به نام خدا

ساختمان داده ها

جلسه یازدهم

دانشگاه صنعتی همدان

گروه مهندسی کامپیوتر

نیم سال دوم 98-1397

فصل چهارم

لیستهای پیوندی و کاربرد آنها

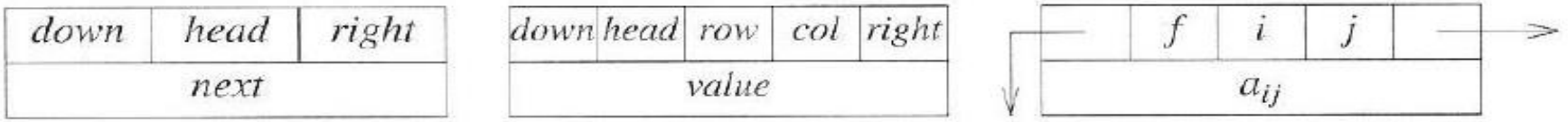
Linked Lists

نمایش ماتریسهای پراکنده با لیستهای پیوندی

- با توجه به خاصیت لیستهای پیوندی (که برای نمایش ساختارهایی که طول متغیر دارند مناسب می باشند) برای پیاده سازی ماتریسهای خلوت نیز می توان از آنها استفاده نمود. زیرا اندازه ماتریسهای خلوت با اعمالی مانند جمع و ضرب تغییر می کند.

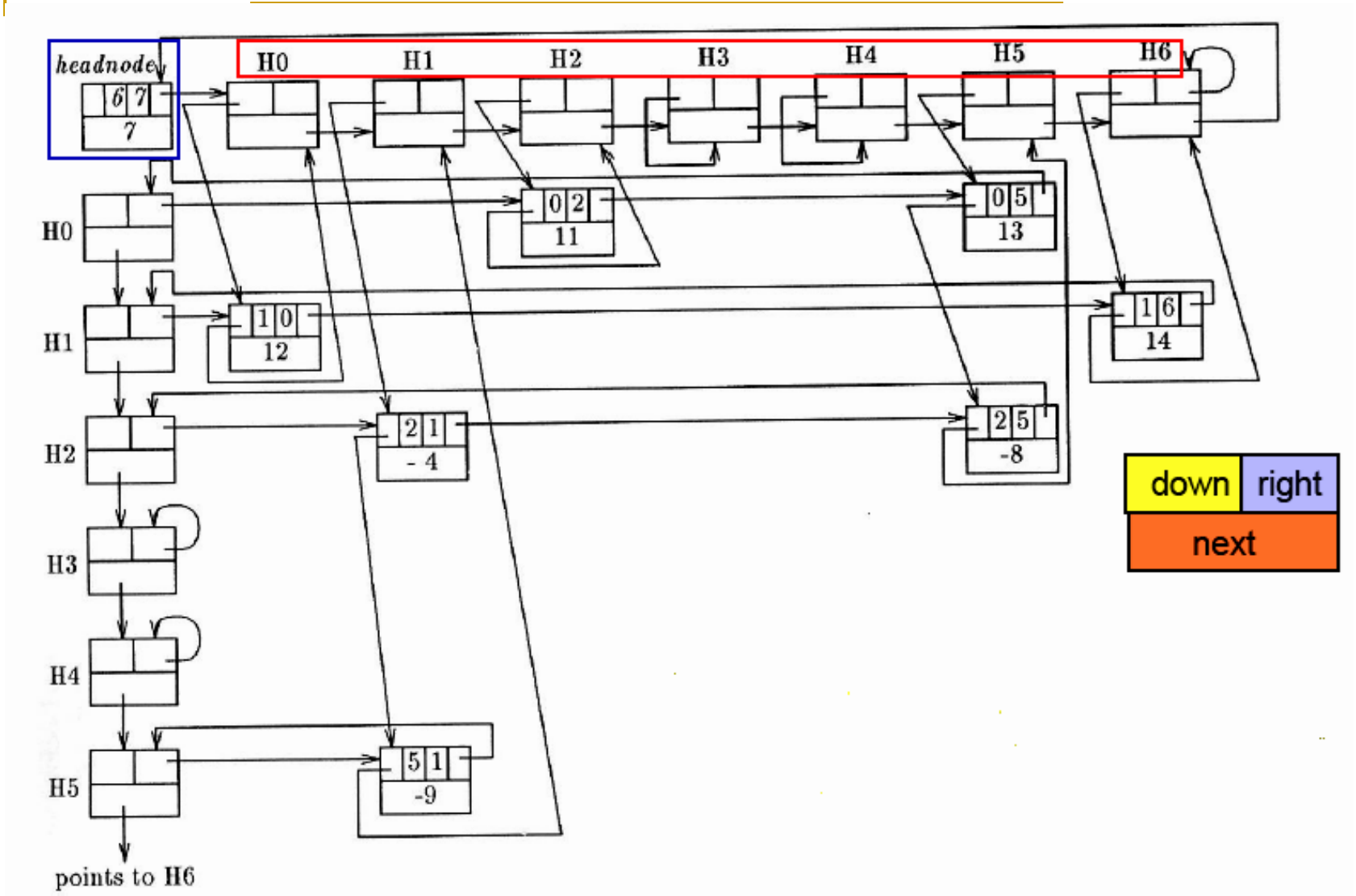
A 6x7 sparse matrix with 7 non-zero terms →

