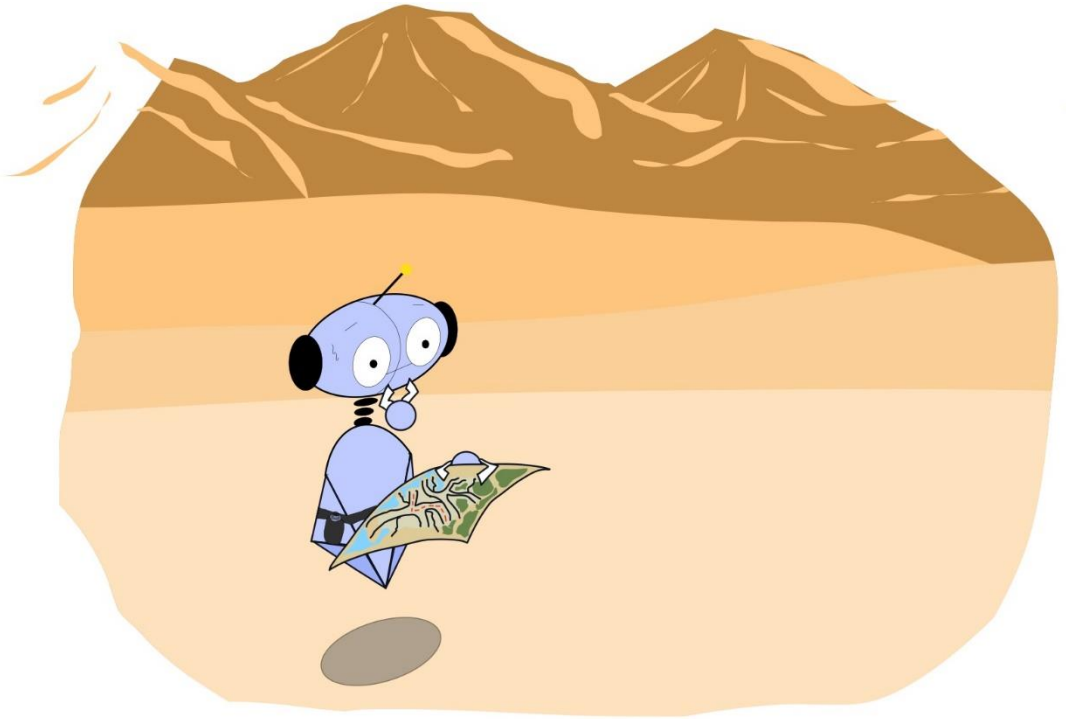
مبانی و کاربردهای هوش مصنوعی

جستجو ناآگاهانه (فصل 3.1 الی 3.4)

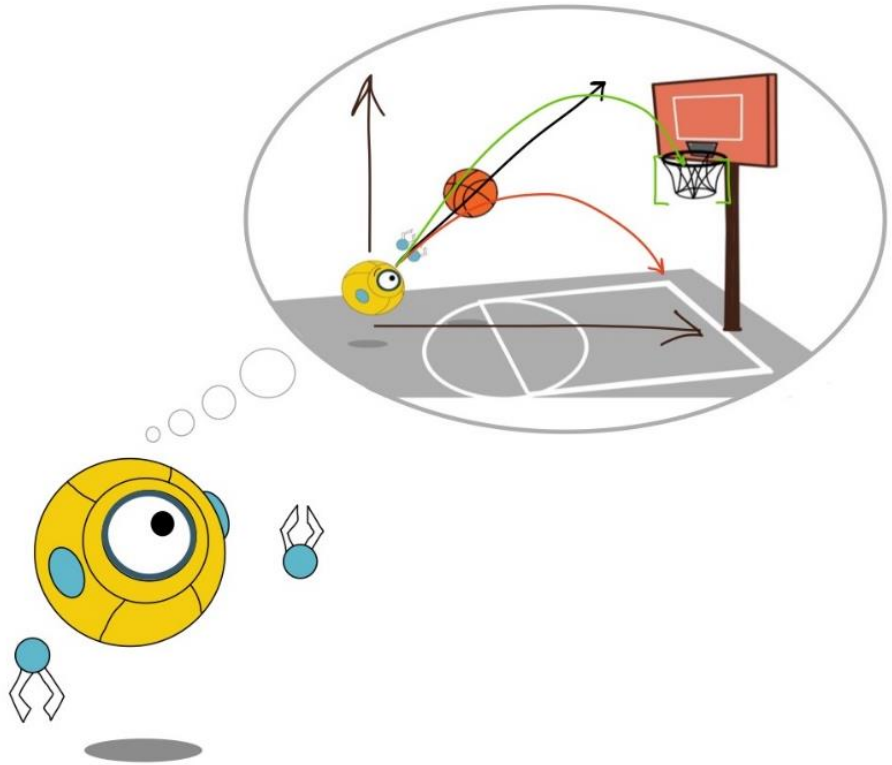


مدرس: مهدی جوانمردی

دانشگاه صنعتی امیرکبیر

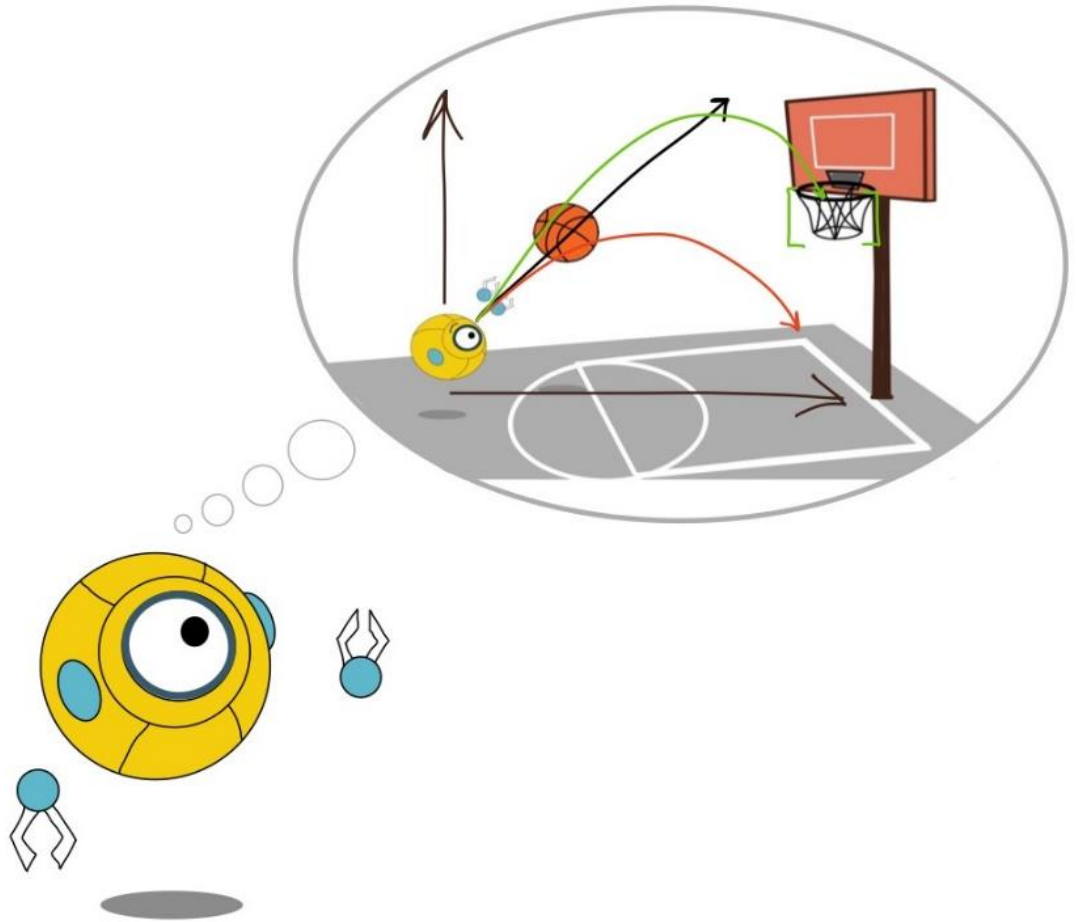
الهام گرفته از محتوای درس هوش مصنوعی دانشگاه برکلی

رئوس مطالب



- عامل‌هایی که برای آینده برنامه‌ریزی می‌کنند
- مسائل جستجو
- روش‌های جستجوی ناآگاهانه
  - جستجوی اول عمق (Depth-first search)
  - جستجوی اول سطح (Breadth-first search)
  - جستجوی هزینه یکنواخت (Uniform-cost search)

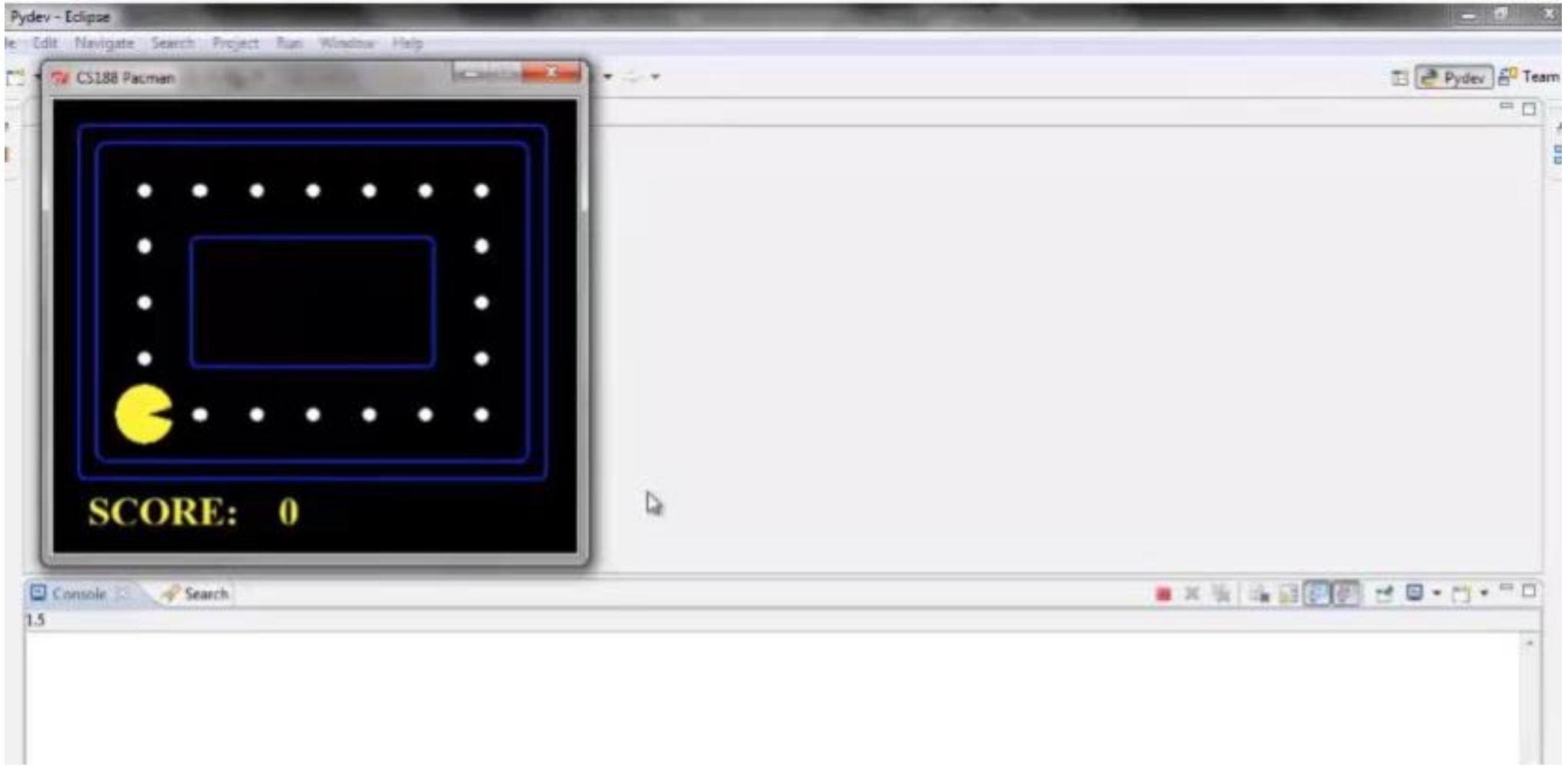
عامل‌هایی که برنامه‌ریزی می‌کنند



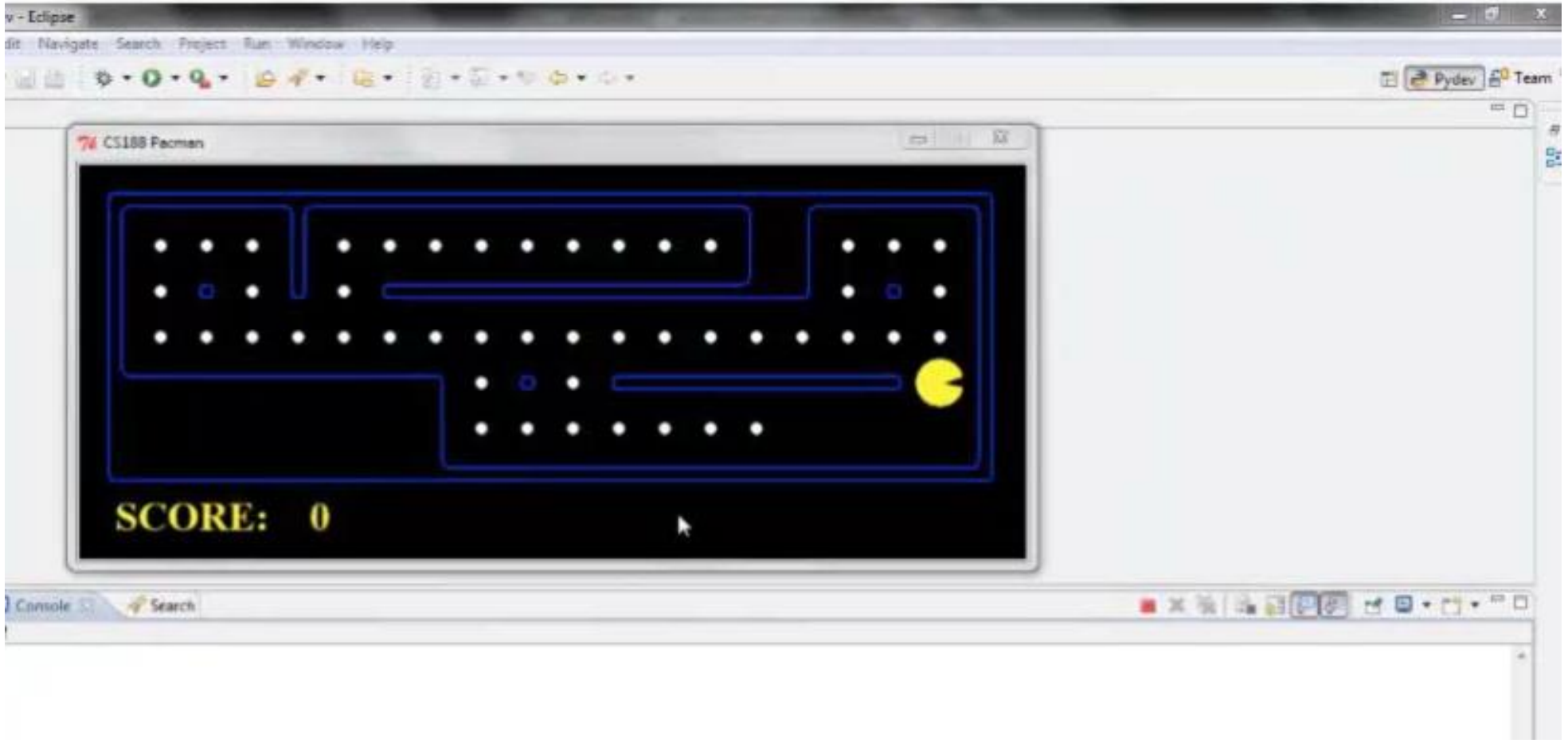
عامل‌های واکنشی

- عامل‌های واکنشی
  - انتخاب عمل بر اساس ادراک فعلی (و شاید حافظه)
  - ممکن است حافظه داشته و یا مدلی از وضعیت فعلی جهان را نگهداری کند
  - عواقب آتی اقداماتش را در نظر نمی‌گیرد
  - در نظر می‌گیرد که جهان چگونه «هست»
- آیا یک عامل واکنشی می‌تواند عقلانی (منطقی) باشد؟

