ejBloque3VA-Sept-18.pdf

realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Bactracking y Ramificación y Poda

Dado un número positivo N queremos construir un array de longitud $2*N$ tal que todos los elementos entre 1 y N se encuentran en el array dos veces y que el número de elementos que aparece entre esas dos apariciones es exactamente igual al elemento en cuestión. Veamos un ejemplo:
Input: $N = 3$
Output:
3 1 2 1 3 2
2 3 1 2 1 3

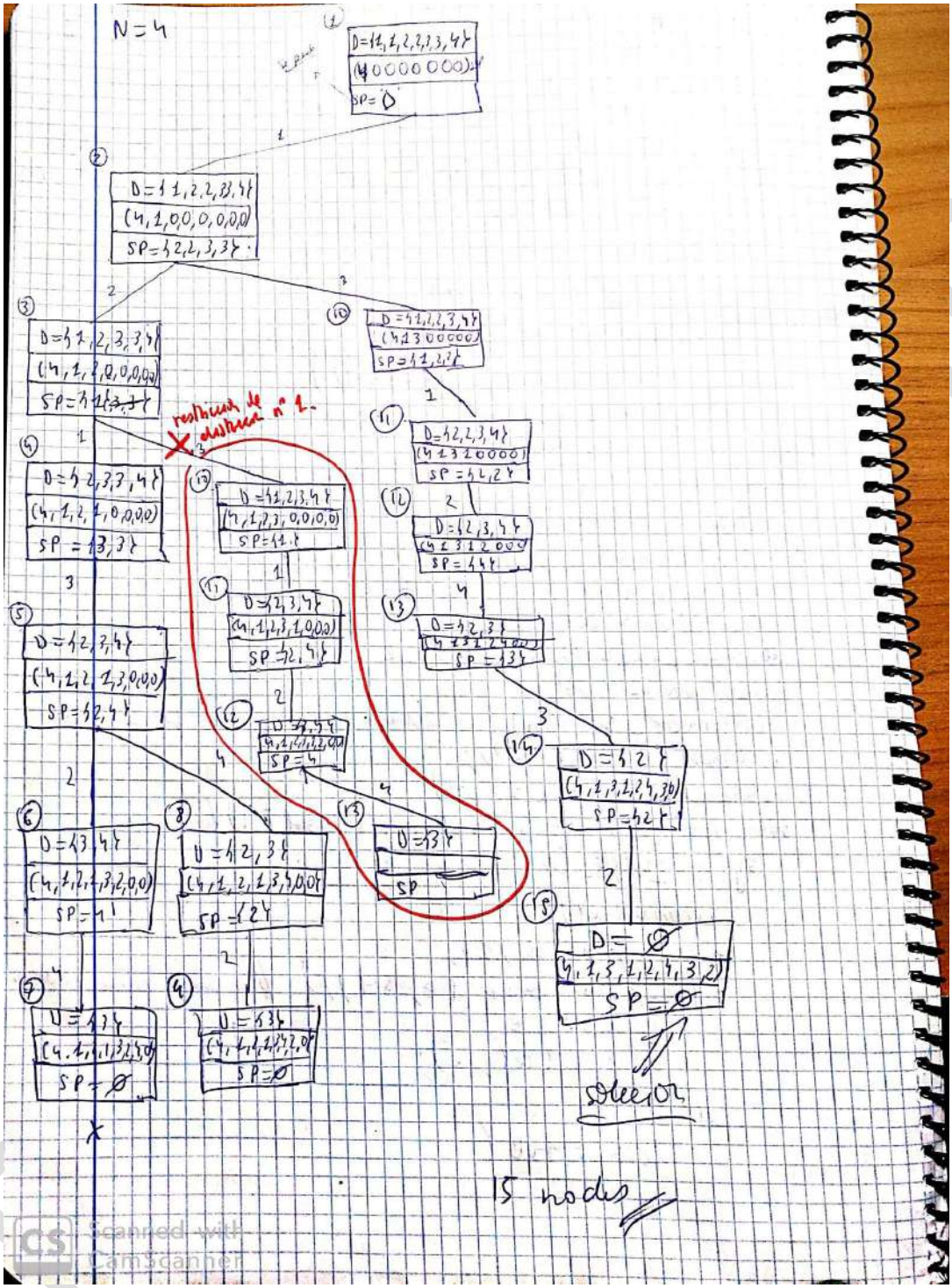
1. [6p] Diseñar un algoritmo de Backtracking para resolver este problema.
2. [4p] Construir el árbol de expansión para encontrar UNA solución para N=4.

Sept 2018

Bloque III : Backtracking y Poda.

$N > 0$

→ La estructura solución será un array de dimensión $2*N$, que empezaremos rellenando con el número N, y cada N niveles del árbol introducimos el valor N. En cada nodo, actualizaremos el conjunto SP, donde estén los números que se pueden poner en esa posición del array. Ramificamos por los elementos del conjunto S, que van a estar ordenados de menor a mayor. Si en alguna posición S es vacío y quedan elementos aún por poner retrocedemos. Llegamos a una solución final cuando $S = \emptyset$ y N = Números disponibles $= \emptyset$.



