* 1. Write a function “canSum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments.

function canSum(targetSum, numbers) {

if (targetSum === 0) {

return true;

}

if (targetSum < 0) {

return false;

}

for (let num of numbers) {

const remainder = targetSum - num;

if (canSum(remainder, numbers) === true) {

return true;

}

}

return false;

}

function canSumMemo(targetSum, numbers, memo = {}) {

if (targetSum in memo) {

return memo[targetSum];

}

if (targetSum === 0) {

return true;

}

if (targetSum < 0) {

return false;

}

for (let num of numbers) {

const remainder = targetSum - num;

if (canSumMemo(remainder, numbers, memo) === true) {

memo[targetSum] = true;

return true;

}

}

memo[targetSum] = false;

return false;

}

* 1. Write a function “HowManyWaySum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Then return total no. of ways

function howManyWaySum(target, numbers) {

  if (target == 0) return 1;

  if (target < 0) return 0;

  var total = 0;

  for (let num of numbers) {

    const remains = target - num;

    total += howManyWaySum(remains, numbers);

  }

  return total;

}

function howManyWaySumMemo(target, numbers, memo = {}) {

  if (target in memo) return memo[target];

  if (target == 0) return 1;

  if (target < 0) return 0;

  var total = 0;

  for (let num of numbers) {

    const remains = target - num;

    total += howManyWaySumMemo(remains, numbers, memo);

    memo[target] = total;

  }

  memo[target] = total;

  return total;

}

* 1. Write a function “HowWaySum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Then return how it sum

function howSum(targetSum, numbers) {

  if (targetSum === 0) {

    return [];

  }

  if (targetSum < 0) {

    return null;

  }

  for (let num of numbers) {

    const remainder = targetSum - num;

    const remainderResult = howSum(remainder, numbers);

    if (remainderResult !== null) {

      return [...remainderResult, num];

    }

  }

  return null;

}

function howSumMemo(targetSum, numbers, memo = {}) {

  if (targetSum in memo) {

    return memo[targetSum];

  }

  if (targetSum === 0) {

    return [];

  }

  if (targetSum < 0) {

    return null;

  }

  for (let num of numbers) {

    const remainder = targetSum - num;

    const remainderResult = howSumMemo(remainder, numbers, memo);

    if (remainderResult !== null) {

      memo[targetSum] = [...remainderResult, num];

      return memo[targetSum];

    }

  }

  memo[targetSum] = null;

  return null;

}

* 1. Write a function “bestWaySum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Then return the best sum

function bestSum(target, numbers) {

  if (target == 0) return [];

  if (target < 0) return null;

  let shortComb = null;

  for (let num of numbers) {

    const remainder = target - num;

    const result = bestSum(remainder, numbers);

    if (result != null) {

      let combination = [...result, num];

      if (shortComb == null || combination.length < shortComb.length) {

        shortComb = combination;

      }

    }

  }

  return shortComb;

}

function bestSumMemo(target, numbers, memo = {}) {

  if (target in memo) return memo[target];

  if (target == 0) return [];

  if (target < 0) return null;

  let shortComb = null;

  for (let num of numbers) {

    const remainder = target - num;

    const result = bestSumMemo(remainder, numbers, memo);

    if (result != null) {

      let combination = [...result, num];

      if (shortComb == null || combination.length < shortComb.length) {

        shortComb = combination;

      }

    }

  }

  memo[target] = shortComb;

  return shortComb;

}

2.1. Write a function “canConstruct (target,words)” that takes in a target and an array of string as arguments.

function canConstruct(target, words) {

if (target == "") return true;

for (let word of words) {

if (target.indexOf(word) == 0) {

const suffix = target.slice(word.length);

if (canConstruct(suffix, words) == true) {

return true;

}

}

}

return false;

}

function canConstructMemo(target, words, memo = {}) {

if (target in memo) {

return memo[target];

}

if (target == "") return true;

for (let word of words) {

if (target.indexOf(word) == 0) {

const suffix = target.slice(word.length);

if (canConstructMemo(suffix, words, memo) == true) {

memo[target] = true;

return true;

