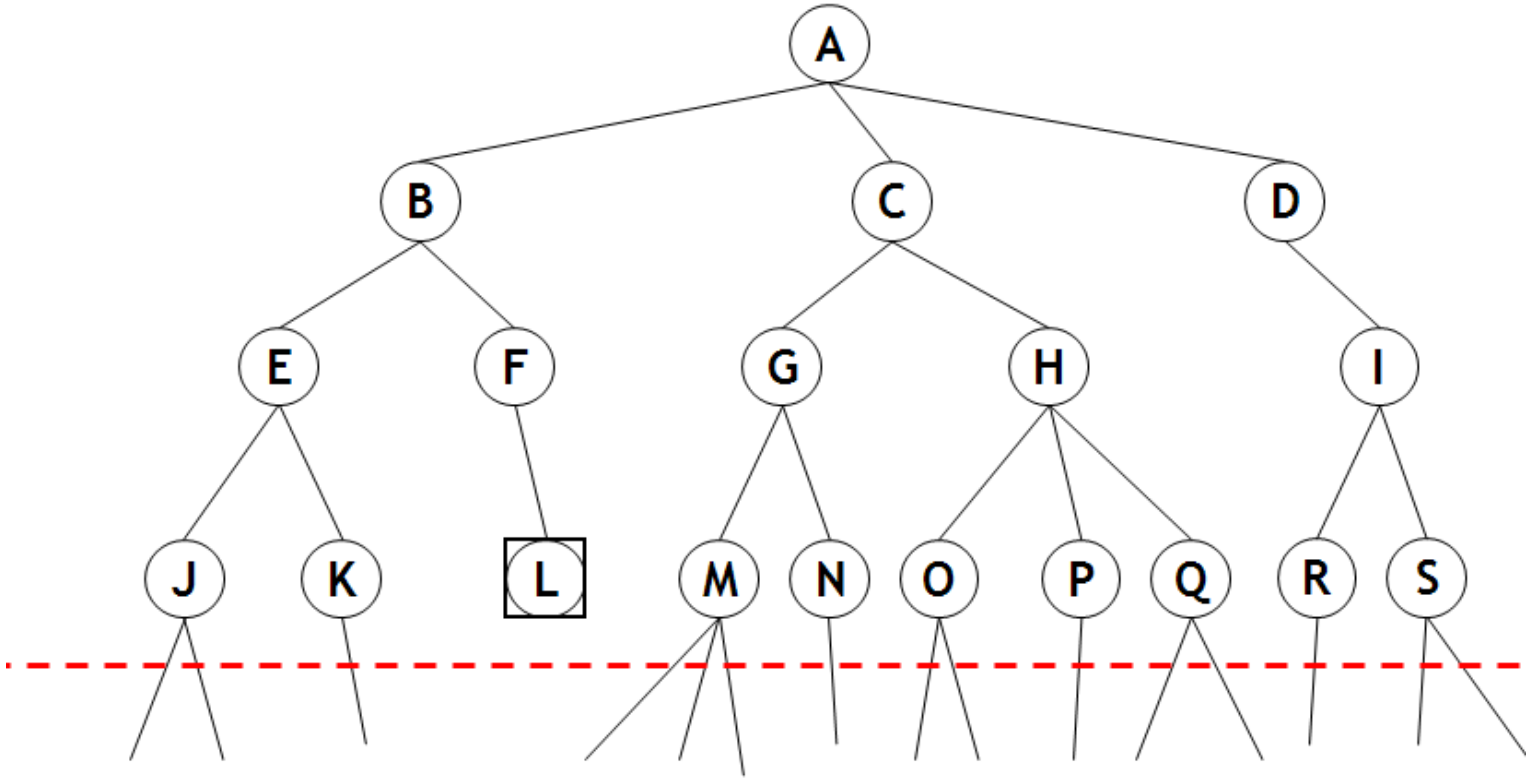
# جستجوی عمیق شونده تکراری



# جستجوی عمیق شونده تکراری



# جستجوی عمیق شونده تکراری




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

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

# جستجوی عمیق شونده تکراری

**کامل بودن:** بله

در صورتی که فاکتور انشعاب محدود باشد

