UNIRIO

RELATÓRIO DE ANÁLISE E PROJETO DE ALGORITMOS

EMPARELHAMENTO ESTÁVEL

ALUNOS: Celso Pacheco Junior, Eduardo Sebastião, Leon Nascimento

Rio de Janeiro, 2020

1. Introdução

Suponha-se que um determinado estudante foi aceito pela universidade A e encontra-se na lista de espera da universidade B (que ele prefere em relação à A). Ou suponhamos que um estagiário tenha sido aceito em um programa de estágio A e esteja na lista da empresa B (que ele também prefere em relação a A).

Não importa o motivo, se o estudante ou o estagiário venha a ser chamado pela sua primeira opção, eles abandonarão a sua opção atual e a deixariam com uma vaga ociosa. Isso levaria a opção abandonada a também chamar algum candidato de sua lista de espera, e deixaria uma vaga ociosa em outro lugar. Essa situação poderia fazer com que terminássemos tendo vagas ociosas em alguns lugares e alunos interessados em outros, mas que não foram aceitos em nenhuma universidade.

Para resolver este problema, vamos considerar um processo seletivo no qual **n** candidatos devem ser designados para **m** universidades. Cada candidato deverá listar as universidades por ordem de preferência, sem a possibilidade de empates, e excluir aquelas em que não deseja estudar. Similarmente, a universidade monta sua ordem de preferência, podendo levar em conta o valor da nota do vestibular, por exemplo.

Assim, poderíamos chegar a uma situação onde haja dois candidatos X e Y e duas universidades A e B com uma vaga cada uma. O candidato X prefere a universidade A e o candidato Y prefere a universidade B, mas a universidade A prefere o candidato Y e a universidade B prefere o candidato X. Nesse caso, não existe nenhuma distribuição que venha a satisfazer todas as preferências, sejam elas de alunos ou universidades. Portanto, se consideramos que as universidades existem atendem aos estudantes, e não o contrário, vamos designar X para universidade A e Y para a universidade B. Assim, o desejo dos candidatos terá prioridade em relação ao das universidades.

O problema em questão é evitar que ocorra uma designação de candidatos para universidades onde há dois candidatos X e Y designados para as universidades A e B, respectivamente, embora Y prefira a universidade A à universidade B, e a universidade A prefira o candidato Y ao candidato X .

Supondo que a situação acima ocorra, o candidato Y tentaria transferir-se para a universidade A, que o admitiria em detrimento do candidato X. Essa mudança seria vantajosa tanto para Y quanto para A. Dessa forma, a distribuição original é considerada instável, pois pode ser subvertida por Y e A agindo juntos, de maneira que ambos sejam beneficiados.

A condição essencial que deve ser satisfeita para uma designação de candidatos para universidades é que ela não tenha instabilidades. Entretanto, não é possível afirmar que em 100% dos casos a possibilidade de se obter uma designação sem instabilidades (ou seja, um "emparelhamento estável") para qualquer combinação de candidatos, universidades e preferências.

Este é o problema do emparelhamento estável, que apresentaremos neste trabalho.

2. Apresentação do Problema e Algoritmo

Dado um conjunto de preferências, é possível uma atribuição entidade-candidato de tal forma que para toda entidade **E**, e para todo possível candidados **C** que não esteja pareado com a entidade E, pelo menos uma das duas situações a seguir acontece:

- **E** prefere cada um de seus possíveis candidatos em vez de **C** ou;
- **C** prefere sua situação atual do que vincular-se a **E**

Assim, objetiva-se que, dados n entidades (E) e n candidatos (C), um <u>emparelhamento estável</u>, onde um emparelhamento é definido como um subconjunto contido no conjunto cartesiano C x E, onde cada elemento de E e C aparece no máximo uma única vez. Em formalismo matemático, temos que um emparelhamento é definido por:

(1) $S \subseteq C \times E$, onde S contém uma tupla (c_i, e_j) onde c_i e e_j aparecem no máximo 1 vez.

Por sua vez, um emparelhamento estável é definido como um emparelhamento onde todos os elementos de C e E aparecem exatamente uma vez. No entanto, um emparelhamento perfeito pode não ser estável. Considere a situação de uma possível lista de preferências entre entidades em C e E, abaixo:

Pessoa/Preferência	**1º**	**2º**	**3º**
Alfredo	Zilda	Yolanda	Ximênia
Bernardo	Ximênia	Yolanda	Zilda
Clodoaldo	Zilda	Ximênia	Yolanda

Pessoa/Preferência	1º	2º	3º
Zilda	Bernardo	Alfredo	Clodoaldo
Yolanda	Alfredo	Bernardo	Clodoaldo
Ximênia	Alfredo	Clodoaldo	Bernardo

Se considerarmos um emparelhamento dos conjuntos {Alfredo, Bernardo, Clodoaldo} e {Zilda, Yolanda e Ximênia}, é possível que um emparelhamento formado, digamos (Alfredo, Yolanda) não seja tão ideal quanto um emparelhamento alternativo como (Alfredo, Zilda), uma vez que embora a Yolanda deseje ficar emparelhada com o Alfredo, este desejaria ficar emparelhado com outro candidato em C, como a Zilda.

