Índice

Segment Tree: Lazy Propagation and Lowest Common Ancestor

Rubén Guzman, Mauricio Carazas, Fernando Ramírez, Alexander Baylon

15 de abril del 2021

1. Introducción

Un *segment tree*, conocido también como un *statistic tree*, es una estructura de datos basada en arboles utilizada para almacenar información en intervalos o **segmentos**, realizar operaciones en estos de forma efectiva y ser lo suficientemente flexibles para modificarse. El *segment tree* fue inventado por Jon Bentley en 1977 en su artículo "Solutions to Klee's rectangle problems".

2. Definición

Un segment tree es una estructura de datos que permite realizar consultas en un rango de forma efectiva, estas consultas incluyen la suma de elementos consecutivos $a[l..r]$, o hallar el elemento menor en dicho rango en tiempo $\mathcal{O}(n \log n)$. El segment tree nos permite modificar el arreglo al reemplazar un elemento, o incluso cambiar los elementos de todo un sub segmento a cualquier valor, o añadir un valor a todos los elementos de este sub segmento.

0	1	2	3	4	5	6	7
3	5	1	2	7	8	2	4

2.1. Forma Simple

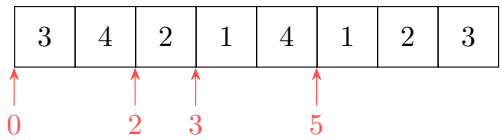
Se tiene un arreglo $a[0...n-1]$, el segment tree tiene que ser capaz de hallar la suma de los elementos entre los índices l y r:

$$\sum_{i=l}^{r} = a[i]$$

, y también cambiar los valores de los elementos en el arreglo:

$$a[i] = x$$

El segment tree debe ser capaz de procesar estas dos operaciones en tiempo $\mathcal{O}(n \log n)$.



Definición 1 Una reina es un par de numeros enteros (α, β) con $0 \leq \alpha, 0 \leq \beta$.
Para el resto de estas definiciones se asume que $q = (\alpha, \beta)$ y donde sea apropiada una segunda reina $q_1 = (\alpha_1, \beta_1)$.

Problema 1. n-Queens PROBLEMA: $\langle n \rangle$ donde n es un entero ≥ 1.
SOLUCIÓN: Un conjunto de Q de reinas las cuales caben en el tablero de tamaño n tal que: $|Q| = n$; y por cada dos reinas distintas $q_1, q_2 \in Q$, q_1 no ataca a q_2. Adicionalmente, para cada dos reinas distintas $(\alpha_1, \beta_1), (\alpha_2, \beta_2) \in Q$, ambas $\alpha_1 + \beta_1 \neq \alpha_2 + \beta_2 \pmod{n}$ y $\alpha_1 - \beta_1 \neq \alpha_2 - \beta_2 \pmod{n}$, entonces se dice que Q es una solución "modular" del problema de n-Queens.

