

"DP on stocks"

DP-35

Best time to buy and sell stock.

Problem

Best time to buy and sell stock ..

arr[] → [7, 1, 5, 3, 6, 4]

Maximize the profit

Brute force → try every possible sell and buy way and find the maximum profit but time → $O(N^2)$

Idea

7 1 5 3 6 4.

int maxprofit (vector<int> nums)

{

 int mini = nums[0];

 int ans = 0;

 int n = nums.size();

 for (int i=1; i<n; i++) {

 int cost = num(i) - mini;

 ans = max(ans, cost);

 mini = min(mini, num(i));

 }

 return ans;

"if you are selling it on the ith day. you must buy it on the minimum price from (1st - (i-1)th day)"

Time complexity → $O(n)$

DP-36

Best time to buy and sell stocks, II

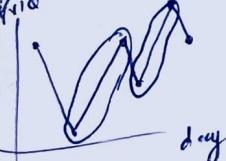
Same problem just add condition.

can purchase and sell any number of times

my solution

except
(7 1 5 3 6 4)

Price



int maxprofit (vector<int> nums)

 int n = nums.size();

 int maxi = 0;

 for (int i=0; i<n-1; i++) {

 if (num[i+1] > num[i]) {

 maxi += (num[i+1] - num[i]));

 }

 } else { continue; }

 return maxi;

Logic

We must consider only that part which give us benefit

$O(n)$ $O(1)$

official example $\begin{matrix} \uparrow & 1 & 5 & 3 & 6 & 4 \end{matrix}$ (can buy and sell at any date)
 so we have (a lot of ways) → Try all ways → (Best answer)

How to write recursion?

(i) Express in term of (index, buy) variable that tell us either i can buy or not.

(ii) Explore possibilities on that day like

(iii) Take the max profit made.

(iv) Base case

(i) $f(\text{mid}, \text{buy})$

$f(0, 1) \rightarrow$ [starting on 0th day with buy 1]
 can give me np maximum profit]

(buy + 1 → can buy
 → 0 → can't buy)

(ii) $y(\text{buy}) \{$ take / not take
 profit¹ = $\{$
 else $\{$ profit² = \sim

$y(\text{buy}) \{$ take / not take
 profit¹ = $\{$
 else $\{$ profit² = \sim

else {
 profit = max {
 prices(mid) + $f(\text{mid}+1, 0)$ } → take (purchased)
 } → not take (not purchased)
 } → sold
 } → don't sell

(ii) Base case → when array is finished no more profit

$y(\text{mid} = n) \text{ return } 0;$

Memorization and Recursion code

```
int fun<int mid, int buy, vector<int> &nums, vector<vector<int>> &dp)
{
    if (mid == n) return 0;
    if (dp[mid][buy] != -1) return dp[mid][buy];
    int profit = 0;
    if (buy)
        profit = max(-nums[mid] + fun(mid+1, 0, nums, n, dp),
                      0 + fun(mid+1, 1, nums, n, dp));
    else {
        Profit = max(-nums[mid] + fun(mid+1, 1, nums, n, dp),
                     0 + fun(mid+1, 0, nums, n, dp));
    }
    return dp[mid][buy] = profit;
}
```

Tabulation

steps to convert memorization to tabulation

(i) convert base case $dp(0) = 0$ (and $i=n$) return 0, $dp(n)(0) = dp(n)(1) = 0$

(ii) write loop condition

(i) we are traversing from n to 0

and $dp(0)(1)$ would be our final ans

for (int i = n-1; i >= 0; i--)

for (int j = 0; j < 2; j++)

return $dp(0)(1)$

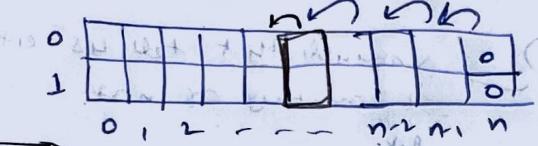


Fig-1

(iii) copy memo, recursion condition

(iv) convert fun(-) into dp to complete code

Code

```

int maxProfit(vector<int>& nums) {
    int n = nums.size();
    vector<vector<int>> dp(n+1, vector<int>(2, 0));
    dp[0][0] = dp[0][1] = 0;
    for (int i = n-1; i >= 0; i--) {
        for (int j = 0; j < 2; j++) {
            int profit = 0;
            if (j == 0)
                profit = max(-nums[i] + dp[i+1][0], dp[i+1][1]);
            else
                profit = max(-nums[i] + dp[i+1][1], dp[i+1][0]);
            dp[i][j] = profit;
        }
    }
    return dp[0][1];
}

