به نام پروردگار دانایی

هوش مصنوعی

درس پنجم: الگوریتم‌های جستجوی ناآگاهانه

سید کاوه احمدی

مرور

- فرموله‌سازی مسئله
  - توصیف گراف فضای حالت
  - فضای حالت یک مدل ریاضی از بیان مسئله است (زبان فرمال)
- درخت جستجو
- معیارهای ارزیابی

استراتژی‌های جستجوی ناآگاهانه

- جستجوی عرضی (Breadth First Search)
- جستجوی هزینه یکنواخت (Uniform Cost Search)
- جستجوی عمقی (Depth First Search)
- جستجوی عمقی محدود شده (Depth Limited Search)
- جستجوی عمیق کننده تکراری (Iterative Deepening Search)
- جستجوی دو طرفه (Bidirectional Search)

جستجوی اول سطح (BFS)

- Breadth-first Search (BFS)
  - ابتدا گره ریشه گسترش می‌یابد سپس تمام گره‌هایی که توسط ریشه تولید شده‌اند گسترش می‌یابند و سپس اولاد آنها.
    - تمام گره‌های عمق d قبل از گره‌های عمق d+1 بسط داده می‌شوند.
  - گره‌ها در یک صف قرار می‌گیرند و به ترتیب بسط داده می‌شوند.
  - BFS کم عمق‌ترین وضعیت هدف را پیدا می‌کند.

جستجوی اول سطح

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```


جستجوی اول سطح

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in proble  ACTIONS(node.STATE) do
      child ← CHILD-N  roblem, node, action)
                        d or frontier then
                       ld.STATE) then return SOLUTION(child)
                       frontier)
```

انتخاب سطحی‌ترین (shallowest) گره از frontier. مفهوم صف.

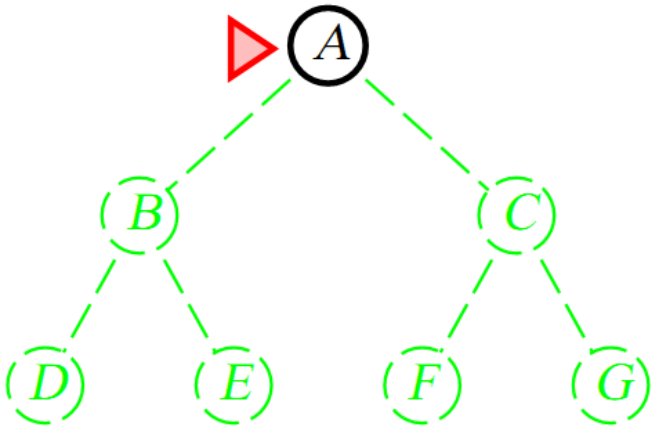
جستجوی اول سطح

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
frontier ← a FIFO queue with *node* as the only element
explored ← an empty set
loop do
  if EMPTY?(*frontier*) **then** return
  node ← POP(*frontier*) /* chooses the shallowest node in *frontier* */
  add *node*.STATE to *explored*
  for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    child ← CHILD-NODE(*problem*, *node*, *action*)
    if *child*.STATE is not in *explored* or *frontier* **then**
      if *problem*.GOAL-TEST(*child*.STATE) **then return** SOLUTION(*child*)
      frontier ← INSERT(*child*, *frontier*)

بررسی هدف بودن گره‌ها در زمان تولید آنها

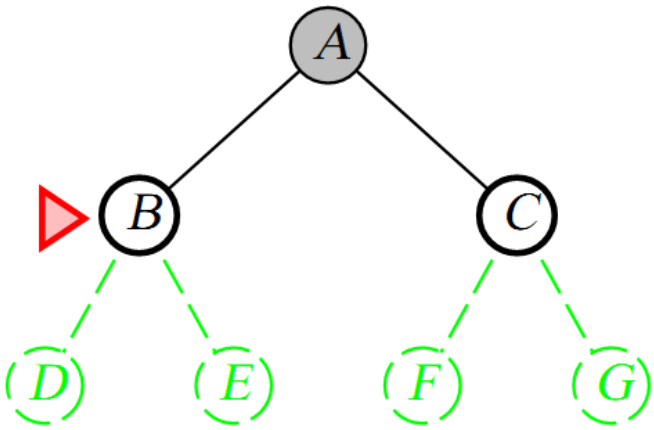
جستجوی اول سطح



Frontier (Queue): A

Explored:

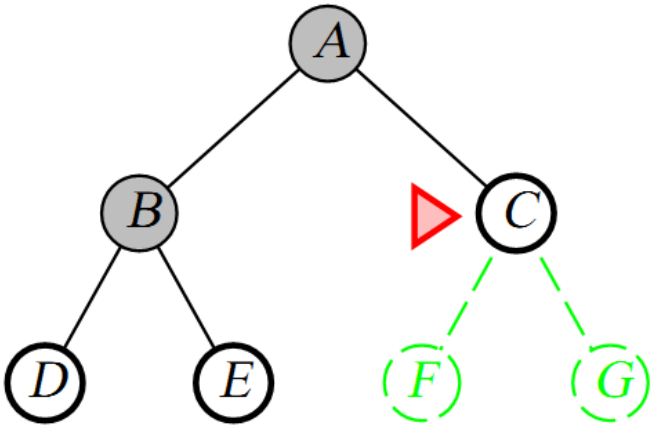
جستجوی اول سطح



Frontier (Queue): B C

Explored: A

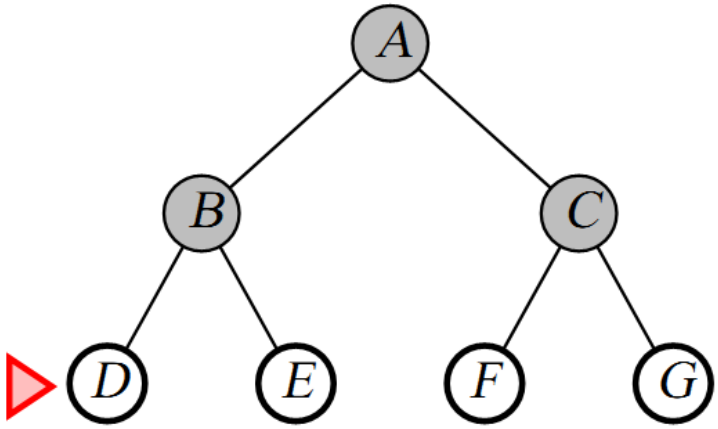
جستجوی اول سطح



Frontier (Queue): C D E

Explored: A B

جستجوی اول سطح



Frontier (Queue): D E F G

Explored: A B C

جستجوی اول سطح

- **کامل است؟**
  - بله به شرط متناهی بودن فاکتور انشعاب (b).
