Exercício 48. Considere o seguinte problema computacional de decisão: *Dadas uma Máquina de Turing* $\boldsymbol{M}$ *e uma entrada* $\boldsymbol{x}$ *para* $\boldsymbol{M}$*, a computação de* $\boldsymbol{M}$ *para* $\boldsymbol{x}$ *usa todos os estados de* $\boldsymbol{M}$*?* Este problema é decidível? Justifique sua resposta.

Para provar que este problema não é decidível, será demonstrado que se ele o fosse, o problema da parada também seria.

Suponhamos que o problema seja decidível. Sendo assim, dada uma máquina M e uma entrada x, deve haver uma máquina M' que retorna "sim"caso M utilize todos os estados para computação de x ou "não"caso contrário. Para isso M' deverá emular M para entrada x (ou de algum outro modo "entender o comportamento"de M para x).

Sendo assim, temos que se M não para para entrada x sem utilizar todos os estados (e.g. fica "presa"em um estado q_i, sem ter passado antes por um estado q_j, sempre andando com o cabeçote para direita), a máquina M' deveria detectar essa ocorrência e retornar "não", caso contrário M' não para também. Porém sabemos que o problema da parada não é decidível, logo essa detecção não é possível.

Portanto, por contradição, M' não existe e o problema não é decidível.

Exercício 58. Considere o problema complementar ao Problema da Primalidade, i.e. o problema de decidir se um dado inteiro não-negativo n não é primo. Mostre de duas formas que este problema está em NP dua duas formas: exiba um algoritmo não-determinístico polinomial, e também exiba um verificador polinomial para o problema.

Algoritmo não determinístico (verdadeiro é primo, falso não é primo):

DecideÉPrimo_NãoDeterminístico(x):

1: **if** $x < 2$ **then**
2: **return false**
3: **else if** $x == 2$ **then**
4: **return true**
5: **end if**
6: *divisor* $\leftarrow$ natural $y \in [2, x)$ determinado não deterministicamente
7: **if** $x \pmod{divisor} == 0$ **then**
8: **return false**
9: **else**
10: **return true**
11: **end if**

O tempo do algoritmo é polinomial, tendo em vista que as linhas 1, 3 e 6 são O(1) e a linha 7 tem complexidade polinomial (operador resto, não sei a complexidade exata)

Verificador se é primo ou não (verdadeiro é primo, falso não é primo), considerando o divisor como um natural não negativo

---

VerificadorÉPrimo(x, *divisor*):

1: **if** $x < 2$ **or** $x ==$ *divisor* **then**
2: **return false**
3: **end if**
4: **if** $x \pmod{divisor} == 0$ **then**
5: **return false**
6: **else**
7: **return true**
8: **end if**

A complexidade do verificador é polinomial, pela mesma argumentação anterior.

Para o complemento da linguagem, tendo em vista que ela é decidível, basta apenas inverter o retorno das funções

Exercício 59. Mostre que *2SAT* está em **P**.

Lembrando que o problema *SAT* é definido pela decisão de, dada uma expressão booleana na formal normal conjuntiva (CNF), se ela possui ou não uma valoração tal que a expressão é verdadeira. O *2SAT* faz a restrição tal que todas as clausulas da CNF tenham dois literais.

Para mostrar que *2SAT* está em **P**, devemos escrever um algoritmo que decide *2SAT* em tempo polinomial.

A ideia do algoritmo é criar um grafo orientado de relações entre os literais, de modo que se uma "contradição"nas arestas do grafo for encontrada, a expressão não é satisfazível.

Para isso, podemos observar que todas as operações do tipo $x + y$ (usarei . para o "e"e + para o "ou") podem ser escritas como $(\neg x \to y).(\neg y \to x)$, onde a função $\to$ é a relação de implicação definida por:

$\to$: (F, F) = V; (F, V) = V; (V, F) = F; (V, V) = V

Dessa maneira, para toda clausula, será adicionado no grafo um vértice emulando a relação de implicação entre os literais da clausula. Por exemplo, a CNF

$(a + b).(b + c).(\neg a + \neg c)$

Resultaria nas relações

$\neg a \to b \quad \neg b \to a$
$\neg b \to c \quad \neg c \to b$
$a \to \neg c \quad c \to \neg a$

Os nós V do grafo seriam $\{a, \neg a, b, \neg b, c, \neg c\}$ e as arestas E seriam $\{(\neg a, b), (\neg b, a), (\neg b, c), (\neg c, b), (a, \neg c), (c, \neg a)\}$.

O grafo deve ser lido como uma série de implicações do tipo "se a é verdadeiro, então $\neg c$ também deve ser verdadeiro"para o caso da relação $a \to \neg c$. Isso é verdade para todo o caminho de implicações do grafo ("já que $\neg c$ deve ser verdadeiro, então b também deve ser verdadeiro pela relação $\neg c \to b$").

Logo podemos chegar a uma contradição caso um vértice x seja alcançável a partir de $\neg x$ e também x seja alcançável a partir de $\neg x$ (i.e. x e $\neg x$ são vértices fortemente conexos), tendo em vista que tanto x quanto $\neg x$ devem ser verdadeiros.

Assim, uma condição necessária para *2SAT* ser satisfazível é: para toda variável x, os vértices x e $\neg x$ não devem ser fortemente conexos.

Para demonstrar que essa condição também é suficiente, faremos uma valoração dos literais baseado no grafo, tendo como premissa que a condição acima é satisfeita.

Antes disso, devemos computar os componentes fortemente conexos (CFC) do grafo (conjunto de vértices que são todos fortemente conectados entre si). Assim podemos ordenar esses conjuntos topologicamente (i.e. ordCFC[u] $\leq$ ordCFC[w] se se há um caminho de u para w, caso u e w não estejam desconexos), fazendo ordCFC[x] o índice do CFC que o vértice x está.

Essa ordenação é necessária para definir a ordem de valoração, tendo em vista que se x for alcançável a partir de $\neg x$, o contrário não pode ser verdade e vice-versa. Assim devemos primeiro valorar x e só depois $\neg x$.

