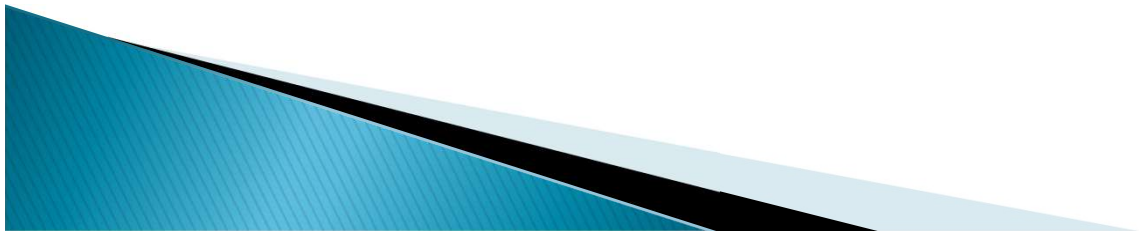
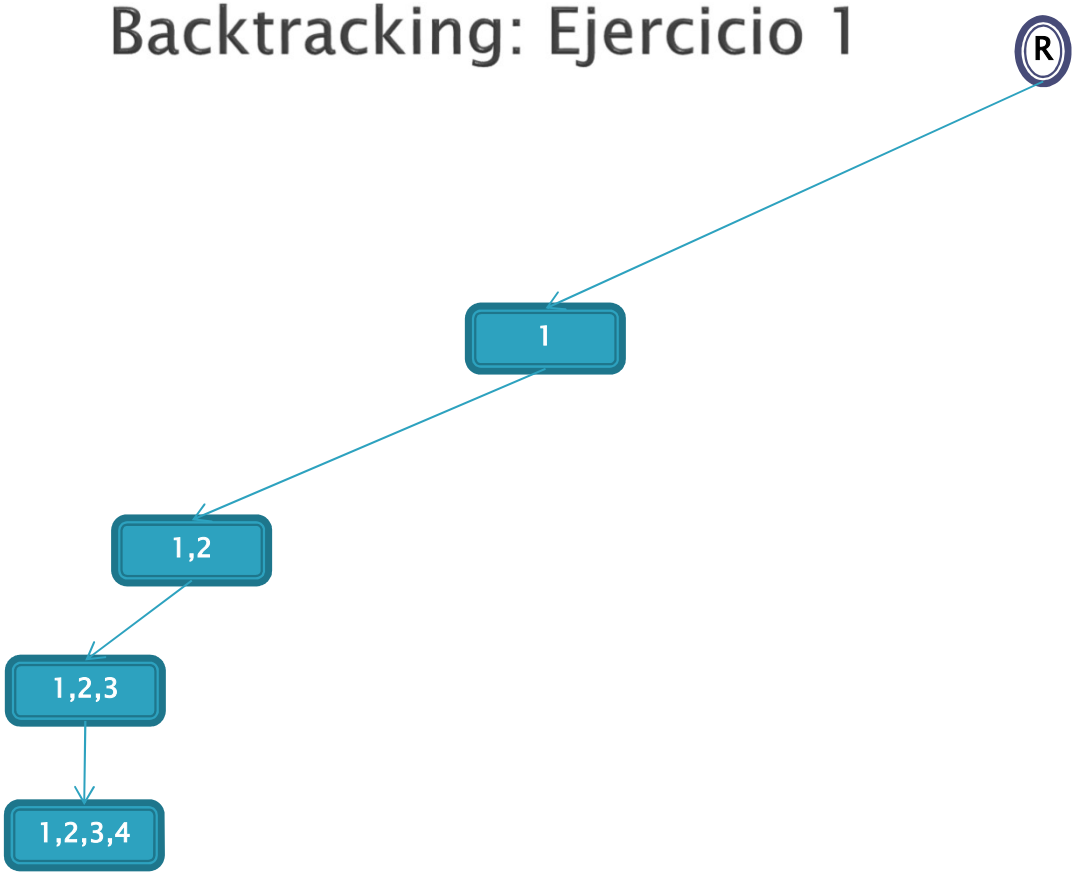
Backtracking: Ejercicio 1

Se tienen `N` elementos distintos almacenados en una estructura de acceso directo (por ejemplo, un vector con los números `1`, `2`, `3`, `4` y `5`, o la cadena `abcdefg`) y se quiere obtener todas las formas distintas de colocar esos elementos, es decir, hay que conseguir todas las permutaciones de los `N` elementos. Diseñar un algoritmo que use Backtracking para resolver el problema.

