Trabalho Prático 1

Algoritmos I

Fernando Eduardo Pinto Moreira - 2019054536

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

fernandoepm@ufmg.br

1. Introdução

O problema proposto foi implementar um sistema de logística para entrega de vacinas contra a COVID-19 em postos de vacinação de uma determinada cidade. Tais vacinas são armazenadas em centros de distribuição, onde a temperatura, que precisa ficar abaixo dos -60°C, é mantida. Porém, ao se deslocar tais vacinas para os postos de vacinação, a temperatura pode alterar e, com base nisso, o objetivo deste trabalho é calcular quantos e quais postos são alcançáveis, além de determinar se há alguma rota que percorra um mesmo posto mais de uma vez.

2. Implementação

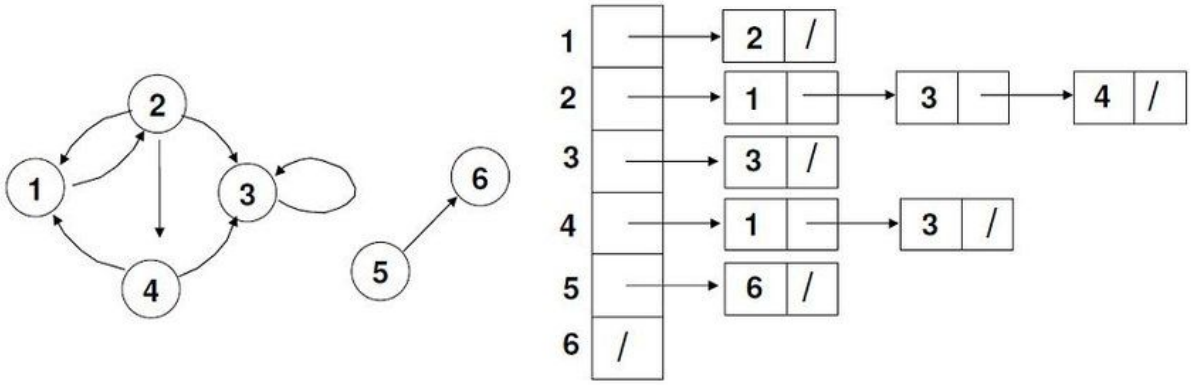
2.1. Estrutura de Dados

A implementação do programa teve como base um grafo como estrutura de dados. A escolha de tal estrutura se deu devido à especificação do problema, em que se devia calcular a distância de certos postos de vacinação aos centros de distribuição e checar se havia alguma rota que percorresse um mesmo posto mais de uma vez.

O grafo, então, foi implementado utilizando listas de adjacência em detrimento de uma matriz de adjacência, uma vez que cada posto ou centro pode ter nenhum, um ou mais postos adjacentes, ou seja, aplicar listas de adjacência é mais viável em termos de espaço, pois não precisaremos de uma matriz com um número fixo de linhas e colunas.

Tal estrutura foi feita com um vector de vector, (vector<vector<int>>), sendo criadas duas dessas estruturas, uma para os Centros de Distribuição e outra para os Postos de Vacinação.

Abaixo, uma ilustração de como um Grafo implementado com listas de adjacência funciona:



2.2. Modelagem computacional

A ideia geral do código é, a partir de uma quantidade de centros de distribuição, uma quantidade de postos de vacinação e o incremento de temperatura, ler do usuário os vértices adjacentes dos centros de distribuição e dos postos de vacinação e, então, calcular a quantidade e quais postos são alcançáveis, além de checar se há alguma rota que percorre um mesmo posto mais de uma vez.

Com isso, é feita essa leitura e, após isso, uma função que checa quais postos são alcançáveis foi chamada. Esta função é recursiva e se chama "dfs". Esta função implementa o algoritmo "Depth-First Search" em grafos, pegando todos os postos que estão dentro de uma distância k, sendo k o valor da distância máxima que pode ser percorrida. Esta distância máxima é calculada através da temperatura por meio da função "calculaDistMax". Tal função "dfs" insere no array criado anteriormente (e com todos os seus elementos sendo igual a zero) podendo haver repetições. Este array então é trabalhado no main, passando os seus valores para um vector e, após isso, passando os valores deste vector para um set. Isso foi necessário para que não houvesse repetições de valores e para que a saída fosse impressa de forma ordenada. Com isso, todos os postos alcançáveis são impressos na tela. Porém, antes de imprimir esses valores, é impresso a quantidade de postos alcançáveis, que é realizada bastando pegar o tamanho do set e imprimir este tamanho.

Após realizada a impressão de quantos e quais postos são alcançáveis, deve-se calcular se há alguma rota que percorre um mesmo posto mais de uma vez, e isso foi feito pela função "possuiCiclo". Esta, primeiramente, marca todos os vértices como não visitados e, após isso, chama a função recursiva "possuiCicloUtil", que começa marcando o vértice analisado como visitado e depois é chamada recursivamente para

os vértices filhos do analisado. Caso, nesse processo de chamar os filhos, o programa encontrou um vértice já visitado, então conclui-se que há um ciclo e o programa retorna verdadeiro. Caso contrário, retorna falso. Este processo é feito para todos os vértices que representam os postos de vacinação.

