Apunte Certamen 2

Marcelo Paz

Analisis y Diseño de Algoritmos 6 de diciembre de 2023

1. Importante

- **Backtracking:** Tipo de algoritmo que se utiliza para resolver problemas de satisfacción de restricciones (CSP).
- **Algoritmos Probabilísticos:** Son aquellos que introducen elementos al azar dentro de su lógica.

  Tipos de algoritmos probabilísticos:

  - **Algoritmos Monte Carlo (Probable que mienta):** Da la respuesta exacta, pero puede dar una solución errada, con probabilidad baja (puede mentir).
  - **Algoritmos Las Vegas (Probable que NO encuentre solución):** Funciona similar a Monte Carlo, pero cuando no puede dar una respuesta correcta, lo admite (no miente, dice que no puede dar la respuesta correcta).

    Se distinguen 2 sub-tipos en general:

    - **Algoritmos de Sherwood:** Siempre encuentra una solución correcta. Si el azar no beneficia a la ejecución, esta tomará más tiempo.
    - **Algoritmos que, a veces, no dan respuesta:** A veces no es capaz de dar una solución, lo cual admite.
  - **Algoritmos Genéticos:** Es una metaheurística que se inspira en la evolución y selección natural para resolver problemas de optimización, busca elegir al individuo más fuerte.
    - **Fitness:** Función de aptitud, nos dice que tan apta es la solución para el problema (que tan cerca estamos de la solución real).
  - **Teoría de la Información:** Trata sobre clasificación de problemas, mide tiempo y espacio utilizando modelos de cómputo.
    - **Entropía(mínimo teórico):** Mide la cantidad de información que tiene un mensaje.

      $$h = \sum_{i=1}^{n} p_i \cdot \log_2 \left(\frac{1}{p_i} \right)$$

    - **Kolmogorov(mínimo real):** Mide la cantidad de información que tiene un mensaje.

      $$K = \text{mín} \{ l(M) : U(M) = x \}$$

      Donde $l(M)$ es la longitud del mensaje y $U(M)$ es el programa que genera el mensaje M.

- **Reducciones:** Son métodos para demostrar alguna característica de un problema.
  - **Reducción Turing:** Es una reducción para demostrar que un problema es imposible de resolver, usando otro problema que ya se sabe que es imposible de resolver. Ejemplo:

    $$\text{HP} \leq_T \text{EP}$$

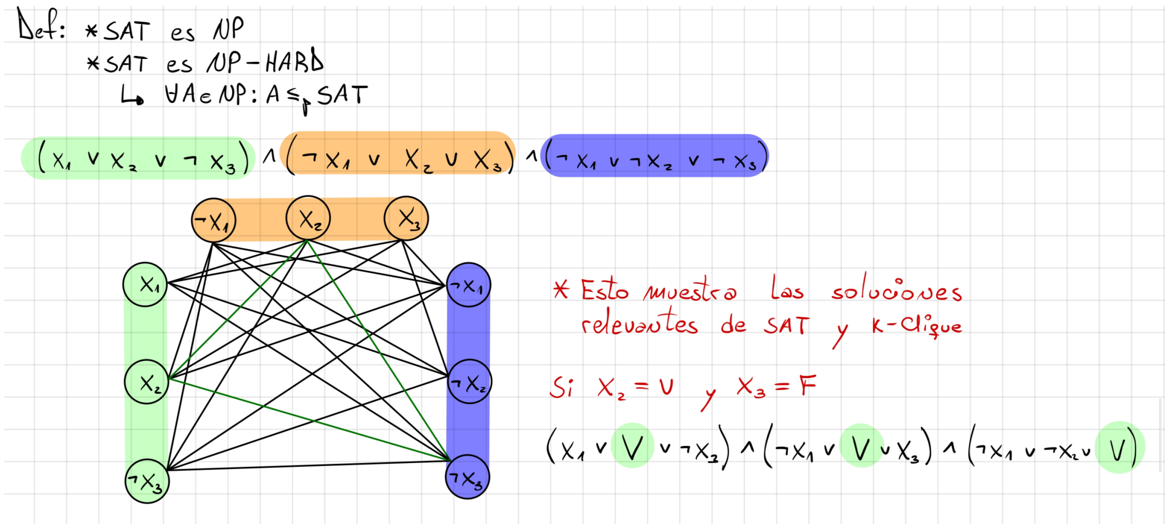
    Donde HP es el problema de parada y EP es el problema del vacío.

