INSTITUTO FEDERAL DE MINAS GERAIS

Bacharelado em Ciência da Computação

Disciplina: Matemática Discreta

Trabalho Prático

Professor: Diego Mello da Silva

Formiga-MG
28 de novembro de 2018

Sumário

1 Informações Gerais

Este documento descreve a especificação do Trabalho Prático da disciplina Matemática Discreta, e deve ser seguido de forma a contemplar os itens considerados na avaliação por parte do professor da disciplina. O trabalho deve ser feito em **grupo de até 3 alunos** e tem o valor de **30 pontos**, distribuídos na forma de requisitos que devem ser contemplados para completar o trabalho prático. A data limite do trabalho deverá ser combinada em sala de aula. O trabalho deverá ser entregue até as **23:59 hs da data limite**. Trabalhos entregues após este prazo serão desconsiderados e não serão corrigidos. O documento é organizado como segue. Na Seção 2 será apresentado o problema que este trabalho prático pretende resolver; na Seção 3 serão apresentados os requisitos de *software* que, se implementados, permitirão resolver o problema proposto; na Seção 4 serão apresentados os critérios de avaliação do trabalho e respectiva pontuação. **Trabalhos com suspeita de plágio, cópia ou adaptação de códigos fontes pré-existentes não autorizado pelo professor da disciplina serão desconsiderados e valerão zero. O professor reserva-se ao direito de não corrigir trabalhos que (i) não compilam na plataforma alvo da correção (Linux Ubuntu ou similar), (ii) que possuem problemas de *memory leaky*, (iii) problemas de alocação dinâmica de memória, ou (iv) que não possuem documentação suficiente para compilação e execução. Nestes casos, a nota do trabalho será zero**. O código fonte e arquivos anexos devem ser enviados para o professor em tarefa a ser disponibilizada na plataforma Google Classroom pelo professor da disciplina.

2 O Problema Proposto

Esta seção descreve um problema que será apelidado de ***Problema da Mínima Difusão Parcial em Grafos Direcionados*** (PMDPGD), um problema novo que foi proposto e discutido em sala de aula. É importante dizer que nenhuma investigação prévia foi feita sobre o problema sob o ponto de vista de fundamentos teóricos, de sua dificuldade (i.e., pertinência à classe de problemas $\mathcal{NP}$-Difícil), da existência de algoritmos eficientes para resolvê-lo, da existência de casos especiais que sejam de fácil resolução, nem de sua aplicação prática para modelar situações reais sendo, portanto, utilizado apenas como *estudo de caso* para aplicação de algoritmos e estruturas de dados que manipulam objetos que se relacionam representadas por relações $R \subseteq A \times A$, e descrito a partir de breves *insights* feitos em colaboração com os alunos em sala de aula.

Os conceitos necessários serão apresentados a seguir. Na subseção 2.1 serão discutidos brevemente os conceitos sobre grafos, e sua representação computacional com ênfase sobre a representação por matriz de incidência. Na subseção 2.2 serão apresentados, de maneira um pouco mais formal, algumas definições importantes para compreender o PMDPGD. Na suseção 2.3 o PMDPGD será apresentado mais formalmente. Algumas propostas de algoritmos para solucionar o PMDPGD serão esboçadas na subseção 2.4. Na seção 2.5 serão apresentados um conjunto de arquivos que serão usados para validar experimentalmente os algoritmos propostos. Por fim, a seção 2.6 discutirão algumas competências que espera-se que o grupo desenvolva ao término do trabalho prático.

2.1 Grafos, Definições e sua Representação Computacional

Grafos são estruturas matemáticas usadas para representar relacionamento entre coisas. Um grafo $G = (V, E)$ é representado por um conjunto V de vértices ou nós, e um conjunto E de arcos ou arestas que ligam nós do conjunto V. Trata-se de uma estrutura combinatória muito utilizada em Ciência da Computação para modelar diversas situações do mundo real relacionadas a problemas envolvendo roteirização (de mensagens em uma rede, de veículos em rodovias, de trilhas e pontos de solda em placas de circuitos impressos, de rotas aéreas), fluxos (de corrente elétrica, de produtos em uma rede logística, de distribuição de água e energia, etc), atribuição (de tarefas a executores, de profissionais em locais de demanda, de disciplinas à sala de aulas, etc), para representação de redes sociais, dentre outras aplicações. De maneira formal, um grafo pode ser definido como

Definição 1 (Grafo) *Um grafo $G = (V, E)$ consiste em um conjunto* ***não vazio*** *de* ***vértices*** *(ou* ***nós****) V e um conjunto de* ***arestas*** *(ou* ***arcos****) E. Cada arco tem um ou dois nós associados a ela, denominados de* ***extremidades****. Um arco liga suas extremidades.*

Um grafo pode ser representado graficamente por círculos representando os nós da rede, com arcos ligandos estes nós quando existem relacionamentos entre eles. Um grafo pode ser direcionado se o arco que representa os objetos que se relacionam tiver uma orientação de 'quem se relaciona com quem'. De maneira formal, denominados grafos direcionados de dígrafos, que são definidos como

Definição 2 (Grafo Direcionado, Grafo Orientado ou Dígrafo) *Um grafo* ***orientado*** *(direcionado ou dígrafo) $G = (V, E)$ consiste em um conjunto não vazio de vértices (ou nós) V e um conjunto de arestas (ou arcos) orientados E. Cada arco orientado está associado a um* ***par ordenado*** *de nós (u, v). Tal arco orientado começa em u e termina em v.*

Para exemplificar ambos os casos, sejam os grafos dados pela Figura 1 que modela o mapa com 128 cidades da América do Norte, Figura 2 que modela o mapa com 5565 sedes de município do Brasil, Figura 3 que modela uma rede de *backbones* localizado no Brasil, Figura 4 que modela uma rede de distribuição de água de uma determinada localidade onde a água é distribuída a partir de uma planta de tratamento de água em direção à tanques e locais de redistribuição cujo modelo considera a capacidade de fluxo de água nos dutos da rede e, por fim, Figura 5 que modela o grafo das páginas web de um site hipotético. Em todos os casos, independente do contexto onde foram usados, eles modelam relacionamentos existentes entre os elementos mapeados sob a forma de rede, ora considerando orientação de quem se relaciona com quem, ora assumindo que se i relaciona com j então j também se relaciona com i.

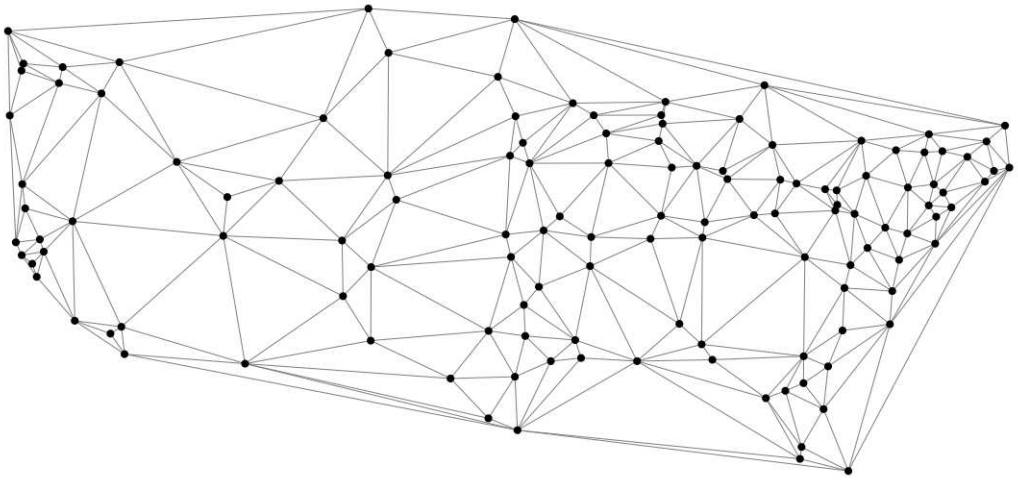


Figura 1: Mapa de 128 cidades da América do Norte segundo algum relacionamento que as interliga