Control4-17-18-Resuelto.pdf

realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Alumno: _____ **Grado:** _____
Grupo: _____

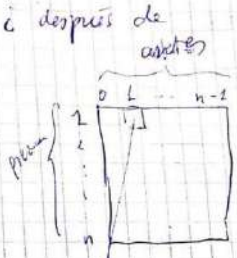
Se desea ensamblar n piezas de un vehículo de forma que el orden de ensamblaje de las mismas en las que se realiza influye en los costes debido a los movimientos que se ve obligado a realizar el brazo robótico. Para ello se tiene una matriz cuadrada C que nos proporciona los costes de ensamblaje, de forma que C_{ij} es el coste de ensamblar la pieza i después de haber ensamblado ya j piezas con $1 \leq i \leq n$ y $0 \leq j \leq n-1$. El objetivo de este problema es minimizar el coste de ensamblado de todas las piezas. Se pide:

1. Escribir el esquema de vuelta atrás que mejor se adapta a este problema.
2. Hacer una descripción detallada del árbol de búsqueda de la solución estableciendo:
   a) Contenido general de un estado y contenidos del estado inicial, de un estado intermedio y de un estado final.
   b) Esquema de ramificación de forma que no se repitan nodos.
   c) Mostrar el árbol de expansión implícito para el ejemplo de este ejercicio utilizando **Backtracking**.
3. Diseñar la(s) funciones de coste (cotas) necesarias para resolver este problema mediante la técnica de Ramificación y Poda. Mostrar el árbol de expansión implícito para el ejemplo de este ejercicio utilizando **Ramificación y Poda**.

Ejemplo:

$$\begin{array}{c} \\ i=1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{bmatrix} 3 & 2 & 1 \\ 3 & 3 & 2 \\ 4 & 4 & 3 \end{bmatrix} \end{array} \quad j$$

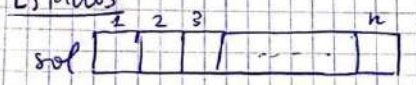
→ Ej → Ensamblar n piezas de un vehículo 22/1/20
Orden influye en los costes
C[i,j] coste de ensamblar la pieza i después de haber ensamblado j piezas
minimizar coste total de ensamblado.



coste de ensamblar la pieza 1, y antes haber ensamblado 1 pieza.

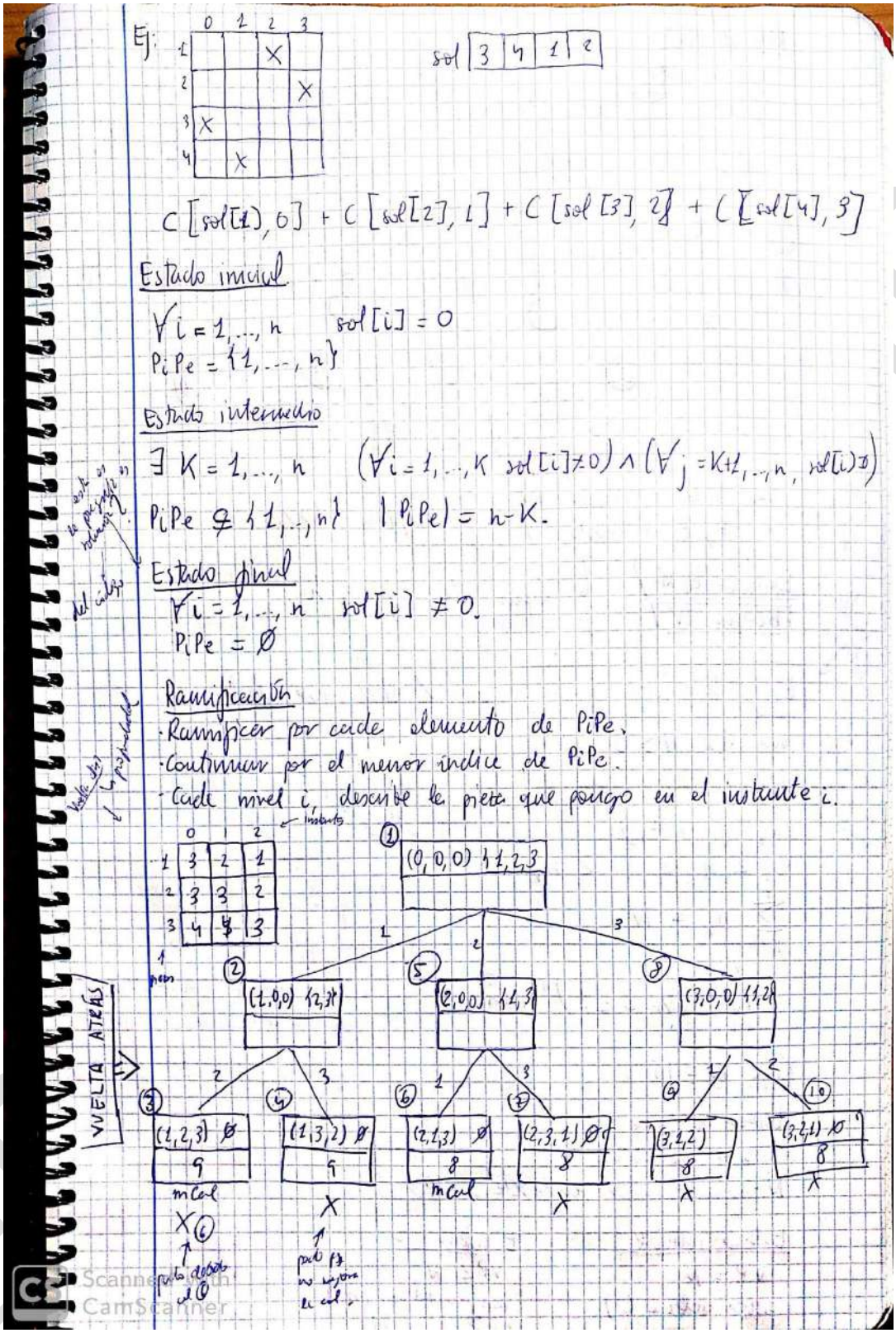
n piezas en n instantes
Orden:
Fuerza Bruta: $P_n = n! \in EXP$

