Listas lineares

Aula 09

Ivone P. Matsuno Yugoshi
ivone.matsuno@ufms.br

Ronaldo Fiorilo dos Santos
ronaldo.santos@ufms.br

Universidade Federal de Mato Grosso do Sul
Câmpus de Três Lagoas
Bacharelado em Sistemas de Informação

Algoritmos e Programação II

Slides baseados no material do Prof. Fábio Henrique Viduani Martinez - FACOM/UFMS

Definição

- Uma lista linear é uma estrutura de dados que armazena um conjunto de informações que são relacionadas entre si
  - nomes e telefones de uma agenda telefônica, as informações bancárias dos funcionários de uma empresa, as informações sobre processos em execução pelo sistema operacional, etc
- cada informação contida na lista é um registro contendo os dados relacionados, chamados de **célula**
- usamos um desses dados como uma chave para realizar diversas operações sobre essa lista
- dados que acompanham a chave são irrelevantes e participam apenas das movimentações das células, podemos imaginar então que uma lista linear é composta apenas pelas chaves das células e que as chaves são representadas por números inteiros

Definição

- uma **lista linear** é um conjunto de $n \geq 0$ células $c_1, c_2, \ldots, c_n$ determinada pela ordem relativa desses elementos:
  - (i) se $n > 0$ então c_1 é a primeira célula;
  - (ii) a célula c_i é precedida pela célula c_{i-1}, para todo i, $1 < i \leq n$.
- as operações básicas sobre uma lista linear são as seguintes:
  - busca;
  - inclusão; e
  - remoção.
- dependendo da aplicação, muitas outras operações também podem ser realizadas sobre essa estrutura

Definição

- listas lineares podem ser armazenadas na memória de duas maneiras distintas:
  - alocação estática ou sequencial: os elementos são armazenados em posições consecutivas de memória, com uso da vetores;
  - locação dinâmica ou encadeada: os elementos podem ser armazenados em posições não consecutivas de memória, com uso de ponteiros
- o problema que queremos resolver é que define o tipo de armazenamento a ser usado, dependendo das operações sobre a lista, do número de listas envolvidas e das características particulares das listas
- já vimos as operações básicas sobre uma lista linear em alocação sequencial

Definição

- as células de uma lista linear em alocação encadeada encontram-se dispostas em posições aleatórias da memória e são ligadas por ponteiros que indicam a posição da próxima célula da lista
- um campo é acrescentado a cada célula da lista indicando o endereço do próximo elemento da lista

Definição

- definição de uma célula de uma lista linear encadeada:

```
struct cel {
    int chave;
    struct cel *prox;
};
```

- definir um novo tipo de dados para as células de uma lista linear em alocação encadeada:

```
typedef struct cel celula;
```

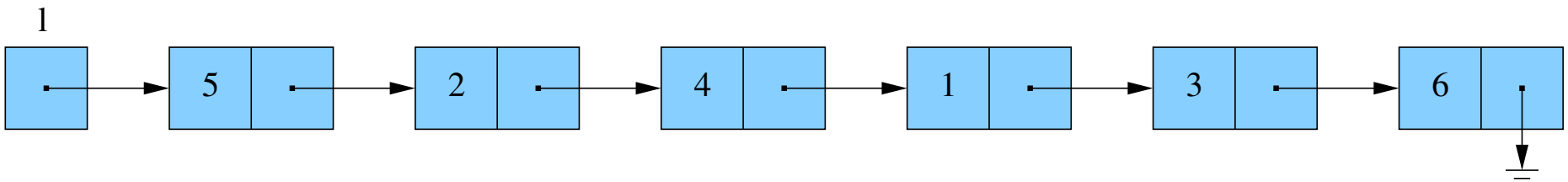
- uma célula `c` e um ponteiro `p` para uma célula podem ser declarados da seguinte forma:

```
celula c;
celula *p;
```

Definição

- se `c` é uma célula então `c.chave` é o conteúdo da célula e `c.prox` é o endereço da célula seguinte
- se `p` é o endereço de uma célula então `p->chave` é o conteúdo da célula apontada por `p` e `p->prox` é o endereço da célula seguinte
- Se `p` é o endereço da última célula da lista então `p->prox` vale `NULL`

Definição



Definição

- o endereço de uma lista encadeada é o endereço de sua primeira célula

Definição

- o endereço de uma lista encadeada é o endereço de sua primeira célula
- se `p` é o endereço de uma lista, podemos dizer que "`p` é uma lista" ou ainda "considere a lista `p`"

Definição

- o endereço de uma lista encadeada é o endereço de sua primeira célula
- se `p` é o endereço de uma lista, podemos dizer que "`p` é uma lista" ou ainda "considere a lista `p`"
- quando dizemos "`p` é uma lista", queremos dizer que "`p` é o endereço da primeira célula de uma lista"

Definição

- Uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa:

Definição

- Uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa:
  - **com cabeça**: a primeira célula serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante; a primeira célula é a **cabeça** da lista

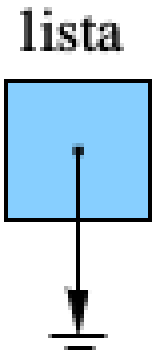
Definição

- Uma lista linear pode ser vista de duas maneiras diferentes, dependendo do papel que sua primeira célula representa:
  - **com cabeça**: a primeira célula serve apenas para marcar o início da lista e portanto o seu conteúdo é irrelevante; a primeira célula é a **cabeça** da lista
  - **sem cabeça**: o conteúdo da primeira célula é tão relevante quanto o das demais
- Nesta aula veremos listas lineares **sem cabeça**

Definição

- para criar uma lista vazia `lista` sem cabeça, basta escrever as seguintes sentenças:

```
celula *lista;
lista = NULL;
```



Operações sobre listas lineares

- De acordo com cada aplicação são necessárias operações para manipular lista linear
- As principais operações utilizadas são:
  - Mostrar todos os elementos da lista
  - Busca de um valor de campo chave na lista
  - Remoção de um elemento da lista
  - Inserção de um elemento da lista
  - Outras
- Na sequência serão discutidas essas operações

Mostrar todos os elementos de uma lista

- para imprimir o conteúdo de todas as células de uma lista linear podemos usar a seguinte função:

```
void imprime_lista(celula *lst)
{
   celula *p;

   for (p = lst; p != NULL; p = p->prox)
      printf("%d\n", p->chave);
}
```

- se `lista` é uma lista linear sem cabeça, a chamada da função deve ser:

```
imprime_lista(lista);
```

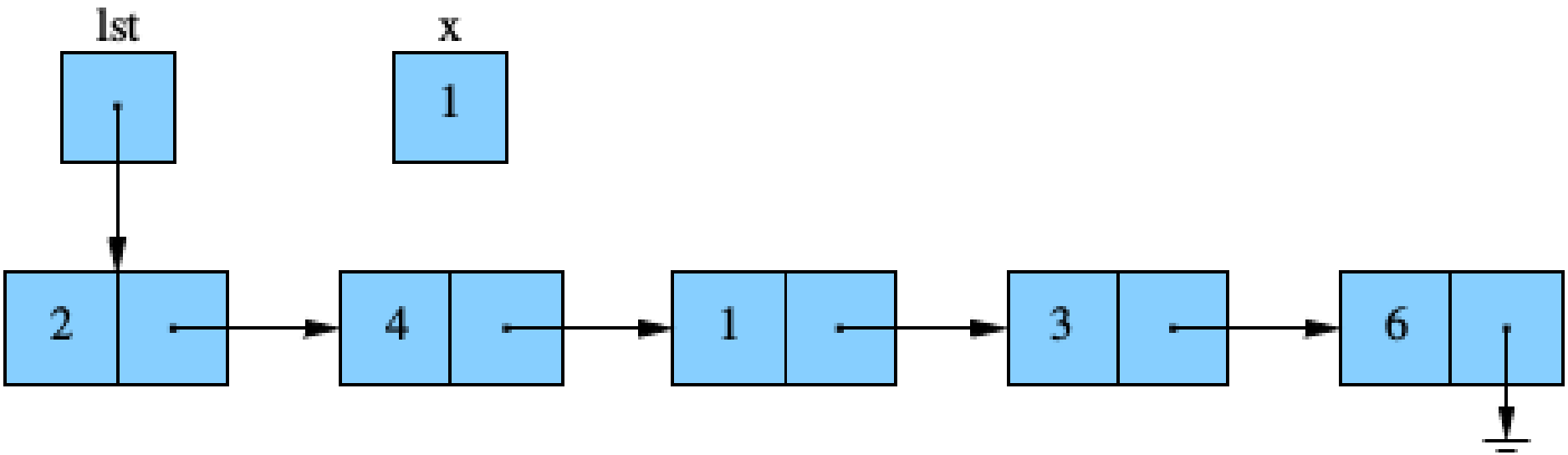
  - `lista` é um ponteiro para o primeiro elemento da lista

