6. Backtracking

Antonio Pérez Carrasco

Universidad Rey Juan Carlos

Introducción

Introducción

- Backtracking (o vuelta atrás) técnica de más amplia utilización
- Resolución de un gran número de problemas, especialmente de optimización
- Existen problemas que requieren el estudio exhaustivo de un conjunto conocido a priori de posibles soluciones, en las que tratamos de encontrar una o todas las soluciones y por tanto también la óptima

Introducción

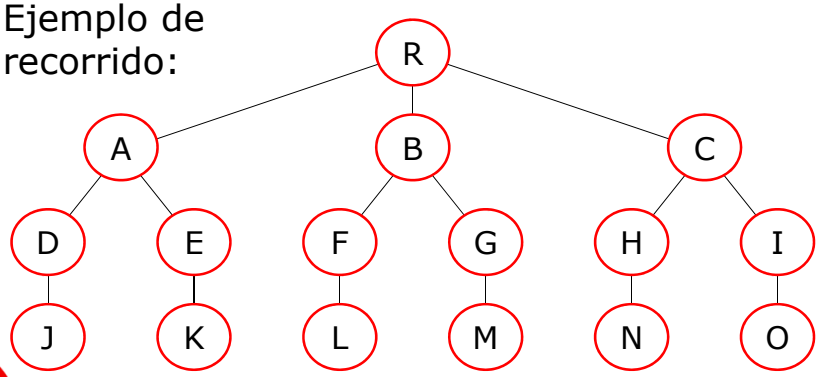
- Backtracking: manera sistemática de generar todas las posibles soluciones dentro de un espacio de búsqueda
- Las soluciones se resuelven por etapas, construyendo gradualmente una solución
- La solución es una n-tupla $[x_1, x_2, ..., x_n]$ donde cada x_i de este vector es elegido en cada etapa de entre un conjunto finito de alternativas, satisfaciendo las restricciones del problema

Introducción

- El espacio de soluciones se puede estructurar en forma de árbol de búsqueda, llamado espacio de búsqueda, donde en cada nivel se toma la decisión de la etapa correspondiente.
- Es un método de "fuerza bruta" pero "inteligente"

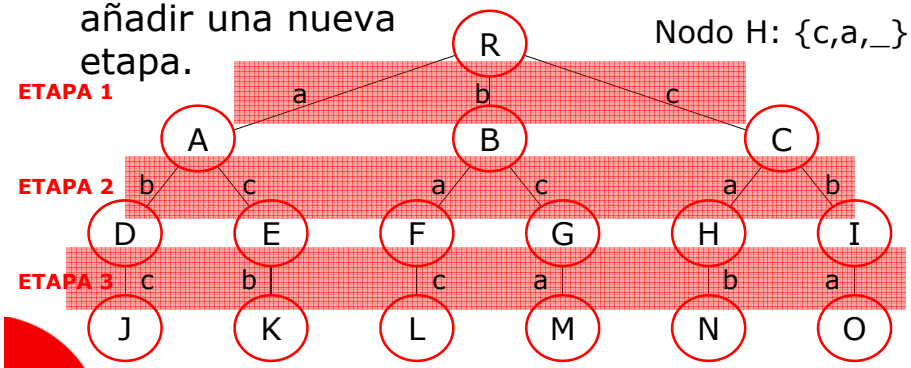
Introducción

- La estrategia que sigue, se asemeja a un recorrido en profundidad (por la izquierda) dentro de un árbol implícito, el árbol de búsqueda.
- Ejemplo de recorrido:



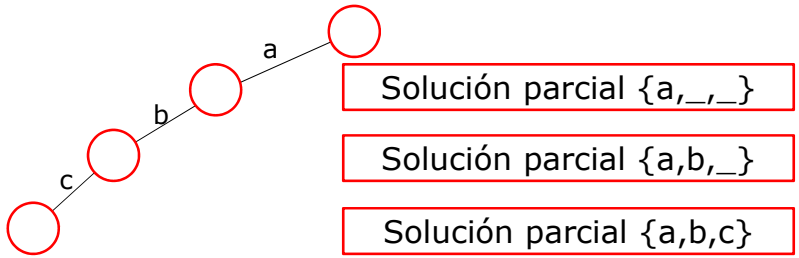
Introducción

- Este árbol es conceptual o implícito.
- Cada nodo de nivel k representa una parte de la solución, con k etapas ya realizadas.
- Sus hijos son las prolongaciones posibles al añadir una nueva etapa.



Introducción

- Para examinar el conjunto de posibles soluciones se ha de recorrer el árbol construyendo soluciones parciales a medida que se avanza en el recorrido.



