لیست پیوندی-۲

سید مهدی وحیدی پور

با تشکر از دکتر جواد سلیمی

دانشگاه کاشان- دانشکده مهندسی برق و کامپیوتر

لیستهای پیوندی

- اشاره گرها
- لیست ها
- لیست های دایره ای
- پشته ها و صفهای پیوندی
- چند جمله ای ها
- روابط هم ارزی
- لیستهای دو پیوندی و لیست های تعمیم یافته

چند جمله ای ها

- بازنمایی چند جمله ایها با لیست های پیوندی

$$A(x) = a_{m-1}x^{e_{m-1}} + \cdots + a_0 x^{e_0}$$

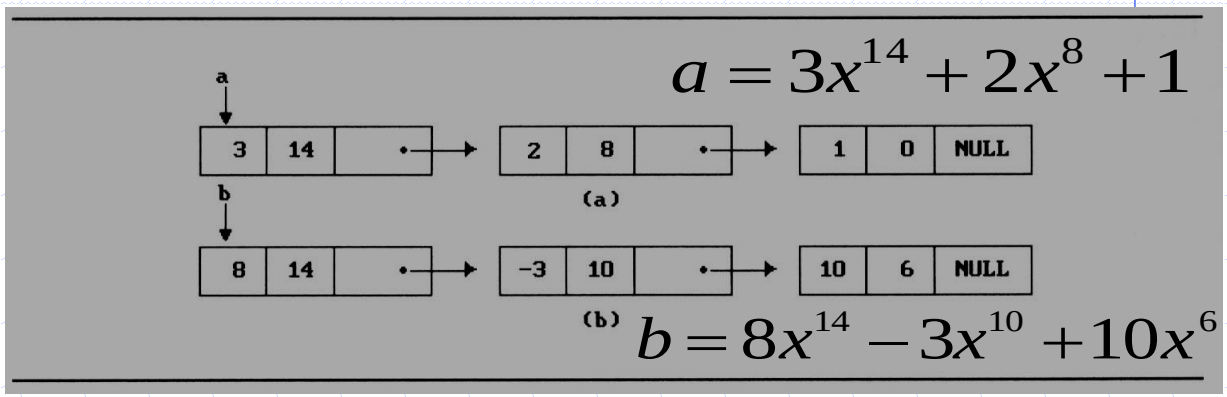
- ai ها ضرایب غیر صفر و ei ها توانهای صحیح غیر منفی هستند به قسمی که

$e_{m-1} > e_{m-2} > \ldots > e_1 > e_0 \geqq 0$.

- هر عبارت با یک گره که شامل سه فیلد ضریب، توان و اشاره گر به گره بعدی است نشان داده می شود.

چند جمله ای ها

```
typedef struct poly_node *poly_pointer;
typedef struct poly_node {
     int coef;
     int expon;
    poly_pointer link;
};
poly_pointer a,b,d;
```