Caso o valor da distância máxima seja igual a 0, o programa não executa nenhuma dessas funções e imprime na tela "0", "*" e "0".

3. Análise de Complexidade de Tempo

Começaremos a análise pela leitura da quantidade de centros de distribuição, da quantidade de postos de vacinação e do incremento de temperatura. Todas essas operações possuem custo constante, ou seja, O(1). Após isso, os postos adjacentes de cada centro e de cada posto são lidos, com também custo constante O(1).

Posteriormente, a função "dfs" é chamada para cada um dos postos adjacentes de cada um dos centros. Logo, a complexidade é uma constante vezes a complexidade da função "dfs". Esta função "dfs" é recursiva e, para cada chamada, a distância máxima k é reduzida em 1. Pode-se notar que nenhum vértice é percorrido mais de duas vezes. Portanto, a complexidade de tempo desta função é O (n).

Após verificar quantos e quais postos são alcançáveis, o programa então chama a função "contemCiclo" para checagem de ciclos. Esta função é da ordem do número de vértices, que neste caso é igual à quantidade de postos de vacinação mais o número de arestas. Portanto, a complexidade é linear, ou seja, O(n).

Assim, a complexidade do total do programa é:

$$O(1) + O(1) + O(n) + O(n) = \mathbf{O(n)}$$

4. Pseudo-código

```
caculaDistMax (incrementoTemp):
   Retorne 30/incrementoTemp

dfs (k, vertice, verticePai, tree, vetor, contador):
   If (k < 0) Retorne
   Endif
   Passa o valor de vertice para vetor[contador]
   Adiciona 1 ao contador
   For (i até vertice[contador])
      If (i é diferente de verticePai)
```

```
        Chama dfs (k - 1, i, vertice, tree, vetor, contador)
      Endif
    Endfor

possuiCicloUtil (v, visitado[], pilhaRec, tree):
    If (vértice não está marcado como visitado)
      Marque o vértice como visitado
      For (i igual ao início do vetor tree até o final do vetor tree)
        Chame possuiCicloUtil (i, visitado, pilhaRec, tree)
        If (vértice não está marcado como visitado e possuiCicloUtil é verdadeiro)
          Retorne Falso
        Else if (pilhaRec na posição vértice é verdadeiro)
          Retorne Verdadeiro
        Endif
      Endfor
      Remove o vértice da pilha de recursão
      Retorne Falso

possuiCiclo (qntVertices, tree):
    Marca todos os vértices como não visitado e como não parte da pilha de recursão
    For (i=1 até qntVertices)
      Chama possuiCicloUtil(i, visitado, pilhaRec, tree)
      If (possuiCicloUtil é verdadeiro)
        Retorne Verdadeiro
      Endif
    Endfor
    Retorne Falso

main():
    Inicializa CD como um vector de vector
    Inicializa PV como um vector de vector
    Lê quantidade de centros, quantidade de postos e incremento de temperatura
    Chama função distMax para calcular distância máxima dado a temperatura
    Inicializa vector line_CD como uma linha do vector de vector CD
    Inicializa vector line_PV como uma linha do vector de vector PV
    Inicializa set setPV
    For (i=0 até quantidade de centros)
      Lê os postos adjacentes a cada Centro
    Endfor
    For (i=0 até quantidade de postos)
```

```
    Lê os postos adjacentes a cada Posto
    Adiciona os postos adjacentes em um vector de vector auxiliar tree
  Endfor
  If (distância máxima > 0)
    Inicializa array auxiliar com todos os elementos iguais a 0
    Inicializa vector auxiliar para armazenar os postos alcançáveis
    If (quantidade de centros = 1)
      Chama dfs (distMax-1, 1, -1, tree, vetor, contador)
    Else if (quantidade de centros > 1)
      Chama dfs (distMax-1, x, -1, tree, vetor, contador), sendo x cada posto
adjacente a cada Centro
    Endif
    Copia todos os elementos do array para o vector de postos alcançáveis
    Copia todos os elementos do vector de postos alcançáveis para o set
    Chama possuiCiclo para todos os postos de vacinação
    If (possuiCiclo = verdadeiro) imprima 1
    Else imprima 0
    Endif
  Else
    Imprima 0, *, 0
  Endif
```

5. Conclusão

Pode-se notar, depois da implementação do programa que, desde a escolha da estrutura de dados, até a implementação do programa, tudo é pensado para que o código funcione e compile de forma desejada, mas tão importante quanto isso, que este seja eficiente e que tenha o menor custo possível. Por isso, foi utilizado um grafo por meio de listas de adjacência que possui algumas operações com menor custo.

Além disso, percebe-se que o problema final foi resolvido e a logística foi realizada com sucesso, isto é, dependendo do incremento da temperatura, podemos saber quantos e quais postos poderão ser alcançados e ainda sabemos se há alguma rota que percorre um mesmo posto mais de uma vez, ajudando assim no combate a este vírus tão cruel que vem nos assolando há vários meses.