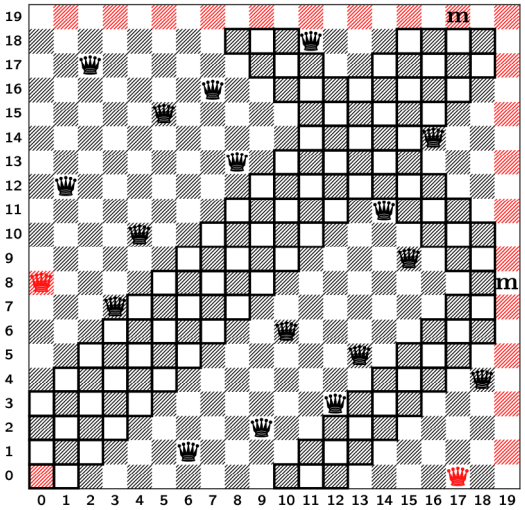
Problema 2. n-Queens Completion PROBLEMA: $M_2 = \langle n, P \rangle$ donde n es un entero y P es un conjunto de Q de reinas las cuales caben en el tablero de tamaño n y no existen dos reinas en P que tengan la misma columna o fila.
SOLUCIÓN: Un conjunto S_2 de reinas el cual es una solución al problema de n-Queens para n tal que $P \subseteq S_2$.
Problema 3. PROBLEMA: $M_3 = \langle n, P, C, R \rangle$ donde $\langle n, P \rangle$ es una instancia del *Problema 2*, y C, R son conjuntos de enteros tal que $|R| = |C|$ y por cualquier reina $(\alpha_1, \beta_1), (\alpha_2, \beta_2) \in P, \alpha \notin C$ y $\beta \notin R$.
SOLUCIÓN: Un conjunto S_3 de reinas tal que: $|S_3| = |C| + |P|$; $P \subseteq S_3$; no existen dos reinas en S_3 que se ataquen una a la otra; y para cada par $(\alpha, \beta) \in S_3 \backslash P$ tenemos $\alpha \in C, \beta \in R$.
Problema 4. Diagonales Excluidas PROBLEMA: $M_4 = \langle n, C, R, D^-, D^+ \rangle$ donde $M_3 = \langle n, \{\}, C, R, \rangle$ es una instancia del *Problema 3* D^-, D^+ son un conjunto de enteros con $D^- \subseteq \{-(n-1), ..., n-1\}$, $D^+ \subseteq \{0, ..., 2n-2\}$.
SOLUCIÓN: Un conjunto de reinas S_4 el cual es una solución a M_3 y adicionalmente: para cualquier reina $(\alpha, \beta) \in S_4$ tenemos que $\alpha - \beta \notin D^-$, $\alpha + \beta \notin D^+$.
Problema 5. PROBLEMA: Un conjunto $M_5 = \{M_{4,a} | 0 \leq a \leq |M_5|\}$, donde cada M_4 es una instancia del *Problema 4* con $D^+ = \{\}$.
SOLUCIÓN: Un conjunto $S_5 = \{S_{4,a} | 0 \leq a \leq |M_5|\}$ donde cada S_4 es una solución a $M_{4,a}$ y adicionalmente: para cualquier $\{S_{4,a}, S_{4,a} \subseteq S_5\}$, y cualquier $(\alpha_a, \beta_a) \in S_{4,a}$, $(\alpha_b, \beta_b) \in S_{4,b}$, tenemos que $(\alpha_a, \beta_a) \neq (\alpha_b, \beta_b)$.
Problema 6. 1-in-3-SAT Restringido PROBLEMA: Un par $M_6 = \langle V, C \rangle$ donde C es un conjunto(de *clausulas*) tal que cada cada $c \in C$ es un conjunto de tres variables, $c = \{v_i, v_j, v_k\}$; y donde $V = \{v | \exists c \in C \cdot v \in c\}$ es el conjunto de todas las variables contenidas en cualquier clausula. Cada variable $v \in V$ ocurre a lo mucho en tres clausulas en C.
SOLUCIÓN: Una asignación de verdad $S_6 : V \rightarrow \{true, false\}$ tal que para todo $c = \{v_i, v_j, v_k\} \in C$, S_6 mapea exactamente uno de v_i, v_j, v_k a *true* y los otros dos a *false*.

3. Demostración de NP-completo

Se realiza una serie de reducciones del *Problema 6* al *Problema 2*. Cada reducción sera *polinomial* y *parsimoniosa*, probando así que el *Problema 2* es *NP-Complete* y *#P-Complete*.

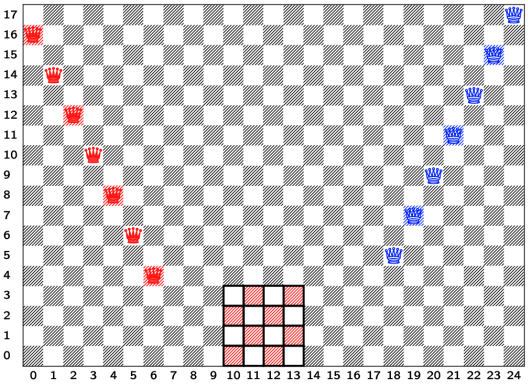
Definición 2 - Reducción del Problema 3 al Problema 2 Esta reducción toma una instancia del *Problema 3* y la $n'-$incrusta en la posición $(0, 0)$ de una tabla basada en la *Figura 1*(donde n' se define como parte de la construcción).

