<https://leetcode.com/problems/distribute-money-to-maximum-children/description/>

You are given an integer money denoting the amount of money (in dollars) that you have and another integer children denoting the number of children that you must distribute the money to.

You have to distribute the money according to the following rules:

All money must be distributed.

Everyone must receive at least 1 dollar.

Nobody receives 4 dollars.

Return *the****maximum****number of children who may receive****exactly***8 *dollars if you distribute the money according to the aforementioned rules*. If there is no way to distribute the money, return -1.

**Example 1:**

**Input:** money = 20, children = 3

**Output:** 1

**Explanation:** The maximum number of children with 8 dollars will be 1. One of the ways to distribute the money is:

- 8 dollars to the first child.

- 9 dollars to the second child.

- 3 dollars to the third child.

It can be proven that no distribution exists such that number of children getting 8 dollars is greater than 1.

**Example 2:**

**Input:** money = 16, children = 2

**Output:** 2

**Explanation:** Each child can be given 8 dollars.

**Constraints:**

1 <= money <= 200

2 <= children <= 30

**Attempt 1: 2025-09-06**

**Solution 1: Math (60 min)**

class Solution {

    public int distMoney(int money, int children) {

        // Ensure each child gets at least $1

        int tmp = money - children;

        // Not enough money for minimum distribution

        if(tmp < 0) {

            return -1;

        }

        // Each count7 represents a child getting $8 ($1 base + $7)

        int count7 = tmp / 7;

        // Remaining money after distributing $7 increments

        int remain = tmp % 7;

        // If all children get exactly $8 and no money left

        if(count7 == children && remain == 0) {

            return count7;

        }

        // Special case: If one child would get $4 ($1 base + $3 remaining)

        // after (children-1) get $8

        if(count7 == children - 1 && remain == 3) {

            // Adjust to avoid $4 by reducing one $8 recipient

            return count7 - 1;

        }

        // If count7 < children-1: we can safely give $8 to count7 children

        // (because there are at least two children left to distribute the

        // leftover without creating $4).

        // e.g money = 41, children = 5

        // If count7 >= children-1: we cannot give more than children-1

        // (unless perfect case) because if we give children-1 and r=3 we

        // have a problem? But the special case already handles that.

        // Actually, if count7 >= children-1, we can only give children-1

        // in general

        // e.g money = 30, children = 5

        return Math.min(children - 1, count7);

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to Deepseek**

In addressing LeetCode problem 2591, "Distribute Money to Maximum Children," the goal is to distribute money among children such that all money is distributed, each child receives at least $1, no child receives exactly $4, and the number of children receiving exactly $8 is maximized. Below is a Java solution derived from the search results, followed by an explanation and complexity analysis.

**Java Solution**

class Solution {

public int distMoney(int money, int children) {

// Ensure each child gets at least $1

money -= children;

if (money < 0) {

return -1; // Not enough money for minimum distribution

}

int count7 = money / 7; // Each count7 represents a child getting $8 ($1 base + $7)

int remaining = money % 7; // Remaining money after distributing $7 increments

// If all children get exactly $8 and no money left

if (count7 == children && remaining == 0) {

return count7;

}

// Special case: If one child would get $4 ($1 base + $3 remaining) after (children-1) get $8

if (count7 == children - 1 && remaining == 3) {

return count7 - 1; // Adjust to avoid $4 by reducing one $8 recipient

}

// In general, the number of $8 recipients cannot exceed children-1 (to avoid issues) or count7

return Math.min(children - 1, count7);

}

}

**Explanation**

**Initial Check for Minimum Distribution**:

Subtract the number of children from the total money to account for the minimum $1 per child. If the result is negative, return -1 as it's impossible to meet the minimum requirement .

**Calculate Potential $8 Recipients**:

Each child receiving $8 requires an additional $7 beyond the base $1. Thus, count7 = money / 7 estimates the maximum children who could get $8 .

**Check for Perfect Distribution**:

If count7 equals the number of children and there is no remaining money (remaining == 0), all children receive exactly $8, so return count7 .

**Handle the Forbidden $4 Case**:

If exactly children - 1 children would get $8 and the remaining money is $3, one child would end up with $4 ($1 base + $3), which is forbidden. To avoid this, reduce the count of $8 recipients by one (e.g., distribute the extra $3 differently among multiple children) .

**General Case**:

The number of $8 recipients cannot exceed children - 1 (to ensure leftover money doesn't cause violations) or count7. Use Math.min(children - 1, count7) to cap the result .

