Trabalho Prático 3

Algoritmos I

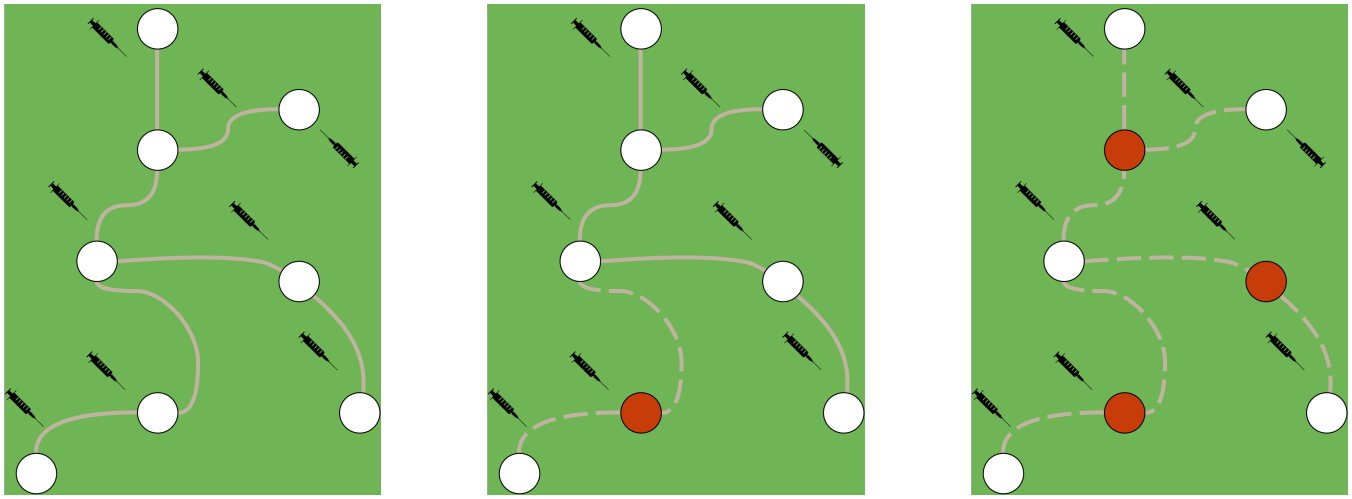
Entrega: 08/09/2021

1 Introdução

Dentro dos esforços de vacinação contra a COVID-19, a secretaria de saúde de um estado brasileiro que contém uma grande área florestal composta por pequenas vilas agrícolas necessita realizar a tarefa de construir depósitos de vacinas para as suas vilas.

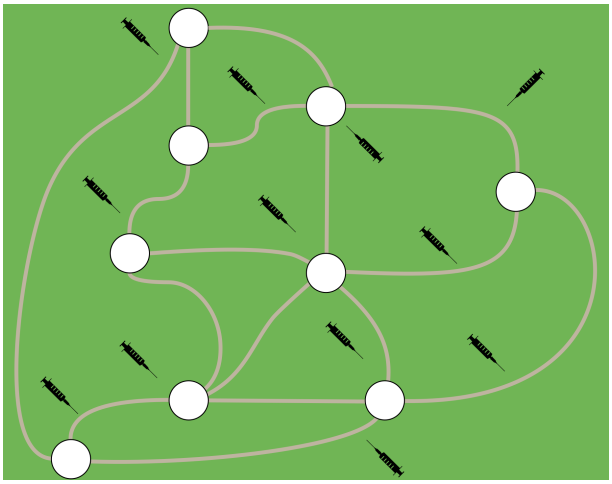
As vilas da região são definidas por um conjunto de pontos $P = \{p_1, p_2, \ldots p_n\}$ e um conjunto de caminhos $C = \{c_1, c_2. \ldots, c_m\}$, cada caminho conectando dois pontos $c_i = (p_j, p_{j'})$ como ilustrado na Figura 1(a). A partir de qualquer ponto da região é possível alcançar todos os demais pontos seguindo uma sequência de caminhos consecutivos $(c_k, c_{k+1}, c_{k+2}, ...)$, que são chamados de trilhas. A secretaria de saúde deverá construir depósitos de vacinas sobre a malha de vilas da região de forma que para todo caminho c_i existente entre duas vilas quaisquer (ou dois pontos) $(p_k, p_{k'})$ pelo menos uma das vilas tenha um depósito de vacinas 1(b) (caminhos tracejados). Um exemplo de construção de depósitos de vacinas pode ser visto na figura 1(c).

