Dynamic Programming

Video - 127

Note :- This playlist is only for explanation of Qns & solutions.

See my "DP Concepts & Qns" playlist for understanding DP from scratch...

Leetcode - 3562 Hard

codestorywithmik

CSwithMIK

codestorywithMIK

Motivation :-

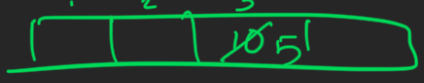
The best way to solve a big problem is to break it into smaller easy subproblems.

Rest everything will start making sense.

3562. Maximum Profit from Trading Stocks with Discounts

Hard | Topics | Companies | Hint

Present



You are given an integer `n`, representing the number of employees in a company. Each employee is assigned a unique ID from 1 to `n`, and employee 1 is the CEO. You are given two **1-based** integer arrays, `present` and `future`, each of length `n`, where:

- `present[i]` represents the **current** price at which the `i^th` employee can buy a stock today.
- `future[i]` represents the **expected** price at which the `i^th` employee can sell the stock tomorrow.

The company's hierarchy is represented by a 2D integer array `hierarchy`, where `hierarchy[i] = [u_i, v_i]` means that employee `u_i` is the direct boss of employee `v_i`.

Additionally, you have an integer `budget` representing the total funds available for investment.

However, the company has a discount policy: if an employee's direct boss purchases their own stock, then the employee can buy their stock at **half** the original price (`floor(present[v] / 2)`).

Return the **maximum** profit that can be achieved without exceeding the given budget.

Note:

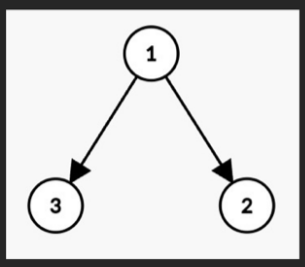
- You may buy each stock at most **once**.

- You **cannot** use any profit earned from future stock prices to fund additional investments and must buy only from `budget`.

Input: `n = 3, present = [4,6,8], future = [7,9,11], hierarchy = [[1,2],[1,3]], budget = 10`

Output: 10

Explanation:



Budget = 10

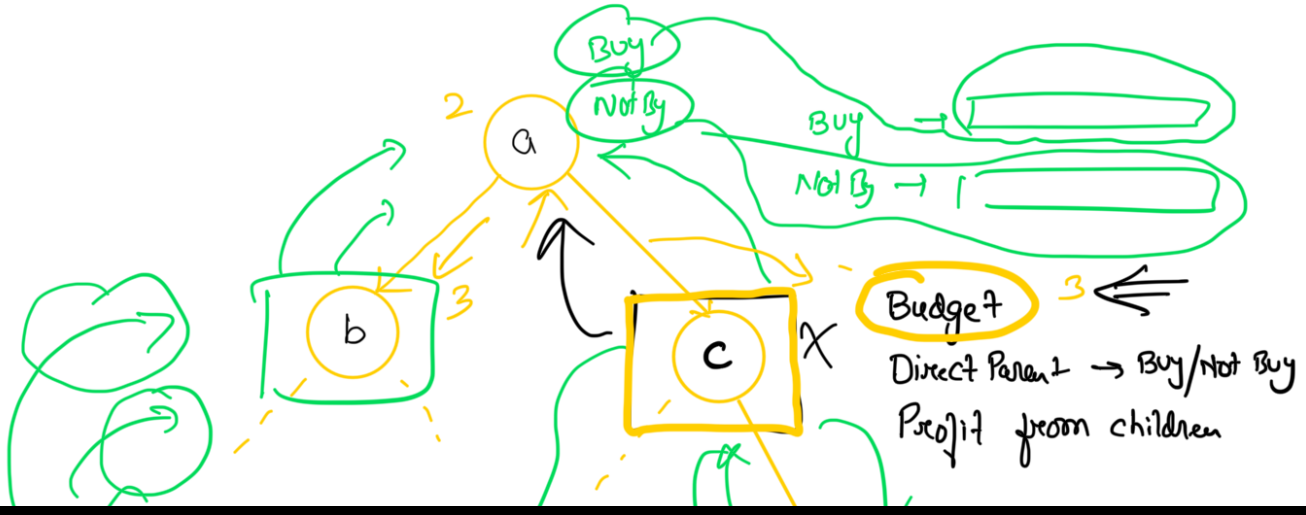
Employ => 2
P: 9 − 3 = 6
B = 6 − 3 = 3

Emp 3
P: 11 − 4 = 7
B = 6 − 4 = 2

- Employee 1 buys the stock at price 4 and earns a profit of `7 - 4 = 3`.
- Employee 3 would get a discounted price of `floor(8 / 2) = 4` and earns a profit of `11 - 4 = 7`.
- Employee 1 and Employee 3 buy their stocks at a total cost of `4 + 4 = 8 <= budget`. Thus, the maximum total profit achieved is `3 + 7 = 10`.

