Computer Simulation Techniques: The definitive introduction!

Harry Perros

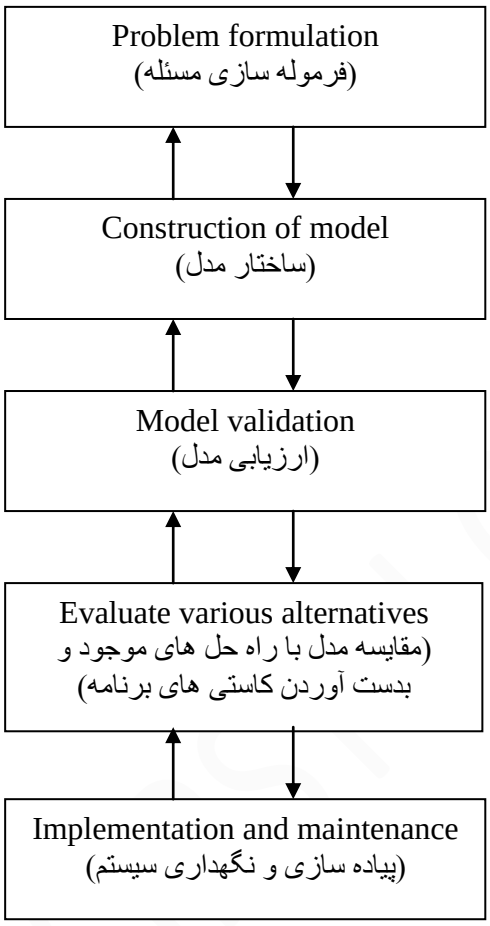
مترجم : جلیل قویدل

گردآورنده : فاطمه علی محمّدبیگی

فصل اول :

مقدمه (Introduction)

برای اینکه یک مساله را در مهندسی نرم افزار حل کنیم باید یک سری مراحل طی کنیم:



مهم ترین مسئله ای که در پیاده سازی برنامه است، ساخت مدل از یک سیستم است. یک مدل نمایشی از یک سیستم واقعی در جهان واقعیت است. در حالت کلی مدلها به سه دسته تقسیم می شوند:

1. **Iconic** : دقیقاً مشخصات یک سیستم واقعی را کپی می کند، اما در مقیاس کوچکتر. مثلاً ساخت یک مدل کوچک از یک هواپیمای بزرگ با همان مشخصات یا ساخت یک نقشه از یک شهر، کشور و ...
2. **Analogue** : در شبیه سازی یک سیستم، شبیه به سیستم اصلی را به جای آن قرار می دهند. مثلاً به جای یک پمپ هیدرولیک از یک پمپ برقی استفاده می کنند.(جایگزین کردن یک سیستم با مشابه اصلی اش)

3. **Symbolic** : این مدل مشخصات یک سیستم زندگی واقعی را با استفاده از تعدادی نماد مثل نمادهای ریاضی و برنامه های کامپیوتری نشان می دهد. این مدل ها به دو دسته مدلهای قطعی (Deterministic) و مدلهای آماری یا احتمالی (Stochastic) تقسیم می شوند.

مدلهای Deterministic مدلهایی هستند که در آنها از عامل احتمال استفاده نمی شود.

مدلهای Stochastic مدلهایی هستند که احتمال در آنها نقش اساسی را ایفا می کند.

از جمله مدلهای Deterministic می توان به روش های حل مسئله خطی، روش های غیرخطی و روشهای برنامه ریزی پویا اشاره کرد.

نکته : منظور از برنامه ریزی، روش های معمول کدنویسی نیست بلکه روشهای ریاضی است.

نمونه هایی از مدلهای Stochastic تئوری صف، پروسس های آماری و تکنیک های شبیه سازی هستند. تکنیک های شبیه سازی تا حد زیادی از تصادف استفاده می کنند، قابلیت یادگیری خوبی دارند، به سادگی قابل پیاده سازی هستند و در دامنه وسیعی از مسائل کاربرد دارند.

When every thing fails then simulate:

زمانی که شما کاری را انجام می دهید و fail می شوید، از شبیه سازی استفاده می کنید و وقتی به یک مدل همه چی تمام رسیدید آن را پیاده سازی می کنید.

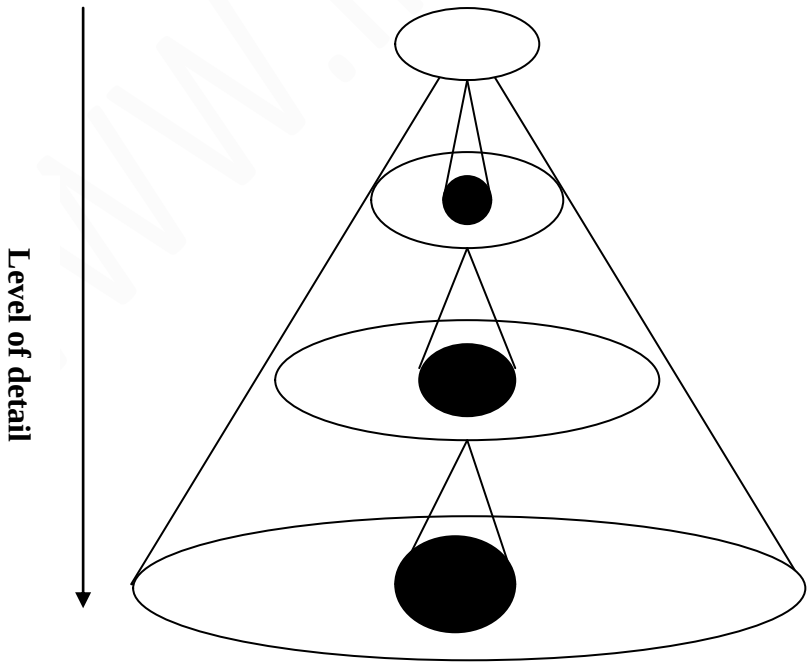
هر سیستم واقعی که با استفاده از تکنیک های شبیه سازی بررسی می شود به عنوان یک سیستم به آن نگاه می شود. یک سیستم مجموعه ای از موجودیت ها است که به صورت منطقی به مربوط هستند و تمام آن موجودیت ها به نحوی خاصیتی دارند که به کار ما مربوط می شود.

مثلاً در پیاده سازی سیستم فقط به sub-system های اصلی آن که با آن در ارتباط هستند توجه می کنیم و بقیه زیرسیستم ها را اجازه می دهیم که به صورت کلی باقی بمانند.

هر سیستم یک سری ویژگی هایی دارد:

1. **Environment** : هر سیستمی که شما در جهان واقعی مشاهده می کنید جزئی از یک سیستم بزرگ تر است که این سیستم بزرگ تر را محیط آن سیستم گویند. مثلاً قسمت آموزش جزئی از دانشگاه است، پس دانشگاه محیط سیستم آموزش است.
2. **Interdependency** : وابستگی اجزا با یکدیگر. هر سیستمی که شما در نظر می گیرید با سیستم های دیگر در ارتباط است.
3. **Sub-systems** : هر سیستم از تعدادی زیرسیستم تشکیل شده است.
4. **Organization** : یک نظم خاصی در سیستم برقرار است. در حالت کلی هر سیستم از اجزائی تشکیل شده است که بسیار منظم هستند و با همدیگر در تعامل هستند تا بتوانند هدف سیستم را برآورده سازند.
5. **Change** : حالت فعلی یک سیستم به طور معمول در طی دوره های طولانی زمانی تغییر می کند.

هرم مدیریت Beard :



کل جزئیات سیستم در اختیار گرفته نمی شود، بلکه فقط آن قسمت از جزئیات که به آن احتیاج داریم استفاده می کنیم.

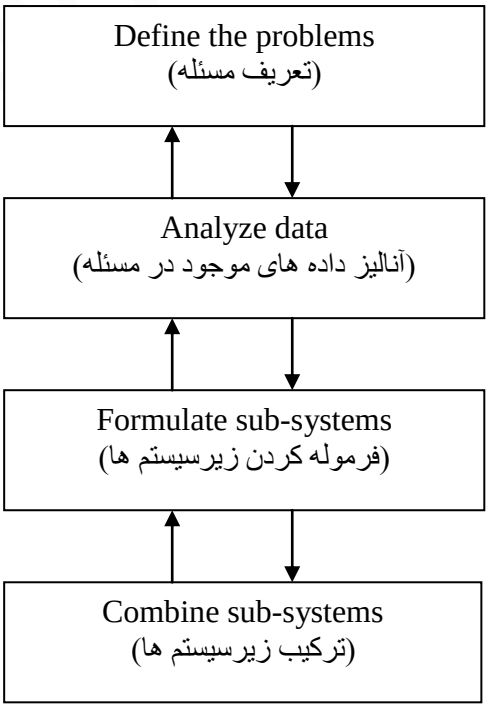
- ✓ هرچه به سمت پایین بیاییم جزئیات بیشتر می شود.
- ✓ معمولاً شبیه سازی برای سیستم هایی به کار برده می شود که فعلاً موجود نیستند.

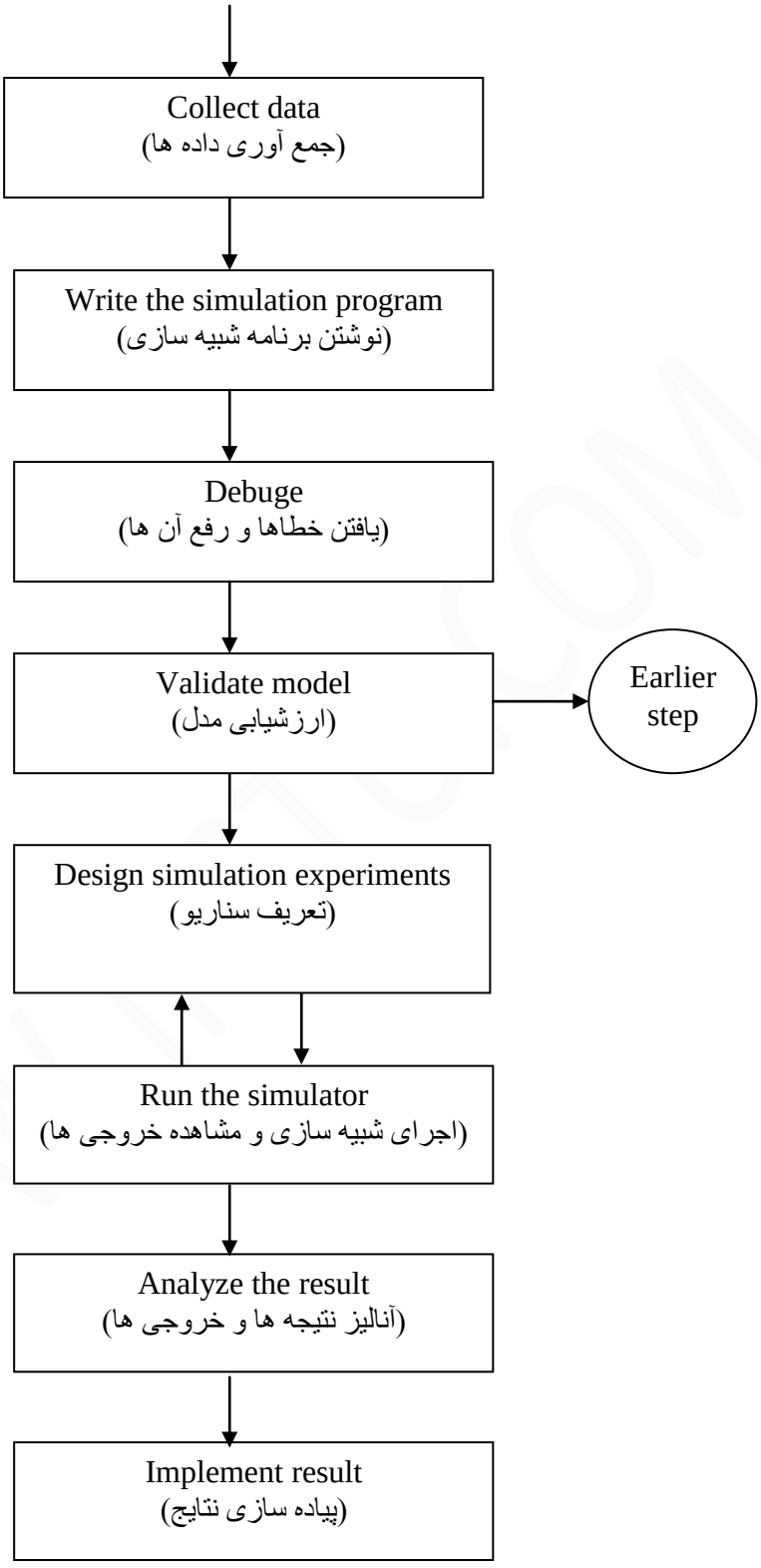
متغیرهای موجود در سیستم به دودسته غیرقابل کنترل (Uncontrollable) و قابل کنترل (Controllable) تقسیم می شوند:

1. **متغیرهای غیرقابل کنترل** : متغیرهایی هستند که به عنوان ورودی سیستم در نظر گرفته می شوند و قابل دست کاری نیستند.
2. **متغیرهای قابل کنترل** : متغیرهایی هستند که قابل دست کاری هستند و باید برای آنها مقدار بهینه یافت شود.

