Estrutura de Dados II - 2020/2

Trabalho Prático T1

29 de junho de 2021

Leia atentamente **todo** esse documento de especificação. Certifique-se de que você entendeu tudo que está escrito aqui. Havendo dúvidas ou problemas, fale com o professor o quanto antes. As dúvidas podem ser sanadas usando o fórum de dúvidas do AVA.

1 Objetivo

O objetivo deste trabalho é implementar um algoritmo de agrupamento, chamado de agrupamento de espaçamento máximo, utilizando uma MST (*Minimal Spanning Tree*).

2 O problema de Agrupamento

Agrupamento[1] (ou *clustering*) é uma tarefa de aprendizado não-supervisionado que tem por objetivo encontrar grupos naturais em bases de dados. Mais especificamente, o objetivo é encontrar grupos de objetos (e.g., pessoas, carros, livros, filmes, músicas...) tais que:

1. objetos do mesmo grupo sejam "similares";
2. objetos de grupos diferentes não sejam "similares".

O problema de agrupamento é extremamente subjetivo, uma vez que o conceito de *similaridade* pode variar significativamente dependendo do domínio sendo estudado. Suponha, por exemplo, que nossos objetos sejam pontos do espaço real bidimensional. A Figura 1 apresenta possíveis estruturas de grupos naturais. Em vários casos, há uma estrutura clara de grupos, mas cada uma tem características diferentes das demais.
Devido a essa subjetividade, há na literatura uma grande variedade de algoritmos de agrupamento. Cada algoritmo tenta encontrar grupos satisfazendo um certo conjunto de características. Infelizmente, não há um algoritmo capaz de se adaptar a qualquer tipo de situação. Dessa forma, o problema de agrupamento ainda é alvo de muitos pesquisadores em diferentes áreas do conhecimento.
Neste trabalho, não teremos um domínio específico de interesse. Vamos assumir que os objetos de interesse são pontos do $\mathbb{R}^m$ e que a dissimilaridade entre dois pontos seja medida pela distância euclidiana entre eles. Também não tentaremos resolver o problema de agrupamento no caso geral. Vamos focar em uma variante específica e bem definida do problema, a qual será descrita a seguir.

2.1 Agrupamento de Espaçamento Máximo

Um versão popular do problema de agrupamento é o *Agrupamento de Espaçamento Máximo*. Esse problema é caracterizado da seguinte forma:
Entrada: um conjunto de pontos do espaço m-dimensional $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ e um inteiro k ($1 \leq k \leq n$).
Objetivo: Encontrar k subconjuntos de $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, denotados por $C_1, \ldots, C_k$, tais que:

1. $C_i \neq \emptyset$, $i = 1, \ldots, k$;
2. $C_i \cap C_j = \emptyset$, para $i \neq j$;

[1] https://en.wikipedia.org/wiki/Cluster_analysis
[2] https://scikit-learn.org/stable/modules/clustering.html

