CONJUNTOS
REPRESENTAÇÃO E OPERAÇÕES

RAUL H.C.LOPES

1. Introdução

1.1. **Conjuntos: operações.** Operações típicas com conjuntos:

- pertinência;
- união dos conjuntos de uma coleção;
- interseção dos conjuntos de uma coleção;
- diferença de conjuntos;
- construção de um novo conjunto a partir de um conjunto e um elemento dado;
- separação dos elementos de um conjunto em relação a uma propriedade dada;
- contagem de elementos de um conjunto;
- ordenação dos elementos de um conjunto;
- identificação de todos os elementos em um intervalo dado.

Questões que influenciam na decisão sobre a representação de conjuntos em um programa:

- Algoritmos que trabalham com conjuntos fixos de dados demandam o uso de estruturas que sejam eficientes em termos de:
  - tempo de construção das etsruturas de dados para armazenar os objetos em questão;
  - tempo necessário para consultar a estrutura em relaçãoa alguma propriedade, por exemplo, presença de um objeto dado;
  - espaço necessário para armazenar a estrutura.
- Algoritmos que lidam conjuntos de dados, cujos elementos variam durante uma execução demandam estruturas dinâmicas e tempos eficientes de atualização, inserção e retirada de objetos da estrutura.

Uma outra questão importante diz respeito ao número de dimensões dos objetos representados: a ordenação de coordenadas de pontos na linha tende a demandar algoritmos mais simples do que a ordenação de pontos no espaço com $d > 1$ dimensões.

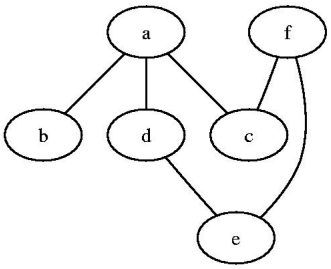

FIGURA 1. Um grafo não dirigido

Considerando ainda algoritmos com conjuntos estáticos de dados em uma dimensão, os algoritmos clássicos de ordenação, quicksort ou heapsort, por exemplo, associados a algoritmos como pesquisa binária fornecem um framework que eficiente tanto em termos de espaço linear, quanto em relação aos tempos de consulta, logarítmico no caso de pertinência.

1.2. **Grafos e árvores.** Conjuntos de dados dinâmicos e até mesmo conjuntos estáticos com dados representadoos em uma mais de uma dimensão geralmente demandam estruturas e algoritmos mais complexas: entre as opções disponíveis estão árvores e hashes.

Definição 1. *Um grafo* $G \stackrel{\triangle}{=}$ *um par* (V, E) *onde* V *é um conjunto de vértices e* $E \subseteq \{x, y : x, y \in V : \{x, y\}\}$*, chamado conjunto de pares. Dado um grafo* G*,* $V.G$ *denotará o conjunto de vértices e* $E.G$ *denotará o conjunto de arestas.*

A figura 1 mosta uma grafo onde

$$V = \{a, b, c, d, e, f\}$$
$$E = \{\{a, d\}, \{d, e\}, \{e, f\}, \{f, c\}, \{c, a\}\}$$

Grafos em que os elementos de E são pares ordenados, ou seja $E \subseteq VxV$, são chamados grafo dirigidos, veja um exemplo na figura 2

Definição 2. *Vértices* $v, u \in V.G$ *são adjacentes em* $G \stackrel{\triangle}{=} \{u, v\} \in E$*.*

Definição 3. *Grau de um vértice* $v \in V.G \stackrel{\triangle}{=}$*número de vértices a ele adjacentes em* G*.*

Definição 4. *Caminho em um grafo* $G \stackrel{\triangle}{=}$*seqüência de vértices tal que se* v_{i+1} *sucede* v_i *na seqüência, então* $\{v_i, v_{i+1}\} \in E.G$*.*

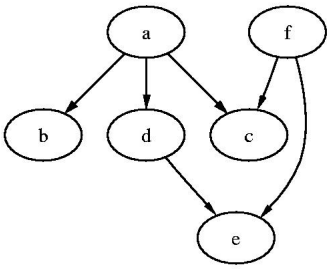

FIGURA 2. Grafo dirigido

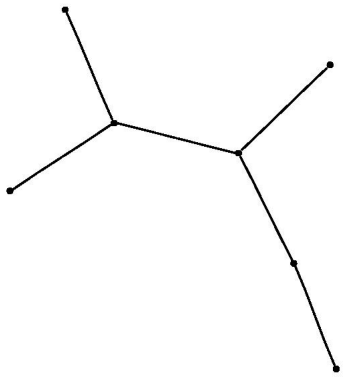

FIGURA 3. Uma árvore

Definição 5. *Árvore* $\stackrel{\triangle}{=}$*grafo em que existe exatamente um caminho conectando qualquer par de vértices.*

O grafo da figura 3 é uma árvore.

Definição 6. *Uma árvore com raiz é uma árvore em que um vértice distinto é designado como raiz.*

Usualmente os vértices de uma árvore com raiz são chamados de nós.