تفاوت بین متغیرهای قابل کنترل و غیرقابل کنترل و تمایز بین آنها بستگی به محدوده مسئله مورد مطالعه دارد.

برای مطالعه یک سیستم برای شبیه سازی یک سری مراحل باید طی شود:




Collect data
(جمع آوری داده ها)
Write the simulation program
(نوشتن برنامه شبیه سازی)
Debuge
(یافتن خطاها و رفع آن ها)
Validate model
(ارزشیابی مدل)
Earlier step
Design simulation experiments
(تعریف سناریو)
Run the simulator
(اجرای شبیه سازی و مشاهده خروجی ها)
Analyze the result
(آنالیز نتیجه ها و خروجی ها)
Implement result
(پیاده سازی نتایج)

یکی دیگر از مشخصه های متغیرهای مرتبط با شبیه سازی این است که آیا آنها در طی شبیه سازی تغییری می کنند یا نه؟ براین اساس متغیرها به دو دسته تقسیم می شوند:

1. Exogenous
2. Exdogenous

متغیرهایی که در طی فرآیند شبیه سازی تحت تاثیر هیچ عاملی قرار نمی گیرند Exogenous نامیده می شوند. اگر متغیری از شبیه سازی در طی شبیه سازی مقدارش با توجه به متغیرهای دیگر تعیین شود آن متغیر Exdogenous نامیده می شود.

به عنوان مثال در یک صف که دارای یک سرویس دهنده (Server) است، متغیرهای اشاره شده به این صورت می توانند دسته بندی شوند:

1. Exogenous variables :

1.1. The time interval between two arrivals.

1.1. فاصله زمانی بین دو مشتری

1.2. The service time of a costumer.

1.2. زمانی که طول می کشد به مشتری یک سرویس ارائه شود و از سرویس خارج شود.

1.3. Number of server

1.3. تعداد سرورهایی که در طول برنامه تغییر نمی کنند.

1.4. Priority discipline

1.4. ترتیب اولویت ها(آشنا بودن با Server)

2. Exdogenous variables

2.1. Mean waiting time in queue

2.1. مدت زمانی که شما در صف می مانید.

2.2. Mean number of customers in the queue

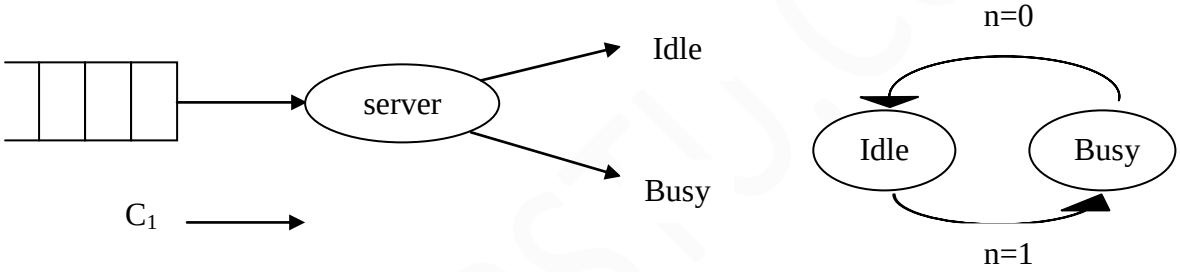
2.2. تعداد افراد در صف.

زمان سرویس دهی:

1. سرعت سرویس دهنده

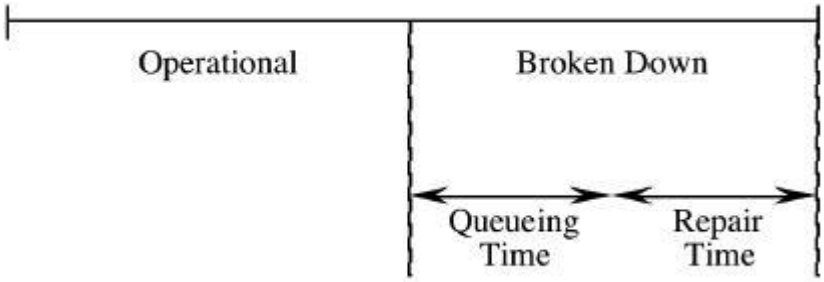
2. اولویت سرویس

متغیرهای فوق بسته به آزمایشی که قصد انجام آن را داریم می توانند قابل کنترل و یا غیرقابل کنترل باشند. به عنوان مثال اگر بخواهیم تاثیر تعداد سرویس دهنده ها را به میانگین زمان انتظار در صف بررسی کنیم، در این صورت تعداد سرویس دهنده ها یک متغیر قابل کنترل خواهد بود. متغیرهای باقی مانده مانند زمان بین رسیدن دو مشتری و یا زمانی که طول می کشد سرویس ارائه شود، غیرقابل کنترل باقی خواهد ماند.

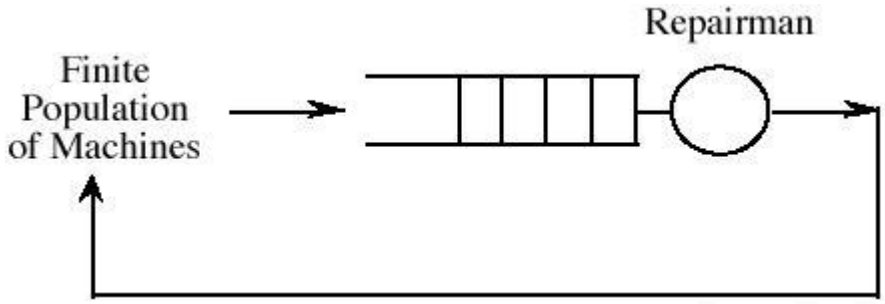


تعدادی از متغیرها در یک سیستم اهمیت بالایی دارند که حالت سیستم را مشخص می کنند که چنین متغیرهایی، متغیرهای حالت نامیده می شوند. این متغیرها ستون فقرات مدل شبیه سازی هستند. در هر لحظه در طی دوره شبیه سازی با استفاده از متغیرهای حالت می توان حالت سیستم را تعیین کرد.

مثال- کارخانه ای را در نظر بگیرید که تعداد n ماشین در حال کار هستند. هر ماشینی برای مدت زمان مشخصی کار کرده و بعد از آن خراب می شود. در این کارخانه تنها یک تعمیرکار وجود دارد. ماشین هایی که خراب می شوند در صف قرار می گیرند تا تعمیرکار آنها را تعمیر کند. ماشین ها به صورت FIFO تعمیر می شوند و هیچ اولویتی در صف وجود ندارد.



در حالت کلی برای محاسبه زمانی که طول می کشد تا ماشین بتواند دوباره کار کند باید زمانی را که تعمیرکار روی ماشین صرف می کند و همچنین زمانی که ماشین در صف قرار دارد مشخص باشد، با داشتن این پارامترها می توان کارایی صف را بدست آورد.



برای سادگی کار فرض می کنیم که زمان کارکرد هر یک از ماشین ها برابر با 10واحد زمانی است. همچنین زمان تغییر هر یک از ماشین ها برابر با 5 واحد زمانی است. به عبارت دیگر زمان کارکرد ماشین ها همه با هم برابر است.

اولین و مهم ترین کار در شبیه سازی و ساختن یک مدل شبیه سازی از یک سیستم، تشخیص رخدادهایی است که اتفاق افتادن آنها حالت سیستم را تغییر می دهد. برای این کار باید متغیرهای حالت سیستم را تعیین کنیم. انتخاب متغیرهای حالت بستگی به نوع معیارهای کارایی دارد که ما قصد بدست آوردن آنها را در سیستم مورد مطالعه داریم.

$n = 0$
$n = 1$
$n \geq 1$

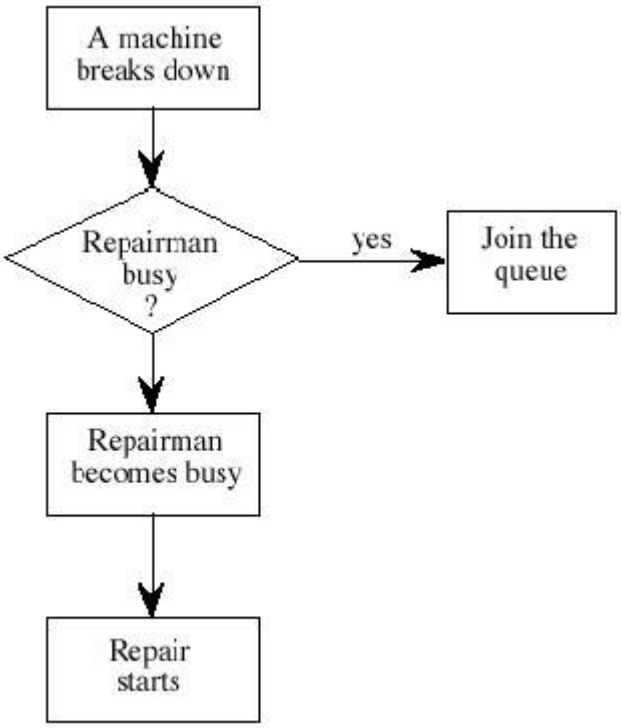
$1 \rightarrow$ در حال تعمیر
$n-1 \rightarrow$ در حال انتظار

n = number of customers in queue

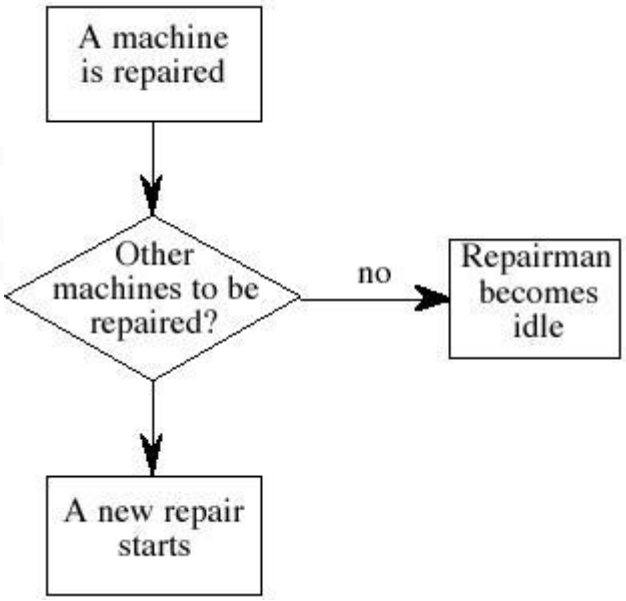
رخدادهایی که n را تغییر می دهند عبارت اند از:

1. خراب شدن یکی از ماشین ها
2. تعمیر شدن یکی از ماشین ها

Flowchart خراب شدن ماشین:



Flowchart تعمیر شدن ماشین:



به منظور اینکه این دو رخداد اصلی در مدل شبیه سازی با هم تعامل داشته باشند، نیاز به مجموعه ای از متغیرها داریم که با نام Clocks (ساعت ها) شناخته می شوند که تاریخچه زمان هایی را نشان خواهند

