Miguel Pedregosa Pérez, Enero 2019

README

NOTA: Este archivo NO es la documentación del proyecto, es la introducción imprescindible a los planteamientos de los diferentes módulos.

Esta implementación del juego de las Letras ha sido realizada de forma libre, sin seguir el guión propuesto, aunque inspirándose ligeramente en él.
Se han diseñado dos programas independientes:

- Estandarizar Diccionario: sirve para generar diccionarios en formato .txt (únicamente con caracteres ASCII de la 'a' a la 'z' minúsculas) a partir de archivos que contienen caracteres especiales del español: á, é, í, ó, ú, ü y ñ. Dado que el diccionario de la RAE se puede encontrar fácilmente en archivos separados por letras y con irregularidades como caracteres numéricos o no alfanuméricos como ',' o '-', se puede obtener un archivo estandarizado con el diccionario completo de la RAE pasándole al programa un archivo fruto de la concatenación de todos estos.

- Letras: programa correspondiente al propio juego. La forma correcta de ejecutarlo será:

    -$ ./bin/letras <diccionario> <n_letras> <modalidad>

  Se recomienda utilizar el archivo ./data/rae_std.txt como diccionario.

MÓDULOS

El resumen de los módulos implementados se puede ver en la propia documentación del proyecto en ./doc, aunque hay algunos aspectos que quiero resaltar.

1- Implementación del T.D.A. Diccionario

La idea principal del almacenamiento del diccionario en memoria es la creación de un objeto parecido a una matriz donde, para buscar una palabra con los caracteres c1, c2, c3… cn, simplemente haya que acceder a

    diccionario[c1][c2][c3]...[cn]

Esto es posible gracias a que el tipo **char** es un tipo numérico, y que en ASCII los caracteres del alfabeto latino estándar están ordenados, solamente hará falta un ajuste del tipo: el índice del carácter 'x' estará en la posición 'x'-'a'.
Con esta aproximación el acceso a la posición de memoria correspondiente a una palabra es tremendamente rápido, ya que es un algoritmo recursivo de aritmética de punteros O(n), donde n es en número de letras de la palabra, lo cual no es un problema ya que:
1- Cada paso del algoritmo es extremadamente rápido
2- n se mantiene bastante pequeño

3- La eficiencia en el caso medio del algoritmo de búsqueda es mucho mejor a cualquier algoritmo que funcione por comparación, por motivos que se explicarán más adelante.

Este algoritmo se podría ver como un conjunto de tablas hash en forma de árbol de profundidad n (ver esquema adjunto).

Para disminuir el enorme espacio en memoria que esto ocuparía, se opta por una reserva dinámica de espacio: La posición diccionario[x1]...[xn] estará reservada únicamente si existen palabras que comienzan por x1...xn (asumiendo que toda palabra comienza por sí misma). Así, podemos dar un ejemplo de por qué la eficiencia de la búsqueda en este TDA en el caso medio mejora bastante, con una mejora inversamente proporcional al tamaño del diccionario.

Sea una palabra un conjunto ordenado de caracteres
Sea P una palabra
Sea D un objeto del TDA Diccionario
Sabemos que P no ha sido registrada en D
Sabemos que P={‘a’,’r’,’b’,’o’,’l’}
Sabemos que D solo tiene una palabra registrada: {‘a’,’b’,a’,’d’}
Procedemos a buscar P en D
Comenzamos por la primera iteración:
D[‘a’] está reservada, proseguimos
D[‘a’][‘r’] no está reservada, paramos
Con solo dos iteraciones podemos saber que la palabra P no ha sido registrada en D.

