محمد پیام تائبی ۴۰۰۱۰۴۸۶۷ تمرین اول هوش مصنوعی

۱) الف) غلط، این تعریف Agent است و عاملی که رفتار عقلانی دارد برای شرایط در هر حالت محیط می‌تواند کنشی را انتخاب کند که تابع هدف ما را بیشینه کند.

ب) صحیح، عامل عقلانی با داشتن بینش از محیط می‌تواند تصمیم بگیرد که کدام Action برای ما بهتر است و معیار سنجش ما را بیشینه می‌کند.

ج) غلط، طبق اسلایدها احتمال رفته شدن بدون sideway ۱۴٪ هست که یعنی تقریبا به ۷ random restart نیاز داریم اما با اعمال sideways moves احتمال به ۹۴٪ می‌رسد که تقریبا با ۱ بار اجرا به جواب می‌رسیم پس تاثیر sideways moves بیشتر است. البته اگر موقع داشتن sideway فقط یک بار اجرا کنیم ولی برای اجرای random restart محدودیتی نداشته باشیم، random restart قطعا به جواب می‌رسد ولی sideway با احتمال ۹۴٪ به جواب می‌رسد که باعث می‌شود گزاره صحیح شود

د) صحیح، کلا الگوریتم گفته شده وقتی که branching factor محدود باشد و وزن یال‌ها کران پایین مشخصی داشته باشند روشی بهینه و کامل است.

ه) صحیح، هر دو مقدار ثابتی از حافظه نیاز دارند و مسیر پیموده شده تا این state را نگه نمی‌دارند از طرفی در Simulated Annealing ابتدا حرکت‌های رندوم زیاد و در گذر زمان این مقدار را کم می‌کنیم که باعث می‌شود در دام‌های محلی نیفتیم. همچنین در الگوریتم ژنتیک از mutation یا جهش استفاده می‌کنیم که همین کار را انجام می‌دهد.

۲) الف) ابتدا state ها را مشخص می‌کنیم به گونه‌ای که مکان هر کاوشگر (bot) را به شکل (x,y) نگه داریم در کل n تا کاوشگر داریم پس:

state: $(x_1,y_1),(x_2,y_2),\dots,(x_n,y_n)$

که واضحا

initial state: $\underbrace{(1,1)}_{bot\ 1}\ \underbrace{(1,2)}_{bot\ 2}\ \dots\ \underbrace{(1,n)}_{bot\ n}$

و

goal state: $\underbrace{(n,n)}_{bot\ 1}\ \underbrace{(n,n-1)}_{bot\ 2}\ \dots\ \underbrace{(n,1)}_{bot\ n}$

مکان های قابل دستیابی →

$$\begin{bmatrix} 1,1 & 1,2 & \dots & 1,n \\ 2,1 & 2,2 & \dots & 2,n \\ \vdots & & & \vdots \\ n,1 & n,2 & \dots & n,n \end{bmatrix}_{n\times n}$$

← نمایش داده این جدول را کردیم که سو تفاهمی رخ نده. توی صورت سوال به شکل (y,x)

Actions: هر bot می‌تواند در ۴ جهت حرکت کند یا کلا حرکت نکند یعنی:

$$\forall\ bot_i : \underbrace{(x_i,y_i)}_{\text{مکان فعلی}} \rightarrow \underbrace{(x_i,y_i\pm1),\ (x_i\pm1,y_i)\ \text{یا}\ (x_i,y_i)}_{\text{مکان بعدی}}$$

Action ها را می‌توان به شکل n حرف پشت هم به شکل (N,U,D,L,R) مدل کرد که هر حرف مخصوص bot مورد نظرش است یعنی:

Action: $\underbrace{A_1}_{for\ bot_1}\ \underbrace{A_2}_{for\ bot\ 2}\ \dots\ \underbrace{A_n}_{for\ bot\ n}$ ، $A_i \in \{N,U,D,L,R\}$

N: حرکت نکند، U: بالا، D: پایین، L: چپ، R: راست

پس هر bot ۵ حالت مختلف دارد که کلاً n ، bot داریم پس

5^n حالت مجاور برای هر state تعریف می‌شود. اما دقت کنید یک حالت خاص

هست که صحیح bot حرکت نکند یعنی Action: NNNN...N که به همان

state فعلی می‌رود این Action را از Action های مجاز حذف

می‌کنیم چون بیهوده است پس هر state ، $(5^n - 1)$ همسایه

دارد که تعداد زیادی از آنها valid و تعدادی هم unvalid هستند که باید

چک شود پس دقت کنید A_5 یک اکشن نیست بلکه $A_1 A_2 \dots A_n$ هم با هم یک اکشن حساب می‌شود

Actions: $\{A_1 A_2 \dots A_n : A_i \in \{N, U, D, L, R\}\} - \{NNNN\dots N\}$

Action cost $= 1$ $\rightarrow$ طبق اصلاح شده‌ی سوال در کوئرا و همچنان

سوال اصلی (سوال از سوال ۲۷ فصل ۳ کتاب راسل آمده است)

باید هزینه را هزینه زمانی فرض کنیم یعنی فارغ از اینکه در هر اکشن چند تا ربات حرکت کنند هر $Act = A_1 \dots A_n$ یک هزینه ثابت دارد

Goal state: $(n, n)\ (n, n-1)\ \dots\ (n, 1)$

ب) ابتدا تابع is_state_valid() را تعریف می‌کنیم که یک state

بگیرد و می‌گوید که مجاز هست یا خیر اگر اعداد آن خارج از محدوده

$[1, n]$ باشد یا ۲ bot در یک خانه باشند false برمی‌گرداند و در غیر این

صورت True برمی‌گرداند.

Number of valid states:

باید n تا bot را در n^2 خانه قرار دهیم:

$$\underbrace{\binom{n^2}{n}}_{\text{انتخاب n تا خانه}} \times \underbrace{n!}_{\text{چیدن bot ها}} = \frac{(n^2)!}{(n^2-n)!}$$

$$\frac{(n^2)!}{(n^2-n)!} = \underbrace{(n^2)(n^2-1)(n^2-2)(n^2-3)\cdots(n^2-n+1)}_{\text{تا } n}$$

$$\rightarrow O\left(\binom{n^2}{n} n!\right) = O\left((n^2)^n\right) = O\left(n^{2n}\right)$$

$$\rightarrow \boxed{\text{number of state valid} = O\left(n^{2n}\right)}$$

ج) همانطور که در الف گفته شد هر حالت $5^n - 1$ همسایه دارد که تعدادی از آن unvalid است پس **کران بالای برای $5^n - 1$ Branching factor است**

د) ساده ترین تابع Heuristic فاصله منهتن است یعنی به شکل زیر تعریف می‌شود:
تابع h_i برای bot i ام است که می‌خواهد از $(1, i)$ به خانه $(n, n-i+1)$ برود

$$h_i(x,y) = \underbrace{n - x}_{\text{حداقل حرکت برای رسیدن به سطر } n} + \underbrace{\| n-i+1-y \|}_{\text{حداقل حرکت برای رسیدن به ستون } n-i+1}$$

بدیهی است که چون در هر جهت فقط حرکت می‌کنیم فاصله کمتری از فاصله منهتن نمی‌توان ارائه داد از طرفی در هر گام نمی‌توان بیشتر از یک حرکت برای bot_i انجام داد پس اگر هزینه واقعی را h^* فرض کنیم قطعاً $h_i \leq h_i^*$ که یعنی h_i یک admissible heuristic هست.

