بسمه تعالی

هوش مصنوعی

حل مسئله ـ۳

نیمسال اوّل ۱۴۰۴-۱۴۰۳

دکتر مازیار پالهنگ

آزمایشگاه هوش مصنوعی

دانشکدهٔ مهندسی برق و کامپیوتر

دانشگاه صنعتی اصفهان

یادآوری

- مثال جهانگرد
- تدوین هدف و مسئله
- شرایط محیط برای یک عامل مسئله حل کن:
  - مشاهده پذیر، قطعی، شناخته شده
- تدوین مسئله
  - حالت اولیه، مجموعهٔ اعمال ممکن، مدل انتقال، هدف، هزینهٔ مسیر
- چند مثال:
  - دنیای جارو، جورچین ۸، مسیریابی، گردشگری، فروشندهٔ دوره گرد
- جستجو برای حل
  - ایجاد درخت، مجموعهٔ پیشگام
  - جستجوی بهترین نخست

جستجو برای حل

- اگر به درخت جستجوی مثال دقت شود، رأس همدان دوبار دیده می شود.
- به چنین رئوسی، رئوس تکراری گفته می شود.
- رئوس تکراری باعث ایجاد مسیر حلقوی می شوند.
- وجود چنین رئوسی باعث می شود که درخت بی نهایت بزرگ شود.
- ولی فضای حالت محدود است.
- مسیرهای حلقوی حالت خاص مسیرهای زائد هستند.
- مسیر زائد هنگامی وجود دارد که بیش از یک مسیر بین دو حالت وجود دارد.

جستجو برای حل

- گاهی می توان مسئله را به گونه ای تدوین کرد که دارای تکرار نباشد.
- بطور مثال در مسئلهٔ ۸ وزیر، اگر هر وزیر را بتوان در هر ستونی از صفحهٔ شطرنج گذاشت، در این حالت هر وضعیت قرار گیری n وزیر در صفحه دارای $n!$ مسیر مختلف خواهد بود.
- ولی اگر وزیر را فقط بتوان در چپترین ستون خالی قرار داد فقط یک مسیر وجود دارد.
- در برخی از مسائل که اعمال برگشت پذیر هستند، همانند مثال همدان، حالتهای تکراری اجتناب ناپذیر هستند.

- مثالی است که "**جامعه ای که تاریخ خود را فراموش کند، محکوم به تکرار آن است.**"
- برای اجتناب از مسیرهای زائد، لازم است مکانهائی که بوده ایم را به خاطر بسپاریم.
- استفاده از ساختمان داده ای به نام **مجموعهٔ اکتشاف شده** (explored set)
- یا لیست بسته
- الگوریتمی که از این مجموعه استفاده می کند جستجوی گرافی نامیده می شود.
- در الگوریتم بهترین نخست از ساختمان دادهٔ reached استفاده شده

- مثالی است که "**جامعه ای که تاریخ خود را فراموش کند، محکوم به تکرار آن است.**"
- برای اجتناب از مسیرهای زائد، لازم است مکانهائی که بوده ایم را به خاطر بسپاریم.
- استفاده از ساختمان داده ای به نام **مجموعهٔ اکتشاف شده** (explored set)
- یا لیست بسته
- الگوریتمی که از این مجموعه استفاده می کند جستجوی گرافی نامیده می شود.
- در الگوریتم بهترین نخست از ساختمان دادهٔ reached استفاده شده
- اگر رئوس تکرار بررسی نشوند جستجوی درختی نامیده می شود.

الگوریتمهای جستجوی درختی

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

- یک مصالحه آن است که فقط دورها را چک کنیم.

