Estudo do problema do caminho mais longo

Allan Cordeiro Rocha de Araújo
(1)Universidade Federal de Roraima
allanps32008@hotmail.com

Paulo Fábio dos Santos Ramos
(1)Universidade Federal de Roraima
paulofabyo@gmail.com

Resumo *Será feito um estudo do problema do caminho mais longo, assim como aplicação prática dele, analisando sua versão exata e uma versão aproximada com complexidade de tempo melhor. Também é feito uma comparação gráfica entre os dois, em relação ao tempo para certas entradas no algoritmo.*

Palavras-Chave: *Problema NP-Completo, Complexidade, NP-difícil, Grafo.*

Abstract *It will be done a study of the problem of the longest path, just like a practical application of it, analyzing its exact version and a linear time solution with better time complexity. Also a graphical comparison between the two, in relation to the time for certain entries in the algorithm is made.*

Keywords: *NP-Complete problem, Complexity, NP-hard, Graph.*

1 Introdução

O objetivo deste artigo é, a partir da versão exata do problema do caminho mais longo, implementar uma versão sua aproximada usando a técnica de ordenação topológica em grafos, para otimização da resolução do problema em questão, e mostrar suas respectivas complexidades com intuito de provar a otimização de tal problema.

2 Problemas NP-Completo

São problemas de decisão, ou seja, são problemas que possuem resposta sim-não para eles. Um problema B é NP-completo se todos os problemas NP se transformam polinomialmente em A, ou seja, se for possível resolver um problema NP-completo com um algoritmo determinístico em tempo polinomial, então todos os problemas da classe NP também podem ser resolvidos em tempo polinomial, porém isso não é possível.

3 Problemas NP-Difícil

Problemas NP-Difíceis são problemas computacionalmente difíceis de se resolver. Um problema H é NP-difícil se existe um problema B € NP-completo que pode ser reduzido para H em tempo polinomial, ou seja, H não é provado para ser NP.

4 Descrição formal e aplicações

Dado um grafo G=(V,E), onde V é o número de vértices e E o número de arestas, e um k € N. Questão: Qual o maior caminho simples em G começando de k?

Algumas aplicações no mundo real podem ser: análise de temporização tática (STA) e suas variantes, em automa-

ção de projetos eletrônicos e achar os caminhos críticos em um circuito digital.

5 Subproblema

O caminho mais longo é encontrar um caminho simples de valor máximo em um grafo, sem vértices repetidos. Ele é um problema NP-difícil, na verdade é uma redução do problema de decisão Hamiltoniano: onde em um grafo G=(V,E) o seu caminho hamiltoniano é um caminho n-1, onde n é o número de vértices V de G, saída sim para caso exista e não para caso não exista.

6 Versão exata do problema

Determinar o caminho mais longo em um determinado grafo consiste em verificar todos os caminhos que são possíveis a partir de um vértice inicial afim de obter o maior percurso entre eles.

Com isso, é proposto o seguinte algoritmo implementado em C++, que percorre uma matriz de adjacência verificando todos os caminhos possíveis, usando um vetor que guarda o caminho atual como um auxiliar para evitar que o mesmo vértice seja visitado duas vezes. Nessa implementação ele faz uso dos pesos das arestas para verificação de qual caminho mais longo foi encontrado e também guarda o número de vértices que fazem parte do mesmo.

```
      int longestPath(int node){
         int cam[MAX_NODO], tam=0,max=0, dest, i, peso=0;
         //Inicializa um vetor com valor -1
         for(i=0; i<MAX_NODO; i++)
            cam[i] = -1;

         dest = 0;
         while(1){
            if(matrix[node][dest] > 0){//Verifica se existe uma aresta entre nodo e dest
               if(verificarPath(cam, tam, dest) == 1){//Caso exista uma aresta, verifica se o
destino já está no caminho
                  peso += matrix[node][dest];

                  cam[tam] = node;//salva os nodos que foram visitados
                  node = dest;
                  tam++;

                  dest=0;
                  continue;
               }
            }
            dest++;
            do{
               if(dest >= MAX_NODO){//verifica se o tamanho maximo da matriz ja foi alcançado,
para então realizar o backtracking
                  if(peso>pesoPath){//verifica se na execução atual foi encontrado um caminho
maior
                     for(i=0; i<tam; i++)
                        path[i] = cam[i];
                     path[i] = node;
                     max = tam;
                     pesoPath = peso;
                  }
                  tam--; //realização do backtracking
                  dest = node;
                  node = cam[tam];
                  peso -= matrix[node][dest];

                  dest++;
               }
            }while(dest >= MAX_NODO);

            if(tam < 0){//caso o tamanho do vetor seja negativo, é dito que todas as possibili-
dades ja foram testadas
               return max;
            }
         }
      }
```

Figura 1 : Função da implementação da versão exata

7 Versão aproximada do problema e Complexidade de tal

Nesta implementação foi usado a linguagem de programação C++, usando o GNU GCC Compiler para compilar os programas. No código em si foi usado suas estruturas de dados, vetor de lista (lista de adjacência) e pilhas das bibliotecas <list> e <stack> respectivamente.