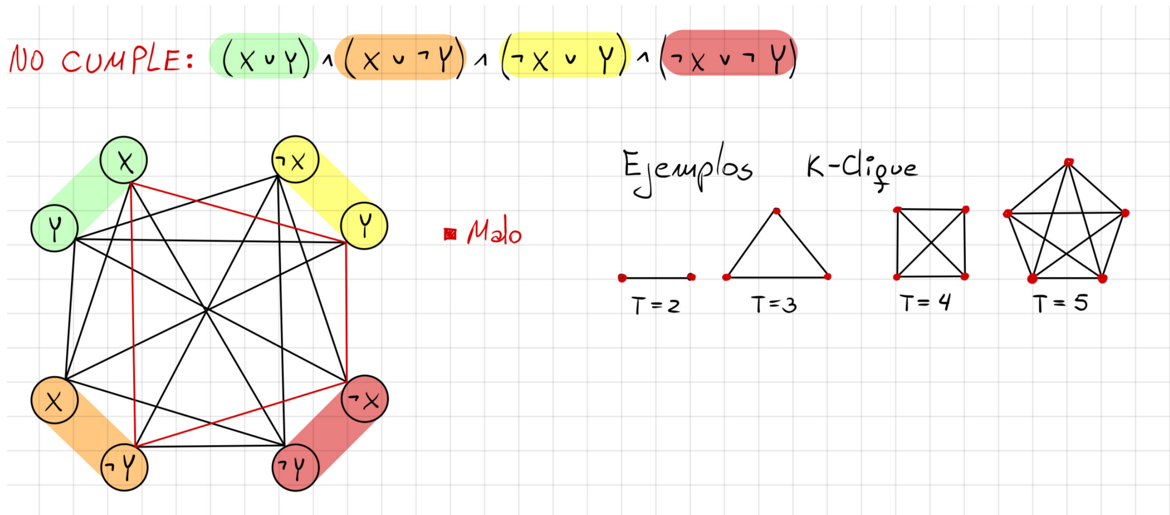
  - **Reducción Karp:** Es una reducción para demostrar que un problema es difícil de resolver, usando otro problema que ya se sabe que es difícil de resolver. Ejemplo:

    $$\text{3-SAT} \leq_p \text{K-Clique}$$

    Donde 3-SAT es el problema de satisfacción de 3 variables y K-Clique es el problema de encontrar un clique de tamaño K.

Problema visto en clase:





2. Ejercicios:

Certamen 2

1. Sean A, B, C matrices de $n \times n$. Considere el siguiente algoritmo probabilístico para saber si $A \cdot B = C$:

```
for (i = 1; i <= k; i++) {
    Generar un vector aleatorio x de n x 1 con entradas en {0 , 1}
    if (A*(Bx) != Cx)
        return false;
}
return true;
```

Responda:

- **a)** Tipo (MonteCarlo o las Vegas)

  Respuesta: MonteCarlo, pues el algoritmo puede mentir. Ejemplo:

  Sea A = $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, B = $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, C = $\begin{pmatrix} 0 & 2 \\ 0 & 4 \end{pmatrix}$, k = 1 y x = $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

$$A \cdot B =^? C$$
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} =^? \begin{pmatrix} 0 & 2 \\ 0 & 4 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \neq \begin{pmatrix} 0 & 2 \\ 0 & 4 \end{pmatrix}$$

  Primero:

$$\begin{aligned} A \cdot (B \cdot x) &= \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \\ &= \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 2 \\ 4 \end{pmatrix} \end{aligned}$$

  Luego:

$$\begin{aligned} C \cdot x &= \begin{pmatrix} 0 & 2 \\ 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 2 \\ 4 \end{pmatrix} \end{aligned}$$

  Como $A \cdot (B \cdot x) = C \cdot x$, el algoritmo retorna true, pero la respuesta correcta es que $A \cdot B \neq C$.

  Por por lo tanto, el algoritmo puede mentir, por lo que es MonteCarlo.

- **b)** Si $k \ll n$, es más rápido este algoritmo que simplemente multiplicar $A \cdot B$ y compararlo con C? Justifique con comportamiento asintótico.
  Respuesta: Si, pues el algoritmo tiene una complejidad de $O(k \cdot n^2)$, mientras que multiplicar $A \cdot B$ y compararlo con C tiene una complejidad de $O(n^3)$.

