به نام خدا

دانشگاه تهران

پردیس دانشکده های فنی

دانشکده مهندسی برق و کامپیوتر

تمرین شماره‌ی 1

بخش کتبی

مدرسین:

دکتر فدائی

دکتر یعقوب‌زاده

نگارش:

فاطمه محمدی 810199489

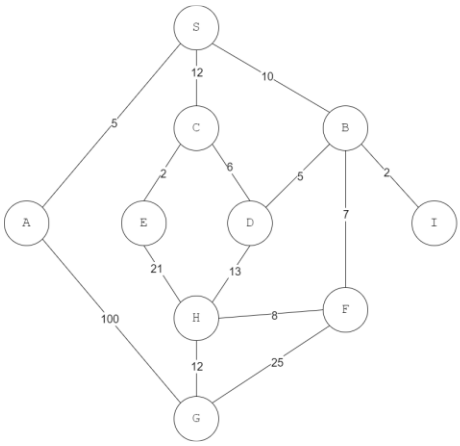
بخش کتبی

Agents

Environment	CS-GO	Sudoku
Fully/Partially Observable	Partially Observable	Fully Observable
Deterministic/Stochastic	Stochastic (شخصا اطلاع کافی از این بازی ندارم، اما اگر فرض میشد تنها مورد غیر قطعی حرکات تیم حریف میبود، Strategic میشد، اما چون احتمالا نه تنها حرکات تیم مقابل، حرکات تیم خودمون هم ممکنه غیر قطعی باشد، محیط را Stochastic در نظر گرفته ایم.)	Deterministic
Episodic/Sequential	Sequential	Sequential (با توجه به اینکه اتمام سودوکو ملاک است نه پر کردن تک خانه، بازی به صورت دنباله ای از اکشن پر کردن خانه x ام است.)
Static/Dynamic/Semi-Dynamic	Dynamic (با توجه به اینکه باقی عامل ها جز محیط محسوب میشوند و ممکن است وضعیت خود را تغیر دهند.)	Semi-Dynamic (فرض کرده ایم با گذشت زمان امتیاز کسر میشود در غیر این صورت Static است.)
Discrete/Continuous	Continuous	Discrete
Single-agent/Multi-agent	Multi-agent	Single-agent (فرض شده مسابقه رقابتی بین دو یا چند عامل نمی باشد.)

Search

سوال اول



الف)

پارت اول سوال:

:BFS

- BFS به طور کلی مناسب این گراف نمیباشد، چرا که هزینه هر یال متفاوت است، درنتیجه نمیتواند راه حل بهینه را پیدا کند. تنها complete است.

برای مثال فرض شود، Initial State راس S است و Goal State راس G میباشد؛ در این صورت یکی از مسیر هایی که ممکن است BFS پیدا کند و به جواب برسد(اگر اولویت را بین حروف رعایت کنیم) به صورت S->A->G باشد که هزینه رسیدن به G در این مسیر برابر با 105 است درصورتی که هزینه رسیدن به این راس میتوانست کمتر باشد.)

:DFS

- DFS به صورت کلی اولین راه حل را میفرستد و بهینه نیست.

در این مثال نیز با فرض، Initial State راس S است و Goal State راس G میباشد؛ مسیری که DFS پیدا میکند به صورت S->A->G میباشد که هزینه آن 105 است و بهینه نمیباشد.

(البته اگر فرض میکردیم Initial State راس C است و Goal State راس D میباشد، به صورت اتفاقی هردو میتوانستند مسیر بهینه را پیدا کنند، اما به صورت کلی نمیتوان انتظار داشت مسیر بهینه را در این گراف بین دو نود پیدا کنند.)

پارت دوم سوال:

- در صورتی جواب بهینه ملاک اصلی ما نباشد، و بدانیم که Space محدود و فاقد loop است، DFS میتواند بسیار مناسب باشد چرا که Space Complicity آن خطی ((O(bm)) است.
- همچنین در صورتی که جواب های متنوع بسیاری داشته باشیم، DFS میتواند از BFS سریع تر و مناسب تر باشد.(اما همچنان بهینه بودن خود راه حل نباید ملاک مهم و اصلی ما باشد.)
- در صورتی فضا (و زمان) مشکلات اصلی ما نباشند و جواب بهینه اهمیت داشته باشد و بدانیم که اولا branch factor محدود است و دوما هزینه یال ها برابرند(مثبت غیر صفر)، استفاده از BFS میتواند راه حل مناسبی باشد.

