تمرین تئوری اول هوش مصنوعی رادین شاه دائی ۴۰۱۱۰۶۰۹۶

سوال اول)

الف) درست است. فرض کنید گرافی داریم که شامل یک دور است. از یک راس بیرون این دور شروع می‌کنیم و با روش DFS به این دور در راس v می‌رسیم. پس از اینکه کل دور پیمایش شد به راس v بر می‌گردیم و دوباره این دور را پیمایش می‌کنیم. همین روند به طور نامتناهی ادامه پیدا می‌کند و درخت جست و جوی DFS نامتناهی می‌شود.

ب) درست است، داریم:

$$f(n), g(n) < h^*(n) \rightarrow 0.7f(n) < 0.7h^*(n) \text{ و } 0.2g(n) < 0.2h^*(n)$$
$$0.7f(n) + 0.2g(n) < 0.9h^*(n) < h^*(n)$$

ج) نادرست، اگر در local beam search همواره بهترین k فرزند را انتخاب کنیم، به احتمال زیاد این k فرزند در یک اپتیمای local گیر می‌کنند. از این رو از روش انتخاب تصادفی استفاده می‌شود که بین حالات مختلف تمایز بیشتری قائل شویم و احتمال همگرایی به local min/max کاهش یابد.

د) درست است. فضای حالتی را در نظر بگیرید که n استیت پشت سر هم آمده‌اند تا به استیت هدف برسیم. در جست و جوی DFS مستقیما و پس از طی n مرحله به استیت هدف می‌رسیم. اما در جست و جوی IDS قبل از اینکه به عمق n برسیم، تمام عمق‌های قبل از آن چک می‌شوند و دوباره از اول جست و جو تکرار می‌شود. در نتیجه پیچیدگی زمانی IDS در این فضا از $O(n^2)$ است درحالی که برای DFS از $O(n)$ بود.

هـ) نادرست، به وضوح برای هر دو حالت، اگر درجه انشعاب نامتناهی باشد، نمی‌توانیم بیش از یک طبقه پیش برویم و در طبقه‌ی اول (همسایه‌های استیت شروع) گیر می‌کنیم. در نتیجه در هر دو جست و جوی BFS و IDS این شرط را داریم که درجه انشعاب یا b باید متناهی باشد.

و) درست، اگر تابع هزینه یا $g(n)$ را معادل $-depth(n)$ قرار دهیم و هر بار از $fringe$، کمترین g را انتخاب کنیم، در هرمرحله بیشترین عمق انتخاب می‌شود که معادل جست و جوی DFS است.

ز)

- Fully observable هست زیرا کل زمین بازی را مشاهده می‌کنیم.

- Single agent هست زیرا فقط یک عامل هوشمند داریم که می‌خواهد به هدف برسد.

- Deterministic هست زیرا در هر استیت می‌دانیم با انجام یک حرکت به چه استیتی خواهیم رفت. (در ورژنی از بازی که من با آن آشنا هستم در هر مرحله مهره‌ی بعدی نیز مشخص است. https://tetris.com/play-tetris)

- Episodic نیست و sequential است زیرا هر استیت به استیت‌های قبلی و حرکت‌های انجام شده بستگی دارد.

- discrete است زیرا زمین بازی یک جدول متنهای است و نمی‌توان مهره‌ای را جایی بین خانه‌های جدول قرار داد.

سوال دوم)

در بخش‌های BFS و DFS این سوال در نظر گرفته می‌شود اگر چند راس عمق یکسان داشتند، اولویت انتخاب از fringe راسی‌ست که هزینه یال ورودی به آن که با آن یال به fringe وارد شده کمتر است. محاسبات درون چک‌نویس انجام شده‌است و در اینجا وارد نشده‌است.

DFS: S -> B -> G

Greedy: S -> A -> E -> G (argmin(h) between neighbors)

UCS: S -> B -> A -> E -> F -> C -> G (argmin(g) between neighbors)

A*: S -> A -> E -> G (argmin(g+h) between neighbors)

BFS: S -> B -> A -> C -> E -> G

سوال سوم)

آ) مقادیر توابع fitness در زیر آمده است:

$Fitness(x_1) = 47$, $Fitness(x_2) = 19$, $Fitness(x_3) = 74$, $Fitness(x_4) = 75$

ب) عملیات crossover برای x_3 و x_4:

First ½ x_4 + Second ½ x_3 = 37914548 → fitness = 64 (c1)

First ½ x_3 + Second ½ x_4 = 89230977 → fitness = 85 (c2)

عملیات CDEF crossover برای x_1 و x_3:

First ¼ x_1 + Middle ½ x_3 + Last ¼ x_1 = 21234565 → fitness = 35 (c3)

First ¼ x_3 + Middle ½ x_1 + Last ¼ x_3 = 89832948 → fitness = 86 (c4)

ج)

72 on c1 → fitness = 68

47 on c2 → fitness = 88

15 on c3 → fitness = 50

75 on c4 → fitness = 96

د) بله، مجموع fitness ها به مقدار ۸۷ واحد افزایش داشته است.

