You are given an array of N integers. Find the *minimum* number of integers you need to delete from the array such that the absolute difference between each pair of integers in the remaining array will become equal.

Input Format

- The first line of input contains a single integer T denoting the number of test cases. The description of T test cases follows.
- The first line of each test case contains an integer N.
- The second line of each test case contains N space-separated integers $A_1,A_2,\ldots,A_N$.

Output Format

For each test case, print a single line containing one integer - the minimum number of integers to be deleted to satisfy the given condition.

Constraints

- $1 \le T \le 10^4$
- $1 \le N \le 10^5$
- $1 \le A_i \le 10^9$
- Sum of N over all test cases does not exceed $5 \cdot 10^5$.

INPUT

3

2

1 2

5

2 5 1 2 2

4

1 2 1 2

OUTPUT

```
0
2
2
```

****

2] Finally, progress reached the Madoka family and she decided to play with her little sister in the sensational game Space Arrays.

The rules of the game are as follows:

- **Initially, a sequence $a_1,a_2,\ldots,a_N$ is given.**
- **The players alternate turns.**
- **In each turn, the current player must choose an index i and increment a_i, i.e. change a_i to a_i+1.**
- **Afterwards, if there is no permutation $p_1,p_2,\ldots,p_N$ of the integers 1 through N such that $a_i \le p_i$ holds for each valid i, the current player loses.**
- **Otherwise, the game continues with the next turn.**

Madoka is asking you to help her — tell her if the first player (the player that plays in the first turn) or second player wins this game if both play optimally.

Input

- **The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.**
- **The first line of each test case contains a single integer N.**
- **The second line contains N space-separated integers $a_1,a_2,\ldots,a_N$.**

Output

For each test case, print a single line containing the string "First" if the first player wins or "Second" if the second player wins (without quotes).

Constraints

- $1 \le T \le 2 \cdot 10^4$
- $1 \le N \le 2 \cdot 10^5$
- The sum of N over all test cases doesn't exceed $2 \cdot 10^5$
- $1 \le a_i \le N$ for each valid i

INPUT

```
4
4
1 2 3 3
4
1 1 3 3
5
1 2 2 1 5
3
2 2 3
```

OUTPUT

```
First
Second
Second
Second
```

****

3] Chef and his best friend Aleksa are into mathematical games these days. Today, they have some (≥ 0) black cells represented as B, and a white cell represented as W, lying randomly in a straight line. They have decided to play with these cells. In a move, a player chooses some (> 0)

black cells lying on any one side of the white cell and remove them. It should be noted that a player is not allowed to choose black cells from both side of the given white cell. Both the players alternate their moves, and play optimally. The player who is unable to move in his respective turn will lose the game.

Aleksa, being a girl, has a habit of playing first. But Chef is fairly smart himself, and will not play the game if he is going to lose it. Therefore, he wants to know the winner beforehand. Can you tell him who is going to win the game for the given configuration of cells?

Input

First line of input contains a single integer T denoting the number of test cases. First and the only line of each test case contains a string S consisting of the characters 'B' and 'W', denoting black and white cells, respectively.

Output

For each test case, output "Chef" if chef wins the game for the given configuration. Else print "Aleksa". (quotes for clarity only).

Constraints

1 ≤ T ≤ 10 1 ≤ |S| ≤ 10000 S contains exactly 1 white cell.

Scoring

Subtask 1: 1 ≤ T ≤ 10, 1 ≤ |S| ≤ 10 : (30 pts) Subtask 2: 1 ≤ T ≤ 10, 1 ≤ |S| ≤ 100 : (30 pts) Subtask 3: 1 ≤ T ≤ 10, 1 ≤ |S| ≤ 10000 : (40 pts)

Sample 1:

Input

```
3
W
```

```
    BW
    BWBB
Output
    Chef
    Aleksa
    Aleksa
```

****

4] Chefs from all over the globe gather each year for an international convention. Each chef represents some country. Please, note that more than one chef can represent a country.

