Um estudo sobre a otimalidade dinâmica de *BSTs*

George Edson Albuquerque Pinto
Victor Almeida Campos
Universidade Federal do Ceará, Fortaleza-CE, Brasil.
{georgeapinto, campos}@lia.ufc.br

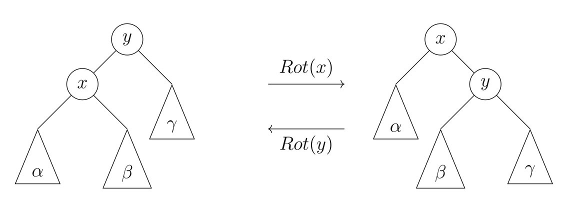
Parallelism, Graphs and Optimization
ParGO Research Group

1. Introdução

As *árvores binária de busca* ou, do inglês, *Binary Search Tree* - *BST* são uma estrutura de dados clássica. Uma *BST* armazena um conjunto de chaves. Além disso, suporta operações como: busca, inserção e remoção de chaves. Apesar de décadas de pesquisa, uma questão fundamental sobre as *BSTs* permanece sem solução: qual é a “melhor estrutura *BST*”? Esse problema ainda não é resolvido, mesmo para o caso em que não é permitido inserções e remoções (neste trabalho vamos focar nesse problema).

Dada uma *BST* T com as chaves $\{1,\dots,n\}$, uma busca por uma chave s em T é realizada com um ponteiro percorrendo T iniciando na raiz e podendo mover-se para o nó filho esquerdo/direito ou para o nó pai, e realizar rotações entre o nó apontado pelo ponteiro e seu nó pai – temos na figura 2 um exemplo de rotação; no entanto, o ponteiro deve, em algum momento da busca, tocar s. O *custo* de uma busca é a quantidade total de nós distintos tocados. Para uma sequência de buscas $S=\langle s_1,\dots,s_m\rangle$ em uma *BST* T_0, a busca por s_i é realizada em T_{i-1} retornando T_i. Culik e Wood mostraram que a quantidade de rotações na transformação de T_{i-1} em T_i na busca por s_i pode ser feita com até duas vezes a quantidade de nós tocados [1]. Denotamos por $OPT(S)$ o menor custo para buscar S em alguma árvore T_0.

Figura 2: Rotação em uma *BST*



Um algoritmo *BST* é *offline* se a sequência de buscas é conhecida a princípio e é *online* se as buscas são conhecidas incrementalmente. Um algoritmo *BST online* é $f(n)$-competitivo se seu custo sobre uma sequência S é limitado por uma função $f(n)OPT(S)$ para todo S. Dizemos que um algoritmo *BST* $\mathcal{A}$ é *dinamicamente ótimo* se $\mathcal{A}$ é $\mathcal{O}(1)$-competitivo.

Para uma sequência de m buscas em uma *BST* balanceada estática, um algoritmo *BST* é $\mathcal{O}(\ln n)$-competitivo. Para o problema em que a árvore é estática, existe um algoritmo de programação dinâmica capaz de encontrar uma árvore ótima em tempo $\mathcal{O}(n^2)$ [2].

Sleator e Tarjan apresentaram a *árvore Splay* [3] e conjecturaram que essa árvore é dinamicamente ótima. No entanto, ainda não foi provado que árvores *Splay*, ou qualquer outra *BST*, é dinamicamente ótima. Alguns limitantes para $OPT(S)$ foram propostos. Lucas propôs um algoritmo *BST offline* guloso para encontrar um limite superior, e conjeturou que seu algoritmo fornece uma aproximação de fator constante para $OPT(S)$ [4]. Demaine et al. também propuseram um algoritmo para o limite superior [5]. Para o limite inferior Wilber propôs dois limites [6], e mais recentemente Demaine et al. apresentaram a árvore Tango, que é $\mathcal{O}(\ln\ln n)$-competitiva [7], e Wang, Derryberry e Sleator propuseram a árvore *multisplay* [8], que também é $\mathcal{O}(\ln\ln n)$-competitiva e é uma combinação das árvores *Splay* e Tango.

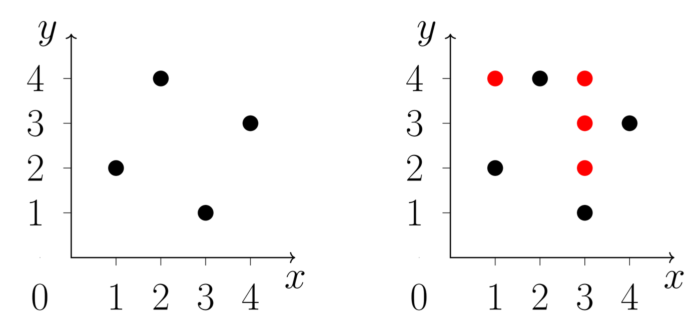
2. Definições

Um *ponto* p se refere a um ponto em 2D com coordenadas inteiras (p_x,p_y) tal que $1\le p_x\le n$ e $1\le p_y\le m$. Usamos $\square ab$ para denotar o *retângulo* alinhado aos eixos definido por dois pontos a e b não alinhados horizontalmente/verticalmente, isto é, $\square ab=\{(x,y)\in\mathbb{Z}\times\mathbb{Z}\mid \min\{x_a,x_b\}\le x\le\max\{x_a,x_b\}\ e\ \min\{y_a,y_b\}\le y\le\max\{y_a,y_b\}\}$. Dizemos que $\square ab$ é um $\boxslash$*-retângulo* se a inclinação da linha $\overline{ab}$ é positiva ou $\boxbslash$*-retângulo* se a inclinação da linha $\overline{ab}$ é negativa.

