CIT 2000 Estructuras de Datos Solemne 1 PAUTA

Tiempo: 100 minutos
Profesores: Juan Giadach
Roberto Konow
Leandro LLanza

Pregunta 1

Existe una técnica de compresión que consiste en reemplazar una secuencia de caracteres iguales, por uno solo y la cantidad de veces que aparece en esa secuencia. Por ejemplo, si la secuencia original es `GGOOOOOOOOOLLL`, la comprimida es `G2O9L3`.

Asumiendo que se tiene una cola en la que hay caracteres (sólo letras, no números), se desea construir una `ListaComprimida` con la secuencia comprimida de caracteres de dicha cola, se pide:

a) Definir la estructura de datos `Nodo`, útil para este problema. (5 puntos)

```
struct Nodo
{
   char caracter;
   int repeticiones;
   Nodo *next;
   Nodo(char c = '\0') : caracter(c), repeticiones(1), next(NULL) {}
};
```

b) Definir el TDA `ListaComprimida` (con los atributos y métodos necesarios). (5 puntos)

```
class ListaComprimida
{
   private:
      Nodo *cabeza;
      Nodo *cola;

   public:
      ListaComprimida();
      bool vacia() { return (cabeza == NULL || cola == NULL); }
      void insertarCaracter(char);
      void imprimeComprimida();

      void imprimeOriginal();
};

ListaComprimida::ListaComprimida() : cabeza(NULL), cola(NULL) {}

void ListaComprimida::insertarCaracter(char c)
{
   if (this->vacia())
   {
      this->cabeza = this->cola = new Nodo(c);
   }
   else
   {
      if (this->cola->caracter == c)
         this->cola->repeticiones++;
      else
      {
         this->cola->next = new Nodo(c);
         this->cola = this->cola->next;
      }
   }
}
```

```
void ListaComprimida::imprimeComprimida()
{
   Nodo *aux = this->cabeza;
   while(aux != NULL)
   {
      cout << aux->caracter << aux->repeticiones;
      aux = aux->next;
   }
   cout << endl;
}
```

c) Programar la función principal de dicho proceso, asumiendo la existencia de la cola con los caracteres ya ingresados y que los métodos de `Nodo` y `ListaComprimida` están construidos. (10 puntos)

```
void principal(Cola& l)
{
  ListaComprimida *lc = new ListaComprimida();
  while(!l.vacia())
  {
    char chr;
    lc->insertarCaracter((chr = l.dequeue()));
  }
  cout << "Secuencia comprimida: "; lc->imprimeComprimida();
  cout << "Secuencia original: "; lc->imprimeOriginal();
}

int main(int argc, char *argv[])
{
  if (argc != 2) return -1;
  Cola q;
  for (int i = 0; i < strlen(argv[1]); i++)
  {
    char chr = argv[1][i];
    // cout << "Insertado caracter " << chr << endl;
    q.enqueue(argv[1][i]);
  }
  principal(q);
}
```

d) Implementar el método (que es parte del TDA `ListaComprimida`), que imprime la secuencia original. (10 puntos)

```
void ListaComprimida::imprimeOriginal()
{
   Nodo *aux = this->cabeza;
   while(aux != NULL)
   {
      for(int i = 0; i < aux->repeticiones; i++)
         cout << aux->caracter;
      aux = aux->next;
   }
   cout << endl;
}
```

Pregunta 2

La empresa de trenes metropolitanos de Santiago necesita hacer un estudio en su línea 1, específicamente, conocer la cantidad de pasajeros que ingresan durante todo un día a cada una de sus estaciones.
Para ello necesita un TDA que permita manejar esta información y que al final del día entregue un informe con el nombre de la estación y la cantidad de pasajeros que ingresó, sin importar el orden en que sean informadas.
Cada cierto tiempo, el sistema general de torniquetes necesita llamar a un método que reciba el nombre de la estación y el número de pasajeros que ingresó en el último periodo.

Defina e implemente el TDA que responde a este requerimiento. (25 puntos)

```
struct nodoe
{
   int conteo;
   char nombre[50];
   nodoe *next;
   nodoe(char *nomb) :  conteo(0), next(NULL)
   {
      strcpy(nombre, nomb);
   }
};

class RecopiladorMetro
{
   private:
      nodoe *head;
      nodoe* existe(char *);

   public:
      RecopiladorMetro() : head(NULL) {}
      void recibeInfoTorniquete(char *, int);
      void informeEstacion(char *);
      void informeCompleto();
};

nodoe* RecopiladorMetro::existe(char *estacion)
{
   nodoe *aux = this->head;
   while(aux != NULL)
   {
      if (strcmp(aux->nombre, estacion) == 0)
         return aux;
      aux = aux->next;
   }
   return NULL;
}

void RecopiladorMetro::recibeInfoTorniquete(char* estacion, int conteo_parcial)
// Como un insertar
{
   if (this->head == NULL)
   {
      this->head = new nodoe(estacion);
      this->head->conteo = conteo_parcial;
   }
   else
   {
      nodoe *match = this->existe(estacion);
      if (match != NULL)
```

```
      {
         match->conteo += conteo_parcial;
      }
      else
      {
         nodoe *n = new nodoe(estacion);
         n->conteo = conteo_parcial;
         n->next = this->head;
         this->head = n;
      }
   }
}

void RecopiladorMetro::informeEstacion(char *estacion)
{
   nodoe *info = this->existe(estacion);
   if (info == NULL)
   {
      cout << "La estacion especificada no existe en el registro" << endl;
   }
   else
   {
      cout << "Informe estacion " << info->nombre << endl;
      cout << "Pasajeros totales del dia: " << info->conteo << endl;
   }
}
void RecopiladorMetro::informeCompleto()
{
   nodoe *aux = this->head;
   while(aux != NULL)
   {
      this->informeEstacion(aux->nombre);   //   Ineficiente   en   complejidad
computacional, pero reutiliza cÃ³digo
      aux = aux->next;
   }
}
```

Pregunta 3

Usted cuenta única y exclusivamente con el TDA `Cola` y necesita saber la cantidad de elementos que contiene una cola. Sin saber como está implementada, desarrolle una función que resuelva su problema. (5 puntos)

```
int largo(Cola& c)
{
   Cola aux;
   int cont = 0;
   while(!c.empty())
   {
      aux.enqueue(c.dequeue());
      cont++;
   }
   while(!aux.empty())
      c.enqueue(aux.dequeue());
   return cont;
}
```

Usted cuenta única y exclusivamente con el TDA `Pila` y necesita saber la cantidad de elementos que contiene una pila. Sin saber como está implementada, desarrolle una función que resuelva su problema. (bonus 5 puntos)

```
int largo(Pila& p)
{
   Pila aux;
   int cont = 0;
   while(!p.empty())
   {
      aux.push(p.pop());
      cont++;
   }
   while(!aux.empty())
      p.push(aux.pop());

   return cont;
}
```