SCC0503 (Algoritmos e Estruturas de Dados II) — Prof. Moacir P. Ponti Junior

Trabalho 2

Implemente sua atividade sozinho sem compartilhar, olhar código de seus colegas, ou buscar na Internet. Procure usar apenas os conceitos já vistos nas aulas.

Problemas de Planejamento Urbano

O prefeito de uma cidade do interior do Estado (cujo nome pretendo esquecer) está planejando melhorias para a população local. Entre as melhorias propostas, está a implantação de um hospital e o recapeamento de diversas ruas. Conversando com especialistas em teoria dos grafos, o prefeito descobriu estar diante de dois problemas clássicos: localização de facilidades e árvore geradora mínima.

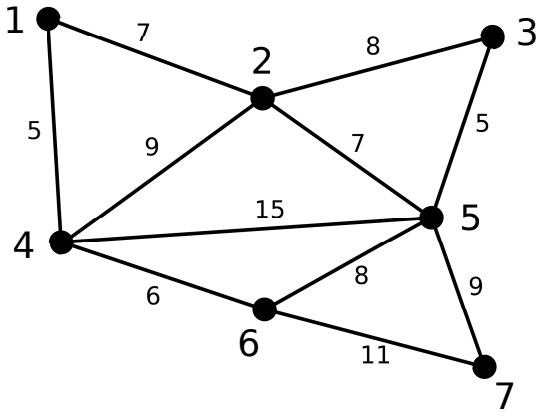

Figura 1: Exemplo de representação da cidade. Os vértices correspondem aos bairros e as arestas às ruas que conectam esses bairros.

Problema 1: Localização de Facilidades

Suponha que a cidade seja modelada por meio de um grafo $G = (V, A)$, conforme ilustrado na Figura 1, e cada vértice representa um bairro. O problema da localização de serviços consiste em determinar um local (no caso, um bairro) cuja distância aos demais bairros seja a menor possível. Deve-se, portanto, descobrir a distância mínima de um vértice u qualquer a todos dos demais vértices.

Assim, para o grafo da Figura 1, o vértice 5 representa o bairro onde o hospital deve ser construído, uma vez que sua localização, em termos de distância, é conveniente para todos os moradores (ou, em outras palavras, para todo vértice $v \in V$, $d(5, v)$ é a menor possível).

Problema 2: Árvore Geradora Mínima

As obras do hospital consumiram mais recursos que o previsto. Por isso, foi decidido que será necessário economizar no recapeamento das ruas. Assim, deseja-se ao menos garantir que todos os bairros estejam conectado entre si através de ruas em boas condições. Em outras palavras, deve ser possível partir de um bairro e chegar aos demais por meio de ruas em boas condições.

A Figura 2 mostra quais seriam as ruas recapeadas pelo prefeito.

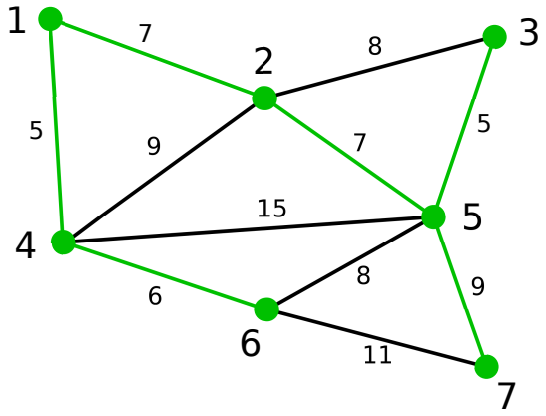

Figura 2: Mapa das ruas recapeadas (em verde).

Tarefa

Escreva um programa que auxilie o prefeito na tomada de decisões. Primeiramente, o programa deve ser capaz de apontar em qual bairro (vértice) o hospital deve ser implantado. Em seguida, deve apresentar as ruas que precisam ser recapeadas, levando em consideração a necessidade de manter o grafo conexo. Utilize como estrutura de dados para a conectividade dos vértices uma lista de adjacência.

Para o problema de localização de facilidades, utilize o algoritmo de Dijkstra e, para o problema de árvore geradora mínima, utilize o algoritmo de Prim.

Exemplo de entrada

A entrada do programa será o número de vértices seguido do número de arestas, seguidos das arestas que representam a conexão entre dois dispositivos e seus respectivos pesos. Na forma:

```
<n Vertices>\n
<m Arestas>\n
<u_1 v_1 p_1>\n
...
<u_M v_M p_M>
```

Por exemplo, para o grafo da Figura 1, temos:

```
7
11
1 2 7
1 4 5
2 3 8
2 4 9
2 5 7
3 5 5
4 5 15
4 6 6
5 6 8
5 7 9
6 7 11
```

Note que estamos trabalhando com grafos não dirigidos. Portanto, é necessário fazer o tratamento correto da entrada pois, caso seja fornecido a aresta `1 2`, subintende-se que `2 1` também exista.

