

Instituto Politécnico Nacional

Escuela Superior de Cómputo

* **UNIDAD DE APRENDIZAJE:** ANÁLISIS DE ALGORITMOS
* **PROFESOR:** EDGARDO ADRIÁN FRANCO MARTÍNEZ
* **ALUMNO:**

BARRERA PÉREZ CARLOS TONATIHU



## **Ejercicio 8: Diseño de soluciones greedy**

## 

* **GRUPO:** 3CM3

# Scarecrow

## Redacción

Taso owns a very long field. He plans to grow different types of crops in the upcoming growing season. The area, however, is full of crows and Taso fears that they might feed on most of the crops. For this reason, he has decided to place some scarecrows at different locations of the field. The field can be modeled as a 1×N grid. Some parts of the field are infertile and that means you cannot grow any crops on them. A scarecrow, when placed on a spot, covers the cell to its immediate left and right along with the cell it is on. Given the description of the field, what is the minimum number of scarecrows that needs to be placed so that all the useful section of the field is covered? Useful section refers to cells where crops can be grown. Input Input starts with an integer T (≤ 100), denoting the number of test cases. Each case starts with a line containing an integer N (0 < N < 100). The next line contains N characters that describe the field. A dot (.) indicates a crop-growing spot and a hash (#) indicates an infertile region.

Output For each case, output the case number first followed by the number of scarecrows that need to be placed.

## Análisis

## Código

|  |
| --- |
| #include <iostream>  **using** **namespace** std;  int greedy\_scarecrow();  int main(int argc, char **const** \*argv[]) {  ios\_base::sync\_with\_stdio(false);  *// Captura del numero de casos*  int T;  cin >> T;  int r;  **for** (int i=1; i <= T; i++){  r = greedy\_scarecrow();  *// Impresion de resultados*  cout << "Case "<< i << ": " << r << "**\n**";  }  }  int greedy\_scarecrow() {  int N;  cin >> N;  char campo[N];  *// Leemos la cadena en un arreglo*  **for** (int i=0; i < N; i++)  cin >> campo[i];  *// Numero de espantapajaros necesarios*  int contador = 0;    *// Si decidimos tomar o no la casilla*    bool tomar = true;    *// Para contar si tomar la casilla o no*    int aux = 3;  *// Recorremos el arreglo*  **for** (int i = 0; i < N; i++) {  *// Si es un punto lo tomamos y esperamos*  **if** (campo[i] == '.' && tomar){  contador++;  tomar = false;  aux--;  } **else** **if** (!tomar)  aux--;  */\**  *\* Si ya recorrimos tres casillas podemos*  *\* poner otro espantapajaros*  *\*/*  **if** (aux == 0) {  aux = 3;  tomar = true;  }  }  **return** contador;  } |

## Captura de pantalla del juez



# Bear and row

## Redacción

Limak is a little polar bear. He is playing a video game and he needs your help.

There is a row with N cells, each either empty or occupied by a soldier, denoted by '0' and '1' respectively. The goal of the game is to move all soldiers to the right (they should occupy some number of rightmost cells).

The only possible command is choosing a soldier and telling him to move to the right as far as possible. Choosing a soldier takes 1 second, and a soldier moves with the speed of a cell per second. The soldier stops immediately if he is in the last cell of the row or the next cell is already occupied. Limak isn't allowed to choose a soldier that can't move at all (the chosen soldier must move at least one cell to the right).

Limak enjoys this game very much and wants to play as long as possible. In particular, he doesn't start a new command while the previously chosen soldier moves. Can you tell him, how many seconds he can play at most?

**Input**

The first line of the input contains an integer **T** denoting the number of test cases. The description of **T** test cases follows.

The only line of each test case contains a string **S** describing the row with N cells. Each character is either '0' or '1', denoting an empty cell or a cell with a soldier respectively.

**Output**

For each test case, output a single line containing one integer — the maximum possible number of seconds Limak will play the game.

