Computação Científica e Analise de Dados
Projeto Final - 2023.2

Semianéis e generalização de algoritmos via multiplicação matricial

Matheus do Ó Santos Tiburcio
DRE: 121069701

Sumário

Introdução

Muitos algoritmos, apesar de serem construídos para solucionar problemas distintos, podem possuir uma estrutura base bem semelhante que por muitas vezes pode passar despercebida. É surpreendente que, de fato, algoritmos para checar se há um caminho entre um vértice e outro em um dado grafo e algoritmos de geração ou reconhecimento de linguagens possuam coisas em comum na sua estruturação. Este trabalho visa explorar justamente isso.

O foco geral será na estrutura de multiplicação matricial. Acaba por ocorrer que a forma com que é definida exibe um padrão que surge em muitos lugares, alterando somente o que significa somar ou multiplicar. À medida que nos desprendemos do que essas operações significam ou foram ditas significar, novos algoritmos surgem, mostrando que é possível generalizar ideias que antes pareciam tão distintas.

Serão introduzidas, primeiramente, algumas definições importantes e que serão utilizadas durante todo o texto. Posteriormente, alguns exemplos de algoritmos em grafos são mostrados, assim como o que a multiplicação matricial, de fato, está fazendo em cada. Por fim, mais exemplos na área de linguagens formais e teoria dos autômatos, onde a multiplicação toma formas "mais distintas" e gera resultados interessantíssimos.

Algumas definições

Para dar segmento às discussões do texto, algumas definições precisam ser dadas. Serão definições que serão utilizadas por todo o texto, portanto é importante que se tenha um espaço dedicado à elas e que sejam feitas no início.

Multiplicação matricial

Como comentado na introdução, multiplicação matricial será de grande importância para o resto do que será discutido e por isso é interessante que ela seja bem definida.

Dadas duas matrizes A e B de dimensões $m \times p$ e $p \times n$ respectivamente, seu produto $AB = C$, com C sendo uma matriz $m \times n$ é definido como

$$C = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \sum_{k=0}^{p-1} a_{i,k} \times b_{k,j}$$

Com $a_{i,k}$ e $b_{k,j}$ sendo, respectivamente, o elemento da i-ésima linha e k-ésima coluna de A e o elemento da k-ésima linha e n-ésima coluna de B. O primeiro somatório varre as linhas de A, o segundo as colunas de B e o terceiro o produto dos pares de elementos da linha i de A e coluna j de B.

De maneira mais compacta, o elemento $c_{i,j}$ de C é definido por

$$c_{i,j} = \sum_{k=0}^{p-1} a_{i,k} \times b_{k,j}, \ \forall 0 \leq i \leq m-1, 0 \leq j \leq n-1$$

A próxima definição auxiliará a generalizar essa ideia.

Monóides e semianéis

Antes de se definir um semianel, é interessante definir o que é um monóide, uma estrutura mais simples.

Um **monóide** é uma 3-tupla $(S, \oplus, 0)$, onde

S: Um conjunto qualquer não-vazio
$\oplus$: Uma operação binária usualmente chamada de "soma"
0: O elemento neutro de $\oplus$, com $0 \in S$

Que satisfaça as seguintes propriedades para qualquer $x, y, z \in S$:

1. $x \oplus y = z$ (totalidade)
2. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ (associatividade)
3. $0 \oplus x = x \oplus 0 = x$ (elemento neutro da soma)

A assinatura de $\oplus$ é $S \times S \to S$.

Com a definição dada acima, fica mais fácil definir um semianel. Um **semianel** pode ser definido como uma 5-tupla $(S, \oplus, \odot, 0, 1)$ onde

S: Um conjunto qualquer não-vazio
$\oplus$: Uma operação binária usualmente chamada de "soma"
$\odot$: Uma operação binária usualmente chamada de "produto"
0: O elemento neutro de $\oplus$, com $0 \in S$
1: O elemento neutro de $\odot$, com $1 \in S$

Que satisfaça as seguintes propriedades para qualquer $x, y, z \in S$:

1. $(S, \oplus, 0)$ é um monóide com uma propriedade extra
   - $x \oplus y = y \oplus x$ (comutatividade)
2. $(S, \odot, 1)$ é um monóide
3. $\odot$ distribui sob $\oplus$ por ambos os lados de modo que
   - $x \odot (y \oplus z) = x \odot y \oplus x \odot z$
   - $(x \oplus y) \odot z = x \odot z \oplus y \odot z$
4. $0 \odot x = x \odot 0 = 0$

Como soma e multiplicação de matrizes utilizam somente de operações binárias de soma e multiplicação entre elementos, é possível provar que, dado um semianel, o conjunto $Mat(S)$ de matrizes cujos elementos pertencem à S conjunto do semianel formam também um semianel.

Um padrão conhecido é o de matrizes sob o corpo dos reais $(\mathcal{R})$ e as operações usuais de soma e produto. Note que $(\mathcal{R}, +, \times, 0, 1)$ é um semianel.

Isto é um primeiro indício de que, usando a definição de semianéis, uma generalização de multiplicação entre matrizes pode ser feita. De fato, definindo um semianel, se tem tudo que é necessário para definir multiplicação matricial sob esse semianel. A definição é feita abaixo.