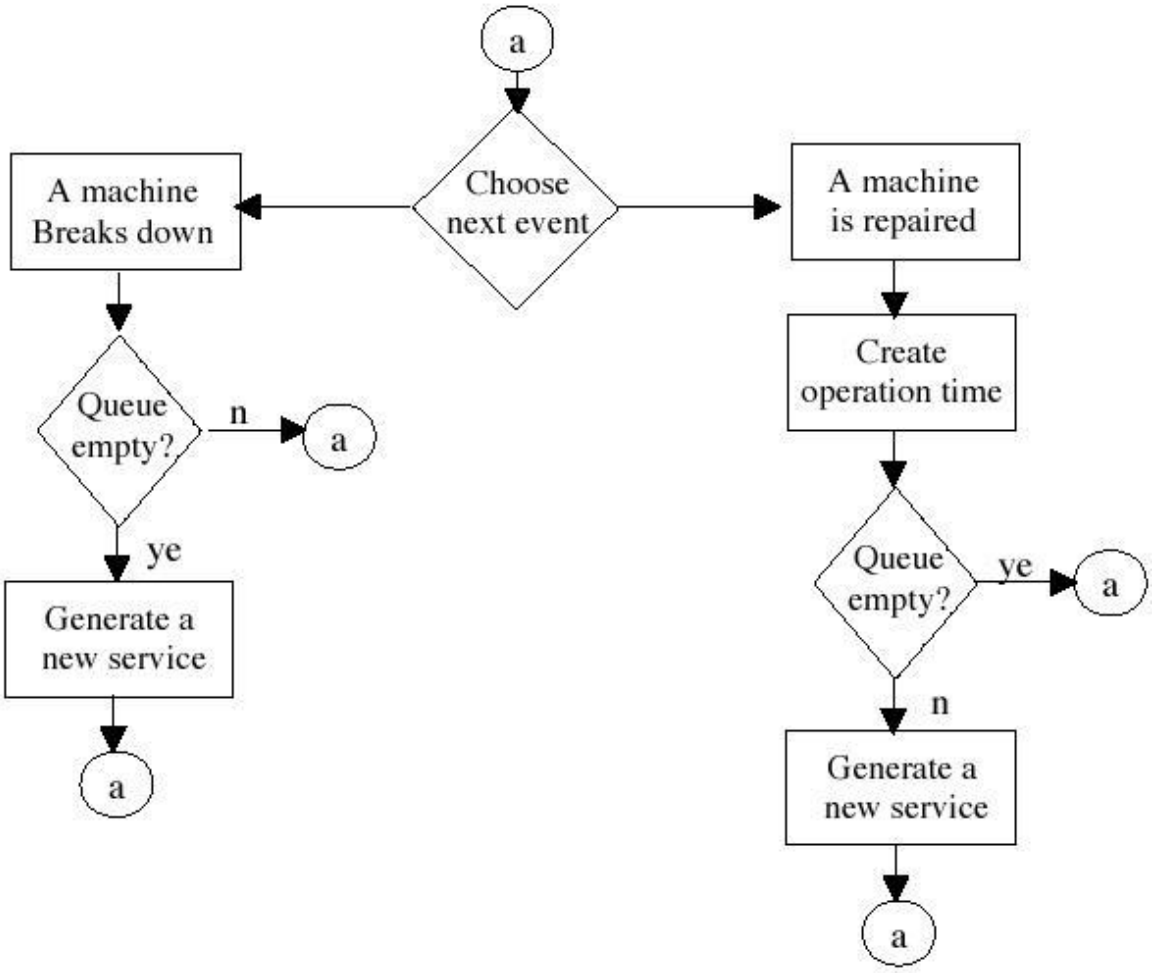
داد که در آنها ماشینی خراب می شود و یا تعمیر می شود. در مورد این مثال خاص نیاز خواهیم داشت که برای هر یک از ماشین ها یک Clock در نظر بگیریم. این ساعت زمانی را نشان خواهد داد که ماشین خراب خواهد شد.

علاوه بر این ساعت ها (Clocks) ساعت دیگری نیاز داریم تا زمانی را نشان دهد که در آن ماشینی که در حال تعمیر است، تعمیرش تمام می شود. پس تا الآن M+1 ساعت خواهیم داشت. هر یک از این ساعت ها زمان رخداد یکی از دو رخداد نشان داده شده را نشان می دهد.

M ساعت اول رخداد خراب شدن ماشین ها و ساعت بعدی زمان تعمیر شدن ماشین در حال تعمیر را نشان می دهد. علاوه بر این ساعت ها یک ساعت کلی یا Master clock خواهیم داشت که زمانی را نشان می دهد که از شبیه سازی سپری شده است.

با استفاده از این ساعت ها، مدل تصمیم می گیرد که کدام یک از اتفاق ها، اتفاق بعدی خواهد بود. پس از یافتن اتفاق بعدی Master clock تا آن اتفاق جلو کشیده می شود.

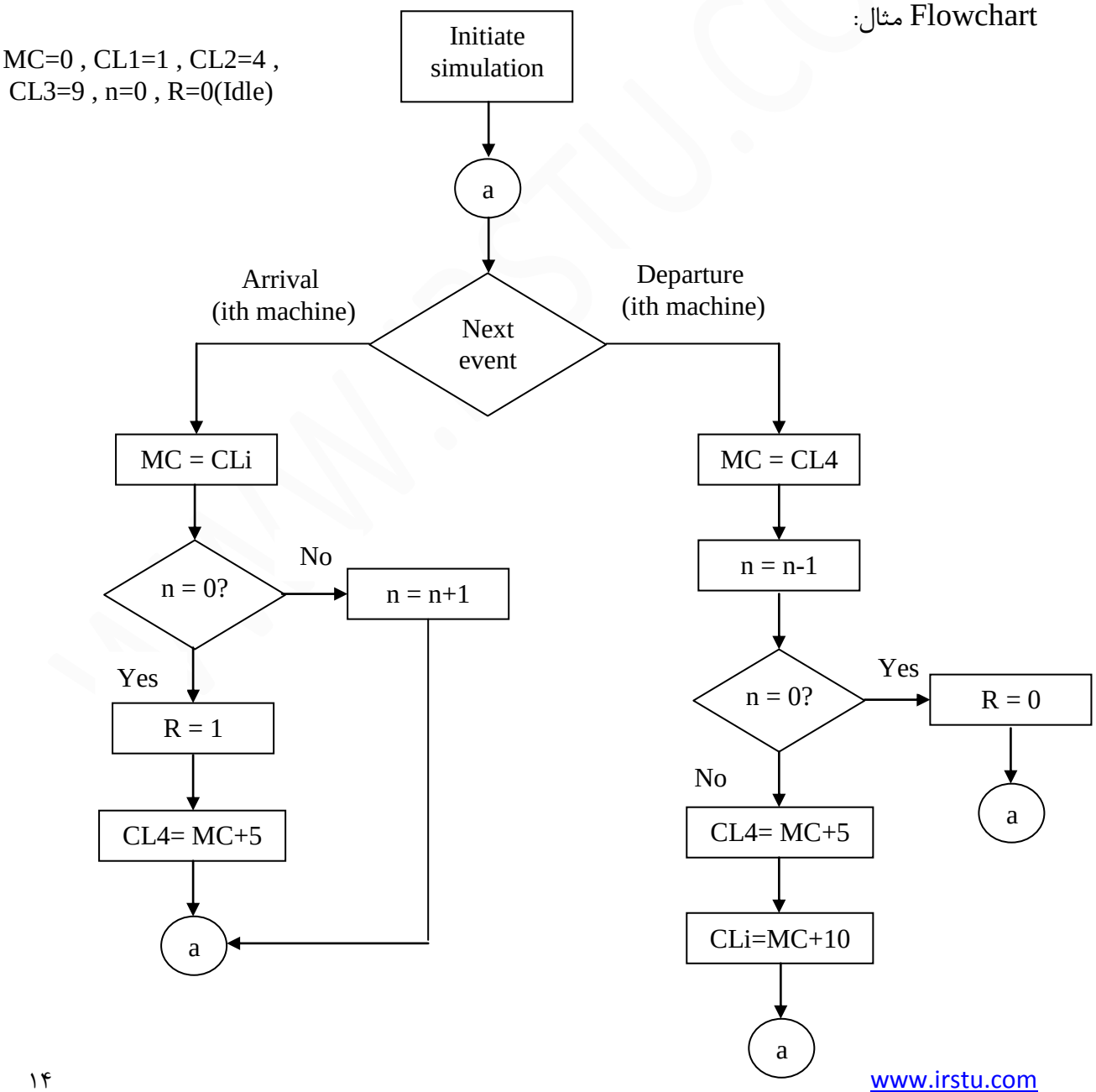
هر دو Flowchart خرابی و تعمیر ماشین را با هم ترسیم می کنیم:



مثال- فرض کنید که مسئله ای با 3 ماشین داشته باشیم. هر یک از این ماشین ها یک ساعت خواهند داشت که به ترتیب آنها را CL1 ، CL2 و CL3 می نامیم. ساعت چهارم یا CL4 زمانی را نشان می دهد که ماشین در حال تعمیر، تعمیرش تمام می شود؛ به عبارت دیگر 3 ساعت اول زمان وارد شدن ماشین ها را به صف نشان می دهد و ساعت چهارم زمان خارج شدن ماشین در حال تعمیر از صف را نشان می دهد. برای سادگی کار فرض می کنیم زمان کارکرد هر یک از ماشین ها 10 واحد زمانی و زمانی که تعمیرکار برای تعمیر یک ماشین صرف می کند 5 واحد زمانی می باشد.

MC	CL1	CL2	CL3	CL4	N	R
0	1	4	9	-	0	Idle
1	-	4	9	6	1	Busy
4	-	-	9	6	2	Busy
6	16	-	9	11	1	Busy
9	16	-	-	11	2	Busy
11	16	21	-	16	1	Busy
16	-	21	26	21	1	Busy
…	…	…	…	…	…	...

مثال: Flowchart



تمرین1: برنامه مثال قبل را با یک زبان برنامه نویسی بنویسید.

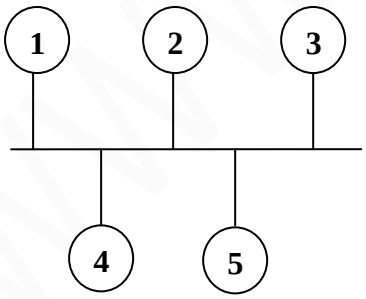
تمرین2: شبیه سازی دستی را برای مسئله تعمیر ماشین ها برای هر یک از شرایط ذیل به صورت جداگانه انجام دهید:

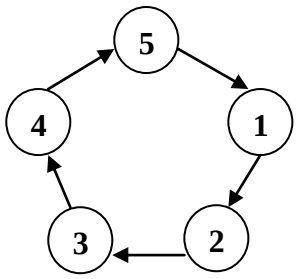
I. زمان تعمیر هر یک از ماشین ها و زمان کارکردشان را متغیر در نظر بگیرید.
II. تعداد تعمیرکارها را تغییر دهید.
III. فرض کنید که سیاست صف FIFO نباشد، ماشینی زودتر تعمیر می شود که زمان تعمیر آن کوتاه تر باشد.

مثال- مکانیزم دسترسی مبتنی بر Token شبکه ای را در نظر بگیرید که شامل تعدادی گره است و این گره ها از طریق یک گذرگاه مشترک به هم متصل شده اند. دسترسی به گذرگاه با استفاده از Token صورت می گیرد؛ یعنی یک گره تا زمانی که Token را در اختیار نگیرد قادر نیست داده ای را ارسال کند. در این مثال یک نسخه ساده سازی شده از این شبکه را شبیه سازی می کنیم.

تنها یک Token وجود دارد که به صورت ترتیب منطقی مشخصی در اختیار گره ها قرار می گیرد. گره ها به صورت منطقی یک حلقه را تشکیل می دهند.

✓ ساختار گره ای به صورت Ring است.





در مورد گره هایی که به صورت Token-based به هم متصل شده اند، ممکن است ترتیب اتصال منطقی با ترتیب اتصال فیزیکی یکسان نباشند. فرض می کنیم که Token هیچ وقت گم نمی شود و هیچ گره ای نمی تواند داده ارسال کند، مگرآنکه Token را در اختیار داشته باشد.

هنگامی که یک گره، Token را دریافت می کند می تواند آن را حداکثر تا دوره زمانی T نگه دارد، در طی این دوره گره می تواند اقدام به ارسال Packet های خود نماید. یک Packet شامل داده و Header است. Header شامل آدرس ارسال کننده، آدرس مقصد و تعدادی فیلد دیگر است.

یک گره Token را زمانی تحویل می دهد که:

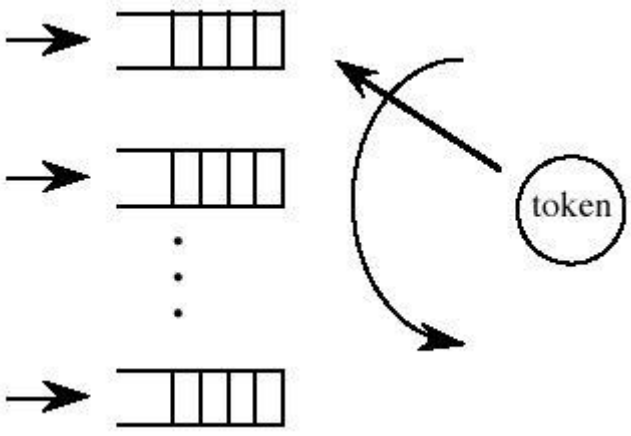
I. زمان T تمام شود.

II. تمام Packet هایش را که در صف قرار دارند قبل از تمام شدن T فرستاده باشد.

III. Token را زمانی دریافت کند که هیچ Packetی برای ارسال نداشته باشد.

اگر زمان T تمام شده باشد و گره در حال ارسال یک Packet باشد، ابتدا ارسال Packet فعلی را تمام می کند و پس از آن Token را تغییر می دهد. منظورمان از تحویل دادن این است که گره Token را در اختیار گره بعد از خود قرار می دهد.

به صورت مفهومی این سیستم را به صورت تعدادی صف می توان در نظر گرفت که برای هر گره یک صف وجود دارد و تنها صفی می تواند داده های خود را ارسال کند که Token را در اختیار داشته باشد. بنابراین Token می تواند به عنوان Server این صف ها در نظر گرفته شود که به صورت دوره ای بین این صف ها جا به جا می شود.



به محض اینکه یک Token در اختیار یک صف قرار می گیرد، Packetهایی که در صف مورد نظر قرار دارند می توانند در شبکه ارسال شوند. بیشترین زمانی که یک صف می تواند Token را نگه دارد T واحد زمانی است و Token برطبق قواعد توضیح داده شده در اختیار صف بعدی قرار می گیرد.

- ✓ زمانی که طول می کشد تا Token از یک صف به صف بعدی جا به جا شود Switch over time نامیده می شود.

