دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

هوش مصنوعی

بهار۱۴۰۴

استاد: احسان تن قطاری

امیرعلی شیخی، علی بختیاری، آرین نوری، بردیا ماندگار، علیرضا ملک حسینی، رادین شاه دایی

تمرین اول | **جست و جو و بهینه سازی** | مهلت ارسال: ۱۷اسفند

- مهلت ارسال پاسخ تا ساعت ۲۳:۵۹ روز مشخص‌شده است.
- در طول ترم امکان ارسال با تاخیر پاسخ همه‌ی تمارین سقف ۴ روز و در مجموع ۱۰ روز، وجود دارد. پس از گذشت این مدت، پاسخ‌های ارسال‌شده پذیرفته نخواهند بود. همچنین، به ازای هر ساعت تأخیر غیر مجاز نیم درصد از نمره ی تمرین کم خواهد شد.
- هم‌کاری و هم‌فکری شما در انجام تمرین مانعی ندارد اما پاسخ ارسالی هر کس حتما باید توسط خود او نوشته شده باشد.
- در صورت هم‌فکری و یا استفاده از هر منابع خارج درسی، نام هم‌فکران و آدرس منابع مورد استفاده برای حل سوال مورد نظر را ذکر کنید.
- لطفا تصویری واضح از پاسخ سوالات نظری بارگذاری کنید. در غیر این صورت پاسخ شما تصحیح نخواهد شد.

سوالات (۱۰۰ نمره)

۱. (۹ نمره) درستی یا نادرستی عبارت‌های زیر را با ذکر دلیل مشخص کنید.

(آ) یکی از عوامل لازم برای شروع کار یک agent در یک محیط داشتن آگاهی کامل به محیط است.

(ب) goal-based agentها احتیاجی به sensor ندارند.

(ج) reflex agentها وقایعی که در اثر کنش‌های آن‌ها در آینده به وجود خواهد آمد را در نظر نمی‌گیرند.

(د) در الگوریتم simulated annealing برای انتخاب حرکت بعدی، بهترین حرکت ممکن را انتخاب می‌کنیم و سپس درباره انجام دادن یا ندادن آن به صورت تصادفی تصمیم‌گیری می‌کنیم.

حل.

(آ) نادرست، عامل‌ها لزوماً به داشتن آگاهی برای محیط نیازی ندارند.

(ب) نادرست، هرعاملی برای دریافت اطلاعات از محیط به حسگر نیاز دارد.

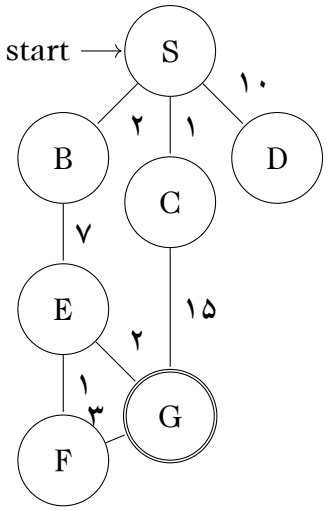
(ج) درست.

(د) نادرست، بهترین حرکت انتخاب نمی‌شود، بلکه حرکت بعدی خود به صورت تصادفی انتخاب می‌شود.

۲. (۱۶ نمره)

بخش اول

گراف جستجوی نشان داده شده در زیر را در نظر بگیرید. S حالت شروع و G حالت هدف است. همه‌ی یال‌ها دوطرفه هستند.



G	F	E	D	C	B	S	node
۰	۱	۱	۷	۱۰	۷	۹	h

برای هر یک از استراتژی‌های جستجوی زیر، مسیری را که بازگردانده می‌شود بنویسید، یا اگر مسیری وجود ندارد، **هیچ** بنویسید. در صورت وجود تساوی، از ترتیب حروف الفبا برای شکستن تساوی استفاده کنید (یعنی، گره‌هایی که نامشان زودتر در الفبا می‌آید ابتدا گسترش داده می‌شوند).

جست و جوی گراف DFS

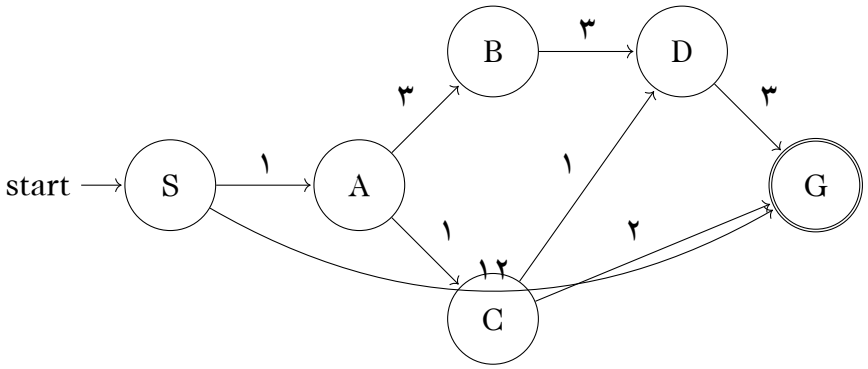
جستجوی گراف BFS

جستجوی گراف UCS

جستجوی A^*

بخش دوم

مسئله جست و جوی زیر را در نظر بگیرید:



به سوالات زیر درباره این مسئله‌ی جستجو پاسخ دهید. در صورت تساوی، ترتیب حروف الفبا رعایت شود.

