Estructures lineals II
Programació 2
Facultat d'Informàtica de Barcelona, UPC

Conrado Martínez

Primavera 2019

- Apunts basats en els d'en Ricard Gavaldà
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

El tipus cua (queue)

La classe queue

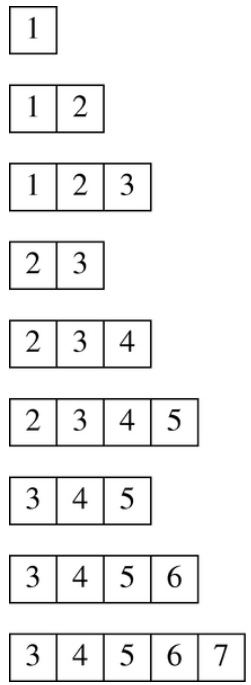
Ofereix tres operacions bàsiques:

- Afegir un nou element al final (*encuar*)
- Treure el primer element (*desencuar*)
- Examinar el primer element (*front*)

FIFO - *First In, First Out*: el primer que ha entrat serà el primer en sortir i és l'únic accessible

Exemple d'evolució d'una cua

```
queue<int> c;
c.push(1); c.push(2); c.push(3);
c.pop();
c.push(4); c.push(5);
c.pop();
c.push(6); c.push(7);
```



Especificació de la classe queue

La classe queue

```
template <class T> class queue {
public:
// Constructores

/* Pre: cert */
/* Post: crea una cua buida  */
queue();

// Destructora
~queue();
```

Especificació de queue

La classe queue

```
// Modificadores

/* Pre: la cua és [a_1,...,a_n], n ≥ 0 */
/* Post: s'afegit l'element x com a últim de la cua, es
         a dir, la cua és ara [a_1,...,a_n,x] */
void push(const T& x);

/* Pre:  la cua és [a_1,...,a_n] i no està buida (n > 0) */
/* Post: s'ha eliminat el primer element de la cua original,
         es a dir, la cua ara és [a_2,...,a_n] */
void pop();
```

Especificació de queue

La classe queue

```
// Consultores

/* Pre: la cua és [a_1,...,a_n] i no està buida (n > 0) */
/* Post: Retorna a_1 */
T front() const;

/* Pre: cert */
/* Post: Retorna cert si i només si la cua està buida */
bool empty() const;

private:
  ...
};
```

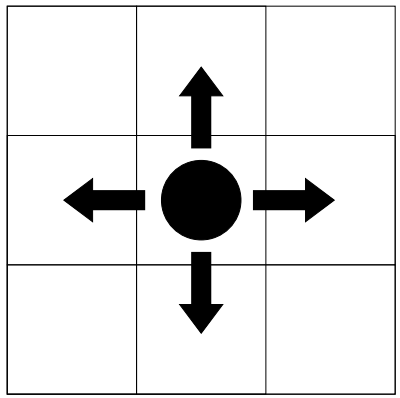
Aplicacions de les cues: laberints

En aquest exemple veurem com trobar un camí, si n'hi ha, entre dues posicions donades d'un laberint. L'ús d'una cua ens permetrà, de fet, trobar un camí de longitud mínima en el cas que existeixin camins alternatius entre les posicions donades.

Aplicacions de les cues: laberints

- Laberints rectangulars: $m \times n$ posicions, organitzades en m files i n columnes
- Cada posició (i, j), $1 \leq i \leq m$, $1 \leq j \leq n$, pot estar lliure o ser una paret
- Donades dues posicions ini i fi, volem una funció que ens digui si existeix un camí entre ini i fi en un laberint L donat
- En els apunts trobereu un exemple més complet, en el qual la funció ens retorna un camí de longitud mínima, si existeix; per a trobar aquest camí es fa servir una pila

Aplicacions de les cues: laberints



- Una posició (i, j) és vàlida si $i \geq 1$, $i \leq m$, $j \geq 1$ i $j \leq n$
- Un camí entre ini i fi és una seqüència de ℓ posicions vàlides lliures adjacents, sent ini la primera posició de la seqüència i fi l'última
- Totes les posicions, exceptes les de la vora, tenen quatre posicions adjacents: al nord, a l'est, al sud i a l'oest.

