به نام پروردگار دانایی

هوش مصنوعی

درس ششم بخش دوم: جستجوهای آگاهانه – بهبود مصرف حافظه در A*

سید کاوه احمدی

بهبود مصرف حافظه در A*

- در A* فضای مصرفی به خاطر استفاده از صف اولویت نمایی است.
- روش‌های بهبود حافظه در A*
  - جستجوی عمیق کننده تکراری A* (Iterative Deepening A* - IDA*)
    - تعمیم جستجوی عمیق کننده تکراری با استفاده از هیوریستیک‌ها
  - جستجوی ساده شده با حافظه محدود A* (Simplified Memory bounded A* - SMA*)
    - کاهش اندازه حافظه به طوریکه با میزان حافظه موجود مطابقت داشته باشد

جستجوی عمیق کننده تکراری A*

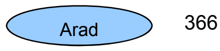
- ساده ترین راه برای کاهش حافظه مورد نیاز A* استفاده از عمیق کننده تکرار در زمینه جست و جوی اکتشافی است.
- در جستجوی IDA* به جای محدوده عمقی از محدوده f-cost استفاده می‌شود.
  - مقدار برش مورد استفاده، عمق نیست بلکه هزینه f(g+h) است.
  - در هر تکرار فقط گره‌های بسط داده می‌شوند که $f(n) \leq$ f-cost باشد.
  - در صورتیکه گره هدف در این ناحیه (کانتور) یافت نشود، جستجو با افزایش ناحیه دنبال خواهد شد.

جستجوی عمیق کننده تکراری A*

- IDA* برای اغلب مسئله‌های با هزینه‌های مرحله‌ای مناسب است و از سربار ناشی از نگهداری صف مرتبی از گره‌ها اجتناب می‌کند.
- IDA* ترکیب جستجوی عمیق کننده تکراری و A* است تا از مزایای هر دو بهره‌مند باشد.
- بهینه و کامل است (اگر f-cost نهایی از C* کمتر نباشد).
- پیچیدگی زمان بستگی به تعداد f-costهای انتخابی دارد. اگر تکرارها زیاد نباشد در محدوده A* عمل می‌کند.
- پیچیدگی فضا تقریبا مانند جستجوی عمقی عمل می‌کند.
  - بخاطر اینکه در هر مرحله فقط گره هایی که f(n) آنها کمتر از f-limit است در حافظه نگهداری می شود به همین دلیل از نظر پیچیدگی حافظه مثل جستجوی عمقی O(bd) است (به خاطر استفاده از پشته).
  - البته در بدترین حالت $O(bf^*/\delta)$ است که δ کمترین هزینه عملگر است

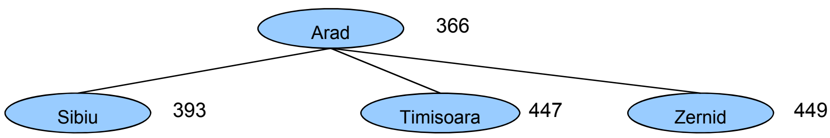
جستجوی عمیق کننده تکراری A*

برش اول f-cost = 400



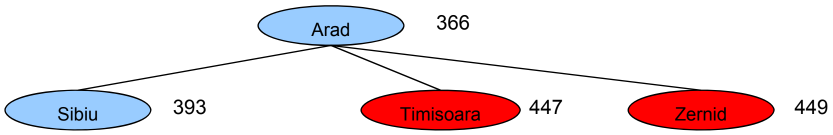
جستجوی عمیق کننده تکراری A*

برش اول f-cost = 400



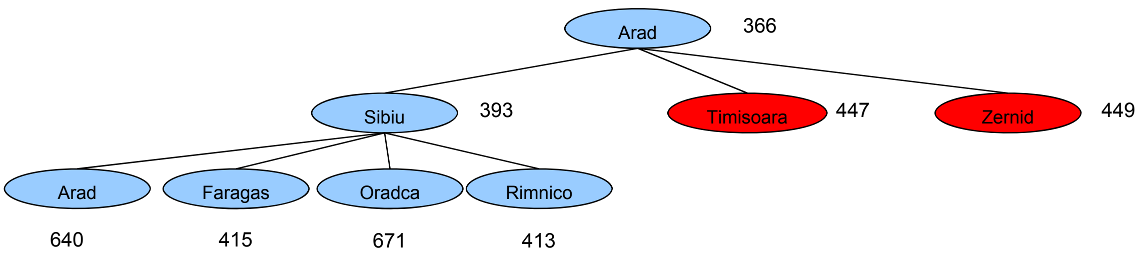
جستجوی عمیق کننده تکراری A*

برش اول f-cost = 400



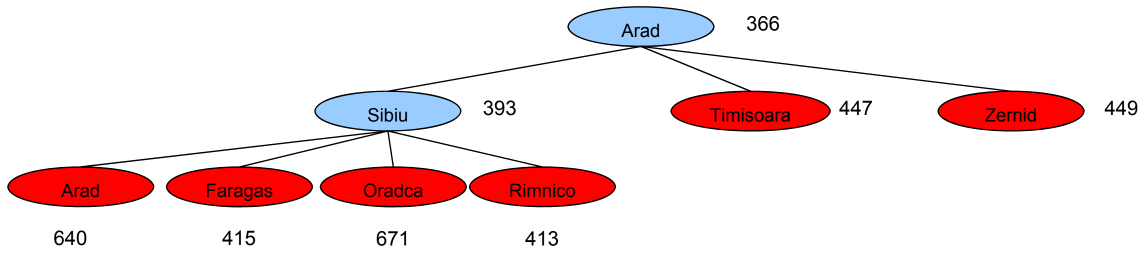
جستجوی عمیق کننده تکراری A*

برش اول f-cost = 400



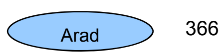
جستجوی عمیق کننده تکراری *A

برش اول f-cost = 400



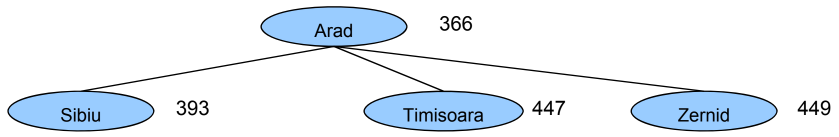
جستجوی عمیق کننده تکراری *A

برش دوم f-cost = 500



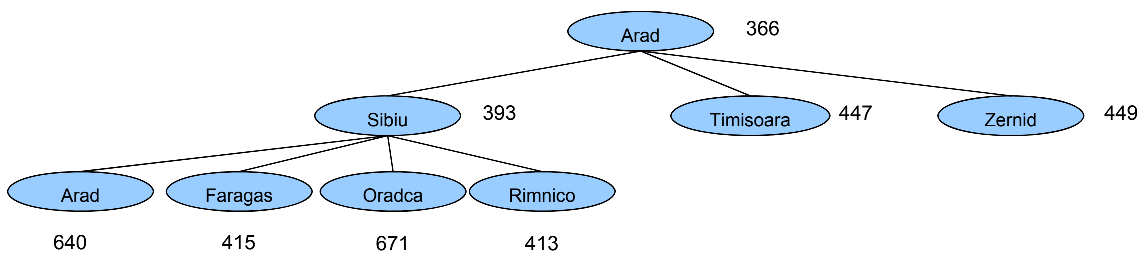
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



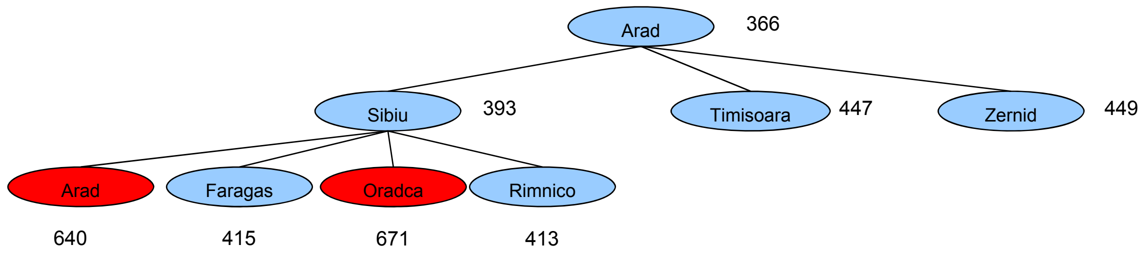
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



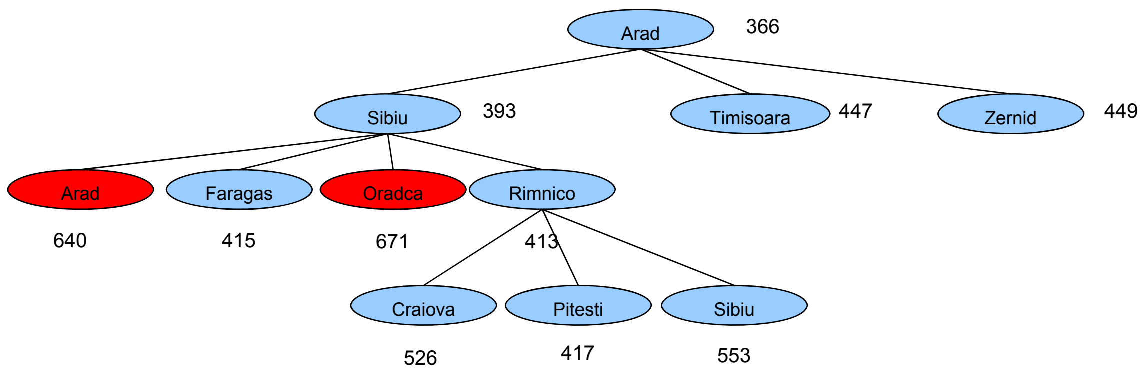
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



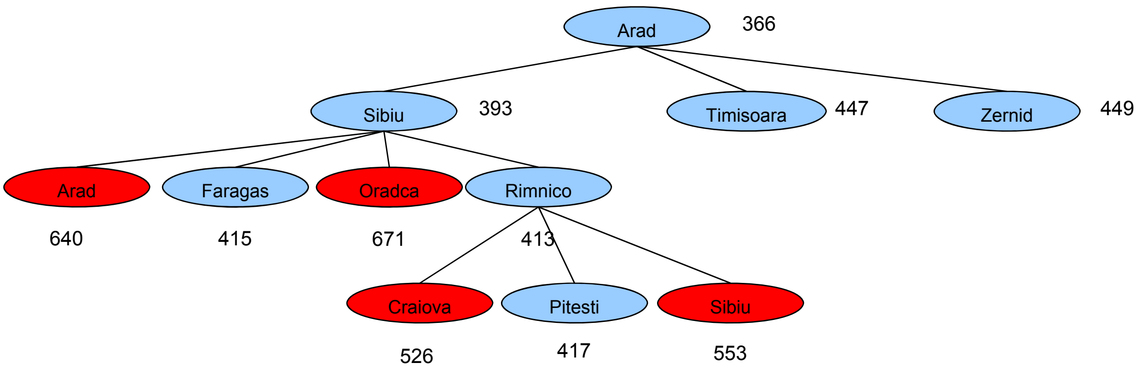
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