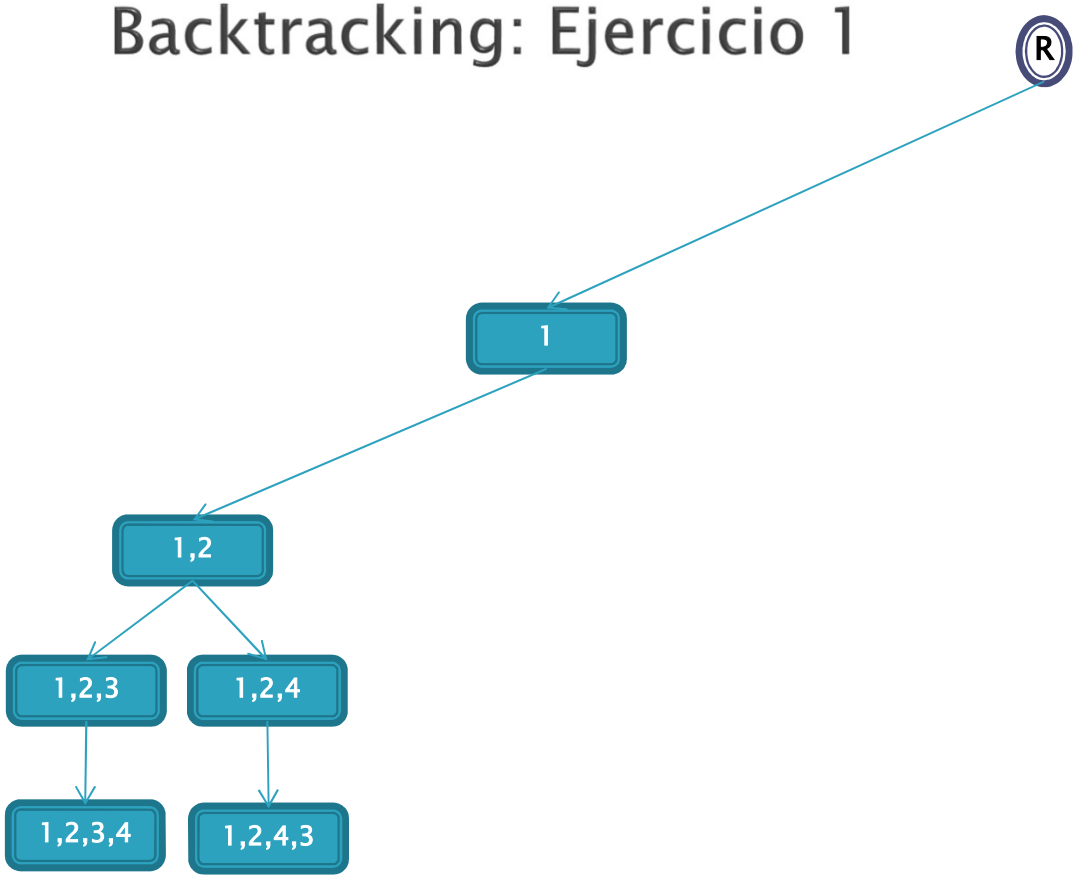


Backtracking: Ejercicio 1



Entrada={1,2,3,4}

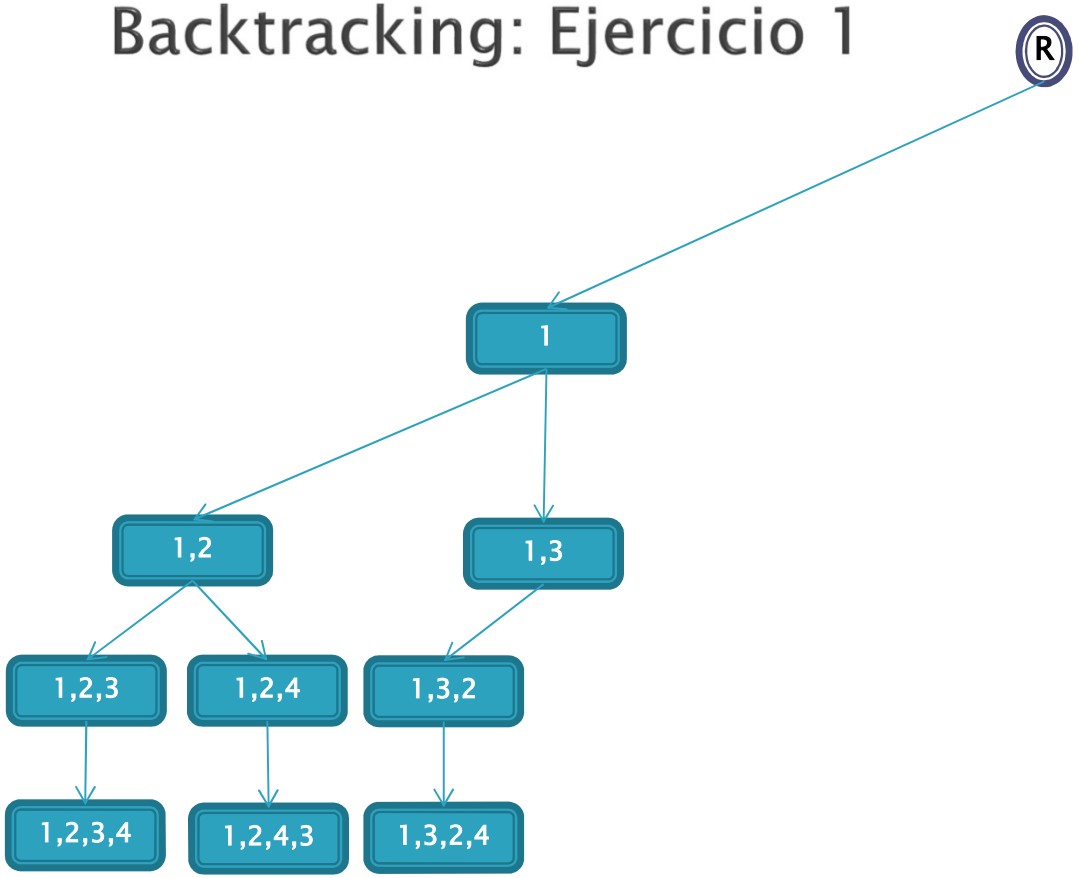
Backtracking: Ejercicio 1



Entrada={1,2,3,4}

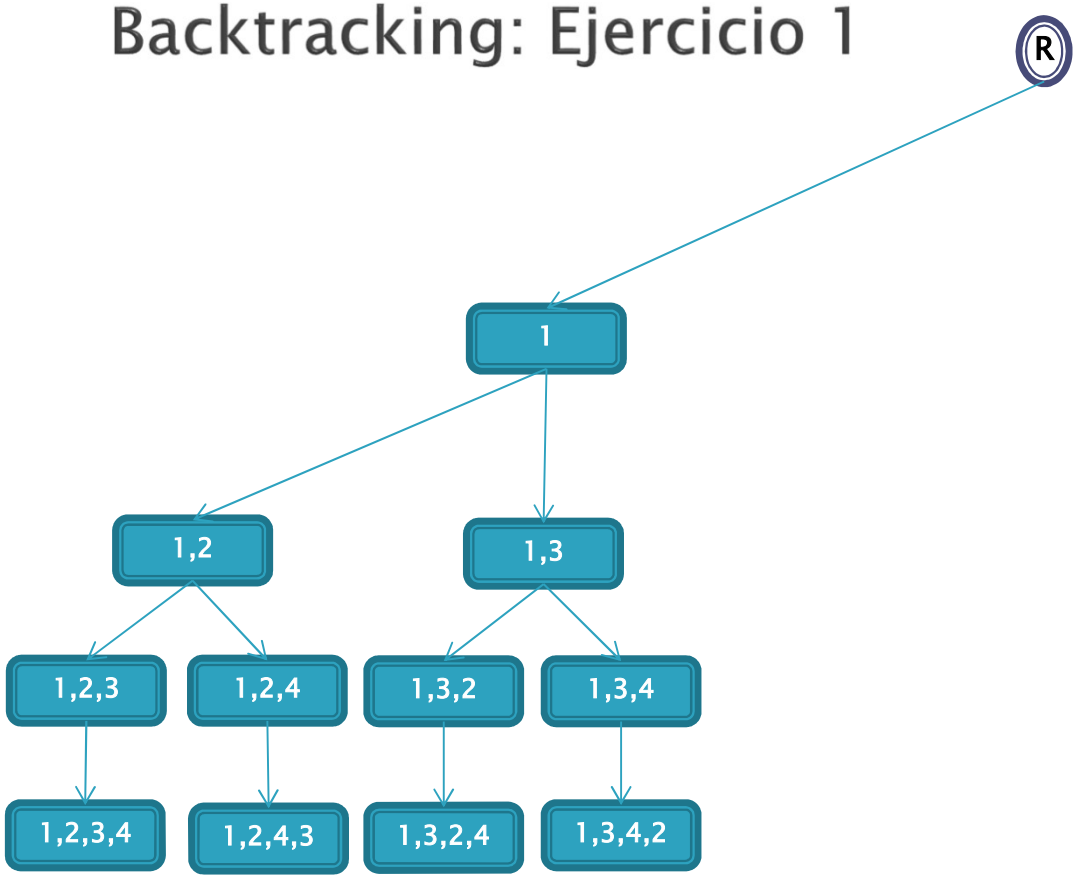
Backtracking: Ejercicio 1

Entrada={1,2,3,4}



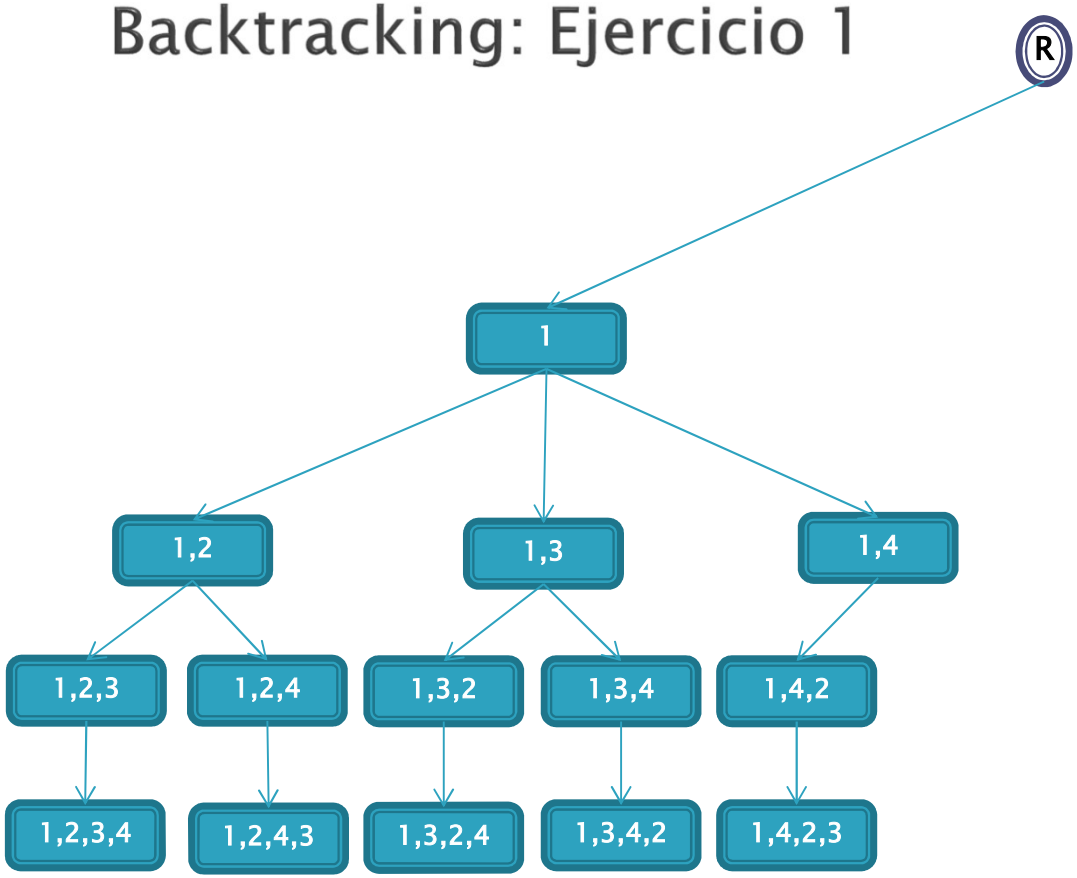
Backtracking: Ejercicio 1

Entrada={1,2,3,4}

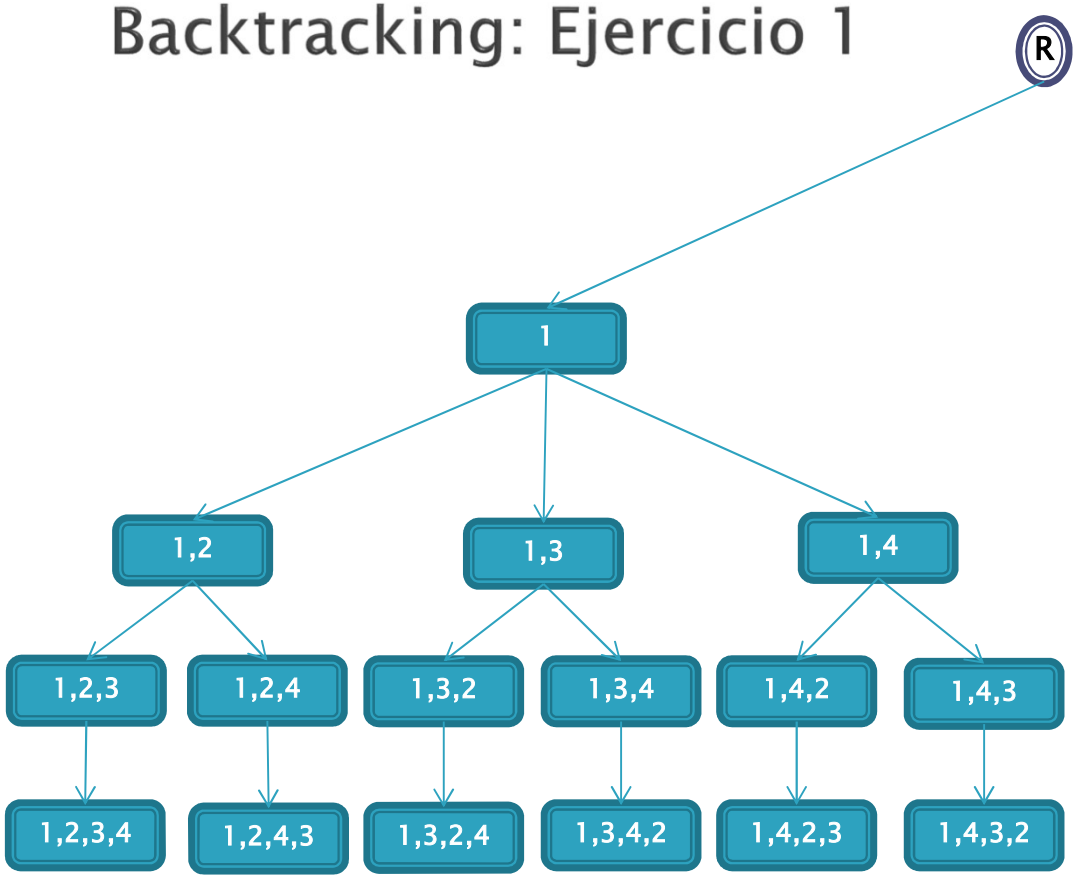


Backtracking: Ejercicio 1

Entrada={1,2,3,4}



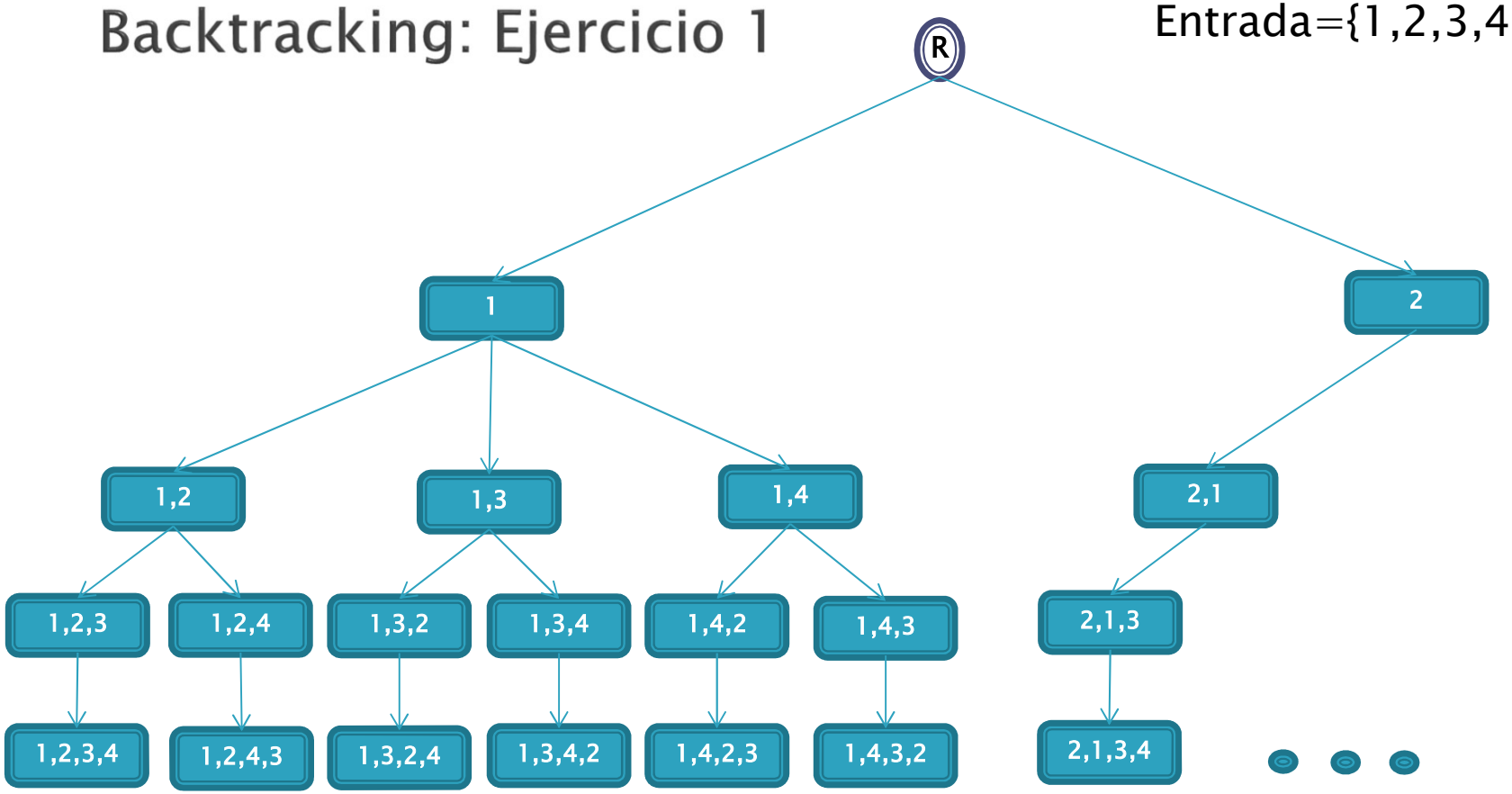
Backtracking: Ejercicio 1



Entrada={1,2,3,4}

Backtracking: Ejercicio 1

Entrada={1,2,3,4}

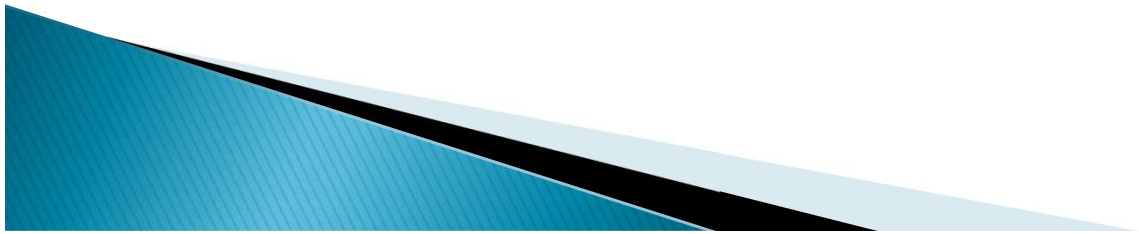


Backtracking: Ejercicio 1

Entrada = [1, 2, 3, 4]									
Sol.	S[1]	S[2]	S[3]	S[4]	Sol.	S[1]	S[2]	S[3]	S[4]
1	1	2	3	4	13	3	1	2	4
2	1	2	4	3	14	3	1	4	2
3	1	3	2	4	15	3	2	1	4
4	1	3	4	2	16	3	2	4	1
5	1	4	2	3	17	3	4	1	2
6	1	4	3	2	18	3	4	2	1
7	2	1	3	4	19	4	1	2	3
8	2	1	4	3	20	4	1	3	2
9	2	3	1	4	21	4	2	1	3
10	2	3	4	1	22	4	2	3	1
11	2	4	1	3	23	4	3	1	2
12	2	4	3	1	24	4	3	2	1

Backtracking: Ejercicio 1

- Entradas [1…N]: el vector de elementos a permutar.
- Válidos [1…N]: indica si se puede tener en cuenta un elemento o no.
- Salida [1…N]: salida válida, generada progresivamente.
- Todas las llamadas de una rama parten de idénticas condiciones.
- Hay que deshacer los cambios después de cada llamada.