کارآئی استراتژیهای جستجو

- استراتژی جستجو ترتیب بسط دادن رئوس را مشخص می نماید.
- استراتژیها بر اساس معیارهای زیر ارزیابی می شوند:
  - کامل بودن
    - آیا الگوریتم ضمنانت می دهد که اگر راه حلی وجود دارد، حلی بیابد و اگر نه بدرستی اعلام شکست کند؟
  - بهینه بودن هزینه
    - آیا الگوریتم حل با کمترین هزینه را می یابد؟
  - پیچیدگی فضا
    - میزان حافظه مصرف شده
  - پیچیدگی زمان
    - زمان صرف شده برای یافتن حل، قابل محاسبه بر حسب واحد زمان، تعداد حالات و اعمال

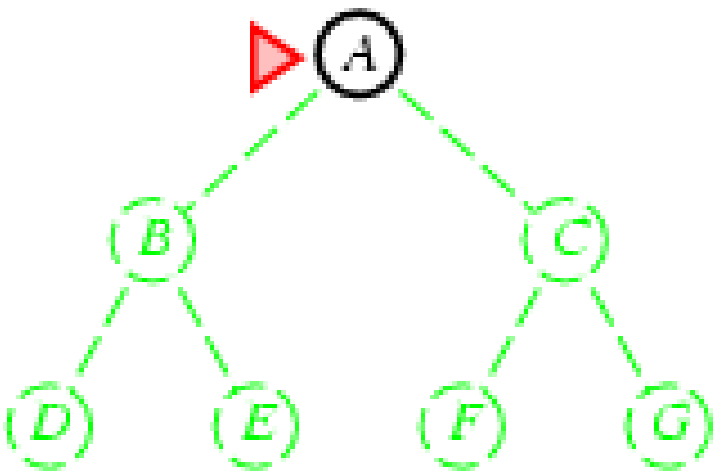
استراتژیهای جستجو

- انواع جستجو:
  - ناآگاهانه
  - آگاهانه

- برای محاسبهٔ پیچیدگی زمان و فضا از پارامترهای زیر استفاده می شود:
- b ضریب انشعاب (حداکثر مقدار آن)
- d عمق کم هزینه ترین حل
- m حداکثر عمق فضای حالت

