Ariane Machado Lima e Fernando Chiu Hsieh

EP 1 de Algoritmos e Estruturas de Dados II - Entrega: 07/05/2018

Exercício Programa 1 - versão 3

2018

-

1 Apresentação

Diversos contextos podem ser modelados com o uso de grafos. Por exemplo, um sistema de transporte no qual os vértices representam estações e as arestas conexões entre elas, ou uma caça ao tesouro na qual os vértices representam ilhas e as arestas rotas marítimas. Essa modelagem nos fornece uma base para resolver problemas relacionados a tais contextos, como 'Qual é a estação mais distante de determinado ponto?' ou 'Qual é a menor rota até a ilha do tesouro?'.

O algoritmo de Dijkstra nos permite computar o custo mínimo e o caminho de custo mínimo entre 2 vértices de um grafo ponderado direcionado de forma eficiente, desde que nenhuma aresta apresente custo negativo. Aplicado-o no contexto do sistema de transporte, podemos identificar o caminho com o menor custo de tempo por exemplo.

2 Algoritmo de Dijkstra

O algoritmo de Dijkstra armazena os vértices ainda não processados (inicialmente todos) em uma fila de prioridades Q. A prioridade de um dado vértice v é inversamente proporcional à estimativa atual da distância do vértice origem s até v ($d[v]$). Inicialmente $d[s] = 0$ e $d[v] = \infty$. Enquanto Q não estiver vazia, remove da fila Q o vértice v com maior prioridade (isto é, com menor $d[v]$) e atualiza as estimativas de distância $d[u]$ e de antecessor $Antecessor[u]$ de cada vértice u adjacente a v. Como já dito, $d[u]$ é a estimativa atual da distância do vértice origem s até u e $Antecessor[u]$ é o vértice antecessor no caminho de menor custo de s até u.

O algoritmo faz uso de uma estratégia gulosa e verifica se é possível melhorar o caminho para se alcançar u se passar por v. O pseudocódigo a seguir resume o algoritmo, que recebe como entrada um grafo direcionado ponderado G e um vértice origem s para a construção da árvore de caminhos de custos mínimos.

```
   Data: G = (V, E), s
1  foreach vértice v ∈ V do
2      d[v] = ∞;
3      Antecessor[v] = −1 ;
4  d[s] = 0 ;
5  Inicialize Q contendo todos os vértices de V;
6  while Q ≠ ∅ do
7      Remova v de Q tal que d[v] é mínimo (vértice de maior prioridade);
8      foreach adjacente u de v do
9      if d[u] > d[v] + pesoAresta(v, u) then
10         d[u] = d[v] + pesoAresta(v, u);
11         Antecessor[u] = v;
```

Para otimizar a velocidade do algoritmo, **você deve implementar Q como um heap binário para gerenciar a fila de prioridades baseado em $d[v]$.**

Mais detalhes sobre o algoritmo de Dijkstra e Heaps Binários encontram-se nos slides de aula e nas referências mencionadas em tais slides.

Seu EP deve utilizar a implementação de **grafos por listas de adjacências.**

3 Formatos dos arquivos fonte, de entrada e saída

Os arquivos fonte serão compilados e linkados de forma a gerar o programa executável `dijkstra`, conforme Makefile disponibilizado. Este Makefile assume a existência dos arquivos `grafo_listaadj.*`, `heap_binario.*` e `dijkstra.c`, ".*" representando ".c" e ".h".

Os arquivos `grafo_listaadj.*` implementam a estrutura de dados de grafos por listas de adjacências, e suas respectivas rotinas de manipulação RESPEITANDO RIGOROSAMENTE O PROTÓTIPO DESSAS ROTINAS DISPONIBILIZADO NO TIDIA, JUNTAMENTE COM O EP. Vocês podem adicionar código no `grafo_listaadj.h` de vocês, só não podem alterar os protótipos das funções. ALÉM DISSO, VOCÊS DEVEM ALTERAR APENAS O TIPOARESTA DE `int` PARA `double` !!!!!