اگر bot های دیگر هم در فضا قرار گیرند قطعاً مسیر حرکت یا سخت تر می‌شود یا تغییری نمی‌کند

یعنی با h^* بیشتر می‌شود یا تغییری نکرده همچنان h_i کران پایینی از آن است.

پس همچنان h_i یک admissible heuristic است که کران پایینی از cost باقی

تا رسیدن به goal را ارائه می‌دهد.

---

e)

طبق h_i تعریف شده در بخش قبل $\max(h_1,\dots,h_n)=h_1$ و $\min(h_1,\dots,h_n)=h_{\frac{n}{2}}$

می‌شود. فرض کنیم شرط برخورد ۲ ارابه تا آنجا که ایجاد نمی‌کند همچنان کران پایینی

هزینه‌ای که باید بدهیم می‌شود $\max(h_1,\dots,h_n)$ و حتی در این محدودیت اعمال شود

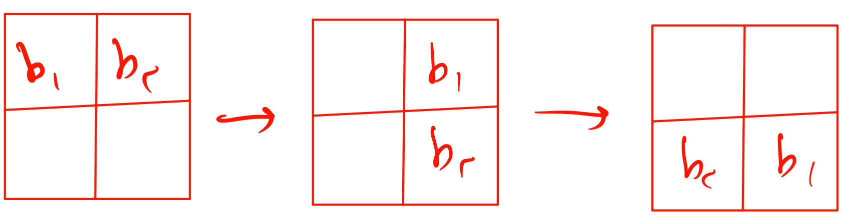
یعنی نتوانیم ۲ bot را در یک خانه قرار دهیم قطعا h^* (مقدار فاصله واقعی) از این

عدد بیشتر می‌شود پس

$$\min(h_1,\dots,h_n) \le \frac{\sum_i h_i}{n} \le \max(h_1,\dots,h_n) \le h^*$$

و این سه کران پایینی برای جواب هستند پس هیوریستیک‌های قابل

قبولی‌اند و اما برای ۳ هیوریستیک دیگر می‌شود مثال نقض زد: فرض کنید $n=2$



این مسئله در ۲ حرکت حل می‌شود

یعنی $h^*=2$

و $h_1=2$, $h_2=2$ که

$\sum h_i = 4 > h^* = 2$

$n\max(h_1,h_2) = 4 > h^* = 2$

$n\min(h_1,h_2) = 4 > h^* = 2$

پس تنها سه تعریف فقط $\min(h_1,\dots,h_n)$, $\max(h_1,\dots,h_n)$ و $\frac{\sum h_i}{n}$ قابل

قبول (admissible) هستند

✡ در آخر فرض کنیم که صورت سوال اصلاح شده نباشد یعنی:

Action-Cost: طبق صورت سوال در یک Action جمع تعداد حروف {U D L R} هزینه را می‌سازد

یعنی Action cost 1 = n - number of "N" in Action

که یعنی جمع عملیات کاوشگرها مورد نظر باشد و زمان مهم نباشد این بازی

حل ساده استراتژی برد بدیهی دارد به طوری که:

bot1 را به مقصد برسانیم با هزینه h_1 سپس bot2 را با h_2 و ... که سوال را بدیهی می‌کند

و در کمترین هزینه ممکن به goal state برسیم و طبق این cost تمام hها به جز

$n \max(h_1 \dots h_n)$ قابل قبول بودند اما اصلا نیازی به الگوریتم سرچ نبود چون

همه را حل بدیهی داشت.

و) مدل استارتی از اول C:

الف)

states: $\underbrace{(x_1, y_1)}_{\text{bot 1}}$ $\underbrace{(x_2, y_2)}_{\text{bot 2}}$ ... $\underbrace{(x_n, y_n)}_{\text{bot n}}$

Valid states → مثل سوال قبل تعریف می‌شود → فقط باید حالاتی که bot_{i+k} به مقصد رسیده ولی bot_i به مقصد نرسیده را unvalid برگردانیم در واقع دارای ترتیب رسیدن bot ها را به مقصد سوال معمول می‌کنیم.

Actions:
در هر کنش، یک ربات انتخاب می‌شود و یکی از ۴ حرکت مجاز را انجام می‌دهد
(واضحا حرکت N به state فعلی می‌رود و آن را حساب نمی‌کنیم)

Acts: $\{A_i : A_i = \{U, D, L, R\}\}$

$U \to x_i{++}$, $D \to x_i{--}$, $L \to y_i{--}$, $R \to y_i{++}$

cost Act = 1 for each Act

goal state → مثل قبل

ب) مثل مثل قبلی است فقط حالاتی که bot_{i+k} به مقصد رسیده ولی bot_i نرسیده

حذف می‌شود که خب می‌توان گفت فضا همچنان $O(n^{2n})$ است

ج) هر state $4n$ همسایه دارد که بخشی از آن unvalid است پس

wrost branching factor = $4n$

طبق مدل معرفی شده اگر h را مثل بخش قبل بگیریم به احتمال زیاد [illegible]

به state هایی که نزدیک جواب اند برویم و در آنجا گیر کنیم چون در h

ترتیب قرارگیری را اعمال نکردیم پس احتمالاً باید h را طور دیگری قرارداد

تا ترتیب قرارگیری bot ها هم در h تاثیرگذار باشد.

