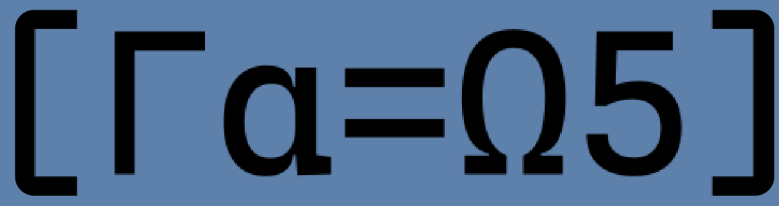
Tipos de Problemas y Paradigmas

Por Ariel Parra



> Hal Burch condujo un analisis en la primavera de 1999 y descubrio que solo existen **16** tipos de problemas para los concursos de programación competitiva.

1. **Dynamic Programming**: Un enfoque para resolver problemas complejos dividiéndolos en subproblemas más simples, almacenando los resultados de estos subproblemas para evitar cálculos redundantes. Se usa típicamente cuando el problema puede ser descompuesto en subproblemas que se solapan.
2. **Greedy (voraz)**: Un algoritmo voraz construye una solución al problema tomando siempre la decisión que parece mejor en ese momento. Un algoritmo codicioso nunca retira sus elecciones, sino que construye directamente la solución final. Por esta razón, los algoritmos codiciosos suelen ser muy eficientes.
3. **Complete Search**: Un método que explora exhaustivamente todas las posibles soluciones para encontrar la mejor. También conocido como búsqueda de fuerza bruta, es útil cuando el espacio de soluciones es relativamente pequeño.

4. **Flood Fill**: Un algoritmo que determina el área conectada a un punto dado en una matriz o gráfico. Se utiliza comúnmente en editores gráficos para rellenar áreas con un color.

5. **Shortest Path**: Un problema que busca encontrar el camino más corto entre dos nodos en un gráfico. Algoritmos conocidos incluyen Dijkstra y Bellman-Ford.

6. **Recursive Search Techniques**: Métodos que resuelven problemas mediante la descomposición en subproblemas similares, utilizando llamadas recursivas para explorar todas las posibles soluciones.

7. **Minimum Spanning Tree**: Un árbol de un gráfico conectado que incluye todos los vértices y tiene el peso total mínimo posible. Algoritmos famosos incluyen Kruskal y Prim.

8. **Knapsack**: Un problema de optimización donde se debe seleccionar un subconjunto de artículos con valor y peso, de manera que el peso total no exceda una capacidad máxima y el valor total sea máximo.

9. **Computational Geometry**: Una rama de la informática que estudia algoritmos para problemas geométricos, como encontrar la intersección de polígonos o la convexidad de un conjunto de puntos.
10. **Network Flow**: Un problema que involucra encontrar el flujo máximo en una red de nodos conectados por aristas, con capacidades específicas en cada arista. El algoritmo de Ford-Fulkerson es un método popular para resolver este problema.
11. **Eulerian Path**: Un camino en un gráfico que visita cada arista exactamente una vez. Si el gráfico es conexo y cada vértice tiene un grado par, el camino es un circuito euleriano.
12. **Two-Dimensional Convex Hull**: Un problema que busca encontrar el envolvente convexo de un conjunto de puntos en el plano bidimensional. El algoritmo de Graham y el algoritmo de Jarvis son técnicas comunes.

13. **BigNums**: Problemas que involucran cálculos con números de gran tamaño que no pueden ser representados por los tipos de datos estándar, requiriendo el uso de bibliotecas especializadas para aritmética de precisión arbitraria.

14. **Heuristic Search**: Algoritmos que buscan soluciones basadas en métodos aproximados en lugar de exactos, utilizando heurísticas para guiar la búsqueda hacia una solución buena en un tiempo razonable.

15. **Approximate Search**: Técnicas para encontrar soluciones aproximadas a problemas complejos donde la solución exacta es impracticable debido a limitaciones de tiempo o espacio.

16. **Ad Hoc Problems**: Problemas que no encajan fácilmente en ninguna categoría estándar de algoritmos, y requieren soluciones específicas para cada caso basadas en una comprensión profunda del problema.