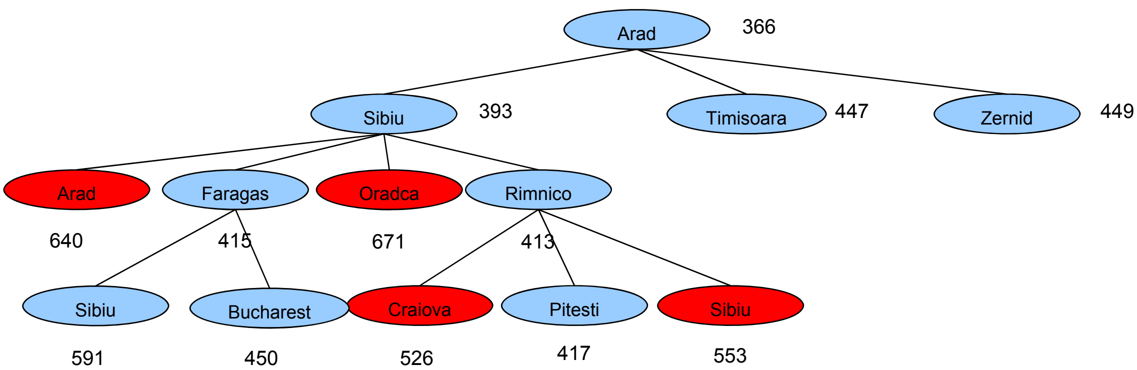
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



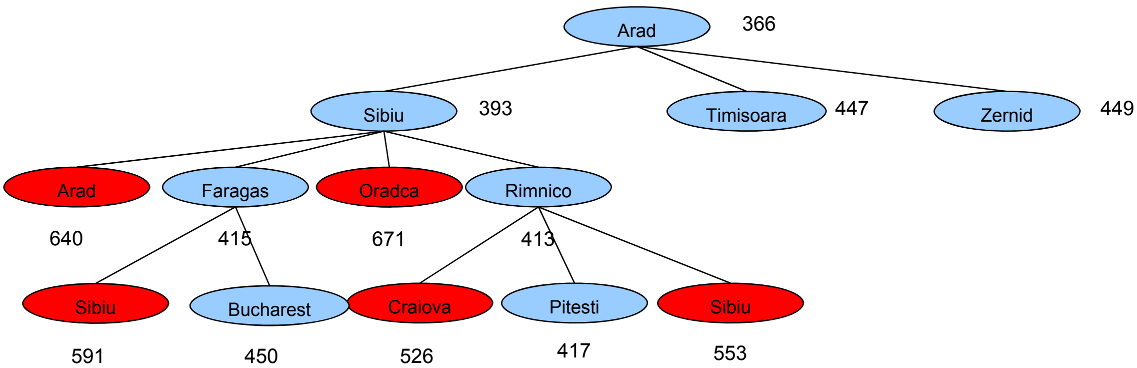
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



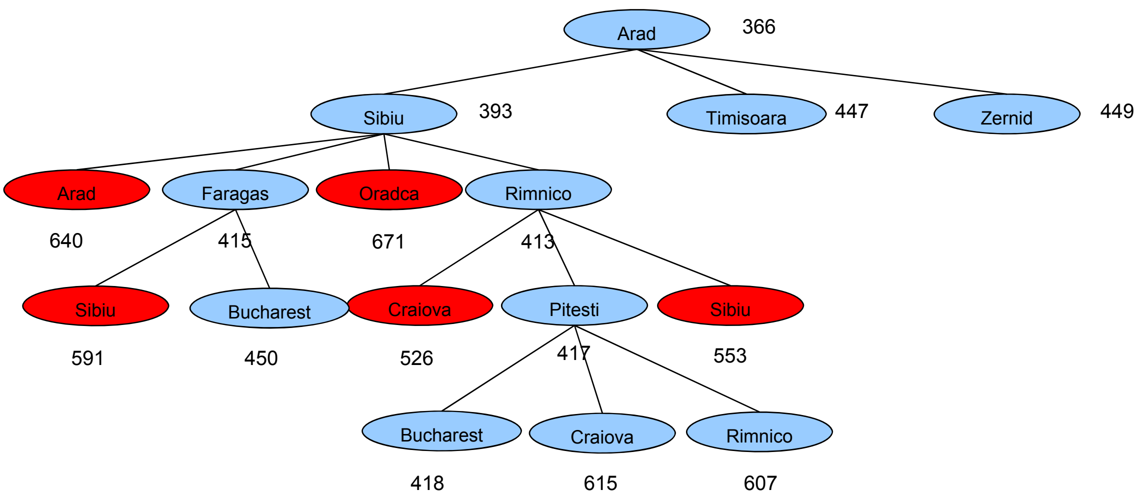
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



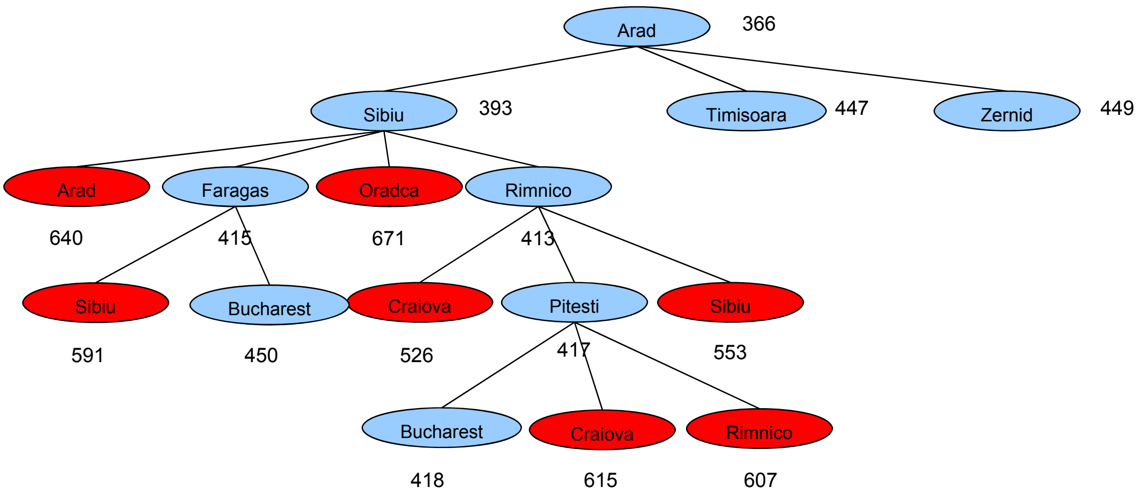
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



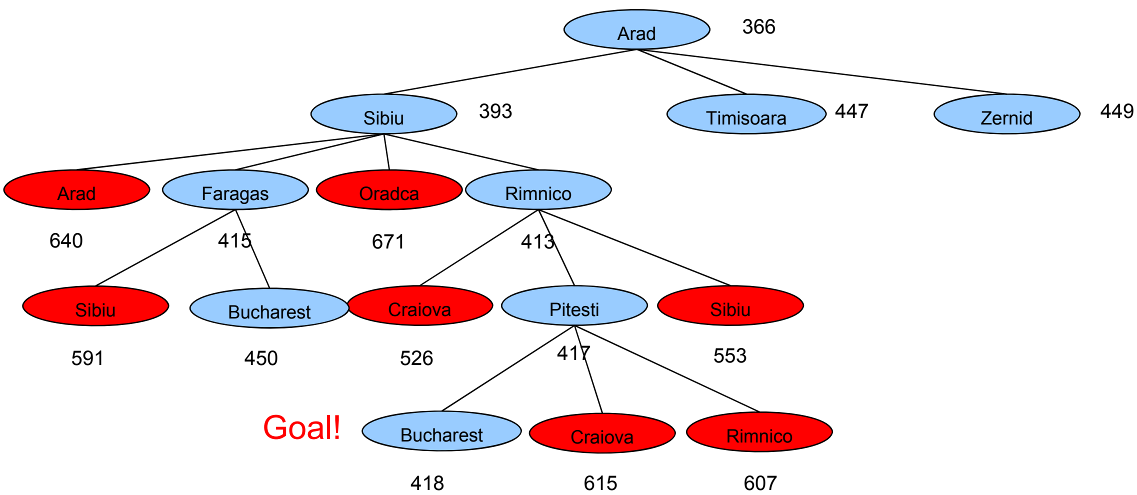
جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



جستجوی عمیق کننده تکراری A*

برش دوم f-cost = 500



همان A* است فقط در هر برش برخی گره ها را در حافظه نگه نمی‌دارد. به جای آن به خاطر وجود برش‌ها، زمان اجرای بیشتری دارد. باید دید برای یک مسئله این یک مزیت است یا اشکال

اشکالات IDA*

- در مسائلی همانند مسئله فروشنده دوره‌گرد که در آن با گراف‌های وزن‌دار مواجه هستیم تکرارها از نظر زمانی ما را دچار مشکل می‌کند.
- هرچه تنوع وزن‌ها در گراف فضای حالت بیشتر باشد، تعداد تکرارها نیز بیشتر خواهد شد و این می‌تواند آنقدر مسئله را کند کند که دیگر کارا نباشد.

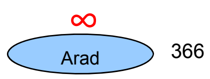
جستجوی اول بهترین بازگشتی (RBFS)

- الگوریتم دیگری برای کاهش استفاده از حافظه در روش A* است.
- ساختار آن شبیه جست و جوی عمقی بازگشتی است به جای اینکه دائما به طرف پایین مسیر حرکت کند:
  - بازگشت را با نگهداری جریان f-limit بهترین مسیر جایگزین از هر جد گره فعلی محدود می‌نماید.
  - در صورتی که گره جاری از این مقدار بیشتر شود، بازگشت به مسیر جایگزین بر می‌گردد.
  - بدین صورت حرکت به عقب f-limit گره جاری را با بهترین هزینه زیر درخت‌ها جایگزین می‌کند.

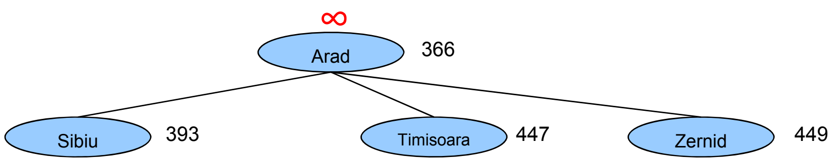
جستجوی اول بهترین بازگشتی (RBFS)

- مقدار f-limit برای هر فراخوانی بازگشتی در بالای هر گره جاری نشان داده شده است.
  - هر گره تا زمانی رشد می‌کند که f-limit آن از سایر گره‌های بسط داده نشده بیشتر است.