Definição 7. *Descendentes de um nóu em uma árvore* $G \stackrel{\triangle}{=}$*conjunto de nós* $x \in V.G$ *tal que* x *não está no caminho que liga a raiz da árvore a* u.

Filhos de um nó u *em uma árvore* $G \stackrel{\triangle}{=}$*conjunto de descentes de* u *que lhe são adjacentes.*

Folha ou nó externo $\stackrel{\triangle}{=}$*nó sem descendentes.*

Nó interno $\stackrel{\triangle}{=}$*nó que tem descendentes.*

Definição 8. *Grau (ou* branching factor*) de um nó em uma árvore com raiz* $\stackrel{\triangle}{=}$*número de filhos do nó.*

Ordem de uma árvore com raiz $\stackrel{\triangle}{=}$*grau máximo de qualquer de seus nós.*

Definição 9. *Profundidade (ou nível) de um nó* $u \stackrel{\triangle}{=}$ *número de arestas do caminho que liga a raiz ao nó* u*.*

Profundidade da árvore $\stackrel{\triangle}{=}$ *máximo das profundidades das folhas.*

Definição 10. *Altura de um nó* $u \stackrel{\triangle}{=}$ *profundidade da subárvore com raiz em* u*.*

Altura da árvore $\stackrel{\triangle}{=}$ *altura da raiz da árvore.*

Árvore tem um interesse especial em computação como estruturas eficientes para representar conjuntos de dados. Nesses casos, seus nós e, possivelmente, suas arestas podem ser rotulados com os dados armazenados.

A figura 4 representa uma árvore binária (ordem 2) de pesquisa de inteiros. Alguns dados sobre essa árvore:

- A raiz é o nó cujo rótulo é 10.
- Sua profundidade é três, a profundidade da folha rotulada com 22.
- Sua ordem é dois, cada nó tem no máximo dois descendentes.
- O nó rotulado com 25 é o único com grau um.
- Os rótulos dos nós estão ordenados da direita para a esquerda.

2. Indexação baseada em comparação

Árvores podem ser usadas para representar conjuntos de dados ou para representar índices que facilitam o acesso rápido a grandes conjuntos de dados. Quando um nó u de uma árvore é rotulado com um valor x diz-se que u contém (ou armazena) a chave x.

Esta seção apresenta árvores de pesquisa, cuja organização está baseada em comparações entre chave. Assume-se de agora em diantes que as chaves armazenadas nas árvores (rótulos dos nós) são retiradas de um algum conjunto A que suporta uma relação de ordem total $\prec$:

$$\forall x, y : x, y \in A : x \prec y \vee y \prec x \vee x = y$$

Definição 11. *Os elementos de uma árvore* G *sobre* A *são os elementos de* A *que ocorrem em rótulos de* G*.*

Definição 12. *Árvore de pesquisa de ordem* $m \stackrel{\triangle}{=}$ *árvore rotulada com elementos de uma conjunto ordenado* A *tal que:*

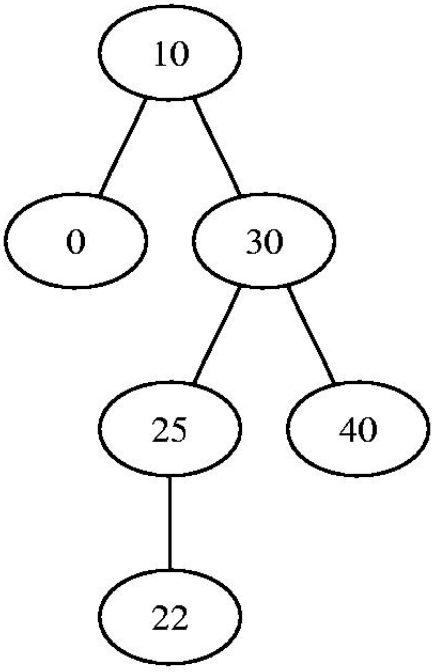

FIGURA 4. Árvore com raiz

- *cada nó tem no máximo m descendentes;*
- *um nó com $k+1$ descendentes contém (é rotulado com) uma seqüência ordenada de $k > 0$ elementos distintos de A;*
- *se $\langle s_1, s_2, \cdots, s_k \rangle$ rotula o nó t e $\langle p_0, p_1, \cdots, p_k \rangle$ é a seqüência de seus descendents, valem as propriedades:*
  - *todos os elementos de p_0 são menores do que s_1.*
  - $\forall i : 0 \leq i < k : \forall x \in p_i : s_i < x < s_{i+1}$
  - *todos os elementos de p_k são maiores do que s_k.*

A árvore da figura 4 é uma árvore binária de pesquisa: tem ordem m.

Uma representação comum de árvores de pesquisa é aquela em que os nós internos representam um índice e as folhas contêm o conjunto representado, possivelmente, repetindo itens que ocorem em nós internos. Essa são chamadas *leaf search trees*, árvores de pesquisa em folha. A figura 5 representa uma árvore de pesquisa em folha equivalente á árvore de pesquisa da figura 4.