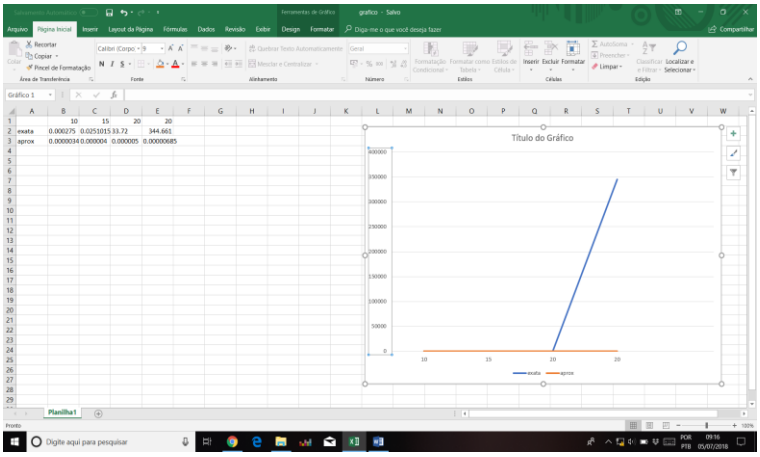
A técnica aplicada para otimização foi usa a estratégia da ordenação topológica, que consiste em organizar um grafo linearmente onde cada nó venha antes de todos aos quais tenha aresta ligando. No cálculo do caminho mais longo é usado um vetor para armazenar as distancias do nó origem para todos os outros nós, é tanto que a distância na posição do nó origem será zero (0) em todas as saídas do programa.

Nó código a seguir, foi colocado apenas as funções para calculo da distância e para a ordenação topológica que é de vital importância para essa resolução.

Para V=número de vértices, E=número de arestas. A complexidade de tempo para a ordenação topológica é O (V+E). Já no cálculo de caminho mais longo em si, para cada vértice, da pilha em ordem topológica, ele executa todos os vértices adjacentes, vértices adjacentes em um grafo é O(E), assim a complexidade de tempo é O(V+E).

```
void Graph::topologicalSortUtil(int v, bool visited[],stack<int> &Stack){
   visited[v] = true;//marca o nó que ele está como visitado

   list<AdjListNode>::iterator i;
   for (i = adj[v].begin(); i != adj[v].end(); ++i){
      AdjListNode node = *i;
      if (!visited[node.getV()])
         topologicalSortUtil(node.getV(), visited, Stack);
   }

   Stack.push(v);//empilha o vertice atual na pilha Stach
}
void Graph::longestPath(int s){
   stack<int> Stack; // uma pilha da classe 'stack' para empilhar os nós visitados
   int dist[V];

   bool *visited = new bool[V];
   for (int i = 0; i < V; i++)
      visited[i] = false;

   for (int i = 0; i < V; i++)
      if (visited[i] == false)
         topologicalSortUtil(i, visited, Stack);

   for (int i = 0; i < V; i++)
      dist[i] = NINF;
   dist[s] = 0;

   while (Stack.empty() == false){ // verifica se a pilha está vazia
      int u = Stack.top();

      Stack.pop(); //desempilha a pilha

      list<AdjListNode>::iterator i;
      if (dist[u] != NINF){
         for (i = adj[u].begin(); i != adj[u].end(); ++i){
            if(dist[i->getV()] < dist[u] + i->getWeight()){
               dist[i->getV()] = dist[u] + i->getWeight();
            }
         }
      }
   }
   cout << "\n";
   for (int i = 0; i < V; i++)
      (dist[i] == NINF)? cout << "INF ": cout << dist[i] << " ";
}
```

8 Análise do tempo de execução dos algoritmos



Os testes foram realizados em uma distro do Linux. Intel core i5, geforce 930m, 6gb de ram, Ubunto versão 16.04. Todos os testes foram executados 7 vezes e foi retirado a media entre o tempo de execução.

Foi feito uma análise da execução dos algoritmos de versão exata e aproximada, dado um certo grafo G com um número de vértices, foi calculado o tempo de execução. Está plotado em um gráfico feito no Excel esses dados, os vértices e o tempo de execução

9 Referências

[1] "Teoria da Complexidade Computacional"; Leticioa Bueno/aa/materiais. Disponível em <http://professor.ufabc.edu.br/~leticia.bueno/classes/aa/materiais/complexidade2.pdf>. Acesso em 05 de julho de 2018.

[2] "Decisão"; Disponível em<https://web.fe.up.pt/~jfo/ensino/io/docs/IOT_decisao.pdf>. Acesso em 05 de julho de 2018.

[3] "Algoritmo de Posicionamento Analítico Guiado a Caminhos Críticos"; Jucemar Luis Monteiro. Disponível em<https://www.lume.ufrgs.br/bitstream/handle/10183/116139/000966425.pdf?sequence=1>. Acesso em 05 de julho de 2018.

[4] "Grafos Topológicos"; Jucemar Luis Monteiro. Disponível em< https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/graph-families.html#topological-numbering>. Acesso em 05 de julho de 2018.

[5] "Longest Path in a Directed Acyclic Graph"; geeks for geeks. Disponível em< https://www.geeksforgeeks.org/find-longest-path-directed-acyclic-graph/>. Acesso em 05 de julho de 2018.