Dado um semianel $S = (U, \oplus, \odot, 0, 1)$, a operação de multiplicação matricial sob S é definida por

$$\bigoplus_{i=0}^{m-1} \bigoplus_{j=0}^{n-1} \bigoplus_{k=0}^{p-1} a_{i,k} \odot b_{k,j}$$

Na forma compacta:

$$c_{i,j} = \bigoplus_{k=0}^{p-1} a_{i,k} \odot b_{k,j}, \ \forall 0 \leq i \leq m-1, 0 \leq j \leq n-1$$

Porém, por motivos que serão melhor explicados posteriormente, uma operação a mais será desejada, chamada de fecho. Semianéis que possuam tal propriedade são chamados de **semianéis fechados**. Um semianel fechado será uma 6-tupla $(S, \oplus, \odot, 0, 1, *)$, onde $(S, \oplus, \odot, 0, 1)$ é um semianel e $*$ é uma operação unária chamada usualmente de fecho cuja definição é

$$x^* = 1 \oplus x \oplus (x \odot x) \oplus .... = \bigoplus_{i=0}^{\infty} x^i$$

Para todo x em S. Aqui a noção de potência é a usual, mas com o produto $\odot$ no lugar do usual $\times$. Desse modo, x^0 é 1 para qualquer x em S.

Ponto fixo

A noção de ponto fixo será de extrema importância para a explicação dos algoritmos que virão. Acredito que um bom jeito de começar é definindo o ponto fixo de uma função.

Seja $f : A \to B$ para algum conjunto A e B, seu ponto fixo é um valor a tal que

$$f(a) = a$$

Repare que para isso $a \in A \cap B$. Além disso, a não necessariamente é único. Tome $f(x) = x$, por exemplo. Qualquer valor x é ponto fixo desta função.

A essência do ponto fixo é essa, não se altera não importe quantas vezes se aplique f. Às vezes o ponto fixo não vem de forma direta, mas pode ser obtido, por exemplo, através de convergência. A ideia é aplicar f sucessivas vezes em um valor qualquer e avaliar se converge ou não. Caso convirja, o resultado é ponto fixo de f. Um jeito de representar isso é

$$\lim_{n \to \infty} f^n(x)$$

Onde f^n é aplicar f n vezes na entrada inicial x. O ponto fixo será o resultado deste limite.

Para ficar mais claro vamos fazer um exemplo. Suponha a matriz $A = \begin{bmatrix} 1 & -\frac{1}{2} \\ 0 & \frac{1}{2} \end{bmatrix}$, se deseja o valor obtido de

$$\lim_{n\to\infty} A^n x$$

De primeiro momento pode parecer um pouco difícil de determinar isso só olhando. Felizmente essa matriz admite uma base formada por seus autovetores. Os autovetores associados à seus autovalores 1 e $\frac{1}{2}$ são, respectivamente, $y = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ e $z = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix}^T$. Por ser possível construir uma base de autovetores, quaisquer valores de x podem ser descritos nesta base $x = ay + bz$ de modo que o limite se torne

$$\lim_{n\to\infty} A^n(ay + bz) = \lim_{n\to\infty} aA^n y + bA^n z$$

Pela definição de autovalores e autovetores, $Mw = \lambda w$, com w autovetor e λ autovalor de M. Portanto

$$\lim_{n\to\infty} a1^n y + b(\frac{1}{2})^n z = ay$$

Desse modo, qualquer múltiplo de y é ponto fixo de A.

O ponto fixo de uma equação em específico será de grande importância.

$$x = a \odot x \oplus 1$$

Repare que seu ponto fixo é a^*! Tome $x = a^*$

$$a^* = a \odot a^* \oplus 1$$

$$a^* = a \odot \bigoplus_0^\infty a^i \oplus 1$$

$$a^* = \bigoplus_1^\infty a^i \oplus 1$$

$$a^* = \bigoplus_1^\infty a^i \oplus a^0$$

$$a^* = \bigoplus_0^\infty a^i$$

Essa equação surgirá em alguns momentos do texto.

Note também que muitas vezes 1 pode ser trocado por outros valores. Nesse caso se obtém

$$a^* \odot y = ax \oplus y$$

Mas esse caso não será tão explorado quanto o primeiro.

Algoritmos em grafos

Após toda a discussão de definições, nesta seção serão demostrados vários exemplos de problemas em grafos que podem ser solucionados usando multiplicação matricial com operações diversas.

Soma dos caminhos

Um modo de representar grafos é com o uso de matrizes de adjacências, tome o seguinte grafo abaixo.

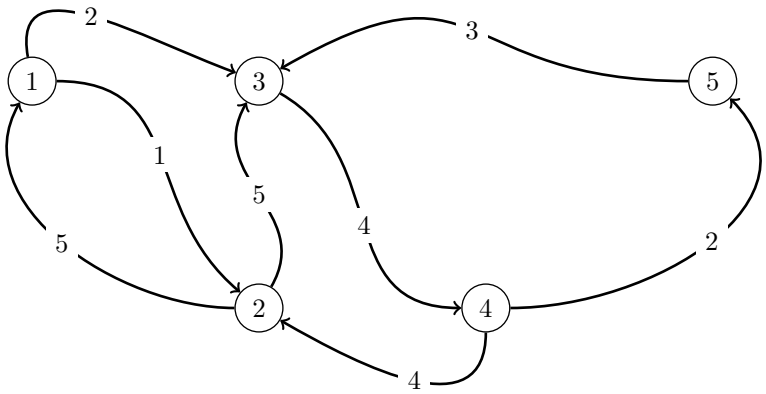

Figura 1: grafo de exemplo

Sua matriz de adjacência será

$$G = \begin{bmatrix} 0 & 1 & 2 & 0 & 0 \\ 5 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 \\ 0 & 4 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \end{bmatrix}$$

Onde o elemento $g_{i,j}$, considerando os índices indo de 1 à 5, indica que há uma aresta entre o vértice v_i e v_j com o peso indicado. Aqui 0 é usado para indicar que não há aresta entre dois vértices.

Usando soma e produto usuais, multiplicar G por sí próprio é encontrar a soma de todos os caminhos de comprimento 2. Entenda comprimento como o número de arestas no caminho. Mas nesse contexto, é tomado o produto dos pesos das arestas no mesmo caminho.