Desta forma, dizemos que este par é um par instável. Consequentemente, todo o emparelhamento S gerador deste par é dito um emparelhamento instável. Um emparelhamento estável, por sua vez, é um emparelhamento onde os seguintes requisitos são atendidos:

1. É um emparelhamento perfeito;
2. Não possui pares instáveis.

Assim, o problema a ser resolvido é dado por:

- Para quaisquer listas de preferências, é possível determinar um emparelhamento estável?
- Dadas as listas de preferências de homens e mulheres, é possível determinar um emparelhamento estável, se um existe?

O algoritmo de Gale-Shapley é um algoritmo que resolve o problema do emparelhamento estável. O algoritmo é descrito abaixo:

```
Algoritmo 1: Algoritmo de Gale-Shapley para o Emparelhamento estável

Inicialize cada membro como livre

Enquanto
(uma entidade está livre e não tentou emparelhamento com todos os candidatos){
        Escolha tal indivíduo m
        w ← O primeiro candidato livre
        Se ( w está disponível){
                Emparelhe m e w
        }
        Senão se (w prefere m ao emparelhamento atual){
                Emparelhe m e w
        }
        Senão {
                Não emparelhar m e w
        }
}
```

O algoritmo apresenta complexidade de tempo de $O(n^2)$, e ele de fato encontra um emparelhamento perfeito pois o laço principal somente termina quando todos os homens possuem um emparelhamento.

O emparelhamento encontrado também é estável pois, pelo laço principal do algoritmo, se existe um pareamento instável, isto significa que a tupla formada pelo emparelhamento (m, w) é instável e (m,w') seria um emparelhamento estável. No entanto, isto é impossível, pois se m fez um par com w' e esta o trocou por m', isto significa que m' estava na frente de m nas preferências de w', o que é uma contradição. Consequentemente, o algoritmo consegue gerar um emparelhamento que atende os requisitos do emparelhamento estável.

3. Experimento computacional

3.1 Ambiente computacional: computador, sistema operacional, linguagem

As instâncias foram executadas em uma máquina com a seguinte especificação:

- Processador: Intel Core i7
- Memória RAM: 16 GB RAM DDR4
- Disco Rígido : HD 8200 RPM
- Sistema Operacional: Windows 10 Home Edition
- Ambiente de execução Java:

  java version "1.8.0_111"

  Java(TM) SE Runtime Environment (build 1.8.0_111-b14)

  Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)

  Comando para execução:

  java -Xms512M -Xmx1524M -Xint-Djava.compiler=NONE -showversion -jar casamento.jar

O algoritmo de Gale-Shapley foi implementado na linguagem Java, com a função abaixo representando a implementação do loop principal do algoritmo:

Trecho 1: Algoritmo Gale-Shapley, implementado em Java

```
public int processarGaleShapley(int[][] homensPrefs, int[][] mulheresPrefs, int[][] ranking, boolean
isDebug) {
        int N = homensPrefs.length;
        int[] indiceUltimaMulherProposta = new int[N];
        int[] parceiroAtual = new int[N];
        Lista solteiros = new Lista();
        int qtdInteracoes = 0;

        // Inicializa todos os homens como solteiros e sem proposta feita e as
        // mulheres sem nenhum parceiro.
        for (int i = N - 1; i >= 0; i--) {
                solteiros.inserirInicio(i);
                indiceUltimaMulherProposta[i] = 0;
                parceiroAtual[i] = -1;
        }
        // Enquanto existir homem solteiro
        int solteiro;
        while ((solteiro = solteiros.getPrimeiro()) != -1) {

                int proponente = solteiro;

                int preferidaIndice = indiceUltimaMulherProposta[solteiro]++;

                int preferida = homensPrefs[solteiro][preferidaIndice];

                // Se a mulher ainda não tem nenhum parceiro
                if (parceiroAtual[preferida] == -1) {
                        parceiroAtual[preferida] = proponente;
                        solteiros.retirarInicio();
                        imprimirCasal(++qtdInteracoes, proponente, preferida, isDebug);
                } else {
                        int pAtual = parceiroAtual[preferida];
                        // Se a mulher prefere seu proponente ao seu atual em O(1)
                        if (ranking[preferida][pAtual] < ranking[preferida][proponente]) {
                                imprimirCasal(++qtdInteracoes, proponente, -1, isDebug);
                        } else {
                                solteiros.retirarInicio();
                                solteiros.inserirInicio(pAtual);
                                parceiroAtual[preferida] = proponente;
                                imprimirCasal(++qtdInteracoes, proponente, preferida, isDebug);
                        }
                }
        }
        return qtdInteracoes;
}
```