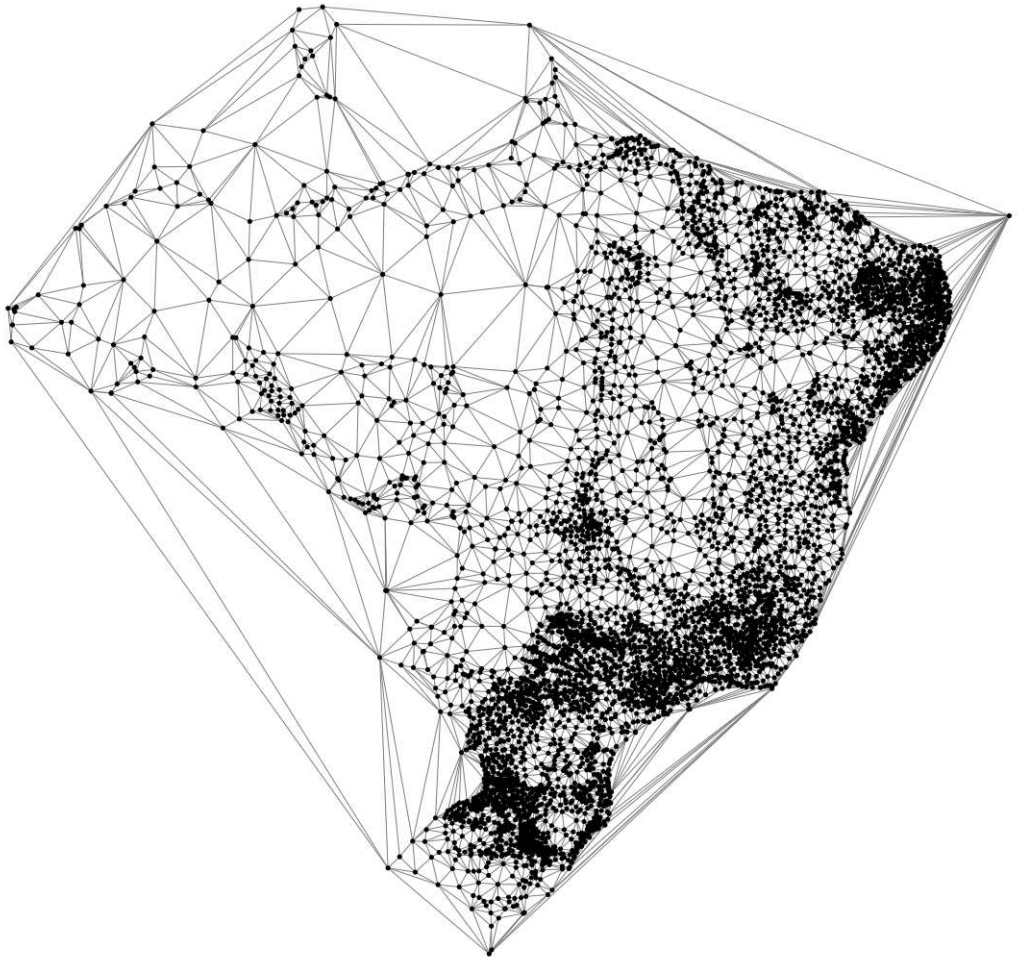


Figura 2: Mapa de 5565 sedes de município do Brasil, resultado de aplicação de algoritmo de triangulação e envoltória convexa

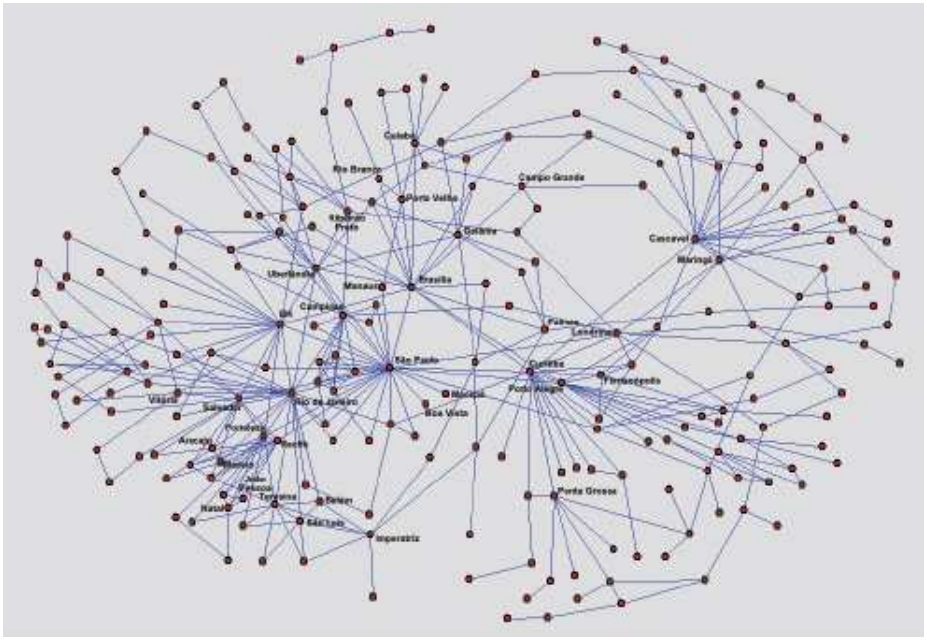


Figura 3: Topologia da rede de *backbones* do Brasil mostrando como se concentra a distribuição de conectividade nos pontos mais importantes

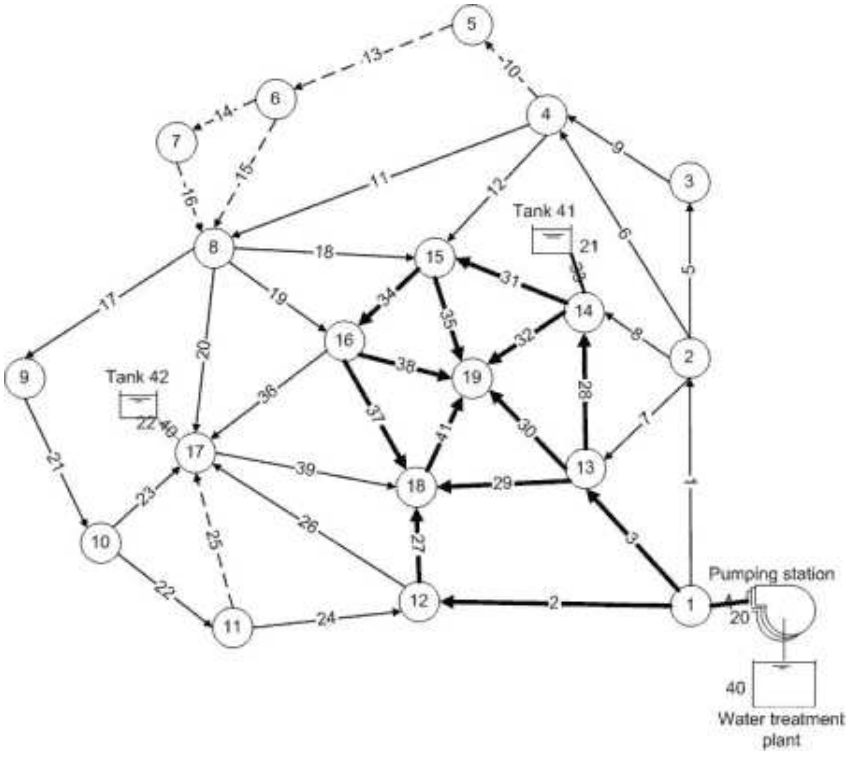


Figura 4: Rede de distribuição de água de uma Planta de Tratamento de Água com Estação de Bombeamento para pontos de redistribuição e tanques. Neste exemplo os arcos do grafo representam os dutos de distribuição que são rotulados com a capacidade ou vazão que suportam, assim como a seta do arco define o sentido da distribuição da água. Note a presença de nós do tipo 'fonte' e do tipo 'sorvedouro' no grafo que representa a rede de distribuição

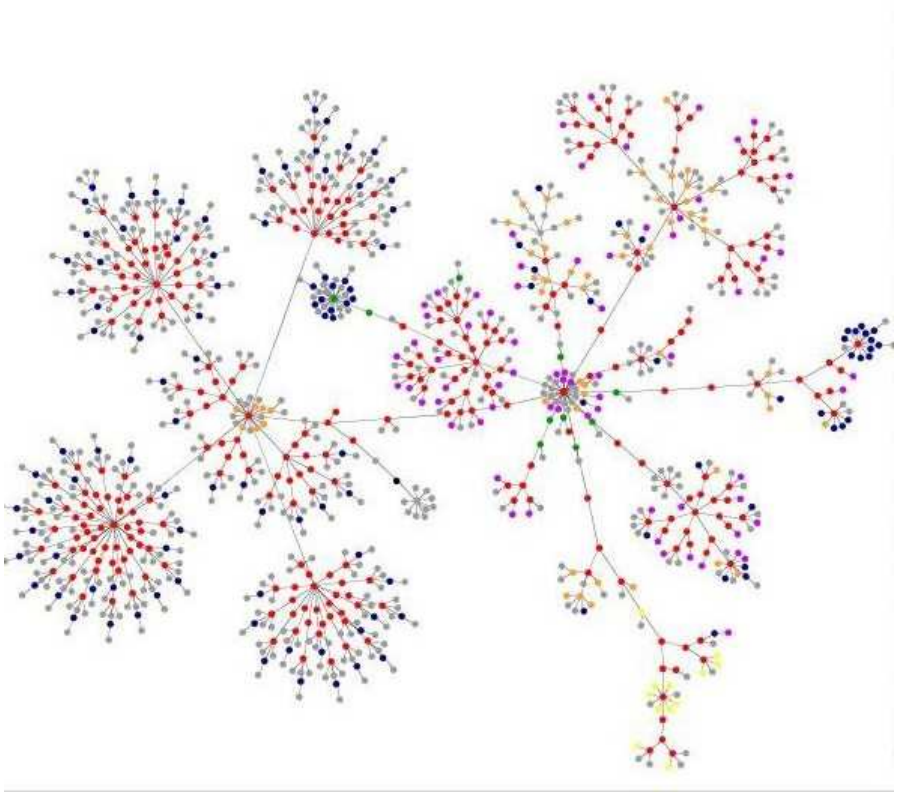
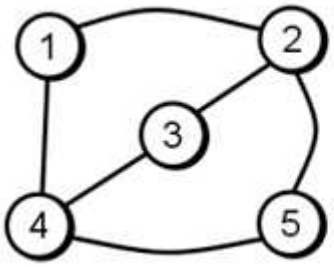