Em sua primeira tarefa, serão fornecidas malhas com caminhos que não formam ciclos, ou seja, existe somente uma única sequência de caminhos entre dois pontos quaisquer. Já em sua segunda tarefa os caminhos podem formar ciclos, exigindo maior atenção do órgão para não se perder. Um exemplo da primeira tarefa pode ser visto na Figura 1 e para a segunda tarefa na Figura 2.



(a) (b) (c)

Figura 1: Exemplificação do problema sem ciclos



(a)

Figura 2: Exemplificação do problema com ciclos

A secretaria de saúde é muito competente na administração de vacinas, mas não possui habilidades em computação. E ela não só quer completar a tarefa, como também a completar da forma mais eficiente possível. Ele quer construir os depósitos de vacinas para as vilas de forma a cobrir todos os caminhos entre as mesmas com o menor número de depósitos de vacinas. Você foi designado para resolver o problema.

2 O que fazer?

Formalmente, uma distribuição de depósitos válida sobre uma trilha $T = (P, C)$ é um subconjunto B de pontos tal que cada caminho da trilha seja incidente em pelo menos um ponto em B. O conjunto B será solução para a nossa primeira tarefa se for o menor dentre todas as distribuições válidas de T. Na primeira parte do trabalho você deverá encontrar **o menor número de depósitos** para atender às vilas, de forma que toda vila tenha um depósito nela mesma ou em uma vila conectada a ela. Observe que as trilhas da primeira tarefa **não possuem ciclos**. Seu programa deverá ler um arquivo contendo a representação das vilas e dos caminhos existentes e retornar a solução exata na saída padrão **em, no máximo, 2 segundos**.

Na segunda tarefa você terá que implementar uma heurística para o problema. Seu programa deverá retornar uma solução aproximada no **máximo duas vezes pior que a solução ótima**. Note que a solução ainda terá que ser válida embora possivelmente não mais a mínima. Semelhante à primeira tarefa, seu programa deverá ler um arquivo contendo a representação das vilas e dos caminhos existentes e retornar a solução aproximada na saída padrão. Vale lembrar que as trilhas agora **podem conter ciclos**, ou seja, mais de uma sequência de caminho entre quaisquer dois pontos. O retorno do programa deve ser dado **em, no máximo, 2 segundos**.

Seu programa deve receber **dois** parâmetros da linha de comando:

1. o identificador da tarefa ("*tarefa1*" (solução ótima na trilha sem ciclos) ou "*tarefa2*" (solução aproximada na trilha com ciclos)

2. o caminho para o arquivo da entrada (especificação da trilha)

3 O arquivo de entrada

O arquivo de entrada para os dois problemas segue o mesmo padrão. A única diferença é que para o primeiro caso temos um grafo sem ciclos e para o segundo não. A primeira linha contém dois inteiros n e m, sendo $0 < n \leq 10^5$ o número de vértices (pontos permitidos para construir depósitos) da trilha e $0 < m \leq n^2$ o número de trilhas entre dois vértices. As m seguintes linhas contem um par de vértices (u, v) que especificam se existe uma trilha entre u e v. Os identificadores dos vértices iniciam em 0 e vão até $n - 1$.

Entrada

```
9 8
0 1
1 2
1 3
3 4
4 5
3 6
6 7
```

4 O arquivo de saída

Há dois tipos de arquivos de saída. Para a primeira tarefa, o arquivo consiste apenas no número mínimo de depósitos necessários para a resolução do problema.

Saída Esperada para a tarefa1

```
3
```

Para a segunda tarefa, além do número de depósitos necessários, o identificador dos vértices que foram escolhidos para a solução devem ser impressos.

Saída Esperada para a tarefa2

```
3
1
4
6
```

É importante que a saída do seu programa siga essa formatação de saída para que a pontuação não seja zerada, não sendo permitido, portanto, incluir textos ou descrições sobre os números gerados na saída, dado que a correção será automatizada.