- **بهینه است؟**
  - بله به شرط آنکه مسیرها هزینه نداشته باشد (هزینه مسیرها به میزان ثابت ε در نظر گرفته می‌شود).
  - اگر در مسئله مسیرها دارای هزینه باشد، BFS کم عمق‌ترین وضعیت هدف را پیدا می‌کند که لزوما پاسخ بهینه نیست.
  - اساسا BFS برای شرایطی که مسیرها هزینه نداشته باشند (یا هزینه‌ها ثابت باشد) به کار گرفته می‌شود.

پیچیدگی زمانی جستجوی اول سطح

- پیچیدگی زمان را براساس تعداد گره‌هایی که باید تولید شود محاسبه می‌کنیم:
  - فرض کنید هر گره، b گره مابعد دارد.
  - یک گره در ریشه‌ی درخت تولید می‌شود (۱ گره).
  - ریشه درخت جستجو b گره در سطح اول تولید می‌کند (b گره).
  - گره‌های سطح اول هرکدام b گره در سطح دوم تولید می‌کنند ($b*b=b^2$ گره).
  - گره‌های سطح دوم (b^2) هرکدام b گره در سطح سوم تولید می‌کنند ($b^2*b=b^3$ گره).
  - ...
  - در سطح d، b^d گره خواهیم داشت.

پیچیدگی زمانی جستجوی اول سطح

- فرض کنید کم‌عمق‌ترین پاسخ در سطح d باشد.
- با فرض بررسی هدف بودن گره هنگام تولید (هنگام ورود به frontier):
  - مطابق الگوریتم داخل اسلاید – ویرایش سوم کتاب
  - هنگام تولید گره‌های سطح d هدف پیدا می‌شود. در بدترین شرایط آخرین گره سطح d هدف است:

$$1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$$

- با فرض بررسی هدف بودن گره هنگام بسط (هنگام خروج از frontier):
  - مطابق ویرایش دوم کتاب
  - در بدترین حالت فقط گره هدف (در سطح d) بسط داده نمی‌شود:

$$1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$$

پیچیدگی حافظه جستجوی اول سطح

- مطابق ویرایش سوم کتاب:
  - $1+b+b^2+\ldots+b^{d-1}$ گره در explored set.
    - تعداد گره‌های غیر برگ درخت با فاکتور انشعاب b و عمق d
  - حداکثر b^d-1 گره در frontier.
    - تعداد گره‌های برگ درخت با فاکتور انشعاب b و عمق d

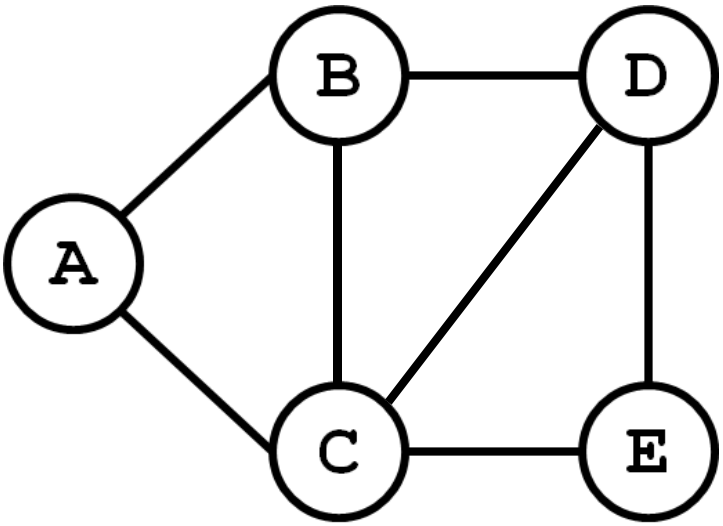
جستجوی اول سطح

- الگوریتم قابل قبول است. اما زمان و فضا نمایی است.
  - با فرض b=10 و d=8 و هر گره ۱۰۰ بایت حافظه بخواهد و در هر ثانیه ۱۰۰۰ گره تولید شود:
    - زمان مورد نیاز یافتن پاسخ: ۳۱ ساعت
    - حافظه مورد نیاز: 11GB

مهندسی ۸۸

- کدام مورد نمی‌تواند ترتیب اجرای الگوریتم BFS روی گراف زیر باشد؟

1. CEDAB
2. BCDAE
3. ABCED
4. ACBDE



مهندسی ۸۸

- گزینه ۳ پاسخ است.
  - با توجه به موارد اشاره شده در مورد نحوه قرارگیری گره‌ها در صف frontier.

مهندسی ۸۷

- ضریب انشعاب یک درخت جستجو ۳ می باشد. حل مسأله در آخرین رأسی که در عمق ۲ جستجو می شود وجود دارد. چه تعداد رأس باید بسط داده شوند تا این رأس بازدید شود در صورتی که از جستجوی عرض نخست استفاده شود؟ (فرض بر این است که حل مسأله بودن یک گره، در زمان باز کردن فرزندان آن گره بررسی می‌گردد.)

1. 13
2. 27
3. 32
4. 37

مهندسی ۸۷

- گزینه ۱ پاسخ است.
- تعداد گره‌های تولید شده برای $b=3$ و $d=2$ (با فرض بررسی هدف بودن گره هنگام بسط آن):
  - $1+3^1+3^2+3(3^2-1)$
  - در سوال، تعداد گره‌های بسط داده شده مورد پرسش قرار گرفته. توجه کنید که گره‌های در عمق $d+1$ هیچگاه بسط داده نخواهند شد. بنابراین تعداد گره‌های بسط داده شده برابر خواهد بود با:
    - $1+3^1+3^2 = 13$

مهندسی ۹۱

- فرض کنید برای مسئله‌ای با جستجوی اول پهنا (breadth first) و تست هدف در لحظه تولید نیاز به بسط دادن (expand) ۳۲ گره باشد. اگر فاکتور انشعاب (branching factor) درخت جستجو ثابت باشد و عمق درخت برابر ۵ و عمق هدف برابر ۴ باشد، کدامیک از گزینه‌ها مقدار فاکتور انشعاب (b) را نشان می‌دهد؟ (فرض کنید ریشه درخت در عمق صفر (۰) واقع شده است)

1. $b = 2$
2. $b > 5$
3. $2 < b < 3$
4. $3 \leq b \leq 5$

مهندسی ۹۱

- گزینه ۴
- پاسخ در عمق ۴ است بنابراین در زمان بسط گره‌های در عمق ۳ پیدا خواهد شد (تست هدف در لحظه تولید است):
  - اگر $b=2$ باشد، تعداد گره‌ها تا عمق ۳ برابر با ۱۵ خواهد بود.
