## Texto Descripción generada automáticamenteO(n)

Binary search: O(log n)

Sorted set/map or priority queue: O(log n) per operation

Prime factorization of an integer, or checking primality or compositeness of

an integer naively: O(sqrt(n))

Reading in n items of input: O(n)

Iterating through an array or a list of n elements: O(n)

Sorting: usually O(n log n) for default sorting algorithms

(mergesort, Collections.sort, Arrays.sort)

Java Quicksort Arrays.sort function on primitives: O(n^2)

## Imprimir data structures

#include "debugging.h"

puede usarse para imprimir data structures

## Trie

Efficient prefix queries. When we query words that share prefixes, such as “how many words start with ‘an’?

// Video con explicación https://youtu.be/GEoUPRYguY4

#include <bits/stdc++.h>

using namespace std;

#define endl '\n'

struct TrieNode {

char val;

bool end;

map<char, TrieNode\*> children; // TreeMap, dict

TrieNode(char c) {

val = c;

end = false;

}

// Tiempo O(N) siendo N la longitud del string

void insert(string word) {

auto curr = this;

for (char c : word) {

if (!curr->children.count(c)) { // TreeMap#get

curr->children[c] = new TrieNode(c); // TreeMap#put

}

curr = curr->children[c]; // TreeMap#get

}

curr->end = true;

}

// Tiempo O(N) siendo N la longitud del string

int countPrefix(string word) {

auto curr = this;

int cnt = 0;

for (char c : word) {

curr = curr->children[c];

if (curr->end) cnt += 1;

}

return cnt - 1;

}

};

int main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

// 1. leer nuestra entrada

int t;

cin >> t;

vector<string> directory(t); // ArrayList

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

cin >> directory[i];

}

// 2. construir nuestro arbol de prefijos

auto trie = new TrieNode('.');

for (string num : directory) {

trie->insert(num);

}

// 3. contar los prefijos de cada numero

int count = 0;

for (string num : directory) {

count += trie->countPrefix(num);

}

cout << count << endl; // print

return 0;

}

# Solución que determina el mayor numero que dos lados pueden tener

Para un problema donde una vez usado un número no se puede volver a utilizar, y se tiene que llenar espacios en la izquierda y derecha de la máxima manera.

#include <bits/stdc++.h>

using namespace std;

const int MAXN=1010;

bool dp[MAXN][MAXN];

int main(){

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

int n,k,e;

cin >> n >> k >> e;

int l=e,r=n-k-e; // l de left y r de righ

dp[0][0]=1;

for(int len=1;len<n+1;++len){

if(len!=k){

for(int i=l;i>=0;i--){

for(int j=r;j>=0;j--){

if(len<=i){

dp[i][j]|=dp[i-len][j];

//cout << dp[i][j] << endl;

}

if(len<=j){

dp[i][j]|=dp[i][j-len];

//cout << dp[i][j] << endl;

}

}

//cout << endl;

}

}

}

int ans=k;

for(int i=0;i<l+1;++i){

for(int j=0;j<r+1;j++){

if(dp[i][j]){

ans=max(ans,i+j+k);

}

}

}

cout << n-ans<<"\n";

}

# Recursividad

## Grid Traveler

Encontrar cuantas rutas existen para llegar a una meta. Es por medio de recursividad.

Solo se puede mover la persona hacia abajo y la derecha, por lo que se limitan las columnas y filas.

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

#include <bits/stdc++.h>

using namespace std;

int gridTraveler(int m,int n,vector<vector<int>> &memo){

int key=m+n;

if(memo[m][n]!=-1){

return memo[m][n];

}

if(m==1 && n==1) return 1;

if(m==0 || n==0) return 0;

memo[m][n]=gridTraveler(m-1,n,memo)+gridTraveler(m,n-1,memo); //resta uno a la fila, resta uno a la columna

return memo[m][n];

}

int main(){

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

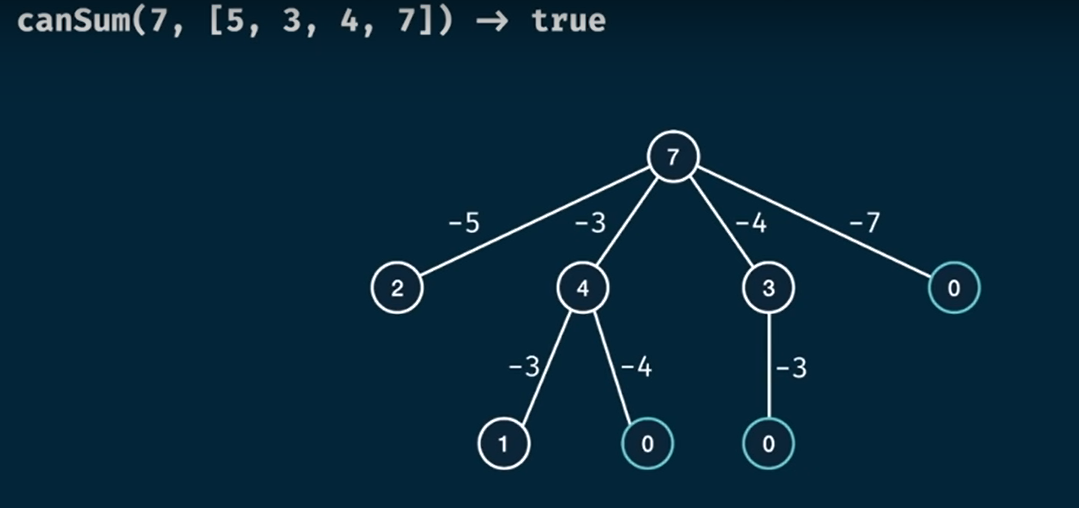
int m=3,n=2;