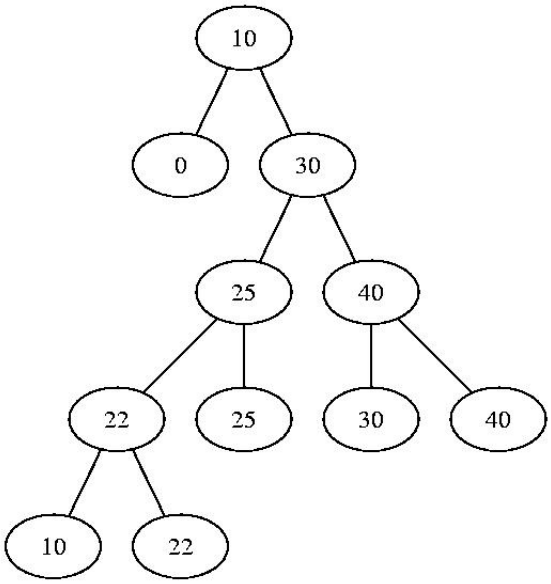

FIGURA 5. Árvore de pesquisa em folha

2.1. **Árvores balanceadas.** Overmars, em [8], apresenta o seguinte conceito de árvore balanceada.

Definição 13. *Uma árvore de pesquisa de ordem m está balanceada quando sua profundidade é menor ou igual do que $c \log n$, onde n é o número de itens na árvore e c é uma constante que depende apenas do tipo da árvore.*

A importância desse conceito está relacionada ao seguinte teorema que define o upper-bound para a pesquisa em uma árvore de pesquisa balanceada.

Exercício 1. *Prove que são necessárias no máximo $O(\log n)$ comparações de chaves para determinar se um dado item ocorre em uma árvore de pesquisa balanceada.*

Overmars propõe classificar árvores de acordo com os seguintes critérios para balanceamento:

- balanceamento de altura: a árvore é balanceada impondo-se restrições na altura de suas sub-árvores.

Definição 14. *Árvore AVL* $\stackrel{\triangle}{=}$ *árvore binária em que, para qualquer nó interno, a altura de sub-árvore esquerda difere da altura da sub-árvore da direita por no máximo um. (Veja* [6] *para algoritmos.)*

- balanceamento de peso: a árvore é mantida balanceada impondo a restrição de que para qualquer nó interno o número de folhas na sub-árvore da esquerda dividido pelo número total de folhas fica dentro de um intervalo fixo. ($BB[\alpha]$*-trees* são exemplo.)
- balanceamento de grau: todas as folhas da árvore têm a mesma profundidade e o balanceamento é obtido variando o grau dos nós internos. A árvore B, [6] e [3], e suas variantes são os exemplos clássicos.
- balanceamento de profundidade: o balanceamento é obtido impondo-se restrições sobre a diferença entre a maior e menor profundidade possível das folhas. αBB*-trees* sãoo exemplo ([7].)

Definição 15. *Árvore-2-3* $\stackrel{\triangle}{=}$ *árvore de pesquisa de ordem* 3 *em que todas as folhas estão no mesmo nível, ver* [1] *e* [6].

Estrutura criada por Hopcroft em 1971, é apresentada com pequenas variações dependendo do texto: veja, por exemplo [5] e [7].

A seguir são introduzidos construtores para os três tipos de nós da árvore-2-3.

Definição 16. *Árvore-2-3 sobre A (denotada* ***Tree.A*** $\stackrel{\triangle}{=}$ *árvore de pesquisa de ordem* 3 *cujos nós são constuídos de acordo com seguintes axiomas:*

(1) *se* $s \in A$ *então* $Leaf.a \in Tree.A$
(2) *se* $p_0, p_1 \in Tree.A$ *e* $s \in A$ *então*

$$Node2(p_0, s,_p 1) \in Tree.A$$

desde que:

- *s seja maior ou igual do que (*$\not\prec$ *na definição original de árvore de pesquisa) qualquer item em* p_0*;*
- *s seja menor do que (*$\prec$*) qualquer item em* p_1*.*

(3) *se* $p_0, p_1 p_2 \in Tree.A$ *e* $s_1, s_2 \in A$ *então*

$$Node3(p_0, s_1,_p 1, s_2, p_2) \in Tree.A$$

desde que:

- s_i *seja maior ou igual do que (*$\not\prec$ *na definição original de árvore de pesquisa) qualquer item em* p_{i-1}*, quando* $1 \leq i \leq 2$*;*

```
data Tree a =  Leaf a
             | Node2 (Tree a, a, Tree a)
             | Node3 (Tree a, a, Tree a, a, Tree a)
    deriving (Eq,Read,Show)
```

FIGURA 6. Inserção árvore-2-3

- s_2 *seja menor do que* ($\prec$) *qualquer item em* p_2.

Essa definição introduz uma árvore de pesquisa em folha: os nós internos servem apenas de índice para as folha.