**بهینگی:** بله

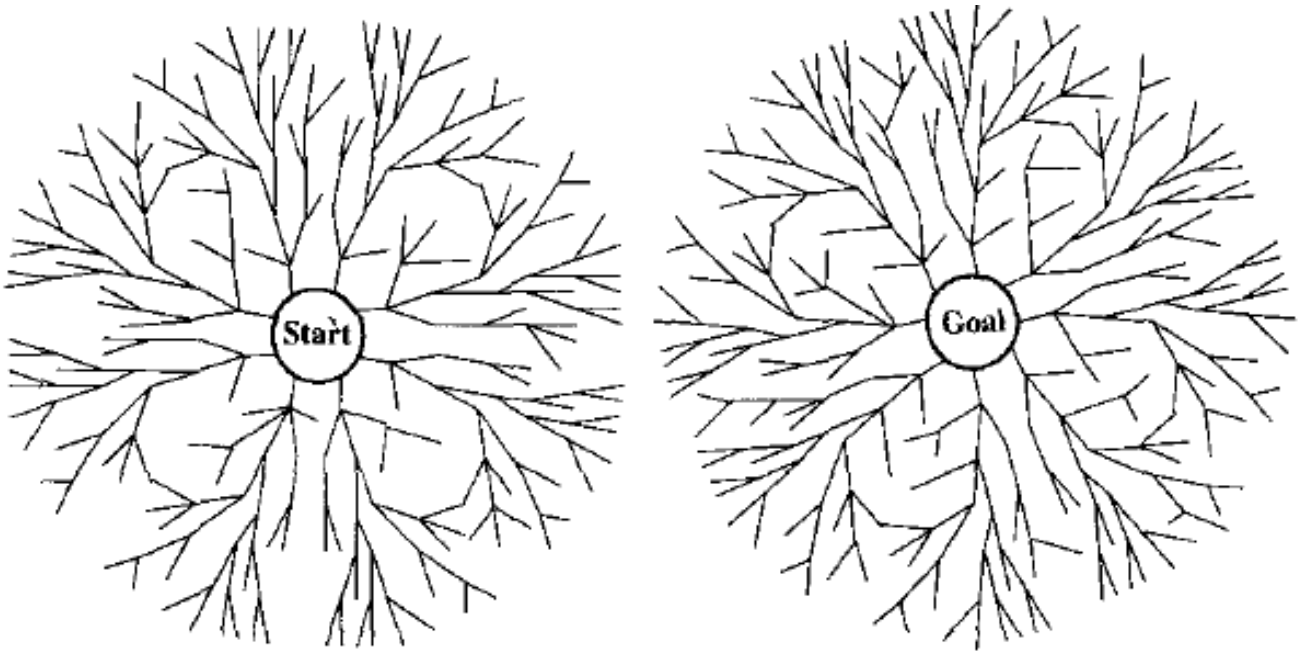
وقتی که هزینه مسیر، تابعی غیر نزولی از عمق گره باشد

**پیچیدگی زمانی:** $O(b^d)$

**پیچیدگی فضا:** $O(bd)$

# جستجوی دو طرفه

انجام دو جست و جوی همزمان، یکی از حالت اولیه به هدف و دیگری از هدف به حالت اولیه تا زمانی که دو جست و جو به هم برسند