```

Optimization

we just need only previous ($i+1$) data to calculate next step
so we can space optimize it

also we can do time optimization using first approach

to make it solve in $O(n)$

DP-37

Best time to Buy and sell stocks III

Problem

Statement

Prices = (3, 3, 5, 0, 0, 3, 1, 4) → same as last problem

We have to find maximum profit but with a extra condition

- * We can do maximum 2 transitions

Solution

→ Try all possible ways + Pick the best one

+ Recursion

How to write recursion

①

Express function in term of variables, like mid, buy

But here we also get 1 extra condition to keep track of no of transition. so

* let add another variable (cap) → start with 2 and ~~use~~ with each sold we decrement value of cap.

* when value of cap == 0, no more transition you done

fun(mid, buy, cap){

②

Explore all possibilities on that day

③

Take the maximum profit made

④

Write base case

1st

fun(mid, buy, cap),

mid → traverse the days in Buy and Sell stocks

buy → can tell that we can purchase or not

cap → $\{ 2 \rightarrow \text{we can do sell 2 times} \}$

$\{ 1 + \text{can do sell 1 time} \}, 0 \rightarrow \text{no more buy or sell}$

'y(buy){

profit = max { -num(mid) + fun(mid+1, 0, cap), 0 + fun(mid+1, 1, cap) }

else

profit = max (+num(mid) + fun(mid+1, 1, cap),)

Memorization and Recursive code

```

mit fun ( mit mid, mit buy, mit cap, mit n, vector<mt> nums, vector<vector<vector<mit>>> &dp) {
    'y ( mid == n) return 0;
    'y ( cap == 0) return 0;
    'y ( dp[mid][buy][cap] != -1) return dp[mid][buy][cap];
    'y (buy) {
        profit = max (-nums[mid] + fun(mid+1, 0, cap, n, nums, dp),
                      fun(mid+1, 1, cap, n, nums, dp));
    }
    else {
        profit = max( nums[mid] + fun(mid+1, 1, cap-1, n, nums, dp),
                      fun(mid+1, 0, cap, n, nums, dp));
    }
    return dp[mid][buy][cap] = profit;
}

mit maxprofit (vector<mt> &nums, mit n) {
    vector<vector<vector<mit>>> dp(n+1, vector<vector<mt>>(2,
                                              vector<mt>(3, -1)));
    return fun(0, 1, 2, n, nums, dp);
}

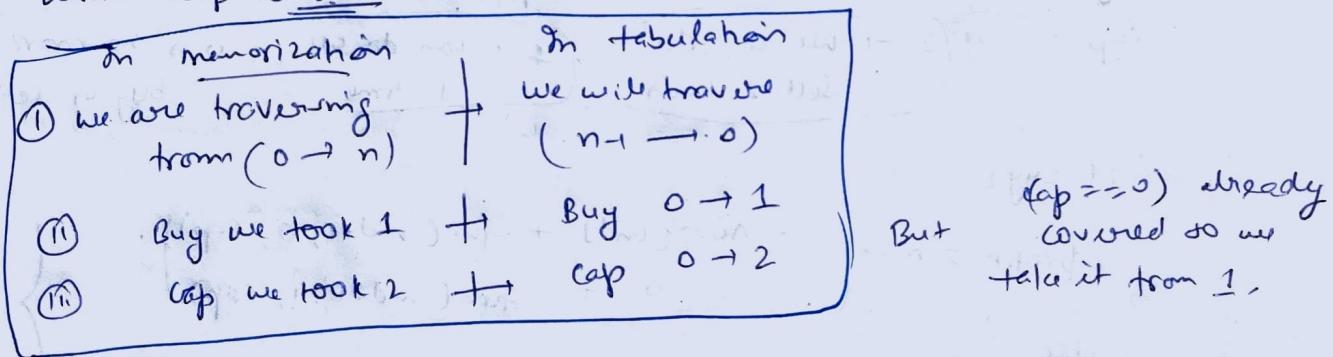
```