Note que essa definição não pode ser considerada como rigorosa porque acaba introduzindo os construtores e a pertinência simultaneamente.

Exercício 2. *Apresente um definição axiomática rigorosa de árvore-2-3.*

Exercício 3. *Propriedades sobre a árvore podem ser provadas de forma indutiva, usando indução sobre a profundidade (ou altura) da árvore. Estabelça um princípio de indução para a árvore-2-3.*

Exercício 4. *Defina um operador de pertinência: dados* $x \in A$ *e* $t \in Tree.A$,

$$x \in t$$

determina se x *ocorre em* t.

Exercício 5. *Defina um predicado que é satistfeito por* $t \in Tree.A$ *quando* t *é uma árvore ordenada.*

Exercício 6. *Defina o predicado depthbal.t que é satisfeito quando* t *é uma* $Tree.A$ *e suas folhas estão todas no mesmo nível.*

A figura 6 apresenta um tipo de dados em Haskell para representar uma árvore-2-3. Note que esse tipo nao estabele as condições de balanceamento e ordenação. Por isso, é "heresia" afirmar que em Haskell, C, ou Java, por exemplo, existem tipos abstratos de dados. Essas linguagens apresentam apenas o suporte para de definição de representação e de algoritmos para manipulação de tipos encapsulados. Um dos fundamentos de tipos abstratos que é a definição lógica de operadores e a relação entre eles não pode ser construída nessas linguagens.

A figura 7 apresenta a inserção em nó interno de grau dois. Note que o construtor **Up** do tipo **NodeSplit** é usado para indicar quebra de nó.[1]

[1] Baixe código completo.

```
nodeinsert x (Node2 (p0,s,p1)) =
    if x<=s
    then case (nodeinsert x p0) of
           (Up (Node2 (p00,s0,p01))) ->
                        Ok (Node3 (p00,s0,p01,s,p1))
           (Ok np0) -> Ok (Node2 (np0,s,p1))
    else case (nodeinsert x p1) of
           (Up (Node2 (p10,s1,p11))) ->
                        Ok (Node3 (p0,s,p10,s1,p11))
           (Ok np1) -> Ok (Node2 (p0,s,np1))
```

FIGURA 7. Inserção árvore-2-3 (nó interno de grau 2)

Exercício 7. *Prove que o algoritmo de inserção em árvore-2-3 preserva a ordenação da árvore.*

Exercício 8. *Prove que o algoritmo de inserção em árvore-2-3 preserva a propriedade de que todas as folhas ficam no mesmo nível.*

Exercício 9. *Prove que a árvore* Tree.A *com profundidade k tem de 2^k a 3^k folhas.*

Exercício 10. *Prove que para determinar se um item ocorre na árvore definida são necessárias $Theta(\lg n)$ comparações.*

Exercício 11. *Prove que para inserir novo item em uma* Tree.A *são necessários $\Theta(N)$ acessos a nós.*

Exercício 12. *Construa um algoritmo de exclusão de itens para a árvore* Tree.A *e prove sua correção.*

2.1.1. *Árvore B.* Uma generalização óbvia da árvore-2-3 é a **árvore-B**.

Definição 17. *Árvore-B de ordem $m \stackrel{\triangle}{=}$ árvore de pesquisa de ordem m satisfazendo as seguintes restrições:*

- *todas as folhas estão no mesmo nível;*
- *o grau da raiz pode variar de 2 a m;*
- *o grau de outros nós internos varia de $m/2$ a m.*

A definição acima é apresentada em [6]. Note essa definição também varia de acordo com o autor: por exemplo, [9] apresenta como árvore-B de ordem aquilo que [6] chama de árvore-B de ordem $2m + 1$.

Variações de árvore-B encontradas na literatura objetivam principalmente reduzir o número de acessos a disco, assumindo que a árvore

esteja armazenada em disco. Entre as estratégias estão: aumentar a ordem da árvore, o que reduz sua profundidade e tentar obter ocupação maior por nó. As seguintes variantes de árvore-B são frequentemente citadas:

- árvore B^+, em [6], chamadas árvore B^* em [2]: árvore-B de pesquisa em folha em que as folhas são parte de uma lista encadeada, facilitando a listagem sequencial dos itens da árvore;
- árvore B^* em [6]: o grau da raiz varia de 2 a $2\lfloor(2m-2)/3\rfloor+1$ e outros nós internos têm grau maior ou igual $(2m-1)/3$.

2.1.2. *Árvore B e a indexação de strings.* Várias alternativas têm sido identificadas para aplicar árvores B à indexação de strings. A indexação de strings traz a dificuldade de que chaves tendem a ter tamanhos diferentes, o que não combina com a idéia de nós de tamanho fixo e número máximo de chaves também fixo. As alternativas são:

- Armazenar nos nós referências para as chaves. Isso, no mínimo duplica o número de acessos a disco para qualquer operação.
- Fixar o tamanho em bytes do nó permitindo que varie o tamanho das chaves. Neste caso, é importante ter um caracter distinto para ser separador dos strings que são as chaves: veja [6]. Esta alternativa ainda pode ser inviável quando os strings têm longas repetiçõesde caracteres.