# جستجوی دو طرفه

**کامل بودن:** بله

اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد

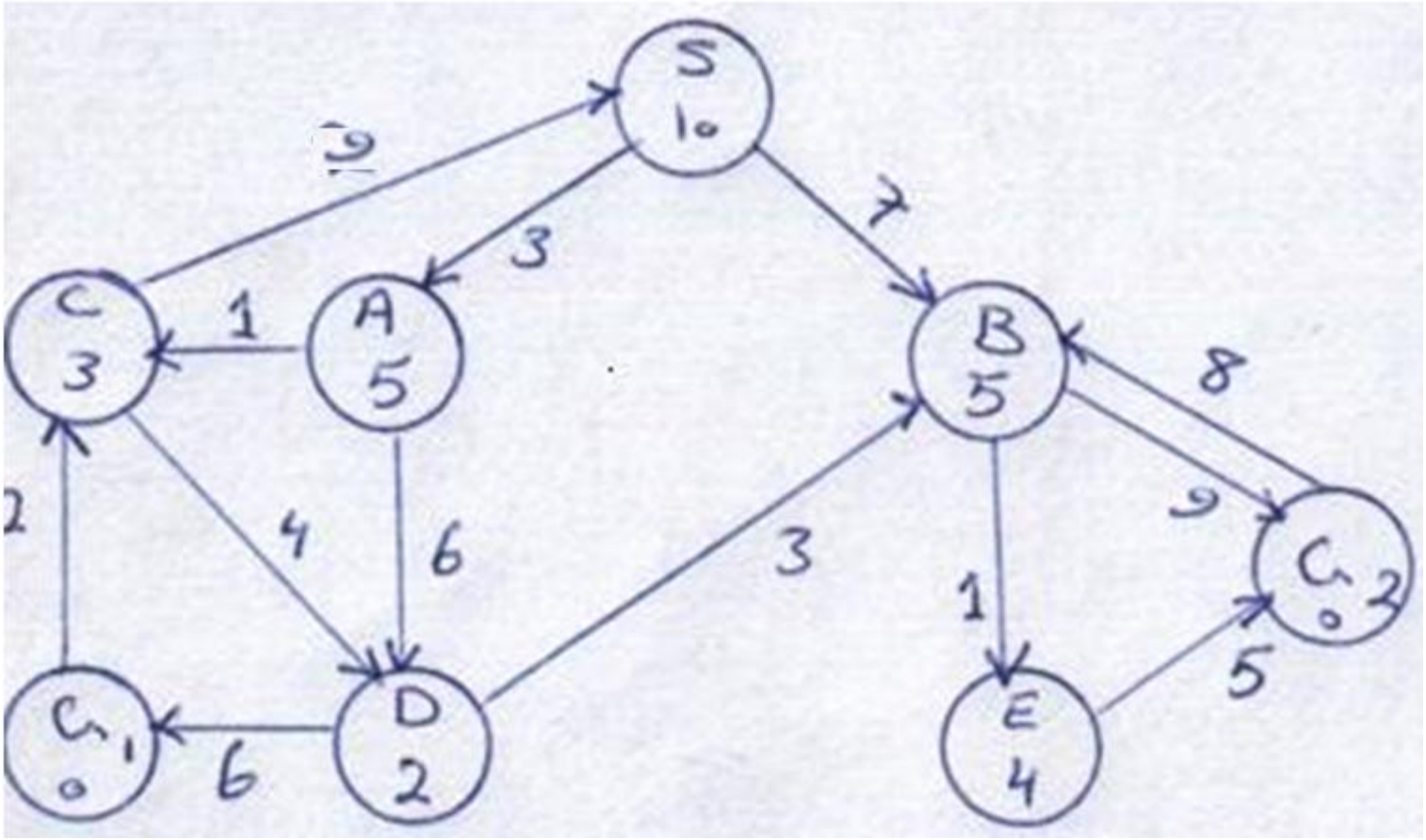
**بهینگی:** بله

اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد

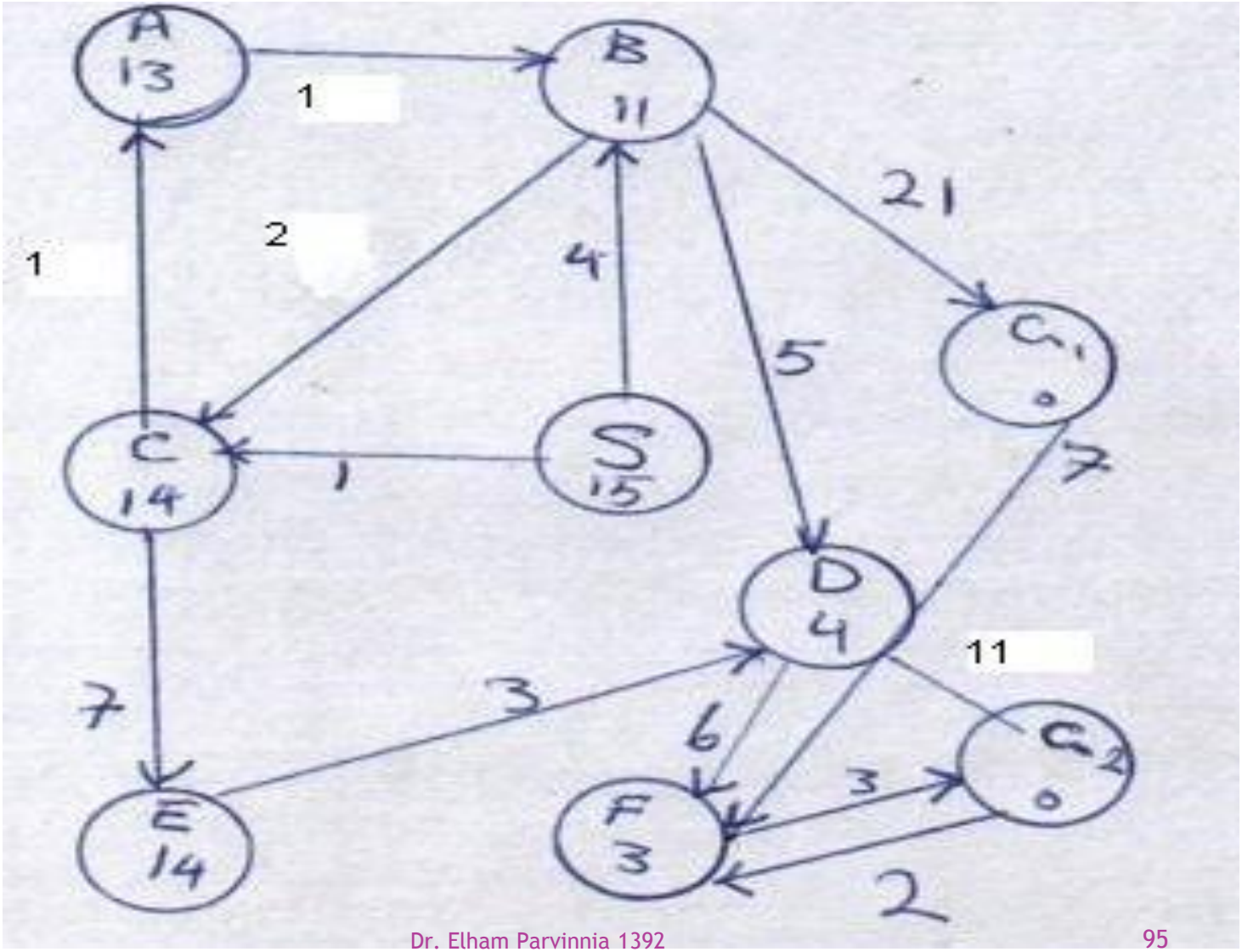
**پیچیدگی زمانی:** $O(b^{d/2})$

**پیچیدگی فضا:** $O(b^{d/2})$

# EXAMPLE1

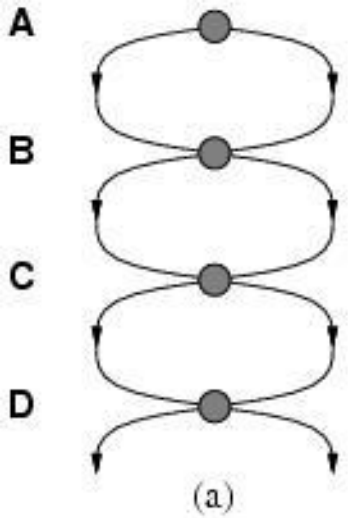


# EXAMPLE2

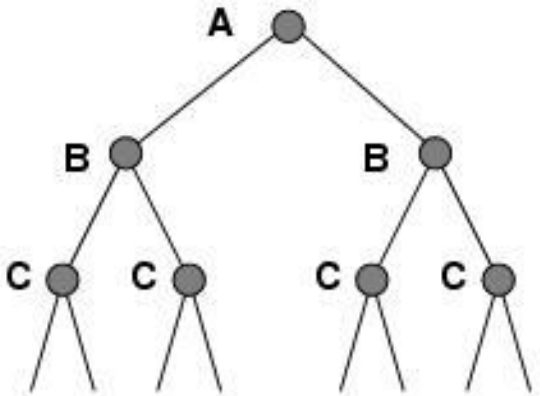