ویدیوی دموی واکنشی بهینه



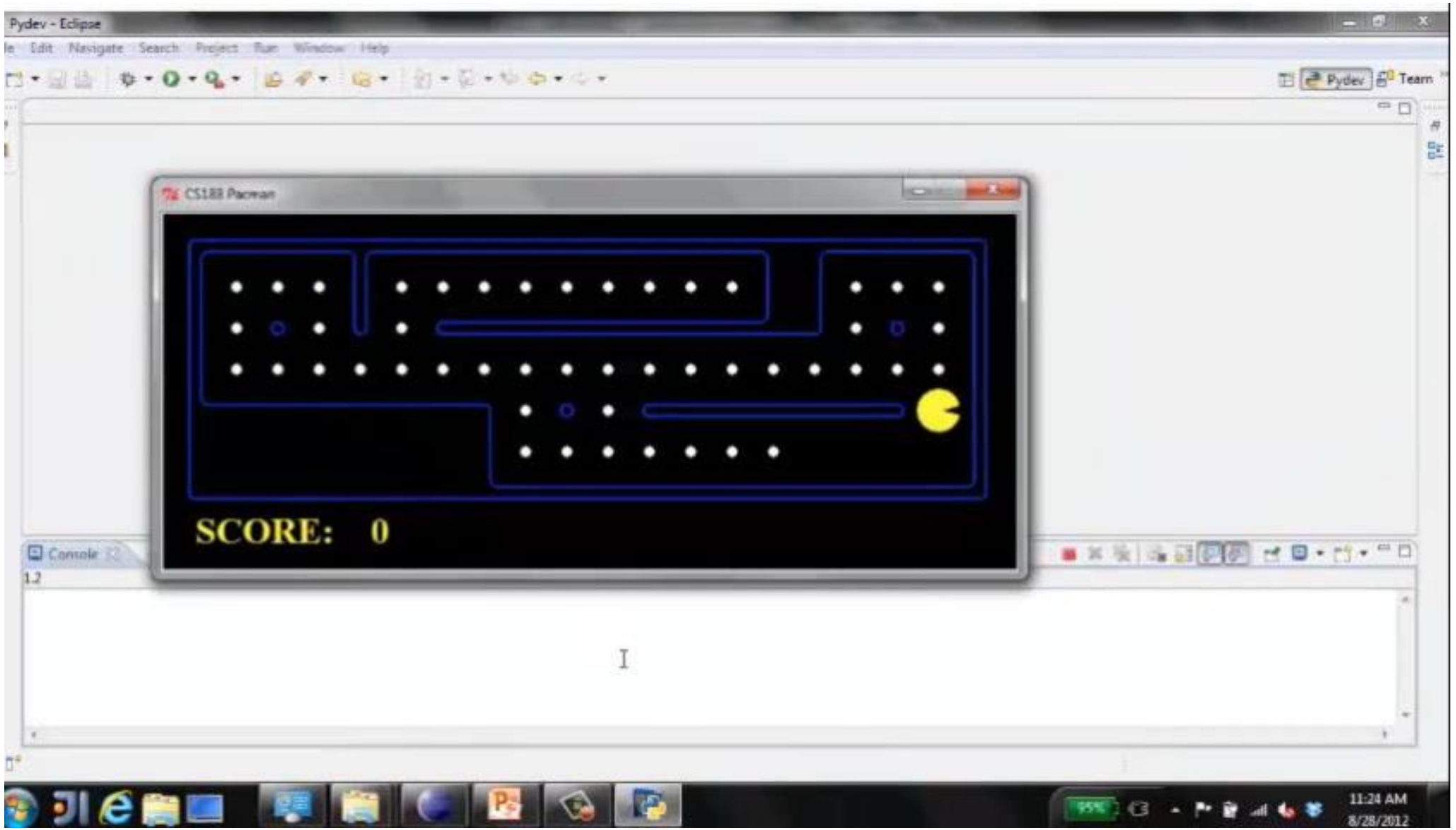
ویدیوی دموی واکنشی ناموفق



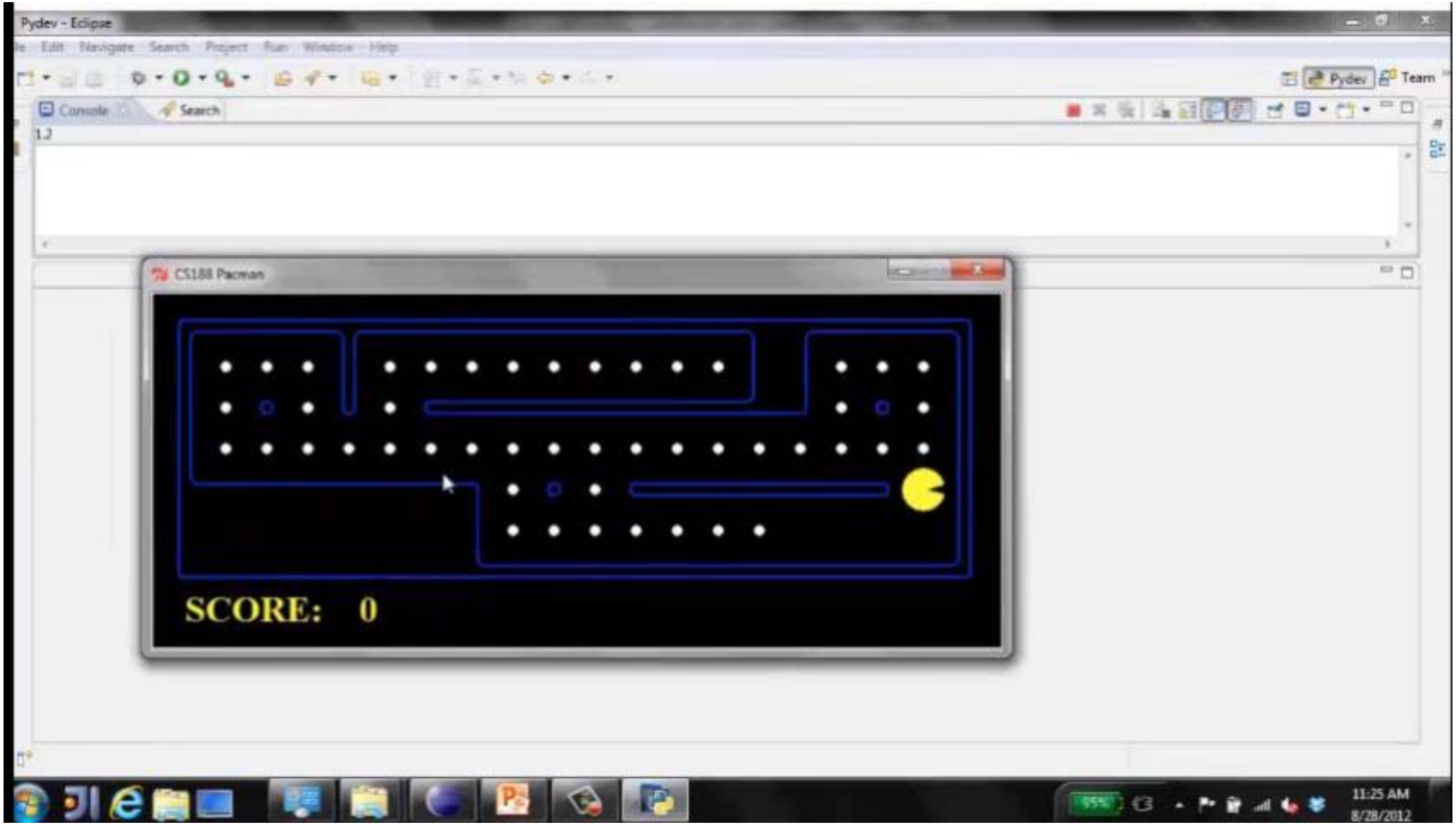
عامل‌های برنامه‌ریز

- عامل‌های برنامه‌ریز
  - می‌پرسد "چه خواهد شد اگر (What if)"
  - تصمیم گیری بر اساس پیامدهای (فرضی) اقدامات
  - باید مدلی از چگونگی تکامل جهان در واکنش به اعمال داشته باشد
  - باید «هدف» را بصورت صریح فرموله کند (آزمون)
  - در نظر می‌گیرد که جهان چگونه «چگونه خواهد بود» اگر...
- برنامه‌ریزی بهینه (optimal) در مقابل برنامه ریزی کامل (complete)
- برنامه‌ریزی (planning) در مقابل برنامه ریزی مجدد (replanning)

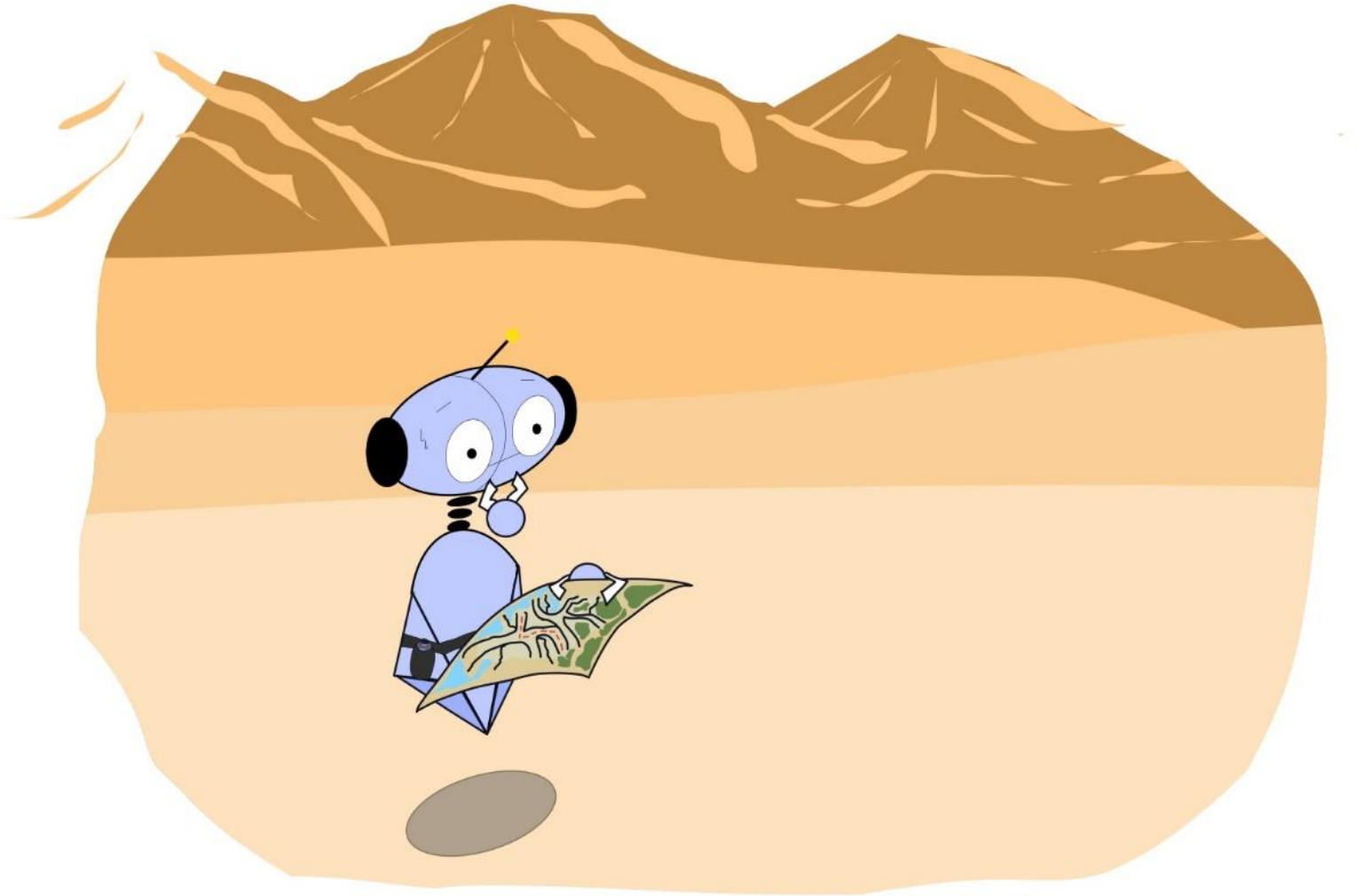
ویدیوی دموی پک‌من برنامه‌ریزی مجدد



ویدیوی دموی پک‌من مغز متفکر



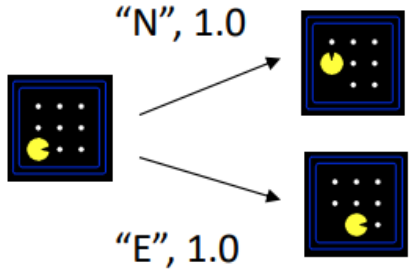
مسئله جستجو



مسائل جستجو (Search problems)

- یک مساله‌ی جستجو شامل موارد زیر است:
  - فضای حالت (state space)

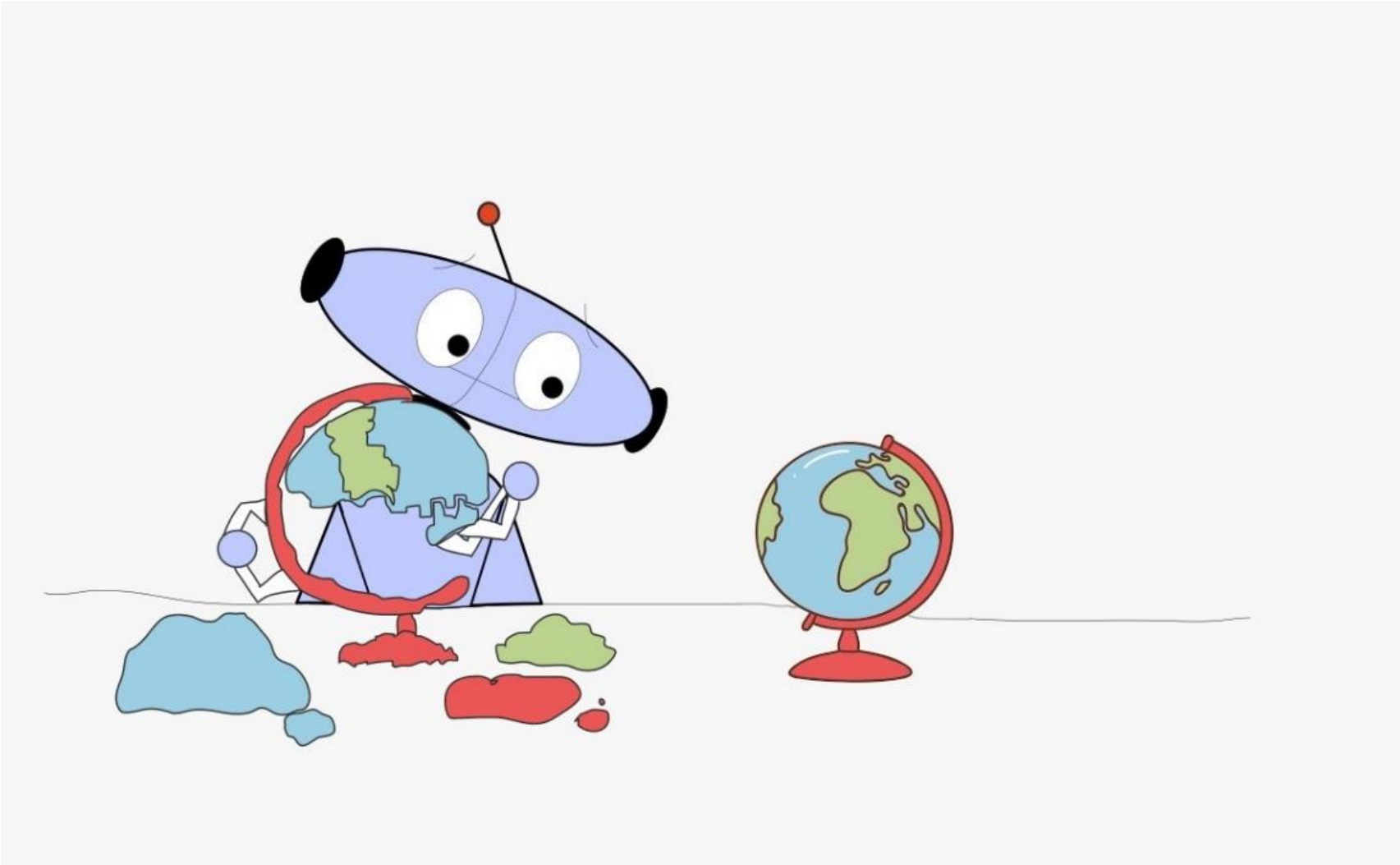
  - تابع پسین (Successor function)، (با اقدامات و هزینه‌ها)



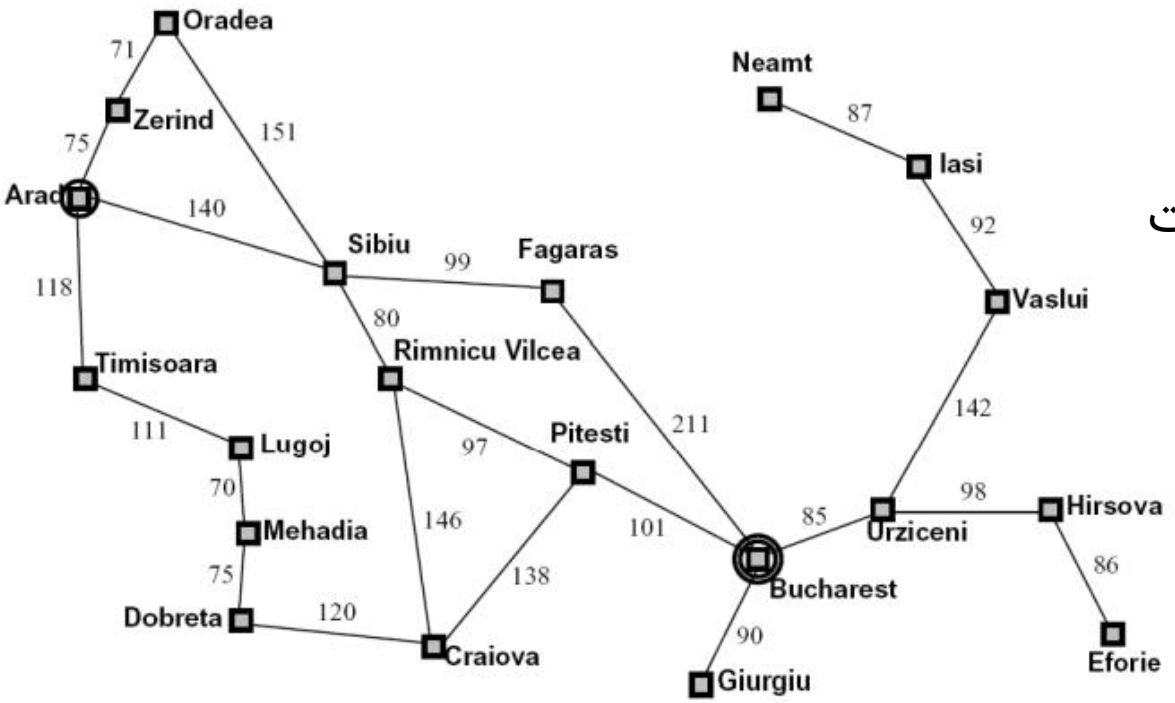
  - حالت شروع و آزمون هدف

- یک راه حل (solution) مجموعه‌ای از اقدامات (برنامه‌ریزی) است که حالت شروع را به حالت هدف تبدیل می‌کند