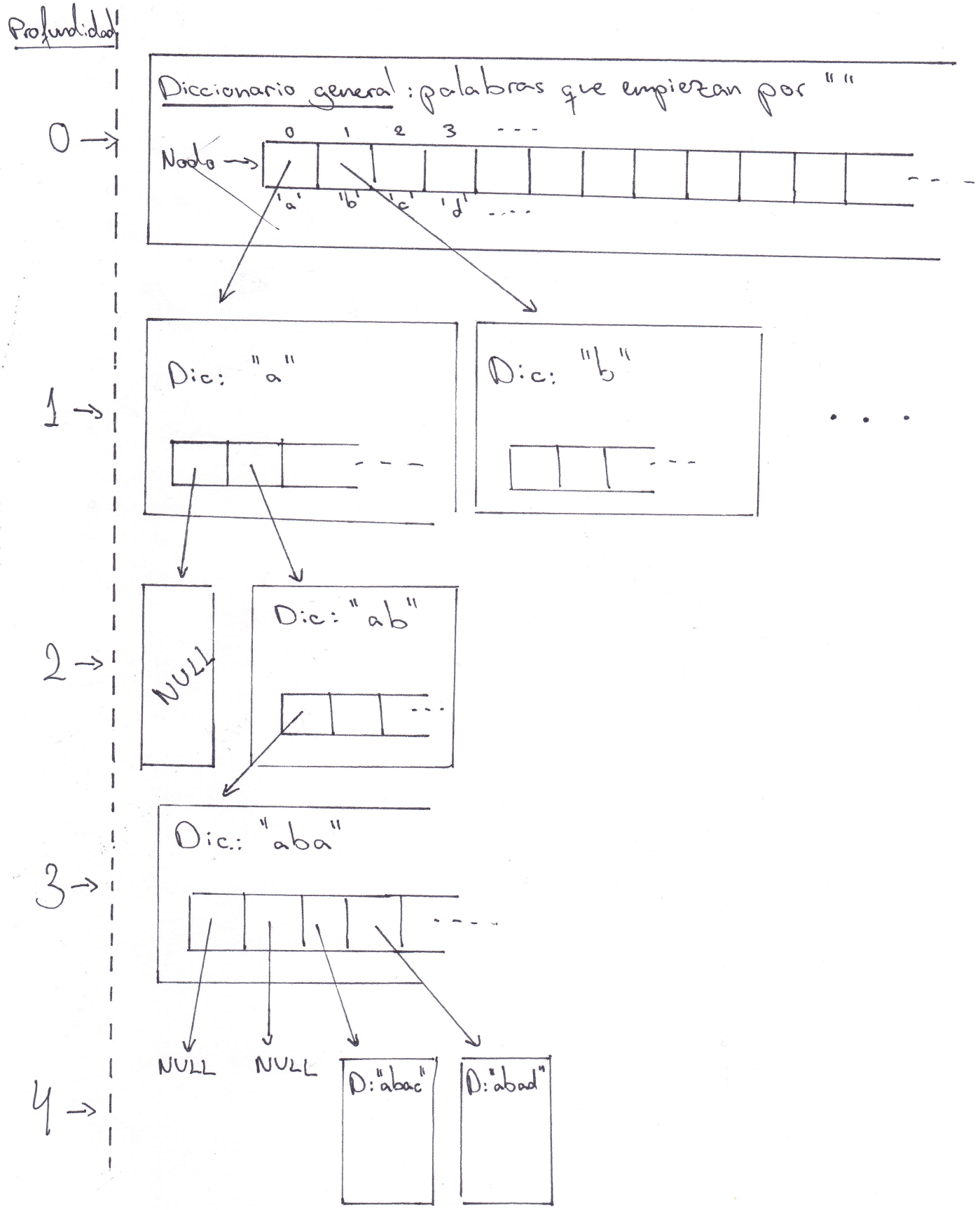
Para facilitar la búsqueda recursiva, cada posición es un struct que contiene, además del puntero a la siguiente profundidad del árbol, un campo set/unset que indica tanto si se ha encontrado una palabra como si hay que seguir buscando por ese camino.

