Conceptos avanzados: back-tracking

Se quieren obtener todas las secuencias de dígitos con tamaño M y que sumen N. Ej: todas las secuencias de tamaño 2 que sumen 2 son 0-2, 2-0 y 1-1.

Idea:

empezar con una colección de dígitos vacía

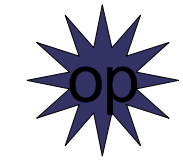
por cada uno de los 10 dígitos

Agregar el dígito como primer elemento de la colección

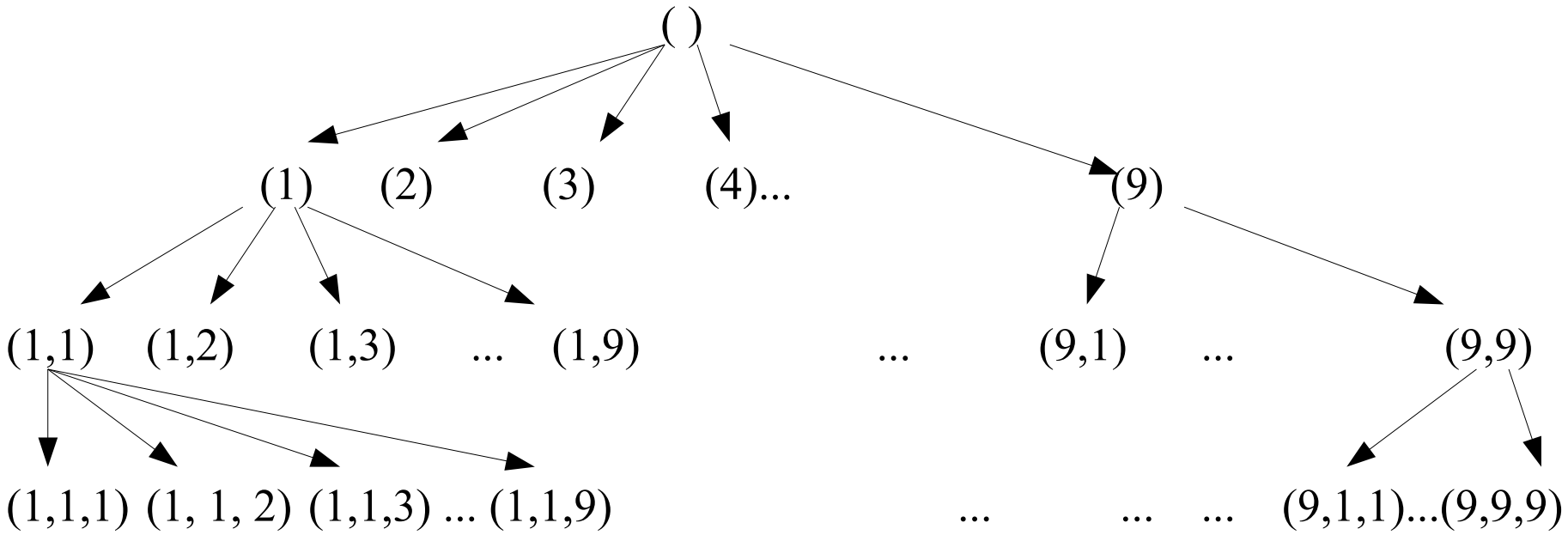
ver si esa colección tiene el tamaño M y también si suma N, si es así, agregarla al resultado

Sacar el elemento de la colección

Conceptos avanzados: back-tracking



Idea vista visualmente:



Se sigue hasta secuencias de tamaño M. En ellas se chequea la condición de suma.

Conceptos avanzados: back-tracking

Técnica de programación para hacer búsqueda sistemática a través de todas las configuraciones posibles dentro de un *espacio de búsqueda*.
Los algoritmos de tipo backtracking construyen posibles soluciones candidatas de manera sistemática. En general, dado una solución candidata s:

Verifican si s es solución. Si lo es, hacen algo con ella (depende del problema).

Construyen todas las posibles extensiones de s, e invocan recursivamente al algoritmo con todas ellas.

Conceptos avanzados: back-tracking

En el caso de ejemplo, se representa una solución candidata como un vector o colección

$$a = (a_1, \ldots, a_k).$$

Las soluciones candidatas se extenderán agregando un elemento al final.

¿Cuándo un candidato es una solución?

$$\sum_{i=0}^{k} a_i = n \text{ y } k = m$$

Conceptos avanzados: back-tracking

El siguiente es un algoritmo genérico de backtracking:

```
Bt(A, k)
    if SOLUCION?(A, k)
    then PROCESAR_SOLUCION(A, k)
    else for each c in SUCESORES(A, k)
    do
        A[k] = c
        Bt(A, k + 1)
```

donde

- SOLUCION?(·) es un método que retorna verdadero sí y sólo sí su argumento es una solución.
- PROCESAR_SOLUCION(·), depende del problema y que maneja una solución.
- SUCESORES(·) es un método que dado un candidato, genera todos los candidatos que son extensiones de éste.

Conceptos avanzados: back-tracking con objetos

GeneradorDeListas

```
>> generar: suma tamanio: tam
^self generarUsandoLista: OrderedCollection new suma: suma tamanio: tam.

>> generarUsandoLista: lista suma: suma tamanio: tamanio
|resultado|
(lista size == tamanio)
        ifTrue:[((lista inject: 0 into: [:a:b| a + b]) = suma)
                ifTrue:[resultado := OrderedCollection with: lista dcopy]
                ifFalse:[resultado:= OrderedCollection new]]
        ifFalse:[ resultado := OrderedCollection new.
                  0 to: 9 do:[:i |  lista addLast: i.
                                    resultado addAll: (self generarUsandoLista: lista suma:
                                       suma tamanio: tamanio).
                                    lista removeLast.]
                                ].
        ^resultado.
```

Conceptos avanzados: back-tracking con objetos

Discusión:

¿Cómo sería una posible forma de encontrar la solución a un SUDOKU dado?

Pensar: forma de representar el sudoku

forma de ir generando las soluciones

forma de saber la secuencia de sucesores correcta

analizar posibles optimizaciones en la generación de sucesores