چنین سیستمی (صف)، سیستم Polling نامیده می شود. در هنگام ساخت یک مدل شبیه سازی استفاده از مدلی به شکل صف بسیار ساده تر از مسئله اولیه شبکه است. رخدادهای ذیل می توانند در این شبیه سازی مدنظر قرار گیرند، برای هر صف یک رخداد ورود (arrival) و رخداد تکمیل سرویس وجود دارد. برای Token یک زمان ورود به یک صف و یک زمان تحویل Token به گره بعدی که با عنوان Time-out شناخته می شود، وجود دارد.

- ✓ زمان ورود و خروج، رخداد هستند.

برای هر صف زمان رسیدن (ورود) Packet بعدی، تعداد مشتریان حاضر در صف و زمان خروج Packet از صف (اگر در حال ارسال باشد) نگه داشته می شود.

برای Token زمان رسیدن به صف بعدی، شماره صفی که Token را در اختیار دارد و Time out نگهداری می شود.

در شبیه سازی دستی زیر فرض می کنیم که شبکه 3 گره داشته باشد، در نتیجه 3صف خواهیم داشت. زمانی که طول می کشد تا بعد از رسیدن یک Packet به یکی از صف ها Packet دیگری به آن وارد شود به ترتیب برای صف های 1و2و3 برابر با 10و15و20 واحد زمانی در نظر می گیریم.

فرض می کنیم که T برابر 15واحد زمانی باشد، زمانی که طول می کشد تا یک گره بتواند یک Packet را ارسال کند برابر با 6 واحد زمانی است و زمانی که طول می کشد تا Token از یک گره به گره بعدی منتقل شود برابر با 1 واحد زمانی است.

برای مقادیر اولیه فرض می کنیم که سیستم در ابتدا خالی است و برای صف های 1و2و3 اولین Packetها به ترتیب در زمان های 2و4و6 وارد می شوند، همچنین فرض می کنیم در زمان صفر Token در اختیار صف 1 است.

در حالتی که ورود و خروج یک Packet در یک صف همزمان باشد فرض می کنیم ورود Packet اول اتفاق می افتد و همچنین اگر Token و Packet همزمان به یک صف برسند فرض می کنیم Packet زودتر رسیده است. برای این مسئله ساعت های زیر را می توان در نظر گرفت:

a. MC : Master Clock

زمانی که از شبیه سازی گذشته است.

b. AT_i : Arrival time clock at queue i , i=1,2,3

زمان ورود Packet بعدی به صف iام

c. DT_i : Departure time clock from queue i , i=1,2,3

زمانی که Packet قبلی از صف خارج می شود.

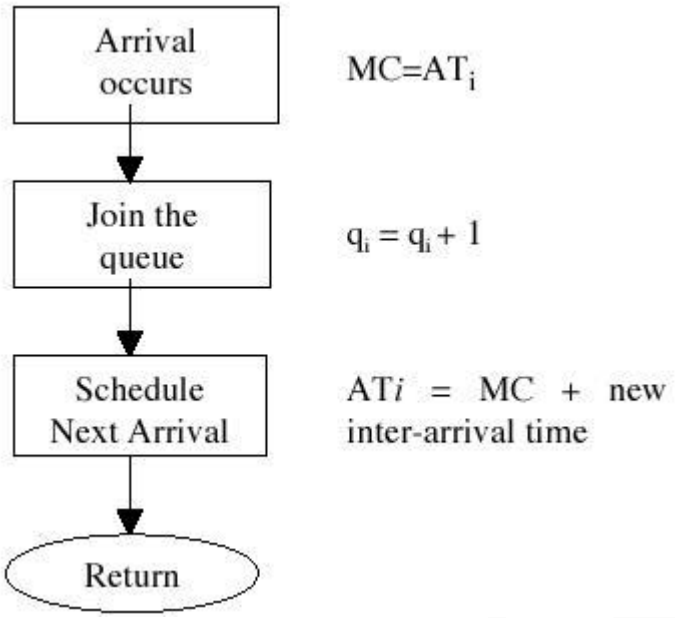
d. T_{out} : Time out clock for token.

زمانی که Node باید Token را تحویل دهد.

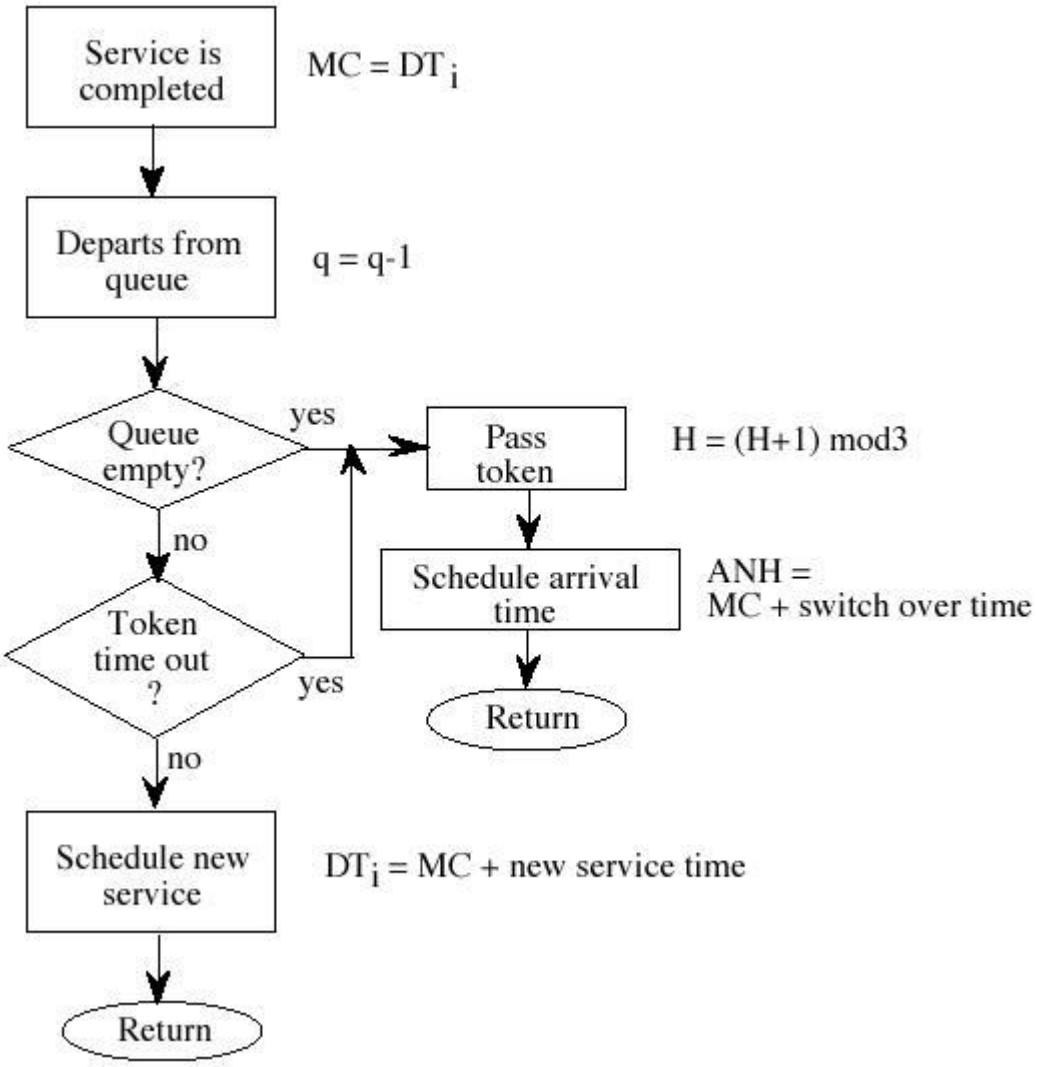
e. ANH : Arrival time clock of token to next queue.

زمانی که قرار است Token در اختیار صف بعدی قرار بگیرد.

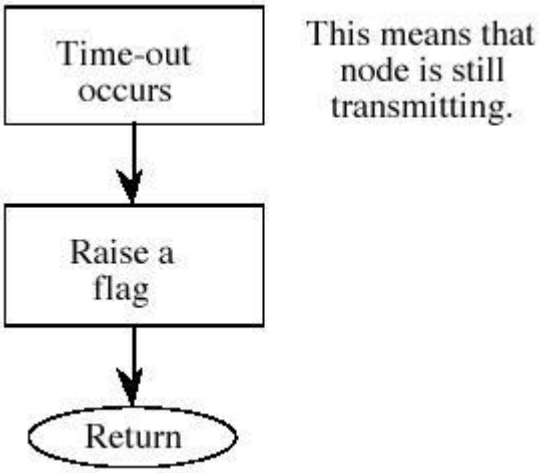
رخداد وارد شدن یک Packet جدید به صف :



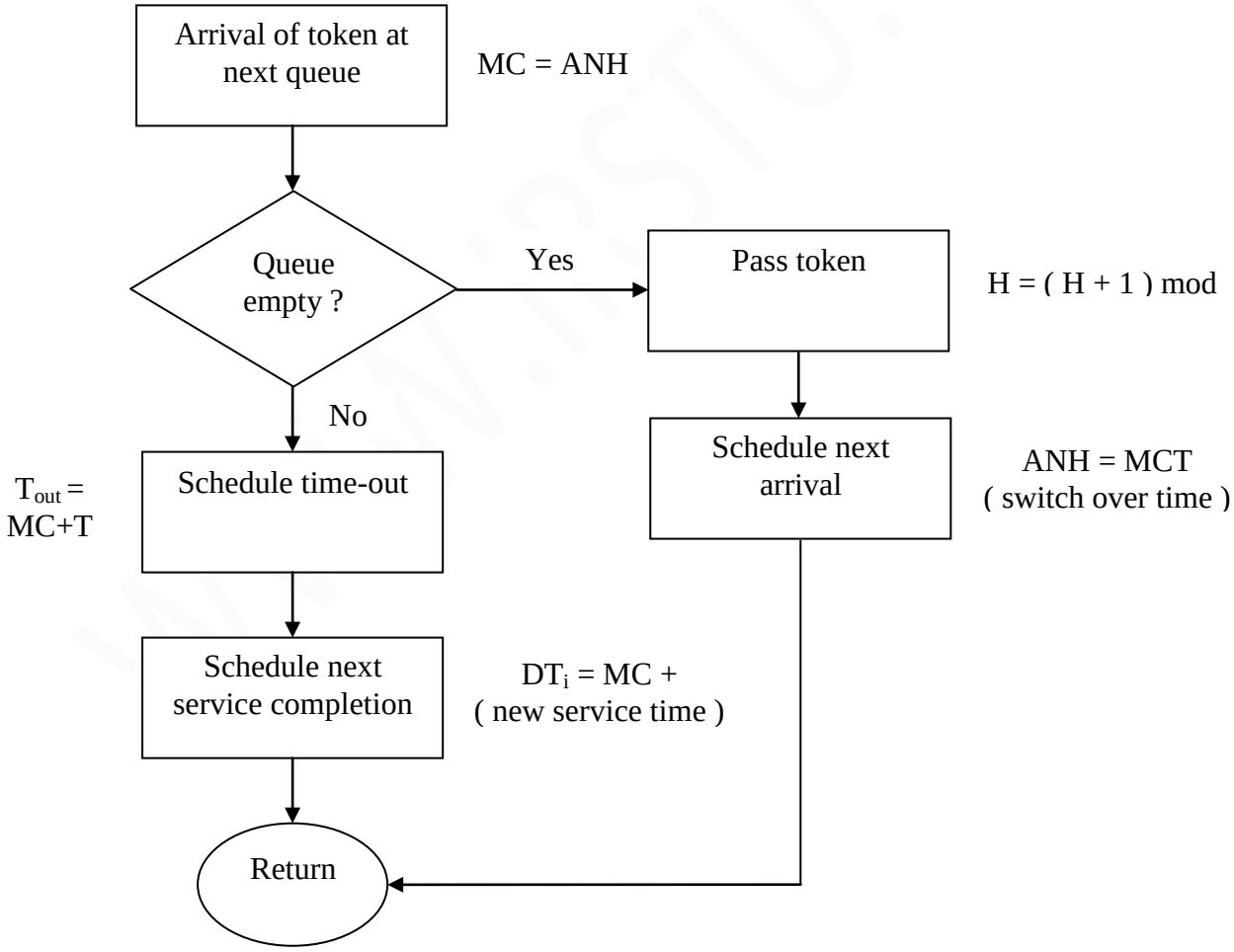
رخداد تمام شدن ارسال یک **Packet** :



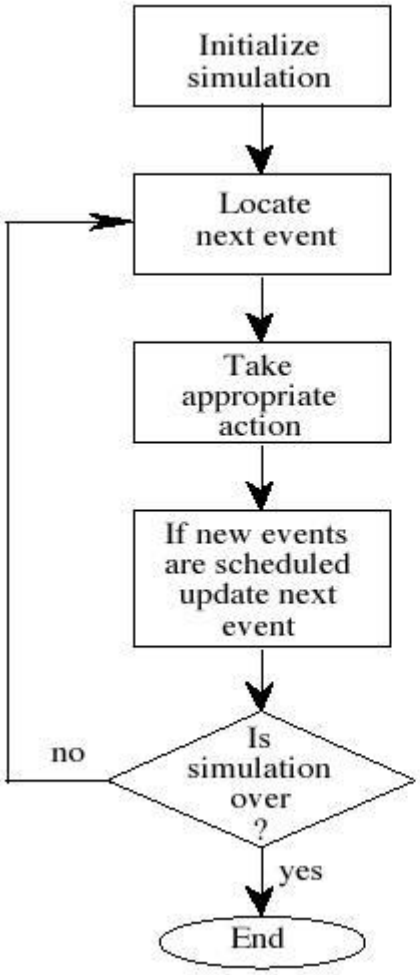
Flowchart اینکه آیا Token به گره بعدی تحویل داده شده یا خیر ؟



Flowchart وقتی Token به یک صف می رسد:



Flowchart کلی :



فصل دوم :

اعداد تصادفی

در این فصل روش های تولید اعداد تصادفی با توزیع Uniform را مورد بررسی قرار خواهیم داد. بر پایه این روش ها در فصل بعد روش های تولید اعداد تصادفی با توزیع های دیگر را بررسی می کنیم.

