Universidade Federal de Minas Gerais **Algoritmos e Estruturas de Dados II**
Instituto de Ciências Exatas **1º Semestre de 2014**
Departamento de Ciência da Computação **Prof. Luiz Chaimowicz e Raquel Prates**

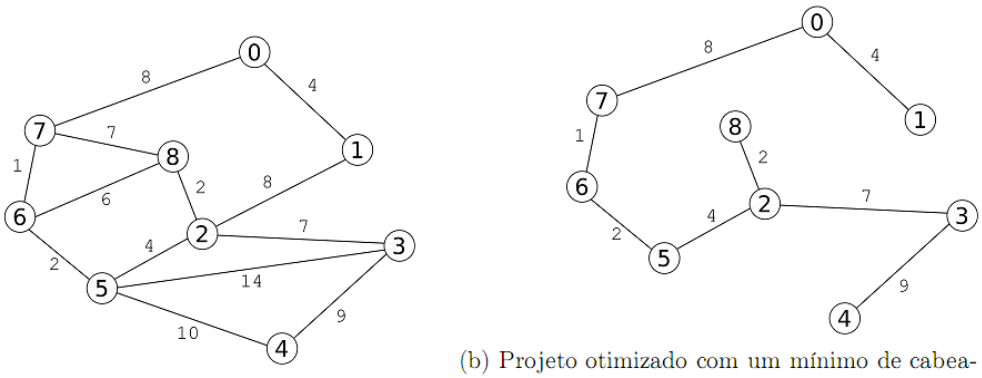
Trabalho Prático1 – Árvore Geradora Mínima

Valor: 10 pontos
Data de devolução: 29/04/2014

O Departamento de Ciência da Computação (DCC) da UFMG está construindo um edifício para abrigar seus laboratórios e deseja criar uma rede de computadores com ligando todas as salas do novo prédio. Um projeto inicial foi proposto, mas financeiramente inviável por utilizar muito cabeamento. O DCC gostaria de implementar uma rede que utilize um comprimento mínimo de cabos, e você deve criar um programa que modifique o projeto inicial para atender a esta exigência.

O projeto de cabeamento deve ser modelado como um grafo em que há um vértice representando cada sala e há uma aresta entre dois vértices se há um cabo ligando as duas salas correspondentes. Cada aresta possui um peso representando o comprimento do cabo que ela descreve.

O problema de encontrar o conjunto de cabos mais curto ligando as salas equivale ao problema de encontrar a *árvore geradora mínima* do grafo. Seu programa deve, portanto, receber um grafo como entrada e produzir como saída uma árvore geradora mínima para este grafo. A Figura 1 contém um exemplo de um grafo representando um projeto de cabeamento original (Figura 1a), e uma árvore geradora mínima representando o projeto otimizado com um mínimo de cabos (Figura 1b).



(a) Projeto de cabeamento original.

(b) Projeto otimizado com um mínimo de cabeamento.

Figura 1: Exemplo de um projeto de cabeamento e sua otimização.

Implementação:

Para implementar o seu grafo, você deverá utilizar uma estrutura de dados chamada *Lista de Adjacência*, na qual é criado um vetor com os vértices e para cada vértice é criada uma lista encadeada contendo os vértices com os quais ele está ligado (a Fig. 2 mostra um exemplo). No caso desse trabalho sua lista deverá armazenar também os pesos das arestas. Crie um TAD Grafo, com as operações necessárias. Um exemplo desse TAD é dado no livro texto do Prof. Nívio Ziviani, 3a Edição [1]. Para entender o TAD e sua implementação, estude as Seções 7.1 e 7.2 (páginas 277 a 295) do livro. Cabe ressaltar que o TAD Grafo deve ser implementado usando uma lista de adjacências usando apontadores, como explicado na Seção 7.2.2.

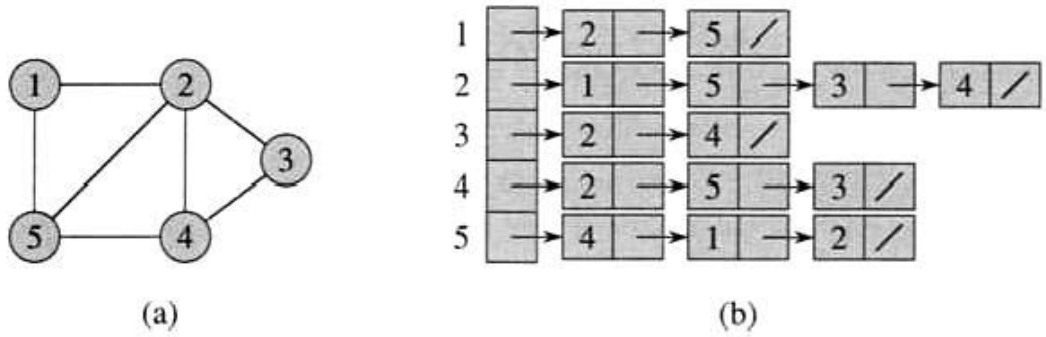


Figura 2: Exemplo de grafo não-direcionado (a) e sua representação usando lista de adjacência (b)