Figura 1: Figura 1



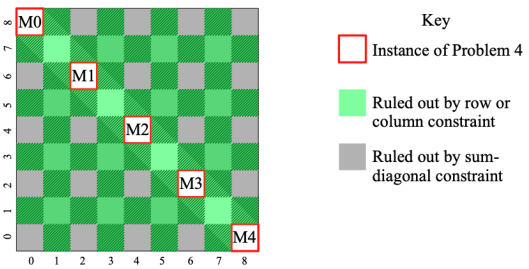
Definición 3 - Reducción del Problema 4 al Problema 3 Para la instancia del *Problema 4* $M_4 = \langle n, C, R, D^-, D^+ \rangle$, el conjunto $Q_{D^-} = \{(6n-3-d, 3n-1-2d) | d \in D^-\}$, el conjunto $Q_{D^+} = \{(2n-2-d, n+2d) | d \in D^+\}$, y el conjunto $M_3 = \langle 7n-3, Q_{D^-} \cup Q_{D^+}, \{c+3n-2 | c \in C\}, R \rangle$.

Figura 2: Figura 2



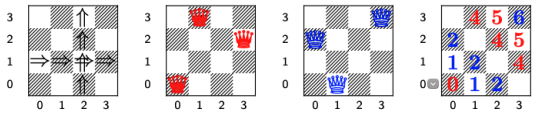
Definición 4 - Reducción del Problema 5 al Problema 4 Si $M_5 = \{M_{4,a} \mid a = 0, 1, 2..., k-1\}$ es una instancia del *Problema 5* donde $k = |M_5|$ y cada $M_{4,a} = \langle n_a, C_a, R_a, D_a^-, \{\} \rangle$. Primero se fija $n' = max\{n_a \mid 0 \leq a \leq k\}$.Se posiciona cada artilugio $M_{4,a}$ en la posición (α_a, β_a) en una cuadrícula $n \times n$ y define la instancia del *Problema 4* como $M_4 = \langle n, C, R, D^-, D^+ \rangle$.

Figura 3: Figura 3



Definición 5 - Reducción de 1-in-3-SAT al Problema 5 Para un entero $\delta \geq 0$ y $G_a = \langle n_a, C_a, R_a, D_a^-, \{\}\rangle$, se define $G_a \oplus \delta_a \stackrel{def}{=} \langle n_a + \delta_a, \{c + \delta_a \mid c \in C_a\}, R_a, \{d + \delta_a \mid d \in D_a^-\}, \{\}\rangle$

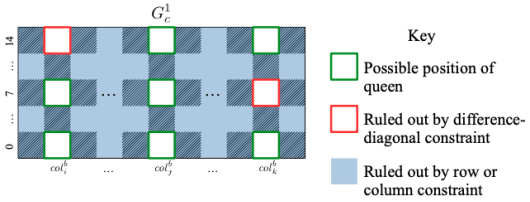
Figura 4: Figura 4



G^1, G^2, G^3 **- Tres artilugios para una clausula 1-in-3-SAT**

Se presentan tres artilugios asociados con la clausula $c = \{v_i, v_j, v_k\}$, los cuales son G_c^1, G_c^2, G_c^3.

Figura 5: Figura 4



Definición 6 - Reducción de 1-in-3-SAT restringido al Problema 5 Se considera una instancia de $M_6 = \langle V, C\rangle$ de 1-in-3-SAT restringido. Para una clausula $c = \{v_i, v_j, v_k \in C\}$ se reescribe $M_{5,c}$ para la instancia del *Problema 5*, $M_{5,c} \stackrel{def}{=} \{G_i^0, G_j^0, G_k^0, G_c^1, G_c^2, G_c^3\}$. Para la reducción de la instancia entera M_6 se establece:

$$M_5 \stackrel{def}{=} \bigcup_{c \in C} M_{5,c}$$

4. Soluciones

4.1. Algoritmo por Fuerza bruta

4.1.1. Explicación

Evidentemente la solución más simple de resolver este problema es verificar todos los posibles subconjuntos existentes en el tablero de tamaño N y para cada uno de ellos comprobar si las posiciones del subconjunto adquirido son validas, sin embargo esto representaría un algoritmo con tiempo exponencial, ya que buscar todos los posibles subconjuntos tendría un tiempo de ejecución de orden $O(2^n \times n)$ y si para cada subconjunto tenemos que hallar todas las posibles variaciones de sus elementos esto tendría un tiempo de ejecución O(n) por lo cual esta solución tiene un tiempo de ejecución total de $O(2^n n)$ lo cual en una escala mucho mayor se vuelve una solución insostenible.

