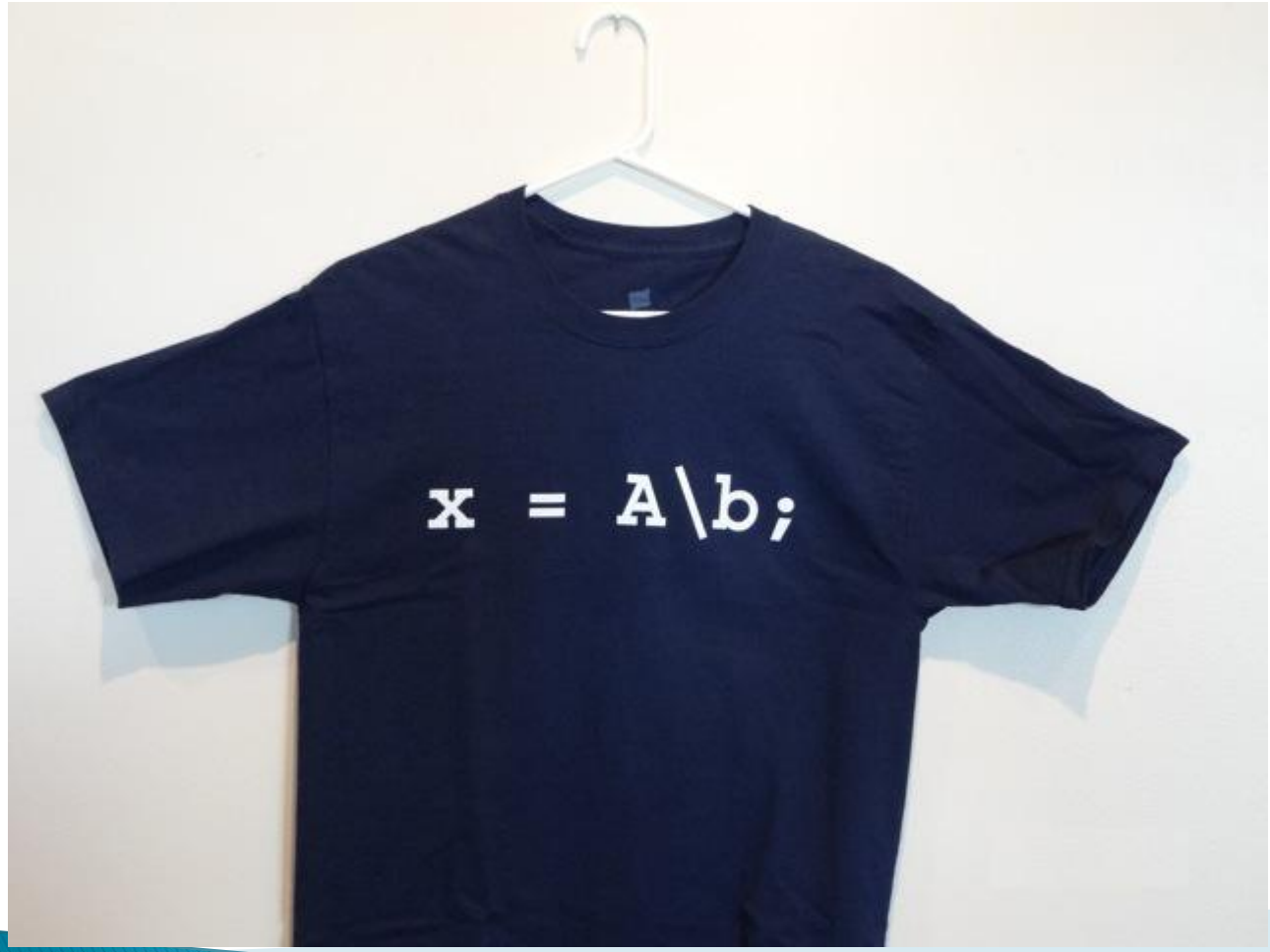
高等数值算法与应用(2a)

– 稀疏矩阵直接解法 (补充内容1) –

喻文健

from T.A. Davis's book: *Direct Methods for Sparse Linear Systems* , and Y. Saad and J.R. Gilbert's course slides



