**1. You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.**

Increment the large integer by one and return the resulting array of digits.

Example 1:

Input: digits = [1,2,3]

Output: [1,2,4]

Explanation: The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1,2,4].

Example 2:

Input: digits = [9]

Output: [1,0]

Explanation: The array represents the integer 9.

Incrementing by one gives 9 + 1 = 10.

Thus, the result should be [1,0].

#include <stdio.h>

int\* plusOne(int\* digits, int digitsSize, int\* returnSize) {

int carry = 1;

for (int i = digitsSize - 1; i >= 0; i--) {

int sum = digits[i] + carry;

digits[i] = sum % 10;

carry = sum / 10;

}

if (carry == 1) {

int\* newDigits = (int\*)malloc(sizeof(int) \* (digitsSize + 1));

newDigits[0] = 1;

for (int i = 0; i < digitsSize; i++) {

newDigits[i + 1] = digits[i];

}

\*returnSize = digitsSize + 1;

return newDigits;

} else {

\*returnSize = digitsSize;

return digits;

}

}

int main() {

int digits[] = {1, 2, 3};

int digitsSize = sizeof(digits) / sizeof(digits[0]);

int returnSize;

int\* result = plusOne(digits, digitsSize, &returnSize);

printf("Incremented digits: ");

for (int i = 0; i < returnSize; i++) {

printf("%d", result[i]);

}

printf("\n");

free(result);

return 0;

}

**2. You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position. Return true if you can reach the last index, or false otherwise.**

Example 1:

Input: nums = [2,3,1,1,4]

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [3,2,1,0,4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

#include <stdbool.h>

bool can\_jump(int\* nums, int numsSize) {

int reachable = 0; // Tracks the farthest reachable index

for (int i = 0; i < numsSize; i++) {

if (i > reachable) { // If current index is beyond reachable, it's not possible to proceed

return false;

}

reachable = fmax(reachable, i + nums[i]); // Update reachable with the farthest possible jump

}

return reachable >= numsSize - 1; // True if the last index is reachable

}

3. Given an integer array nums, find the subarray with the largest sum, and return its sum.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]

Output: 6

Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input: nums = [1]

Output: 1

Explanation: The subarray [1] has the largest sum 1.

Example 3:

Input: nums = [5,4,-1,7,8]

Output: 23

Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

\*

#include <stdio.h>

int maxSubArray(int\* nums, int numsSize) {

int current\_sum = nums[0]; // Keep track of the current subarray sum

int max\_sum = nums[0]; // Keep track of the maximum subarray sum found so far

for (int i = 1; i < numsSize; i++) {

// If the current sum is negative, discard it and start a new subarray

current\_sum = fmax(nums[i], current\_sum + nums[i]);

max\_sum = fmax(max\_sum, current\_sum);

}

return max\_sum;

}

int main() {

int nums1[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

int nums2[] = {1};

int nums3[] = {5, 4, -1, 7, 8};

printf("Maximum subarray sum for nums1: %d\n", maxSubArray(nums1, sizeof(nums1) / sizeof(nums1[0])));

printf("Maximum subarray sum for nums2: %d\n", maxSubArray(nums2, sizeof(nums2) / sizeof(nums2[0])));

printf("Maximum subarray sum for nums3: %d\n", maxSubArray(nums3, sizeof(nums3) / sizeof(nums3[0])));

return 0;

}