Algoritmos y Estructuras de Datos (VJ1215) - Universitat Jaume I
Evaluación continua - 2022/2023

30 de noviembre de 2022

Nombre:

La duración de esta prueba es de 80 minutos. La prueba es individual. No puedes consultar libros, apuntes ni dispositivos electrónicos. Al finalizar entrega tu solución junto con el enunciado (no es necesario que entregues todas las hojas). Pon tu nombre en todo lo que entregues.

Ejercicio 1 — 4,5 puntos

Tenemos que repartir $n \geq 1$ jugadores entre $q \geq 1$ equipos. Cada jugador puede ser asignado a un solo equipo. No puede quedar ningún jugador sin asignar. Puede haber equipos a los que no les demos ningún jugador.

Nos dan una matriz *beneficio* de talla $(n+1) \times q$ en la que *beneficio*$[j][e]$ es el beneficio que obtenemos si asignamos exactamente una cantidad de jugadores j al equipo e, para $j = 0, \ldots, n$ y para $e = 0, \ldots, q-1$.

Necesitamos una función que calcule cuál es la máxima suma de beneficios que podemos obtener, repartiendo los jugadores de forma óptima:

```
float maximoBeneficio(const vector<vector<float> > & beneficio)
```

Por ejemplo, con $n = 5$ jugadores a repartir, $q = 3$ equipos y esta matriz *beneficio*:

cantidad de jugadores \ **equipo**	0	1	2
0	0	0	0
1	30	10	20
2	50	60	20
3	60	70	40
4	65	80	80
5	65	85	100

el resultado sería $50 + 60 + 20 = 130$, correspondiente a asignar dos jugadores al equipo 0, dos al equipo 1 y uno al equipo 2:

cantidad de jugadores \ **equipo**	0	1	2
0	0	0	0
1	30	10	**20**
2	**50**	**60**	20
3	60	70	40
4	65	80	80
5	65	85	100

Empleando Programación Dinámica, diseña e implementa un algoritmo eficiente para resolver el problema a) de forma recursiva y b) de forma no recursiva. Escribe tus soluciones empleando el lenguaje C++.

Indica el coste temporal y espacial de cada una de tus dos soluciones en el peor caso, en función de n y q.

	Recursiva	**No recursiva**
Coste temporal	$O($ $)$	$O($ $)$
Coste espacial	$O($ $)$	$O($ $)$

Ejercicio 2 — 1,5 puntos

Teorema Maestro: La solución de la ecuación $T(N) = aT(N/b) + \theta(N^k \log^p N)$, con $a \geq 1$, $b > 1$ y $p \geq 0$, es

$$T(N) = \begin{cases} O(N^{\log_b a}) & \text{si } a > b^k \\ O(N^k \log^{p+1} N) & \text{si } a = b^k \\ O(N^k \log^p N) & \text{si } a < b^k \end{cases}$$

El siguiente algoritmo recursivo encuentra el máximo en un vector de talla $n \geq 1$. Utilizando el Teorema Maestro, analiza su coste temporal en el peor caso en función de n. Muestra paso a paso cómo aplicas el teorema.

```
int maximo(const vector<int> & v, int inicio, int fin) {
// Devuelve el maximo valor encontrado entre las posiciones inicio y fin, ambas inclusive
   int resultado;
   if (fin - inicio <= 3) {
      resultado = v[inicio];
      for (int i = inicio + 1; i <= fin; i++)
         if (v[i] > resultado)
            resultado = v[i];
   } else {
      int medio = (inicio + fin) / 2;
      int cuartoIzquierda = (inicio + medio) / 2;
      if (maximo(v, inicio, cuartoIzquierda) > maximo(v, cuartoIzquierda + 1, medio))
         resultado = maximo(v, inicio, cuartoIzquierda);
      else
         resultado = maximo(v, cuartoIzquierda + 1, medio);
      for (int i = medio + 1; i <= fin; i++)
         if (v[i] > resultado)
            resultado = v[i];
   }
   return resultado;
}

int maximo(const vector<int> & v) {
   return maximo(v, 0, v.size() - 1);
}
```

Ejercicio 3 — 1,5 puntos

En un vector, a partir de la posición 1 inclusive, tenemos un montículo binario de mínimos (*min-heap*) de talla n.

a) ¿Cómo implementarías una operación que reciba como argumento la posición i de un elemento en el vector, siendo $1 \leq i \leq n$, y elimine ese elemento del montículo? Explica claramente tu solución. No es necesario que la implementes, pero puedes hacerlo si lo prefieres.

b) ¿Cuál sería el coste temporal de tu solución en el peor caso, en función de n? No es necesario que lo justifiques.

Nombre:

Ejercicio 4 — 2,5 puntos

Un programador está implementando el algoritmo de Huffman en el constructor de la siguiente clase `Huffmann`, tal como se ha pedido en las prácticas de la asignatura:

```
class Huffman{
   struct Nodo{
      char   caracter;
      float  frecuencia;
      Nodo * izquierdo;
      Nodo * derecho;
      Nodo * padre;
      char   bit;
      Nodo(char, float);
   };
   Nodo * raiz;
   map<char, Nodo *> hojas;
 public:
   Huffman(const vector< pair<char, float> > &);
   string codificar(const string &) const;
   string decodificar(const string &) const;
};
```

Ayúdale, respetando lo que ya ha hecho y completando lo que le falta en la implementación que aparece a continuación. Escribe tu solución en C++, indicando qué pondrías en los puntos suspensivos dentro del bucle `while` y qué pondrías a continuación de ese bucle, si pondrías algo. No puedes modificar nada más ni añadir código en otros sitios.

```
Huffman::Nodo::Nodo(char c, float f)
  : caracter{c}, frecuencia{f}, izquierdo{nullptr}, derecho{nullptr}, padre{nullptr} {
}
Huffman::Huffman(const vector< pair<char, float> > & frecuencias) {

   if (frecuencias.size() < 2)
      throw string("Necesitamos al menos dos caracteres con sus frecuencias.");

   class ComparadorNodos {
   public:
      bool operator() (Nodo * n1, Nodo * n2) const {
         return n1->frecuencia > n2->frecuencia;
      }
   };
   priority_queue<Nodo *, vector<Nodo *>, ComparadorNodos> colaPrioridad;

   for (auto dato : frecuencias) {
      Nodo * hoja = new Nodo(dato.first, dato.second);
      colaPrioridad.push(hoja);
      hojas[dato.first] = hoja;
   }

   while (colaPrioridad.size() > 1) {
      Nodo * aux = colaPrioridad.top();
      colaPrioridad.pop();
      ...
   }
   ...
}
```