  - اگر $b=3$ باشد، در عمق ۳ بین ۱۴ تا ۴۰ گره خواهیم داشت.
  - اگر $b=5$ باشد، در عمق ۳ بین ۳۲ تا ۱۰۶ گره خواهیم داشت.
- بنابراین فاکتور انشعاب بین ۳ تا ۵ قابل قبول خواهد بود.

جستجو با هزینه یکسان (UCS)

- Uniform-cost Search (UCS)
  - اگر مسیرها دارای هزینه (بزرگ‌تر از ۰) باشد، جستجوی اول سطح، کم عمق‌ترین هدف را پیدا می‌کند که لزوما کم هزینه‌ترین مسیر نیست.
  - اصلاح جستجوی اول سطح برای یافتن کم هزینه‌ترین هدف.
  - همواره گرهی بسط داده می‌شود که کم هزینه‌ترین مسیر را داشته باشد. نه لزوما کم‌عمق‌ترین گره.
    - صف بر اساس هزینه مسیر مرتب می‌شود (صف اولویت): کم هزینه‌ترین مسیر اول!
      - پیاده‌سازی با heap – درخت نیمه مرتب

جستجو با هزینه یکسان

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

```
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with node as the only element
explored ← an empty set
loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
            frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
            replace that frontier node with child
```

جستجو با هزینه یکسان

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with *node* as the only element
explored ← an empty set
loop do
  if EMPTY?(*frontier*) **then return** failure
  node ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  add *node*.STATE to *explored*
  for e… …ode.STATE) **do**
    child ← CHILD-NODE(*problem*, *node*, *action*)
    if *child*.STATE is not in *explored* or *frontier* **then**
      frontier ← INSERT(*child*, *frontier*)
    else if *child*.STATE is in *frontier* with higher PATH-COST **then**
      replace that *frontier* node with *child*

انتخاب کم هزینه‌ترین گره. مفهوم صف اولویت.

جستجو با هزینه یکسان

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with *node* as the only element
explored ← an empty set
loop do
  if EMPTY?(*frontier*) **then return** failure
  node ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  add *node*.STATE to *explored*
  for each *action* **in** *problem*.ACTIONS… …STATE) **do**
    child ← CHI…
    if *child*.STAT… …n
      frontier …
    else if *child*.STATE is in *frontier* with higher PATH-COST **then**
      replace that *frontier* node with *child*

بررسی هدف بودن گره‌ها در زمان بسط آنها (زمان خروج از frontier)

جستجو با هزینه یکسان

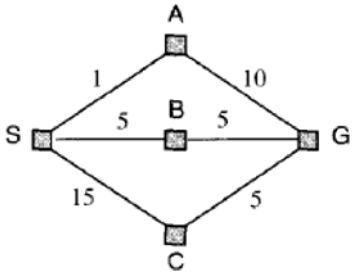
function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with *node* as the only element
explored ← an empty set
loop do
if EMPTY?(*frontier*) **then return** failure
node ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */

همواره گره ها با کمترین هزینه دسترسی در frontier مرتب می‌شوند. یعنی اگر گره‌ای در frontier باشد و مسیری پیدا شود که با هزینه کمتر به آن گره برسد، frontier بر اساس هزینه کمتر به روز می‌شود.

if *child*.STATE is not in *explored* or *fron*
frontier ← INSERT(*child*, *frontier*)
else if *child*.STATE is in *frontier* with higher PATH-COST **then**
replace that *frontier* node with *child*

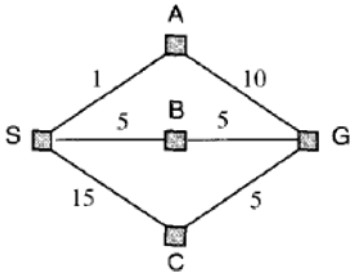
جستجو با هزینه یکسان

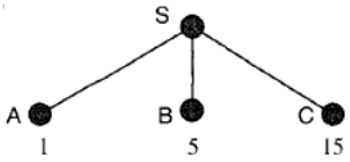


Frontier (Queue): S

Explored:

جستجو با هزینه یکسان

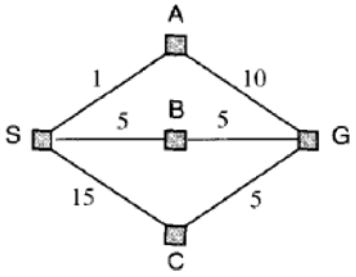


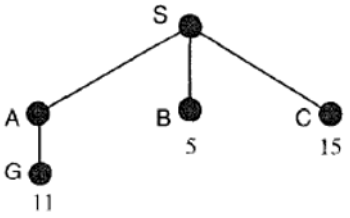


Frontier (Queue): A B C

Explored: S

جستجو با هزینه یکسان



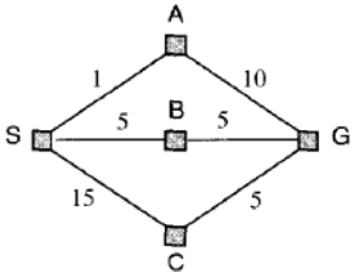


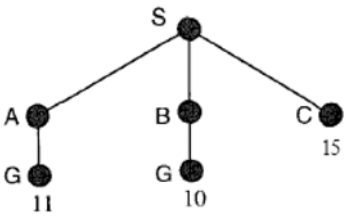
Frontier (Queue): B G_A C

Explored: S A

بررسی هدف بودن گره هنگام بسط انجام می‌شود. در اینجا هدف در frontierقرار گرفته اما هنوز هدف بودن آن مورد بررسی قرار نگرفته.

جستجو با هزینه یکسان





- گره G در frontier قرار داشت اما چون از طریق گره دیگری و با هزینه کمتر به G رسیدیم، اولویت گره G در frontier تغییر می‌کند.
- در مرحله بعد G از frontier خارج می‌شود و هدف بودن آن بررسی می‌شود. هدف است و الگوریتم به پایان می‌رسد.

