Detalharemos agora cada módulo (função) do código-fonte:

→ aloc_int

Essa função irá alocar dinamicamente a matriz principal do programa. Isso significa dizer que ela irá ter um custo assintótico de espaço de $O(n^2)$, pois terá uma matriz *nxn* onde será alocado cada posição uma única vez.

Como o procedimentos para a alocação requer apenas uma passagem de *0* a *n-1*, em tempo teremos então, o custo de *O(n)*. (Obs.: a matriz é alocada com "calloc", isso significa que ela já está preenchida inteiramente com o valor "0").

→ free_aloc_int

Essa função irá deslocar a matriz principal do programa, para isso teremos o processo inverso descrito em "aloc_int", então em tempo também teremos *O(n)*, como não há nenhum espaço alocado nesse módulo, o custo será constante isso é, *O(1)*.

→ rlc_binary

Essa função é o esqueleto de todo o código fonte, ela colhe todos os valores passados no arquivo em um vetor único *"numbers_arc" (*que possui um custo de espaço constante *O(1)*). Após isso ela irá preencher somente a primeira linha e primeira coluna da *"matriz_aloc"* (o custo dessa matriz em espaço já foi detalhado e definido em *"aloc_int"*, o custo de tempo será detalhado posteriormente) com os valores passados como argumento na primeira linha do arquivo.

Assim quando obtivermos esse vetor (*"numbers_arc"*), teremos os vetores *"par_eixo_x"* e *"par_eixo_y"* onde ficarão respectivamente os valores de x e y de forma sincronizada, ou seja, a posição 0 dos dois vetores terá exatamente o primeiro par da entrada de dados, esses dois vetores acima terão custo de tempo e espaço respectivamente da ordem de *O(1)* e *O(n)*, já que foram alocados dinamicamente.

Por fim, os vetores *"par_eixo_x"* e *"par_eixo_y"* irão percorrer toda a matriz *"matriz_aloc"* computando os pares e preenchendo-os com o valor "1".

Note que para o preenchimento dessa matriz, é necessário passar *n vezes* em cada posição da matriz *nxn*, logo o custo assintótico de tempo gasto nessa parte será $O(n^3)$.

Para os módulos a seguir use:

Caso alguma das condições que serão descritas abaixo for *falsa*, ou seja, quando a condição descrita não é satisfeita será alocado os pares faltantes, que impedem a relação de ser verdadeira, em uma variável chamada *"vetor_resposta"* de tamanho constante que será detalhado no módulo *"main"*. Tal computação é realizada em tempo de execução durante a verificação da condição por isso não é gasto um tempo mais do que já está sendo consumido. Caso o módulo não realize essa computação será descrito no mesmo.

→ reflexiva

Definição: $R \text{ é reflexiva} \Leftrightarrow \forall\, x \text{ em } A\,,\ (x,x) \in R$

Essa função verifica se a diagonal principal está *inteiramente preenchida* com o valor "1".

Como é necessário verificar cada posição da matriz nxn, teremos um custo assintótico de tempo representado por $O(n^2)$ e como não há nenhuma alocação teremos custo constante de espaço, ou seja, *O(1)*.

→ irreflexiva

Definição: $R \text{ é irreflexiva} \Leftrightarrow \forall\, x \text{ em } A\,,\ (x,x) \notin R$

Essa função verifica se a diagonal principal possui algum valor *diferente* de "1".

Custo assintótico de tempo e espaço já detalhados em *"reflexiva"*.

→ simetrica

Definição: $R \text{ é simetrica} \Leftrightarrow \forall\, x \text{ e } y \text{ em } A,\ se\,(x,y) \in R,\ então\,(y,x) \in R$

Essa função verifica se o elemento da *"[i][j] == [j][i]"* para toda a matriz.

Custo assintótico de tempo e espaço já detalhados em *"reflexiva"*.

→ anti_simetrica

Definição: $R \text{ é anti-simetrica} \Leftrightarrow \forall\, x \text{ e } y \text{ em } A,\ se\,(x,y) \in R \text{ e } (y,x) \in R,\ então\ x = y$

Essa função verifica se o elemento da *"[i][j] != [j][i]"* a menos que *i e j sejam iguais*.

Custo assintótico de tempo e espaço já detalhados em *"reflexiva"*.

→ assimetrica

Definição: $R \text{ é assimetrica} \Leftrightarrow \forall\, x \text{ e } y \text{ em } A,\ se\,(x,y) \in R,\ então\,(y,x) \notin R$

Essa função verifica se o elemento da *"[i][j] != [j][i]"* . Se *i = j*, então não pode haver conexão. Esse bloco apenas retornar "F" ou "V", pois pela definição já dada em *"assimetrica"* quando essa relação for falsa, seus pares ordenados que fariam a sentença verdadeira já foram reproduzidos nos blocos *"irreflexiva"* e *"antissimétrica"*.

