TADCOCINA.pdf

Anónimo

Análisis de Algoritmos y Estructuras de Datos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

TAD COCINA

// Una empresa de muebles de cocina necesita un TAD para representar el conjunto de muebles
//colocados en la pared de una cocina. Una cocina se crea con una longitud positiva
// y un mueble colocado en la pared se identifica por el par formado por su posicion
// y su anchura. La posicion es la distancia desde su lateral izquierdo al extremo derecho de la pared
// El tad debe soportar las siguiente operaciones
// -crear una cocina vacia con una longitud dada
//- determinar si un mueble de una cierta anchura puede colocarse en una posicion dada
// - añadir un mueble de una determinada longitud a una posicion dada
// - devolver el mueble inésimo de la cocina empezando a contar por la izquierda
// - eliminar el mueble enesimo de la cocina si existe
// - mover el mueble inesimo de la cocina si existe hacia la izquierda hasta que se junte con el mueble i-1 esimo o el extremo izquierdo
//de la pared
//- destruir la cocina

// Especificacion 30%
// diseñar estructura de datos 10%
// implementar operaciones 60%

//Definicion : una cocina es una secuencia de muebles colocados en la pared en una posicion concreta, teniendo en cuenta su anchura.
// La posicion es la distancia desde el lateral izquiedo al extremo derecho de la pared , y la anchura es el ancho del mueble

//Especificaciones:
//explicit Cocina (double longitudCocina); OK
//postCondicion: crea una cocina vacia con una longitud determinada

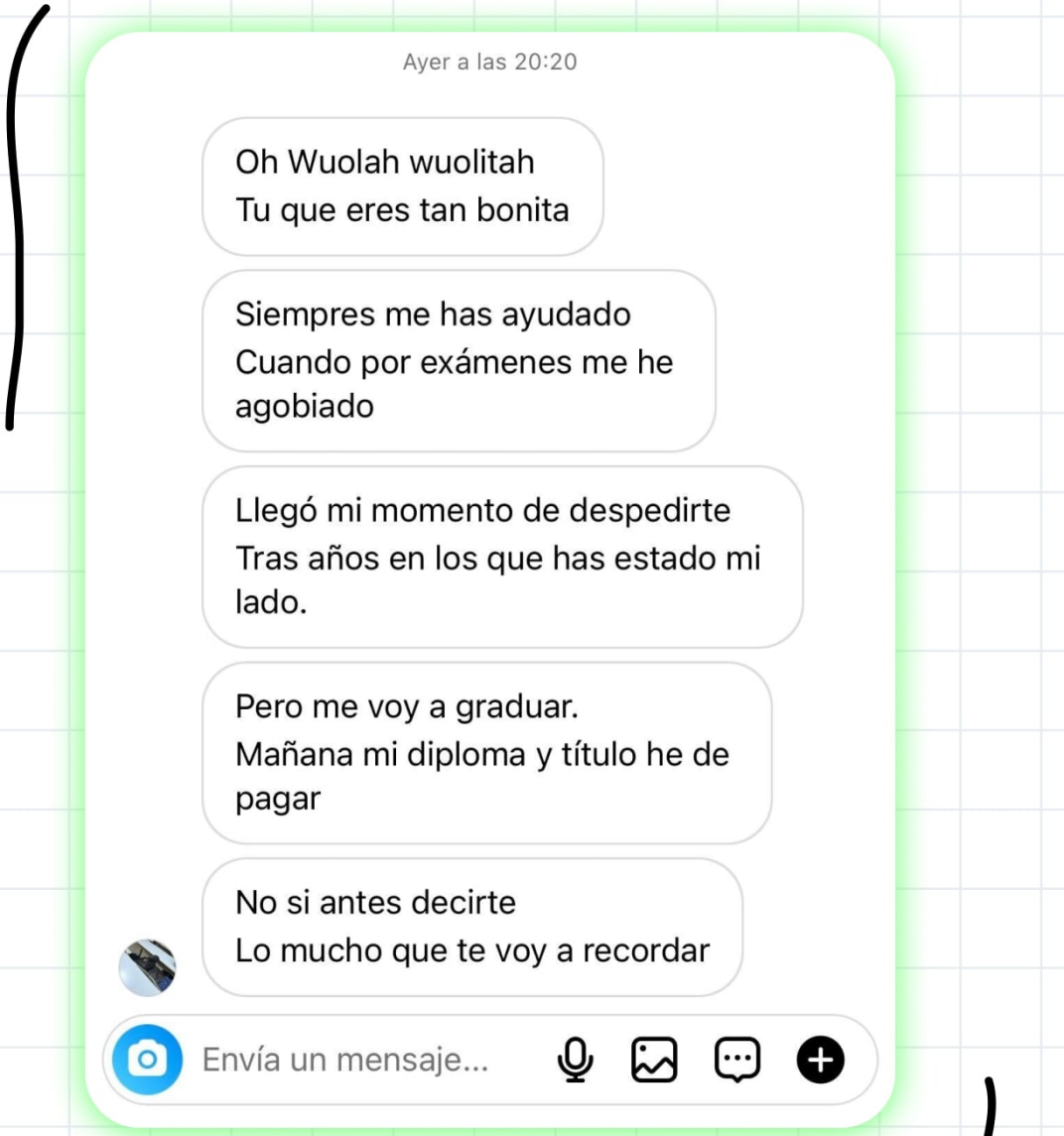
//bool comprobarSiPuedeColocarMueble (double anchura , double posicionAcolocar) const ; //OK
//preCondicion: la cocina no puede estar vacia
//postCondicion: devuelve true, si el mueble se puede colocar en la posicion dada, en caso contrario, false