Existem vários algoritmos para encontrar a árvore geradora mínima de um grafo. Neste trabalho, você deve implementar o algoritmo a seguir, que começa construindo a árvore geradora mínima a partir de um único vértice, e a cada passo inclui na árvore uma aresta de menor peso que conecte mais um vértice à árvore. O algoritmo para quando todos os vértices forem incluídos na árvore. Note que uma árvore é um caso especial de grafo, então o TAD Grafo pode ser usado para representar tanto o grafo *G* de entrada quanto a árvore geradora mínima *A* de saída.

```
1: function AGM(G)
2:     Crie um grafo A;
3:     Arestas(A) = vazio;
4:     Vertices(A) = vazio;
5:     Escolha um vértice v qualquer em Vertices(G);
6:     Inclua v em Vertices(A);
7:     while Vertices(A) ≠ Vertices(G) do
8:         Escolha uma aresta (u, v) com menor peso possível em Arestas(G), tal que a aresta
   conecte um vértice já em Vertices(A) a um vértice ainda fora de Vertices(A);
9:         Inclua a aresta (u, v) em Arestas(A);
10:        if vértice u ainda não está em Vertices(A) then
11:            Inclua u em Vertices(A);
12:        else if vértice v ainda não está em Vertices(A) then
13:            Inclua v em Vertices(A);
14:        end if
15:    end while
16:    return A;
17: end function
```

Para entender o funcionamento do algoritmo acima, leia as Seções 7.8 e 7.8.1 do livro texto (3ª edição, páginas 312-315). O livro do Cormen, 3a edição [2], também possui uma boa explicação sobre o algoritmo no Capítulo 23.

Atenção: Entender a definição e implementação do TAD Grafo, assim como o funcionamento do algoritmo de geração da Árvore Geradora Mínima faz parte deste trabalho!

Entrada e saída:

O seu programa irá ler da entrada o grafo representando o projeto inicial de cabeamento. A primeira linha da entrada contém o número *v* de vértices e o número *e* de arestas do grafo, separados por um espaço em branco. A partir da segunda linha, a entrada apresenta a lista das arestas e seus

pesos, na forma v_i v_j $p_{i,j}$, significando que há uma aresta entre os vértices v_i e v_j , cujo peso é $p_{i,j}$.

Como saída, seu programa deverá imprimir a árvore geradora mínima, respeitando o mesmo formato do grafo de entrada: a primeira linha deverá conter o número de vértices e arestas da árvore geradora mínima, e a partir da segunda linha deve seguir-se a lista de vértices pertencentes à árvore. A lista de arestas da saída deve estar ordenada de forma que:

i) em cada aresta o menor vértice sempre aparece antes do maior vértice, ou seja, em toda aresta v_i v_j $p_{i,j}$, temos que $v_i < v_j$. Por exemplo, uma aresta entre os vértices 0 e 3 com peso 7 deve ser representada como 0 3 7 e **NUNCA** como 3 0 7.

ii) arestas são ordenadas em ordem crescente por seu menor vértice, e quando o menor vértice de duas arestas é o mesmo, elas são ordenadas em ordem crescente pelo segundo vértice. Por exemplo, a aresta 0 7 8 vem antes da aresta 2 3 7 porque o primeiro vértice da primeira aresta (0) é menor que o primeiro vértice da segunda (2). Já a aresta 2 3 7 vem antes de 2 5 4 porque elas têm o primeiro vértice igual (2), mas estão ordenadas pelo segundo vértice (3 < 5).

A Tabela 1 contém uma representação da entrada e da saída correspondendo ao exemplo da Figura 1. Note que a saída está apropriadamente ordenada.

Entrada	**Saída**
9 13 0 1 4 0 7 8 1 2 8 2 3 7 2 8 2 2 5 4 3 4 9 3 5 14 4 5 10 5 6 2 6 7 1 6 8 6 7 8 7	9 8 0 1 4 0 7 8 2 3 7 2 5 4 2 8 2 3 4 9 5 6 2 6 7 1

Tabela 1: Exemplo de entrada e saída para o projeto da Figura 1.

A entrada deve ser lida do teclado (usando, por exemplo, `scanf`), e a saída deve ser impressa na tela (usando `printf`). Ao realizar seus testes, você pode querer ler a entrada de um arquivo (por exemplo, file.in) e escrever a saída em um arquivo (por exemplo file.out). Nesse caso, você pode utilizar uma linha de comando como a seguir, que usa o arquivo file.in como a entrada padrão em vez do teclado, e usa o arquivo file.out como a saída padrão, em vez da tela:

```
./executavel < file.in > file.out
```

O que deve ser entregue:

- Código fonte do programa em C (bem identado e comentado).
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:
    1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
    2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
    3. Estudo de Complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O).
    4. Testes: descrição dos testes realizados e listagem da saída (não edite os resultados).
    4. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
    5. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso

  Obs1: Apesar de esse trabalho ser bem simples, a documentação pedida segue o formato da documentação que deverá ser entregue nos próximos trabalhos.

  Obs2: Um exemplo de documentação será postado no Moodle.

Forma de Entrega:

O trabalho (código e documentação) deverá ser entregue no PRATICO. Para isso, você deverá ter se cadastrado no Prático (http://aeds.dcc.ufmg.br/) e na turma (Turma N - Raquel Prates e Chaimowicz - AEDS2).

Comentários Gerais:

1 Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.

2 Clareza, identação e comentários no programa também vão valer pontos.

3 O trabalho é individual.

4 Trabalhos copiados serão penalizados conforme anunciado.

5 Penalização por atraso: $(2^d - 1)$ pontos, onde d é o número de dias (úteis) de atraso. Note que após dois dias úteis, o trabalho não pode mais ser entregue.

Referências:

[1] N. Ziviani, *Projeto de algoritmos com implementações em Pascal e C*, Cengage Learning, 3a Edição Revista e Ampliada, 2011.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Algoritmos: Teoria e Prática*, Elsevier, Tradução da 3a Edição Americana, 2012.