Os arquivos `heap_binario.*` implementam a estrutura de dados de heap binário baseado no menor valor (menor a chave, maior a prioridade) e suas respectivas rotinas de manipulação.

O arquivo `dijkstra.c` implementa o algoritmo de Dijkstra de caminhos mínimos e possui a função `main` responsável por executar o algoritmo sobre uma dada entrada.

Esse programa será executado da seguinte forma:

dijkstra <arquivo de entrada> <arquivo de saída>

Exemplo:

```
dijkstra entrada_teste1.txt saida_teste1.txt
```

<arquivo de entrada>: Conterá na primeira linha o vértice origem s e nas linhas seguintes a representação do grafo direcionado ponderado G. O grafo será descrito utilizando o seguinte formato: a 1ª linha (1ª linha da descrição do grafo, 2ª linha do arquivo) conterá o número de vértices e arestas, nesta ordem; as demais linhas conterão as especificações das arestas.

Mais especificamente, o formato do arquivo será:

s

n m

o_1 d_1 $custo_1$

o_2 d_2 $custo_2$

...

sendo:

s (inteiro): número do vértice origem;

n, m (inteiros): número de vértices e de arestas, respectivamente;

o_i, d_i (inteiros): vértices origem e destino, respectivamente, da i-ésima aresta (i = 1...m);

$custo_i$ (double): custo (ou peso) da i-ésima aresta (i = 1...m)

<arquivo de saída>: Conterá o resultado da execução do programa, que será representado por |V| linhas. A i-ésima linha conterá os campos:

v_i d_i $Antecessor_i$

sendo:

v_i (inteiro): o número do i-ésimo vértice;

d_i (double): a distância do vértice s ao vértice v_i no caminho de custo mínimo; se essa distância for igual a infinito, escrever o valor correspondente a DBL_MAX.

$Antecessor_i$ (inteiro): o número do vértice antecessor do vértice v_i no caminho de

custo mínimo de s até v_i. se não houver antecessor, imprimir -1 (que é o combinamos ser VERTICE_INVALIDO).

Observações:

1. Os vértices do grafo serão indexados a partir de 1;

2. Todos os campos são separados por um espaço (tanto no arquivo de entrada quanto no arquivo de saída).

3. Nos casos em que a árvore de caminhos mínimos não é única, as arestas selecionadas por cada implementação poderão ser eventualmente diferentes. Basta que uma árvore correta seja fornecida que o programa será considerado correto.

4. Para representar um valor double infinito, use DBL_MAX definido na biblioteca <float.h>. Para imprimir o valor de DBL_MAX utilize a expressão "%.2e".

4 Entrega do trabalho

O trabalho DEVERÁ ser individual.

O código-fonte deverá ser escrito na linguagem C, compilável no compilador gcc.

Todos os arquivos fonte deverão estar em um diretório cujo nome deve ser o nome completo do aluno sem espaços. Ex: "ArianeMachadoLima". Esta pasta deve ser compactada usando o programa tar com a opção -z, gerando o arquivo compactado contendo o nome do aluno com a extensão `.tar.gz`.

Exemplo: pasta ArianeMachadoLima compactada com o comando:

```
tar cvz ArianeMachadoLima -f ArianeMachadoLima.tar.gz
```

Este arquivo compactado deverá ser postado no TIDIA, na Atividade "EP 1". O prazo para entrega é **07/05/2018**.

Evidência de plágio entre trabalhos não apenas implicará na nota zero no trabalho, como também sujeitará os alunos envolvidos às medidas disciplinares cabíveis.

Observação: Uma alternativa para desenvolver o exercício programa em windows é utilizar o Cygwin (https://www.cygwin.com/), que é uma coleção de ferramentas que implementam o ambiente linux para o Windows.