مسائل جستجو مدل هستند



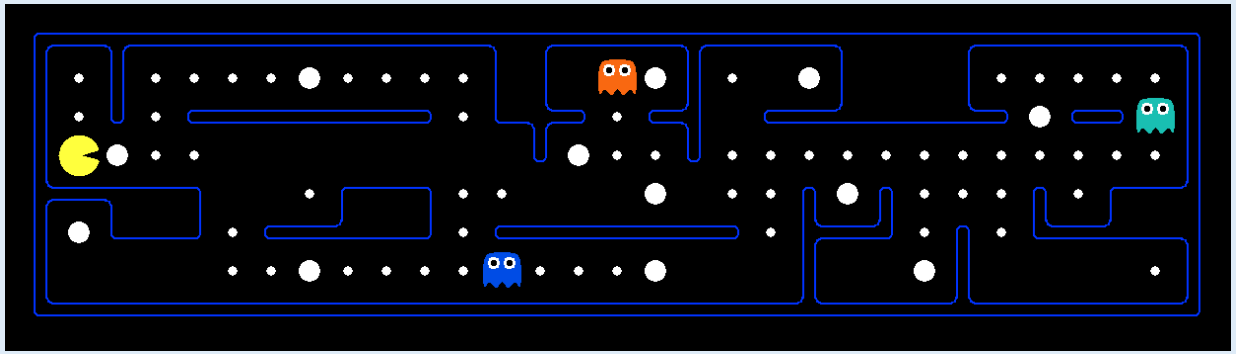
مثال: مسافرت در رومانی



- فضای حالت
  - شهرها
- تابع پسین
  - جاده‌ها: به شهر مجاور بروید با هزینه = مسافت
- حالت شروع
  - شهر آراد
- تست هدف
  - آیا حالت == بخارست است؟
- راه حل؟

فضای حالت چیست؟

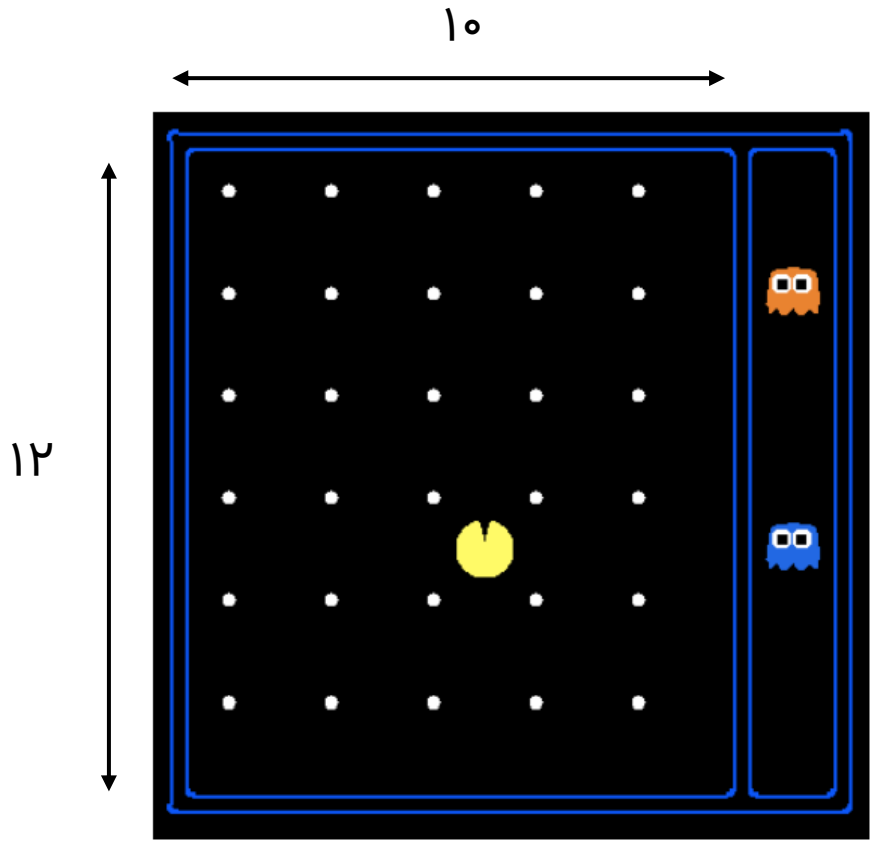
حالت جهان (world state) شامل همه‌ی جزئیات محیط است



حالت جستجو (search state) فقط جزئیات مورد نیاز برای برنامه‌ریزی را نگه می‌دارد (انتزاع)

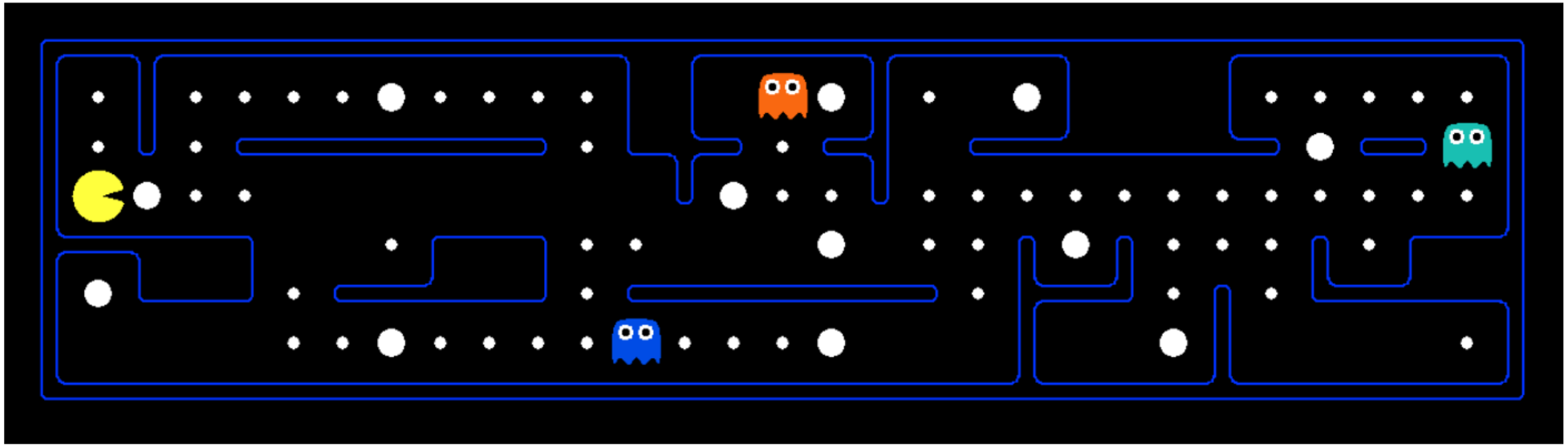
- مساله: مسیریابی
  - حالات: مکان(X, Y)
  - اقدامات: شمال، جنوب، شرق، غرب
  - تابع پسین: فقط مکان به روزرسانی شود
  - تست هدف: پایان = (x,y) باشد
- مساله: خوردن همه نقطه‌ها
  - حالات: {(X, Y)، حالت بولی نقطه‌ها}
  - اقدامات: شمال، جنوب، شرق، غرب
  - تابع پسین: بروزرسانی مکان و شاید بولی یک نقطه
  - تست هدف: false بودن بولی همه نقاط

اندازه‌ی فضای حالت



- حالت جهان:
  - مکان‌های عامل: ۱۲۰
  - تعداد غذا: ۳۰
  - مکان روح‌ها: ۱۲
  - جهت عامل: شمال، جنوب، شرق، غرب
- چه تعداد حالت؟
  - حالت جهان؟ ۴ × ۱۲ × ۲ × ۳۰ = ۱۲۰ × ۲
  - حالت‌های مسیریابی؟ ۱۲۰
  - حالت‌های خوردن همه نقطه‌ها؟ ۱۲۰ × ۲ × ۳۰

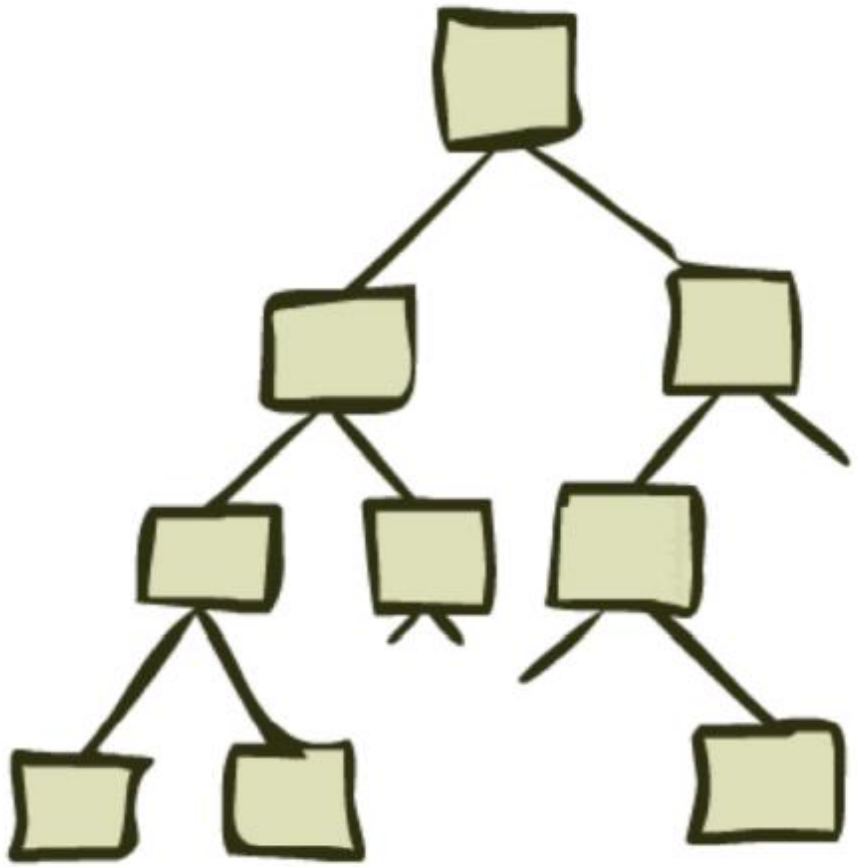
آزمونک: گذرگاه امن



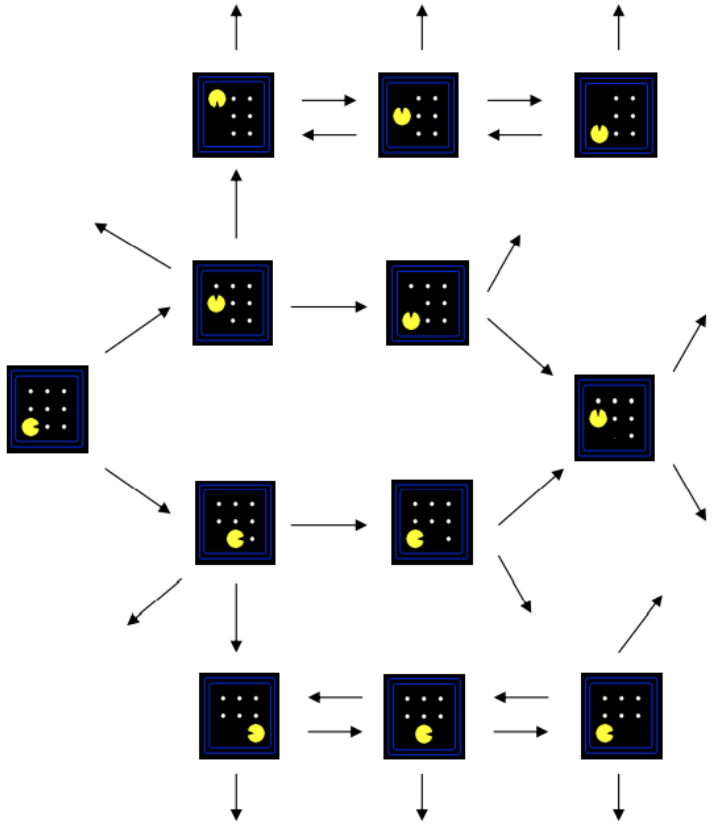
- مساله: تمام نقاط را بخورید در حالی که ارواح را همیشه ترسانده‌اید.
- فضای حالت باید چه چیزی را مشخص کند؟

(موقعیت عامل، بولی‌های نقطه‌ای، بولی‌های قرص قدرت، زمان ترس باقیمانده)

گراف فضای حالات و درخت جستجو



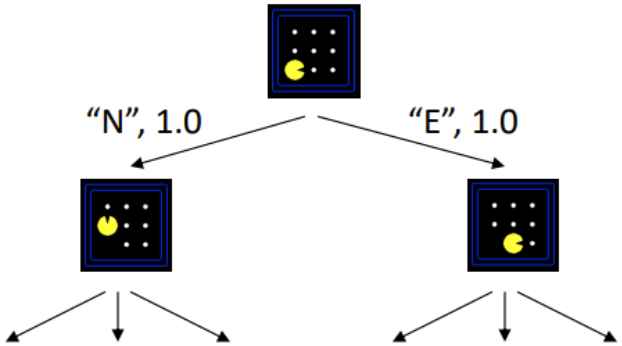
گراف فضای حالت



- گراف فضای حالت: نمایش ریاضیاتی یک مسئله جستجو
  - گره‌ها پیکربندی جهان را نشان میدهند (خلاصه‌شده)
  - یال‌ها نشان دهنده پسین‌ها هستند (نتایج اعمال)
  - آزمون هدف مجموعه ای از گره های هدف است (شاید فقط یک گره باشد)

- در گراف فضای حالت، هر حالت فقط یک بار رخ می‌دهد!
- ما به ندرت می‌توانیم این نمودار را کامل در حافظه بسازیم (خیلی بزرگ است)، اما این یک ایده قابل استفاده است.

درخت‌های جستجو

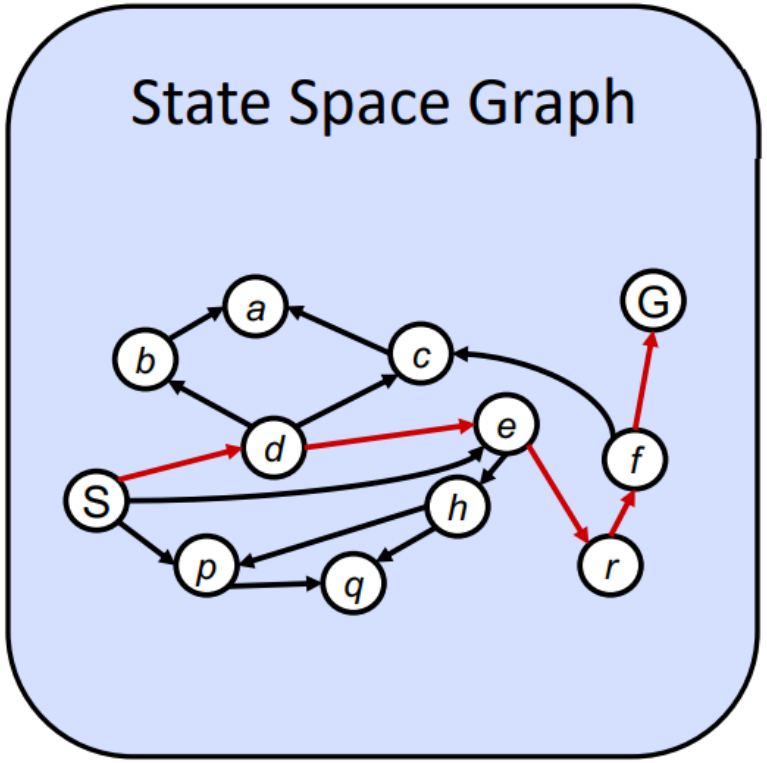


این حالت کنونی / شروع است.