Frontier (Queue): G_B ~~G_A~~ C

Explored: S A B

جستجو با هزینه یکسان

- در BFS بررسی هدف در هنگام تولید یا بسط گره در نتیجه‌ی جستجو تاثیری نخواهد داشت.
- در UCS همانطور که در مثال قبل مشخص است، بررسی هدف بودن گره باید هنگام بسط گره صورت گیرد در غیر این صورت ممکن است به هدف بهینه نرسیم.
- در UCS اولین راه حل پیدا شده، بهترین راه‌حل است مشروط بر اینکه هزینه مسیر با ادامه مسیر کاهش پیدا نکند (هزینه همه عمل‌ها غیر منفی باشد)
- وجود مسیرهای با طول صفر ممکن است منجر به ایجاد گذر تهی (مسیر نامتناهی با طول صفر) می‌شود.
- بنابراین در UCS هزینه مسیرها باید مثبت باشد.

جستجو با هزینه یکسان

- اگر تابع هزینه مسیر را با g(n) نمایش دهیم (کمترین هزینه از مبدا گره n):
  - جستجوی اول سطح همان جستجو با هزینه یکسان با g(n) = DEPTH(n) است.
- اگر هزینه همه اعمال یکسان باشد، UCS همانند جستجوی اول سطح است.

جستجو با هزینه یکسان

- کامل است اگر هزینه هر مرحله حداقل $\varepsilon > 0$ باشد.
- بهینه است: اول هزینه‌های کمتر توسعه داده می‌شود.
- پیچیدگی زمان و حافظه: $O\left(b^{1+\lfloor C^*/\varepsilon \rfloor}\right)$
  - C^* هزینه پاسخ بهینه است و ε کمترین وزن در یال‌های گراف.
  - طول بهینه: فرض کنید همه مسیر تا هدف شامل یال‌های با هزینه ε باشد: $C^*/\varepsilon = d$
    - پیچیدگی برابر خواهد بود با: $O(b^{1+d})$
  - مشابه پیچیدگی BFS با بررسی هدف هنگاه بسط (در UCS نیز دیدیم بررسی هدف باید هنگام بسط باشد)

آی تی ۸۹

- کدام عبارت برای حل یک مسئله با روش جستجو غلط است؟

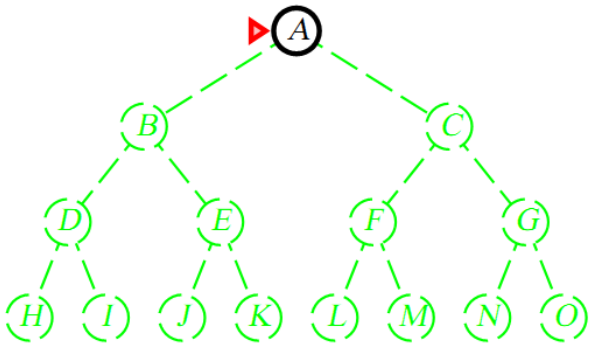
1. هزینه از یک حالت تا حالت بعدی باید مشخص باشد
2. حالت‌های بعدی هر حالت باید مشخص باشند
3. حالت‌های هدف باید مشخص باشد
4. حالت شروع باید مشخص باشد

آی تی ۸۹

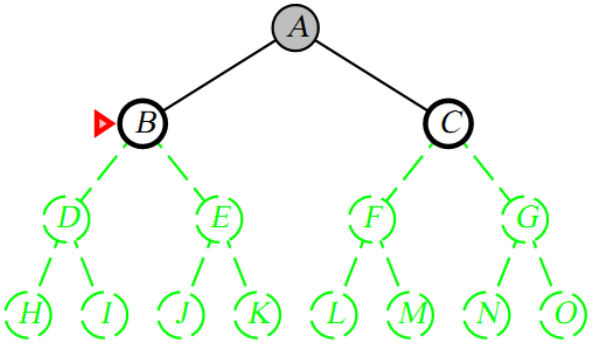
- گزینه ۱
  - در صورت عدم وجود هزینه یال، می‌توان از الگوریتم‌های ناآگاهانه BDS و DFS بهره گرفت. بنابراین وجود هزینه یال «ماست» نیست!

جستجوی اول عمق

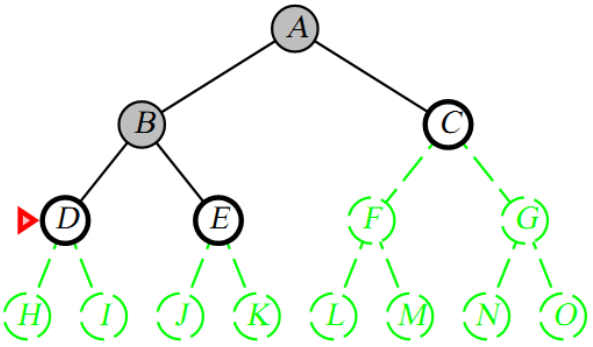
- Depth-first Search (DFS)
  - همواره عمیق‌ترین گره در حاشیه درخت جستجوی کنونی توسعه داده می‌شود.
    - جستجو بلافاصله تا عمیق‌ترین سطح درخت گسترش می‌یابد.
  - زمانی که به گره‌ی رسید که مابعد ندارد، به عقب بازگشته و کم‌عمق‌ترین گره حاشیه‌ای که هنوز توسعه نیافته را گسترش می‌دهد.
  - این استراتژی از ساختمان داده‌ی پشته برای نگهداری گره‌های حاشیه‌ای استفاده می‌کند.



