PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMIA
Primer Examen
(Primer semestre de 2013)

Horario 0581: prof. Andrés Melgar

Duración: 3 horas
Nota:

- No se permite el uso de material de consulta.
- No se ofrecerá asesoría en la parte teórica
- **La presentación, la ortografía y la gramática influirán en la calificación.**

Puntaje total: 20 puntos

Cuestionario:

PARTE TEÓRICA

Responda las siguientes preguntas según los conceptos vistos en clase

Pregunta 1 (1 puntos) El número de comparaciones usado por el algoritmo de búsqueda secuencial en el peor de los casos es n mientras que en el peor de los casos la búsqueda binaria realiza $log_2 n$ comparaciones. Basada en esta información, ¿se puede afirmar que siempre es mejor realizar una búsqueda binaria que una búsqueda secuencial?. **Justifique adecuadamente su respuesta**.

Pregunta 2 (2 puntos) ¿Qué es el principio de optimalidad? ¿En qué situaciones debe ser usado?

Pregunta 3 (2 puntos) ¿En qué se diferencian las estrategias algoritmicas de tipo *top down* y *bottom up*?

PARTE PRÁCTICA

Pregunta 4 (5 puntos) El problema del cambio exacto. Problema adaptado de la URL `http://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=2512`

- *Vendedor: Serán catorce dólares.*
- *Comprador: Aquí hay un billete de veinte.*
- *Vendedor: Lo siento, no tengo cambio.*
- *Comprador: OK, aquí hay uno de diez y uno de cinco. Quédese con el cambio.*

Cuando se viaja a lugares remotos, a menudo es útil llevar dinero en efectivo, en caso se necesite comprar algo de alguien que no acepta tarjetas de crédito o débito. También es útil llevar una variedad de denominaciones, en caso de que el vendedor no tenga cambio. A pesar de tener dinero en varias denominaciones, puede que no consiga la cantidad exacta, y tendrá que pagar un poco más del precio completo. El mismo problema puede surgir incluso en zonas urbanas, por ejemplo, con máquinas expendedora que

no devuelven cambio.

Por supuesto, usted desea minimizar la cantidad que paga (aunque usted debe pagar por lo menos tanto como el valor del elemento). Por otra parte, a pesar de pagar el importe mínimo, usted desea minimizar el número de monedas o billetes que utiliza.

Especificación de la Entrada: La primera línea de la entrada contiene un entero que especifica el número de casos a seguir. Cada caso de prueba comienza con una línea que contiene un entero, el precio en centavos de el artículo que desea comprar. El precio no superará los 10 000 centavos de dólar (es decir, $ 100). La siguiente línea contiene un entero n, el número de billetes y monedas que tiene. El número n es a lo sumo 100. Las siguientes n líneas contienen cada uno un entero, el valor en centavos de cada billete o moneda que usted tiene. Tenga en cuenta que las denominaciones pueden ser cualquier número de centavos, que no se limitan a los valores de las monedas y billetes que normalmente utilizamos en Perú. Sin embargo, ningún billete o moneda tendrá un valor superior a 10 000 centavos ($ 100). El valor total de los billetes y monedas será siempre igual o mayor que el precio del artículo.

Ejemplo de la Entrada:

```
1
1400
3
500
1000
2000
```

Especificación de la Salida: Para cada caso de prueba, la salida de una sola línea que contiene dos enteros: el importe total pagado (en centavos de dólar), y el número total de monedas y billetes utilizados.

Ejemplo de la Salida:

```
1500 2
```

Se le pide que elabora un programa en ANSI C que resuelva el problema antes mencionado. ***Una restricción del problema es que este debe ser ejecutado rápidamente (i.e. 1 segundos) para una lista grande de casos.***

```c
#include <stdio.h>
#include <stdlib.h>
#define PRECIO_MAXIMO 10000
#define PAGO_MAXIMO 20001
#define NUM_MAX_MONEDAS 101
#define max(a,b) ((a)>(b)?(a):(b))
#define min(a,b) ((a)<(b)?(a):(b))

void leer_caso(int *precio, int *num_monedas, int *denominaciones);
void resolver_caso(int *precio, int *num_monedas, int *denominaciones, int *monedas);
void inicializa_monedas(int *Monedas);

int main(int argc, char** argv) {
    int num_casos, precio, num_monedas, denominaciones[NUM_MAX_MONEDAS], monedas[
        PAGO_MAXIMO];
    int i;

    scanf("%d", &num_casos);
    for (i = 1; i <= num_casos; ++i) {
        inicializa_monedas(monedas);
        leer_caso(&precio, &num_monedas, denominaciones);
        resolver_caso(&precio, &num_monedas, denominaciones, monedas);
```

```
22          mostrar_resultado(precio, monedas);
23      }
24
25      return (EXIT_SUCCESS);
26  }
27
28  void leer_caso(int *precio, int *num_monedas, int *denominaciones) {
29      int i;
30      scanf("%d", precio);
31      scanf("%d", num_monedas);
32      for (i = 0; i<*num_monedas; i++)
33          scanf("%d", &denominaciones[i]);
34  }
35
36  void resolver_caso(int *precio, int *num_monedas, int *denominaciones, int *monedas) {
37      int valor_anterior, maximo_valor, indice_moneda;
38      /*Monedas sera un arreglo en donde el indice (i) me indique el valor
39       * que se consigue con la menor cantidad de monedas (M[i]) monedas */
40      monedas[0] = 0; /*para que funcione para el primer caso*/
41      maximo_valor = 0; /* sin ninguna moneda*/
42
43      /*se recorren las k denominaciones*/
44      for (indice_moneda = 0; indice_moneda<*num_monedas; indice_moneda++) {
45
46          /*con la denominacion actual k, buscamos la mejor combinacion:
47           precio cercano al real y con la menor cantidad de monedas*/
48          for (valor_anterior = min(maximo_valor, PRECIO_MAXIMO); valor_anterior >= 0;
                valor_anterior--) {
49
50              /*valor que se consigue con la denominacion actual */
51              int nuevo_valor = valor_anterior + denominaciones[indice_moneda];
52              if (monedas[valor_anterior] + 1 < monedas[nuevo_valor]) {
53                  monedas[nuevo_valor] = monedas[valor_anterior] + 1;
54                  maximo_valor = max(maximo_valor, nuevo_valor);
55              }
56          }
57      }
58  }
59
60  /*Monedas sera un arreglo en donde el indice (i) indicara el valor
61   * que se consigue con (M[i]) monedas. Inicialmente este arreglo
62   * estara inicializado con un valor superior a la cantidad maxima de
63   * moneda sa conseguir */
64  void inicializa_monedas(int *Monedas) {
65      int i;
66      for (i = 0; i < PAGO_MAXIMO; ++i)
67          Monedas[i] = NUM_MAX_MONEDAS;
68  }
69
70  void mostrar_resultado(int precio, int *monedas) {
71      int indice = precio;
72      /*se busca la primer entrada en el arreglo que tenga un valor de monedas
73       llenado por el algoritmo de programacion dinamica, buscamos a partir
74       del precio*/
75      while (1) {
76          if (monedas[indice] < NUM_MAX_MONEDAS) break;
77          indice++;
78      }
79      printf("%d %d\n", indice, monedas[indice]);
80  }
```

