6 Búsqueda exhaustiva

ALGORÍTMICA Y COMPLEJIDAD

Camilo Palazuelos Calderón

Curso 2023-2024

Vuelta atrás (VA)

- VA realiza una exploración sistemática de las posibles soluciones del problema
  - Analiza todas las posibilidades, lo que conlleva un *coste temporal exponencial*
  - Suele utilizarse para atacar problemas de la clase de complejidad NP-completo

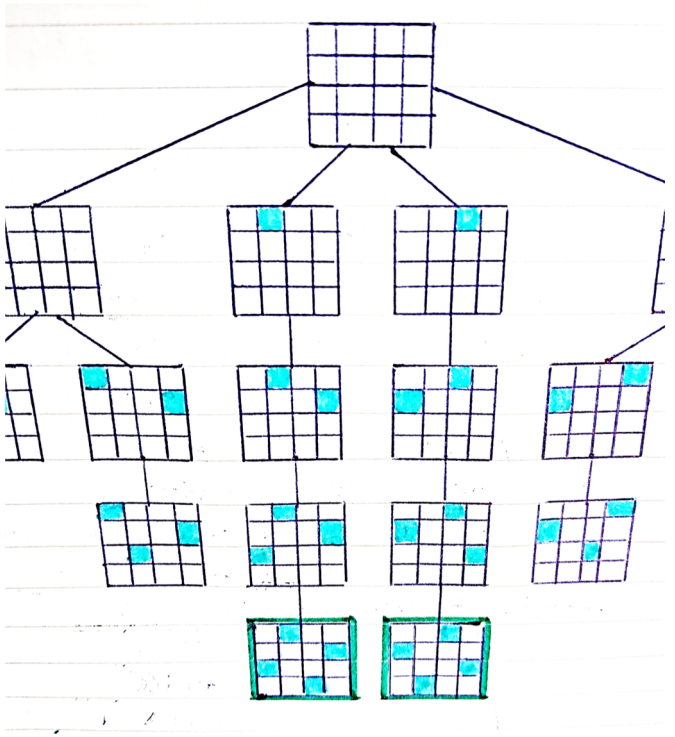
- El diseño recursivo es la manera natural de pensar en VA
  - Se recorre un *árbol de búsqueda* en profundidad, que se maneja implícitamente a través de la recursión
  - Cuando, desde un nodo, no se puede seguir buscando la solución, se produce la *vuelta atrás*

- El coste temporal es prácticamente imposible de calcular *a priori*
  - Aunque depende del número de *nodos explorados*
  - *Cota superior:* $O(a^{f(n)})$, donde a es el número de decisiones y $f(n)$, el tamaño de la solución

El problema de las *n* reinas, I

- Queremos colocar *n* reinas en un tablero *n* x *n* de manera que no se coman entre sí
  - No puede haber más de una reina por fila
  - No puede haber más de una reina por columna
  - No puede haber más de una reina por diagonal

- La solución puede expresarse en forma de array
  - Por ejemplo, *A*[1] = 2 significa que la reina de la fila 1 está colocada en la columna 2



El problema de las *n* reinas, II

- Solución:

```
REINAS(n):
    A[1..n] ← (0,…,0)
    if REINAS(A[1..n], 1)
        return A[1..n]
    else
        return NULL
```

```
REINAS(A[1..n], f):
    for c ← 1 to n
        A[f] ← c
        if ESFACTIBLE(A[1..n], f)
            if f = n
                return TRUE
            else if REINAS(A[1..n], f + 1)
                return TRUE
    return FALSE
```

```
ESFACTIBLE(A[1..n], f):
    for i ← 1 to f – 1
        if A[i] = A[f] or |A[i] – A[f]| = |i – f|
            return FALSE
    return TRUE
```

- Coste temporal: $O(n^n)$

Coloración de grafos

- Dados un grafo G y un número entero positivo m, ¿es G m-coloreable?
  - Es decir, ¿se pueden pintar con colores los vértices de G de modo que no haya dos adyacentes con el mismo color usando m colores?

- Solución:

```
COLOREAR(A[1..n, 1..n], m):
    B[1..n] ⟵ (0,…,0)
    if COLOREAR(A[1..n, 1..n], B[1..n], m, 1)
        return B[1..n]
    else
        return NULL
```

```
COLOREAR(A[1..n, 1..n], B[1..n], m, v):
    for i ⟵ 1 to m
        B[v] ⟵ i
        if ESFACTIBLE(A[1..n, 1..n], B[1..n], v)
            if v = n
                return TRUE
            else if COLOREAR(A[1..n, 1..n], B[1..n], m, v + 1)
                return TRUE
    return FALSE
```

```
ESFACTIBLE(A[1..n, 1..n], B[1..n], v):
    for i ⟵ 1 to v – 1
        if A[i, v] = 1 and B[i] = B[v]
            return FALSE
    return TRUE
```