$$\begin{bmatrix} 0 & 0 & 11 & 0 & 0 & 13 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 14 \\ 0 & -4 & 0 & 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -9 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



(a) head node (b) typical node (c) set up for a_{ij}

ماتریسهای خلوت پیوندی



```
struct Triple { int value, row, col; };
class Matrix;
class MatrixNode {
  friend class Matrix;
  friend istream& operator>>(istream&, Matrix&);
  MatrixNode *down, *right;
  bool head;  // a head node or not
  union { // anonymous union
    MatrixNode *next;
    Triple triple;
  };
public:
  MatrixNode(bool b, Triple *t) :head(b) { // ctor
    if(b) { right = down = next = this; }
    else triple = *t;
  }
};

typedef MatrixNode *MatrixNodePtr;
```

```
class Matrix {
  friend istream& operator>>(istream&, Matrix&);
  MatrixNode *headnode;
public:
  ~Matrix();  // dtor
};
```

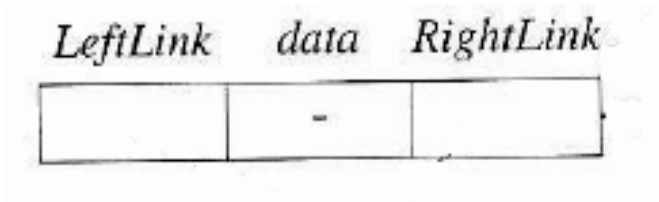
خواندن یک ماتریس خلوت

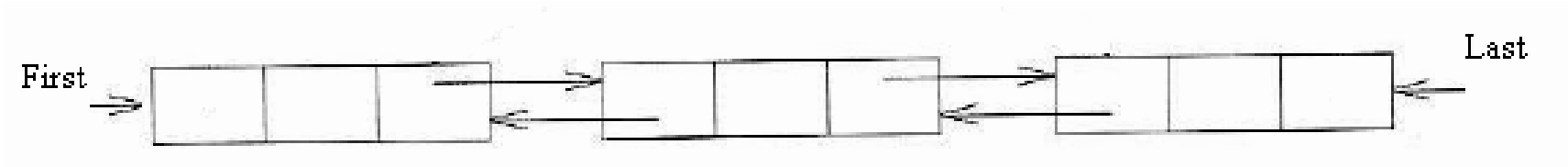
```
istream& operator>>(istream& is, Matrix& matrix)
{   Triple s; int p;     is >> s.row >> s.col >> s.value;
if (s.row > s.col) p = s.row; else p = s.col;
     matrix.headnode = new MatrixNode(FALSE, &s);
if (p==0) {matrix.headnode->right = matrix.headnode; return is;}
       MatrixNodePtr *head = new MatrixNodePtr[p];
for (int i = 0; i < p; i++)    head[i]  = new MatrixNode(TRUE, 0);

    int CurrentRow = 0; MatrixNode *last = head[0];
for (i = 0; i < s.value; i++)
    {          Triple t;     is >> t.row >> t.col >> t.value;
               if (t.row > CurrentRow) {
                      last->right = head[CurrentRow];
                       CurrentRow = t.row;
                      last = head[CurrentRow]; }
              last = last->right = new MatrixNode(FALSE, &t);
             head[t.col]->next = head[t.col]->next->down = last;
 }
    last->right = head[CurrentRow];
    for (i = 0; i < s.col; i++) head[i]->next->down = head[i];
     for (i = 0; i < p-1 ; i++) head[i]->next = head[i+1];
    head[p-1]->next = matrix.headnode;
    matrix.headnode->right =  head[0];
    delete [] head;
    return is;
}
```

لیستهای دو پیوندی

- لیستهای تک پیوندی فقط در یک جهت می توانند پیمایش شوند
  - اضافه کردن یک گره جدید به قبل از یک گره کار مشکلی است.
  - حذف یک گره هم مقداری پیچیده می شود.
- یک راه حل برای این مشکلات دو پیوندی کردن لیستها است. در این صورت حرکت در دو جهت امکان پذیر می باشد.