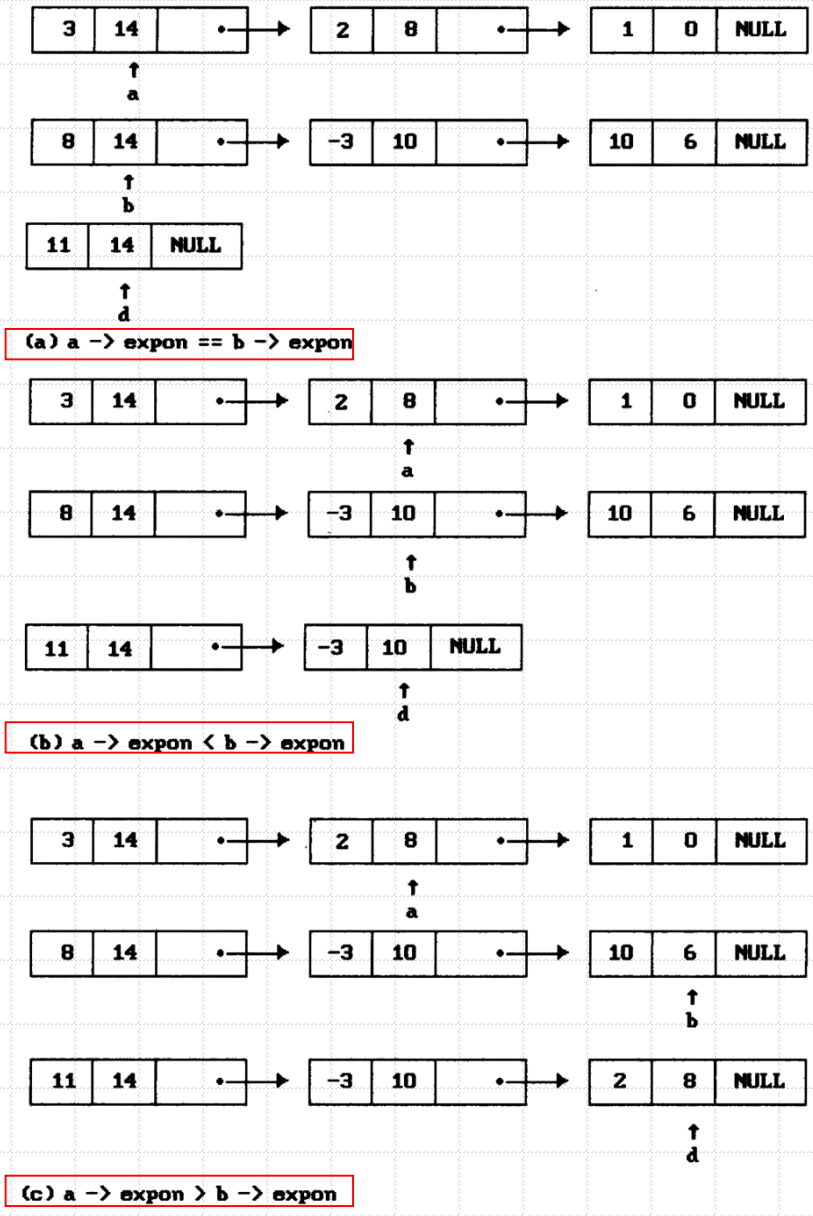
coef	expon	link

جمع چند جمله ای ها

- فرض کنیم که a و b اشاره گرهایی به ابتدای چند جمله ای ها باشند.
  - اگر توان دو چند جمله ای با هم برابر باشد، ضرایب با هم جمع می‌شوند و گره جدیدی تشکیل می‌شود، همچنین اشاره‌گرها را به گره های بعدی در a و b حرکت می دهیم.
  - اگر توان چند جمله ای a کمتر از توان متناظر در چند جمله ای b باشد، آنگاه یک جمله مشابه این جمله ایجاد و آنرا به نتیجه ، یعنی d ، اضافه می کنیم و اشاره گر را به جمله بعدی در b منتقل می کنیم.
  - اگر a -> expon > b -> expon باشد عملی مشابه را بر روی a انجام می دهیم .

جمع چند جمله ایها

$d = a+b$



جمع چند جمله ایها

```
poly_pointer padd(poly_pointer a, poly_pointer b)
{
/* return a polynomial which is the sum of a and b */
   poly_pointer front, rear, temp;
   int sum;
   rear = (poly_pointer)malloc(sizeof(poly_node));
   if (IS_FULL(rear)) {
      fprintf(stderr, "The memory is full\n");
      exit(1);
   }
   front = rear;
   while (a && b)
      switch (COMPARE(a->expon,b->expon)) {
         case -1: /* a->expon < b->expon */
               attach(b->coef,b->expon,&rear);
               b = b->link;
               break;
         case 0: /* a->expon = b->expon */
               sum = a->coef + b->coef;
               if (sum) attach(sum,a->expon,&rear);
               a = a->link;  b = b->link; break;
         case 1: /* a->expon > b->expon */
               attach(a->coef,a->expon,&rear);
               a = a->link;
      }
   /* copy rest of list a and then list b */
   for (; a; a = a->link) attach(a->coef,a->expon,&rear);
   for (; b; b = b->link) attach(b->coef,b->expon,&rear);
   rear->link = NULL;
   /* delete extra initial node */
   temp = front; front = front->link;  free(temp);
   return front;
}
```