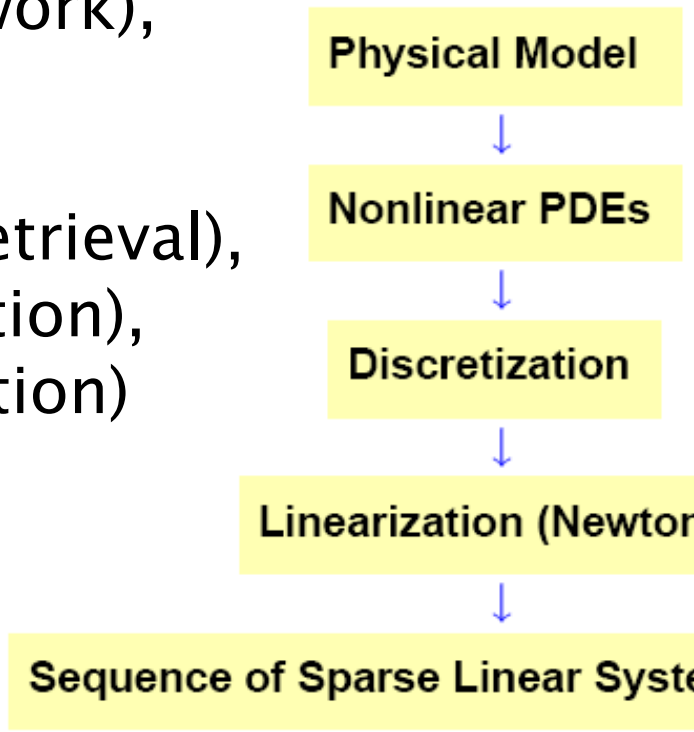
\ --"It's amazing how much numerical linear algebra is contained in that single character"

Outline

- 大规模稀疏矩阵概述
- 矩阵填入元与图
- 稀疏下三角阵的Lsolve算法
- 稀疏矩阵LU分解与矩阵重排序
- ~~稀疏对称正定阵的Cholesky分解~~

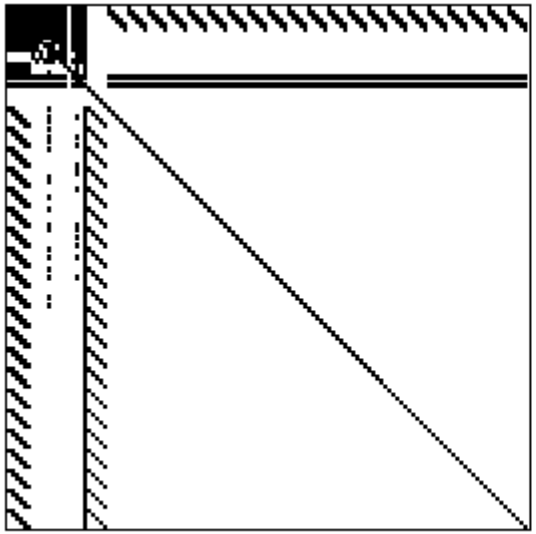
大规模稀疏矩阵

- 稀疏矩阵的应用领域
  - 结构工程(structural engineering),
    计算流体力学(computational fluid dynamics),
    油藏数值仿真(reservoir simulation),
    电力网络(electrical network),
    优化问题(optimation),
    Google PageRank,
    信息抽取(information retrieval),
    电路仿真(circuit simulation),
    器件仿真(device simulation)