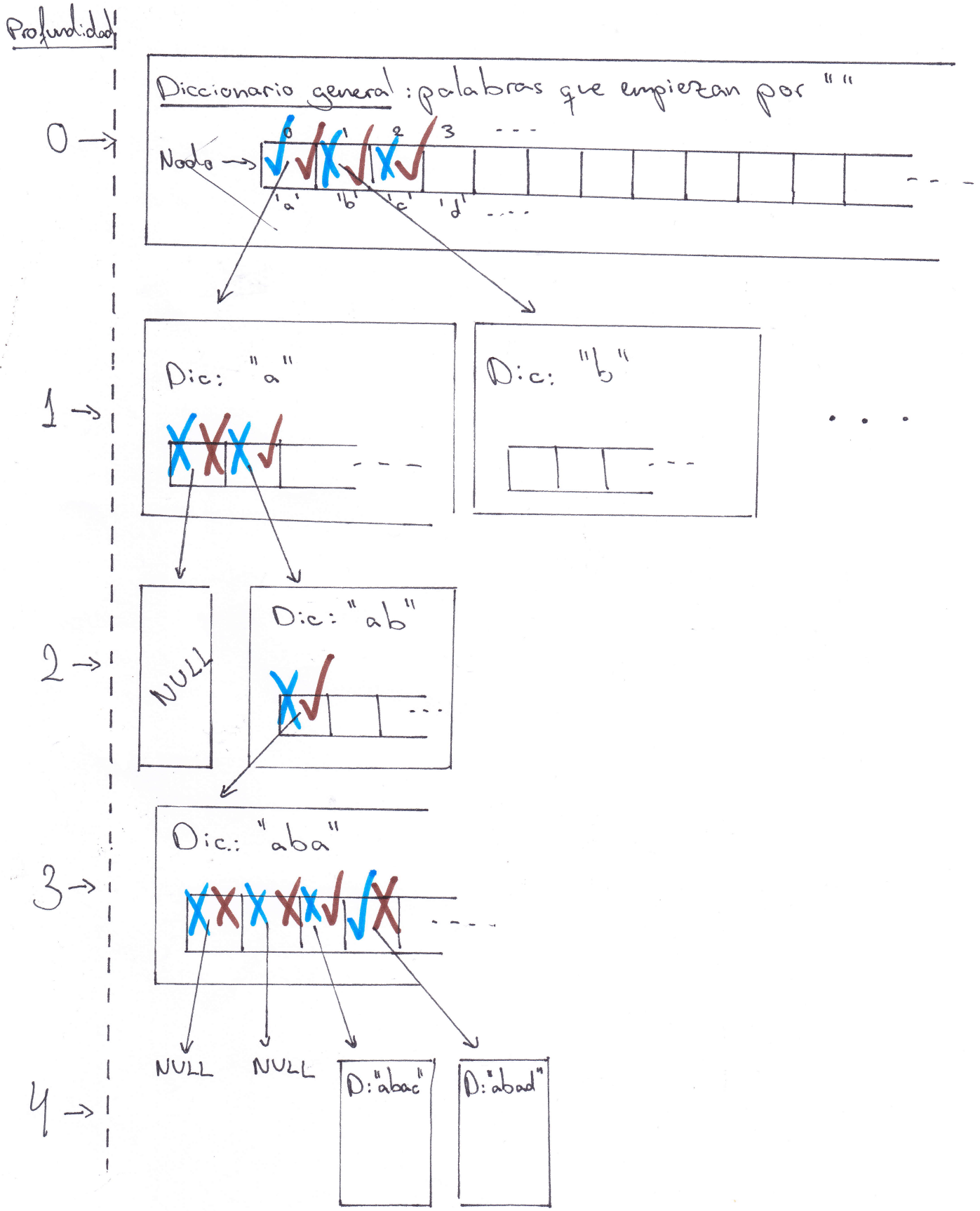
2- Implementación del TDA Juego

Se ha implementado un módulo adicional para que administre el transcurso de una partida.

ANEXOS

Se adjunta un esquema de una parte del diccionario procedente de rae_std.txt.
Se adjunta una copia del mismo con la información de búsqueda previamente mencionada: Azul para el campo “aquí hay palabra” y marrón para “seguir buscando por aquí”.


Profundidad
Diccionario general: palabras que empiezan por " "
0 →
Nodo →
0 1 2 3 ...
'a' 'b' 'c' 'd' ....
1 →
Dic: " a "
Dic: " b "
2 →
NULL
Dic: " ab "
3 →
Dic: " aba "
NULL NULL
D: "abac"
D: "abad"
4 →


Profundidad
0
1
2
3
4
Diccionario general: palabras que empiezan por " "
Nodo
0
1
2
3
'a'
'b'
'c'
'd'
Dic: "a"
D:c: "b"
NULL
Dic: "ab"
Dic: "aba"
NULL
NULL
D: "abac"
D: "abad"