۳) آ)

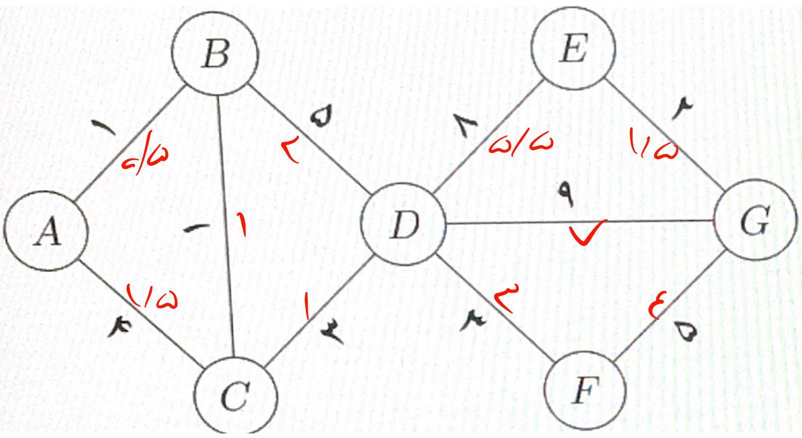
A second, slightly stronger condition called **consistency** (or sometimes **monotonicity**) is required only for applications of A* to graph search.[9] A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$\boxed{h(n) \le c(n, a, n') + h(n')\ .}$$

$n \rightarrow n'$

طبق تعریف یکنوایی (consistency) باید به‌ازای هر یال a→b داشته باشیم

$$|\Delta h_{a\to b}| \le w_{ab}$$



Node	h_1	h_2
A	۹/۵	۱۰
B	۹	۱۲
C	۸	۱۰
D	۷	۸
E	۱/۵	۱
F	۴	۴/۵
G	۰	۰

که برای h_1 در تمام یال های گراف، این ویژگی برقرار است. (Δh را با رنگ قرمز روی تمام یال ها مشخص کرده ام)

اما برای h_2 مثال نقض وجود دارد:

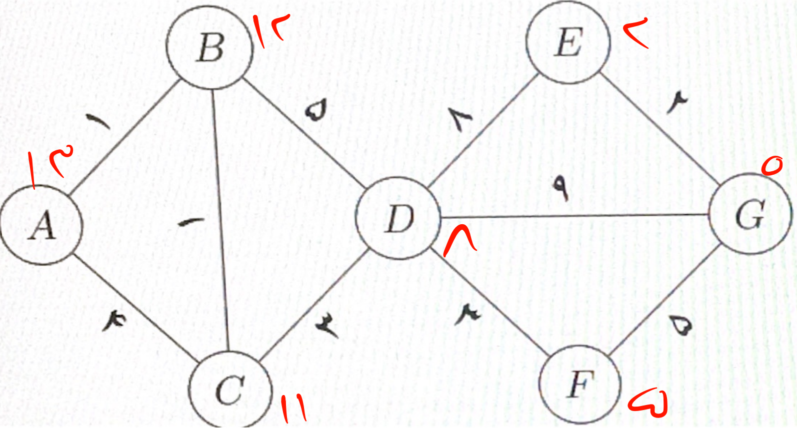
$A \rightarrow B$

$|h_A - h_B| = ۲$, $w_{AB} = ۱$ $\longrightarrow$ h_2 is not consistent

پس h_1 یکنوا است اما h_2 یکنوا نیست.

حال قابل قبول بودن h ها را بررسی می‌کنیم: وقتی که h مقدار آن پایین‌تر از فاصله واقعی باشد می‌توان گفت h admissible (قابل قبول) است.

که در گراف زیر کمترین فاصله واقعی هر راس از G را مشخص کرده ام تا به بررسی h بپردازم



Node	h_1	h_2
A	✓ ۹/۵	✓ ۱۰
B	✓ ۹	✓ ۱۲
C	✓ ۸	✓ ۱۰
D	✓ ۷	✓ ۸
E	✓ ۱/۵	✓ ۱
F	✓ ۴	✓ ۴/۵
G	✓ ۰	✓ ۰

که مشخص است تمام اعداد داده شده در جدول بالا بزرگ‌تر از عدد متناظر قرمز رنگ در شکل گراف است.

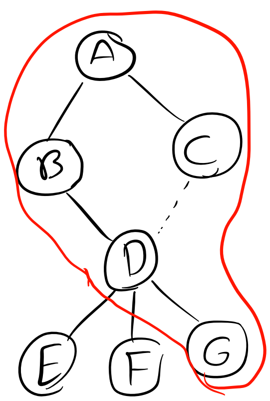
پس h_1 و h_2 هر ۲ قابل قبول (admissible) هستند.

ب)

	A-B-D-G	A-C-D-G	A-B-C-D-F-G
DFS ۱۳ جست‌وجوی عمق اول	✓	✓	✓
BFS ۱۴ جست‌وجوی سطح اول	✓	✓	✗ ✓
Dijkstra ۱۵ جستجوی هزینه یکنواخت	✗	✗	✓
جست‌وجوی A^* با تابع اکتشافی h_1	✗	✗	✓
جست‌وجوی A^* با تابع اکتشافی h_2	✗	✗	✓