جست و جوی گراف DFS

جستجوی گراف BFS

جستجوی گراف UCS

جستجوی A^* با تابع هیوریستیک consistent

حل.

بخش اول

جست و جوی گراف DFS

G - F - E - B - S

جستجوی گراف BFS

G - C - S

جستجوی هزینه یکنواخت

G - E - B - S

جستجوی A^*

G - E - B - S

بخش دوم

جستجوی گراف BFS

G - S

جستجوی گراف UCS

G - C - A - S

جست و جوی گراف DFS

G - D - B - A - S

جستجوی A^*

G - C - A - S

۳. (۱۸ نمره) یک ربات در خانه‌ی S در صفحه شطرنجی زیر قرار دارد. این ربات در هر حرکت می‌تواند به یکی از چهار جهت اصلی حرکت کند. شما باید به او کمک کنید تا در سریع‌ترین زمان به خانه‌ی E برسد. در این صفحه تعدادی خانه غیرقابل عبور وجود دارد که با X مشخص شده‌اند. سایر خانه‌ها از جنس جاده (R)، سبزه (G)، یا آب (W) هستند که زمان عبور از آنها به ترتیب ۱، ۲، و ۳ واحد است. همچنین فرض کنید زمان عبور از خانه‌ی شروع صفر است.

S	R	G	R	X
R	W	R	X	R
R	R	W	G	R
X	R	X	X	G
R	R	R	R	**E**

(آ) مسئله را به صورت یک گراف مدل کنید.

(ب) یک جستجوی گرافی با الگوریتم جستجوی هزینه یکنواخت روی این گراف انجام دهید و مسیر خروجی و تعداد گره‌های باز شده را مشخص کنید. همچنین مشخص کنید در هر مرحله کدام گره باز می‌شود و مقدار هر گره چگونه بروزرسانی می‌شود. در طی جستجو فرض کنید در صورت برابر بودن مقدار چند راس، اولویت حرکت به صورت «راست، پایین، چپ، بالا» است.

(ج) یک تابع اکتشافی به نام h_1 برای این مسئله ارائه دهید که قابل قبول و یکنوا باشد. همچنین این ویژگی‌ها را برای h_1 بررسی و اثبات کنید.

(د) یک جستجوی گرافی با الگوریتم A^* و تابع اکتشافی h_1 انجام دهید و مسیر خروجی و تعداد گره‌های باز شده را مشخص کنید. همچنین مشخص کنید در هر مرحله کدام گره باز می‌شود و مقدار هر گره چگونه بروزرسانی می‌شود. در طی جستجو فرض کنید در صورت برابر بودن مقدار چند راس، اولویت حرکت به صورت «راست، پایین، چپ، بالا» است.

(ه) تعداد گره‌های باز شده توسط این دو الگوریتم را مقایسه کنید. کدام الگوریتم عملکرد بهتری دارد؟ درباره‌ی علت آن توضیح دهید.

(و) اکنون فرض کنید ربات می‌تواند حرکت‌های قطری نیز انجام دهد (یعنی علاوه بر چهار جهت اصلی، به صورت مورب نیز حرکت کند). آیا همچنان تابع اکتشافی h_1 قابل قبول و یکنوا باقی می‌ماند؟ اگر بله، آن را نشان دهید و در غیر این صورت، یک تابع اکتشافی جدید به نام h_2 ارائه دهید که قابل قبول و یکنوا باشد.

حل.

(آ) گراف این مسئله به این صورت است که به‌ازای هر خانه‌ی صفحه‌ی شطرنجی یک رأس در نظر می‌گیریم و بین رأس‌های متناظر با خانه‌های همسایه، دو یال جهت‌دار قرار می‌دهیم. با فرض اینکه عبور از هر خانه معادل ورود به آن است، وزن هر یال (u, v) برابر هزینه‌ی عبور از خانه‌ی متناظر با رأس v است (همچنین می‌توان فرض کرد عبور از هر خانه معادل خروج از آن است و در آن صورت، وزن یال (u, v) برابر هزینه‌ی عبور از خانه‌ی متناظر با رأس u خواهد بود، به استثنای یال‌های مجاور S که وزن صفر دارند). برای سادگی می‌توان خانه‌های غیرقابل عبور را در گراف در نظر نگرفت.

