Logotipo

Descripción generada automáticamente con confianza mediaLogotipo

Descripción generada automáticamenteINSTITUTO POLITECNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ANÁLISIS Y DISEÑO DE ALGORITMOS

PRÁCTICA 08

Rodrigo García Mayorga

2012630554

# Introducción

El **Backtracking** o **vuelta atrás** es una técnica algorítmica fundamental en la resolución de problemas de búsqueda y toma de decisiones. Se basa en la exploración sistemática de todas las posibles soluciones mediante un enfoque de prueba y error, descartando aquellas opciones que no cumplen con las condiciones del problema en el menor tiempo posible. Este enfoque resulta especialmente útil en problemas de tipo **combinatorio**, como los rompecabezas, las permutaciones, el problema de las N reinas, los laberintos, y la coloración de grafos, entre otros.

La estrategia de Backtracking se implementa generalmente mediante **recursividad**, donde el algoritmo explora una solución parcial, y si esta no conduce a una solución válida, retrocede ("backtrack") y prueba una alternativa diferente. Esta metodología reduce considerablemente el número de soluciones candidatas que deben evaluarse, al eliminar tempranamente aquellas ramas del árbol de decisiones que no llevan a una solución válida.

El objetivo de esta práctica es aplicar el método de Backtracking en la resolución de un problema clásico, comprendiendo su estructura, funcionamiento y ventajas frente a otros métodos de fuerza bruta. Se busca fortalecer la capacidad de diseñar algoritmos eficientes para la resolución de problemas complejos mediante una adecuada estructuración de la lógica recursiva y el uso adecuado de estructuras de datos auxiliares.

# Desarrollo

### Descripción General

Este código implementa una solución para determinar si una cadena es un "scramble" de otra. Dos cadenas son consideradas "scramble" si:

* Tienen la misma longitud
* Una puede ser transformada en la otra mediante divisiones y reordenamientos recursivos

### Estructura del Código

Función Principal `isScramble`

bool isScramble(char\* s1, char\* s2)

Casos Base

* Verifica si las cadenas tienen diferente longitud
* Comprueba si las cadenas son idénticas
* Maneja el caso de cadenas de longitud 1

Verificación de Caracteres

int count[26] = {0};

for (int i = 0; i < len; i++) {

count[s1[i] - 'a']++;

count[s2[i] - 'a']--;

}

* Usa un array para contar la frecuencia de cada carácter
* Asegura que ambas cadenas contengan los mismos caracteres

Proceso Recursivo

1. División de Cadenas:

* Divide las cadenas en todas las posibles posiciones
* Prueba dos casos:
  + Sin intercambio: s1[0:i], s1[i:n] con s2[0:i], s2[i:n]
  + Con intercambio: s1[0:i], s1[i:n] con s2[n-i:n], s2[0:n-i]

2. Manejo de Subcadenas:

strncpy(left1, s1, i);

left1[i] = '\0';

strcpy(right1, s1 + i);

3. Ejemplo de Uso

Entrada:

s1 = "great"

s2 = "rgeat"

Proceso:

1. División en "g|reat"

2. "reat" se divide y reordena

3. Se obtiene "rgeat"

Salida: true

4. Complejidad

* Tiempo: O(n!), donde n es la longitud de las cadenas
* Espacio: O(n) para la recursión

5. Limitaciones

* Máximo 30 caracteres por cadena
* Solo procesa letras minúsculas
* No maneja caracteres especiales

6. Función Main

int main() {

char s1[31], s2[31];

// Lectura de entrada y llamada a isScramble

// Impresión del resultado

}

Esta implementación proporciona una solución completa al problema de determinar si dos cadenas son scrambles entre sí, utilizando un enfoque recursivo con división y conquista.

# Bibliografía

* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
* Brassard, G., & Bratley, P. (1996). *Fundamentals of Algorithmics*. Prentice Hall.
* Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
* Skiena, S. S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer.

## Enlace del código (github):

<https://github.com/LowisN/ADA_Practica-7>