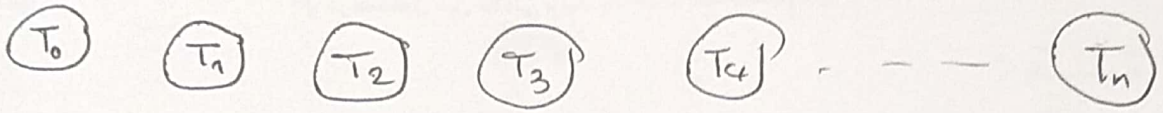
۱- ۱) در این الگوریتم، اعداد را با توجه به مقداری که دارند، در یکی از لیست‌های پیوندی (سطل) قرار داده و با یک الگوریتم stable sort مرتب‌سازی سطل‌ها پرداخته و در آخر هر سطل را با concate می‌کنیم. از آن جایی که برای سطل‌ها آرایه‌ای از لیست‌های پیوندی را داریم و ساختمان داده‌ها خطی است، می‌توانیم از geometric decomposition استفاده کنیم و مرتب‌سازی هر سطل را به یک نخ بسپاریم. همچنین با توجه به ثابت بودن r در هر سطل یا ورودی می‌توانیم از تجزیه recursive این ساختمان داده به تعداد مشخص سطل و سپردن مرتب‌سازی آن‌ها به هر نخ موازی‌سازی را انجام دهیم.

۲) ~~در صورت قبل از نخ داده شده~~ بعد

۳) اگر اعداد آرایه به طور یک نواخت در این بازه پخش شده باشند، آنگاه تعداد n خانه‌های هر سطل کم بوده و منطقی نیست که به هر نخ مرتب‌سازی یک سطل را بسپاریم (geometric decomposition)، زیرا سپردن حجم کمی از کار به تعداد زیادی نخ باعث می‌شود سربار مربوط به context switch زمان اجرا را از اجرای سریال بیشتر و غیربهینه‌تر کند. بنابراین در این حالت برای این الگوریتم بهتر است از recursive data decomposition استفاده کنیم و به صورت بازگشتی این ساختمان داده را پردازش کرده و در آخر نتایج را concate کنیم. اگر پکنواخت هم توزیع شده بود نیز می‌توانیم با توجه به در نظر گرفتن اندازه هر سطل، کار را بین نخ‌ها پخش کنیم. هر چند حالتی که n بسیار بزرگ باشد و به طور یک نواخت در [1..r] پخش شده باشند، و به طوری که طول لیست‌های پیوندی برابر باشد، geometric decomposition گزینه بهتری است.