(ب) الگوریتم UCS را در حالت جستجوی گرافی اجرا می‌کنیم. در این پیاده‌سازی، در صورت برابر بودن مقدار g برای چند گره، به صورت FIFO عمل می‌کنیم. همچنین اگر یک گره در صف وجود داشت و مسیر دیگری به آن یافتیم، آن را مجدداً در صف درج نمی‌کنیم و g آن را برابر مینیمم g فعلی و g جدید قرار می‌دهیم. مراحل این الگوریتم به صورت زیر است:

Step ۱: Queue:
Node: (0, 0), $g = 0$
Expanding Node: (0, 0) with $g = 0$

Step ۲: Queue:
Node: (0, 1), $g = 1$
Node: (1, 0), $g = 1$
Expanding Node: (0, 1) with $g = 1$

Step ۳: Queue:
Node: (1, 0), $g = 1$
Node: (0, 2), $g = 3$
Node: (1, 1), $g = 4$
Expanding Node: (1, 0) with $g = 1$

Step ۴: Queue:
Node: (2, 0), $g = 2$
Node: (0, 2), $g = 3$
Node: (1, 1), $g = 4$
Expanding Node: (2, 0) with $g = 2$

Step ۵: Queue:
Node: (0, 2), $g = 3$
Node: (2, 1), $g = 3$
Node: (1, 1), $g = 4$
Expanding Node: (0, 2) with $g = 3$

Step ۶: Queue:
Node: (2, 1), $g = 3$
Node: (1, 1), $g = 4$
Node: (0, 3), $g = 4$
Node: (1, 2), $g = 4$
Expanding Node: (2, 1) with $g = 3$

Step ۷: Queue:
Node: (1, 1), $g = 4$
Node: (0, 3), $g = 4$
Node: (1, 2), $g = 4$
Node: (3, 1), $g = 4$
Node: (2, 2), $g = 6$
Expanding Node: (1, 1) with $g = 4$

Step ۸: Queue:
Node: (0, 3), $g = 4$
Node: (1, 2), $g = 4$
Node: (3, 1), $g = 4$
Node: (2, 2), $g = 6$
Expanding Node: (0, 3) with $g = 4$

Step ۹: Queue:
Node: (1, 2), $g = 4$
Node: (3, 1), $g = 4$
Node: (2, 2), $g = 6$
Expanding Node: (1, 2) with $g = 4$

Step ۱۰: Queue:

Node: (3, 1), $g = 4$
Node: (2, 2), $g = 6$
Expanding Node: (3, 1) with $g = 4$

Step ۱۱: Queue:
Node: (4, 1), $g = 5$
Node: (2, 2), $g = 6$
Expanding Node: (4, 1) with $g = 5$

Step ۱۲: Queue:
Node: (2, 2), $g = 6$
Node: (4, 2), $g = 6$
Node: (4, 0), $g = 6$
Expanding Node: (2, 2) with $g = 6$

Step ۱۳: Queue:
Node: (4, 2), $g = 6$
Node: (4, 0), $g = 6$
Node: (2, 3), $g = 8$
Expanding Node: (4, 2) with $g = 6$

Step ۱۴: Queue:
Node: (4, 0), $g = 6$
Node: (4, 3), $g = 7$
Node: (2, 3), $g = 8$
Expanding Node: (4, 0) with $g = 6$

Step ۱۵: Queue:
Node: (4, 3), $g = 7$
Node: (2, 3), $g = 8$
Expanding Node: (4, 3) with $g = 7$

Step ۱۶: Queue:
Node: (4, 4), $g = 7$
Node: (2, 3), $g = 8$
Expanding Node: (4, 4) with $g = 7$

Path: (0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (4, 1), (4, 2), (4, 3), (4, 4)
Expanded Nodes: 16

(ج) تابع $h_۱$ را به صورت زیر تعریف می‌کنیم. این تابع همان فاصله‌ی منهتنی هر خانه با خانه‌ی E است.

$$h_۱((x, y)) = |x_E - x| + |y_E - y|$$

برای اثبات قابل قبول بودن $h_۱$ باید نشان دهیم $\forall s \in S : h_۱(s) \leq h^*(s)$. فرض کنید مسیر بهینه از s به E به صورت $t_۱, \ldots, t_k$ است. داریم:

$$h^*(s) = \sum_{i=۱}^{k} c((t_i, t_{i+۱})) \geq k$$

همچنین از آنجایی که با هر حرکت فاصله منهتنی با E حداکثر یک واحد کاهش می‌یابد، k حداقل برابر با $h_۱(s)$ است. با ترکیب این دو نامساوی حکم اثبات می‌شود:

$$h^*(s) = \sum_{i=۱}^{k} c((t_i, t_{i+۱})) \geq k \geq h_۱(s)$$

برای اثبات سازگار بودن h_1 باید نشان دهیم به ازای هر دو خانه‌ی همسایه‌ی s، s' نامساوی زیر برقرار است:

$$h_1(s) - h_1(s') \leq c((s, s'))$$

می‌دانیم به ازای هر s، s' همسایه، با حرکت از s به s' فاصله‌ی منهتنی با E - یعنی مقدار h_1 - حداکثر یک واحد کاهش می‌یابد و همچنین $c((s, s'))$ حداقل برابر یک است. با ترکیب این دو نامساوی حکم اثبات می‌شود:

$$h_1(s) - h_1(s') \leq 1 \leq c((s, s'))$$

پس تابع اکتشافی h_1 هم قابل قبول و هم سازگار است.

