Clase Metodológica: Recursión (backtrack)

Dr.C. Alejandro Piad Morffis
Profesor Principal

Plan E :: Ciencia de la Computación :: Programación

Agenda

- Contexto global: Recursión en Ciencia de la Computación.
- Contexto local: Backtrack en el tema Recursión
- Objetivos y habilidades
- Método: orientado a problemas
- Guión de la clase
- Prácticas y estudio independiente

Contexto global

Recursión en Ciencia de la Computación. Dos miradas:

- Estrategia de solución de problemas
  - Divide y vencerás, backtrack, programación dinámica.
  - Puntos de contacto con EDA, MD, IA, y DAA.

- Paradigma de cómputo
  - Funciones primitivas recursivas, máquinas de Turing, gramáticas
  - Puntos de contacto con MD, Compilación, DAA.

Recursión es un tema central en la Ciencia de la Computación. Este es el primer encuentro de los estudiantes con el paradigma en su máxima expresión, por lo que es un buen punto para crear expectativas de cuán útil es este conocimiento.

Contexto local

Backtrack dentro del tema de Recursión.

- Antecedentes:
  - Algoritmos iterativos básicos (ciclos y arrays)
  - Algoritmos recursivos de una sola rama (búsqueda binaria, recurrencias, ...)
  - Paradigma divide & conquer (mergesort, quicksort, ...)
- **Backtrack para problemas de satisfacibilidad**
- Posteriormente
  - Backtrack para optimización
  - Problemas NP-Hard (intuiciones)
  - Árboles y grafos

Contexto local

El tema Recursión se enmarca en el paradigma de **la búsqueda como estrategia de solución general de problemas**.

Hasta este punto, los estudiantes han visto dos estrategias generales de solución:

- constructiva (e.g., problemas matemáticos)
- búsqueda en espacios explícitamente definidos (e.g., búsqueda en arrays)

Con recursión han visto ambos enfoques (búsqueda binaria y ordenación respectivamente).

El siguiente paso es **búsqueda en espacios combinatorios que están implícitamente definidos**.

Objetivos

Programación está en la eterna tensión entre dos habilidades:

- Una de alto nivel (**conceptualización**): encontrar soluciones efectivas y eficientes a problemas computacionales
- Y otra de bajo nivel (**implementación**): codificar esas soluciones en un lenguaje que pueda entender una computadora

Pregunta central de la clase

¿Cómo explorar un espacio combinatorio de forma exhaustiva para encontrar un objeto que satisface cierta propiedad?

Habilidades

Alto nivel (conceptualización)

- identificar un problema computacional como un problema de satisfacibilidad
- definir un espacio combinatorio de todas las posibles soluciones a dicho problema
- definir un criterio de parada (noción de satisfacibilidad)

Bajo nivel (implementación)

- implementar un método recursivo que explora el espacio combinatorio
- implementar el método portal que inicia la búsqueda
- implementar criterios de poda relevantes

Método orientado a problemas

Para motivar la clase, se comienza con un problema sencillo de explicar, por ejemplo...

> ***El cuadrado mágico***: *Dado una matriz de $N \times N$, ubicar los números de* 1 *a* N^2 *tal que todas las filas, columnas y diagonales sumen lo mismo.*

El estilo de la clase es *expositivo-interactivo*: se construye la solución al problema a partir de la interacción con los estudiantes.

El método de exposición es guíado por la solución a este problema concreto, ilustrando los conceptos relevantes (caso de parada, paso recursivo, poda, etc.,) y generalizándolos posteriormente.

Guión de la clase

- Presentar el problema
- Discutir estrategias de solución constructivas (ver por qué fallan)
- Replantear como problema de búsqueda, suponiendo que tuviéramos el conjunto de todos los cuadrados mágicos en un array
- Implementar la solución "mágica" donde se enumeran todos los posibles arrays...

```
foreach(var square in AllSquares()) {
    if (IsCorrect(square)) {
        return square;
    }
}
```

Guión de la clase

- En este punto llegamos a la conclusión de que si pudiéramos enumerar todos los posibles cuadrados, la solución al problema sería trivial (asumiendo que `IsMagical` es fácil de programar).
- Ahora el problema es hacerlos llegar a la noción intuitiva de que enumerar todos los cuadrados implica **aplicar y deshacer mútiples decisiones**.
- Esto lo logramos tratando de enumerar en pizarra los cuadrados de 2x2, viendo que cuando llegamos al final de un cuadrado, es necesario dar un paso hacia atrás y tomar otro camino.

Guión de la clase

Introducir el patrón general de backtrack:

```
void Backtrack(solution, step) {
    if (IsComplete(solution)) {
        BaseCase(solution);
    }

    foreach(decision in Options(solution, step)) {
        Apply(decision, solution);
        Backtrack(solution, Next(step));
        Undo(decision, solution);
    }
}
```

Guión de la clase

- Aplicar el patrón general al problema del cuadrado mágico.
- Determinar condición de parada, decisiones recursivas, y qué significa aplicar / deshacer.
- Ver el método portal que inicia la búsqueda (qué significa el paso 0)
- Ejecutar el programa en pantalla imprimiendo todos los pasos recursivos.

Hasta este punto deben tener una idea intuitiva de la estructura de una solución con backtrack.

Ahora toca generalizar.

Guión de la clase

Explicar la estructura formal de una solución con backtrack:

```
==> paso recursivo
-------------------------
| d1 | d2 | d3 | ...    |
-------------------------
 d11  d21  d31 \
 d12  d22  d32 | ciclo iterativo
 d13  d23  d33 /
```

Formalizar los elementos a reconocer en el problema:

- secuencia de decisiones a tomar
- opciones en cada paso
- criterio de satisfacibilidad

Guión de la clase

A modo de conclusiones parciales...

- Motivar con la idea de que backtrack es suficientemente poderoso como para ser un paradigma de cómputo en sí mismo (que verán más adelante en Prolog y en IA).
- Recalcar la idea de que logramos resolver el problema porque convertimos una pregunta de *construir un objeto complejo* en una pregunta de *enumerar todos los objetos e identificar el correcto*.

Aquí es importante lograr que los estudiantes vean la potencia de este cambio de punto de vista, cómo nos permitió resolver "fácilmente" un problema que parecía impenetrable.

Prácticas y estudio independiente

Programación tiene dos tipos de clases prácticas:

- En aula: clases conceptuales, solución de problemas tipo en equipo y de conjunto con el profesor.
- En el laboratorio: trabajo individual y aclaración de dudas.

Orientar ejercicios típicos de backtrack para satisfacibilidad:

- Sudoku
- N reinas
- Camino del caballo
- ...