ه) طبق تعریف تابع fitness، واضح است که کروموزوم بهینه به صورت 99900999 می‌باشد. اگر mutation نکنیم و فقط crossover داشته باشیم، برای رسیدن به این رشته‌ی بهینه باید حداقل در یکی از کروموزوم های اولیه، رقم پر ارزش آن 9 می‌بود که اینگونه نیست. پس نمی‌توانیم به رشته‌ی بهینه برسیم.

سوال چهارم)

الف) ۱) این تابع admissible نیست. مثال نقضی را در برا n=4 در نظر بگیرید. 2 تا از مهره‌ها در یک خانه هستند و 2 دیگر در خانه‌ای مجاور مهره‌های قبلی قرار دارند. واضحا با 1 حرکت به استیت goal می‌رسیم. تابع heuristic ما اما مقدار 4 = 2 * 2 را گزارش می‌کند. این مقدار برای n > 2 بیشتر از *h است که نشان می‌دهد این heuristic، admissible نیست. با توجه به اینکه admissible بودن شرط لازم monotonic بودن است، پس این heuristic، monotonic هم نیست.

۲) این تابع admissible است؛ اثبات: اگر فاصله‌ی دو مهره‌ی نزدیکترین 4 باشد، با توجه به اینکه در سریع ترین حالت ممکن این دو مهره با شروع به حرکت به سمت دیگر در ۲ مرحله به یکدیگر می‌رسند، پس حداقل تعداد حرکات لازم برای رسیدن به استیت goal برابر ۲ حرکت است که مقدار تابع heuristic ما نیز می‌باشد. در شرایطی که تابع heuristic مقدار صفر را داشته باشد، واضح است که overestimate نکرده‌است. پس این تابع heuristic، admissible می‌باشد. اما این تابع monotonic نیست. در نظر داریم که برای monotonic بودن باید داشته باشیم که با توجه به اینکه هزینه‌ی هر حرکت 1 است، اختلاف مقدار تابع heuristic دو استیت متوالی از 1 نباید بیشتر بشود. اما مثالی را در نظر بگیرید که n خیلی بزرگ است و فاصله نزدیک‌ترین دو مهره برابر 3 است. این یعنی با انجام دو حرکت به خانه‌ی میانی فاصله‌شان می‌رسند و تابع heuristic برابر 0 می‌باشد. حال فرض کنید این دو مهره را یک قدم به عقب می‌بریم. در این حالت فاصله‌ی نزدیک ترین دو مهره برابر 5 می‌شود و تابع heuristic مقدار 2 را می‌گیرد. اختلاف h دو استیت متوالی از 1 بیشتر شد پس این تابع heuristic، monotonic نیست.

۳) این تابع admissible نیست. به مثال نقض زیر برای n=4 توجه کنید، هر x نشان دهنده‌ی یک مهره است.

	x		
x	G		x
	x		

در این مثال با ۲ حرکت می‌توانیم به استیت goal برسیم. (در این استیت همه‌ی مهره ها در خانه‌ی G قرمز قرار دارند) اما مقدار تابع heuristic مقدار 2.5 را گزارش می‌کند که این یعنی overestimate کرده است. با توجه به اینکه اگر تابعی admissible نباشد، monotonic هم نیست، پس این تابع monotonic هم نیست.

۴) با توجه به اینکه monotonic بودن شرط کافی برای admissible بودن است، اگر اثبات کنیم تابع ما monotonic است، اثبات کردیم که admissible هم هست. همانطور که در تابع دوم استدلال شد، برای monotonic بودن باید اثبات کنیم اختلاف تابع heuristic دو استیت متوالی از 1 بیشتر نمی‌شود. مقدار حداکثری $|x_i - x_j|$ در هر مرحله، با توجه به اینکه این دو مهره در بهترین حالت ۲ خانه به هم از نظر مختصات x نزدیک می‌شوند، به اندازه‌ی 2 تغییر می‌کند. دقیقا همین استدلال را برای $|y_i - y_j|$ به کار می‌بریم. در نتیجه ماکسیموم این دو تغییر مقدار 2 می‌باشد که نصف آن 1 است. پس این تابع heuristic، monotonic است پس admissible هم هست.