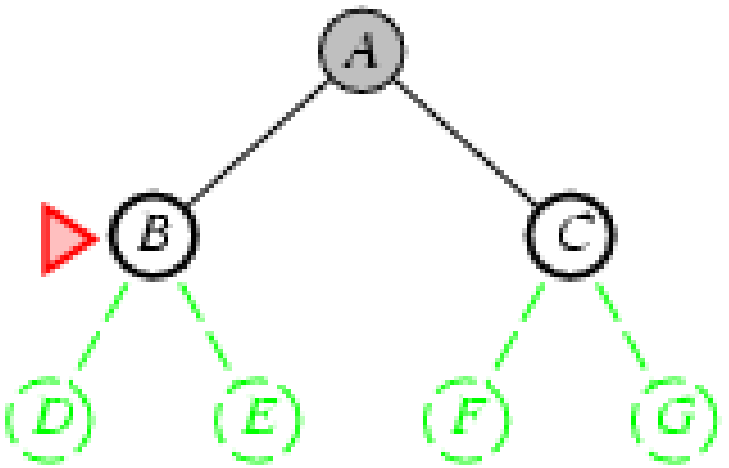
جستجوی عرض نخست

- بسط دادن کم عمق ترین گرهٔ رسیده و بسط داده نشده



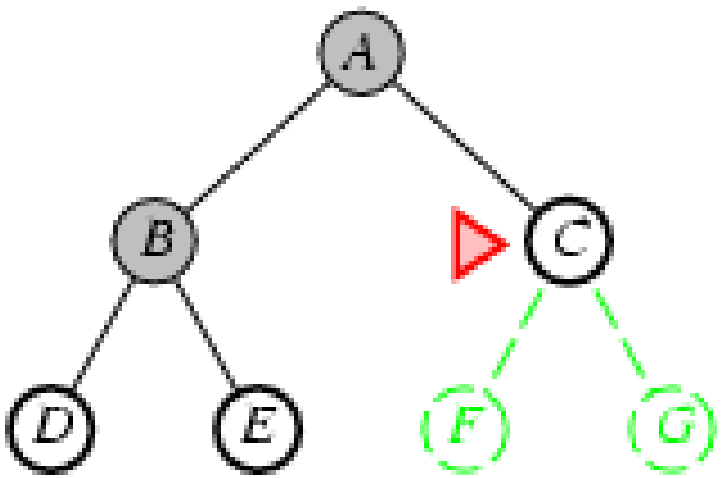
جستجوی عرض نخست

- بسط دادن کم عمق ترین گرهٔ رسیده و بسط داده نشده



جستجوی عرض نخست

- بسط دادن کم عمق ترین گرهٔ رسیده و بسط داده نشده



جستجوی عرض نخست

- می توان از جستجوی بهترین نخست با $f(n)$ برابر با عمق n استفاده کرد.
- می توان کارآئی بهتری داشت.
- frontier می تواند یک صف FIFO باشد.
- چون بر حسب عمق مرتب می کنیم.

