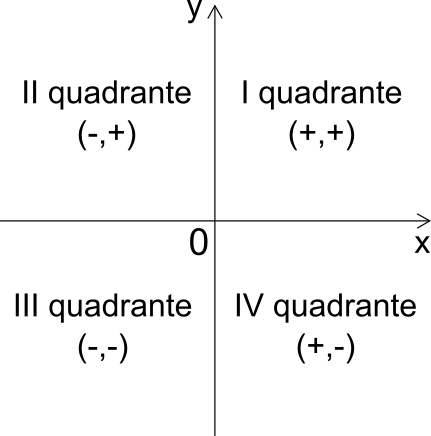
**Attività 8 - Queue**

Realizzare la propria implementazione della struttura dati CODA con le funzioni di libreria viste a lezione:

* newQueue() → q
  + Post: q = nil
* emptyQueue(q) → b
  + Post: se q=nil allora b = true altrimenti b = false
* enqueue(e, q) → q’
  + Post: se q = nil allora q’ = <e> altrimenti   
    se q = <a1, a2, … an> con n > 0 allora q’ = <a1, a2, … an, e>
* dequeue(q) → a
  + Pre: q = <a1, a2, …, an-1, an> n>0 (q ≠ nil)
  + Post: a = a1 e l’elemento a1 viene rimosso da q

Dopodiché, implementare le seguenti funzioni:

1. **Coda di punti**: Creare una coda i cui item sono *ADT punto* (l’ADT punto è stato implementato nelle esercitazioni passate). Le coordinate sono inserite da tastiera ma non è noto a priori il numero di punti da inserire. Prevedere un opportuno input che interrompa l’inserimento dei dati da parte dell’utente.
2. **Sottocode**: A partire dalla coda Q appena realizzata, implementare una funzione che divide la coda in quattro sottocode Q1, Q2, Q3 e Q4 di cui ciascuna conterrà i punti di uno dei quattro quadranti del piano Cartesiano. (*NOTA, al termine della divisione, la coda Q non ha più ragione di esistere*)
   1. Il piano Cartesiano è convenzionalmente diviso in quattro quadranti così ordinati:
3. **Spezzata**: Scelta una delle quattro sottocode realizzate al passo precedente, calcolare la lunghezza della *spezzata* risultante dalla congiunzione dei punti presenti nella coda secondo l’ordine con il quale appaiono nella coda stessa.

*NOTA: la coda non deve essere distrutta.*

* 1. Esempio: se la coda è quella del primo quadrante mostrata qui di seguito a sinistra, la spezzata ottenuta sarà quella mostrata a destra la cui lunghezza sarà pari a 24.23

|  |  |  |
| --- | --- | --- |
|  | **X** | **Y** |
| *Head ->* | 2 | 0 |
|  | 1.2 | 9 |
|  | 4 | 3.1 |
|  | 0.8 | 3 |
| *Tail ->* | 3 | 8 |

1. **Concatenazione**: Implementare una funzione di concatenazione tra code queue concatena (queue q1, queue q2) che restituisce una nuova coda ottenuta per concatenazione delle due date code in input. Con tale funzione di concatenazione, costruire una coda Q’ a partire dalle sottocode Q1, Q2, Q3 e Q4 assicurando che gli elementi appaiano in ordine discendente di quadrante Q’ = *head->* (Q4, Q3, Q2, Q1) *<-tail (NOTA. Al termine della concatenazione, le sottocode non hanno più motivo di esistere. Q’ non è un clone di Q perché inevitabilmente non potremmo ricostruire l’esatto ordine degli item nella coda originale Q).*
2. **Scheduling dei processi*[avanzato]*** Modificare l’ADT punto in modo che esso piuttosto che rappresentare le coordinate di un punto cartesiano, rappresenti un processo in memoria di cui siano noti un ID numerico e il *CPU-burst,* ossia numero di cicli di clock della CPU di cui il processo ha bisogno per la sua esecuzione e terminazione (*basta semplicemente vedere la modifica di ADT punto in ADT processo tale che quella che era l’ascissa del punto diventi un int che rappresenta l’ID numerico del processo mentre l’ordinata un int per il CPU-burst del processo*).

Un processo resta nella coda di memoria per il tempo necessario alla sua esecuzione, ossia quando la CPU ha eseguito il processo per un numero di cicli di clock corrispondenti al suo CPU-burst. Ogni volta che la CPU è libera, il primo processo nella coda viene estratto e assegnato alla CPU. Per una equa distribuzione delle risorse, i processi non possono monopolizzare la CPU e quindi dopo un numero fissato di cicli di CPU, se essi non terminano, devono lasciare la CPU e ritornare nella coda dei processi riaccodandosi ai processi presenti con criterio FIFO (first-in, first-out)

Realizzare pertanto una coda di processi inseriti in memoria con dati inseriti da tastiera (*oppure inserire proceduralmente un numero N di processi con ID intero progressivo, e CPU-burst random con le funzioni: srand(time(NULL)); cpu\_burst=rand()%100;* ).

Una volta realizzata la coda dei processi, impostare da tastiera il numero di cicli di CPU e simulare l’esecuzione dei processi con i riaccodamenti necessari fino all’esaurimento di tutta la coda dei processi.

ESEMPIO. Sia data la seguente coda di processi ( (1, 10) , (2, 3) , (3, 8) , (4, 5) ) e numero di cicli CPU per esecuzione pari a 4. Quello che avverrà sarà:

* Viene estratto il processo con ID: 1 e CPU-burst: 10. Dopo 4 cicli di clock non è terminato, quindi viene reinserito nella coda con lo stesso ID e CPU-burst pari a 6 =10-4  
  Quindi la coda diventa ( (2, 3) , (3, 8) , (4, 5) (1, 6) )
* Viene estratto il processo ID: 2 e CPU-burst: 3 che tuttavia in un solo passo di esecuzione viene eseguito totalmente e termina
* Viene estratto il processo ID: 3 e CPU-burst: 4 che dopo 4 cicli non è terminato e ritorna in coda con CPU-burst: 4=8-4;  
  La coda sarà: ( (4, 5), (1, 6), (3, 4))
* Viene estratto il processo ID:4, CPU-burst: 4 che viene riaccodato con CPU-burst 1=5-4  
  La coda sarà: ( (1, 6), (3, 4), (4, 1) )
* Viene estratto il processo ID:1 e CPU-burst 6 che ritorna in coda con CPU-burst 2=6-4

La coda sarà ( (3, 4), (4, 1), (1,2))

* Si continua con i processi ID:3 e poi ID4 e poi ID1 che terminano tutti entro il numero di cicli di CPU per processo e la coda si svuota.

*Curiosità: questo algoritmo di schedulazione dei processi in memoria è realmente esistente ed è noto come Algoritmo di Schedulazione dei Processi Round-Robin.*