اعداد تصادفی در بسیاری از کارها مانند برنامه نویسی، طراحی الگوریتم و همچنین در مورد رمزنگاری کاربرد دارد. در شبیه سازی، اعداد تصادفی به منظور ایجاد تصادف در مدل ایجاد شده به کار می روند. به عنوان مثال می توان مسئله تعمیر ماشین ها را عنوان کرد، در این مثال فرض کردیم که هر ماشین مدت زمان مشخصی را کار می کند در حالی که واقعیت این گونه نیست. وقتی بتوانیم اعداد تصادفی تولید کنیم می توانیم زمان خراب شدن ماشین ها را تصادفی فرض کرده و مدل خود را به واقعیت نزدیک تر کنیم.

برای ایجاد اعداد تصادفی باید بتوانیم از توزیع های احتمالی مختلف توزیع یکنواخت را تولید کنیم.

اعداد تصادفی که در کامپیوتر تولید می شوند در واقع شبه تصادفی هستند. هنگامی که از اعداد تصادفی صحبت می شود منظور فقط یک عدد تصادفی نیست بلکه دنباله ای از اعداد تصادفی است که تشکیل یک توزیع احتمالی را می دهد. دو روش برای تولید اعداد تصادفی وجود دارد:

یکی استفاده از یک وسیله فیزیکی است که بتواند اعداد تصادفی تولید کند. اعدادی که این گونه تولید می شوند اعداد تصادفی واقعی نامیده می شوند. یک تولید کننده اعداد تصادفی واقعی اعدادی را تولید می کند که غیرقابل پیش بینی و غیرقابل تولید مجدد هستند، چنین تولید کننده هایی در طبیعت یافت می شوند.

به عنوان مثال زمان سپری شده میان تشعشع دو ذره رادیواکتیو در طول یک دهه، یک منبع شناخته شده تصادفی است یا روند تبادل اطلاعات میان انسان و کامپیوتر نیز تصادفی است.

اعداد تصادفی واقعی برای استفاده در رمزنگاری و کارهای شبیه آن بسیار مناسب است اما چنین اعدادی در شبیه سازی قابل استفاده نیستند چرا که در شبیه سازی نیاز داریم که بتوانیم دنباله اعداد تصادفی ایجاد شده را دوباره تولید کنیم، به علاوه تولید و ذخیره سازی اعداد تصادفی واقعی بسیار زمان بر است و هزینه زیادی دارد.

یک راه حل جایگزین استفاده از الگوریتم های ریاضی برای تولید اعداد تصادفی است. الگوریتم های زیادی وجود دارند که می توانند با استفاده از یک روند معین اعداد تصادفی تولید کنند، این الگوریتم ها با گرفتن یک مقدار اولیه به نام Seed (دانه) دنباله ای از اعداد تصادفی را تولید کنند، تا زمانی که Seed عوض نشود دنباله تولید شده نیز عوض نمی شود. از آنجایی که این اعداد تصادفی می توانند دوباره تولید شوند شبه تصادفی نامیده می شوند. چون این اعداد تصادفی در هر زمان که بخواهیم قابل تولیداند دیگر نیازی نیست که آنها را ذخیره کنیم. دقت کنید که اصطلاح شبه تصادفی به اعداد تصادفی گفته می شود که هم یکنواخت بوده و هم در بازه [0و1] باشند. تمام اعداد تصادفی دیگر که در بازه ای خارج از بازه [0و1] تولید شوند Random variates یا Stochastic variates نامیده می شوند. برای سادگی از این به بعد به اعداد شبه تصادفی نیز تصادفی می گوییم.

در حالت کلی یک الگوریتم تولید اعداد تصادفی باید شرایط زیر را داشته باشد:

a. توزیع یکنواخت (Uniform) داشته باشد.
b. مستقل آماری باشند.
c. قابل تولید مجدد باشند.
d. برای هر طول دلخواه تکرار نشود.

اولین روش تولید اعداد تصادفی توسط Von Neuman ارائه شد. این روش که روش Mid-Square (مربع میانه) نامیده می شود بدین صورت عمل می کند که در آن عدد تصادفی جدید ارقام وسطی مجذور عدد تصادفی قبلی است.

به عنوان مثال فرض کنید مقدار Seed = 5772156649 باشد؛

$Seed^2$ = 3331738, 7378338971, 4909201

x_0 = Seed

x_1 = 7378338971

$x_i+1 = (x_i)^2$

معایب روش Von Neuman :

1. به مقدار Seed حساس است.
2. تولید اعداد میانی که ممکن است در یک حلقه قرار بگیرند و تمام اعداد تکراری شوند.

مثلاً اگر Seed = 6500 در نظر بگیریم عددهای بعدی برابر 250 خواهند بود.

Seed = 6500

$Seed^2 = 42, 250, 00 \rightarrow (250)^2 = 06, 250, 00$

صفر را اضافه می کنیم تا بتوانیم 3 رقم مورد نظر را جدا کنیم.

روش هم نهشتی (Congruential) :

این روش، روش بسیار معروفی است و نسخه های مختلفی از آن ارائه شده و استفاده می گردد. مزایای این روش سادگی، سرعت زیاد و تولید دنباله اعداد تصادفی است که از لحاظ آماری برای شبیه سازی قابل قبول است. فرمول این روش به صورت بازگشتی بوده و به شکل زیر نوشته می شود:

$$x_{i+1} = (ax_i + c) \bmod m$$

که در آن a ، c و m اعداد غیرمنفی هستند.

مثال:

$x_0 = 0$

$a = c = 7 \quad \rightarrow x_{i+1} = (7x_i + 7) \bmod 10$

$m = 10 \quad = 7,6,9,0,7,\ldots$

این روش، روش هم نهشتی ترکیبی (Mixed) نامیده می شود.

یک نسخه ساده تر این روش دارای فرمول زیر می باشد:

$$x_{i+1} = (ax_i) \bmod m$$

که به آن روش هم نهشتی ضربی می گویند.

اعدادی که با روش هم نهشتی تولید می شوند بین صفر تا m-1 قرار دارند. برای اینکه این اعداد در بازه [0و1] قرار گیرند کافی است تا آنها را بر m-1 تقسیم کنیم.

به تعداد اعداد تصادفی متوالی که تولید می شوند و پس از آن دنباله اعداد شروع به تکرار می کند دوره (Period) گفته می شود. اگر دوره برابر m باشد می گوییم مولد دارای دوره کامل است. تئوری های اطلاعات نشان داده اند که دوره وابسته به m است، هرچه m بزرگتر باشد دوره نیز بزرگ تر خواهد بود به ویژه اگر a ، c و m شرایط زیر را داشته باشند مولد بهتری خواهیم داشت و دوره آن کامل تر خواهد بود.

a. m و c عامل مشترک نداشته باشند.

b. $a \overset{r}{\equiv} 1$ a به پیمانه r با 1 هم نهشت باشد، اگر r عامل اول m باشد. یعنی اگر r یک عدد اول باشد که m بر آن بخش پذیر است، a-1 نیز بر r بخش پذیر باشد.

c. $a \overset{4}{\equiv} 1$ اگر m مضربی از 4 باشد. یعنی اگر m مضربی از 4 باشد a-1 نیز بر 4 بخش پذیر باشد.

مهم است که بدانیم a ، c و m نمی توانند هر مقداری داشته باشند و باید شرایط فوق را دارا باشند. مثلاً اعداد زیر می توانند برای a ، c و m انتخاب شوند؛

a = 314 , 159 , 256

c = 453 , 809 , 245

$m = 2^{32}$ (for a 32-bit machine)

همان گونه که گفتیم برای شروع اعداد تصادفی نیاز به یک مقدار اولیه (Seed) داریم اما نشان داده می شود که پس از تولید تعداد محدودی از اعداد دیگر Seed تاثیر چندانی بر دنباله اعداد تولید شده ندارد. برای این روش نیاز به یک عمل ضرب، یک جمع و یک تقسیم داریم. به دلیل اینکه $m = 2^{32}$ در نظر گرفته شده است دیگر نیازی به عمل تقسیم نداریم. عبارت داخل پرانتز پس از محاسبه چه سرریز رخ دهد و چه رخ ندهد عددی بین صفر تا $2^{32}-1$ خواهد بود که باقیمانده آن نسبت به m خود آن عدد است.

به عنوان مثال، کامپیوتری را فرض کنید که اعداد آن از صفر تا 99 هستند. اگر m=100 در نظر بگیریم و a=8 ، x_i=2 و c=10 باشند آنگاه $ax + c = 26$ می شود که باقیمانده آن نسبت به m خود آن عدد است.

به عنوان مثال دیگر اگر x_i=20 باشد آنگاه $ax_i + c = 170$ خواهد بود که مقدار 1 آن برای سرریز دور ریخته می شود و عدد 70 همان باقیمانده نسبت به m خواهد بود.

روش های هم نهشتی عمومی (کلی) :

روش هم نهشتی ترکیبی حالت خاصی از روش هم نهشتی عمومی است. فرمول این روش به صورت زیر نوشته می شود:

$$x_{i+1} = f(x_i, x_{i-1}, \dots) \bmod m$$

که در آن f تابعی از اعداد تصادفی قبلی است. حالت خاصی از این روش، روش هم نهشتی مربعی است:

$$x_{i+1} = (a_1 x_i^2 + a_2 x_{i-1} + c) \bmod m$$

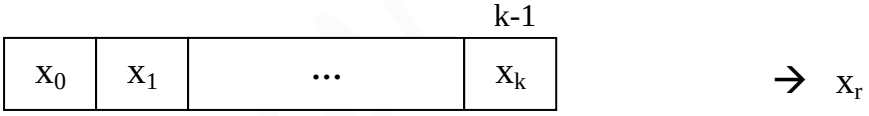
حالت خاص دیگری از این روش هنگامی است که f به صورت زیر باشد:

$$f(x_i, x_{i-1}, \dots, x_{i-k}) = a_1 x_i + a_2 x_{i-1} + \dots + a_k x_{i-k}$$

که به آن روش هم نهشتی تجمعی گویند.

مولدهای اعداد تصادفی مرکب :

این مولدها از ترکیب دو مولد اعداد تصادفی به دست می آیند. این ترکیب بدین دلیل صورت می گیرد که امیدواریم با ترکیب دو مولد، مولد جدیدی بدست بیاوریم که از لحاظ آماری رفتار بهتری از دو مولد ترکیبی قبلی داشته باشند.



$0 \le r \le k-1$

یک مثال خوب برای مولدهای اعداد تصادفی مرکب، مولدهایی هستند که از ترکیب دو مولد هم نهشتی ایجاد می شوند. در این مولدها ابتدا یک آرایه به طول k با استفاده از اعداد تولید شده از مولد یک پر می شود پس از آن مولد دو عددی بین صفر تا k-1 تولید می کند که این عدد اندیس آرایه ای است که توسط مولد یک پر شده بود اگر اسم آرایه x و عدد تولید شده توسط مولد دوم r باشد. x_r اولین عدد تصادفی خروجی مولد مرکب خواهد بود. پس از آن مولد اول یک عدد تصادفی جدید تولید می کند که جایگزین x_r می شود و این روند ادامه پیدا می کند.

مولد تولید اعداد تصادفی Tousworthe :

مولدهای Tousworthe مولدهایی هستند که از بدست آوردن باقیمانده نسبت به 2 یک مولد هم نهشتی جمعی بدست می آید.

$$x_i = (a_1x_{i-1} + a_2x_{i-2} + \ldots + a_nx_{i-n}) \bmod 2$$

که در آن x_i می تواند صفر یا یک باشد. این نوع تولید کننده ها دنباله ای از صفر و یک ها را تولید می کنند. در این حالت می توان گفت ضرایب نیز باینری هستند که در این صورت محاسبه x_i تبدیل به جمع کردن تعدادی از بیت های قبلی و بدست آوردن باقیمانده آنها نسبت به 2 می شود. این عملیات معادل عملیات XOR است.

$$a_1 = 1, a_2 = 1 \rightarrow x_i = (x_{i-1} + x_{i-2}) \text{ m0d } 2 \rightarrow x_i = x_{i-1} \oplus x_{i-2}$$

بیت های تولید شده کنار هم قرار گرفته و تولید اعداد بزرگ تر را می نمایند. اگر L بیت کنار هم قرار بگیرند می توانند اعدادی بین صفر تا 2^L-1 ایجاد کنند.

