UNIVERSIDADE FEDERAL DO MATO GROSSO – CAMPUS ARAGUAIA II
CIÊNCIA DA COMPUTAÇÃO

INDEPENDENT SET PARA VERTEX COVER

BARRA DO GARÇAS
2023

ALDO TEIXEIRA DA SILVA JUNIOR
MARIA LUISE BRITTO
TAINÁ ISABELA MONTEIRO
WILLYAN JOSUÉ BASTOS

INDEPENDENT SET PARA VERTEX COVER

Trabalho de
Projeto e Análise de Algoritmos
sobre algoritmo do problema
Independent Set para Vertex
Cover.

Dr. Robson da Silva

BARRA DO GARÇAS
2023

SUMÁRIO

1. DEFINIÇÃO

Imagine um conjunto de salas interligadas por corredores. Um guarda postado numa sala é capaz de vigiar todos os corredores que convergem sobre a sala. Queremos determinar o número de guardas capazes de vigiar todos os corredores.
O problema do Vertex-Cover está em NP (classe de complexidade não determinística polinomial). Isso significa que, embora não tenhamos uma maneira eficiente de verificar a solução para o problema em tempo polinomial, podemos verificar em tempo polinomial se uma solução proposta é válida.

Em termos simples, o problema do Vertex-Cover pergunta se, dado um grafo não direcionado, existe um conjunto de vértices tal que cada aresta do grafo tenha pelo menos um de seus vértices no conjunto. Em outras palavras, é possível selecionar um subconjunto de vértices de modo que todos os vértices do grafo estejam conectados a pelo menos um vértice no subconjunto. Enquanto isso, o Independent Set pode ser definido pelo maior conjunto de vértices independente de um grafo.

Embora não tenhamos um algoritmo conhecido que resolva o problema do Vertex-Cover em tempo polinomial, podemos verificar se uma solução proposta é válida em tempo polinomial. Portanto, o problema está classificado em NP.

Sob essa perspectiva, foram feitas implementações de algoritmos em c + + a fim de provar cada caso específico dentro da classe de complexidade.

2. DETERMINANDO SE B'(Vertex-Cover) está em NP

Implementação de um algoritmo ***C(I,S)***, que recebe como parâmetro ***I*** que é uma instância do problema Vertex-Cover e ***S*** é uma solução para a instância ***I***. ***C*** deve verificar em tempo polinomial se ***S*** é uma solução para instância ***I*** do problema Vertex-Cover;

```cpp
bool isVertexCover(vector<pair<int, int>>& I, vector<int>& S) {
    // Verifica cada aresta em I
    for (const auto& edge : I) {
        int u = edge.first;
        int v = edge.second;

        // Verifica se nem u nem v estão presentes em S
        if (find(S.begin(), S.end(), u) == S.end() && find(S.begin(), S.end(), v) == S.end()) {
            cout << "Está não é uma solução válida para a instância I do problema Vertex-Cover." << endl;
            return false;
        }
    }
    cout << "Está é uma solução válida para a instância I do problema Vertex-Cover." << endl;
    return true;
}
```

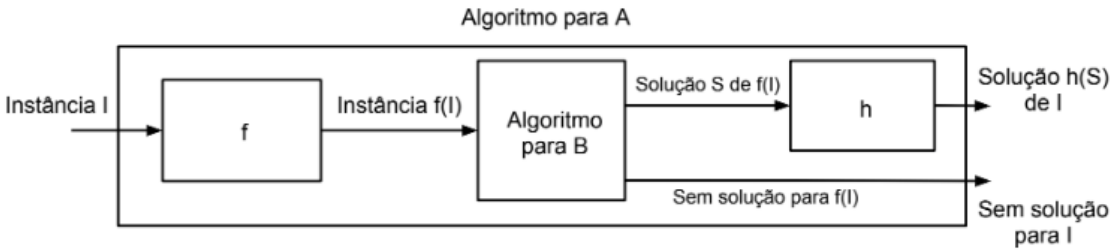
A função isVertexCover recebe uma instância I do problema Vertex-Cover que possui como entrada:

- ➢ Um vetor de pares de inteiros, onde cada par representa uma aresta do grafo;
- ➢ Uma solução S representada por um vetor de inteiros, que contém os vértices selecionados como solução proposta pelo usuário.

A função tem complexidade O(N), ela verifica se cada aresta em I tem pelo menos um de seus vértices em S, retornando true se for o caso de atribuição verdadeira ou false, caso contrário.