**Complexity Analysis**

**Time Complexity**: (O(1)), as the solution involves only basic arithmetic operations and conditional checks .

**Space Complexity**: (O(1)), using a constant amount of extra space .

**Example Walkthrough**

**Example 1**: money = 20, children = 3

After base distribution: money = 20 - 3 = 17

count7 = 17 / 7 = 2, remaining = 3

Special case applies: count7 = 2 (== children - 1 = 2) and remaining = 3 → return 2 - 1 = 1 .

**Example 2**: money = 16, children = 2

After base distribution: money = 16 - 2 = 14

count7 = 14 / 7 = 2, remaining = 0

Perfect distribution: return 2 .

This solution efficiently handles edge cases while maximizing $8 distributions through mathematical reasoning.

**I am not very sure how is the logic below ? Can you show me 2 examples respectively mapping 2 options as 'children - 1' and 'count7'**

// In general, the number of $8 recipients cannot exceed children-1 (to avoid issues) or count7

return Math.min(children - 1, count7);

Certainly! Let's break down the logic behind Math.min(children - 1, count7) and provide two examples that illustrate when children - 1 is chosen and when count7 is chosen.

**Understanding the General Case**

After giving each child at least $1, we have money = money - children (which is non-negative). Then:

count7 = money / 7: This is the number of children who can be given an additional $7 (making their total $8) if we ignore constraints.

remaining = money % 7: The leftover money after giving as many $7 as possible.

However, we cannot have a child receiving exactly $4. Also, we must distribute all money without violating the rules.

Why Math.min(children - 1, count7)?

**count7 is the upper bound from available money**: We cannot have more than count7 children getting $8 because we don't have enough money to give more than that.

**But we cannot always use all count7**: Sometimes we have to limit to children - 1 to avoid problems with the leftover money.

Specifically:

If we try to give count7 children exactly $8, the leftover money is remaining = money % 7.

This leftover must be distributed to the other children - count7 children (who currently have $1 each).

However, if we give all leftover to one child, that child would get 1 + remaining. If remaining == 3, that child gets $4 which is forbidden.

Also, if there are multiple children left, we can distribute the leftover in a way that avoids $4 (e.g., giving $1 to several children). But if we have only one child left and remaining=3, we have a problem.

Therefore, to avoid the risk of having a child with $4, we often cannot give $8 to all count7 children. Instead, we may need to limit to children - 1 so that there is more than one child to distribute the leftover to (avoiding a concentration of $3 in one hand).

But note: if count7 is less than children - 1, then we can safely give $8 to count7 children without worrying about leftover causing $4? Why?

Because if there are at least two children who are not getting $8, we can distribute the leftover money (which is less than $7) without creating a $4. For example, if remaining=3, we can give $1 to three different children (so each gets $2, $2, $2) or $2 to one and $1 to another (so $3 and $2) — but we must avoid $4. Actually, we can always distribute the leftover without creating a $4 as long as there is more than one child to give to? However, if there is only one child left and remaining=3, we have a problem.

So the rule is:

If we have exactly children - 1 children getting $8 and remaining=3, we must reduce by one (handled in the special case).

In all other cases, we can give at most children - 1 to avoid having only one child left for leftover? Actually, no: if we have more than one child left, we can always distribute the leftover without creating a $4. Therefore, the only problem is when we have exactly one child left and remaining=3.

Therefore, if we give $8 to k children, then there are children - k children who currently have $1. The leftover is r. We need to distribute r to these children - k children without any child getting exactly $4.

How to avoid?

If k = children - 1, then only one child is left. If r=3, we cannot give $3 to that child (becomes $4). So we cannot have k = children -1 when r=3.

For any other k, we have at least two children to distribute the leftover. Then we can always avoid $4? For example:

If r=3: we can give $1 to three children (each becomes $2) or $2 to one and $1 to another (becomes $3 and $2). No $4.

Similarly, for any r between 0 and 6, we can distribute without creating $4 if we have at least two children.

Therefore, the only forbidden case is when k = children - 1 and r=3.

So why in the general case we return Math.min(children-1, count7)?

Actually, we can sometimes give more than children-1? For example, if count7 = children and r=0, we give all children $8 (which is allowed).

Also, if count7 = children but r>0, we cannot give all children $8 (because we have leftover). Then we must reduce.

But the code already handles the perfect case (count7==children and r==0) and the special case (count7==children-1 and r==3). So in the general case, we are left with:

If count7 < children-1: we can safely give $8 to count7 children (because there are at least two children left to distribute the leftover without creating $4).

If count7 >= children-1: we cannot give more than children-1 (unless perfect case) because if we give children-1 and r=3 we have a problem? But the special case already handles that. Actually, if count7 >= children-1, we can only give children-1 in general? Why?

Wait, let's see the code flow:

If perfect case: return count7.

Else if special case (count7==children-1 and r==3): return count7-1.

Else: return Math.min(children-1, count7).