جستجوی اول بهترین بازگشتی (RBFS)

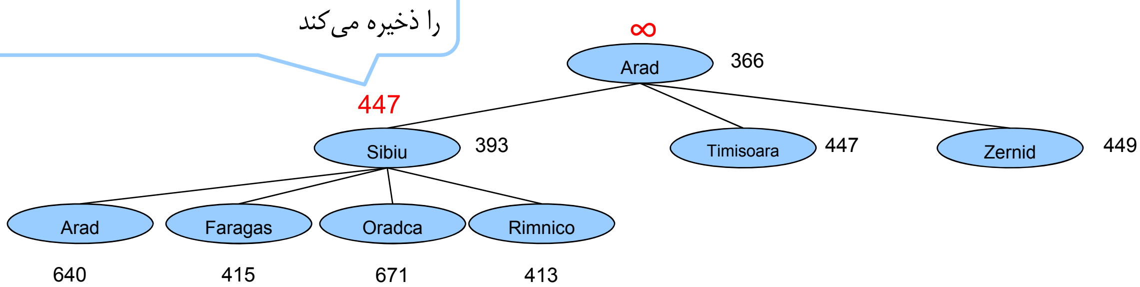


جستجوی اول بهترین بازگشتی (RBFS)

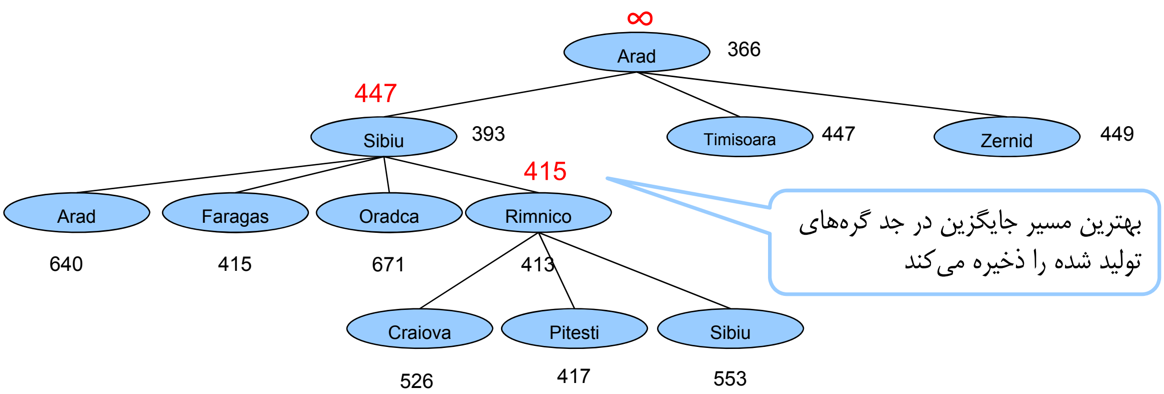


جستجوی اول بهترین بازگشتی (RBFS)

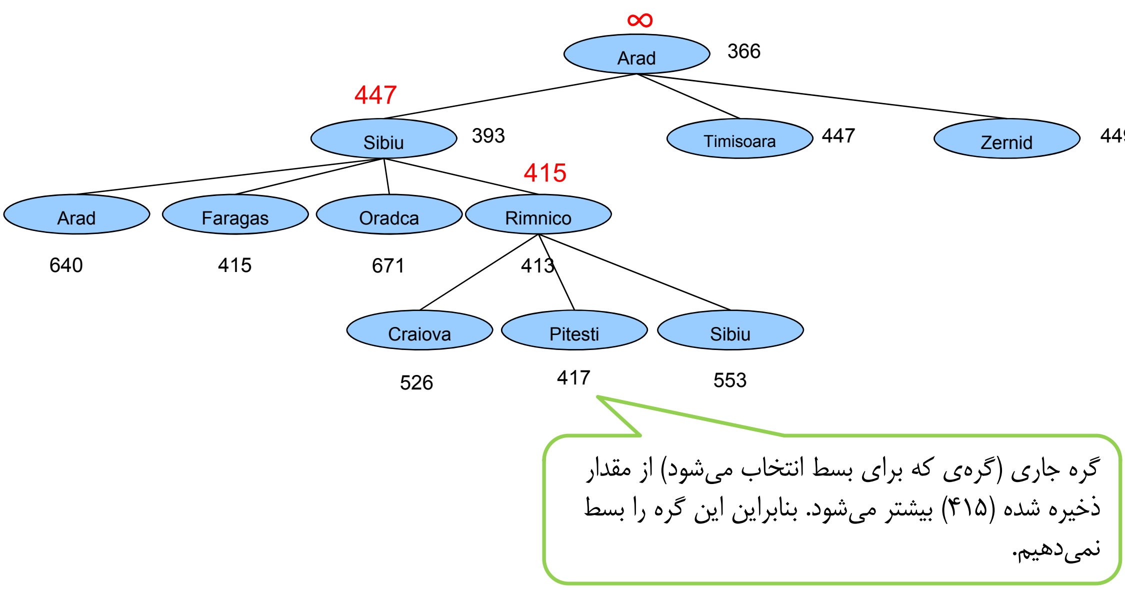
بهترین مسیر جایگزین در جد گره‌های تولید شده را ذخیره می‌کند



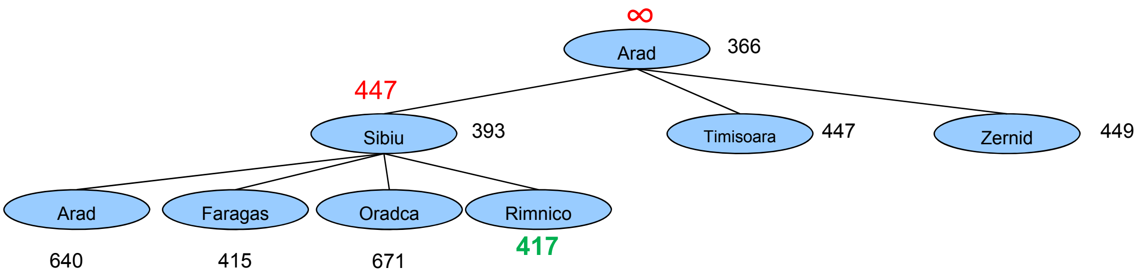
جستجوی اول بهترین بازگشتی (RBFS)



جستجوی اول بهترین بازگشتی (RBFS)

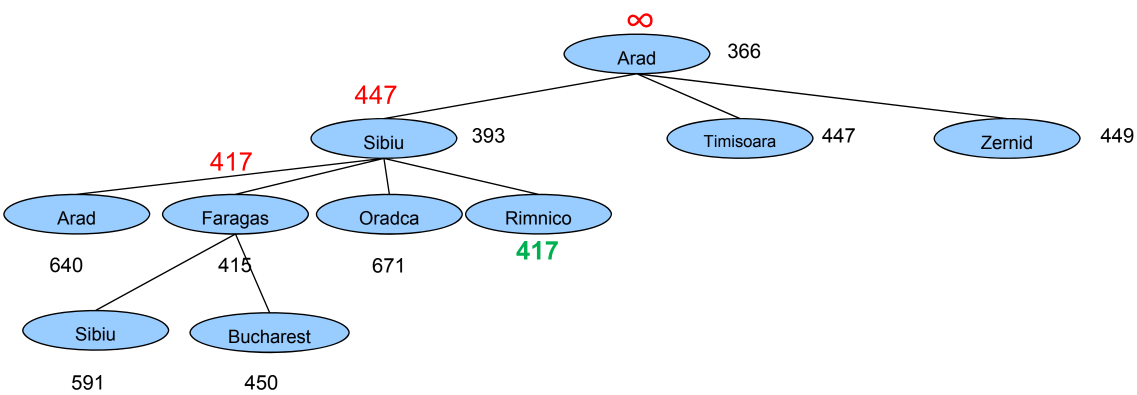


جستجوی اول بهترین بازگشتی (RBFS)

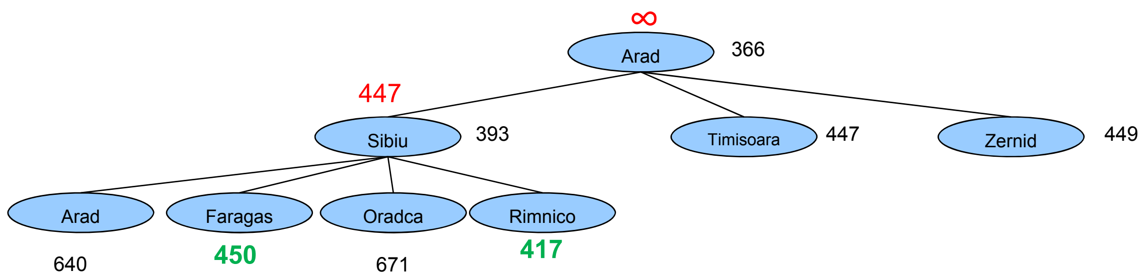


فهمیدیم بهترین f با انتخاب Rimnico، ۴۱۷ خواهد بود. این مقدار را به عنوان f جدید گره ذخیره می‌کنیم.

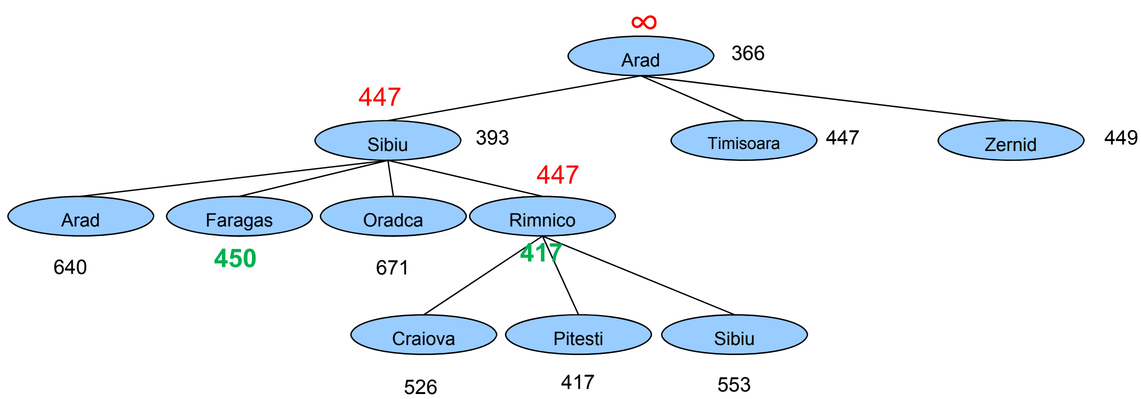
جستجوی اول بهترین بازگشتی (RBFS)