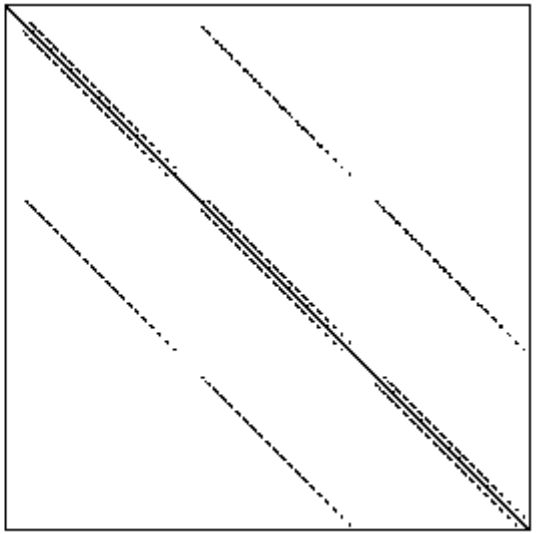


有限差分
有限元
边界元

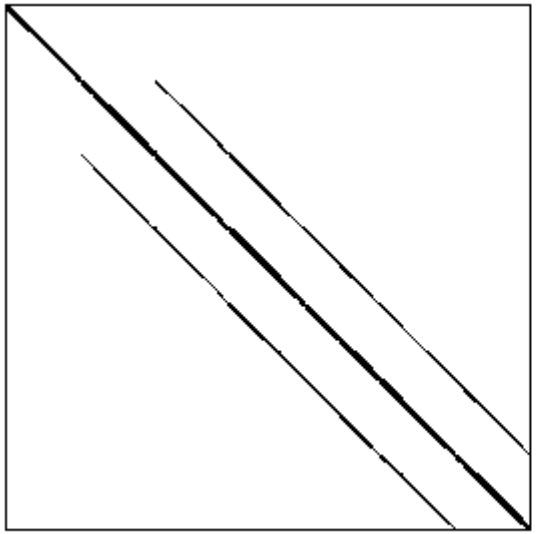
Nonzero patterns of a few sparse matrices



