Dynamic Programming ---Coins in a line

Telescope Scheduling

Coin in a line

0/1 Knapsack

How many coins must be in a line

In this game, an **even number**, n, of coins, **of various face value**, are **placed in a line**.

Does this require dynamic programming or greedy algorithm, explain and benefit of dynamic or greedy w.r.t. this problem

False Start 1: Greedy Method

A natural greedy strategy is "always choose the largest valued available coin."

- But this doesn't always work:
  - [5, 10, 25, 10]: Alice chooses 10
  - [5, 10, 25]: Bob chooses 25
  - [5, 10]: Alice chooses 10
  - [5]: Bob chooses 5
- Alice's total value: 20, Bob's total value: 30. (Bob wins, Alice loses)

- Applies to a problem that at first seems to require a lot of time (possibly exponential), provided we have:
  - **Simple subproblems:** the subproblems can be defined in terms of a few variables, such as j, k, l, m, and so on.
  - **Subproblem optimality:** the global optimum value can be defined in terms of optimal subproblems
  - **Subproblem overlap:** the subproblems are not independent, but instead they overlap (hence, should be constructed bottom-up).

Explain the problem and goal

In this game, an **even number**, n, of coins, **of various face value**, are **placed in a line**.

Alice and Bob are 2 players. They are required to choose coins only from leftist side or rightest side. Alice firstly chooses, try her best to do dynamic optimal strategy to make herself get maximum total value than Bob, and Bob will choose coin optimally to minimize what Alice can get next time.

The goal is to find an optimal selection strategy to let Alice get Maximum total value in this game than Bob.

Explain the players strategies

Alice must choose based on the following reasoning:

Case 1: Base case:

If j = i + 1, then she should pick the larger of V[i] and V[j], and the game is over.

$$M_{i,i+1} = \max\{V[i], V[i+1]\}.$$

Case 2:

If Alice chooses coin i, then Bob will choose V[i + 1] or V[j], and he will do a strategy to minimize total coins value that Alice can get after. So in this case, Alice can get:

$$\min\{M_{i+1,j-1},\ M_{i+2,j}\} + V[i].$$

Case 3:

If Alice chooses coin j, then Bob will choose V[i] or V[j-1]], and he will do a strategy to minimize total coins value that Alice can get after. So in this case, Alice can get:

$$\min\{M_{i,j-2},\ M_{i+1,j-1}\} + V[j].$$

So:

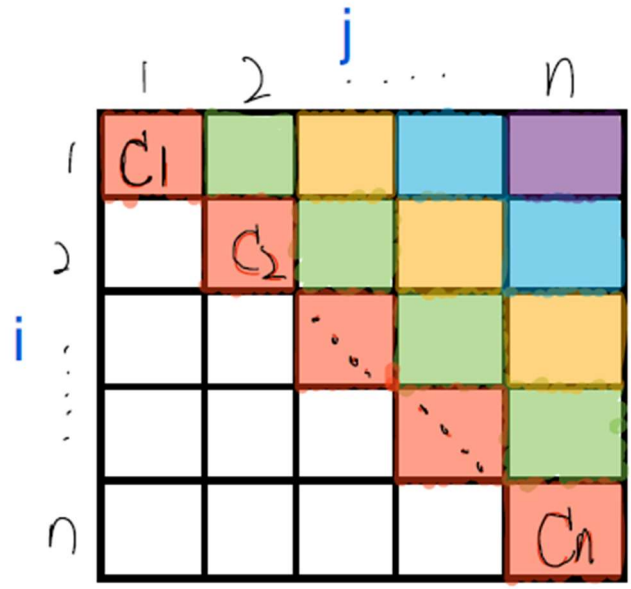
$$M_{i,j} = \begin{cases} \text{the maximum value of coins taken by Alice, for coins} \\ \text{numbered } i \text{ to } j, \text{ assuming Bob plays optimally.} \end{cases}$$

Therefore, the optimal value for Alice is determined by $M_{1,n}$.

$$M_{i,j} = \max\left\{\min\{M_{i+1,j-1},\ M_{i+2,j}\} + V[i],\ \min\{M_{i,j-2},\ M_{i+1,j-1}\} + V[j]\right\}.$$

In addition, for $i = 1, 2, \ldots, n-1$, we have the initial conditions

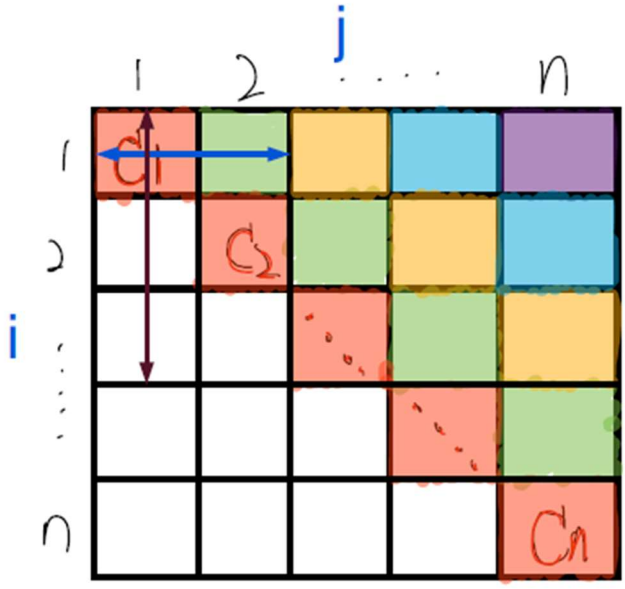
Give examples of



Base case 1, we just have only 1 coin, Alice just choose C[i].

Base case 2, choose from 2 coins, Alice can do, Mi,i+1 = Max{C[i], C[i+1]}.

Because for each Mi,j i >= j，那么这个矩阵只会被填充上半区。



For orange，length = ⟷ = 2, 对于所有orange这一层的循环，其j的大小会随着i的变化而变化，也就是j = i + length，而i的变化范围也如图所示为n-length。

You do not have to memorize M(i,j) calculation, but do be able to explain the players strategies

Given its code, explain it

```
Algo: Coins (C)
Input: list C of integer n≥ 0 coins
Output: maximum value that Alice, the first player, can obtain from C
        assuming both players play perfectly

M ← new n x n Matrix
for i← to n
  M[i, i] ← C(i)   // only one coin, pick it (base case)

for i ← 1 to (n-1) do // two coins to choose from (base case)
     M[i, i+1] ← max { C(i), C(i+1) }

for Length ← 2 to (n-1) do
  for i ← 1 to (n – Length)
    j ← i + Length
    M[i, j] ← max {
                C(i) + { min { M[i+2, j], m[i+1, j-1] } ,
                C(j) + { min { M[i+1, j-1], M[i, j-2] } }
return M[1,n]
```

Algorithm runtime and explain it

Since there are O(n) iterations in this algorithm and each iteration runs in O(n) time, the total time for this algorithm is O(n^2).

Greedy or dynamic and why

Dynamic programming is better than Greedy Algorithm for the "Coins in a Line" problem.

Reason:

DP can construct simple subproblems, create subproblem optimality, and solve the trouble of subproblem overlap. In this context, DP can globally explore all possible choices and consequences, ensuring a globally optimal solution.

While Greedy Algorithm seems faster, it may not guarantee global optimality, because it always does locally optimal choices without considering the global impact.

通用回答，仅需替换红色部分问题名字.