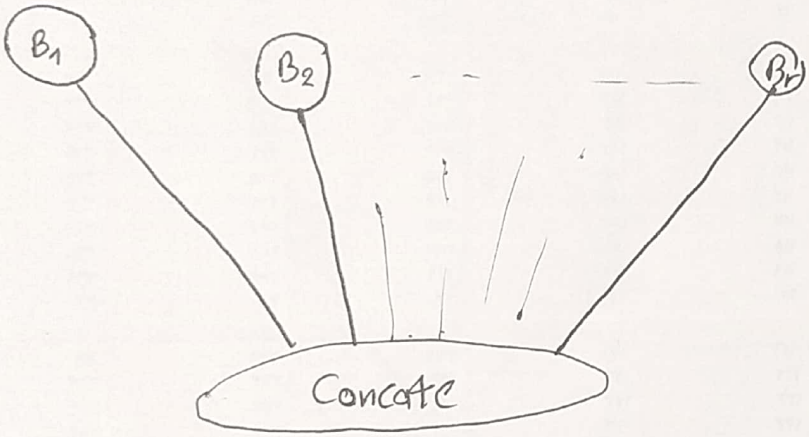


~~geometric~~ ۱-۲) ~~[illegible]~~

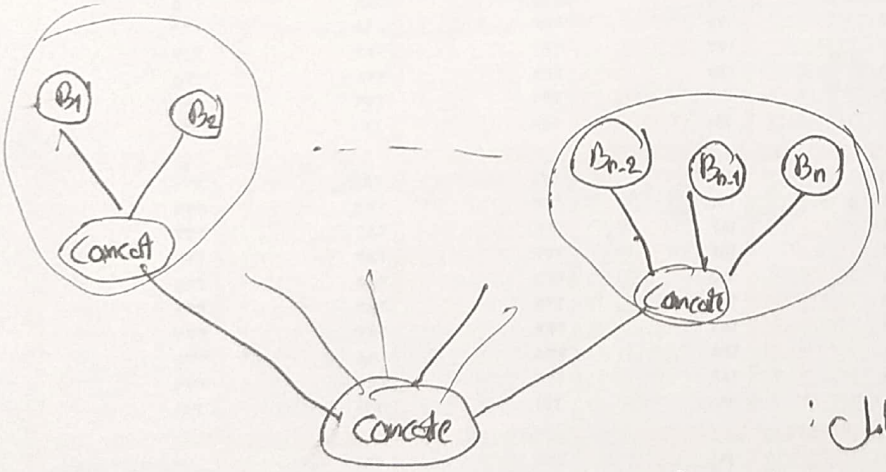
تعریف: در ~~[illegible]~~ وظیفه شامل یک یا چند سطح است

ارتباط: برای geometric

انباشت



برای recursive



نگاشت: بر عهده سیستم شامل:

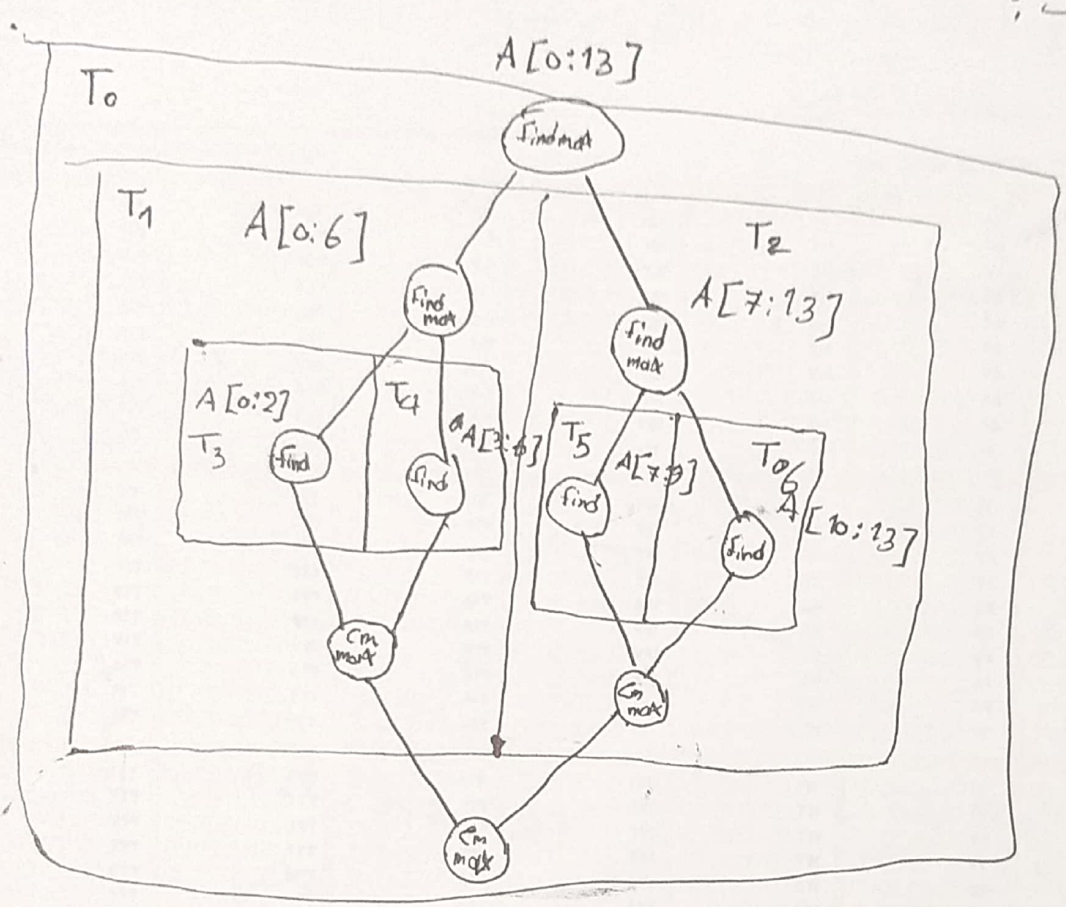
One to one

One to many

many to many

۳- تجزیه: به طور بازگشتی به تجزیه آرایه پرداخته و تا رسیدن به خرد پایه آن را تقسیم می‌کنیم و هر یک وظیفه مرتب سازی آن را دارد.

ارتباط و اتنات:

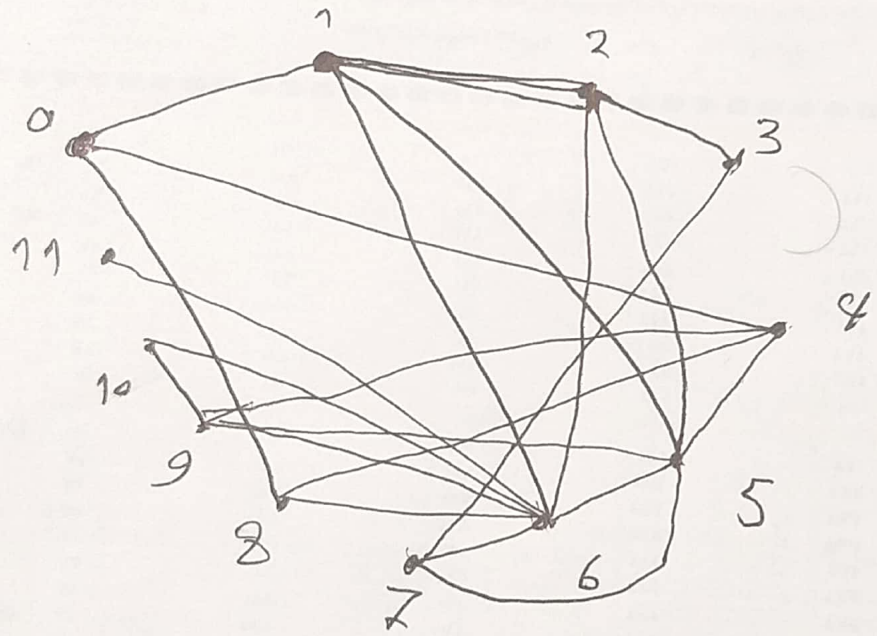


حل نگاشت: بر عهده OS

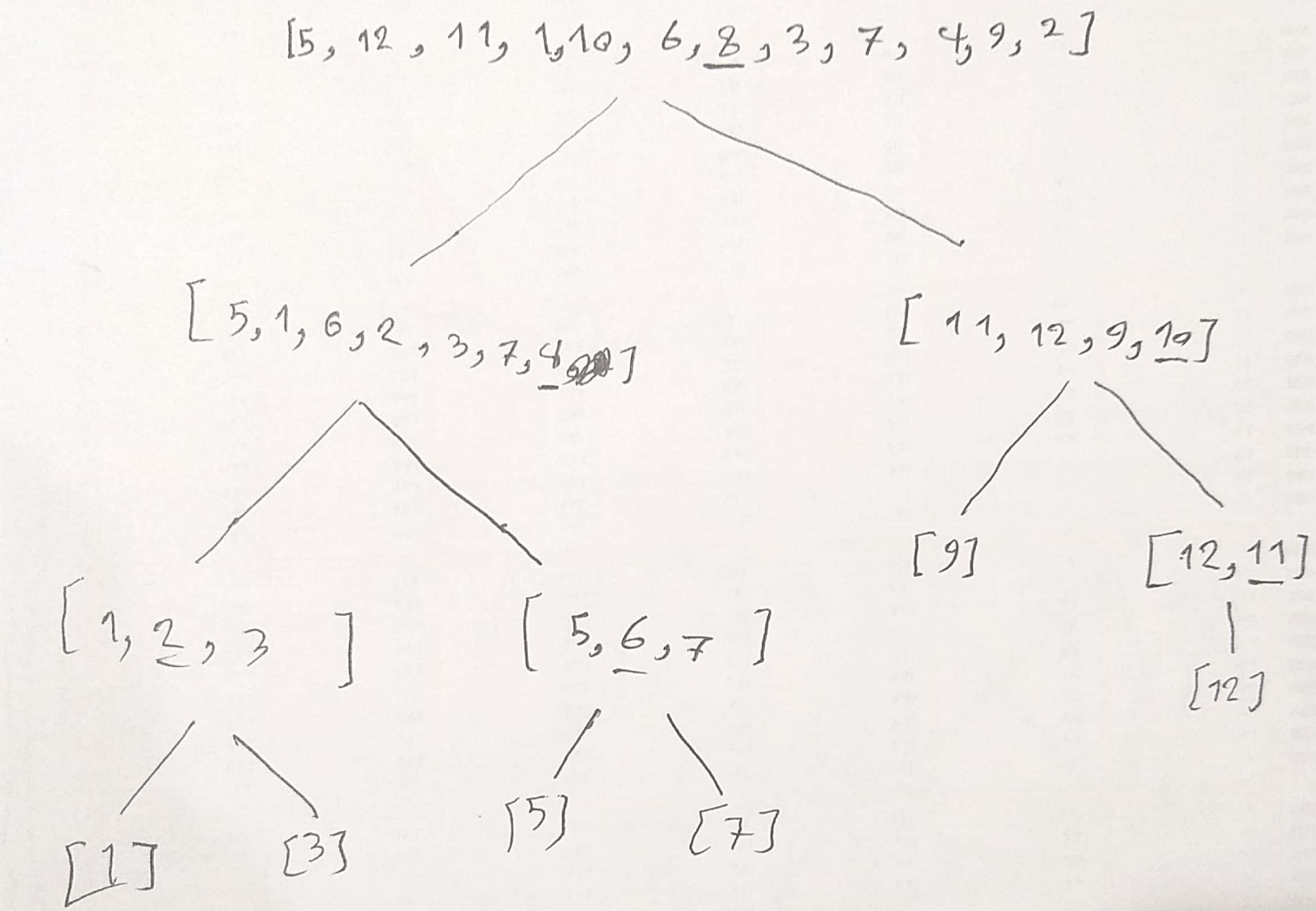
۱) بله، تجزیه Geometric. آرایه را به تعداد مشخص نخ تقسیم کرده و در آخر نتایج را با یک دیگر مقایسه می‌کنیم.

۴- از آنجایی که هر واحد پردازشی برای انجام کار خود به کل آرایه B نیاز دارد، بنابراین هر پردازنده به داده داخل تمام پردازنده های دیگر نیاز دارد و گراف وظایف یک گراف کامل با 12 گره است. در چنین توان این گراف را با حذف محاسباتی که به صفر ختم می شود خلاصه تر کرد و هر تسک آ به تسک ن را مرتبط دانست به شرطی که A[0][j]×B[j] مساوی صفر نباشد و از ضرب های بیهوده خودداری کنیم.

