«بنام خدا»

تمرین هوش مصنوعی HW3

Year: Month: Date: (۹۹۵۲۱۴۹ سینا علی نژاد) Subject

سوال ۱-

الف)

initialize: Self.frontier = [S]

expanded = [] , S.g = ۰ , next = S

1) self.frontier $\underset{Sort}{=}$ [D, A] , exp = [S]

(D: ۷ , A: ۱۲)

D.g = ۲ , A.g = ۳

P(D) = P(A) = S next = D

2) frontier $\underset{Sort}{=}$ [B, E, A]

(B: ۷ , E: ۹ , A: ۱۲)

exp = [S, D]

S.g = ۰ , D.g = ۲ , A.g = ۳ , B.g = ۳ , E.g = ۴

P(B) = P(E) = D P(D) = P(A) = S

next = B

3) frontier = [C, E, A]

(C: ۷ , E: ۷ , A: ۱۲)

exp = [S, D, B]

D.g = ۲ , A.g = ۳ , B.g = ۳ , E.g = ۴, C.g = ۵

P(E) = P(C) = B , P(B) = D , P(D) = P(A) = S

next = C

4) frontier = [E, G, A] , exp = [S, D, B, C]

(E: ۷ , G: ۹ , A: ۱۲)

G.g = ۹ , C.g = ۵ , E.g = ۴ , B.g = A.g = ۳

D.g = ۲ P(E) = P(C) = B , P(B) = D , P(D) = P(A) = S

P(G) = C next = E

Scanned with CamScanner

Year: Month: Date: ( ) Subject

5) frontier = [G, A] exp = [S, D, B, C, E]
v ۱۲

G.g = ۷ , E.g = ۱۰ , B.g = A.g = ۳ , D.g = ۲ , C.g = ۵

P(G) = E , P(E) = B , P(B) = D , P(D) = S

P(A) = S , P(C) = B next = G → finish

shortest path = S D B E G : ۷

(ب)

initialize: frontier = [S] , exp = [ ]
next = S

1) frontier = [D, A] , exp = [S]
sort ۵ h ۹

next = D P(D) = S

2) frontier = [E, B] , exp = [S, D]
۳ ۴

next = E P(E) = D

3) frontier = [G] , exp = [S, D, E]
۰

next = G → finish , P(G) = E

answer: SDEG : ۹

Scanned with CamScanner

همانطور که مشخص است الگوریتم A* جواب درست را داده است اما الگوریتم حریصانه جواب درست یا بهینه نداده است. ۷ < ۹

مزیت الگوریتم حریصانه در این است که حافظه کمتری نیاز دارد زیرا در frontier فقط همسایه‌های گرهی که expand شده ذخیره می‌شود و فقط اعراف خود را می‌بیند و بهترین را انتخاب می‌کند.

از نظر زمانی هم الگوریتم حریصانه باز هم بهتر است ولی لزوماً جواب درست را نمی‌دهد.

الگوریتم A* مسیری که تا الان طی شده را نیز در نظر می‌گیرد و با آگاهی از گذشته و آینده پیش‌بینی شده بهترین تصمیم را می‌گیرد.

سوال ۲-

الف)

BCDFG ⟶ $h^*(B) = 12$

shortest Path

$\longrightarrow h(B) \leq h^*(B) = 12$

$h(B) \geq 0 \longrightarrow 0 \leq h(B) \leq 12$

ب)

AB: $10 \leq 1 + h(B) \longrightarrow h(B) \geq 9$ (۱)

BA: $h(B) \leq 1 + 10 \longrightarrow h(B) \leq 11$ (۲)

Scanned with CamScanner

Year: Month: Date: ( ) Subject

BC: $h(B) \leq 1+9 \rightarrow h(B) \leq 10$ (۳)

CB: $9 \leq 1+h(B) \rightarrow h(B) \geq 8$ (۴)

BD: $h(B) \leq 5+7 \rightarrow h(B) \leq 12$ (۵)

DB: $7 \leq 5+h(B) \rightarrow h(B) \geq 2$ (۶)

(۱) ∩ (۳) ∩ (۴) ∩ (۵) ∩ (۶) ⇒ $9 \leq h(B) \leq 10$

ج) بین B و C ابتدا C به انتها می‌رود پس:

$f(C) < f(B) \longrightarrow 4+9 < 1+h(B)$

$\rightarrow h(B) > 12$

بین B و D ابتدا B به انتها می‌رود پس

$f(B) < f(C) \rightarrow 10+h(B) < 7+7 \rightarrow h(B) < 13$

$12 < h(B) < 13$

اینجا اگر از مسیر ACB

محاسبه می‌کردیم:

$f(B) = 5+h(B) > 1+h(B)$

پس $f(B)$ را آپدیت نمی‌کنیم.