Time = $O(N \times 2 \times 3)$

Space = $O(N \times 2 \times 3) + O(N)$

Tabulation steps to convert memorization into tabulation

- ① change Base case , here we can neglect because they already have
- ② write loop condition same value '0'.



ans returned as $\text{ans} = \text{fun}(0, 1, 2, n, \text{nums}, \text{dp}) \rightarrow \text{return } \text{dp}[0][1][2];$

Tabulation code

```

int maxProfit ( vector<int> &nums, int n) {
    vector<vector<vector<int>> dp( n+1, vector<vector<int>>(2,
        vector<int> (3, 0)));
    for ( int mid = n-1; mid >= 0; mid--) {
        for ( int buy = 0; buy < 2; buy++) {
            for ( int cap = 0; cap < 3; cap++) {
                if (buy) {
                    dp[mid][buy][cap] = max ( -nums[mid] + dp[mid+1][0][cap],
                        dp[mid+1][1][cap]);
                } else {
                    dp[mid][buy][cap] = max ( 0, nums[mid] + dp[mid+1][1][cap-1],
                        dp[mid+1][0][cap]);
                }
            }
        }
    }
    return dp[0][1][2];
}

```

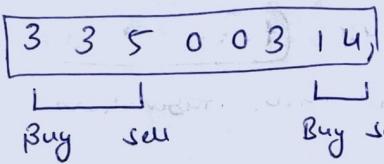
Another approach

Instead of using $dp[N][2][3]$

we can use $dp[N][4]$

$\xleftarrow{\text{mid}}$ $\xrightarrow{\text{trans}}$

idea :-



so recursion function

```

fun( mid, trans) {
    if (mid == n || trans == 4)
        return 0;
    if (trans-1-2 == 0) → buy
        max { - prices(mid) + fun( mid+1, trans+1), }
        0 + fun( mid+1, trans)
    else
        max { + prices(mid) + fun( mid+1, trans+1) }
        0 + fun( mid+1, trans)
}

```

- ① we know the maximum limit is 2.
- ② so we can say that we can have maximum 2 Buy 2 Sell
- ③ with each buy and sell we will increment value of '+' and once it reaches 4 no more transaction will be possible

Memorization + recursion approach :

```

int fun( int mid, int t, int n, vector<int> &nums,
         vector<vector<int>> &dp ) {
    if( mid == n || t == 4 ) return 0;
    if( dp[mid][t] != -1 ) return dp[mid][t];
    int profit = 0;
    if( t * 1.2 == 0 ) {
        profit = max( -nums[mid] + fun( mid+1, t+1, n, nums, dp ),
                      0 + fun( mid+1, t, n, nums, dp ) );
    } else {
        profit = max( nums[mid] + fun( mid+1, t+1, n, nums, dp ),
                      0 + fun( mid+1, t, n, nums, dp ) );
    }
    return dp[mid][t] = profit;
}

int maxProfit( vector<int> &nums) {
    int n = nums.size();
    vector<vector<int>> dp( n+1, vector<int>( 4, -1 ) );
    return fun( 0, 0, n, nums, dp );
}

```

Tabulation

↳ How to convert memorization into tabulation

① step 1 → declare required size dp. → Here we will need $n \times 5$ dp
[Because we have to store base $t=4 \rightarrow 0$]

② convert Base cases of memo into tabulation

```

for( int i = 0 ; i < 5 ; i++ ) {
    dp[0][i] = 0;
}
    
```

```

for( int i = 0 ; i < n ; i++ )
    dp[i][4] = 0;
    
```

③ convert recursion into (loops) → Just opposite traversing of memorization
on Memo

ind → $(0 \rightarrow n)$ → $(n-1 \rightarrow 0)$] on tabulation
 $+ 1$ $(0 \rightarrow 4)$ → $(3 \rightarrow 0)$] on tabulation

iv) copy Paste
Main part
of memo

```

for( int i = n-1 ; i >= 0 ; i-- ) {
    for( int j = 3 ; j >= 0 ; j-- ) {
        ;
    }
}
    
```