Um par de pontos $(a,b)\in P$ é *arboreamente satisfeito* em P se, ou a e b são alinhados horizontalmente/verticalmente, ou existe pelo menos um ponto de $P\backslash\{a,b\}$ em $\square ab$. Um conjunto de pontos P é arboreamente satisfeito se todos os pares de pontos em P são arboreamente satisfeitos. A *visão geométrica de uma execução BST* E é o conjunto de pontos $P(E)=\{(x,y)\mid x\in\tau_y\}$, onde τ_y são os nós tocados na busca por s_i.

Seja a *visão geométrica de uma sequência de buscas* o conjunto de pontos $P(S)=\{(s_1,1),\dots,(s_m,m)\}$. Na figura 4(a) temos um exemplo da visão geométrica da sequência de buscas $S=\{3,1,4,2\}$. Na figura 4(b) temos um conjunto de pontos P' arboreamente satisfeito tal que $P(S)\subseteq P'$.

Figura 4: visão geométrica de uma sequência de buscas e conjunto arboreamente satisfeito



(a) (b)

Lema 2.1 ([5]). *O conjunto de pontos $P(E)$ para qualquer execução* BST E *é arboreamente satisfeito.*

Lema 2.2 ([5]). *Para qualquer conjunto de pontos arboreamente satisfeito P, existe uma execução* BST E *com $P(E)=P$.*

Os lemas 2.1 e 2.2 mostraram que uma execução E de uma sequência de buscas S é equivalente ao conjunto arboreamente satisfeito P tal que $P=P(E)$ e $P(S)\subseteq P(E)$.

Denotamos por $minASS(S)$ o tamanho no menor superconjunto arboreamente satisfeito de $P(S)$. Assim, temos que $OPT(S)=minASS(S)$.

O problema do superconjunto arboreamente satisfeito *online* (ASS *online*) é projetar um algoritmo que receba um conjunto de pontos $(s_1,1),\dots,(s_m,m)$ nesta ordem. Depois de receber (s_i,i), o algoritmo gera um conjunto de pontos P_i tal que $\{(s_1,1),\dots,(s_i,i)\}\cup P_1\cup\ldots\cup P_i$ seja arboreamente satisfeito.

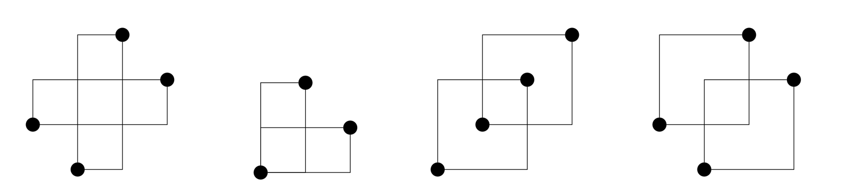
Lema 2.3 ([5]). *Para qualquer algoritmo ASS* online A, *existe um algoritmo* BST *online* A' *tal que, em qualquer sequência de buscas, o custo de A' é limitado por uma constante multiplicativa do custo de A.*

3. Limitantes

3.1 Limites inferiores

Dois retângulos $\square ab$ e $\square cd$ são *independentes* $(a,b,c,d\in P)$ em P se os retângulos não são arboreamente satisfeito e nenhum canto de qualquer dos retângulos estiver estritamente dentro do outro.

Figura 6: Retângulos independentes e dependentes



(a) (b) (c) (d)

Temos exemplos de retângulos independentes nas figuras 6(a) e 6(b), e de retângulos dependentes em 6(c) e 6(d). Denotamos a quantidade máxima retângulos independente que podem ser formado no conjunto de pontos P por $maxIRB(P)$.

Wilber propôs dois limites inferiores para $OPT(S)$ em uma *BST* T [6]. O primeiro limite depende de uma árvore binária P com as mesmas chaves de T, chamada de *árvore de limite inferior*. Esta árvore tem sua estrutura fixa. O segundo limite proposto por Wilber não depende da árvore de limite inferior nem de T. Baseado nesses algoritmos, Demaine et al. apresentaram uma construção de um conjunto de retângulos independentes para cada limite de Wilber [5].