- Coste temporal: $O(m^n)$

El problema del ciclo hamiltoniano

- Dado un grafo G no dirigido y conexo, ¿existe, al menos, un ciclo hamiltoniano en G?
  - Es decir, ¿se pueden recorrer todos los vértices de G pasando solo una vez por cada uno de ellos (salvo por el primero, al que se debe volver)?
- Solución:

```
HAMILTONIANO(A[1..n, 1..n]):
    B[1..n] ← (0,…,0)
    if HAMILTONIANO(A[1..n, 1..n], B[1..n], 1)
        return B[1..n]
    else
        return NULL
```

```
HAMILTONIANO(A[1..n, 1..n], B[1..n], v):
    for i ← 1 to n
        B[v] ← i
        if ESFACTIBLE(A[1..n, 1..n], B[1..n], v)
            if v = n
                return TRUE
            else if HAMILTONIANO(A[1..n, 1..n], B[1..n], v + 1)
                return TRUE
    return FALSE
```

```
ESFACTIBLE(A[1..n, 1..n], B[1..n], v):
    if v = 1
        return TRUE
    if A[B[v – 1], B[v]] = 0
        return FALSE
    if v = n and A[B[v], B[1]] = 0
        return FALSE
    for i ← 1 to v – 1
        if B[i] = B[v]
            return FALSE
    return TRUE
```

- Coste temporal: $O(n^n)$

Ramificación y poda (RP)

- RP realiza una exploración sistemática de las posibles soluciones del problema
  - Analiza todas las posibilidades cuyo coste potencial (*cota*) sea menor que el *coste* de la mejor solución hasta el momento
  - Suele utilizarse para atacar problemas de la clase de complejidad NP-completo y requerir un coste temporal exponencial, aunque menor que el requerido por VA

- El diseño iterativo es la manera natural de pensar en RP
  - Se recorre el árbol de búsqueda con la estrategia *primero el mejor* usando una cola de prioridad
  - Cuando, desde un nodo, la cota para llegar a la solución es mayor que el coste de la mejor solución hasta el momento, *se poda*, es decir, dejan de encolarse los descendientes del nodo

- El coste temporal es prácticamente imposible de calcular *a priori*
  - Aunque depende del número de *nodos explorados*
  - *Cota superior:* $O(a^{f(n)})$, donde a es el número de decisiones y $f(n)$, el tamaño de la solución

```
LABERINTO(A[1..f, 1..c], x, y):
    A_mejor ← NULL
    p_mejor ← ∞
    C ← ()
    INSERTAR((A[1..f, 1..c], x, y, 1), C)
    while not VACÍA(C)
        (A'[1..f, 1..c], x', y', p) ← EXTRAERMÍN(C)
        A'[x', y'] ← p
        for i ← 1 to 4
            (x'', y'') ← MOVER(x', y', i)
            if ESFACTIBLE(A'[1..f, 1..c], x'', y'') and COTA(…) < p_mejor
                if A'[x'', y''] = –2
                    if p + 1 < p_mejor
                        A_mejor ← COPIAR(A'[1..f, 1..c])
                        p_mejor ← p + 1
                else
                    INSERTAR((COPIAR(A'[1..f, 1..c]), x'', y'', p + 1), C)
    return A_mejor
```

Salir de un laberinto lo antes posible

- Cota: Camino recorrido + distancia Manhattan hasta la salida
  - $D_M((x_{ini}, y_{ini}), (x_{fin}, y_{fin})) = |x_{ini} - x_{fin}| + |y_{ini} - y_{fin}|$
  - Si $p_{mejor} = \infty$ (aún no se ha encontrado la primera solución), no hace falta comprobar si la cota es mejor que p_{mejor}
- Coste temporal: $O(3^{fc})$

```
JUEGODEL15(A[1..n, 1..n] , x, y):
    m_mejor ← ∞
    C ← ()
    INSERTAR((COPIAR(A[1..n, 1..n]), x, y, 0), C)
    while not VACÍA(C)
        (A'[1..n, 1..n], x', y', m) ← EXTRAERMÍN(C)
        for i ← 1 to 4
            (A''[1..n, 1..n], x'', y'') ← MOVER(A'[1..n, 1..n], x', y', i)
            if COTA(A''[1..n, 1..n], x'', y'', m + 1) < m_mejor
                if ESSOLUCIÓN(A''[1..n, 1..n], x'', y'')
                    if m + 1 < m_mejor
                        m_mejor ← m + 1
                else
                    INSERTAR((A''[1..n, 1..n], x'', y'', m + 1), C)
    return m_mejor
```

El juego del 15

- Dada una cajita formada por n^2 casillas ocupadas números del 1 al $n^2 - 1$ (hay un hueco que "mover"), ¿cuál es el número mínimo de movimientos para ordenar los números?

- ¿Cómo diseñamos la función de cota?
  - N.º de fichas que faltan por ordenar… *¡No!*
  - Dist. Manhattan de las fichas hasta su posición final… *¡No!*
  - Combinación de ambas estrategias… *¡Sí!*

- Coste temporal: $O(4^{n^2})$
  - Tal y como está resuelto, ¿puede darse el caso de pasar 2 o más veces por una solución parcial concreta (configuración de A)?