Sendo assim, para toda variável x da CNF, se ordCFC[x] < ordCFC[$\neg x$], então fazemos $\neg x = V$, caso contrário, $x = V$ (caso ordCFC[x] == ordCFC[$\neg x$], a hipótese é violada). Para demonstrar que com essa valoração nenhuma contradição é encontrada no grafo, suponhamos que $x = V$ (o mesmo pode ser feito analogamente para $\neg x = V$):

Temos que $\neg x$ não é alcançável a partir de x, pois ordCFC[x] < ordCFC[$\neg x$], logo não há a contradição direta em x.

Outro modo de haver uma contradição, seria se dois vértices y e $\neg y$ fossem alcançáveis a partir de x, assim tanto y quanto $\neg y$ devem ser verdadeiros. Caso haja essa alcançabilidade, $\neg x$ também é alcançável a partir de y e $\neg y$, pela propriedade do grafo de implicações de que, se $x \rightarrow y$, então também $\neg y \rightarrow \neg x$ (pelos pares de relações de implicação que são adicionadas ao grafo). Sendo assim, por transitividade x é alcançável a partir de $\neg x$, contradizendo a hipótese.

Assim foi demonstrada que a condição dada anteriormente é não só necessária como suficiente para mostrar que um problema *2SAT* é satisfazível.

ANÁLISE COMPLEXIDADE TEMPORAL:

A construção do grafo é linear no tamanho da entrada, sendo n o número de literais e m o número de clausulas, o número de vértices gerado é $2n$ e de arestas no máximo $4m$.

O tempo para a ordenação topológica de um grafo G=(V,E) é O($|V| + |E|$). A verificação da condição para satisfatibilidade é O($|V|$) após a ordenação topológica, assim como a valoração dos literais.

Assim sendo, a complexidade do algoritmo é O($|V| + |E|$)=O($2n + 4m$) = O($n+m$). Logo *2SAT*$\in$**P**

EXEMPLO:

Dada a CNF

$(a + b).(b + c).(\neg a + c)$

Temos o grafo de implicações

G = (V, E) = ({a, $\neg a$, b, $\neg b$, c, $\neg c$}, {($\neg a$, b), ($\neg b$, a), ($\neg b$, c), ($\neg c$, b), (a, c), ($\neg c$, $\neg a$)}). Ordenando topologicamente os vértices temos

ordCFC[c]=1, ordCFC[a]=2, ordCFC[$\neg b$]=3,
ordCFC[b]=4, ordCFC[$\neg a$]=5, ordCFC[$\neg c$]=6

Tendo em vista que nenhum par de vértices x e $\neg x$ ficou no mesmo índice, a expressão deve ser satisfazível.

Assim sendo, valoramos os literais com base no índice de menor para maior, fazendo $c = V$, $a = V$, e então $\neg b = V$. Com isso temos uma valoração que satisfaz a expressão booleana.

$(V + V).(F + V).(F + V) = V$

Exercício 60. Mostre que o problema de decidir se um grafo possui ciclo hamiltoniano é **NP**-completo.

Para um problema ser **NP**-completo, ele deve:

1. O problema deve pertencer a classe **NP**

2. O problema deve ser **NP**-difícil (i.e. qualquer problema da classe **NP** deve ser reduzível a ele).

Para demonstrar que o problema do ciclo hamiltoniano num grafo direcionado é **NP**-completo, tomarei como presunção que é sabido que o 3SAT é **NP**-completo e então farei a redução do 3SAT para ele. Além disso, irei descrever um verificador em tempo polinomial para uma solução do problema.

VERIFICADOR

A variável *caminho* é um vetor de tuplas, em que o primeiro valor é o vértice de origem e o segundo o vértice de chegada. Também a notação de índices da linguagem C é usada, assim como "[ ]"para indicar uma lista vazia e as pseudo funções add(vetor, valor) e pop(vetor, índice).

VerificadorCicloHamiltoniano(G=(V,E), *caminho*):

```
1: if |caminho| != |V| then
2:    return false
3: end if
4: first ← caminho[0][0]
5: next ← caminho[0][0]
6: visited ← [ ]
7: while caminho não for vazio do
8:    v ← caminho[0][0]
9:    u ← caminho[0][1]
10:   if (v, u) ∉ E then
11:      return false
12:   end if
13:   if v ∈ visited or v != next then
14:      return false
15:   end if
16:   if u == first then
17:      return true
18:   end if
19:   next ← u
20:   add(visited, v)
21:   pop(caminho, 0)
22: end while
```

O algoritmo primeira checa se o tamanho do caminho é válido, tendo em vista que deve se passar exatamente por $|V|$ arestas para fechar um ciclo hamiltoniano. Após isso, vai removendo os nós do caminho em sequência e validando-o, checando

se a aresta é valida, então checando se o vértice de origem já foi visitado e se era o vértice de destino da aresta anterior. Caso isso seja validado, os valores dos vértices já visitados e o valor de qual deve ser o próximo vértice de origem são atualizados, assim como a aresta removida do *caminho*. A condição de parada é quando o vértice de destino é o mesmo do primeiro vértice de origem, fechando assim o ciclo hamiltoniano.

As linhas 1 e 7 são executadas $|V|$ vezes. A linha 12 tem limite superior de $|V|-1$ execuções em uma iteração (caso |visited| == $|V|-1$), a linha 10 tem limite superior de $|E|$ para cada iteração e todas as outras tem complexidade O(1) (as operações de remover e adicionar tem O(1) caso seja mantida uma lista encadeada, tendo em vista que operam apenas no primeiro ou último elemento).

Sendo assim o verificador tem complexidade de tempo de $O(|V||E|)$, que está contido em $O(V^3)$. Logo o verificador é em tempo polinomial, portanto o problema do ciclo hamiltoniano é **NP**.

REDUÇÃO

Para a redução, primeiro é necessário definir o conceito de *gadget* ao se transformar o domínio de um problema para o outro. No caso do 3SAT para o ciclo hamiltoniano, os *gadgets* serão mecanismos ou padrões utilizados para transformas tanto os literais quanto as clausulas das expressões em nós e arestas do grafo, de modo que, a expressão seja satisfazível se e somente se o grafo possuir um ciclo hamiltoniano.

O primeiro *gadget* será utilizado para gerar vértices e arestas a partir dos literais e segue na figura 1.

