A B A K Ó S
PUC Minas Instituto de Ciências Exatas e Informática

ICEI - PUC Minas

Bruno Pena Baêta[1]
Diogo Castro[2]
Felipe Tadeu Góes Guimarães[3]
João Quintão[4]

Resumo

Neste trabalho iremos apresentar uma solução para se determinar o número máximo de caminhos disjuntos em arestas existentes em um grafo. Neste problema, dado um grafo e um par de vértices, sendo uma origem e outro o destino, deseja-se encontrar no final a quantidade de caminhos distintos em arestas entre os dois vértices dados. Além disso deve-se listar cada um dos caminhos encontrados. Utilizaremos o Algoritmo de Edmonds-Karp para determinar os caminhos disjuntos e para comparação vamos utilizar os caminhos disjuntos que achamos manualmente.

Palavras-chave: caminhos disjuntos, Edmonds-Karp

[1]Bacharel em Ciências da Computação,
Programa de graduação em Ciências da Computação da PUC Minas, Brasil.
[2]Bacharel em Ciências da Computação,
Programa de graduação em Ciências da Computação da PUC Minas, Brasil.
[3]Bacharel em Ciências da Computação,
Programa de graduação em Ciências da Computação da PUC Minas, Brasil.
[4]Bacharel em Sistemas de Informação,
Instituto de Ciências Exatas e de Informática da PUC Minas, Brasil.

1 INTRODUÇÃO

A princípio, achar em um grafo direcionado todas as rotas que não se cruzam em arestas, dados o vértice de raiz e o vértice de destino, parece algo trivial e sem muitas implicações plausíveis para problemas reais.

Porém, após esse primeiro contato com assunto, vimos o quão importante é este problema pois, existem muitos problemas reais que podem ser modelados como caminhos disjuntos em arestas. Por exemplo, um sistema de suporte à decisão em gerenciamento de fluxo de tráfego aéreo, ou ainda, redes de roteamento de transmissão múltipla obtidas a partir de uma rede geral de comunicações.

Para acharmos uma solução para tal problema utilizaremos uma modelagem que inferi os grafos como ponderados, ou seja, todas as arestas terão peso 1 e iremos usar um dos algoritmos de fluxo da teoria de grafos, o Algoritmo de Edmonds-Karp para encontrar todos os caminhos disjuntos em arestas.

2 METODOLOGIA

A resolução do problema é dividida em dois arquivos, *graph.cpp* que é responsável pela representação e operações em grafos e *main.cpp* que lida com as entradas e saídas.

O arquivo *graphGenerator.cpp*(BHOJASIA,) foi criado com o propósito de gerar grafos com proporções maiores para testar os limites dos algoritmos implementados.

Em *main.cpp*, é importado o arquivo *graph.cpp*, que contém todas as bibliotecas padrão da linguagem C++ usadas no projeto. A monitorização do tempo de execução do programa é realizada no arquivo *main.cpp* e conta o tempo que o programa demorou para achar os caminhos disjuntos.

A classe graph, definida em *graph.cpp*, possui 5 campos, um vetor 2D de inteiross, que representa a matrix de adjacência, um inteiro que representa o número de caminhos disjuntos calculados antecipadamente, um inteiro que representa o número de caminhos disjuntos obtido pelo algoritmo, um vetor de tuplas, que armazena os caminhos encontrados pelo algoritmo e por fim há dois inteiros que guardam o valor do número de vértices e arestas respectivamente.