ب) هر دو استیت همسایه در مسئله جدید را به صورت دوتایی مرتب (x_i , x_j) در نظر می‌گیریم. همچنین هر دو استیت همسایه در مسئله‌ی قبلی را (y_i , y_j) در نظر می‌گیریم. از (x_i , x_j) به (y_i , y_j) یک تابع یک‌به‌یک وجود دارد.

حال این ادعا را اثبات می‌کنیم. در مسئله‌ی جدید تعداد مهره‌ها کمتر از تعداد مهره‌ها در مسئله‌ی قدیمی است، در نتیجه در زمین‌های یکسان می‌توان استیت x_i مسئله‌ی جدید را می‌توان به یک استیت y_i مسئله‌ی قدیمی map کرد. اگر بتوانیم از استیت x_i به x_j برویم و همچنین x_i به y_i مپ بشود و x_j به y_j مپ بشود، از y_i می‌توان به y_j رفت. زیرا حرکات ما در مسئله‌ی جدید زیر مجموعه‌ی حرکات ما در مسئله‌ی قدیمی است. فقط باید دقت کنیم که مهره‌های اضافه را در دو استیت y درست و در جای یکسانی بچینیم. دقت کنید که استیت‌های y که از استیت‌های x بدست آمده‌اند دقیقا همان مهره‌های استیت x را در جای خودشان دارند.

با توجه به اینکه می‌توان هر دو استیت همسایه در مسئله‌ی جدید را به دو استیت همسایه در مسئله‌ی قدیمی مپ کرد (دقت کنید که مسئله‌ی قدیمی relax تر است) و تابع h برای استیت‌های همسایه در مسئله‌ی قدیمی monotonic محسوب می‌شد، نتیجه می‌گیریم که تابع h برای مسئله‌ی جدید نیز monotonic است.

سوال پنجم)

الف) هر استیت را یک حالت از ماتریس دودویی n در n می‌گیریم. رنگ سیاه را با مقدار 0 و رنگ سفید را با مقدار 1 نمایش می‌دهیم. برای حالت اولیه مسئله همان ماتریس دودویی اولیه و حالت نهایی نیز ماتریس تمام یک می‌باشد. در هر استیت، می‌توان یک زیر ماتریس را انتخاب کرد تا کل درایه‌های آن را not کنیم. برای انتخاب این زیر جدول C(n+1,2)^2 حالت داریم. (انتخاب خطوط شروع و پایان هر یک از ضلع‌های زیر ماتریس) پس ضریب انشعاب این مسئله نیز C(n+1,2)^2 می‌باشد.

ب) n^2 تا درایه داریم که هرکدام را می‌توان با یکی از اعداد 0 و 1 پر کرد. در نتیجه فضای مسئله از $O(2^{n^2})$ می‌باشد.

ج) تابع heuristic انتخاب شده به صورت زیر است:

$$h(M) = 1 - \frac{2\sum_{i=0}^{n}(i \times IsTopLeftSquare(i))}{n(n+1)}$$

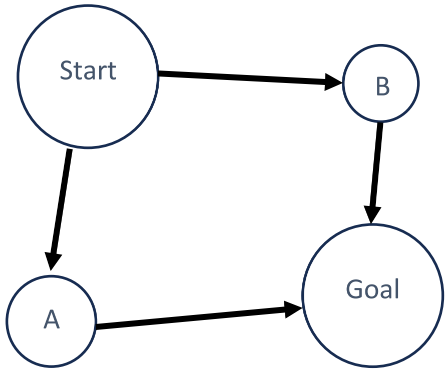
تابع IsTopLeftSquare(i) یک $Indicator\ Function$ است که برای هر i چک می‌کند که مربعی سفید به طول ضلع i که راس $Top\ Left$ آن درایه‌ی (0,0) ماتریس باشد. اگر اینگونه بود خروجی 1 می‌دهد و در غیر این صورت خروجی 0 می‌دهد.

