سیستم‌های توزیع‌شده

تمرین سری سوم

استاد: دکتر کمندی

پارسا محمدپور

پاییز ۱۴۰۳

پرسش یک

مراحل اجرای الگوریتم FloodSet را برای چهار فرآیند و دو خطا دنبال کنید:

- مقادیر اولیه فرآیندها به ترتیب برابر 1، 0، 0 و 0 هستند.
- فرض کنید که فرآیندهای یک و دو معیوب هستند، به طوریکه فرآیند یک در دور اول پس از ارسال به فرآیند دو از کار می‌افتد و فرآیند دو در دور دوم پس از ارسال به فرآیندهای یک و سه، از کار می‌افتد.

ابتدا توضیح مختصری در رابطه با الگوریتم فلادست[1] می‌دهیم. این الگوریتم برای وقت‌هایی است که ما خطای توقف[2] داریم. در این الگوریتم، فرض می‌کنیم که تعداد f پراسس خطادار[3] داریم. این الگوریتم در f+1 مرحله اجرا می‌شود و پراسس‌ها در انتهای این مرحله تصمیم می‌گیرند. در این الگوریتم، هر پراسس دارای یک متغیر از جنس **مجموعه** به نام w است که در هر بار اجرای الگوریتم آن را بروزرسانی می‌کند و در انتها با توجه به مقدار آن تصمیم‌گیری انجام می‌شود. در این الگوریتم، در هر مرحله، هر پراسس این مقدار (مقدار متغیر w) را به تمام پراسس‌های دیگر ارسال می‌کند. مقدار اولیه موجود در این متغیر، در ابتدا برابر با یک مجموعه تک عضوی است که تنها مقدار آن، تصمیم اولیه خود پراسس است. نحوه بروزرسانی این متغیر برای هر پراسس به این صورت است که در این متغیر، اجتماع مقدار فعلی این متغیر با مقداری که از پراسس‌های دیگر دریافت کرده است را قرار می‌دهد. نحوه تصمیم‌گیری پراسس‌ها در انتهای مرحله f+1 ام به این صورت است که اگر مجموعه w برابر با مجموعه تک‌عضوی[4] باشد، مقدار تصمیم این پراسس، تنها عضو این مجموعه است؛ در غیر اینصورت، تصمیم پراسس برابر با مقدار تصمیم دیفالت[5] می‌باشد. طبق اسلایدها تعریف این الگوریتم به صورت زیر است:

***states*$_i$:**
rounds $\in \mathbb{N}$, initially 0
decision $\in V \cup \{unknown\}$, initially *unknown*
$W \subseteq V$, initially the singleton set consisting of i's initial value

***msgs*$_i$:**
if *rounds* $\leq f$ then send W to all other processes

***trans*$_i$:**
rounds := *rounds* + 1
let X_j be the message from j, for each j from which a message arrives
$W := W \cup \bigcup_j X_j$
if *rounds* $= f + 1$ then
  if $|W| = 1$ then *decision* := v, where $W = \{v\}$
  else *decision* := v_0

حالا طبق تعریف‌های صورت گرفته، به انجام الگوریتم برای حالت گفته شده می‌رویم. برای این کار، برای هر مرحله جدولی شامل مقدار متغیرهای هر پراسس را نشان می‌دهیم. همچنین می‌دانیم به علت اینکه دو تا پراسس خراب داریم، پس این الگوریتم در طی سه مرحله اجرا می‌شود و در پایان مرحله سه، تصمیم‌گیری انجام می‌شود. پس داریم:

- مقدار اولیه، یا همان مرحله صفرم، طبق الگوریتم به صورت زیر است:

پراسس / متغیر	یک	دو	سه	چهار
w	{1}	{0}	{0}	{0}
r	0	0	0	0
decision	unknown	unknown	unknown	unknown

چون در این مرحله، r مخالف با f+1 ،که تعداد کل مراحل است، می‌باشد، پسپراسس‌ها برای هم پیام ارسال می‌کنند تا در مرحله بعدی پیام‌ها پردازش شوند. این پیام‌ها به صورت زیر هستند:

---

[1] FloodSet algorithm
[2] stop failure
[3] faulty
[4] singleton
[5] default

پراسس ارسال کننده / پراسس دریافت کننده	یک	دو	سه	چهار
یک	-	Process fail	Process fail	Process fail
دو	{1}	-	{0}	{0}
سه	Process fail	{0}	-	{0}
چهار	Process fail	{0}	{0}	-