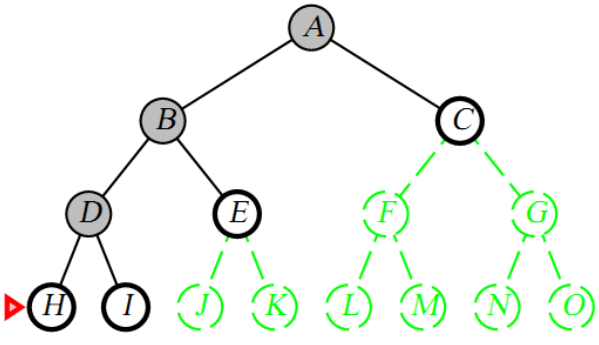
Frontier: A



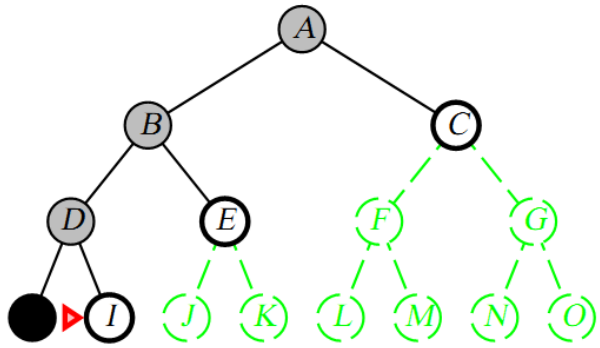
Frontier: B C



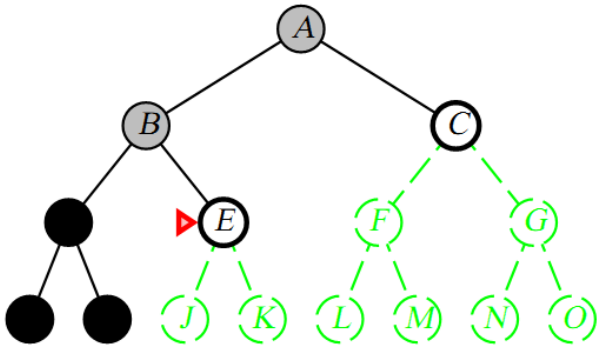
Frontier: D E C



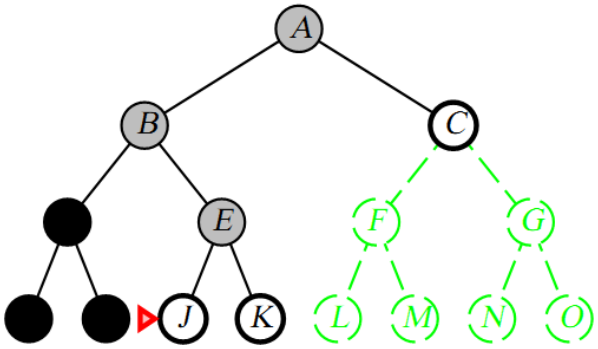
Frontier: H I E C



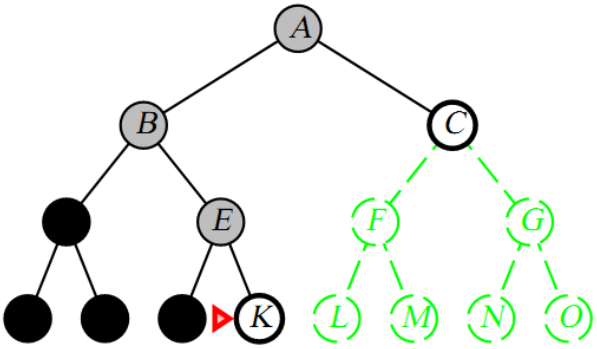
Frontier: I E C



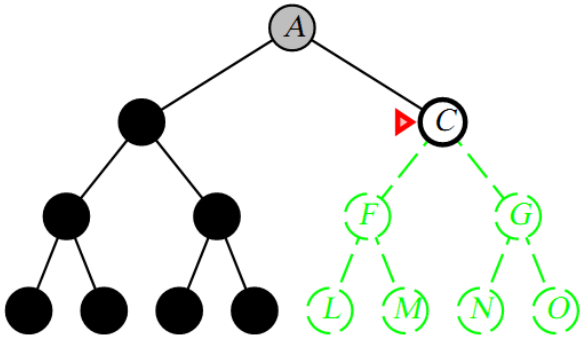
Frontier: E C



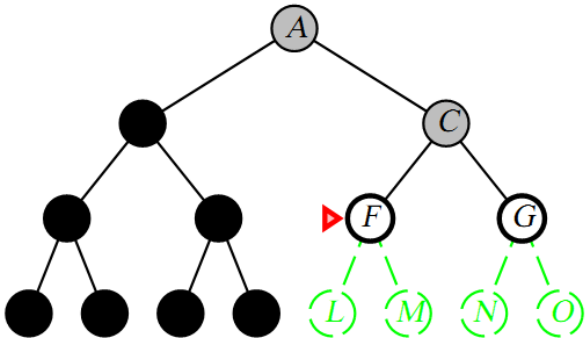
Frontier: J K C



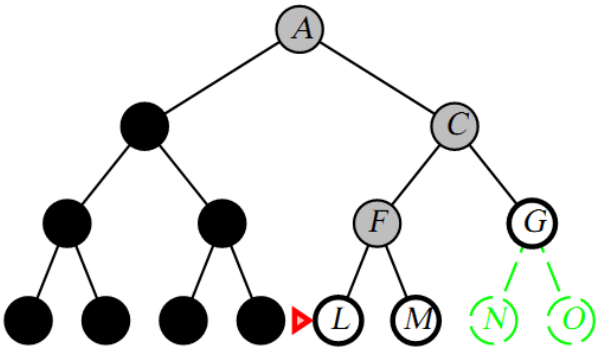
Frontier: K C



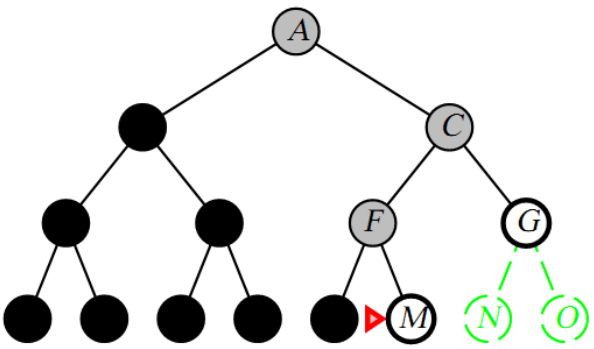
Frontier: C



Frontier: F G



Frontier: L M G



Frontier: M G

جستجوی اول عمق

- در هر لحظه یک مسیر از ریشه تا یک برگ را به همراه نودهای همزادش نگه می‌دارد.
  - یک نود گسترش یافته، به محض اینکه همهٔ نوادگانش کاملاً بررسی شدند، از حافظه خارج می‌شود.
- پیاده‌سازی بازگشتی ممکن است.
  - تابع بازگشتی خود را به ازای همه فرزندان فراخوانی کند.

جستجوی اول عمق

- کامل است؟
  - برای فضای حالت گرافی که متناهی باشد کامل است (با کمک explore set).
  - برای فضای حالت درختی اگر زیر درخت چپ عمق نامحدود داشته باشد و از جستجوی درختی استفاده کنیم، جستجو هرگز خاتمه نمی‌یابد.