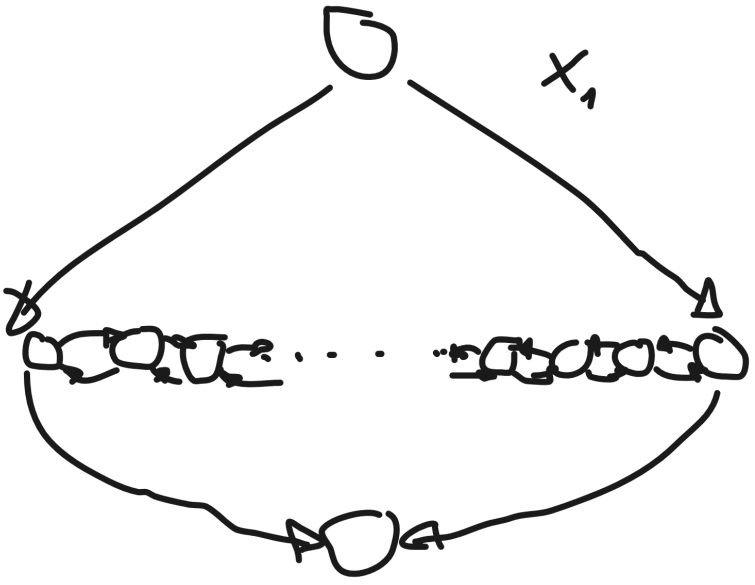

Figura 1: *Gadget* dos literais

O gadget dos literais consiste em um nó "fonte"e um nó "de chegada", estando entre eles um número suficiente de nós intermediários que são ligados ao seu adjacente (ida e volta). O número necessário de nós intermediários será especificado mais tarde.

É possível perceber que nos nós intermediários, para haver um caminho hamiltoniano (irei me referir a caminho hamiltoniano, tendo em vista que para obter o grafo do ciclo a partir do grafo do caminho é trivial ao final da demonstração), o caminho deve ir apenas para esquerda ou apenas para direita, caso contrário, o nó é repetido no caminho e não há caminho hamiltoniano.

A ideia do *gadget* é que se uma variável for verdadeira, o caminho percorrido nos nós intermediários deve ser da esquerda para direita (por convenção), caso contrário deve ser da direita para esquerda. Esse *gadget* preserva a propriedade do literal ser verdadeiro ou falso.

O *gadget* das clausulas irá seguir uma ideia semelhante de, se a clausula c_i é validada por um literal x_j quando este é verdadeiro, haverá uma aresta partindo de um nó intermediário de x_j, chegando no vértice de c_i e então voltando ao nó intermediário adjacente a direita àquele a partir de c_i.

Caso o literal em c_i seja $\neg x_j$, então a direção das arestas é invertida (parte do nó a direita e chega naquele a esquerda). Esse *gadget* é esquematizado na figura 2, exemplificando o caso em que a clausula c_j possui x_n e a clausula c_i possui $\neg x_n$.

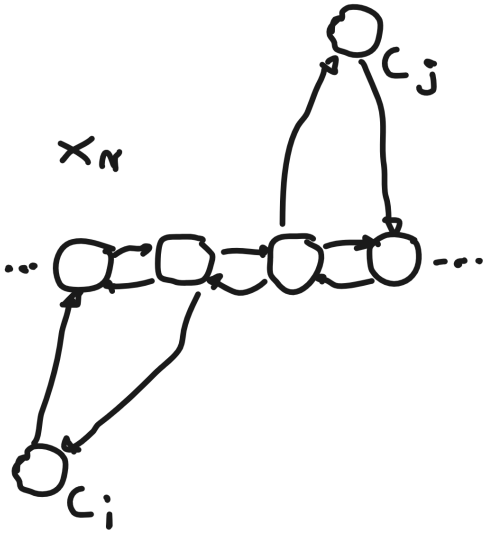

Figura 2: *Gadget* das clausulas

Vale lembrar que, nesse caso, é possível perceber que os nós intermediários de x_n não podem fazer seu caminho passando em c_j e em c_i, pois isso quebraria a propriedade do caminho hamiltoniano como argumentado anteriormente. Portanto,

no caso de x_n ser verdadeiro, por exemplo, a clausula c_i deverá ser alcançada a partir dos nós intermediários de outro literal. Caso as clausulas já tenham sido validadas (i.e. já estiverem no caminho) basta "ingorá-las"e continuar o caminho pelos nós intermediários normalmente.

A partir desses dois *gadgets* é possível estabelecer relações e propriedades deles para se certificar de que haja um ciclo hamiltoniano no grafo se e somente se a expressão for satisfazível. No algoritmo a seguir estão implementados os *gadgets* e tambés essas restrições e relações entre eles. (Obs.: todas as operações de criação e adição de nós e arestas no algoritmo são sobre o grafo G).

Redutor3SATparaCicloHamiltoniano(Φ):

1: n $\leftarrow$ número de literais em Φ
2: k $\leftarrow$ número de clausulas em Φ
3: G(V, E) $\leftarrow$ ((), ())
4: **for** i em [0, n) **do**
5: Crie 2k nós intermediários $x_{i,j}$
6: Crie o nó $x_{i,source}$
7: Adicione os vértices $(x_{i,source}, x_{i,0})$ e $(x_{i,source}, x_{i,2k-1})$
8: Crie o nó $x_{i,end}$
9: Adicione os vértices $(x_{i,0}, x_{i,end})$ e $(x_{i,2k-1}, x_{i,end})$
10: **for** j em [0, 2k-1) **do**
11: Adicione os vértices $(x_{i,j}, x_{i,j+1})$ e $(x_{i,j+1}, x_{i,j})$
12: **end for**
13: **if** i == n-1 **then**
14: Adicione o vértice $(x_{n-1,end}, x_{n-1,source})$
15: **else if** i != 0 **then**
16: Adicione o vértice $(x_{i-1,end}, x_{i,source})$
17: **end if**
18: **end for**
19: **for** j em [0, k) **do**
20: Crie o nó c_j
21: **for** cada literal x_i da clausula **do**
22: **if** $\neg x_i$ esta na clausula c_j **then**
23: Crie as arestas $(x_{i,2*j+1}, c_j)$ e $(c_j, x_{i,2*j})$
24: **else**
25: Crie as arestas $(x_{i,2*j}, c_j)$ e $(c_j, x_{i,2*j+1})$
26: **end if**
27: **end for**
28: **end for**
29: **return** G(V, E)

O *for* da linha 4 adiciona os *gadgets* dos literais, além de conectar o nó do fim de um com o inicial do próximo e o último nó final com o nó inicial, para fechar o ciclo hamiltoniano. Sua complexidade é de O($|n|$), sendo n o número de literais da expressão.