- مقدار متغیرها در پایان مرحله یک:

پراسس‌ها در این مرحله با توجه به مقادیر دریافتی، مقدار متغیرهایشان را بروزرسانی می‌کنند. در انتهای این مرحله (پس از پایان بروزرسانی متغیرها)، مقدار این متغیرها برابر است با:

پراسس / متغیر	یک	دو	سه	چهار
w	Failed	{0,1}	{0}	{0}
r	Failed	1	1	1
decision	Failed	unknown	unknown	unknown

چون در این مرحله، r مخالف با f+1 ،که تعداد کل مراحل است، می‌باشد، پسپراسس‌ها برای هم پیام ارسال می‌کنند تا در مرحله بعدی پیام‌ها پردازش شوند. این پیام‌ها به صورت زیر هستند:

پراسس ارسال کننده / پراسس دریافت کننده	یک	دو	سه	چهار
یک	-	Process failed	Process failed	Process failed
دو	Process failed	-	Process fail	Process fail
سه	Process failed	{0,1}	-	{0}
چهار	Process failed	Process fail	{0}	-

- مقدار متغیرها در پایان مرحله دو:

پراسس‌ها در این مرحله با توجه به مقدار دریافتی، متغیرهای خود را بروزرسانی می‌کنند. در انتهای این مرحله، در پایان مرحله دوم (پس از بروزرسانی متغیرها)، مقدار این متغیرها برابر است با:

پراسس / متغیر	یک	دو	سه	چهار
w	Failed	Failed	{0,1}	{0}
r	Failed	Failed	2	2
decision	Failed	Failed	unknown	unknown

چون در این مرحله، r مخالف با f+1 ،که تعداد کل مراحل است، می‌باشد، پسپراسس‌ها برای هم پیام ارسال می‌کنند تا در مرحله بعدی پیام‌ها پردازش شوند. این پیام‌ها به صورت زیر هستند:

پراسس ارسال کننده / پراسس دریافت کننده	یک	دو	سه	چهار
یک	-	Process failed	Process failed	Process failed
دو	Process failed	-	Process failed	Process failed
سه	Process failed	Process failed	-	{0}
چهار	Process failed	Process failed	{0,1}	-

- مقدار متغیرها در پایان مرحله سه (مرحله آخر و تصمیم‌گیری):
پراسس‌ها در این مرحله با توجه به مقدار دریافتی، متغیرهای خود را بروزرسانی می‌کنند. در انتهای این مرحله، در پایان مرحله سوم (پس از بروزرسانی متغیرها و انجام تصمیم‌گیری)، مقدار این متغیرها برابر است با:

پراسس / متغیر	یک	دو	سه	چهار
w	Failed	Failed	{0,1}	{0,1}
r	Failed	Failed	3	3
decision	Failed	Failed	0	0

پس در این حالت، پراسس‌های بدون خطا، پراسس‌های سه و چهار، به ترتیب تصمیم **صفر و صفر** را می‌گیرند. از سه تا شرطس که داشتیم، شرط بالاخره تصمیم‌گرفتن[۱]، یکسان بودن تصمیم‌ها و یا موافق بودن[۲] و شرط قابل قبول بودن[۳] **هر سه رعایت شده‌اند**. چون تصمیم همه پراسس‌هایی که تصمیم می‌گیرند، یکسان است پس موافق بودن همچنین چون تصمیم‌گیری شده است، پس شرط بالاخره تصمیم‌گیری رعایت شده است. همچنین چون مقدار تصمیم اولیه همه پراسس‌ها یکسان نیست، پس شرط قابل قبول بودن هم به انتفای مقدم برقرار است.

[۱] termination
[۲] agreement
[۳] validity

پرسش ۲

الگوریتم EIGByz را در نظر بگیرید. اجراهای صریح بسازید و نشان دهید که الگوریتم با شرایط زیر می‌تواند نتایج اشتباه بدهد:

الف) هفت گره، دو خطا و دو دور.