Um conjunto de pontos P é $\boxslash$*-satisfeito* se todo par de pontos $(a,b)\in P$ que formam um $\boxslash$-retângulo $\square ab$ é arboreamente satisfeito. Denotamos como $minASS_{\boxslash}(P)$ o tamanho do menor superconjunto $\boxslash$-satisfeito de P. Demaine et al. propuseram o algoritmo guloso *SIGNEDGREEDY* [5] para calcular $minASS_{\boxslash}(P)$, onde o conjunto de pontos P é varrido incrementalmente na coordenada y, e na linha i, adiciona a quantidade mínima de pontos para P ser $\boxslash$-satisfeito. Denotamos por $add_{\boxslash}(P)$ o conjunto de pontos adicionados pelo algoritmo. Seja $minASS_{\boxtimes}$ o tamanho do menor conjunto $\boxtimes$-satisfeito Z com relação a X.

Lema 3.1 ([5]). *Para qualquer conjunto de pontos P, existe um conjunto de $\boxslash$-retângulos independentes $IRB_{\boxslash}(P)$, onde $|IRB_{\boxslash}(P)|=|add_{\boxslash}(P)|$.*

Lema 3.2 ([5]). *Para qualquer conjunto de pontos P, $minASS_{\boxslash}(P)=|add_{\boxslash}(P)|+|P|$.*

Teorema 3.3 ([5]). *Se P contém um conjunto de retângulos independentes I, então $minASS_{\boxtimes}(P)\ge\frac{|I|}{2}+|P|$.*

3.2 Limites superiores

Lucas apresentou o algoritmo *BST offline* guloso *GREEDY-FUTURE* [4] que segue dois princípio: (1) tocar apenas os nós no caminho de busca (caminho entre a raiz e a chave buscada), e (2) reorganizar o caminho de busca, movendo a próxima chave a ser buscada o mais próximo da raiz possível. O principal aspecto “guloso” do *GREEDY-FUTURE* é tocar apenas as chaves do caminho de busca. Lucas conjeturou que seu algoritmo fornece uma aproximação de fator constante para $OPT(S)$. E alguns anos depois Munro conjecturou que o *GREEDY-FUTURE* tem custo $OPT(S)+\mathcal{O}(m)$ [9].

Demaine et al. apresentaram o algoritmo guloso *GREEDYASS* [5] para o problema do superconjunto arboreamente satisfeito (ASS), onde uma linha horizontal varre o conjunto de pontos incrementalmente na coordenada y. Na linha i, adiciona a quantidade mínima de pontos para tornar o conjunto de pontos, onde $y\le i$, arboreamente satisfeito. Esse algoritmo é *online*, porque suas decisões dependem apenas do passado. Na figura 4(b) temos a saída do *GREEDYASS* para as buscas $\{3,1,4,2\}$. Pelo lema 2.3, o *GREEDYASS* pode ser transformado em um algoritmo *BST online* com custo assintoticamente igual.

Baseado no primeiro limite de Wilber, Demaine et al. propuseram a *árvore Tango* [7], que é uma *BST* $\mathcal{O}(\ln\ln n)$-competitiva. Para alcançar este desempenho, uma árvore de limite inferior P é dividida em árvores auxiliares menores balanceadas de tamanho no máximo $\log n$. Posteriormente Wang, Derryberry e Sleator apresentaram a árvore *multisplay* [8], que também é $\mathcal{O}(\ln\ln n)$-competitiva e é uma combinação das árvores *Splay* e Tango.

4. Plano de trabalho

- Revisar a bibliográfica do estado da arte sobre otimalidade dinâmica;
- Redigir o texto de dissertação e apresentar até março de 2020 contendo *survey*

Referências

[1] K. Culik and D. Wood, “A note on some tree similarity measures,” *Inf. Process. Lett.*, vol. 15, no. 1, pp. 39–42, 1982.

[2] D. E. Knuth, “Optimum binary search trees,” *Acta Inf.*, vol. 1, no. 1, pp. 14–25, 1971.

[3] D. D. Sleator and R. E. Tarjan, “Self-adjusting binary search trees,” *J. ACM*, vol. 32, no. 3, pp. 652–686, 1985.

[4] J. M. Lucas, “Canonical forms for competitive binary search tree algorithms,” *Rutgers University, Department of Computer Science, Laboratory for Computer Science Research*, 1988.

[5] E. D. Demaine, D. Harmon, J. Iacono, D. M. Kane, and M. Pătraşcu, “The geometry of binary search trees,” in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, 2009, pp. 496–505.

[6] R. E. Wilber, “Lower bounds for accessing binary search trees with rotations,” *SIAM J. Comput.*, vol. 18, no. 1, pp. 56–67, 1989.

[7] E. D. Demaine, D. Harmon, J. Iacono, and M. Pătraşcu, “Dynamic optimality - almost,” in *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, 2004, pp. 484–490.

[8] C. C. Wang, J. Derryberry, and D. D. Sleator, “O(log log n)-competitive dynamic binary search trees,” in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, ser. SODA ’06. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006, pp. 374–383.

[9] J. I. Munro, “On the competitiveness of linear search,” in *Proceedings of the 8th Annual European Symposium on Algorithms*, ser. ESA ’00. London, UK, UK: Springer-Verlag, 2000, pp. 338–345.