Isso se dá pela definição de multiplicação matricial. Neste caso é

$$c_{i,j} = \sum_{k=0}^{p-1} a_{i,k} \times b_{k,j}$$

Na matriz de adjacências, $a_{i,k} \times b_{k,j}$ faz o produto dos pesos de uma aresta de v_i para v_k e de v_k para v_j. O somatório só indica que todos esses caminhos entre v_i e v_j que passem por algum vértice v_k são somados.

$$G^2 = \begin{bmatrix} 5 & 0 & 5 & 8 & 0 \\ 0 & 5 & 10 & 20 & 0 \\ 0 & 16 & 0 & 0 & 8 \\ 20 & 0 & 26 & 0 & 0 \\ 0 & 0 & 0 & 12 & 0 \end{bmatrix}$$

De fato, repare que o único caminho de tamanho 2 de 5 para 4 é acessando as aresta $5 \xrightarrow{3} 3$ e $3 \xrightarrow{4} 4$. Como seus pesos são respectivamente 3 e 4 se tem 3×4.

Para o caso G^3 a ideia é a mesma. De G^2 a soma dos caminhos de tamanho dois já foi obtida, agora se quer caminhos de tamanho 3. $g^2_{i,j}$ Já guarda os caminhos da forma $v_i \to v_k \to v_j$, multiplicando por G se obtém caminhos entre v_i e v_j que passem por dois vértices intermediários v_k e v_l de modo que $a_{i,k} \times b_{k,j}$ representa o produto da soma dos caminhos da forma $v_i \to v_k \to v_l$ e $v_l \to v_j$. O somatório, como antes, soma todos esses produtos.

Caso se queira encontrar a soma de todos os caminhos de tamanho 1 e 2, basta somar G e G^2. De modo geral, caso queira a soma todos os caminhos de tamanho $k \leq n$ até n basta fazer

$$\sum_{i=k}^{n} G^i$$

Repare que essa ideia é bem parecida com a de fecho e poderia ser determinado encontrando o ponto fixo da equação [1]. O problema é que, como sempre são feitas somas, o somatório da definição de fecho não necessariamente converge aqui. Uma aplicação dessas ideias é em grafos computacionais.

Grafos computacionais

Grafos computacionais são utilizados para calcular derivadas parciais de maneira mais rápida. Nele, cada vértice é uma função e cada aresta $a \to b$ representa a derivada parcial de a em relação à b. Um exemplo de grafo computacional é dado abaixo para $f(x) = (sen\ x)(y^2 + z + 2)(1 + cos\ y)(y + x)$. No grafo cada fragmento da função é representado por nós q_i.

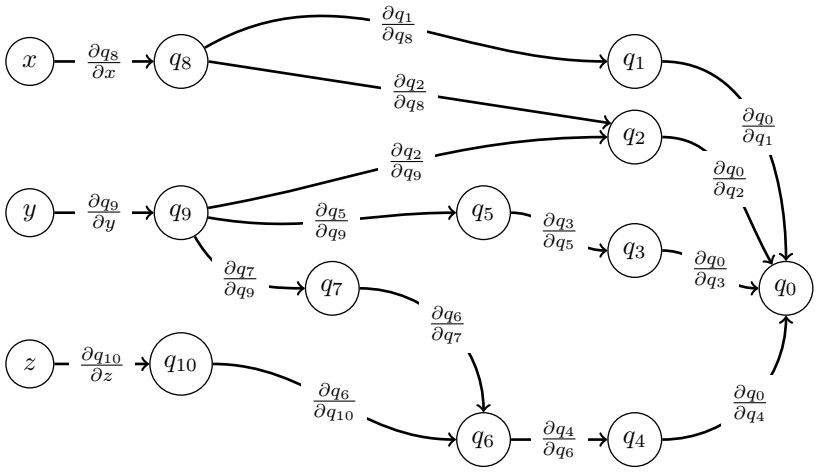

Figura 2: grafo computacional de $f(x)$

Com q_i sendo

$q_0 = f(x) = (senx)(y^2 + z + 2)(1 + cosy)(y + x)$
$q_1 = senx = sen(q_8)$
$q_2 = y + x = q_8 + q_9$
$q_3 = y^2 + z + 2 = q_5 + 2$
$q_4 = 1 + cosy = 1 + q_6$
$q_5 = cosy = cos(q_9)$
$q_6 = y^2 + z = q_7 + q_{10}$
$q_7 = y^2 = q_9^2$
$q_8 = x$
$q_9 = y$
$q_{10} = z$

Renomeie x, y, z como q_{11}, q_{12}, q_{13}, respectivamente, matriz de adjacência desse grafo é

$$G = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial q_0}{\partial q_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial q_0}{\partial q_2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial q_0}{\partial q_3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{\partial q_0}{\partial q_4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{\partial q_3}{\partial q_5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{\partial q_4}{\partial q_6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial q_6}{\partial q_7} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{\partial q_1}{\partial q_8} & \frac{\partial q_2}{\partial q_8} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{\partial q_2}{\partial q_9} & 0 & 0 & \frac{\partial q_5}{\partial q_9} & 0 & \frac{\partial q_7}{\partial q_9} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial q_6}{\partial q_{10}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial q_8}{\partial q_x} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial q_9}{\partial q_y} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{\partial q_{10}}{\partial q_z} & 0 & 0 & 0
\end{bmatrix}$$

Onde a entrada $m_{i,j}$ indica se há uma aresta de q_i a q_j, com i, j números entre 0 e 13.

Pela regra da cadeia, as derivadas parciais de uma função em relação à outra é justamente o produto de todas as arestas pelo caminho entre ambos. No caso de haver mais de um caminho, se soma o resultado final do produto destes caminhos.