Backtracking: Ejercicio 1

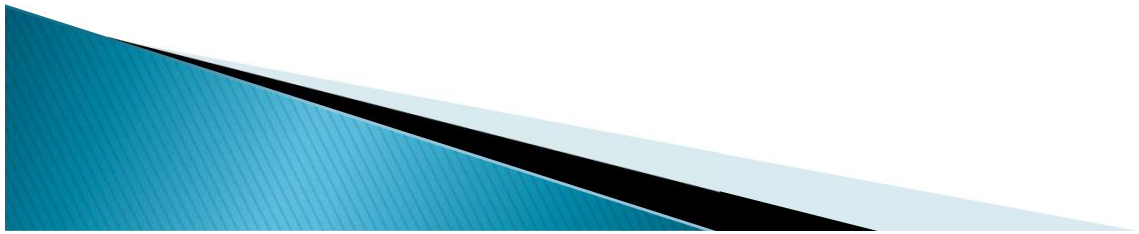
```
const N = ...
tipos bool = array[1... N] de booleano
tipos int = array[1... N] de entero


proc Inicializar (E/S validos: bool)
   desde i=1 hasta N hacer validos[i] = true fdesde
fproc

proc Ejercicio1 (E/S entrada, salida :int ; E/S validos :bool, E k:entero)
   var i : entero
   si k > N entonces
      Escribir salida //Si k (Profundidad) > Total Números se muestra resultado
   sino
      desde i=1 hasta N hacer
         si validos[i] entonces //Si el número es candidato
            validos[i] = false //Candidato a false
            salida[k] = entrada[i]//Se coge el valor como resultado parcial
            Ejercicio1(entrada, salida, validos, k+1) //Se baja en profundidad
            validos[i] = true //Se restauran los datos según profundidad
         fsi
      fdesde
   fsi
Fproc
```

PROGRAMA PRINCIPAL

Inicializar (validos)
Ejercicio1 (entrada, salida, validos, 1)

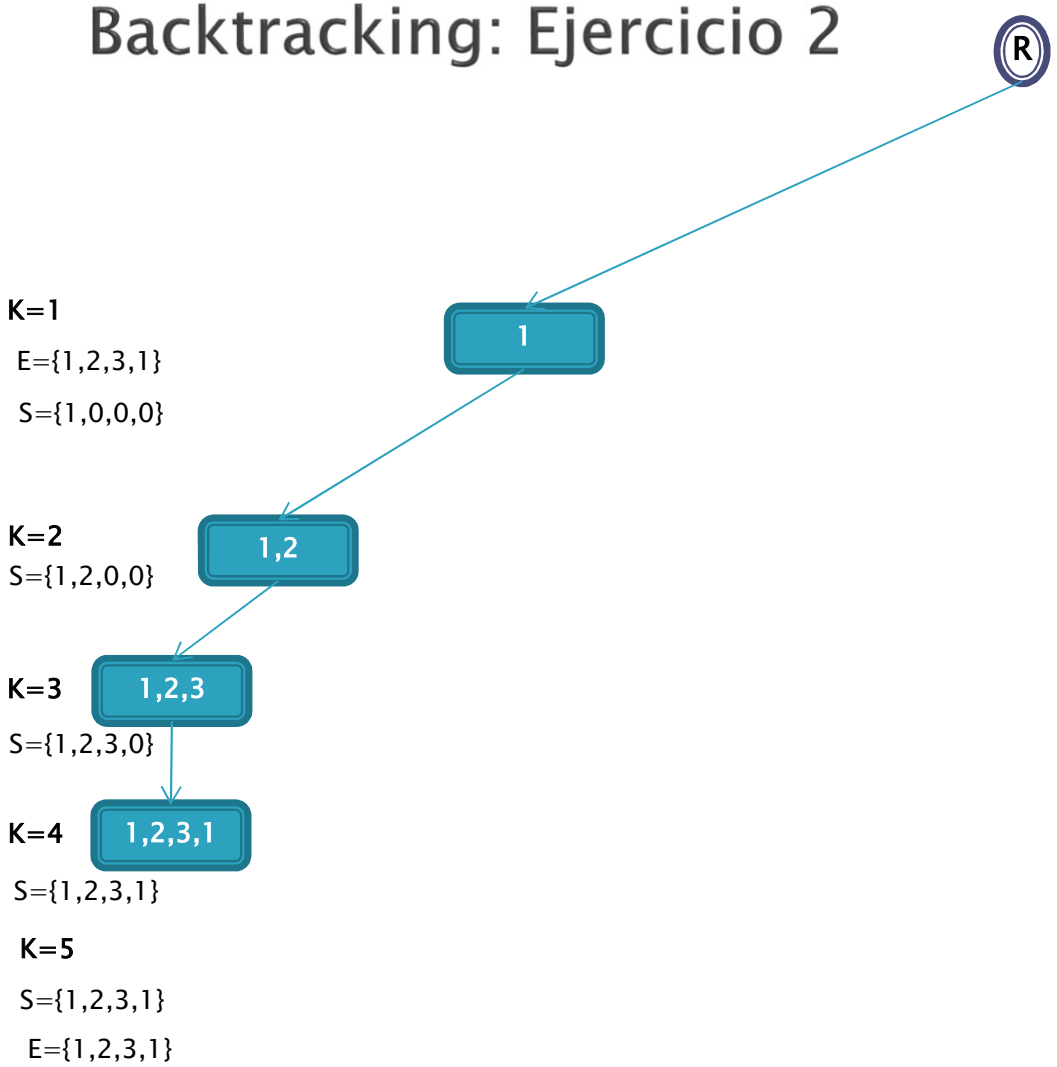
Backtracking: Ejercicio 2

Resolver el problema anterior considerando la posibilidad de que los elementos se repitan entre sí (por ejemplo, el vector `1, 2, 3, 1` o la cadena `acabada`).