Figura 5: Páginas de um *site* e *hyperlinks*, representando um provável diagrama navegacional entre as páginas e para quais sites ela provê acesso

Sob o ponto de vista de implementação, grafos são também estruturas de dados úteis para armazenar os relacionamentos entre coisas em sistemas computacionais, implementados sob três formas clássicas: **lista de adjacência**, **matriz de incidência** e **matriz de adjacência**. Para o propósito deste trabalho apenas a matriz de adjacência será abordada neste documento. Uma matriz de adjacência é uma forma simples de representar grafos não direcionados. Trata-se de um arranjo bidimensional em memória que possui dimensão $n \times n$, onde n é a a cardinalidade de $|V|$. As linhas e colunas representam os índices dos nós. Dado a linha i e coluna j da matriz, se o elemento da posição (i, j) da matriz vale 1 então existe um arco que liga os nós i e j em G (isto é, $(i, j) \in E$). De maneira similar, se o elemento da posição (i, j) for igual a 0 não existe relacionamento entre os nós correspondentes no grafo G. Por representar um grafo não direcionado, se o elemento da posição (i, j) da matriz de adjacência vale 1, então o elemento da posição (j, i) também vale 1 (isto é, ela é simétrica). Para exemplificar, seja a Figura 6 que possui um grafo de exemplo, com sua matriz de adjacência correspondente. Observe que o grafo possui 5 nós; logo a matriz de adjacência será de dimensões 5×5. Verifique a correspondência que há entre os elementos da matriz e a presença de arcos no grafo G.



	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Figura 6: Representação do grafo G **não-direcionado** por matriz de adjacência

Seja agora uma **adaptação da matriz de adjacência para representação de grafos direcionados**. Nesta adaptação, se existe um arco direcionado que parte do nó i em direção ao nó j no grafo G, então na matriz correspondente teremos o valor 1 definido na posição (i, j) da matriz; caso contrário teremos 0. Observe que nesta adaptação da matriz de adjacência não há simetria, a menos que para cada 'arco de ida' exista um 'arco de retorno' (isto é, se existe valor 1 na posição (i, j) da matriz que representa o grafo, não necessariamente haverá um valor 1 na posição (j, i), a menos que na topologia original do grafo exista o 'arco de retorno' correspondente). Desta forma, podemos adaptar o conceito de matriz de adjacência para representar grafos direcionados. Seja um grafo direcionado G. De maneira geral, se existe um arco que parte do nó i para o nó j, então na i-ésima linha, j-ésima coluna da matriz que representa G espera-se existir um valor 1; caso contrário espera-se valor 0, que representa o caso onde não existe tal arco (i, j) em G.

Para finalizar os fundamentos sobre grafos, sejam as seguintes definições que serão úteis para discutir a variante do problema apresentado.

Definição 3 (Grau de um Nó de um Grafo Não-Orientado) *O **grau** de um nó v de um grafo não orientado, indicado por $\delta(v)$ é o número de arcos incidentes a ele, exceto um laço em v, que contribui duas vezes para o grau de v e deve ser contado apenas uma vez.*

A generalização acima serve apenas para grafos não orientados, onde a presença do arco ligando os nós i e j indica que tanto i relaciona com j, quanto j relaciona com i. Para o caso de dígrafos, é preciso considerar a contabilização do grau em dois casos separados: quando os arcos partem de um nó v arbitrário, e quando os arcos chegam em v. Formalmente,

Definição 4 (Grau de um Nó de um Dígrafo) *Em um dígrafo, o **grau de entrada** de um nó v, indicado por $\delta_{in}(v)$ é o número de arcos que tem v como nó final. O **grau de saída** de um nó v, indicado por $\delta_{out}(v)$, é o número de arestas que tem v como nó inicial.*

A partir dessa definição de grau de dígrafos, podemos classificar um dado nó $v \in V$ do grafo direcionado de acordo com o grau de entrada e de saída de v. Seja a

Definição 5, que lida com nós especiais de um dígrafo G que são caracterizados por serem apenas extremidades de origem de arcos:

Definição 5 (Nó Fonte) *Em um dígrafo $G = (V, E)$, um nó $v \in V$ é denominado **nó fonte** se não existem arcos que chegam em v, mas existem arcos que partem de v para outros nós $k \in V$, com $k \neq v$. Em outras palavras, um nó $v \in V$ é fonte se as seguintes condições forem garantidas:*

- $\exists k \in V, k \neq v\big((v, k) \in E\big)$
- $\delta_{in}(v) = 0$
- $\delta_{out}(v) \neq 0$

Assim, um nó fonte é um nó do grafo de onde originam-se ligações para outros nós. De maneira similar existe um tipo de nó de interesse para o problema que são apenas extremidades que chegam arcos, formalizados pela Definição 6:

Definição 6 (Nó Sorvedouro) *Em um dígrafo $G = (V, E)$, um nó $v \in V$ é denominado **nó sorvedouro** se não parte nenhum arco originado em v, mas chegam arcos originados de outros nós $k \in V$ direcionados para v. Em outras palavras, um nó $v \in V$ é sorvedouro se as seguintes condições forem garantidas:*

- $\forall k \in V, k \neq v\big((v, k) \notin E\big)$
- $\delta_{in}(v) \neq 0$
- $\delta_{out}(v) = 0$

Um nó $v \in V$ que não é nem um nó fonte, nem um nó sorvedouro, é um nó intermediário da rede. De posse dessas informações pode-se revisitar o grafo apresentado na Figura 4. Nele, o nó rotulado como 1 é um nó fonte; os nós rotulados como 17 e 19 são nós sorvedouros; e os demais nós são intermediários.

2.2 Definição de Difusão e Outros Conceitos Importantes

Seja $G = (V, E)$ um grafo direcionado onde V é o conjunto de vértices ou nós do grafo, e E é o cojunto de arcos onde cada arco $(i, j) \in E$ representa um relacionamento existentes do nós $i \in V$ para o nó $j \in V$. Seja um **nó influenciador** um nó arbitrário $v \in V$ onde existe pelo menos um arco que parte de v em direção à outro nó $k \in V$, seja ele um nó fonte ou intermediário. Formalmente,

Definição 7 (Nó Influenciador) *Em um dígrafo, um nó $v \in V$ é denominado de* ***nó influenciador*** *se $\exists k \in V\big((v,k) \in E\big)$, com $k \neq v$, e $\delta_{out}(v) \neq 0$.*

De maneira análoga, um **nó influenciado** é um nó que é uma extremidade de chegada de um arco no dígrafo G. Formalmente,

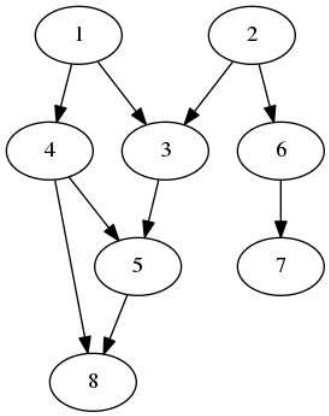
Definição 8 (Nó Influenciado) *Em um dígrafo, um nó $v \in V$ é denominado de* ***nó influenciado*** *se $\exists k \in V\big((k,v) \in E\big)$, com $k \neq v$, e $\delta_{in}(v) \neq 0$.*

De posse dos conceitos de nó influenciador e de nó influenciado, apresentaremos o **conceito de difusão** proposto neste trabalho.

Definição 9 (Difusão) *Seja $D \subseteq V$ algum subconjunto formado apenas por nós influenciadores. Uma difusão é uma propagação recursiva de informações originada em um nó influenciador $v \in D$, que difunde-se adiante para cada nó k atingível a partir de v.*

Entende-se por propagação recursiva todo procedimento que repete a propagação da informação recebida pelo nó k para os nós que são atingíveis diretamente por k, isto é, aqueles que são extremidades de chegada dos arcos que partem de k. Uma difusão encerra quando, em algum momento da propagação adiante, atinge-se um nó sorvedouro que não tem para quem difundir a informação recebida.

Seja o conjunto $A(v)$ o conjunto formado por todos os **nós que são atingíveis** por meio de difusão, originando-se em v. São computados considerando todos os nós que são extremidades de chegada de um arco acessados diretamente de v, ou como extremidade de chegada de algum arco que parte de outro nó k atingível por v por propagação adiante. Para exemplificar, seja a Figura 7. Na imagem da esquerda, um grafo é apresentado com 8 nós. Na direita, os conjuntos dos nós atingíveis a partir de cada nó $v \in \{1,2,3,4,5,6,7,8\}$.