در مولد مرکب معرفی شده در بخش قبل یکی از مولدها می تواند مولد Tousworthe باشد ولی هر دو آنها نمی توانند. مولدهای Tousworthe مستقل از کامپیوتر مورد استفاده و تعداد بیت های آن هستند و دوره بسیار طولانی دارند، اما از آن جایی که بیت تولید می کنند بسیار کند هستند.

مولدهای Lagged Fibonacci Generator (LFG) :

این مولدها بهبود یافته مولدهای هم نهشتی هستند. این مولدها از دنباله معروف فیبونانچی الهام گرفته شده اند. در حالت کلی LFG به صورت :

$$x_n = (x_{n-j} \; O \; x_{n-k}) \bmod m$$

تعریف می شود که در آن $0 < j < k$ می باشد. در این مولد عنصر بعدی با ترکیب دو تا از عناصر قبلی با استفاده از عملگر O بدست می آیند. O می تواند جمع، تفریق، ضرب و یا حتی XOR باشد. اگر O عملگر جمع باشد آنگاه LFG تجمعی یا ALFG نامیده می شود. اگر O ضرب باشد LFG، LFG ضربی یا MLFG نامیده می شود. معمولاً در بیشتر موارد از ALFG استفاده می شود که بسیار سریع بوده و پیاده

سازی آن ساده است. این مولد دارای دوره m^k-1 خواهد بود اگر m اول باشد، اما اگر m اول نباشد مثلاً $m=2^{32}$ یا $m=2^{64}$ در نظر گرفته شود دوره آن $(2^k-1) \times 2^{m-1}$ می باشد. در MLFG اگر $m=2^{32}$ یا $m=2^{64}$ باشد دوره برابر $(2^k-1) \times 2^{m-3}$ خواهد بود.

در حالت کلی LFGها مولدهای اعداد تصادفی خوبی هستند و تقریباً کارایی آنها با کارایی مولدهای هم نهشتی یکسان است. این مولدها می توانند به صورت موازی پیاده سازی شوند اما به مقدار Seed بسیار حساس هستند، یعنی اگر Seed خوبی انتخاب نشود کارایی این الگوریتم ها بسیار پایین خواهد آمد و پیدا کردن Seed مناسب بسیار مشکل است.

مولد Mersenne Twister :

روش MT یکی از روش های مهم تولید اعداد تصادفی است که کارایی خوبی دارد و ماکسیمم دوره آن برابر با $2^{19937}-1$ می باشد که این دوره بسیار بیشتر از دوره بسیاری از مولدهای دیگر است. خروجی این روش دارای خواص آماری خوبی است. این روش دنباله ای از بیت ها را تولید می کند که تعداد آنها به اندازه دوره این روش است. دنباله بیت ها در این روش به بخش های 32بیتی تقسیم می شوند که این بخش های 32بیتی همان اعداد تصادفی تولید شده در این روش خواهند بود. رابطه زیر رابطه بازگشتی این روش است:

$$\boldsymbol{x_{k+n} = x_{k+m} \oplus (x_k^u \,|x_{k+1}^l)\, A\,, k \geq 0}$$

x عددی است که دارای w بیت است.

x_k^u ، w-r بیت پرارزش x_k است. $(0 \leq r \leq w)$

x_{k+1}^l ، r بیت کم ارزش x_{k+1} است.

$\oplus$ علامت XOR می باشد.

| علامت الحاق می باشد، یعنی بیت ها را به هم می چسباند.

n درجه بازگشت الگوریتم و m عددی صحیح است که $1 \leq m \leq n$ و A آخرین عنصر رابطه یک ماتریس $w \times w$ است که به صورت زیر تعریف می شود:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & \cdots & a_0 \end{pmatrix}$$

در این روش به تعداد n، Seed استفاده می کنیم:

$$x_0 , x_1 , \ldots , x_{n-1}$$

عمل ضرب در ماتریس می تواند به سادگی به صورت زیر انجام شود:

$$xA = \begin{cases} shiftright(x) \text{ if } x_0 = 0 \\ shiftright(x) \oplus a \text{ if } x_0 = 1 \end{cases}$$

که در آن:

$$a = \{ a_{w-1} , a_{w-2} , \ldots , a_0 \} , x = \{ x_{w-1} , x_{w-2} , \ldots , x_0 \}$$

پس از عمل ضرب عملیات زیر نیز روی x انجام می شود:

1. $y = x \oplus (x >> u)$
2. $y = y \oplus ((y << s) \Lambda b)$
3. $y = y \oplus ((y << t) \Lambda c)$
4. $z = y \oplus (y >> l)$

که در آن b و c ماسک های بیتی به اندازه کلمه حافظه هستند. مثلاً در 00001111 بیت های صفر حذف می شوند.

L ، s ، t و u اعداد صحیح از پیش تعیین شده هستند. الگوریتم MT به شرح زیر است:

1. Bit Mask Initialization

$$u \leftarrow \underbrace{1\cdots1}_{w-r}\underbrace{0\cdots0}_{r} : \text{(The bitmask for upper } w\text{-}r \text{ bits)}$$

$$ll \leftarrow \underbrace{0\cdots0}_{w-r}\underbrace{1\cdots1}_{r} : \text{(The bitmask for lower } r \text{ bits)}$$

$$a \leftarrow [a_{w-1}, a_{w-2}, \cdots, a_1, a_0] : \text{(the last row of the matrix } A)$$

2. Seed Initialization

   i ← 0

   (x[0] , x[1] , … , x[n-1]) ← “ any non-zero initial values “

3. Compute $(x_k^u \,|x_{k+1}^l)$

   y ←(x[i] Λ u) V (x[(i+1) mod n] Λ ll)

4. Multiplication whit A:

$$x[i] \leftarrow x[\,(i+m) \bmod n\,] \oplus (y >> 1) \oplus \begin{cases} 0 \text{ if } y_0 = 0 \\ a \text{ if } y_0 = 1 \end{cases}$$

5. Shift Operation

   y ← x[i]

   y ← y ⊕ (y >> u′)

   y ← y ⊕ ((y << s) Λ b)

   y ← y ⊕ ((y << t) Λ c)

   y ← y ⊕ (y >> l)

   Output y

6. i = i + 1 mod n and goto step 3.

مثال – داده های زیر در مسئله داده شده اند.

x[0] = 110111 , x[1] = 110101 , x[2] = 111001

u = 111000 , ll = 000111

a = [101001] , b = 010101 , c = 010110

s = 1 , t = 2 , l = 4

حل –

w = 6 , r = 3 , i ← 0 , u′ = 6 – 3 = 3

1 ≤ m ≤ 3 → m = 2

y = (x[0] Λ u) V (x [(0 + 1) mod 3] Λ ll)

y = 110101

$x[0] \leftarrow x[(0+2) \bmod 3] \oplus (y >> 1) \oplus a$

$x[0] \leftarrow 111001 \oplus 011010 \oplus 101001$

$x[0]$ تغییر می کند $\rightarrow x[0] = 001010$

پیاده سازی مرحله 5 الگوریتم :

1. $y \leftarrow 001010$
2. $y \leftarrow 001010 \oplus (y >> u') \rightarrow y \leftarrow 001010 \oplus 000001 = 001011$
3. $y \leftarrow 001011 \oplus ((y << 1) \wedge b) \rightarrow y \leftarrow 001011 \oplus 010110 \wedge b$
   $y \leftarrow 001101$
4. $y \leftarrow y \oplus ((y << t) \wedge c)$
5. $y \leftarrow y \oplus (y >> l)$

output y

بقیه حل بر عهده دانشجو می باشد.

آزمون های آماری برای بررسی مولدهای تولید اعداد :

در این بخش به پنج آزمون آماری مختلف خواهیم پرداخت که عبارتند از:

1. frequency test,
2. serial test,
3. autocorrelation test,
4. runs test,
5. chi-square test.

سه آزمون اول تصادفی بودن دنباله ای از بیت ها را بررسی می کند در حالی که آزمون های 4و5 بررسی می کند که آیا اعداد در بازه صفر و یک به صورت تصادفی توزیع شده اند یا خیر. آزمون آماری شامل فرض کردن یک فرضیه که آن را در آمار hypothesis می نامیم می باشد.

آزمون فرضیه :

آزمون فرضیه در آمار برای بررسی اینکه آیا یک فرضیه خاص درست است یا نه مورد استفاده قرار می گیرد فرضیه hypothesis گفته می شود و معمولاً یک ادعا در مورد توزیع احتمالی یک یا چندین

متغیر تصادفی است. فرضیه آماری که مورد آزمون قرار می گیرد null hypothesis نامیده می شود که آن را با H_0 نشان می دهیم و عکس این فرضیه H_a نامیده می شود. به عنوان مثال ادعایی که ۳۰٪ ماشین ها قرمز هستند null hypothesis می باشد و عکس این قضیه H_a خواهد بود.

برای بررسی یک فرضیه ابتدا یک سری داده جمع آوری می کنیم سپس بر اساس آن داده یک آزمون فرضیه آماری انجام می دهیم. به عنوان مثال برای بررسی این ادعا که ۳۰٪ ماشین ها قرمز هستند باید یک سری داده در مورد ماشین ها جمع آوری کنیم و بر اساس این داده ها باید P_{red} را حساب کنیم. مشخص است که به احتمال زیاد P_{red} محاسبه شده ۳۰٪ نخواهد بود. P_{red} هیچ وقت ۳۰٪ نخواهد بود. آزمون آماری به ما کمک خواهد کرد تا بفهمیم آیا این تفاوت بین ادعای ما و عدد بدست آمده به خاطر نمونه گیری بوده است و یا واقعاً ۳۰٪ ماشین ها قرمز نیستند.

Real Situation	Decision	
H_0 is True	Valid	Type I Error
H_0 is not True	Type II Error	Valid

جدول خطای نوع ۱ و نوع ۲

در بررسی فرضیه ها دو نوع خطا وجود دارد:

1. خطای نوع 1
2. خطای نوع 2

خطای نوع 1 هنگامی اتفاق می افتد که H_0 واقعاً درست باشد ولی آزمون آماری ما بگوید غلط است و برعکس خطای نوع 2 زمانی اتفاق می افتد که H_0 ما در واقعیت غلط باشد اما آزمون آماری بگوید درست است. معمولاً به خطای نوع 1، False negative و به خطای نوع 2، False positive گفته می شود. احتمال رخداد خطای نوع 1 را با α نشان می دهند و به آن Level of significance یا درجه اطمینان می گویند.

1. Frequency test (Monobit test):

این آزمون یکی از آزمون های مهم و پایه ای در تولید اعداد تصادفی است و اگر یک مولد تولید اعداد تصادفی این آزمون را پاس نکند به احتمال زیاد نمی تواند از عهده آزمون های سخت تر بربیاید. در یک دنباله اعداد تصادفی از اعداد، تعداد صفر و یک ها معمولاً یکسان است. این آزمون این مسئله را بررسی می کند. برای انجام این کار از تابع خطای مکملی یا complementary error function = erfc استفاده می شود.

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2}\, dt$$

مراحل این آزمون به شرح زیر است:

1. m عدد تصادفی تولید شده توسط مولد را به صورت بیتی پشت سرهم می نویسیم، پس لذا به جای صفرها، 1- می نویسیم. فرض کنید تعداد صفر و یک ها n باشد، پس از آن $S_n = x_1 + x_2 + \dots + x_n$ را حساب می کنیم.
2. $S_{obs} = \frac{|S_n|}{\sqrt{n}}$ را حساب می کنیم.
3. $P_{value} = erfc \frac{|S_{obs}|}{\sqrt{2}}$. اگر P-value کوچکتر از 0.01 باشد H_0 را رد می کنیم در غیر اینصورت آن را قبول می کنیم.

(مثال 1011010101

S_n=1-1+1+1-1+1-1+1-1+1=2

$$S_{obs} = \frac{|S_n|}{\sqrt{n}} = \frac{|2|}{\sqrt{10}} = 0.6324$$

$$P_{value} = erfc \frac{|S_{obs}|}{\sqrt{2}} = 0.5271 > \frac{1}{100}$$

در نتیجه اعداد تصادفی هستند.

Serial test.2 :

2^k راه مختلف برای در کنار هم چیدن k بیت وجود دارد. هر یک از این ترکیب ها شانس مساوی برای رخ دادن در یک دنباله تصادفی از بیت ها دارند. فرض کنید e نشان دهنده n بیت متوالی تولید شده توسط یک مولد اعداد تصادفی باشد و k عددی باشد $\boldsymbol{k < \lfloor log_2^n \rfloor - 2}$.

