سؤال ۱. Binarization of CSP

- الف) برای این‌که یک محدودیت سه‌گانه تعریف کنیم، سه متغیر A، B و C را به شکل زیر تعریف می‌کنیم:

$$A + B = C$$

یک متغیر جدید به نام AB تعریف می‌کنیم. اگر دامنه‌ی A و B مجموعه اعداد N باشد، آنگاه دامنه‌ی AB، مجموعه‌ی $N \times N$ خواهد بود. حال سه محدودیت دوگانه داریم:

۱. یکی بین A و AB که بیانگر این است که مقدار A باید برابر با اولین عضو دوتایی[۱] AB باشد.
۲. یکی بین B و AB که بیانگر این است که مقدار B باید برابر با دومین عضو دوتایی AB باشد.
۳. در نهایت یکی که بیانگر این است که جمع دو عضو باید برابر با مقدار C باشد.

همان‌طور که نشان داده شد، توانستیم یک محدودیت سه‌گانه را به محدودیت دوگانه تبدیل کنیم. هم‌چنین می‌توانیم یک محدودیت چهارگانه متغیرهای A، B، C و D را کاهش دهیم. ابتدا باید مراحل بالا را برای A، B و C انجام دهیم تا محدودیت‌های دوگانه ایجاد شود و سپس با دوباره اضافه کردن D یک محدودیت سه‌گانه جدید ایجاد می‌شود که همانند فرآیندهای بالا قابل تبدیل به محدودیت دوگانه است.

به همین ترتیب با استقرا می‌توان نتیجه گرفت که هر محدودیت nگانه را می‌توان به محدودیت (n - 1)گانه تبدیل کرد.

نکته: می‌توان در مرحله‌ی محدودیت دوگانه توقف کرد. زیرا هر محدودیت یگانه را می‌توان به راحتی با حذف کردن آن از دامنه‌ی متغیر اعمال کرد.

- ب) چون متغیر D، محدودیت یگانه دارد و مقدار آن از پیش تعیین شده، کاری به آن نداریم و باید به متغیرها و دامنه‌های زیر توجه کرده و با توجه به توضیحات داده شده در قسمت (الف) عمل کنیم:

$A \in \{1, 2, 5\}$
$B \in \{1, 4, 5, 6, 7\}$
$C \in \{10, 12\}$

$A + B = C$
$A < B$

یک متغیر جدید به نام AB در نظر می‌گیریم که عضو اول آن از دامنه‌ی متغیر A و عضو دوم آن از دامنه‌ی متغیر B است. حال با ضرب دکارتی دامنه‌ی A در دامنه‌ی B داریم:

$$\begin{aligned} AB \in \{&(1,1,10),(1,1,12),(1,4,10),(1,4,12),(1,5,10),(1,5,12),(1,6,10),(1,6,12),(1,7,10),(1,7,12),\\ &(2,1,10),(2,1,12),(2,4,10),(2,4,12),(2,5,10),(2,5,12),(2,6,10),(2,6,12),(2,7,10),(2,7,12),\\ &(5,1,10),(5,1,12),(5,4,10),(5,4,12),(5,5,10),(5,5,12),(5,6,10),(5,6,12),(5,7,10),(5,7,12)\} \end{aligned}$$

حال با اعمال محدودیت‌های دوگانه به مجموعه جواب زیر می‌رسیم:

$$AB \in \{(5, 7, 12)\}, D = 11$$

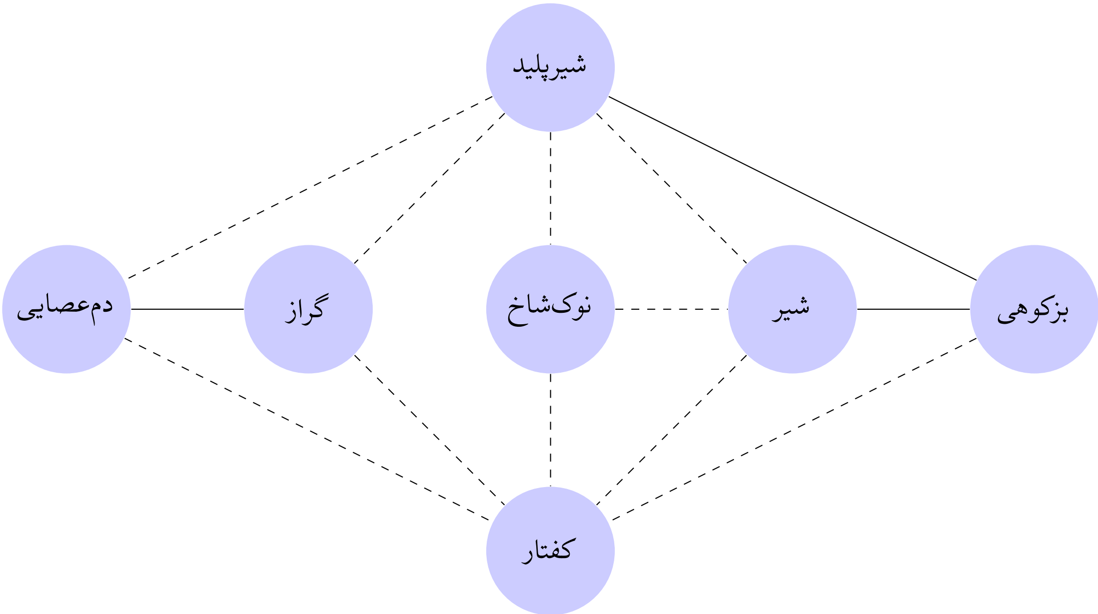
[۱] زوج مرتب

سؤال ۲. CSP

- الف)

۱. متغیرها = { شیر، شیر پلید، دم‌عصایی، گراز، کفتار، نوک‌شاخ، بزکوهی }
۲. دامنه هر کدام = { ۱، ۲، ۳، ۴ }

نکته: این دامنه بدون در نظر گرفتن محدودیت‌هایی است که در صورت سوال ذکر شده است. جلوتر به دامنه‌ی هر کدام خواهیم رسید.



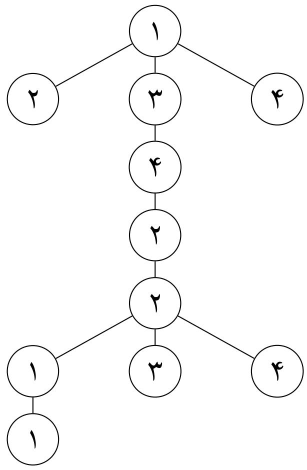
شکل ۱: نقطه‌چین بیان‌گر این است که نباید در یک خانه قرار بگیرند. خط هم بیان‌گر این است که یا حتما باید در یک خانه قرار بگیرند و یا نمی‌توانند همسایه یا در یک خانه قرار گیرند

حیوان	دامنه
شیر	۱
نوک‌شاخ	۲ ۳ ۴
بزکوهی	۳ ۴
شیر پلید	۲ ۳ ۴
کفتار	۲ ۳ ۴
دم‌عصایی	۱ ۲ ۳ ۴
گراز	۱ ۲ ۳ ۴

- ب)

مرحله	متغیر	مقادیر حذف شده همسایه	بک‌ترک
۱	شیر = ۱	–	–
۲	نوک‌شاخ = ۲	شیرپلید $\neq$ ۲ و کفتار $\neq$ ۲	–
۳	شیر پلید	بزکوهی $\neq$ ۳ و ۴	✓
۴	نوک‌شاخ = ۳	شیرپلید $\neq$ ۳ و کفتار $\neq$ ۳	–

