Estudo do caminho mais longo

Allan Cordeiro Rocha de Araújo, Paulo Fábio dos Santos Ramos

[1]Centro de ciência e tecnologia– Universidade Federal de Roraima (UFRR)
Boa Vista– RR – Brasil

allanps32008@gmail.com, paulofabyo@gmail.com

***Abstract**. It will be done a study of the problem of the longest path, just like a pratical application of it, analyzing its exact version and a linear time soluction with better time complexity. Also a graphical comparsion between the tow, about the time for certain entries in the algorithm is made.*

***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.*

1. Introdução

O objetivo deste artigo é, demonstrar uma comparação feita em relação ao tempo de execução de um algoritmo aproximado para o problema do caminho mais longo em relação a versão exata do problema. A versão de aproximação foi implementada usando a técnica de ordenação topológica em grafos, para otimizar a resolução do problema em questão.

2. Problemas NP

Fazendo a análise da complexidade de algoritmos, conseguimos identificar diferentes classes de problemas, são elas P, NP, NP-dificil e NP-completo. A classe P de problemas, são aqueles que conseguem ser resolvidos em tempo polinomial, dado uma instancia do problema, ele sempre encontrara uma solução após determinado tempo. Problemas NP (*nondeterministic polynomia*l) são aqueles que, dada uma entrada para o problema, é possível verificar se aquela entrada é faz parte de uma solução do problema. Ou seja, consiste na classe dos problemas de decisão, pois somente é necessário dizer se a solução proposta é válida ou não, podemos falar que P $\subset$ NP.

Um problema é NP-Completo se ele for um problema NP-Difícil e também se ele for da classe NP, somente problemas de decisão podem ser classificados como NP-Completos. O conhecimento dos problemas NP-Completos se deu graças a Steven Cook, que na década de 70 conseguiu provar a existência do primeiro algoritmo dessa classe, conhecido como SAT (*Problema de satisfatibilidade booleana*), e a partir dele, se tornou provar a existência de outros problemas NP-Completos a partir da redução polinomial, pois dado um problema X e um problema Y tal que Y é NP-Completo, se provarmos que X é tão difícil quanto Y, X também é NP-Completo, isso consiste em dizer também que, 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 o que tornaria valida a questão “P = NP?”, porém isso não é possível.

3. Problemas NP-Difícil

Problemas NP-Difíceis são computacionalmente difíceis de se resolver, e até então não existe nenhum algoritmo que é capaz de resolver tais problemas em tempo polinomial. Um problema H é NP-difícil se existe um problema B $\in$ NP-completo, que pode ser transformado em H em tempo polinomial, ou seja, H é NP-Difícil caso seja uma variação de B, porém H não se encontra na classe 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 $\in$ N, percorrer todos os caminhos possíveis em tal grafo afim de encontrar um caminho cujo o peso de suas arestas seja o máximo possível.

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

5. Sub-problema

O problema do caminho mais longo consiste em encontrar um caminho simples de valor máximo em um determinado grafo, sem que haja repetição de vértices. Ele é um problema NP-difícil devido ao alto custo de calcular todos os possíveis caminhos, na verdade é uma redução do problema de decisão Hamiltoniano, onde, em um grafo G=(V,E) o caminho hamiltoniano é um aquele que possui n-1 elementos, onde n é o número de vértices de G, a saída para o problema de decisão Hamiltoniano é sim para caso exista tal caminho, 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 possibilidades ja foram testadas
         return max;
      }
   }
}
```

7. Versão aproximada do problema

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 cálculo 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];
```

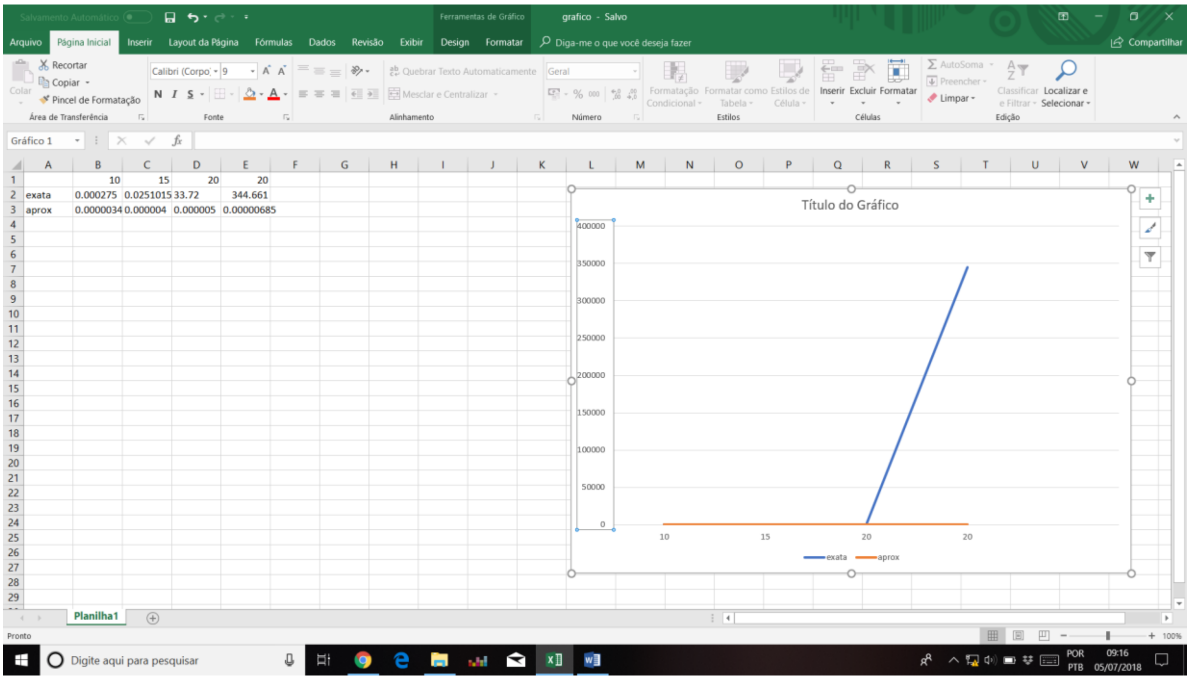
```
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    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 de tempo de execução

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

Foi feita a análise do tempo de execução dos algoritmos de versão exata e aproximada. Os testes foram feitos inicialmente em grafo com 10 vértices e 18 arestas, logo em seguida foram adicionados mais 5 vértices e 15 arestas, totalizado 15 vértices e 33 arestas. Até então o algoritmo para versão exata conseguiu calcular sem muita dificuldade, quando aumentamos o grafo para 20 vértices e 62 arestas, o mesmo já começou a demorar em torno de 30 segundos para apontar qual o caminho mais longo, aumentando somente 8 arestas à tal vértice, a demora foi de mais de 5 minutos para o cálculo, enquanto que a versão aproximada manteve um excelente tempo de execução para grafos com 30 vértices e 275 arestas.

Abaixo está plotado um gráfico feito no Excel desses dados.



Referências

"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.

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

“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.

“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.

“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.