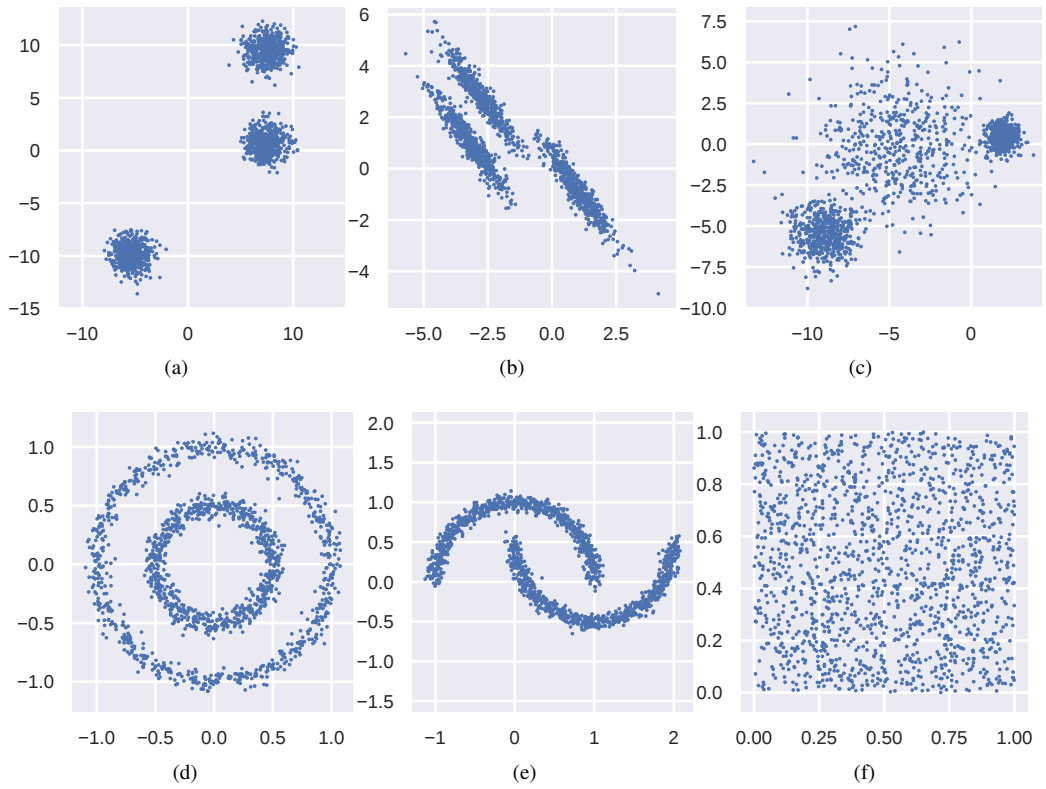

Figura 1: Exemplos de vários tipos de grupos naturais: (a) grupos circulares; (b) grupos alongados e de mesmo tamanho; (c) grupos com densidades diferentes; (d) grupos circulares e concêntricos; (e) grupos não convexos; (f) sem estrutura de grupos. Exemplos baseados na documentação da biblioteca *Scikit-learn*.[2]

3. $C_1 \cup \cdots \cup C_k = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$; e
4. o mínimo das distâncias entre dois pontos pertencentes a grupos diferentes deve assumir o maior valor possível.

Saída: os grupos $C_1, \ldots, C_k$.
Lembrando que a distância euclidiana entre dois vetores $\mathbf{x} \in \mathbb{R}^m$ e $\mathbf{y} \in \mathbb{R}^m$ é dada por

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{m} (\mathbf{x}(i) - \mathbf{y}(i))^2}, \tag{1}$$

onde $\mathbf{x}(i)$ denota a i-ésima componente do vetor $\mathbf{x}$.

2.2 Algoritmo para encontrar o agrupamento de espaçamento máximo

Apesar de parecer complexo e abstrato, o problema de agrupamento de espaçamento máximo é um dos problemas de agrupamento mais simples de se resolver (do ponto de vista computacional). O algoritmo que você deve implementar é descrito abaixo.
Dados $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ e k:

1. Obtenha as distâncias entre todos os pares de pontos. Você pode, por exemplo, armazenar esses valores em uma matriz $\mathbf{Y}$, onde $\mathbf{Y}(i, j) = d(\mathbf{x}_i, \mathbf{x}_j)$.
2. Podemos pensar que a matriz $\mathbf{Y}$ descreve um grafo G completo, não-direcionado e ponderado cujo conjunto de vértices é $V = \{1, \ldots, n\}$, onde o peso de uma aresta $\{i, j\}$ é $\mathbf{Y}(i, j)$. Assim, o segundo passo do algoritmo é computar a árvore geradora mínima de G usando o famoso *Algoritmo de Kruskal*

(veja `https://en.wikipedia.org/wiki/Kruskal%27s_algorithm`). Uma árvore geradora mínima (*minimum spanning tree - MST*) é um subconjunto de arestas de um grafo não-direcionado conexo e ponderado (isto é, com peso nas arestas). Por ser uma árvore, uma MST não pode ter ciclos. Por ser mínima, a soma dos pesos das arestas tem de ser a menor possível.

3. Remova as $k-1$ arestas de maior peso da árvore geradora mínima. As k componentes conexas restantes formam o resultado esperado.

A prova de que este algoritmo resolve o problema do agrupamento de espaçamento máximo não está no escopo deste curso. Para as pessoas interessadas, a prova pode ser encontrada no livro *Algorithm Design* dos autores *Jon Kleinberg* e *Éva Tardos*.