آینده‌های محتمل

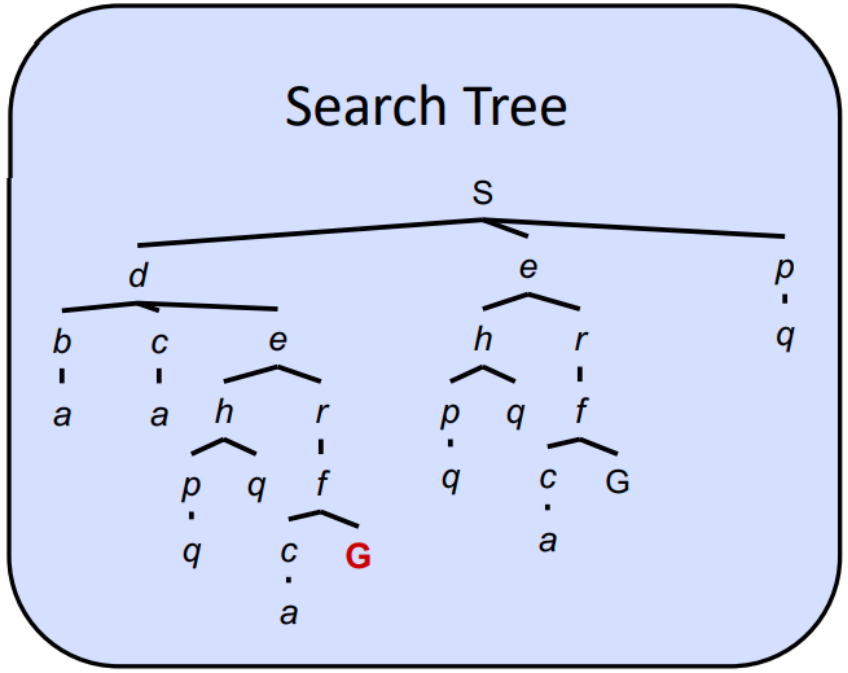
- درخت جستجو:
  - درخت «چه می شد اگر(what if)» از برنامه‌ها و نتایج آن‌ها
  - حالت شروع گره ریشه است
  - فرزندان نشان دهنده پسین‌ها
  - گره‌ها حالت‌ها را نشان می‌دهند، اما مطابق با **برنامه**هایی هستند که به آن حالت‌ها دست می‌یابند
  - برای اکثر مسائل، ما هرگز نمی‌توانیم کل درخت را بسازیم (حتی از گراف جستجو هم بزرگ‌تر است!)

گراف فضای حالت در مقابل درخت‌های جستجو



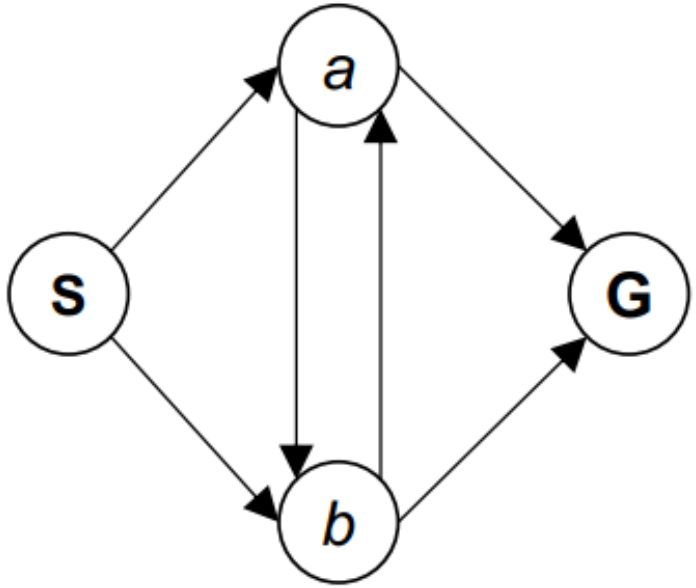
هر گره در درخت جستجو یک مسیر کامل در نمودار فضای حالت است.

ما هر دو را بنا به تقاضا– و تا حد امکان کوچکتر می‌سازیم.



آزمونک: گراف فضای حالت در مقابل درخت‌های جستجو

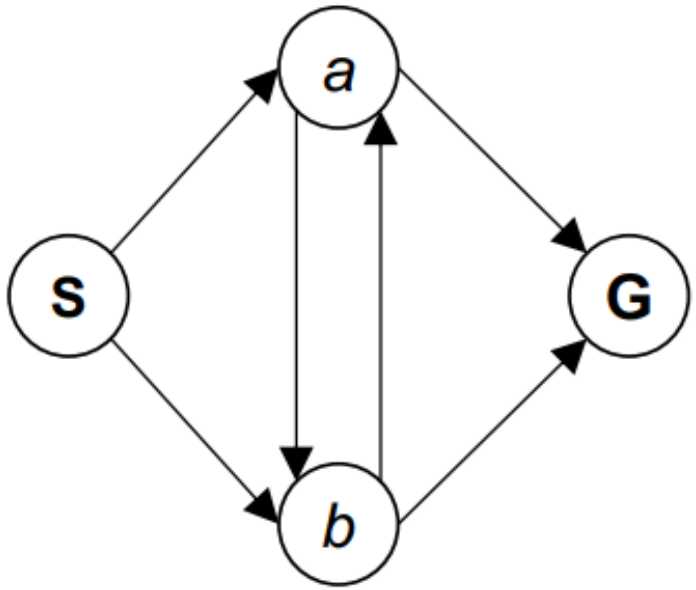
این گراف 4 حالته را در نظر بگیرید.



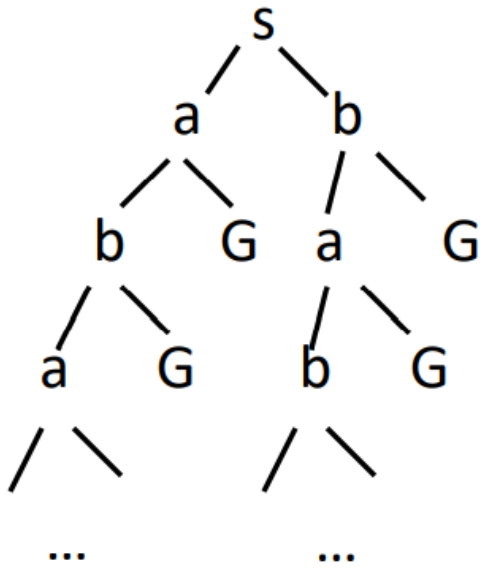
اندازه درخت جستجو چقدر است؟
(با فرض شروع از حالت S)

آزمونک گراف فضای حالت در مقابل درخت‌های جستجو

این گراف 4 حالته را در نظر بگیرید.

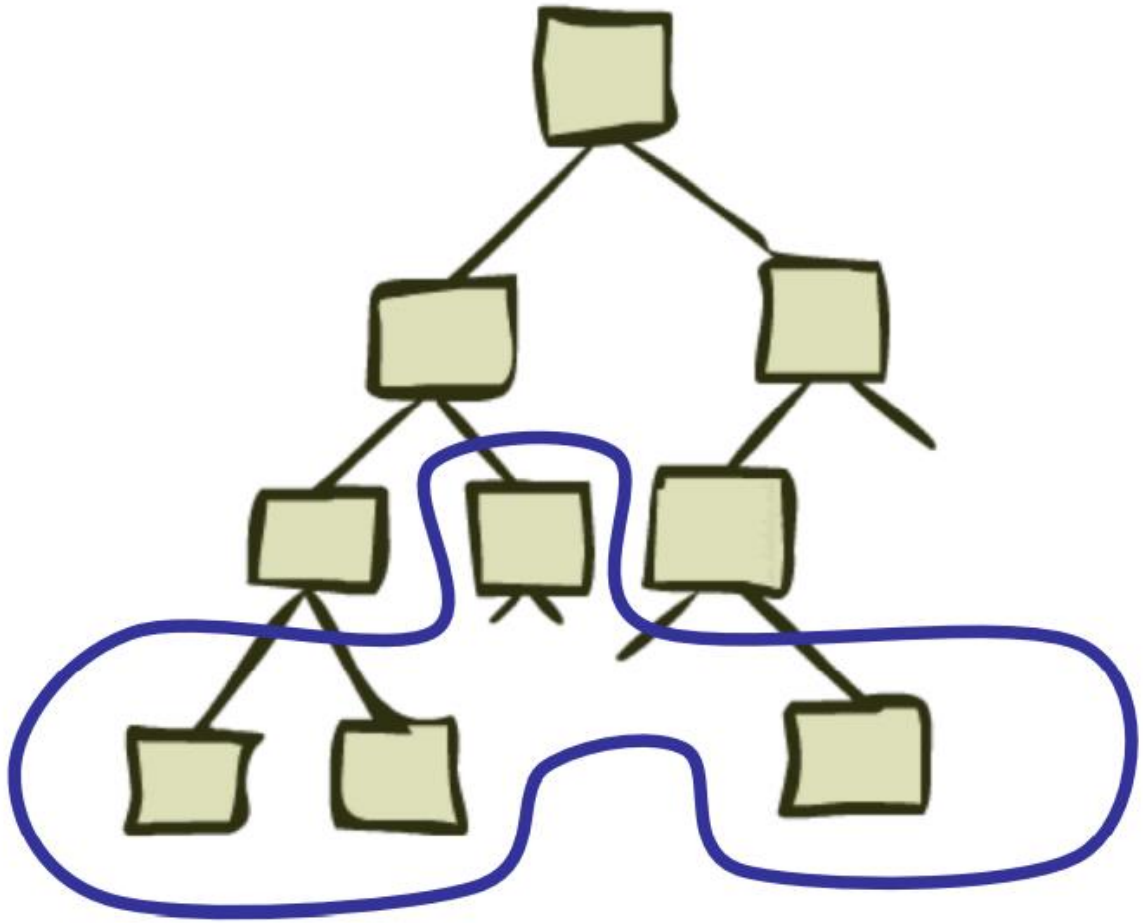


اندازه درخت جستجو چقدر است؟
(با فرض شروع از حالت S)

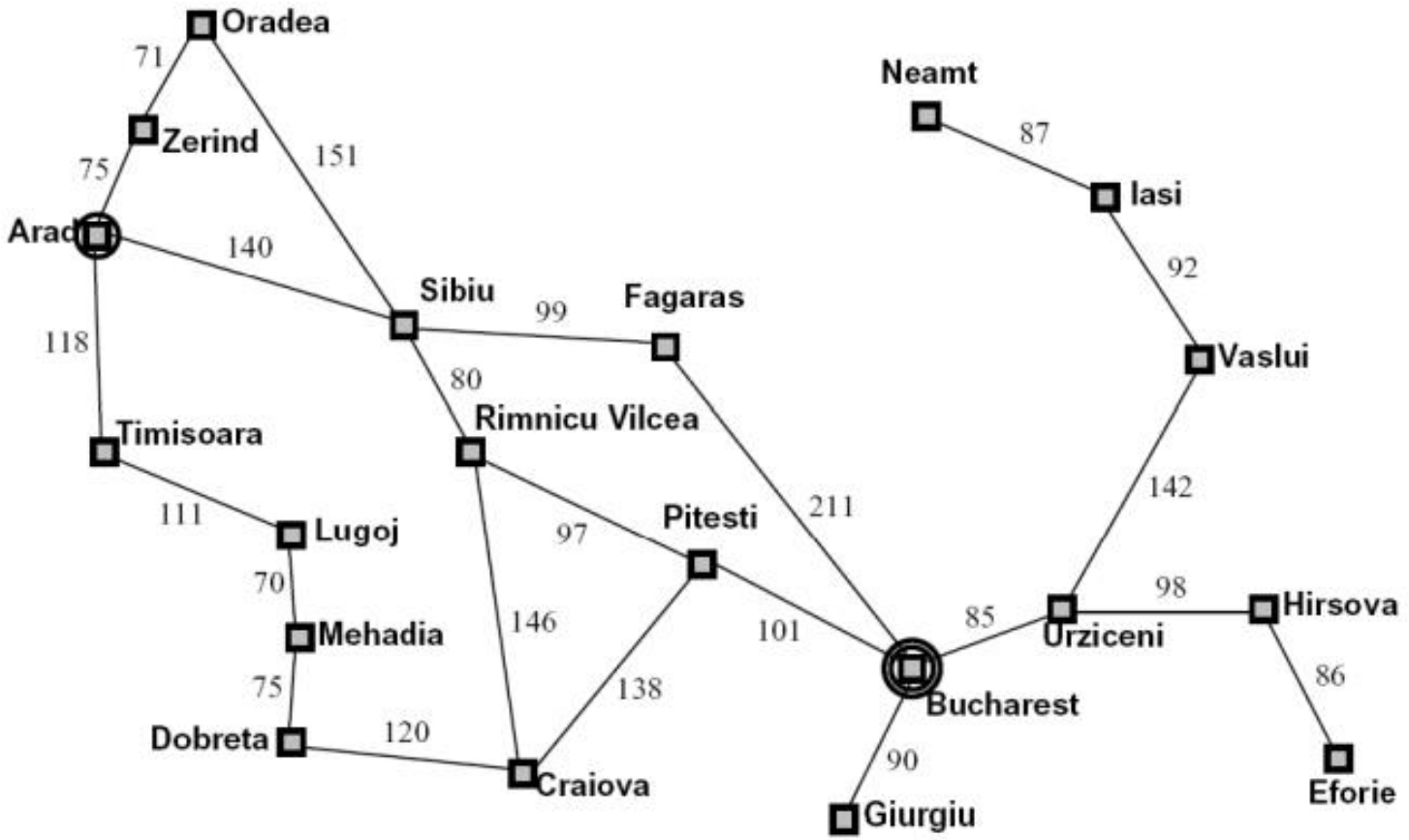


نکته مهم: تعداد زیادی ساختار تکراری در درخت جستجو!

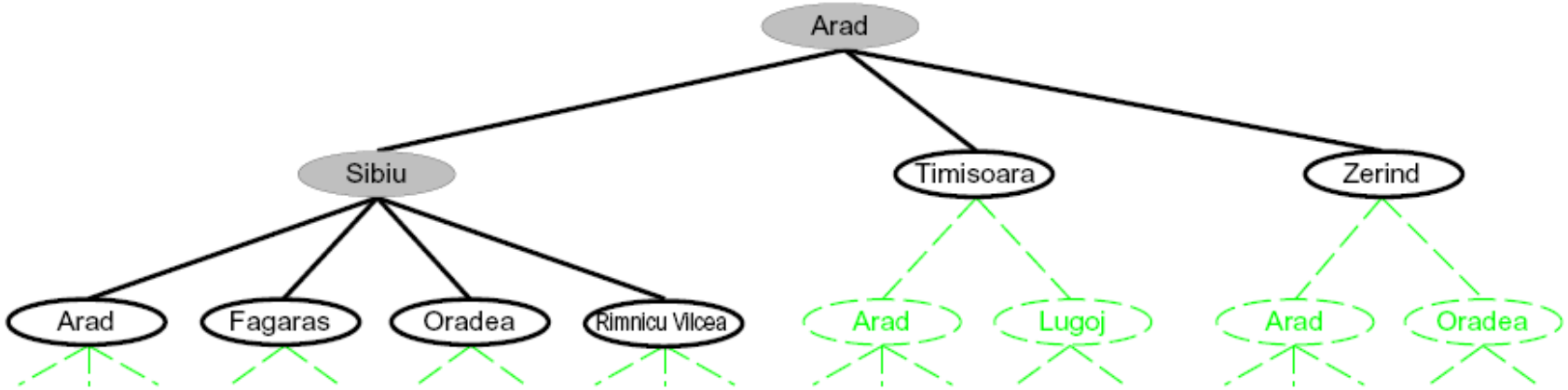
جستجوی درختی



مثال جستجو: رومانی



جستجو با درخت جستجو



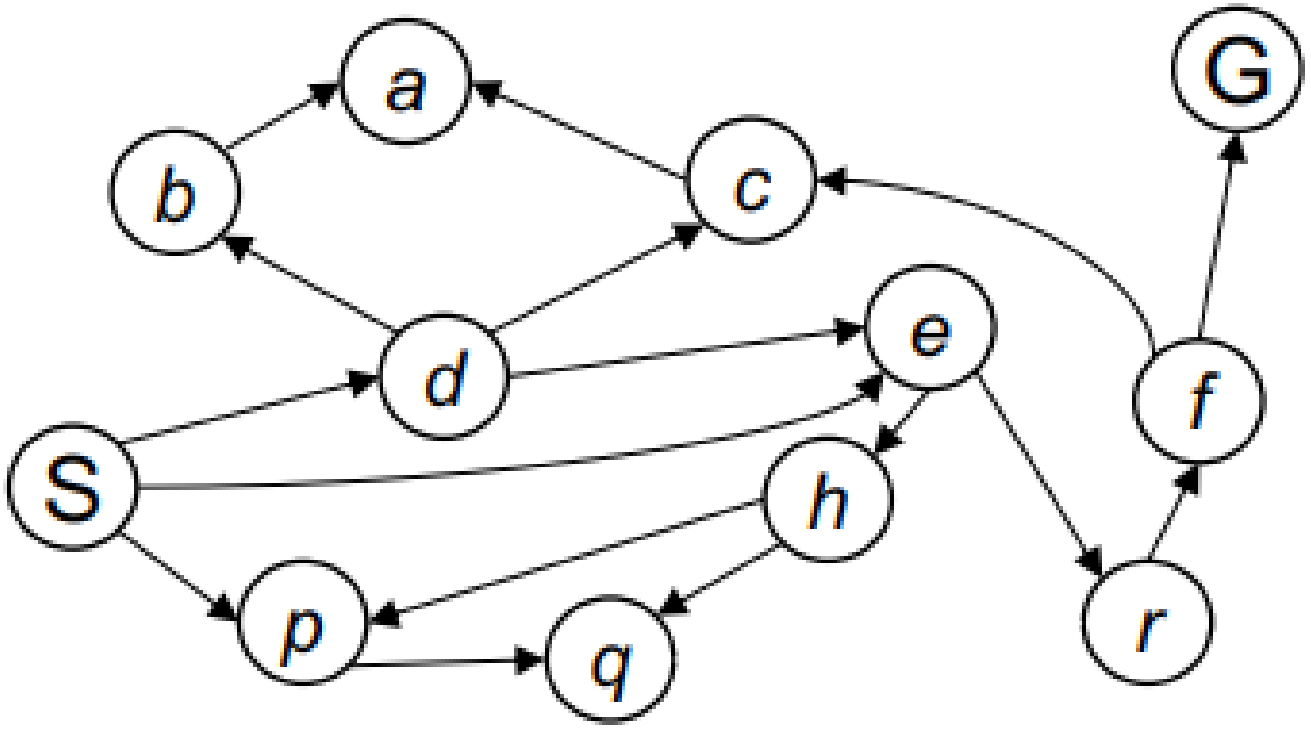
- جستجو:
  - بسط برنامه‌ریزی‌های بالقوه (گره‌های درخت)
  - لبه‌ای (fringe) از برنامه‌ریزی‌های جزئی در دست بررسی را در حافظه نگهدارید
  - سعی کنید تا حد امکان تعداد کمتری از گره‌های درختی را بسط دهید