Each of them presents their best dish to the audience. The audience then sends emails to a secret and secure mail server, with the subject being the name of the chef whom they wish to elect as the "Chef of the Year".

You will be given the list of the subjects of all the emails. Find the country whose chefs got the most number of votes, and also the chef who got elected as the "Chef of the Year" (the chef who got the most number of votes).

If several countries got the maximal number of votes, consider the country with the lexicographically smaller name among them to be a winner. Similarly if several chefs got the maximal number of votes, consider the chef with the lexicographically smaller name among them to be a winner.

The string $A = a_1a_2...a_n$ is called lexicographically smaller then the string $B = b_1b_2...b_m$ in the following two cases:

- there exists index $i \le \min\{n, m\}$ such that $a_j = b_j$ for $1 \le j < i$ and $a_i < b_i$;
- A is a proper prefix of B, that is, $n < m$ and $a_j = b_j$ for $1 \le j \le n$.

The characters in strings are compared by their ASCII codes.

Refer to function strcmp in C or to standard comparator < for string data structure in C++ for details.

Input

The first line of the input contains two space-separated integers N and M denoting the number of chefs and the number of emails respectively. Each of the following N lines contains two space-separated strings, denoting the name of the chef and his country respectively. Each of the following M lines contains one string denoting the subject of the email.

Output

Output should consist of two lines. The first line should contain the name of the country whose chefs got the most number of votes. The second line should contain the name of the chef who is elected as the "Chef of the Year".

Constraints

- **1 ≤ N ≤ 10000 (10**^{4}**)**
- **1 ≤ M ≤ 100000 (10**^{5}**)**
- **Each string in the input contains only letters of English alphabets (uppercase or lowercase)**
- **Each string in the input has length not exceeding 10**
- **All chef names will be distinct**
- **Subject of each email will coincide with the name of one of the chefs**

INPUT

```
1 3
Leibniz Germany
Leibniz
Leibniz
Leibniz
```

OUTPUT

```
Germany
Leibniz
```

****

5]You are an administrator of a popular quiz website. Every day, you hold a quiz on the website.

There are N users (numbered 1 through N) and M admins (numbered $N+1$ through $N+M$). For each quiz, each user is allowed to attempt it at most once. Sometimes, admins attempt the quizzes for testing purposes too; an admin may attempt a quiz any number of times.

On a certain day, the quiz was attempted K times. For each i ($1 \le i \le K$), the i-th of these attempts was made by the person with number L_i. Find all the users who attempted the quiz more than once, in order to disqualify them.

Input

- The first line of the input contains a single integer T denoting the number of test cases. The description of T test cases follows.
- The first line of each test case contains three space-separated integers N, M and K.
- The second line contains K space-separated integers $L_1, L_2, \ldots, L_K$.

Output

For each test case, print a single line containing an integer D denoting the number of users who should be disqualified, followed by a space and D space-separated integers denoting their numbers in ascending order.

Constraints

- $1 \le T \le 10$
- $0 \le N, M \le 10^5$
- $1 \le N+M, K \le 10^5$
- $1 \le L_i \le N+M$ for each valid i

Input

```
3
```

```
1 1 1
1
1 1 2
1 1
5 5 10
2 5 2 5 2 4 10 10 10 10
```

Output

```
0
1 1
2 2 5
```

****

6] An N-*bonacci sequence* is an infinite sequence $F_1, F_2, \dots$ such that for each integer $i > N$, F_i is calculated as $f(F_{i-1}, F_{i-2}, \dots, F_{i-N})$, where f is some function. A *XOR* N-*bonacci sequence* is an N-bonacci sequence for which $f(F_{i-1}, F_{i-2}, \dots, F_{i-N}) = F_{i-1} \oplus F_{i-2} \oplus \dots \oplus F_{i-N}$, where $\oplus$ denotes the bitwise XOR operation.

Recently, Chef has found an interesting sequence $S_1, S_2, \dots$, which is obtained from prefix XORs of a XOR N-bonacci sequence $F_1, F_2, \dots$. Formally, for each positive integer i, $S_i = F_1 \oplus F_2 \oplus \dots \oplus F_i$. You are given the first N elements of the sequence F, which uniquely determine the entire sequence S.