2.3 Exemplo ilustrativo

Para ilustrar o funcionamento do algoritmo descrito, vamos analisar um exemplo. Nesse caso, queremos encontrar o agrupamento de espaçamento máximo com $k = 3$ para o conjunto de 10 pontos (de A a J) pertencentes ao $\mathbb{R}^2$ que são apresentados na Figura 2.

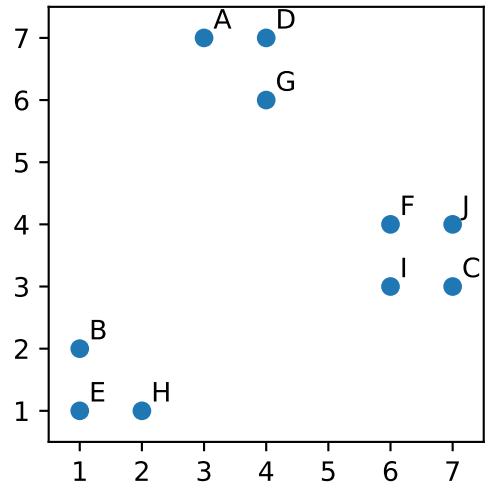

Figura 2: Coordenadas no $\mathbb{R}^2$ dos pontos denotados por $A, B, \ldots, J$.

De acordo com o Passo (1) do algoritmo, precisamos calcular a distância entre cada par de pontos. Essas distâncias são apresentadas na matriz abaixo (os valores foram arredondados para duas casas decimais para facilitar a apresentação).

$$\mathbf{Y} = \left(\begin{array}{c|cccccccccc} & A & B & C & D & E & F & G & H & I & J \\ \hline A & & & & & & & & & & \\ B & 5.39 & & & & & & & & & \\ C & 5.66 & 6.08 & & & & & & & & \\ D & 1.00 & 5.83 & 5.00 & & & & & & & \\ E & 6.32 & 1.00 & 6.32 & 6.71 & & & & & & \\ F & 4.24 & 5.39 & 1.41 & 3.61 & 5.83 & & & & & \\ G & 1.41 & 5.00 & 4.24 & 1.00 & 5.83 & 2.83 & & & & \\ H & 6.08 & 1.41 & 5.39 & 6.32 & 1.00 & 5.00 & 5.39 & & & \\ I & 5.00 & 5.10 & 1.00 & 4.47 & 5.39 & 1.00 & 3.61 & 4.47 & & \\ J & 5.00 & 6.32 & 1.00 & 4.24 & 6.71 & 1.00 & 3.61 & 5.83 & 1.41 & \end{array}\right) \quad (2)$$

Seguindo o Passo (2) do algoritmo, a Figura 3 apresenta a árvore geradora mínima resultante.

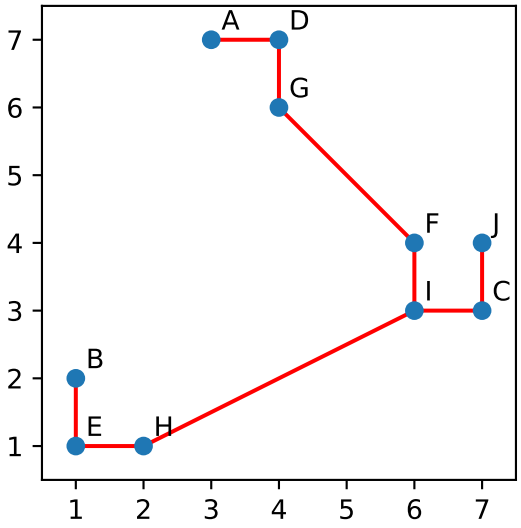

Figura 3: Árvore Geradora Mínima do grafo de exemplo. As arestas da árvore são representadas pelas linhas em vermelho.

Por fim, após seguir o Passo (3) e remover as $k - 1 = 2$ arestas de maior peso da árvore geradora mínima, tem-se os 3 grupos apresentados na Figure 4.

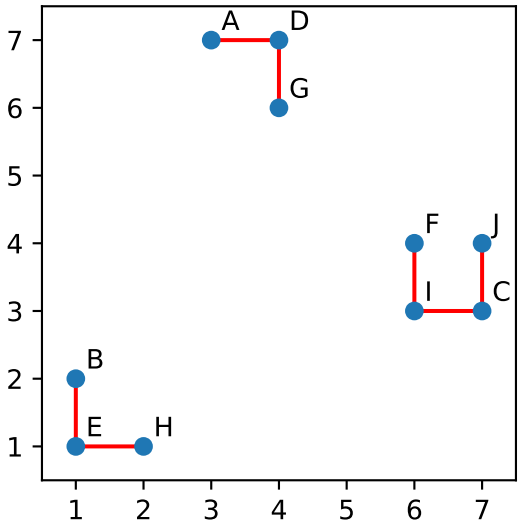

Figura 4: Grupos resultantes. Cada grupo é uma componente conexa.