BFS Tree

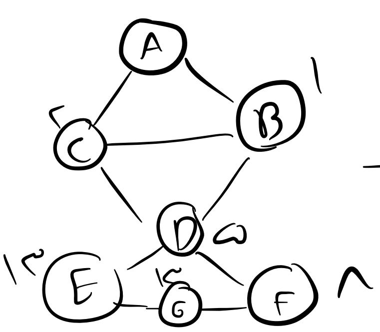
۲ جواب ممکن



DFS Tree → تمام مسیرهای ممکن را ملاک است

به خصوص چون مشخص نیست از کدام همسایه راس شروع می‌کند به ادامه گراف

Dijkstra



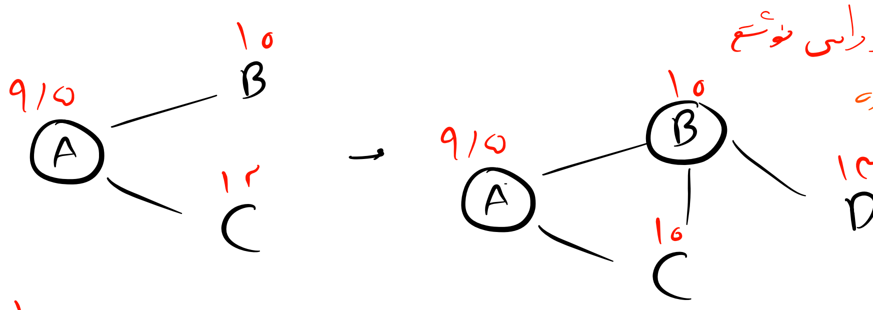
→ best path = A B C D F G

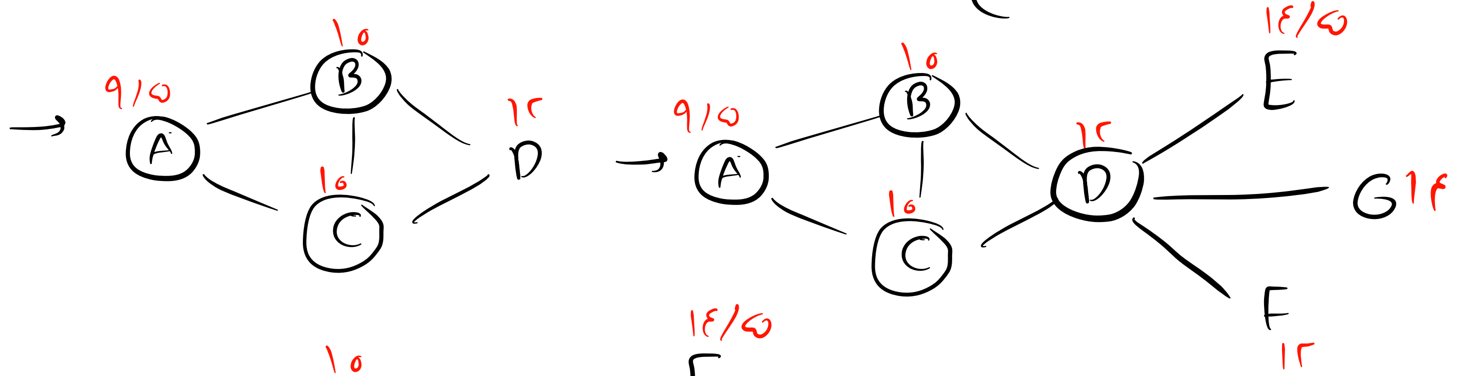
Dijkstra is optimal → پس واقعا مسیر درست را می‌دهد ABCDFG

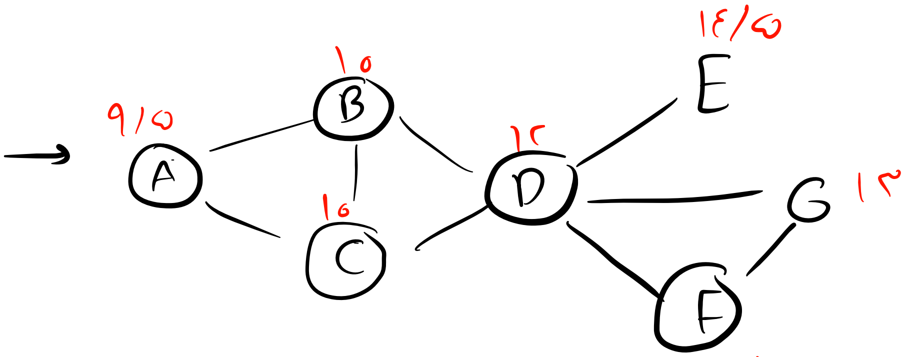
h_1 A^*:

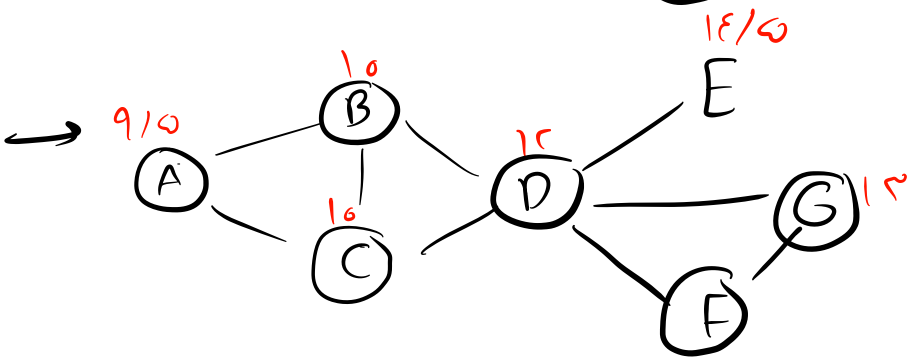
$h+g$ را با قرمز بالای هر راس نوشتم

دور نود expand شده دایره کشیدم.









به G رسیدیم پس

جواب پیدا شد

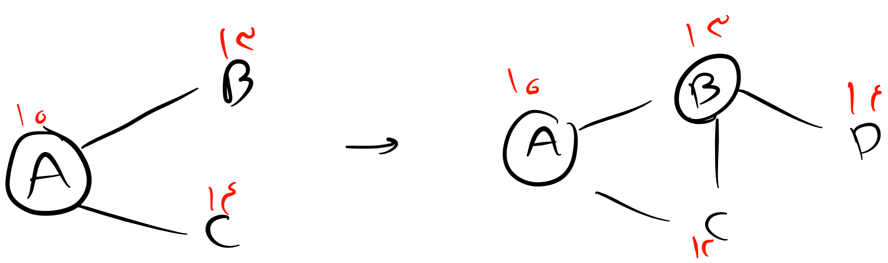
که با استفاده از درختی که داریم پدران G را به شکل برعکس می‌نویسیم:

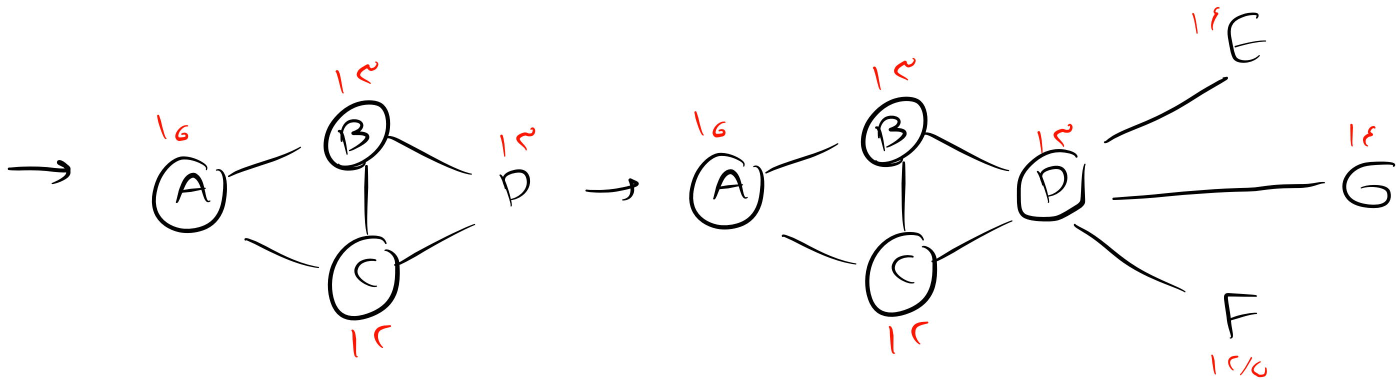
A B C D F G

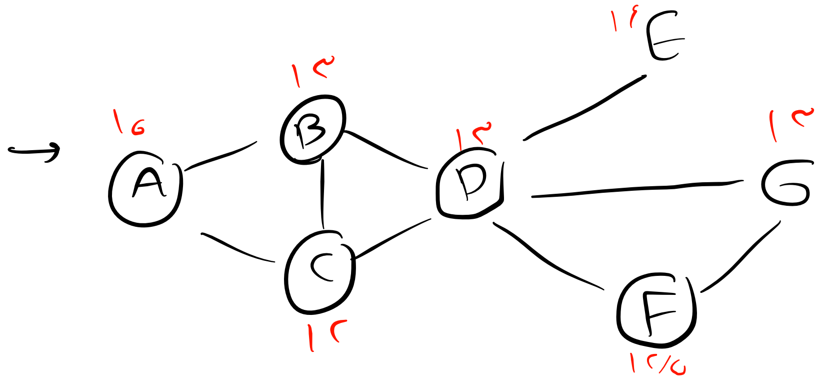
دقت کنید h_1 هم قابل قبول بود هم یکنوا.