Exemplo de saída

A saída deve apresentar o vértice no qual será instalado o hospital e a sequência de arestas inseridas na árvore geradora mínima. Por exemplo:

```
5
1 4
4 6
1 2
2 5
5 3
5 7
```

Fique atento à quebra de linha esperada. Para o exemplo anterior, as quebras de linha são da forma:

```
5\n
1 4\n
4 6\n
1 2\n
2 5\n
5 3\n
5 7\n
```

Caso haja empate na decisão sobre em qual vértice deverá estar o hospital, ambos os vértices devem ser impressos **um em cada linha** e respeitando a ordem *crescente* de numeração. Em caso de empate na escolha das arestas que serão colocadas na árvore, ou seja, caso haja duas arestas (u_i, v_i) e (u_j, v_j), ambas com mesmo peso, deve-se utilizar como desempate o seguinte critério: escolhe-se primeiro a aresta tal que $u_i < u_j$.

Critérios

O projeto será avaliado principalmente levando em consideração:

1. Processamento correto das entradas e saídas do programa;
2. Realização das tarefas descritas;
3. Bom uso das técnicas de programação;
4. Boa endentação e uso de comentários no código;
5. Boa estruturação e modularização do código.

Implementação

As seguintes funções devem, **obrigatoriamente**, ser implementadas:

- `dijkstra()`;
- `prim()`;
- `find_best_vertex()`;
- `print_best_vertex()`;
- `print_tree()`.

A função `find_best_vertex()` deve ser a responsável por determinar o(s) vértice(s) no(s) qual(is) o hospital pode ser construído. Esse(s) vértice(s) deve(m) ser impresso(s) pela função `print_best_vertex()`. Já a árvore gerada pelo algoritmo de Prim deve ser impressa pela função `print_tree()` e **não** pode ser impressa pela função `prim()`.

Deve-se observar que os parâmetros de entrada e o valor de retorno dessas funções podem variar, dependendo sa sua implementação. Sinta-se livre para utilizar a biblioteca implementada em aula.

Você poderá criar outras funções se quiser ou precisar lembre-se de tornar a função criada útil para os propósitos de reuso e abstração, e de comentá-la corretamente.

Restrições:

- Não deverá ser utilizada qualquer variável global.
- Não poderão ser utilizadas bibliotecas com funções prontas (a não ser aquelas para entrada, saída e alocação dinâmica de memória).
- Não serão aceitos trabalho que implementem o Algoritmo de Floy-Warshall e/ou o Algoritmo de Kruskal.
- Também serão desconsiderados os trabalhos que utilizam a estrutura de dados Matriz de Adjacência.

Uso de cabeçalhos e Makefile:

- O arquivo de cabeçalho (`.h`) deverá conter toda a implementação das funções para manipulação do grafo, incluindo estrutura de dados, percurso e outros.
- O arquivo fonte (`.c` ou `.cc`) deverá conter apenas o processamento referente à entrada e saída dos dados.
- Mais informações sobre como usar Makefiles e gerar bibliotecas pode ser encontrado em: `http://wiki.icmc.usp.br/images/0/0a/ApostilaMakefiles2011.pdf`

Dicas

As estruturas podem ser modificadas, de forma que o grafo contenha a lista de adjacência, e haja um arranjo de vértices para armazenar as informações de cada um. Uma sugestão:

```
struct digraph {
   int V;             // total de vertices
   int A;             // total de arestas
   VertexList *list;  // arranjo de vertices (a ser alocado)
};

typedef struct {
  int visited;   // -1 se vertice nao foi visitado ou maior ou igual a 0 se fo
  int pred;      // armazena indice do vertice predecessor no percurso
  Link adj;      // ponteiro para a lista de adjacencia
} VertexList;
```

ATENÇÃO

Leia atentemente os item a seguir:

- O projeto deverá ser entregue apenas pelo Sistema de Submissão de Programas (SSP)[1], escolhendo a opção `Trabalho2`. Há um casos de teste para serem baixados na página da disciplina. O sistema receberá trabalhos **apenas** entre os dias 02/05, às 0h00, e 09/05 as 23h59. Não serão aceitos trabalhos com atraso.
- Todos os arquivos do projeto (`Makefile`, arquivos `.h` e arquivos `.c`) deverão ser armazenados no mesmo diretório. Esse diretório deverá ser compactado com a extensão `.zip`. Não compacte os arquivos, mas sim o diretório. Isso facilita a correção e, portanto, agiliza a entrega das notas.
- Não utilize acentos nos nomes de arquivos e diretórios.

[1] `http://netuno.icmc.usp.br/ssp01/`

- Lembre-se: por padrão, a função `main()` deve ter valor de retorno. Assim, faça da seguinte maneira:

```
int main()
{
   ...
   return 0;
}
```

  que evita erros na execução do comando `make run`.

- Dúvidas conceituais deverão ser colocadas nos horários de atendimento. Dificuldades em implementação, por favor, envie e-mail para o estagiário PAE com o assunto `[trab2]duvida`, anexando o código, especificando o problema e apresentando o número USP.

- A detecção de cópia de parte ou de todo código-fonte, de qualquer origem, implicará reprovação direta no trabalho. Partes do código cujas **ideias** foram desenvolvidas em colaboração com outro(s) aluno(s) devem ser devidamente documentadas em comentários no referido trecho. O que **NÃO** autoriza a cópia de trechos de código nem a codificação em conjunto. Portanto, compartilhem ideias, soluções, modos de resolver o problema, mas não o código. Qualquer dúvida entrem em contato com o professor.