You should answer Q queries. In each query, you are given an index k and you should calculate S_k. It is guaranteed that in each query, S_k does not exceed 10^{50}.

Input

- The first line of the input contains two space-separated integers N and Q.
- The second line contains N space-separated integers $F_1, F_2, \dots, F_N$.
- The following Q lines describe queries. Each of these lines contains a single integer k.

Output

For each query, print a single line containing one integer S_k.

Constraints

- $1 \le N, Q \le 10^5$
- $0 \le F_i \le 10^9$ for each i such that $1 \le i \le N$
- $1 \le k \le 10^9$

Sample 1:

```
Input
3 4
0 1 2
7
2
5
1000000000

OUTPUT
3
1
0
0
```

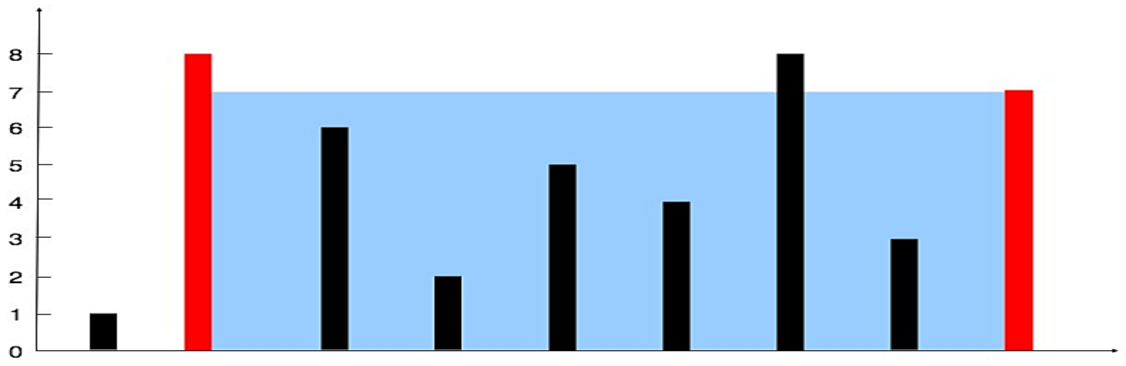
****

7] You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.



nput: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]
Output: 1

Constraints:

- **n == height.length**
- **2 <= n <= 105**
- **0 <= height[i] <= 104**

****

8] Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

- **'.' Matches any single character.**
- **'*' Matches zero or more of the preceding element.**

The matching should cover the entire input string (not partial).

Example 1:

```
Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
```

Example 2:

```
Input: s = "aa", p = "a*"
Output: true
Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it
becomes "aa".
```

Example 3:

```
Input: s = "ab", p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".
```

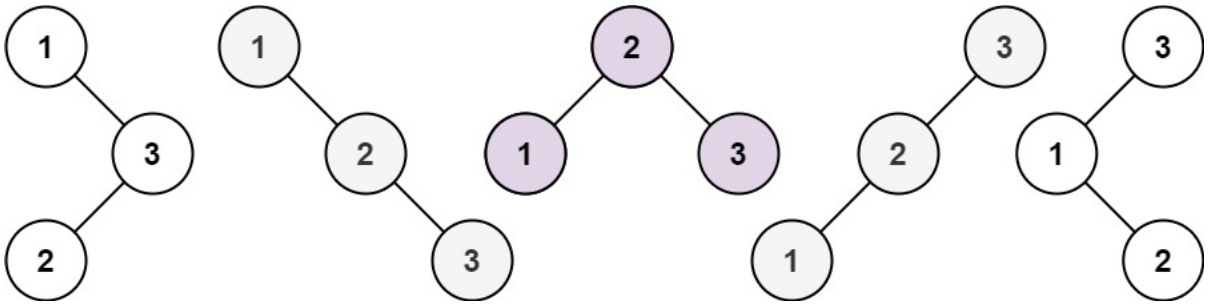
Constraints:

- 1 <= s.length <= 20
- 1 <= p.length <= 30
- s contains only lowercase English letters.
- p contains only lowercase English letters, '.', and '*'.
- It is guaranteed for each appearance of the character '*', there will be a previous valid character to match.