- در این لیست برای هر گره دلخواه p داریم:

P==p->llink->rlink==p->rlink->llink

کلاس لیستهای دو پیوندی

```
class DblListNode {
friend class DblList;
private:
    int data;
    DblListNode *llink, *rlink;
};

class DblList {
public:
    void Insert(DblListNode*, DblListNode*);
    void Delete(DblListNode*);
private:
    DblListNode *first;  // points to head node
};
```

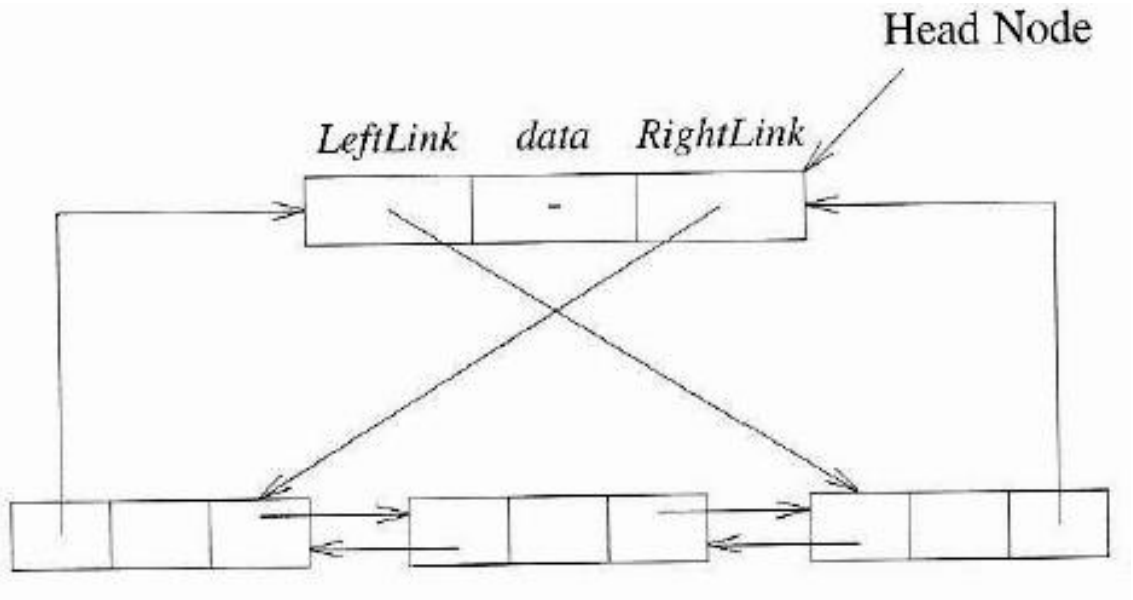
توابع حذف و اضافه کردن

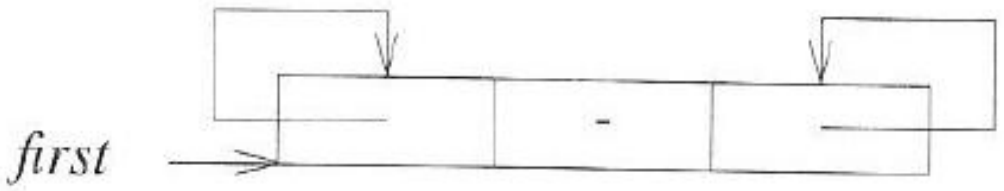
```
void DblList::Insert(DblListNode *p, DblListNode *x)
// insert node @p@ to the right of node @x@
{
    p->llink = x; p->rlink = x->rlink;
    x->rlink->llink = p; x->rlink = p;
}

void DblList::Delete(DblListNode *x)
{
      x->llink->rlink = x->rlink;
      x->rlink->llink = x->llink;
      delete x;

}
```

لیستهای دو پیوندی حلقوی





```
DblList::DblList() { // ctor
   first = new DblListNode;  // allocate the head node
   first->llink = first;
   first->rlink = first;
}
void DblList::InsertR(DblListNode *p, DblListNode *x) {
// insert node p to the right of node x
   p->llink = x; p->rlink = x->rlink;
   x->rlink->llink = p; x->rlink = p;
}
void DblList::InsertL(DblListNode *p, DblListNode *x) {
// insert node p to the left of node x
   p->rlink = x; p->llink = x->llink;
   x->llink->rlink = p; x->llink = p;
}
```

```
void DblList::Delete(DblListNode *x)
{
    if (x == first) cerr << "Deletion of head node not permitted" <<
endl;
    else {
      x->llink->rlink = x->rlink;
      x->rlink->llink = x->llink;
      delete x;
    }
}
```