Com multiplicação matricial usual, foi definido que G^n possui os caminhos de tamanho n entre cada par de vértices do grafo. Apesar de não haver convergência sempre, é notável que o maior caminho do grafo é de comprimento 5, portanto basta computar

$$\sum_{i=1}^{5} G^i$$

Não é necessariamente G^*, mas soluciona o que era desejado: encontrar as derivadas parciais. Nesse contexto, a parcial de q_j em relação à q_i estará na entrada $g^5_{i,j}$. Note que a matriz ficou bem esparsa. Isso muitas vezes pode ocorrer em matrizes de adjacências de grafos e há métodos de multiplicação matricial mais rápidos focados em matrizes esparsas, mas que não serão discutidos aqui.

Caminho mínimo

Os semianéis usados até agora foram $(S, +, *, 0, 1)$, com S podendo ser os reais, por exemplo. A primeira mudança de operações será feita agora.

As operações agora serão de mínimo, denotado por min e a usual de soma. O semianel em uso será $(\mathcal{R} \cup \{+\infty\}, \min, +, +\infty, 0)$, que também é conhecido como "semianel minplus" devido a suas operações ou "semianel tropical" em homenagem ao cientista computacional brasileiro Imre Simon. No lugar dos reais, outros conjuntos podem ser usados como, por exemplo, os naturais. A

multiplicação matricial agora toma a forma

$$\min_{i=0}^{m-1} \min_{j=0}^{n-1} \min_{k=0}^{p-1} a_{i,k} + b_{k,j}$$

Na forma compacta:

$$c_{i,j} = \min_{k=0}^{p-1} a_{i,k} + b_{k,j}$$

A interpretação de G^2 agora é um pouco diferente. $a_{i,k}+b_{k,j}$, para cada k, soma os pesos das arestas no caminho de v_i para v_k e de v_k para v_j. Após esta soma, o mínimo entre eles é obtido. Desse modo, G^2 representa o caminho de tamanho 2 de menor peso entre v_i e v_j. Para fazer sentido, as arestas inexistentes antes representadas por 0, serão agora representadas por $+\infty$, o elemento neutro de min. O único caso em que essa troca não ocorre, é nas entradas da diagonal principal, pois pode-se considerar que um caminho de tamanho 0 entre um vértice e si próprio não tem custo.

$$G = \begin{bmatrix} 0 & 1 & 2 & +\infty & +\infty \\ 5 & 0 & 5 & +\infty & +\infty \\ +\infty & +\infty & 0 & 4 & +\infty \\ +\infty & 4 & +\infty & 0 & 2 \\ +\infty & +\infty & 3 & +\infty & 0 \end{bmatrix}$$

Semelhante à ideia apresentada para a soma de caminhos, para se obter o caminho mínimo de tamanho entre 1 e n, basta somar $\min_{i=0}^{n} G^i$.

Para se obter o, de fato, menor caminho entre todos os pares de vértice, pode-se usar a ideia de fecho. Este é o primeiro caso em que, de fato, o fecho converge, mas somente porque não há ciclos de peso negativo no grafo ou auto-arestas (de um vértice para si mesmo) com pesos diferentes de 0.

Para provar a convergência, suponha que o número de vértices de um grafo é igual a n. Caso se compute G^{n+1}, analisando $a_{i,k} + b_{k,j}$ se obtém o custo do caminho mínimo de tamanho n entre v_i e v_k e o custo da aresta entre v_k e v_j que por fim são somados. Mas como o caminho é de tamanho n e não há ciclos negativos por hipótese, o k que tornará o custo mínimo é justamente $k = j$. Note que, para esse argumento funcionar, não se pode ter auto-arestas com valores diferentes de 0, já que, como $k = j$, se obteria um valor maior sempre, se $g_{j,j} > 0$ ou menor sempre, se $g_{j,j} < 0$. De maneira semelhante, caso houvessem ciclos negativos o mínimo poderia sempre diminuir passando por esse ciclo.

No grafo da figura 1 se tem esse caso que converge, portanto o caminho mínimo entre todos os pares vértices pode ser descrito como

$$G^* = \min_{i=0}^{\infty} G^i$$

Como há 5 vértices nele, calcular, qualquer G^n com $n \geq 5$ será exatamente igual à G^5 fazendo com que $\min(G^5, G^n) = G^5$. O grafo de caminhos mínimos da figura 1 é mostrado abaixo. É um grafo completo! Para evitar desenhar todas as arestas, os pesos estão na forma (a, b) de modo que a indique o peso da aresta

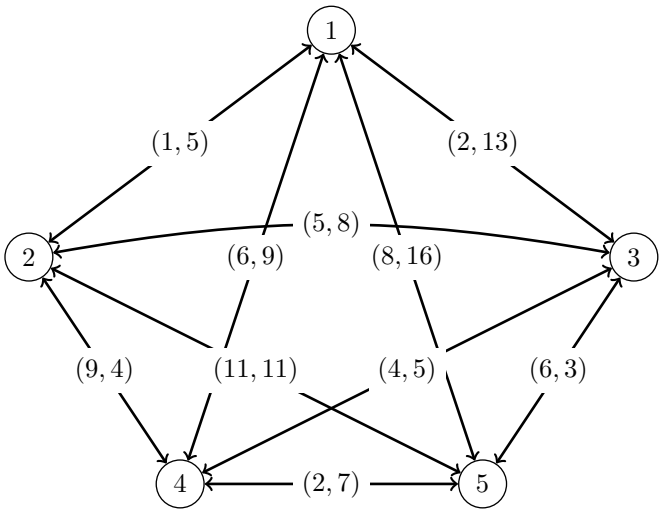

Figura 3: Caminhos mínimos do grafo 1

do vértice de menor índice para o de maior e b do vértice de maior índice para o de menor. A matriz de adjacências do grafo da figura 3 é

$$G^* = \begin{bmatrix} 0 & 1 & 2 & 6 & 8 \\ 5 & 0 & 5 & 9 & 11 \\ 13 & 8 & 0 & 4 & 6 \\ 9 & 4 & 5 & 0 & 2 \\ 16 & 11 & 3 & 7 & 0 \end{bmatrix}$$