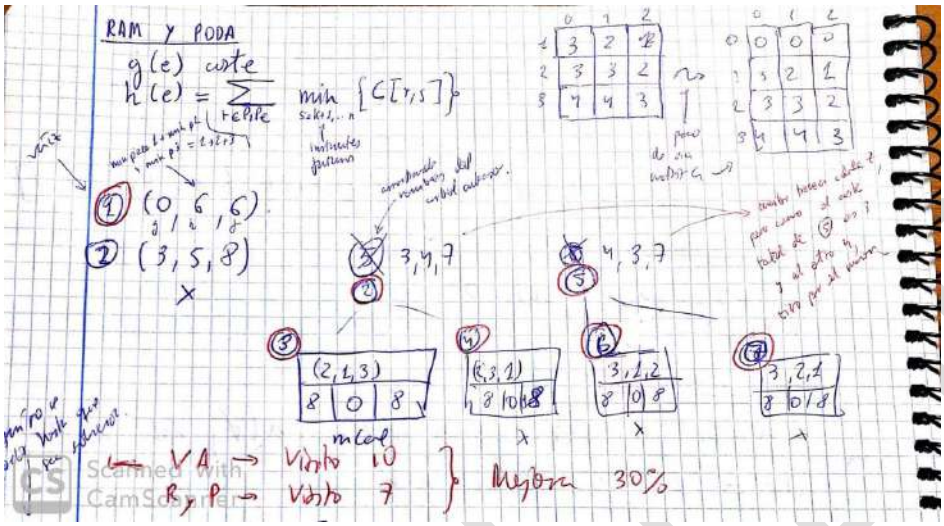
Estados



sol[p] = q la pieza q se ensambla en el instante p
y antes he ensamblado p-1 piezas
piezas pendientes de ensamblar (Pi Pe)

$$\text{Coste} = \sum_{r=1}^{n} C[sol[r], r-1]$$





ejBloque3RyP-Feb19.pdf

realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Bloque III: Ramificación y Poda

Dado un tablero de 3 × 3 con 8 fichas (numeradas del 1 al 8) y un espacio vacío ya situados; el objetivo es colocar los números en las casillas para que coincidan con la configuración final utilizando el espacio vacío.

1	2	3
5	6	
7	8	4

conf. inicial

1	2	3
5		6
7	8	4

conf. final

En cada movimiento podemos deslizar una de las cuatro casillas adyacentes (izquierda, derecha, arriba y abajo) en el espacio vacío. El objetivo de nuestro problema es alcanzar un estado final dado con el menor número de movimientos. En el ejemplo mostrado, los tres movimientos posibles inicialmente son mover la ficha 3, la ficha 6 o la ficha 4 al espacio vacío produciéndose una nueva configuración.

1. [6p] Hacer una descripción detallada de los elementos de la solución. Establecer cuál es la estructura solución, restricciones, proceso de ramificación, y definir claramente la función de estimación para cada nodo y su complejidad (indicar al menos un ejemplo del cálculo de la cota).
2. [4p] Expandir el árbol con la configuración inicial del ejemplo, indicando, claramente, el orden de generación de los nodos y las cotas utilizadas y/o calculadas para llegar a la configuración final.

Febrero 2019

Bloque III: Ram y Pod

1) En cada estado a tener en cuenta:
- el tablero
- el conjunto de los movimientos posibles MP.
- $f(e) = g(e) + h(e)$
  - $g(e)$ = nº movimientos realizados
  - $h(e)$ = nº movimientos posibles de fichas descolocadas respecto a conf. final

Ram: pasamos por cada uno de los de MP,

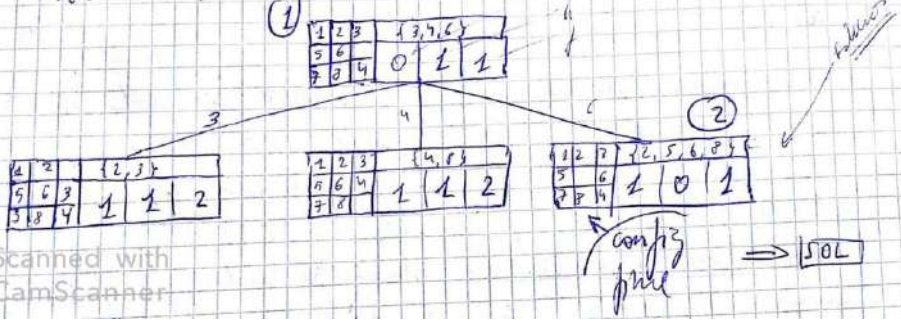
$f(e) \in \mathcal{O}(1)$

Ej. cálculo de función de cota:

Para conf. inicial, f(inicial) valdría:

$g(\text{inicial}) = 0$
$h(\text{inicial}) = 1$
$\Big\} f(\text{inicial}) = 0 + 1 = 1.$

2)