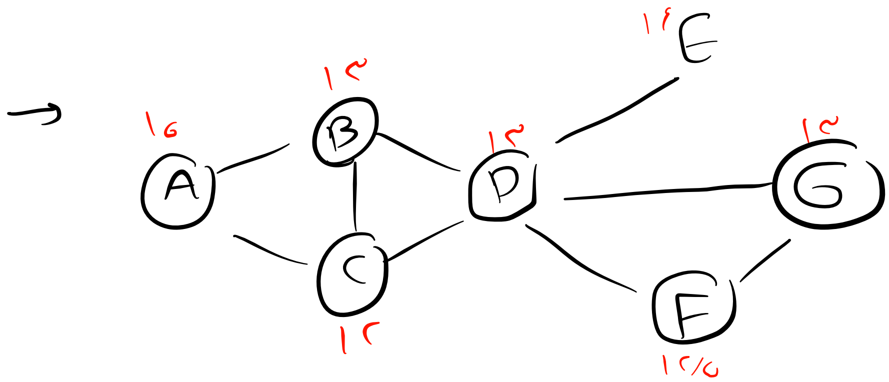
درضمن ما Graph search زدیم یعنی راس‌های تکراری را مجددا اضافه نکردیم و راس‌هایی که expand شوند به closed list میروند و دوباره expand نمی‌شوند

h_2 A^*:









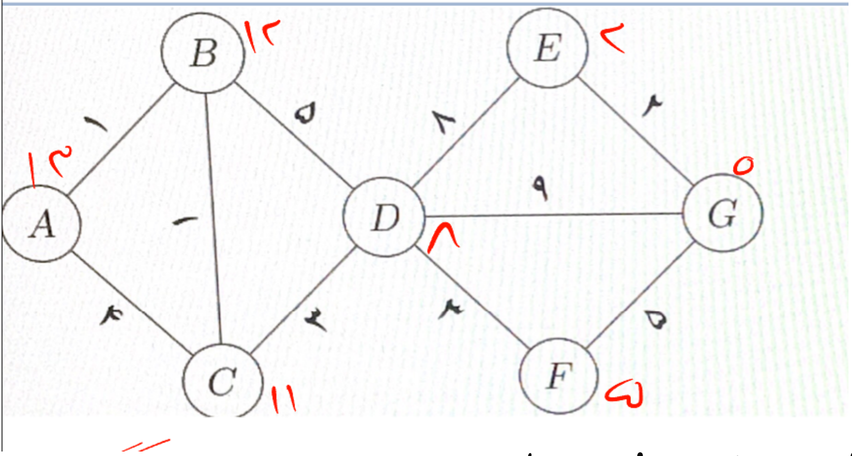
به G رسیدیم و مسیر ما شد

A B C D F G

اینجا هم مسیر درست پیدا شد اما چون h_1 یکنوا نیست جستجوی گراف optimal نیست، و هیچ به جواب بهینه نمی‌رسد اما اگر جستجوی درختی انجام بدهیم مشابه admissible بودن نیز دیدیم که می‌شود تضمین کرد که h_1 درست کار می‌کند اما در این مثال شانسی درست کار کرد البته حتی بدون monotonic بودن هم می‌شود سرچ گراف زد اگر این بخش را به کد اضافه کنیم:

```
elif g[m] > g[n] +  G.weight(n,m):
        g[m] = g[n] + weight
        parents[m] = n
        if G.is_visited(m):  → in closed_list
            G.clear(m,False)
            queue.add(m)  → add to fringe agin
```

در واقع در این کد اگر حالتی که قبلاً expand شده پیدا شود که با مسیر بهتری می‌توان به آن رسید دوباره آن را باز می‌کند که باگ monotonic نبودن را حل می‌کند.



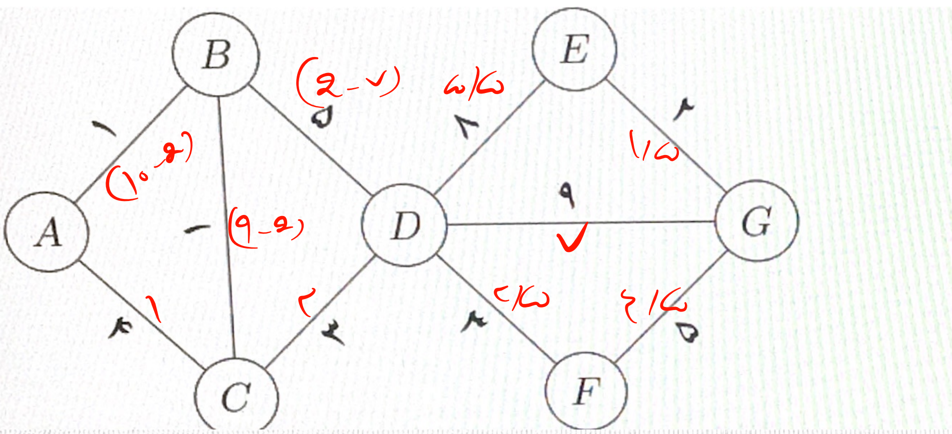
ج) در گراف روبه رو فاصله واقعی هر راس از G را نوشته ایم که واضحاً h_2 باید در هر راس کران پایینی از این عدد باشد تا بتواند admissible باشد پس

h_2 in B $\le$ ۱۲ → این مقدار از ۵ تا ۱۲ مقدار قابل قبولی است

د) باید مثل جنس h_1 و h_2 چک کنیم که $\forall i,j \quad \|\Delta h_{i,j}\| \le w_{ij}$

که h ها در شکل مشخص کردم که باید برای تمام یال ها کمتر از وزن یال باشد

پس:



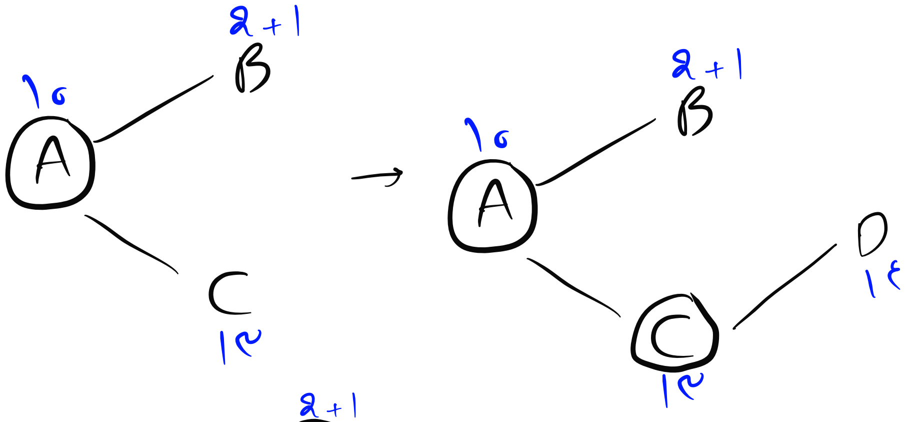
Node	A	B	C	D	E	F	G
h_3	۱۰	x	۹	۷	۱/۵	۴/۵	۰

$$\big||10-x|\big| \le 1 \,,\quad |7-x| \le 5 \,,\quad |9-x| \le 1$$

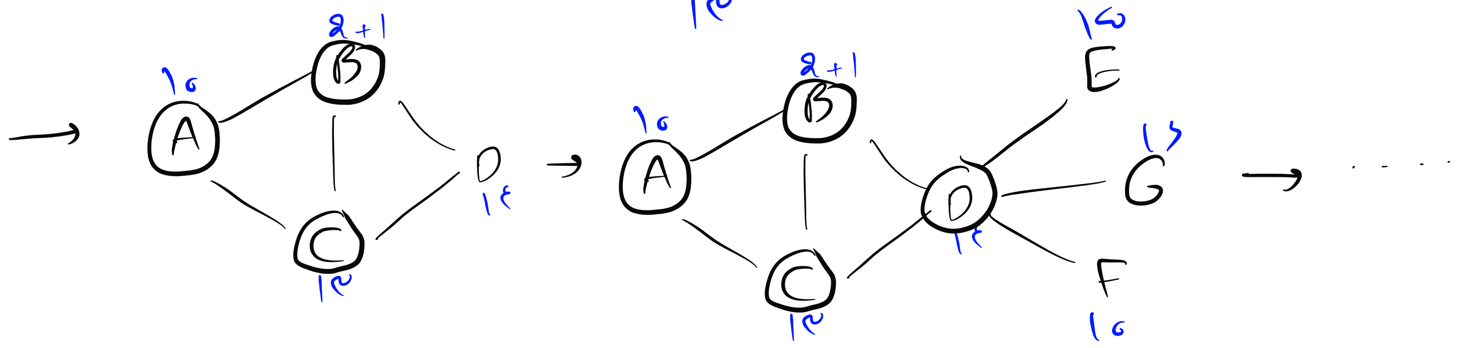
$$\to 9 \le x \le 10$$

→ پس مقدار مورد نظر بین ۹ تا ۱۰ است

۵)



$$۱۳ \leq d+1$$



$$d+1 \leq ۱۴$$

$$\rightarrow \quad ۱۲ \leq d \leq ۱۳$$

جواب ندارد چون همواره مسیر $A \rightarrow B$ هزینه کمتری از $A \rightarrow C \rightarrow B$

دارد یعنی غیر ممکن است که $parent[B] = C$ بشود

اما با این حال اگر $۱۲ \leq h_B \leq ۱۳$ باشد می‌توان گفت که

اول A بعد C بعد B بعد D و ... باز می‌شوند اما خب چون

B توسط A باز شده و نه توسط C پس مسئله جواب ندارد

۴) الف) کافی است به هر کلمه یک شماره دهیم و جایگاه کلمه را مشخص می‌کند در واقع n کلمه مختلف داریم که $n!$ state مختلف می‌سازد

ب) هر Action را جابه‌جا کردن ۲ کلمه فرض کنید پس حالت‌های مجاور یک state همه جملاتی هستند که جای ۲ کلمه با هم در آن‌ها عوض شده است. واضحا هر state $\binom{n}{2}$ همسایه دارد

مثال:

هرچه	زودتر	خواهش	می‌کنم	آزاد	کنید	من	را
۸	۷	۶	۵	۴	۳	۲	۱

هرچه	زودتر	من	می‌کنم	آزاد	کنید	خواهش	را
۸	۷	۲	۵	۴	۳	۶	۱

می‌کنم	زودتر	خواهش	هرچه	آزاد	کنید	من	را
۵	۷	۶	۸	۴	۳	۲	۱

ج) Hill climbing از یک حالت شروع کند و همواره بهترین همسایه را انتخاب می‌کند و جلو می‌رود (حالت greedy دارد) که باعث می‌شود با توجه به نقطه شروع به نزدیک‌ترین Local minimum برسیم که خب واضحا می‌تواند در بسیاری از مسائل جواب مسئله نباشد. علاوه بر این ممکن است در دور بیفتد و نتواند جواب پیدا کند یعنی این ایده complete نیست برای حل این مشکل مقداری احتمال را وارد حل می‌کنیم یعنی از stochastic Hill climbing استفاده می‌کنیم که در هر

گام با احتمالی random walk می‌زنه که باعث می‌شود در local minimum گرفتار نشود علاوه بر این وقتی در لوپ بیوفتیم یا به local minimum برسیم برنامه را دوباره از اول اجرا می‌کند (از نقطه دیگری random restart می‌زنیم) در ضمن می‌توان از sideway move برای جلوگیری از لوپ هم استفاده کرد که همه این‌ها در کنار هم با اینکه در حالت کلی همچنان complete نیستند اما می‌توان از آن برای حل مسئله استفاده کرد

د) ابتدا مسئله را مدل می‌کنیم:

خواهش (۸) ملیکا (۷) هرچه (۶) زودتر (۵) من (۴) را (۳) آزاد (۲) کنید (۱)

از یک ترتیب شانسی $\{a_1, a_2, \dots, a_8\}$ که initial_state هست شروع می‌کنیم و هدف این هست که به ۱۲۳۴۵۶۷۸ برسیم

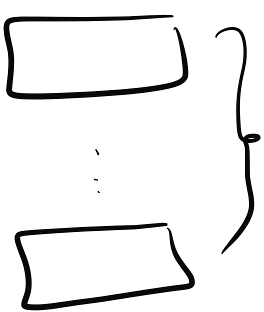
دقت کنیم که به هر کلمه یک شماره دادیم و ترتیب خاص (از آن اعداد جواب مسئله‌ها) یعنی جمله‌ی:

ملیکا (۷) کنید (۱) من (۴) هرچه (۶) زودتر (۵) آزاد (۲) را (۳) خواهش (۸) معادل عدد

۸۳۲۵۶۴۱۷ هست.

