1406. Stone Game III (Recursion - Memo - Bottom Up - O(1)Space)

27 May 2023 06:47 AM

1406. Stone Game III

Hint

Hard 1.3K 31

Companies

Alice and Bob continue their games with piles of stones. There are several stones **arranged in a row**, and each stone has an associated value which is an integer given in the array `stoneValue`.

Alice and Bob take turns, with Alice starting first. On each player's turn, that player can take `1`, `2`, or `3` stones from the **first** remaining stones in the row.

The score of each player is the sum of the values of the stones taken. The score of each player is `0` initially.

The objective of the game is to end with the highest score, and the winner is the player with the highest score and there could be a tie. The game continues until all the stones have been taken.

Assume Alice and Bob **play optimally**.

Return `"Alice"` *if Alice will win,* `"Bob"` *if Bob will win, or* `"Tie"` *if they will end the game with the same score.*

Example 1:

```
Input: values = [1,2,3,7]
Output: "Bob"
Explanation: Alice will always lose. Her best move will be
to take three piles and the score become 6. Now the score
of Bob is 7 and Bob wins.
```

Example 2:

```
Input: values = [1,2,3,-9]
Output: "Alice"
Explanation: Alice must choose all the three piles at the
first move to win and leave Bob with negative score.
If Alice chooses one pile her score will be 1 and the next
move Bob's score becomes 5. In the next move, Alice will
take the pile with value = -9 and lose.
If Alice chooses two piles her score will be 3 and the
next move Bob's score becomes 3. In the next move, Alice
will take the pile with value = -9 and also lose.
Remember that both play optimally so here Alice will
choose the scenario that makes her win.
```

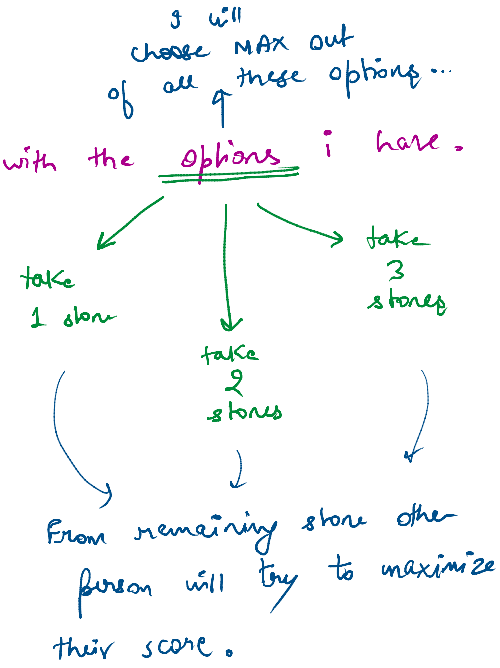
Example 3:

Input: values = [1,2,3,6]
Output: "Tie"
Explanation: Alice cannot win this game. She can end the game in a draw if she decided to choose all the first three piles, otherwise she will lose.

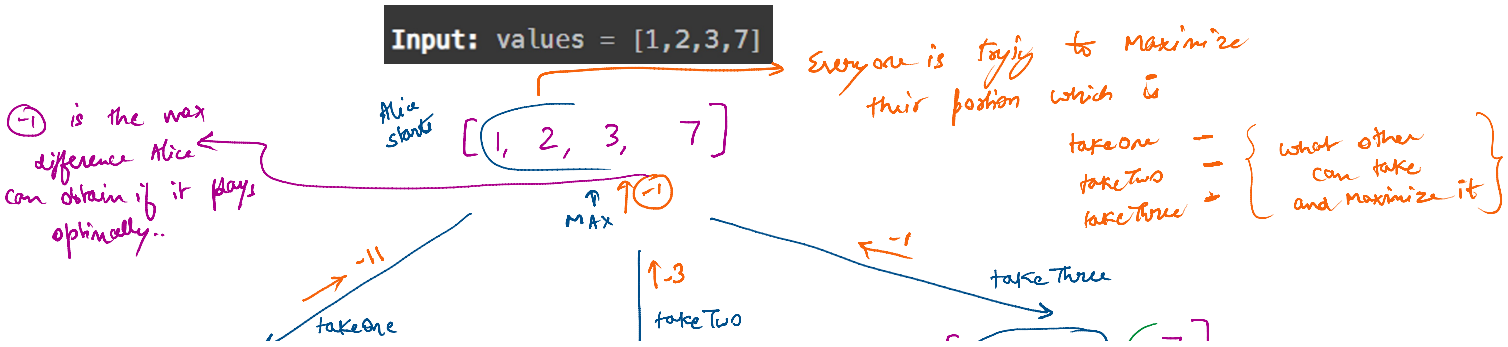
Constraints:

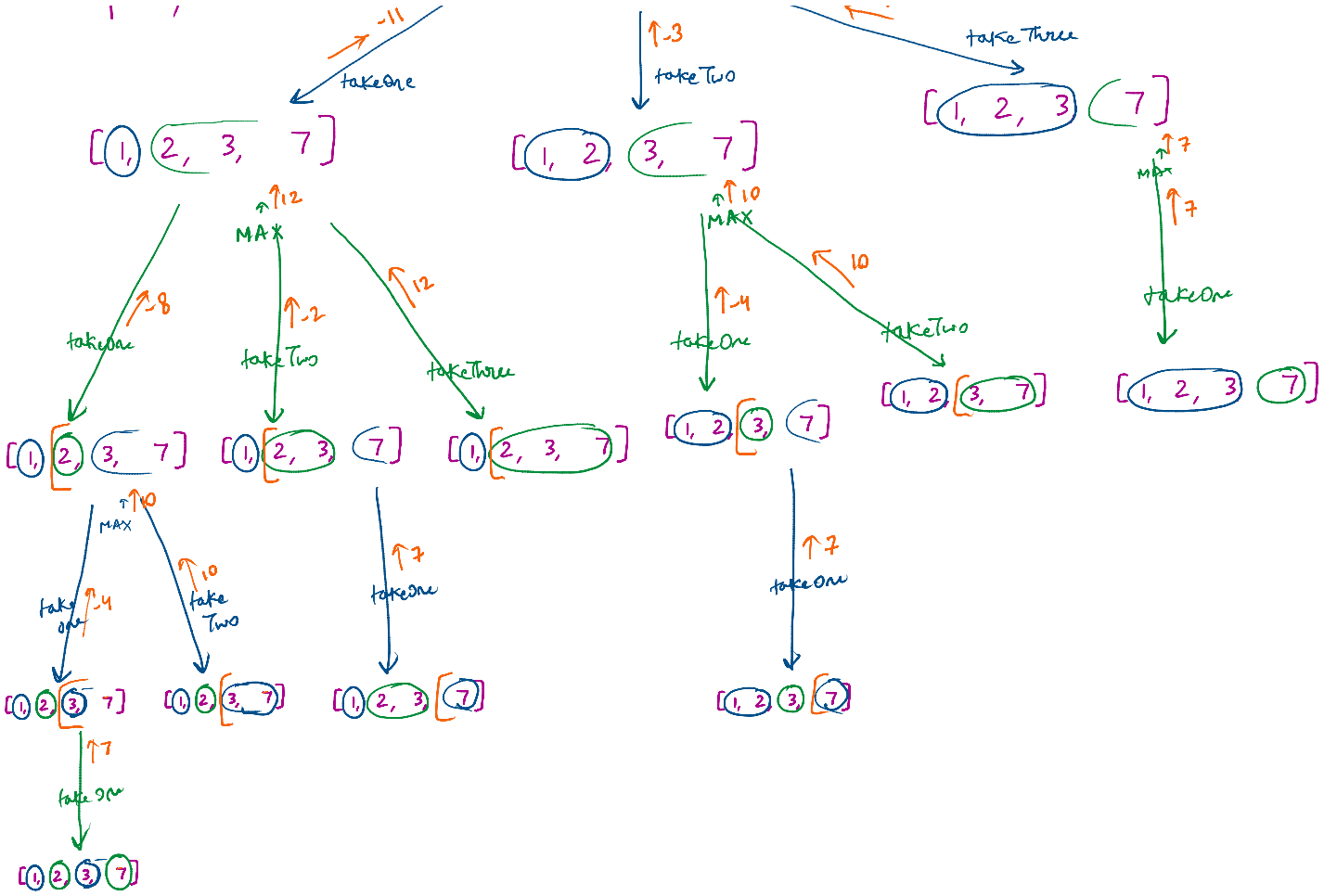
- `1 <= stoneValue.length <= 5 * 10^4`
- `-1000 <= stoneValue[i] <= 1000`