جمع چند جمله ای ها

```
void attach(float coefficient, int exponent, poly_pointer *ptr){
/* create a new node with coef = coefficient and expon =
   exponent, attach it to the node pointed to by ptr.  Ptr is updated
   to point to this new node */
   poly_pointer temp;
   temp = (poly_pointer) malloc(sizeof(poly_node));
   /* create new node */
   if (IS_FULL(temp)) {
     fprintf(stderr, "The memory is full\n");
     exit(1);
   }
   temp->coef = coefficient;/* copy item to the new node */
   temp->expon = exponent;
   (*ptr)->link = temp;        /* attach */
   *ptr = temp;                /* move ptr to the end of the list */
}
```

جمع چند جمله ای ها الگوریتم جمع چند جمله ای با فاکتور ثابت بهینه است.

تحلیل جمع چند جمله ای ها

$$A(x)(=a_{m-1}x^{e_{m-1}}+\cdots+a_0x^{e_0})+B(x)(=b_{n-1}x^{f_{n-1}}+\cdots+b_0x^{f_0})$$

- **جمع ضرایب**

$0 \le$ additions $\le min(m, n)$

where m (n) denotes the number of terms in A (B).

- **مقایسه توان ها**

extreme case:

$e_{m-1} > f_{m-1} > e_{m-2} > f_{m-2} > \ldots > e_1 > f_1 > e_0 > f_0$

$m+n-1$ comparisons

- **ایجاد گره جدید برای d**

extreme case: maximum number of terms in d is $m+n$

$m + n$ new nodes

summary: $O(m+n)$

چند جمله ای ها

$e(x) = a(x) * b(x) + d(x)$

```
poly_pointer a, b, d, e;
...
a = read_poly();
b = read_poly();
d = read_poly();
temp = pmult(a, b);
e = padd(temp, d);
print_poly(e);
```

- read_poly()
- print_poly()
- padd()
- psub()
- pmult()

temp برای ذخیره یک مقدار میانی استفاده شده است با بازگرداندن گره های این چند جمله ای می توان از گره های آن برای نگهداری چندجمله ایهای دیگر استفاده کرد.