A seguir considere que o objetivo consiste em indexar os sufixos de

aabaadac

As figuras a seguir mostram uma *leaf search* árvore-B de ordem 3. Note que:

- as folhas contêm todos os sufixos ordenados da esquerda para a direita;
- embora sejam mostrados partes dos sufixos, a respresentação real deveria conter em todos os nós referências para os respectivos sufixos.

As figuras em apresentam os estágios sucessivos da árvore. A figura 8 após a inserção dos 3 primeiros sufixos: note que o sufixo 1, iniciado com *ab*, ocorre em nó interno, como índice e e em nó externo, como informação. Além disso, só a parte inicial de cada sufixo é exibida. Em uma implementação, provavelmente uma referêcia para o sufixo **baadac** apareceria no lugar de **ba**.

A figura 9 mostra a árvore com os primeiros 4 sufixos.

As figuras 10, 11 e 12 mostram as árvores, respectivamente com 5, 6 e 8 sufixos.

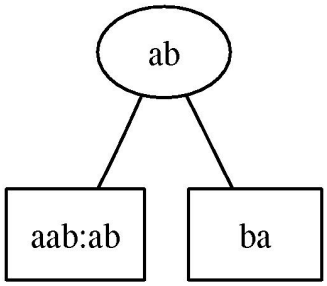

FIGURA 8. A árvore-B com três sufixos

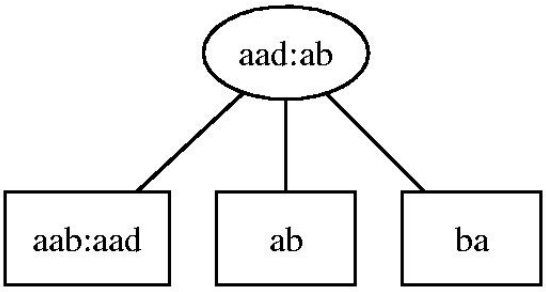

FIGURA 9. A árvore-B com quatro sufixos

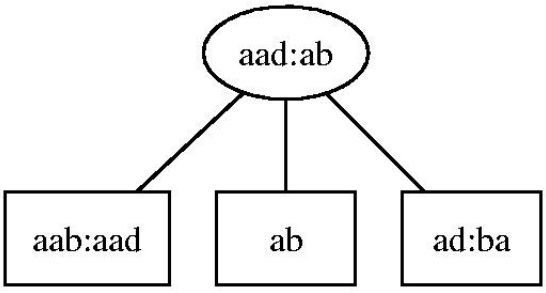

FIGURA 10. A árvore-B com cinco sufixos

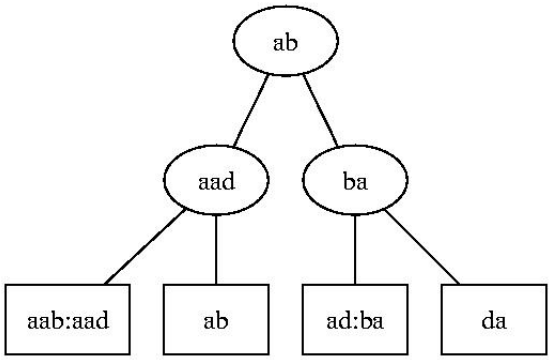

FIGURA 11. A árvore-B com seis sufixos

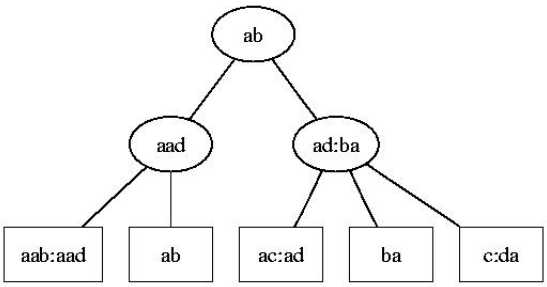

FIGURA 12. A árvore-B com oito sufixos

Ferragina e Grossi, em [4], propõem uma representação alternativa onde os nó internos contêm pares que definem os limites inferior e superior dos descendentes respectivos. A figura 13 mostra a árvore com os 8 sufixos. Eles denominam essa estrutura *String B-Tree* simples. É interessante que essa idéia já aparece em estrutura de indexação multi-dimensional como *k-d-b-tree* e *R-tree*. A versão completa da *String B-tree* representa os nós internos em uma árvore digital.