O *for* da linha 19 adiciona os *gadgets* das clausulas, mantendo a importante

propriedade de que o único modo de que duas clausulas não compartilham o mesmo nó (linhas 23 e 25). Sendo assim o único modo de validar (i.e. não violar as propriedades do caminho hamiltoniano) um *gadget* de literal passando por um *gadget* de clausula é passando pela clausula e já retornando para o nó intermediário do literal de onde se veio (i.e. caso o caminho siga para um nó de outro literal, não há caminho hamiltoniano pois não haverá modo de alcançar o nó "de retorno"da clausula sem quebrar a propriedade de validação do *gadget* do literal). Isso faz com que, caso a expressão não seja satisfazível, não é possível validar todos os *gadgets* de clausula, ou seja, não há ciclo hamiltoniano.

A complexidade desse *for* é O($|m|$), sendo m o número de clausulas da expressão.

Portanto, como argumentado, há um caminho hamiltoniano no grafo se e somente se a expressão booleana for satisfazível. Sendo a expressão valorada de modo a seguir os caminhos dos nós intermediários do literal no ciclo hamiltoniano (esquerda para direita verdadeira, direita para esquerda falso).

Sendo assim, a complexidade do redutor é O($|m| + |n|$), portanto o SAT3 é reduzido em tempo polinomial para o problema do ciclo hamiltoniano, logo podemos afirmar que este é **NP**-completo.

EXEMPLO

Para a CNF (utilizando "."para "e"e "+"para "ou")

$(x_1 + x_2 + \neg x_3).(\neg x_1 + x_2 + \neg x_3).(x_1 + \neg x_2 + x_3)$

O grafo resultante é apresentado na figura 3

Sendo a expressão satisfeita pela valoração $x_1 = V$, $x_2 = V$, $x_3 = F$, o ciclo hamiltoniano se inicia em $x_{1,s}$, vai da esquerda para direita nos seus nós intermediários, passando por c_1 e c_3, então nos nós intermediários de x_2 faz o mesmo passando por c_2 e então em x_3 apenas vai da esquerda para direita, sem passar pelo nó de nenhuma clausula e ao final retornando de $x_{3,t}$ para $x_{1,s}$, formando um ciclo hamiltoniano.

O contrário também poderia ser feito, encontrar um ciclo hamiltoniano no grafo e a partir disso valorar a expressão, porém isso é mais difícil de fazer e explicar visualmente (i.e. rodar algoritmo "no olho"não é fácil).

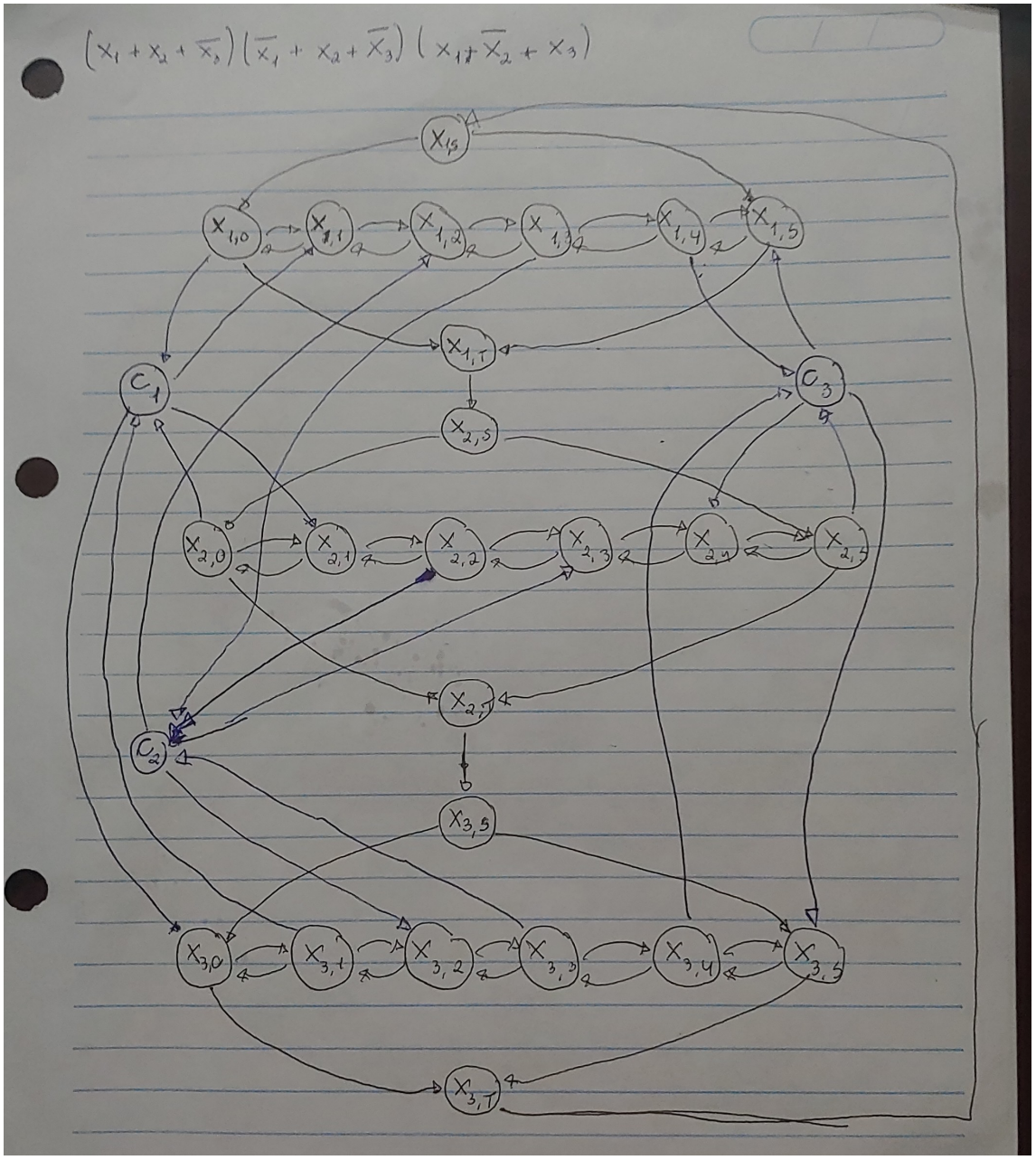

Figura 3: Exemplo de grafo gerado

Exercício 61. Seja **2COLOUR** o problema de, dado um grafo simples **G**, decidir se os vértices de **G** podem ser coloridos usando no máximo 2 cores de modo que vértices adjacentes não recebam a mesma cor. Mostre que **2COLOUR** está em **P**.