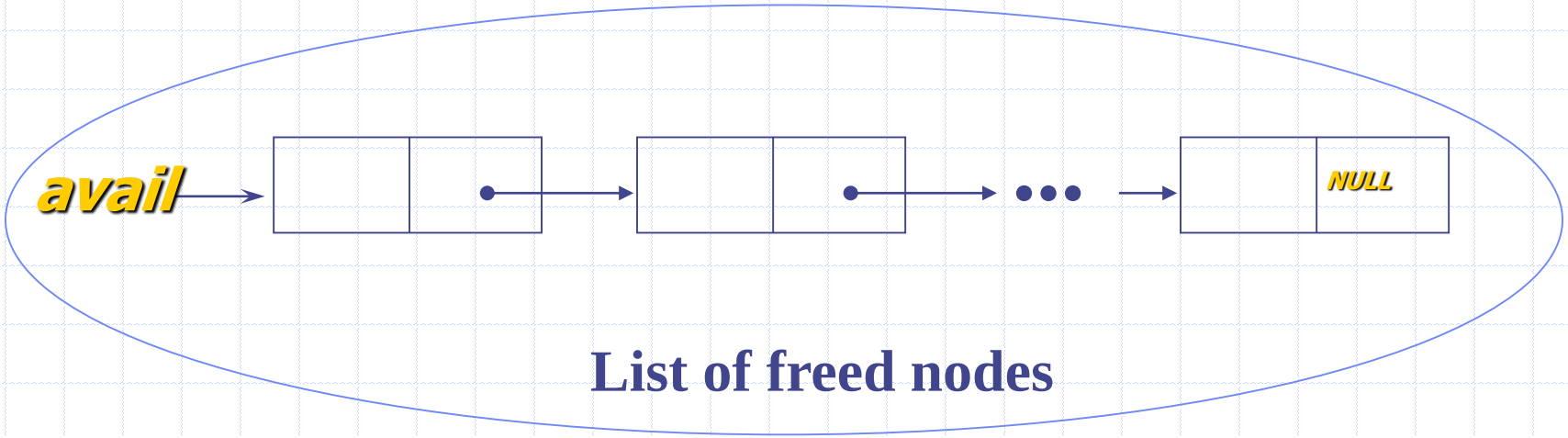
چند جمله ای ها

- حذف یک چند جمله ای

```
void erase (poly_pointer *ptr){
/* erase the polynomial pointed to by ptr */
   poly_pointer temp;
   while ( *ptr){
     temp = *ptr;
     *ptr = (*ptr) -> link;
     free(temp);
   }
}
```

لیست دایره ای

- با ذخیره گره هایی که حذف شده اند به صورت زنجیر، یک الگوریتم حذف کارا برای لیست های دایره ای به دست می آوریم.
- به جای استفاده از توابع ***malloc*** و ***free*** از تابع های ***get_node*** و ***ret_node*** استفاده می کنیم.



لیست دایره ای

- هرگاه احتیاج به گره جدیدی داشته باشیم لیست آزاد را بررسی می کنیم. اگر این لیست خالی نباشد آنگاه از یکی از گره های موجود در آن استفاده می کنیم. فقط وقتی لیست خالی است از دستور ***malloc*** برای ایجاد گره جدید استفاده می کنیم.

```
poly_pointer get_node(void)
/* provide a node for use */
{
   poly_pointer node;
   if (avail) {
      node = avail;
      avail = avail->link;
   }
   else {
      node = (poly_pointer) malloc(sizeof(poly_node));
      if (IS_FULL(node)) {
         fprintf(stderr, "The memory is full\n");
         exit(1);
      }
   }
   return node;
}
```

لیست دایره ای

- برگرداندن یک گره

avail متغیری از نوع ***listNode **** است که به زنجیری از گره هایی که حذف شده اند اشاره می کند.

این زنجیر یا لیست را لیست حافظه آزاد می نامیم.

در آغاز ***avail*** برابر ***NULL*** است.

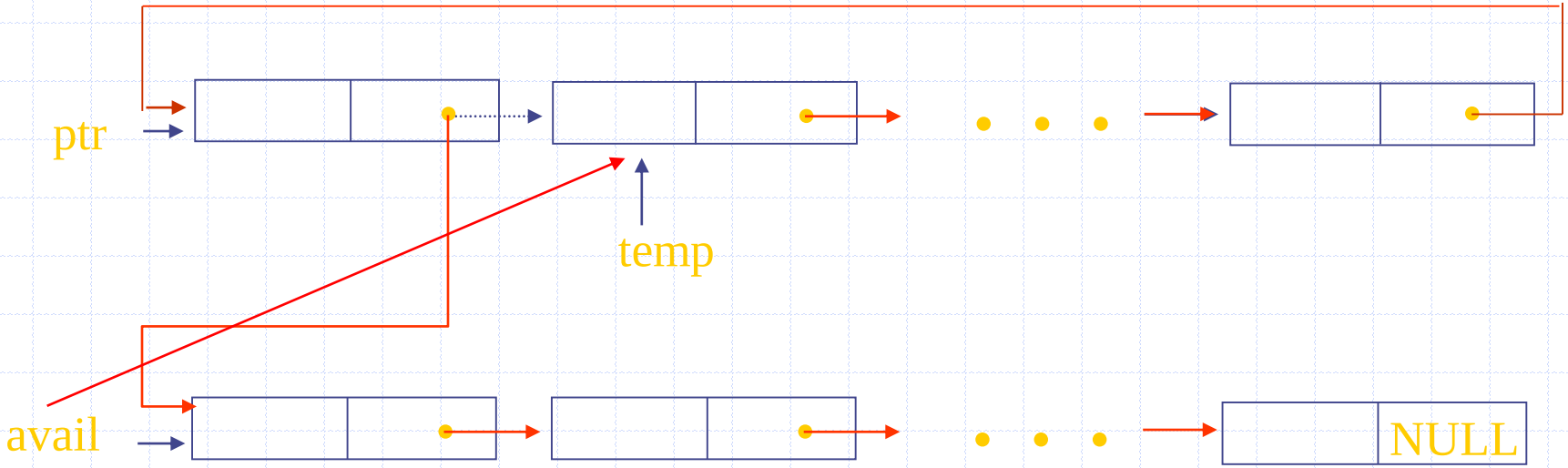
اضافه کردن ptrبه ابتدای لیست

```
void ret_node(poly_pointer ptr)
{
/* return a node to the available list */
   ptr->link = avail;
   avail = ptr;
}
```

```
void cerase(poly_pointer *ptr)
{
/* erase the circular list ptr */
   poly_pointer temp;
   if (*ptr) {
      temp = (*ptr)->link;
      (*ptr)->link = avail;
      avail = temp;
      *ptr = NULL;
   }
}
```

