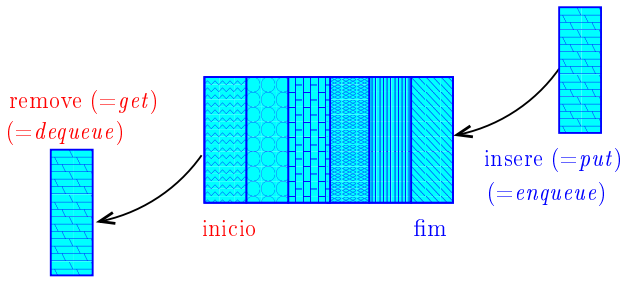
Melhores momentos

AULA 15

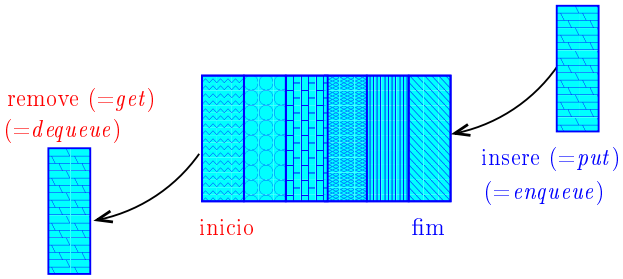
Filas

Uma **fila** (=*queue*) é uma lista dinâmica em que todas as inserções são feitas em uma extremidade chamada de **fim** e todas as remoções são feitas na outra extremidade chamada de **início**.



Filas

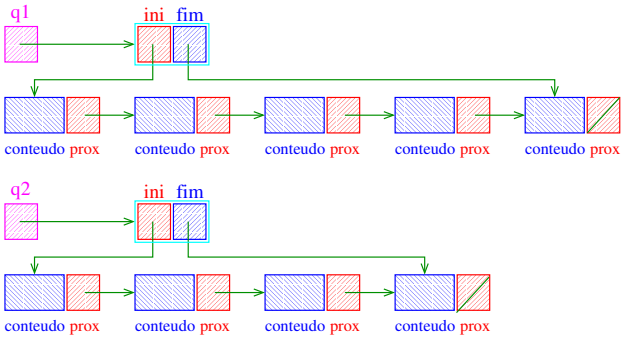
Assim, o primeiro objeto a ser removido de uma fila é o primeiro que foi inserido. Esta política de manipulação é conhecida pela sigla **FIFO** (=*First In First Out*)



AULA 16

FilaS implementadaS em listaS encadeadaS

As filas serão armazenada em listas encadeadas.



FilaS implementadaS em listaS encadeadaS

Uma fila `q` é um ponteiro para uma `struct` com campos `ini` e `fim`.

Para cada fila `q` há ponteiros `ini` e `fim`.

`q->ini->conteudo` é o primeiro elemento da fila `q`.

`q->fim->conteudo` é o último elemento da fila `q`.

A fila `q` está vazia se "`q->ini == NULL`".

Interface item.h

```
/*
 * item.h
 */
typedef int Item;
```

Interface queue.h

```
/*
 * queue.h
 * INTERFACE: funcoes para manipular filas
 * ATENCAO: Esta interface permite que
 * varias filas sejam utilizadas.
 */
typedef struct queue *Queue;

Queue queueInit(int);
int queueEmpty(Queue);
void queuePut(Queue, Item);
Item queueGet(Queue);
void queueFree(Queue);
```

distancias

A função distancias recebe um um inteiro n, uma matriz A representado um rede de estradas entre n cidades e uma cidade c e devolve um vetor d que registra a distancia da cidade c a cada uma das outras: d[i] é a distância de c a i.

```
int *
distancias (int n, int **A, int c) {
  int *d; /* d[i] = distancia de c a i*/
  int j;
  Queue q;
```

distancias

```
/* aloque vetor de distancias */
d = mallocSafe(n* sizeof(int));

q = queueInit(n); /* crie uma fila */

/* inicialize o vetor de distancias */
for (j = 0; j < n; j++)
  d[j] = n; /* distancia n = infinito */
d[c] = 0;


queuePut(q,c); /* coloque c na fila */
```

distancias

```
while (!queueEmpty(q)) {
  int i = queueGet(q);
  int di = d[i];
  for (j = 0; j < n; j++)
      if (A[i][j] == 1 && d[j] > di+1) {
          d[j] = di + 1;
          queuePut(q,j);
      }
}
queueFree(q);
return d;
}
```