****

9] Given an integer n, return *the number of structurally unique BST's (binary search trees) which has exactly* n *nodes of unique values from* 1 *to* n.

Example 1:



Input: n = 3

Output: 5

Example 2:

Input: n = 1

Output: 1

Constraints:

- **1 <= n <= 19**

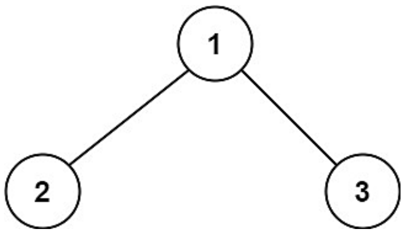
****

**10]	A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.**

The path sum of a path is the sum of the node's values in the path.

Given the root of a binary tree, return *the maximum path sum of any non-empty path*.

Example 1:

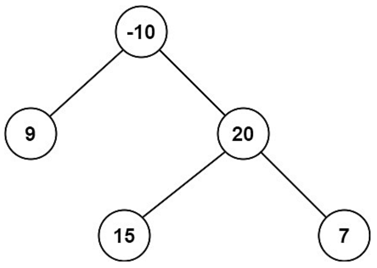


Input: root = [1,2,3]

Output: 6

Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.

Example 2:



Input: root = [-10,9,20,null,null,15,7]

Output: 42

Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.

Constraints:

- The number of nodes in the tree is in the range [1, 3 * 104].
- -1000 <= Node.val <= 1000

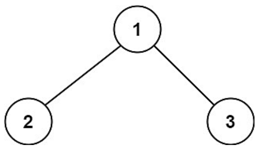
****

11] A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

Given the root of a binary tree, return *the maximum path sum of any non-empty path.*

Example 1:

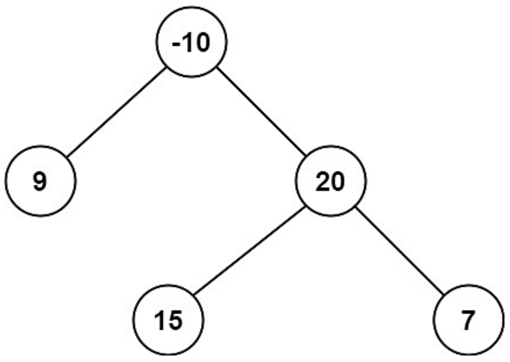


Input: root = [1,2,3]

Output: 6

Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.

Example 2:



Input: root = [-10,9,20,null,null,15,7]

Output: 42

Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of 15 + 20 + 7 = 42.

Constraints:

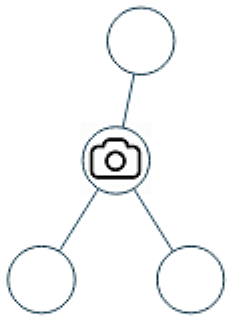
- **The number of nodes in the tree is in the range [1, 3 * 104].**
- **-1000 <= Node.val <= 1000**
- **0 <= amount <= 104**

****

12] You are given the root of a binary tree. We install cameras on the tree nodes where each camera at a node can monitor its parent, itself, and its immediate children.

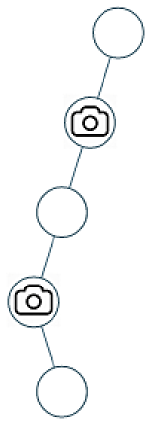
Return *the minimum number of cameras needed to monitor all nodes of the tree*.

Example 1:



Input: root = [0,0,null,0,0]
Output: 1
Explanation: One camera is enough to monitor all nodes if placed as shown.

Example 2:



Input: root = [0,0,null,0,null,0,null,null,0]
Output: 2
Explanation: At least two cameras are needed to monitor all nodes of the tree. The above image shows one of the valid configurations of camera placement.