حذف یک لیست دایره ای

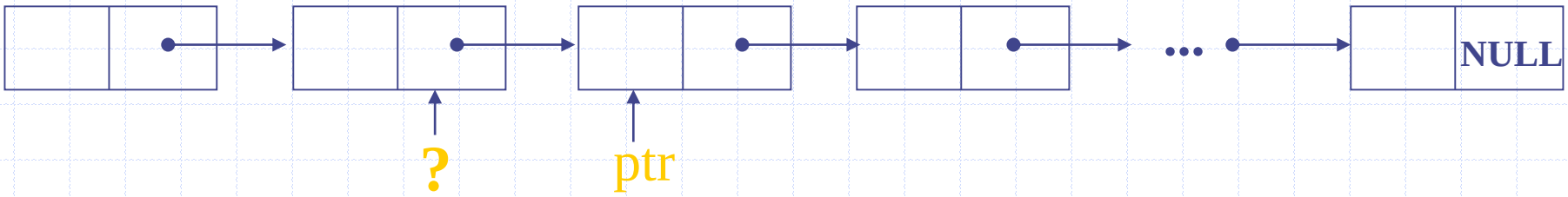
لیست دایره ای را می توان در زمان ثابت (O(1)) مستقل از تعداد گره های لیست حذف کرد.



لیست های دو پیوندی

- لیست های تک پیوندی بعضی از مشکلات را ایجاد می‌کند زیرا فقط می‌توانیم در جهت پیوندها حرکت کنیم.

مثال تنها راه یافتن گره ماقبل ptr پیمایش از ابتدای لیست می باشد. برای حذف یک گره دلخواه باید آدرس گره قبل را بدانیم.



- اگر نیازمند پیمایش لیست از هر دو جهت باشیم بهتر است از لیست دو پیوندی استفاده کنیم.

