ESTRATEGIAS DE PROGRAMACIÓN

Algoritmos y Programación

Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

28 de septiembre de 2023

Estrategias

- Fuerza bruta (*brute force*)
- Vuelta atrás (*backtracking*)
- Voráz (*greedy*)

- Divide y vencerás
  - Reduce y vencerás

Técnica

- Programación Dinámica

Fuerza Bruta (*Brute Force*)

- Basada en la definición del problema
  - Evalúa todas las posibles combinaciones que resuelven el problema

- Fortalezas:
  - Amplia aplicabilidad
  - Simple
  - Genera soluciones razonables para algunos problemas

- *Debilidades*:
  - Algoritmos poco eficientes
  - En general, con poco esfuerzo podemos hacerlo mucho mejor !

https://en.wikipedia.org/wiki/Brute-force_search

Programación mediante fuerza bruta

- En general, no resulta práctico generar todas las combinaciones antes de procesarlas porque:
  - Pueden ocupar demasiado <u>espacio</u> y realmente sólo necesitamos procesar una combinación cada vez (no necesitamos tenerlas todas a la vez).
  - El <u>tiempo</u> que tardamos en generar todas las combinaciones puede retrasar mucho el comienzo de su procesamiento.

Soluciones (1/3)

- Utilizar código <u>iterativo</u> (bucles anidados) para generar y comprobar las combinaciones.

  - <u>Ventaja</u>
    - Fácil de programar

  - <u>Inconvenientes</u>
    - Procesar más o menos combinaciones requiere añadir o quitar bucles.

      → Dificulta programar el caso general.

Soluciones (2/3)

- Utilizar código <u>recursivo</u> para generar y comprobar las combinaciones.
  - <u>Ventajas</u>
    - Fácil de programar
  - <u>Inconvenientes</u>
    - El límite de llamadas recursivas del intérprete/compilador de nuestro lenguaje de programación.
    - Cuando el número de combinaciones es muy grande no podemos detener la ejecución del programa, guardar el estado del programa (por ejemplo, en un fichero) y continuar más adelante su ejecución.

Soluciones (3/3)

- Utilizar un iterador
  - Es un componente que evalúa perezosamente las combinaciones; lo llamamos cuando necesitamos la "siguiente" combinación.

  - Ventaja
    - Podemos detener la ejecución, guardar estado y continuar.

  - Inconveniente
    - No está disponible en el soporte estándar de todos los lenguajes (pero podemos construirlo fácilmente).

Python: Generadores (*lazy evaluation*)

```
x = (j for j in range(5) if j % 2 == 0)      # Generator expression
print(x)           # <generator object <genexpr> at ….>
print(next(x))  # 0
print(next(x))  # 2
print(next(x))  # 4

x = (x for x in range(5))
print(list(x))              # [0, 1, 2, 3, 4]
```

Si no quedan más elementos y llamamos a next() eleva la excepción StopIteration

Podemos llamar al constructor de lista para depurar la expresión generadora y ver la lista completa

Python: Escribiendo nuestras funciones generadoras

```
def numbers(n):
    for j in range(n):
        yield j
```

Función generadora

```
g = numbers(2)
print(g)            # <generator object numbers at 0x….>

print(next(g))      # 0
print(next(g))      # 1

print(list(numbers(5)))        # [0, 1, 2, 3, 4]
```

Construcción de nuestro iterador para para Fuerza Bruta

¿ Cómo podemos construir un iterador para calcular todas las combinaciones de N números ?

Ejemplo:

Datos de entrada: 10 20 30 40

Supongamos que nuestro objetivo es encontrar qué números debo elegir para que su producto minimice la distancia a un valor X

Ejemplo

Datos de entrada: 10   20   30 40

Ejemplo

Datos de entrada:	10	20	30	40
	-	-	-	x
	-	-	x	-
	-	x	-	-
	x	-	-	-

Combinaciones de 1 elemento

Ejemplo

Datos de entrada: 10 20 30 40

-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-

Combinaciones de 1 elemento

+

Combinaciones de 2 elementos

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40
-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-

Combinaciones de 1 elemento

\+

Combinaciones de 2 elementos

\+

Combinaciones de 3 elementos

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40
-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-
x	x	x	x

Combinaciones de 1 elemento

\+

Combinaciones de 2 elementos

\+

Combinaciones de 3 elementos

\+

Combinaciones de 4 elementos

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40
-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-
x	x	x	x

¿ Cómo podemos generarlas ?