Caminho máximo

Assim como se é possível determinar o caminho mínimo entre todos os vértices, é possível determinar o caminho máximo, isto é, de maior custo entre todos os vértices. Toda a explicação acima ainda é válida, mas o semianel será alterado para $(\mathcal{R} \cup \{-\infty\}, \max, +, -\infty, 0)$. Assim como o semianel de usado no caminho mínimo, este também é chamado de "semianel tropical" ou "semianel maxplus" devido a suas operações. A multiplicação matricial será

$$\max_{i=0}^{m-1} \max_{j=0}^{n-1} \max_{k=0}^{p-1} a_{i,k} + b_{k,j}$$

Compactamente:

$$c_{i,j} = \max_{k=0}^{p-1} a_{i,k} + b_{k,j}$$

Fecho transitivo

O problema do fecho transitivo é determinar se, dado um vértice v_i de um grafo G, é possível chegar em algum vértice v_j de G. Aqui, portanto, o problema não é achar o caminho mínimo entre dois vértices, mas achar pelo menos um caminho. O problema se resume a fazer checagens da existência de arestas entre vértices no grafo, que pode ser tratado usando a matriz de adjacência

de uma maneira semelhante à feita para o problema do caminho mínimo entre todos os pares de vértices, mas em um semianel diferente. As operações que serão utilizadas serão os conectivos lógicos "e"($\wedge$) e "ou"($\vee$) para representarem soma e produto, respectivamente. O semianel completo será $(\{0,1\}, \wedge, \vee, 1, 0)$ que por muitas vezes pode ser chamado de "semianel booleano". Aqui trate 1 como `TRUE` e 0 como `FALSE`. A definição de multiplicação matricial fica

$$\bigvee_{i=0}^{m-1} \bigvee_{j=0}^{n-1} \bigvee_{k=0}^{p-1} a_{i,k} \wedge b_{k,j}$$

Na forma compacta:

$$c_{i,j} = \bigvee_{k=0}^{p-1} a_{i,k} \wedge b_{k,j}$$

Assim, $a_{i,k} \wedge b_{k,j}$ responde a pergunta de se há uma aresta entre v_i e v_k e uma aresta entre v_k e v_j em G de modo que o caminho $v_i \to v_k \to v_j$ exista. Após isso, $\vee$ é passado em todas essas checagens a fim de achar pelo menos uma que seja verdadeira. Ou seja, que haja pelo menos um k tal que $v_i \to v_k \to v_j$ exista.

Tome o grafo da imagem 1 como exemplo novamente. Para tratar esse semianel é preciso tornar todas as entradas da matriz de adjacências original em 1 e 0. Isso pode ser feito fazendo todas as entradas com valores distintos de 0, ou seja, em que exista uma aresta de um dado peso, se tornarem 1 e o restante se mantendo 0. Outro modo de tratar esse semianel é o expandindo para todos os reais: $(\mathcal{R}, \wedge, \vee, 1, 0)$ e redefinindo as operações para que qualquer valor diferente de 0 seja tratado como `TRUE`. A primeira opção será a utilizada aqui. A matriz de adjacências do grafo fica:

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Semelhante como foi no problema do caminho mínimo, aqui G^n indica se há um caminho de tamanho n entre cada par de vértices.

Esse caso, diferente do caminho mínimo, sempre converge! Como a exigência é a de que haja pelo menos um caminho entre os vértices, basta somar as potências de G até o seu número de vértices. O ponto fixo fica

$$G^* = \bigvee_{i=0}^{\infty} G^i$$

Pelo argumento acima, o ponto fixo será $\bigvee_{i=0}^{n} G^i$, n número de vértices de G. A ideia intuitiva é de que primeiro se checará se o vértice está diretamente ligado a outro (G), depois se há um vértice entre ambos que os liga (G^2), depois se há dois entre ambos que os liga (G^3) e assim sucessivamente.

Linguagens formais e teoria dos autômatos

Os próximos problemas utilizam de alguns conceitos de linguagens formais e teoria dos autômatos que serão explicados a seguir.

Definições

Um alfabeto Σ é um conjunto finito de símbolos quaisquer. Símbolos podem formar palavras, ou strings, e sua definição é justamente essa: uma sequência finita de símbolos. O tamanho de uma palavra é o número de símbolos que ela possui. Dentre elas, há uma palavra especial chamada palavra vazia que denotarei por ε e é definida como a única palavra de tamanho 0.

O conjunto de todas as palavras que podem ser formadas com símbolos somente de Σ é denotada por Σ^*. A intuição de $*$ é a mesma de antes, se tem todas as palavras de tamanho 0 (Σ^0), cujo único elemento é ε, as palavras de tamanho 1 (Σ^1) e assim sucessivamente. De maneira mais compacta:

$$\Sigma^* = \bigcup_{i=0}^{+\infty} \Sigma^i$$

Uma linguagem é um conjunto de palavras que possuam símbolos vindos de um alfabeto em específico. Repare que por si só Σ^* é uma linguagem.

Abaixo há alguns exemplos destes conceitos.

$\Sigma = \{\mathtt{a} \mid \mathtt{a}$ é um símbolo do alfabeto latino$\}$
$L = \{w \mid$ palavras do português$\}$

$\Sigma = \{\mathtt{a}, \mathtt{h}, \mathtt{m}\}$
$L = \{w \mid w = (\mathtt{mua} \cup \varepsilon)(\mathtt{ha})^*\} = \{w | w$ é uma string que pode começar com "`mua`" ou não e depois é uma sequência de 0 ou mais strings "`ha`"$\}$

as chaves da notação de conjuntos usual serão muitas vezes omitidas, mas tenha em mente que as operações estão sendo feitas em conjuntos. Além disso, as palavras "palavra" e "string" significam o mesmo e ambas serão usadas.

