Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II
2º Semestre de 2018
Profs. Luiz Chaimowicz, Gisele Pappa, Antônio Loureiro e Cristiano Arbex

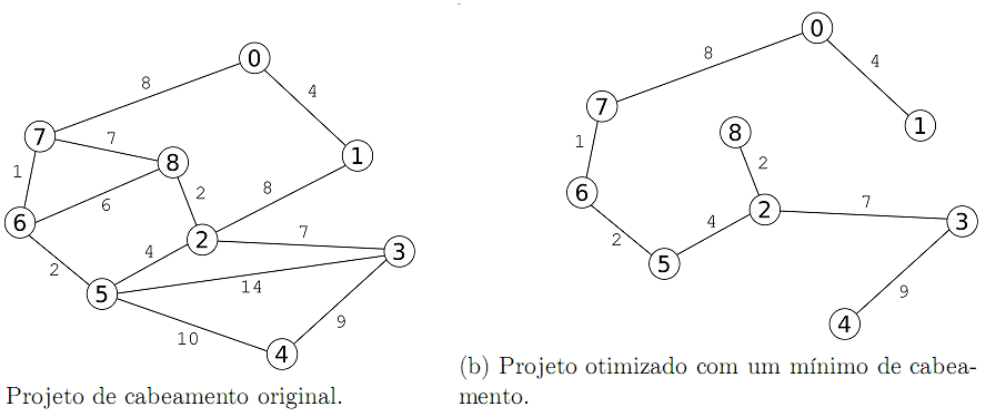
Trabalho Prático 2 – Árvore Geradora Mínima

Valor: 15 pontos
Data de devolução: 30/10/2018

O Departamento de Ciência da Computação (DCC) da UFMG planeja refazer a rede de computadores com fio que liga todas as salas do novo prédio. Um projeto inicial foi proposto, mas é 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). Note na figura que a árvore não contém ciclos, pois isto representaria um gasto desnecessário em cabos.



(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:

A forma mais simples de implementar um grafo com *N* vértices é definindo uma matriz *N x N* onde tanto a posição *(i, j)* quanto a posição *(j, i)* contêm o peso da aresta entre os vértices *i* e *j*. Se uma aresta não existe entre *i* e *j*, coloque um flag qualquer como, por exemplo, -1 na posição *(i, j)* para indicar isto. O problema desta implementação é que se muitas arestas não existem, há muito desperdício de memória.

Por isso, vamos utilizar uma estrutura alternativa: você deve implementar 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.

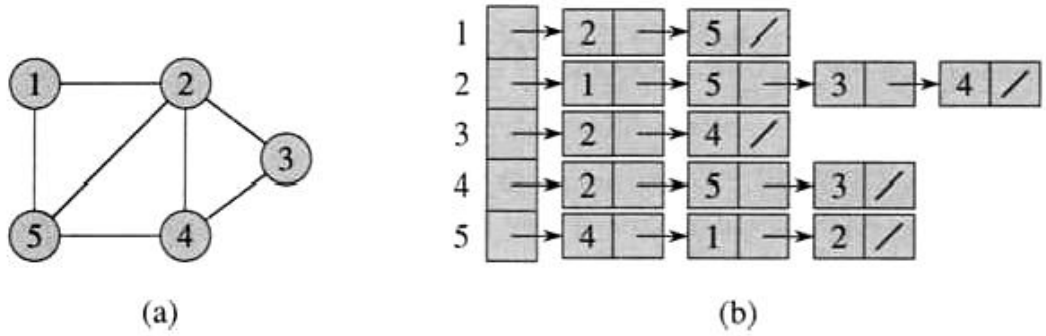


Figura 2: Exemplo de grafo não-direcionado (a) e sua representação usando lista de adjacência (b)

Existem vários algoritmos para encontrar a árvore geradora mínima de um grafo. Neste trabalho, você deve implementar o algoritmo de Krushkal, detalhado a seguir. O algoritmo 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 tanto para representar 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
```

Em resumo, o algoritmo funciona da seguinte forma. A cada passo, escolha a aresta de menor peso dentre as que ainda não foram escolhidas para entrar na árvore. Caso esta aresta conecte dois vértices que já estão na árvore (ambos na árvore, não apenas um deles), então escolha uma nova aresta, pois a inclusão dela criaria um ciclo e deixaríamos de ter uma árvore. Se você quiser mais detalhes sobre como implementar o algoritmo, procure o algoritmo de Kruskal na Internet.

Entrada e saída:

O seu programa irá ler de um arquivo texto 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 escrever em um arquivo 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) para cada aresta, o menor vértice deverá aparecer sempre antes do maior vértice, ou seja, para 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) as arestas deverão ser ordenadas em ordem crescente considerando seu menor vértice. Quando o menor vértice de duas arestas for o mesmo, a ordenação deverá considerar o 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.

Ao realizar seus testes, você deve ler a entrada de um arquivo (por exemplo, entrada.txt) e escrever a saída em um arquivo (por exemplo saida.txt). Você deve passar os nomes do arquivo de entrada e saída através de argumentos de linha de comando (exemplo Linux e Windows):

```
./executavel entrada.txt saida.txt

executavel.exe entrada.txt saida.txt
```

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 Moodle. Uma tarefa será criada para a entrega.

ATENÇÃO: O trabalho será avaliado em Linux. Evite utilizar bibliotecas que só compilam no Windows, como, por exemplo, windows.h. Se tiver na dúvida, pergunte a nós que podemos te sugerir alguma mudança caso necessário.

ATENÇÃO 2: Lembre-se que seu trabalho será avaliado por um corretor automático. É importante que a saída seja **exatamente** como pedido (sem enfeites desnecessários).

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.