$A(1) = \{4,8,5,3\}$
$A(2) = \{3,5,8,6,7\}$
$A(3) = \{5,8\}$
$A(4) = \{5,8\}$
$A(5) = \{8\}$
$A(6) = \{7\}$
$A(7) = \varnothing$
$A(8) = \varnothing$

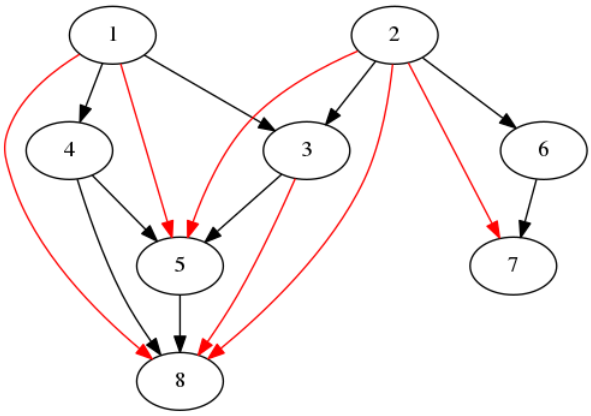
Figura 7: Grafo orientado com 8 nós, e respectivos conjuntos $A(v)$, com $v \in \{1,2,3,4,5,6,7,8\}$

Em termos práticos, o alcance de uma difusão pode ser determinado pelo fecho transitivo de um dígrafo. Seja R uma relação binária no conjunto de nós do grafo, isto é $R \subseteq V \times V$. Logo, a relação R é o próprio conjunto E. Uma relação é denominada transitiva quando $\forall x \forall y \forall z \big((x, y) \in R \wedge (y, z) \in R \rightarrow (x, z) \in R \big)$, ou seja, será transitiva se quando existir um arco do nó x direcionado para y, e existir um arco do nó y direcionado para z, sempre existir também um arco do nó x direcionado para o nó z. Se uma relação R não é transitiva, pode-se determinar quais arcos faltam em R para que a relação tenha essa propriedade calculando-se o **fecho transitivo** de R, denominado R^*. O fecho transitivo é o menor subconjunto dos arcos que não estão em R, mas se estivessem, tornariam R uma relação transitiva. Em outras palavras, $R \cup R^*$ é uma relação transitiva. Um procedimento simples (não necessariamente eficiente) para computar o fecho transitivo de uma relação R é dado no Algoritmo 1, em pseudocódigo, com instruções alto nível sobre como computar o fecho.

Algorithm 1 CALCULA-FECHO-TRANSITIVO($G = (V, E)$)

```
1: R* ← E
2: while ( R* não for uma relação transitiva ) do
3:     Inspecionar os pares ordenados de R*
4:     Adicionar novos pares em R* se necessário
5: end while
6: return R*
```

Uma vez conhecido o conceito de fecho transitivo, e apresentado um procedimento para computá-lo, podemos revisitar o exemplo apresentado na Figura 7. Suponha que o procedimento dado no Algoritmo 1 seja aplicado sobre o grafo com 8 nós. Após o término do procedimento, espera-se obter o dígrafo apresentado a seguir, na Figura 8.



$A(1) = \{4, 8, 5, 3\}$
$A(2) = \{3, 5, 8, 6, 7\}$
$A(3) = \{5, 8\}$
$A(4) = \{5, 8\}$
$A(5) = \{8\}$
$A(6) = \{7\}$
$A(7) = \varnothing$
$A(8) = \varnothing$

Figura 8: Grafo original, acrescido dos arcos do fecho transitivo, e respectivos conjuntos $A(v)$, com $v \in \{1, 2, 3, 4, 5, 6, 7, 8\}$

Observando a Figura 8, observa-se que todos os nós k atingíveis a partir de um dado nó v pode ser facilmente descobertos após calcular-se o fecho transitivo correspondente. De maneira mais formal, podemos dizer que:

Definição 10 (Conjunto $A(v)$ de Nós Atingíveis a Partir de v) *Dados uma relação R e o fecho transitivo R^* correspondente, então o conjunto de nós atingíveis a partir de $v \in V$, denotado por $A(v)$, é dado por:*

$$A(v) = \left\{ \; k \in V \mid \exists k \text{ tal que } (v,k) \in (R \cup R^*) \; \right\}$$

Desta forma, para saber quais nós são atingíveis a partir de v, calcula-se o fecho transitivo da relação R original, e faz-se a união entre o fecho e a relação original. Pela transitividade, todo nó k adiante alcançável a partir de v ou será uma extremidade de chegada de um arco original de R, ou será uma extremidade de chegada de um arco de R^*, em ambos os casos originado em v.

2.3 O Problema da Mínima Difusão Parcial em Grafos Direcionados

Seja $D \subseteq V$ um subconjunto de nós influenciadores. Denominaremos a respectiva difusão iniciada nos nós de D de **difusão total** se, partindo de cada nó $k \in D$, a propagação correspondente alcançar todos os nós do grafo. Em outras palavras, uma difusão completa é aquela em que

$$\left| \bigcup_{v_i \in D} A(v_i) \right| = \left| A(v_1) \cup A(v_2) \cup \cdots \cup A(v_i) \cup \cdots \cup A(v_n) \right| = \left| V \right|$$

Uma condição suficiente para que se atinja uma difusão total consiste em construir um subconjunto $D \subseteq V$ contendo todos os nós fonte de V. Dessa forma, qualquer nó intermediário ou sorvedouro será alcançável a partir da propagação dos nós de D. Seja agora $D \subseteq V$ um subconjunto de nós influenciadores. A difusão iniciada a partir destes nós será denominada **p-difusão parcial** se a quantidade de nós alcançáveis a partir dessa difusão for pelo menos p percento do total de nós do grafo, com $1 \leq p \leq 100$. Em outras palavras, uma p-difusão parcial é aquela em que

$$\left| \bigcup_{v_i \in D} A(v_i) \right| = \left| A(v_1) \cup A(v_2) \cup \cdots \cup A(v_i) \cup \cdots \cup A(v_n) \right| \geq \left\lceil \frac{p}{100} \cdot \left| V \right| \right\rceil$$

O problema proposto neste trabalho será apelidado de **Problema da Mínima Difusão Parcial em Grafos Direcionados**. Ele consiste em determinar o menor subconjunto $D \subseteq V$ que inicie uma p-difusão parcial, isto é, que propague a informação passada adiante e seja capaz de alcançar **no mínimo** p percento dos nós do grafo G.

Uma possível aplicação do problema está em difusão de campanhas de *marketing*. Aqui segue um exemplo hipotético, simplista, mas suficiente para que se discuta os princípios do problema. Seja no exemplo um empresa que deseja divulgar um novo produto utilizando uma rede social minimalista. Suponha que o departamento de *marketing* conclui que, para que o lançamento do produto seja considerado um sucesso, então ele deve ser conhecido e comentado por pelo menos 40% das pessoas

do universo de discurso, neste caso, da rede social minimalista. Suponha também que cada pessoa que é informada sobre o produto possa passar adiante essa informação para outras pessoas que ela consiga influenciar como possíveis compradoras. A empresa designada para fazer a publicidade recebe uma quantidade unitária de recursos financeiros para cada pessoa que ela aborda; a partir daí a informação flui naturalmente, boca a boca, da pessoa que recebeu a informação sobre o produto para outra pessoa que ela consegue influenciar. Um mapeamento direto pode ser feito deste exemplo para o Problema da Mínima Difusão Parcial em Grafos Direcionados: a rede social é representada por um grafo G, onde os nós de G representam pessoas, e os arcos de G representam outras pessoas que podem ser influenciadas pela opinião de pessoas que já conhecem o produto em divulgação. Quando uma pessoa é informada sobre o novo produto, ela repassa adiante a informação para as pessoas que ela é capaz de influenciar. Desta forma, o problema pode ser reformulado para responder à seguinte pergunta: *dados $|V|$ pessoas nesse universo de discurso, quais seriam aquelas que, com menor custo em recursos financeiros, eu deveria abordar para conseguir fazer com que a campanha se espalhe para pelo menos 40% das pessoas deste universo de discurso?* Formalmente, a resposta está na resolução do 40-Difusão Mínima.

2.4 Algoritmos Propostos

Nesta seção serão apresentados alguns procedimentos computacionais para resolver o problema, mais especificamente, um procedimento geral, que não entra em detalhes sobre como fazer a escolha dos nós influenciadores onde a difusão deve iniciar, e dois procedimentos que descrevem possibilidades de escolha. O primeiro deles utiliza uma estratégia gulosa, míope, que é baseada no total de nós alcançáveis a partir de um dado nó $v \in V$, e um segundo procedimento que utiliza uma estratégia probabilística, baseada na aleatoriedade.

2.4.1 Procedimento Geral

Com as informações dadas sobre o problema, é possível propor um procedimento computacional capaz de resolver este problema. Antes, devemos ressaltar que (i) pode não ser a única forma de resolver o problema, visto que pouca ou nenhuma investigação foi feita ainda; (ii) pode não ser a forma mais eficiente de resolvê-lo sob o ponto de vista de complexidade de algoritmos, mas não é objetivo do trabalho buscar uma forma eficiente de fazê-lo, mas sim uma forma eficaz; (iii) ele é utilizado mais como um objeto didático para aprendizagem dos conceitos sobre conjuntos e relações, do conteúdo de Matemática Discreta, do que como ferramenta efetiva de otimização em redes. Dadas as simplificações apresentadas, o procedimento pode ser feito da seguinte maneira: primeiramente, determina-se quais dos nós do grafo são nós fonte, isto é, aqueles que possuem arcos de saída, mas não possuem arcos de chegada, e guarda-os em um subconjunto $F \subseteq V$. O conjunto F guarda, portanto, os nós que se estimulados, terão potencial para iniciar uma propagação de informações adiante de forma a alcançar todos os nós do grafo. Em seguida, dado o grafo R que representa a relação original computa-se o fecho transitivo R^* de R utilizando o Algoritmo 1 ou outro algoritmo mais eficiente contido na literatura para determinar que nós do grafo são alcançáveis a partir de um dado nó $v \in V$, isto é, determinamos

para cada nó $v \in V$ qual é o conjunto $A(v)$ correspondente. O conjunto F computado acima possui apenas nós influenciadores; dele devemos sacar um subconjunto $D \subseteq F$ mínimo de influenciadores que seja capaz de difundir a informação iniciada por eles adiante a pelo menos p percento dos nós de V. Aqui, neste ponto onde D é calculado, é que o procedimento se diferencia de acordo com a estratégia usada. O pseudo-código do procedimento apresentado até o momento é dado a seguir, no Algoritmo 2.