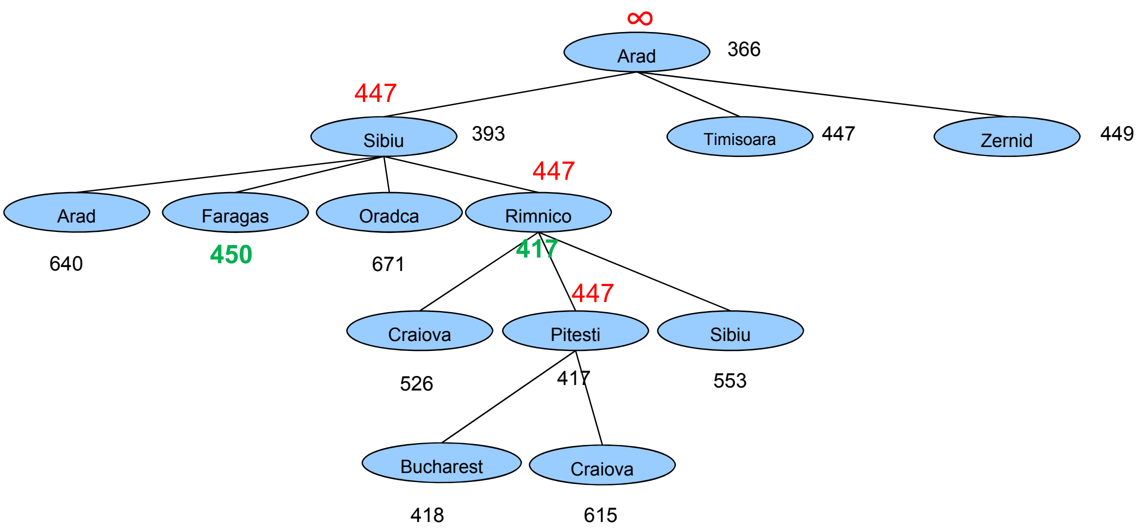
جستجوی اول بهترین بازگشتی (RBFS)



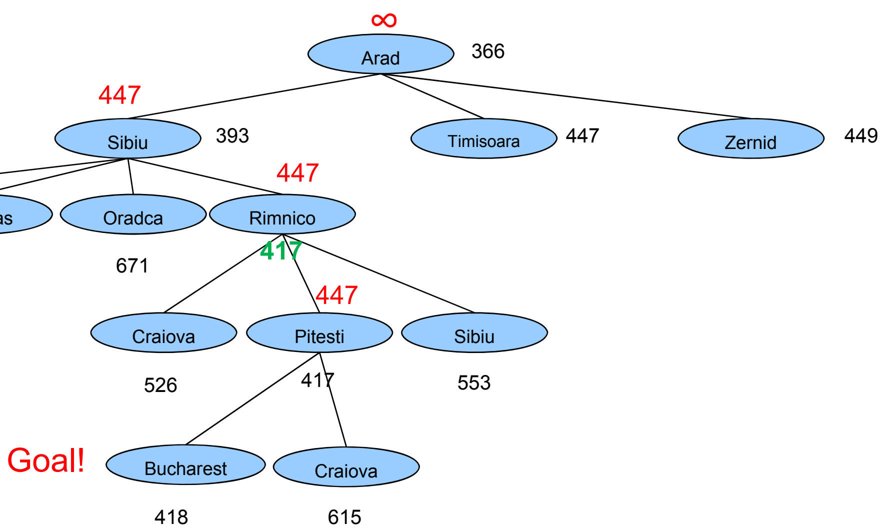
جستجوی اول بهترین بازگشتی (RBFS)



جستجوی اول بهترین بازگشتی (RBFS)



جستجوی اول بهترین بازگشتی (RBFS)

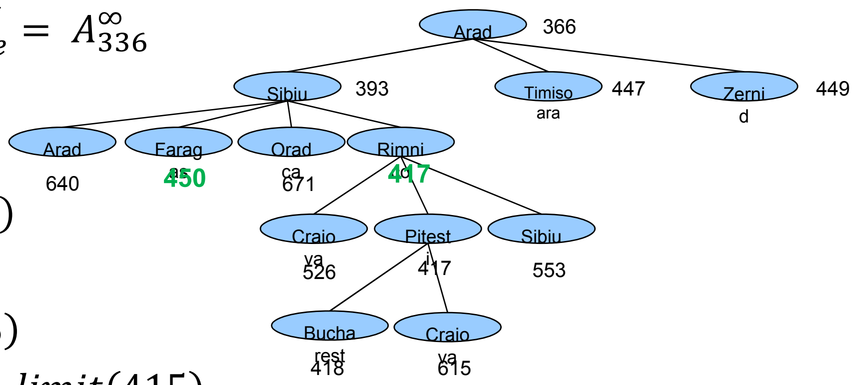


الگوریتم RBFS

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
   return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do  /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```


الگوریتم RBFS

- $Initial\ State\colon A_{f-value}^{f-limit} = A_{336}^{\infty}$
- $RBFS(A, \infty)$
- $Best\colon S_{393}, Alt\colon T_{447}$
- $Recurse\colon RBFS(S, 447)$
- $Best\colon R_{413}, Alt\colon F_{415}$
- $Recurse\colon RBFS(R, 415)$
- $Best\colon P_{417}, exeeds\ f-limit(415)$
- $Return\langle failure, 417\rangle$
- $Unwind\ to\ RBFS(S, 447)$
- $Update\ best\ leaf\ value\ (417)\ to\ forgotten\ subtree\ at\ R$
- $Best\colon F_{415}, Alt\colon R_{417}$
- $Recurse\colon RBFS(F, 417)$
- …



جستجوی اول بهترین بازگشتی (RBFS)

- کامل و بهینه است (مشابه *A)
- پیچیدگی زمانی: تعیین پیچیدگی زمانی آن به دقت تابع اکتشافی و میزان تغییر بهترین مسیر در اثر بسط گره‌ها بستگی دارد.
- پیچیدگی فضا: تابع خطی از عمق عمیق‌ترین راه حل بهینه است O(bd).
- RBFS عملا از ساختار داده‌ای پشته استفاده می‌کند.
- RBFS تا حدی از *IDA کارآمدتر است، اما گره‌های زیادی تولید می‌کند.

اشکالات *IDA و RBFS

- *IDA و RBFS در معرض افزایش توانی پیچیدگی قرار دارند که در جست و جوی گراف‌ها مرسوم است، زیرا نمی‌توانند حالت‌های تکراری را در غیر از مسیر فعلی بررسی کنند. لذا، ممکن است یک حالت را چندین بار بررسی کنند.
- *IDA بین هر تکرار فقط یک عدد را نگهداری می‌کندکه هزینه فعلی f است.
- RBFS می‌تواند حداکثر از b*d خانه حافظه استفاده کند حتی اگر حافظه بیشتری وجود داشته باشد.
- *IDA و RBFS از فضای اندکی استفاده می‌کنند و این امر از کارایی آنها کم می‌کند. اگر حافظه بیشتری وجود داشته باشد، این الگوریتم‌ها راهی برای استفاده از آنها بلد نیستند.
  - الگوریتم‌های *Memory Bounded A و *Simplified MA این توانایی را دارند.

جستجوی حافظه محدود ساده *SMA

- *SMA از یک صف اولویت با طول مشخص شده برای نگهداری گره‌ها استفاده می‌کند و همواره بهترین برگ را بسط می‌دهد تا حافظه پر شود.
- در این نقطه بدون از بین بردن گره های قبلی نمی‌تواند گره جدیدی اضافه کند.
- در این شرایط یک گره را از انتهای صف خارج می‌کند.
  - بدیهی است گره‌های انتهای صف، f بزرگتری دارند و امید به رسیدن به هدف از طریق آنها کمتر است.
  - گره‌هایی که به این طریق از صف حذف می‌شوند، گره‌های فراموش‌شده (forgotten nodes) یا گره های بدون امید (unpromise) (با هزینه بالا و قدیمی) نامیده می‌شوند.

جستجوی حافظه محدود ساده *SMA

- انتقادی که به این روش وارد می‌شود این است که ممکن است به دلیل تخمین نادرست تابع اکتشاف، گره حذف شده اتفاقا روی مسیر بهینه باشد. در این شرایط الگوریتم مسیر رسیدن به پاسخ بهینه را گم خواهد کرد.
  - برای حل این مشکل و همچنین برای اجتناب از جستجوی مجدد زیردرخت‌هایی که از حافظه حذف شده‌اند، گره‌های اجدادی نگه داشته می‌شوند و در آنها اطلاعاتی در مورد کیفیت بهترین مسیر در زیر درخت فراموش شده، نگهداری می‌شود.
- پس جد زیر درخت فراموش شده، کیفیت بهترین مسیر در آن زیر درخت را می داند و فقط در صورتی زیردرخت فراموش شده را دوباره تولید می کند که تمام مسیرهای دیگر گران‌تر از مسیر فراموش شده باشد.
- همچنین چون امکان جستجوی مسیرهای با طول بیشتر از مقدار حافظه وجود ندارد، این مسیرها با هزینه ∞ مشخص می‌شوند.

جستجوی حافظه محدود ساده SMA*

- اگر مقدار f تمام برگ‌ها یکسان باشد، الگوریتم ممکن است یک گره را هم برای بسط و هم برای حذف انتخاب کند. این مشکل با حذف بهترین برگ جدید و حذف بدترین برگ قدیمی حل می‌شود.
- SMA* کامل است اگر عمق سطحی‌ترین گره هدف کمتر از اندازه حافظه باشد.
- SMA* بهترین الگوریتم همه منظوره برای یافتن حل‌های بهینه است.

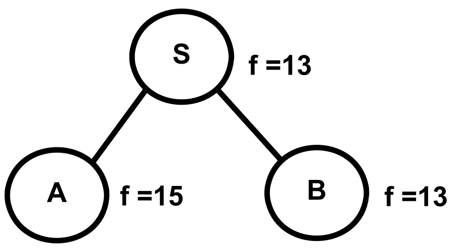
جستجوی حافظه محدود ساده SMA*

- ممکن است SMA* مجبور شود دائما بین مجموعه‌ای از مسیرهای حل کاندید تغییر موضع دهد، در حالی که بخش کوچکی از هر کدام در حافظه جا شود.
- زمان اضافی مورد نیاز برای تولید تکراری بعضی از گره‌ها به این معناست که مسئله‌هایی که برای A* قابل حل هستند (با توجه به محدودیت حافظه) برای SMA* غیر قابل حل می‌شوند.
- محدودیت‌های حافظه ممکن است مسئله‌ها را از نظر زمان محاسباتی، غیر قابل حل کند.
- زمانی که حافظه موجود برای درخت جستجو کامل کافی باشد جستجو کارآ بهینه Optimally efficient است.