Code

```

int maxProfit ( vector<int> &nums, int n){
    vector<vector<int>> dp( n+1, vector<int>( 5, 0));
    for( int i= 0 ; i<5 ; i++){
        dp[0][i] = 0;
    }
    for( int j=0 ; j<n ; j++){
        dp[j][4] = 0;
    }
    for( int i= n-1 ; i>=0 ; i--){
        for( int j=3 ; j>=0 ; j--){

            if( j+1 == 0){
                dp[i][j] = max( -nums[i] + dp[i+1][j+1], dp[i+1][j]);
            } else {
                dp[i][j] = max( nums[i] + dp[i+1][j+1], dp[i+1][j]);
            }
        }
    }
    return dp[0][0];
}

```

time = $O(N \times S)$
space = $O(N \times S)$

DP → 38 Best time to Buy and sell stock 4

Problem statement

- We have given an $\text{nums} = [3, 2, 6, 5, 0, 3]$, and $K=2$
- And we have to calculate maximum profit and we can do maximum K transactions.

ideal logic → In the previous ques we can do maximum transactions upto 2 but now it will be K so only 1 change is required to get correct answer



Memorization + Recursion

```

mit fun ( mit mid, mit buy, mit k, int n, vector<mit> & nums,
          vector<vector<vector<mit>>> & dp) {
    if (mid == n || k == 0) return 0;
    if (dp[mid][buy][k] != -1) return dp[mid][buy][k];
    mit profit = 0;
    if (buy) {
        profit = max(-nums[mid] + fun(mid+1, 0, k, n, nums, dp),
                      0 + fun(mid+1, 1, k, n, nums, dp));
    } else {
        profit = max(nums[mid] + fun(mid+1, 1, k-1, n, nums, dp),
                      0 + fun(mid+1, 0, k, n, nums, dp));
    }
    return dp[mid][buy][k] = profit;
}

mit maxProfit ( mit k, mit n, vector<mit> & nums) {
    vector<vector<vector<mit>>> dp(n+1, vector<vector<mit>>(2,
                                              vector<mit>(<K+1, -1))));
    return fun(0, 1, K, n, nums, dp);
}

```

↓ convert memorization to tabulation. yes/no

① declare required size vector "dp(N+1)[2][K+1]" → (0-K) for storing
 ↓
 (0-n-1) for storing
 n-1 for base case
 kth for base case

② convert base case into tabulation

(i) $(\text{mid} == n) = 0$

```

for (mit i=0; i<2; i++) {
    for (mit j=0; j<=K; j++) {
        dp[n][i][j] = 0;
    }
}

```

if ($K == 0$) return 0;

```

for (mit i=0; i<n; i++) {
    for (mit j=0; j<2; j++) {
        dp[i][j][0] = 0;
    }
}

```

we can avoid

writing this code just by

"declaring array with initial value '0'". [∴ we are updating]

③ convert recursion to loops

Namo

mid → 0 → n

→

Tabulation

$(n-1 \rightarrow 0)$	1st loop
$(0 \rightarrow 1)$	2nd loop
$(0 \rightarrow K)$	3rd loop

buy → 1 → 0

→

trans → K → 0

Code

```

mit fun( mit k, mit n, vector<mit> &num) {
    vector<vector<vector<mit>>> dp(n+1, vector<vector<mit>>(2,
        vector<mit>(k+1, 0)));
}

// no need to write base case

for (mit i = n-1; i >= 0; i--) {
    for (mit j = 0; j <= 1; j++) {
        for (mit m = 1; m <= k; m++) {
            if (j == 1) {
                dp(i)(j)(m) = max(-num[i] + dp(i+1)(0)(m), dp(i+1)(1)(m));
            } else {
                dp(i)(j)(m) = max(num[i] + dp(i+1)(1)(m-1), dp(i+1)(0)(m));
            }
        }
    }
}
return dp(0)(1)(k);
    
```

→ Just with the 3 variable input you gave in memo fun(...)

Another approach

- like previous method we can use $dp(n+1)(2 \times k) \rightarrow \text{vector}$ instead of $dp(N+1)(2)(k)$ or
- watch striver video for DP solution

DP-39 Best time to buy and sell stock with cooldown.