لیست دو پیوندی

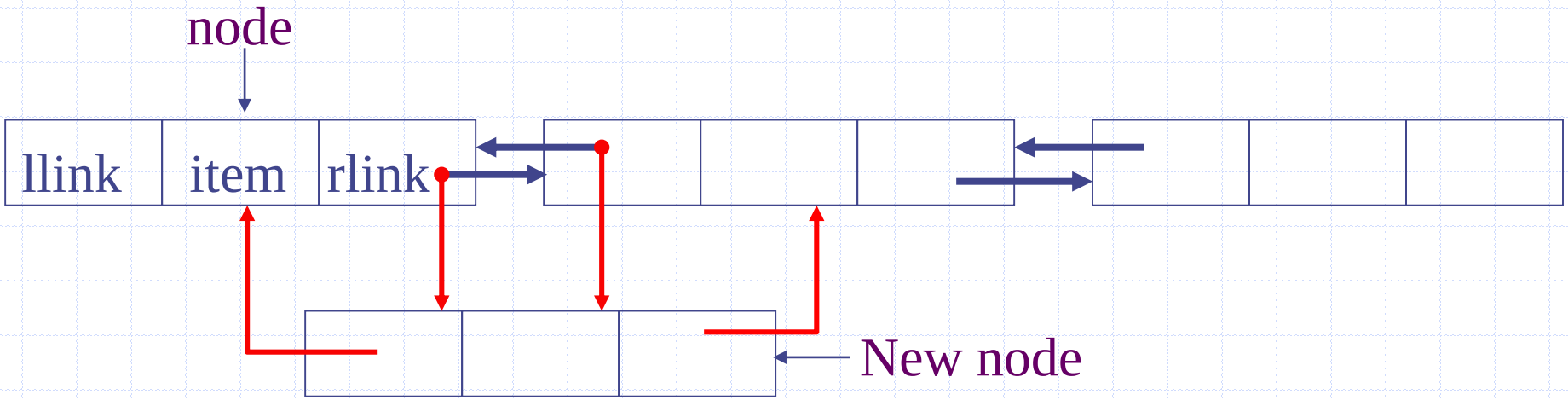
- یک گره در یک لیست دو پیوندی حداقل سه فیلد داده ای دارد:
  - فیلد data
  - فیلد llink (اشاره گر به چپ)
  - فیلد rlink (اشاره گر به راست)
- لیست دوپیوندی می تواند دایره ای باشد یا دایره ای نباشد. می تواند دارای گره سر باشد یا نباشد.

```
typedef struct node *node_pointer;
typedef struct node{
   node_pointer llink;
   element item;
   node_pointer rlink;
};
```

لیست دو پیوندی

- اضافه کردن گره در یک لیست دوپیوندی

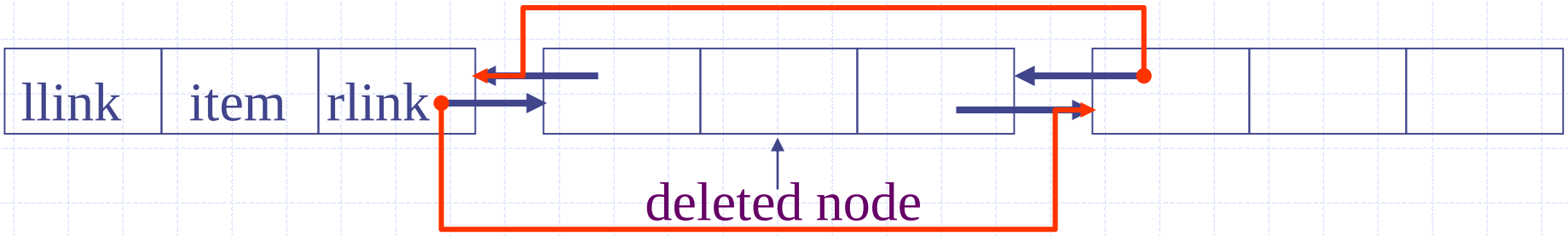
```
newnode->llink = node;
newnode->rlink = node->rlink;
node->rlink->llink = newnode;
node->rlink = newnode;
```



لیست دو پیوندی

- حذف کردن گره در یک لیست دوپیوندی

```
deleted->llink->rlink = deleted->rlink;
deleted->rlink->llink = deleted->llink;
free(deleted);
```



نمونه سوالات در لیست‌های پیوندی

- لیست یکطرفه / لیست یکطرفه حلقوی / لیست دوطرفه/ لیست دوطرفه (دوپیوندی) حلقوی
  - درج عنصر ptr در ابتدای لیست
  - حذف عنصر اول از لیست
  - درج عنصر ptr به عنوان آخرین عنصر
  - حذف عنصر آخر لیست
  - معکوس کردن لیست
  - اتصال دو لیست به هم
  - تفکیک یک لیست به دو لیست جداگانه از محل ptr
  - پیاده‌سازی پشته/صف با لیست