UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ciencias de la Ingeniería y Tecnología

Unidad Valle de las Palmas

Análisis de Algoritmos

Meta 4.1
Backtracking, y branch and bound.

José Humberto Moreno Mejía
Paola Gómez Faustino

OCTUBRE 2024

Algoritmo de Vuelta Atrás (Backtracking)

El algoritmo de vuelta atrás es una técnica de búsqueda utilizada para resolver problemas de toma de decisiones, como la generación de combinaciones, permutaciones, y la resolución de rompecabezas. Funciona explorando todas las posibles soluciones y retrocediendo cuando se encuentra una solución que no cumple con las condiciones del problema.

Funcionamiento

1. **Elección**: Seleccionar un elemento o una opción.
2. **Validación**: Verificar si la opción seleccionada es válida.
3. **Solución**: Si la opción es válida y completa la solución, se guarda como una solución.
4. **Retroceso**: Si la opción no lleva a una solución, se retrocede y se elige otra opción.

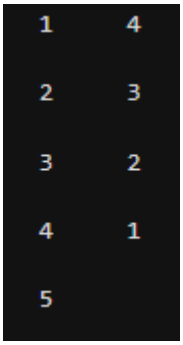
Ejemplo Matlab:

```
function SumaSubconjuntos(conjunto, suma)
    n = length(conjunto);
    % Llamar a la función de backtracking
    encontrar_subconjuntos(conjunto, suma, n, []);
end

function encontrar_subconjuntos(conjunto, suma, n, sub)
    % Comprobar si la suma del subconjunto actual es igual a la suma deseada
    if sum(sub) == suma
        disp(sub);
    end

    % Recorrer los elementos del conjunto
    for i = 1:n
        % Evitar el uso de elementos ya considerados
        if ~ismember(conjunto(i), sub)
            % Agregar el elemento actual al subconjunto
            sub = [sub, conjunto(i)];
            % Llamar recursivamente para el siguiente elemento
            encontrar_subconjuntos(conjunto, suma, n, sub);
            % Retirar el último elemento (retroceso)
            sub(end) = [];
        end
    end
end

% Ejemplo de uso
conjunto = [1, 2, 3, 4, 5];
suma = 5;
SumaSubconjuntos(conjunto, suma);
```

Problema: Suma de Subconjuntos

El objetivo es encontrar todos los subconjuntos de un conjunto dado cuya suma sea igual a un número específico.

Salida:



La salida muestra todos los subconjuntos del conjunto {1, 2, 3, 4, 5} que suman exactamente 5.

Cada línea representa un subconjunto cuya suma coincide con el valor objetivo. El algoritmo explora diferentes combinaciones y utiliza el retroceso para probar todas las posibilidades hasta encontrar todas las soluciones válidas.3

Algoritmo de Ramificación y Poda (Branch and Bound)

El algoritmo de ramificación y poda es una técnica de optimización utilizada para resolver problemas de decisión, como la búsqueda de la solución óptima en problemas de programación lineal y problemas de optimización combinatoria. Se basa en dividir el problema en subproblemas más pequeños (ramificación) y descartar aquellos que no pueden producir mejores resultados que la mejor solución encontrada hasta el momento (poda).

Funcionamiento

1. **Dividir**: Se divide el problema en subproblemas más pequeños.
2. **Evaluar**: Se evalúan los subproblemas y se calcula una cota para cada uno.
3. **Podar**: Se descartan aquellos subproblemas cuya cota es peor que la mejor solución conocida.
4. **Repetir**: Se repite el proceso hasta encontrar la solución óptima.

Ejemplo Matlab:

```
function Mochila(pesos, valores, capacidad)
    n = length(valores);
    % Inicializar el valor máximo
    valor_maximo = 0;
    % Llamar a la función de ramificación y poda
    valor_maximo = branch_and_bound(pesos, valores, capacidad, n, 0, 0);
    disp(['Valor máximo: ', num2str(valor_maximo)]);
end

function max_valor = branch_and_bound(pesos, valores, capacidad, n, peso_actual, valor_actual)
    % Si se ha alcanzado el número de elementos
    if n == 0
        max_valor = valor_actual;
        return;
    end

    % Omitir el elemento actual (ramificación)
    max_valor = branch_and_bound(pesos, valores, capacidad, n-1, peso_actual, valor_actual);

    % Incluir el elemento actual si no se excede la capacidad
    if peso_actual + pesos(n) <= capacidad
        max_valor_incluido = branch_and_bound(pesos, valores, capacidad, n-1, peso_actual + pesos(n), valor_actual + valores(n));
        max_valor = max(max_valor, max_valor_incluido); % Podar
    end
end

% Ejemplo de uso
pesos = [2, 3, 4, 5];
valores = [3, 4, 5, 6];
capacidad = 5;
Mochila(pesos, valores, capacidad);
```

Problema: Problema de la Mochila

El objetivo es encontrar el valor máximo que se puede obtener al seleccionar elementos sin exceder la capacidad de la mochila.

Salida:

```
Valor máximo: 7
```

La salida muestra el valor máximo que se puede obtener al seleccionar elementos del conjunto de pesos y valores, sin exceder la capacidad de la mochila.

Elementos considerados:

- Peso 2, Valor 3
- Peso 3, Valor 4
- Peso 4, Valor 5
- Peso 5, Valor 6

Al examinar todas las combinaciones posibles y aplicar la técnica de ramificación y poda, el algoritmo determina que la mejor combinación para maximizar el valor sin superar el peso total de 5 es seleccionar los elementos con peso 2 (valor 3) y peso 3 (valor 4), lo que da un valor total de 7.

En resumen:

- **Vuelta Atrás**: Encuentra todos los subconjuntos que suman a un valor específico, mostrando combinaciones válidas.
- **Ramificación y Poda**: Optimiza la selección de elementos para maximizar el valor total sin exceder la capacidad, mostrando el valor máximo alcanzado.