- En este recorrido podemos obtener dos resultados posibles:

Introducción

- Éxito: llega a una solución (una hoja del árbol).
  - Piden una solución: finaliza aquí
  - Piden todas las soluciones o la mejor: seguirá explorando el árbol en búsqueda de soluciones alternativas.

Introducción

- Fracaso: en alguna etapa la solución parcial construida hasta el momento no se puede completar.
  - Vuelve atrás eliminando los elementos que se hubieran añadido en cada etapa a partir de ese nodo.
  - Si existe uno o más caminos aún no explorados que puedan conducir a solución, el recorrido del árbol continúa por ellos.

Introducción

- Estrategia: recorrido en profundidad
- En cada momento, el algoritmo se encontrará en un cierto nivel k, con una solución parcial $(x_1,...,x_k)$, $k \leq n$.

- Proceso:
  - 1º Si puede añadirse un elemento x_{k+1} la solución parcial se avanza al nivel k + 1

Introducción

- Estrategia: recorrido en profundidad
- En cada momento, el algoritmo se encontrará en un cierto nivel k, con una solución parcial $(x_1,...,x_k)$, $k \leq n$.

- Proceso:
  - 2º Si no, se prueban otros valores válidos para x_k

Introducción

- Estrategia: recorrido en profundidad
- En cada momento, el algoritmo se encontrará en un cierto nivel k, con una solución parcial $(x_1,...,x_k)$, $k \leq n$.

- Proceso:
  - 3º Si no existe ningún valor válido, se retrocede al nivel anterior (deshaciendo la solución parcial, eliminando los elementos que se hubieran añadido)

Introducción

- Proceso:
  - Se continúa con este proceso hasta que:
    - La solución parcial sea una solución del problema (solución completa)
      - Si sólo se busca una solución, el algoritmo termina
      - Si se buscan varias soluciones o la óptima, continúa hasta realizar el recorrido completo
    - No queden más posibilidades por probar, es decir, no hay solución.

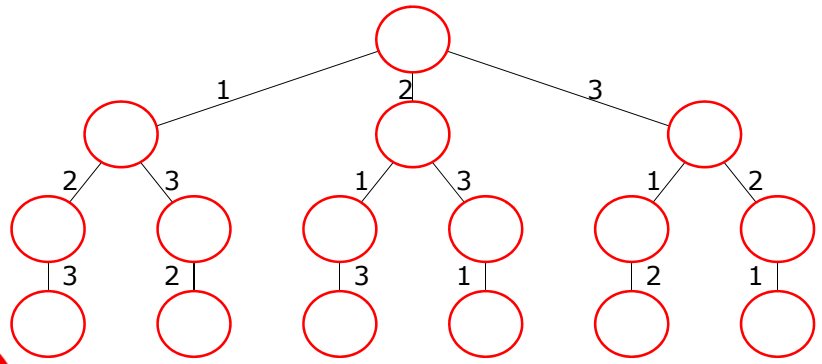
Introducción

- Complejidad
  - Los algoritmos de backtracking son bastante ineficientes debido a que realizan una búsqueda exhaustiva en el espacio de soluciones del problema
  - En general, se tienen tiempos con órdenes de complejidad factoriales o exponenciales
  - Depende del tamaño del espacio de soluciones

Árbol de búsqueda

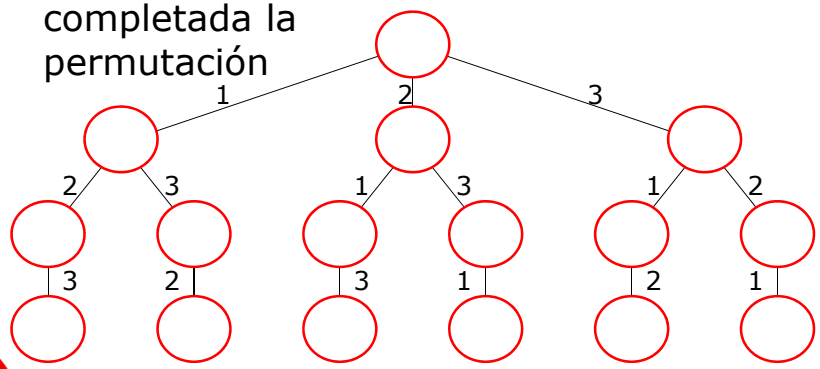
Árbol de búsqueda

- Permutaciones
  - Árbol de búsqueda generado con todas las permutaciones de un conjunto:
  - **{1,2,3}, {1,3,2}, {2,1,3}, {2,3,1}, {3,1,2}, {3,2,1}**