مراحل این الگوریتم به شرح زیر است:

6. k-1 بیت اول e را به آخر آن کپی کنید و آن را e_1' بنامید. k بیت اول e را به آخر آن اضافه کنید و آن را e_2' بنامید.k بیت اول e را به آخر آن اضافه کنید و آن را e_3' بنامید.

7. تعداد تکرار هر کدام از ترکیب های k و k-1 و k-2 بیتی را به ترتیب در e_1'، e_2'و e_3' حساب کنید و آن ها را برای ترکیب iام به ترتیب f_i، f_i' وf_i''بنامید.

8.

$$;\boldsymbol{S_k^2 = \frac{2^k}{n}\sum_i f_i^2 - n}$$

$$;\boldsymbol{S_{k-1}^2 = \frac{2^{k-1}}{n}\sum_i f_i'^2 - n}$$

$$.\boldsymbol{S_{k-2}^2 = \frac{2^{k-2}}{n}\sum_i f_i''^2 - n}$$

9.

$$;\boldsymbol{\Delta S_k^2 = S_k^2 - S_{k-1}^2}$$

$$.\boldsymbol{\Delta^2 S_k^2 = S_k^2 - 2S_{k-1}^2 + S_{k-2}^2}$$

10.

$$;\text{P-value}_1 = \text{Incomplete Gamma}\ (\boldsymbol{2^{k-2}, \Delta S_k^2/2})$$

$$.\text{P-value}_1 = \text{Incomplete Gamma}\ (\boldsymbol{2^{k-3}, \Delta^2 S_k^2/2})$$

اگر P-value و P-value_2 از 0.01 کوچک تر بودند H_0 را رد می کنیم در غیر این صورت آن را قبول می کنیم.

Gammanic :

$$\begin{cases} p(a,x) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} \; dt \\ \Gamma(a) = \int_0^\infty e^{-t}\, t^{x-1} \qquad dt \end{cases}$$

مثال- فرض کنید :

e = 0011011101 → n = 10

k باید حتماً از فرمول $k < [log_2^n] - 2$ بدست آید اما در اینجا ما آن را 3 در نظر می گیریم. → k = 3

k-1 = 2 → e_1' = 001101110100

k-2 = 1 → e_2' = 00110111010

k-3 = 0 → e_3' = e

تعداد تکرار هر یک از ترکیبات k = 3 را در e_1' پیدا می کنیم:

C_1 = 000 → f_1 = 0

C_2 = 001 → f_2 = 1

C_3 = 010 → f_3 = 1

C_4 = 011 → f_4 = 2

C_5 = 100 → f_5 = 1

C_6 = 101 → f_6 = 2

C_7 = 110 → f_7 = 2

C_8 = 111 → f_8 = 1

تعداد تکرار هر یک از ترکیبات k = 2 را در e_2' پیدا می کنیم :

C_1' = 00 → f_1' = 1

C_2' = 01 → f_2' = 3

C_3' = 10 → f_3' = 3

C_4' = 11 → f_4' = 3

تعداد تکرار هر یک از ترکیبات k = 1 را در e_3' پیدا می کنیم :

$C_1'' = 0 \quad \rightarrow \quad f_1'' = 4$

$C_2'' = 1 \quad \rightarrow \quad f_2'' = 6$

$$S_3^2 = \frac{2^3}{10} = (0 + 1 + 1 + 4 + 1 + 4 + 4 + 1) - 10 = 2.8$$

$$S_2^2 = \frac{2^2}{10} = (1 + 9 + 9 + 9) - 10 = 1.2$$

$$S_1^2 = \frac{2}{10} = (16 + 36) - 10 = 0.4$$

$$\Delta S_k^2 = S_3^2 - S_2^2 = 2.8 - 1.2 = 1.6$$

$$\Delta^2 S_k^2 = S_3^2 - 2S_2^2 + S_1^2 = 0.8$$

P-value$_1$ = Incomplete Gamma $(2^{k-2}, \Delta S_k^2/2)$ = (2 , 0.8) = 0.9057

P-value$_1$ = Incomplete Gamma $(2^{k-3}, \Delta^2 S_k^2/2)$ = (1 , 0.4) = 0.8805

3. Autocorrelation test :

فرض کنید e نشان دهنده n بیت باشد که توسط یک مولد اعداد تصادفی ایجاد شده است. می گوییم e به تعداد d چرخش پیدا کرده است اگر e را d بار به سمت چپ بچرخانیم.

مثال:

1001101 , e = 2 → e_d = 0110110

الگوریتم این روش به شرح زیر است:

1. فرض کنید e_i نشان دهنده رشته بیتی باشد که از چرخاندن e به تعداد i بار بدست آمده باشد. با $1 \le d \le \frac{n}{2}$ تعداد بیت هایی را پیدا می کنیم که بین e_i و e_{i+d} متفاوت است.

$$A(d) = \sum_{i=0}^{n-d-1} e_i \oplus e_{i+d}$$

2.

$$S = \frac{2(A(d) - \frac{n-d}{2})}{\sqrt{n-d}}$$

3.اگر $n-d \geq 10$ با درجه اطمینان 95% باشد می توان گفت که e تصادفی است به شرطی که $S \leq 1.96$ باشد در غیر این صورت H_0 را نمی پذیریم در نتیجه دنباله بیت ها نیز تصادفی نخواهد بود.

مثال:

$e = 01011 \quad , \quad d = 2$

$$A_{(2)} = \sum_{i=0}^{5-2-1} e_i \oplus e_{i+d} = (01011) \oplus (01101) + (10110) \oplus (11010) + (01101) \oplus (10101) = 6 + 12 + 24 = 42$$

$$S = \frac{2(A(d) - \frac{n-d}{2})}{\sqrt{n-d}} = \frac{2(42 - \frac{5-2}{2})}{\sqrt{5-2}} = \frac{81}{\sqrt{3}} = 46.8 \not< 1.96$$

در نتیجه تصادفی نیست.

فصل سوم :

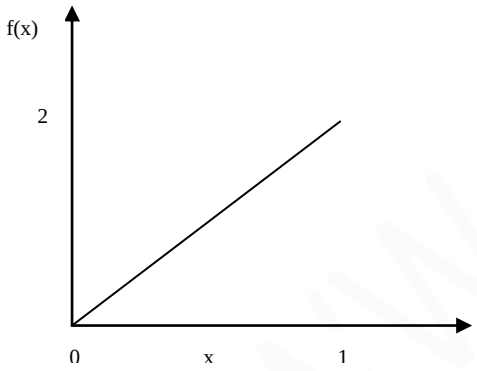
Random Variates

در این فصل به روش های تولید اعداد تصادفی خواهیم پرداخت که از توزیع های مختلف آماری پیروی می کند.

1. روش تبدیل معکوس :

این روش تنها در مواردی قابل استفاده است که از تابع چگالی تجمعی بتوان معکوس گرفت. فرض کنید می خواهیم اعداد تصادفی تولید کنیم که دارای تابع چگالی احتمال $f(x)$ هستند و تابع چگالی تجمعی آن را با $F(x)$ نشان می دهیم. دقت کنید که $F(x)$ در بازه $[0,1]$ قرار دارد. از این خاصیت برای بدست آوردن Random Variate ها استفاده خواهیم کرد. ابتدا یک عدد تصادفی r را تولید کرده و آن را مساوی $F(x)$ قرار می دهیم $(r = F(x))$. پس از آن معکوس F را بدست می آوریم $(x = F^{-1}(x))$.

به عنوان مثال فرض کنید که می خواهیم اعداد تصادفی تولید کنیم که از تابع چگالی احتمال $f(x) = 2x$ و $0 \le x \le 1$ پیروی می کند.



در ابتدا تابع چگالی تجمعی را بدست می آوریم:

$$\boldsymbol{F(x)} = \int_{\mathbf{0}}^{\boldsymbol{x}} \boldsymbol{f(t)dt} = \int_0^x 2t\,dt = x^2 \qquad 0 \le x \le 1$$

$x^2 = r \rightarrow x = \sqrt{r}$

توزیع یکنواخت :

$$\boldsymbol{F(x)} = \int_{\boldsymbol{a}}^{\boldsymbol{x}} \boldsymbol{f(t)dt} = \int_{\boldsymbol{a}}^{\boldsymbol{x}} \frac{\mathbf{1}}{\boldsymbol{b-a}}\boldsymbol{dt} = \frac{\mathbf{1}}{\boldsymbol{b-a}}\boldsymbol{t}\Big|_a^x = \frac{\boldsymbol{x-a}}{\boldsymbol{b-a}}$$

$$F(x) = \frac{x-a}{b-a} = r \rightarrow x - a = r(b-a) \rightarrow x = a + (b-a)r$$

$$E(x) = \int_a^b xf(x)\,dx = \int_a^b \frac{x}{a-b}dx = \frac{b+a}{2}$$

$$Var(x) = \int_a^b \big(x - E(x)\big)^2 F(x)\,dx = \frac{(b-a)^2}{12}$$

تابع توزیع نمایی :

$$f(x) = ae^{-ax} \qquad a > 0\,, x \geq 0$$

$$F(t) = \int_0^x f(t)\,dt = \int_0^x a\,e^{-at}\,dt = 1 - e^{-ax}$$

$$E(t) = \int_0^\infty ae^{-at}t\,dt = \frac{1}{a}$$

$$Var(t) = \int_0^\infty \big(t - E(x)\big)^2 e^{-at}t\,dt = \frac{1}{a^2}$$

$$e^{-ax} = r = F(x) \rightarrow\ 1 - r = e^{-ax} \rightarrow\ x = -\frac{1}{a}log(1-r) \longrightarrow \begin{cases} 0 \leq r \leq 1 \\ 1 \geq 1\text{-}r \geq 0 \end{cases}$$

در نتیجه → $x = -\frac{1}{a}log_e(r)$

یادآوری

$$log_e = ln$$

تابع چگالی توزیع نرمال :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\mu)}{\sigma^2}} \qquad -\infty < x < +\infty$$

که در آن σ یک عدد مثبت است و μ و σ پارامتر های این توزیع هستند. امید ریاضی و واریانس این توزیع به ترتیب برابر μ و σ^2 هستند.

نرم افزارمتلب

(Matlab)

نام متغیر حتماً باید با اسم شروع شود در ادامه اش می توان از (_) و عدد هم استفاده کرد اما در C نام متغیر می توانست با (_) هم شروع شود.

اعداد در متلب از یک شروع می شوند در حالی که در C از صفر شروع می شوند.

نرم افزار متلب به حروف کوچک و بزرگ حساس است.

تعریف متغیر:

بعد از تعریف متغیر حتماً باید کلید Enter را زد.

x = 10

تعریف آرایه:

از کاما یا Space می توان برای فاصله استفاده کرد.

x = [5 6 7]

x = [5 , 6 , 7]

مثال)

x = [5 , 6 , 7]

y = [1 , 2 , 3]

z = [x y]

output → 5 6 7 1 2 3

دستور زیر مقادیر آرایه y را منفی می کند:

z = [x -y]

output → 5 6 7 -1 -2 -3

دستور زیر عنصر 5 آرایه z را بدست می آورد:

z (5)

Output → z = -2

دستور زیر عنصر 5 آرایه z را حذف می کند:

z (5) = [ ]

دستور زیر از عنصر اول شروع می کند، یکی یکی تا 10 جلو می رود و چاپ می کند:

seyyed = 1 : 1 : 10

output → 1 2 3 4 5 6 7 8 9 10

مثال)

seyyed = 1 : 1.5 : 10

output → 1 2.5 4 5.5 7 8.5 10

مثال)

seyyed = 1 : 2 : 10

output → 1 3 5 7 9

مثال)

seyyed = 10 : -1 : -10

output → 10 9 8 … -8 -9 -10

اگر آخر دستور (؛) بگذاریم عمل مورد نظر را انجام می دهد اما نتیجه را نمایش نمی دهد.

در دستور زیر خط اول مقادیر - 10 تا 10 را تولید می کند؛ خط دوم تک تک اعضای آرایه seyyed را به توان 2 می رساند و در متغیر box می ریزد؛ در خط سوم plot نمودار دو بعدی را با مقادیر محور xها (seyyed) و yها (box) رسم می کند:

seyyed = -10 : 0.1 : 10 ;

box = seyyed . ^ 2 ;

plot (seyyed , box)

دستور زیر از صفر تا 100 طوری فاصله ایجاد می کند که 50 نقطه بدست بیاید:

s = linspace (0 , 100 , 50)

دستور زیر ترانهاده ماتریس را بدست می آورد (جای سطر ها و ستون ها را عوض می کند) :

a = [1 5 0 11]

b = a′

دستور زیر معادله دو مجهولی را حل می کند:

[x , y] = solve (′ x + 8 * y = 12 ′ , ′ x – y = 1 ′)

Output → x = 20.9 y = 11.9

دستور زیر معادله ای با دو جواب را حل می کند:

[x , y] = solve (′ x + sin (y) = 0 ′ , ′ x + cos (y) = 1 ′)

Output → x = 1 y = -1.2*pi

x = 0 y = 0

مثال)معادله دو مجهولی

x = solve (′ x^2 + 2x – 4 ′) → x = solve (′ x ^ 2 + 2 * x – 4)

output → 5 ^ (1 . 2) – 1 , -5 ^ (1 . 2) – 1

دستور ترسیم نمودار:

ezplot (′ sin (tan (y)) ′)

تولید اعداد تصادفی:

تولید یک عدد تصادفی بین صفر و یک:

a = rand

تولید هزار عدد تصادفی بین صفر و یک در یک سطر و هزار ستون:

a = rand (1 , 1000)

اگر a = rand (2 , 1000) بود عدد تصادفی در دوسطر و هزار ستون چاپ می شد.