A mensuração do tempo é realizada de acordo com o seguinte loop na função main do programa, descrita abaixo. No trecho a seguir, algumas partes da função referentes à argumentos de entrada do programa e rotinas de carregamento de dados dos arquivos de referência foram omitidas para manter a concisão do relatório, sem prejuízo do processamento desejado. Devido ao ambiente de execução Java implementar diversas otimizações, como por exemplo a compilação dinâmica, a mensuração do tempo de execução nesta linguagem torna-se um desafio. Para obter uma medida de tempo de execução mais precisa, foi necessário desabilitar os recursos de otimização do ambiente de execução Java. Para isto foram utilizados parâmetros avançados com a finalidade de desabilitar o recurso de compilação dinâmica, tornando o tempo de execução mais consistente. Os parâmetros utilizados foram: ***-Xint*** e -***Djava.compiler=NONE***

Trecho 2: Demonstração do algoritmo de mensuração de tempo, ressaltando a tomada de n-plicatas no processo

```
public static void main(String[] args) throws IOException {

        // Leitura dos parâmetros de tela
        lerParametrosTela(args);

        // - - - - - - - - - - - - - - - - - - - - - - -
        // CARREGANDO ARQUIVO CORRESPONDENTE A INSTÂNCIA NA MEMÓRIA
        // - - - - - - - - - - - - - - - - - - - - - -
        LoadManager lm = new LoadManager();
        Preferencia p = lm.importar(Matrizes.getFileName(tipo, tamanho));
        int[][] prefH = p.getPrefHomens();
        int[][] prefM = p.getPrefMulheres();
        int[][] ranking = p.getRanking();

        // Trechos de códigos omitidos ...

        CasamentoEstavel c = new CasamentoEstavel();
        int iteracoes = 0;

        // - - - - - - - - - - - - - - - - - - - - - - -
        // MENSURAÇÃO DO TEMPO
        // - - - - - - - - - - - - - - - - - - - - - - -
        long estimatedTime = 0;

        for (int i = 0; i < execucoes; i++) {
                long startTime = System.nanoTime();
                iteracoes = c.processarGaleShapley(prefH, prefM, ranking);
                estimatedTime += (System.nanoTime() - startTime);
        }
        estimatedTime = (estimatedTime / execucoes);

        // - - - - - - - - - - - - - - - - - - - - - - -
        // IMPRESSÃO DOS RESULTADOS
        // - - - - - - - - - - - - - - - - - - - - - - -
        System.out.println("Tempo de execução: " + estimatedTime + " nanosegundos.");
}
```

3.2 Instância Teste

Podemos demonstrar a corretude do algoritmo com o seguinte exemplo:

Trecho 3: Exemplo de dados de entrada utilizados pelo programa

```
WORST CASE INSTANCE STABLE MARIAGE
N      5
MEN'S PREFERENCES
   0    1    2    3    4
   1    2    3    0    4
   2    3    0    1    4
   3    0    1    2    4
   0    1    2    3    4
WOMEN'S PREFERENCES
   1    2    3    4    0
   2    3    4    0    1
   3    4    0    1    2
   4    0    1    2    3
   0    1    2    3    4
<Generated by Adriana C. F. Alvim, 2019>
```

Ao executar o programa com o arquivo acima, temos o seguinte resultado:

```
RANKING
4   0   1   2   3
3   4   0   1   2
2   3   4   0   1
1   2   3   4   0
0   1   2   3   4
RESULTADO
1 - m:0 w:0
2 - m:1 w:1
3 - m:2 w:2
4 - m:3 w:3
5 - m:4 w:0
6 - m:0 w:1
7 - m:1 w:2
8 - m:2 w:3
9 - m:3 w:0
10 - m:4 w:1
11 - m:0 w:2
12 - m:1 w:3
13 - m:2 w:0
14 - m:3 w:1
15 - m:4 w:2
16 - m:0 w:3
17 - m:1 w:0
18 - m:2 w:1
19 - m:3 w:2
20 - m:4 w:3
21 - m:0 w:4
Tempo de execucao: 21000000 nanosegundos.
```

Representando o seguinte emparelhamento:
{ (0,4), (4,3), (3,2), (2,1), (1,0) }

Que é um emparelhamento perfeito (cada um dos membros aparece exatamente uma vez) e estável (nenhum dos elementos w prefere outro elemento em m).

3.3. Resultados

As instâncias fornecidas para o teste foram classificadas em 3 categorias: BEST, onde o emparelhamento estável é facilmente encontrado; HARD, onde o emparelhamento estável exige o esgotamento de todos os casos, e ALE, representando uma mistura aleatória de preferências. A tabela a seguir exibe os resultados em nanosegundos para a média dos valores em 1000-plicata das diferentes instâncias do problema.