Dentro os métodos da classe graph, se destacam graph_from_txt(), vector<int> obtain_successors(int vertex, vector<vector<int» graph) , bool breadth_first_search(vector<vector<int» graph,int rootVertex, int targetVertex, vector<int> &parents) e void find_unique_paths(). O primeiro se responsabiliza por criar um grafo a partir do padrão de .txt usado pelo grupo, que segue o padrão a seguir:
n1 n2 n3
v0 v1
v0 v2
...
vn v(n1-1)

n1 representa o número de vértices, n2 representa o número de arestas, n3 representa o número de caminhos disjuntos, em seguida, todas a n2 arestas são representadas pelo vértice de saída e de entrada.
O segundo obtém os n vértices sucessores de um vértice vertex em um grafo graph, o terceiro realiza uma busca em largura em um grafo graph, do vértice rootVertex até o vértice targetVertex, preenchendo o array parents com os pais de cada vértice e retorna se encontrou caminho entre os vértices rootVertex e targetVertex. Por fim, o último encontra os caminhos disjuntos de um grafo, tendo como grafo source o que se encontra na posição 0 e grafo terminal o que se encontra na última posição. Ele faz isso colocando todas as arestas com peso 1 e obtendo o fluxo máximo desse grafo por meio do método de Edmonds-Karp(GEEKSFORGEEKS,).Ao se obter o fluxo máximo, um grafo novo somente com as arestas que possuem fluxo, eliminando todas as arestas não utilizadas pelo grafo, em seguida, os caminhos são extraídos desse grafo e são inseridos no vetor de caminhos da classe.

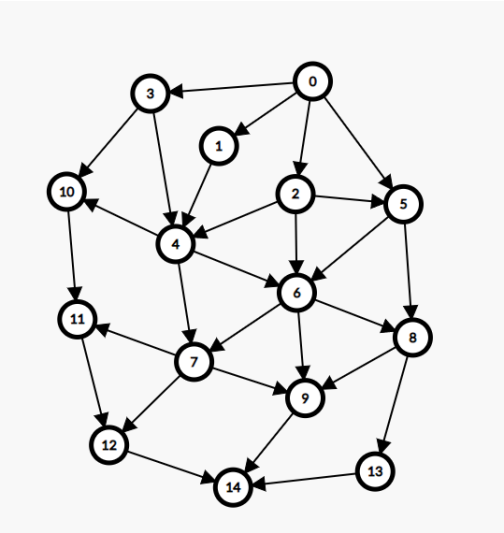
3 RESULTADOS

Nesta seção serão apresentados os resultados dos testes realizados usando nosso programa e os dados de teste criados automaticamente e manualmente.

3.1 Algoritmo para se encontrar número máximo de caminhos disjuntos

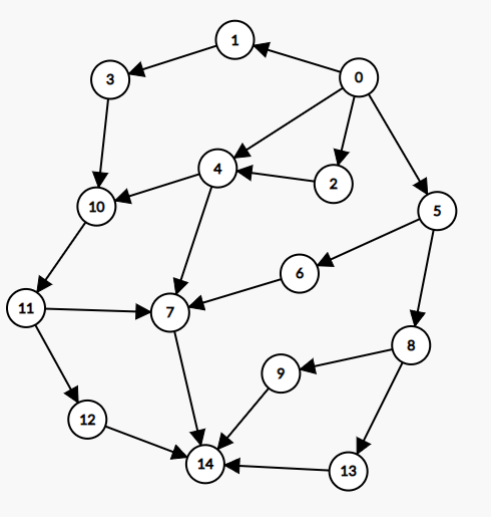
O algoritmo implementado (complexidade $O(n^4)$ ou $O(EV^3)$) resolve esse problema de forma ótima. Ele realiza a busca em largura (complexidade $O(n^2)$ ou $O(V^2)$ dado ao uso de matriz de adjacência) no máximo $O(n^2)$ $O(EV)$ vezes.

1. O primeiro grafo conta com 15 vértices e 29 arestas:



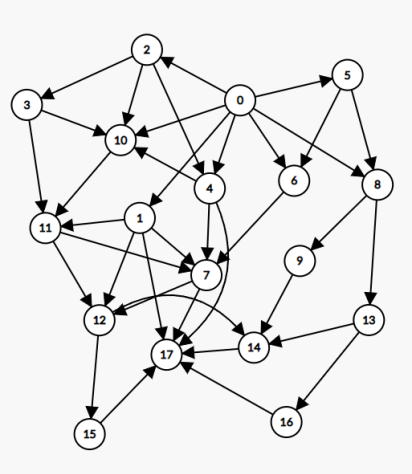
Solução ótima: 3 caminhos
Solução encontrada pelo algoritmo: 3 caminhos
Caminhos encontrados pelo algoritmo:
0 5 6 9 14
0 1 4 7 12 14
0 2 4 6 8 13 14