3. DETERMINANDO SE NP É REDUTIVEL PARA C EM TEMPO POLINOMIAL

Supondo que o problema A' (Independent Set) é NP-Completo, vamos verificar se A' é redutível para B' (Vertex–Cover) em tempo polinomial.



Provaremos isso com base na imagem acima em implementações de algoritmos.

3.1. Implementação de função *B* que recebe a instância *f(I)* do problema Vertex-Cover, com isso, encontramos a solução ótima por meio de um algoritmo determinístico para ele e retornar a resposta *S* para o problema Vertex-Cover;

```cpp
vector<int> findVertexCover(vector<pair<int, int>> &fI)
{
    // Conjunto para armazenar o conjunto de vértices do Vertex-Cover
    vector<int> vertexCover = {};

    // inicializa um vetor de arestas
    vector<pair<int, int>> arestas = fI;
    // Verifica cada aresta em fI
    for (const auto &e : arestas)
    {
        int u = e.first;
        int v = e.second;

        // Se u e v forem diferentes de 0 são adicionados a resposta
        if (u != 0 && v != 0)
        {
            vertexCover.push_back(u);
            vertexCover.push_back(v);
        }

        // Remove todas as arestas adjacentes a u e v do vetor de arestas
        for (auto &i : arestas)
        {
            int x = i.first;
            int y = i.second;
            if (x == u || x == v || y == u || y == v)
            {
                i.first = 0;
                i.second = 0;
            }
        }
    }

    cout
        << "Solução Vertex-Cover: ";
    for (const auto &vertex : vertexCover)
    {
        cout << vertex << " ";
    }
    cout << endl;

    return vertexCover;
}
```

A função ***findVertexCover*** recebe uma instância do problema Vertex-Cover representada pela lista de arestas ***f(I)*** e retorna a solução ***S***, que é o conjunto de vértices do Vertex-Cover.

A função começa inicializando um vetor vazio Vertex-Cover para armazenar os vértices do Vertex-Cover, em seguida, é criado um vetor de arestas arestas que é inicializado com os valores de ***f(I)***.

A função percorre cada aresta no vetor arestas. Para cada aresta, são obtidos os vértices ***u*** e ***v***.

Se tanto ***u*** quanto ***v*** forem diferentes de 0, significa que a aresta não foi removida anteriormente, então os vértices u e v são adicionados ao Vertex-Cover, depois remove todas as arestas adjacentes a ***u*** e ***v*** do vetor arestas marcando os vértices correspondentes como 0.

Após percorrer todas as arestas em arestas, a função exibe a solução do Vertex-Cover no formato "Solução Vertex-Cover: v1 v2 v3 ..." onde v1, v2, v3, etc são os vértices presentes em Vertex-Cover.

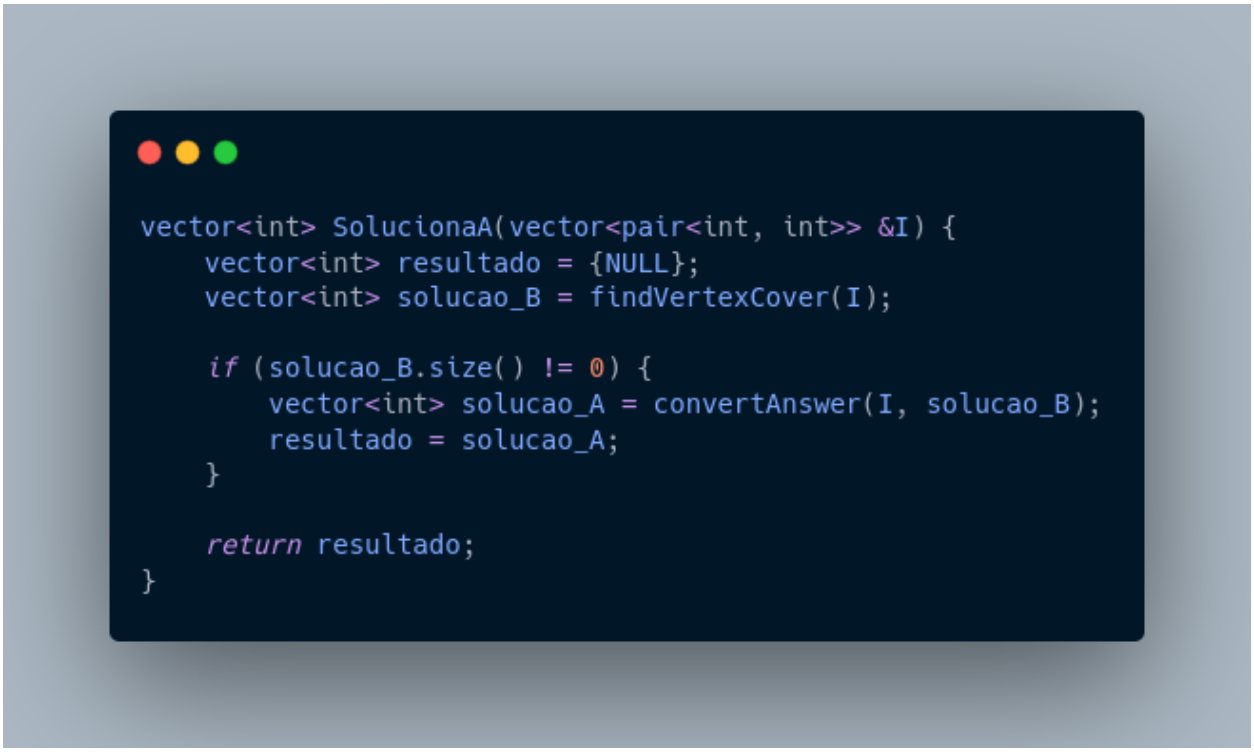
Implementamos uma função ***h***, para caso a instância ***f(I)*** do problema Vertex-Cover tenha solução, essa função h receberá a resposta S do problema Vertex-Cover e será convertida para a resposta do problema Independent set. No caso da instância ***f(I)*** do problema Vertex-Cover não tenha solução, o programa imprimirá “Não existe solução para a instância I”. A complexidade de tempo dessa função é $\mathbf{O(N^2)}$, desconsiderando o ***for*** utilizado para imprimir o vetor, onde N é o número de arestas na instância I do problema Independent Set.