مرحله	متغیر	مقادیر حذف شده همسایه	بک‌ترک
۵	شیر پلید	بزکوهی ≠ ۳	–
۶	کفتار	–	–
۷	بزکوهی	شیر پلید ≠ ۴ و کفتار ≠ ۴	–
۸	شیر پلید	دم‌عضایی ≠ ۲ و گراز ≠ ۲	–
۹	کفتار	–	–
۱۰	گراز	–	–
۱۱	دم‌عصایی	–	–
۱۲	بزکوهی = ۴	–	–
۱۳	شیر پلید = ۲	–	–
۱۴	کفتار = ۲	–	–
۱۵	دم‌عصایی = ۱	گراز ≠ ۳ و ۴	–
۱۶	گراز	–	–
۱۷	گراز = ۱	–	–



شکل ۲: سطح اول: شیر، سطح دوم: نوک شاخ، سطح سوم: بزکوهی، سطح چهارم: شیر پلید، سطح پنجم: کفتار، سطح ششم: دم‌عصایی و سطح هفتم: گراز

سؤال ۳. CSP

- الف)

۱. متغیرها: { کلاس ۱، کلاس ۲، کلاس ۳، کلاس ۴، کلاس ۵ }

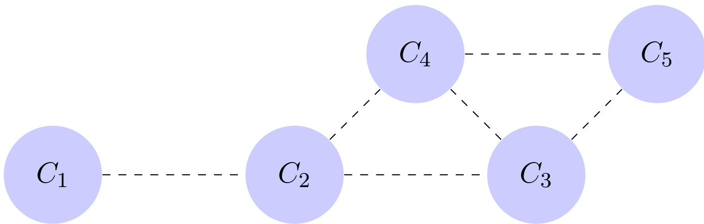
۲. دامنه‌ها:

$C_1 \in \{C\}$
$C_2 \in \{B, C\}$
$C_3 \in \{A, B, C\}$
$C_4 \in \{A, B, C\}$
$C_5 \in \{B, C\}$

۳. محدودیت‌های دوگانه:

$C_1 \neq C_2,\ C_2 \neq C_3,\ C_2 \neq C_4, ,\ C_3 \neq C_4,\ C_4 \neq C_5,\ C_3 \neq C_5$

۴. گراف محدودیت‌ها:



شکل ۳: نقطه‌چین بیان‌گر این است که نباید به‌طور هم‌زمان انجام شوند.

- ب) با توجه به این که برای کلاس ۱ فقط پرفسور C (محدودیت یگانه) وجود دارد، با اعمال آن همه‌ی کلاس‌های دیگر به محدودیت یگانه تبدیل می‌شوند و جواب برابر حالت زیر است:

$C_1 = Professor\ (C)$
$C_2 = Professor\ (B)$
$C_3 = Professor\ (A)$
$C_4 = Professor\ (C)$
$C_5 = Professor\ (B)$

سؤال ۴. Consistency

- الف) **توضیح الگوریتم:** یک متغیر در مسئله CSP، Arc Consistent است، اگر برای هر مقدار درون دامنه‌اش، همه‌ی محدودیت‌های دوگانه را ارضا کند. معروف‌ترین الگوریتم برای Arc Consistency الگوریتم AC-3 است. برای این که همه‌ی متغیرها را Arc Consistent کند، این الگوریتم یک مجموعه از arcها را در نظر می‌گیرد [۲]. ابتدا این مجموعه شامل همه‌ی arcهای مسئله CSP است. سپس الگوریتم یک arc مانند $(X_i,\ X_j)$ به‌طور دل‌خواه pop می‌کند و X_i را با توجه به X_j، consistent می‌کند. اگر در این مرحله دامنه‌ی D_i بدون تغییر بماند، الگوریتم سراغ arc بعدی می‌رود. در غیر این صورت، همه‌ی arcهای $(X_k,\ X_i)$ را اضافه می‌کند (که X_k همسایه‌ی X_i است). حتی اگر X_k را قبلا در نظر گرفته‌ایم، باید این کار را انجام دهیم، زیرا ممکن است تغییرات در دامنه‌ی D_i باعث تغییر (کاهش) در دامنه‌ی D_k شود. اگر دامنه‌ي D_i تهی شود، به این معنی است که جواب consistent برای این مسئله وجود ندارد و الگوریتم خطا برمی‌گرداند. در غیر این صورت، این کار را آن‌قدر ادامه می‌دهیم تا به حالتی برسیم که هیچ arcی در مجموعه وجود نداشته باشد و به جوابی معادل با جواب CSP اصلی با سرعت بیش‌تر. می‌رسیم، زیرا دامنه‌ی متغیرها کوچک‌تر می‌شود.

```
function AC-3(csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
   removed ← false
   for each x in DOMAIN[Xi] do
      if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
         then delete x from DOMAIN[Xi];  removed ← true
   return removed
```

