Estructura de Datos y Algoritmos

Implementación de tipos de datos recurrentes lineales

1. El TAD Cola

Una cola es un caso particular de lista en la cual las inserciones tienen lugar por un extremo (el final de la cola) y los borrados tienen lugar por el otro de los extremos (el principio o frente de la cola).

Otro nombre para una cola es FIFO que significa first-in-first-out, o sea que el primero insertado es el primero que se quita.

El TAD cola incluye las siguientes operaciones:

1. creo_vacia(s): crea una cola vacia.
2. es_vacia(s): retorna true si la cola es vacia, en otro caso retorna false.
3. primero(s): retorna la información del elemento que esta en el frente de la cola.
4. quitar(s): quita el primer elemento de la cola (elemento del frente).
5. encolar(s,x): agrega el elemento x en el final de la cola.
6. imprimo(s) : imprime los elementos de la cola del frente al fondo.

Podemos implementar las operaciones utilizando las operaciones de listas encadenadas.

1. creo_vacia(s): es la misma operación.
2. es_vacia(s): es la misma operación. otro caso retorna false.
3. primero(s): es la misma operación.
4. quitar(s): es siguiente con posición 1.
5. encolar(s,x): es insertar en la posición largo de la lista + 1.
6. imprimo(s) : es la misma operación.

Veremos una sola representación: con **listas encadenadas**.

La representación con arreglos es ineficiente pues quitar el primero implica correr todos los elementos del array una posición. Calcularemos el tiempo de las operaciones suponiendo la cola tiene largo n.

1.1. Representación de colas con listas encadenadas

La representación de la información es:

```
struct nodo_cola {
        int x;
        nodo_cola *sig;};

typedef nodo_cola *cola;

struct cabezal{
        cola frente;
        cola fondo;
};
```

El cabezal tiene punteros al primer y al ultimo elemento insertados. De esta forma la inserción y el borrado tienen ambos O(1). Veamos la implementación de las operaciones:

```
cabezal creo_vacia()
{         cabezal c;
          c.frente= NULL;
          return c;}
```

es O(1) y $\Omega(1)$.

```
bool es_vacia(cabezal l)
{
      return (l.frente==NULL); }
```

es O(1) y $\Omega(1)$.

```
int primero (cabezal l)
{   return (l.frente)->x; }
```

es O(1) y $\Omega(1)$.

```
void quitar (cabezal &l)
{     cola q;
      q=l.frente;
      if (l.frente == l.fondo)
        l.frente = NULL;
        l.fondo = NULL;
      else l.frente = (l.frente) ->sig;
      delete q;
}
```

es O(1) y $\Omega(1)$.

```
void encolar (cabezal &l,int a)
```

```
{     cola q;

      q = new nodo_cola;
      q->x=a;
      q->sig=NULL;
      if (es_vacia(l)) {l.frente=q;l.fondo=q;}
      else {l.fondo->sig=q; l.fondo=q;}
}
```

es O(1) y $\Omega(1)$.

```
void imprimo(cabezal l)
{     cola q;
      if (es_vacia(l)) cout << "Cola vacia\n";
      else
           { cout << "\n"<< "Cola :"<< "\n";
             q=l.frente;
             while (q! =NULL)
             {
               cout << q->x << " ";
               q=q->sig;
             }
             cout << "\n";}
}
```

es O(n) y $\Omega(n)$.