⇒ If we just try to build the tree with the options i have.

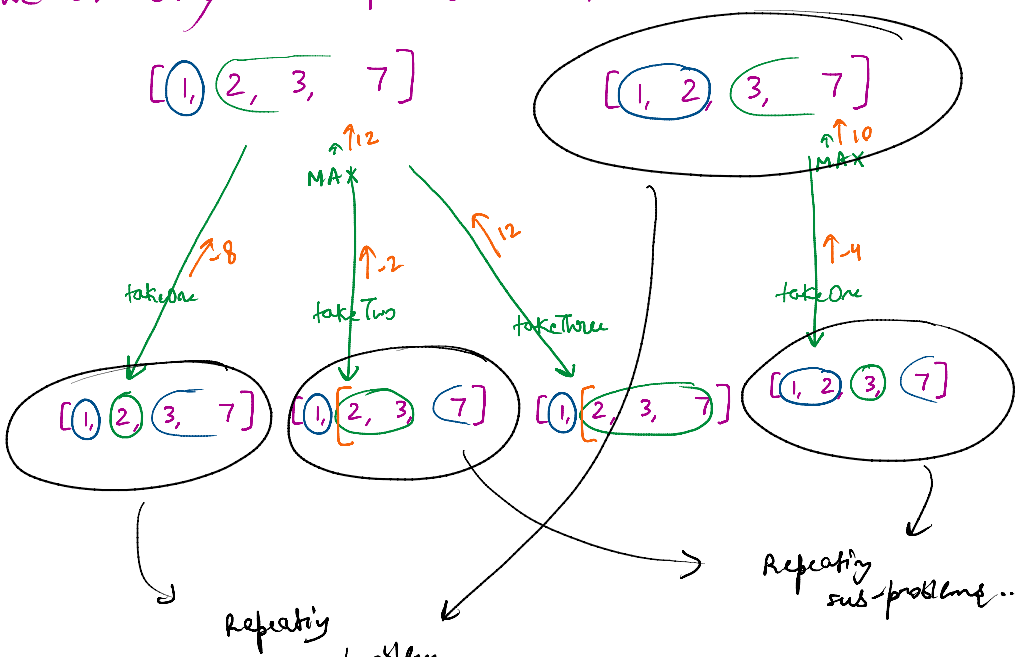


Input: values = [1,2,3,7]





We can easily see repeating sub-problems :)



Repeating sub-problems..

Repeating sub-problems..

Thus, we can think of Simple Recursion & then apply Memoization on top of it !!