با توجه به روند این الگوریتم و نقل قول کردن پراسس‌ها برای همدیگر، اجرایی میسازیم که در آن با توجه به موارد گفته شده در صورت سوال، به تناقض بخوریم. روند اجرای این الگوریتم را مرحله به مرحله نشان می‌دهیم. در این الگوریتم فرض می‌شود که گراف ارتباطی[۱]، گرافی کاملی است. برای این کار باید ابتدا مقدار تصمیم اولیه هر پراسس را مشخص کنیم. برای این اجرا از کد، درنظر می‌گیریم که مقدار اولیه پراسس‌های یک و دو برابر با صفر، مقدار اولیه پراسس‌های سه و چهار هم برابر با صفر و مقدار اولیه بقیه پراسس‌ها برابر با یک باشد. همچنین پراسس‌های یک و دو دارای خطای بیزانسی[۲] هستند. پس داریم:

- انتهای مرحله صفر (مقداردهی اولیه):

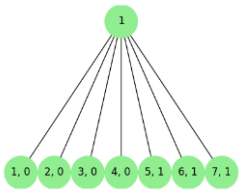
هر پراسس مقدار خود را دارد. و همچنین پیام‌هایی که برای دیگران ارسال می‌کنند، همه به جز پراسس های یک و دو (که خطا دار هستند)، بقیه مقدار خودشان را ارسال می‌کنند و پراسس‌های خطا دار، مقدار صفر را برای پراسس‌های سه و چهار و مقدار یک را برای پراسس‌های پنج و شش و هفت ارسال می‌کنند. پس خواهیم داشت:

ارسال کننده / دریافت کننده	یک	دو	سه	چهار	پنج	شش	هفت
یک	0	0	0	0	1	1	1
دو	0	0	0	0	1	1	1
سه	1	1	0	0	1	1	1
چهار	1	1	0	0	1	1	1
پنج	0	0	0	0	1	1	1
شش	0	0	0	0	1	1	1
هفت	0	0	0	0	1	1	1

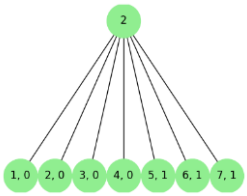
- انتهای مرحله یک:

در انتهای این مرحله پیام‌های دریافتی از جانب سایر پراسس‌ها پردازش می‌شوند و مقدار موجود در ساختمان داده[۳] را بروزرسانی می‌کنند. پس این ساختمان داده برای هر پراسس برابر است با:

- پراسس یک:

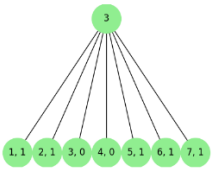


- پراسس دو:

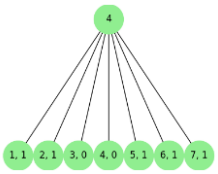


[۱] communication graph
[۲] byzantine failure
[۳] data structure

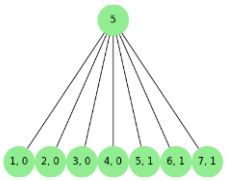
- پراسس سه:



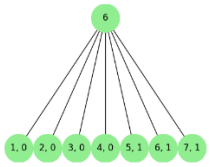
- پراسس چهار:



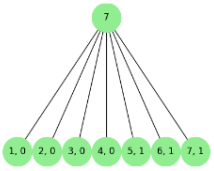
- پراسس پنج:



- پراسس شش:



- پراسس هفت:



(در انتهای این مرحله هم مشخص است که شرط موافق بودن نقض می‌شود در این مرحله اگر مرحله آخر بود. چون پراسس‌های سه و چهار تصمیم یک را می‌گرفتند و بقیه پراسس‌های غیرخطادار، تصمیم صفر را می‌گرفتند)
حالا در این مرحله، باز هم پراسس‌ها باید برای یکدیگر پیام ارسال کنند. پیام‌های ارسالی را به این شکل درنظر می‌گیریم:

ارسال کننده / دریافت کننده	یک	دو	سه	چهار	پنج	شش	هفت
یک	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)
دو	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)

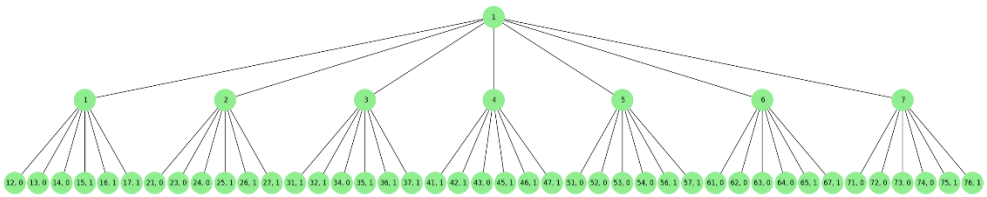
سه	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)
چهار	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)
پنج	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)
شش	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)
هفت	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,1), (2,1), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)	(1,0), (2,0), (3,0), (4,0), (5,1), (6,1), (7,1)

با توجه به این پیام‌ها، هر کدام از پراسس‌ها با دریافت پیام، ساختار داده‌ای خودشان را در مرحله بعدی بروزرسانی می‌کنند.