جستجوی بهترین نخست

Figure 3.7

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

Figure 3.9

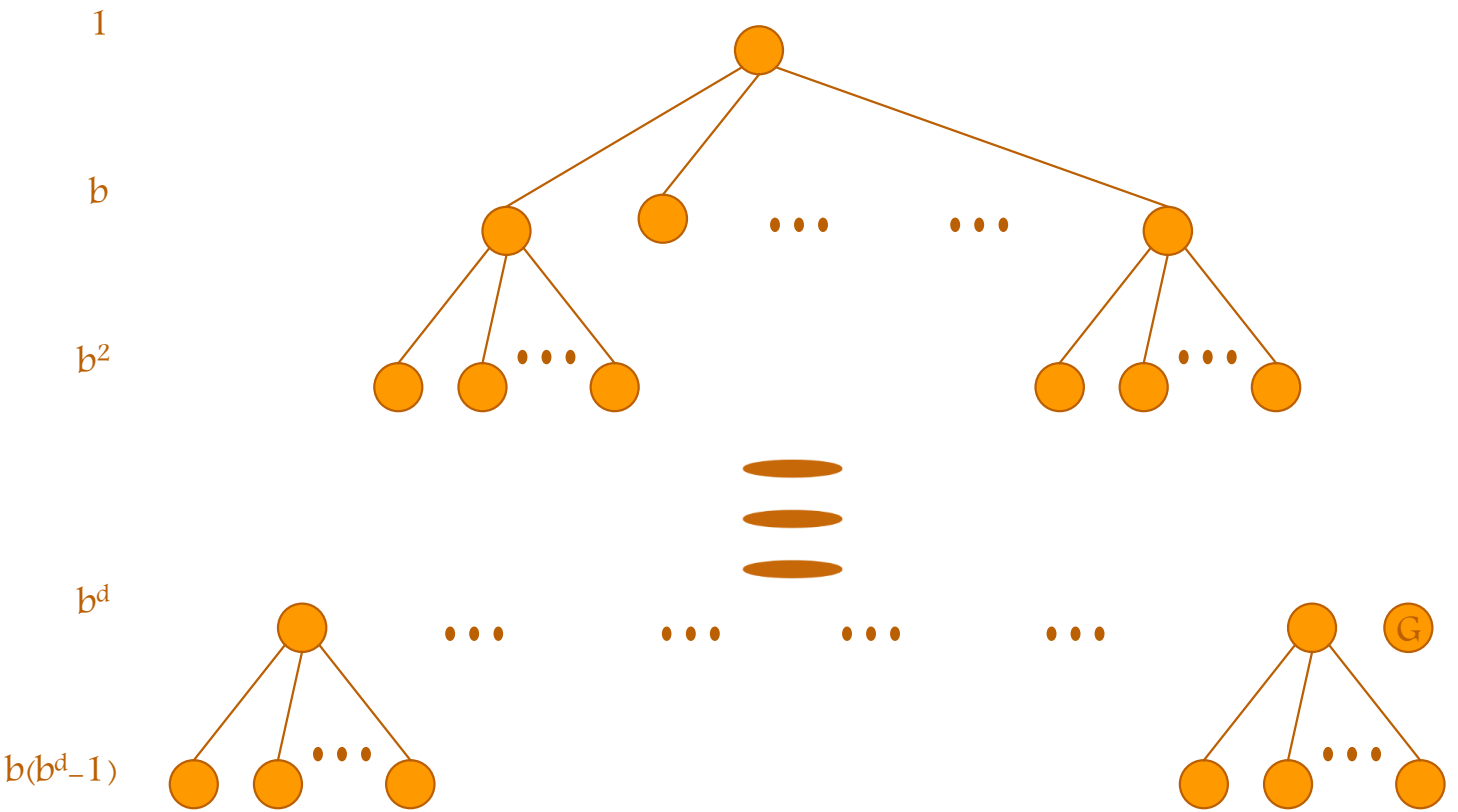
```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

قبل از قرار دهی در لیست پیشگام تست هدف انجام می شود.

فقط حالت به لیست رسیده شده اضافه می شود

جستجوی عرض نخست

- کامل
  - بله اگر b محدود باشد
- بهینه
  - بله (اگر هزینه برای هر مرحله برابر باشد)


1
b
b²
bᵈ
b(bᵈ−1)
G

جستجوی عرض نخست

- پیچیدگی زمان
  - اگر گره هنگام ایجاد تست هدف نشود
  - $1+b+b^2+b^3+\ldots+b^d+b(b^d-1)=O(b^{d+1})$
  - اگر بعد از ایجاد تست هدف شود
  - $O(b^d)$
- پیچیدگی فضا
  - اگر گره هنگام ایجاد تست هدف نشود
  - $O(b^{d+1})$
  - اگر بعد از ایجاد تست هدف شود
  - $O(b^d)$

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

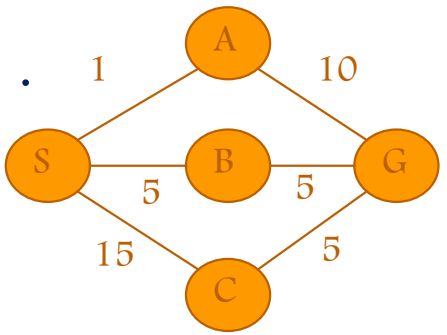
Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

جستجوی هزینه یکنواخت یا دایسترا

- رأس بسط داده نشده با کم ترین هزینه (از ریشه تا این رأس – $g(n)$) را بسط بده
- frontier صفی است که برحسب هزینه مرتب شده است.
- اگر همهٔ هزینه های مراحل مساوی باشند این الگوریتم مشابه عرض نخست است.
- در این روش، گره هنگام انتخاب از صف تست هدف می شود نه هنگام ایجاد شدن و قبل از داخل صف رفتن.
- چون ممکن است بعداً گره ای با هزینهٔ کمتر یافت شود.

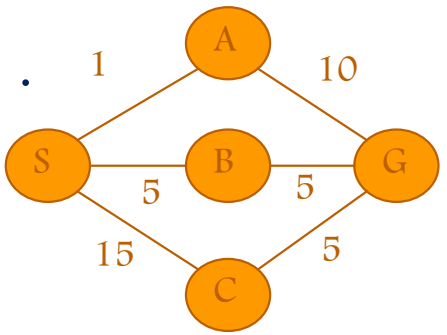
جستجوی هزینه یکنواخت