  Pues el algoritmo tiene un ciclo que se repite k veces, y en cada iteración se multiplica una matriz de $n \times n$ con un vector de $n \times 1$, lo cual tiene una complejidad de $O(n^2)$, por lo que la complejidad del algoritmo es $O(k \cdot n^2)$.

  Mientras que multiplicar $A \cdot B$ y compararlo con C tiene una complejidad de $O(n^3)$, pues se multiplica una matriz de $n \times n$ con otra matriz de $n \times n$, lo cual tiene una complejidad de $O(n^3)$, y luego se compara con otra matriz de $n \times n$, lo cual tiene una complejidad de $O(n^2)$, por lo que la complejidad de multiplicar $A \cdot B$ y compararlo con C es $O(n^3)$.

2. Sea una cadena de ADN de largo 10^9. La cantidad de adeninas en la cadena son 300 millones, timinas 200 millones, citosinas son 100 millones y guaninas el resto. Cuál es el mínimo teórico de información que posee la cadena?

   El mínimo teórico es la entropía, la cual se calcula de la siguiente forma:

$$h = \sum_{i=1}^{n} p_i \cdot \log_2\left(\frac{1}{p_i}\right)$$

   Para calcular la entropía, primero debemos calcular la probabilidad de cada elemento, la cual se calcula de la siguiente forma:

   Para adeninas:

$$\begin{aligned} p_a &= \frac{300 \cdot 10^6}{10^9} \\ &= 0,3 \end{aligned}$$

   Para timinas:

$$\begin{aligned} p_t &= \frac{200 \cdot 10^6}{10^9} \\ &= 0,2 \end{aligned}$$

   Para citosinas:

$$\begin{aligned} p_c &= \frac{100 \cdot 10^6}{10^9} \\ &= 0,1 \end{aligned}$$

   Para guaninas:

$$\begin{aligned} p_g &= 1 - p_a - p_t - p_c \\ &= 1 - 0,3 - 0,2 - 0,1 \\ &= 0,4 \end{aligned}$$

   Remplazando:

$$\begin{aligned} h &= \sum_{i=1}^{n} p_i \cdot \log_2\left(\frac{1}{p_i}\right) \\ &= \left(p_a \cdot \log_2\left(\frac{1}{p_a}\right) + p_t \cdot \log_2\left(\frac{1}{p_t}\right) + p_c \cdot \log_2\left(\frac{1}{p_c}\right) + p_g \cdot \log_2\left(\frac{1}{p_g}\right)\right) \\ &= \left(0,3 \cdot \log_2\left(\frac{1}{0,3}\right) + 0,2 \cdot \log_2\left(\frac{1}{0,2}\right) + 0,1 \cdot \log_2\left(\frac{1}{0,1}\right) + 0,4 \cdot \log_2\left(\frac{1}{0,4}\right)\right) \\ &= \left(0,3 \cdot \log_2\left(\frac{10}{3}\right) + 0,2 \cdot \log_2\left(\frac{10}{2}\right) + 0,1 \cdot \log_2\left(\frac{10}{1}\right) + 0,4 \cdot \log_2\left(\frac{10}{4}\right)\right) \end{aligned}$$

3. Backtracking

- Es una meta-heurística que trata de podar el árbol de búsqueda, el cuál se va generando de manera dinámica.
- Se descartan soluciones intermedias que se puede determinar no llegarán a una solución.
- La búsqueda se hace en profundidad.

  Para realizar backtracking, necesitamos:

  1. Punto de partida del árbol.
  2. Función de rechazo.
  3. Función de aceptación.
  4. Funciones de hijo (primero y siguiente).
  5. Función de Output (completar).

Algorithm 1: PseudoCódigo Backtracking

```
Función Backtracking(c):
    if reject(P,c) then
        return;
    if accept(P,c) then
        output(P,c);
        exit(0);
    s = first(P,c);
    while s ≠ ∧ do
        Backtracking(s);
        s = next(P,s);
```

Eficiencia de backtracking

- **Función de rechazo:** Mientras más cerca de la raíz, mejor.
- **Función de aceptación:** Mientras más cerca de la raíz, mejor.
- **Funciones de hijo (primero y siguiente):** Mientras más restrictiva, mejor.
- **Función de Output (completar):** Mientras más eficiente, mejor.