3. RADIX BASED TREE

Esta seção da indexação de strings via árvores *trie* e suas variantes. A característica fundamental dessa organização de dados é de que a indexação baseia-se num processo de distribuição dos strings por buckets

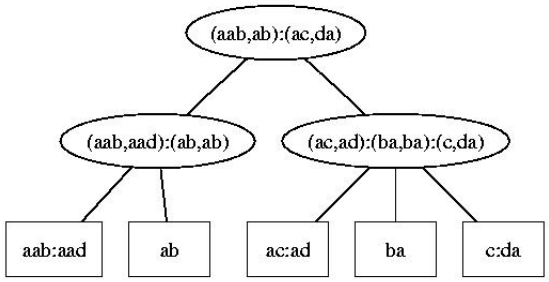

FIGURA 13. A *String B-tree* simples

de forma análoga aos algoritmos de ordenação por distribuição. Todos os exemplos desta seção apresentam indexação dos sufixos de um texto dados. Nas figuras, os rótulos das folhas definem o sufixo ali encontrado: por exemplo, por uma folha com rótulo t_4 indicam que o sufixo obtido pela exclusão dos quatro caracteres iniciais do texto está presente na referida folha.

3.1. **Trie.** A árvore *trie* foi a provavelmente a primeira estrutura proposta para indexar texto e certamente a primeira baseada em distribuição do texto baseada no alfabeto: ver [6] para história e algoritmos para trie e pesquisa digital em geral.

Os nós externos de umna trie são strings ou referências para strings. Cada nó interno é mapeamento de um alfabeto, o alfabeto usado para construir o texto, para nós internos ou externos. A figura 14 apresenta uma *trie* com todos os sufixos de **baaaaz**. Note que, embora apresentada em [6] como estrutura para indexação de strings, ela pode ser triviamente aplicada á indexação de sufixos. Para tal, basta assegura que nenhum sufixo de um texto é prefixo de outro sufixo.

Exercício 13. *Prove que se o caracter final de um texto é distinto de todos os outros, então nenhum sufixo do texto é prefixo de outro sufixo do texto.*

A trie é uma estrura extremamente eficiente em termos de pesquisa: $O(n)$ passos são necessários para determinar se um string de tamanho n existe em uma trie. O algoritmo de pesquisa é trivial: basta manter a qualquer momento uma referência p_s para o string e uma referência p_t para a trie. A invariante estabelecerá que o caminho percorrido da raiz até antes do nó p_t é igual ao prefixo que precede a p_s para o string. Assumindo que ao início de uma transição p_s designe o caracter x, após

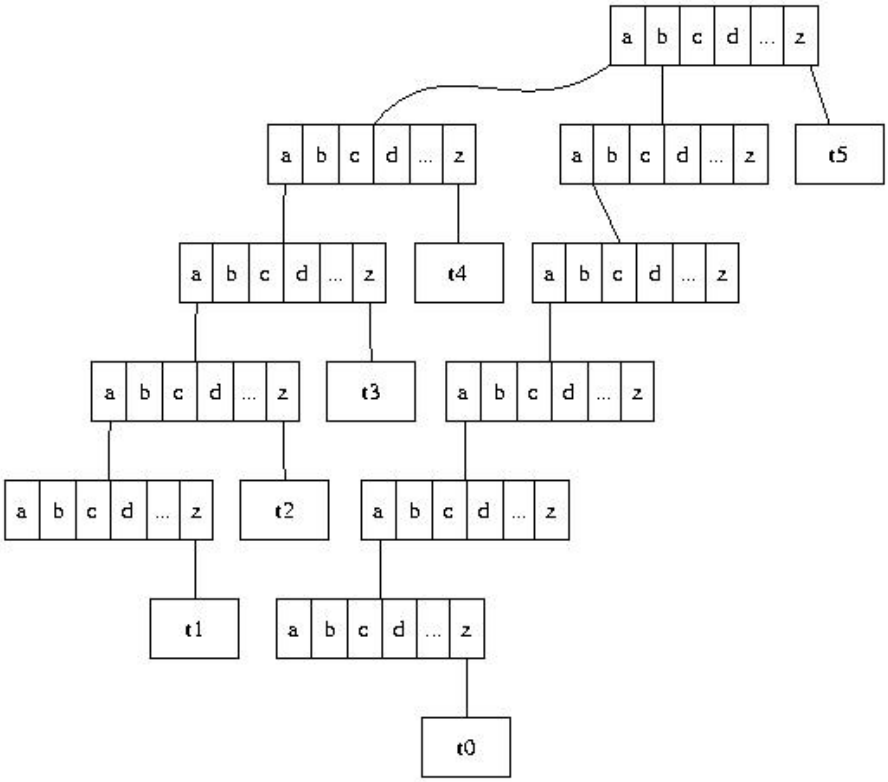

FIGURA 14. Trie para sufixos de **baaaaz**

a mesma p_s terá avançado uma posição e p_t referenciará $p_t(x)$. Quando $p_t(x)$ designa um string basta determinar se esse string é igual ao string procurado.

Note que:

- as folhas da trie fornecem a ordenação dos strings indexados.
- é trivial realizar uma pesquisa parcial em que se procura um prefixo de uma palavra: afinal, o prefixo de uma palavra ocorre no texto indexado se ocorrer como prefixo de um de seus sufixos.