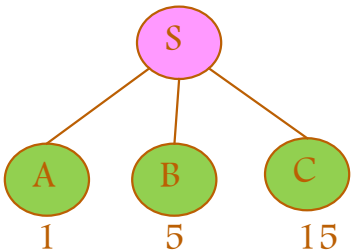
- .



جستجوی هزینه یکنواخت

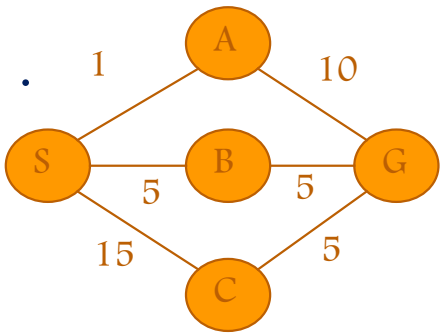
- .

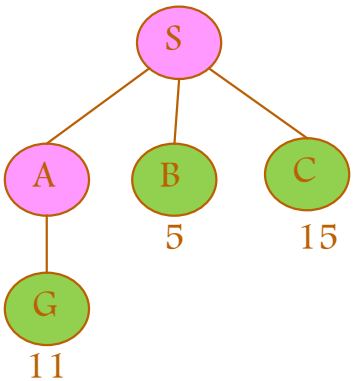




جستجوی هزینه یکنواخت

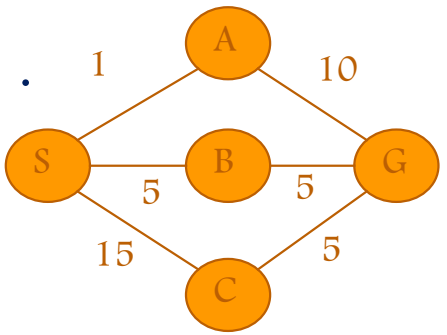
- .

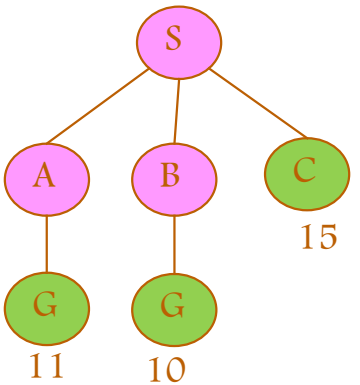




جستجوی هزینه یکنواخت

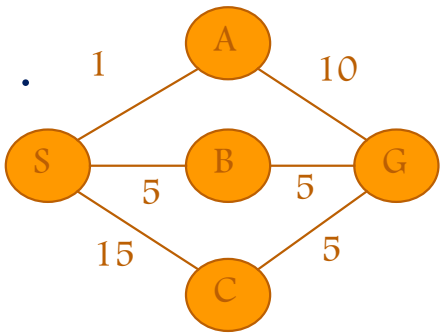
- .

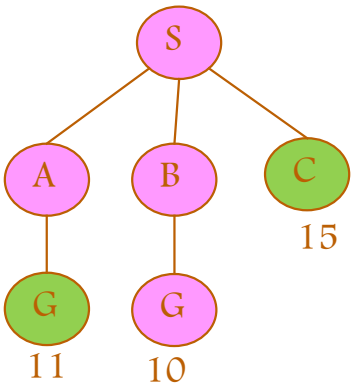




جستجوی هزینه یکنواخت

- .





جستجوی هزینه یکنواخت

- .

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, PATH-COST)

جستجوی هزینه یکنواخت

- کامل؟
  - بله اگر هزینهٔ هر مرحله بزرگتر یا مساوی ε باشد.
  - اگر هزینه صفر باشد در حلقهٔ بی نهایت قرار می گیرد.
- زمان:
- چون با هزینهٔ مسیر هدایت می شود نه با عمق تحلیل مقداری پیچیده
- در بدترین حالت:
  - تعداد رئوس با g هزینهٔ حل بهینه (هزینهٔ هر مرحله حداقل ε و C^* هزینهٔ حل بهینه)

$$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$$

جستجوی هزینه یکنواخت

- فضا: تعداد رئوس با g کوچکتر یا مساوی با هزینهٔ حل بهینه

$$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$$

- بهینه: در صورت افزایشی بودن هزینهٔ مراحل بله
- در صورتی که هزینهٔ مراحل یکسان باشد جستجوی هزینه یکنواخت همانند جستجوی عرض نخست است.

خلاصه

- جستجو برای حل
- جستجوی گرافی و درختی
- کارآئی الگوریتمهای جستجو
- جستجوی عرض نخست
- جستجوی هزینه یکنواخت


والسلام