Pseudocode PD, Backtracking, Branch and Bound

Teorema de Optimalidad de Mitten

El objetivo básico en la programación dinámica consiste en 'descomponer" un problema de optimización en k variables a una serie de problemas con menor número de variables más fáciles de resolver.

Levenstein(O(m*n))

Depen de l'objectiu del problemes

```
for i = 0, 1, 2, ..., m:
    E(i, 0) = i
for j = 1, 2, ..., n:
    E(0, j) = j
for i = 1, 2, ..., m:
    for j = 1, 2, ..., n:
        E(i, j) = min{E(i − 1, j) + 1, E(i, j − 1) + 1, E(i − 1, j − 1) + diff(i, j)}
return E(m, n)
```

Floyd-Washall (O(V**3))

```
for i = 1 to n:
   for j = 1 to n:
      dist(i, j, 0) = ∞
for all (i, j) ∈ E:
   dist(i, j, 0) = ℓ(i, j)
for k = 1 to n:
   for i = 1 to n:
      for j = 1 to n:
         dist(i, j, k) = min{dist(i, k, k − 1) + dist(k, j, k − 1),  dist(i, j, k − 1)}
```

Knapsack PD (O(n*W))

```python
# A Dynamic Programming based Python Program for 0-1 Knapsack problem
# Returns the maximum value that can be put in a knapsack of capacity W
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W+1)] for x in range(n+1)]

    # Build table K[][] in bottom up manner
    for i in range(n+1):
        for w in range(W+1):
            if i==0 or w==0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]
```

TSP PD(O(n**2*2**n))

```
function algorithm TSP (G, n)
  for k := 2 to n do
    C({k}, k) := d_{1,k}
  end for

  for s := 2 to n-1 do
    for all S ⊆ {2, . . . , n}, |S| = s do
      for all k ∈ S do
        C(S, k) := min_{m≠k,m∈S} [C(S\{k}, m) + d_{m,k}]
      end for
    end for
  end for

  opt := min_{k≠1} [C({2, 3, . . . , n}, k) + d_{k,1}]
  return (opt)
end
```

Backtracking(Global Strategy)

```
algorithm backtrack():
  if (solution == True)
    return True
  for each possible moves
    if(this move is valid)
      select this move and place
      call backtrack()
      unplace that selected move
      increment the given choice in the for loop
    else
      return False
```

Backtracking Grafos(Strategy)

```
Bactracking Enum(X,num)
  variables L: ListaComponentes
  inicio
     si EsSolución (X) entonces num = num+1
        mostrarSolución (X)
     sino
        L  = Candidatos (X)
        mientras ¬Vacía (L) hacer
           X[i + 1] =  Cabeza (L); L =  Resto (L)
           BacktrackingEnum (X, num)
```

```
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.
   Do following for every tried row.
    a) If the queen can be placed safely in this row
       then mark this [row, column] as part of the
       solution and recursively check if placing
       queen here leads to a solution.
    b) If placing the queen in [row, column] leads to
       a solution then return true.
    c) If placing queen doesn't lead to a solution then
       unmark this [row, column] (Backtrack) and go to
       step (a) to try other rows.
3) If all rows have been tried and nothing worked,
   return false to trigger backtracking.
```