الگوریتم جستجوی SMA*

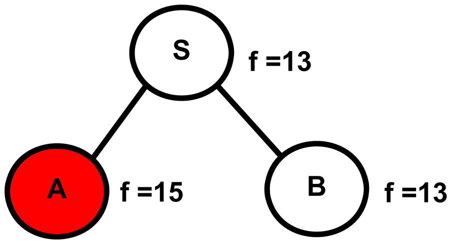
```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  local variables: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
         f(s) ← ∞
     else
         f(s) ← MAX(f(n), g(s)+h(s))
     if all of n's successors have been generated then
         update n's f-cost and those of its ancestors if necessary
     if SUCCESSORS(n) all in memory then remove n from Queue
     if memory is full then
         delete shallowest, highest-f-cost node in Queue
         remove it from its parent's successor list
         insert its parent on Queue if necessary
     insert s on Queue
  end
```


نکات مربوط به جستجوی SMA*



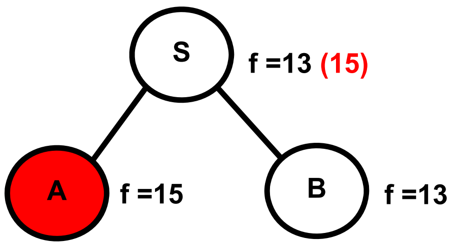
- با فرض وجود حافظه به اندازه ۳ گره:
  - اگر حافظه پر شد،

نکات مربوط به جستجوی *SMA



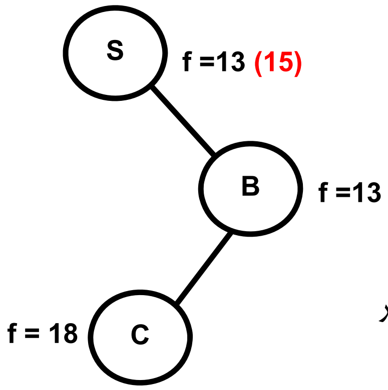
- با فرض وجود حافظه به اندازه ۳ گره:
  - اگر حافظه پر شد،
    - گره با بشترین f را حذف کن

نکات مربوط به جستجوی *SMA



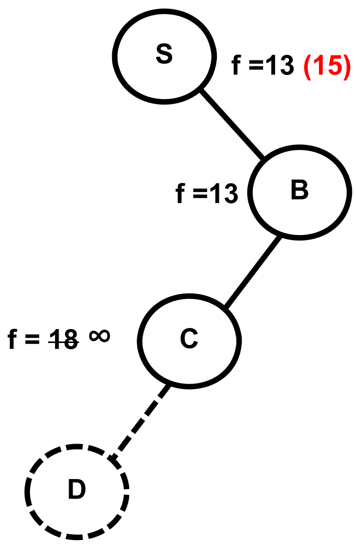
- با فرض وجود حافظه به اندازه ۳ گره:
  - اگر حافظه پر شد،
    - گره با بشترین f را حذف کن
    - بهترین گره فراموش شده را در هر گره والد با خاطر بسپار!

نکات مربوط به جستجوی SMA*



- با فرض وجود حافظه به اندازه ۳ گره:
  - اگر حافظه پر شد،
    - گره با بشترین f را حذف کن
    - بهترین گره فراموش شده را در هر گره والد با خاطر بسپار!
  - همانطور که مشخص است، گره‌های فرزند تک به تک به صف اضافه می‌شوند.
    - جلوگیری از سرریزی حافظه
    - امکان بررسی حذف گره در صورت نیاز

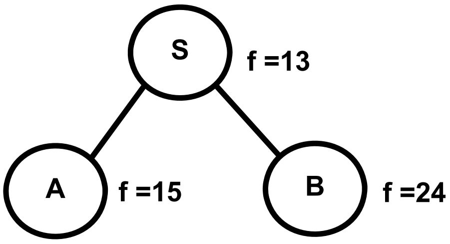
نکات مربوط به جستجوی SMA*



- با فرض وجود حافظه به اندازه ۳ گره:
  - مسیرهای طولانی: دست بردار!
    - امکان رسیدن به هدف از این مسیر با توجه به محدودیت حافظه وجود ندارد.
    - هزینه مسیر C برابر ∞ در نظر گرفته می‌شود.

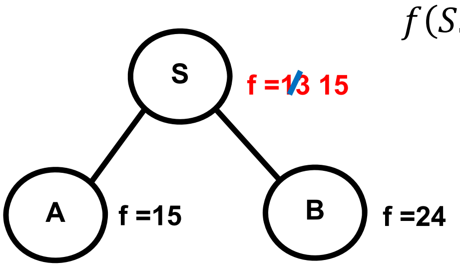
نکات مربوط به جستجوی SMA*

- به روز کردن مقادیر f
  - **اگر** تمامی فرزندان M_i گره N پیمایش شده‌اند
  - و $\forall i: f(S..M_i) > f(S..N)$



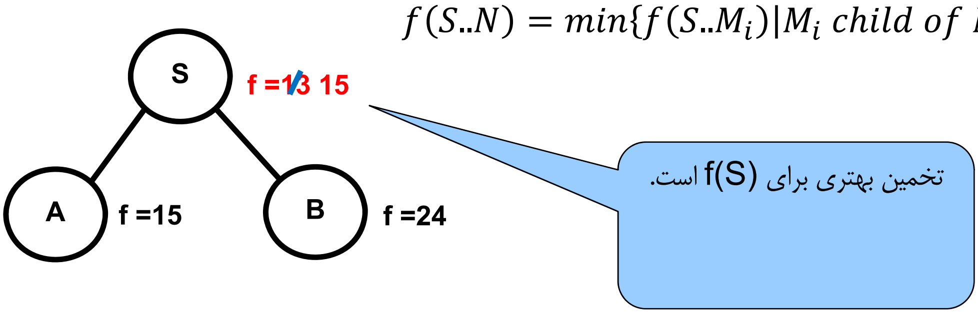
نکات مربوط به جستجوی SMA*

- به روز کردن مقادیر f
  - **اگر** تمامی فرزندان M_i گره N پیمایش شده‌اند
  - و $\forall i: f(S..M_i) > f(S..N)$
  - **آنگاه** $f(S..N) = min\{f(S..M_i) | M_i \; child \; of \; N\}$



نکات مربوط به جستجوی SMA*

- به روز کردن مقادیر f
  - **اگر** تمامی فرزندان M_i گره N پیمایش شده‌اند
  - **و** $\forall i: f(S..M_i) > f(S..N)$
  - **آنگاه** $f(S..N) = min\{f(S..M_i)|M_i\ child\ of\ N\}$

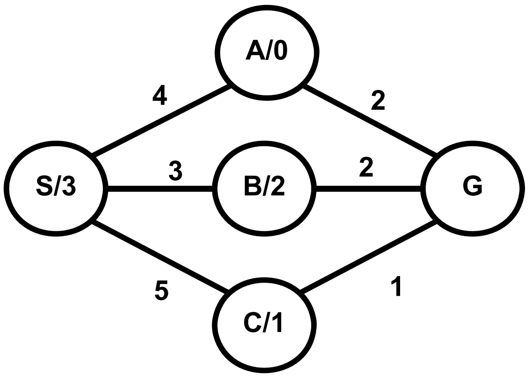


نکات مربوط به جستجوی SMA*

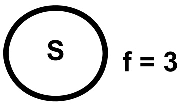
- بررسی هدف بودن:
  - در هر مرحله بزرگترین f انتخاب شده و بررسی می‌شود آیا هدف است یا خیر.
  - جستجو به نحوی پیش می‌رود که در هر گره پدر هم می‌دانیم با چه fای به هدف خواهیم رسید.

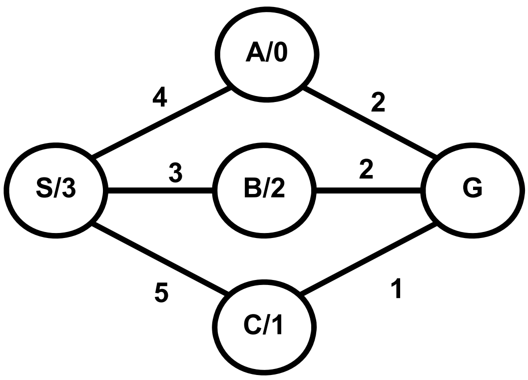
یک مثال از جستجوی SMA*

- با فرض وجود حافظه برای ذخیره حداکثر ۳ گره

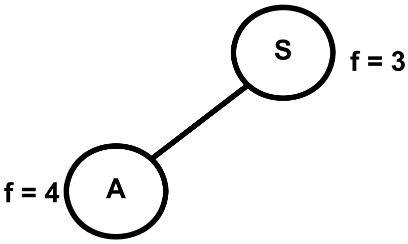


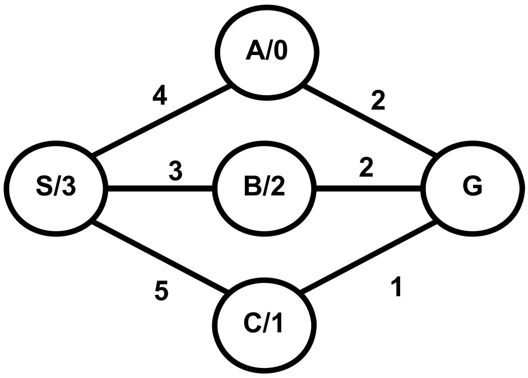
یک مثال از جستجوی SMA*



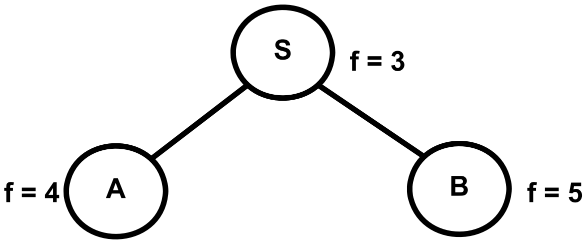


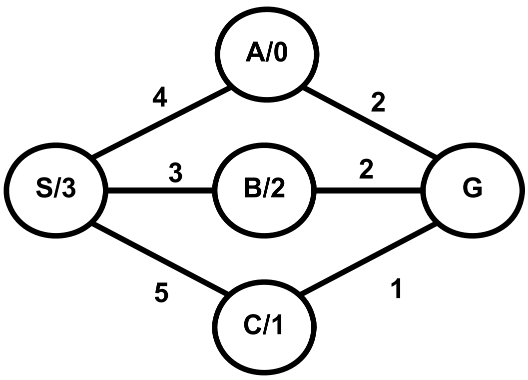
یک مثال از جستجوی SMA*



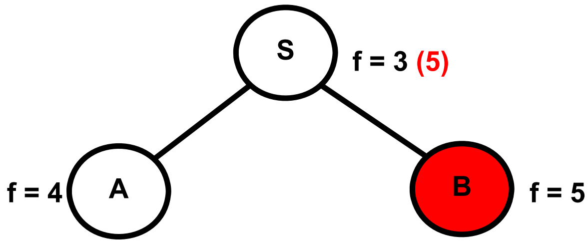


یک مثال از جستجوی SMA*

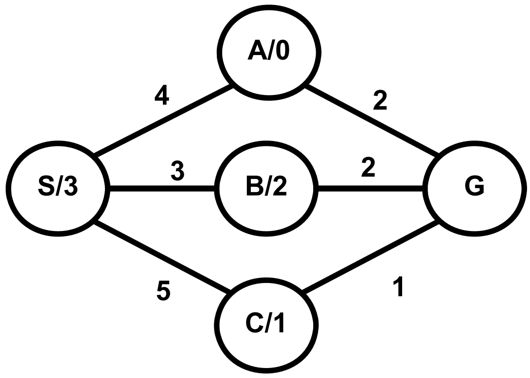




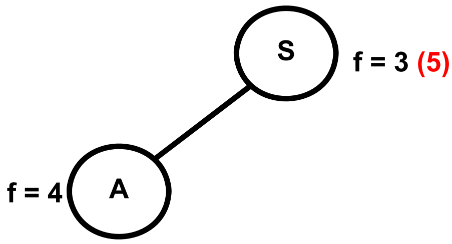
یک مثال از جستجوی *SMA

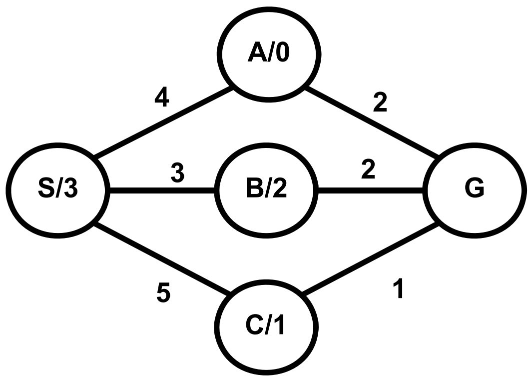


حافظه پر شده. گره با بیشترین f حذف می‌شود و کمترین f فراموش شده در گره پدر ذخیره می‌شود.