Embora eficiente, em termos de pesquisa a trie tende a ser ineficiente em termos de espaço. Na figura 14, por exemplo, os nós internos apresentam um aproveitamento de no máximo 20%. Uma solução trivial para resolver esse problema consiste em representar cada nó como lista

linear: cada nó interno da trie agora pode ser representado por um nó interno de árvore binária, como na figura 15. A raiz dessa árvore, por exemplo, que para encontrar strings iniciados com o caracter a, deve-se se seguir a referência que leva para o nó abaixo da raiz. Para encontrar strings iniciados com outros caracteres, tenta-se o mesmo procedimento, a partir da sub-árvore á direita da raiz atual.

Exercício 14. *Apresente uma definição formal da* trie *com compactação de nó interno.*

Apesar de mais eficiente em termos de uso de espaço essa representação ainda apresenta nós internos redundantes: os cinco nós internos abaixo do nó com rótulo b são redundantes na medida em que são usados como ponto de distinção entre strings diferentes. A trie da figura 16 apresenta a característica de que qualquer nó interno tem, ao menos, dois descendentes.

Exercício 15. *Prove que determinar se um string de tamanho n ocorre em uma* trie *compactada demanda $O(Km)$, onde K é o número de caracters do alfabeto.*

A figura 17 apresenta uma trie compactada para os sufixos de **baaaabaac**.

Exercício 16. *Como seria possível representar uma trie compactada em usando nós de árvore binária.*

3.2. **PATRICIA.** Morrisson apresentou em 1968, veja em [6], a idéia de indexar os sufixos de um texto. Para tal, ele propôs uma estrutura chamada árvore PATRICIA que é uma trie compactada com as seguintes restrições:

- o texto é visto como um string de bits;
- todo nó interno tem exatamente dois descendentes;
- um nó interno apresenta uma inteiro b e particiona as folhas abaixo dele em dois grupos: um grupo, cujos strings têm bit b igual a zero, e o grupo de strings com bit b igual a 1.

As figuras 18 a 21 apresentam os estágios intermediários da construção da árvore PATRICIA para o texto **baaad**.

Referências

1. Alfred V. Aho, Jonh E. Holpcroft, and John D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.
2. R. Bayer and K. Unterauer, *Prefix B-trees*, ACM Transaction on Databse Systems **2** (1977), 11–26.
3. D. Comer, *The ubiquitous B-Tree*, ACM Computing Surveys **11** (1979), 121–137.

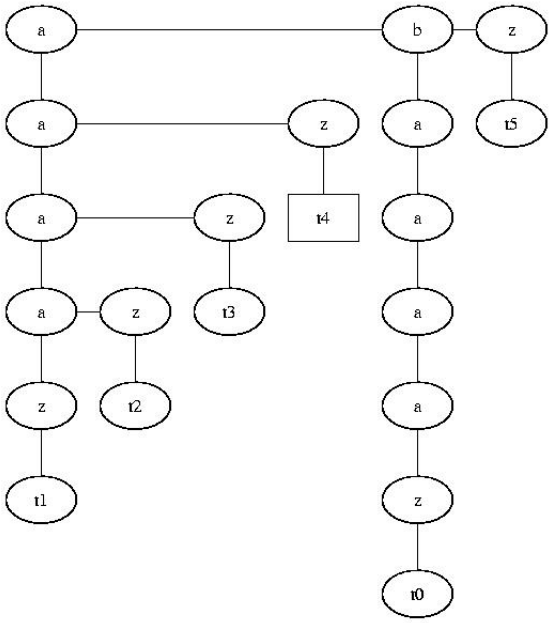

FIGURA 15. Trie com compactação de nível

4. P. Ferragina and R. Grossi, *The string B-tree: A new data structure for string search in external memory and its applications*, Journal of ACM **46** (1999), 236–280.
5. Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, and Clifford Stein, *Introduction to algorithms*, MIT Press, 2001.
6. Donald E. Knuth, *The art of computer programmimg, volume 3: Sorting and searching*, Addison-Wesley, 1998.
7. Mark H. Overmars, *The design and analysis of dynamic data structures*, Lecture Notes in Computer Science, Springer-Verlag, 1983.
8. _____, *The design of dynamic data structures*, Springer-Verlag, 1983.
9. Niklaus Wirh, *Algorithms and data structures*, ?, 1990.

