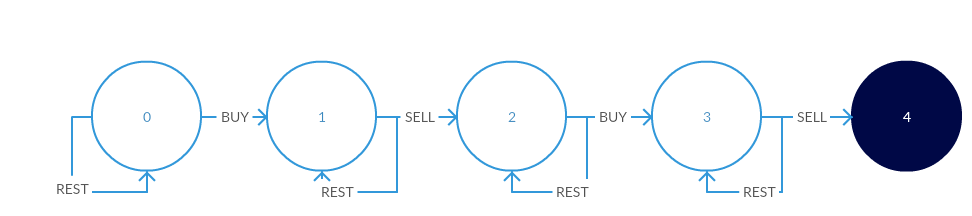
## BestVersionState graph

This approach can be used for all the problems based on stock prices.

The idea is to design a state machine that correctly describes the problem statement.



**Intuition behind the state diagram:**  
We begin at state 0, where we can either rest (i.e. do nothing) or buy stock at a given price.

* If we choose to rest, we remain in state 0
* If we buy, we spend some money (price of the stock on that day) and go to state 1

From state 1, we can once again choose to do nothing or we can sell our stock.

* If we choose to rest, we remain in state 1
* If we sell, we earn some money (price of the stock on that day) and go to state 2

This completes one transaction for us. Remember, we can only do *atmost* 2 transactions.

From state 2, we can choose to do nothing or buy more stock.

* If we choose to rest, we remain in state 2
* If we buy, we go to state 3

From state 3, we can once again choose to do nothing or we can sell our stock for the last time.

* If we choose to rest, we remain in state 3
* If we sell, we have utilized our allowed transactions and reach the final state 4

**Going from the state diagram to code**

# fullVersion

class Solution {

public:

int maxProfit(vector<int> &prices) {

// f[k, ii] represents the max profit up until prices[ii] (Note: NOT ending with prices[ii]) using at most k transactions.

// f[k, ii] = max(f[k, ii-1], prices[ii] - prices[jj] + f[k-1, jj]) { jj in range of [0, ii-1] }

// = max(f[k, ii-1], prices[ii] + max(f[k-1, jj] - prices[jj]))

// f[0, ii] = 0; 0 times transation makes 0 profit

// f[k, 0] = 0; if there is only one price data point you can't make any money no matter how many times you can trade

if (prices.size() <= 1) return 0;

else {

int K = 2; // number of max transation allowed

int maxProf = 0;

vector<vector<int>> f(K+1, vector<int>(prices.size(), 0));

for (int kk = 1; kk <= K; kk++) {

int tmpMax = f[kk-1][0] - prices[0];

for (int ii = 1; ii < prices.size(); ii++) {

f[kk][ii] = max(f[kk][ii-1], prices[ii] + tmpMax);

tmpMax = max(tmpMax, f[kk-1][ii] - prices[ii]);

maxProf = max(f[kk][ii], maxProf);

}

}

return maxProf;

}

}

};