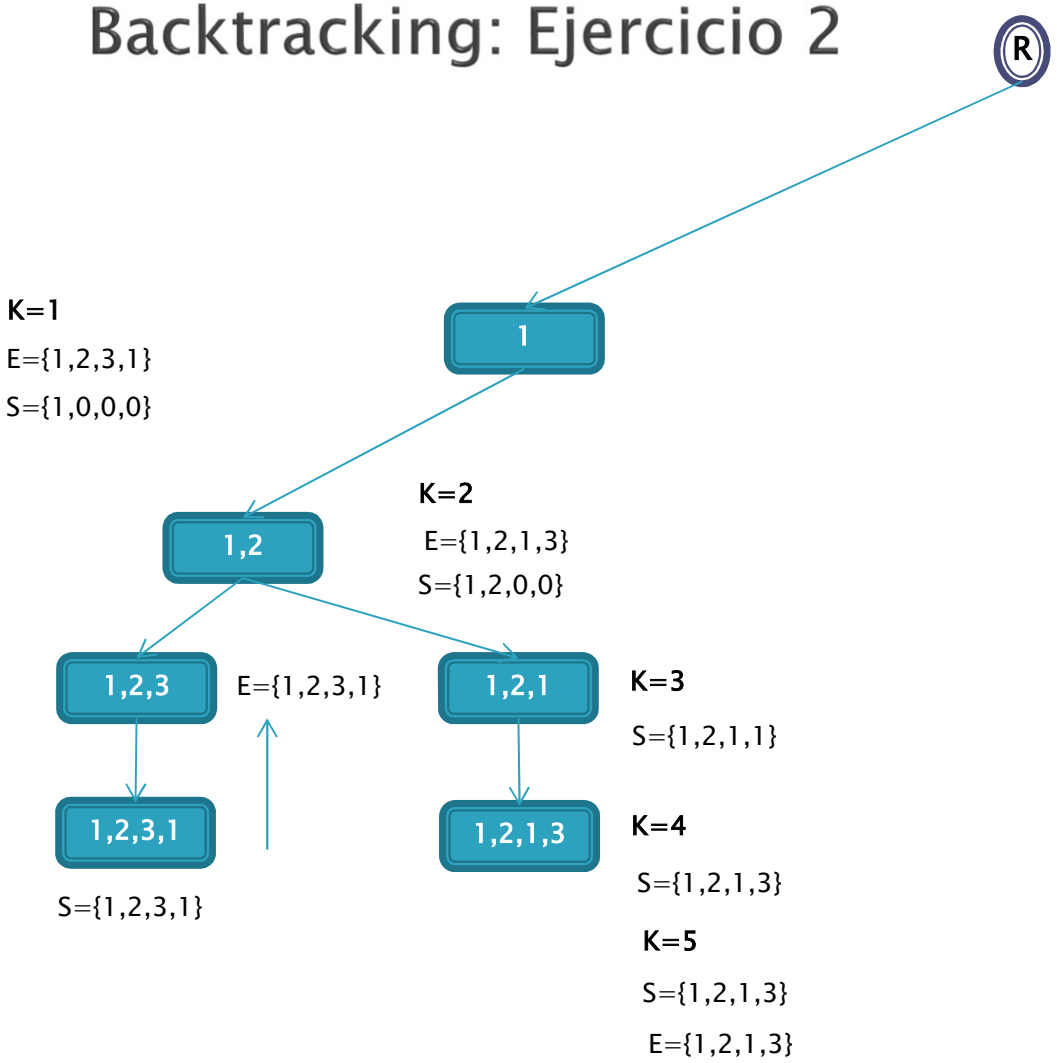
Backtracking: Ejercicio 2

Entrada={1,2,3,1}



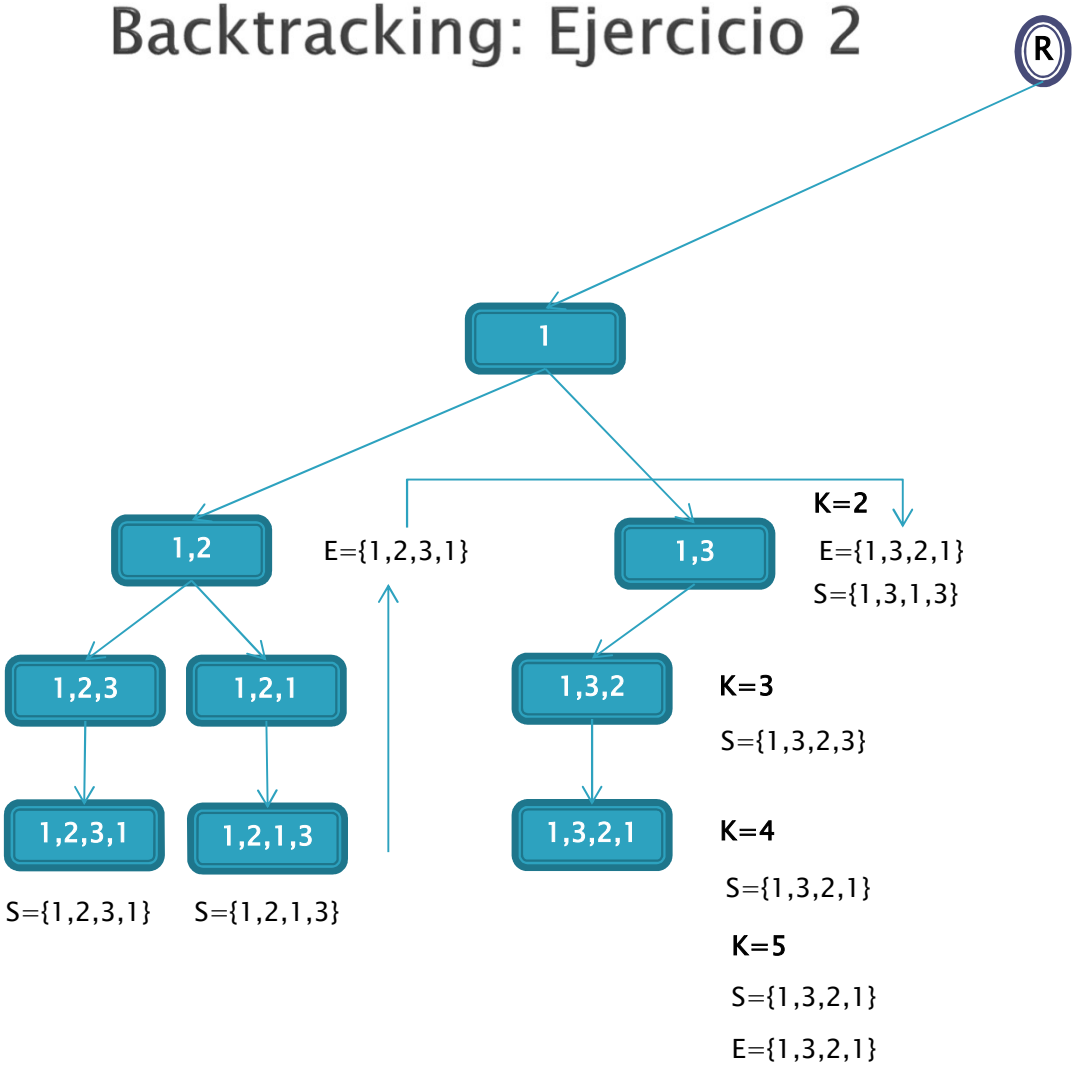
Backtracking: Ejercicio 2

Entrada={1,2,3,1}



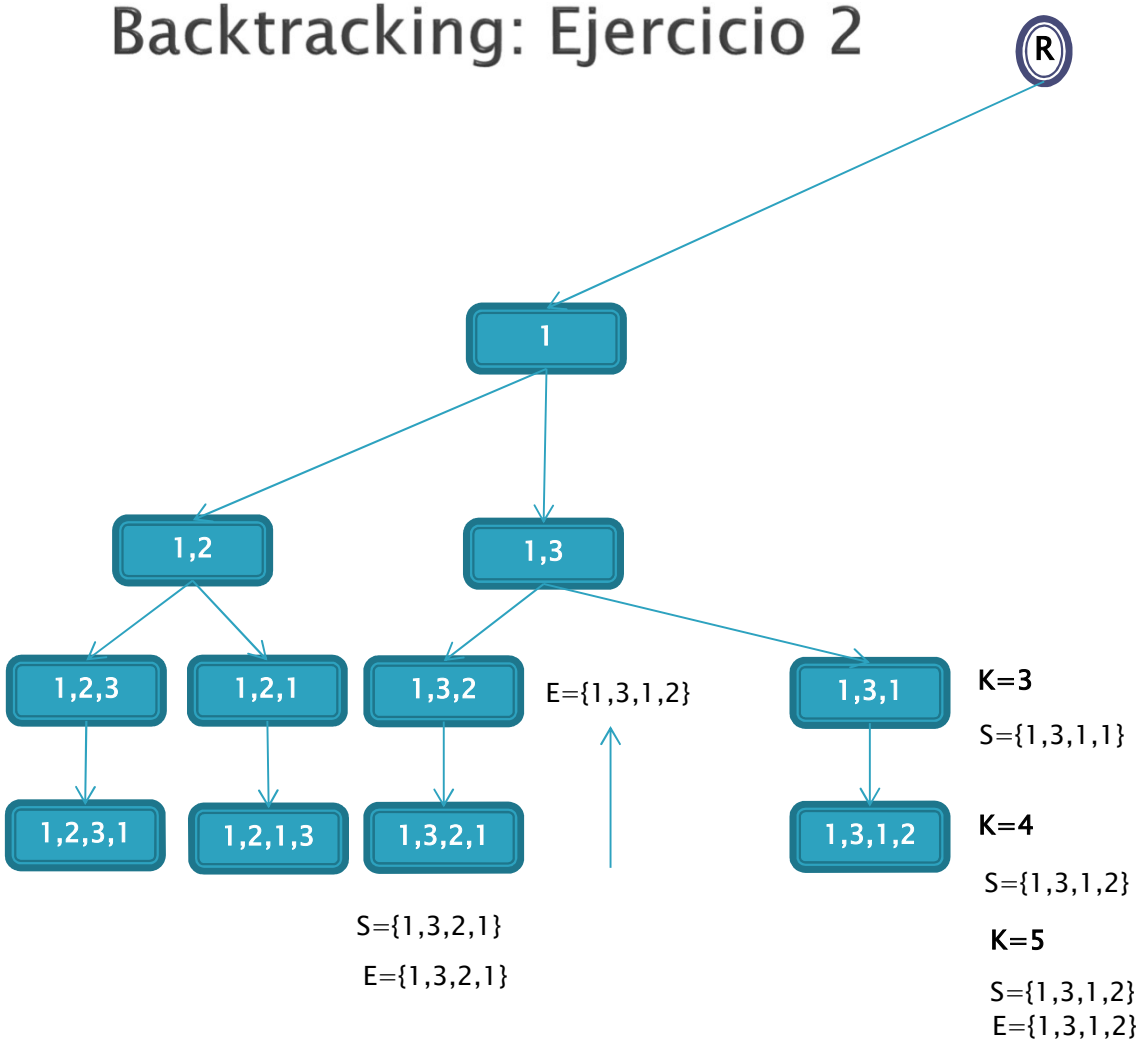
Backtracking: Ejercicio 2

Entrada={1,2,3,1}



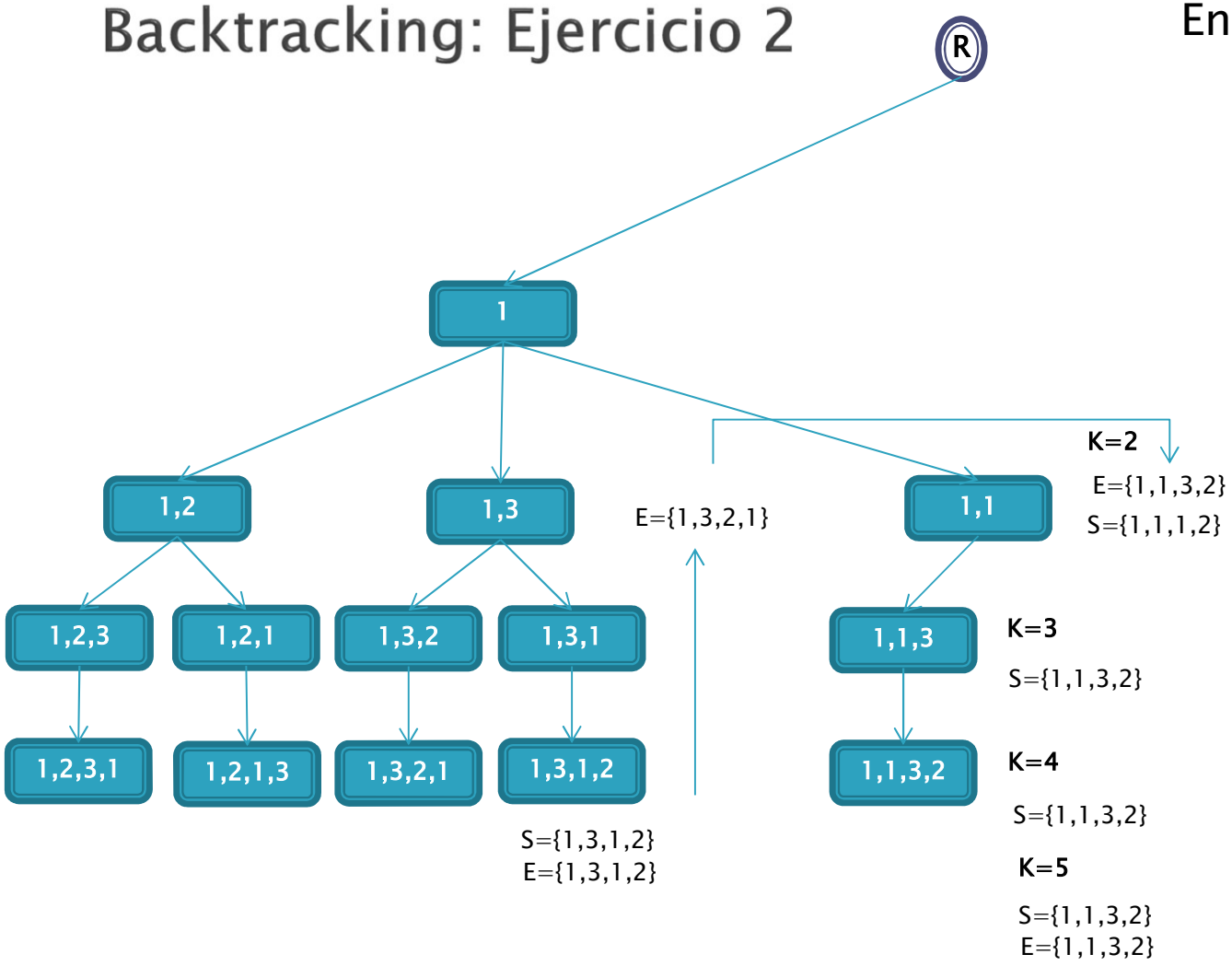
Backtracking: Ejercicio 2

Entrada={1,2,3,1}

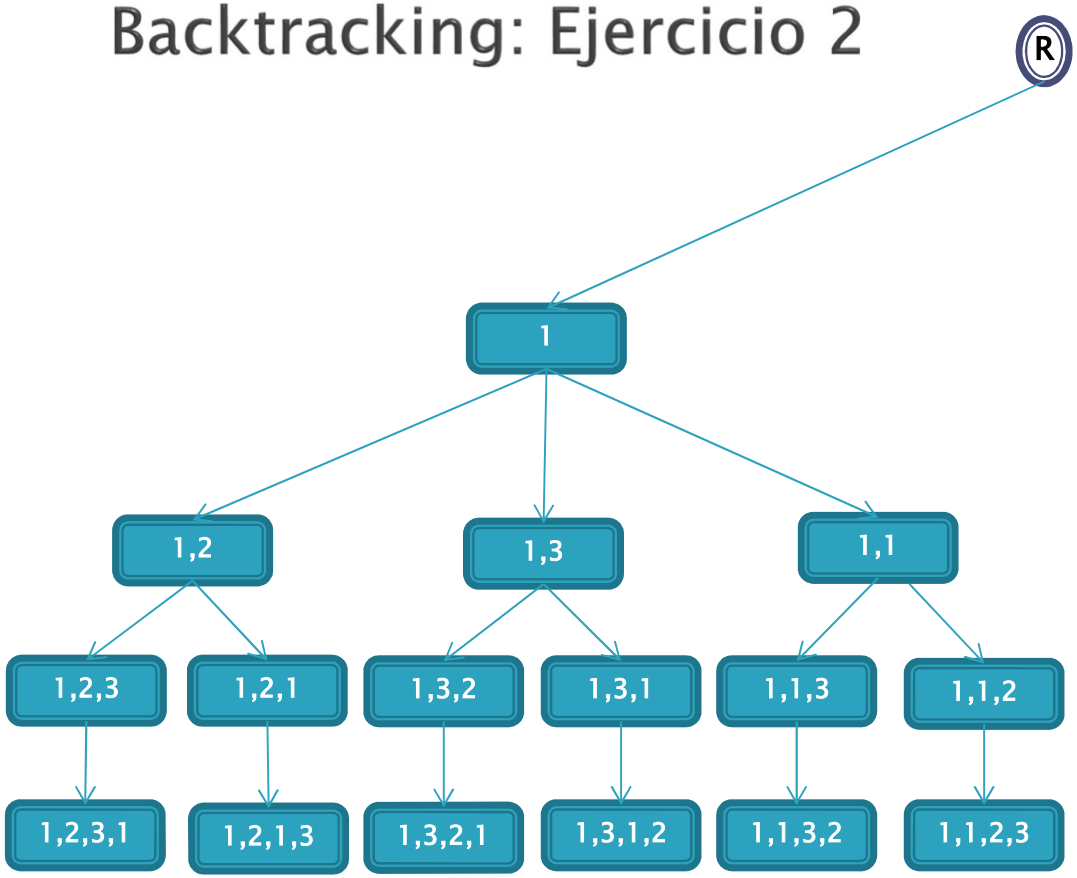


Backtracking: Ejercicio 2

Entrada={1,2,3,1}

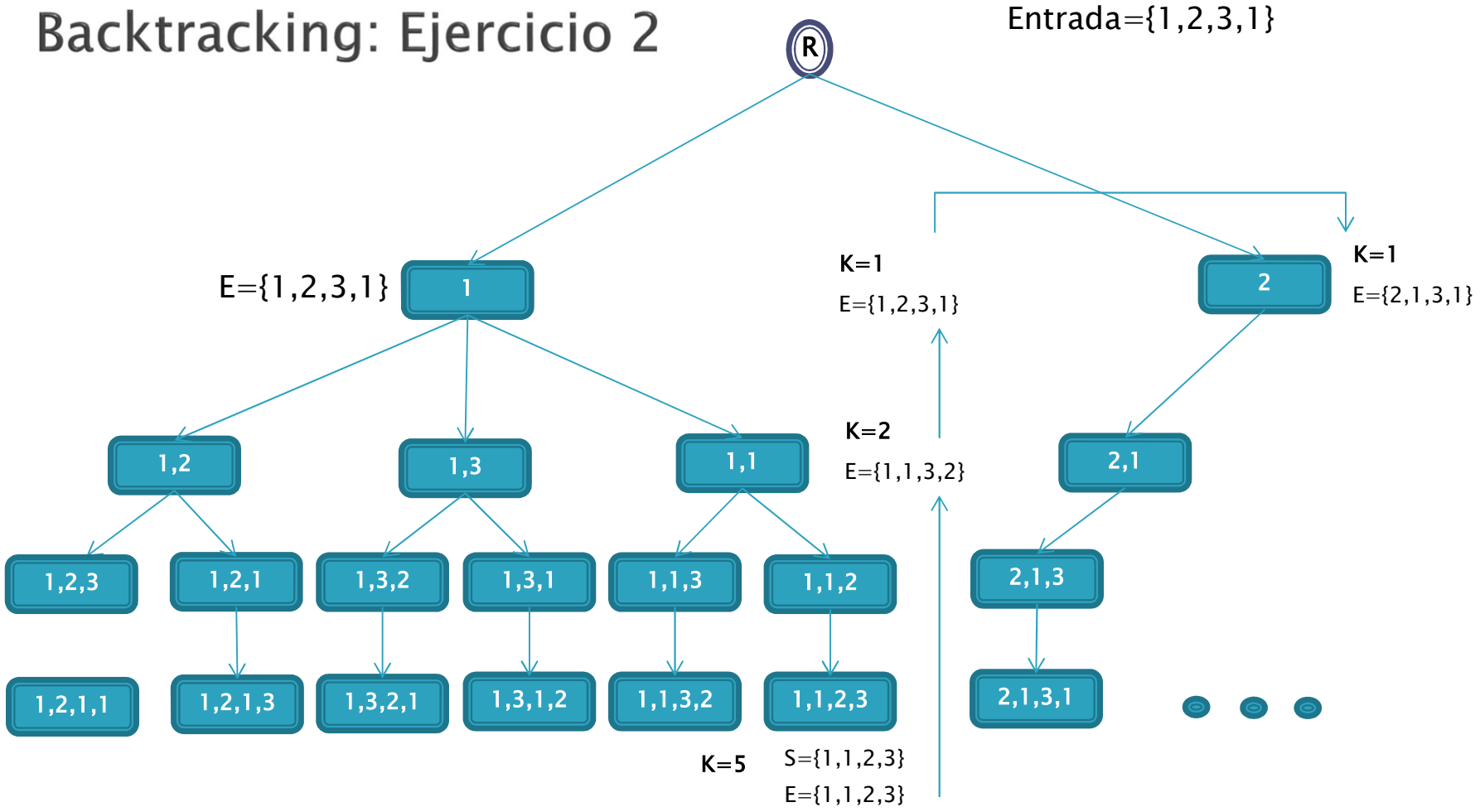


Backtracking: Ejercicio 2



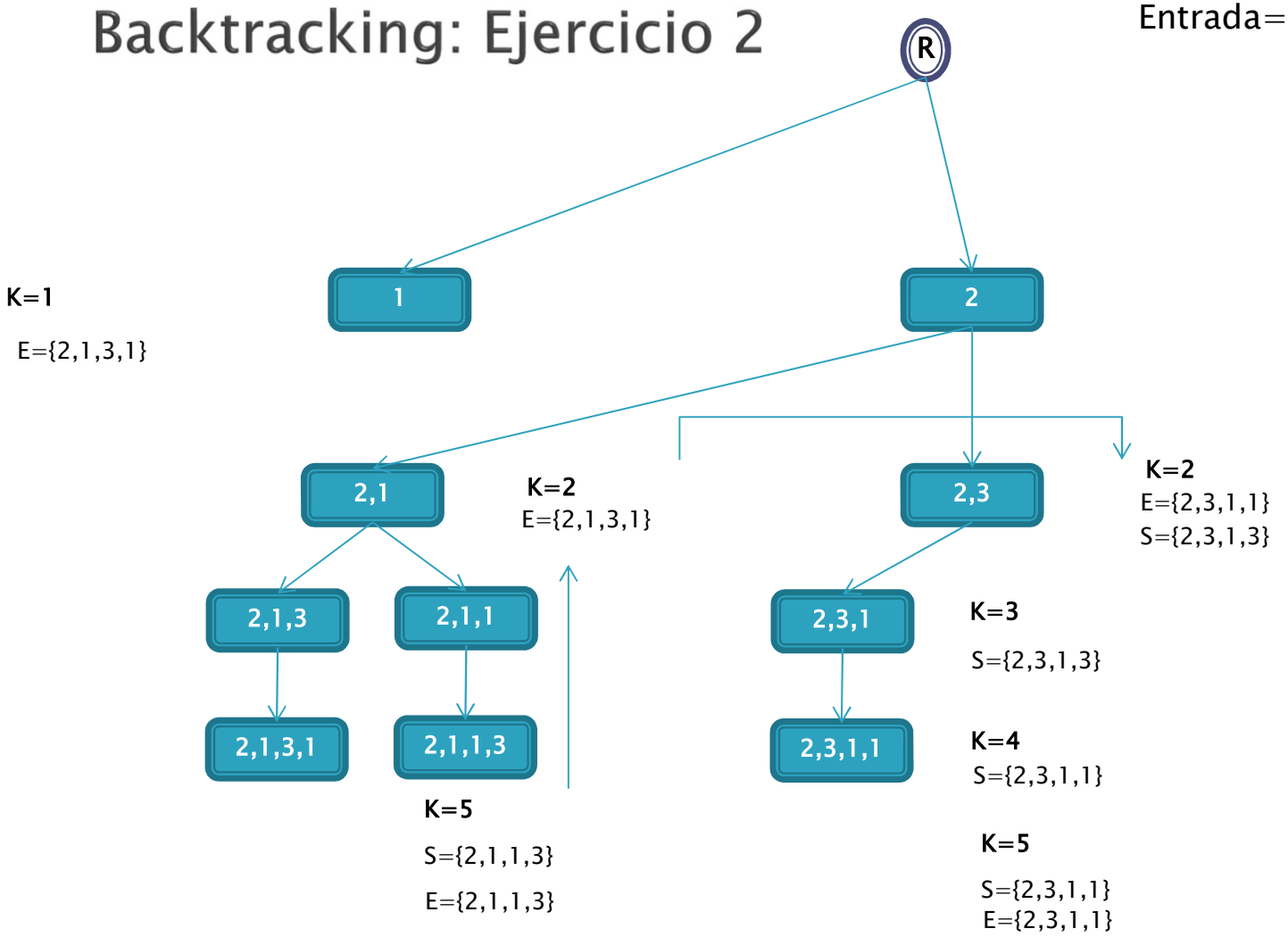
Entrada={1,2,3,1}

Backtracking: Ejercicio 2



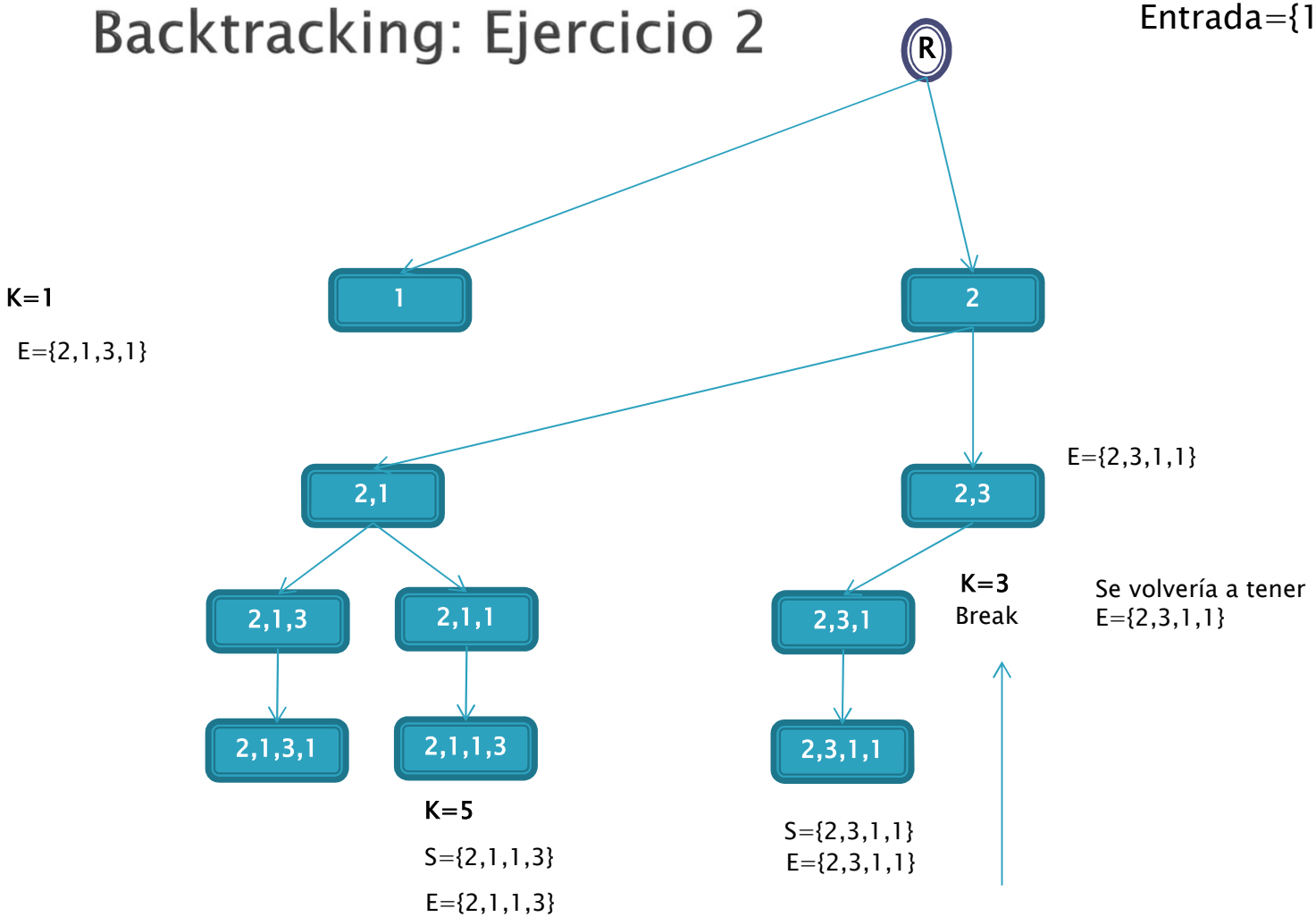
Backtracking: Ejercicio 2

Entrada={1,2,3,1}



Backtracking: Ejercicio 2

Entrada={1,2,3,1}



Backtracking: Ejercicio 2

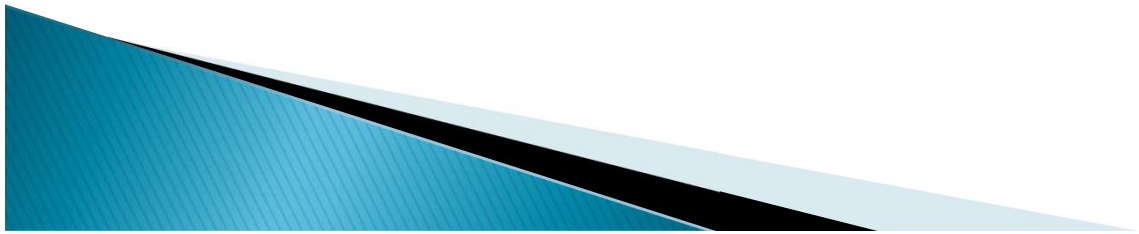
Entradas= [1, 2, 3, 1]									
Sol.	S[1]	S[2]	S[3]	S[4]	Sol.	S[1]	S[2]	S[3]	S[4]
1	1	2	3	1	13	3	1	2	1
2	1	2	1	3	14	3	1	1	2
3	1	3	2	1	~~15~~	~~3~~	~~2~~	~~1~~	~~1~~
4	1	3	1	2	16	3	2	1	1
6	1	1	2	3	~~17~~	~~3~~	~~1~~	~~1~~	~~2~~
5	1	1	3	2	~~18~~	~~3~~	~~1~~	~~2~~	~~1~~
7	2	1	3	1	~~19~~	~~1~~	~~1~~	~~2~~	~~3~~
8	2	1	1	3	~~20~~	~~1~~	~~1~~	~~3~~	~~2~~
9	2	3	1	1	~~21~~	~~1~~	~~2~~	~~1~~	~~3~~
~~10~~	~~2~~	~~3~~	~~1~~	~~1~~	~~22~~	~~1~~	~~2~~	~~3~~	~~1~~
~~11~~	~~2~~	~~1~~	~~1~~	~~3~~	~~23~~	~~1~~	~~3~~	~~1~~	~~2~~
~~12~~	~~2~~	~~1~~	~~3~~	~~1~~	~~24~~	~~1~~	~~3~~	~~2~~	~~1~~

Backtracking: Ejercicio 2

Entradas = [1, 2, 3, 1]				
Sol.	S[1]	S[2]	S[3]	S[4]
1	1	2	3	1