Busca em listas lineares sem cabeça

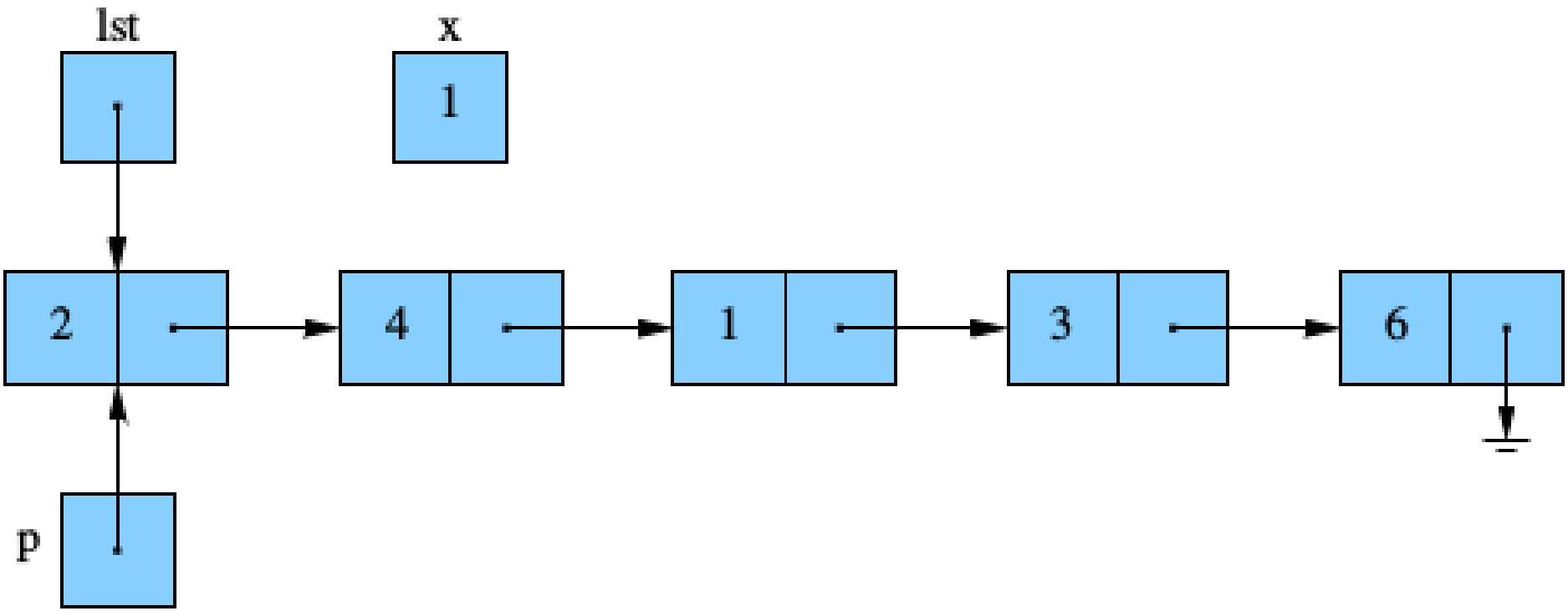
- busca não-recursiva

```
celula *busca_S(int x, celula *lst)
{
   celula *p;

   p = lst;
   while (p != NULL && p->chave != x)
      p = p->prox;

   return p;
}
```

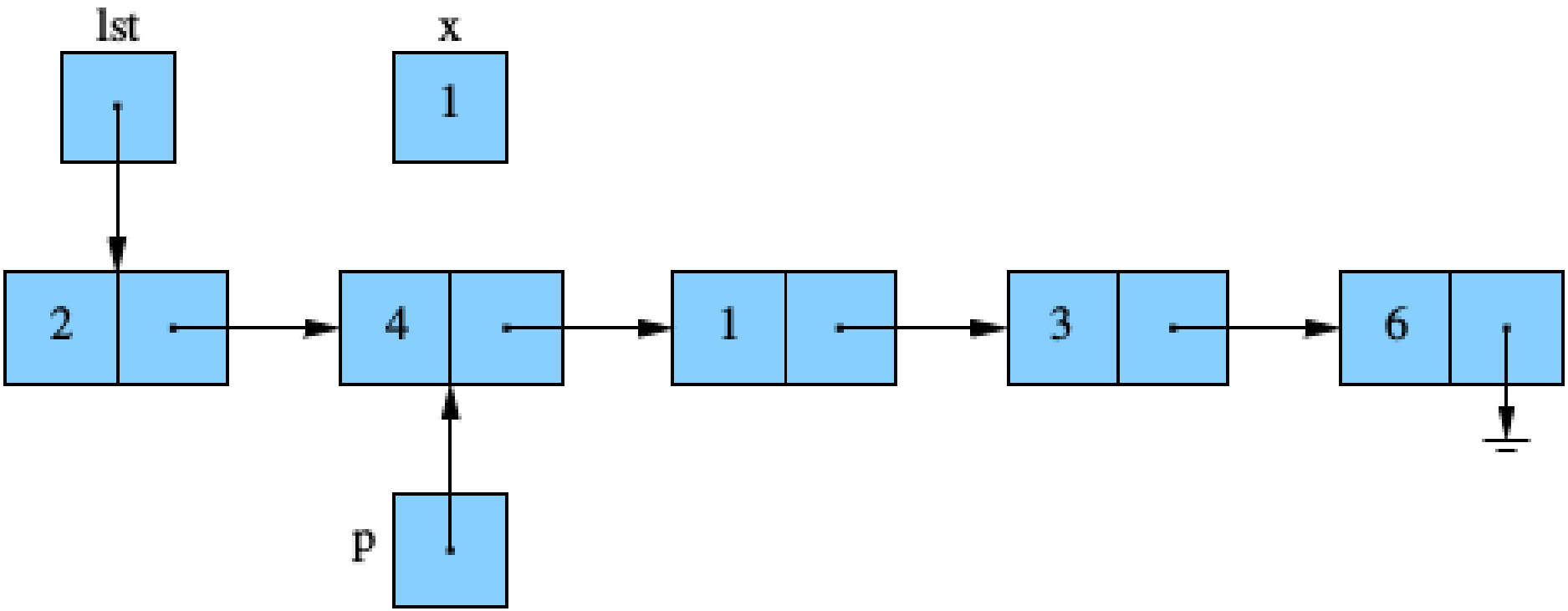
Busca em listas lineares sem cabeça



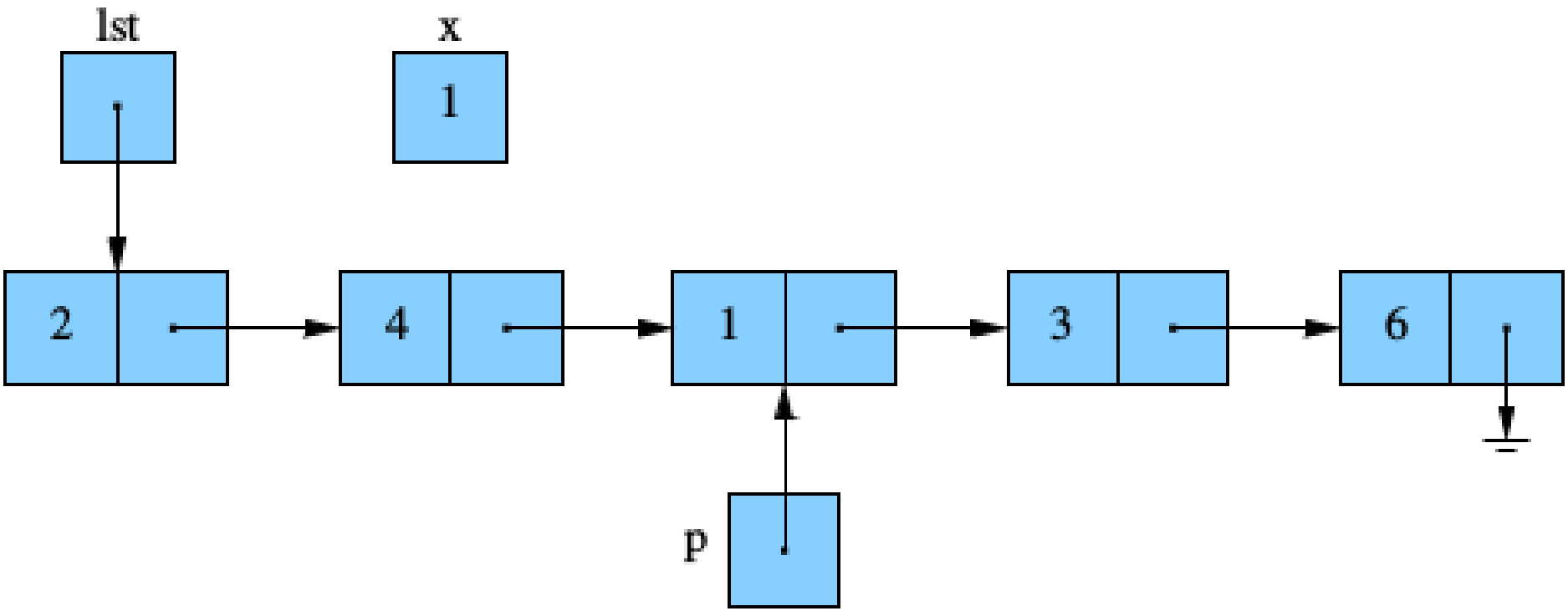
Busca em listas lineares sem cabeça



Busca em listas lineares sem cabeça



Busca em listas lineares sem cabeça



Busca em listas lineares sem cabeça

- busca recursiva

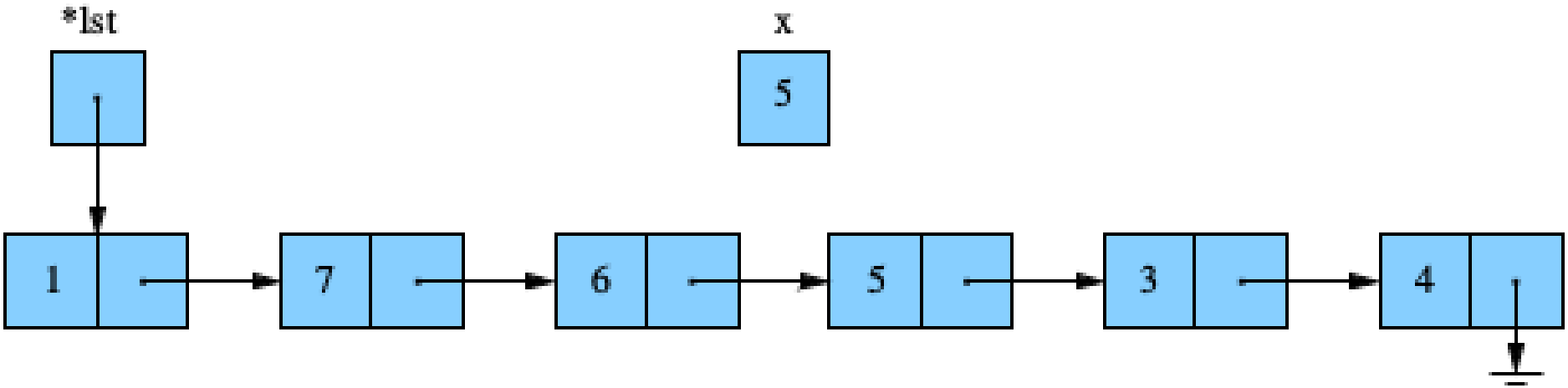
```
celula *buscaR_S(int x, celula *lst)
{
   if (lst == NULL)
      return NULL;

   if (lst->chave == x)
      return lst;

   return buscaR_S(x, lst->prox);
}
```