Expressões regulares

O último exemplo dava uma linguagem representada na forma $(\mathtt{mua} \cup \varepsilon)(\mathtt{ha})^*$. Expressões desse tipo, que só usam concatenação de palavras, união e $*$, são chamadas de expressões regulares. Essas expressões dizem, de maneira compacta, exatamente quais palavras são ou não aceitas pela linguagem descrita por elas. Linguagens que podem ser descritas por essas expressões são chamadas de linguagens regulares.

Outro modo de representar essas linguagens é através de um tipo de grafo chamado de autômato. Nele, os vértices são denotados "estados" e as arestas possuem como peso símbolos, de maneira que a palavra dada a esse autômato vá sendo "consumida" à medida que as transições de estados são feitas. Tome

a linguagem da expressão regular $(\texttt{mua} \cup \varepsilon)(\texttt{ha})^*$ de exemplo. Um autômato possível para ela é

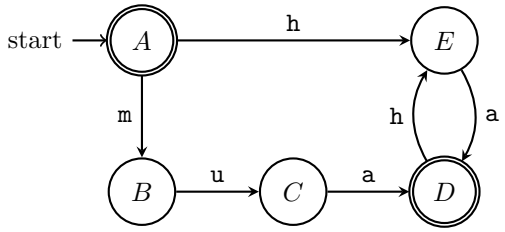

Figura 4: Autômato de exemplo da linguagem reconhecida por $(\texttt{mua} \cup \varepsilon)(\texttt{ha})^*$

Os estados marcados são denotados estados finais e o estado com um "start" ao lado é o estado inicial. Caso, após consumir toda a palavra, o estado atual seja um final, a palavra é aceita pela linguagem. Abaixo um exemplo com a palavra `muahaha` representando a sequência de transição de estados. Acima da seta há o que foi consumido.

$$A \xrightarrow{\texttt{m}} B \xrightarrow{\texttt{u}} C \xrightarrow{\texttt{a}} D \xrightarrow{\texttt{h}} E \xrightarrow{\texttt{a}} D \xrightarrow{\texttt{h}} E \xrightarrow{\texttt{a}} D$$

Figura 5: transições da palavra `muahaha`

Como o último estado é um final, a palavra `muahaha` é aceita.

Encontrar o padrão de strings aceitas por uma linguagem, portanto, pode ser tratado encontrando todos os caminhos do estado inicial para algum final. É um problema semelhante ao do fecho transitivo anteriormente discutido. O semianel que será utilizado é $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \varepsilon)$, com união de conjuntos e concatenação de strings sendo as operações de união e concatenação, respectivamente, e $\mathcal{P}(\Sigma^*)$ denotando o conjunto das partes de Σ^*. E a multiplicação matricial fica

$$\bigcup_{i=0}^{n-1} \bigcup_{i=0}^{m-1} \bigcup_{i=0}^{p-1} a_{i,k} \cdot b_{k,j}$$

Onde cada entrada da matriz C resultante é

$$c_{i,j} = \bigcup_{i=0}^{p-1} a_{i,k} \cdot b_{k,j}$$

Semelhante aos grafos computacionais, $a_{i,k} \cdot b_{k,j}$ está tomando o produto entre as arestas do grafo que nesse caso é a concatenação. A ideia intuitiva é de que se precisa consumir a string presente em $a_{i,k}$ e depois a string presente em $b_{k,j}$. A união faz o papel de agrupar todos os caminhos possíveis entre um estado e outro. Outro modo de ver é: agrupar todas as strings que podem ser consumidas para se chegar ao outro estado. A expressão regular então é a união das strings dos caminhos de qualquer tamanho do estado inicial para cada estado final. Em outras palavras, a expressão regular do autômato é a união das entradas $a^*_{i,j}$,

com i sendo o estado inicial e j sendo cada estado final, do fecho de sua matriz de transição. A matriz A que representa as transições do autômato é

$$A = \begin{bmatrix} \emptyset & \texttt{m} & \emptyset & \emptyset & \texttt{h} \\ \emptyset & \emptyset & \texttt{u} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \texttt{a} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \texttt{h} \\ \emptyset & \emptyset & \emptyset & \texttt{a} & \emptyset \end{bmatrix}$$

Onde as linhas e colunas são referentes A, B, C, D e E, nesta ordem. Aqui $\emptyset$, elemento neutro da união, é utilizado no lugar de 0 para representar a inexistência de arestas. Semelhante a como ocorreu em outros casos, a potência A^n determina caminhos de tamanho n entre os estados.

Infelizmente a estratégia de reduzir o fecho para uma soma finita aqui não será suficiente. Repare que para qualquer n ímpar ≥ 3 há pelo menos uma palavra deste tamanho que pertence à linguagem, é um conjunto infinito. Logo uma solução que resta é encontrar o ponto fixo da equação

$$X = AX \cup \varepsilon$$

Que é exatamente A^*. Felizmente há modos de se fazer isso, seja resolvendo diretamente o sistema linear do autômato ou tomando fórmulas fechadas [6]. O resultado, ou um dos possíveis, seria justamente $(\texttt{mua} \cup \varepsilon)(\texttt{ha})^*$.

Algoritmo CYK e geração de linguagens

Aqui um algoritmo de reconhecimento de palavras em linguagens é introduzido, assim como algumas definições pertinentes.

Um nível acima na hierarquia

Nem toda linguagem pode ser representada por uma expressão regular ou um autômato como mostrado acima. De fato, as expressões regulares são o último (penúltimo dependendo do autor[4]) nível de uma hierarquia conhecida como "hierarquia de Chomsky"[4]. O nível seguinte abrange as linguagens livres de contexto, amplamente utilizada em linguagens de programação. Uma possível definição é dada abaixo.