Implementação queue.c

```
/*
 * FILA: uma implementacao em lista
 * encadeada
 */
typedef struct queueNode* Link;
struct queueNode {
  Item conteudo;
  Link prox;
};
struct queue {
  Link ini, fim;
};
typedef struct queue *Queue;
```

Implementação queue.c

```
static Link
new(Item item, Link prox)
{
  Link p = mallocSafe(sizeof *p);
  p->conteudo = item;
  p->prox = prox;
  return p;
}
```

Implementação queue.c

```
Queue
queueInit(int n)
{
  Queue q = mallocSafe(sizeof *q);
  q->ini = NULL;
  return q;
}

int
queueEmpty(Queue q)
{
  return q->ini == NULL;
}
```

Implementação queue.c

```
void
queuePut(Queue q, Item item)
{
  if (q->ini == NULL)
  {
      q->ini = new(item, NULL);
      q->fim = q->ini;
      return;
  }
  q->fim->prox = new(item, NULL);
  q->fim = q->fim->prox;
}
```

Implementação queue.c

```
Item
queueGet(Queue q)
{
  Link p = q->ini;
  Item item = q->ini->conteudo;

  q->ini = q->ini->prox;
  free(p);
  return item;
}
```

Implementação queue.c

```
void
queueFree(Queue q)
{
  while (q->ini != NULL)
  {
      Link t = q->ini->prox;
      free(q->ini);
      q->ini = t;
  }
  free(q);
}
```

Análise de algoritmo

Programming Pearls: Algorithm Design Techniques,
Jon Bentley, Addison-Wesley, 1986

Segmento de soma máxima

Um **segmento** de um vetor $v[0..n-1]$ é qualquer subvetor da forma $v[e..d]$.

Problema: Dado um vetor $v[0..n-1]$ de números inteiros, determinar um segmento $v[e..d]$ de soma máxima.

Entra:

	0									$n-1$
v	31	−41	59	26	−53	58	97	−93	−23	84

Segmento de soma máxima

Sai:

	0		2				6			$n-1$
v	31	−41	59	26	−53	58	97	−93	−23	84

$v[e..d] = v[2..6]$ é segmento de soma máxima.

$v[2..6]$ tem soma **187**.

Segmento de soma máxima

Problema (versão simplificada): Determinar a soma máxima de um segmento de um dado vetor $v[0..n-1]$.

Entra:

	0									$n-1$
v	31	−41	59	26	−53	58	97	−93	−23	84

Sai:

	0		2				6			$n-1$
v	31	−41	59	26	−53	58	97	−93	−23	84

A soma máxima é **187**.

Algoritmo café-com-leite

```
void segMax3(int v[],int n,int *e,int *d,
                        int *sMax){
  int i, j, k, s;
1 *sMax = 0; *e = *d = -1;
2 for (i = 0; /*1*/ i < n; i++)
3   for (j = i; j < n; j++) {
4       s = 0;
5       for (k = i; /*2*/ k <= j; k++)
6           s += v[k];
7       if (s > *sMax){
8           *sMax = s; *e = i; *d = j;
        }
    }
}
```

Correção de algoritmos

Estrutura "típica" de demonstrações da correção de algoritmos iterativos através de suas **relações invariantes** consiste em:

1. verificar que a relação vale no início da primeira iteração;
2. demonstrar que se a relação vale no início da iteração, então ela vale no final da iteração (com os papéis de alguns atores possivelmente trocados);
3. concluir que, se relação vale no início da última iteração, então a a relação junto com a condição de parada implicam na correção do algoritmo.

Correção

Relação **invariante** chave:

(i0) em /*1*/ vale que: $v[*e..*d]$ é um segmento de soma máxima com $*e < i$. ♡

	*e		i				*d			$n-1$
v	31	−41	59	26	−53	58	97	−93	−23	84

Correção

Mais relações **invariantes**:

(i1) em /*1*/ vale que:

$*\texttt{sMax} = \texttt{v}[*\texttt{e}] + \texttt{v}[*\texttt{e}+1] + \texttt{v}[*\texttt{e}+2] + \cdots + \texttt{v}[*\texttt{d}];$

(i2) em /*2*/ vale que:

$\texttt{s} = \texttt{v}[\texttt{i}] + \texttt{v}[\texttt{i}+1] + \texttt{v}[\texttt{i}+2] + \cdots + \texttt{v}[\texttt{k}-1].$