2	1	2	1	3
3	1	3	2	1
4	1	3	1	2
5	1	1	3	2
6	1	1	2	3
7	2	1	3	1
8	2	1	1	3
9	2	3	1	1
10	3	2	1	1
11	3	1	2	1
12	3	1	1	2

Backtracking: Ejercicio 2

- Entradas [1…N]: el vector de elementos a permutar.
- Salida [1…N]: salida válida, generada progresivamente.
- Todas las llamadas de una rama parten de idénticas condiciones.
- Se intercambian las entradas para evitar soluciones repetidas.
- Hay que deshacer los cambios después de cada llamada.



Backtracking: Ejercicio 2

PROGRAMA PRINCIPAL

Ejercicio2 (entrada, salida, 1)

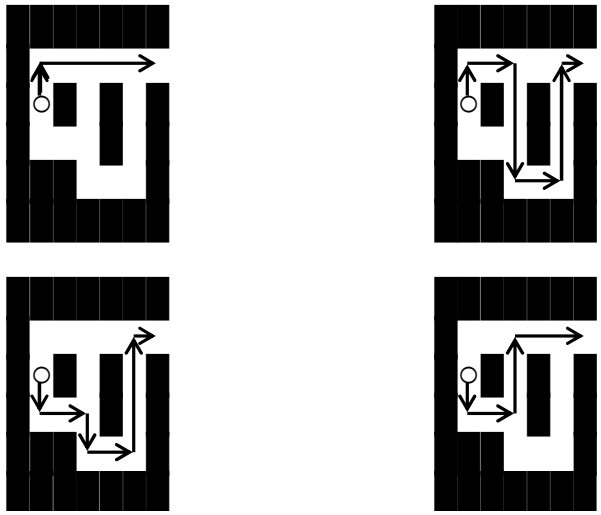
```
const N = ...
tipos int = array[1... N] de entero

proc Intercambiar(E entrada:in, E i,j:entero)
   var a: entero
   a=entrada[i]; entrada[i]=entrada[j]; entrada[j]=entrada[i];
fproc

proc Ejercicio2 (E entrada, salida: E k: entero)
   var i, j, saltar_iteracion: entero
   si k>N entonces Escribir salida //Si k(Profundidad)>Total Números mostrar resultado

   sino
      desde i=k hasta N hacer
         saltar_iteracion=0
         desde j=k hasta i-1 hacer //Si ya está repetido salgo
             si  entrada[i]=entrada[j]entonces saltar_iteracion = 1; salir desde
             fsi
         fdesde
         si  saltar_iteracion = 0 entonces
             salida[k]=entradas[i] //Se toma como resultado parcial
             Intercambiar(entrada,i,k) //Se intercambian los no seleccionados
             Ejercicio2(entrada, salida, k+1)
             Intercambiar (entrada, k, i)//Se restauran los datos originales