Aplicacions de les cues: laberints

```
xxxxxxxxxx
x.x.x.x..x     hi ha un camí de longitud 11 (mínima)
x...x.x.xx     entre (2,2) i (6,1)
x.x.x...xx
x.x.x.x..x
xxx...x..x     no hi ha cap camí entre (2,2) i (8,7)
x.x.xxx..x
x.x.x..xxx
x...x.xx.x
xx.xx.x..x
x..x..x..x
xx.x.xxxxx
xx...x..xx
x.......xx
xxxxxxxxxx
```

Matriu de caràcters representant un laberint de $m = 13$ files i $n = 8$ columnes ('X'=paret, '.'=posició lliure)

Aplicacions de les cues: laberints

- La classe `Laberint` ens ofereix mètodes com ara
  - `marcar`(p): deixa una marca a una posició vàlida p d'un laberint,
  - `lliure`(p): torna cert si la posició p és vàlida i està lliure,
  - `marcada`(p): torna cert si la posició p s'ha marcat
  - ...
- La classe `pos` és molt senzilla, representa un parell $(fila, columna)$ i ens dona una constructora `pos(i,j)`, les consultores `fila` i `col`, `es_igual` per a comparar, etc.

Aplicacions de les cues: laberints

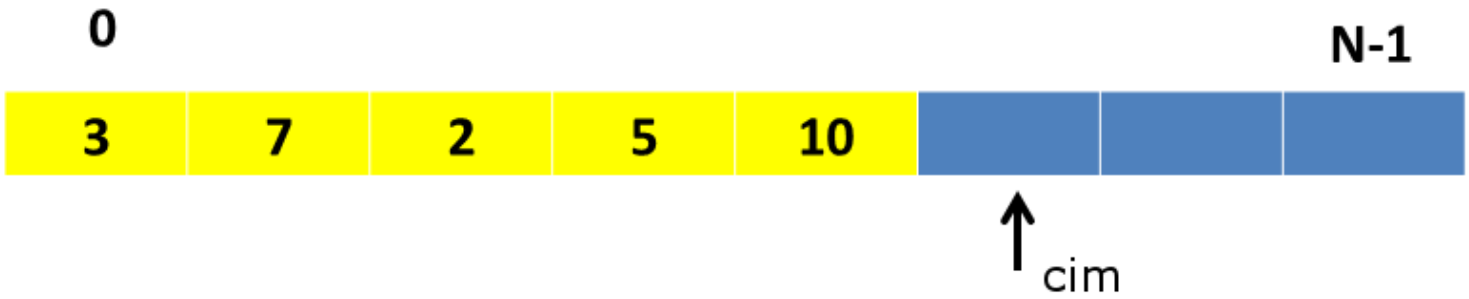
```
// Pre: L.lliure(ini), L.lliure(fi),
//        totes les posicions vàlides de L sense marcar
bool busca_cami(const Laberint& L, pos ini, pos fi) {
  queue<pos> Q;
  Q.push(ini); L.marcar(ini);
  bool cami_trobat = false;
  while (not Q.empty() and not cami_trobat) {
    pos p = Q.front(); Q.pop();
    L.marcar(p); // marquem la posició p
    if (p.es_igual(fi)) cami_trobat = true;
    else {
      ...
    }
  }
  // 'cami_trobat'==true si i només si existeix un camí
  // entre 'ini' i 'fi'
  return cami_trobat;
}
```

Aplicacions de les cues: laberints

```
// p = (i,j) != fi
pos nord(p.fila()-1, p.col());
if (L.lliure(nord) and not L.marcada(nord)) {
    Q.push(nord); L.marcar(nord);
}
pos est(p.fila(), p.col()+1);
if (L.lliure(est) and not L.marcada(est)) {
    Q.push(est); L.marcar(est);
}
pos sud(p.fila()+1, p.col());
if (L.lliure(sud) and not L.marcada(sud)) {
    Q.push(sud); L.marcar(sud);
}
pos oest(p.fila(), p.col()-1);
if (L.lliure(oest) and not L.marcada(oest)) {
    Q.push(oest); L.marcar(oest);
}
```

Implementacions amb vectors

Implementació de piles amb vectors



- Invariant de la representació:

$$0 \leq \texttt{cim} \leq \texttt{elems.size}() = \texttt{MAX\_SIZE}$$

- Alternativament, sense mida màxima fent servir `push_back()`

Implementació de piles amb vectors

stack.hpp

```
template <class T> class stack {
public:
  ...
private:
  vector<T> elems;
  int cim;
  static const int MAX_SIZE = 100;
};
```