(د) الگوریتم A^* را در حالت جستجوی گرافی اجرا می‌کنیم. در این پیاده‌سازی، در صورت برابر بودن مقدار g برای چند گره، به صورت FIFO عمل می‌کنیم. همچنین اگر یک گره در صف وجود داشت و مسیر دیگری به آن یافتیم، آن را مجدداً در صف درج نمی‌کنیم و g آن را برابر مینیمم g فعلی و g جدید قرار می‌دهیم. مراحل این الگوریتم به صورت زیر است:

Step ۱: Queue:
Node: (0, 0), $f = 8$
Expanding Node: (0, 0), $f = 8$

Step ۲: Queue:
Node: (0, 1), $f = 8$
Node: (1, 0), $f = 8$
Expanding Node: (0, 1), $f = 8$

Step ۳: Queue:
Node: (1, 0), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Expanding Node: (1, 0), $f = 8$

Step ۴: Queue:
Node: (2, 0), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Expanding Node: (2, 0), $f = 8$

Step ۵: Queue:
Node: (2, 1), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Expanding Node: (2, 1), $f = 8$

Step ۶: Queue:
Node: (3, 1), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Node: (2, 2), $f = 10$
Expanding Node: (3, 1), $f = 8$

Step ۷: Queue:
Node: (4, 1), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Node: (2, 2), $f = 10$
Expanding Node: (4, 1), $f = 8$

Step ۸: Queue:
Node: (4, 2), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Node: (2, 2), $f = 10$
Node: (4, 0), $f = 10$
Expanding Node: (4, 2), $f = 8$

Step ۹: Queue:
Node: (4, 3), $f = 8$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Node: (2, 2), $f = 10$
Node: (4, 0), $f = 10$
Expanding Node: (4, 3), $f = 8$

Step ۱۰: Queue:
Node: (4, 4), $f = 7$
Node: (0, 2), $f = 9$
Node: (1, 1), $f = 10$
Node: (2, 2), $f = 10$
Node: (4, 0), $f = 10$
Expanding Node: (4, 4), $f = 7$

Path: (0, 0), (1, 0), (2, 0), (2, 1), (3, 1), (4, 1), (4, 2), (4, 3), (4, 4)
Expanded Nodes: 10

(ه) با استفاده از تابع اکتشافی h_1 که ویژگی‌های قابل قبول بودن و سازگاری را دارد، هر دو الگوریتم optimal خواهند بود. مسیر خروجی این دو الگوریتم در این مثال یکسان است ولی الگوریتم UCS با باز کردن ۱۶ گره به این مسیر رسیده، در حالی که الگوریتم A^* فقط ۱۰ گره را باز کرده است. بنابراین الگوریتم A^* کارایی بیشتری دارد و بهینه‌تر است.
دلیل این موضوع به informed بودن این الگوریتم برمی‌گردد؛ یعنی علاوه بر استفاده از تابع g که هزینهٔ رسیدن به هر گره را نشان می‌دهد، با داشتن تابع اکتشافی که تخمینی از هزینه‌ی ادامه‌ی مسیر را فراهم می‌کند، جستجو به شکل موثرتری انجام می‌شود و گره‌هایی که به هدف نزدیک‌تر هستند، زودتر مورد توجه قرار می‌گیرند. اما الگوریتم UCS که به چنین تخمینی دسترسی ندارد، باید مسیرهای بیشتری را بررسی کند و عملاً مجبور است تمام مسیرهای ممکن با g یکسان را گسترش دهد!

(و) خیر، هیچ یک از ویژگی‌های قابل قبول بودن و سازگاری برای تابع h_1 در این حالت برقرار نیست. برای نشان دادن این موضوع کافی است مثال نقض ارائه دهیم.

رد کردن قابل قبول بودن h_1:

$$h((۳, ۱)) = ۴ > h^*((۳, ۱)) = ۳$$

رد کردن سازگاری h_1:

$$h((۳, ۱)) - h((۴, ۲)) = ۲ > c(((۳, ۱), (۴, ۲))) = ۱$$

تابع h_2 را به صورت زیر تعریف می‌کنیم:

$$h_2((x, y)) = \max\{|x_E - x|, |y_E - y|\}$$

برای اثبات قابل قبول بودن $h_۲$ باید نشان دهیم $\forall s \in S : h_۲(s) \leq h^*(s)$.
فرض کنید مسیر بهینه از s به E به صورت $t_۱, \ldots, t_k$ است. داریم:

$$h^*(s) = \sum_{i=۱}^{k} c((t_i, t_{i+۱})) \geq k$$

همچنین از آنجایی که با هر حرکت مقدار تابع $h_۲$ حداکثر یک واحد کاهش می‌یابد، k حداقل برابر با $h_۲(s)$ است. با ترکیب این دو نامساوی حکم اثبات می‌شود:

$$h^*(s) = \sum_{i=۱}^{k} c((t_i, t_{i+۱})) \geq k \geq h_۲(s)$$

برای اثبات سازگار بودن $h_۲$ باید نشان دهیم به ازای هر دو خانه‌ی همسایه‌ی s, s' نامساوی زیر برقرار است:

$$h_۲(s) - h_۲(s') \leq c((s, s'))$$

می‌دانیم به ازای هر s, s' همسایه، با حرکت از s به s' مقدار تابع $h_۲$ حداکثر یک واحد کاهش می‌یابد و همچنین $c((s, s'))$ حداقل برابر یک است. با ترکیب این دو نامساوی حکم اثبات می‌شود:

$$h_۲(s) - h_۲(s') \leq ۱ \leq c((s, s'))$$

پس تابع اکتشافی $h_۲$ هم قابل قبول و هم سازگار است.