اگر منظور از خانه های سفید، عدد صفر است گراف به صورت زیر در می آید:



ب - در هر سطح یک Pivot انتخاب می کنیم و بعد از قرار دادن Pivot ، آرایه را به دو قسمت قبل و بعد از Pivot تقسیم می کنیم.



۶- به بررسی حلقه for دوم و خط ۷ی پردازیم. خط ۷ عناصر آرایه را دو به دو جدا و با هم جمع می‌کند برای مثال در n=۱ داریم

$$\begin{cases} A[0] = A[0] + A[1] \\ A[1] = A[1] + A[0] \end{cases} \quad \begin{cases} A[2] = A[2] + A[3] \\ A[3] = A[3] + A[2] \end{cases}$$

در n=۲ داریم:

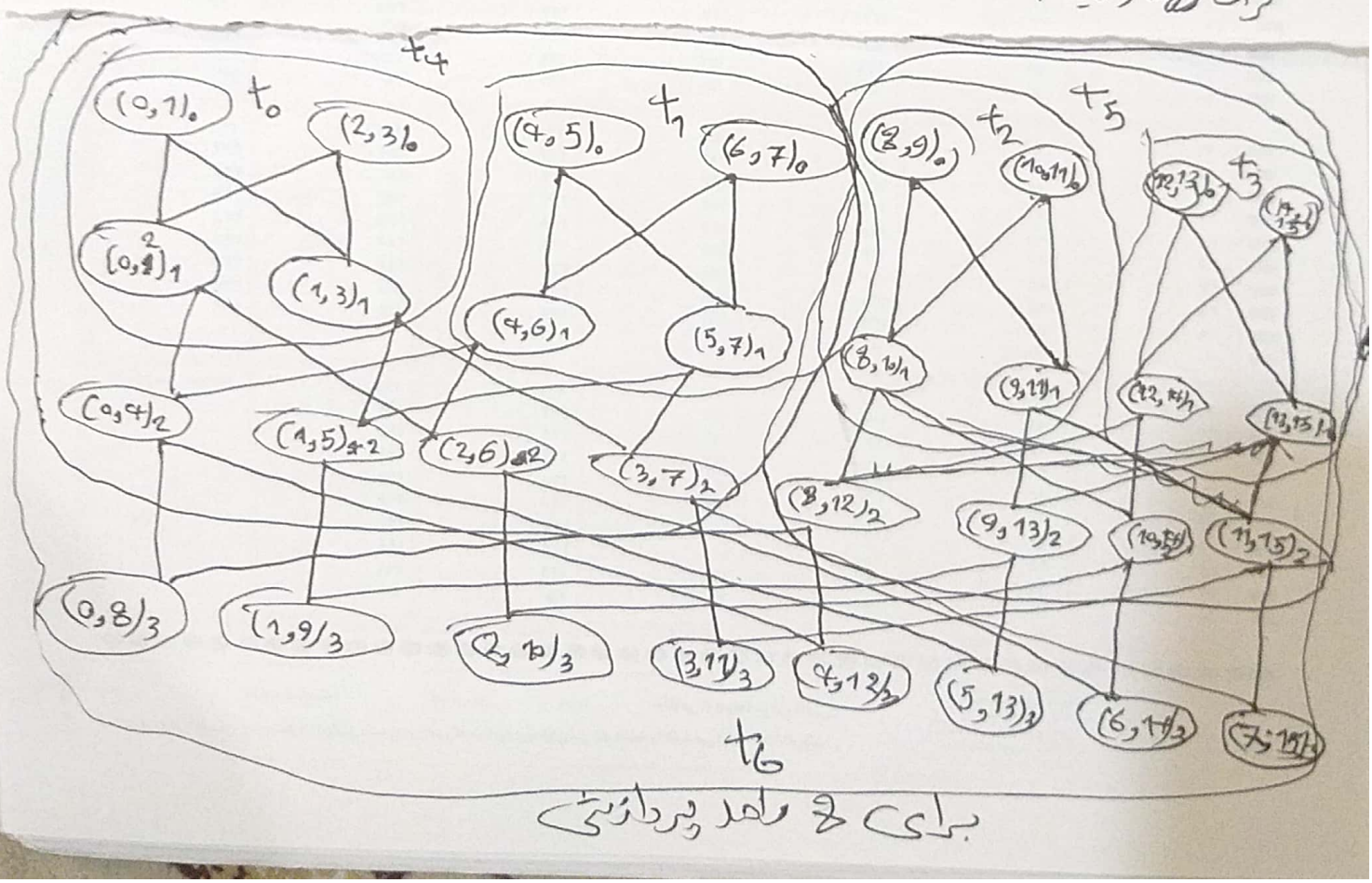
$$\begin{cases} A[0] = A[0] + A[2] \\ A[2] = A[2] + A[0] \end{cases} \quad \begin{cases} A[1] = A[1] + A[3] \\ A[3] = A[3] + A[1] \end{cases}$$