Algorithm 2 PROCEDIMENTO-P-DIFUSAO($G = (V, E)$, p)

1: $F \leftarrow \varnothing$
2: **for all** ($v \in V$) **do**
3: **if** (v é um nó fonte) **then**
4: $F \leftarrow F \cup \{v\}$
5: **end if**
6: **end for**
7: Calcula o fecho transitivo de G
8: $D \leftarrow \varnothing$
9: **repeat**
10: Selecione um nó $f \in F$ usando uma das estratégias propostas
11: $F \leftarrow F - \{f\}$
12: $D \leftarrow D \cup \{f\}$
13: $N \leftarrow \bigcup_{v \in D} A(v)$
14: **until** ($|N| \geq (p/100) \cdot |V|$)
15: **return** D

O algoritmo funciona como segue. Na linha 1, um conjunto F de fontes é inicialmente definido como vazio. No trecho entre as linhas 2 e 6 descreve uma varredura no conjunto V de nós do grafo em busca daqueles que são nós fonte e, portanto, potencializam uma difusão. Cada nó encontrado é incluído no conjunto F; ao final do trecho F conterá todos os nós fonte do grafo G. Em seguida calcula-se, na linha 7, o fecho transitivo do grafo G considerando-o como uma relação onde os arcos do grafo indicam quem se relaciona com quem. Este procedimento é necessário para determinar que nós são alcançáveis a partir de cada nó $v \in V$ do grafo, ou seja, é usado para determinar $A(v)$. Em seguida, na linha 8, o conjunto D é criado inicialmente vazio, mas será preenchido com alguns dos nós fonte de F conforme o procedimento avança, até que estes nós de D sejam capazes de gerar um difusão que atinge pelo menos p-percento do total de nós de V. Isso é feito de maneira iterativa no trecho de código compreendido entre as linhas 9 e 13. Na linha 10, utilizando ou a estratégia gulosa, ou a estratégia aleatória, escolhe-se um nó fonte f que pertence ao conjunto de todos os nós fontes F. Na linha 11, o nó fonte f escolhido é removido de F para evitar que seja escolhido novamente em uma iteração futura, se necessária. O nó f sorteado é então adicionado ao conjunto D na linha 12. Esta atualização de D causa o efeito de aumentar o número de nós alcançáveis a partir da difusão em construção; para saber quais nós são agora atingíveis, faz-se a união de todos os conjunto $A(v)$ para cada nó $v \in D$ de forma que o conjunto final N resultante da união contenha todos os nós que são atingíveis de uma propagação que inicia-se nos

nós guardados em D. Se porventura este total de nós alcançáveis é da ordem de pelo menos p-percento do total de nós V, o procedimento deve encerrar pela condição de parada do laço descrito na linha 14; caso contrário a iteração se repete, executando a seleção de mais um nó fonte $f \in F$ na linha 10. O procedimento segue até que pelo menos p-percento do total de nós tenha recebido a informação passada via difusão partindo dos nós que pertencem à D. Este conjunto é então retornado na linha 15.

É importante ressaltar que a descrição algorítmica do procedimento em pseudo-código serve apenas com o propósito de orientar o implementador sobre o funcionamento macro do método. Detalhes de implementação poderão diferir da descrição em alto nível apresentada (como por exemplo, na escolha das estruturas de dados que serão escolhidas a partir da linguagem utilizada para armazenar as informações do solucionador), mas deve seguir a idéia principal em linhas gerais.

Observe ainda que, independente da estratégia de escolha feita na linha 10, não é possível garantir que a sequência de escolhas feitas seja sempre aquela que gera o conjunto D de menor cardinalidade para qualquer caso (isto é, para qualquer grafo de entrada). Por esse motivo, o procedimento proposto não é exato nem ótimo, mas sim o que se conhece na literatura por **método heurístico** (isto é, aquele que se baseia em alguma observação, *feeling* ou estrutura do problema para resolvê-lo, sem garantir otimalidade). Apesar dessa fragilidade, em geral as respostas computadas pelos métodos heurísticos costumam ser subótimas mas de boa qualidade, computadas por métodos eficientes em termos computacionais. Este balanço entre qualidade da solução e tempo de execução normalmente advoga a favor do uso de métodos heurísticos para resolver problemas combinatórios de grande complexidade onde ou é difícil modelá-lo matematicamente para resolvê-lo de maneira exata, porém ineficiente, ou não é conhecido sequer um procedimento exato para resolvê-lo, ou porque o método heurístico é rápido suficiente para compensar uma eventual perda de qualidade da solução.

2.4.2 Procedimento Guloso

A seleção de um nó fonte f a partir de F de maneira gulosa deve levar em conta uma escolha que é localmente ótima, sem considerar seu impacto no restante da construção da solução. Em outras palavras, ela é feita de tal maneira a considerar o que é melhor no momento da escolha, mesmo que mais adiante essa escolha coloque o procedimento construtivo em uma situação de sub-otimalidade.

Neste trabalho propõe-se que o procedimento guloso funcione de tal maneira. Seja F o conjunto de todos os nós fontes do grafo G. Em uma dada iteração do procedimento geral, a escolha feita na linha 11 do algoritmo levará em conta que o nó fonte f a ser escolhido será aquele que mais alcança outros nós no grafo. Em outras palavras, f é escolhido dentre todos os nós $v \in F$ tal que $|A(f)| > |A(v)|$. A cada iteração do laço compreendido dentre as linhas 10 e 14, o nó fonte que mais alcança nós do grafo é escolhido, removido de F, e inserido em D. Na próxima iteração, se necessária, o nó restante de F que mais alcança outros nós é também escolhido, sendo removido de F e inserido em D. O procedimento segue até que o conjunto D possua nós que, juntos, alcançam mais do que p-percento dos nós do grafo G, sempre escolhendo para compor o conjunto D os nós fontes f em **ordem decrescente de cardinalidade de** $A(f)$, ou seja, primeiro o que mais alcança outros nós, seguido do segundo que mais alcança nós, e assim segue até atingir o percentual p especificado.

Um detalhe importante ocorre quando mais de um nó fonte empata na quantidade de nós que consegue alcançar: neste caso, sorteia-se um dos nós envolvidos no empate de maneira uniforme, e usa-o. O aspecto guloso do procedimento está no fato de que uma escolha local nem sempre leva à um ótimo global. **Tente descobrir o porquê. Se não conseguir, procure o seu professor**.

Dica: pense em um grafo onde muitos nós influenciados são comuns à mais de um nó fonte, tal como apresentado na Figura 9. Se seguirmos o procedimento à risca, para $p = 60$ (isto é, o algoritmo deve parar quando selecionar uma quantidade de nós fonte que difunde a informação para 50% de 15 nós, isto é, pelo menos $\lceil 0.6 \cdot 15 \rceil = \lceil 9 \rceil = 9$ nós) existe alguma outra solução construída de outra maneira que é 'melhor do que' a construída pelo procedimento guloso? Qual?

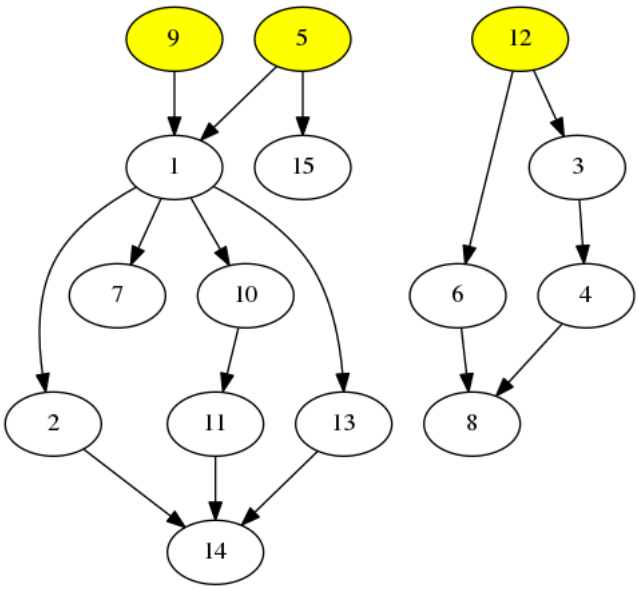


Figura 9: Dígrafo com nós fonte (em amarelo), influenciadores e influenciados

2.4.3 Procedimento Aleatório

Neste procedimento não existe uma ordem de prioridade de seleção de nós fonte, como existe no caso do procedimento guloso. De posse do conjunto F, seleciona-se um nó qualquer do conjunto de maneira equiprovável, isto, é, a escolha de qual nó será selecionado independe de quantos nós ele alcança pois a chance de qualquer nó ser selecionado é da ordem de $1/|F|$. Isso pode ser feito utilizando-se um gerador de números aleatórios uniforme, tal como disponibilizado na maioria das linguagens de programação.