//void insertarMueble(mueble m);
// preCondicion: no puede haber otro mueble desde la posicion hasta lo q mide el mueble
//postCondicion: añade a la cocina el mueble m , en una posicion determinada

```
//const mueble muebleEnesimo ( int enesimo ) const ;
//postCondicion: devuelve el mueble que se incluyó... Si no existe... ?

//void eliminarMuebleEnesimo( int enesimo);
//preCondicion: la cocina no puede estar vacia
//postCondicion: elimina el mueble enesimo de la cocina, si no existe el enesimo no hace
nada

//void moverMuebleEnesimo ( int enesimo);
//preCondicion: la cocina no puede estar vacia
//postCondicion: mueve la posicion respecto a la cocina del mueble si existe hacia la
izquierda hasta que se junte con el mueble i-1 esimo o el extremo izquierdo
//de la pared

//~Cocina();
//postCondicion: destruye la cocina

typedef struct{
   double posicionEnPared;
   double anchura;
}mueble;

class Cocina{
public:
   explicit Cocina (double longitudCocina): longitudPared(longitudCocina),
longitudParedDisponible(longitudCocina){}; // OK
   bool comprobarSiPuedeColocarMueble(double anchuraMueble, double
posicionAcolocar) const;
   void insertarMueble(const mueble& m);
   const mueble& muebleEnesimo (int n) const;
   void eliminarMuebleEnesimo( int enesimo);
   void moverMuebleEnesimo ( int enesimo);
   ~Cocina();
private:
   lista<mueble> cocina_();
   const double longitudPared;
   double longitudParedDisponible;
};

bool Cocina::comprobarSiPuedeColocarMueble(double anchura, double posicionAcolocar)
const {

   bool haySitio = false;
```

Que no te escriban poemas de amor cuando terminen la carrera 🚩🚩🚩🚩🚩🚩

(a nosotros por suerte nos pasa)



WUOLAH

```
   lista<mueble>::posicion m;

   if (longitudParedDisponible >= anchura){
      haySitio = true;
      m = cocina_.primera();

      while(m != cocina_.fin() && haySitio){
         if(!((posicionEnPared < posicionAcolocar && posicionEnPared+anchura <
posicionAcolocar) || (posicionEnPared>posicionAcolocar &&
posicionAcolocar+anchura>posicionEnPared)))
            haySitio = false;

         m = cocina_.siguiente(m);
      }
   }

   return haySitio;
}

void Cocina::insertarMueble(const mueble& m){
   // insertar ordenando la lista
   lista<mueble>::posicion p ;
   for( p = cocina_.primera(); p != cocina_.fin(); p= cocina.siguiente(p)){
      if (cocina.elemento(p).posicionEnPared < m.posicionEnPared &&
      m.posicionEnPared < cocina.elemento(cocina.siguiente(p)).posicionEnPared){
         cocina_.insertar(m, p);
      }
   }
   longitudParedDisponible = longitudPared - m.anchura;
}

const mueble& Cocina::muebleEnesimo (int n) const{
   // con lista ordenada
   return cocina_.elemento(n);
}

void Cocina::eliminarMuebleEnesimo(int enesimo){
   longitudParedDisponible += cocina_.elemento(enesimo).anchura;
   cocina_.eliminar(enesimo);
}

void Cocina::moverMuebleEnesimo(int enesimo){ // en listas empieza en 1 o cero?

   if(enesimo == 1){
```

```
      cocina_.elemento(enesimo).posicionEnPared = 0;
   }else if((cocina_.elemento(enesimo-1).posicionEnPared + cocina_.elemento(enesimo-
1).anchuraMueble) != cocina_.elemento(enesimo).posicionEnPared){
      cocina_.elemento(enesimo).posicionEnPared = cocina_.elemento(enesimo-
1).posicionEnPared + cocina_.elemento(enesimo-1).anchuraMueble);
   }
}

template <typename T >
Cocina::~Cocina(){
   ~cocina_();
}
```