شکل ۴: شبه کد الگوریتم AC-3

بهبود الگوریتم: الگوریتم AC-3 هر یال $(X_k,\ X_i)$ را هر موقع که مقداری از دامنه‌ی X_i حذف می‌شود را در مجموعه قرار می‌دهد. حتی اگر هر مقدار X_k با چندین مقدار باقی‌مانده‌ی مربوط به X_i consistent باشد. فرض کنید برای هر یال $(X_k,\ X_i)$، ما تعداد مقادیر باقی‌مانده‌ی مربوط به X_i که با هر مقدار X_k سازگار است را نگهداری می‌کنیم. ایده‌ی اساسی این است که محدودیت‌ها را پیش‌پردازش کنیم تا برای هر مقدار X_i مقادیری از X_k را نگهداری کنیم که یک یال از X_k به X_i مقداری خاصی از X_i را ارضا کند.این داده ساختار در زمان متناسب با اندازه مسئله محاسبه شود. سپس، پس از این‌که مقدار X_i حذف شود، تعداد مقادیر مجاز برای هر یال $(X_k,\ X_i)$ که در آن ذخیره شده است را یکی کاهش می‌دهیم. همان‌طور که مشخص است و توضیح داده شد، زمان اجرای این الگوریتم $O(n^2d^2)$ است.

- ب) خیر. فرض کنید یک مسئله‌ی CSP داریم که به شرح زیر است:

$Variables = \{X, Y\}$
$D_X = D_Y = \{1,\ 2,\ 3,\ 4\}$

محدودیت‌های آن هم $X < 4$ و $X = Y$ هستند. اگر چه این مسئله 2-consistent است اما 1-consistent نیست. زیرا به ازای $X = 4$ شرط (محدودیت) اول نقض می‌شود.

- ج) همان‌طور که در اسلاید شماره ۳۱ جلسه ۷ و ۸ آمده، اثبات می‌کنیم که این مسئله جواب دارد.

[۲] در مجموعه ترتیب مهم نیست

اثبات: اگر یک مسئله CSP، Strong n-consistent باشد، به این معنی است که

$$(n-1)-consistent,\ (n-2)-consistent,\ ...,\ 1-consistent$$

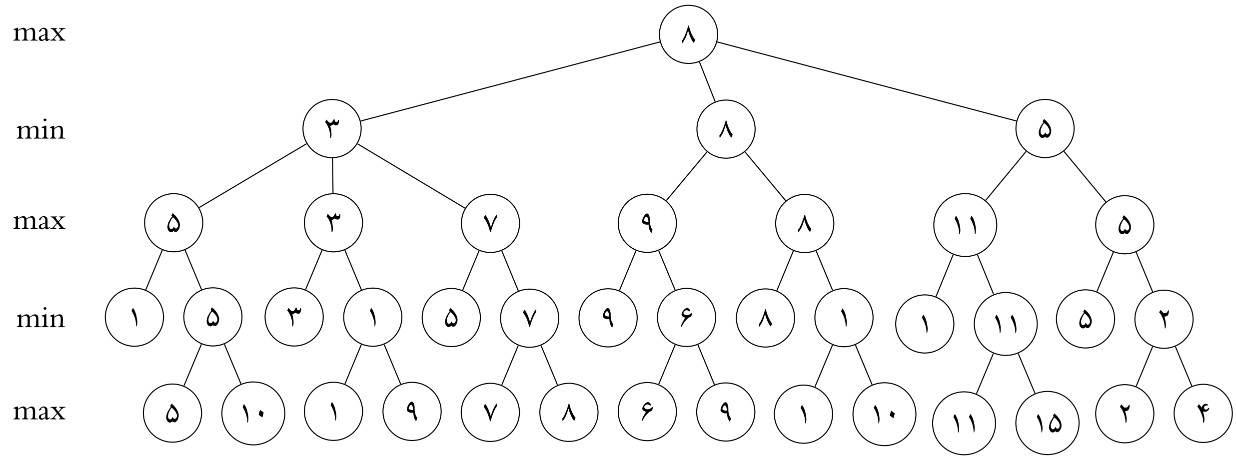
است. حال می‌توانیم مسئله را به شکل زیر حل کنیم:

۱. یک مقدار consistent برای X_1 انتخاب می‌کنیم.
۲. چون 2-consistent نیز هست، تضمین می‌شود که مقدار consistent برای X_2 نیز پیدا می‌شود.
۳. برای X_3 هم همانند قسمت (۲) چون 3-consistent است، تضمین می‌شود که مقدار قابل قبول برای آن پیدا می‌شود.
۴. همین‌طور مراحل ادامه می‌یابد. برای هر متغیر X_i تنها باید داخل دامنه‌ی با سایز d آن به دنبال مقدار consistent با $X_1, ..., X_{i-1}$ بگردیم.

نکته: زمان اجرای این الگوریتم $O(n^2d)$ است.

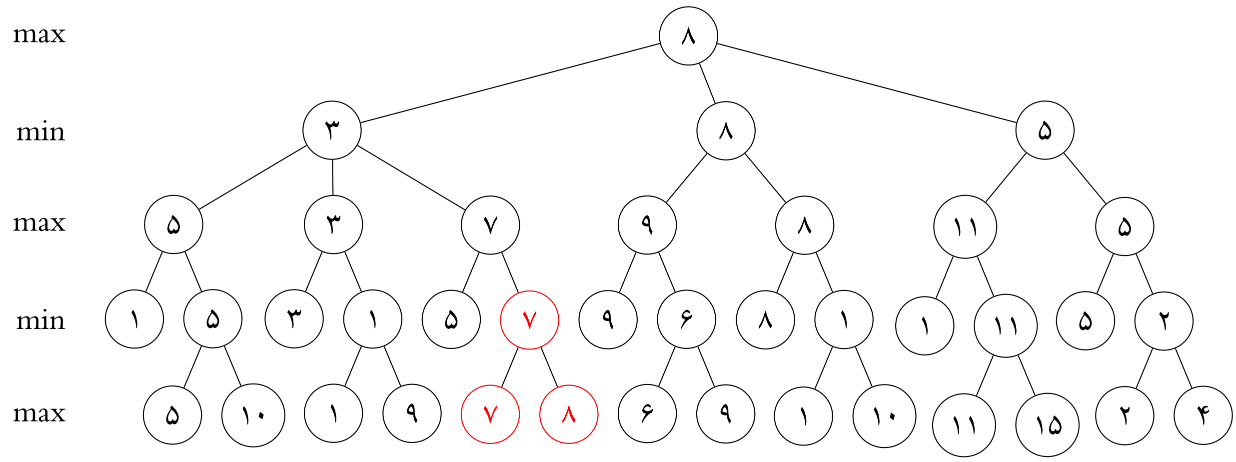
سؤال ۵. Minimax

- الف)



شکل ۵: مقدار ریشه برابر با هشت است.

- ب)



شکل ۶: گره‌هایی که رنگ آن‌ها قرمز است، حذف می‌شوند.