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40
-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-
x	x	x	x

0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40
-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-
x	x	x	x

0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Ejemplo

Datos de entrada: 10 20 30 40

-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-
x	x	x	x

0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40				
-	-	-	x	0	0	0	1
-	-	x	-	0	0	1	0
-	x	-	-	0	0	1	1
x	-	-	-	0	1	0	0
-	-	x	x	0	1	0	1
-	x	-	x	0	1	1	0
x	-	-	x	0	1	1	1
-	x	x	-	1	0	0	0
x	-	x	-	1	0	0	1
x	x	-	-	1	0	1	0
-	x	x	x	1	0	1	1
x	-	x	x	1	1	0	0
x	x	-	x	1	1	0	1
x	x	x	-	1	1	1	0
x	x	x	x	1	1	1	1

Ejemplo

Datos de entrada: 10 20 30 40

10	20	30	40
-	-	-	x
-	-	x	-
-	x	-	-
x	-	-	-
-	-	x	x
-	x	-	x
x	-	-	x
-	x	x	-
x	-	x	-
x	x	-	-
-	x	x	x
x	-	x	x
x	x	-	x
x	x	x	-
x	x	x	x

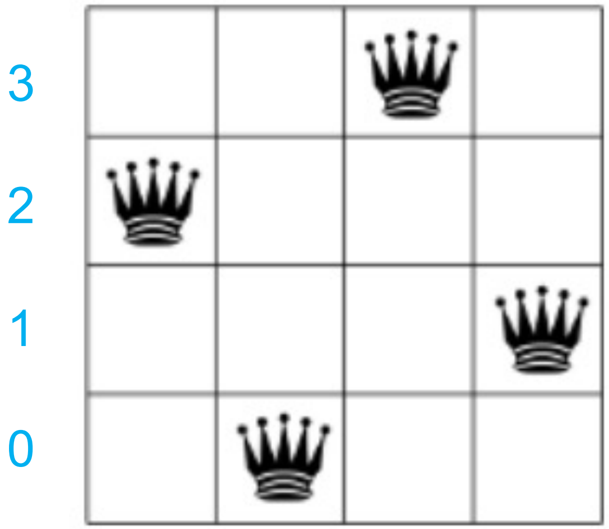
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

En resumen

- En este problema la infraestructura mínima necesaria para calcular todas las combinaciones de N elementos es un contador en base 2

Podemos generalizarlo fácilmente para resolver más problemas de combinatoria

Otros problemas de combinatoria



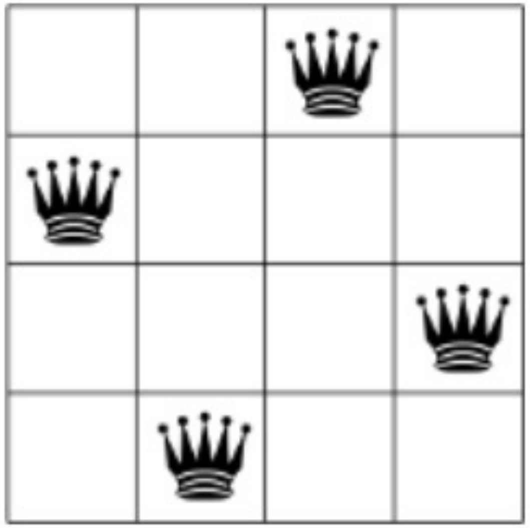
N-Queens

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku

Podemos utilizar los dígitos de un número en base 4 para representar en qué fila colocamos cada reina. En este ejemplo el número sería el [2, 0, 3, 1]

Otros problemas de combinatoria



N-Queens Problem

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Sudoku

Podemos resolver Sudoku utilizando 81 dígitos de un número en base 9