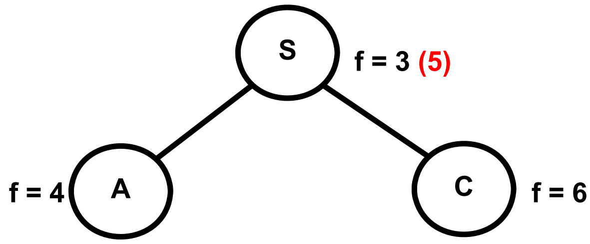


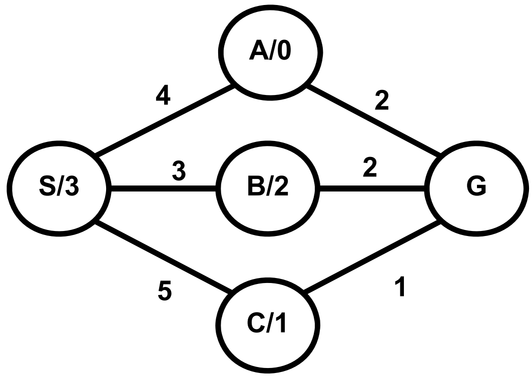
یک مثال از جستجوی *SMA





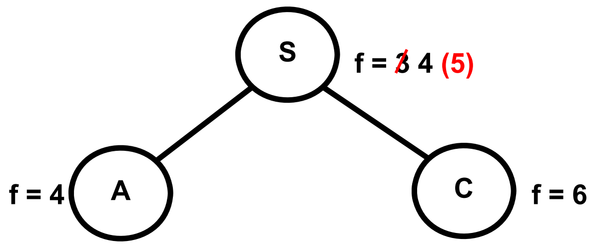
یک مثال از جستجوی SMA*

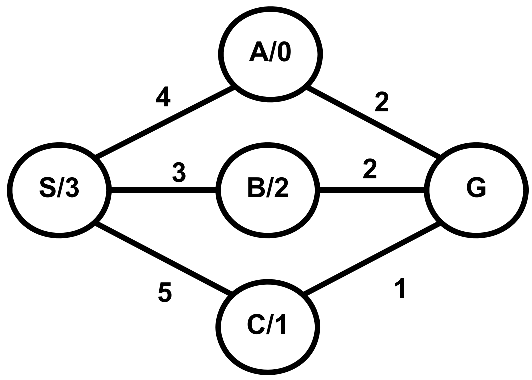




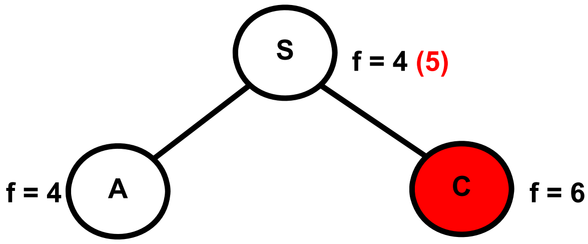
یک مثال از جستجوی SMA*

تمامی فرزندان S پیمایش شده‌اند. f آن با کمترین f فرزندانش به روز می‌شود.

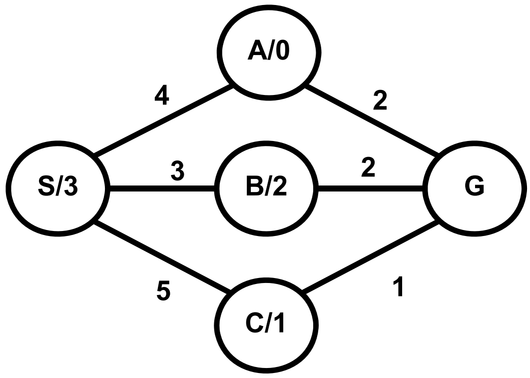




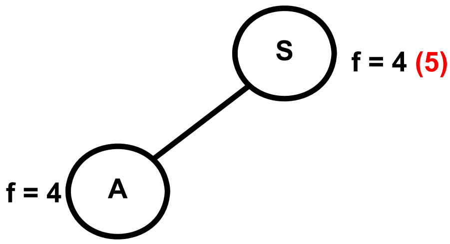
یک مثال از جستجوی SMA*

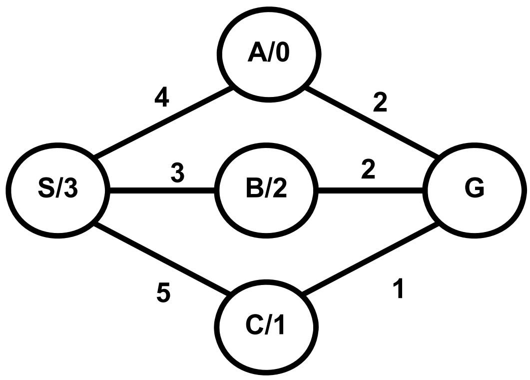


حافظه پر شده. گره با بیشترین f حذف می‌شود و کمترین f فراموش شده در گره پدر ذخیره می‌شود. (در اینجا چون ۵ ذخیره شده کمتر است، تغییری ایجاد نمی‌شود.)

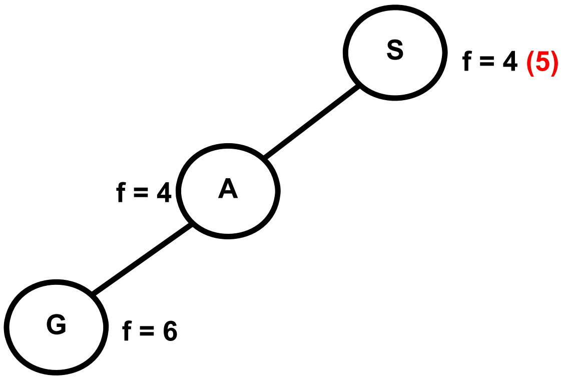


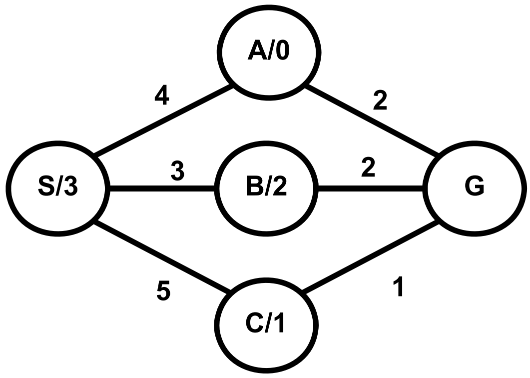
یک مثال از جستجوی SMA*





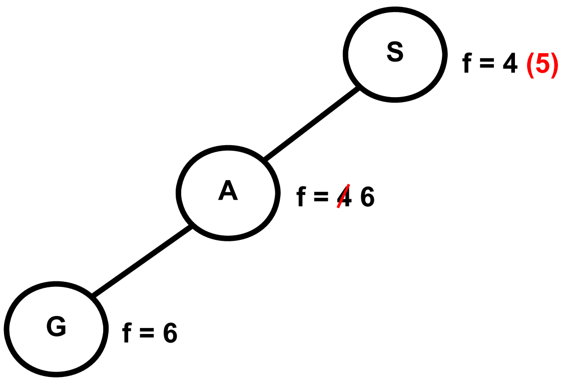
یک مثال از جستجوی SMA*

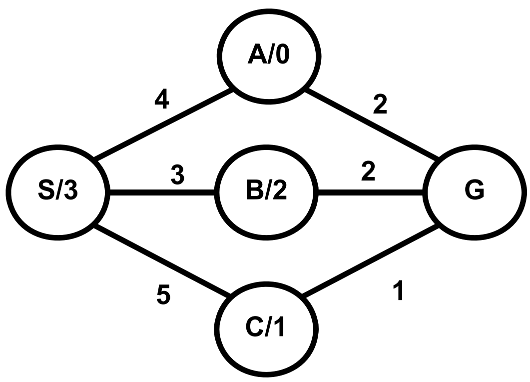




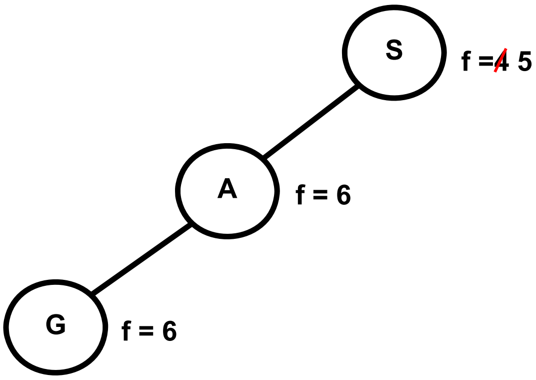
یک مثال از جستجوی SMA*

تمامی فرزندان A پیمایش شده‌اند. f آن با کمترین f فرزندانش به روز می‌شود.

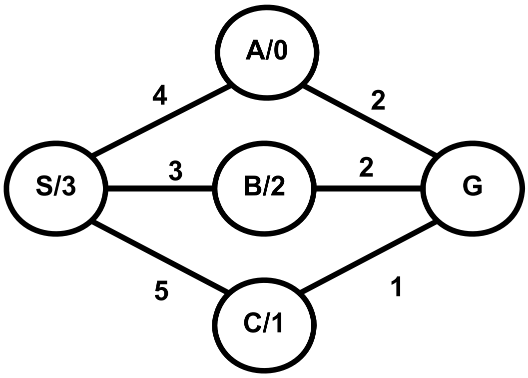




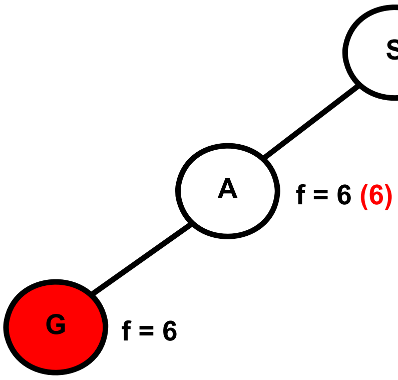
یک مثال از جستجوی SMA*

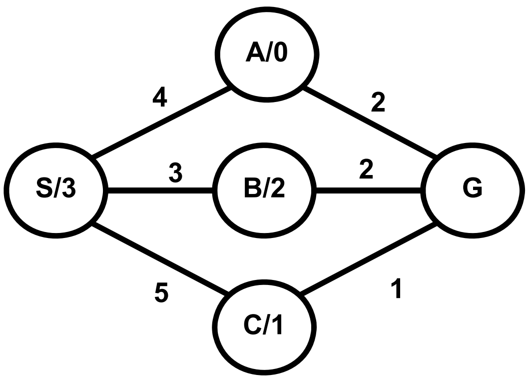


با توجه به تغییر f فرزند S (یعنی A)، f آن نیز باید مجددا به روز شود. f فرزند فراموش شده‌ی S کمترین f در بین فرزندان آنرا دارد. بنابراین f گره S با این مقدار به روز می‌شود.