# اجتناب از حالتهای تکراری

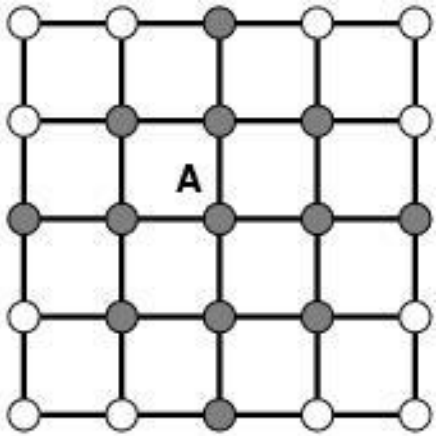
وجود حالتهای تکراری در یک مسئله قابل حل، میتواند آن را به مسئله غیر قابل حل تبدیل کند



(a)



(b)



(c)

# اجتناب از حالتهای تکراری

سه راه برای حل مشکل حالات تکراری وجود دارد:

- به حالتی که هم اکنون از آن آمده اید، برنگردید.
  - داشتن تابع بسطی (یا مجموعه عملگرها) که از تولید مابعدهایی که مشابه حالتی هستند که در آنجا نیز والدین این گره ها وجود دارند، جلوگیری کند.
- از ایجاد مسیرهای دوار بپرهیزید.