حلقه دوم را به ۸ تسک تقسیم می‌کنیم که هرتسک کد زیر را اجرا می‌کند:

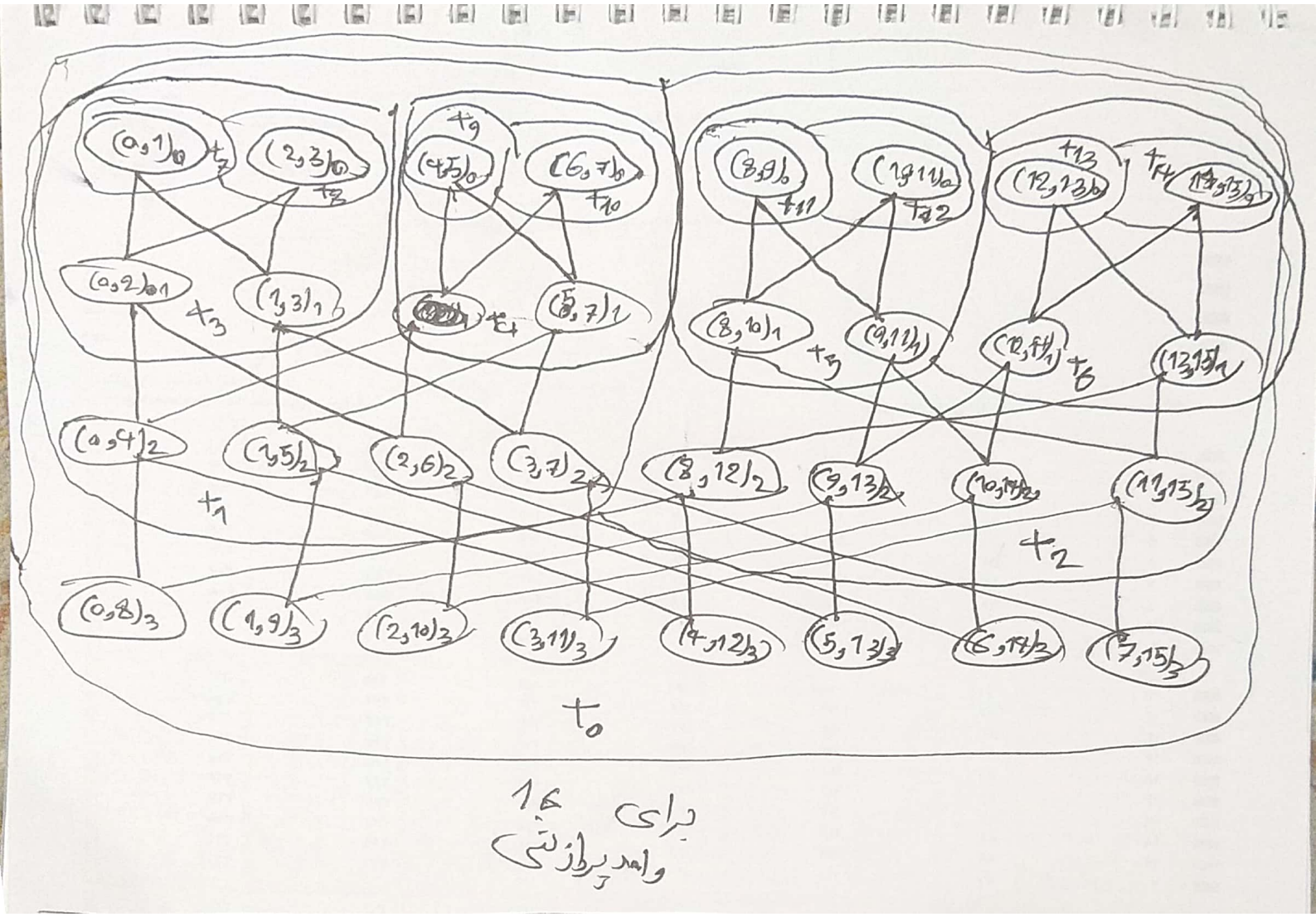
$$A[i] = A[i] + A[i \text{ xor } 2^j]$$

$$A[i \text{ xor } 2^j] = A[i \text{ xor } 2^j] + A[i]$$

گراف وظایف:



برای ۸ واحد پردازشی



برای ۱۶ واحد پردازشی

۷- فرض کنید که یک واحد پردازشی منتظر رسیدن نتیجه پردازش یک واحد پردازشی دیگر بوده تا براساس آن خروجی و درازای شرط ارضا شده تصمیم بگیرد که کاری را انجام دهد. این سناریو بیشتر در دستورات switch-case دیده می‌شود. اگر هر case پردازش سنگینی داشته باشد، می‌توان با کمک تجزیه speculative این مشکل را تاحدودی حل کرد. قبل از رسیدن نتیجه پردازش واحد قبلی، می‌توان پردازش هر case را به یک پردازنده سپرد و با رسیدن نتیجه واحد قبلی نتیجه آن case را به عنوان خروجی بدهیم. این روش تاحدودی زمان اجرا را بهبود می‌دهد، اما باید به این نکته توجه داشت که مقدار حاصل توصیه‌ای از محاسبات ساده‌ای بود، بنابراین بهتر است این تکنیک را برای caseهایی که احتمال رخ دادن آن‌ها بیشتر است استفاده کنیم.