So in the else case, if count7 >= children-1, we return children-1. But why not more?

For example, suppose children=5, money=40:

Base: 40-5=35.

count7=35/7=5, which equals children. But r=0? Actually 35%7=0. So it is perfect -> returns 5.

Another example: children=5, money=41:

Base: 41-5=36.

count7=36/7=5 (since 5\*7=35), r=1.

Not perfect (r!=0), and not special (count7=5 is not children-1=4? So it goes to else.

Then return Math.min(4,5)=4.

Why can't we give 5 children $8? Because we have leftover $1. If we give 5 children $8, we need to distribute the leftover $1 to one of them? But that would make one child $9. However, that is allowed. So why not 5?

Wait, the rules: we want to maximize the number of children with exactly $8. But if we give 5 children $8, we have $1 left. We must give that $1 to one child, making it $9. So we have 4 children with $8 and one with $9. So the number of children with exactly $8 is 4.

Therefore, we cannot have 5 because we have leftover. So we must reduce to 4.

In general, if we try to give $8 to all count7 children, but there is leftover, then we have to distribute the leftover to the children. This might reduce the number of exactly $8 children.

Therefore, the maximum number of $8 children is limited by:

The available money: count7

And the need to distribute leftover without reducing the count: which often forces us to children-1 if there is leftover.

But actually, if we have leftover, we can never have all children at $8. So the maximum is at most children-1 if there is leftover? Not exactly: we can have more than children-1 only if there is no leftover (perfect case).

So in the code, after handling the perfect case and the special case, the general rule is that we cannot exceed children-1 because there is leftover. Also, we cannot exceed count7. Therefore, we take the minimum.

Now, examples:

**Example 1: When children - 1 is chosen (because count7 is larger)**

money = 41, children = 5

After base: 41 - 5 = 36

count7 = 36 / 7 = 5, remaining = 1

Not perfect (because remaining !=0)

Not special (count7=5 != children-1=4)

So return Math.min(4,5)=4.

Explanation: We can give 4 children $8 (using 4\*7=28 from the 36). Then we have 36-28=8 left. Now we have 5 children: 4 have $8, one has $1. We need to distribute the $8 to the children without creating $4. We can give to the last child: but if we give $3, it becomes $4 -> forbidden. So we cannot give $3. Instead, we can give $2 to several children? Actually, we have one child with $1 and four with $8. We can give the $8 to the children who already have $8? That would make them more than $8. But we want to maximize the number of exactly $8.

Alternatively, we can give the leftover to the children who are not at $8. There is only one such child. So we have to give the $8 to that child. But we cannot give $3 (becomes $4). So we must avoid $3. How to distribute? We can give $1 to the child repeatedly? But we have to distribute all.

Actually, we can give up to $7 to the poor child without making it $8? But if we give $7, it becomes $8. So that is good. Then we have 5 children with $8? But we have $8 leftover. So we can give $7 to the poor child to make it $8. Then we have $1 left. Then we have to give that $1 to one of the $8 children, making it $9. So we end with 4 children at $8 and one at $9.

Therefore, the maximum is 4.

**Example 2: When count7 is chosen (because it is smaller than children-1)**

money = 30, children = 5

After base: 30 - 5 = 25

count7 = 25 / 7 = 3 (since 3\*7=21), remaining=4

Not perfect, not special (count7=3 !=4)

So return Math.min(4,3)=3.

Explanation: We can give 3 children $8. Then we have 25-21=4 left. There are 2 children who still have only $1. We can distribute the $4 to these two without creating $4: for example, give $2 to each (so they become $3 and $3). Then we have 3 children with $8 and 2 with $3. So no $4, and we have 3 exactly $8.

Can we give more than 3? If we try to give 4 children $8, we need 4\*7=28. But we only have 25. So we cannot. Therefore, the maximum is 3.

**Conclusion**

The general return Math.min(children-1, count7) is because:

If count7 is less than children-1, we can safely use count7 (there are enough children to distribute the leftover without creating $4).

If count7 is greater than or equal to children-1, we cannot use more than children-1 (unless perfect) because having leftover forces us to reduce the count to avoid issues (as in example1).

The code already handles the perfect case and the special case, so the else case covers all others where the maximum is limited by either the money (count7) or the number of children minus one (children-1).

Refer to

[L1103.Distribute Candies to People (Ref.L2591)](note://WEB708d8b378a3c00aaddd0f19115825df0)

[L1716.Calculate Money in Leetcode Bank (Ref.L2591)](note://WEBe491030c5682ca6c890dc52ef2f2f9d9)

[L2305.Fair Distribution of Cookies (Ref.L410,L473,L698)](note://WEBccd4ba98d71625f4c769ccee45f02d79)