Remoção de determinado elemento em listas lineares sem cabeça

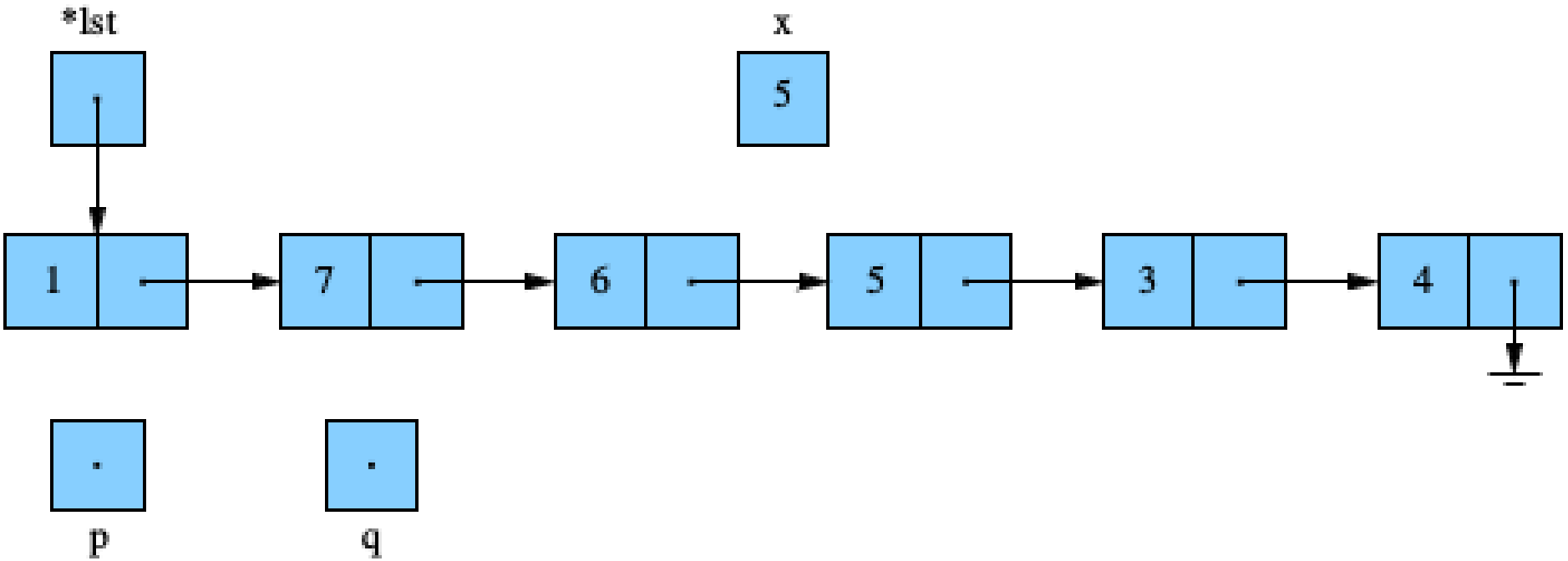
- busca seguida de remoção

```
void busca_remove_S(int x, celula **lst)
{
   celula *p, *q;

   p = NULL;
   q = *lst;
   while (q != NULL && q->chave != x) {
      p = q;
      q = q->prox;
   }
   if (q != NULL)
      if (p != NULL) {
         p->prox = q->prox;
      }
      else {
         *lst = q->prox;
      } free(q);
   }
```

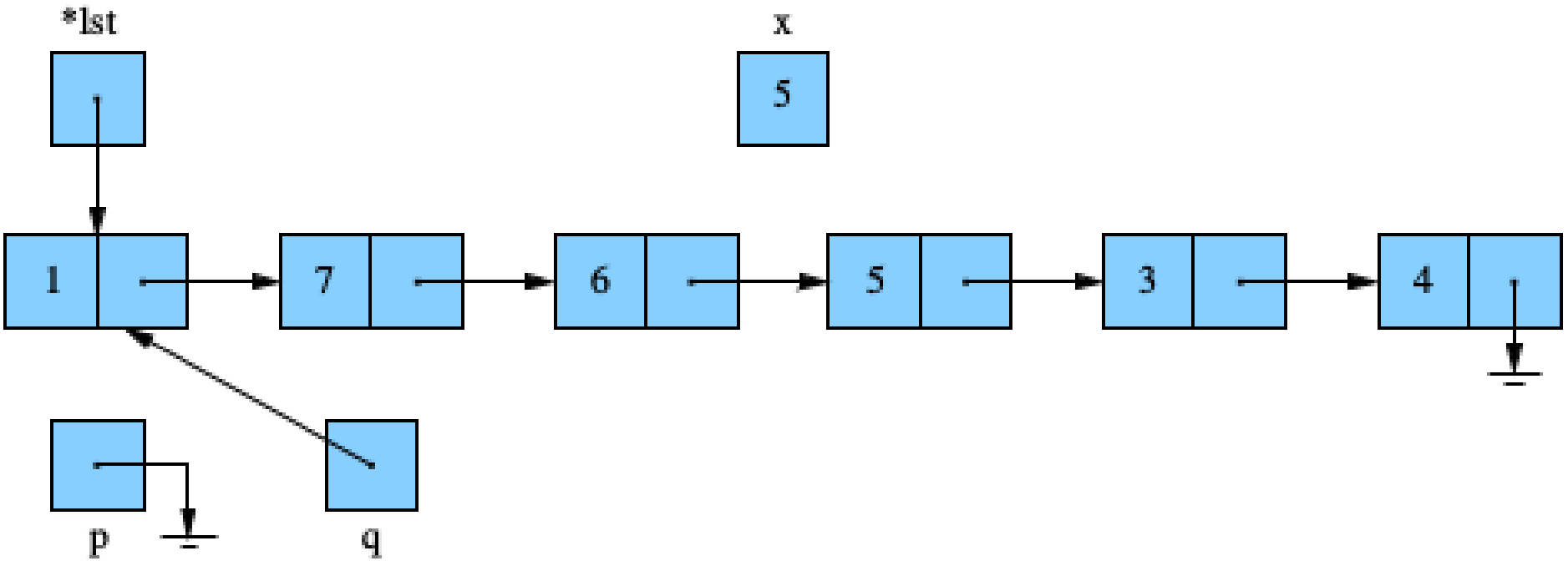
Remoção de determinado elemento em listas lineares sem cabeça



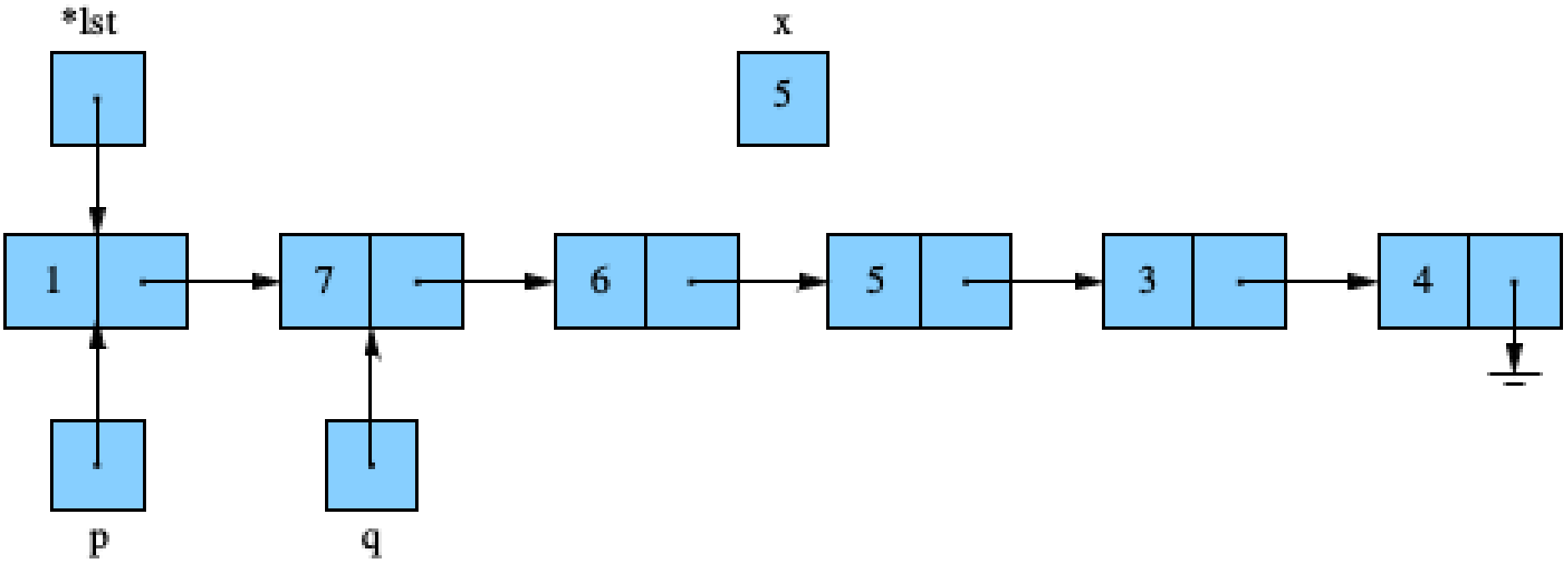
Remoção de determinado elemento em listas lineares sem cabeça