2. O segundo grafo conta com 15 vértices e 21 arestas:



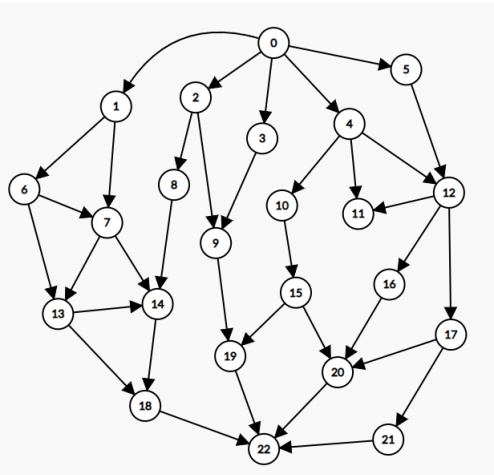
Solução ótima: 3 caminhos
Solução encontrada pelo algoritmo: 3 caminhos
Caminhos encontrados pelo algoritmo:
0 5 8 9 14
0 4 7 14
0 1 3 10 11 12 14

3. O terceiro grafo conta com 18 vértices e 37 arestas:



Solução ótima: 6 caminhos
Solução encontrada pelo algoritmo: 6 caminhos
Caminhos encontrados pelo algoritmo:
0 10 11 7 17
0 6 14 17
0 1 17
0 4 17
0 8 13 16 17
0 5 6 7 12 15 17

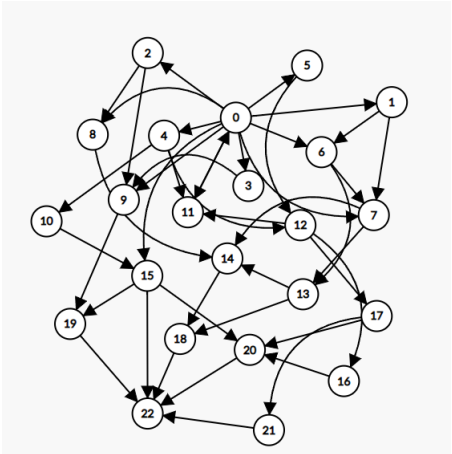
4. O quarto grafo conta com 23 vértices e 36 arestas:



Solução ótima: 4 caminhos
Solução encontrada pelo algoritmo: 4 caminhos
Caminhos encontrados pelo algoritmo:
0 4 10 15 20 22
0 5 12 17 21 22

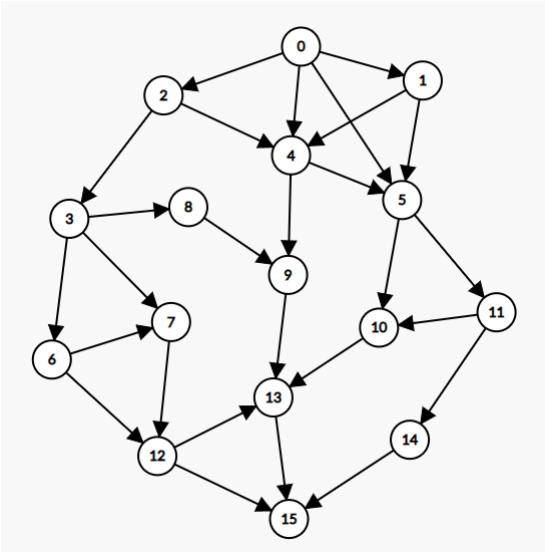
0 2 9 19 22
0 1 6 13 18 22

5. O quinto grafo conta com 23 vértices e 44 arestas:



Solução ótima: 5 caminhos
Solução encontrada pelo algoritmo: 5 caminhos
Caminhos encontrados pelo algoritmo:
0 15 22
0 9 19 22
0 6 13 18 22
0 4 10 15 20 22
0 5 12 17 21 22