3.2. Implementação de função *h* que recebe a resposta *S* do problema *B'* e converte para resposta do problema *A'*.

```cpp
vector<int> convertAnswer(vector<pair<int, int>> &I, vector<int> &S)
{
    // Inicia um vetor vazio para a resposta
    vector<int> response;

    // Percorre todos os vertices do grafo I
    for (const auto &i : I)
    {

        int u = i.first;
        int v = i.second;

        // Se o vertice u não estiver presente na solução, e não estiver na respota, ele é adicionado ao
vetor response
        if (find(S.begin(), S.end(), u) == S.end() && find(response.begin(), response.end(), u) ==
response.end())
        {
            response.push_back(u);
        }
        // Se o vertice v não estiver presente na solução, e não estiver na respota, ele é adicionado ao
vetor response
        if (find(S.begin(), S.end(), v) == S.end() && find(response.begin(), response.end(), v) ==
response.end())
        {
            response.push_back(v);
        }
    }

    // Retorna a resposta
    return response;
}
```

A função **convertAnswer** recebe um vetor de pares inteiros representando um grafo ***I***, onde cada par representa uma vertice do grafo, e recebe também um vetor ***S*,** que contém os vértices pertencentes à solução **vertex-cover** de ***I***. Com isso, o algoritmo percorre todo o grafo ***I*** comparando se os vértices já estão na solução ***S*,** se não estiver, o vértice é inserido em um vetor que representa a solução do **independent-set.** Por ter um laço for, a complexidade desse algoritmo é O(n).

3.3. Implementação que verifica se o problema *A'* tem solução com o algoritmo *B*



```
vector<int> SolucionaA(vector<pair<int, int>> &I) {
    vector<int> resultado = {NULL};
    vector<int> solucao_B = findVertexCover(I);

    if (solucao_B.size() != 0) {
        vector<int> solucao_A = convertAnswer(I, solucao_B);
        resultado = solucao_A;
    }

    return resultado;
}
```

A função ***SolucionaA*** tem como parâmetro ***I*** que é um grafo qualquer. É criada a variável ***resultado*** para receber a resposta final, a variável ***solucao_B*** recebe o vertex cover correspondente a instância dada, logo em seguida é verificado se a solução existe, se sim, a solução do problema B' é convertida para a solução do problema ***A'*** e colocada na variável ***solução A*** e então ***resultado*** recebe ***solucao_A***, caso não exista, retorna um vetor vazio, que significa que não existe solução para a instância ***I***.

Obs: Não existe uma conversão do problema ***A'*** para o problema ***B'***, pois ambos são extraídos de um grafo, ou seja, não há necessidade de conversão.

Conclui-se que a redução Independent Set → Vertex-Cover é um NP-Completo, pois

- ***c*** está em ***NP***;
- Todo problema em ***NP*** é redutível para ***c*** em tempo polinomial.