جستجو با درخت جستجو

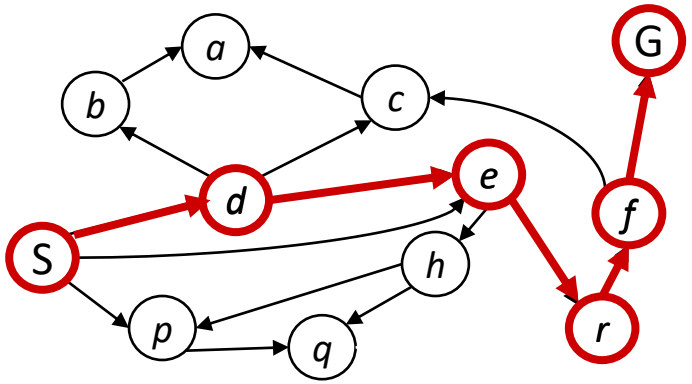
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

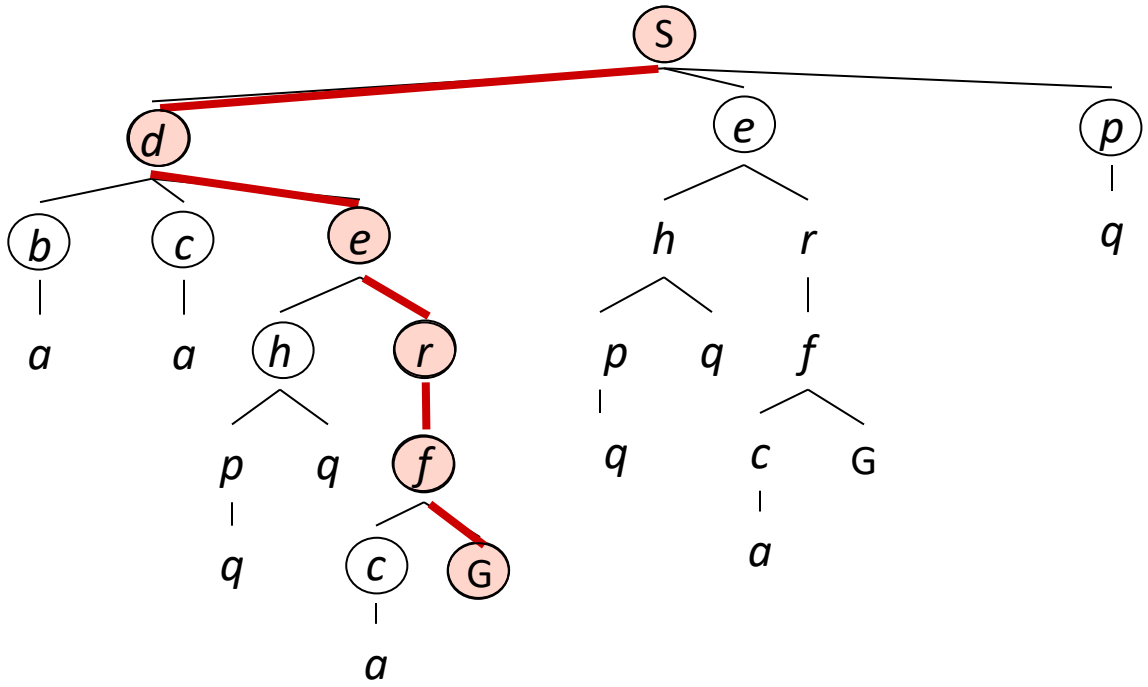
- ایده‌های مهم:
  - لبه
  - بسط دهی (expansion)
  - استراتژی اکتشاف (exploration)
- سوال اصلی: کدام گره‌های لبه را کاوش (explore) کنیم؟

مثال: جستجو درختی



مثال: جستجو درختی





~~s~~

~~s → d~~

s → e

s → p

s → d → b

s → d → c

~~s → d → e~~

s → d → e → h

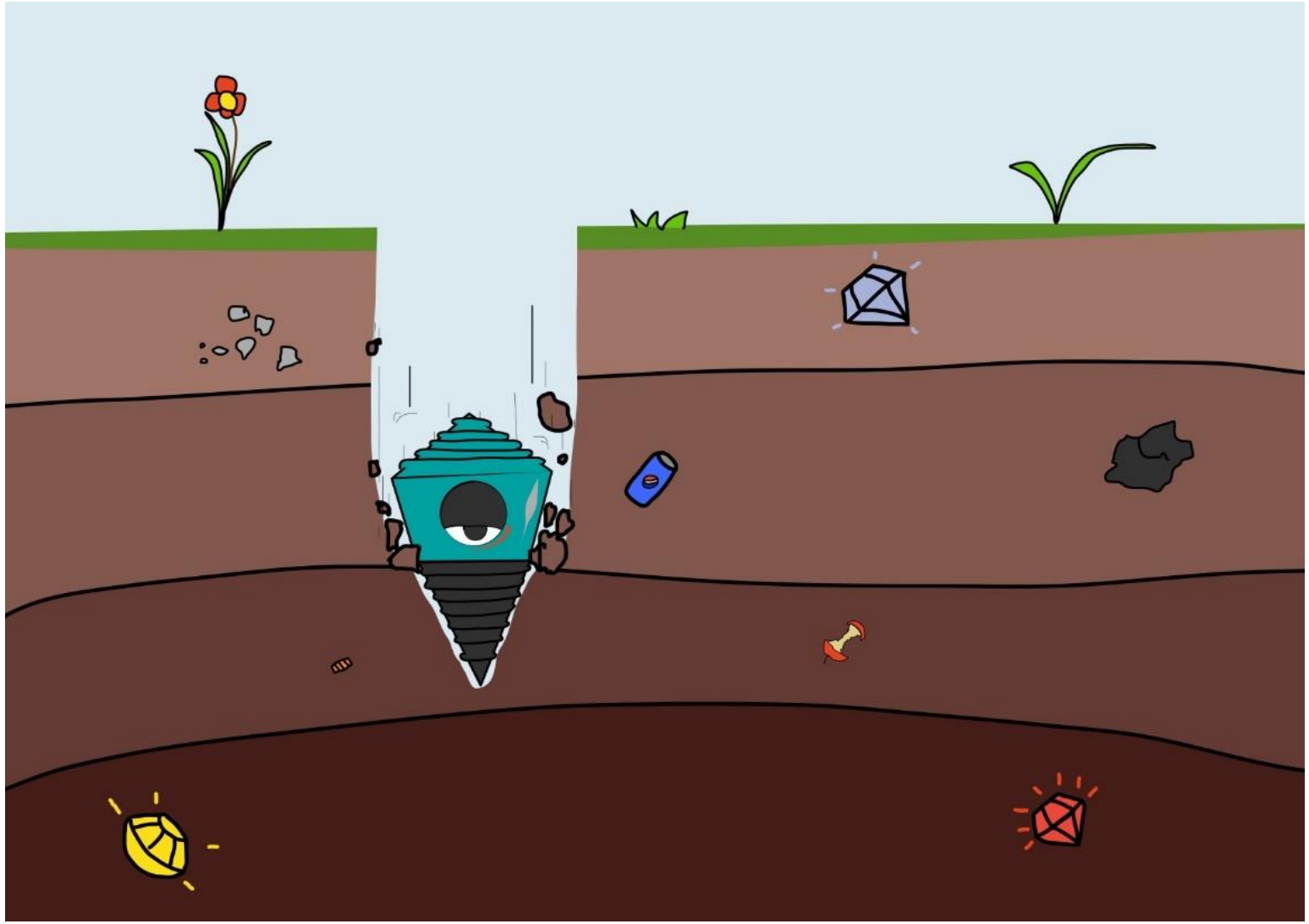
~~s → d → e → r~~

~~s → d → e → r → f~~

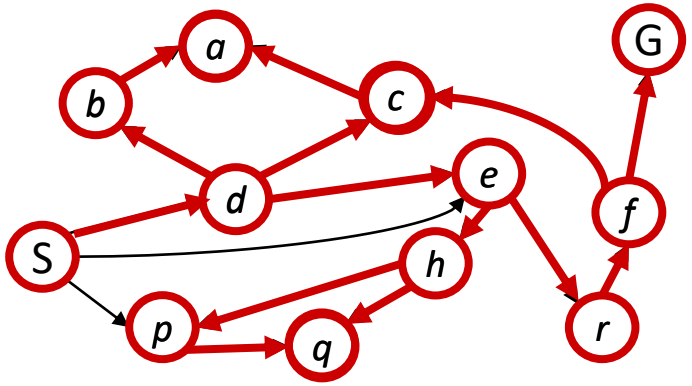
s → d → e → r → f → c

~~s → d → e → r → f → G~~

جستجوی اول عمق (DFS)

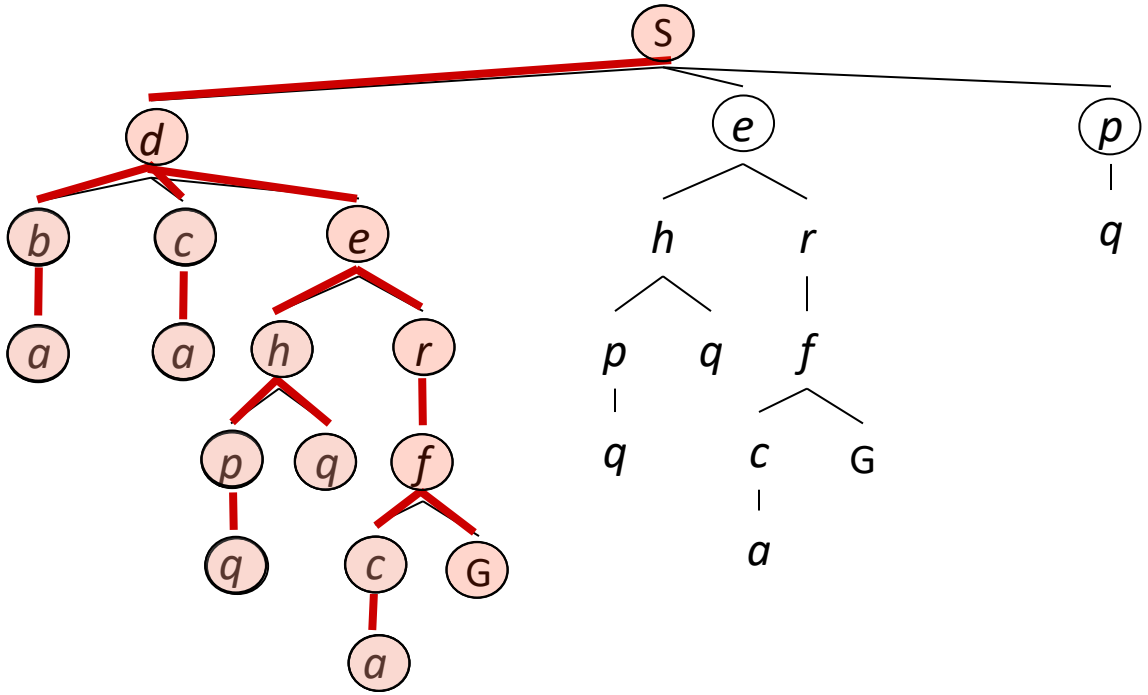


جستجوی اول عمق (DFS)

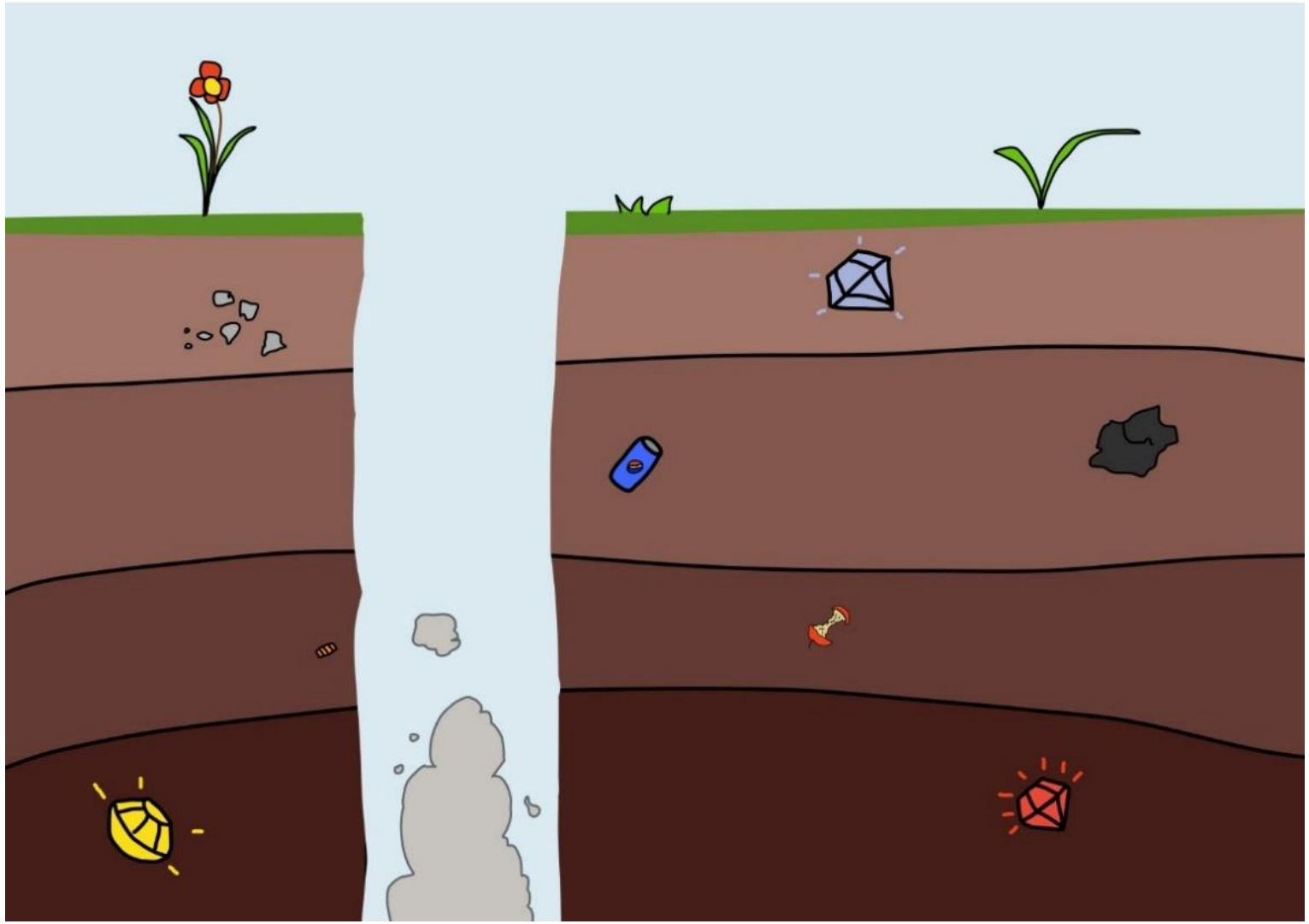


استراتژی:
ابتدا عمیق‌ترین گره را گسترش دهید.

پیاده سازی:
Fringe یک پشته LIFO است.