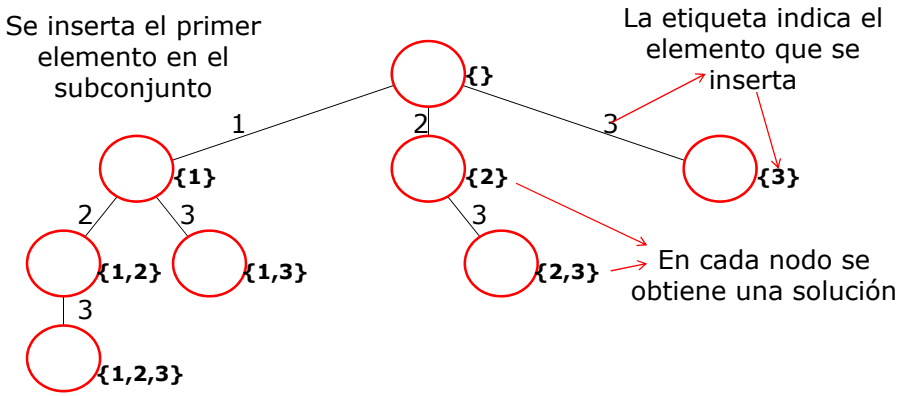
Árbol de búsqueda

- Permutaciones
  - El nivel/etapa de la llamada indica la posición del nuevo elemento a añadir
  - En el ultimo nivel (en las hojas) tenemos completada la permutación



Árbol de búsqueda

- Subconjuntos – Subconjunto en cada nodo



Soluciones:
subconjuntos de tamaño variable

Esquema general

Esquema general

```
void Backtracking ( int n , int i, Valor [ ] solucion )
{
   for (int k =0; k<n; k++)
   {
      << generar estado >>;
      if ( estado valido )
      {
         << incluirlo en solucion >>;
         if (estado es hoja )
             mostrar/guardar( solucion );
         else
             Backtracking(n, i+1, solucion );
         << borrarlo de solucion >>;
      }
   }
}
```

Explorar alternativas (candidatos)

Solución: permutación parcial construida

Si es nodo hoja, imprimo solución

Si no, bajo un nivel

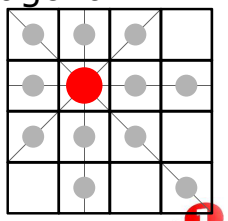
Exploraremos más caminos diferentes

N-Reinas

N-Reinas

- Dado un tablero de ajedrez de tamaño nxn, encontrar todas las formas de colocar n reinas de modo que no se amenacen:
  - No pueden estar en la misma la misma fila
  - No pueden estar en la misma columna
  - No pueden estar en la misma diagonal
- Posibles planteamientos:
  - Probar todas las formas de colocar n reinas en un tablero y para cada solución probar si es valida.

$\binom{n^2}{n}$ n=4, 1.820
n=8, $4{,}42 \cdot 10^{10}$

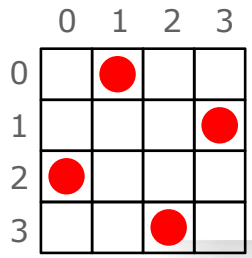


N-Reinas

- Posibles planteamientos:
  - Restricciones:
    - Sólo puede haber una reina por cada columna: Esto reduce las posibilidades a n^n (hay n formas de colocar una reina en una columna, y hay n columnas)
      - n=8 , 16.777.216
      - n=4 , 256
    - Además, no puede haber dos reinas en la misma fila: búsqueda de una permutación de n elementos con n! posibilidades (aún sigue siendo elevado)
      - n=8 , 40.320
      - n=4 , 24