**Constraints**

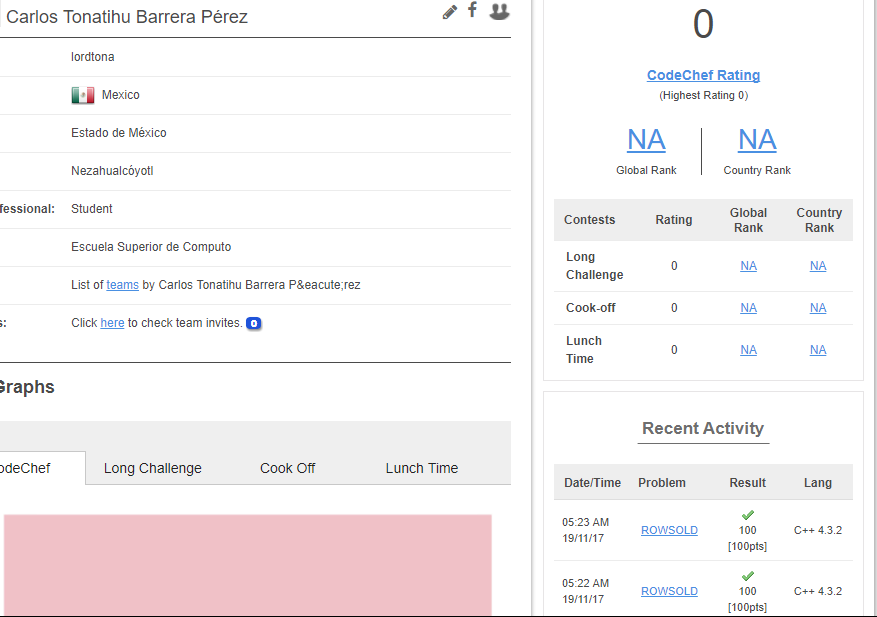
* 1 ≤ **T** ≤ 5
* 1 ≤ N ≤ 105 (N denotes the length of the string **S**)

## Análisis

## Código

|  |
| --- |
| #include <iostream>  #include <string>  **using** **namespace** std;  long greedy\_soldados();  int main(int argc, char **const** \*argv[]) {  ios\_base::sync\_with\_stdio(false);  *// Capturamos el numero de casos*  int T;  cin >> T;  **while** (T--) {  cout << greedy\_soldados() << "**\n**";  }  **return** 0;  }  long greedy\_soldados() {  string soldados;  *// Leemos la cadena*  cin >> soldados;  long tam = soldados.size();  long movimientos = 0;  */\**  *\* Para contar grupos de ceros que maximicen*  *\* el numero de movimientos*  *\*/*  bool grupo = true;  long total = 0;  *// Leemos la cadena del final al inicio*  **for** (long i = tam-1; i > -1; i--) {  */\**  *\* Si vemos un cero contamos un grupo y cuando*  *\* encontremos un uno termina ese grupo*  *\*/*  **if** (soldados[i] == '0' && grupo) {  movimientos++;  grupo = false;  } **else** **if** (soldados[i] == '1' && !grupo) {  grupo = true;  }  */\**  *\* Encontramos un soldado y tenemos que calcular*  *\* sus movimientos*  *\*/*  **if** (soldados[i] == '1'){  total += movimientos + (tam - 1 - i);  tam--;  }  }  **return** total;  } |

## Captura de pantalla del juez



# Permutación hermosa

## Redacción

A Loro le encantan las permutaciones. Una permutación de longitud nn es una secuencia de nn números enteros positivos distintos, ninguno mayor que n.

Asumiremos que el valor de mcd(a,b)mcd(a,b) es el máximo común divisor de aa y bb.

Para Loro, el elemento pipi de la permutación p1,p2,...,pnp1,p2,...,pn es buenobueno si mcd(i,pi)>1mcd(i,pi)>1. Loro considera una permutación hermosahermosa, si tiene exactamente kk elementos buenosbuenos. Desafortunadamente Loro no conoce ninguna permutación hermosahermosa. Ayúdalo a encontrar al menos una de ellas.

# **Entrada**

La linea de entrada contiene dos enteros nn y kk.

# **Salida**

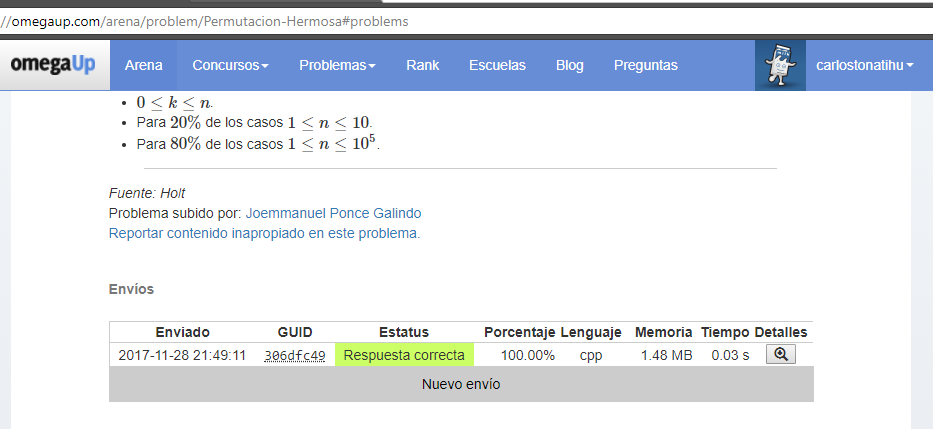
En una sola linea imprime cualquier permutación hermosahermosa ó -1, si no existe ninguna.