4.1.2. Algoritmo

El algoritmo es el siguiente :

`Input:`

```
    Permutacion de posicion actual

Output:
    Falso si cumple
    Verdadero si no cumple

    Paso 1:
        a. Se generan todas las permutaciones posibles para el tablero
    Paso 2:
        a. Al restar los indices de dos posiciones , se puede verificar si
        estan en la misma diagonal
    Paso 3:
        a. Si se encuentra fuera de la diagonal se imprime la solucion
```

4.1.3. Implementación

La siguiente implementación está hecha en python, donde la forma de recorrer todos los subconjuntos posibles de conjuntos es hacer una permutación por todas las filas y columnas del tablero. La forma en la que se les da una representación es a través de índices en el eje x , y , permitiendo el manejo de la permutación , además de dar un valor el cual pueda calcular la distancia en los ejes , siendo una forma sencilla para comprobar si se encuentra en diagonal mediante la resta de los índices.

- Método

```
import itertools as it

def es_solucion(perm):
  for(i1, i2) in  it.combinations(range(len(perm)),2):
    if abs(i1 - i2) == abs(perm[i1] - perm[i2]):
      return False
  return True

for perm in it.permutations(range (8)):
  if es_solucion(perm):
    print (perm)
```

4.1.4. Análisis

Este algoritmo se puede ver como un algoritmo estándar de permutación , el cual usa las propiedades bases como que solo se puede encontrar una reina por columna y fila (dentro de las permutaciones generadas) y aritmética básica para calcular si se encuentra amenazada de manera diagonal o no.

4.2. Algoritmo con Backtracking

4.2.1. Explicación

Ya hemos podido ver como un algoritmo de fuerza bruta resulta totalmente ineficiente cuando se habla de una cantidad de posiciones a gran escala. Por lo que hemos encontrado un algoritmo Pseudo-polinomial basado en backtracking. Este algoritmo se ve como

una solución bastante eficaz para nuestro problema, ya que puede encontrar el resultado deseado únicamente basándose en recorrer los caminos y cortar los que no tengan salida. A simple vista el N-Reinas parece haber sido solucionado con este método, pero como hemos mencionado antes este es un algoritmo pseudo-polinomial lo cual significa que la complejidad de este algoritmo es un polinomio en el valor número de su entrada, pero no necesariamente en la longitud de su entrada (el número de bits requerido para poder representarlo) por lo que se llega al mismo problema de que a una gran escala este problema se vuelve ineficaz.
La lógica que tiene este algoritmo es poder recuperar comprobaciones ya hechas anteriormente para poder usarlas en el cálculo de comprobaciones posteriores, evidentemente con comprobación nos referimos a comprobar si las reinas cumplen las reglas determinadas [4].

4.2.2. Algoritmo

El algoritmo es el siguiente:

```
Input:
    Arreglo A de tamaño n*n
Output:

    Paso 1:
        a. Crear una matriz booleana de tamaño n*n
    Paso 2:
        a. Calcular las diagonales para saber si es valido
    Paso 3:
        a. Agregar valor posible al conjunto
    Paso 4:
        a. Eliminar si no cumple
    Paso 5:
        a. Finalmente se imprime los conjuntos resultados
```

O(nM)

4.2.3. Implementación

La siguiente implementación está hecha en python, donde la forma de recorrer todos los subconjuntos posibles de conjuntos es hacer un check de las posibilidades dentro de cada columna, para poder ir actualizando el camino y tener marcado donde regresar.

- Método

```
def puede_ser_solucion(perm):
  i = len(perm) - 1
  for j in range(i):
    if i - j == abs(perm[i] - perm[j]):
      return False
  return True
def backtracking(perm, n):
  if len(perm) == n:
    print(perm)
    exit()
```

```
    for k in range (n):
      if k not in perm:
        perm.append(k)
        if puede_ser_solucion(perm):
          backtracking(perm,n)
          perm.pop()

backtracking(perm = [], n = 20)
```

4.3. Algoritmo Cuántico

En [6] se propuso un algoritmo cuántico para solucionar el problema de las n-reinas en un tiempo $O(n^8)$. Tiempo después se propuso otro algoritmo que no solo buscaba solucionar el problema de las n-reinas, sino también dos de sus problemas derivados: completación de las n-reinas y n-reinas con diagonales bloqueados.