Recursion :

```
class Solution {
public:
    int stoneGame(const vector<int>& stoneValue, int i)
    {
        int n = stoneValue.size();
        if(i>=n) return 0;

        int takeOne = stoneValue[i] - stoneGame(stoneValue, i+1);

        int takeTwo = INT_MIN;
        if(i+1 < n) takeTwo = stoneValue[i] + stoneValue[i+1] - stoneGame(stoneValue, i+2);

        int takeThree = INT_MIN;
        if(i+2 < n) takeThree = stoneValue[i] + stoneValue[i+1] + stoneValue[i+2] - stoneGame(stoneValue, i+3);

        return max({takeOne, takeTwo, takeThree});
    }

    string stoneGameIII(vector<int>& stoneValue)
    {
       int value = stoneGame(stoneValue,0);
       if(value > 0)
          return "Alice";
       else if(value  < 0)
          return "Bob";
       else
          return "Tie";
    }
};
```

Base case when your index reaches beyond array bounds.

take One stone i.e. Current stone

Considering the fact that other person will also try to maximize their score

take two stones i.e. this & next

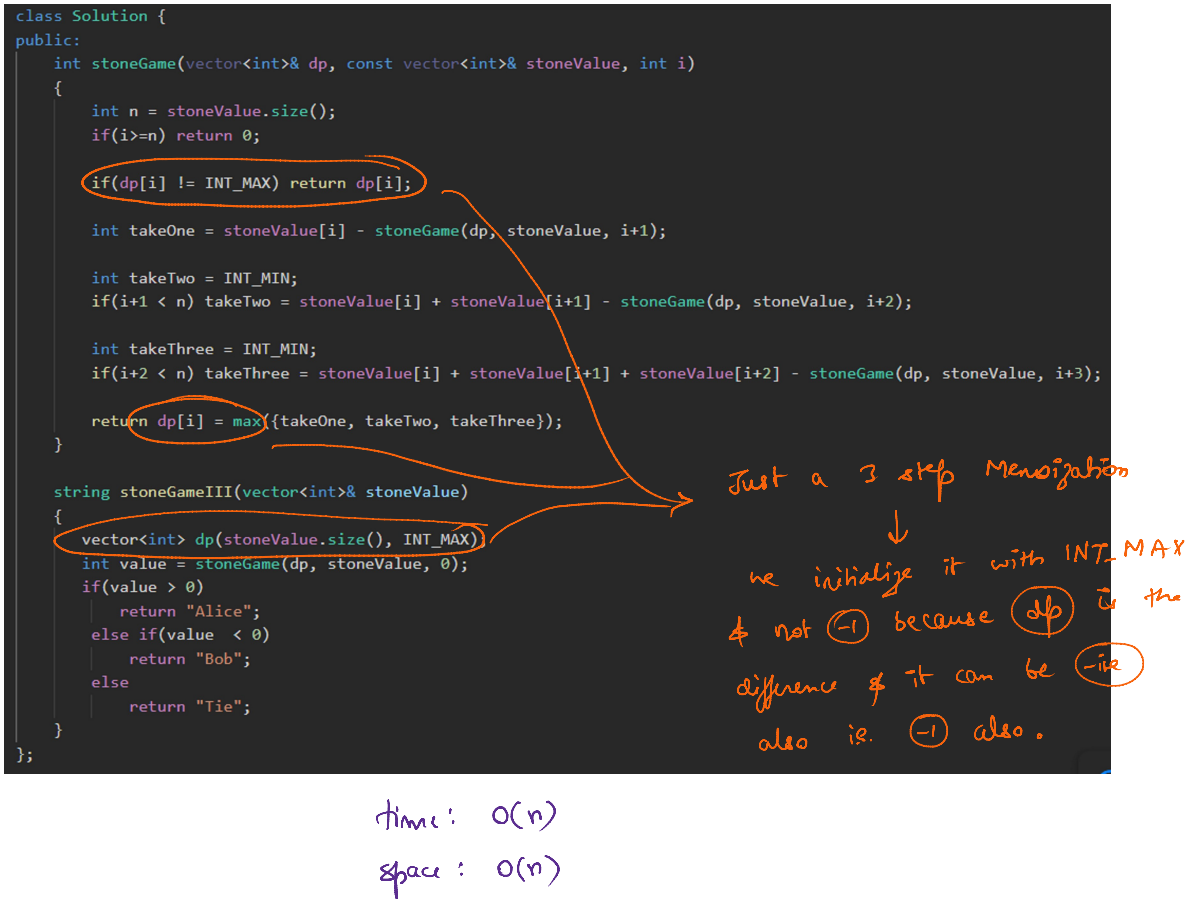
take three stones i.e. this, next, next to next

↳ take max of what you can take.