}

}

}

memo[target] = false;

return false;

}

2.2. Write a function “howManyWayConstruct( (target,words)” that takes in a target and an array of string as arguments. Then return total posibilities

function howManyWayConstruct(target, words) {

if (target == "") return 1;

var total = 0;

for (let word of words) {

if (target.indexOf(word) == 0) {

const suffix = target.slice(word.length);

total += howManyWayConstruct(suffix, words);

}

}

return total;

}

function howManyWayConstructMemo(target, words, memo = {}) {

if (target == "") return 1;

var total = 0;

for (let word of words) {

if (target.indexOf(word) == 0) {

const suffix = target.slice(word.length);

total += howManyWayConstructMemo(suffix, words, memo);

}

}

return total;

}

2.3. Write a function “allWayConstruct(target,words)” that takes in a target and an array of string as arguments. Then return all posiblities

function allWayConstruct(target, words) {

if (target == "") return [[]];

var result = [];

for (let word of words) {

if (target.indexOf(word) == 0) {

const suffix = target.slice(word.length);

const suffixWays = allWayConstruct(suffix, words);

const ways = suffixWays.map((keys) => [word, ...keys]);

result.push(...ways);

}

}

return result;

}

console.log(allWayConstruct("abcdef", ["abc", "def", "a", "bc", "b", "c"]));

function allWayConstructMemo(target, words, memo = {}) {

if (target in memo) return memo[target];

if (target == "") return [[]];

var result = [];

for (let word of words) {

if (target.indexOf(word) == 0) {

const suffix = target.slice(word.length);

const suffixWays = allWayConstructMemo(suffix, words, memo);

const ways = suffixWays.map((keys) => [word, ...keys]);

result.push(...ways);

}

}

memo[target] = result;

return result;

}

2.4. Write a function “bestWayConstruct(target,words)” that takes in a target and an array of string as arguments. Then return best way

function bestWayConstruct(target, words) {

  if (target == "") return [];

  var best = null;

  for (let word of words) {

    if (target.indexOf(word) == 0) {

      const suffix = target.slice(word.length);

      const suffixWays = bestWayConstruct(suffix, words);

      const ways = [word, ...suffixWays];

      if (!best || ways.length < best.length) best = ways;

    }

  }

  return best;

}

function bestWayConstructMemo(target, words, memo = {}) {

  if (target in memo) return memo[target];

  if (target == "") return [];

  var best = null;

  for (let word of words) {

    if (target.indexOf(word) == 0) {

      const suffix = target.slice(word.length);

      const suffixWays = bestWayConstructMemo(suffix, words, memo);

      const ways = [word, ...suffixWays];

      if (!best || ways.length < best.length) {

        memo[target] = ways;

        best = ways;

      }

    }

  }

  memo[target] = best;

  return best;

}

* 1. Write a function “canSum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Using tablation method

function canSumTablation(target,numbers){

const table=Array(target+1).fill(false);

table[0]=true;

for(let i=0;i<target;i++){

if(table[i]==true){

for(let num of numbers){

if(i+num<=target){

table[i+num]=true

}

}

}

}

return table[target]

}

* 1. Write a function “HowManyWaySum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Then return total no. of ways, Using tablation method
  2. Write a function “HowWaySum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Then return how it sum, Using tablation method

function howSumTablation(target,numbers){

const table=Array(target+1).fill(null);

table[0]=[];

for(let i=0;i<target;i++){

if(table[i]!=null){

for(let num of numbers){

if(i+num<=target){

table[i+num]=[...table[i],num]

}

}

}

}

return table[target]

}

* 1. Write a function “bestWaySum(targetSum,numbers)” that takes in a targetSum and an array of numbers as arguments. Then return the best sum, Using tablation method

function bestSumTablation(target,numbers){

const table=Array(target+1).fill(null);

table[0]=[];

for(let i=0;i<target;i++){

if(table[i]!=null){

for(let num of numbers){

if(i+num<=target){

var comb=[...table[i],num]

if(table[i+num]==null || comb.length<table[i+num].length)

table[i+num]=comb

}

}

}

}

return table[target]

}