Tries

Jonatan Ahumada Fernández

28 de marzo de 2019

Objetivos de la presentación

- ¿Qué son los tries?
- ¿Qué problemas resuelven?
- ¿Cómo se podrían representar sus nodos?
- Algunos conceptos transversales

Patricinio

- Eric Demaine del MIT
- Erik Demaine. 6.851 Advanced Data Structures. Spring 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, `https://ocw.mit.edu`. License: Creative Commons BY-NC-SA.

¿Quién creó los tries?

- Edward Fredkin
- "digital philosophy"
- "Trie Memory" 1960. –> no disponible :(
- reTRIEval

String matching

text T x String P ——strings over Σ

- ocurrencias de un patrón en un texto
- contarlos, imprimirlos, su posición, etc.

Tipos de algoritmo

- Naive
- Rabin Karp
- Knuth Norris Pratt
- etc.

Data Structure version

- "Static DS"
- Nos dan T, "preprocesamos T"
- Query P
- ::Objetivo: O(P), O(T) space:::

Problema del predecessor

- T_1, T_2,..T_k
- Responde a la pregunta dónde iría mi nueva String P dentro de $\{T_n\}$
- "binary search will not work so well if the strings are very long"
- Por eso tenemos los Tries!

En términos más simples

- insert
- delete
- predecessor
- sucessor

Derinición

- Un "rooted tree" con ramas nombradas con una letra del alphabeto
- "root to leef paths"
- caracter de terminación: "add new letter $ to end of every string"

Qué nos gusta?

- que tiene precedesor
- usos: títulos de libros, etc

Ejemplo

- {ana, ann, anna, anne}
- no tine raiz
- el segundo nivel es todo el alphabeto (en este caso: a, e, n)

Problemas

Posibles representaciones de los nodos

- array
- BST
- HashTable
- Van Emde Boas **
- Weight Balanced BST's
- Leaf Trimming **

array

- serian tamaño Σ
- query rápido
- query: O(P)
- Space: O(T Σ)
- el espacio puede ser un problema

// T= # nodes in a trie

BST

- en vez de tener nodos para cada palabra del alphabeto
- se hace binario
- el tamaño de este arbol sería height: log Σ
- Query(P log\ Sigma)
- space O(T)

hashTable

- ¡no puedes resolver el problema del predecesor!
- Bono: ¿por qué?

Trays (lewendowsky 2006)

- portmanteau: tries + arrays
- sí predecesoor y sucesor
- Query: O(p+log Σ)
- Space: O(T)
- VER Weight-Balanced BST!!

Weight-Balaced BST (Martin Farach Colton)

- weight: # descendant leaves
- cada dos bordes: a) avancas una letra en P o b) reducimos el número de canditados a 2/3
- can only cut in child boundaries
- Query: O(P + log K)

Referencias

- R. Boyer and J. Moore. A fast string searching algorithm. Communications of the ACM,

20(10):762772, 1977.

- R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. IBM Journal

of Research and Development, 31:249260, March 1987.

- D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. SIAM Journal of

Computing, 6(2):323350, 1977.