Finding the max difference possible & As Alice is starting first thus this difference is of (Alice - Bob)

depending on difference

Memoization :



time: O(n)

space: O(n)

Code:

```
class Solution {
public:
    int stoneGame(vector<int>& dp, const vector<int>& stoneValue, int i)
    {
        int n = stoneValue.size();
        if(i>=n) return 0;
        if(dp[i] != INT_MAX) return dp[i];
        int takeOne = stoneValue[i] - stoneGame(dp, stoneValue, i+1);
        int takeTwo = INT_MIN;
        if(i+1 < n) takeTwo = stoneValue[i] + stoneValue[i+1] - stoneGame(dp, stoneValue, i+2);
        int takeThree = INT_MIN;
        if(i+2 < n) takeThree = stoneValue[i] + stoneValue[i+1] + stoneValue[i+2] - stoneGame(dp, stoneValue,
i+3);

        return dp[i] = max({takeOne, takeTwo, takeThree});
    }
    string stoneGameIII(vector<int>& stoneValue)
    {
       vector<int> dp(stoneValue.size(), INT_MAX);
       int value = stoneGame(dp, stoneValue, 0);
       if(value > 0)
           return "Alice";
        else if(value  < 0)
            return "Bob";
        else
            return "Tie";
    }
};
```

⇒ Can we optimize it ? → To optimize DP → we would need to convert to Tabulation (Bottom Up)

"Simple" Conversion of Recursive code ⟷ Only then we can think of Optimizations :)

Bottom Up:

```
class Solution {
public:
    string stoneGameIII(vector<int>& stoneValue)
    {
        int n = stoneValue.size();
        vector<int> dp(stoneValue.size()+1, 0);

        for (int i = n - 1; i >= 0; i--) {
            int takeOne = stoneValue[i] - dp[i + 1];

            int takeTwo = INT_MIN;
            if (i + 1 < n) takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[i + 2];

            int takeThree = INT_MIN;
            if (i + 2 < n) takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[i + 3];

            dp[i] = max({takeOne, takeTwo, takeThree});
        }

        int value = dp[0];
        if(value > 0)
            return "Alice";
        else if(value  < 0)
            return "Bob";
        else
            return "Tie";
    }
};
```

Going from back as the values from back will be used

Exactly same as Recursion

Exact same stuff.

time : O(n)
space : O(n)

CODE:

```
class Solution {
public:
    string stoneGameIII(vector<int>& stoneValue)
    {
        int n = stoneValue.size();
        vector<int> dp(stoneValue.size()+1, 0);
        for (int i = n - 1; i >= 0; i--) {
            int takeOne = stoneValue[i] - dp[i + 1];
            int takeTwo = INT_MIN;
            if (i + 1 < n) takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[i + 2];
            int takeThree = INT_MIN;
            if (i + 2 < n) takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[i + 3];
            dp[i] = max({takeOne, takeTwo, takeThree});
        }
        int value = dp[0];
        if(value > 0)
            return "Alice";
        else if(value  < 0)
            return "Bob";
        else
            return "Tie";
```

```
    }
};
```

⇒ we can easily see to compute dp(i) we only need

```
for (int i = n - 1; i >= 0; i--) {
    int takeOne = stoneValue[i] - dp[i + 1];

    int takeTwo = INT_MIN;
    if (i + 1 < n) takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[i + 2];

    int takeThree = INT_MIN;
    if (i + 2 < n) takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[i + 3];

    dp[i] = max({takeOne, takeTwo, takeThree});
}
```

thus, we can only take space of (3) rather than taking entire space of (n)

thus Reducing the space from O(n) to O(1)