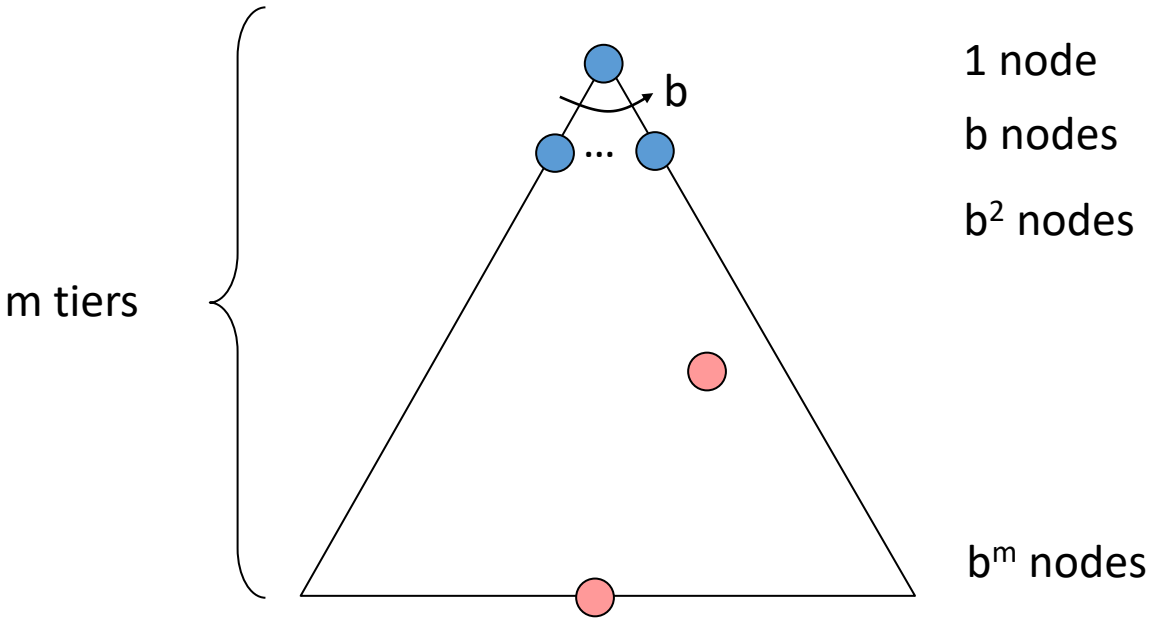


ویژگی‌های الگوریتم جستجو



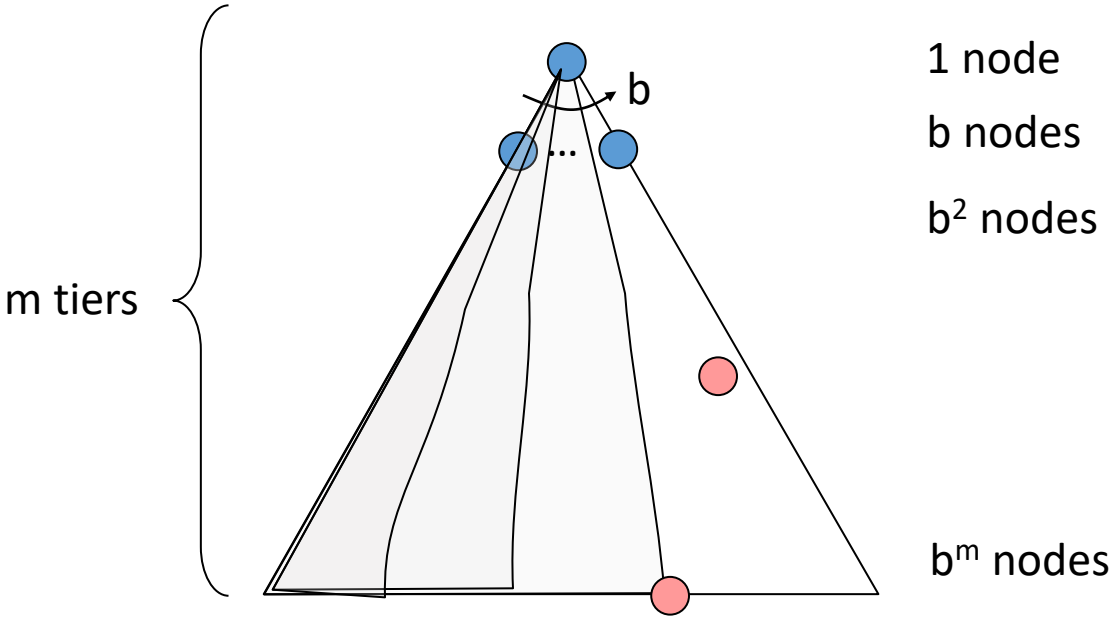
ویژگی‌های الگوریتم جستجو

- کامل بودن (complete): تضمین می‌کند که اگر راه‌حلی وجود داشته باشد آن‌را پیدا کند.
- بهینه (optimal): تضمین می‌کند که مسیر با کمترین هزینه را پیدا کند.
- پیچیدگی زمانی؟
- پیچیدگی فضا؟
- کاریکاتور درخت جستجو:
  - b ضریب انشعاب
  - m حداکثر عمق
  - راه حل ها در اعماق مختلف



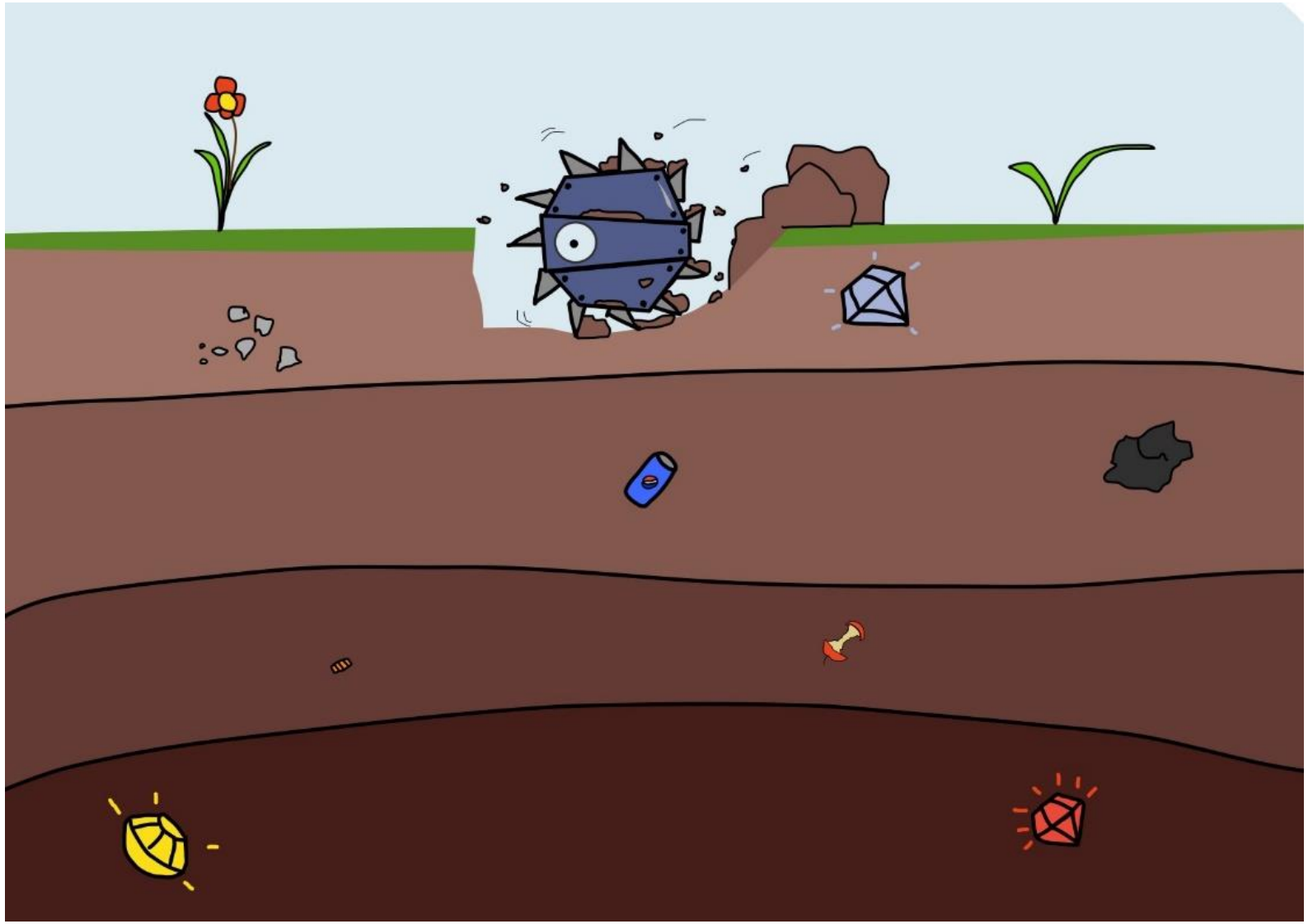
- تعداد گره‌ها در کل درخت:
  - $1 + b + b^2 + ... b^m = O(b^m)$

ویژگی‌های جستجوی اول عمق (DFS)

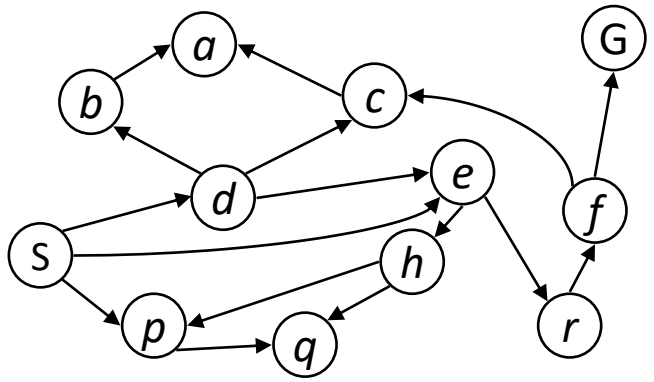


- DFS چه گره‌هایی را بسط می‌دهد؟
  - برخی از پیشوندهای سمت چپ درخت
  - می‌تواند کل درخت را پیمایش کند (به شرط عمق محدود)
  - اگر m متناهی باشد، جستجو $O(b^m)$ زمان می‌برد
- لبه چه اندازه فضا اشغال می‌کند؟
  - فقط گره‌های خواهر و برادر (siblings) در مسیر ریشه‌ی درخت را شامل می‌شود، بنابراین O(bm)
- آیا کامل است؟
  - m می‌تواند بینهایت باشد، بنابراین فقط در صورتی که از دورها جلوگیری کنیم (در ادامه بیشتر توضیح داده می‌شود)
- آیا بهینه است؟
  - خیر، بدون در نظر گرفتن عمق یا هزینه، "سمت چپ ترین" راه حل را پیدا می‌کند

جستجوی اول سطح (BFS)

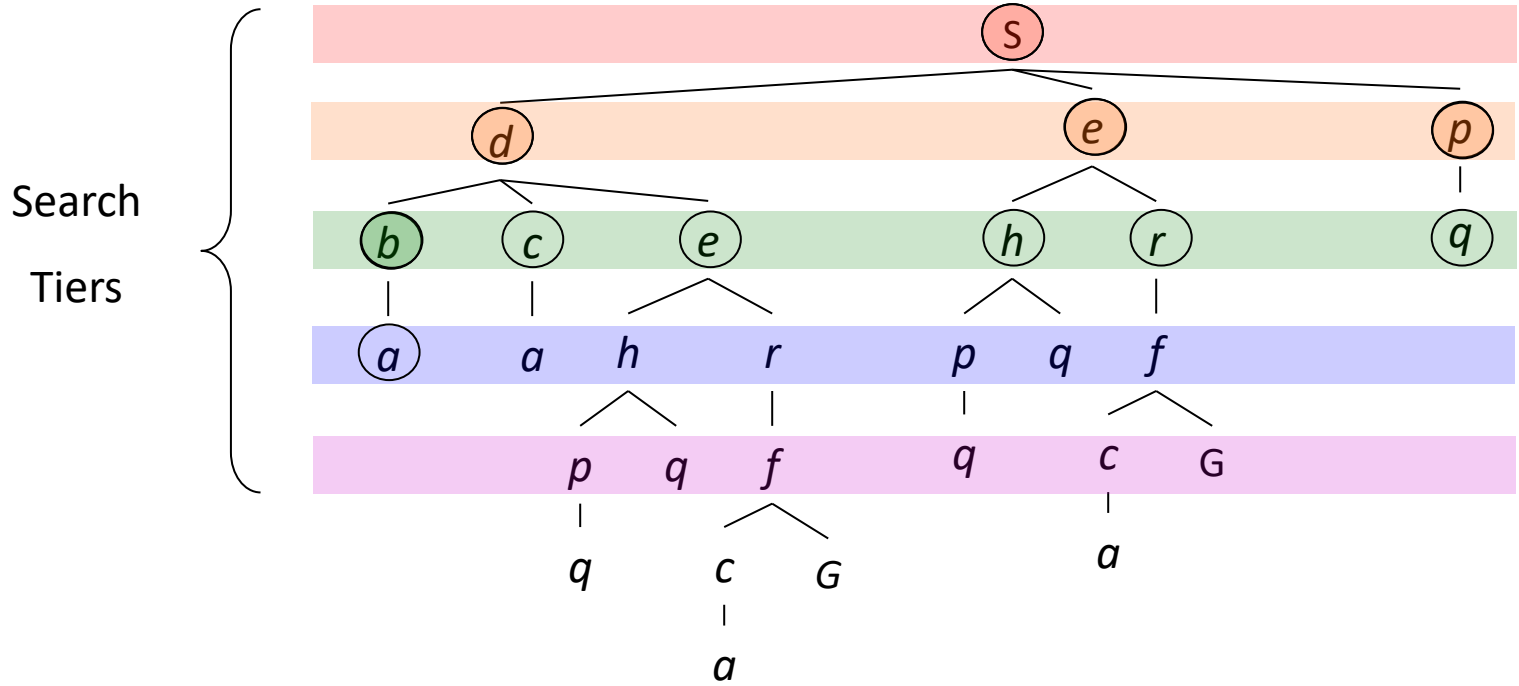


جستجوی اول سطح (BFS)

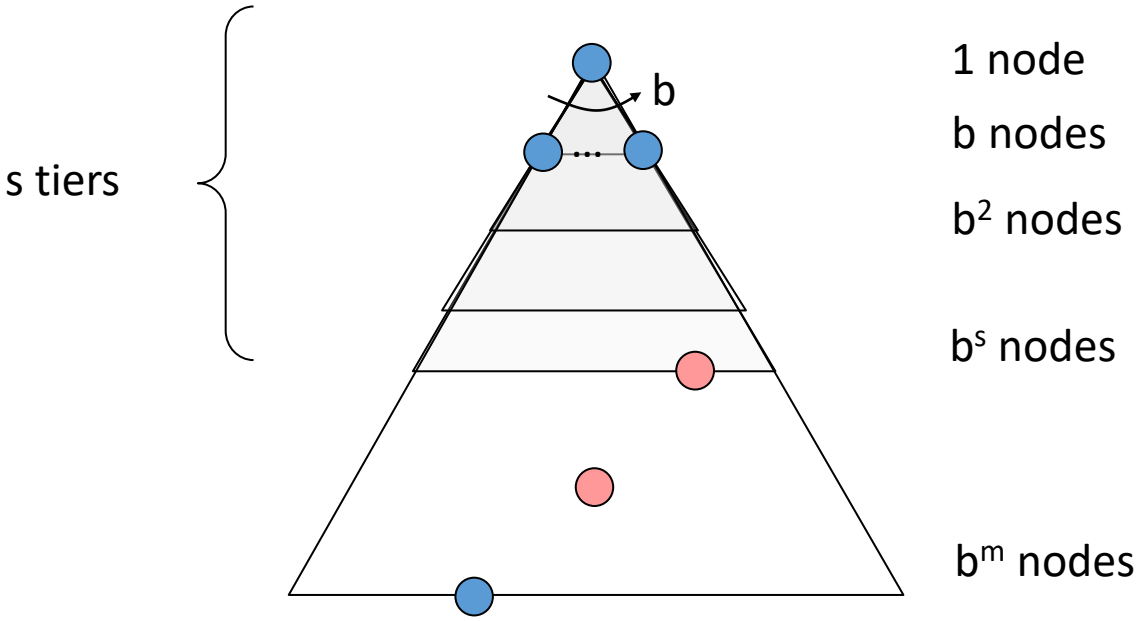


استراتژی:
ابتدا کم عمق‌ترین گره را گسترش دهید.

پیاده سازی:
Fringe یک صف FIFO است.

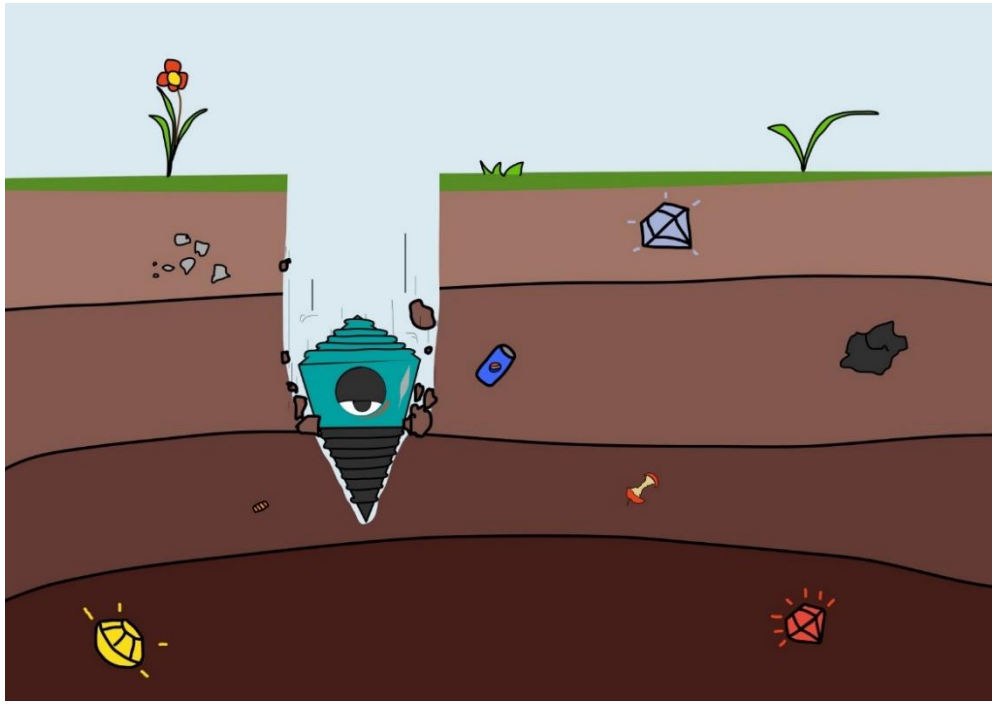


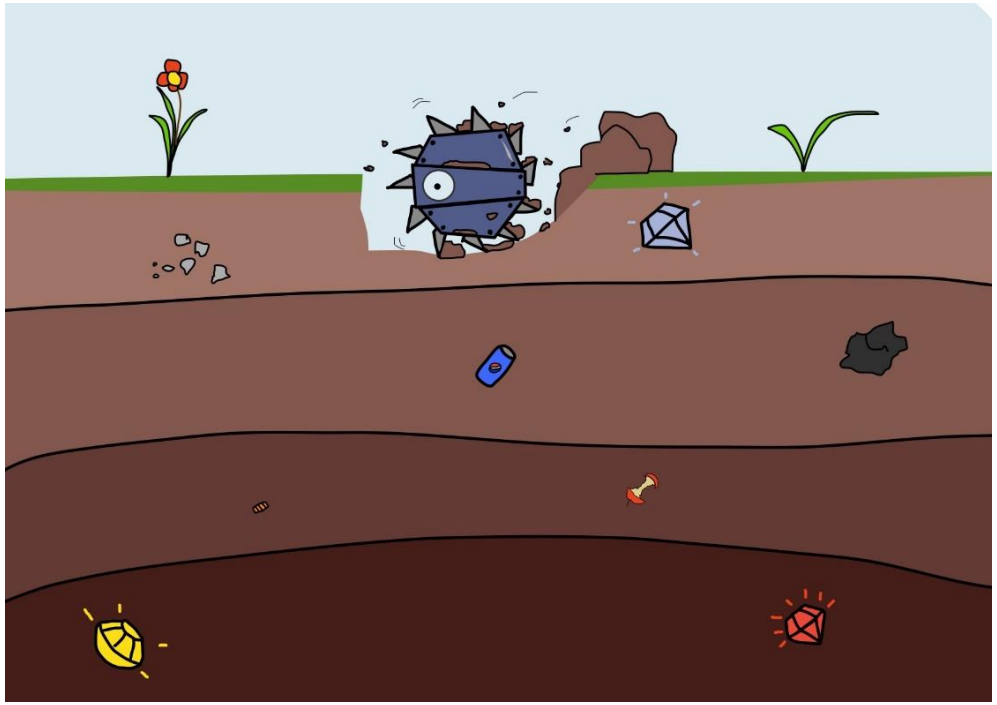
ویژگی‌های جستجوی اول سطح (BFS)