- بهینه است؟
  - خیر
  - BFS برای مسائل بدون هزینه مسیر بهینه بود (و USC برای مسائل با هزینه مسیر).

جستجوی اول عمق

- حافظه؟
  - هر گره پس از پویش تمام نوادگان بلافاصله حذف می‌شود.
  - حداکثر به میزان ذخیره یک مسیر از ریشه به یک برگ و گره‌های خواهر و برادر توسعه نیافته‌ی گره‌های داخل مسیر
  - برای یک فضای حالت با فاکتور انشعاب b و حداکثر عمق m، جستجوی اول عمق فضایی به میزان bm+1 نیاز دارد. (O(bm) در برابر $O(b^d)$ در BFS)
- زمان؟
  - در بدترین حالت $O(b^m)$
    - $1 + b + b^2 + \ldots + b^m$

جستجوی اول عمق

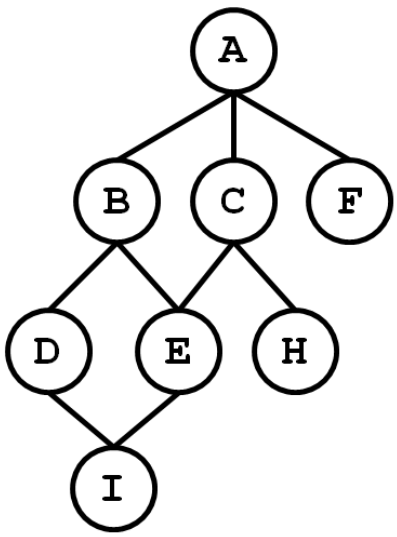
- اگر مقدار m خیلی بزرگتر از d باشد این روش عملکرد بسیار بدی خواهد داشت.
- اگر اهداف با تراکم زیاد در درخت جستجو پراکنده شده باشند، این روش از جستجوی سطحی بهتر عمل می‌کند.

جستجوی عقبگرد (Backtrack)

- مانند جستجوی عمقی است، با این تفاوت که در هر لحظه بجای تولید همه فرزندان فقط یک فرزند تولید می شود، در عوض بخاطر می سپارد که بعداً کدام فرزندان باید گسترش یابند.
  - گره‌های خواهر برادر گره جاری در پشته ذخیره نمی‌شوند.
- در این حالت پیچیدگی حافظه به O(m) کاهش می یابد، در عوض زمان بیشتری برای بازگشت به عقب و توسعه یک زیر شاخهٔ دیگر صرف می شود.

مهندسی ۸۱

- اگر در گراف روبرو جستجوی اول عمق (DFS) را از راس C شروع کنیم، به ترتیب کدام گره‌ها دیده می‌شوند؟ (فرض کنید فرزندان یک گره بر اساس ترتیب حروف الفبا انتخاب می‌شوند)



1. ABCDEFHI
2. CABDIEFH
3. CAEHBFID
4. CABDEHIF

مهندسی ۸۱

- گزینه ۲

مهندسی ۹۰

- روی یک توری n*n که هر خانه به چهار همسایه خود متصل است، خانه میانی را نقطه شروع جستجو و نقطه (۰, ۰) در نظر می‌گیریم. گره هدف در موقعیت (x, y) است. در این الگوریتم جستجوی A بدون تست تکراری بودن حالات حداکثر $\left(\left(4^{x+y+1}-1\right)/3\right)-1$ گره و جستجوی B با تست تکراری بودن حالات حداکثر 2(x+y)(x+y+1)-1 گره را قبل از یافتن جواب بسط می‌دهند. کدام یک از گزینه‌های زیر در مورد این دو الگوریتم صحیح است؟

1. A و B هر دو الگوریتم اول پهنا (Breadth first) هستند
2. A و B هر دو الگوریتم اول عمق (Depth first) هستند
3. A الگوریتم اول پهنا (Breadth first) و B الگوریتم اول عمق (Depth first) است
4. A الگوریتم اول عمق (Depth first) و B الگوریتم اول پهنا (Breadth first) است

مهندسی ۹۰

- گزینه ۱
  - در جستجوی سطحی، تعداد گره‌های تولید شده تابعی از فاکتور انشعاب و عمق پاسخ است.
  - در جستجوی عمقی، تعداد گره‌های تولید شده تابعی از فاکتور انشعاب و عمق درخت است.
  - در مسئله فوق، معیار ارزیابی از X و Y کمک گرفته که وابسته به عمق پاسخ است و در هیچکدام از توابع سخنی از عمق درخت نشده.
  - می‌توان با کمی هوشمندی نتیجه گرفت که هر دوی روش‌ها سطحی هستند

جستجوی عمقی محدود شده (Depth-limited Search)

- DFS در حالتی که درخت جستجو عمق نامحدود داشته باشد کامل نیست.
- مسأله درخت‌های نامحدود می‌تواند به وسیله محدود کردن عمق جستجو بهبود یابد.
- یعنی گره‌های با عمق l (limitation) به عنوان برگ در نظر گرفته شوند.
- محدودیت عمل می‌تواند براساس دانش مسئله باشد.
  - در مسئله رومانی از آنجا که ۲۰ شهر داریم، بنابراین اگر پاسخی وجود داشته باشد طول آن حداکثر ۱۹ است.