Para demonstrar que **2COLOUR** está em **P**, faremos a redução do problema para o **2SAT** em tempo polinomial, tendo em vista que sabemos que este está em **P** (exerc. 59).

O *gadget* da redução é feito notando que os vértices conectados do grafo devem ter cores diferentes, o que é diretamente traduzido para a função booleana XOR. Este pode ser representado por meio da CNF abaixo (usando "."para "e"e "+"para "ou").

XOR(x_1, x_2): $(x_1 + x_2).(\neg x_1 + \neg x_2)$

Sendo assim, o grafo satisfaz **2COLOUR** se e somente a expressão **2SAT** criada a partir das arestas do grafo for satisfazível, com a valoração verdadeira representando uma cor para os nós do grafo e a falsa representando outra. O algoritmo é descrito a seguir

---

Redutor2COLOURpara2SAT($G = (V, E)$):

1: $\Phi \leftarrow$ V
2: **for** cada aresta (v_i, v_j) em E **do**
3: $\quad \Phi \leftarrow \Phi.(v_i + v_j).(\neg v_i + \neg v_j)$
4: **end for**
5: **return** Φ

A complexidade do algoritmo é O($|E|$), portanto a redução é polinomial. Sendo assim, **2COLOUR** está em **P**.

Exercício 62. Seja **3COLOUR** o problema de, dado um grafo simples **G**, decidir se os vértices de **G** podem ser coloridos usando no máximo 3 cores de modo que vértices adjacentes não recebam a mesma cor. Mostre que **3COLOUR** é **NP**-completo.

Para demonstrar que **3COLOUR** é **NP**-completo, faremos a redução do SAT para ele em tempo polinomial, tendo em vista que sabemos que **SAT** é **NP**-completo. Além disso também iremos mostrar um verificador em tempo polinomial para uma solução do problema.

VERIFICADOR

O algoritmo abaixo recebe um grafo não direcionado e um vetor com a cor para cada vértice do grafo (no máximo 3 cores diferentes) e retorna se a coloração do grafo é valida ou não.

Verificador3COLOUR($G = (V, E)$, V_{cores}):

1: **if** $|V_{cores}|$!= $|V|$ **or** número de cores em V_{cores} maior que 3 **then**
2: **return false**
3: **end if**
4: **while** V não for vazio **do**
5: $v_i \leftarrow$ primeiro vértice de V
6: $c_i \leftarrow$ cor do vértice v_i em V_{cores}
7: **for** cada aresta u em E com v_i **do**
8: $v_j \leftarrow$ vértice em u que não é v_i
9: $c_j \leftarrow$ cor do vértice v_j em V_{cores}
10: **if** $c_j == c_i$ **then**
11: **return false**
12: **end if**
13: Remover u de E
14: **end for**
15: Remover v_i de V
16: Remover cor de v_i de V_{cores}
17: **end while**
18: **return true**

A linha 1 e a 4 são executadas $|V|$ vezes. Já a linha 7 é executada $|E|$ vezes para uma representação por lista de adjacência, por exemplo. A linha 9 tem complexidade $O(|V|)$ caso não haja nenhuma ordenação, sendo as outras linhas no máximo $O(1)$.

Portanto a complexidade do verificador é $O(|V|^2 + |E|)$, logo o algoritmo tem complexidade polinomial. Portanto **3COLOUR** está em **NP**.

REDUÇÃO

Os *gadgets* que serão utilizados transformarão as propriedades das portas NOT e OR em propriedades de satisfazibilidade do **3COLOUR**. Com essas portas é

possível construir todas outras, como AND, XOR, etc. O AND por exemplo pode ser escrito (utilizando "+"como OR) como

$\text{AND}(x_1, x_2)$: $\neg(\neg x_1 + \neg x_2)$

Para a construção dos *gadgets* será utilizado um "triângulo de cores"no grafo, com 3 vértices, um representando o neutro (N), outro representando a valoração verdade (V) e outro a valoração falsa (F). O grafo será construído de modo que essas sejam as três cores utilizadas e que, se a expressão for satisfazível, o grafo pode ser colorido com 3 cores (com a cor do vértice V ou F representando sua valoração na expressão).

O *gadget* da negação é apresentado na figura 4, assim como o "triângulo de cores". É possível notar que x_1 deve ser colorido com V ou F e $\neg x_1$ com o complementar, caso contrário a coloração não será válida. Vale também notar que todos os literais e resultados de expressões devem ser ligados ao nó N, para garantir que tenham coloração V ou F.

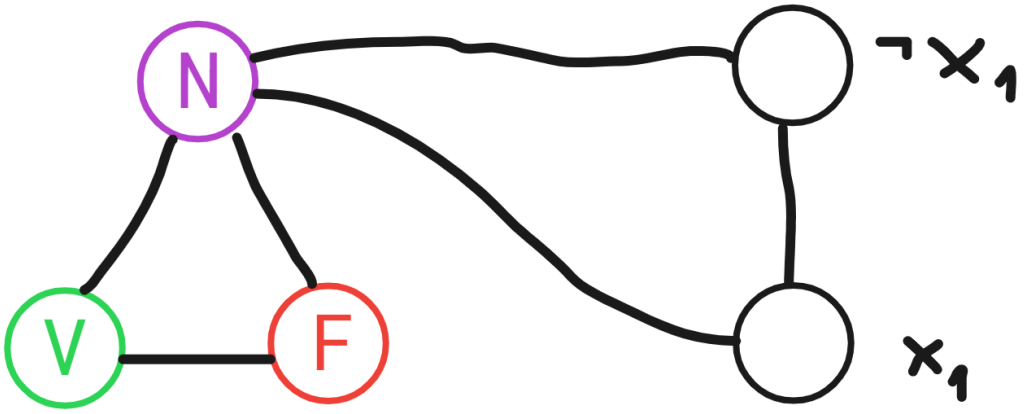

Figura 4: *Gadget* da negação

O *gadget* do OR é apresentado na figura 5. As arestas em roxo representam as ligações para manter as propriedades de V e F dos literais ou expressões. É possível perceber que, se p e q forem falsos (F), os nós 2 e 3 deverão um ser V e outro N e, independente de qual terá qual valor, o nó $p + q$ será F. Assim o nó 1 deverá ser N e o nó 4 V, para finalizar a coloração corretamente.