Implementació de piles amb vectors

stack.hpp

```
template <class T> class stack {
public:
  stack() {
     cim = 0;
     elems = vector<T>(MAX_SIZE);
  };
  ~stack() {};
```

Implementació de piles amb vectors

stack.hpp

```
  ...
void push(const T& x) {
  if (cim == MAX_SIZE) ... // ERROR: pila plena
  elems[cim] = x;
  ++cim;
}

void pop() {
  if (cim == 0) ... // ERROR: pila buida
  --cim;
}
```

Implementació de piles amb vectors

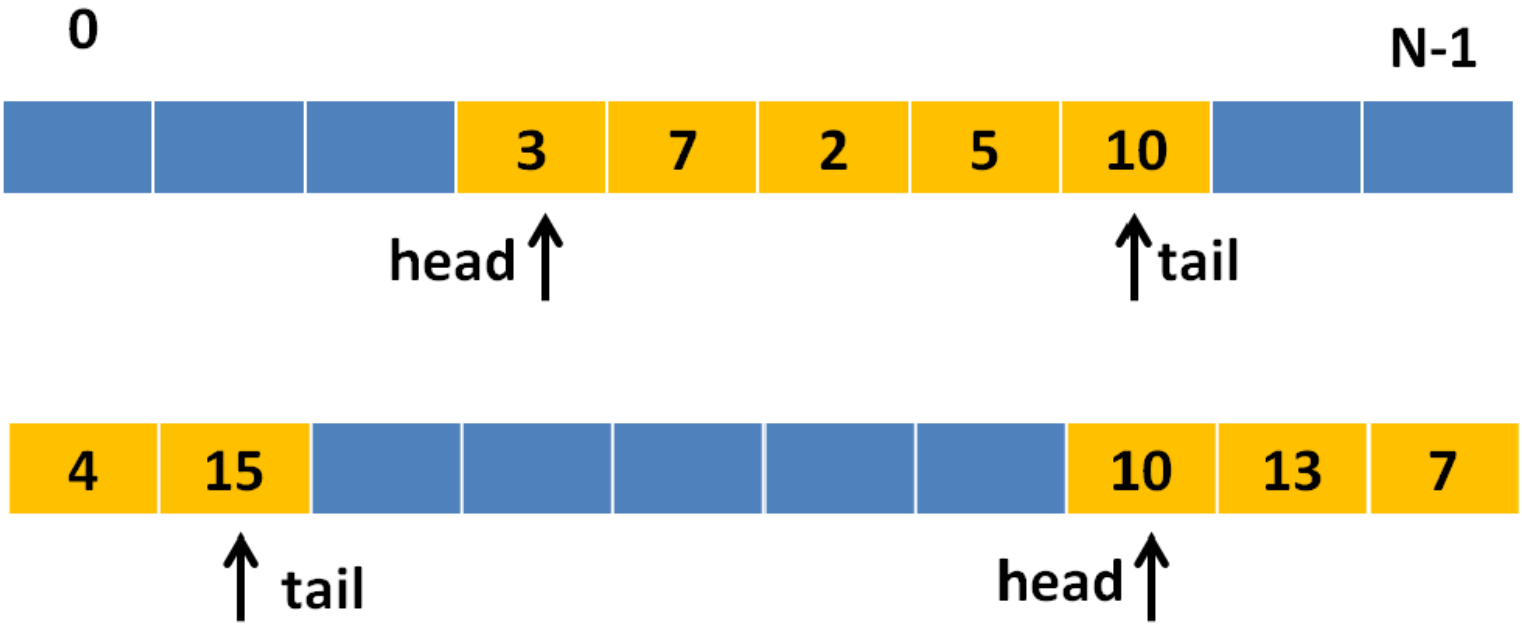
stack.hpp

```
  ...
T top() const {
  if (cim == 0) ... // ERROR: pila buida
  return elems[cim - 1];
}

bool empty const {
  return cim == 0;
}

private:
   ...
};
```

Implementació de cues amb vectors: Cua circular



- push: `tail = (tail+1)%N`
- pop: `head = (head+1)%N`
- size: `(tail-head+1)%N`
- ambigüetat quan tail = head-1. Buida o plena? $\Rightarrow$ afegir un comptador d'elements o no omplir mai (`size < N` sempre)