- انتهای مرحله دو (باید تصمیم‌گیری شود):

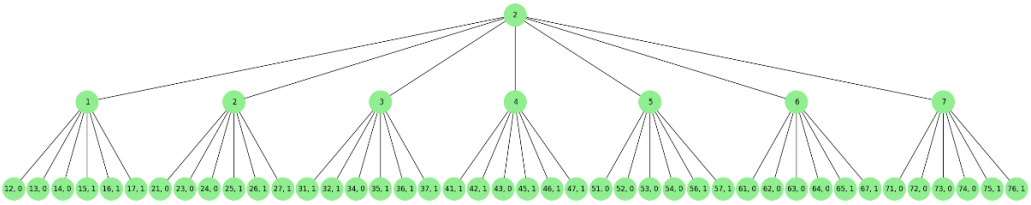
در انتهای این مرحله پیام‌های دریافتی از جانب سایر پراسس‌ها پردازش می‌شوند و مقدار موجود در ساختمان داده را بروزرسانی می‌کنند. پس این ساختمان داده برای هر پراسس برابر است با:

- پراسس یک:



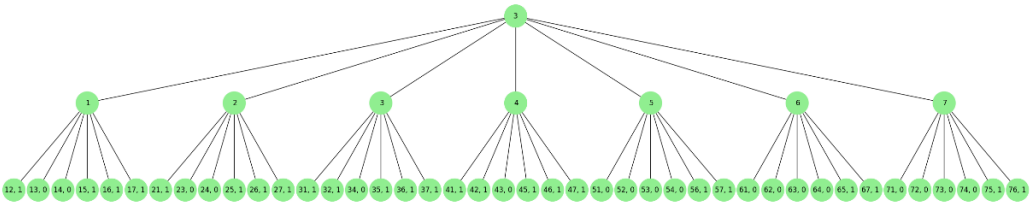
پس با توجه به شکل، تصمیم گره یک (باتوجه به نحوه تصمیم‌گیری) برابر است با صفر. (البته چون گره یک خطادار است، پس اصلا مهم نیست)

- پراسس دو:



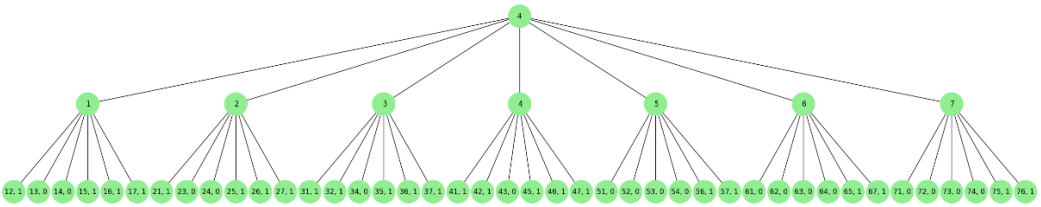
پس با توجه به شکل، تصمیم گره دو (باتوجه به نحوه تصمیم‌گیری) برابر است با صفر. (البته چون گره دو خطادار است، پس اصلا مهم نیست)

- پراسس سه:



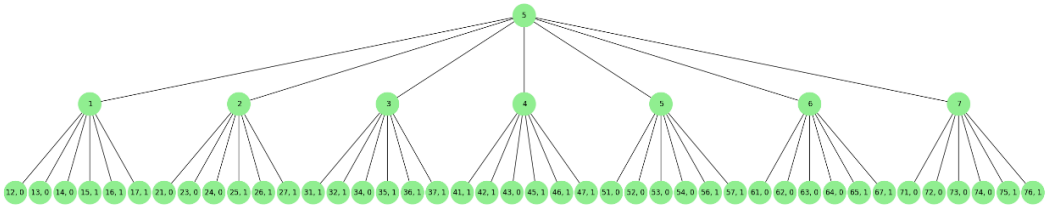
پس با توجه به شکل، تصمیم گره سه (باتوجه به نحوه تصمیم‌گیری) برابر است با یک.