O segundo caso é quando p e q são verdadeiros (V), os nós 2 e 3 deverão um ser F e outro N e, independente de qual terá qual valor, o nó $p + q$ será V. Assim o nó 1 deverá ser F e o nó 4 N (ou vice-versa), para finalizar a coloração corretamente.

O última caso é quando p é verdadeiro (V) e q é falso (F) (ou vice-versa), o nó 4 deverá ser N e o nó 1 F, consequentemente $p + q$ deverá ser V. Assim o nó 3 deverá ser N e o nó 2 F (para o caso de $p = V$ e $q = F$ ou vice-versa se os valores forem invertidos). É possível observar também que, caso o nó 1 fosse ligado ao F ao invés do V, a saída se tornaria $p.q$.

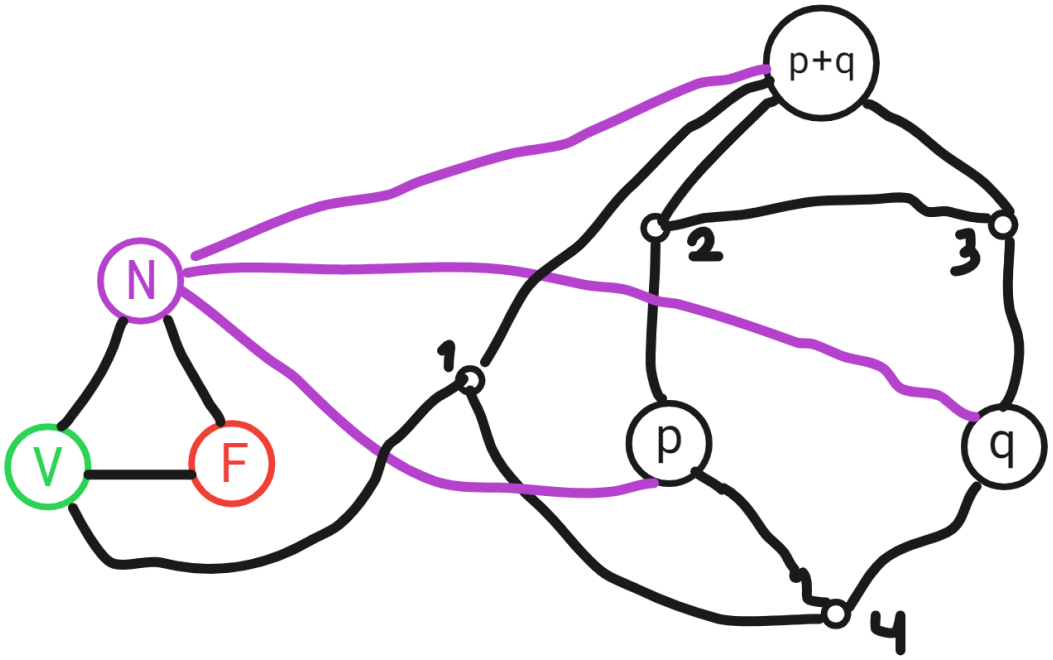

Figura 5: *Gadget* do OR

Com todos esses casos cobertos, é possível afirmar que o *gadget* se comporta corretamente para todas valorações de p e q. Importante lembrar que, para simular toda expressão, basta "ligar"o nó de saída de um *gadget* no nó de entrada do outro.

Por fim, para checar se a expressão é ou não satisfazível, basta ligar o nó final de saída da expressão ao N e ao F. Sendo assim, só será possível colorir o grafo se for possível colorir o resultado da expressão com V (i.e. validar a expressão).

A partir desses dois *gadgets* é possível reduzir o **SAT** para o problema **3COLOUR**, sendo que a expressão é satisfazível se e somente se é possível colorir o grafo com três cores.

A complexidade de tempo da redução é linear, tendo em vista que, após a criação dos vértices dos literais, basta converter as operações para OR e negação (linear) e criar as arestas e os nós dos gadgets (linear).

Portanto, com a redução e o verificador polinomiais, é possível afirmar que **3COLOUR** é **NP**-completo.

Exercício 63. Seja **G** um grafo simples. Uma clique em **G** é um conjunto não-vazio $X \subseteq V(G)$ tal que $uv \in E(G)$ para quaisquer $u, v \in V(G)$. A cardinalidade da maior clique em **G** é denotada por $\omega(G)$. Um conjunto independente em **G** é um conjunto não-vazio $X \subseteq V(G)$ tal que $uv \notin E(G)$ para quaisquer $u, v \in V(G)$. A cardinalidade do maior conjunto independente em **G** é denotada por $\alpha(G)$. Uma cobertura por vértices em **G** é um conjunto não-vazio $X \subseteq V(G)$ tal que, para toda aresta $e \in E(G)$, ao menos um dos extremos de e está em X. A cardinalidade da maior cobertura por vértices em **G** é denotada par $\tau(G)$. Seja **MAXCLIQUE** a versão em problema de decisão do problema de otimização de, dado um grafo **G**, encontrar $\omega(G)$. Seja **MAXIS** a versão em problema de decisão do problema de otimização de, dado um grafo **G**, encontrar $\alpha(G)$. Seja **MINVC** a versão em problema de decisão do problema de otimização de, dado um grafo **G**, encontrar $\tau(G)$. Mostre que os três problemas — **MAXCLIQUE**, **MAXIS**, e **MINVC** — são **NP**-completos.

Para demonstrar que todos os problemas são **NP**-completos, primeiro iremos mostrar que **MAXCLIQUE** o é e então fazer a redução desse para os outros, além de descrever um verificador polinomial para todos.

