20 Algoritmos de Fuerza Bruta - Explicacion y Ejemplos I/O

1. Busqueda lineal

Idea clave: Recorrer secuencialmente la lista hasta encontrar el objetivo o agotar los elementos.
Complejidad: O(n) tiempo, O(1) espacio
Cuando usar: Listas pequenas o no ordenadas, o cuando el costo de ordenar no compensa.
Notas/podas/alternativas: Si la lista esta ordenada, usar busqueda binaria.

Ejemplo:
Entrada: arr = [5, 2, 9, 1, 7], x = 1
Salida: indice = 3

2. Busqueda de patron en texto (naive)

Idea clave: Probar el patron en cada posicion posible del texto y comparar caracter a caracter.
Complejidad: O(n*m) tiempo, O(1) espacio
Cuando usar: Textos cortos o como baseline para comparar con KMP/Boyer-Moore.
Notas/podas/alternativas: Peor caso cuando el texto tiene muchos prefijos repetidos (ej: 'aaaa...').

Ejemplo:
Entrada: texto = 'abacaabaccabacaba', patron = 'abaca'
Salida: posiciones = [0, 10]

3. Subset Sum (suma de subconjuntos)

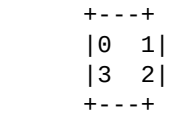
Idea clave: Enumerar todos los subconjuntos y revisar si su suma alcanza el objetivo.
Complejidad: O(2^n) tiempo, O(1)-O(n) espacio
Cuando usar: n pequeno (<= 25 aprox) o como baseline/benchmark.
Notas/podas/alternativas: Meet-in-the-middle y DP pueden acelerar; podas si numeros son positivos.

Ejemplo:
Entrada: nums = [3, 4, 5, 2], target = 7
Salida: existe = True, ejemplo = [3,4] o [5,2]

4. TSP por fuerza bruta

Idea clave: Probar todas las permutaciones de ciudades fijando un origen para evitar duplicados.
Complejidad: O(n!) tiempo, O(n) espacio
Cuando usar: n muy pequeno (<= 10) o para validar heuristicas.
Notas/podas/alternativas: Usar poda de costo parcial y simetrias; comparar con nearest neighbor.

Ejemplo:
Entrada: coords = [(0,0),(1,0),(1,1),(0,1)]
Salida: ruta optima (0->1->2->3->0) o equivalente, longitud = 4.0



5. Generacion de permutaciones

Idea clave: Listar todas las ordenaciones de n elementos.
Complejidad: O(n!*n) con impresion, O(n) espacio en backtracking
Cuando usar: Necesitas explorar todos los ordenes posibles de ejecucion o asignacion.
Notas/podas/alternativas: Backtracking clasico; evitar duplicados si hay elementos repetidos.

Ejemplo:
Entrada: A = [1,2,3]
Salida: permutaciones = [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

6. Problema de las N-Reinas (backtracking)

Idea clave: Colocar reinas fila por fila y retroceder si hay ataques.

Complejidad: Exponencial; soluciones crecen segun N
Cuando usar: Ejemplo tipico de poda y restricciones; ensenar backtracking.
Notas/podas/alternativas: Verificar columnas y diagonales antes de bajar de nivel.

Ejemplo:
Entrada: N = 4
Salida: soluciones (una): [1,3,0,2] (columnas por fila)

```
Fila-> 0 1 2 3
       . Q . .
       . . . Q
       Q . . .
       . . Q .
```

7. Busqueda de colisiones en hash (demostrativo)

Idea clave: Generar entradas hasta encontrar dos con el mismo hash en un espacio reducido.
Complejidad: O(k) tiempo segun tamano del espacio; memoria opcional
Cuando usar: Solo en demos educativas con hashes toy o espacios pequenos.
Notas/podas/alternativas: Para hashes criptograficos reales la fuerza bruta es impractica.

Ejemplo:
Entrada: hash(x) = x mod 10, buscar colision entre 0..50
Salida: colision: 7 y 17 (ambos dan 7)

8. Coin Change por enumeracion

Idea clave: Probar todas las combinaciones hasta alcanzar el monto exacto.
Complejidad: Exponencial en numero de monedas y monto
Cuando usar: Montos pequenos o como baseline frente a DP.
Notas/podas/alternativas: DP para contar o minimizar monedas es mas eficiente.

Ejemplo:
Entrada: monedas = [1,2,5], monto = 5
Salida: combinaciones validas: [5], [2,2,1], [2,1,1,1], [1,1,1,1,1]

9. Max Clique por enumeracion

Idea clave: Revisar todos los subconjuntos de vertices y verificar si forman un clique.
Complejidad: O(2^n) tiempo
Cuando usar: Grafos pequenos o para validar heuristicas.
Notas/podas/alternativas: Podar si el subconjunto actual no puede superar el mejor.

Ejemplo:
Entrada: G con V={A,B,C,D} y E={(A,B),(B,C),(A,C)}
Salida: clique maximo = {A,B,C} de tamano 3

10. Multiplicacion de matrices (naive)

Idea clave: Para cada celda C[i][j] sumar A[i][k]*B[k][j] sobre k.
Complejidad: O(n^3) tiempo, O(1) extra
Cuando usar: Matrices pequenas o como definicion base.
Notas/podas/alternativas: Strassen u otras tecnicas aceleran para matrices grandes.