Constraints:

- The number of nodes in the tree is in the range [1, 1000].
- Node.val == 0

****

13]   Store, there are n items to sell. Each item has a price. However, there are some special offers, and a special offer consists of one or more different kinds of items with a sale price.

You are given an integer array price where price[i] is the price of the ith item, and an integer array needs where needs[i] is the number of pieces of the ith item you want to buy.

You are also given an array special where special[i] is of size n + 1 where special[i][j] is the number of pieces of the jth item in the ith offer and special[i][n] (i.e., the last integer in the array) is the price of the ith offer.

Return *the lowest price you have to pay for exactly certain items as given, where you could make optimal* *use of the special offers*. You are not allowed to buy more items than you want, even if that would lower the overall price. You could use any of the special offers as many times as you want.

Example 1:

```
Input: price = [2,5], special = [[3,0,5],[1,2,10]], needs = [3,2]
Output: 14
Explanation: There are two kinds of items, A and B. Their prices are $2 and $5 respectively.
In special offer 1, you can pay $5 for 3A and 0B
In special offer 2, you can pay $10 for 1A and 2B.
You need to buy 3A and 2B, so you may pay $10 for 1A and 2B (special offer #2), and $4 for 2A.
```

Example 2:

```
Input: price = [2,3,4], special = [[1,1,0,4],[2,2,1,9]], needs = [1,2,1]
Output: 11
Explanation: The price of A is $2, and $3 for B, $4 for C.
You may pay $4 for 1A and 1B, and $9 for 2A ,2B and 1C.
You need to buy 1A ,2B and 1C, so you may pay $4 for 1A and 1B (special offer #1), and $3 for 1B, $4 for 1C.
You cannot add more items, though only $9 for 2A ,2B and 1C.
```

Constraints:

- n == price.length == needs.length
- 1 <= n <= 6
- 0 <= price[i], needs[i] <= 10
- 1 <= special.length <= 100
- special[i].length == n + 1
- 0 <= special[i][j] <= 50

***Given a

14] string s representing a valid expression, implement a basic calculator to evaluate it, and return *the result of the evaluation*.

Note: You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().

Example 1:

Input: s = "1 + 1"
Output: 2

Example 2:

Input: s = " 2-1 + 2 "
Output: 3

Example 3:

Input: s = "(1+(4+5+2)-3)+(6+8)"
Output: 23

Constraints:

- 1 <= s.length <= $3 * 10^5$
- s consists of digits, '+', '-', '(', ')', and ' '.
- s represents a valid expression.
- '+' is not used as a unary operation (i.e., "+1" and "+(2 + 3)" is invalid).
- '-' could be used as a unary operation (i.e., "-1" and "-(2 + 3)" is valid).
- There will be no two consecutive operators in the input.
- Every number and running calculation will fit in a signed 32-bit integer.

****

15] You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:

- An integer x.
  - Record a new score of x.
- '+'.
  - Record a new score that is the sum of the previous two scores.
- 'D'.
  - Record a new score that is the double of the previous score.
- 'C'.

- Invalidate the previous score, removing it from the record.

Return *the sum of all the scores on the record after applying all the operations.*

The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.

Example 1:

```
Input: ops = ["5","2","C","D","+"]
Output: 30
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.
```

Example 2:

```
Input: ops = ["5","-2","4","C","D","9","+","+"]
Output: 27
Explanation:
"5" - Add 5 to the record, record is now [5].
"-2" - Add -2 to the record, record is now [5, -2].
"4" - Add 4 to the record, record is now [5, -2, 4].
"C" - Invalidate and remove the previous score, record is now [5, -2].
"D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].
"9" - Add 9 to the record, record is now [5, -2, -4, 9].
"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].
"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].
The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.
```

Example 3:

```
Input: ops = ["1","C"]
Output: 0
Explanation:
```

```
"1" - Add 1 to the record, record is now [1].
"C" - Invalidate and remove the previous score, record is now [].
Since the record is empty, the total sum is 0.
```

Constraints:

- 1 <= operations.length <= 1000
- operations[i] is "C", "D", "+", or a string representing an integer in the range [-3 * 104, 3 * 104].
- For operation "+", there will always be at least two previous scores on the record.
- For operations "C" and "D", there will always be at least one previous score on the record.

