746. Min Cost Climbing Stairs

Problem description

You are given an integer array `cost` where `cost[i]` is the cost of `i`th step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor*.

Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: Cheapest is: start on cost[1], pay that cost, and go to the top.
```

Example 2:

```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: Cheapest is: start on cost[0], and only step on 1s, skipping cost[3].
```

Constraints:

- `2 <= cost.length <= 1000`
- `0 <= cost[i] <= 999`

arnav-vijayakar ★ 259 December 28, 2019 11:43 AM

For people having confusion regarding question. the stair case is something like this:
Input: cost = [10, 15, 20]

```
                                  __________
                             ___ | Final step
                        ___ | 20
                   ___ | 15
_________ | 10
First step
```

The logic of the solution

Let’s go throw an example:

Input: cost = [1,100,1,1,1,100,1,1,100,1]

cost = [0, 1,100,1,1,1,100,1,1,100,1, 0]

0	1	100	1	1	1	100	1	1	100	1	0
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*

from a stair i , we must jump to stair $\begin{cases} i+1, if\ cost[i+1] \le cost[i+2] \\ i+2, otherwise \end{cases}$

In other words, we jump to the stair that have the minimum cost.

Let’s say $f(i)$ is the final cost to climb to the top from stair i , then

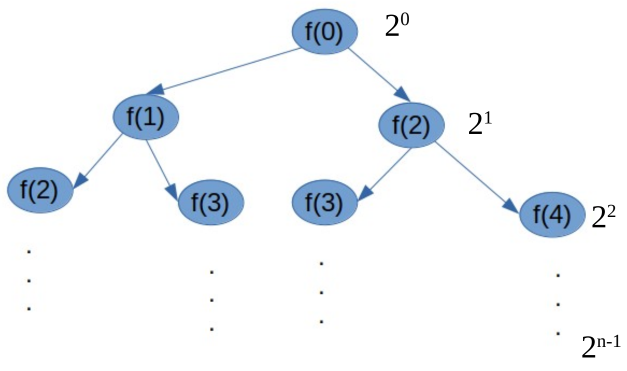
$f(i) = cost[i] + min(f(i+1), f(i+2))$ (***hint from leetcode***)

Naive Solution

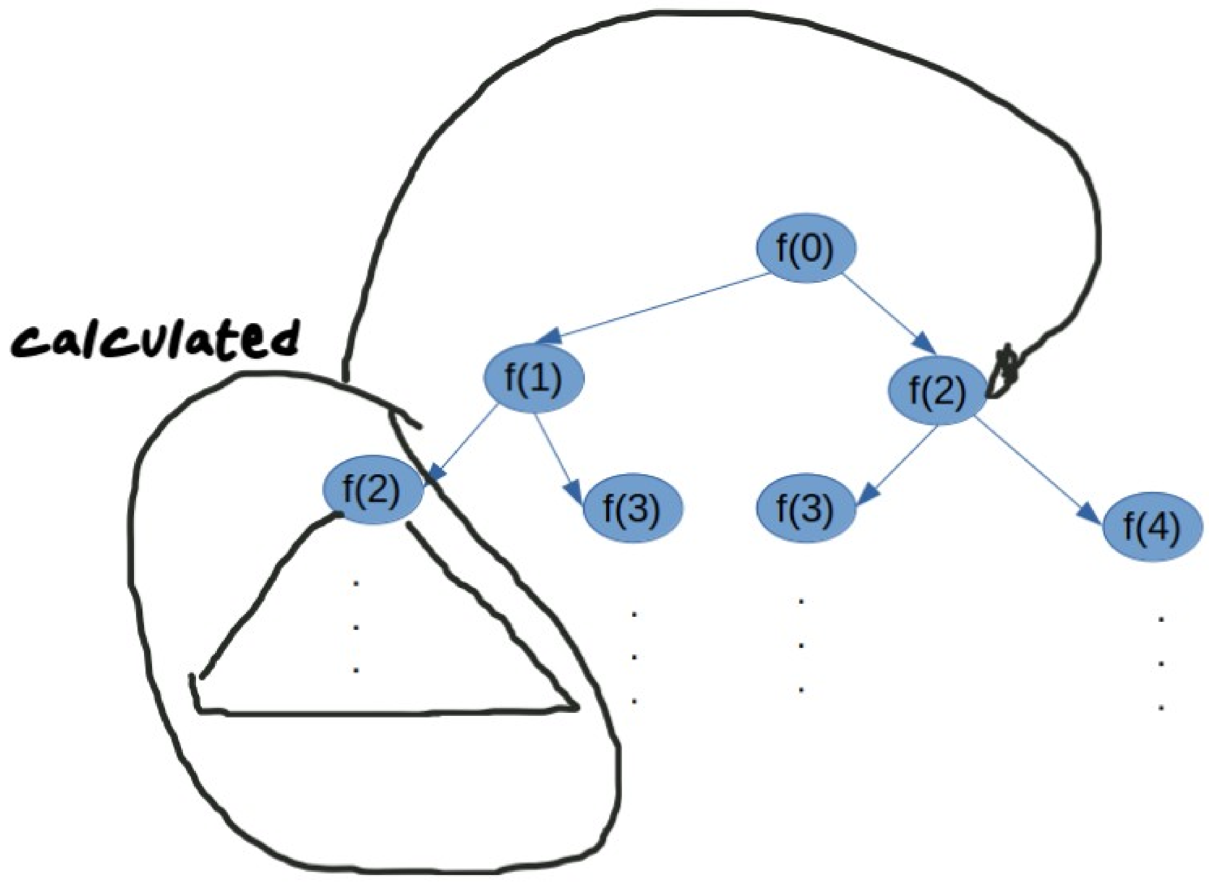
C++: $O(2^{n-1})$

```
int f(vector<int>& cost, int i, int n) {
        if (i == n-1) return 0;
        if (i == n - 2) return cost[i];
        return cost[i] + min(f(cost, i+1, n), f(cost, i+2, n));
}
int minCostClimbingStairs(vector<int>& cost) {
       auto it1 = cost.begin();
       cost.insert(it1, 0);
       auto it2 = cost.end();
       cost.insert(it2, 0);
       int n = cost.size();
       return f(cost, 0, n);
}
```

Time complexity



Solution with memoization



The idea is to store the calculated $f(i)$ in a data structure (like a map), in order to not be recalculated.

C++: $O(n)$

```
unordered_map<int, int> memo;
int f(vector<int>& cost, int i, int n) {
        if (i == n-1) return 0;

        if (memo.find(i) != memo.end()) return memo[i];

        if (i == n - 2) return memo[i] = cost[i];
        return memo[i] = cost[i] + min(f(cost, i + 1, n), f(cost, i + 2, n));
}
int minCostClimbingStairs(vector<int>& cost) {
         auto it1 = cost.begin();
         cost.insert(it1, 0);
         auto it2 = cost.end();
         cost.insert(it2, 0);
         int n = cost.size();
         return f(cost, 0, n);
}
```