- پراسس چهار:



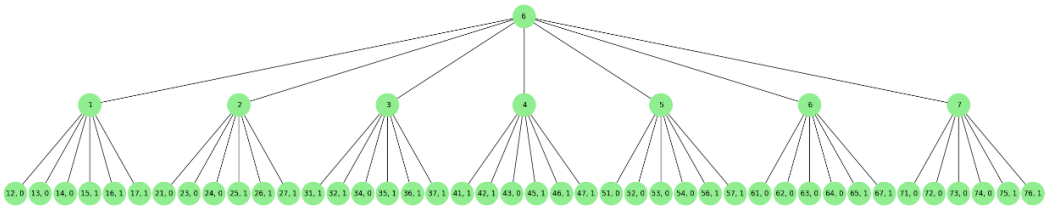
پس با توجه به شکل، تصمیم گره چهار (باتوجه به نحوه تصمیم‌گیری) برابر است با یک.

- پراسس پنج:



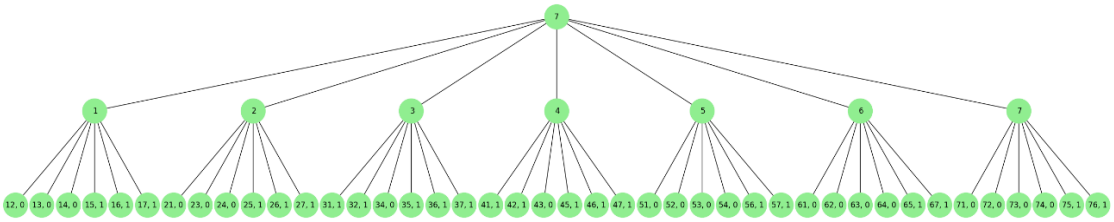
پس با توجه به شکل، تصمیم گره پنج (باتوجه به نحوه تصمیم‌گیری) برابر است با صفر.

- پراسس شش:



پس با توجه به شکل، تصمیم گره شش(باتوجه به نحوه تصمیم‌گیری) برابر است با صفر.

- پراسس هفت:



پس با توجه به شکل، تصمیم گره هفت (باتوجه به نحوه تصمیم‌گیری) برابر است با صفر.

پس تصمیم‌پراسس‌ها در انتهای این الگوریتم، به ترتیب 0، 0، 1، 1، 0، 0 و 0 است.

پس با توجه به تصمیم‌ها، **قانون موافق بودن، نقض شده**[1] است. چون پراسس‌های سه و چهار تصمیم یک گرفته اند؛ درحالیکه پراسس‌های پنج، شش و هفت تصمیم صفر گرفته اند. (پراسس‌های یک و دو هم خطادار هستند پس اصلا مهم نیستند؛ چون قانون موافق بودن برای پراسس‌هایی تعریف می‌شود که خطا نداشته باشند)

[1] violation in agreement requirement

ب) شش گره، دو خطا و سه دور.

برای اینکه در این حالت به نقض یکی از شرایط (اتمام[۱]، توافق[۲] و قابل قبول بودن[۳]) برسیم، سناریو را بدین شکل طراحی می‌کنیم که همه پراسس‌ها (حتی پراسس‌های خطادار) با مقدار یک شروع کنند، سپس هر سری پراسس‌های خطادار، مقدار خلاف مقدار واقعی را برای بقیه ارسال کنند. در این حالت با توجه اینکه طبق شرط قابل قبول بودن، باید همه پراسس‌های سالم، به تصمیم یک برسند، اما در انتها همه به تصمیم صفر می‌رسند و شرط قابل قبول بودن از بین می‌رود. دلیل شهودی آن هم برای این است که زیردرخت‌ها پراسس‌های سالم، چون در مرحله سوم، دارای چهار تا برگ هستند که دو تا از آنها تحت تاثیر پراسس‌های خطادار هستند، پس مقدار تصمیم‌شان صفر می‌شود. زیر درخت‌های پراسس‌های خطادار طبیعتا چون از پراسس‌های خراب تاثیر گرفته‌اند، پس مقدار تصمیم‌شان صفر است. بنابراین تصمیم پراسس‌های سالم هم صفر می‌شود.

شکل‌های زیر حاصل پیاده‌سازی کد قسمت بعدی در محیط پای‌چارم[۴] هستند که درخت تصمیم نهایی هر پراسس را در مرحله نهایی نشان می‌دهد. درخت پراسس‌ها را به دلیل زیاد بودن حجمشان نیاوردیم وگرنه آن‌ها هم با تغییر مقدار یک متغییر نشان داده می‌شوند. پراسس‌ههای خطادار هم پراسس‌های سه و چهار هستند.