Botanup optimized...

```
class Solution {
public:
    string stoneGameIII(vector<int>& stoneValue)
    {
        int n = stoneValue.size();
        vector<int> dp(3, 0);

        for (int i = n - 1; i >= 0; i--) {
            int takeOne = stoneValue[i] - dp[(i + 1) % 3];

            int takeTwo = INT_MIN;
            if (i + 1 < n) takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[(i + 2) % 3];

            int takeThree = INT_MIN;
            if (i + 2 < n) takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[(i + 3) % 3];

            dp[i % 3] = max({takeOne, takeTwo, takeThree});
        }

        int value = dp[0];
        if(value > 0)
            return "Alice";
        else if(value  < 0)
            return "Bob";
        else
            return "Tie";
    }
};
```

Only code change to use only space of (3) → constant rather than the space of (n) :)

time: O(n)

Space : O(1)

CODE:

```
class Solution {
public:
    string stoneGameIII(vector<int>& stoneValue)
    {
       int n = stoneValue.size();
       vector<int> dp(3, 0);
       for (int i = n - 1; i >= 0; i--) {
            int takeOne = stoneValue[i] - dp[(i + 1) % 3];
            int takeTwo = INT_MIN;
            if (i + 1 < n) takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[(i + 2) % 3];
            int takeThree = INT_MIN;
            if (i + 2 < n) takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[(i + 3) % 3];
            dp[i % 3] = max({takeOne, takeTwo, takeThree});
        }
       int value = dp[0];
       if(value > 0)
           return "Alice";
        else if(value  < 0)
            return "Bob";
        else
            return "Tie";
    }
};
```

C++

```
class Solution {
public:
    string stoneGameIII(vector<int>& stoneValue)
    {
       int n = stoneValue.size();
       vector<int> dp(3, 0);
       for (int i = n - 1; i >= 0; i--) {
            int takeOne = stoneValue[i] - dp[(i + 1) % 3];
            int takeTwo = INT_MIN;
            if (i + 1 < n) takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[(i + 2) % 3];
            int takeThree = INT_MIN;
            if (i + 2 < n) takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[(i + 3) % 3];
            dp[i % 3] = max({takeOne, takeTwo, takeThree});
        }
       int value = dp[0];
       if(value > 0)
           return "Alice";
        else if(value  < 0)
            return "Bob";
        else
            return "Tie";
    }
};
```

JAVA

```java
class Solution {
    public String stoneGameIII(int[] stoneValue) {
        int n = stoneValue.length;
        int[] dp = new int[3];
        for (int i = n - 1; i >= 0; i--) {
            int takeOne = stoneValue[i] - dp[(i + 1) % 3];
            int takeTwo = Integer.MIN_VALUE;
            if (i + 1 < n)
                takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[(i + 2) % 3];
            int takeThree = Integer.MIN_VALUE;
            if (i + 2 < n)
                takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[(i + 3) % 3];
            dp[i % 3] = Math.max(Math.max(takeOne, takeTwo), takeThree);
        }
        int value = dp[0];
        if (value > 0)
            return "Alice";
        else if (value < 0)
            return "Bob";
        else
            return "Tie";
    }
}
```

PYTHON:

```python
class Solution:
    def stoneGameIII(self, stoneValue):
        n = len(stoneValue)
        dp = [0] * 3
        for i in range(n - 1, -1, -1):
            takeOne = stoneValue[i] - dp[(i + 1) % 3]
            takeTwo = float('-inf')
            if i + 1 < n:
                takeTwo = stoneValue[i] + stoneValue[i + 1] - dp[(i + 2) % 3]
            takeThree = float('-inf')
            if i + 2 < n:
                takeThree = stoneValue[i] + stoneValue[i + 1] + stoneValue[i + 2] - dp[(i + 3) % 3]
            dp[i % 3] = max(takeOne, takeTwo, takeThree)
        value = dp[0]
        if value > 0:
            return "Alice"
        elif value < 0:
            return "Bob"
        else:
            return "Tie"
```