Ejemplo:
Entrada: A=[[1,2],[3,4]], B=[[2,0],[1,2]]
Salida: C=[[1*2+2*1, 1*0+2*2],[3*2+4*1, 3*0+4*2]] = [[4,4],[10,8]]

11. Busqueda ingenua en lista no ordenada

Idea clave: Verificar cada posicion hasta hallar coincidencia.
Complejidad: O(n)
Cuando usar: Caso generico de busqueda sin preprocesamiento.

Notas/podas/alternativas: Equivalente a busqueda lineal.

Ejemplo:
Entrada: arr=['a','c','f','b'], x='f'
Salida: indice = 2

12. Closest Pair of Points (fuerza bruta)

Idea clave: Calcular distancia entre todos los pares y tomar la minima.
Complejidad: O(n^2) tiempo
Cuando usar: n moderado o para comparar con divide-and-conquer.
Notas/podas/alternativas: El algoritmo clasico O(n log n) es preferible para n grande.

Ejemplo:
Entrada: P={(0,0),(3,4),(7,1)}
Salida: par mas cercano = ((0,0),(3,4)), distancia = 5

13. Max Subarray ingenuo

Idea clave: Evaluar suma de todas las subarreglos posibles y elegir la maxima.
Complejidad: O(n^3) o O(n^2) con prefijos
Cuando usar: Listas pequenas o para ilustrar mejora hacia Kadane.
Notas/podas/alternativas: Kadane logra O(n).

Ejemplo:
Entrada: arr=[-2,1,-3,4,-1,2,1,-5,4]
Salida: max suma = 6, subarray = [4,-1,2,1]

14. SAT por enumeracion

Idea clave: Probar todas las asignaciones de n variables booleanas y chequear clausulas.
Complejidad: O(2^n)
Cuando usar: Formulas pequenas o testing.
Notas/podas/alternativas: SAT solvers modernos usan CDCL, heuristicas y learned clauses.

Ejemplo:
Entrada: F = (x1 OR NOT x2) AND (x2 OR x3) con n=3
Salida: satisfacible, por ejemplo x1=True, x2=False, x3=True

15. Mochila 0/1 ingenua

Idea clave: Probar incluir o no cada item y quedarse con el mejor valor sin exceder capacidad.
Complejidad: O(2^n) tiempo
Cuando usar: n pequeno o como baseline frente a DP.
Notas/podas/alternativas: DP pseudo-polinomial O(n*W) si pesos enteros.

Ejemplo:
Entrada: pesos=[2,3,4], valores=[4,5,6], W=5
Salida: mejor valor = 9 con items [2,3]

16. Sudoku solver (backtracking)

Idea clave: Rellenar celdas vacias probando 1..9 y retroceder si violan restricciones.
Complejidad: Exponencial en peor caso
Cuando usar: Instancias concretas, n pequeno (9x9).
Notas/podas/alternativas: Usar seleccion de variable mas restringida acelera mucho.

Ejemplo:
Entrada: Tablero 9x9 con ceros como vacios (ejemplo pequeno)
Salida: Tablero completo que cumple filas, columnas y cajas

```
Fila/Col -> restricciones de 1..9
```

20 Algoritmos de Fuerza Bruta - Explicacion y Ejemplos I/O

17. Hamiltonian path/cycle

Idea clave: Probar permutaciones de vertices y verificar si hay camino/ciclo que visita todos.
Complejidad: O(n!)
Cuando usar: Grafos pequenos o docencia de NP-completo.
Notas/podas/alternativas: Relacion cercano con TSP (sin pesos).

Ejemplo:
Entrada: G: 1-2-3-4 completo menos arista (2,4)
Salida: camino Hamiltoniano: 2-1-3-4

18. LCS ingenuo (Longest Common Subsequence)

Idea clave: Enumerar subsecuencias de una cadena y verificar si son subsecuencia de la otra.
Complejidad: O(2^n + 2^m) aprox
Cuando usar: Cadenas cortas; mostrar mejora con DP O(n*m).
Notas/podas/alternativas: No confundir con substring (contiguo).

Ejemplo:
Entrada: s='ABCBDAB', t='BDCABA'
Salida: una LCS posible = 'BCBA' de longitud 4

19. Generacion de combinaciones k

Idea clave: Listar todas las combinaciones de tamano k de un conjunto de n.
Complejidad: O(C(n,k))
Cuando usar: Exploracion exhaustiva de grupos o equipos.
Notas/podas/alternativas: Backtracking o iteradores lexicograficos.

Ejemplo:
Entrada: n=5, k=3, elementos=[A,B,C,D,E]
Salida: 10 combinaciones, ej: [A,B,C], [A,B,D], ...

20. Criptogramas aritmeticos

Idea clave: Asignar digitos a letras y probar todas las combinaciones respetando restricciones.
Complejidad: Factorial en numero de letras distintas
Cuando usar: Instancias pequenas o con muchas restricciones de poda.
Notas/podas/alternativas: Enviar acarreo, evitar ceros a la izquierda.

Ejemplo:
Entrada: SEND + MORE = MONEY
Salida: Solucion clasica: 9567 + 1085 = 10652