VERIFICADORES

Para verificar uma solução do **MAXCLIQUE**, recebendo um grafo $G = (V, E)$ e tamanho da clique k, basta fazer todas as combinações de subgrafos com k vértices e checar se algum desses subgrafos é uma clique. É possível perceber que todas essas verificações são $O(|V|^k k^2)$, tendo em vista que há no máximo $|V|^k$ subgrafos com k nós e k^2 arestas para checar a existência em cada subgrafo. Sendo assim, tendo em vista que k é uma constante, **MAXCLIQUE** está em **NP**.

Para verificar uma solução do **MAXIS**, recebendo um grafo $G = (V, E)$ e o tamanho do conjunto independente k, basta tomar o grafo complementar do recebido e fazer a mesma verificação do **MAXCLIQUE** com o tamanho mínimo da clique igual a k.

A argumentação é: se X é conjunto de vértices independentes de tamanho k de G, então no grafo complementar X deve formar uma clique de tamanho pelo menos k, tendo em vista que então haverá arestas entre todos os vértices de X. Caso o conjunto X não exista, não haverá clique de tamanho pelo menos k.

O complementar do grafo tem complexidade $O(|E|)$, logo a redução é polinomial e o tempo para verificação é o mesmo de **MAXCLIQUE**. Sendo assim **MAXIS** está em **NP**.

Para verificar uma solução do **MINVC**, recebendo um grafo $G = (V, E)$ e o tamanho da cobertura por vértices k, basta tomar o grafo complementar do recebido e fazer verificação do **MAXCLIQUE** para o tamanho máximo do conjunto independente de $|V| - k$.

A argumentação é: se S é a cobertura por vértices de tamanho k de G, então $S - V$ devem formar um conjunto independente, portanto ao tomar o grafo complementar, todos os vértices de $S - V$ devem ser conectados, formando uma clique de tamanho

pelo menos $|V| - k$. Caso S não exista, a clique de tamanho pelo menos $|V| - k$ também não existirá.

Pela mesma argumentação do **MAXIS** sobre complexidade, **MINVC** está em **NP**.

A argumentação para os problemas acima podem ser também "invertidas", levando a um "se e somente se"para todos esses problemas (e.g. existe uma clique de tamanho k em G se e somente se houver um conjunto independente de tamanho k no complementar de G). Portanto, como todas as reduções valem para os dois lados, se um dos três problemas for demonstrado **NP**-completo, todos o são.

REDUÇÃO MAXCLIQUE

Dado uma expressão **3SAT** com n literais e m clausulas, é possível reduzir o problema para **MAXCLIQUE** utilizando *gadgets* que transformar as clausulas e os literais em nós e arestas em G, fazendo com que a expressão seja satisfazível se e somente se houver uma clique de tamanho m no grafo G formado.

O *gadget* das clausulas cria nós rotulados como $x_{i,a,r}$ em que i é o número do literal, a o número da clausula e r o booleano do literal i na clausula a (i.e. $r = V$ se x_i na clausula j, $r = F$ se $\neg x_i$ na clausula j).

Após isso, são criadas as arestas entre os nós $x_{i,a,r}$ e $x_{j,b,t}$ (fazendo todas combinações possíveis) se:

1. $i \neq j$ e $a \neq b$
2. $i == j$ e $a \neq b$ e $r == t$

Vale notar que se dois nós estão conectados, então ambos podem ser assinalados verdadeiros simultaneamente (não há nenhum x conectado com $\neg x$). Sendo assim, dado uma clique no grafo, todas variáveis podem ser assinaladas verdadeiras ao mesmo tempo.

Assim, tendo em vista que só a ligações entre literais de clausulas diferentes, caso haja uma clique de tamanho m, ela possui exatamente um nó de cada clausula. Sendo assim, a expressão é satisfazível, bastando assinalar V para os literais dos nós da clique.

Por outro lado, se a expressão é satisfazível, basta tomar um literal de cada clausula que é verdadeiro e adiciona-lo a um conjunto de nós X. Ao final X deve formar uma clique de tamanho m, tendo em vista que todos os literais são V ao mesmo tempo (i.e. podem ser conectados entre si).

Portanto, a expressão é satisfazível se e somente se há uma m clique no grafo.

A redução é feita em tempo polinomial, tendo em vista que a criação do grafo é $O(m^2)$ (criar os vértices de cada clausula e para os 3 literais de cada clausula, criar os vértices com todas outras clausulas). (Obs.: tendo em vista que m é uma entrada para o problema, a complexidade do verificador do **MAXCLIQUE** não é polinomial, sendo $O(|V|^m m^2)$).

Exercício 66. Considere o seguinte problema: *Dado um conjunto U com $3m$ elementos e uma família $\mathcal{F}$ de n subconjuntos de U, cada qual com exatamente três elementos de U, é possível selecionar m subconjuntos mutualmente disjuntos em $\mathcal{F}$ de modo que a união de todos os subconjuntos selecionados seja igual a U?* Mostre que este problema é NP-completo.

Para mostrar que este problema é *NP*-completo, primeiro será demonstrado um verificador de tempo polinomial, mostrando que ele está em **NP**, e então a redução do **3SAT** para esse problema (3-dimensional matching).

Obs.: irei me referir aos conjuntos de $\mathcal{F}$ como triplas, pelo fato de haver três elementos neles.

VERIFICADOR

O verificador recebe uma instância do problema (U e $\mathcal{F}$) e também a solução X. Para verificação, basta checar se $X \subseteq \mathcal{F}$ e se todos os elementos de U ocorrem uma e somente uma vez em X, respeitando a disjunção mútua e a união de todos subconjuntos formarem U. Caso isso seja verdade, a solução é válida.

A complexidade do algoritmo é $O(|U||\mathcal{F}|)$ já que para checar $X \subseteq \mathcal{F}$ a complexidade é $O(|\mathcal{F}|)$ e checar para um elemento de U se ele ocorre uma e somente uma vez em X, a complexidade é $O(|U|)$, resultando em $O(|U||\mathcal{F}|)$ (tendo em vista que $|X| \leq |\mathcal{F}|$).

Portanto o problema descrito é **NP**.

REDUÇÃO

Dada uma instância do **3SAT**, expressão ϕ com n variáveis e m clausulas, a redução para o 3-dimensional matching (**3DM**) utiliza três *gadgets*. O primeiro *gadget*, relacionado à restrição das variáveis (V ou F), cria um conjunto de $2m$ triplas para cada variável x_i, como indica a figura 6, sendo que triplas "adjacentes"são definidas com cores diferentes, indicando se a variável é verdadeira ou falsa. Para cada variável devem ser selecionadas todas as triplas de uma cor.