3 Entrada e Saída

3.1 Entrada

Os dados a serem agrupados deverão ser lidos de arquivos texto organizados da seguinte forma:

1. Cada linha do arquivo representará um ponto do $\mathbb{R}^m$ e será composta por $m + 1$ campos separados por vírgulas. O primeiro elemento será um identificador único do ponto (cadeia de caracteres) e os demais m valores reais indicarão a coordenada do ponto no $\mathbb{R}^m$.

2. o comprimento dos identificadores poderá ser arbitrariamente longo.

A seguir, o conteúdo do arquivo relativo ao exemplo da seção anterior:

```
A,3.0,7.0
B,1.0,2.0
C,7.0,3.0
D,4.0,7.0
E,1.0,1.0
F,6.0,4.0
G,4.0,6.0
H,2.0,1.0
I,6.0,3.0
J,7.0,4.0
```

3.2 Saída

A saída do trabalho deverá ser salva em um arquivo, também de texto, organizado em linhas. Cada linha do arquivo será relativa a um grupo resultante e seus elementos deverão estar separados por vírgulas. Os seguintes requisitos devem ser satisfeitos:

1. Os elementos de cada grupo deverão estar ordenados em ordem lexicográfica;
2. Os grupos estarão ordenados, lexicograficamente, de acordo com o primeiro de seus respectivos elementos.

Seguindo esse padrão, a saída do exemplo da seção anterior é:

```
A,D,G
B,E,H
C,F,I,J
```

3.3 Execução do trabalho

Para testar seu trabalho, o professor executará comandos seguindo o seguinte padrão.

```
tar -xzvf <nome_arquivo>.tar.gz
make
./trab1 <nome_arquivo_entrada> k <nome_arquivo_saida>
```

É extremamente importante que vocês sigam esse padrão. Seu programa não deve solicitar a entrada de nenhum valor e também não deve imprimir nada na tela.
Por exemplo, se o nome do arquivo recebido for `2004209608.tar.gz`, os dados de entrada estiverem em `entrada.txt`, o número de grupos desejados for 3 e o nome do arquivo de saída for `saida.txt`, o professor executará:

```
tar -xzvf 2004209608.tar.gz
make
./trab1 entrada.txt 3 saida.txt
```

4 Detalhes de implementação

A seguir, alguns detalhes, comentários e dicas sobre a implementação. Muita atenção aos usuários do Sistema Operacional Windows.

- o trabalho deve ser implementado em C. A versão do C a ser considerada é a presente no Sistema Operacional Ubuntu 18.04 (ou Ubuntu 20.04).
- o caractere de nova linha será o `\n`.

- Seu programa deve ser, obrigatoriamente, compilado com o utilitário `make`. Crie um arquivo `Makefile` que gera como executável para o seu programa um arquivo de nome `trab1`.
- É possível que mais de uma árvore geradora mínima exista para um grafo. O professor fará o possível para que isso não ocorra nos arquivos de teste. Fiquem à vontade para quebrar empates de forma arbitrária.
- A linguagem C possui algumas funções que podem ser úteis na leitura dos arquivos. Em especial, sugere-se o estudo cuidadoso das funções `getline` e `strtok`.
- A estrutura de dados *disjoint-set* mencionada no pseudo-código da Wikipedia é a estrutura de *union-find* vista em sala. O código dessa estrutura está disponível no AVA. Adapte a versão mais eficiente do algoritmo para as suas necessidades e use-a no seu trabalho.
- O algoritmo de Kruskal requer que as distâncias entre pares de pontos sejam ordenadas. Recomenda-se o uso da função `qsort` do C para tal tarefa.
- Ao longo do desenvolvimento do trabalho, certifique-se que o seu código não está vazando memória testando-o com o `valgrind`. Não espere terminar o código para usar o `valgrind`, incorpore-o no seu ciclo de desenvolvimento. Ele é uma ferramenta excelente para se detectar erros sutis de acesso à memória que são muito comuns em C. Idealmente o seu programa deve sempre executar sem nenhum erro no `valgrind`.
- Não é necessária nenhuma estrutura de dados muito elaborada para o desenvolvimento deste trabalho. Todas as estruturas que você vai precisar foram discutidas em aula ou no laboratório. Veja os códigos disponibilizados pelo professor para ter ideias. Prefira estruturas simples a coisas muito complexas. Pense bem sobre as suas estruturas e algoritmos antes de implementá-los: quanto mais tempo projetando adequadamente, menos tempo depurando o código depois.