Cuando usar backtracking

- BackTracking está considerado para resolver Problemas de Satisfacción de Restricciones (CSP). Éste se define como los problemas consistentes de una tripleta (X, D, C).

  X= Conjunto de variables.

  D = Conjunto de dominios.

  C = Conjunto de restricciones.

- Cada variable $x_i \in d_i$ y debe satisfacer c_i, el cual está formado por una relación entre elementos de X.

Ejemplos de problemas CSP

Aritmética Verbal

Dado un problema de aritmética verbal, encontrar la solución.

Coloración de mapas

Dado un mapa, colorear las regiones de tal forma que dos regiones adyacentes no tengan el mismo color.

Crucigramas

Dado un crucigrama, encontrar las palabras que lo completan.

Sudoku

Dado un tablero de 9×9, colocar los números del 1 al 9 de tal forma que no se repitan en la misma fila, columna o submatriz de 3×3.

Problema de las n reinas

Dado un tablero de $n \times n$, colocar n reinas de tal forma que no se ataquen entre ellas.

4. Algoritmos Probabilísticos

- Los algoritmos probabilísticos son aquellos que introducen elementos al azar dentro de su lógica.
- Puede que el tiempo, la memoria o la respuesta sean afectados positivamente (o negativa con baja probabilidad) por el azar.

Tipos

- Algoritmos Numéricos.
  - Estos algoritmos dan una respuesta aproximada al problema que se quiere resolver.
  - Su precisión mejora conforme se realizan más ciclos de iteración.
- **Algoritmos Monte Carlo** (Puede mentir).
  - Se utilizan cuando no existen formas eficientes de resolver un problema de otra manera.
  - Estos algoritmos dan la respuesta exacta, pero puede dar una solución errada, con probabilidad baja.
  - Mientras más larga la ejecución, mayor es la probabilidad de que la respuesta se la correcta.

    Se le dice a un algoritmo tipo Monte Carlo p-correcto si:
    - La solución regresada es correcta con probabilidad p>0,5, no importando el dato de entrada.
    - p puede depender del tamaño de la entrada, pero no de los datos de la entrada.
- **Algoritmos Las Vegas** (No miente, dice que no puede dar la respuesta correcta).
  - El algoritmo tipo Las Vegas funciona similar a Monte Carlo, pero cuando no puede dar una respuesta correcta, lo admite.

    Se distinguen 2 sub-tipos en general:
    - Siempre encuentra una solución correcta. Si el azar no beneficia a la ejecución, esta tomará más tiempo.
    - A veces no es capaz de dar una solución, lo cual admite.

5. Algoritmos Genéticos

- Algoritmo genético es una metaheurística que se inspira en la evolución y selección natural para resolver problemas de optimización.
- En términos coloquiales, genera una población inicial y la hace evolucionar miles o millones de años, para finalmente elegir al individuo más fuerte.

En términos generales, los tópicos ligados a un algoritmo genético son los siguientes:

1. **Población Inicial:** La Población Inicial consta de soluciones (ya sean parciales y/o totales).
2. **Función de aptitud:**La Fitness Function o Función de Aptitud nos dice que tan apta es la solución para el problema.
3. **Algoritmo de combinación (sexo):** Básicamente acá es que los individuos con mejor fitness tienen mayores posibilidades de procrear.
4. **Algoritmo y tasa de Mutación:** Con alguna probabilidad (generalmente muy baja), cada bit de hijos puede cambiar.
   - Mutación bit por string.
   - Flip.
   - Límite (para números).
   - No uniforme.
   - Uniforme.
   - Gausiano.
   - Shrink.
5. **Selección:** Hay varias versiones sobre supervivencia a la siguiente generación. Una común es que sobrevivan los mejores (siempre mismo tamaño).

6. Búsqueda Informada (Heurística)

La búsqueda no informada trata ciegamente de encontrar una solución.

- Fuerte: Se usa heurística para tratar de resolver el problema lo mejor posible, pero no se asegura la solución.
- Débil: Heurística se conjuga con un método riguroso para llegar a la mejor solución. Muchas veces sigue siendo infactible de resolver en el peor caso.

7. Complejidad Computacional

- La complejidad computacional trata sobre clasificación de problemas.

  Dificultad inherente.

  Clases de complejidad y sus relaciones.
- Mide Tiempo y Espacio utilizando modelos de cómputo.

8. Teoría Algorítmica de la Información

9. Dureza, Completitud y Reducciones

10. Clases de Complejidad