Pregunta 5 (5 puntos) El problema de los productos de dígitos. Problema adaptado de la URL `http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=934`

Dado un número entero N no negativo, se pide encontrar el mínimo número natural Q. El número Q es todo aquel número en el cual el producto de sus dígitos es igual a N.

Especificación de la Entrada: La primera línea de entrada contiene un número entero positivo, que es el número de conjuntos de datos. Cada línea subsiguiente contiene un conjunto de datos que consta de un número entero no negativo N ($0 \leq N \leq 10^9$).

Especificación de la Salida: Para cada conjunto de datos, escriba una línea que contiene el número Q correspondiente o -1 si es que el número Q no existe.

Ejemplo de la Entrada:

```
3
1
10
123456789
```

Ejemplo de la Salida:

```
1
25
-1
```

Se le pide que elabora un programa en ANSI C que resuelva el problema antes mencionado. ***Una restricción del problema es que este debe ser ejecutado rápidamente (i.e. 3 segundos) para una lista grande de casos.***

```c
#include <stdio.h>
#include <stdlib.h>

void solucionar_caso(int n);

int main(int argc, char** argv) {
    int numero_casos, i, n;
    scanf("%d", &numero_casos);
    for (i = 0; i < numero_casos; i++) {
        scanf("%d", &n);
        if (n < 10)
            printf("%d\n", n);
        else
            solucionar_caso(n);
    }
    return (EXIT_SUCCESS);
}

void solucionar_caso(int n) {
    int divisores[] = {9, 8, 7, 6, 5, 4, 3, 2};
    char numeroQ[10];
    int indiceQ = 0, i;

    for (i = 0; i < 8; i++)
        while (n % divisores[i] == 0) {
            n = n / divisores[i];
            numeroQ[indiceQ++] = divisores[i] + '0';
        }

    if (n != 1)
        printf("-1\n");
    else {
        indiceQ--;
```

```
34          while (indiceQ >= 0) {
35              printf("%c", numeroQ[indiceQ]);
36              indiceQ--;
37          }
38          printf("\n");
39      }
40
41  }
```

Pregunta 6 (5 puntos) El algoritmo Comb Sort. Problema adaptado de la URL `http://es.wikipedia.org/wiki/Comb_sort`

En ciencias de la computación, el comb sort es un algoritmo de ordenamiento relativamente simple diseñado por Wlodzimierz Dobosiewicz en 1980. Posteriormente fue redescubierto y popularizado por Stephen Lacey y Richard Box en un artículo publicado por la revista Byte en abril de 1991.

A continuación se presenta una implementación del algoritmo en ANSI C:

```
1  void combsort(int *arr, int size) {
2      float shrink_factor = 1.247330950103979;
3      int gap = size, swapped = 1, swap, i;
4
5      while ((gap > 1) || swapped) {
6          if (gap > 1)
7             gap = gap / shrink_factor;
8
9          swapped = 0;
10         i = 0;
11
12         while ((gap + i) < size) {
13             if (arr[i] - arr[i + gap] > 0) {
14                 swap = arr[i];
15                 arr[i] = arr[i + gap];
16                 arr[i + gap] = swap;
17                 swapped = 1;
18             }
19             ++i;
20         }
21     }
22 }
```

Se le pide que:

a) Indique cuál es la idea del algoritmo.

b) Compare el algoritmo Comb Sort con el algoritmo de ordenación Bubble Sort.

c) Compare el algoritmo Comb Sort con el algoritmo de ordenación Quick Sort.

Profesores del curso: Andrés Melgar

Pando, 18 de mayo de 2012