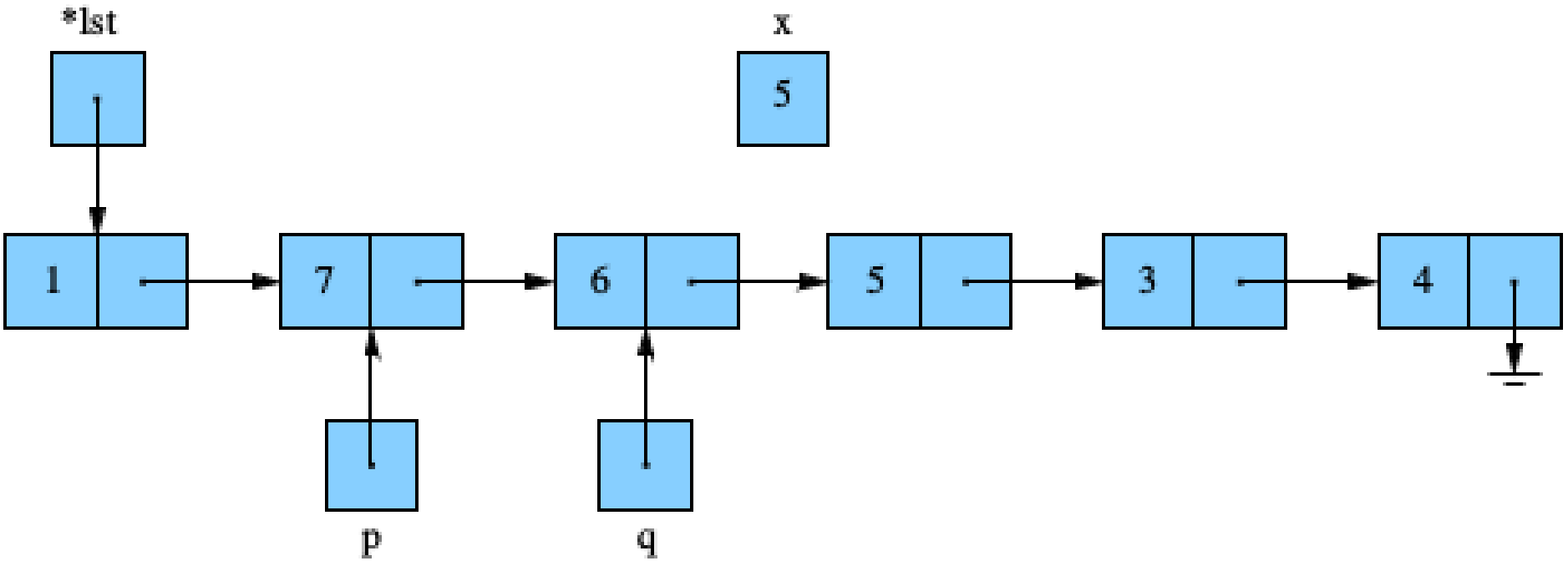
Remoção de determinado elemento em listas lineares sem cabeça



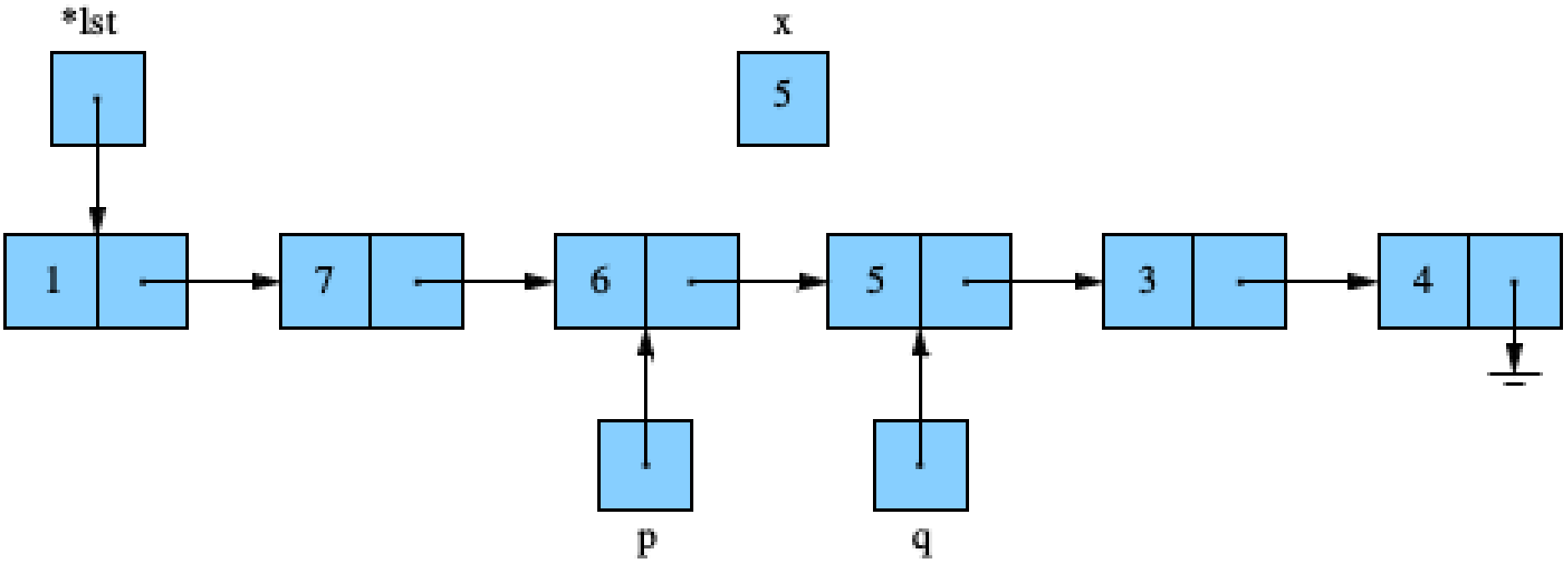
Remoção de determinado elemento em listas lineares sem cabeça



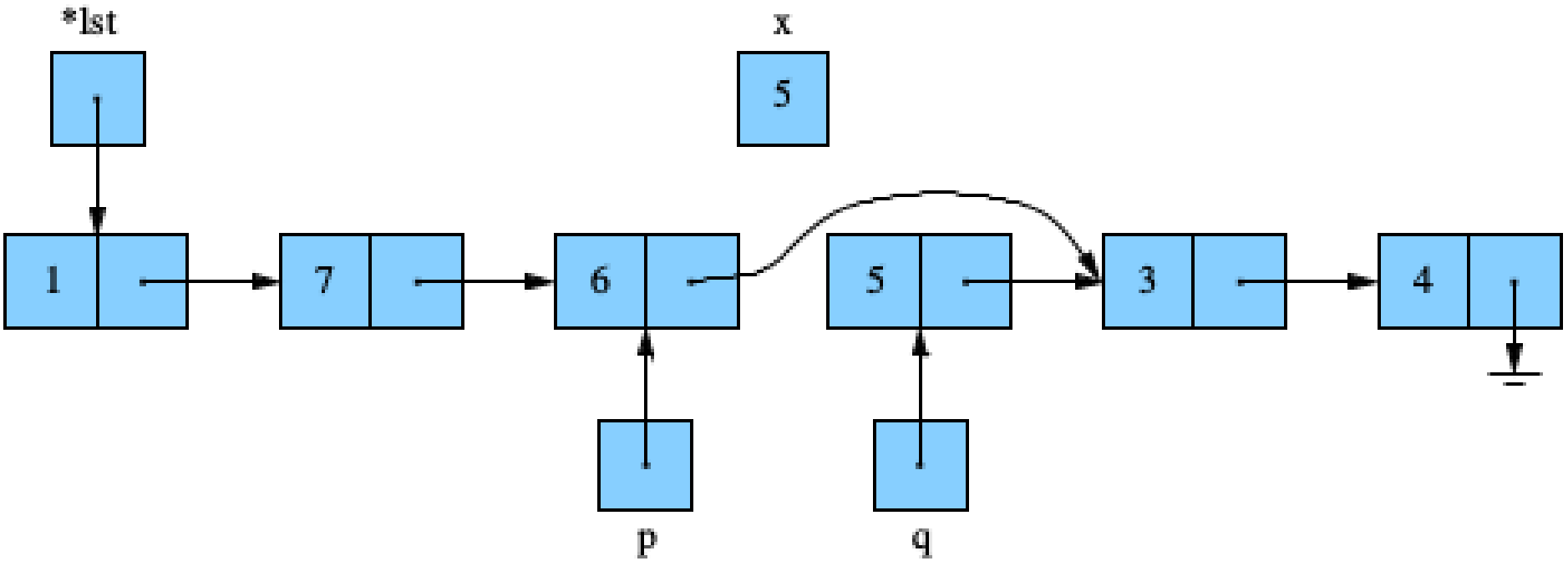
Remoção de determinado elemento em listas lineares sem cabeça



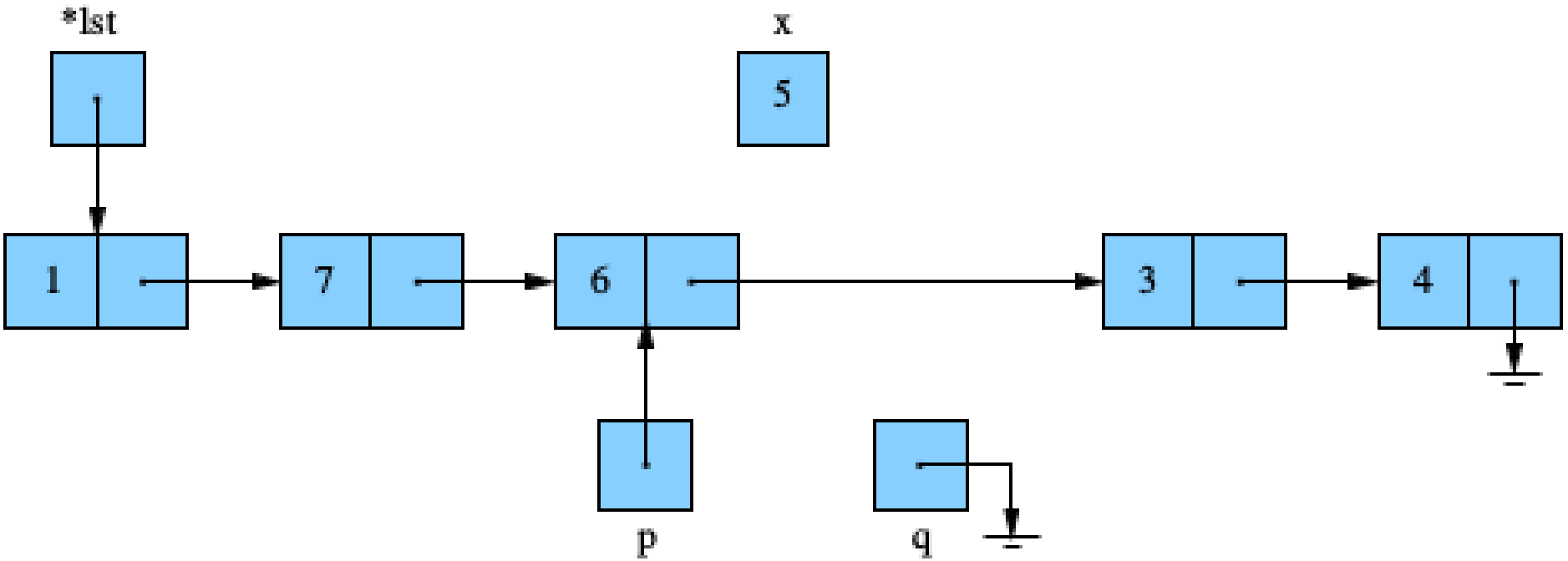
Remoção de determinado elemento em listas lineares sem cabeça