۴. (۱۸ نمره)

فرض کنید یک عامل شبیه خودرو قصد دارد از یک هزارتو مانند تصویر پایین خارج شود. این عامل دارای جهت‌گیری است و در هر لحظه به یکی از چهار جهت شمال (N)، جنوب (S)، شرق (E) یا غرب (W) نگاه می‌کند. عامل می‌تواند با یک عمل، یا به جلو حرکت کند (با سرعت قابل تنظیم) یا دور بزند. عمل‌های چرخش شامل چرخش به چپ و چرخش به راست هستند که جهت عامل را ۹۰ درجه تغییر می‌دهند. چرخش فقط زمانی مجاز است که سرعت عامل صفر باشد (و بعد از چرخش همچنان صفر باقی می‌ماند).

عمل‌های حرکتی شامل افزایش سرعت (fast)، حفظ سرعت (maintain) و کاهش سرعت (slow) هستند:

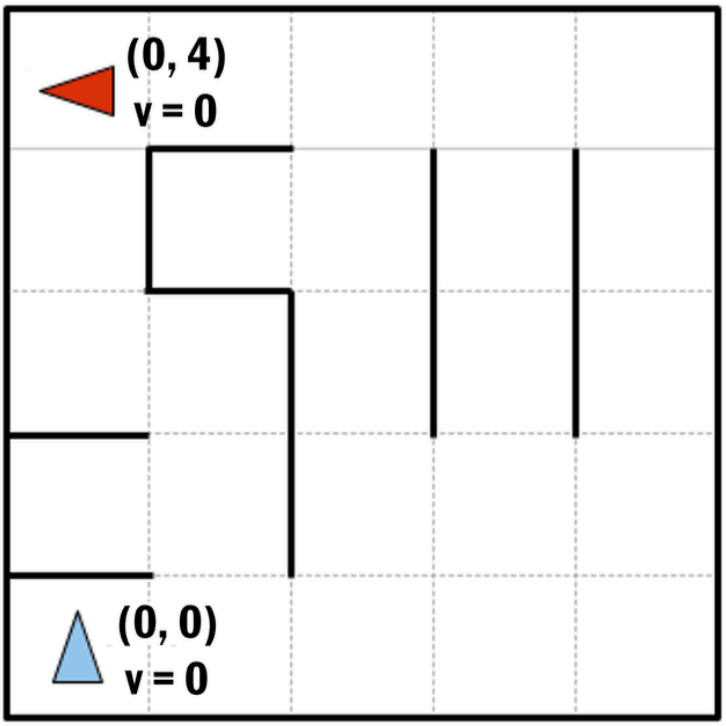
- **Fast** سرعت را ۱ واحد افزایش می‌دهد و سپس عامل به تعداد خانه‌های برابر با سرعت جدید خود حرکت می‌کند.
- **Slow** سرعت را ۱ واحد کاهش می‌دهد و سپس عامل به همان تعداد خانه حرکت می‌کند.
- **Maintain** سرعت فعلی را حفظ کرده و عامل را به همان تعداد خانه‌های حرکت قبلی جابجا می‌کند.

هر حرکتی که منجر به برخورد با دیوار شود غیرمجاز است. هیچ حرکتی نباید سرعت را کمتر از صفر یا بیشتر از یک مقدار حداکثر ($V_{\max}$) کند.

هدف: عامل باید برنامه‌ای پیدا کند که آن را در کمترین تعداد گام زمانی ممکن، در حالت سکون روی خانه‌ی خروج قرار دهد.

(الف) فرض کنید عامل می‌خواهد مسیر سمت چپ (leftmost path) را از نقطه‌ی شروع $(۰, ۰)$ رو به شمال تا خانه‌ی هدف $(۰, ۴)$ رو به غرب طی کند. دنباله‌ی اعمال لازم برای رسیدن به این هدف را بنویسید.

(ب) اگر شبکه‌ی $M \times N$ باشد، اندازه‌ی فضای حالات چقدر است؟ فرض کنید که همه‌ی حالات ممکن از حالت شروع قابل دسترسی هستند.



شکل ۱ : نمایی از هزارتو برای سوال چهارم

(پ) بیشترین مقدار عامل انشعاب (branching factor) در این مسئله چقدر است؟ یک حالت مثال بزنید که در آن عامل انشعاب بیشینه رخ دهد و لیست اعمال ممکن را مشخص کنید.