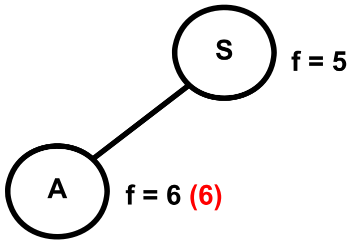


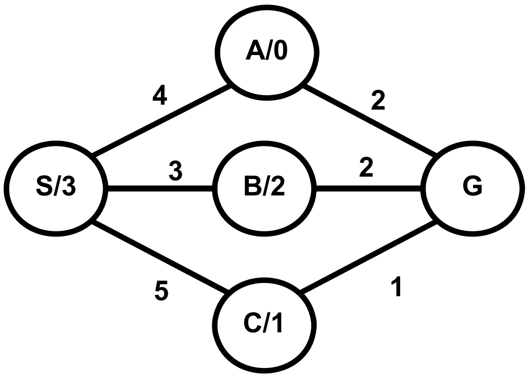
یک مثال از جستجوی SMA*



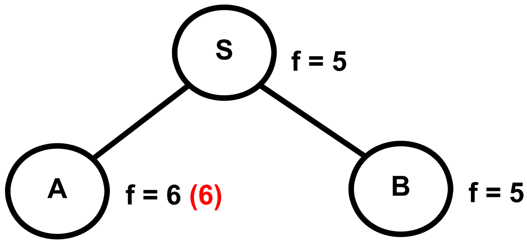


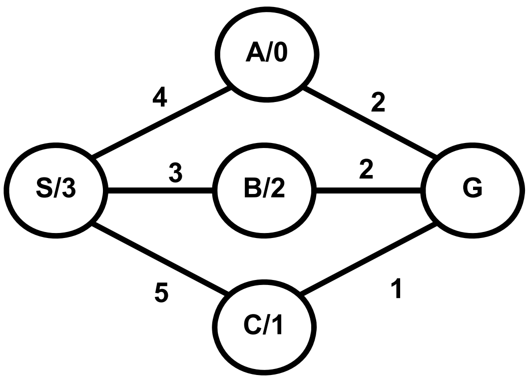
یک مثال از جستجوی SMA*



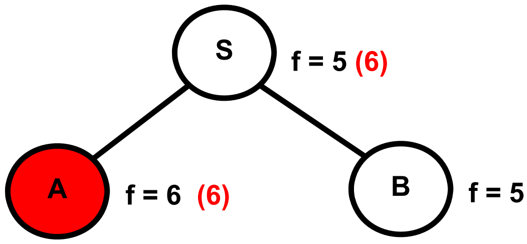


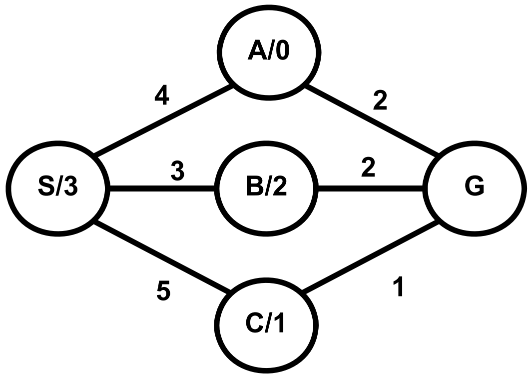
یک مثال از جستجوی SMA*



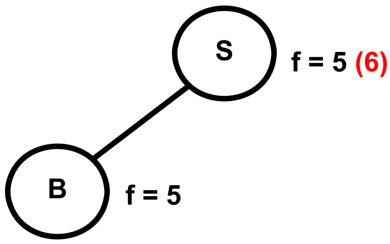


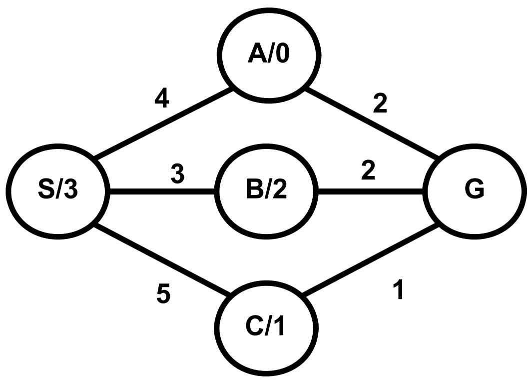
یک مثال از جستجوی SMA*



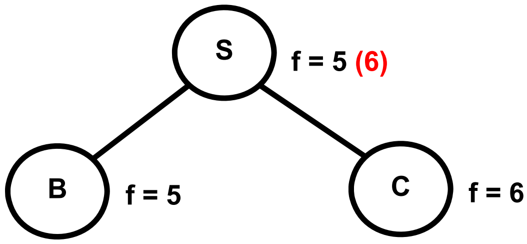


یک مثال از جستجوی SMA*

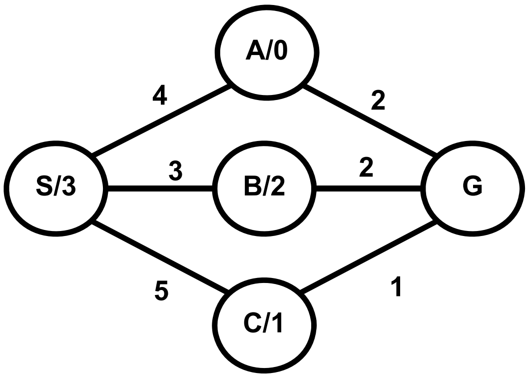




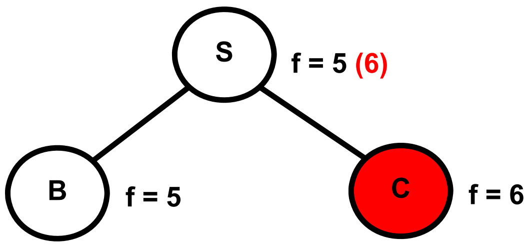
یک مثال از جستجوی SMA*

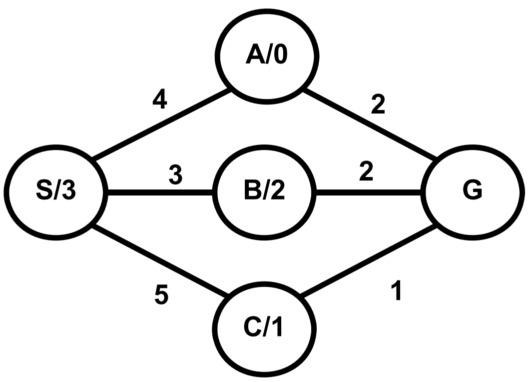


در این مرحله با توجه با اینکه همه فرزندان S یکبار پیمایش شده‌اند و می‌دانیم بهترین f فرزندان S برابر ۵ است (به دلیل به روز کردن f در S)، به نظر می‌رسد که تولید مجدد گره C دست کم در این مرحله ضرورتی ندارد. این مورد در الگوریتم اصلی نیامده است. اما می توان به آن فکر کرد!

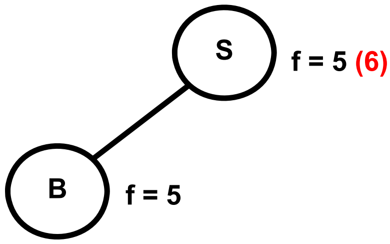


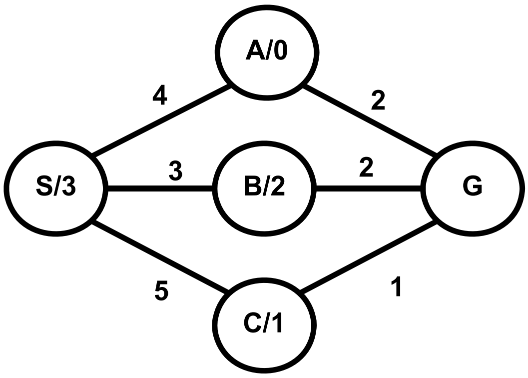
یک مثال از جستجوی SMA*



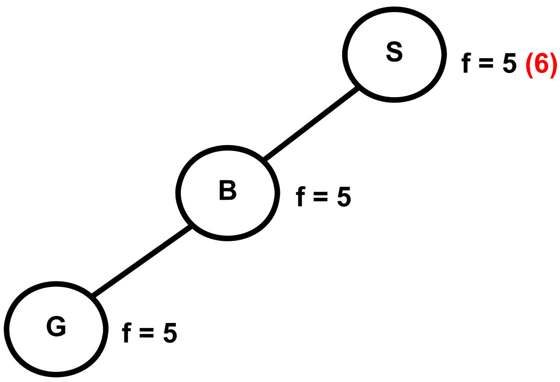


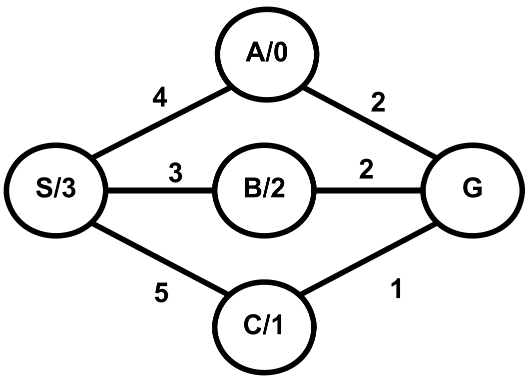
یک مثال از جستجوی SMA*



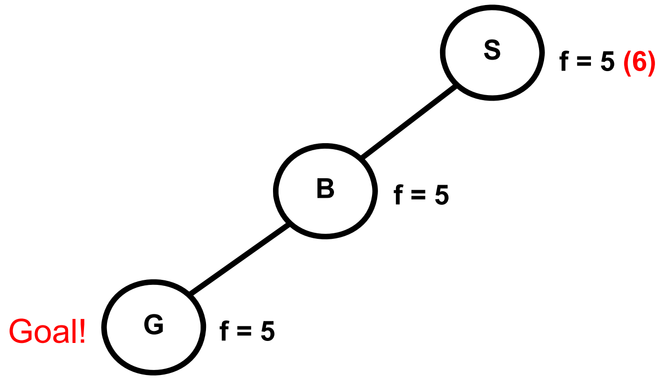


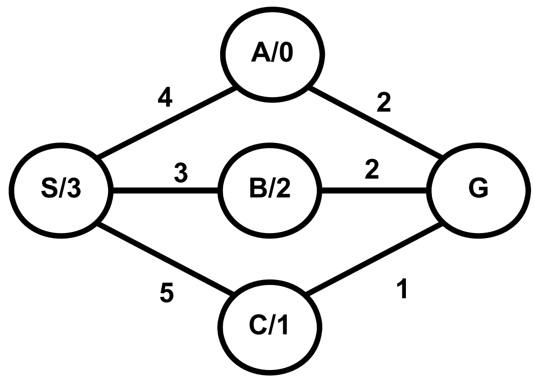
یک مثال از جستجوی SMA*





یک مثال از جستجوی *SMA





هوش مصنوعی ۸۷

- در مقایسه بین روش‌های مختلف جستجو از نظر حافظه‌بری اگر بخواهیم روش‌ها را از پیچیده‌ترین تا ساده‌ترین مرتب نماییم، کدام گزینه در اغلب موارد صحیح است؟