Remoção de determinado elemento em listas lineares sem cabeça



Remoção de determinado elemento em listas lineares sem cabeça



Remoção de determinado elemento em listas lineares sem cabeça

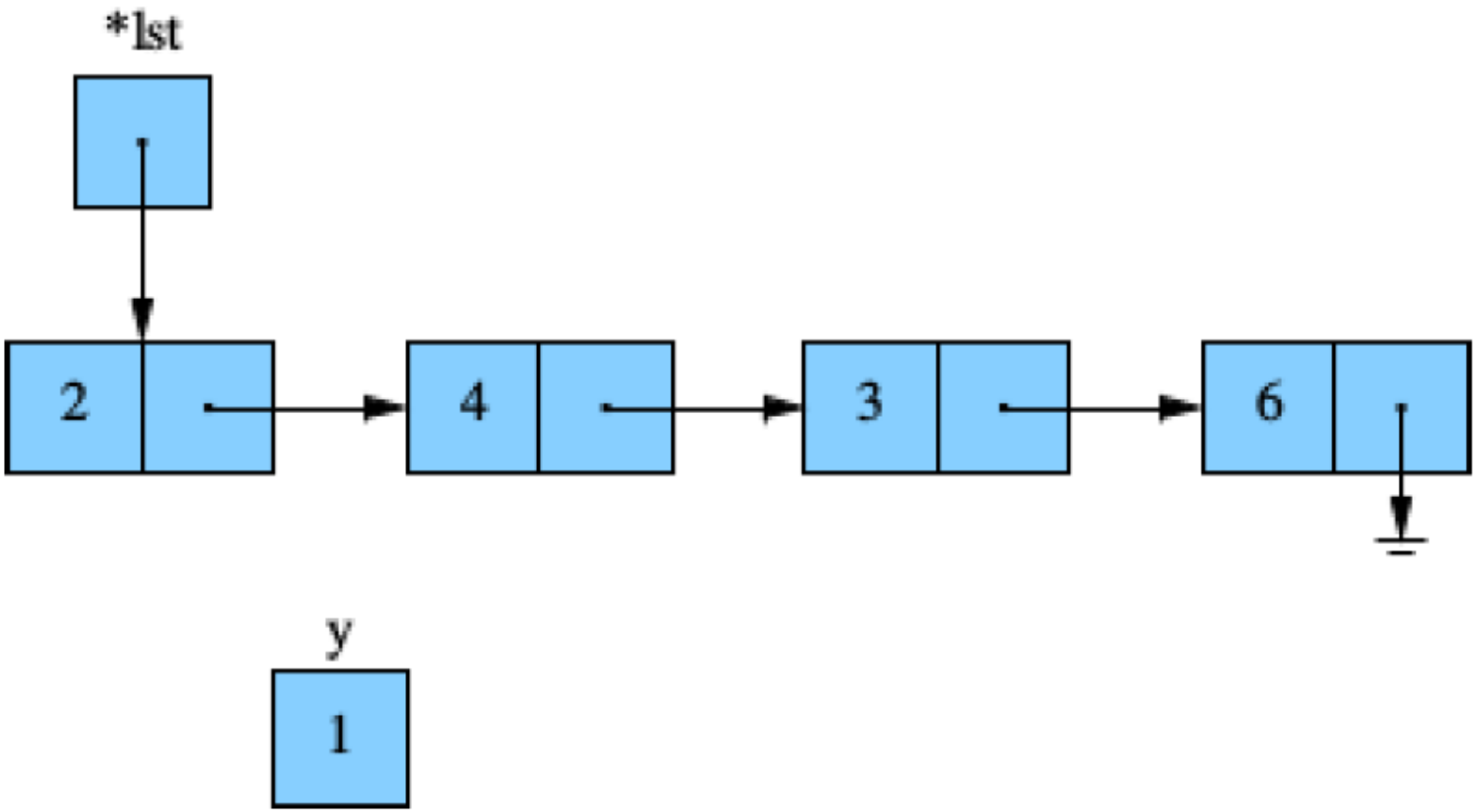
- uma chamada à função `busca_remove_S` é ilustrada abaixo, para um número inteiro `x` e uma `lista`:

```
busca_remove_S(x, &lista);
```

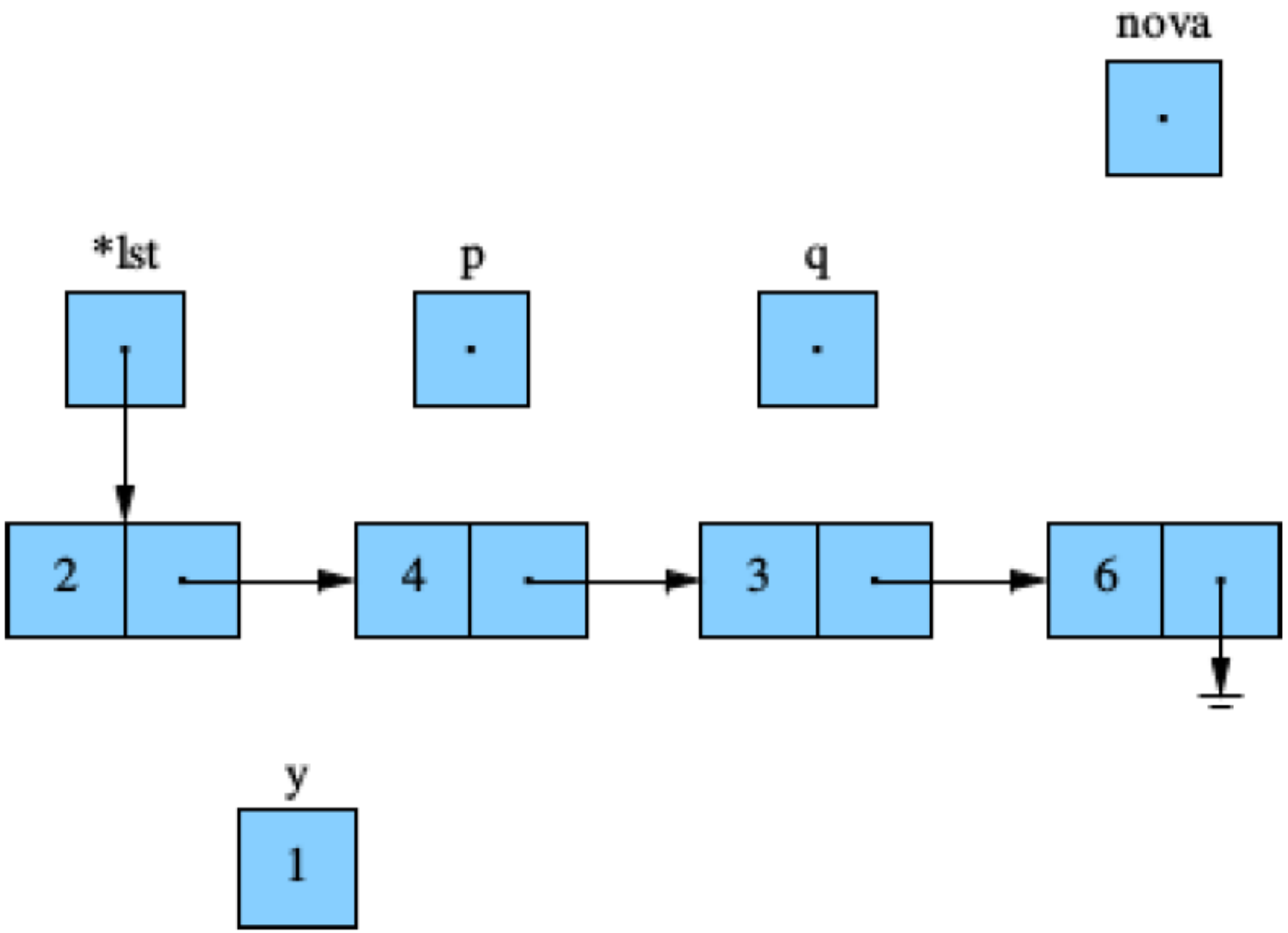
Inserção de determinado elemento em listas lineares sem cabeça

```
void busca_insere_S(int x, celula **lst)
{
   celula *p, *q, *nova;

   p = NULL; q = *lst;
   while (q != NULL && q->chave != x) {
      p = q; q = q->prox;
   }
   if(q == NULL){
      nova = (celula *) malloc(sizeof (celula));
      nova->chave = x;
      nova->prox = NULL;

      if(p!=NULL)
         p->prox=nova;
      else
         *lst=nova;
   }
   else
      printf("elemento já cadastrado");
}
```