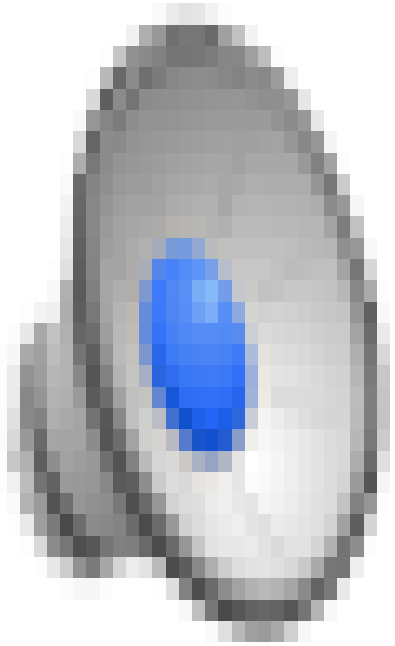
- BFSچه گره‌هایی را بسط می‌دهد؟
  - تمام گره‌های بالاتر از سطحی‌ترین راه‌حل را پردازش می‌کند
  - فرض کنید کم عمق‌ترین راه‌حل در عمق S باشد
  - جستجو $O(b^s)$ زمان می‌برد
- لبه چه اندازه فضا اشغال می‌کند؟
  - تقریبا آخرین ردیف در بر می‌گیرد، بنابراین $O(b^s)$
- آیا کامل است؟
  - اگر راه‌حلی وجود داشته باشد s باید محدود باشد، بنابراین بله!
- آیا بهینه است؟
  - فقط در صورتی که هزینه‌ها همه 1 باشد (در آینده بیشتر در مورد هزینه‌ها صحبت می‌شود)

تقابل DFS و BFS

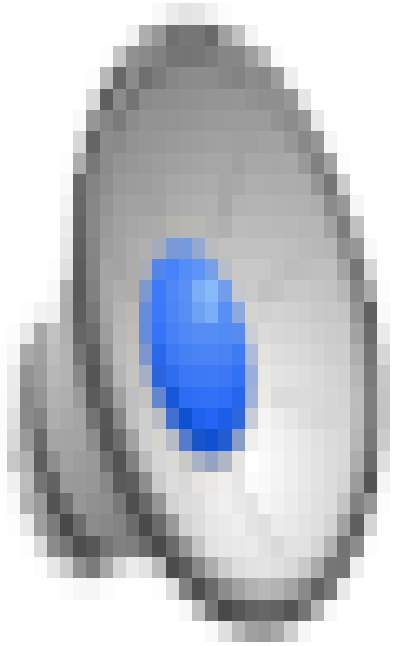




ویدیوی Demo Maze Water DFS/BFS(قسمت اول)

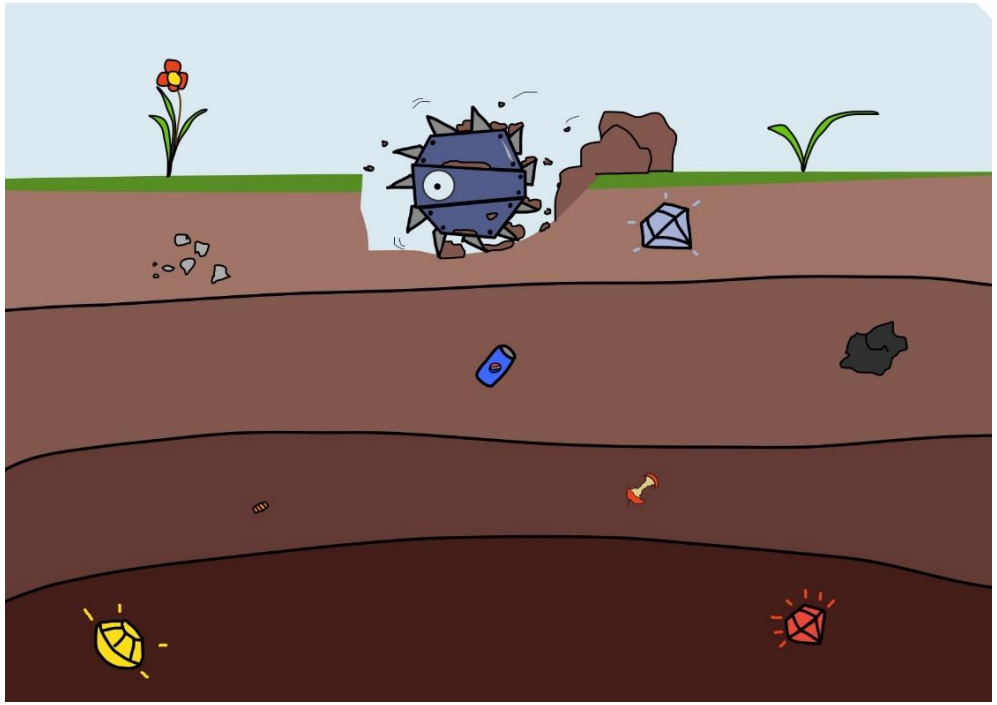


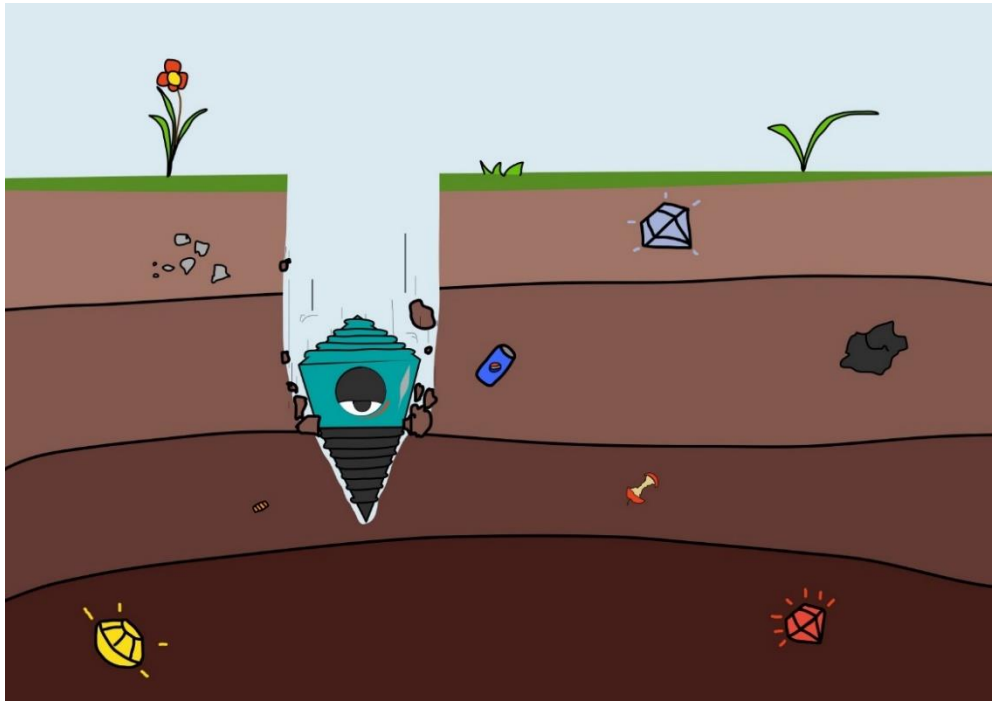
ویدیوی Demo Maze Water DFS/BFS(قسمت دوم)



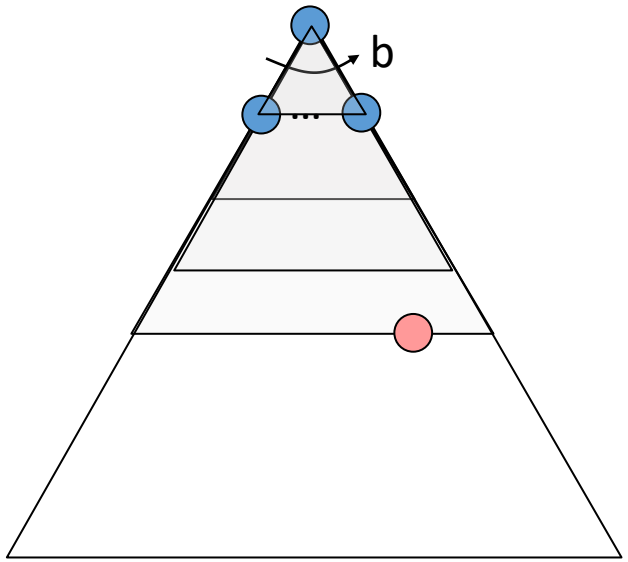
آزمونک: تقابل DFS و BFS

- چه زمانی BFS بهتر از DFS عمل می کند؟
- چه زمانی DFS بهتر از BFS عمل می کند؟





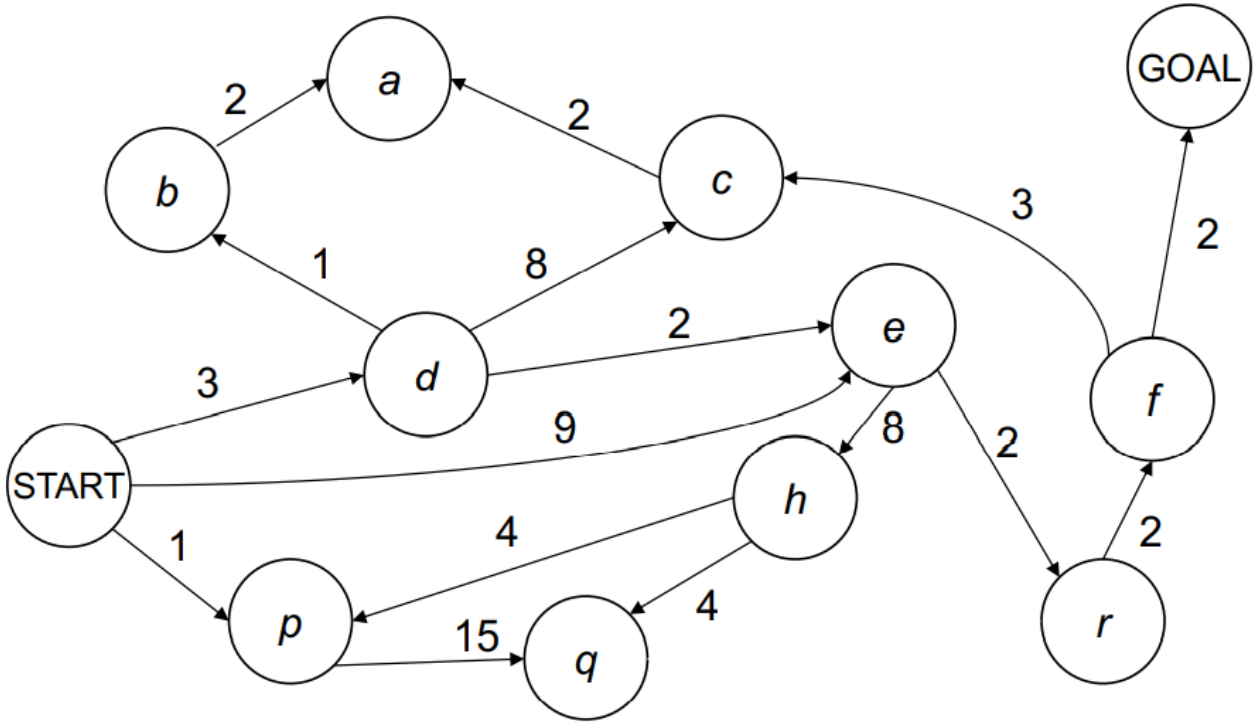
عمیق‌سازی تکرارشونده (iterative deepening)



- ایده: مزیت فضای DFS را با مزایای زمان/ راه حل کم عمق BFS همزمان داشته باشیم
  - یک DFS با محدودیت عمق 1 اجرا کنید. اگر راه‌حلی وجود نداشت...
  - یک DFS با محدودیت عمق 2 اجرا کنید. اگر راه‌حلی وجود نداشت...
  - یک DFS با محدودیت عمق 3 و ...
- آیا این کار بیهوده تکراری نیست؟
  - به طور کلی بیشترین کار در جستجوی پایین‌ترین سطح اتفاق می‌افتد،

    بنابراین خیلی بد نیست!

جستجوی حساس به هزینه

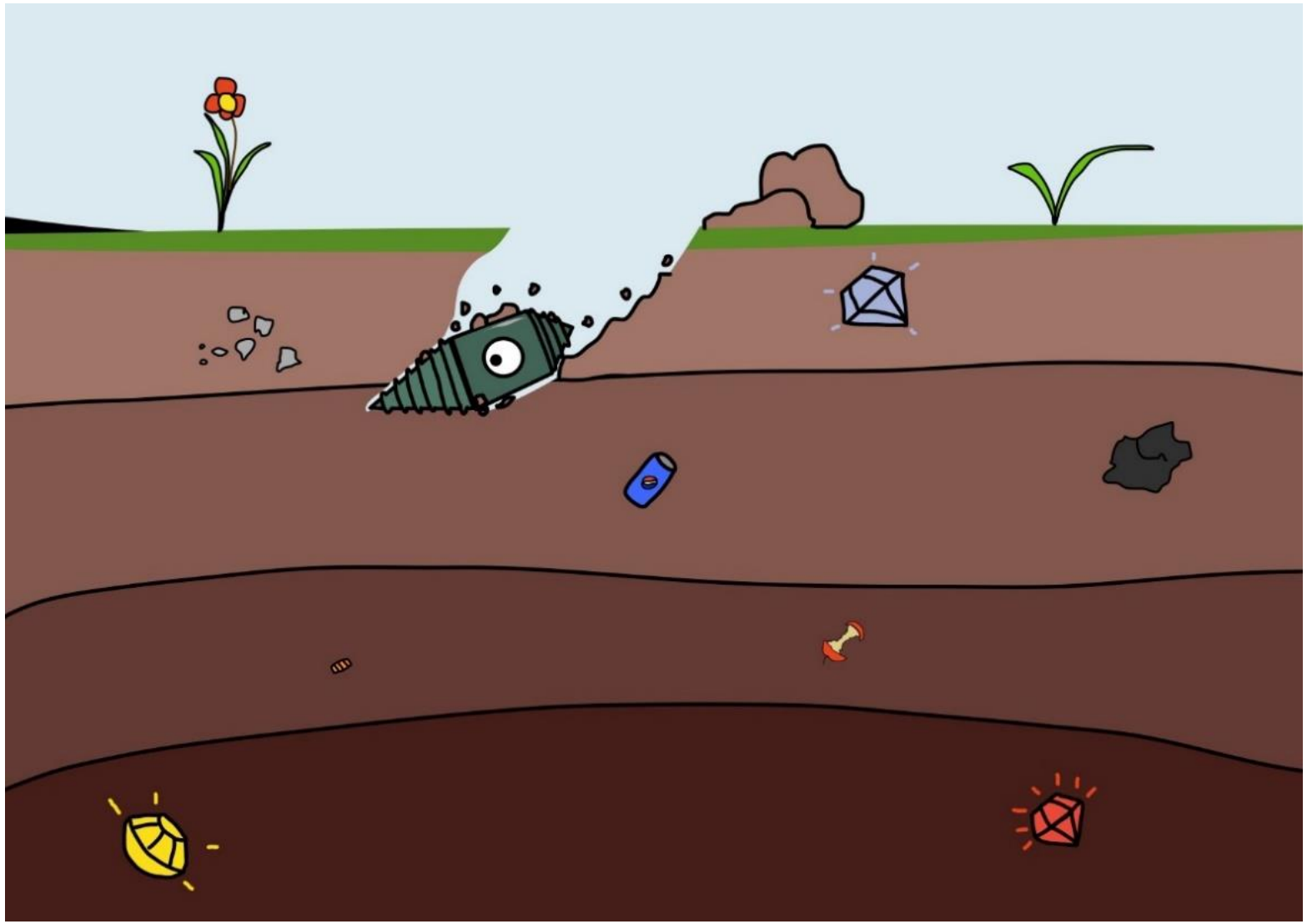


BFS کوتاه ترین مسیر را از نظر تعداد اعمال پیدا می‌کند.