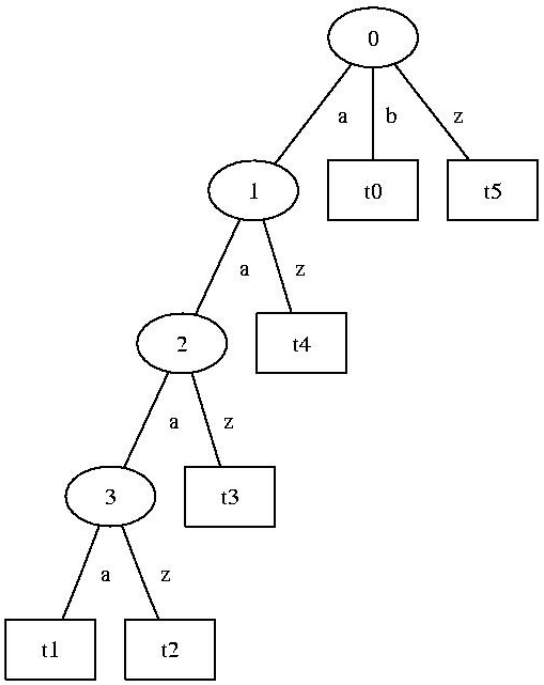

FIGURA 16. Trie compactada para sufixos de **baaaaz**

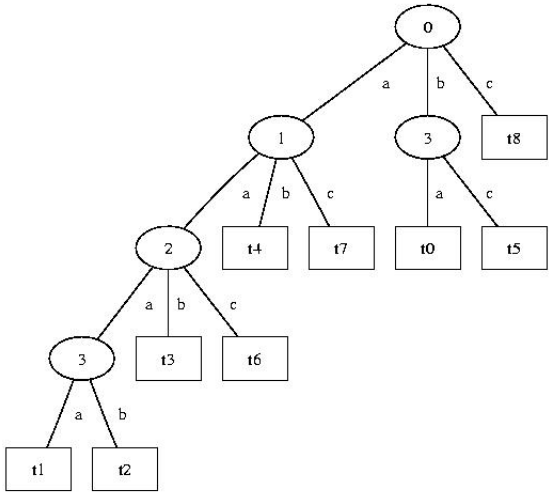

FIGURA 17. Trie compactada para sufixos de **baaaabaac**

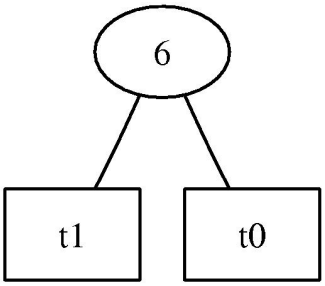

FIGURA 18. PATRICIA para dois sufixos de **baaad**

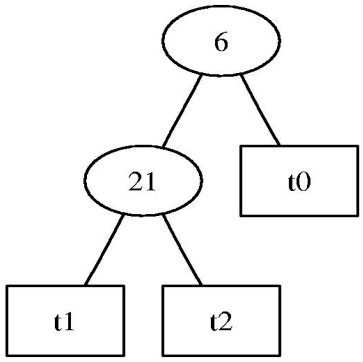

FIGURA 19. PATRICIA para três sufixos de **baaad**

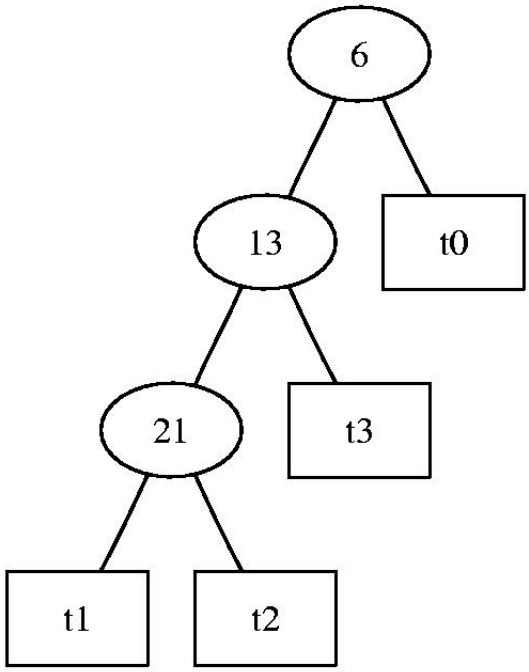

FIGURA 20. PATRICIA para quatro sufixos de **baaad**

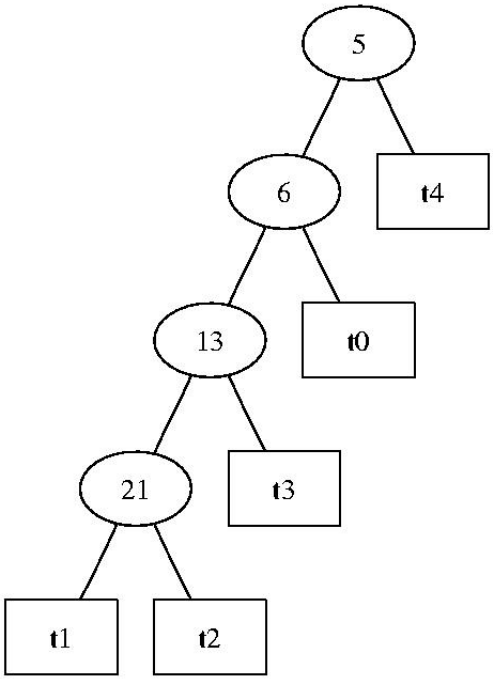

FIGURA 21. PATRICIA para os sufixos de **baaaaz**