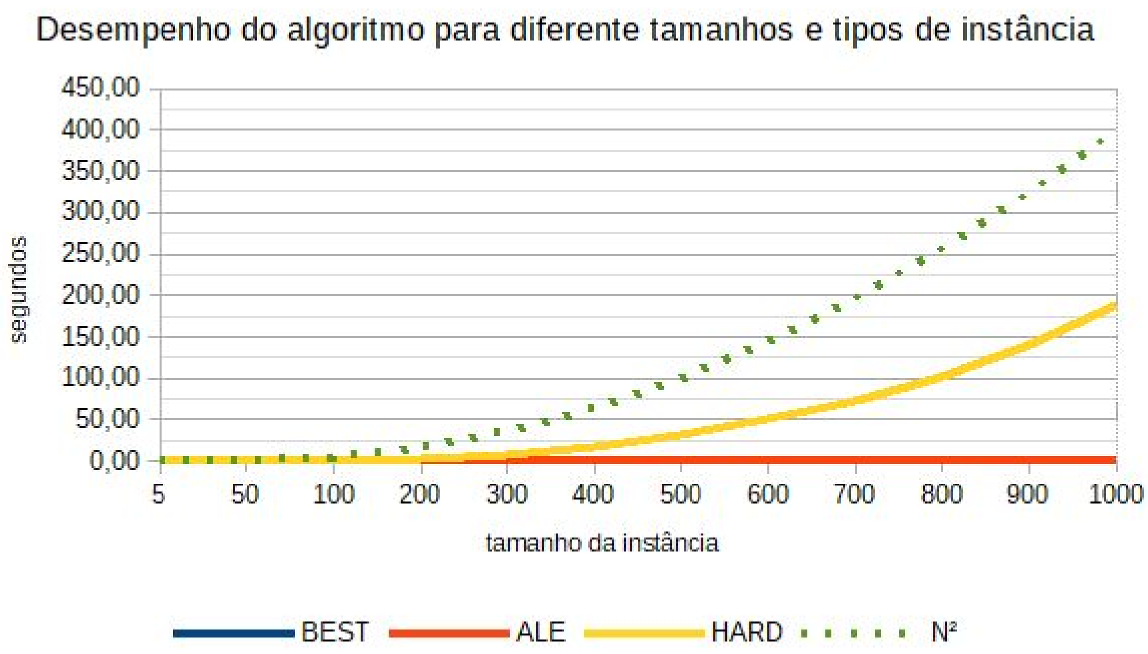
ITERAÇÕES POR INSTÂNCIA

N	ALE	BEST	HARD
5	6	5	21
50	228	50	2451
100	485	100	9901
200	1011	200	39801
300	1737	300	89701
400	2228	400	159601
500	3922	500	249501
600	5314	600	359401
700	4222	700	489301
800	5313	800	639201
900	5965	900	809101
1000	7091	1000	999001

TEMPO EXECUÇÃO POR INSTÂNCIA

N	ALE	BEST	HARD
5	2338	1460	5659
50	36085	12522	667322
100	82561	26144	3534894
200	222936	50293	14203892
300	493257	84581	38609025
400	552855	141150	70513064
500	889712	225305	95552361
600	1295995	232714	141176125
700	1194015	220484	208560036
800	1623796	341855	298235105
900	1616880	281435	367439839
1000	2286857	301788	451868813

Através destes dados, é possível observar que o algoritmo executa, no pior dos casos, em $O(n^2)$, uma vez que, dentro das nossas instâncias de teste, é possível encontrar um valor θ para o qual todas as instâncias estejam limitadas por $\theta . n^2$ (fig 1). Para a figura abaixo, utilizou-se o valor $\theta = \frac{1}{2500}$ para facilitar a visualização.



4. Considerações finais

O algoritmo do emparelhamento estável de Gale-Shapley fornece uma solução ao desafio de criar emparelhamentos que tenham os atributos de serem emparelhamentos perfeitos enquanto mantém a estabilidade de todos os pares gerados. Através deste experimento, foi possível observar que o aumento da complexidade em pior caso se comporta conforme o teorizado, crescendo em um fator de $O(n^2)$.

5. Referências

1. Gale, D.; Shapley, L. S. (1962). "College Admissions and the Stability of Marriage". *American Mathematical Monthly*. **69** (1): 9–14. doi:10.2307/2312726. JSTOR 2312726.
2. Jon Kleinberg e Eva Tardos. Algorithm Design. Addison-Wesley, 2005.
3. Oracle. Launches a Java application. Disponível em: https://docs.oracle.com/javase/7/docs/technotes/tools/windows/java.html. Acessado em 10/10/2020.