2.5 *Benchmarks*

Os problemas clássicos em computação são geralmente explorados por meio de diferentes métodos computacionais, que variam em complexidade, convergência e uso

de memória. Isto posto, é comum realizar comparações entre os métodos existentes ou suas variantes. Para realizar esta avaliação de maneira objetiva é comum fazer uso de *benchmarks*. Em Ciência da Computação, e mais especificamente em otimização combinatória, os *benchmarks* consistem em conjuntos de instâncias para um dado problema combinatório onde são disponibilizados arquivos de entrada, valores ótimos (quando conhecidos) para cada entrada, ou melhor resultado documentado. Em geral descrevem o formato de leitura de suas instâncias (binária ou ASCII), assim como os dados que elas contêm e a fonte das mesmas (muitas vezes as instâncias são oriundas de artigos publicados em revistas, conferências ou *journals* da área). É possível buscar na literatura especializada diversos *benchmarks* para problemas clássicos em grafos. Para exemplificar, seguem a URL de alguns:

- `http://www.cs.uky.edu/ai/benchmark-suite/vertex-cover.html`. Composto por 04 instâncias (`vc1`, `vc2`, `vc3` e `vc4`) que contêm a descrição de grafos contendo 100 nós;
- `http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm`: Contêm 8 conjuntos de instâncias contendo 5 instâncias cada fornecidas no formato `ASCII DIMACS`. Além das instâncias descritas, apresenta outro conjunto de instâncias de grafos gerados aleatoriamente utilizado pela primeira vez na dissertação de mestrado de Muthubharathi Periannan.
- `https://turing.cs.hbg.psu.edu/txn131/file_instances/vertex_cover/clique_complement/hamming.tar.gz`. Contêm 6 arquivos de instância no formato `ASCII DIMACS` para o problema da Clique Máxima. Podem ser usados como entrada pelos algoritmos.
- `http://www.ru.is/faculty/eyjo/vertexcover.html`. Contêm arquivos de entrada e tabelas para diferentes problemas combinatórios em grafos.

A tabela apresentada na Figura 10 resume algumas informações sobre algumas destas instâncias. A coluna **Instância** refere-se ao nome do arquivo que descreve o grafo correspondente. As colunas **Nós** e **Arcos** referem-se, respectivamente, ao número de nós e arcos do grafo; a coluna **Densidade** refere-se ao percentual de arcos que o grafo descrito pela instância possui em relação ao total de arcos de um grafo completo[1].

[1] O número de arcos de um grafo completo é dado por $n(n-1)/2$

Figura 10: Listagem de instâncias para problemas de grafos com quantidade de nós, arcos e densidades dos *benchmarks* apresentados

Instância	**Nós**	**Arcos**	**Densidade**
`vc1`	100	200	04.04 %
`vc2`	100	200	04.04 %
`vc3`	100	200	04.04 %
`vc4`	100	200	04.04 %
`frb30-15-1.mis`	450	17827	17.64 %
`frb30-15-2.mis`	450	17874	17.71 %
`frb56-25-1.mis`	1400	109676	11.1 %
`frb56-25-2.mis`	1400	109401	11.1 %
`graph250-01.txt`	250	15562	49.99 %
`graph250-02.txt`	250	23343	74.99 %
`graph500-01.txt`	500	62375	50.00 %
`graph500-01.txt`	500	99880	80.06 %
`hamming6-2.clq-compliment.txt`	64	192	09.52 %
`hamming6-4.clq-compliment.txt`	64	1312	65.07 %
`hamming8-2.clq-compliment.txt`	256	1312	04.01 %
`hamming8-4.clq-compliment.txt`	256	11776	36.07 %
`hamming10-2.clq-compliment.txt`	1024	5120	00.07 %
`hamming10-4.clq-compliment.txt`	1024	89600	17.10 %

O formato dos arquivos dos *benchmarks* acima é denominado `ASCII DIMACS`, definido pelo *Center for Discrete Mathematics & Theoretical Computer Science* (DIMACS), entidade formada pela colaboração entre as universidades de Princeton e Rutgers com as empresas AT&T, Bell Labs e NEC e com sede no campus da Universidade Rutgers. O formato é simples, e consiste na descrição de grafos em arquivos texto plano ASCII onde cada linha do arquivo inicia-se com um único caracter que indica o conteúdo do restante da linha. Para exemplificar, seja o arquivo `frb56-25-1.mis`, que representa um grafo com 1400 nós e 109.676 arcos. A primeira linha indica os dados do grafo, e as linhas subsequentes indicam, uma a uma, os arcos do grafo. Observe que a linha que descreve as características do grafo inicia-se com um caracter '`p`', enquanto que as linhas que descrevem as ligações entre os nós do grafo iniciam-se pelo caracter '`e`'. Descritos desta forma é fácil carregar os nós e arcos de um grafo em memória, mais especificamente em uma matriz de adjacência, usando qualquer linguagem de programação que dê suporte à manipulação de arquivos.

```
p edge 1400 109676
e 1 2
e 1 3
```

```
e 1 4
e 1 5
e 1 6
e 1 7
e 1 8
e 1 9
e 1 10
e 1 11
e 1 12

.
.
.

e 1395 1398
e 1395 1399
e 1395 1400
e 1396 1397
e 1396 1398
e 1396 1399
e 1396 1400
e 1397 1398
e 1397 1399
e 1397 1400
e 1398 1399
e 1398 1400
e 1399 1400
```

Benchmarks possuem um papel fundamental na análise de novos algoritmos. Seja o exemplo de problema apresentado neste documento. Será que é possível determinar, **sem experimentação**, qual dos dois procedimentos, guloso ou aleatório, computa melhores resultados? A resposta inicial é não, porque em ambos os casos existem elementos probabilísticos envolvidos, que não nos permite analisar a complexidade dos métodos. Nenhum dos métodos é determinístico e pode dar respostas diferentes mesmo quando utilizado sobre o mesmo grafo de entrada.

Como determinar qual dos dois algoritmos é melhor se não é possível saber o resultado de cada execução independente? Uma abordagem muito comum consiste em escolher um *benchmark* para o problema, e realizar uma análise experimental usando os algoritmos em questão sobre suas instâncias. Cada algoritmo é executado uma certa quantidade de vezes para cada instância (por exemplo, 100 vezes em cada grafo), os resultados são anotados e ao término da execução de cada algoritmo realiza-se uma análise para verificar qual dos algoritmos melhor se comportou diante das diferentes instâncias usadas. Embora não se possa generalizar o resultado, esta abordagem permite analisar o comportamento dos algoritmos testados sobre diferentes condições. Especificamente para o problema proposto, pode-se analisar o resultado dos métodos, por exemplo, em entradas com diferentes densidades; isto é, com diferentes percentuais de arcos em relação à um grafo completo; com diferentes topologias, isto é, como a forma de ligar os nós do grafo afetam o resultado; e com diferentes tamanhos, isto é, como os métodos se comportam quando o número de nós dos grafos de entrada descritos em cada instância aumentam?

Para cada execução independente de cada instância é comum coletar pelo menos duas variáveis: qualidade da solução e tempo de execução (*runtime*). A qualidade da solução refere-se a alguma medida que mostra o quão distante as soluções computadas pelo método ficaram em relação ao ótimo. No caso do problema proposto pode-se considerar como medida de qualidade a cardinalidade do conjunto D calculado: quanto menor $|D|$, melhor será a solução. O tempo de execução refere-se à quanto tempo o método demora para executar em um sistema operacional específico e máquina específica. Para conseguir uma medida sem interferências externas os métodos devem executar todas as instâncias sob a mesma condição. Após a coleta faz-se a análise dos resultados, geralmente utilizando estatística.

2.6 Competências Desenvolvidas com o Trabalho Prático

As subseções apresentadas acima contextualizaram o uso de grafos para modelagem de problemas do mundo real em computação. Mais especificamente elas apresentaram (i) o que são grafos, (ii) como podem ser representados computacionalmente, (iii) o problema de otimização combinatória apresentado e (iv) um procedimento geral para computar soluções pro problema proposto, e dois procedimentos auxiliares que usando diferentes estratégias em conjunto com o procedimento geral. Ao término deste trabalho o grupo terá:

- Compreendido e implementado grafos como relações $R \subseteq A \times A$, usando representação por matriz de adjacência;
- Compreendido o problema combinatório denominado Problema da Mínima Difusão Parcial em Grafos Direcionados (PMDPGD) que ocorre em grafos onde se deseja encontrar o menor conjunto de nós fonte que consegue difundir informação para pelo menos p-percento de nós do grafo;
- Compreendido e implementado um algoritmo guloso para o PMDPGD;
- Compreendido e implementado um algoritmo aleatório para o PMDPGD;
- Compreendido o conceito de *benchmark* para problemas combinatórios;
- Exercitado a leitura e escrita de arquivos em linguagem computacional utilizando formatos de arquivos texto plano de entrada e saída no formato ASCII DIMACS;
- Compreendido o conceito de entrada de dados via linha de comando em console;
- Executado um pequeno experimento com ambos os algoritmos sobre as instâncias do *benchmark* indicado, seguido de análise de resultados.

A próxima seção especifica os requisitos do trabalho que, se completamente implementados, completarão a entrega do presente trabalho prático.