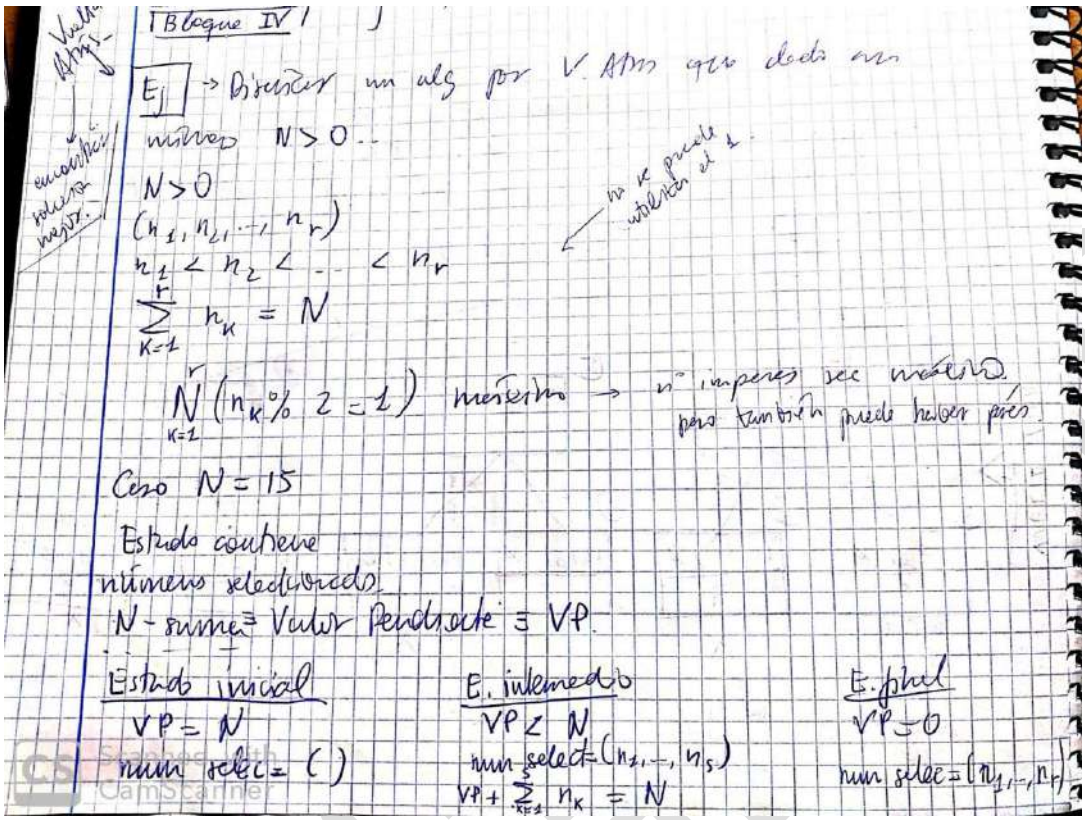
ejVAtipo-examen.pdf

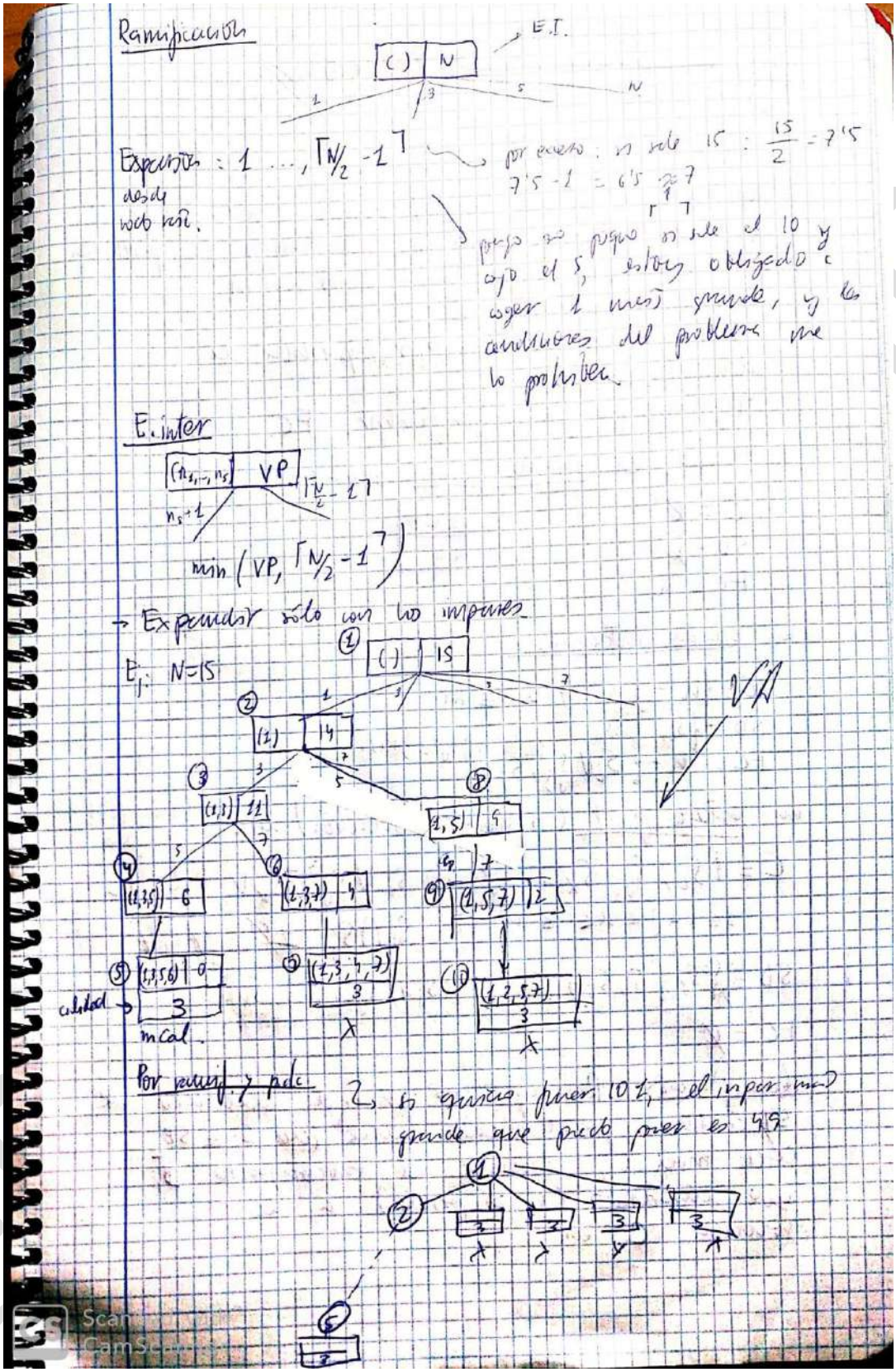
realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga





ejBloque3VA-Sept19.pdf

realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Bloque III: Vuelta atrás

Sean $S_1, \ldots, S_m$ una colección de m conjuntos, donde $S_i \subseteq \{1, \ldots, n\}$, $1 \leqslant i \leqslant m$. Se dice que $C \subseteq \{1, \ldots, n\}$ es un impactador de $S_1, \ldots, S_m$ si, y sólo si, $C \cap S_i \neq \emptyset$ para todo S_i, $1 \leqslant i \leqslant m$.

- [6p] Implementar un algoritmo de vuelta atrás para determinar si existe un impactador C de tamaño $|C| \leqslant k$, para un cierto k y una cierta colección de conjuntos $S_1, \ldots, S_m$.
- [4p] Construir el árbol de búsqueda para $k = 2$ y conjuntos $\{1,2\}, \{1,3\}, \{1,5\}, \{2,3,4\}, \{4,6\}$

Examen Sept 2019

Bloque III: Vuelta Atrás.

$S_1, \ldots, S_m$ colección de m conjuntos

$S_i \subseteq \{1, \ldots, n\} \quad \forall i \in \{1, \ldots, m\}$

$C \subseteq \{1, \ldots, n\}$ es impactador de $S_1, \ldots, S_m \iff C \cap S_i \neq \emptyset \quad \forall i \in \{1, \ldots, m\}$

a) $|C| \leq K$.

En cada estado vamos a actualizar la colecc $S_1, \ldots, S_m$ el conjunto C y su tamaño.

El estado inicial estará constituido por todos los $S_1 \ldots S_m$ intactos, $C = \emptyset$ y $t = 0$. ($t = |C|$)

Un estado intermedio estará formado por $S_1, \ldots, S_m$ con algunos derivados ya impactados, y el conjunto C con un tamaño variable, dependiendo de por donde hayamos ramificado.

Llegaremos a un estado final, si obtenemos un conjunto C con tamaño $t \leq K$, y los S_i están completamente impactados ($\forall i \in \{1, \ldots, m\} : S_i = \emptyset \ldots$).

Ramificamos por cada S_i, cogemos su primer elemento, lo eliminamos del resto de S_i y expandimos por ahí en alguno de los expansores más encontrados con un conjunto C de tamaño K, y aún queden S_i sin impactar retrocedemos.