5 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até as 23:59h do dia 30/07/2021. Não serão aceitos trabalhos após essa data.
- **Prazo para tirar dúvidas**: Para evitar atropelos de última hora, você deverá tirar todas as suas dúvidas sobre o trabalho até o dia 28/07/2021. A resposta para dúvidas que surgirem após essa data fica a critério do professor.
- **Grupo**: O trabalho pode ser feito em grupos de até três pessoas. Atenção ao fato de que duas pessoas que estiverem no mesmo grupo nesse trabalho não poderão estar no mesmo grupo nos próximos trabalhos.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado, no formato `.tar.gz`, com todo o seu trabalho. A sua submissão deve incluir todos os arquivos de código e um `Makefile`, como especificado anteriormente. Além disso, inclua um arquivo de relatório (descrito adiante). **Somente uma pessoa do grupo deve enviar o trabalho no AVA. Coloque a matrícula de todos integrantes do grupo (separadas por vírgula) no nome do arquivo do trabalho.**
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

6 Relatório de resultados

Após terminar de implementar e testar o seu programa você deverá escrever um relatório sobre o trabalho. As seguintes seções são obrigatórias:

1. **Introdução:** breve descrição do conteúdo do documento e dos resultados obtidos.
2. **Metodologia:** descrição das principais decisões de implementação, incluindo uma justificativa para os algoritmos e estruturas de dados escolhidos.

3. **Análise de complexidade:** uma breve análise de complexidade (assim como vista em aula) sobre as principais partes (passos) de sua implementação.

4. **Análise empírica:** para alguns dos casos de teste disponibilizados pelo professor, meça o tempo total que seu programa demora para executar (incluindo tempos de leitura e escrita de arquivos). Após isso, crie uma tabela mostrando a porcentagem desse tempo que foi gasta em cada uma das seguintes etapas: leitura dos dados; cálculo das distâncias; ordenação das distâncias; obtenção da MST; identificação dos grupos; escrita do arquivo de saída. Os valores obtidos nessa tabela estão de acordo com sua análise de complexidade?

O relatório deve ser entregue em formato `.pdf`.

7 Avaliação

- Assim como especificado no plano de ensino, o trabalho vale 10 pontos. Sete pontos referentes à parte de implementação e três pontos referentes ao relatório.
- A parte de implementação será avaliada de acordo com a fração e tipos de casos de teste que seu trabalho for capaz de resolver de forma correta. Casos *pequenos* e *médios* (3 pontos) serão utilizados para aferir se seu trabalho está correto. Casos *grandes* (3 pontos) serão utilizados para testar a eficiência do seu trabalho. Casos *muito grandes* (1 ponto) serão utilizados para testar se seu trabalho foi desenvolvido com muito cuidado e tendo eficiência máxima como objetivo. Todos os casos de teste serão projetados para serem executados em poucos minutos (no máximo 5) em uma máquina com 16GB de RAM.
- Trabalhos com erros de compilação receberão nota zero.
- Trabalhos que usem **variáveis globais** ou a primitiva **goto** receberão nota zero.
- Trabalhos que gerem *segmentation fault* para algum dos casos de teste disponibilizados no AVA serão severamente penalizados na nota.
- Trabalhos com *memory leak* (vazamento de memória) sofrerão desconto na nota.
- Organização do código e comentários valem nota. Trabalhos confusos e sem explicação sofrerão desconto na nota.
- Caso seja detectada **cópia** (entre alunos ou da Internet), todos os envolvidos receberão nota zero. Caso as pessoas envolvidas em suspeita de cópia discordem da nota, amplo direito de argumentação e defesa será concedido. Neste caso, as regras estabelecidas nas resoluções da UFES serão seguidas.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre cópia, acima.)