Backtracking

El backtracking es un algoritmo general para encontrar un conjunto de soluciones a un problema computacional donde se va creando de forma incremental un conjunto de candidatos que formarán parte de una solución final. El algoritmo explora los elementos de un conjunto y selecciona/descarta subconjuntos de candidatos que pertenecen/no-pertenecen a la solución.

Este algoritmo debe ser aplicado a problemas donde existan elementos considerados "candidatos parciales de la solución" para permitir realizar verificaciones rápidas si dichos candidatos pertenecen a la solución ó no. No resulta útil para conjuntos no ordenados, donde la exploración es total (explorar todos los candidatos). Entonces, el punto clave de los algoritmos de backtracking es: descartar/seleccionar rápidamente las soluciones inválidas/válidas.

Visto de otro modo, la solución deseada debe expresarse como una n-tupla $(x_1, ..., x_n)$ donde los x_i son elegidos de algún conjunto finito S_i. Usualmente el problema a resolver requiere encontrar un vector que maximice/minimice/satisfaga una función criterio $F(x_1, ..., x_n)$. A veces, se busca todos los vectores que satisfagan F.

En un algoritmo de backtracking con datos de entrada P a resolver se pueden definir cinco funciones: aceptar, rechazar, primero, próximo y solución. Cada uno opera de la siguiente forma:

1. Inicio (P): Retorna el candidato parcial de la raíz del problema con datos P; sirviendo como inicialización
2. Aceptar (P, C): Retorna **true** si el candidato parcial C es una solución de P, y falso en caso contrario
3. Rechazar (P, C): Retorna **true** si un candidato parcial C no debería continuar su exploración ó búsqueda de más candidatos
4. Primero (P, C): Extrae el primer elemento/componente del candidato parcial C llamado s
5. Próximo (P, s): Genera el próximo conjunto de candidatos posterior a s
6. Solución (P, c): El subconjunto $c \in P$ se considera solución.

La llamada inicial del algoritmo se hace como Backtrack (P, Inicio (P)), donde Backtrack se puede escribir como:

```
void Backtrack (Set P, C)
  if Rechazar (P, C) then Abortar ( )
  if Aceptar (P, C) then Solucion (P, C)
  Set s = Primero (P, C)
  while not s.EsVacio() do
    Backtrack (P, s)
    s = Proximo (P, s)
  end
end
```

Es posible clasificar un algoritmo de backtracking de acuerdo al tamaño del subconjunto C solución:

- Una solución: Cuando el algoritmo encuentre una solución, su ejecución finaliza. En este enfoque el algoritmo se queda con la la primera solución que consigue.
- Solución óptima: Cuando el algoritmo explora todos los subconjuntos de soluciones posibles y se queda con la òptima para el problema a resolver.
- Todas las soluciones: El algoritmo colecta todos los subconjuntos encontrados en la exploración y forman parte de la solución.

Por ejemplo: permutar los caracteres de un String *strText*

```
void Permute (String strText, Integer iStartIndex, iLength)
  if iStartIndex == iLength then
    Print (strText)
  else
    for Integer iJ = iIndex to iLength do
      swap (ref strText, iIndex, iJ)
      Permute (strText, iIndex + 1, iLength)
      swap (ref strText, iIndex, iJ) //backtrack
    end
end
```

La llamada para un **String** $strText$ = "*Hello*" sería Permute(strText, 1, 5).

Existen diversos algoritmos clásicos que se resuelven con algoritmos de backtracking: rompecabezas, laberintos, permutaciones, problemas de las 8-reinas, crucigramas, Sudoku, problema de la mochila, problema del agente viajero, etc.

esmitt ramírez/2014