Paradigmas de Resolución de algoritmos

No hay una fuente que diga de manera clara cuantos tipos de paradigmas existen, pero varias fuentes nombran hasta 12 paradigmas: Divide and Conquer, Dynamic Programming, Greedy Algorithms, Complete Search (Brute Force), Heuristic Search, Graph Search, Approximation Algorithms, Probabilistic Algorithms, Backtracking, Meet-in-the-Middle, Simulation & Game Theory.

Pero nostros nos enfocaramos principalmente en estos cuatro paradigmas:

- **Complete Search**: La Búsqueda Completa son métodos que implican una búsqueda exhaustiva a través de todas las posibilidades, como el backtracking y la **fuerza bruta** (brute force).
- **Divide and Conquer**: Dividir y Conquistar son técnicas que descomponen problemas en subproblemas más pequeños, los resuelven de manera independiente y combinan sus soluciones.
- **Greedy**: los algoritmos voraces son aquellos que toman decisiones localmente óptimas en cada paso con la esperanza de encontrar una solución global óptima.
- **Dynamic Programming** La Programación Dinámica son enfoques que resuelven problemas descomponiéndolos en subproblemas más simples y almacenando los resultados para evitar cálculos redundantes. (Usar más espacio para menos velocidad)

Tipo de problema y con que paradigma se puede resolver

Problem Type \ paradigms	Complete Search	Divide and Conquer	Greedy	Dynamic Programming
Ad-Hoc	✓	✓	✓	✓
Greedy			✓	
Computational Geometry	✓	✓		
Dynamic Programming				✓
BigNums				✓
Two-Dimensional	✓	✓		✓
Eulerian Path			✓	✓
Minimum Spanning Tree			✓	

Problem Type \ paradigms	Complete Search	Divide and Conquer	Greedy	Dynamic Programming
Knapsack			✓	✓
Network Flow			✓	✓
Flood Fill	✓			
Shortest Path	✓	✓	✓	✓
Approximate Search			✓	
Complete Search	✓			
Recursive Search Techniques	✓	✓		
Heuristic Search	✓		✓	

Tips para problemas Ad Hoc

1. **Escribe tus ideas**: Anota cualquier observación útil para asegurarte de no olvidarla y poder referirte a ella más tarde.
2. **No te quedes atascado**: Si una idea no te lleva a una solución completa, sigue explorando otras opciones para no perder tiempo.
3. **Dibuja casos pequeños**: Crear y analizar casos pequeños te ayudará a entender mejor el problema, especialmente si tienes dificultades para depurar, comenzar o avanzar en el problema.
4. **Explora diferentes perspectivas**: Aborda el problema desde varios ángulos. Experimenta con fórmulas, visualizaciones y distintos enfoques hasta que encuentres una solución.
5. **Identifica patrones en las soluciones**: Con el tiempo, reconoce patrones comunes en las soluciones de problemas. En programación competitiva, ciertos idioms y técnicas se repiten; guarda estos patrones en un archivo de encabezado para facilitar su uso futuro.

Etiquetas de problemas en Codeforces

1. **implementation**: Problemas que requieren soluciones directas y lógicas. (Ad Hoc)
2. **math**: Problemas con fórmulas y cálculos matemáticos. (Knapsack, BigNums, Computational Geometry)
3. **greedy**: Algoritmos que eligen soluciones locales óptimas esperando que sean globales.
4. **Dynamic Programming**: Resolución dividiendo en subproblemas y evitando cálculos repetidos.
5. **data structures**: Uso eficiente de estructuras de datos. (Complete Search, Ad Hoc)
6. **brute force**: Exploración exhaustiva de todas las soluciones posibles. (Complete Search)
7. **constructive algorithms**: Construcción explícita y paso a paso de la solución. (Ad Hoc)
8. **graphs**: Problemas sobre caminos y estructuras de grafos. (Flood Fill, Shortest Path, Recursive Search Techniques, Eulerian Path, Minimum Spanning Tree), (Network Flow)
9. **sortings**: Ordenamiento eficiente de datos. (Ad Hoc)
10. **binary search**: Búsqueda eficiente en conjuntos ordenados.(Complete Search, Greedy)
11. **dfs**: Algoritmos de búsqueda en profundidad y similares. (Recursive Search Techniques, Flood Fill)
12. **trees**: Problemas que involucran árboles jerárquicos. (Minimum Spanning Tree, Recursive Search Techniques)
13. **strings**: Manipulación de cadenas de caracteres. (Ad Hoc)
14. **number theory**: Propiedades numéricas como divisibilidad. (BigNums, Knapsack)