  - داشتن تابع بسطی (یا مجموعه عملگرها) که از تولید مابعدهای یک گره که مشابه اجداد آن گره است، جلوگیری کند.
- حالتی را که قبلاً تولید شده است، مجدداً تولید نکنید.
  - این مسئله باعث می شود که هر حالت در حافظه نگه داری شود، و پیچیدگی فضای O(b^d) داشته باشد.

# جستجو با اطلاعات ناقص

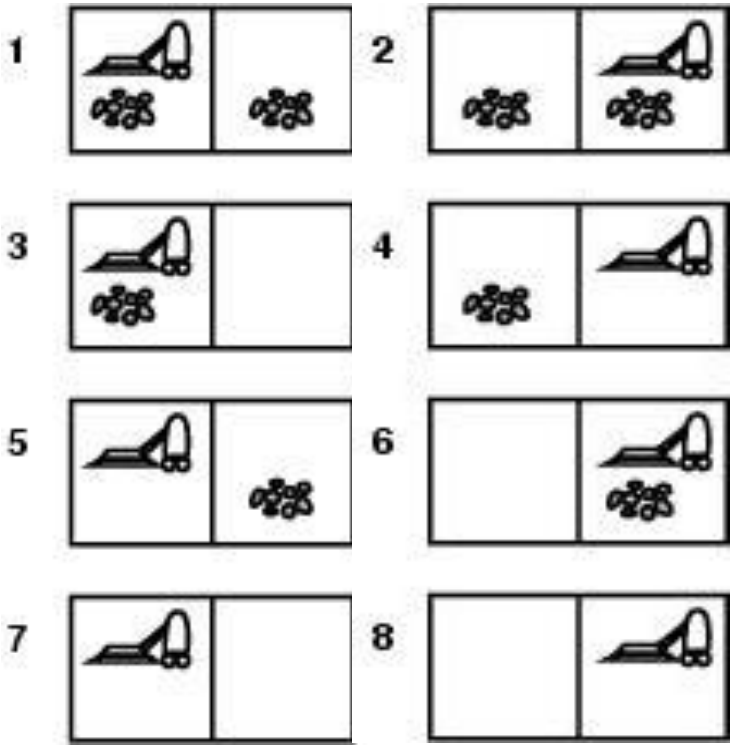
- **مسئله های فاقد حسگر:** اگر عامل فاقد حسگر باشد، میتواند در یکی از چند حالت اولیه باشد و هر فعالیت میتواند آن را به یکی از چند حالت جانشین ببرد

