Pràctica 2

Estructures encadenades: pila, cua i llista

Niub – Nom
16799506 - Rubén Ballester Bautista
16856733 – Oriol Rabasseda Alcaide
Grup de Pràctiques F – Parella 1
Professora:
Maria Salomó

Exercici 1:

Costos Computacionals TAD Linked Queue:

- Constructor: O(1)
- Destructor: O(1n
- size(): O(1)
- empty() : O(1)
- first() : O(1)
- show(): O(n) Depén de la mida de la cua
- enqueue(): O(1)
- dequeue(): O(1)

Observacions dels TAD:

Hem implementat el TAD LinkedQueue a partir de un template Node prèviament declarat. Com a observació, existeix un problema al eliminar els objectes residus d'un node, ja que si ens passen un element de tipus «built-in» aleshores no es poden destruir i tenim un problema a l'hora de tractar-los. Seria convenient tindre objectes per a cada tipus predefinit per C++ de manera que el «delete» no tingui que discriminar el tipus que se li passa.

Al TAD LinkedQueue no hem implementat un node fantasma per evitar crear un node innecessari sense informació. Hem optat per fer una petita comprovació de si està buit o no a l'hora de fer un enqueue i si és buit inicialitzar els punters de «front» i de «rear».

Com el TAD LinkedQueue és una implementació d'un TAD cua genèric de ints en aquest cas, hem tingut en compte excepcions com pila buida a l'hora de fer el dequeue(), front() o el show().

Cal comentar que el destructor és O(n) ja que ha d'eliminar tots els nodes que utilitza i açò implica un recorregut per els «n» elements de la cua amb un cost computacional d'O(n).

És una bona observació dir que necessitem un node d'enllaçament individual i no cal un node més gran.

Per últim cal dir que a l'hora d'eliminar elements amb dequeue() hem hagut de duplicar un node, canviar el seu adreçament de manera que quan fèiem el delete no eliminés també el element al qual apuntava.

Observació de l'algorisme del main():

El generador de nombres random genera sempre el mateix cada vegada que compila.