15. **combinatorics**: Cálculo de combinaciones y permutaciones. (Knapsack)
16. **special**: Problemas fuera de otras categorías. (Ad Hoc)
17. **geometry**: Problemas con figuras geométricas. (Computational Geometry , Two-Dimensional Convex Hull)
18. **bitmasks**: Uso de operaciones a nivel de bits. (Ad Hoc)
19. **two pointers**: Recorridos eficientes con dos punteros. (Greedy)
20. **dsu**: Estructura para particiones dinámicas. (Minimum Spanning Tree)
21. **shortest paths**: Encontrar caminos más cortos en grafos.
22. **probabilities**: Cálculo de probabilidades (Ad Hoc).
23. **divide and conquer**: División de problemas en partes más pequeñas.
24. **hashing**: Uso de funciones hash para mapeo eficiente. (Ad Hoc)
25. **games**: Problemas de teoría de juegos. (Ad Hoc)
26. **flows**: Encontrar flujos máximos en redes. (Network Flow)
27. **interactive**: Problemas con interacción dinámica. (Heuristic Search)
28. **matrices**: Manipulación de matrices. (Ad Hoc)

29. **string suffix structures**: Estructuras para procesar sufijos de cadenas. (Ad Hoc)
30. **fft**: Transformación rápida de Fourier. (BigNums)
31. **graph matchings**: Emparejamientos máximos en grafos bipartitos. (Network Flow, Minimum Spanning Tree)
32. **ternary search**: Búsqueda en funciones unimodales. (Greedy, Complete Search)
33. **expression parsing**: Análisis y evaluación de expresiones. (Ad Hoc)
34. **meet-in-the-middle**: Dividir el problema en dos partes y combinar soluciones. (Complete Search, Dynamic Programming)
35. **2-sat**: Resolver problemas de satisfacibilidad con 2 variables. (Ad Hoc)
36. **chinese remainder theorem**: Resolver sistemas de congruencias. (Ad Hoc, BigNums)
37. **schedules**: Optimización de secuencias y horarios. (Knapsack, Greedy)

Problems por tema en el Junior Training & SuperVision Sheet

> Junior Training & SuperVision son documentos de Google sheets creado por Mostafa Saad con multiples links.

- Para el Junior Training Consulte la página de la hoja (Topics1). Tiene las mismas hojas de problemas (CF-A a CF-D3) ordenados por categoría y nivel, son alrededor de 950 problemas.
- Columna Calidad de ideas: P5 (importante), P4 (muy interesante), P3 (interesante), P2 (mejor), P1 (bien), Vacío (normal)
- Puedes entrenar usando el orden ciego y usar la página de Temas como guía para saltarte algunos problemas.
- **Ventaja:** dominar el algoritmo hasta resolver algunos problemas difíciles en poco tiempo.
- **Desventaja:** descubrir el algoritmo detrás del problema es una habilidad importante. Dado que conoces el tema, pierdes espacio para mejorar esta habilidad.
- **Desventaja:** enfocarte en un algoritmo específico permite resolver muchos de ellos más fácilmente. Sin embargo, al resolver concursos reales, tu mente no está activa en el tema específico.
- Links para el: Junior Training Sheet ⮭ y el Supervision Problems Lists ⮭

Problemas

- **799A** Carrot Cakes ⮭
- **268A** Games ⮭

Referencias

- Sukhanov, N. (2014). *Codeforces tags distribution*. Recuperado de https://codeforces.com/blog/entry/14565
- Tiwari, A. (2021). *What are Ad Hoc Problems in Competitive Programming?*. Recuperado de https://www.geeksforgeeks.org/what-are-ad-hoc-problems-in-competitive-programming/
- USACO. (2013). *2.1 Programming Contest Problem Types*. Recuperado de https://github.com/voxsim/usaco/blob/master/section1.1/2.Programming Contest Problem Types.txt
- USACO. (2013). *2.1 Programming Contest Problem Types*. Recuperado de https://usaco.training/