Thought Process

Boss(CEO) → Buy stock / not buy stock.



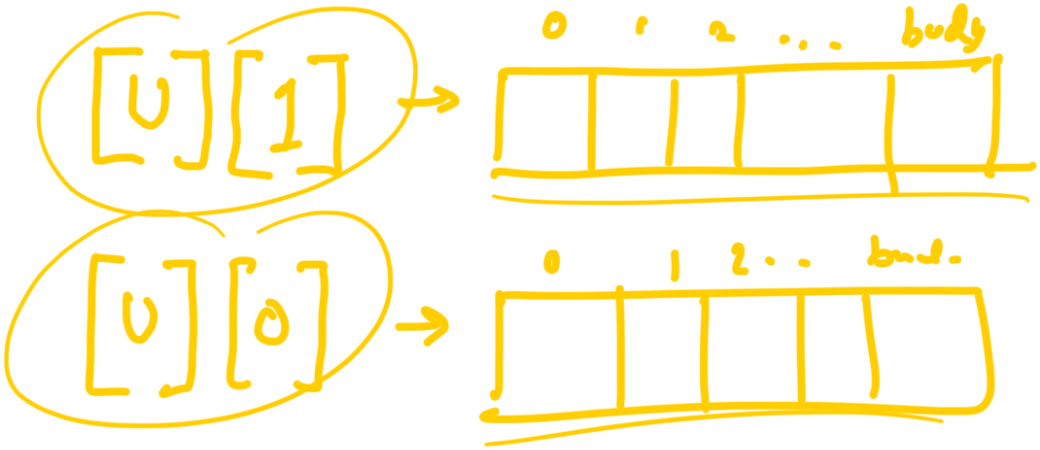
Budget = 8

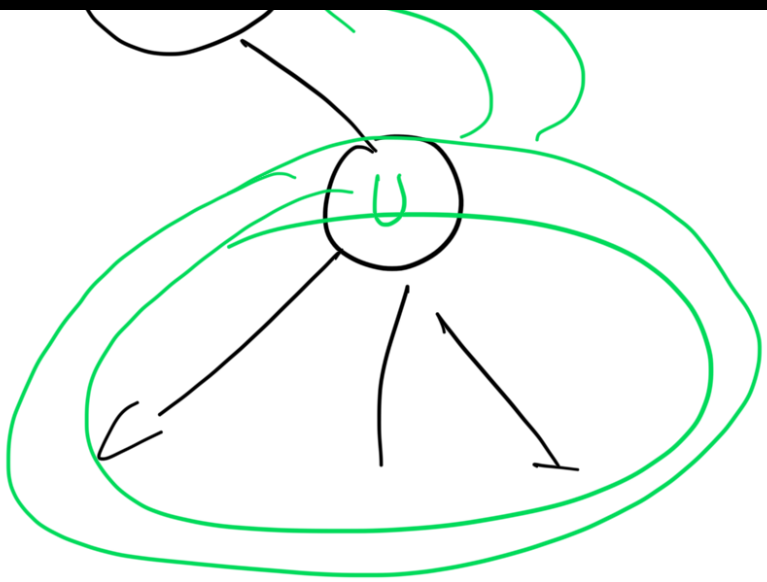
U, Buy/Not By, budget → max profit

[U][1][b]

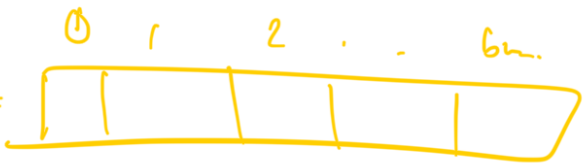
[U][0][b]