## Análisis

## Código

|  |
| --- |
| #include <stdio.h>  int main(int argc, char **const** \*argv[]) {  int N, K;  scanf("%d %d", &N, &K);  **if** (K>=N){  printf("%d", -1);  **return** 0;  }  **if** (K == N-1)  printf("%d ", 1);  **else**  printf("%d ", N);  int i;  **for** (i = 2; i<=K+1; i++)  printf("%d ", i);  **if** (K != N-1){  printf("%d ", 1);  **for** (int j = i; j<N; j++)  printf("%d ", j);  }  **return** 0;  } |

## Captura de pantalla del juez



# Función horrible

## Redacción

Considera una función con NN elementos:

g(x)=f1(x)+f2(x)+f3(x)+..fN(x)g(x)=f1(x)+f2(x)+f3(x)+..fN(x)

Donde fi(x)fi(x) puede ser de la forma AxBAxB (tipo 1) o AeBxAeBx (tipo 2) y se cumple que 1≤A,B≤10181≤A,B≤1018 para cualquiera de los tipos.

Ahora tienes un conjunto de MM xx's enteras positivas de las cuales quieres escoger KK de ellas tales que se maximice el valor de:

Z=∑j=0Kg(xj)Z=∑j=0Kg(xj)

, donde xjxj es la j-ésima x que escogiste.

# Problema

Dada la descripción de la función g(x)g(x) y el conjunto de xx's que puedes escoger, determina cuales tienes que tomar tal que se maximice el valor ZZ.

# **Entrada**

En la primera línea 3 enteros NN , MM , KK indicando el número de elementos de la función , el número de XX's que tienes y el número de xx's que debes tomar.

En las siguientes NN líneas se describe la función gg. Para el i-ésimo elemento de la función, se da un entero para expresar de que tipo es (1 o 2) y los valores AA y BB del elemento fi(x)fi(x).

En la siguiente fila MM enteros que x1x1 , x2x2 , ⋯⋯ , xMxM expresan el conjunto de x con el que cuentas.

# **Salida**

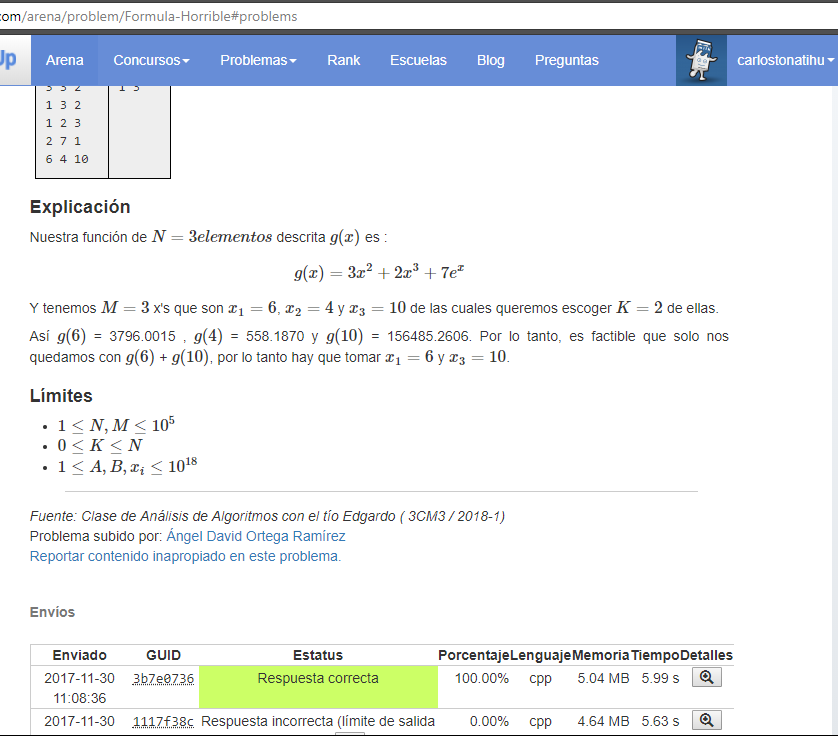
Una línea con KK enteros ordenados p1p1, p2p2 ,⋯⋯ , pkpk indicando que tomaste xp1xp1, xp2xp2, , ⋯⋯ , xpkxpk. Si existe más de una solución, cualquiera se tomará como válida.