(ت) آیا فاصله منهتن بین مکان عامل و مکان خروج، یک هیوریستیک مجاز (admissible) است؟

اگر نیست، یک حالت مثال بزنید که در آن این هیوریستیک مقدار بیشتری نسبت به هزینه واقعی بدهد و موارد زیر را مشخص کنید:

- مقدار هیوریستیک در آن حالت
- هزینه واقعی رسیدن از آن حالت به هدف

(ث) آیا این هیوریستیک مجاز است؟ (تعداد چرخش‌های مورد نیاز برای قرارگیری رو به هدف)

اگر نیست، یک حالت مثال بزنید که در آن این هیوریستیک مقدار بیشتری نسبت به هزینه واقعی بدهد و موارد زیر را مشخص کنید:

- مقدار هیوریستیک در آن حالت
- هزینه واقعی رسیدن از آن حالت به هدف

(ج) آیا این هیوریستیک مجاز است؟ (فاصله منهتن تقسیم بر $V_{\max}$)

اگر نیست، یک حالت مثال بزنید که در آن این هیوریستیک مقدار بیشتری نسبت به هزینه واقعی بدهد و موارد زیر را مشخص کنید:

- مقدار هیوریستیک در آن حالت
- هزینه واقعی رسیدن از آن حالت به هدف

حل. **الف)** ترتیب عملیات‌های مورد نیاز برای پیمودن leftmost path به این صورت است: Right, fast, slow, left, fast, maintain, slow, left, fast, slow, right, fast, maintain, slow, left

ب) در هنگام شروع حرکت متحرک می‌تواند در هر یک از $M \times N$ خانهٔ موجود باشد و در هر کدام از این خانه‌ها جهت متحرک می‌تواند به سمت شمال، جنوب، شرق یا غرب باشد، پس ۴ مقدار ممکن دارد و همچنین سرعت آن می‌تواند $V_{\max}$ مقدار مختلف داشته باشد، پس اندازهٔ فضای حالت مسئله برابر است با:

$$M \times N \times V_{\max} \times ۴$$

پ) بیش‌ترین عامل انشعاب زمانی رخ می‌دهد که متحرک با سرعت صفر در یک خانه در جهتی که روبه‌رویش دیوار نیست قرار دارد و حداکثر سرعت مجاز یا همان $V_{\max}$ بیش‌تر از یک است. در این حالت، اعمال ممکن عبارت‌اند از: گردش به راست، گردش به چپ، افزایش سرعت، و حفظ سرعت. پس حداکثر مقدار عامل انشعاب مسئله برابر ۴ است.

ت) فاصلهٔ منهتن یک هیوریستیک مجاز نیست. حالتی را در نظر بگیرید که متحرک در خانهٔ $(۴, ۴)$ باشد و جهت آن به سمت چپ باشد و سرعت آن برابر ۴ باشد. در این حالت، متحرک با یک عملیات `maintain` به مقصد می‌رسد، ولی فاصلهٔ منهتن تا مقصد برابر ۴ است. پس مقدار هیوریستیک برابر ۴ و هزینهٔ واقعی برابر ۱ است.

ث) این هیوریستیک مجاز است، زیرا متحرک برای رسیدن به مقصد حتماً باید جهت خود را رو به مقصد تغییر دهد، پس با صرف‌نظر از سرعت متحرک، حداقل به تعداد تغییر جهت‌های موردنیاز برای قرار گرفتن رو به مقصد عملیات نیاز داریم.

ج) این هیوریستیک نیز مجاز است، زیرا متحرک نهایتاً با سرعت $V_{\max}$ می‌تواند حرکت کند و حتی در آن شرایط نیز حداقل به تعداد $\frac{\text{فاصلهٔ منهتن}}{V_{\max}}$ عملیات `maintain` نیاز دارد تا به مقصد برسد، حتی با صرف‌نظر از عملیات‌های موردنیاز برای چرخش. پس همواره مقدار این هیوریستیک کم‌تر از هزینهٔ واقعی تا مقصد است و این هیوریستیک مجاز است.

۵. (۲۱ نمره) درستی یا نادرستی عبارت‌های زیر را با ذکر دلیل مشخص کنید.

(آ) الگوریتم‌های جستجوی محلی به دلیل نگهداری پیکربندی‌های کامل (complete configuration) حافظه بیشتری مصرف می‌کنند.

(ب) در الگوریتم‌های جستجوی محلی، برخلاف الگوریتم‌های جستجوی سیستماتیک، هر وضعیت یک جواب احتمالی برای مسئله را نشان می‌دهد.

(ج) در یک فضای جستجوی متناهی، الگوریتم Random Walk در صورت دادن زمان کافی، قطعاً به جواب مسئله (در صورت وجود) می‌رسد.

(د) نمی‌توان تحت هیچ شرایطی تضمین کرد که الگوریتم Simulated Annealing از بهینه محلی فرار می‌کند.

(ه) الگوریتم Simulated Annealing با $T = \infty$ معادل Random Walk است.

(و) الگوریتم Local Beam Search معادل k بار اجرای همزمان الگوریتم Hill-Climbing Search با نقاط شروع متفاوت است.

(ز) پیچیدگی زمانی هر مرحله از الگوریتم Local Beam Search با ضریب انشعاب b از $O(bk)$ است.

حل.

(آ) نادرست؛ در الگوریتم‌های جستجوی محلی برخلاف الگوریتم‌های جستجوی سیستماتیک، در هر لحظه تنها وضعیت جاری نگهداری می‌شود و نیازی به ذخیره‌ی تاریخچه‌ی وضعیت‌ها نیست. بنابراین این الگوریتم‌ها مصرف حافظه‌ی بهینه‌تری دارند.