- ج) با توجه به اسلاید شماره ۲۷ جلسه ۹ و ۱۰ ، چون لایه‌ی یکی مانده به آخر min است،بنابراین باید گره[۳]ها به ترتیب صعودی مرتب شده باشند که همین اتفاق افتاده است.

---

[۳] node

سؤال ۶. Minimax

- الف) حد بالای تعداد گره‌های انتهایی درخت minimax برابر با $n!$ است. حد بالا برای تمامی گره‌ها برابر با $\Sigma_{i=1}^{n} i!$ است. این مقدار تفاوت خیلی زیادی (از لحاظ مرتبه‌ای) با $n!$ ندارد.

  حد بالا برای تعداد حالت‌های متمایز بازی برابر با 3^n است. چون هر کدام از خانه‌ها یا خالی هستند، یا X و یا O هستند.

- ب) در این حالت هیچ بازی‌ای سریع تمام نمی‌شود. اگر حالت‌های تکراری را در نظر نگیریم دقیقا $\Sigma_{i=1}^{n} i!$ حالت داریم.

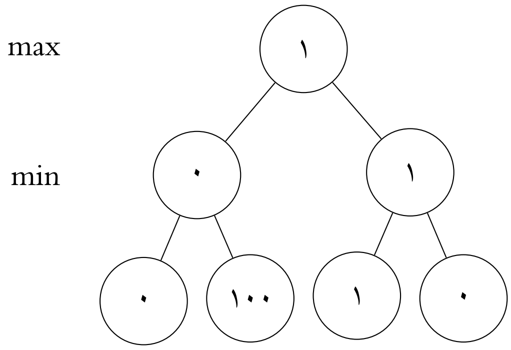
در آخر بازی خانه‌ها بین دو بازیکن تقسیم می‌شود. یعنی $\lceil n/2 \rceil$ خانه به بازیکن اول و $\lfloor n/2 \rfloor$ خانه به بازیکن دوم می‌رسد. بنابراین یک حد پایین خوب برای تعداد حالت‌های متمایز همان تعداد حالت‌های انتهایی متمایز است که برابر با $\binom{n}{\lceil n/2 \rceil}$ می‌باشد.

- ج) اگر برای حالت s تابع $X(s)$ را تعداد حالات بردی که در آن هیچ Oای وجود ندارد و تابع $O(s)$ را تعداد حالات بردی که در آن هیچ Xای وجود ندارد در نظر بگیریم، تابع زیر می‌تواند یک تابع ارزیابی برای آن به حساب بیاید:

$$evaluation - function(s) = X(s) - O(s)$$

سؤال ۷. Minimax vs Expectimax

- الف)



همان‌طور که می‌بینیم در درخت بالا اگر از الگوریتم minimax استفاده کنیم، جوابی برابر با ۱ خواهیم داشت. اما اگر از الگوریتم expectimax استفاده کنیم مقدار ۵۰ را می‌گیرد.

اگر بازیکن اول انتظار داشته باشد که بازیکن دوم به طور تصادفی حرکن کند، بهتر است از الگوریتم expectimax استفاده کند.

- ب) طبق تعریف حرکت بهینه کمینه به معنی حرکتی است که کوچک‌ترین مقدار ممکن را می‌گیرد. بازی تصادفی شامل حرکاتی است که لزوما بهینه نیستند. با فرض این که هیچ دو tie با یک‌دیگر برابر نیستند، می‌توان گفت الگوریتم expectimax یک الگوریتم زیربهینه[۴] است. میانگین‌گیری در برابر کمینه‌گیری خروجی را افزایش می‌دهد.

  با این حساب، می‌توانیم ثابت کنیم که درخت بازی‌ای وجود ندارد که در آن مقدار ریشه برای آن برای expectimax کوچک‌تر از minimax باشد. یکی از آن‌ها بازی بهینه است و دیگری بازی زیربهینه است.

  اگر بازیکن دوم به طور بهینه بازی کند بازیکن اول باید از الگوریتم minimax استفاده کند.

---

[۴] sub-optimal