1. RBFS→Breadth First →SMA* →A*
2. Breadth First →A* →RBFS →SMA*
3. RBFS →Breadth First →A* →SMA*
4. Breadth First →A* →SMA* →RBFS

هوش مصنوعی ۸۷

- گزینه ۲ صحیح است.
  - جستجوی اول سطح به دلیل ناآگاهانه بودن و استفاده از صف (یا صف اولویت) نیاز به فضای حافظه زیادی دارد.
  - A* به دلیل آگاهانه بودن معمولا نیاز به فضای کمتری نسبت به جستجوی اول سطح دارد.
  - RBFS به دلیل استفاده از پشته، مصرف نمایی حافظه را به خطی تقلیل می‌دهد. اما در صورت وجود حافظه‌ی بیشتر قادر نیست از آن استفاده کند. (حداکثر می‌تواند b*d گره را در حافظه نگه دارد).
  - SMA* می‌تواند صف اولویت را تا اندازه ممکن گسترش دهد و فقط در صورت پر شدن حافظه عناصر را (با حفظ خصوصیات در گره پدر) حذف می‌کند. از این منظر از همه بهتر است.
    - RBFS از نظر زمان مناسب نیست.
    - IDA* چون در هر تکرار فقط یک مقدار f را ملاک قرار می‌دهد، نسبت به RBFS ضعیف‌تر عمل می‌کند.

آی تی ۸۵

- کدام یک از جملات زیر صحیح است؟

1. الگوریتم SMA* همیشه سریعتر از A* به جواب می‌رسد.
2. جستجوی کور همیشه نیاز به حافظه کمتری نسبت به جستجوهای مطلع دارد.
3. الگوریتم اول عمق همیشه با صرف مقدار کمتری از حافظه نسبت به الگوریتم اول پهنا به جواب می‌رسد.
4. الگوریتم جستجوی A* (با هر هیوریستیکی) همیشه تعداد کمتری گره نسبت به هر الگوریتم مطلع دیگر (با هر هیوریستیکی) بسط می‌دهد.

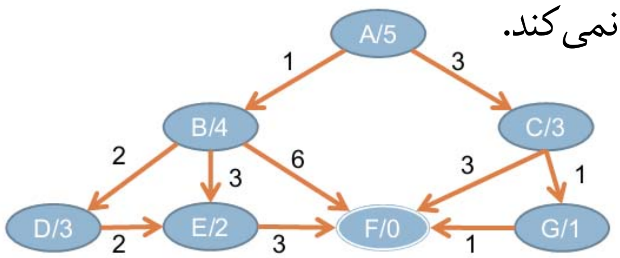
آی تی ۸۵

- گزینه ۳ صحیح است

هوش مصنوعی

- حاصل جستجوی *SMA با حداکثر ۳ خانه حافظه روی گراف زیر چیست؟ (A نقطه شروع است و F گره هدف است و اعداد روی یال‌ها هزینه‌ی مسیر و اعداد داخل دایره‌ها مقدار h گره مورد نظر است. ترتیب ملاقات فرزندان به ترتیب حروف الفبا است.)

1. ACF
2. ABF
3. ACGF
4. *SMA پاسخی برای این مسئله پیدا نمی‌کند.

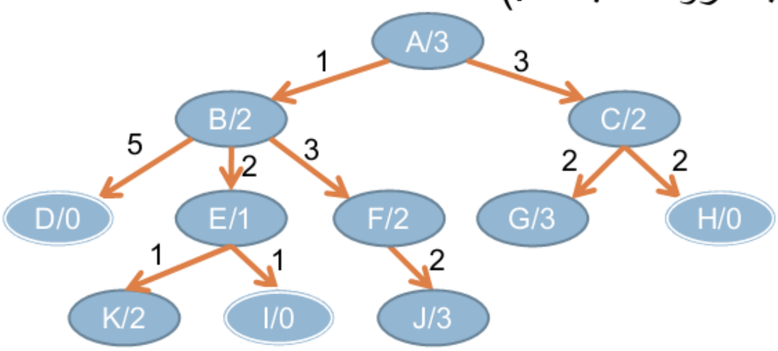


هوش مصنوعی

- گزینه ۱ صحیح است.
  - نکته تستی: کافی است مسیرهای با طول بیشتر از ۳ را حذف کرده، *A جستجو کنید

مکاترونیک ۸۶

- حاصل جستجوی *SMA با حداکثر ۳ خانهٔ حافظه بر روی گراف زیر چیست؟ (A نقطهٔ شروع است و گره های D,H,I هدف است و اعداد روی یال ها هزینهٔ مسیر و اعداد داخل دایره ها مقدار h گره مورد نظر است. ترتیب ملاقات فرزندان به ترتیب حروف الفباست.)



۱. ABD
۲. ACH
۳. ABEI
۴. *SMA پاسخی برای این مسأله پیدا نمی کند.

مکاترونیک ۸۶

- گزینه ۲ صحیح است

هوش مصنوعی ۸۳

- کدام یک از موارد زیر در خصوص روش جستجوی RTA* (Real Time A*) در مقایسه با روش A* صحیح‌تر است؟

1. RTA* اغلب تمایل بیشتری به ادامه مسیر جاری دارد.
2. RTA* همواره مسیرهای کوتاهتری را می‌یابد.
3. RTA* اغلب تمایل کمتری به ادامه مسیر جاری دارد
4. RTA* همواره مسیرهای طولانی را می‌یاد.

هوش مصنوعی ۸۳

- گزینه ۱ صحیح است
  - RTA* کمتر backtrack می‌کند

هوش مصنوعی ۸۲

- نقطه ضعف روش جستجوی IDA* چیست؟
  1. کامل نبودن
  2. دوباره کاری
  3. کارایی پایین
  4. مصرف حافظه زیاد

هوش مصنوعی ۸۲

- گزینه ۲ صحیح است

هوش مصنوعی ۹۲

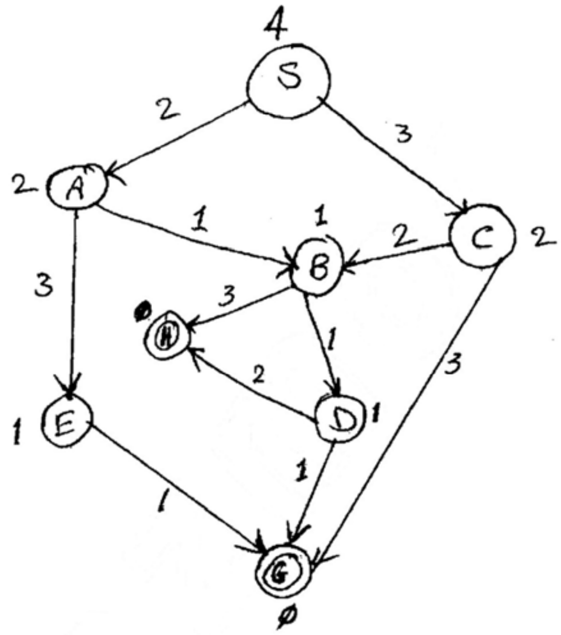
۱۱۷- در گراف زیر، H و G گره‌های هدف و S گره شروع است. کدام یک از موارد زیر به ترتیب از چپ به راست، گره‌های ملاقات شده توسط جستجوی A^* را نشان می‌دهد؟ هزینه هر انتقال از یک گره به گره دیگر روی یال واصل و هزینه تخمینی هر گره تا هدف در کنار آن گره نوشته شده است. در شرایط مساوی به گره‌ای که زودتر تولید شده است، اولویت دهید.

۱) SABCDG

۲) SACBDH

۳) SABDH

۴) SABDG

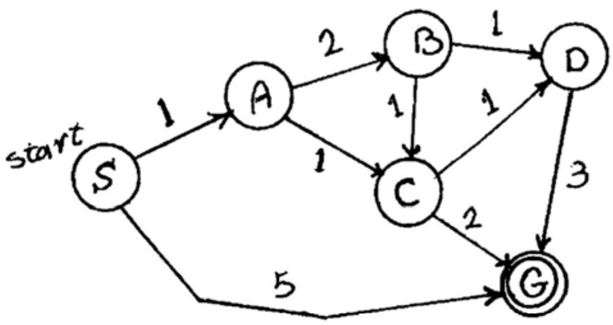


هوش مصنوعی ۹۲

گزینه ۴ صحیح است (بررسی کنید).

هوش مصنوعی ۹۲

۱۱۸- در گراف جستجوی زیر، S گرهٔ شروع و G گرهٔ هدف است. پیمایش گره‌ها در شرایط مساوی بر اساس ترتیب الفبا صورت می‌گیرد. بر اساس دو تابع هیوریستیک نشان داده شده در جدول زیر، کدام گزاره صحیح است؟



گره	h_1	h_2
S	۳	۴
A	۳	۲
B	۴	۳
C	۲	۱
D	۳	۱
G	۰	۰

۱) هر دو تابع هم admissible و هم consistent هستند.

۲) هر دو تابع admissible هستند. اما فقط h_1، consistent است.

۳) هیچ یک از دو تابع admissible نیست. اما فقط h_2، consistent است.

۴) هیچ یک از دو تابع consistent نیست. اما فقط h_2، admissible است.

هوش مصنوعی ۹۲

- گزینه ۴ صحیح است.
  - در h_1، مقدار تخمین زده شده برای گره B از هزینه واقعی بیشتر است. پس h_1 قابل قبول نیست بنابراین یکنوا هم نیست. h_2 قابل قبول هست اما یکنوا نیست (بررسی کنید).

آی تی ۹۲

۸۲- فرض کنیم $h_1(n)$، $h_2(n)$ و $h_3(n)$ سه تابع مکاشفه‌ای قابل قبول باشند. یک الگوریتم A^* با کدام یک از توابع $h(n)$ زیر جواب بهینه را تولید می‌کند؟

۱) $h(n) = h_1(n) + h_2(n) + h_3(n)$

۲) $h(n) = h_1(n) * h_2(n) * h_3(n)$

۳) $h(n) = \max(\min(h_1(n), h_2(n), h_3(n)), h_1(n) * h_2(n) * h_3(n), h_1(n) + h_2(n) + h_3(n))$

۴) $h(n) = \min(\max(h_1(n), h_2(n), h_3(n)), h_1(n) * h_2(n) * h_3(n), h_1(n) + h_2(n) + h_3(n))$

آی تی ۹۲

- گزینه ۴ صحیح است.