حالا باید الگوریتم ژنتیک را پیاده‌سازی کنیم.

۱) ابتدا k حالت رندوم درست می‌کنیم



k تا

سپس با استفاده از تابعی که داشتیم (evaluation) عددی به هر حالت
نسبت می‌دهیم سپس fitness function به دست می‌آید. (واضحا احتمال
نسبت داده شده به هر حالت از رابطه $\frac{v_i}{\sum v_i}$ به دست می‌آید که
v_i مقدار به دست آمده از تابع evaluation داده شده در صورت سوال است)

در گام بعدی تا k state از این state ها را مستقلاً با احتمال داده
شده انتخاب می‌کنیم (واضحا حالت تکراری هم مجاز هست) که به این عملیات
selection می‌گویند

در گام بعدی به طور شانسی هر ۲ تای از k حالت را با هم جفت
می‌کنیم.

حال باید به ازای هر جفت ۲ حالت جدید تولید کنیم (crossover)
چون اعداد متضاد اند و state از نوع Permutation هست از
order crossover استفاده می‌کنیم که در زیر شرح داده شده است:

Order crossover (OX1) [edit]

The order crossover goes back to Davis[1] in its original form and is presented here in a slightly generalized version with more than two crossover points. It transfers information about the relative order from the second parent to the offspring. First, the number and position of the crossover points are determined randomly. The resulting gene sequences are then processed as described below:

Procedure	*Example*
• Let be given two permutations of the same set	$P_0 = (A, B, C, D, E, F, G, H, I, J)$ and $P_1 = (B, D, A, H, J, C, E, G, F, I)$
• and a random selection of gene segments in P_0. Here from gene position 1 to 2 and from 6 to 8.	$P_0 = (\underline{A, B}, C, D, E, \underline{F, G, H}, I, J)$
• As a child permutation, a permutation is generated that contains the selected gene segments of P_0 in the same position.	$P_C = (A, B, ?, ?, ?, F, G, H, ?, ?)$
• The remaining missing genes are now also transferred, but in the order in which they appear in P_1.	$P_{\text{missing}} = \{C, D, E, I, J\}$ $P_{\text{in order from } P_1} = (D, J, C, E, I)$
• This results in the completed child genome.	$P_C = (A, B, \underline{D, J, C}, F, G, H, \underline{E, I})$

که طبق عکس ابتدا از s_1 تعدادی حرف به شکل رندوم انتخاب می‌کنیم
سپس باقی حروف را در مجموعه $S_{missing}$ بریزید سپس ترتیب
این اعضا را از s_2 برداریم و طبق همان ترتیب در s_1 قرار دهیم
حال s_1 همان حالتی است که به دنبال آن بودیم (child 1) برای
دومین child هم دقیقا همین کار را باید کرد یعنی همان حالتی که از s_1
برداشتیم را این دفعه از s_2 برمی‌داریم و باقی را از s_1 پر می‌کنیم:

مثال:

S_1 = ۲ ۳ ۴ ۱ ۵ ۷ ۶ ۸

S_2 = ۱ ۸ ۷ ۶ ۲ ۳ ۴ ۵

random select → S_1 = ۲ ۳ ۴ ۱ ۵ ۷ ۶ ۸ S_2 = ۱ ۸ ۷ ۶ ۲ ۳ ۴ ۵

→ C_1 = ۲ ؟ ۴ ۱ ؟ ؟ ۶ ؟

?set = {۳, ۵, ۷, ۸}

sort ? from s_2 = {۸, ۷, ۳, ۵}

Final C_1 = ۲ ۸ ۴ ۱ ۷ ۳ ۶ ۵

C_2 = ۱ ؟ ۷ ۶ ؟ ؟ ۴ ؟

?set = {۸, ۲, ۳, ۵}

sort ? from s_1 = {۲, ۳, ۵, ۸}

C_2 = ۱ ۲ ۷ ۶ ۳ ۵ ۴ ۸

به طور خلاصه به طور رندوم k تا عدد را انتخاب می‌کنیم و جایگاه این k تا را در جمله
دوم پیدا می‌کنیم و به همان ترتیب جمله دوم در جمله اول قرار می‌دهیم و برعکس.

حال به مرحله mutation می‌رسیم

از بین mutation های معروف (swap, insert, scramble, inverse)

در این مثال چون از نوع جایگشت هست swap منطقی‌ترین است

یعنی باید اگر طبق یک احتمالی تصمیم به جهش (mutation) گرفتیم،

باید ۲ خانه شانسی را انتخاب کنیم و جای آن ۲ را با هم عوض کنیم.

- ۵

$\textcircled{T}$

$$f: R^n \to R,\ f_{(x)} = \|x\|_2^2 = x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2$$

$$\to \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 2x_2 \\ \vdots \\ 2x_n \end{bmatrix} = 2x \to \boxed{\nabla f = 2x}$$

$$f: R^n \to R,\ f_{(x)} = \|Ax\|_2^2 = \langle Ax \cdot Ax \rangle = x^T A^T A x$$

$$\to \nabla f = \frac{\partial f}{\partial x} = \frac{\partial x^T A^T A x}{\partial x} = 2A^T A x \to \boxed{\nabla f = 2A^T A x}$$

$$f: R^n \to R,\ f_{(x)} = \|Ax - b\|^2 + \gamma \|x\|^2$$

$$\to f_{(x)} = \langle Ax - b \cdot Ax - b \rangle + \gamma \|x\|^2$$

$$= (x^T A^T - b^T)(Ax - b) + \gamma \|x\|^2$$

$$= x^T A^T A x - x^T A^T b - b^T A x + b^T b + \gamma \|x\|^2$$

$$= \left(x^T A^T A x - 2x^T A^T b + b^T b\right) + \gamma \|x\|^2$$

$$\frac{\partial x^T A^T A x}{\partial x} = 2A^T A x,\quad \frac{\partial x^T A^T b}{\partial x} = A^T b,\quad \frac{\partial \|x\|^2}{\partial x} = 2x$$

$$\to \frac{\partial f}{\partial x} = \frac{\partial x^T A^T A x}{\partial x} - 2\frac{\partial x^T A^T b}{\partial x} + \frac{\partial b^T b}{\partial x} + \gamma \frac{\partial \|x\|^2}{\partial x}$$

$$\rightarrow \frac{\partial f}{\partial x} = 2A^TAx - 2A^Tb + 0 + 2\gamma x$$

$$\rightarrow \nabla f = 2A^TAx - 2A^Tb + 2\gamma x$$

$$\rightarrow \boxed{\nabla f = 2A^T(Ax - b) + 2\gamma x}$$

ب) طبق الگوریتم معرفی شده کافی است برای α های مختلف الگوریتم را پیاده سازی کنیم

Algorithm: Gradient Descent
Given:
Function f, initial point x_0, step size $\alpha > 0$
Initialize:
$x \leftarrow x_0$
Repeat until convergence:
$x \leftarrow x - \alpha \nabla_x f(x)$