****

16] Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Notes:

- You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

```
Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]
```

```
Explanation
MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false
```

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, peek, and empty.
- All the calls to pop and peek are valid.

Follow-up: Can you implement the queue such that each operation is amortized O(1) time complexity? In other words, performing n operations will take overall O(n) time even if one of those operations may take longer.

****

17] Design an algorithm that collects daily price quotes for some stock and returns the span of that stock's price for the current day.

The span of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backward) for which the stock price was less than or equal to today's price.

- For example, if the price of a stock over the next 7 days were [100,80,60,70,60,75,85], then the stock spans would be [1,1,1,2,1,4,6].

Implement the StockSpanner class:

- StockSpanner() Initializes the object of the class.
- int next(int price) Returns the span of the stock's price given that today's price is price.

Example 1:

```
Input
["StockSpanner", "next", "next", "next", "next", "next", "next", "next"]
[[], [100], [80], [60], [70], [60], [75], [85]]
Output
```

```
[null, 1, 1, 1, 2, 1, 4, 6]

Explanation
StockSpanner stockSpanner = new StockSpanner();
stockSpanner.next(100); // return 1
stockSpanner.next(80);  // return 1
stockSpanner.next(60);  // return 1
stockSpanner.next(70);  // return 2
stockSpanner.next(60);  // return 1
stockSpanner.next(75);  // return 4, because the last 4 prices (including today's price of 75) were less than or equal to today's price.
stockSpanner.next(85);  // return 6
```

Constraints:

- 1 <= price <= 10^5
- At most 10^4 calls will be made to next.

****

18] Given an array of distinct integers candidates and a target integer target, return *a list of all unique combinations of* candidates *where the chosen numbers sum to* target. You may return the combinations in any order.

The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]

Example 3:

Input: candidates = [2], target = 1
Output: []

Constraints:

- 1 <= candidates.length <= 30
- 2 <= candidates[i] <= 40
- All elements of candidates are distinct.
- 1 <= target <= 500

**A 19]

string is called a happy prefix if is a non-empty prefix which is also a suffix (excluding itself).

Given a string s, return *the longest happy prefix of* s. Return an empty string "" if no such prefix exists.

Example 1:

Input: s = "level"
Output: "l"
Explanation: s contains 4 prefix excluding itself ("l", "le", "lev", "leve"), and suffix ("l", "el", "vel", "evel"). The largest prefix which is also suffix is given by "l".

Example 2:

Input: s = "ababab"
Output: "abab"
Explanation: "abab" is the largest prefix which is also suffix. They can overlap in the original string.

Constraints:

- 1 <= s.length <= 105
- s contains only lowercase English letters.

****

20] Design a HashSet without using any built-in hash table libraries.

Implement `MyHashSet` class:

- `void add(key)` Inserts the value `key` into the HashSet.
- `bool contains(key)` Returns whether the value `key` exists in the HashSet or not.
- `void remove(key)` Removes the value `key` in the HashSet. If `key` does not exist in the HashSet, do nothing.

Example 1:

```
Input
["MyHashSet", "add", "add", "contains", "contains", "add", "contains",
"remove", "contains"]
[[], [1], [2], [1], [3], [2], [2], [2], [2]]
Output
[null, null, null, true, false, null, true, null, false]

Explanation
MyHashSet myHashSet = new MyHashSet();
myHashSet.add(1);      // set = [1]
myHashSet.add(2);      // set = [1, 2]
myHashSet.contains(1); // return True
myHashSet.contains(3); // return False, (not found)
myHashSet.add(2);      // set = [1, 2]
myHashSet.contains(2); // return True
myHashSet.remove(2);   // set = [1]
myHashSet.contains(2); // return False, (already removed)
```

Constraints:

- `0 <= key <= 10`^{6}
- At most `10`^{4} calls will be made to `add`, `remove`, and `contains`.