(ب) درست؛ برخلاف الگوریتم‌های جستجوی سیستماتیک که ممکن است جواب مسئله را به تدریج بسازند، در الگوریتم‌های جستجوی محلی از یک جواب اولیه شروع می‌کنیم و سعی می‌کنیم آن را بهبود ببخشیم. بنابراین هر وضعیت، یک جواب احتمالی برای مسئله را نشان می‌دهد.

(ج) درست؛ در یک فضای جستجوی متناهی، الگوریتم Random Walk در صورت دادن زمان کافی تمام فضا را جستجو می‌کند. به عبارتی اگر $T \to \infty$ آنگاه احتمال بررسی هر وضعیت نیز به یک میل می‌کند. بنابراین این الگوریتم به جواب (در صورت وجود) خواهد رسید.

(د) درست؛ الگوریتم Simulated Annealing به گونه‌ای طراحی شده تا از بهینه‌های محلی فرار کند. اگرچه زمان‌بندی‌ای وجود دارد که همگرایی این الگوریتم به جواب بهینه کلی را در بی‌نهایت تضمین کند (Logarithmic Cooling)، ولی به دلیل ماهیت تصادفی الگوریتم، هیچ تضمینی وجود ندارد که در همه‌ی اجراها از همه‌ی بهینه‌های محلی فرار کند.

(ه) درست؛ در الگوریتم Simulated Annealing یک همسایه از وضعیت جاری به طور تصادفی انتخاب و ΔE این حرکت محاسبه می‌شود. اگر $\Delta E > ۰$ بود این حرکت همواره پذیرفته می‌شود. حال اگر $T = \infty$ باشد، هر حرکت با $\Delta E \leq ۰$ نیز به احتمال یک پذیرفته می‌شود. پس می‌توان گفت که در هر مرحله یک حرکت کاملاً تصادفی انجام می‌شود که این معادل الگوریتم Random Walk است.

(و) نادرست؛ در الگوریتم Local Beam Search لزوماً در هر مرحله رابطه یک به یکی بین k وضعیت جدید و k وضعیت قبلی وجود ندارد و ممکن است برخی از وضعیت‌ها هیچ فرزند مناسبی تولید نکنند و برخی دیگر فرزندان بیشتری ایجاد کنند. در نتیجه، به تدریج نقاط شروع بهتر می‌توانند غالب شوند. بنابراین نمی‌توان این الگوریتم را با k بار اجرای الگوریتم Hill-Climbing Search شبیه‌سازی کرد.

(ز) درست؛ در هر مرحله از الگوریتم Local Beam Search هر یک از k وضعیت موجود b فرزند ایجاد می‌کند و سپس باید k بهترین فرزند از میان آنها انتخاب شوند که با پیاده‌سازی بهینه، در حالت متوسط این عملیات در $O(bk)$ انجام می‌شود.

۶. (۱۸ نمره) الگوریتم ژنتیک را در نظر بگیرید که در آن از کروموزوم‌هایی با طول ۸ استفاده می‌کنیم. هر ژن می‌تواند مقدار ۰ یا ۱ را بگیرد. تابع fitness را برای کروموزوم‌ها به این صورت تعریف می‌کنیم.

$$X = \overline{x_۷x_۶x_۵x_۴x_۳x_۲x_۱x_۰}$$

$$f(X) = \sum_{i=۰}^{۷} (x_i \times (X \ll i))_۲$$

معنی تابع بالا این است که هربار رشته کروموزوم اولیه را i بار به سمت چپ شیفت می‌دهد و در بیت i ام ضرب می‌کند، سپس معادل دهدهی هر حالت را محاسبه می‌کند و با یکدیگر جمع می‌کند. در نظر داشته باشید که هنگام شیفت دادن کروموزوم، به اندازه کافی حافظه برای ذخیره اعداد جدید داریم و بیت‌ها بیرون ریخته نمی‌شوند.

جمعیت اولیه‌ی کروموزوم‌های ما به صورت زیر هستند:

$$X_۰ = ۱۱۰۰۱۱۰۱$$
$$X_۱ = ۰۱۱۰۱۰۰۱$$
$$X_۲ = ۱۰۱۱۰۱۱۰$$
$$X_۳ = ۰۱۰۱۰۱۱۱$$

(آ) ابتدا تابع fitness را برای جمعیت موجود حساب کنید.

