956. Tallest Billboard II DP II Reducing DP States II Hashing II 0/1 Knapsack

24 June 2023 07:19 AM

956. Tallest Billboard

Hard 903 29

Companies

You are installing a billboard and want it to have the largest height. The billboard will have two steel supports, one on each side. Each steel support must be an equal height.

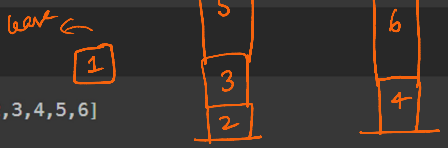
You are given a collection of `rods` that can be welded together. For example, if you have rods of lengths `1`, `2`, and `3`, you can weld them together to make a support of length `6`.

Return *the largest possible height of your billboard installation*. If you cannot support the billboard, return `0`.

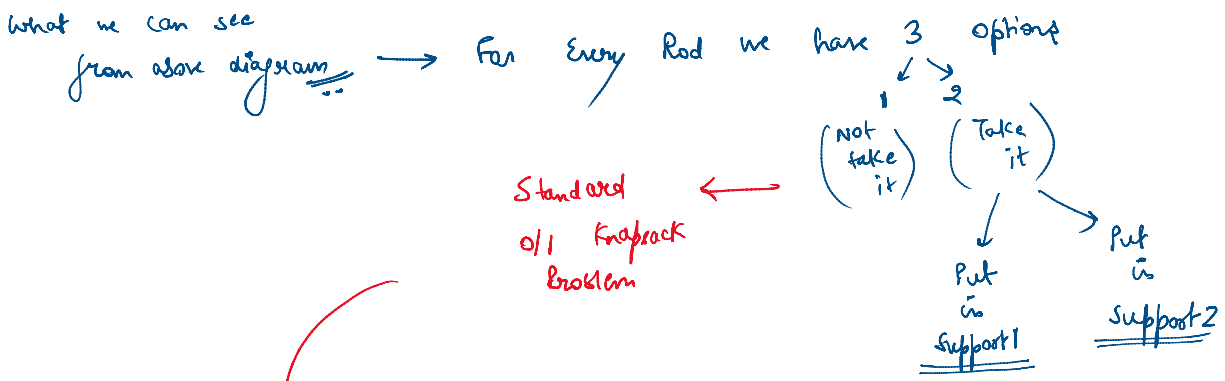
Example 1:

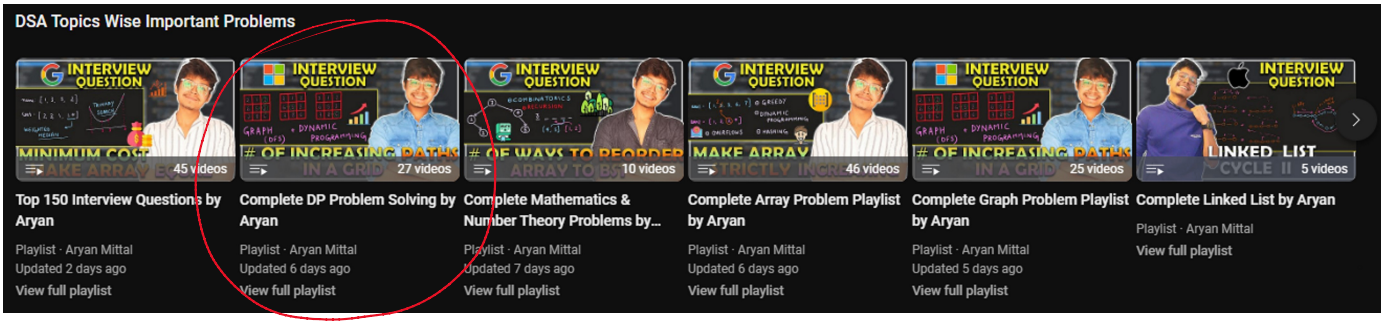

```
Input: rods = [1,2,3,6]
Output: 6
Explanation: We have two disjoint subsets {1,2,3} and {6}, which have
the same sum = 6.
```

Example 2:

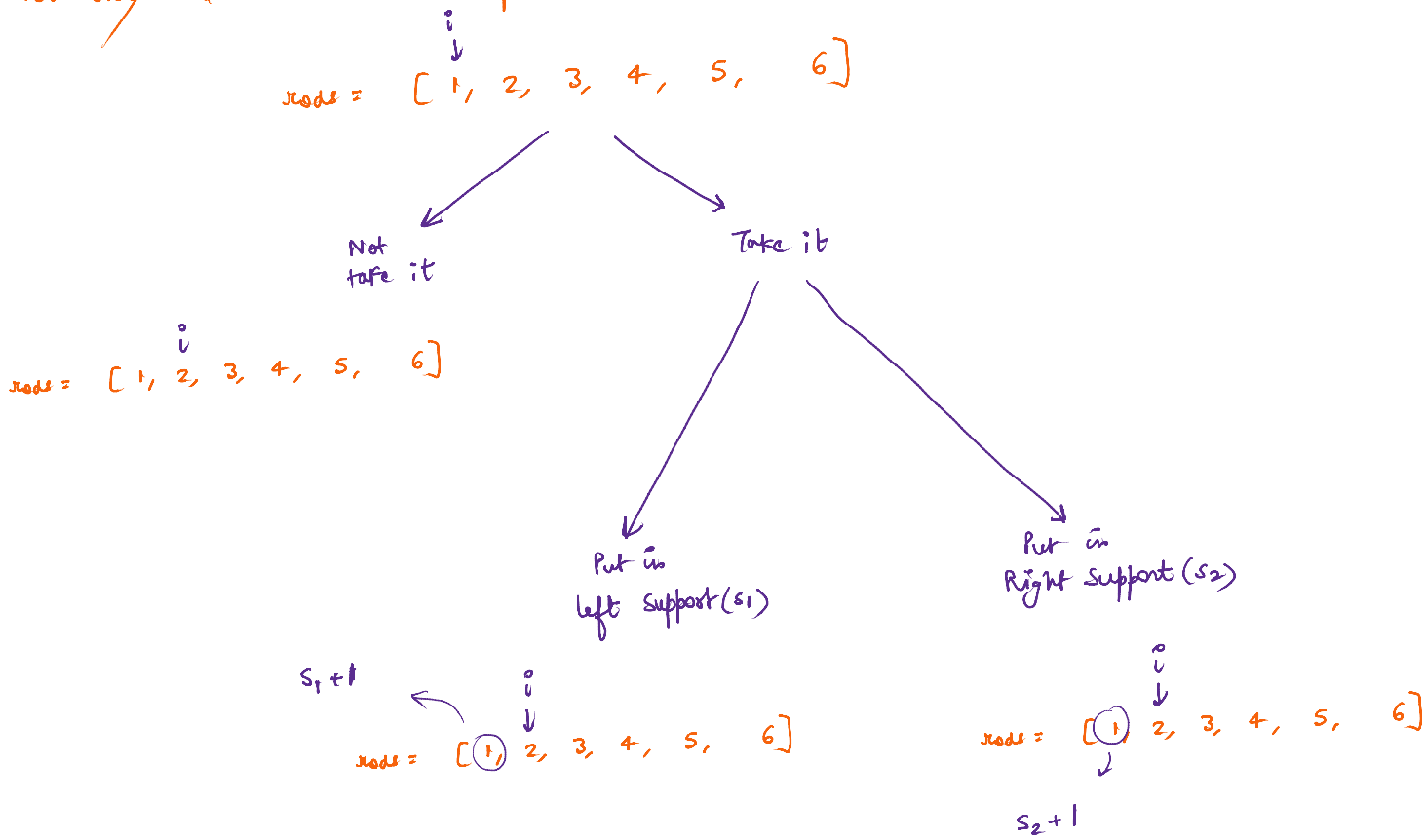



```
Input: rods = [1,2,3,4,5,6]
Output: 10
Explanation: We have two disjoint subsets {2,3,5} and {4,6}, which have
the same sum = 10.
```