Scanned with CamScanner

Year: Month: Date: ( ) گزارش کد Subject

توضیح تابع Solve Simple Back Tracking :

در این تابع ابتدا به کمک تابعی به اسم getNextLocation یک خانهٔ خالی به ما داده می‌شود. اگر خانه‌ای خالی نبود پس مسئله حل شده، Return True می‌شود، در غیر اینصورت روی همهٔ مقادیر ۱ تا dim یک حلقه زده شده و هر کدام که Safe باشد، انتخاب می‌شود و در آن خانه قرار داده می‌شود، سپس به طور بازگشتی این تابع صدا زده می‌شود، اگر نهایتاً کل جدول پر شود تمام است اما اگر به جایی برسیم که هیچ گزینه‌ای برای یک خانه ممکن نبود، یک مرحله به عقب رفته و در حلقه‌ای که گفتیم، عدد بعدی را امتحان می‌کنیم.

تابع isSafe : این تابع طبق قوانین بازی سودوکو بررسی می‌کند که آیا یک عدد در یک خانه می‌تواند قرار بگیرد یا نه.

check_row, check_column, check_box

تابع getRemainingValues :

این تابع به کمک تابع getDomain دامنه مقادیر خانه‌ها را آپدیت می‌کند. برای خانه‌های پر '0' می‌گذارد.

تابع getDomain(row,col) : این تابع دامنه مقادیر را برای خانهٔ واقع در ردیف row، ستون col آپدیت می‌کند. بدین صورت که ابتدا همهٔ مقادیر را برای آن در نظر می‌گیرد سپس با توجه به مقادیر واقع در آن ردیف، آن ستون و آن باکس از این مقادیر حذف می‌کند.

Scanned with CamScanner

Year: Month: Date: ( ) Subject

تابع SolveCspBackTracking :

این تابع با Solve SimpleBackTracking فرق‌هایی دارد. در اینجا روی همه مقادیر دامنه یکی مسئله حلقه زده نمی‌شود بلکه روی همان دامنه مقادیر مربوط به آن خانه حلقه داریم. بعد از اینکه به این خانه مقدار دادیم، باید دوباره دامنه مقادیر بروز شود.

self.rv = self.getRemainingValues()

در این مسئله، شاید خیلی با این روش بهبودی حاصل نشود، بعد از تابع گذاشتن مشخص شد که در حالت simple ۷ میلی ثانیه و در حالت Csp ۹ میلی ثانیه حل شد. در واقع اینجا بیشتر داشتیم ولی در مسائل بزرگتر بهتر بودن روش Csp خود را بیشتر نشان می‌دهد.

در واقع اینجا می‌دانیم چه مقادیری می‌توانیم در هر خانه بگذاریم ولی از آن طرف باید دامنه مقادیر خانه‌ها را آپدیت کنیم که این خودش سربار است.

راه‌های برای بهبود:

۱- هر بار همه خانه‌ها را آپدیت نکنیم، مثلاً وقتی خانه‌ای را پر دامنه مقادیر

Scanned with CamScanner

Year: Month: Date: ( ) Subject

می‌کنیم، این خانه روی خانه‌های هم ردیف و هم ستون و هم باکس خودش تاثیر می‌گذارد و کافیست فقط همان را تغییر دهیم.

۲- می‌توان از روش‌های forward checking و یا در حالت بهتر Arc consistency استفاده کرد، بدین صورت که در هر مرحله بین خانه‌های هم ردیف و هم ستون و هم باکس چک می‌کنیم که برای هر انتخاب از یکی، برای آن یکی هم انتخابی داشته باشیم، اگر برای دو خانه هم ردیف مثلاً دامنه مقادیر زیر را داشتیم، پس نمی‌توانیم برای یکی از آن‌ها مقداری قرار دهیم.

$$d_a = d_b = [1]$$

با این کار زودتر آگاه می‌شویم که باید برگردیم و مقدار دیگری امتحان کنیم.

در روش forward checking هم هر موقع خانه‌ای را آپدیت کردیم، چک می‌کنیم خانه‌ای دامنه مقادیر خالی نداشته باشد. در اینجا هم یک مرحله جلوتر را می‌بینیم.

۳- ایجاد یک ordering مناسب برای مثل خانه‌ای را برای پر کردن انتخاب کند، که دامنه مقادیر کمتری بتوان برای آن قرار داد.

★ مورد ۳ را در تابع getNextLocation انجام دادم که باعث شد در حالت Csp از ۲۴ میلی ثانیه به ۸ میلی ثانیه رسید، تعداد expandedNodes هم از ۱۱۰ به ۴۴ رسید که پیشرفت خوبی است.

Scanned with CamScanner

Year: Month: Date: ( ) Subject

مورد ۱ رو هم انجام دادم و توابع update_domains_on_change و update_domains_on_removal برای همین مورد نوشته شده.

برای فایل b.txt تعداد expanded Nodes از ۹۷۶ م

عدد ۸۵ رسید که پیشرفت بزرگی است.