(ب) حالا طبق مشاهداتی که از خروجی تابع و معادله آن داشتید بگویید در واقع این تابع چه کاری را انجام می‌دهد؟ به ازای چه ورودی‌ای این تابع بیشینه می‌شود؟

(ج) بعد از حساب کردن تابع fitness برای هر کروموزوم، می‌خواهیم selection را انجام دهیم. ابتدا توضیح دهید این کار چه کمک به ما در رسیده به مقدار بهینه می‌کند و سپس احتمال انتخاب هر کروموزوم را حساب کنید و به کمک اعداد زیر از چپ به راست به ترتیب selection را انجام دهید.

$$۰/۷۴۸, ۰/۳۶۲, ۰/۹۱۵, ۰/۵۳۱$$

(د) عمل بعدی در الگوریتم ژنتیک crossover است. به طور خلاصه این الگوریتم چگونه به ما در رسیدن به جواب مطلوب کمک می‌کند؟ حالا کروموزوم‌های حاصل از مرحله قبل را در نظر بگیرید. می‌خواهیم به کمک تنها یک برش این کار را انجام دهیم. فرض کنید شما محل برش را تعیین می‌کنید تا دو به دو کروموزوم‌ها با هم ترکیب شوند. در صورتی که بخواهیم حریصانه عمل کنیم، برش را چگونه می‌زنیم؟ حال فرض کنید برش را طوری بزنیم که کروموزوم‌ها را نصف کند، با ترکیب کروموزوم‌های اولی با دومی و سومی با چهارمی کروموزوم‌های جدید را بسازید.

(ه) در گام پایانی برای رسیدن به جمعیت جدید می‌خواهیم بر روی چهار کروموزوم حاصل از بخش قبلی عمل mutation را انجام دهیم. به طور خلاصه این عمل چه کمکی به ما می‌کند؟ حالا برای انجام این کار در رشته i ام، بیت i ام از سمت راست را تغییر دهید. در نهایت کروموزوم‌های جدید را بنویسید. آیا می‌توان گفت به مقدار بهینه نزدیک‌تر شدیم؟

(و) با جمعیت اولیه موجود و بدون در نظر گرفتن عمل mutation آیا می‌توان به مقدار بهینه رسید؟ علت خود را توضیح دهید.

حل.

(آ) به ترتیب به خروجی‌های ۴۲۰۲۵، ۱۱۰۲۵، ۳۳۱۲۴ و ۷۵۶۹ می‌رسیم.

(ب) هدف این تابع حساب کردن مجذور رشته‌های داده شده در مبنای دهدهی است و به ازای رشته‌ی تمام یک بیشینه می‌شود.

(ج) به ازای رشته iام، مجموع تابع از رشته‌ی اولیه تا رشته‌ی i را حساب می‌کنیم و بر مجموع خروجی‌ها تقسیم می‌کنیم. بازه‌های احتمال به این صورت خواهند بود:

$$(۰, ۰/۴۴۸۳), (۰/۴۴۸۳, ۰/۵۶۵۹), (۰/۵۶۵۹, ۰/۹۱۹۳), (۰/۹۱۹۳, ۱/۰)$$

در نتیجه به ترتیب رشته‌های دوم، صفرم، دوم و اول انتخاب خواهند شد.

$$A_۰ = ۱۰۱۱۰۱۱۰$$

$$A_۱ = ۱۱۰۰۱۱۰۱$$

$$A_۲ = ۱۰۱۱۰۱۱۰$$

$$A_۳ = ۰۱۱۰۱۰۰۱$$

به طور خلاصه این عمل در بالاتر بردن ارزش کلی جمعیت با نگه داشتن رشته‌های باارزش‌تر و حذف رشته‌های بی‌ارزش‌تر، احتمال رسیدن به پاسخ بهینه را بالا می‌برد.

(د) به کمک مرحله‌ی crossover می‌توانیم اطلاعات جمعیت را ترکیب کنیم و راه‌حل‌های بهینه‌تر تولید کنیم. اگر فرض greedy بودن را رسیدن به بهترین رشته بگذاریم بهتر است ترکیب را به گونه‌ای انجام دهیم که بین هر جفت رشته دو بیت سمت چپ از رشته‌ی اول انتخاب شود و باقی از رشته‌ی دوم با این کار به عدد ۱۱۱۱۱۱۰۱ در بین جمعیت داده شده می‌رسیم که بیشترین حالت ممکن است. در ادامه با ترکیب کروموزوم‌ها به کمک بریدن آن‌ها از وسط، به این جمعیت می‌رسیم:

$$B_۰ = ۱۰۱۱۱۱۰۱$$

$$B_1 = 11000110$$

$$B_2 = 10111001$$

$$B_3 = 01100110$$

(ه) این عملیات با ایجاد تغییرات تصادفی در کروموزوم‌ها از همگرایی زودرس جلوگیری کرده و تنوع جمعیت را حفظ می‌کند.

$$C_0 = 10111100$$

$$C_1 = 11000100$$

$$C_2 = 10111101$$

$$C_3 = 01101110$$

به ترتیب خروجی تابع برای اعداد بالا برابر ۳۵۳۴۴، ۳۸۴۱۶، ۳۵۷۲۱ و ۱۲۱۰۰ خواهد بود. به طور کلی خروجی تابع برای جمعیت داده شده رشد قابل توجهی داشته، با این وجود بهترین رشته در جمعیت سابق کمی نسبت به بهترین رشته در جمعیت فعلی پسرفت داشته.

(و) خیر، نیاز داریم تا رشته‌ها طوری انتخاب کنیم که به رشته‌ی ۱۱۱۱۱۱۱۱ برسیم و برای این کار باید بتوان دو رشته را طوری انتخاب کرد که crossover روی آن‌ها رشته‌ی تمام یک را نتیجه دهد که با جمعیت فعلی غیر ممکن است.