- این تابع monotonic و admissible است. این ادعا را اثبات می‌کنیم:

حداکثر مقدار این تابع نسبت به M یا $\underset{M}{argmin}(h)$ در M کامل یک می‌باشد. با توجه به اینکه در تمام i ها از $n \sim 1$ خروجی 1 می‌دهد، مقدار Σ برابر $\frac{n(n+1)}{2}$ می‌باشد که در h را برابر 1 می‌کند. همچنین اگر تغییری در M ایجاد کنیم، حداکثر مقدار 1 تغییر می‌کند. با توجه به اینکه هزینه‌ی هر عمل ما 1 می‌باشد، اختلاف h بین دو استیت متوالی (و در واقع بین هر دو استیتی) حداکثر 1می‌باشد. پس این تابع monotonic است $(h(s_1) + h(s_2) \leq 1)$ و چون monotonic بودن شرط کافی برای admissible بودن، تابع معرفی شده monotonic و admissible است.

سوال ششم)

(۱) گراف زیر را در نظر بگیرید:



مقادیر زیر را برای هزینه‌ی مسیرها با تابع heuristic در نظر بگیرید:

Cost(Start → B) = 5

Cost(Start → A) = 4

Cost(A → Goal) = 3

Cost(B → Goal) = 1

h(Start) = 5

h(A) = 1

h(B) = 3

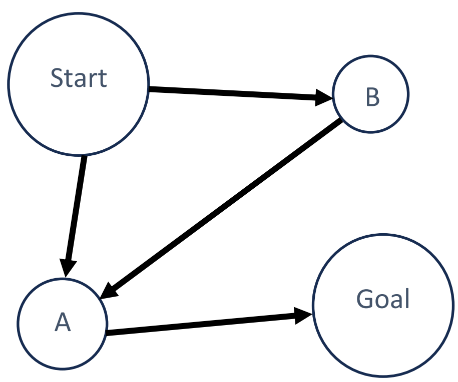
h(Goal) = 0

واضحا این تابع admissible نیست زیرا h(B) مقدار هزینه‌ی مسیر از b به Goal را overestimate می‌کند.

مسیر بهینه به صورت Start → B → Goal است که هزینه‌ی 6 را دارد در حالی که مسیر Start → A → Goal هزینه‌ی 7 دارد.

الگوریتم *A اما این مسیر را پیدا نمی‌کند. این الگوریتم ابتدا راس‌های A و B را به داخل *fringe* می‌برد. سپس راس *A* را باز می‌کند زیرا *f(A)=5* در حالی که *f(B) = 8*، سپس راس *Goal* را به *fringe* برده و آن را باز می‌کند زیرا از این مسیر دارای مقدار *f(Goal) = 7* است. نتیجتا الگوریتم *A** مسیر بهینه را پیدا نکرد.

(۲)گراف زیر را در نظر بگیرید:



مقادیر زیر را برای هزینه‌ی مسیرها با تابع heuristic در نظر بگیرید:

Cost(Start → A) = 5

Cost(Start → B) = 4

Cost(B → A) = 0.5

Cost(A → Goal) = 3.5

h(Start) = 7

h(A) = 2

h(B) = 3.5

h(Goal) = 0

این تابع *heuristic*، *admissible* است زیرا به ازای راس‌های مختلف هیچ‌کدام از مقادیر تابع *heuristic*، اندازه مسیر را *overestimate* نمی‌کند. اما این تابع *monotonic* نیست زیرا اگر از راس *B* به *A* برویم داریم *h(A) + cost(B,A) = 2.5*، اما مقدار *h(B) = 3.5* که باعث می‌شود نامساوی مثلث رعایت نشود.

مسیر بهینه، مسیر *Start → B → A → Goal* است که هزینه‌ی *8* را دارد. مسیر دیگر به هدف، هزینه‌ی *8.5* دارد.

الگوریتم *A** اما این مسیر را پیدا نمی کند. ابتدا راس‌های *A* و *B* را به *fringe* می‌فرستد. سپس راس *A* که *f(A)=7* را باز می‌کند. راس *B* از این مسیر دارای *f(B)=7.5* است. سپس راس *Goal* وارد *fringe* می‌شود که از این مسیر دارای *f(Goal) = 8.5* است. در *fringe* دو راس *B* و *Goal* باقی مانده است که *f* راس *B* کمتر است. این راس باز می‌شود اما همسایه‌اش که راس *A* باشد *visit* شده‌است، در نتیجه دوباره راس *A* وارد *fringe* نمی‌شود. در نهایت راس *Goal* باز می‌شود و خروجی مسیر بهینه = *8.5* داده می‌شود که نادرست است. در نتیجه در *Graph Search* به *monotonic* بودن برای بهینگی الگوریتم *A** نیاز داریم.