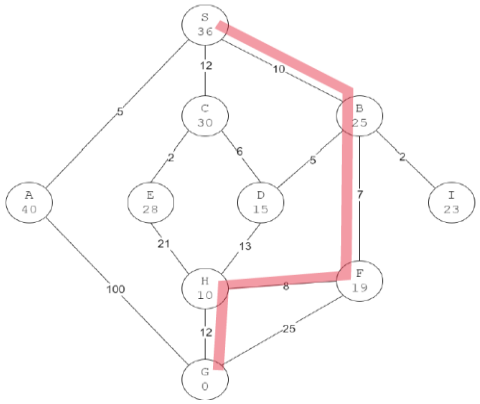
*در واقع BFS مسیر با کمترین تعداد قدم را برمیگرداند و نه ارزان ترین پس اگر یا تعداد قدم ها ملاک باشد و یا هزینه یال ها برابر باشند، این الگوریتم میتواند پاسخگو نیاز های ما باشد.

- به طور کلی اگر دنبال مسیر بهینه باشیم، DFS با توجه به اینکه اولین جوابی که به آن برسد را برمیگرداند، نمیتواند مناسب باشد. همچنین در این حالت اگر هزینه یال ها برابر نباشد، BFS نیز نمیتواند جواب بهینه را برگرداند پس پیشنهاد نمیشود.
- در صورتی که بدانیم هزینه یال ها برابر باشند و دنبال هزینه بهینه نیز باشیم، ترکیب این دو الگوریتم یعنی IDS میتواند مناسب باشد، چرا که هزینه Space خطی دارد که مشکل جدی تری از Time Complexityاست و در عین حال ما را به جواب بهینه میرساند.

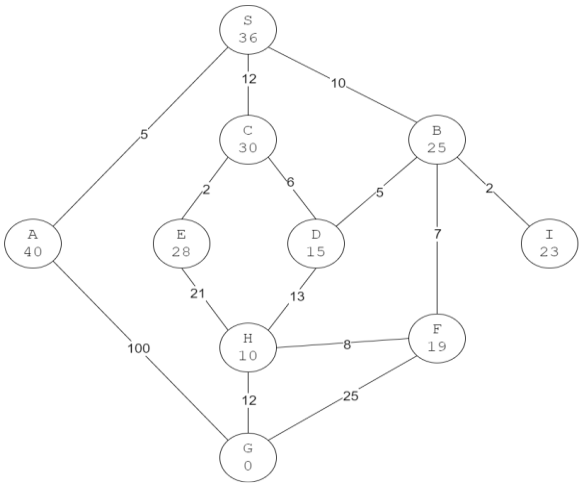
ب)

Frontier (ordered by path cost)	Explored	Current Node	Total Cost	Path
S_0				
A_5, B_{10}, C_{12}	S	S	0	**S**
B_{10}, C_{12}, G_{105}	S, A	A	5	**S->A**
$C_{12}, , I_{12}, D_{15}, F_{17}, G_{105}$	S, A, B	B	10	**S->B**
$I_{12}, E_{14}, D_{15}, F_{17}, G_{105}$	S, A, B, C	C	12	**S->C**
$E_{14}, D_{15}, F_{17}, G_{105}$	S, A, B, C, I	I	12	**S->B->I**
$D_{15}, F_{17}, H_{35}, G_{105}$	S, A, B, C, I, E	E	14	**S->C->E**
F_{17}, H_{31}, G_{105}	S, A, B, C, I, E, D	D	15	**S->B->D**
H_{25}, G_{42}	S, A, B, C, I, E, D, F	F	17	**S->B->F**
G_{37}	S, A, B, C, I, E, D, F, H	H	25	**S->B->F->H**
	S, A, B, C, I, E, D, F, H, G	G	37	**S->B->F->H->G**

با توجه به جدول بالا میتوان گفت مسیر بهینه به صورت S->B->F->H->G با هزینه 37 میباشد.



سوال دوم



الف)

Frontier	Explored	Current Node	Total Cost g(n)	g(n) + h(n)	Path
S_{36}					
B_{35}, C_{42}, A_{45}	S	S	0	36	**S**
$D_{30}, I_{35}, F_{36}, C_{42}, A_{45}$	S, B	B	10	35	**S->B**
$I_{35}, F_{36}, H_{38}, C_{42}, A_{45}$	S, B, D	D	15	30	**S->B->D**
$F_{36}, H_{38}, C_{42}, A_{45}$	S, B, D, I	I	12	35	**S->B->I**
$H_{35}, C_{42}, G_{42}, A_{45}$	S, B, D, I, F	F	17	36	**S->B->F**
G_{37}, C_{42}, A_{45}	S, B, D, I, F, H	H	25	37	**S->B->F->H**
C_{42}, A_{45}	S, B, D, I, F, H, G	G	37	37	**S->B->F->H->G**

ب)

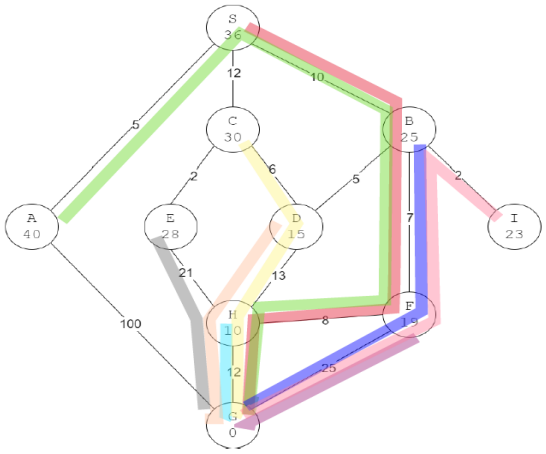
- **Admissible**:

- تعریف:

یک Heuristic h(n) را Admissible میگوییم اگر برای هر هر راس n، h(n) برابر یا کمتر از هزینه واقعی از n به Goal State (G) باشد.