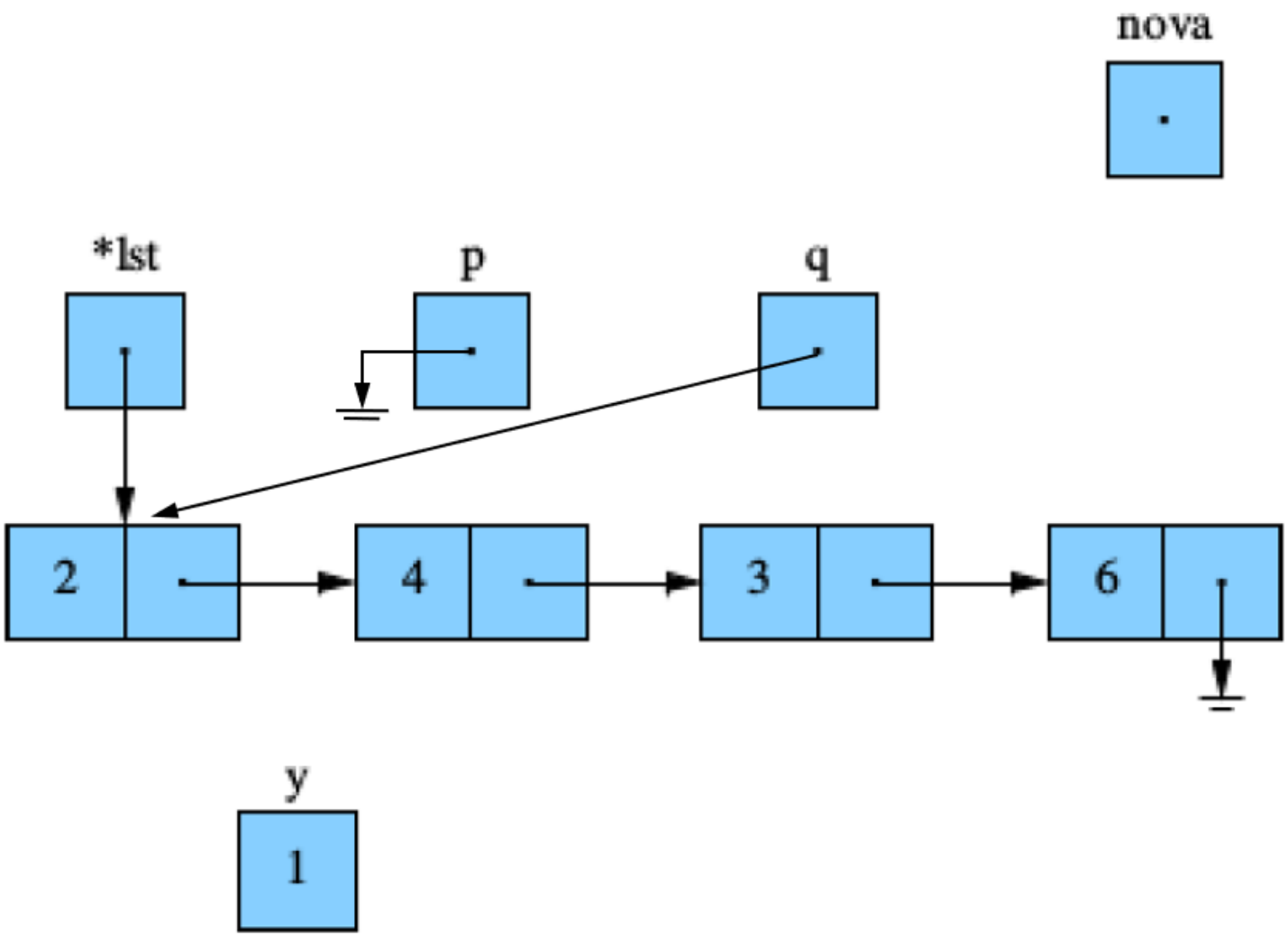
Inserção de determinado elemento em listas lineares sem cabeça



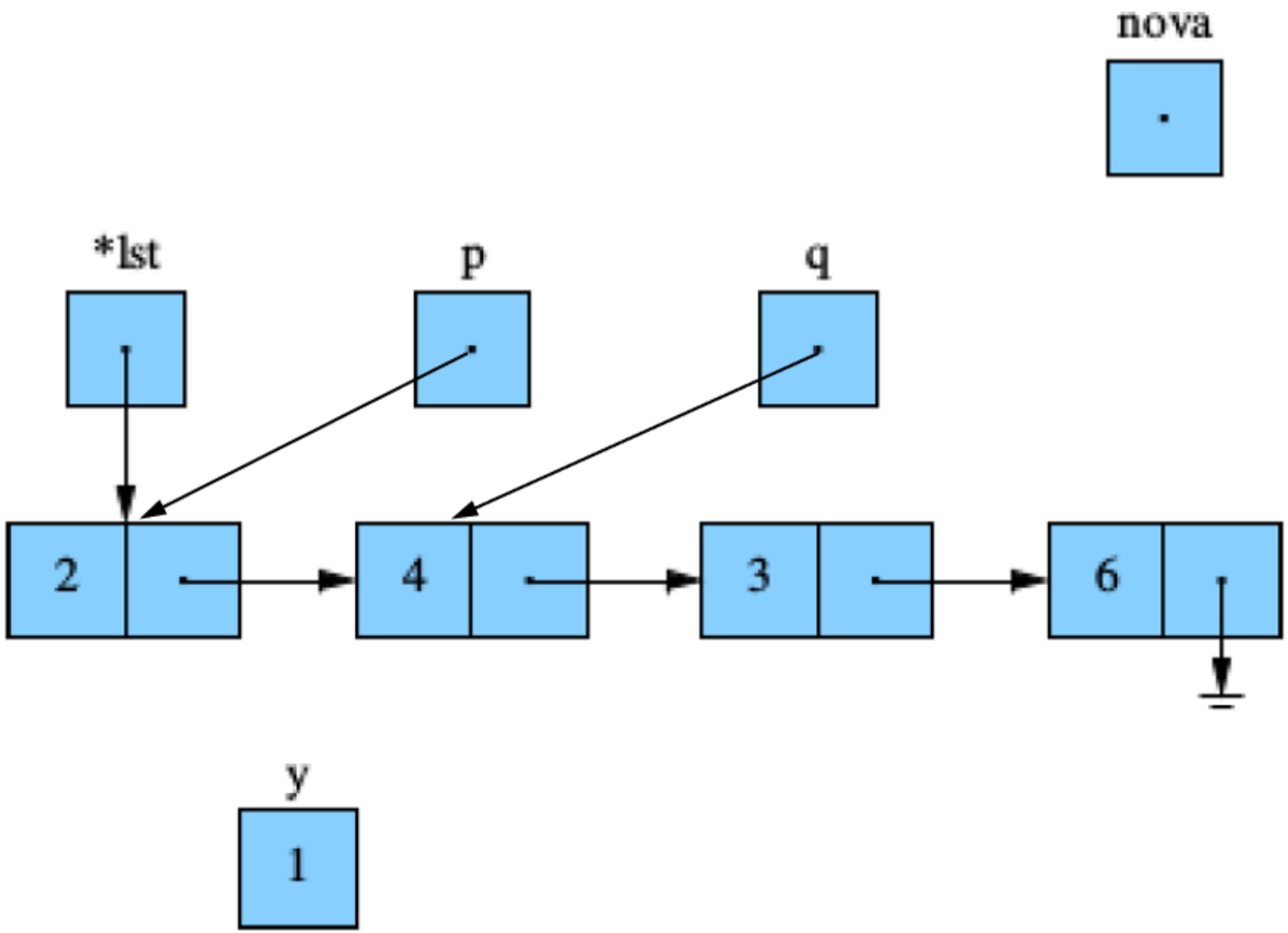
Inserção de determinado elemento em listas lineares sem cabeça



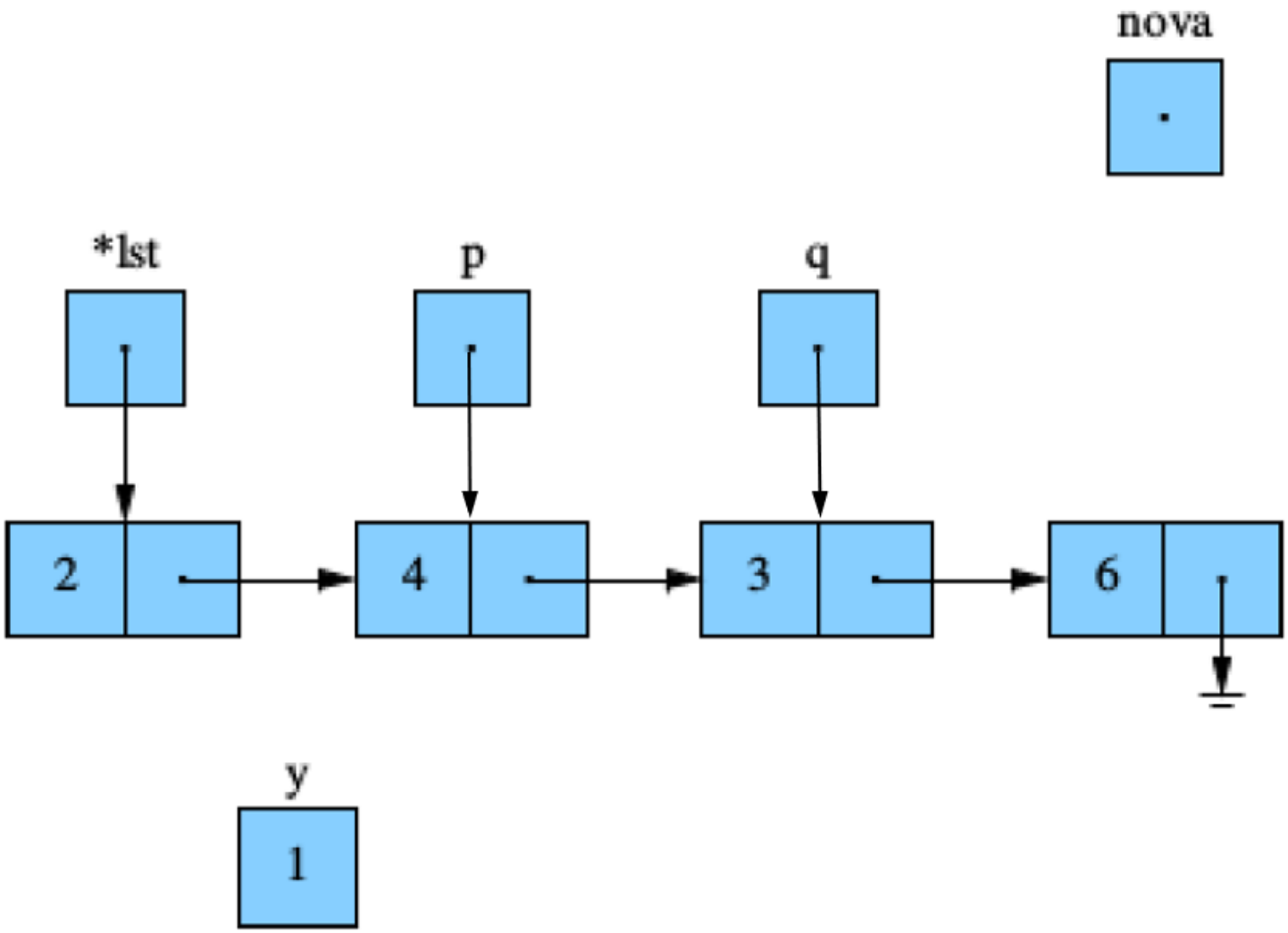
Inserção de determinado elemento em listas lineares sem cabeça