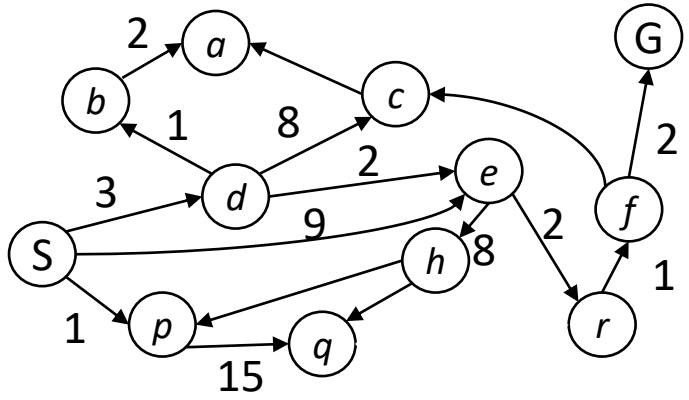
کمترین هزینه را پیدا نمی‌کند. اکنون الگوریتم مشابهی را پوشش می‌دهیم که مسیر کم هزینه را پیدا می‌کند.

چگونه؟

جستجوی هزینه یکنواخت (UCS)

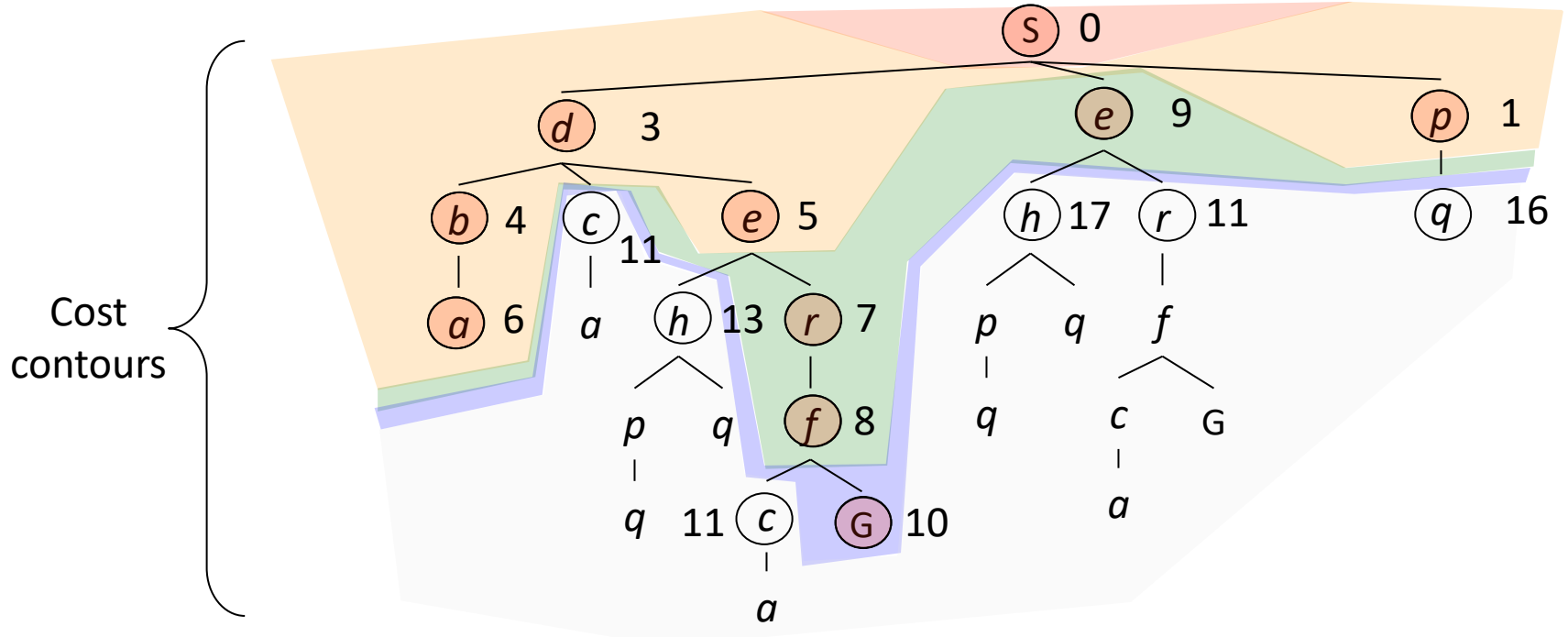


جستجوی هزینه یکنواخت



استراتژی:
ابتدا ارزان‌ترین گره را گسترش دهید

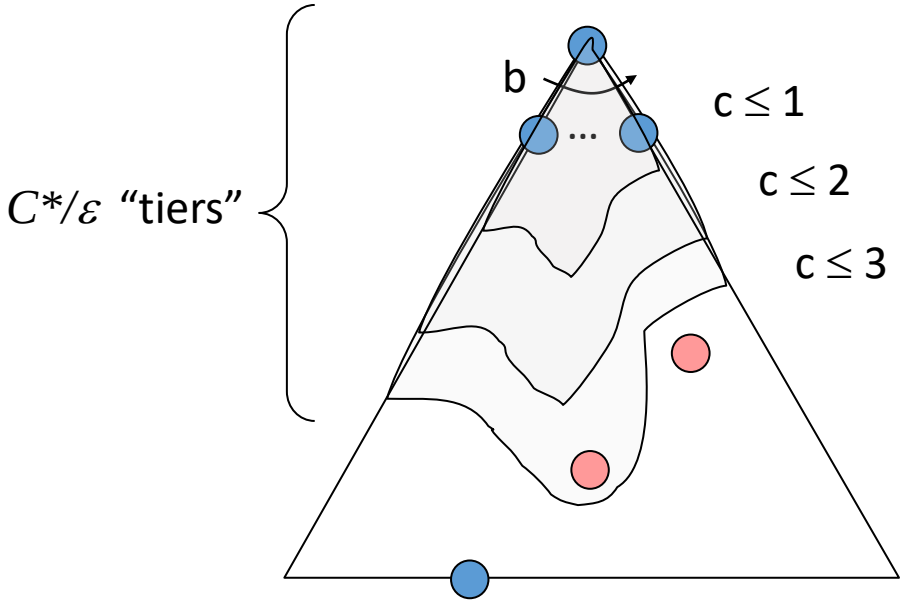
پیاده سازی:
یک صف اولویت است (اولویت: هزینه تجمعی)



ویژگی‌های جستجوی هزینه یکنواخت (UCS)

- UCSچه گره‌هایی را گسترش می‌دهد؟
  - تمام گره‌هایی که هزینه کمتر از ارزان‌ترین راه‌حل دارند را پردازش می‌کند!
  - اگر آن راه حل هزینه C* و کمان حداقل ε هزینه داشته باشد، «عمق مؤثر» تقریباً C*/ε است.
  - جستجو O(b^(C^*/ε)) زمان می‌برد.

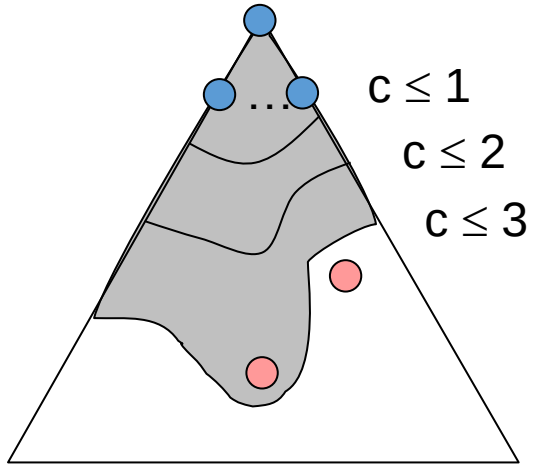
    (به صورت نمایی در عمق موثر)

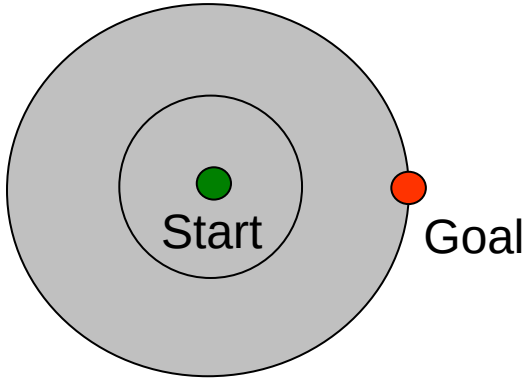


- لبه چه اندازه فضا اشغال می‌کند؟
  - تقریبا آخرین ردیف را دارد، بنابراین O(b^(C^*/ε))
- آیا کامل است؟
  - با فرض اینکه بهترین راه حل دارای هزینه محدود است

    و حداقل هزینه قوس مثبت است، بله!
- آیا بهینه است؟
  - بله! (اثبات در اسلاید بعدی از طریق A*)

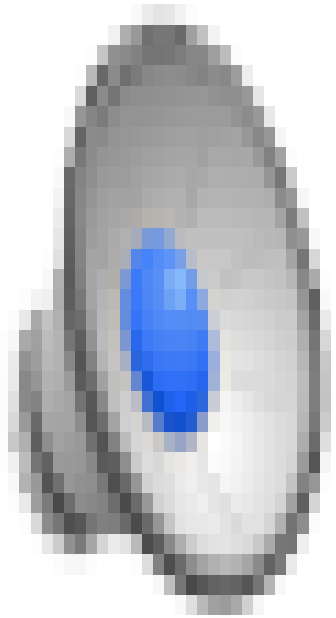
موضوعات هزینه یکنواخت



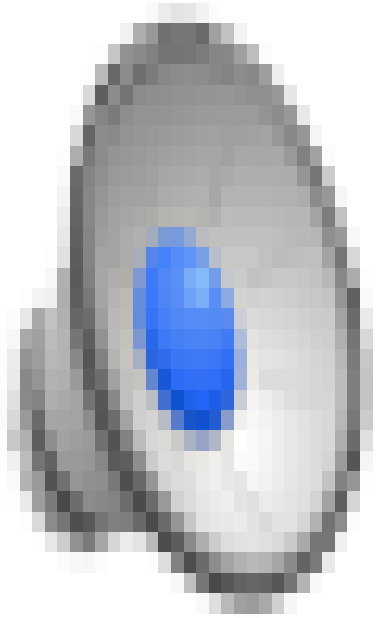


- به یاد داشته باشید: UCS خطوط افزایش هزینه را کاوش می‌کند
  - تمام گره‌هایی که هزینه کمتر از ارزان‌ترین راه‌حل دارند را پردازش می‌کند!
- خوبی: UCSکامل و بهینه است
- موارد بد:
  - گزینه‌ها را در هر جهت بررسی می‌کند
  - اطلاعاتی در مورد مکان هدف در نظر نمی‌گیرد
- به زودی این مشکل را برطرف می‌کنیم!

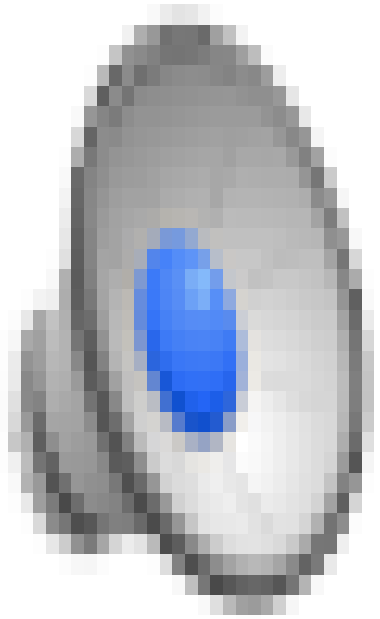
ویدیوی Demo Empty UCS



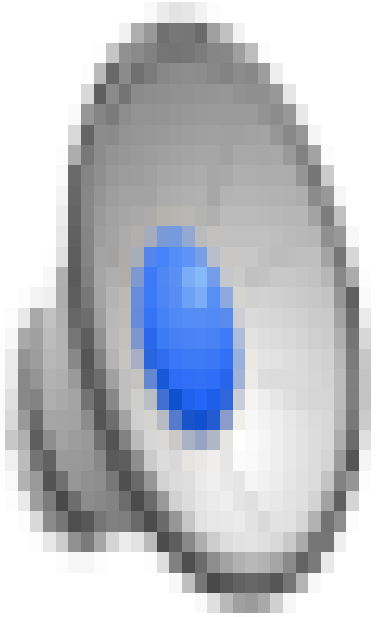
ویدیوی Demo Maze با Deep/Shallow Water --- DFS, BFS یا UCS؟(قسمت اول)



ویدیوی Demo Maze با Deep/Shallow Water --- DFS, BFS یا UCS؟(قسمت دوم)

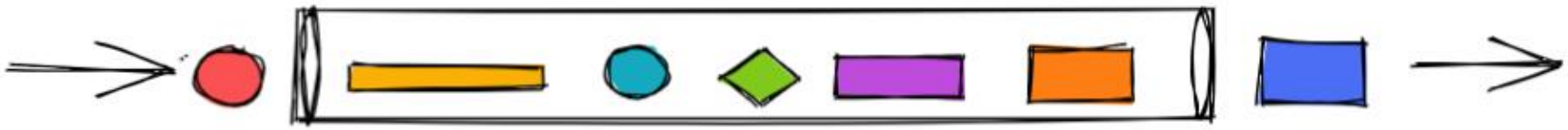


ویدیوی Demo Maze با Deep/Shallow Water --- DFS, BFS یا UCS؟(قسمت سوم)



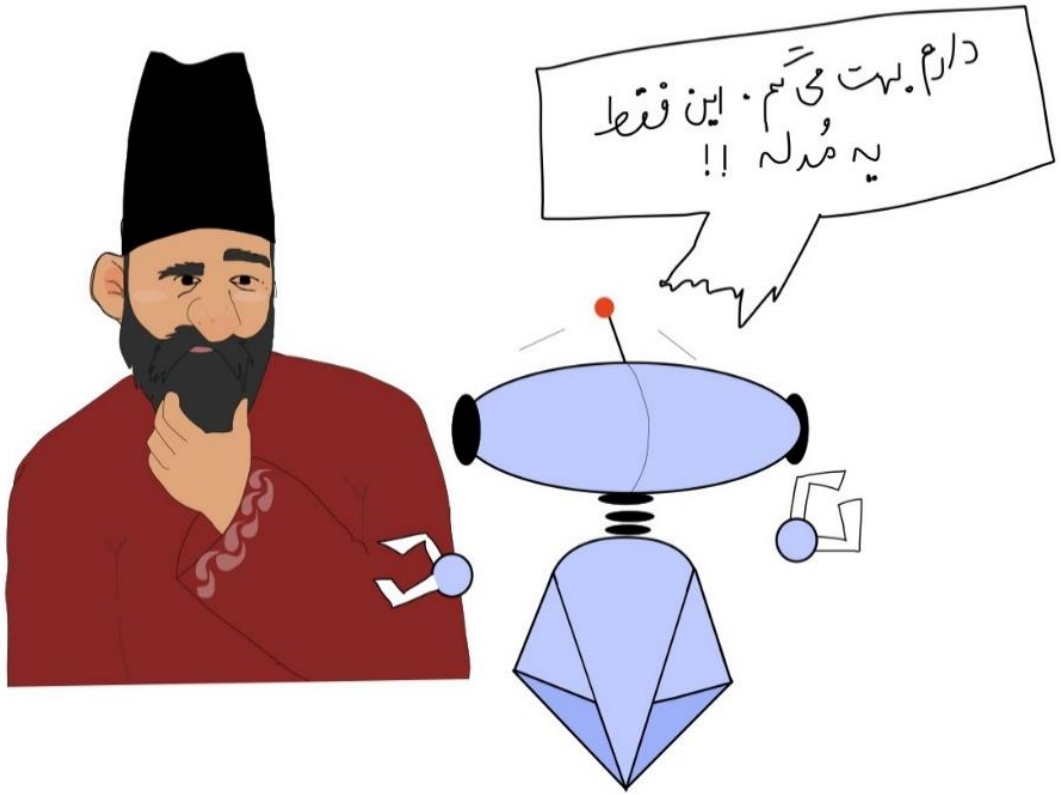
یک صف

- همه این الگوریتم‌های جستجو به جز در استراتژی‌های کاوش لبه یکسان هستند
  - از نظر مفهومی، تمام لبه‌ها صف‌های اولویت هستند. (مثلا مجموعه‌ای از گره‌ها با اولویت‌های پیوست شده)
  - عملاً، برای DFS و BFS، می‌توانید با استفاده از پشته‌ها و صف‌ها، از سربار log(n) از یک صف اولویت واقعی اجتناب کنید
  - حتی می‌توان کد یک پیاده‌سازی را نوشت که یک شیء متغیر صف را بگیرد.



جستجو و مدل

- جستجو بر روی مدل‌های جهان عمل می‌کند و نه خود جهان!!
  - عامل در واقع تمام برنامه‌ریزی‌ها را در دنیای واقعی امتحان نمی‌کند!
  - برنامه‌ریزی همه در "شبیه سازی" است
  - جستجوی شما تنها به اندازه مدل شما خوب است...



جستجو به خطا رفت؟