vector<vector<int>> memo(m+1,vector<int>(n+1,-1));

cout<< gridTraveler(m,n,memo);

}

## CanSum



Determinar si es posible llegar a un target si se tiene valores de un arreglo.

#include <bits/stdc++.h>

using namespace std;

bool canSum(int targetSum, const vector<int> &numbers,unordered\_map<int,bool>&memo){

if(memo.find(targetSum)!=memo.end()) return memo[targetSum]; //diferente de memo.end por como esta escrito la fx de .find()

if(targetSum==0){

memo[targetSum]=true;

return true;

}

if(targetSum<0) return false;

for(auto num:numbers){

int remainder=targetSum-num;

//el remainder no puede ser negativo

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

return true;

}

}

return false; //se regresa false hasta el final, pq

//quiere decir que es imposible que se genere

}

int main(){

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cout.tie(0);

unordered\_map<int,bool> memo;

cout << canSum(7, {2,3},memo) << "\n";

cout << canSum(7, {5,3,4,7},memo) << "\n";

cout << canSum(7, {2,4},memo) << "\n";

cout << canSum(8, {2,3,5},memo) << "\n";

cout << canSum(300, {7,14},memo) << "\n";

return 0;

}

# HowSum

The howSum(targetSum, integers) question is asking us to return a combination of integers that adds up to exactly the targetSum. From the countSum problem, we know that there might more than one combinations, this question is OK with any valid combination.

Esta parte es el núcleo de la recursión. Verifica si es posible alcanzar sum de dos maneras:

Excluyendo el último elemento (llamada a isSubsetSumRec(arr, n - 1, sum)).

Incluyendo el último elemento, reduciendo la suma objetivo (sum - arr[n - 1]) y verificando los n-1 elementos restantes (isSubsetSumRec(arr, n - 1, sum - arr[n - 1])).

//C++ implementation for subset sum

// problem using memoization

#include <bits/stdc++.h>

using namespace std;

// Recursive function to check if a subset

// with the given sum exists

bool isSubsetSumRec(vector<int>& arr, int n, int sum,

vector<vector<int>> &memo) {

// If the sum is zero, we found a subset

if (sum == 0)

return 1;

// If no elements are left

if (n <= 0)

return 0;

// If the value is already

// computed, return it

if (memo[n][sum] != -1)

return memo[n][sum];

// If the last element is greater than

// the sum, ignore it

if (arr[n - 1] > sum)

return memo[n][sum] = isSubsetSumRec(arr, n - 1, sum, memo);

else {

// Include or exclude the last element

return memo[n][sum] = isSubsetSumRec(arr, n - 1, sum, memo) ||

isSubsetSumRec(arr, n - 1, sum - arr[n - 1], memo);

}

}

// Function to initiate the subset sum check

bool isSubsetSum(vector<int>&arr, int sum) {

int n = arr.size();

vector<vector<int>> memo(n + 1, vector<int>(sum + 1, -1));

return isSubsetSumRec(arr, n, sum, memo);

}

int main() {

vector<int>arr = {1, 5, 3, 7, 4};

int sum = 12;

if (isSubsetSum(arr, sum)) {

cout << "True" << endl;

}

else {

cout << "False" << endl;

}

return 0;

}

Con tabulación

//C++ implementation for subset sum

// problem using tabulation

#include <bits/stdc++.h>

using namespace std;

// Function to check if there is a subset of arr[]

// with sum equal to the given sum using tabulation with vectors

bool isSubsetSum(vector<int> &arr, int sum) {

int n = arr.size();

// Create a 2D vector for storing results

// of subproblems

vector<vector<bool>> dp(n + 1, vector<bool>(sum + 1, false));

// If sum is 0, then answer is true (empty subset)

for (int i = 0; i <= n; i++)

dp[i][0] = true;

// Fill the dp table in bottom-up manner

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

for (int j = 1; j <= sum; j++) {

if (j < arr[i - 1]) {

// Exclude the current element

dp[i][j] = dp[i - 1][j];

}

else {

// Include or exclude

dp[i][j] = dp[i - 1][j]

|| dp[i - 1][j - arr[i - 1]];

}

}

}

return dp[n][sum];