Nesse caso, como há somente verificação de condicionais, custo de tempo e espaço são constantes nesse caso o custo assintótico para ambos é de *O(1)*.

→ transitiva

Definição: $R \text{ é transitiva} \Leftrightarrow \forall\, x, y, z \text{ em } A,\ se\,(x,y) \in R,\ e\ (y,z) \in R, então\,(x,z) \in R$

Essa função verifica se *"[i][j] == 1 && [j][z] == 1 && [i][z] == 1"*

Como precisamos analisar as posições *x,y, z* em uma matriz *nxn* é necessário passar *n vezes* por toda a matriz, isso significa que teremos um custo de *nxnxn* o que pode ser representado por $O(n^3)$ em relação ao tempo e como o espaço ocupado é constante temos *O(1)*.

Por fim, como a minha entrada é falsa a disposto da relação transitiva, incluiremos os valores que a tornariam verdadeira para que a saída esperada em *"fecho transitiva"* (que será detalhada em breve) esteja correta, isso novamente ocasiona o uso de uma estrutura de repetição que possui um custo de $O(n^2)$.

→ imprimir_resposta

Essa função irá imprimir todas as respostas necessárias para que as relações descritas acima sejam verdadeiras, isso é feito com um custo assintótico de ordem *O(n)* em tempo e como não há espaço ocupado temos *O(1)*.

→ main

A função main irá efetuar os procedimentos para que a saída esperada seja executada, isto é, analisar se a entrada de dados corresponde a uma das relações propostas no trabalho. Caso seja falso, ela irá imprimir os pares ordenados que seriam capazes de tornar aquela entrada verdadeira.

Para isso iremos alocar estaticamente um vetor chamado *"vetor_resposta"* que terá a função de salvar respostas resultantes caso a relação descrita seja falsa. Como esse vetor independe da entrada de dados, ele terá um custo assintótico de espaço de *O(1)*, assim como de tempo.

Após isso, iremos analisar as relações descritas acima verificando se é verdadeira ou falsa, isso é feito por meio de estruturas condicionais que possuem um custo assintótico de tempo e espaço de *O(1)* por serem constantes. Caso seja falsa, iremos utilizar a variável *"vetor_resposta"* para imprimir os pares que seriam capazes de confirmar a relação, o processo de impressão é feito na função *"imprimir_resposta"* já detalhado.

A seguir temos relação de equivalência e ordem parcial que por definição, relação de equivalência é falsa quando meu conjunto de dados não é transitivo ou reflexiva ou simetrica, ou simultameamente as três relações. Já para ordem parcial, temos falsa quando não é transitiva ou

reflexiva ou antissimétrica. O custo assintótico de tempo e espaço desse bloco novamente por usar apenas estruturas condicionais é de *O(1)*.

Finalmente temos o *"fecho transitivo"*, ela sempre será impressa entretanto com um custo de $O(n^2)$ em tempo, quando minha entrada é transitiva, a impressão será de todos os pares ordenados da minha entrada. Caso seja negativa, analisaremos quando a expressão *"[i][j] == 1 && [j][z] == 1 && [i][z] != 1"* é satisfeita gerando o mesmo custo assintótico de tempo. Isso é feito após a impressão de todos os pares ordenados marcados como "1" na matriz *"matriz_int_aloc"*, lembre-se que como foi dito, quando a *"transitiva"* for falsa incluiremos ao final da impressão o valor de *"1"* para todos os pares impressos. O processo de repetir a expressão usada na *"transitiva"* é explicado pois podemos ter pares transitivos formados a partir da união desses casos, vejamos um exemplo abaixo:

4 1 2 3 4	Pares ordenados: (1,2) ; (2,3); (3,4)
1 2	Transitividade: (1,3) ; (2,4)
2 3	
3 4	

Note que pela definição da relação transitiva como faz parte da minha solução os pares (1,3) e (3,4), pela definição (1,4) também deverá parte da solução, logo a justificativa para que o passo dado no bloco acima seja refeito, isso eliminará a possibilidade de algum par estar ausente, consumindo $O(n^3)$ em tempo.

Então, concluímos que o custo assintótico no pior caso do código-fonte *"relacao.c"* de tempo e espaço respectivamente é da ordem de $\boldsymbol{O(n^3)}$, $\boldsymbol{O(n^2)}$.