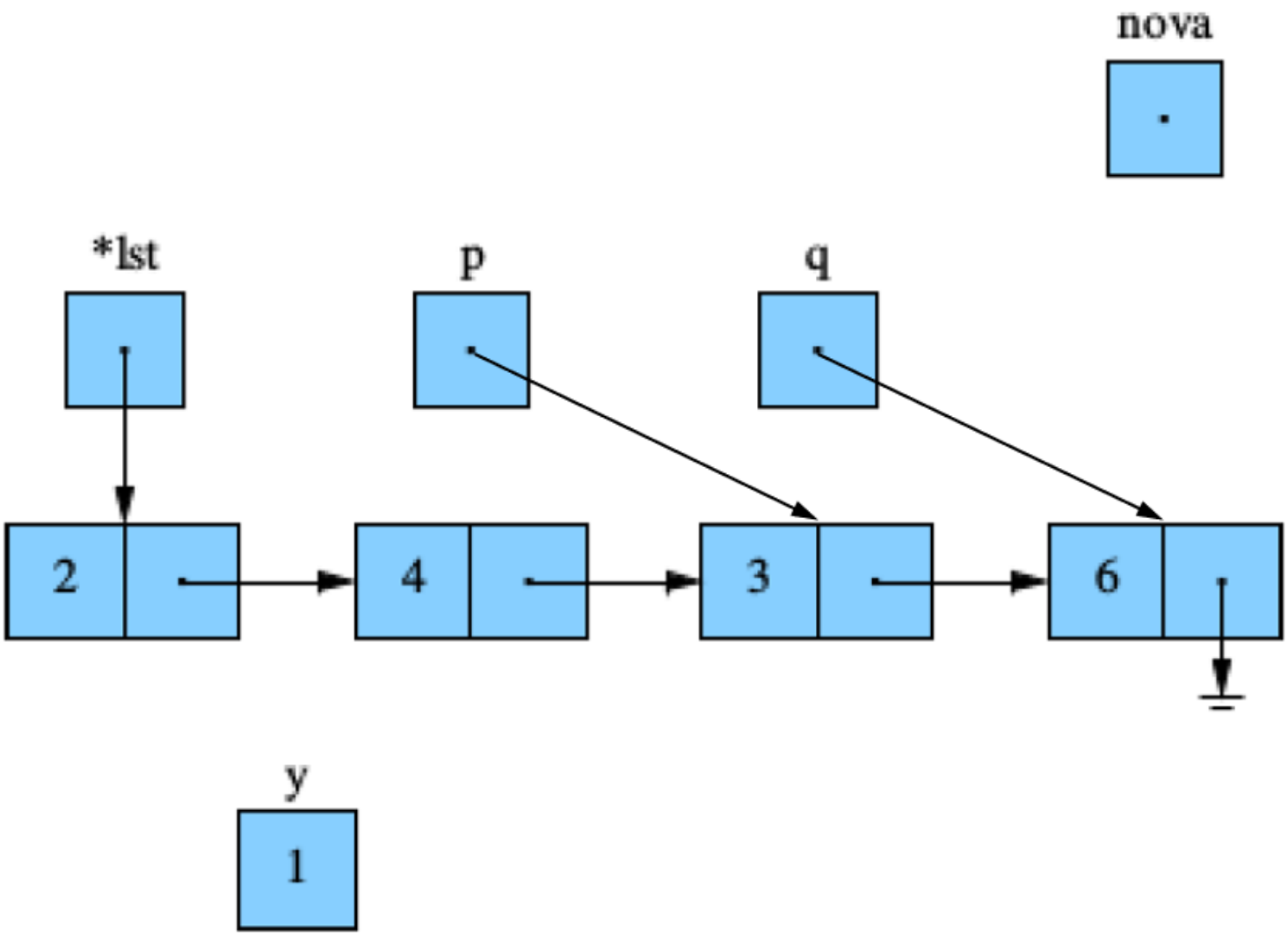
Inserção de determinado elemento em listas lineares sem cabeça



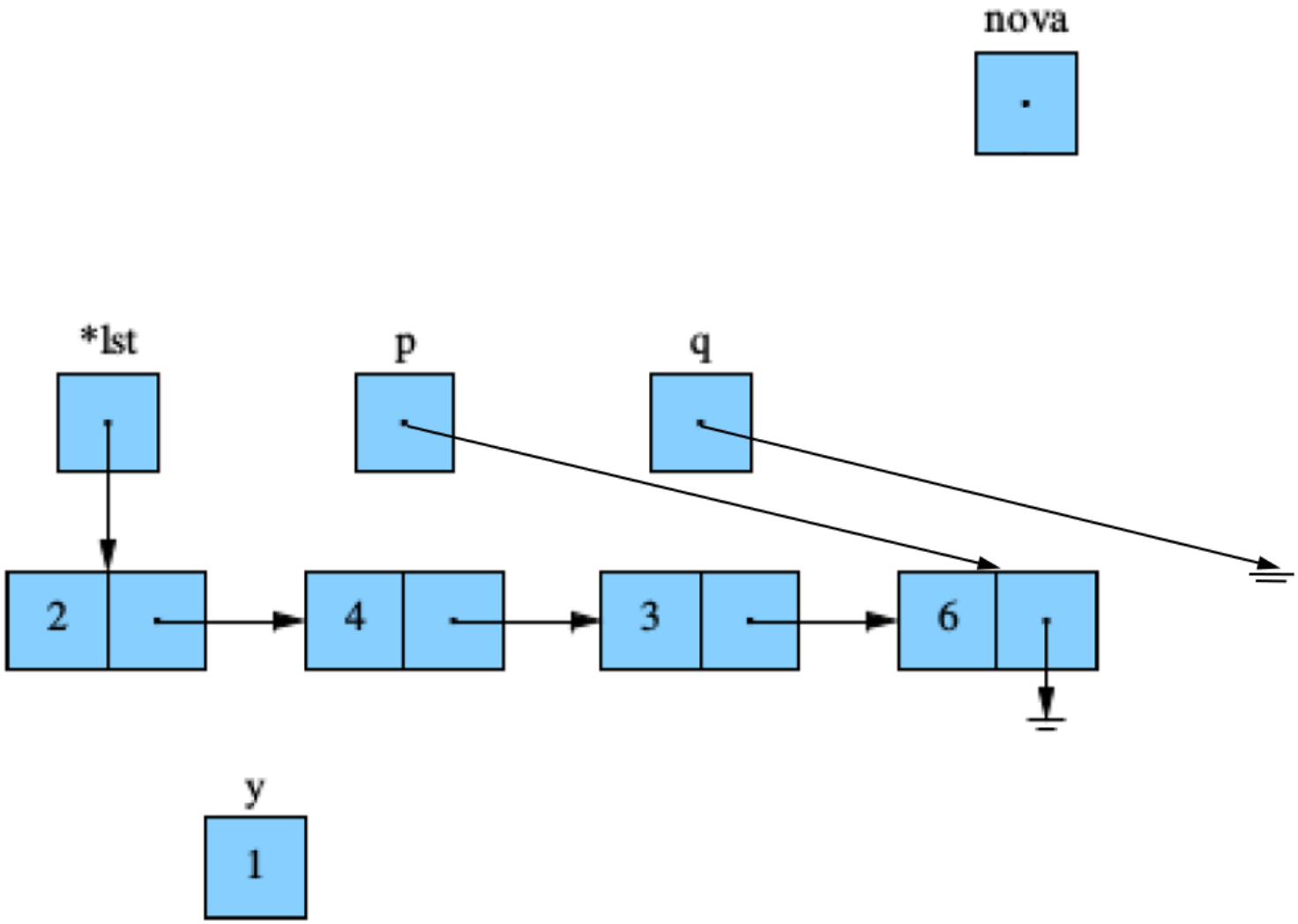
Inserção de determinado elemento em listas lineares sem cabeça