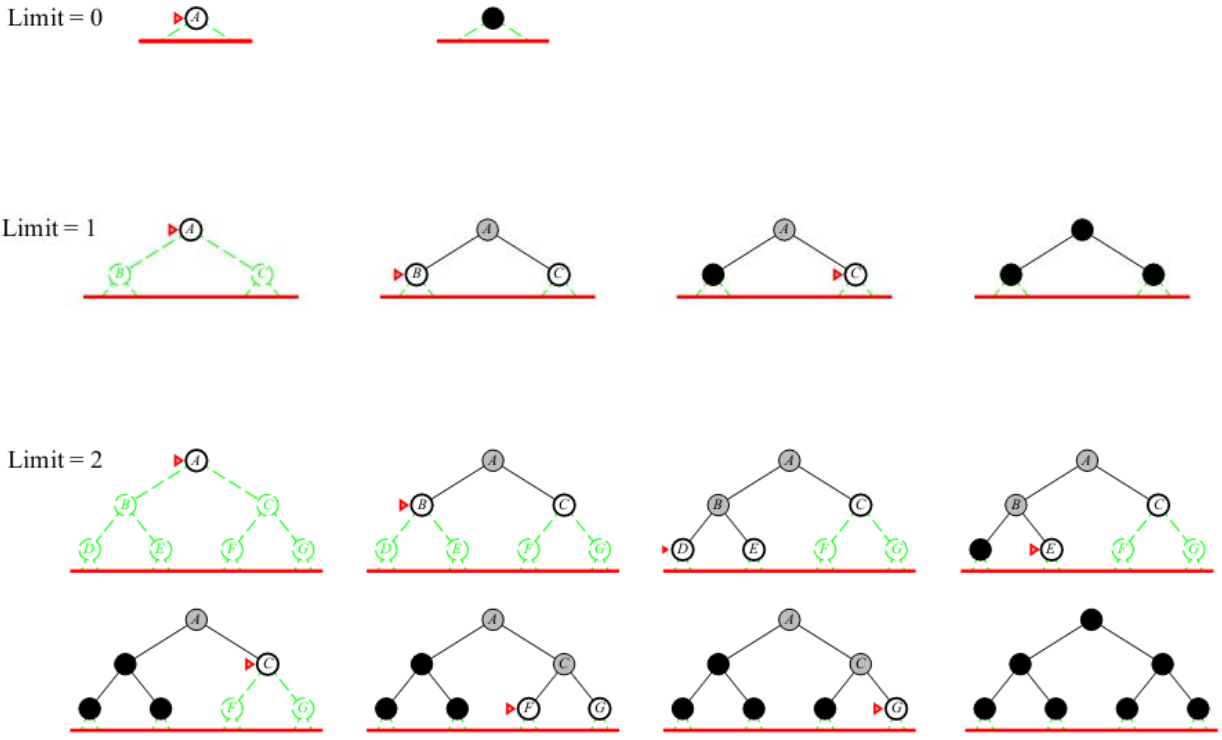
جستجوی عمقی محدود شده (Depth-limited Search)

- کامل است؟
  - اگر l<d یعنی سطحی‌ترین هدف خارج از عمق تعیین شده باشد کامل نخواهد بود. (زمانی ممکن است که d ناشناخته باشد)
- بهینه است؟
  - مسلما خیر
- پیچیدگی زمان: $O(b^l)$
- پیچیدگی حافظه: O(bl)

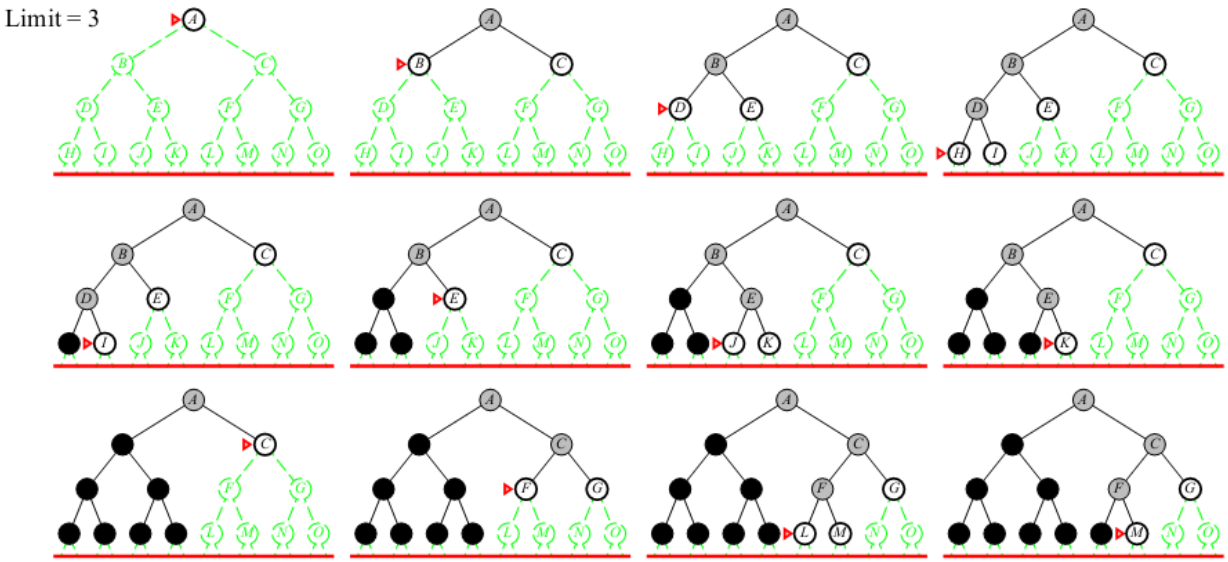
جستجوی عمیق کننده تکراری (Iterative deepening search)

- قسمت دشوار جستجوی عمقی محدود شده، انتخاب یک محدوده خوب است.
- برای بیشتر مسائل، محدوده عمقی مناسب را تا زمانی که مسئله حل نشده است، نمی‌شناسیم.
- جستجوی عمیق کننده تکراری استراتژی است که انتخاب بهترین محدوده عمقی، توسط امتحان کردن تمام محدوده مسیرهای ممکن را انجام می‌دهد.
- مانند این است که جستجوی عمقی محدود را چند بار تکرار کنیم و در هر تکرار محدودیت l را افزایش دهیم.
  - وقتی به کم عمق ترین هدف برسیم الگوریتم به پایان می رسد.

جستجوی عمیق کننده تکراری (Iterative deepening search)



جستجوی عمیق کننده تکراری (Iterative deepening search)



جستجوی عمیق کننده تکراری (Iterative deepening search)

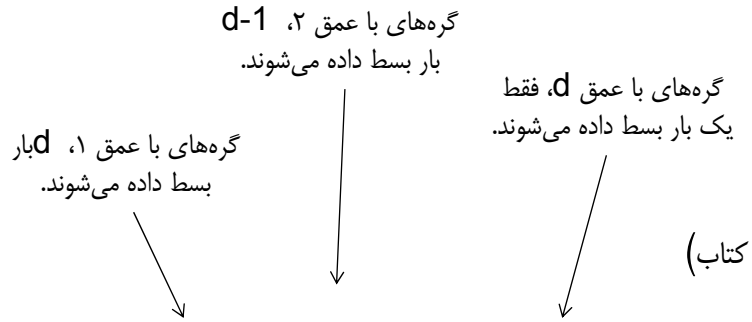
- این جستجو، جستجوی اول عمق و اول سطح را ترکیب می کند.
  - در هر تکرار کل گره‌های عمق مشخص شده (l) را بررسی می‌کند.
- ترکیبی از مزایای جستجوی سطحی و عمقی را دارد.
- این جستجو مانند جستجوی سطحی کامل و بهینه است، اما فقط مزیت درخواست حافظه اندک را از جستجوی عمقی دارد.