         fsi
      fdesde
   fsi
Fproc
```

Backtracking: Ejercicio 5

Se dispone de una tabla laberinto[1..n,1..m] con valores lógicos que representa un laberinto.

El valor TRUE indica la existencia de una pared (no se puede atravesar), mientras que FALSE representa una casilla recorrible.

Para moverse por el laberinto, a partir de una casilla se puede desplazar horizontal o verticalmente, pero solo a una casilla vacía (FALSE). Los bordes de la tabla están completamente a TRUE excepto una casilla, que es la salida del laberinto. Diseñar un algoritmo Backtracking que encuentre todos los caminos posibles que llevan a la salida desde una casilla inicial determinada, si es posible salir del laberinto.

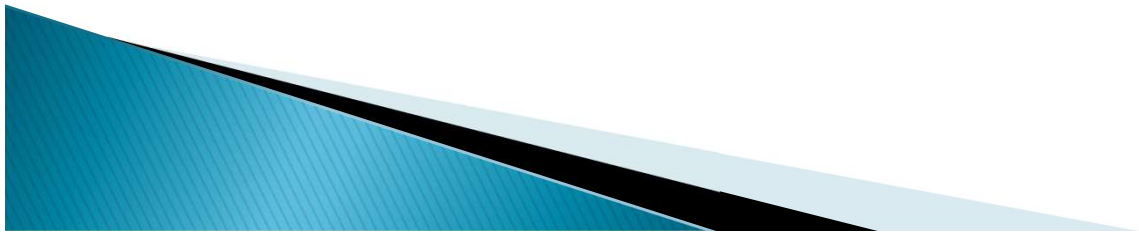


Backtracking: Ejercicio 5

- Estrategia:
  1. Comprobar si la casilla donde está actualmente es la salida, si es así, sale de la función indicando que se ha encontrado la solución; de lo contrario pasa al siguiente paso.
  2. Si es posible, visitar las casilla de izquierda llamando recursivamente a la misma función. Verificar paso 1.
  3. Si no es posible visitar o hallar la salida por paso 2 buscar salida con la casilla de arriba. Verificar paso 1.
  4. Si no es posible visitar o hallar la salida por paso 3 buscar salida con la casilla de la derecha. Verificar paso 1.
  5. Si no es posible visitar o hallar la salida por paso 4 buscar salida con la casilla de abajo. Verificar paso 1.