Al programar en una computadora cuántica se toma bastante atención a la arquitectura de esta, puesto que aun se esta experimentado en esta área y los algoritmos se ven afectados. La arquitectura que se usa afectara a los algoritmos en la forma de hallar la solución, como se introducen los datos con los que trabajar y como se espera la salida, así como la comprobación de que la salida sea correcta, entre otros [7].

4.3.1. Construcción de la simulación

Se usa a cada posición de cada átomo como representación de una reina y este sera estimulado por un rayo de luz. El átomo estará bloqueado en movimiento en los ejes Y y Z. Luego el sistema pasa estado solido, por la interacción de los átomos, el cual es interpretado como la solución del problema. Todo este proceso se aprecia en la Figura 6.

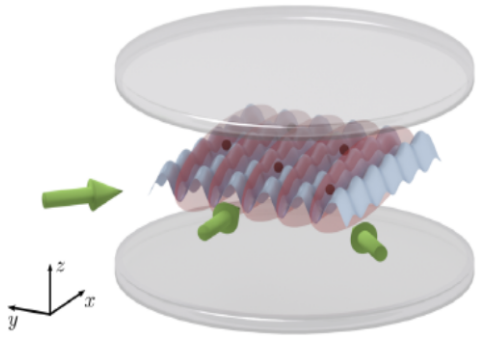


Figura 6: Construcción de la simulación. [7]

4.3.2. Resultados

Se le da una gran importancia a la configuración de la arquitectura, lo bueno y malo encontrado en cada aspecto de esta en el transcurso de los experimentos. En cuanto a los resultados del algoritmo no son totalmente confiables pero se encuentran dentro del parámetro aceptado.

Si se desea recrear el experimento en [7] se encuentra descrita tanto la arquitectura como los parámetros de entrada y una explicación de estos.

5. Conclusión

1. Hemos demostrado que el 'n-Queens Completion' es NP-Completo.
2. La demostración se basa en una reducción del problemas al 3-SAT (revisar secuencia de reducciones).
3. El problema presentado es uno de los más importantes dentro de la Inteligencia Artificial.
4. La solución propuesta usando backtracking es más eficiente que el algoritmo basado en fuerza bruta, además este representa perfectamente la definición de un algoritmo pseudo-polinomial.
5. Las computadoras cuánticas aún son muy inestables, pero se encuentran en una constante mejora; por lo que se estima que en dos décadas, las computadoras cuánticas sean estables y económicamente accesibles como las computadoras personales actuales.

Referencias

[1] Richard M. Karp, *Reducibility among Combinatorial Problems* https://people.eecs.berkeley.edu/ luca/cs172/karp.pdf

[2] Ian P. Gent, Christopher Jefferson, Peter Nightingale, *Complexity of n-Queens Completion.* 2017

[3] Wikipedia, *NP-completo.* https://es.wikipedia.org/wiki/NP-completo

[4] Geeks for Geeks, *Pseudo polynomial in Algorithms* https://www.geeksforgeeks.org/pseudo-polynomial-in-algorithms

[5] Rok Sosic and Jun Gu, *A Polynomial Time Algorithm for the N-Queens Problem.* 1990

[6] Rounak Jha, Debaiudh Das, Avinash Dash, Sandhya Jayaraman, Bikash K Behera, and Prasanta K Panigrahi, *A novel quantum n-queens solver algorithm and its simulation and application to satellite communication using ibm quantum experience..* 2018

[7] Valentin Torggler, Philipp Aumann, Helmut Ritsch, and Wolfgang Lechner, *A Quantum N-Queens Solver.* 2019

If the i-th deletion does not lead to a contraction, we have
$= \mathrm{c}_i + _i - _{i-1}$
$= 1 + (\mathrm{size}_i - 2num_i) - (size_{i-1} - 2num_{i-1})$
$= 1 + (\mathrm{size}_i + 2num_i) - (sum_i - 2(num_i + 1))$
$= 3$
else if the i-th deletion lead to a contraction, we have equation
$\mathrm{num}_{i-1} = num_i + 1 = 1/3size_{i-1} = 1/2size_i$
so
$= \mathrm{c}_i + _i - _{i-1}$
$=(1 + \mathrm{num}_i) + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1})$
$= 2$