- مرتبه بسط حالات مشابه جستجوی سطحی است، به جز اینکه بعضی حالات چند بار بسط داده می شوند.

جستجوی عمیق کننده تکراری (Iterative deepening search)

- به نظر می‌آید که این استراتژی اتلاف کننده است چون برخی وضعیت‌ها را چندین بار تولید می‌کند.
- این هزینه چندان اهمیت ندارد، سطح‌های بالای درخت، گره‌های زیادی ندارد (اگر فاکتور انشعاب محدود باشد)
  - در محاسبه زمان این مورد دیده می‌شود.
- اگر فضای حالت بزرگ و عمق راه حل نامشخص باشد، در میان جستجوهای ناآگاهانه ، جستجوی عمقی تکرارشونده ترجیح داده می شود.

جستجوی عمیق کننده تکراری (Iterative deepening search)

- کامل است؟
  - بله. در صورتی که فاکتور انشعاب محدود باشد.
- بهینه است؟
  - بله (برای مسائل بدون هزینه مسیر).
- حافظه؟
  - O(bd)
- زمان؟ (مطابق ویرایش سوم کتاب)



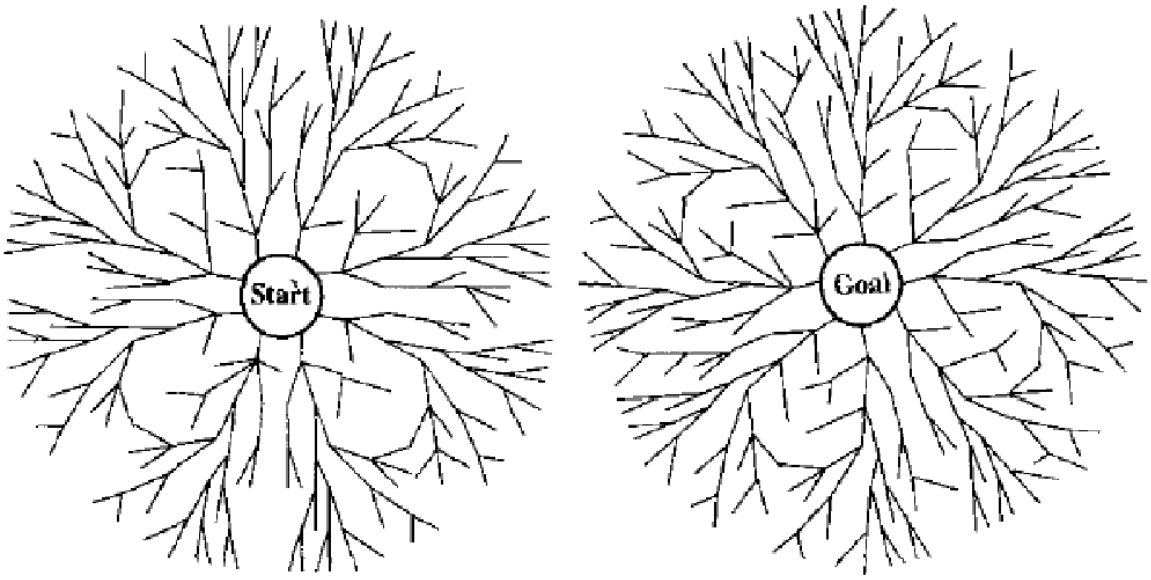
  - $N(IDS) = db + (d-1)b^2 + ... + 1b^d = O(b^d)$

Iterative lengthening search

- IDS برای مسائل بدون هزینه مسیر قابل قبول است.
- ایده Iterative lengthening search همانند IDS است برای مسائل با هزینه مسیر.
  - روش را مطالعه کنید.
  - براس اساس هزینه از مبدا (کمترین مقدار ممکن گرفته می‌شود)
- نشان داده می‌شود که کارا نیست.
  - تمرین کتاب است.

جستجوی دوطرفه

- 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 دوطرفه

- جستجو از سمت هدف به چه معنی است؟
  - ماقبل‌های (predeccessors) یک گره n را گره هایی درنظر می گیریم که n مابعد آنها باشد.
  - جستجو به سمت عقب بدین معناست که تولید ماقبل ها از گرهٔ هدف آغاز شود.
    - برای معمای ۸ می‌توان predeccessors function نوشت.
- زمانی که هدف های متفاوتی وجود داشته باشد این جستجو کارا نیست.
- باید یک راه موثر برای کنترل هر گره جدید وجود داشته باشد تا متوجه شویم که آیا این گره قبلاً در درخت جستجو توسط جستجوی طرف دیگر، ظاهر شده است یا خیر.
- نیاز داریم که تصمیم بگیریم که چه نوع جستجویی در هر نیمه قصد انجام دارد. به عنوان مثال اسلاید قبل انتخاب جستجوی BFS در هر دو طرف را نمایش می دهد.

جستجوی دوطرفه

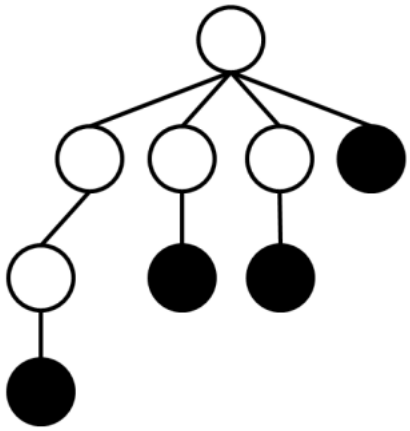
- کامل است؟
  - بله. اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد.
- بهینه است؟
  - بله. اگر هر دو جستجو، عرضی باشند و هزینه تمام مراحل یکسان باشد.
- پیچیدگی زمانی: $O(b^{d/2})$
- پیچیدگی فضا: $O(b^{d/2})$
  - با فرض یکسان بودن درجه ورودی و خروجی گره‌ها

مقایسه استراتژی‌های جستجو:

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete	Yes	Yes	No	Yes, if l>d	Yes	Yes
Time	b^d	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes

مهندسی ۸۵

- در حین انجام یک روش جستجو، درخت جستجوی حاصل به شکل مقابل رشد یافته است. راس‌هایی که نامزد بسط داده شدن هستند به رنگ سیاه مشخص شده‌اند. این جستجو چه روشی می‌تواند باشد؟



1. عمق نخست (dfs)
2. عرض نخست (bfs)
3. هزینه یکنواخت (ucs)
4. تعمیق تکراری (ids)

مهندسی ۸۵

- گزینه ۳