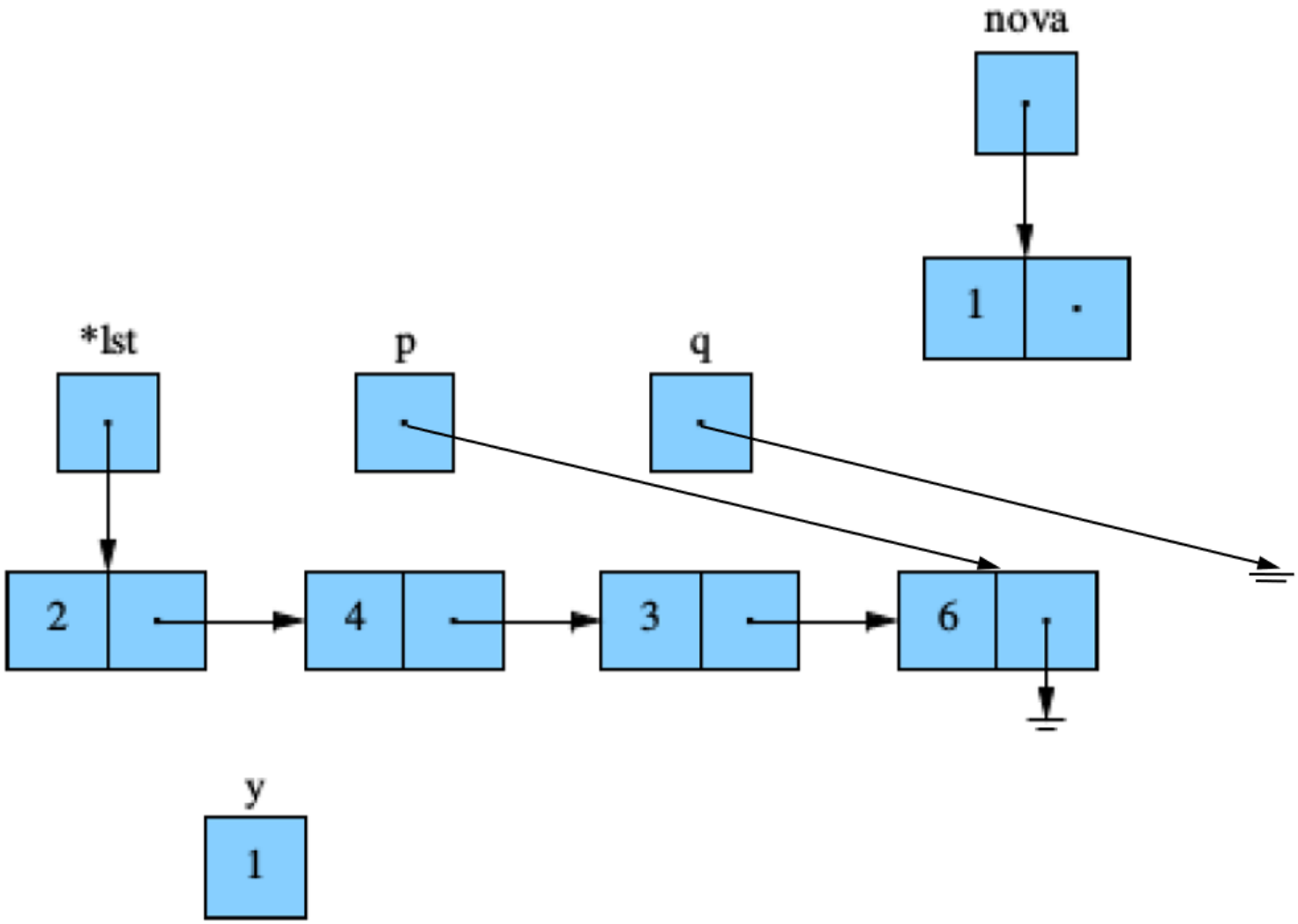
Inserção de determinado elemento em listas lineares sem cabeça



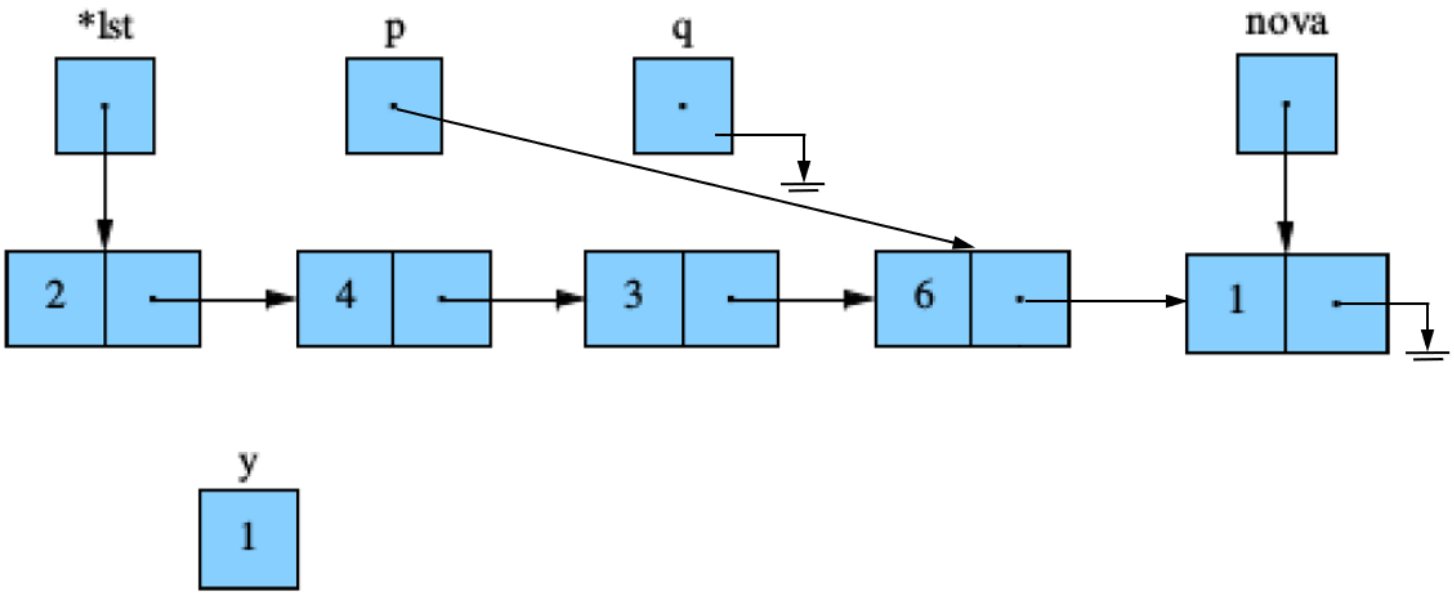
Inserção de determinado elemento em listas lineares sem cabeça



Inserção de determinado elemento em listas lineares sem cabeça



Inserção de determinado elemento em listas lineares sem cabeça



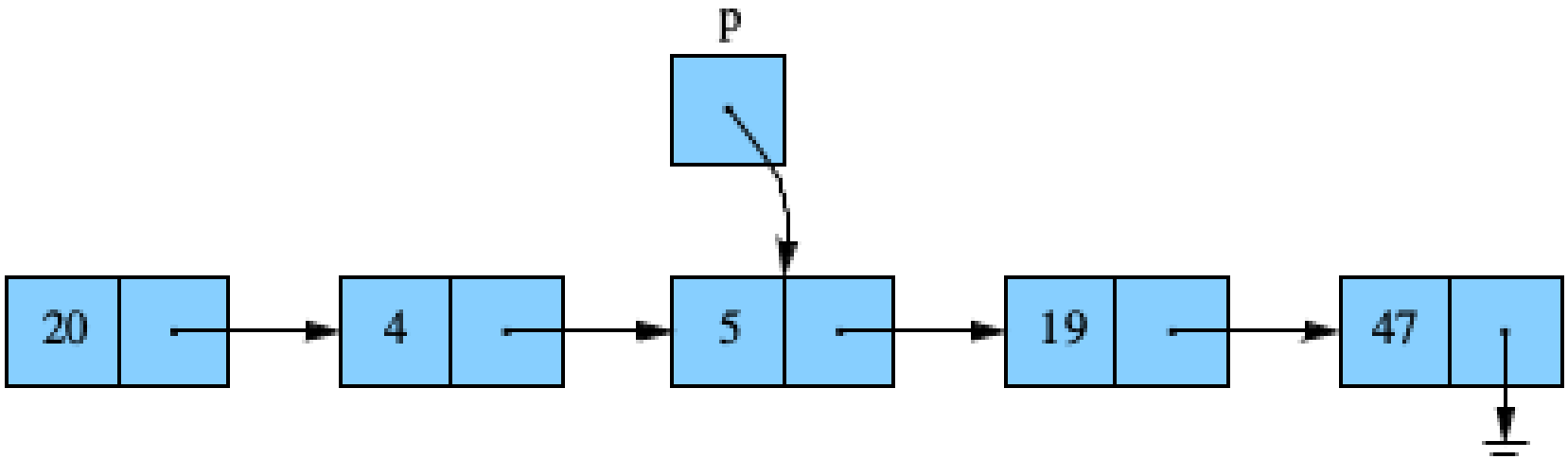
Inserção de determinado elemento em listas lineares sem cabeça

- uma chamada à função **`busca_insere_S`** é ilustrada abaixo, para números inteiros **`y`** e **`x`** e uma **`lista`**:

```
busca_insere_S(x, &lista);
```

Exercícios

1. Se conhecemos apenas o ponteiro `p` para uma célula de uma lista linear em alocação encadeada, como na figura abaixo, e nada mais é conhecido, como podemos modificar a lista linear de modo que passe a conter apenas os valores $20, 4, 19, 47$, isto é, sem o conteúdo da célula apontada por `p`?



Exercícios

2. Sejam S_1 e S_2 dois conjuntos disjuntos de números inteiros. Suponha que S_1 e S_2 estão implementados em duas listas lineares em alocação encadeada. Escreva uma função `uniao` que receba as listas representando os conjuntos S_1 e S_2 e devolva uma lista resultante que representa a união dos conjuntos, isto é, uma lista linear encadeada que representa o conjunto $S = S_1 \cup S_2$. Considere os casos em que as listas lineares encadeadas são com cabeça e sem cabeça.
3. Escreva uma função que encontre uma célula cuja chave tem valor mínimo em uma lista linear encadeada. Considere listas com e sem cabeça e escreva versões não-recursivas e recursivas para a função

Exercícios

4. Seja `lista` uma lista linear com seus conteúdos dispostos em ordem crescente. Escreva funções para realização das operações básicas de busca, inserção e remoção, respectivamente, em uma lista linear com essa característica. Escreva conjuntos de funções distintas para listas lineares com cabeça e sem cabeça. As operações de inserção e remoção devem manter a lista em ordem crescente.
5. Sejam duas listas lineares `lst1` e `lst2`, com seus conteúdos dispostos em ordem crescente. Escreva uma função `concatena` que receba `lst1` e `lst2` e construa uma lista `R` resultante da intercalação dessas duas listas, de tal forma que a lista construída também esteja ordenada. A função `concatena` deve destruir as listas `lst1` e `lst2` e deve devolver `R`. Escreva duas funções para os casos em que as listas lineares encadeadas são com cabeça e sem cabeça.