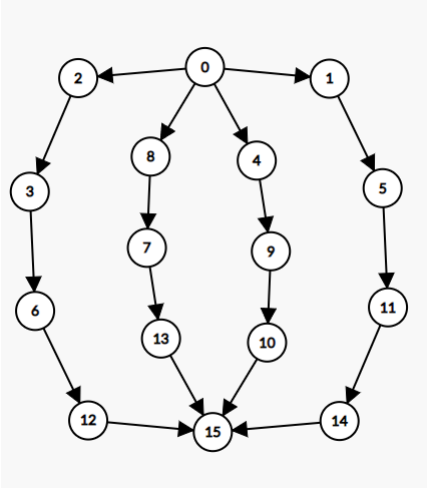
6. O sexto grafo conta com 16 vértices e 27 arestas:



Solução ótima: 3 caminhos
Solução encontrada pelo algoritmo: 3 caminhos

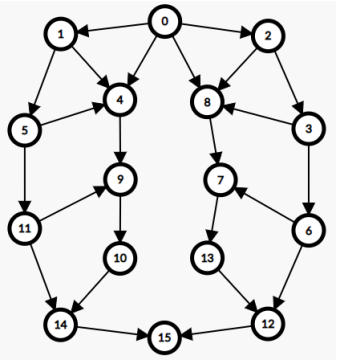
Caminhos encontrados pelo algoritmo:
0 4 9 13 15
0 5 11 14 15
0 2 3 6 12 15

7. O setimo grafo conta com 16 vértices e 18 arestas:



Solução ótima: 4 caminhos
Solução encontrada pelo algoritmo: 4 caminhos
Caminhos encontrados pelo algoritmo:
0 4 9 10 15
0 8 7 13 15
0 1 5 11 14 15
0 2 3 6 12 15

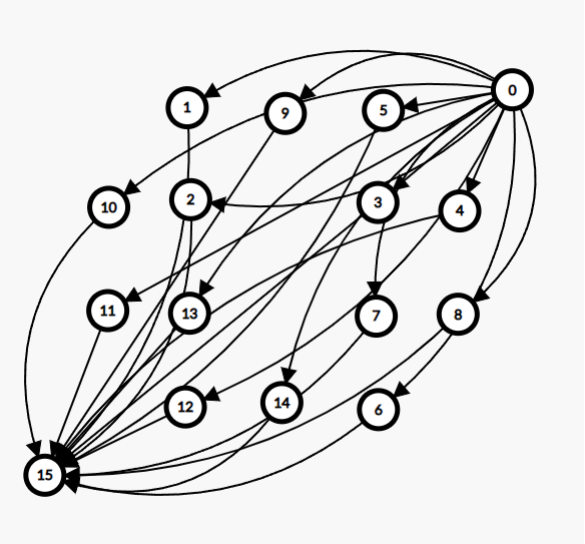
8. O oitavo grafo conta com 16 vértices e 23 arestas:



Solução ótima: 2 caminhos
Solução encontrada pelo algoritmo: 2 caminhos
Caminhos encontrados pelo algoritmo:

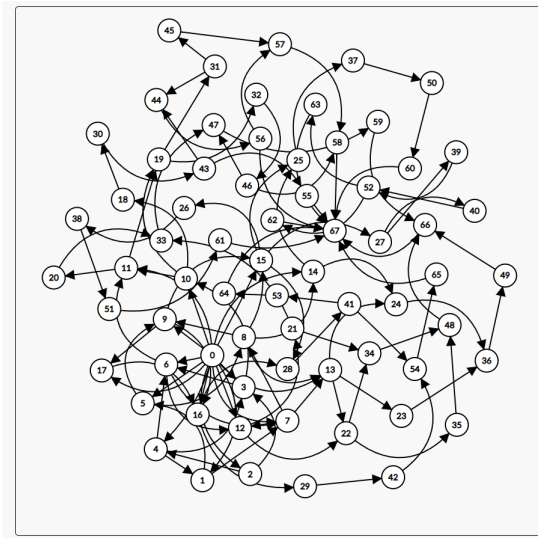
0 1 5 11 14 15
0 2 3 6 12 15

9. O nono grafo conta com 16 vértices e 28 arestas:



Solução ótima: 15 caminhos
Solução encontrada pelo algoritmo: 15 caminhos
Caminhos encontrados pelo algoritmo:
0 1 15
0 2 15
0 3 15
0 4 15
0 5 15
0 6 15
0 7 15
0 8 15
0 9 15
0 10 15
0 11 15
0 12 15
0 13 15
0 14 15

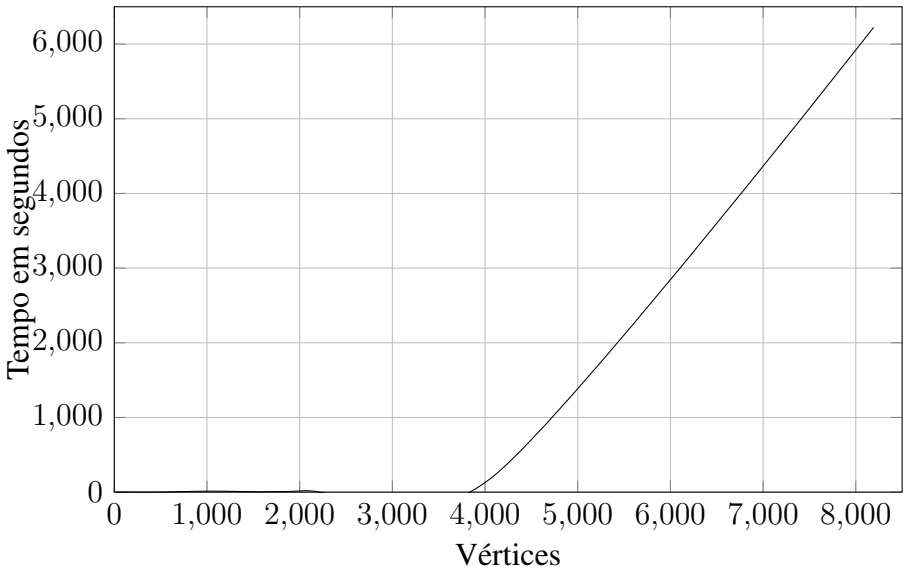
10. O decimo grafo conta com 68 vértices e 118 arestas:



Solução ótima: 9 caminhos
Solução encontrada pelo algoritmo: 9 caminhos
Caminhos encontrados pelo algoritmo:
0 10 18 30 43 55 67
0 11 19 31 44 56 67
0 12 21 34 48 66 67
0 14 25 37 50 60 67
0 15 26 38 51 61 67
0 17 16 28 41 53 64 67
0 16 29 42 54 65 67
0 9 15 27 39 52 62 67
0 4 3 10 19 32 46 58 67

3.2 Análise de vértices por tempo em grafos

Os grafos criados automaticamente nos permitem testar os limites do algoritmo implementado. Foram gerados grafos de 2^n vértices e $4 * 2^n$ arestas para um n de 4 à 14. Isso é um intervalo de 16 a 8192 vértices e 64 a 32768 arestas.



4 CONCLUSÃO

A partir dos resultados obtidos nos testes, podemos concluir que o método proposto é viável, uma vez que o tempo de execução é polinomial e a sua solução é ótima. Além disso, seu tempo de execução não segue $O(n^4)$ para os grafos testados, seguindo um crescimento linear nos grafos de teste.

Referências

BHOJASIA, Manish Kumar. **C++ Program to Generate a Random Directed Acyclic Graph DAC for a Given Number of Edges**. Disponível em: <https://www.sanfoundry.com/cpp-program-construct-random-directed-acyclic-graph/>.

GEEKSFORGEEKS. **Ford-Fulkerson Algorithm for Maximum Flow Problem**. Disponível em: <https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/>.