ایجاد یک ماتریس 1000*2 و قرار دادن یک عدد تصادفی بین صفر تا 10 در هر خانه:

(خود 10 را شامل نمی شود)

b = randint (2 , 1000 , 10)

تعریف ماتریس :

x = [1 2 3 ; 0 2 -1 ; 7 1 9] ;

کل مقادیر ماتریس را صفر می کند:

y = zeros (200 , 100) ;

کل مقادیر ماتریس را یک می کند:

y = ones (200 , 100) ;

کل مقادیر ماتریس را 10 می کند:

y = 10 * ones (200 , 100) ;

x = [1 2 ; 3 4]

y = [2 4 ; 8 9]

جمع و تفریق ماتریسی:

z = x ± y ;

ضرب ماتریسی:

z = x * y ;

به جای ضرب ماتریسی درایه های ماتریس را نظیر به نظیر در هم ضرب می کند:

z = x . * y ;

$$\frac{\mathbf{x}}{\mathbf{y}} = \mathbf{x} * \mathbf{y}^{-1};$$

ترتیب اولویت ها:

+
–
*
/
^
'

دستورات زیر در قسمت Current directory فایلی به نام seyyed ذخیره می کند و دستورات تایپ شده را در فایل کپی می کند:

diary seyyed

تایپ دستورات

diary off

دستور زیر صفحه editor را باز می کند و شما می توانید دستورات متلب را در آن بنویسید و اجرا کنید اما در صفحه فعلی دستورات تک تک اجرا می شوند:

edit

مثال)

$$\sqrt{\frac{1-\sqrt{2\pi}}{1+\sqrt{2\pi}}} = (\left(1-(2*pi).\hat{}\ (0.5)\right)/(1+(2*pi)\hat{}0.5))\hat{}0.5$$

همه عملگر ها از (:) اولویت بالاتری دارند:

1 + 1 : 5

(1 + 1) : 5

Output → 2 3 4 5

1 + (1 : 5) → 1 + [1 2 3 4 5]

Output →2 3 4 5 6

1 : 5′

Output →1 2 3 4 5

ماتریس زیر را در نظر بگیرید:

x = [1 2 3 4 5] ;

y = [1 0 1 6 1] ;

دستور زیر درایه ها را نظیر به نظیر به توان y می رساند:

z = x . ^ y

دستور زیر عناصر 2 تا 4 آرایه x را چاپ می کند:

x (2 : 4)

Output →2 3 4

دستور زیر عناصر 2 تا 4 آرایه y را چاپ می کند:

y (2 : 4)

Output →0 1 6

دستور زیر عنصر دوم تا آخر آرایه را چاپ می کند:

2 : end

دستور زیر از عنصر دوم تا عنصر ما قبل آخر آرایه را چاپ می کند:

2 : end – 1

فرمول → x (شروع : گام حرکت : پایان)

مثال)

x = [1 2 3 4 0 5];

x (1 : 2 : 5)

output →1 3 0

دستور زیر سطر اول و تمام ستون ها را چاپ می کند:

x (1 , :)

دستور زیر ستون اول و تمام سطر ها را چاپ می کند:

x (: , 1)

دستور زیر عنصر اول سطر اول و عنصر اول و دوم سطر دوم را چاپ می کند:

[x (1 , 1) , x (2 , 1 : 2)]

دستور زیر کل سطر ها و ستون ها را چاپ می کند:

x (: , :)

مثال)

(1). r = 2 ;

(2). x = [1 2 3 4] ;

(3). x * (r + 1) ^ 2

(4). x (r + 1) ^ 2

دستور 3، r را با یک جمع کرده و به توان 2 می رساند که جواب 9 می شود، سپس تک تک درایه های آرایه را در 9 بدست آمده ضرب می کند:

x = [9 18 27 36]

دستور 4، r را با یک جمع کرده در نتیجه (3) x بدست می آید یعنی عنصر سوم آرایه،که این عنصر سوم را به توان 2 می رساند یعنی 9. توجه داشته باشید که نتایج بالا برحسب تقدم عملگر ها بدست آمد.

دستور زیر عناصر آرایه را با هم جمع می کند:

a = [1 5 10];

sum (a)

output → 16

حلقه for :

در حلقه زیر شمارنده از 1 تا 100 می شمارد :

for i = 1 : 100

.
.
.

end

در حلقه زیر شمارنده از 1 تا 50 شمرده و دو تا دو تا جلو می رود :

for i = 1 : 2 : 50

.
.
.

end

در حلقه زیر شمارنده به صورت نزولی از 100 شمرده و دو تا دو تا کم می کند :

for i = 100 : -2 : 1

.
.
.

end

مثال) دستور زیر را یک بار با for و یک بار با sum بنویسید.

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\frac{1}{5}-\cdots+\frac{1}{1000}$$

با دستور for :

```
x = 0 ;
s = 1 ;
for i = 1 : 1000
        x = x + s . * ( 1 . / i ) ;
        s = - s ;
end
x
```

با دستور sum :

```
x = 1 : 2 : 1000 ;
y = sum ( 1 . / x – 1 . / ( x + 1 ) ) ;
disp ( y )
```

بعد از اتمام برنامه برای نمایش نتایج از **نام متغیر** یا (**نام متغیر**) **disp** استفاده می کنیم.

مثال) دستور زیر را یک بار با for و یک بار با sum بنویسید.

$$\frac{1}{1^2\times 3^2}+\frac{1}{3^2\times 5^2}+\frac{1}{5^2\times 7^2}+\cdots+\frac{1}{21^2\times 23^2}$$

با دستور for :

```
x = 0 ;
for i = 1 : 2 : 21
        x = x + 1 . / ( i . ^ 2 * ( i + 2 ) . ^ 2 ) ;
end
disp ( x )
```

با دستور sum :

```
x = 1 : 2 : 21 ;
y = sum ( 1 . / ( x . ^ 2 . * ( x + 2 ) . ^ 2 ) ) ;
disp ( y )
```

دستور if :

if شرط

elseif شرط

else

end

علامت ها :

> , < , >= , <= , ~= , == , && , || , ~

~ : not

چاپ پیغام :

disp (' پیغام ')

مثال) برنامه ای بنویسید که یک عدد تصادفی بین 1 تا 10 تولید کند، اگر بزرگتر از 5 بود عبارت بزرگتر از 5 را چاپ کند و اگر کوچکتر از 5 بود عبارت کوچکتر از 5 را چاپ کند.

```
a = rand * 10
if a > 5
        disp ( ' bozorgtar ' );
else
        disp ( ' koochektar ' );
end
```

مثال) ضرایب معادله درجه 2 زیر را به صورت تصادفی بین صفر و 10 تولید کنید و سپس معادله را حل کنید.

$Ax^2 + bx + c = 0$

```
a = rand * 10 ;
b = rand * 10 ;
c = rand * 10 ;
d = b . ^ 2 – 4 . * a . * c ;
if  a ~= 0
        if  ( d >= 0 )
                x1 = ( - b + d . ^ 0.5 ) . / ( 2 . * a ) ;
                x2 = ( - b - d . ^ 0.5 ) . / ( 2 . * a ) ;
        else
                disp ( ′ risheye haghighi nadarad ′ ) ;
        end
elseif  b ~= 0
                x1 = - c . / b ;
else
                disp ( ′ rishe nadarad ′ ) ;
end
```

دستور **switch** :

```
switch  d
        case 1
        .
        .
        case 2
        .
        .
        otherwise
        .
        .
End
```

یا

```
switch  d
        case { 1 , 5 , 10 }
        .
        .
        case { 6 , 7 }
        .
        .
        case 0
        .
        .
        otherwise
        .
        .
end
```

مثال) عددی تصادفی صحیح بین 1 تا 10 تولید کنید، اگر عدد 1 و 5 و 10 بود عبارت good و اگر 6 و 7 بود عبارت bad را چاپ کند در غیر این صورت عبارت هیچکدام را چاپ کند.

برای اینکه عدد صحیح تولید کند از randbit استفاده می کنیم→ a = randbit (1 , 1 , [0 10])

```
switch  a
        case { 1 , 5 , 10 }
                disp ( ' good ' )
        case { 6 , 7 }
                disp ( ' bad ' )
        otherwise
                disp ( ' hichkodam ' )
end
```

اعداد مختلط در متلب :

در نرم افزار متلب می توان اعداد مختلط هم تعریف کرد که از نماد i برای آن استفاده می کنیم :

x = a + bi

a مقدار حقیقی (صحیح) و bi مقدار موهومی (مختلط) می باشد.

توابع دیگر که در این زمینه کاربرد دارند عبارتند از :

دستور زیر قسمت صحیح را بر می گرداند:

real (a)

دستور زیر قسمت موهومی را بر می گرداند:

imag (a)

دستور زیر اندازه عدد مختلط را بر می گرداند :

abs (a)

مثال)

a = 2 + 4i → *output :* $\sqrt{2^2 + 4^2}$

با دستور edit صفحه editor را باز کرده و عبارت ; (′ this is a test ′) disp را تایپ کنید و صفحه را با نام test ذخیره کنید و پنجره editor را ببندید، در صفحه command window عبارت test را تایپ کنید :

Output → this is a test

می توان کدی را نوشت و در current directory ذخیره کرد و بعداً از آن در برنامه استفاده کرد.

مثال) در صفحه editor عبارت (′ please enter a number ′) a = input را نوشته و با نام po ذخیره کنید سپس در صفحه command window ، po را تایپ کنید برنامه از شما یک عدد درخواست می کند و توان 2 عدد را محاسبه می کند و نتیجه را نشان می دهد:

```
a = input ( ′ please enter a number ′ ) ;
disp (′ a ^ 2 : ′ ) ;
disp ( a . ^ 2 : ) ;
```

مثال) عبارت زیر را محاسبه کنید.

$$\frac{1+2}{3}+\frac{3+4}{5}+\frac{5+6}{7}+\cdots+\frac{100+101}{102}$$

```
x = 0 ;
for  i = 1 : 2 : 100
        x = ( i + ( i + 1 ) ) . / ( i + 2 ) + x ;
end
```

output → 189.3788

مثال) برنامه ای بنویسید که ضرایب یک معادله درجه 2 را به عنوان ورودی بگیرد و معادله را حل کند.

ابتدا برنامه را در editor تایپ و ذخیره می کنیم:

```
a = input ( ′ a : ′ ) ;
b = input ( ′ b : ′ ) ;
c = input ( ′ c : ′ ) ;
d = b . ^ 2 – 4 . * a . * c ;
x1 = ( - b + d . ^ 0.5 ) . / ( 2 . * a ) ;
x2 = ( - b - d . ^ 0.5 ) . / ( 2 . * a ) ;
disp ( ′ x1 ′ ) , disp ( x1 ) ;
disp ( ′ x2 ′ ) , disp ( x2 ) ;
```

سپس پنجره editor را بسته و در پنجره command window عبارت moadele را تایپ می کنیم و ورودی های درخواست شده را وارد می کنیم :

حال اگر ورودی ها را a = 8، b = 1، c = 1 دهیم جواب به صورت عدد مختلط می شود؛

اگر ورودی ها را a = 0، b = 1، c = 1 دهیم جواب Inl (بی نهایت) می شود؛

اگر ورودی ها را a = 0، b = 0، c = 0 دهیم جواب Non (تعریف نشده) می شود.

دستور time! :

هر دستوری که بعد از ! بیاید به عنوان دستور سیستم عامل اجرا می شود.

این دستور زمان فعلی سیستم را نشان می دهد و به شما این امکان را می دهد که اگر خواستید زمان جدید وارد کنید اما امکان عوض کردن ساعت سیستم را نمی دهد چون administrator نیستید.

تعریف توابع در متلب :

function [خروجی اول خروجی دوم ... خروجی آخر] = اسم تابع (ورودی اول , ورودی دوم , ... , ورودی آخر)

.
.
.

end

تابع در برنامه متلب می تواند چند خروجی داشته باشد، بلعکس زبان های دیگر برنامه نویسی که فقط یک خروجی دارند.

وقتی تابعی را در فایلی ذخیره کردید، موقع فراخوانی تابعی را فراخوانی می کند که اسم فایل همان اسم تابع باشد.

مثال) در صفحه editor برنامه زیر را تایپ کرده و با نام seyyed ذخیره می کنیم:

```
function  [ a   b ] = seyyed ( s )
        a = s . ^ 2 ;
        b = s . ^ 3 ;
end
```

سپس پنجره editor را بسته و در صفحه command window دستور زیر را تایپ می کنیم :

```
[ t   v ] = seyyed ( 15 )
```

t و v را به جای a و b قرار می دهد و عملیات را انجام می دهد.