5 Avaliação Experimental

Para a avaliação experimental o aluno deverá criar entradas para o problema de forma a avaliar se o tempo de execução do algoritmo está de acordo com a complexidade reportada na documentação, aumentando o número de vértices e arestas da entrada, guardando os tempos de execução e colocando esses valores em gráficos para melhor visualização. É **necessário** executar mais de uma vez o algoritmo para entradas com o mesmo tamanho de pontos e caminhos (no mínimo 5 vezes), guardando o tempo de execução e reportando a **média** e o **desvio padrão** deste tempo. Isso é necessário para observar como o tempo está se comportando dado diferentes execuções e diferentes configurações da trilha.

Você também deverá construir algumas instâncias pequenas para os casos em que o grafo contém ciclos e mostrar qual a solução ótima do problema e qual a solução que seu algoritmo forneceu, comparando se o resultado está dentro da restrição imposta no problema (a solução aproximada não deve ser maior que 2 vezes a solução ótima).

6 O que deve ser entregue

Você deve submeter um arquivo compacto (**zip**) no formato **seu_nome_sua_matrícula** via Moodle contendo:

- todos os arquivos do código *.c* e *.h* que foram implementados,
- um arquivo *makefile*[1] **que crie um executável tp3 em ambiente Linux**,
  - **ATENÇÃO:** O makefile é para garantir que o código está sendo compilado corretamente, de acordo com o modo que vocês modelaram o programa. É **essencial** que ao digitar "make" na linha de comando dentro da pasta onde reside o arquivo makefile, o mesmo compile o programa e gere um executável chamado **tp3**. Portanto, gere seu executável usando o comando "make" para se certificar que ele funciona
- sua documentação.

Sua documentação deve ter até 10 páginas contendo:

- uma breve descrição do problema,
- explicações das estruturas de dados e dos algoritmos utilizados para resolver o problema. Para tal, artifícios como pseudo-códigos, exemplos ou diagramas podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. Não inclua textos de códigos em sua documentação.
- análise de complexidade da solução proposta (espaço e tempo). Cada complexidade apresentada deverá ser devidamente justificada para que seja aceita.
- prova de corretude do algoritmo.
- avaliação experimental.

O seu TP deverá ser entregue de acordo com a data especificada no moodle. A penalidade em porcentagem para os TPs atrasados é dada pela fórmula $2^{d-1}/0.16$.

[1] `https://pt.wikibooks.org/wiki/Programar_em_C/Makefiles`

7 Implementação

7.1 Linguagem, Ambiente e Parâmetros

O seu programa deverá ser implementado na linguagem **C** e poderá fazer uso de funções da biblioteca padrão da linguagem. Trabalhos que utilizem qualquer outra linguagem de programação e/ou que façam uso de bibliotecas que não a padrão não serão aceitos. Além disso, certifique-se que seu código compile e execute corretamente nas máquinas **Linux** dos laboratórios do **DCC**.

ATENÇÃO: O tipo da tarefa (*tarefa1* ou *tarefa2*) e o arquivo da entrada deve ser passado como parâmetros para o programa através da linha de comando (e.g., *$ ./tp3 tarefa1 trilha1.txt*) e imprimir a saída no **stdout** (com printf), não em um arquivo.

ATENÇÃO: certifique-se que o programa execute com os três arquivos passados como exemplos junto com a documentação no Moodle (dataset_tp3.zip), sem alterar os arquivos passados. Isso vai garantir que seu programa vai conseguir ler corretamente os arquivos do corretor automático.

7.2 Testes

A sua implementação passará por um processo de correção automática, o formato da saída de seu programa deverá ser idêntico aquele descrito nas seções anteriores. Saídas diferentes serão consideradas erro para o programa. Para auxiliar na depuração do seu trabalho, será fornecido um pequeno conjunto de entradas e suas saídas. É seu dever certificar que seu programa atenda corretamente para qualquer entrada válida.

7.3 Qualidade do código

É importante prestar atenção para a qualidade do código, mantendo-o organizado e comentado para não surgir dúvidas na hora da correção. Qualquer decisão que não estiver clara dada a documentação e a organização do código será descontada na nota final.

8 Consideração Final

Assim como em todos os trabalhos dessa disciplina é estritamente proibido a cópia parcial ou integra de códigos, seja da internet ou de colegas. Seja honesto! Você não aprende nada copiando código de terceiros. Se a cópia for detectada, sua nota será zerada e o professor será informado para que as devidas providências sejam tomadas.