Uma **linguagem livre de contexto** é uma linguagem que pode ser gerada (ou reconhecida) por pelo menos uma gramática livre de contexto. Gramáticas, assim como expressões regulares e autômatos, geram (ou reconhecem) linguagens.

Uma **gramática livre de contexto** é uma 4-tupla (N, Σ, P, S) onde

N: Conjunto finito de elementos chamados "variáveis" ou "não-terminais".
Σ: Conjunto finito de elementos chamados "tokens" ou "terminais"
P: Conjunto finito de produções da gramática
S: Variável (ou não-terminal) inicial

Satisfazendo

1. $N \cap \Sigma = \emptyset$
2. $S \in N$
3. Toda produção da gramática é da forma
    - $A \to \alpha$, com $A \in N$ e α sequência de variáveis e tokens.

Como exemplo, tome a linguagem gerada pelo autômato da figura 4. Uma possível gramática para essa linguagem seria $(\{A, B, C, D, E\}, \{a, h, m\}, P, A)$, com as produções em P sendo

$$A \to \texttt{m}B$$
$$A \to \texttt{h}E$$
$$B \to \texttt{u}C$$
$$C \to \texttt{a}D$$
$$D \to \texttt{h}E$$
$$E \to \texttt{a}D$$
$$E \to \texttt{a}$$

Figura 6: Possível gramática da linguagem gerada por $(\texttt{mua} \cup \varepsilon)(\texttt{ha})^*$

De maneira mais compacta:

$$A \to \texttt{mua}H$$
$$A \to H$$
$$H \to haH$$
$$H \to \varepsilon$$

Figura 7: Possível versão mais compacta equivalente à figura 6

De fato, como as linguagens livres de contexto estão acima na hierarquia, qualquer linguagem regular é também livre de contexto.

As produções servem como substituição de modo que o lado esquerdo seja trocado pelo direito. A sequência de produções que geram `muahaha` podem ser representadas por o que é chamado de "árvore de derivação", por exemplo

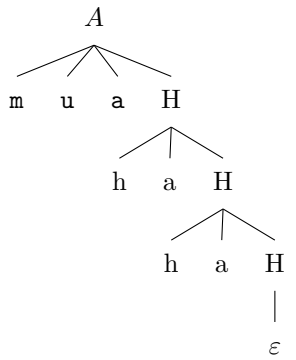

Figura 8: Árvore de derivação da palavra `muahaha`

Uma sequência de produções é chamada de derivação. Dada uma palavra, caso haja pelo menos uma árvore de derivação da gramática cuja as folhas da árvore formem estritamente a palavra, então ela é reconhecida pela gramática.

Seria desejável automatizar esse processo. Quer dizer, fixada uma gramática, para qualquer palavra que seja avaliada uma resposta de "sim" ou "não" deve ser emitida indicando se a palavra pertence ou não à linguagem gerada pela gramática. Os próprios autômatos são uma tentativa de automatização desse processo, porém não são os únicos.

O uso de parsers, que recebem uma palavra e devolvem, muitas vezes, uma (ou mais) árvores de derivação completas caso pertença à linguagem, é um outro modo de automatização. Um exemplo de algoritmo é dado em seguida.

Algoritmo CYK matricial

Antes de introduzir esse algoritmo, é O algoritmo de CYK (Cocke-Younger-Kasami)[4] é um algoritmo de parsing que abrange qualquer linguagem livre de contexto. Para que o algoritmo funcione, a gramática deve estar na forma normal de Chomsky (CNF), como converter uma gramática para essa forma não será discutido aqui, mas toda gramática livre de contexto pode ser convertida para essa forma. A forma normal de Chomsky possui somente 3 tipos de produção

- $S \to \varepsilon$
- $A \to BC$
- $A \to a$

Somente a variável inicial pode produzir a palavra vazia e todas as outras possuem produções que levem à outras duas variáveis ou a um token. O algoritmo, dada uma string de tamanho n, com a ajuda de programação dinâmica mapeia todas as substrings de tamanho 1 a tamanho n, buscando na gramática que variáveis podem produzi-las, caso possam.

O algoritmo, por exemplo, busca todas as variáveis que produzam alguma substring de tamanho 1 (tokens) na palavra, em seguida, para substrings de tamanho 2, verifica se há alguma variável A que produza BC, onde B produz o primeiro token da palavra e C o segundo. Esse raciocínio se segue até o tamanho n em que caso S variável inicial produza essa substring, então a string é aceita. A programação dinâmica ajuda no fato de que, para substrings maiores, pode-se subdividí-las em substrings já previamente calculadas.

Repare que, analisando a árvore de derivação, o CYK faz o parsing iniciando nas folhas e subindo a árvore até a raiz de modo que caso a raiz seja exatamente a variável inicial, então a string é aceita. Esse modo de construção de derivação é chamado de bottom-up em alusão ao fato de ir de baixo para cima.

Repare que há duas ações principais sendo feitas: checar se alguma variável produz tal substring e, caso produza, agrupar todas que consigam. Há aqui

duas operações!

Diferente dos outros semianéis, este será um pouco mais maquiavélico. Tanto as operações, quanto, principalmente, a construção da matriz tem de ser feita com um certo cuidado. O semianel será $(S, \cup, \vdash, \emptyset, \varepsilon)$. A soma será a união usual de conjuntos. $\vdash$ será um construtor de conjuntos com a seguinte definição

$$\alpha \vdash \beta = \{X \mid X \to \alpha\beta\}$$

α e β sequência de tokens e variáveis e X variável da gramática. A multiplicação matricial agora pode ser completamente definida

$$\bigcup_{i=0}^{n-1} \bigcup_{i=0}^{m-1} \bigcup_{i=0}^{p-1} a_{i,k} \vdash b_{k,j}$$