## Análisis

## Código

|  |
| --- |
| #include <iostream>  #include <stdio.h>  #include <stdlib.h>  #include <algorithm>  **using** **namespace** std;  **struct** Nodo {  unsigned long long valor;  int indice;  };  bool comparacion(**struct** Nodo a, **struct** Nodo b) {  **return** (a.valor < b.valor);  }  int main(int argc, char **const** \*argv[]) {  int N, M, K;  cin >> N >> M >> K;  **struct** Nodo \*entrada = (**struct** Nodo\*)malloc(M \* **sizeof**(**struct** Nodo));  unsigned long long tipo, A, B;  **for** (int i = 0; i < N; i++)  cin >> tipo >> A >> B;    **for** (int i = 0; i < M; i++){  cin >> entrada[i].valor;  entrada[i].indice = i+1;  }  **if** (K != 0){  int salida[K];  sort(entrada, entrada+M, comparacion);  int j = 0;  **for** (int i = M-K; i < M; i++)  salida[j++] = entrada[i].indice;  sort(salida, salida+K);  **for** (int i = 0; i < K; i++)  cout << salida[i] << " ";  }    **return** 0;  } |

## Captura de pantalla del juez



# Lazy Loading

## Redacción

Wilson works for a moving company. His primary duty is to load household items into a moving truck. Wilson has a bag that he uses to move these items. He puts a bunch of items in the bag, moves them to the truck, and then drops the items off.

Wilson has a bit of a reputation as a lazy worker. Julie is Wilson's supervisor, and she's keen to make sure that he doesn't slack off. She wants Wilson to carry at least 50 pounds of items in his bag every time he goes to the truck.

Luckily for Wilson, his bag is opaque. When he carries a bagful of items, Julie can tell how many items are in the bag (based on the height of the stack in the bag), and she can tell the weight of the top item. She can't, however, tell how much the other items in the bag weigh. She assumes that every item in the bag weighs at least as much as this top item, because surely Wilson, as lazy as he is, would at least not be so dense as to put heavier items on top of lighter ones. Alas, Julie is woefully ignorant of the extent of Wilson's lack of dedication to his duty, and this assumption is frequently incorrect.

Today there are **N** items to be moved, and Wilson, paid by the hour as he is, wants to maximize the number of trips he makes to move all of them to the truck. What is the maximum number of trips Wilson can make without getting berated by Julie?

Note that Julie is not aware of what items are to be moved today, and she is not keeping track of what Wilson has already moved when she examines each bag of items. She simply assumes that each bagful contains a total weight of at least **k** \* **w** where **k** is the number of items in the bag, and **w** is the weight of the top item.

### **Input**

Input begins with an integer **T**, the number of days Wilson "works" at his job. For each day, there is first a line containing the integer **N**. Then there are **N** lines, the **i**th of which contains a single integer, the weight of the **i**th item, **Wi**.

### **Output**

For the **i**th day, print a line containing "Case #**i**: " followed by the maximum number of trips Wilson can take that day.

### **Constraints**

1 ≤ **T** ≤ 500   
1 ≤ **N** ≤ 100   
1 ≤ **Wi** ≤ 100

## Análisis

## Código

|  |
| --- |
| #include <algorithm>  #include <stdio.h>  **using** **namespace** std;  */\**  *\* Funcion que calcula el maximo de viajes por*  *\* cada caso*  *\*/*  int greedy\_viaje();  int main(int argc, char **const** \*argv[]) {  int T;  *// Captura del numero de casos*  scanf("%d", &T);  *// Calculo e impresion de los resultados*  **for** (int i=1; i<=T; i++) {  printf("Case #%d: %d**\n**", i, greedy\_viaje());  }  **return** 0;  }  int greedy\_viaje() {  int N;  *// Numero de viajes que se realizan*  int viajes = 0;  *// Captura del numero de objetos*  scanf("%d", &N);  int objetos[N];  *// Capturamos el peso de los N objetos*  **for** (int i=0; i < N; i++)  scanf("%d", &objetos[i]);  *// Ordenamos los numeros de menor a mayor*  sort(objetos, objetos+N);  *// Final del arreglo*  int final = N-1;  *// Peso de la bolsa*  int peso = 0;  *// Numero de objetos en la bolsa*  int num\_objetos = 1;  **for** (int inicio = -1; inicio < final;) {  *// Calcula el peso de la bolsa*  peso = objetos[final] \* num\_objetos;  **if** (peso >= 50) {  */\**  *\* Reiniciamos nuestros valores*  *\*/*  peso = 0;  num\_objetos = 1;  */\**  *\* Menos uno para tomar el siguiente objeto*  *\* mas pesado que queda por mover*  *\*/*  final -= 1;  */\**  *\* Sumamos un viaje*  *\*/*  viajes += 1;  } **else** {  */\**  *\* Agregamos un objeto a la bolsa y nos*  *\* desplazamos en el arreglo de objetos*  *\* que faltan por llevar*  *\*/*  num\_objetos++;  inicio++;  }  }  *// Regresamos el numero de viajes*  **return** viajes;  } |

## Captura de pantalla del juez