3 Especificação

Esta seção apresenta uma especificação técnica de elementos que devem ser implementados pelo grupo para atender à especificação de um *software* capaz de resolver um problema de otimização combinatória proposto e apelidado de Problema da Mínima Difusão Parcial em Grafos Direcionados (PMDPGD). Para resolver o PMDPGD o *software* implementará dois algoritmos, um guloso e outro aleatório. A aplicação deverá ser escrita em linguagem `Java`, `C` ou `Python 3.x`, usando os compiladores `javac` para linguagem `Java`, `gcc` para linguagem `C` ou `python3` para a linguagem `Python`. A aplicação será do tipo console onde toda a interação com o usuário será realizada mediante uso de linha de comando, seguindo a sintaxe para código compilado em linguagem C:

```
user@machine$ ./pdif-solver <semente> <p-difusao> <metodo> <entrada> <saida>
```

ou seguindo esta sintaxe, para código compilado em Java

```
user@machine$ java pdif-solver <semente> <p-difusao> <metodo> <entrada> <saida>
```

ou, ainda, seguindo essa sintaxe, para código interpretado em Python 3

```
user@machine$ python3 pdif-solver.py <sem.> <p-dif> <metodo> <entrada> <saida>
```

Independente da linguagem escolhida, um conjunto de parâmetros deve ser informado por linha de comando, onde:

- `<semente>`: valor inteiro de semente informada pelo usuário que será usada para inicializar o gerador de números aleatórios;
- `<p-difusao>`: valor correspondente à p-difusão alvo, ou seja, qual é o valor percentual p de nós que devem ser atingidos na rede para considerar a difusão bem sucedida;
- `<metodo>`: caracter que indica qual método utilizar: '`g`' para algoritmo guloso, '`a`' para método aleatório;
- `<entrada>`: nome do arquivo de entrada contendo a descrição de um grafo no formato `ASCII DIMACS`, já mencionado neste documento;
- `<saida>`: nome do arquivo de saída gerado automaticamente pelo aplicativo e que contêm o log dos calculos realizados (ver requisito de Saída de Dados).

Para exemplificar, seja o caso onde deseja-se executar o aplicativo para resolver o PMDPGD sobre a instância `graph250-01.txt` através do algoritmo guloso gerando como saída o arquivo `solucao.log`. Assuma que a semente do gerador de números aleatórios será iniciada com o número 12345, com valor de $p = 30\%$. A chamada completa seria algo do tipo, se executando a aplicação compilada em C:

```
user@machine$ ./pdif-solver  12345  30 g  graph250-01.txt  solucao.log
```

O código fonte da aplicação será compilado no ambiente operacional Linux, distribuição Ubuntu. Certifique-se que o seu código fonte é compatível com tal ambiente. **Códigos que não compilarem no ambiente mencionado serão desconsiderados, sem excessão**. Toda a implementação deverá ser entregue em um único arquivo de nome `pdif-solver.c` se implementado em `C`, ou `pdif-solver.py` se implementado em `Python3`. Em `Java` pode-se gerar mais arquivos, porém devem ser acompanhados por *shell script* ou instruções sobre sua compilação para gerar o arquivo binário `pdif-solver` correspondente. **O código fonte deverá possuir um cabeçalho em comentários contendo os nomes e matrículas dos integrantes do grupo.** O trabalho deverá ser enviado ao professor em um único arquivo compactado denominado `pratico-dm-2016-XX-YY-ZZ.zip` onde `XX`, `YY` e `ZZ` correspondem aos nomes dos integrantes do grupo. Deverá conter o código fonte da aplicação e um arquivo em formato .pdf contendo os resultados encontrados com a execução da aplicação `vc-solver` sobre os arquivos de *benchmark* descritos neste documento.

As próximas sub-seções detalham cada requisito da aplicação. **De maneira geral, cada requisito deve ser implementado em um procedimento ou função separada**, podendo-se modularizar ainda mais cada requisito em pedaços menores desde que documentados e que facilitem a leitura do código fonte.

3.1 Requisito: Entrada via Arquivo formato `ASCII DIMACS`

Este requisito trata da entrada de dados da aplicação `pdif-solver`. Conforme mencionado, a entrada de dados será feita mediante arquivo ASCII cujo formato será conhecido a priori. Arquivos ASCII são facilmente criados a partir de editores de texto sem formatação, como Bloco de Notas e `Notepad++` da plataforma Windows, e `gedit` da plataforma Linux.

O formato do arquivo de entrada é texto plano, e possui um caracter de controle no início de cada linha para informar o que seguirá. A estrutura do arquivo poderá conter comentários (iniciados pelo caracter '`c`'), informações sobre a dimensão do grafo e número de ligações (caracter '`p`', seguido da string '`edge`') e informações sobre qual arco liga a qual arco (caracter '`e`').

```
c <linha de comentário, ignorada até o Carriage Return (caracter CR)>
p edge <número de nós> <número de arcos>
e <início do arco> <término do arco>
```

Observe que o nome do arquivo é informado como argumento de linha de comando, e não mediante digitação pelo usuário. A leitura de argumentos por linha de comando deverá ser pesquisada pelo grupo, mas sugere-se buscar pelas funções relacionadas com leitura de argumentos em linha de comando para as linguagens `C`, `Java` e `Python3`.

O arquivo informado deverá ser lido e suas informações devem ser armazenadas em estrutura de dados em memória (ver Requisito 3.2). Observe que a dimensão do grafo só será conhecida em tempo de execução; logo a memória necessária para armazenar o grafo deverá ser gerenciada por métodos que redimensionam dinamicamente matrizes e vetores, ou por métodos explícitos de alocação dinâmica.

O grupo deverá pensar em situações problemas que devem ser tratadas na entrada de dados. São exemplos de tais situações: informar um método inexistente, informar um arquivo de entrada inválido, não informar o número de argumentos exigido, etc. Caso alguma delas ocorra a aplicação deverá ser abortada, exibindo uma mensagem de erro para o usuário no terminal. A entrada de dados deverá ser implementada em uma função auxiliar que percorrerá o arquivo lendo seu conteúdo.

3.2 Requisito: Estrutura de Dados Grafo

O grafo que representa o grafo do problema necessita, após sua leitura, de ser representado computacionalmente. A maneira mais simples de representar grafos não direcionados é por meio da chamada matriz de adjacência. Nela, quando um determinado nó u do grafo relaciona-se com outro nó v (isto é, existe arco que liga os nós u e v) então a matriz assume valor 1 nas posições $M[u, v]$ e $M[v, u]$, e 0 nestas posições caso contrário. Essa representação é simples e permite facilmente verificar o grau de um nó i simplesmente somando-se os elementos da i-ésima linha na matriz de adjacência, ou dai i ésima coluna correspondente (dependendo se é δ_{in} ou δ_{out}).

3.3 Requisito: Construção do conjunto F

Neste requisito o grupo deverá implementar uma função descobre, no grafo, quais nós são nós fonte. Isso pode ser feito fazendo-se uma varredura nos graus de entrada e saída de cada nó, e checando se satisfaz as condições descritas na parte teórica deste trabalho. Se satisfizer, então tal nó deve entrar no conjunto F. Ao final da varredura o conjunto F deverá ter sido criado para futura manipulação.

3.4 Requisito: Fecho Transitivo

Neste requisito o grupo deverá implementar algum algoritmo para cálculo de fecho transitivo no grafo G que representa a relação. O resultado do fecho deverá ser agregado ao resultado do grafo original para permitir que se determine a cardinalidade do conjunto $A(v)$ com facilidade, apenas contando o grau de saída de v. Terá utilidade nas estratégias de escolha de nó fonte.

3.5 Requisito: Algoritmo Aleatório

Neste requisito o grupo deverá implementar o algoritmo aleatório para o PMDPGD, conforme descrito na parte teórica deste documento.

3.6 Requisito: Algoritmo Guloso

Neste requisito o grupo deverá implementar o algoritmo guloso para o PMDPGD, conforme descrito na parte teórica deste documento.

3.7 Requisito: Saída em Formato DOT

Neste requisito o grupo deverá implementar uma saída simples em arquivo ASCII contendo uma descrição de dígrafo no formato DOT. O formato DOT corresponde uma notação para descrição de grafos que pode ser interpretado pela ferramenta GraphViz, que gera imagens em diversos formatos com o grafo descrito pelo DOT. Para maiores detalhes, consultar o artigo *Drawing graphs with DOT*[2]. Uma questão importante aqui é que o arquivo deverá ser salvo em arquivo texto plano (ASCII) com o nome indicado pelo parâmetro `saida`, concatenado com a string `.dot`, informado em linha de comando. Ele deverá conter na descrição:

- Todos os nós e arcos do grafo original;
- Os nós que foram selecionados para compor o conjunto D deverão estar destacados em amarelo;
- Os arcos e o círculo dos nós que são atingíveis a partir dos nós $v \in D$ deverão estar destacados em vermelho;
- Os demais arcos, não alcançáveis, deverão estar em cor preta.