Onde cada entrada da matriz C resultante é

$$c_{i,j} = \bigcup_{i=0}^{p-1} a_{i,k} \vdash b_{k,j}$$

Portanto $\vdash$ faz o papel de checar se uma certa substring é produzível pela gramática e retorna o conjunto de quem a produz e a união agrupa todas essas variáveis. A construção da matriz será a seguinte: Dada uma palavra $w = w_1 w_2 \cdot \ldots \cdot w_n$ de tamanho n, construa uma matriz $(n+1) \times (n+1)$ completamente vazia.

$$A = \begin{bmatrix} \emptyset & \cdots & \emptyset \\ \vdots & \ddots & \vdots \\ \emptyset & \cdots & \emptyset \end{bmatrix}$$

Trate os índices indo de 0 a n. Como a palavra possui n de tamanho, terá n substrings de tamanho 1. Para cada elemento $a_{i,i+1}$ preencha com $w_{i+1} \vdash \varepsilon$. Desse modo, todos as variáveis que geram substrings de tamanho 1 serão registradas na matriz A

$$A = \begin{bmatrix} \emptyset & w_1 \vdash \varepsilon & \cdots & \cdots & \emptyset \\ \vdots & \cdots & w_2 \vdash \varepsilon & \cdots & \vdots \\ \vdots & \cdots & \cdots & \ddots & \vdots \\ \emptyset & \cdots & \cdots & \emptyset & \emptyset \end{bmatrix}$$

Após isso, cada multiplicação matricial de potência k irá preencher a diagonal de $a_{i,i+k}$. Caso a variável inicial se encontre em $a_{0,n}$, a palavra é aceita pela linguagem.

Desse modo, A^k representa todas as variáveis que produzem substrings de tamanho k da string original. O fecho aparece novamente, somar essas potências entre 0 e k entrega todas as variáveis que produzem substrings de tamanho 0 até k da string original. Porém repare que, como a palavra tem tamanho n, A^{n+1} não entregará mais nenhuma informação. Deste modo, só se necessita somar as potências até n.

$$A^* = \bigcup_{i=0}^{\infty} A^i = \bigcup_{i=0}^{n} A^i$$

Porém esse semianel não é suficientemente preciso para o algoritmo. Um fato que o algoritmo exige é que todas as subdivisões tem de ser testadas, então para substrings de tamanho 3 o produto $A \vdash A^2$ e $A^2 \vdash A$ precisão ser feitos e somados, porque $\vdash$ não é comutativo. Para tirar essa ambiguidade da fórmula acima, considere "todas as combinações possíveis para n" por $A^{(n)}$ em constrate com A^n.

$$A^* = \bigcup_{i=0}^{\infty} A^{(i)} = \bigcup_{i=0}^{n} A^{(i)}$$

De modo que agora $A^{(3)} = A \vdash A^2 \cup A^2 \vdash A$. Genericamente

$$A^{(n)} = \bigcup_{i=0}^{n} A^{(i)} \vdash A^{(n-i)}$$

Finalmente

$$A^* = \bigcup_{i=0}^{n} \bigcup_{j=0}^{i} A^{(j)} \vdash A^{(j-i)}$$

Tome de exemplo a gramática da figura 6 e a string `muahaha`, sua forma normal de Chomsky, com S símbolo inicial, será

$$\begin{gathered}
S \to MB \\
S \to HE \\
S \to \varepsilon \\
B \to UC \\
C \to AD \\
D \to HE \\
E \to AD \\
E \to \texttt{a} \\
A \to \texttt{a} \\
H \to \texttt{h} \\
M \to \texttt{m}
\end{gathered}$$

Figura 9: Forma normal de Chomsky da gramática da figura 6

Alguns passos do parser são mostrados abaixo

$$A = \begin{bmatrix}
\emptyset & \{M\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \{U\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset
\end{bmatrix}$$

$$A + A^{(2)} = \begin{bmatrix} \emptyset & \{M\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{U\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \{S, D\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \{S, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

$$\bigcup_{i=0}^{4} A^{(i)} = \begin{bmatrix} \emptyset & \{M\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{U\} & \emptyset & \emptyset & \{B\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \{C, E\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \{S, D\} & \emptyset & \{S, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \{C, E\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \{S, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

$$\bigcup_{i=0}^{7} A^{(i)} = \begin{bmatrix} \emptyset & \{M\} & \emptyset & \emptyset & \emptyset & \{S\} & \emptyset & \{S\} \\ \emptyset & \emptyset & \{U\} & \emptyset & \emptyset & \{B\} & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \{C, E\} & \emptyset & \{C, E\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \{S, D\} & \emptyset & \{S, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} & \emptyset & \{C, E\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{H\} & \{S, D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{A, E\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{bmatrix}$$

Como o já foram feitas 7 iterações, tamanho da string `muahaha`, e a entrada $a_{0,8}$ contém a variável inicial S, `muahaha` é aceita pela gramática.

Para uma discussão mais aprofundada e um modo de redução deste problema para multiplicação de matrizes booleanas veja [3].

Conclusões

Certamente esses não foram todos os semianéis possíveis. Há muitos outros problemas que não foram discutidos, mas podem utilizar de multiplicação matricial com diferentes operações. Para uma leitura mais aprofundada no tema, certamente recomendo as fontes na parte de referências.

Referências

[1] Stephen Dolan. Fun with semirings: A functional pearl on the abuse of linear algebra. *SIGPLAN Not.*, 48(9):101–110, sep 2013.

[2] Stephen Dolan. Fun with semirings: A functional pearl on the abuse of linear algebra. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, page 101–110, New York, NY, USA, 2013. Association for Computing Machinery.

[3] Franz Christian Ebert. Cfg parsing and boolean matrix multiplication. 2007.

[4] Dick Grune and Ceriel J. H. Jacobs. *Parsing Techniques - A Practical Guide*. Monographs in Computer Science. Springer, 2008.

[5] "Bogumił Kamiński". "what is the shortest path to alphatensor?".

[6] Daniel J. Lehmann. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76, 1977.