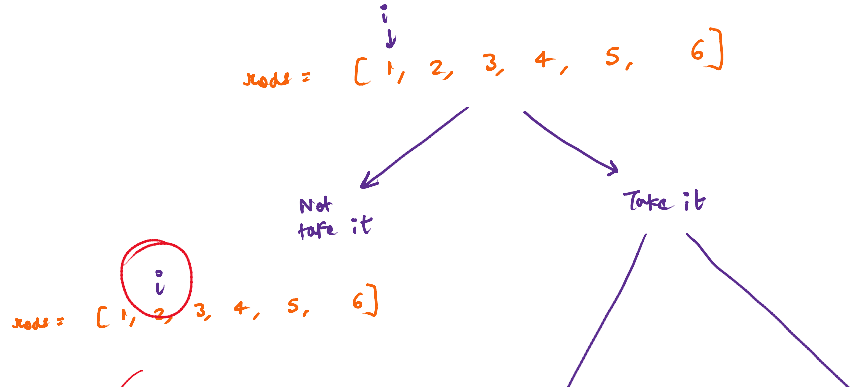


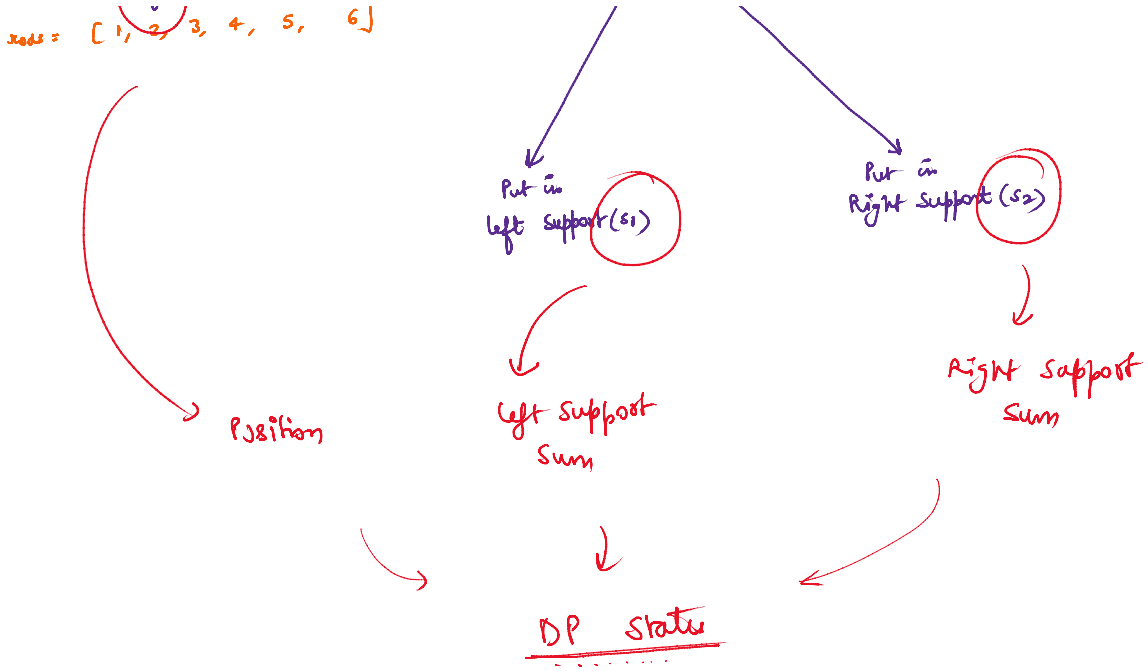
⇒ For Every Rod → we have 3 options ...



↳ we can see from the Tree itself it is a Recursion → kind of same 0/1 knapsack Recursion tree will be made

Repeating sub-problems ← Memoization & DP





Simple Recursion

```
class Solution{
    int ans, n;
    vector<int> rods;
public:
    int tallestBillboard(vector<int>& rods) {
        this->rods = rods;
        n = rods.size();
        ans = 0;

        return solve(0, 0, 0);
    }

    int solve(int i, int s1, int s2) {

        if (i == n) {
            if (s1 == s2)return s1;
            return 0;
        }

        int ans = 0;

        int opt1 = solve(i + 1 , s1 , s2);
        int opt2 = solve(i + 1 , s1 + rods[i], s2);
        int opt3 = solve(i + 1 , s1 , s2 + rods[i]);

        return ans = max({opt1, opt2, opt3});
    }
};
```

Position
Left support Sum
Right support Sum
As we reach end
If both support are equal
Don't take that rod & move on :)
Take Rod, add in support 1
Take Rod, add in Support 2
take max equal Support

```cpp
class Solution{
    int ans, n;
    vector<int> rods;
public:
    int tallestBillboard(vector<int>& rods) {
        this->rods = rods;
        n = rods.size();
        ans = 0;

        return solve(0, 0, 0);
    }

    int solve(int i, int s1, int s2) {

        if (i == n) {
            if (s1 == s2)return s1;
            return 0;
        }

        int ans = 0;

        int opt1 = solve(i + 1 , s1 , s2);
        int opt2 = solve(i + 1 , s1 + rods[i], s2);
        int opt3 = solve(i + 1 , s1 , s2 + rods[i]);

        return ans = max({opt1, opt2, opt3});
    }
};
```

Even if we memoize this by

dp[i][s1][s2]

Constraints:

- `1 <= rods.length <= 20`
- `1 <= rods[i] <= 1000`
- `sum(rods[i]) <= 5000`

$= 20 * 5000 * 5000$

$= 50 * 10^7 = 5 * 10^8$

both time & space not feasible :: with DP too

going to next position

optimize DP state :: dp[i][s1][s2]