ejBloque3RyP-Feb18.pdf

realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

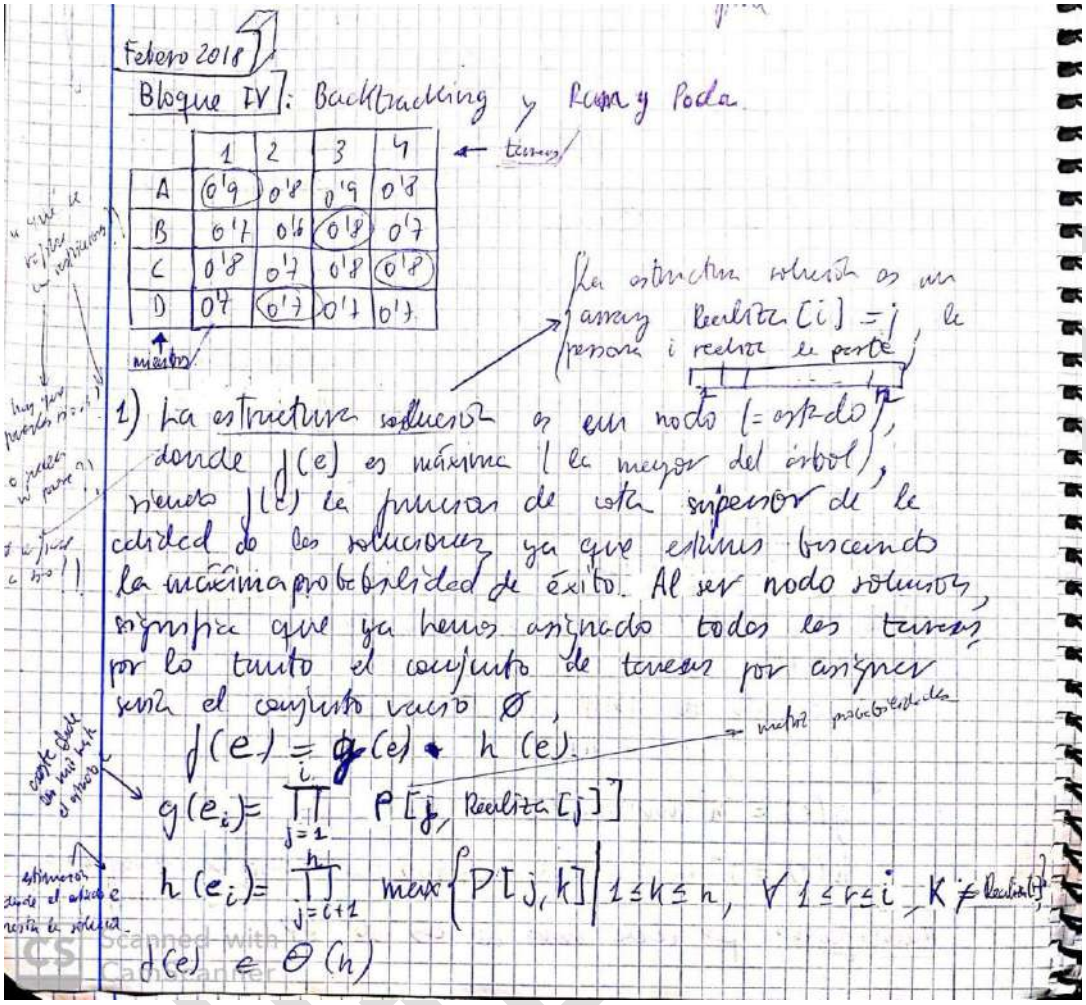
Bloque IV: Backtracking y Ramificación y Poda

Una empresa de software está formando un equipo para llevar a cabo un proyecto software que incluye varias partes. Hay cuatro miembros del equipo: A, B, C y D, y cuatro partes (1, 2, 3 y 4) a desarrollar. Cada miembro del equipo puede desarrollar exactamente una parte. Las cuatro partes deben desarrollarse con éxito para que el proyecto en general tenga éxito, sin embargo, la probabilidad de que un miembro del equipo desarrolle con éxito una parte varía, como se muestra en la tabla de abajo. Por ejemplo, si los miembros del equipo se asignaron a las partes en el orden ABCD, entonces la probabilidad general de finalización exitosa del proyecto es $0,9 * 0,6 * 0,8 * 0,7 = 0,3024$. Aplicar la técnica de **Ramificación y Poda** para encontrar la solución que maximice la probabilidad de éxito. Se pide:

1. [6p] Hacer una descripción detallada del árbol de búsqueda de la solución. Establecer cuál es la estructura solución y las restricciones que presenta el problema. Definir claramente la función de estimación para cada nodo, así como la complejidad de su implementación. Indicar al menos un ejemplo de su cálculo.

2. [4p] Expandir el árbol con la siguiente tabla, indicando, claramente, el orden de generación de los nodos y las cotas utilizadas y/o calculadas.

	1	2	3	4
A	0.9	0.8	0.9	0.8
B	0.7	0.6	0.8	0.7
C	0.8	0.7	0.8	0.8
D	0.7	0.7	0.7	0.7



Febrero 2018

Bloque IV: Backtracking y Ramificación y Poda

	1	2	3	4
A	(0'9)	0'8	0'9	0'8
B	0'7	0'6	(0'8)	0'7
C	0'8	0'7	0'8	(0'8)
D	0'7	(0'7)	0'7	0'7

← tareas

↑ máquinas

La estructura solución es un array Realiza[i] = j, la persona i realiza la parte j

1) La estructura solución es un nodo (= estado), donde f(e) es máxima (la mejor del árbol), siendo f(e) la función de esta superior de la calidad de las soluciones, ya que estamos buscando la máxima probabilidad de éxito. Al ser nodo solución, significa que ya hemos asignado todas las tareas, por lo tanto el conjunto de tareas por asignar será el conjunto vacío ∅.