Para familiarizar, seja o seguinte exemplo de grafo descrito em linguagem DOT, salvo com o nome hipotético de `solucao.dot`.

```
digraph
{
    1 [color=red];
    2 [color=red];
    3 [color=black];
    4 [color=black];
    5 [fillcolor=yellow, style=filled];
    6 [color=black];
    7 [color=red];
    8 [color=black];
    9 [fillcolor=yellow, style=filled];
    10 [color=red];
    11 [color=red];
    12 [color=black];
    13 [color=red];
    14 [color=red];
    15 [color=red];
    1 -> 2 [color=red];
    1 -> 7 [color=red];
    5 -> 1 [color=red];
    9 -> 1 [color=red];
```

[2]Disponível em `https://www.graphviz.org/pdf/dotguide.pdf`

```
    12 -> 6;
    12 -> 3;
    3 -> 4;
    1 -> 10 [color=red];
    4 -> 8;
    6 -> 8;
    10 -> 11 [color=red];
    1 -> 13 [color=red];
    13 -> 14 [color=red];
    11 -> 14 [color=red];
    2 -> 14 [color=red];
    5 -> 15 [color=red];
}
```

Uma vez salvo, pode-se invocar a ferramenta `dot` do GraphViz para gerar uma imagem em formato PNG, pela seguinte linha de comando:

```
user@machine$ dot -Tpng solucao.dot -o solucao.png
```

que resultará, por sua vez, no grafo apresentado na Figura 11, já construída em formato PNG pelo aplicativo.

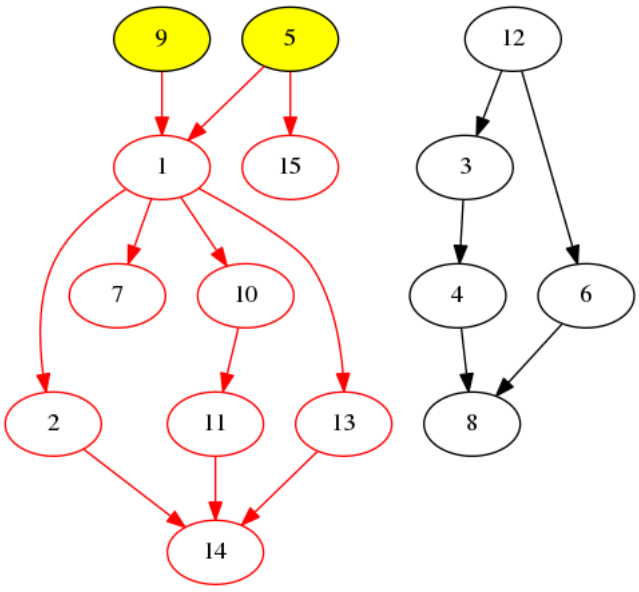


Figura 11: Exemplo de saída de arquivo DOT gerado pela aplicação `dot` do GraphViz

No Linux Ubuntu o GraphViz pode ser instalado usando a seguinte linha de comando:

```
user@machine$ sudo apt-get install graphviz
```

Após instalado, o pacote GraphViz disponibiliza vários aplicativos para conversão de grafos em figuras, como é o caso do `dot`, `neato`, `circo`, `twopi`, `sfdp`, e `fdp`. Para maiores informações, consultar `http://edutechwiki.unige.ch/en/Graphviz`.

3.8 Requisito: Saída em Arquivo de Log

Neste requisito o grupo deverá implementar saída em arquivo de log contendo informações sobre a instância executada, seu tamanho e número de arcos, a semente da execução corrente, e a cardinalidade do conjunto D. Tais informações serão gravadas no parâmetro `<saida>`, concatenado com a string `.log`, definidos para a aplicação `pdif-solver`.

Caso o arquivo de saída informado não exista deve-se criá-lo, inserindo como cabeçalho os dados conforme o modelo abaixo. As colunas são separadas pelo caracter de tabulação, encerrando a linha com um caracter CR.

```
INSTANCE    NODES    ARCS    SEED    METHOD    DSIZE
```

Caso o arquivo já exista, o grupo deverá implementar algoritmo que abre-o para inserir as novas informações no final do arquivo pré-existente. Tal concatenação evitará perder os dados de execuções anteriores. Para exemplificar, seja o exemplo onde ambos os algoritmo foi executado por 3 vezes cada sobre o arquivo `vc1.txt`.

```
INSTANCE    NODES    ARCS    SEED    METHOD    DSIZE
vc1.txt     100      200     12833   g         7
vc1.txt     100      200     12833   g         5
vc1.txt     100      200     12833   g         5
vc1.txt     100      200     12833   a         7
vc1.txt     100      200     12833   a         6
vc1.txt     100      200     12833   a         9
```

3.9 Requisito: Execução do `pdif-solver` em *benchmark*

Neste requisito o grupo deverá, de posse da implementação da aplicação `pdif-solver`, executar ambos os métodos (guloso e aleatório) sobre os arquivos de *benchmark* comentados neste documento e adaptados para o formato ASCII DIMACS. Tais arquivos encontram-se disponíveis na seguinte URL: `https://sites.google.com/a/ifmg.edu.br/diegosilva/disciplinas/2016/matematica-discreta`, com nome de `benchmark-pratico01.zip`.

Por tratar-se de métodos probabilísticos é possível que cada execução independente da aplicação retorne um resultado diferente em termos da cardinalidade do conjunto D. Com apenas uma execução independente não é possível esboçar qualquer conclusão sobre o comportamento do método na instância em questão. Neste caso é preciso executar o mesmo algoritmo sobre a mesma instância uma certa quantidade de vezes ou repetições, e realizar alguma análise de resultados sobre o conjunto de valores observados.

Importante: em cada execução independente do algoritmo é preciso definir um valor diferente de semente para o gerador de números aleatórios. Caso contrário, cada execução retornará a mesma sequência de números aleatórios que a anterior.

Uma forma de fazer essa execução de maneira automática utiliza os chamados *shell scripts* em Linux e Unix. Para exemplificar, seja um código fonte escrito para o `bash` (interpretador de linha de comando padrão do Ubuntu) onde executa-se a aplicação 100 vezes usando o método guloso (caracter '`g`'), cada qual com um valor diferente de semente (dado pelo valor que `$RANDOM` assume na execução corrente) para o arquivo de instância `graph250-01.txt`, e guardando o resultado computado e grafo no formato DOT no arquivo com preâmbulo `resultado`. Para funcionar, este código deve ser salvo em um arquivo com nome `run.sh`, e deve-se executar o comando que altera as permissões do arquivo `run.sh` para torná-lo executável usando o seguinte comando no terminal, estando na mesma pasta de `run.sh`: `chmod +x run.sh`.

```
# Quantidade de repeticoes desejada
NUM_REPETICOES=100
INSTANCIA=graph250-01.txt

# Reseta o contador inicial
CONTADOR=0

# Executa a aplicacao a quantidade de repeticoes especificada
until [ $CONTADOR -ge $NUM_REPETICOES ];
do
    echo "Execucao No. $CONTADOR"
    echo
    ./pdif-solver $RANDOM g $INSTANCIA resultado

    # Incrementa o contador
    let CONTADOR=CONTADOR+1
done
```

3.10 Requisito: Análise de Resultados

Após executar cada instâncias uma quantidade significativa de vezes (100 vezes ou mais) de maneira independente será possível realizar alguma análise sobre os resultados obtidos utilizando-se o arquivo de log gerado no Requisito 3.8. Para realizar a análise sugere-se utilizar algum aplicativo de gerenciamento de planilhas tais como o Excel ou LibreOffice. É preciso separar os dados por instância e método, agrupando as execuções que ocorreram sobre a mesma condição. Exemplo de análises que são feitas em dados coletados está a Análise Exploratória; ela visa conhecer parte dos resultados obtidos por meio dos cálculos de sua média, mediana, quartis, mínimo, máximo, desvio e variância. O grupo deverá consultar algum material de estatística para compreender o conceito e realizar a análise esperada.

3.11 Requisito: Documentação de Código

A documentação de código é importante em qualquer implementação computacional e será considerada neste trabalho. Pede-se que o arquivo que contem o código fonte possua ao menos (i) cabeçalho inicial, contendo nome do aplicativo, membros do grupo com nome e matrícula, instruções de compilação, ambiente de desenvolvimento, data e objetivo do arquivo; (ii) cabeçalho das funções auxiliares e procedimentos implementados no trabalho; (iii) comentários nos principais trechos de código de cada algoritmo, explicando resumidamente o que está sendo codificado a seguir.

3.12 Requisito: Corretude dos Resultados

O requisito mais importante do trabalho é aquele que lida com resultados corretos. Desta forma, é um requisito essencial do trabalho que além da aplicação ser implementada respeitando-se os requisitos anteriores é importante que ela retorne resultados de qualidade. Em virtude disso o grupo deverá testar exaustivamente a aplicação.

4 Barema de Correção

Conforme mencionado, o trabalho prático tem valor de 30 pontos. A correção seguirá o barema apresentado a seguir, que lista os requisitos do trabalho prático e suas respectivas pontuações.

Requisito	**Pontos**
Entrada via Arquivo formato ASCII DIMACS	2
Estrutura de Dados Grafo	2
Construção do Conjunto F	1
Algoritmo Aleatório	5
Algoritmo Guloso	5
Saída em Formato DOT	3
Saída em Arquivo de Log	1
Execução do `pdif-solver` em *benchmark*	2
Análise de Resultados	3
Documentação de Código	1
Corretude dos Resultados	5
TOTAL	**30.0 ptos**

Caso haja necessidade ou suspeita de plágio é reservado o direito à arguição por parte do professor para o grupo ou cada um de seus membros em separado. Trabalhos plagiados, com suspeita de cópia, ou que não compilam/executam no ambiente indicado serão desconsiderados e valerão zero.