- برای بررسی Admissible بودن heuristic در گراف داده شده باید ابتدا هزینه رسیدن به G را برای هر راس n ا پیدا کنیم و با مقدار h(n) آن راس مقایسه کنیم اگر برای همه راس ها هزینه رسیدن به G بزرگتر مساوری h(n) راس ها باشد، heuristic، Admissible است.

طبق پاسخی که از مسئله اول بدست اوردیم، هزینه رسیدن به G برای S را میدانیم که 37 است. و برای باقی راس ها نیز مسیر های بهینه رسم شده اند.



n	Actual Cost from n to G	h(n)	
S	37	36	✓
A	37 + 5 = 42	40	✓
B	7 + 25 = 32	25	✓
C	30	30	✓
D	25	15	✓
E	33	28	✓
F	25	19	✓
H	12	10	✓
I	34	23	✓
G	0	0	✓

با توجه به جدول میتوان گفت heuristic، Admissible است.

- **Consistent**:

- تعریف :

یک Heuristic h(n) را Consistent میگوییم اگر برای هر دو راس n1 و n2 هزینه رسیدن از n1 به n2 بیشتر مساوی h(n1)-h(n2) باشد.
به تعبیر دیگر مقدار f هیچگاه در میان مسیر کاهش پیدا نمیکند.

- برای بررسی Consistent بودن heuristic در گراف داده شده میتوان اختلاف h(n) هر دو راس مجاور را با هزینه واقعی بین آن دو راس مقایسه کرد:
(صفحه بعد)

n_1	n_2	Cost ($n_1 to\ n_2$)	$h(n_1) - h(n_2)$	
S	A	5	40 – 36 = 4	✓
S	B	10	36 – 25 = 11	✗
S	C	12		
A	G	100		
B	D	5		
B	F	7		
B	I	2		
C	D	6		
C	E	2		
D	H	13		
E	H	21		
F	H	8		
F	G	25		
H	G	12		

در سطر دوم مشاهده میشود که هزینه رسیدن S به B در واقعیت کمتر از از h(B) – h(S) است پس **Heuristic داده شده نمیتواند Consistent باشد،** در نتیجه نیازی به پر کردن باقی جدول نمیباشد.

سوال سوم

الف)

در دنیای واقعی و مسائلی که با آنها رو برو میشویم، ممکن است فضای Search ما بسیار بزرگ باشد و استفاده از دیگر روش ها بخاطر تعداد زیاد State ها معقول نباشد، چرا که هزینه Space یا Time ما بیش از حد توانمان باشد؛ در این موارد رویکرد Local Search برتری دارد.

چرا که به کمک الگوریتم های Local Search، میتوان تمام فضای جستجو را بسط ندهیم و در نتیجه Search با کارایی بهتر داشته باشیم که هزینه Time یا Space کمتری میپردازیم اما به یک جواب معقول(که میتواند optimal نباشد) برسیم.

مثلا در مسائل optimization، مثلا یک شبکه عصبی با میلیون ها یال داریم و میخواهیم که هزینه هرکدام از این یال ها را بدست بیاوریم ، با توجه به فضای بزرگ مسئله میتوانیم یک دور رندوم اول همه رو رندوم تغییر بدهیم، بعد با الگوریتم های local search برای هر یال میتونیم با قدم های کوچک مقادیر را بهبود ببخشیم.

به طور خلاصه:

- حافظه کمی مصرف میکنند ولی در عین حال راه حل معقولی (در مدت زمان مناسب و قابل تحملی) در فضای مسئله بزرگ یا نامتناهی(پیوسته) پیدا میکنند که ا لگوریتم های systematic مناسب نیستند.

ب)

1. اجراهای متعدد:
الگوریتم را از n نقطه رندوم مختلف شروع و اجرا کنیم و از میان جواب هایی که برمیگرداند، بهترین جواب از میان n جواب را انتخاب کنیم.
2. پذیرش ریسک:
با یک احتمال کم هر چند وقت یک بار در جهت رو پایین نیز حرکت میکنیم. (اجازه حرکت downhill نیز به Agent میدهیم.) حتی میتوانیم در ابتدا که Agent در محیط ناشناخته تری قرار دارد با احتمال بالاتری به Agent این اجازه را بدهیم و به مرور این احتمال را کاهش بدهیم.