: 3D-vector



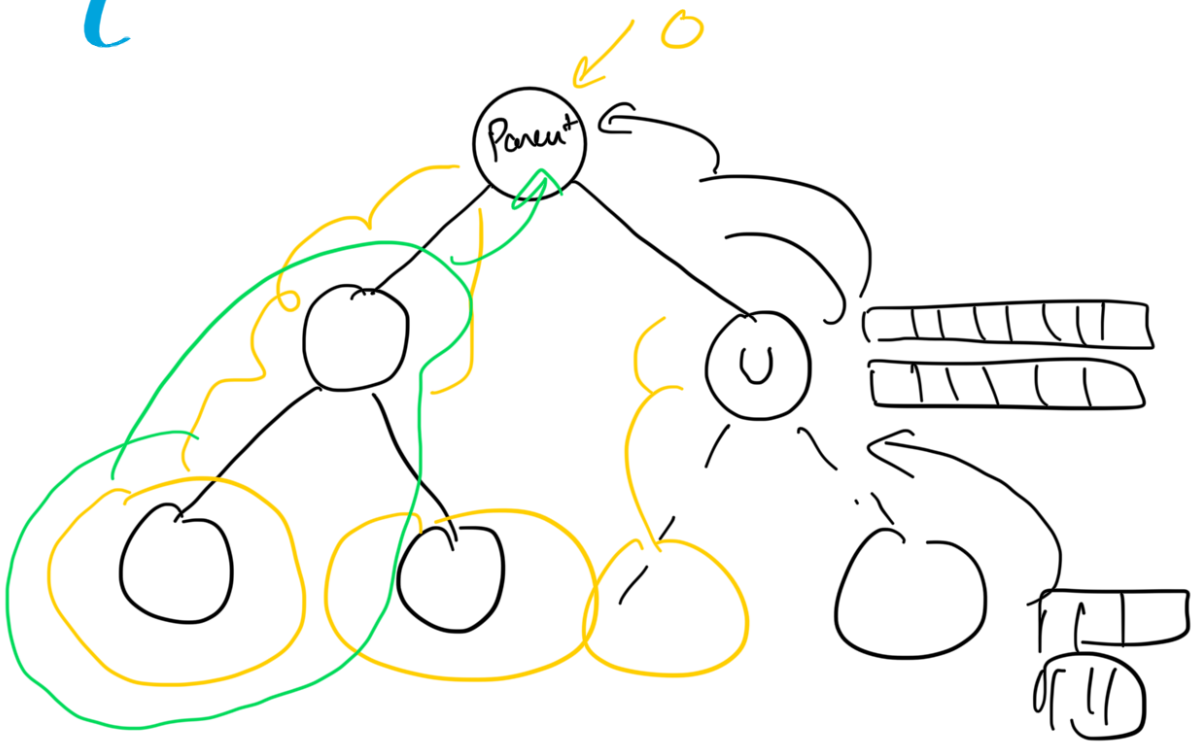


[u] [0] =



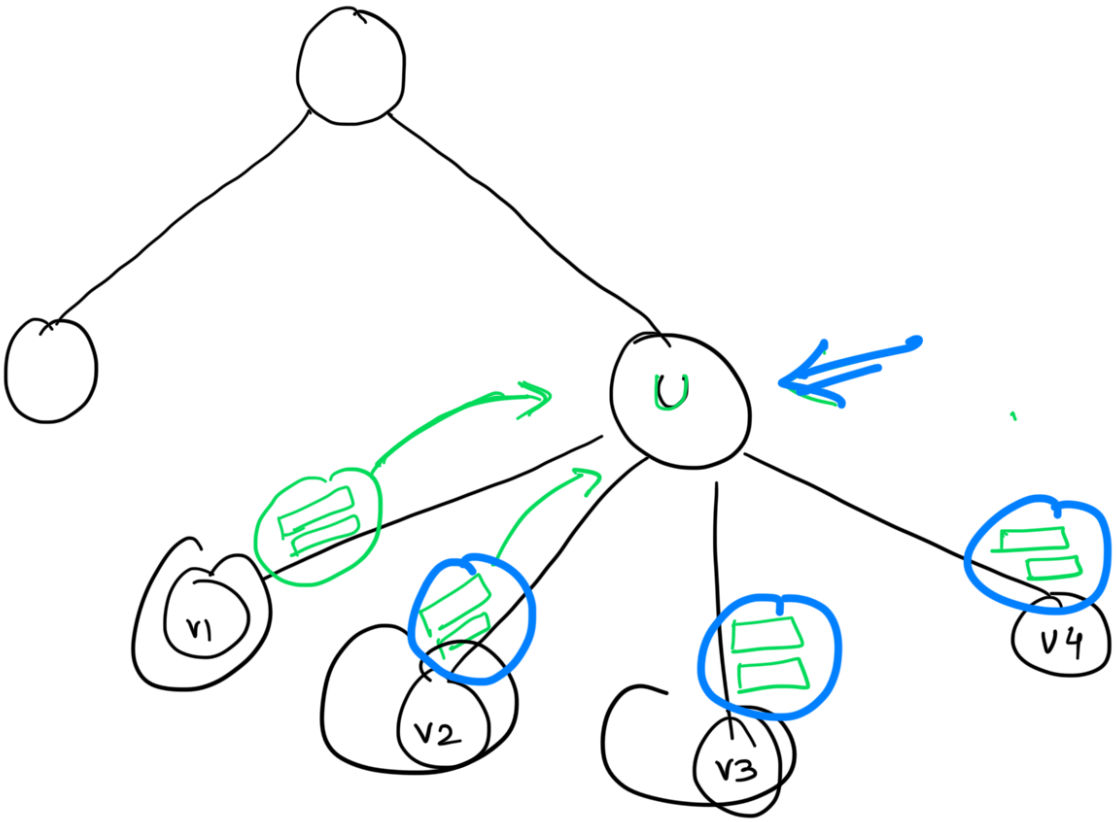
[u] [1) (budget)

How to proceed ??