- درخت پراسس یک بعد از تصمیم‌گیری:

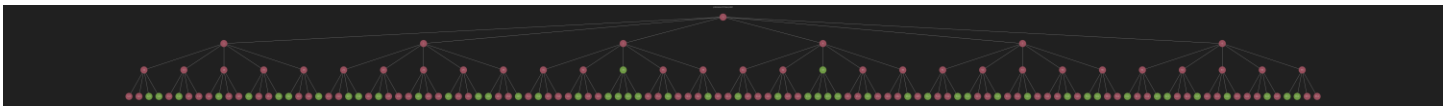
- درخت پراسس دو بعد از تصمیم‌گیری:

- درخت پراسس سه بعد از تصمیم‌گیری:

- درخت پراسس چهار بعد از تصمیم‌گیری:

- درخت پراسس پنج بعد از تصمیم‌گیری:

- درخت پراسس شش بعد از تصمیم‌گیری:



---

[۱] termination
[۲] agreement
[۳] validity
[۴] pycharm

پرسش ۳

الف) الگوریتم EIGByz را پیاده‌سازی کنید.

کدهای پیاده‌سازی این قسمت در پوشه کد[۱] قرار دارند. همچنین نتیجه شبیه‌سازی هم در فایل نوت‌بوک قرار دارد. اما عکس‌های خروجی به دلیل حجم زیاد به صورت فایل عادی قرارداده نشده‌اند.

ب) با استفاده از امضای دیجیتال و کد پیاده‌سازی شده در قسمت الف، الگوریتم را بهبود دهید. فرضیات درنظر گرفته شده و بهبودهای حاصل شده را بیان کرده و کد پیاده‌سازی شده هر دو قسمت را بارگذاری کنید.

در این حالت، ما با استفاده از امضای دیجیتال آر.اس.ای[۲] الگوریتم را پیاده‌سازی کردیم. البته برای این کار، کلید عمومی و خصوصی این را جابجا استفاده کردیم. یعنی کلید عمومی را فقط برای خود پراسس نگه داشتیم و کلید خصوصی را به همه دادیم. (صرفا از لحاظ مفهومی جابجا شده‌اند ئگرنه باز هم یکی خصوصی است و یکی عمومی) سپس در هر پیام، علاوه بر مقدار تصمیم، مقدار امضا شده توسط هر پراسس هم قرار دارد؛ به طوریکه اگر پراسس یک دارد از پراسس دو نقل قول می‌کند (پیام شنیده شده از جانب خودش را برای پراسس دیگری میفرستد) باید مقدار امضا شده توسط پراسس فرستنده اول را امضا کند. یعنی اگر پراسس یک می‌خواهد مقداری که از پراسس دو شنیده است را برای پراسس سه بفرستد، باید پیامی که در مرحله قبلی پراسس دو امضا کرده‌بود و برایش فرستاده بود را امضا کند و به همراه مقداری که ادعا می‌کند، برای پراسس سه بفرستد. بدین شکل هر پراسس هنگاه دریافت پیام، می‌تواند از روی امضای دیجیتال و مقدار فرستاده شده، تایید کند ک آیا آن پیام معتبر است یا خیر. اگر پیام غیرمرتبط دریافت کند، آن را حذف می‌کند و لحاظ نمی‌کند و در درختی که هر پراسس آن را بسط می‌دهد، مقدار تاییدیه[۳] آن گره از درخت را برابر با صفر می‌گذارد و در تصمیم گیری نهایی آن را لحاظ نمی‌کند.

با توجه به تغییرات بوجود آمده در این الگوریتم، صرفا دو مرحله برای تصمیم‌گیری کافی است. همچنین دیگر محدودیتی در تعداد پراسس‌ها خطادار نداریم؛ صرفا یک پراسس سالم وجود داشته باشد، کافیست. همچنین در این حالت در تعداد مرحله بزرگتر از دو، هیچ کدام از خواسته‌ها[۴]، نقض نمی‌شوند.

کدهای این قسمت هم در پوشه کدها قرار دارد. همچنین نتایج هم در فایل نوت‌بوک قرار دارند. اما فایل عکس‌های خروجی به دلیل حجم زیاد قرار داده نشدند.

برای نمایش درخت‌ها، از یک کد که برای پیاده‌سازی مکان قرارگیری نقاط درخت است، استفاده شده است که آن کد از اینترنت کپی شده است. این کد در پوشه اکسترنال[۵] قرار گرفته است.

---

[۱] code
[۲] RSA
[۳] is_authed
[۴] requirements
[۵] external