Caso a variável da figura 6 seja verdadeira, as triplas em vermelho do *gadget* serão utilizados (par); caso seja falsa, as triplas em azul (ímpar). As outras triplas serão utilizadas para satisfazer as clausulas ou "descartadas", caso não seja necessário utilizá-la.

O segundo *gadget*, relacionado às clausulas, é esquematizado na figura 7. Ele cria três triplas para cada clausula, cada uma do tipo $\{c_i, \neg c_i, x_{kj}\}$ onde c_i e $\neg c_i$ são elementos da clausula, x_k o literal presente na clausula e $j = 2i$ se o literal é negado e $j = 2i - 1$ caso contrário. Com esse *gadget* é possível validar clausulas, tomando um dos literais e sua tripla com os elementos da clausula.

O terceito *gadget* trata dos elementos restantes, aqueles que não são utilizados mesmo satisfazendo todas as clausulas. É possível perceber que foram criados $2m$ elementos externos (x na figura 6) para cada variável e que m desses elementos já são utilizados ao definir a variável como verdadeira ou falsa. Porém, caso ele não esteja presente na clausula ou os elementos da clausula já tenha sido utilizados, ele deve ser utilizado por outra tripla. Também é possível perceber que serão utilizados apenas m

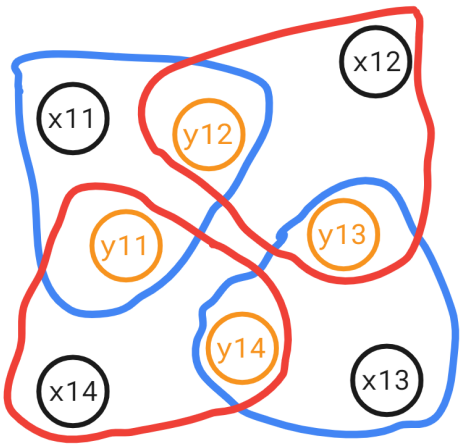

Figura 6: Exemplo de *gadget* para a variável x_1 e uma expressão com $m = 2$. Cada conjunto de elementos contornados representa uma tripla

elementos para validação das clausulas (um para cada clausula), portanto basta criar triplas para utilizar os elementos restantes. Tendo em vista que há nm elementos externos, basta criar $nm - m$ triplas novas, ligando todos os elementos externos à todos novos elementos do tipo $q_i, \neg q_i$, resultando em triplas do tipo $\{q_i, \neg q_i, x_{kj}\}$ para $i \in (1, nm - m]$ e todo x_{kj}.

Sendo assim, U é formado pelo conjunto de elementos criados e $\mathcal{F}$ pelo conjunto de triplas. Pelo modo como foram construídas as relações, é possível perceber que haverá um **3DM** se a expressão dada for satisfazível, tendo em vista que as restrições (variável é verdadeira ou falsa, clausula deve ser satisfazível por meio dos literais) e a consistência (os nós não utilizados pelas clausulas são utilizados em outra tripla) são traduzidas para o **3DM**.

É perceptível também o fato de que, se a fórmula não for satisfazível, a instância do **3DM** também não o será, já que não é possível utilizar todas as clausulas e manter as restrições dos *gadgets* das variáveis ao mesmo tempo.

Logo, a expressão é satisfazível se e somente se há um 3-dimensional matching.

Com relação à complexidade temporal, o passo de construção dos *gadgets* das variáveis é $O(mn)$, o passo de construção dos *gadgets* das clausulas é $O(m)$ e a criação das triplas finais é $O(mn(n(m-1)))$=$O(n^2m^2)$ (para cada elemento externo conectá-lo a $n(m-1)$ triplas).

Portanto a complexidade da redução é $O(n^2m^2)$, sendo assim **3-dimensional matching** é **NP**-completo.

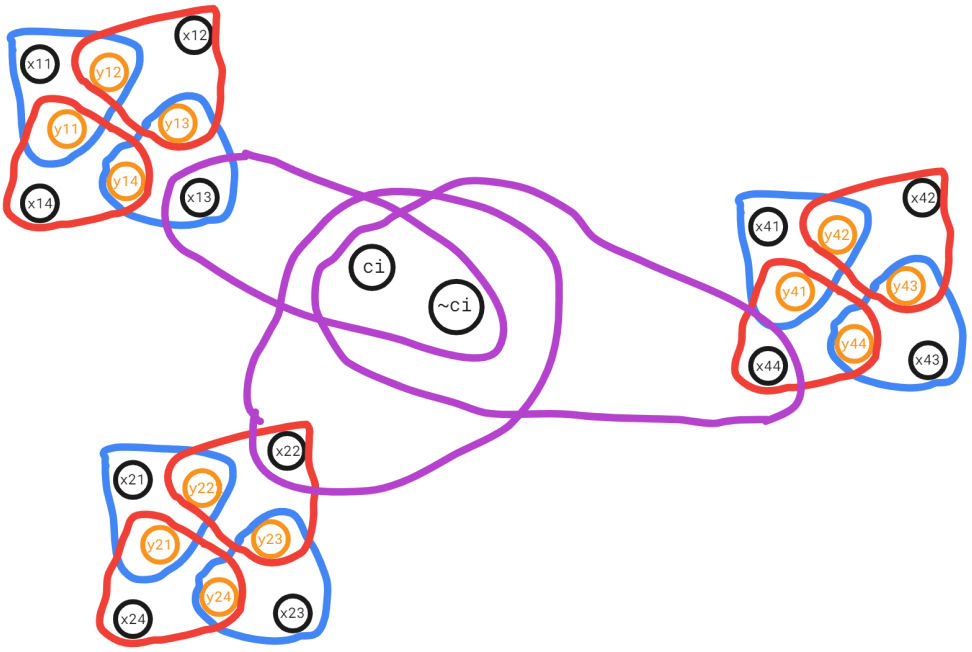

Figura 7: Exemplo de *gadget* para uma clausula $c_i = x_1 + \neg x_2 + \neg x_4$. A clausula é satisfeita se pelo menos um dos literais for verdadeiro.