  6. Si no se encontró la salida, salir de la función indicando que no existe salida del laberinto por la casilla actual.

Backtracking: Ejercicio 5

- Laberinto[F][C]. Matriz que almacena la estructura del laberinto.
  - "#" → Pared del laberinto → No se puede acceder a esta casilla.
  - "O" → Punto de inicio.
  - "S" → Salida.
  - "." → Camino recorrido



Backtracking: Ejercicio 5

```
Proc recorrer(E laberinto[F][C]:Matriz de Carácter, E i:entero, E j:entero){
      //Se comprueba si es solución
      si(laberinto[i][j]=='S')entonces
            imprimir(laberinto);
            laberinto[i][j]='S';
            existeSolucion=true;
            return;
      fSi
      laberinto[i][j]='.';//Se marca el camino

      //izquierda o salida
      si(i-1>=0 && i-1<F &&(laberinto[i-1][j]==' '||laberinto[i-1][j]=='S'))entonces
            recorrer(laberinto, i-1, j);
      //arriba o salida
      si(j+1>=0 && j+1<C && (laberinto[i][j+1]==' ' || laberinto[i][j+1]=='S'))entonces
            recorrer(laberinto, i, j+1);
      //derecha o salida
      si(i+1>=0 && i+1<F && (laberinto[i+1][j]==' ' || laberinto[i+1][j]=='S'))entonces
            recorrer(laberinto, i+1, j);
      //abajo o salida
      si(j-1>=0 && j-1<C && (laberinto[i][j-1]==' ' || laberinto[i][j-1]=='S'))entonces
            recorrer(laberinto, i, j-1);
      laberinto[i][j]=' ';//Se desmarca el camino
fProc
```

PROGRAMA PRINCIPAL

recorrer(laberinto, inicio_x, inicio_y);

Backtracking: Ejercicio 5

Solución