Problem statement →

- Given price of stock for each day $[1, 2, 3, 0, 2]$
- Can do any no of transactions
- But after every sold we can't buy on next day

Idea / logic → We know the recursion part without cooldown condition

$\text{fun}(mid, buy)$

```

if (buy) {
    profit = max(-num[mid] + fun(mid+1, 0),
        0 + fun(mid+1, 1))
}
    
```

```

else {
    profit = max(+num[mid] + fun(mid+1, 1),
        0 + fun(mid+1, 0))
}
    
```

this is sold part

+ add cooldown condition

when we sold we can buy next to next day

$\text{num}[mid] * \text{fun}(mid+2, 1)$

Memorization + Recursion :-

```

mit fun ( mit mid, mit buy , mit n, vector<mit> + nums),
) vector<vector<mit>> & dp) {
    if ( mid >= n) return 0;
    if ( dp[mid][buy] != -1) return dp[mid][buy];
    if ( buy) {
        dp[mid][buy] = max ( -nums[mid] + fun( mid+1, 0, n, nums, dp))
    } else {
        dp[mid][buy] = max [ +nums[mid] + fun( mid+2, 1, n, nums, dp)],
                            0 + fun( mid+1, 0, n, nums, dp)
    }
    return dp[mid][buy];
}

mit maxprofit ( mit n, vector<mit> + nums) {
    Vector<vector<mit>> dp( n+1, vector<mit> (2,-1));
    return fun( 0, 1, n, nums, dp);
}

```

$O(N) \rightarrow \text{Time}$
 $O(N \times 2) \rightarrow \text{Space}$
 $+ O(N)$

* Convert memorization to tabulation

- convert base case (but no need here due to '0' as the value we have)
- copy paste recursion to update)
- convert recursion into loops in opposite fashion
in memo we go $(0 \rightarrow n)$ (so in tabulation $n-1 \rightarrow 0$)

Code :

```

mit maxprofit ( mit n, vector<mit> + nums){
    Vector<vector<mit>> dp( n+2, vector<mit> (2,0));
    for(mit i= n-1; i>=0; i--){
        for (mit j=0; j<2; j++){
            if (j) {
                dp[i][j] = max( -nums[i] + dp[i+1][0], dp[i+1][1]);
            } else {
                dp[i][j] = max ( nums[i] + dp[i+2][1], dp[i+1][0]);
            }
        }
    }
    return dp[0][1];
}

```

$O(N) + \text{time}$
 $O(N \times 2) \rightarrow \text{Space}$

\nwarrow
 * pure space optimization is not
 very good option just because
 we have to keep track of $dp[i+1]$, and $dp[i+2]$

\nearrow just copy returning from memo
 $dp[0][1] \approx \text{fun}(0, 1, n, \text{nums}, \text{dp})$

DP-40

Buy and sell stocks with transaction fees.

Problem statement

Given → a price list of per day [1, 3, 2, 8, 4, 9] and

find max profit?

Buy and sell can be done unlimited times, but each time we sold a transaction fee is charged.

Idea / logic

We know that the only difference is transaction fee so every time we sold we put extra fee on profit.

Memorization & recursion code

```

mit fun (mit mid, mit buy, mit k, mit n, vector<int> & nums,
         vector<vector<int>> & dp) {
    if (mid == n) return 0;
    if (dp[mid][buy] != -1) return dp[mid][buy];
    mit profit = 0;
    for (int i = mid + 1; i < n; i++) {
        if (buy) {
            profit = max(-nums[i] + fun(mid + 1, 0, k, n, nums, dp),
                         0 + fun(mid + 1, 1, k, n, nums, dp));
        } else {
            profit = max((+nums[i]) + fun(mid + 1, 1, k, n, nums, dp) - k,
                         0 + fun(mid + 1, 0, k, n, nums, dp));
        }
    }
    return dp[mid][buy] = profit;
}

mit maxprofit (mit k, vector<int> & nums) {
    mit n = nums.size();
    vector<vector<int>> dp(n + 1, vector<int> (2, -1));
    return fun(0, 1, k, n, nums, dp);
}

```

* can easily convert to tabulation

→ can refer to previous notes.

Time = O(NX2)
Space = O(NX2) + O(M)