DFS

DFS(0, present, future, adj, statesProfit, budget) {



u → { (,) , { , } }

vector<pair<vector, vector>> childrenStatesProfit;

for (int &v : adj[u]) {

```
DFS(v, present, future, adj, statesProfit, budget);
childrenStatesProfit.push_back({statesProfit[v][0],
                                statesProfit[v][1]});
}
```

Calculating for U :-


```
vector<int> bestProfitAtU(budget+1, 0);
```

```
vector<int> childrenProfitIfUNotBought(budget+1, 0);
```

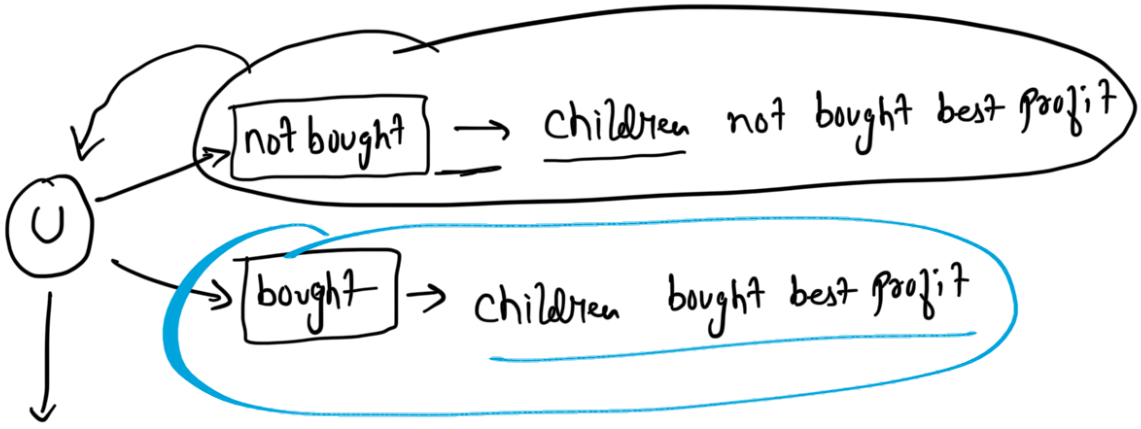
```
for ( auto & child : childrenStatesProfit ) {
    vector<int> temp (buy+1, 0);

    for(used = 0 ; used <= budget ; used++) {
        for (take = 0 ; take+used <= budget ) {
            temp[used+take] = max ( temp[used+take] ,
                childrenProfitIfUNotBought [used]
                + child-first[take] ) ;
        }
    }

    childrenProfitIfUNotBought = temp;
}
```

budget used by prev. children

budget for current children



bestProfitatU

```
for ( int b = 0 ;  b <= budget ;  b++ ) {
    bestProfitatU[b] = max( bestProfitatU[b] ,
```

childrenProfitIfUNotBought[b]);

}

Bought by U :-

Present[U];

for (int parent = 0 ; Parent <= 1 ; Par++) {

Case-1 : U not bought stock

Case-2 : U buy stock

statesProfit[U][parent] = bestProfitAtU ;

}

```
Price = (Parent == 0) ? Present[u] : Present[u]/2;

Profit = future[u] - price;
```

```
//Case-2 When u bought stock

vector<int> childrenProfitIfUBought (budget+1, 0);

for (auto &child : childrenStatesProfit) {
    vector<int> temp(buy+1, 0);

    for(used = 0; used <= budget; used++)) {
        for (take = 0; take+used <= budget) {

            temp[used+take] = max( temp[used+take],
                childrenProfitIfUBought[used]
                + child.first[take]);
        }

    childrenProfitIfUBought = temp;
}
```

budget used by prev. children

budget for current children

```
for (b = price;  b <= budget ; b++) {

    bestProfitatU[b] = max(bestProfitatU[b] ,
                           ChildrenProfitIfUBought[b-price]
                           + profit;
}
```

```
stateProfit[u][parent] = bestProfitatU ;
}
```