- **مسئله های اقتضایی:** اگر محیط به طور جزئی قابل مشاهده باشد یا اگر فعالیتها قطعی نباشد، ادراکات عامل، پس از هر عمل، اطلاعات جدیدی را تهیه میکنند. هر ادراک ممکن، اقتضایی را تعریف میکند که باید برای آن برنامه ریزی شود

- **مسائل خصمانه:** اگرعدم قطعیت در اثر فعالیتهای عامل دیگری بوجود آید، مسئله را خصمانه گویند

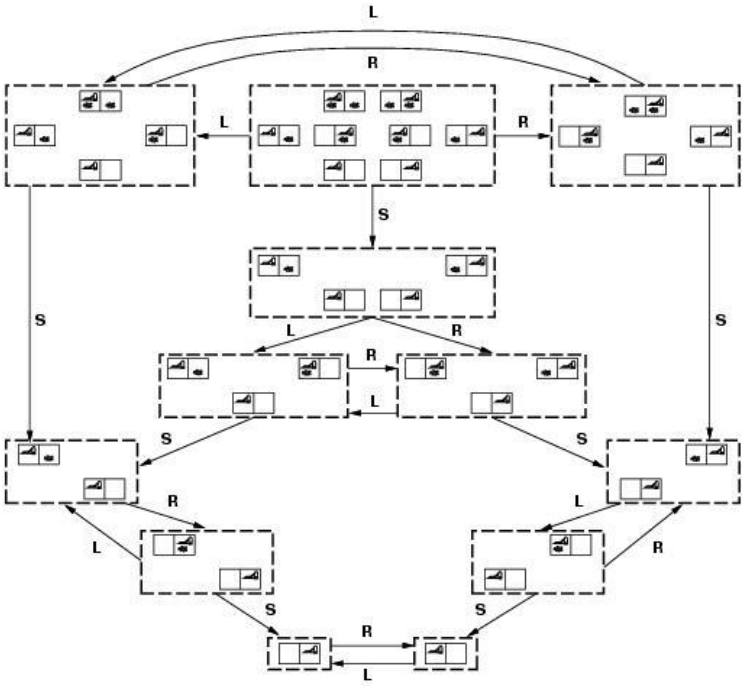
- **مسئله های اکتشافی:** وقتی حالتها و فعالیتهای محیط ناشناخته باشند، عامل باید سعی کند آنها را کشف کند. مسئله های اکتشافی را میتوان شکل نهایی مسئله های اقتضایی دانست

# مثال: دنیای جاروبرقی فاقد حسگر




- عامل جارو تمام اثرات فعالیتهایش را میداند اما فاقد حسگر است.
- حالت اولیه آن یکی از اعضای مجموعه {۱،۲،۳،۴،۵،۶،۷،۸} میباشد
- فعالیت (**Right**) ← {۲،۴،۶،۸}
- فعالیت (**Right,Suck**) ← {۴،۸}
- فعالیت (**Right,Suck,Left,Suck**) تضمین میکند که صرف نظر از حالت اولیه، به حالت هدف، یعنی ۷ برسد

# دنیای جاروبرقی فاقد حسگر




- عامل باید راجع به مجموعه های حالتی که میتواند به آنها برسد استدلال کند. این مجموعه از حالتها را حالت باور گوییم.
- اگر فضای حالت فیزیکی دارای s حالت باشد فضای حالت باور 2^s حالت باور خواهد داشت.