۸. معادله زیر را در نظر بگیرید:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \xrightarrow{\text{decomposition}} \begin{bmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} \\ 0 & U_{2,2} \end{bmatrix}$$

در تبدیل بالا، ماتریس سمت چپ بلاک های ماتریسی دیگی تجزیه شده اند و برای به دست آوردن تجزیه برای بلاک های ماتریسی به طور بازگشتی از الگوریتم خود استفاده می کنیم. تسک ها به صورت زیر تعریف می شوند:

t_1: $A_{1,1} \xrightarrow{\text{decompose}} L_{1,1} U_{1,1}$

t_2: $A_{2,1} = L_{2,1} U_{1,1} \rightarrow L_{2,1} = A_{2,1} U_{1,1}^{-1}$

t_3: $U_{1,2} = L_{1,1}^{-1} A_{1,2}$

t_4: $A_{2,2} = A_{22} - L_{2,1} U_{1,2}$

t_5: $A_{22} \xrightarrow{\text{decompose}} L_{2,2} U_{2,2}$

حال با مشخص شدن وظایف، به مرحله انتساب می‌رسیم. در این مرحله باید توجه کنیم که حجم کاری این وظایف با هم برابر نبوده و وظایف آخری حجم کاری بیشتری دارند و باید به ارتباط هر وظیفه با وظیفه قبلی نیز توجه کنیم.

t_1 P_0	t_2 و t_3 P_1
t_4 P_2	t_5 P_3