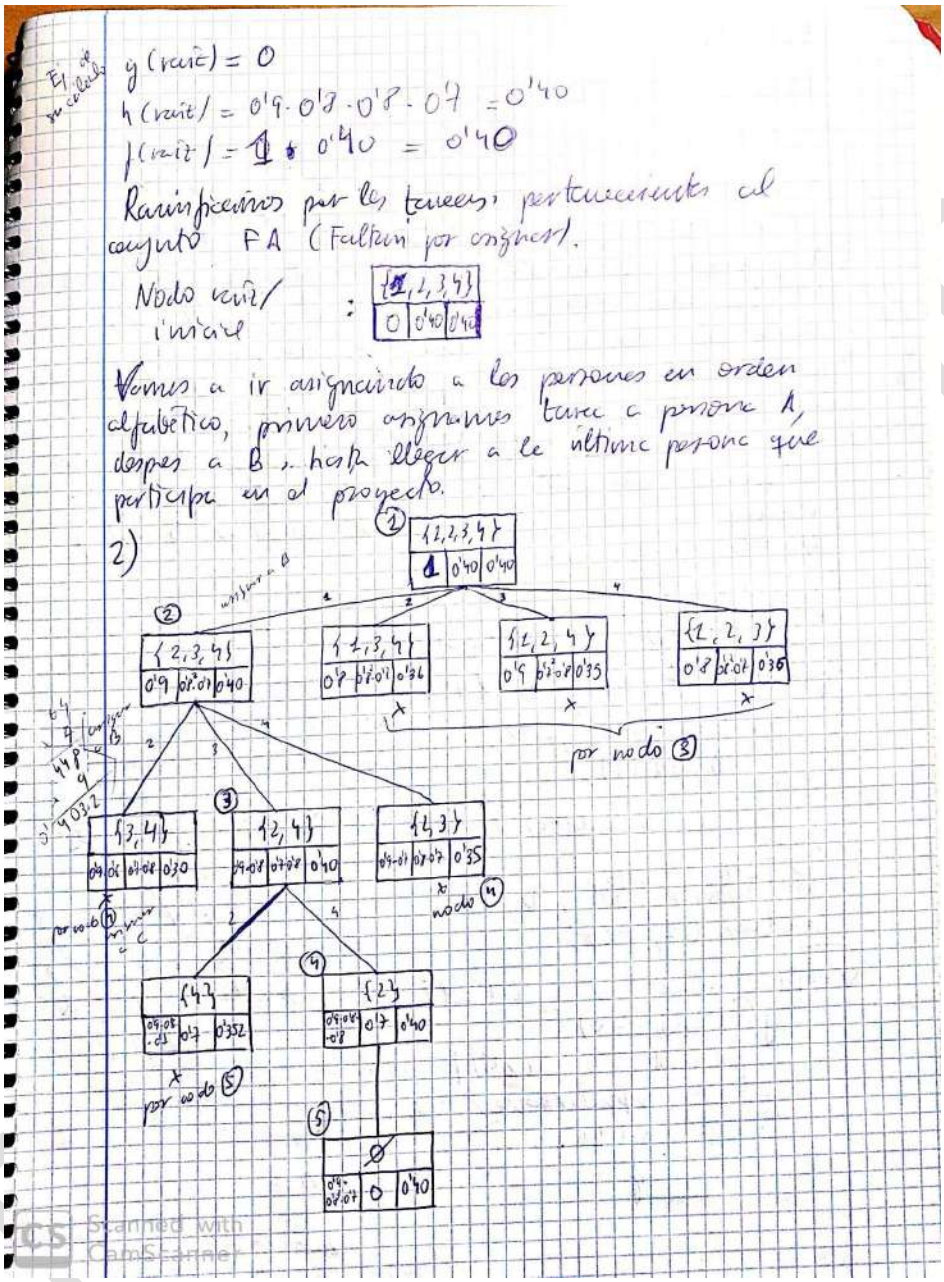
$$f(e_i) = g(e_i) \cdot h(e_i)$$

→ mejor probabilidad

$$g(e_i) = \prod_{j=1}^{i} P[j, Realiza[j]]$$

$$h(e_i) = \prod_{j=i+1}^{n} \max\{P[j,k] \mid 1 \le k \le n,\ \forall\, 1 \le r \le i,\ K \neq Realiza[j]\}$$

$f(e) \in \Theta(h)$



ejBloque3VA-Feb14.pdf

realGCabrones

Analisis y Diseño de Algoritmos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

Bactracking y Ramificación y Poda

Supón te dedicas a organizar bodas. La pareja cuya boda estás organizando en este momento te da la lista con sus n invitados. Además de la lista, la pareja también te proporciona información sobre cómo es la relación entre los distintos invitados. Esta información puede representarse mediante un grafo $G(V, E)$ en el que cada vértice es un invitado, y una arista $(u, v) \in E$ significa que u y v se llevan muy mal. Tu trabajo consiste en distribuir los invitados en mesas (de un máximo K de comensales, que no tienen que estar completamente ocupadas) de forma que ningún par de invitados que se llevan mal compartan mesa.

1. [6p] Dado un número máximo de mesas M, diseña, utilizando la técnica de vuelta atrás, un algoritmo que encuentre una distribución correcta de invitados en las mesas, o devuelva una indicación de que no existe ninguna distribución.

2. [4p] Modifica el algoritmo anterior para que se encuentre la distribución que utiliza el menor número de mesas.

Feb 2017

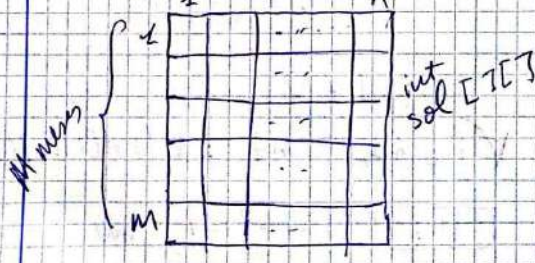
Bloque III: Backtracking y Poda.

1) → Nº mínimo de mesas M
→ Por cada Mesa, K comensales máximo.

```
Solucion backtracking (Solucion sol) {
  if (esSolucion (sol)) return sol;
  else {
    Conj setCand = posContinuacion (sol);
    Solucion solAux = null;
    while (!esVacio (setCand) && solAux == null) {
      cand = seleccione (setCand);
      setCand = setCand - {cand};
      solAux = backtracking (sol + cand);
    }
    return solAux;
  }
}
```

ESQUEMA

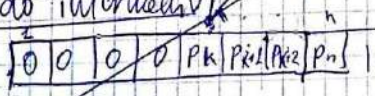
La estructura solución es una matriz de K columnas y M filas, cada fila representa cada una de las mesas disponibles para sentar a los n invitados.