}

int main() {

vector<int> arr = {3, 34, 4, 12, 5, 2};

int sum = 9;

if (isSubsetSum(arr, sum))

cout << "True" << endl;

else

cout << "False" << endl;

return 0;

}

# Number of divisors

Tabla

Descripción generada automáticamente

long long numberOfDivisors(long long num) {

long long total = 1;

for (int i = 2; (long long)i \* i <= num; i++) {

if (num % i == 0) {

int e = 0;

do {

e++;

num /= i;

} while (num % i == 0);

total \*= e + 1;

}

}

if (num > 1) {

total \*= 2;

}

return total;

}

# Sum of divisors

Texto

Descripción generada automáticamente con confianza media

long long SumOfDivisors(long long num) {

long long total = 1;

for (int i = 2; (long long)i \* i <= num; i++) {

if (num % i == 0) {

int e = 0;

do {

e++;

num /= i;

} while (num % i == 0);

long long sum = 0, pow = 1;

do {

sum += pow;

pow \*= i;

} while (e-- > 0);

total \*= sum;

}

}

if (num > 1) {

total \*= (1 + num);

}

return total;

}

Euler´s tontient function

Para saber que numeros son coprimos(su gcd==1)

int phi(int n) {

int result = n;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

while (n % i == 0)

n /= i;

result -= result / i;

}

}

if (n > 1)

result -= result / n;

return result;

}

# Linear Diophantine Equation

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Algoritmo

int gcd(int a, int b, int& x, int& y) {

if (b == 0) {

x = 1;

y = 0;

return a;

}

int x1, y1;

int d = gcd(b, a % b, x1, y1);

x = y1;

y = x1 - y1 \* (a / b);

return d;

}

bool find\_any\_solution(int a, int b, int c, int &x0, int &y0, int &g) {

g = gcd(abs(a), abs(b), x0, y0);

if (c % g) {

return false;

}

x0 \*= c / g;

y0 \*= c / g;

if (a < 0) x0 = -x0;

if (b < 0) y0 = -y0;

return true;

}

# Convolución

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

# Fracciones continuas

// Java program to find large factorials using BigInteger

import java.math.BigInteger;

import java.util.Scanner;

public class Example

{

// Returns Factorial of N

static BigInteger factorial(int N)

{

// Initialize result

BigInteger f = new BigInteger("1"); // Or BigInteger.ONE

// Multiply f with 2, 3, ...N

for (int i = 2; i <= N; i++)

f = f.multiply(BigInteger.valueOf(i));

return f;

}

// Driver method

public static void main(String args[]) throws Exception

{

int N = 20;

System.out.println(factorial(N));

}

}

# Linked List

## Determinar si existe un ciclo dentro de una linked list. Floyd alghoritm

#include <iostream>

using namespace std;

class Node {

public:

int value;

Node\* next;

Node(int value) {

this->value = value;

next = nullptr;

}

};

class LinkedList {

private:

Node\* head;

Node\* tail;

int length;

public:

LinkedList(int value) {

Node\* newNode = new Node(value);

head = newNode;

tail = newNode;

length = 1;

}

~LinkedList() {

Node\* temp = head;

while (head) {

head = head->next;

delete temp;

temp = head;

}

}

void printList() {

Node\* temp = head;

if (temp == nullptr) {

cout << "empty";

} else {

while (temp != nullptr) {

cout << temp->value;

temp = temp->next;

if (temp != nullptr) {

cout << " -> ";

}

}

}

cout << endl;

}

Node\* getHead() {

return head;

}

Node\* getTail() {

return tail;

}

int getLength() {

return length;

}

void makeEmpty() {

Node\* temp = head;

while (head) {

head = head->next;

delete temp;

temp = head;

}

tail = nullptr;

length = 0;

}

void append(int value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

tail = newNode;

} else {

tail->next = newNode;

tail = newNode;

}

length++;

}

//algoritmo que sirve solo para saber si existe un loop

Node\* hasLoop(){

Node\* fast=head;

Node\* slow=head;

while(fast && fast->next){

fast=fast->next->next;

slow=slow->next;

if(fast==slow) return fast;

}

return nullptr;

}

Node\* get(int index) {

if (index < 0 || index >= length) return nullptr;

Node\* temp = head;

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

temp = temp->next;

}

return temp;

}

//algoritmo para saber donde está el loop

Node\* startingLoop(){

Node\* p = hasLoop(); //obtenemos donde existió la intersección

Node\* q = head;

while(p!=q){

p=p->next;

q=q->next;

}

return p;

}

};

int main(){

LinkedList list(1);

//list.append(1);

list.append(2);

list.append(3);

list.append(4);

list.append(5);

list.append(6);

list.append(7);

list.append(8);

list.append(9);

list.append(10);

//list.append(11);

Node\* tail = list.getTail();

tail->next = list.get(5); //para que se agarre el de valor 6

//list.printList();

cout << "este es el valor del nodo que inicia el loop " << list.startingLoop()->value << "\n";

}