Backtracking

Bach. Rodolfo Mercado Gonzales
Programación Competitiva UPC

Backtracking

- Técnica de fuerza bruta basada en la recursividad.
- Permite iterar por un espacio de búsqueda formado por configuraciones (arreglos, matrices, etc).

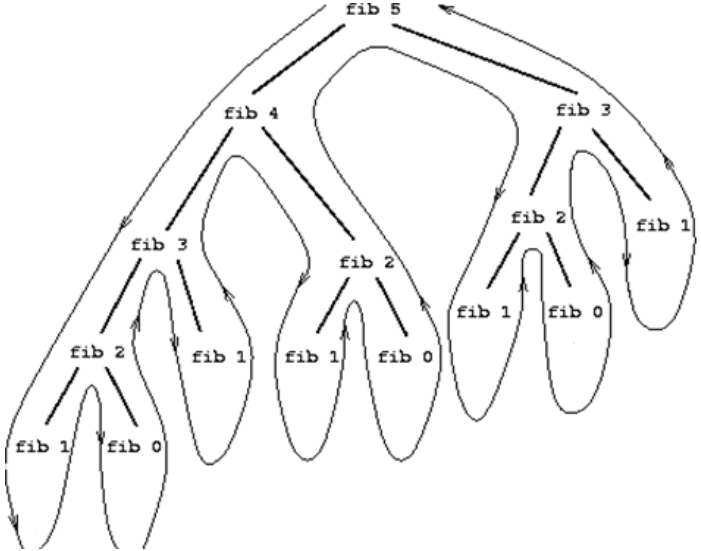
Backtracking

Idea General

Construir una configuración incrementalmente (partimos sin elementos y los vamos agregando paso a paso), probando todas las posibles formas.

Observaciones

- Aprovecharemos el recorrido en profundidad que tienen una función recursiva.
- Cuando ya no se puede expandir en profundidad se da un paso atrás (backtrack).

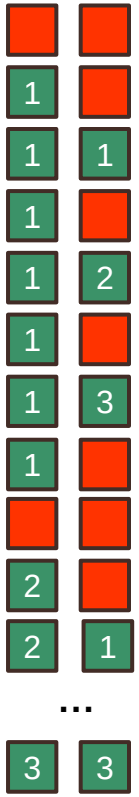


Ejemplo

Generar todas las configuraciones posibles de 2 elementos usando los números del 1 al 3.

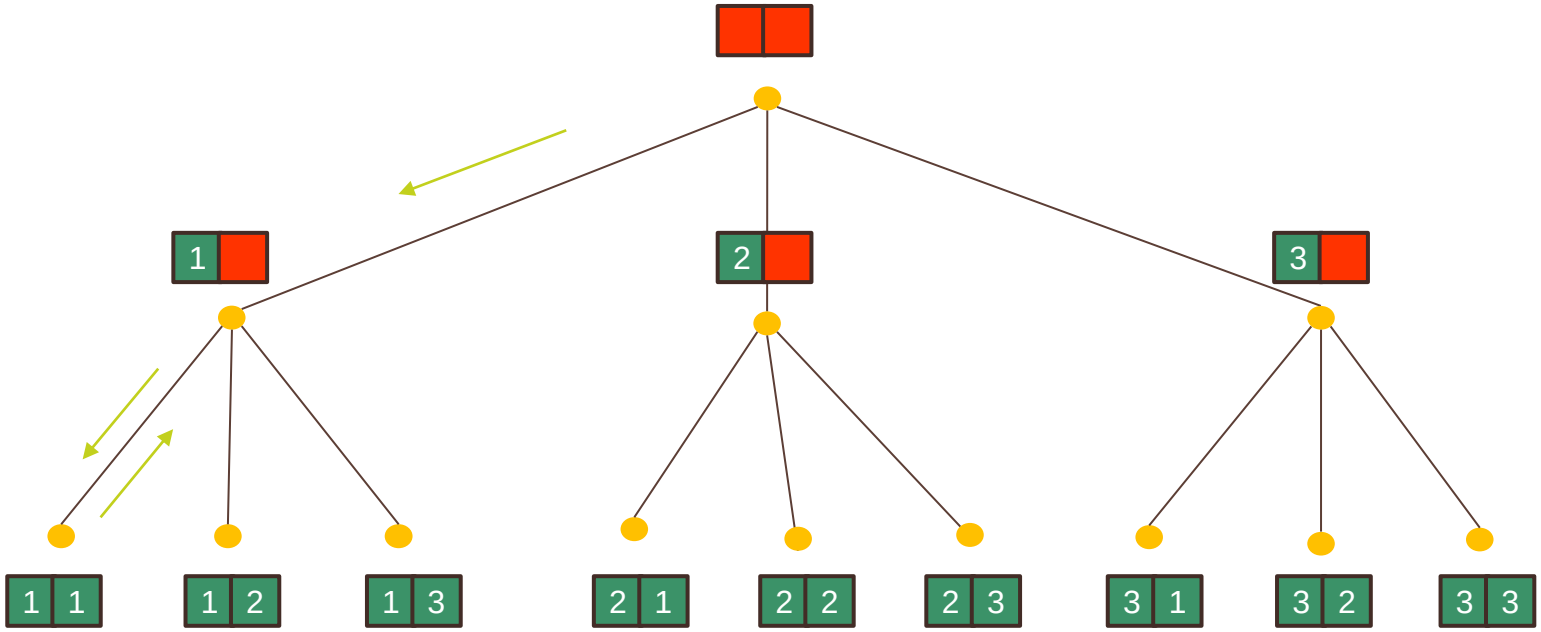
Solución:

Tenemos que generar arreglos de 2 elementos, donde cada elemento puede ser un número del 1 al 3.



Ejemplo

Ahora veámoslo de manera recursiva

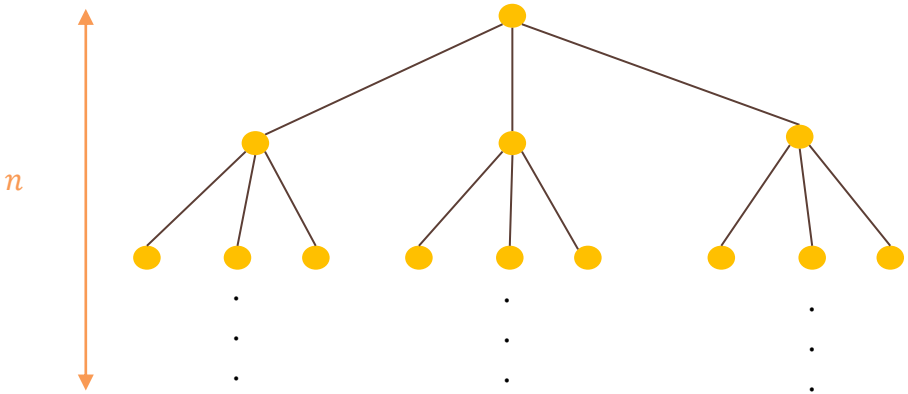


Implementación

1. Testeamos si la configuración buscada ya ha sido encontrada
2. Sino, para cada opción en la posición actual:
   2.1. Incluimos esta opción en nuestra configuración.
   2.2. Recursivamente completamos las siguientes posiciones.
   2.3. Retiramos esta opción de nuestra configuración.

```
void fill_cell(int pos) {
    if (pos == n) { //ya formé una conf de tamaño n
        for (int i = 0; i < n; ++i) cout << " " << conf[i];
        cout << endl;
        return;
    }
    //probamos todas las opciones para la casilla pos
    for (int op = 1; op <= m; op++) {
        conf[pos] = op; //incluimos la opción
        fill_cell(pos + 1);
        conf[pos] = -1; //retiramos la opción
    }
}
```

Análisis del tiempo de ejecución



nivel 0 : m^0 llamadas de m operaciones

nivel 1: m^1 llamadas de m operaciones

nivel 2: m^2 llamadas de m operaciones

$$T(n,m) = (m^0 + m^1 + ..m^{n-1}) * m$$
$$= m * (m^n - 1)/(m - 1)$$

$$\boldsymbol{O}(m^n)$$

Análisis del tiempo de ejecución

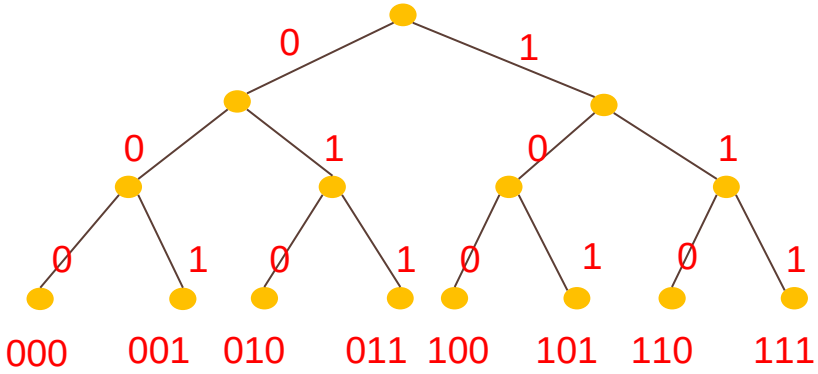
- Generalmente el backtracking puede ser implementado con una complejidad cercana al tamaño del espacio de búsqueda.
- La complejidad de un backtracking generalmente es exponencial o peor.