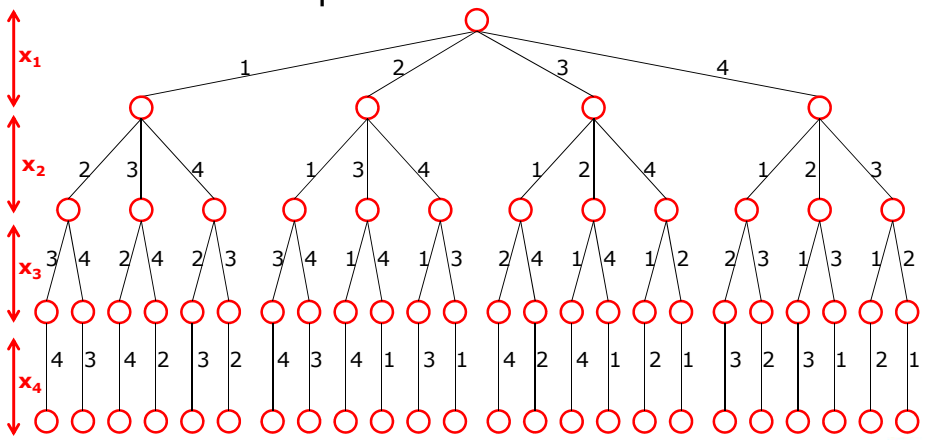
N-Reinas

- Formato de la solución:
  - Numeramos las filas y las columnas de 0 a n-1
  - Numeramos las reinas de 0 a n-1
  - Solución: $(x_0,...,x_{n-1})$
    - La reina i se coloca en la columna i y en cierta fila j (dibujo: {2,0,3,1})
  - Tendremos que buscar las permutaciones válidas.



N-Reinas

- Árbol de búsqueda n=4



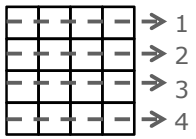
- Busquemos las soluciones válidas...

N-Reinas

- Árbol de búsqueda n=4
  - Empleamos el algoritmo para buscar permutaciones, dos variantes:
    - Se comprueban las restricciones al llegar a una hoja
      - Obliga a generar todo el árbol para ver si la solución es valida
      - Costes factoriales o exponenciales
    - Se puede comprobar si la solución parcial es prometedora (si puede llegar a ser solución final), antes de llegar a una hoja
      - Si la solución parcial no va a poder formar parte de la solución final, no se recorre
      - Podamos el árbol ahorrando cálculos

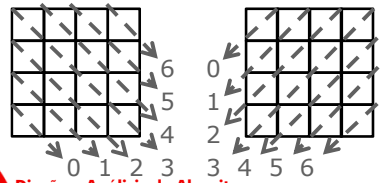
N-Reinas

- Implementación
  - Para verificar que una solución parcial es válida usamos:
    - 1º: un vector f (longitud n) que sirve para comprobar que dos reinas no ocupen la misma fila



N-Reinas

- Implementación
  - Para verificar que una solución parcial es válida usamos:
    - 2º: vector de 2n-1 posiciones para comprobar que dos reinas no ocupan la misma diagonal principal o secundaria:
      - dp: diagonales principales libres (índice del vector=columna-fila+n-1)
      - ds: diagonales secundarias libres (índice del vector=columna+fila)

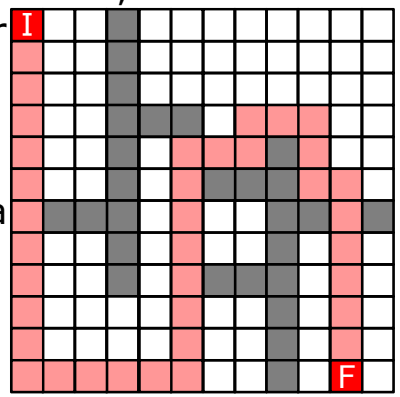


dp	T	T	T	T	T	T	T
ds	T	T	T	T	T	T	T

Otros problemas

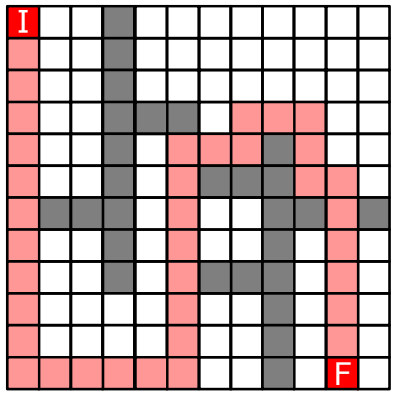
Otros problemas: laberinto

- Encontrar un camino desde una posición inicial hasta una posición final
  - En función del estado actual, ir a un nuevo estado para encontrar el camino solución, compuesto de pasos
  - No repetir pasos (ubicarse en la misma celda está prohibido)
  - Pasos verticales u horizontales



Otros problemas: laberinto

- Encontrar un camino desde una posición inicial hasta una posición final
  - Estado inicial:
    Celda: (0,0)
    Posibles caminos siguientes:
    (0,1), (1,0)



Otros problemas: mochila

- Llenar la mochila con objetos
  - Hacemos un recorrido por los objetos para ver cuál introducir en cada ocasión
  - Podemos insertar siempre aquellos objetos que quepan en la mochila
  - Buscaremos los que maximicen el beneficio
  - Si hay límite de unidades, no podremos sobrepasarlo en ninguna solución

6. Backtracking

Antonio Pérez Carrasco

Universidad Rey Juan Carlos

DLSI 1

http://www.dlsi1.etsii.urjc.es

Diseño y Análisis de Algoritmos