1) $x \leftarrow x - 2\alpha x$

2) $x \leftarrow x - 2\alpha A^TAx$

3) $x \leftarrow x - 2\alpha\left(A^T(Ax - b) + 2\gamma x\right)$

repeat

مقدار α را نیز باید با آزمون و خطا به دست بیاوریم (بهتر است در گذر زمان α را کوچک تر هم کنیم .

Convex functions

ج

- A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if, for any $x, y \in \mathbb{R}^n$ and $0 \leq \theta \leq 1$

$f = \|x\|^2$

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y)$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad f_{(x)} = \|x\|^2 = x^T x$$

$$\theta x + (1-\theta) y = \begin{bmatrix} \theta(x_1 - y_1) + y_1 \\ \theta(x_2 - y_2) + y_2 \\ \vdots \\ \theta(x_n - y_n) + y_n \end{bmatrix} = \theta(x - y) + y$$

$$\rightarrow f_{(\theta x + (1-\theta) y)} = (\theta x^T + (1-\theta) y^T)(\theta x + (1-\theta) y)$$

$$= \theta^2 x^T x + \theta(1-\theta) x^T y + \theta(1-\theta) y^T x + (1-\theta)^2 y^T y$$

$$= \theta^2 f_{(x)} + 2\theta(1-\theta) x^T y + (1-\theta)^2 f_{(y)}$$

$, |\langle x, y \rangle| \leq \|x\| \times \|y\|$ (cauchy schwarz)

$$\rightarrow x^T y \leq (x^T x)^{\frac{1}{2}} (y^T y)^{\frac{1}{2}}$$

$$\rightarrow f(\theta x + (1-\theta) y) \leq \theta^2 f(x) + 2\theta(1-\theta)(x^T x)^{\frac{1}{2}}(y^T y)^{\frac{1}{2}}$$

$$+ (1-\theta)^2 f(y)$$

$$= \theta^2 f(x) + 2\theta(1-\theta)\sqrt{f(x)}\sqrt{f(y)} + (1-\theta)^2 f(y)$$

$$= \underbrace{\left(\theta\sqrt{f(x)} + (1-\theta)\sqrt{f(y)}\right)^2 \leq \theta f(x) + (1-\theta) f(y)}$$

چرا؟ چون تابع $f(x) = x^2$ ، $x \in \mathbb{R}$ یه تابع convex هست یعنی

$$f\left(\theta\sqrt{f(x)} + (1-\theta)\sqrt{f(y)}\right) \leq \theta f(x) + (1-\theta) f(y)$$

$$\rightarrow \left(\theta\sqrt{f(x)} + (1-\theta)\sqrt{f(y)}\right)^2 \leq \theta f(x) + (1-\theta) f(y)$$

پس در نهایت:

$$f(\theta x + (1-\theta) y) \leq \left(\theta\sqrt{f(x)} + (1-\theta)\sqrt{f(y)}\right)^2 \leq \theta f(x) + (1-\theta) f(y)$$

$$\rightarrow f(\theta x + (1-\theta) y) \leq \theta f(x) + (1-\theta) f(y) \checkmark$$

حالا بیاییم ثابت کنیم $f(x) = x^2$ یه تابع محدب است، اثبات:

$\forall\ \theta$ in $[0,1]$:

$$\theta(\theta-1)(x-y)^2 \leq 0 \rightarrow \theta(\theta-1)x^2 - 2\theta(\theta-1)xy + \theta(\theta-1)y^2 \leq 0$$

$\rightarrow (\theta^2-\theta)x^2 + 2\theta(1-\theta)xy + (\theta^2-\theta)y^2 \le 0$

$\rightarrow \theta^2 x^2 + 2\theta(1-\theta)xy + [(1-\theta)^2-(1-\theta)]y^2 \le \theta x^2$

$\rightarrow \theta^2 x^2 + 2\theta(1-\theta)xy + (1-\theta)^2 y^2 \le \theta x^2 + (1-\theta)y^2$

$\rightarrow (\theta x + (1-\theta)y)^2 \le \theta x^2 + (1-\theta)y^2$

$\rightarrow f(\theta x + (1-\theta)y)^2 \le \theta x^2 + (1-\theta)y^2$

$\rightarrow f(x) = x^2$ is convex function we know $f_{(x)}$ is convex, prove $g_{(x)} = f_{(Ax-b)}$ is convex

so we prove: $g_{(\alpha x + (1-\alpha) y)} \le \alpha g_{(x)} + (1-\alpha) g_{(y)}$

$$g_{(\alpha x + (1-\alpha) y)} = f\big(A(\alpha x + (1-\alpha) y) - b\big)$$

$$= f\big(\alpha A x + (1-\alpha) A y - b\big)$$

$$= f\big(\alpha (Ax - b) + (1-\alpha)(Ay - b)\big)$$

Let $Ax - b = x'$, $Ay - b = y'$

$$\rightarrow g_{(\alpha x + (1-\alpha) y)} = f_{(\alpha x' + (1-\alpha) y')}$$

and f is convex so:

$$f_{(\alpha x' + (1-\alpha) y')} \le \alpha f_{(x')} + (1-\alpha) f_{(y')}$$

$$\rightarrow g_{(\alpha x + (1-\alpha) y)} \le \alpha f_{(x')} + (1-\alpha) f_{(y')}$$

$$= \alpha \underbrace{f_{(Ax-b)}}_{g_{(x)}} + (1-\alpha) \underbrace{f_{(Ay-b)}}_{g_{(y)}}$$

→ $g_{(\alpha x + (1-\alpha) y)} \le \alpha g_{(y)} + (1-\alpha) g_{(y)}$

→ $g_{(x)}$ is convex function

ه

$f_{(x)} = \|x\|^2$ is convex (طبق ج)

→ $f_{(Ax-b)} = \|Ax-b\|^2$ is convex (طبق د)