Subconjuntos

Generar todos los subconjuntos de un conjunto de *n* elementos.

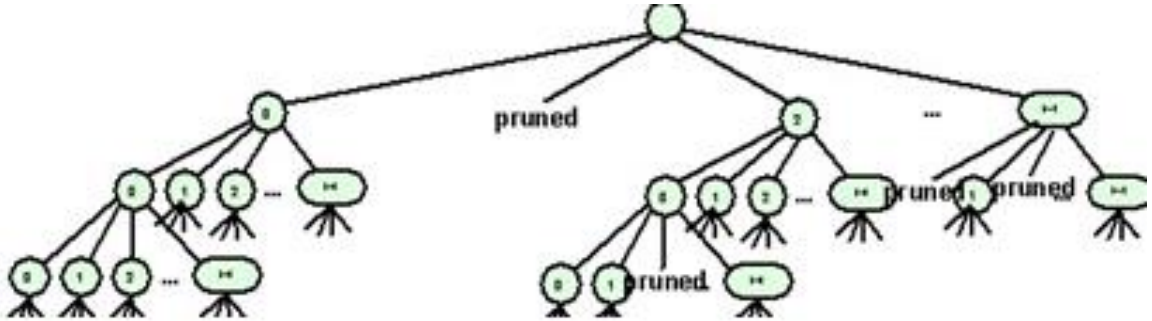
Solución:

Tenemos que generar arreglos de n elementos, donde cada elemento es mapeado por un 0 (no está en el subconjunto) o 1 (está en el subconjunto)



Pruning (poda)

- Es una “validación temprana” que se realiza para reducir nuestro espacio de búsqueda.
- Nos permite saber si una configuración está siendo formada correctamente en cada paso en lugar de esperar a que se haya completado toda la configuración y recién ahí validar.



Estructura general del backtracking

1. Testeamos si la configuración buscada ha sido encontrada
2. Sino, para cada opción válida en la posición actual (verificamos si es posible hacer una poda):
   2.1. Incluimos esta opción en nuestra configuración.
   2.2. Marcamos esta opción como no disponible (en caso de ser necesario).
   2.3. Recursivamente completamos las siguientes posiciones.
   2.4. Retiramos esta opción de nuestra configuración
   2.5. Desmarcamos esta opción, es decir ahora debe figurar como disponible.

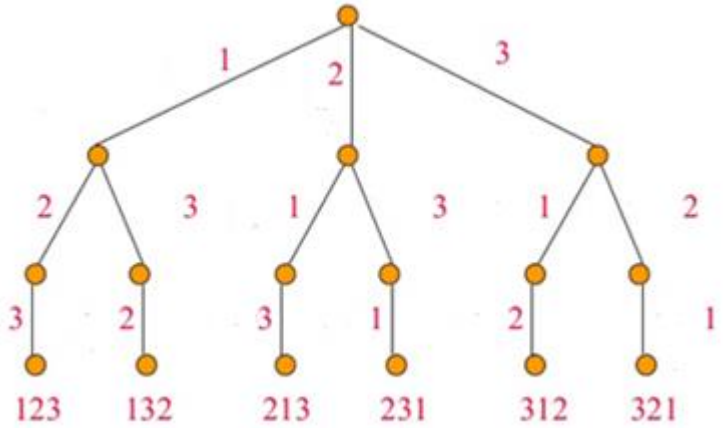
.

Permutaciones

Generar todas las permutaciones de los números del 1 al n.

Solución:

Tenemos que generar arreglos de n elementos, donde cada elemento puede ser un número del 1 al n y *todos deben ser distintos.*



$n = 3$

Referencias

- Steven Skiena – The Algorithm Design Manual
- Topcoder – Computational Complexity
- Antti Laaksonen – Guide to Competitive Programming

“Whether you think you can or think you can’t, you’re right.”

- Henry Ford