Nodo inicial / raíz

El nodo inicial contiene la matriz sol[][] previamente descrita llena de 0's (no hay invitados sentados), y el array Inv (que contiene los invitados de la boda) completo hasta la posición n.

Nodo intermedio



$\forall i \in \{1, \dots, k-1\}$: Inv[i] = 0 // Ya he sentado a esos invitados

- Cada nodo / estado contendrá la matriz sol[][] previamente descrita, que inicialmente estará inicializada a 0, debido a que al principio no hay ningún invitado sentado

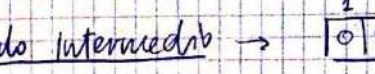
Cada nodo o estado contendrá:

- La matriz sol[][] previamente descrita, que inicialmente estará llena de ceros ya que no hay ningún invitado sentado, pero conforme se vaya expandiendo el árbol se irá llenando de p_i (personas) contenidas en el array Inv[]
- El array de Invitados Inv[], que inicialmente estará lleno, pero que conforme se vayan asignando mesa a cada invitado p_i y Inv[i] = 0, y cuando Inv[] esté completamente lleno de 0's, significará que cada invitado tiene mesa asignada adecuadamente, por tanto sol[][] será una solución a nuestro problema

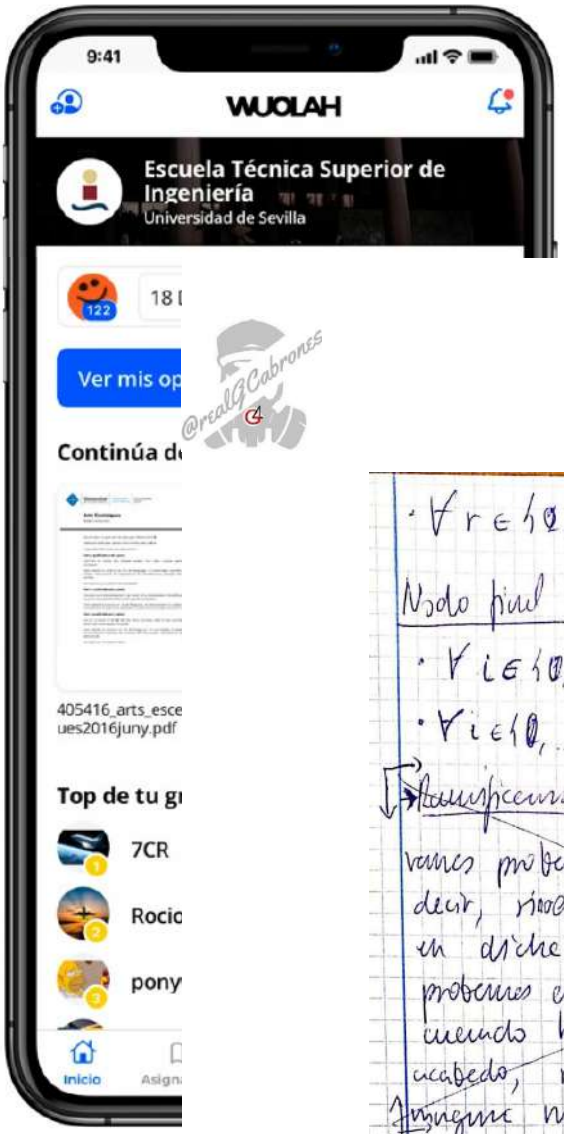
Nodo Raíz / Inicial

- $\forall i \in \{0, \dots, n-1\}$: Inv[i] ≠ 0.
- $\forall i \in \{0, \dots, k-1\}, \forall j \in \{0, \dots, n-1\}$: N(sol[i][j] ≠ 0) = 0

Nodo intermedio →



- $\forall i \in \{0, \dots, k-1\}$: Inv[i] = 0 // Se han sentado a k personas
  $\forall j \in \{k+1, \dots, n-1\}$: Inv[j] ≠ 0



- $\forall r \in \{0,\dots,k-1\}, \forall s \in \{0,\dots,M-1\} : N(sol[r][s] \neq 0) = K$

Nodo final

- $\forall i \in \{0,\dots,n-1\} : Inv[i] = 0$. // Todas personas asignadas a una.
- $\forall i \in \{0,\dots,k-1\}, \forall j \in \{0,\dots,M-1\} : N(sol[i][j] \neq 0) = n$.

Ramificamos por el primer p_i : $Inv[i] \neq 0$, y vamos probando si es compatible en cada mesa, es decir, vamos mirar en 6 de las personas p colocadas en dicha mesa, n es menor de alguno de ellos, probamos en la siguiente mesa, y así sucesivamente, cuando hayamos colocado todos los invitados hemos acabado, ningún invitado no se puede colocar en ninguna mesa retrocedemos.

En cada nivel vamos a asignar mesa al primer p_i: $Inv[i] \neq 0$, ramificamos por cada una de las mesas que tengamos libres, si ramificamos por una mesa en la que se encuentra un menor de el invitado que queremos sentar a la mesa podemos esa rama, y seguimos con la siguiente mesa, así hasta que queden asignados todos los invitados de $Inv[]$.

2) Será el mismo algoritmo solo que en vez de cuando encuentre solución acabe, que baraje en calidad, en este caso la calidad es el nº de mesas usadas para sentar a los n invitados, cuando encontremos todas las soluciones del árbol, nos quedamos con la que mejor calidad tenga, es decir, con la que menor número de mesas use para resolver el problema.