- s1 → Sum of Support 1
- s2 → Sum of Support 2

Identify Similarity in DP state

Ultimately we want Sum(s1) = Sum(s2)

or we can take difference also

Sum(s1) - Sum(s2) = 0

s1 - s2 = 0

diff = 0

↳ Converting s1 & s2 to diff

$0 \le s1 \le 5000$

$0 \le s2 \le 5000$

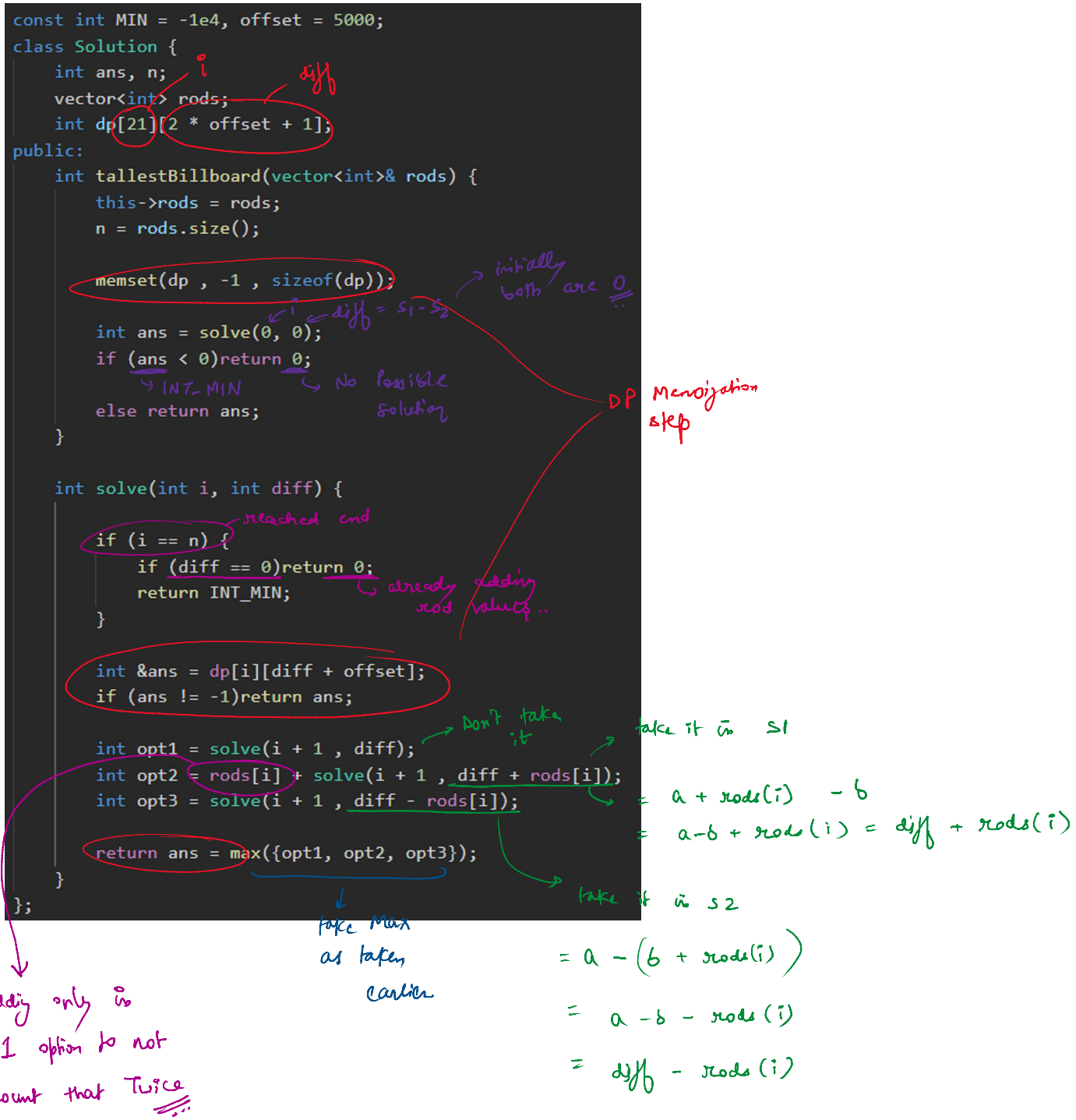
$-5000 \le s1 - s2 \le 5000$

(0 - 5000) (5000 - 0)

$-5000 \le diff \le 5000 \Rightarrow 0 \le diff \le 2 * 5000$

↳ offset = 5000

s1 - s2



CODE

```
const int MIN = -1e4, offset = 5000;
class Solution {
    int ans, n;
    vector<int> rods;
    int dp[21][2 * offset + 1];
public:
    int tallestBillboard(vector<int>& rods) {
```

```
        this->rods = rods;
        n = rods.size();
        memset(dp , -1 , sizeof(dp));
        int ans = solve(0, 0);
        if (ans < 0)return 0;
        else return ans;
    }
    int solve(int i, int diff) {
        if (i == n) {
            if (diff == 0)return 0;
            return INT_MIN;
        }
        int &ans = dp[i][diff + offset];
        if (ans != -1)return ans;
        int opt1 = solve(i + 1 , diff);
        int opt2 = rods[i] + solve(i + 1 , diff +
rods[i]);
        int opt3 = solve(i + 1 , diff - rods[i]);
        return ans = max({opt1, opt2, opt3});
    }
};
```