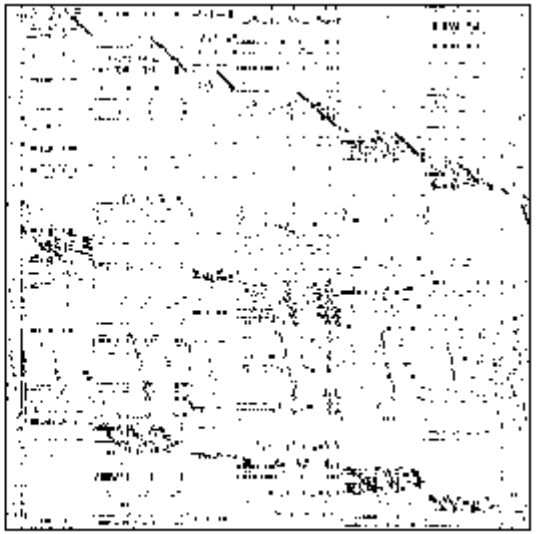
ARC130



SHERMAN5



PORES3

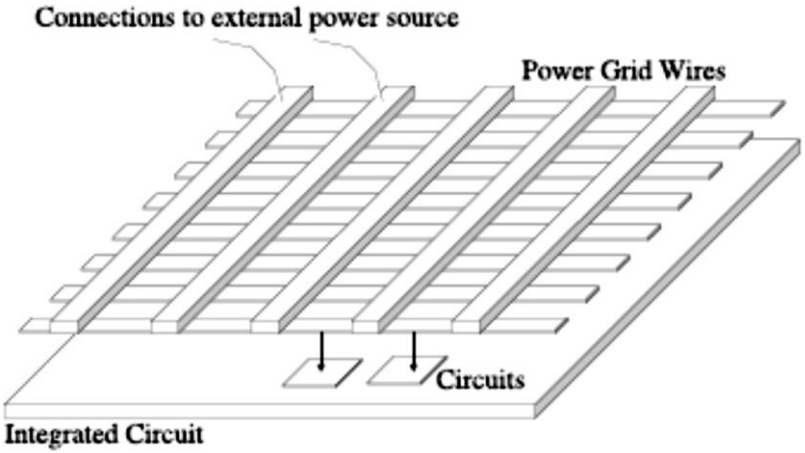


BP_1000

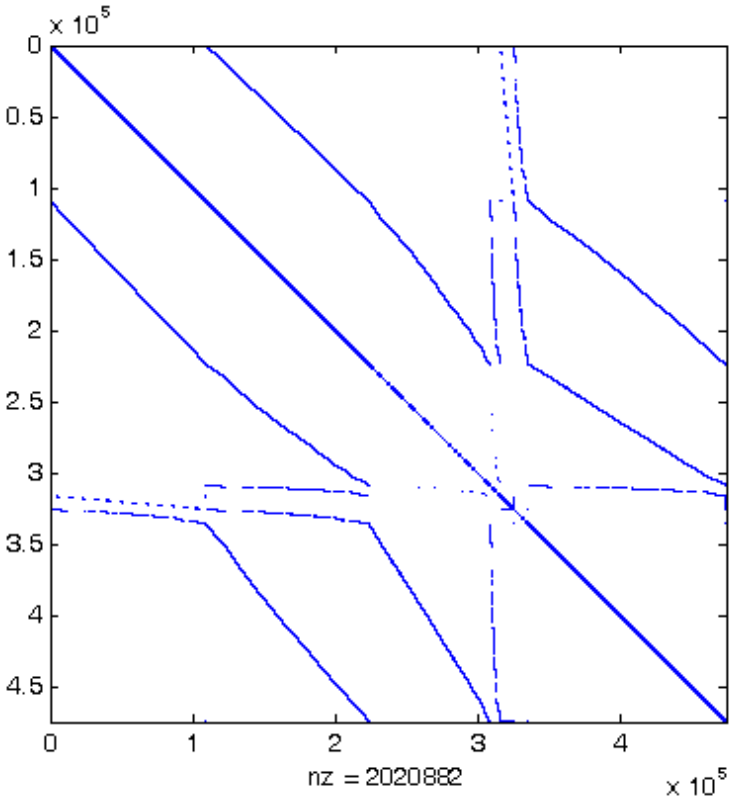
非结构化的

一个实际例子

- 集成电路供电网仿真
  - DC分析：求解电阻网络电路中的节点电压
  - 一个来自IBM公司的例子
    - n= 474,524
    - nnz= 2,020,882
    - 在Matlab中, 占26M内存 ？
    - “\”求解(矩阵时对称正定矩阵), t= 15s
    - “chol”: Cholesky分解系数矩阵 **内存不够用！**？



集成电路供电线网的一部分



稀疏矩阵的运算

- **Recall:**

```
typedef struct SpaFmt {
/*-----------------------------------------
| C-style CSR format - used internally
| for all matrices in CSR format
|----------------------------------------*/
  int n;
  int *nzcount;  /* length of each row */
  int **ja;     /*  to store column indices  */
  double **ma;  /*  to store nonzero entries */
} CsMat, *csptr;
```

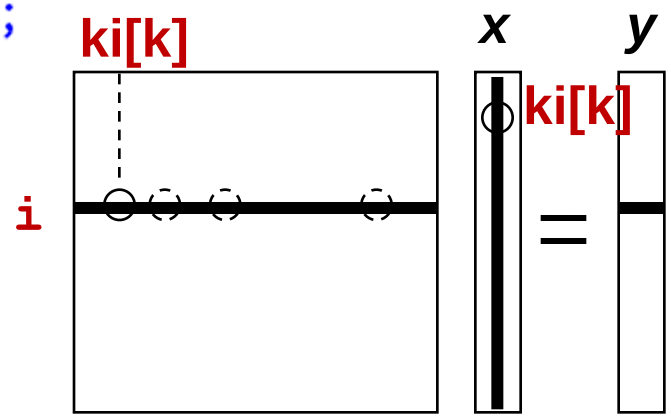
- **Can store rows of a matrix (CSR)**
- **or its columns (CSC)**
- **How to perform the operation $y = A * x$ in each case?**

矩阵与向量的乘法

Matvec – row version

```
void matvec( csptr mata, double *x, double *y )
{
    int i, k, *ki;
    double *kr;
    for (i=0; i<mata->n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];      // 第i行的非零元素值
        ki = mata->ja[i];      // 第i行非零元的列号
        for (k=0; k<mata->nzcount[i]; k++)
            y[i] += kr[k] * x[ki[k]];
    }
    return;
```

计算y=Ax (CSR存储格式)



- Uses sparse dot products

计算量：nnz(A)次乘法

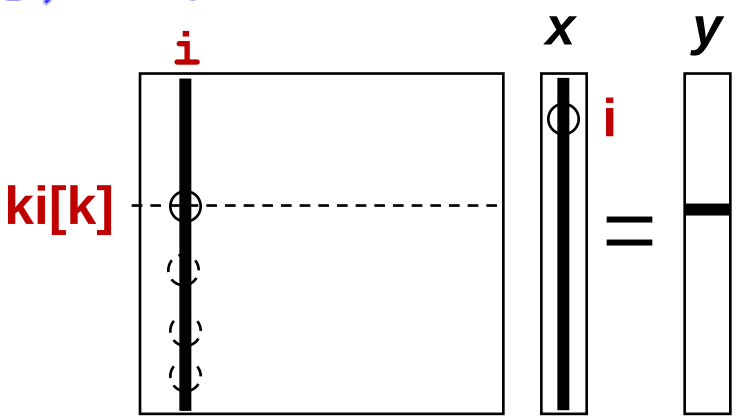
Matvec – Column version

```
void matvecC( csptr mata, double *x, double *y )
{
  int n = mata->n, i, k, *ki;
  double *kr;
  for (i=0; i<n; i++)
    y[i] = 0.0;
  for (i=0; i<n; i++) {
    kr = mata->ma[i];          // 第i列的非零元素值
    ki = mata->ja[i];          // 第i列非零元的行号
    for (k=0; k<mata->nzcount[i]; k++)
      y[ki[k]] += kr[k] * x[i];
  }
  return;
```

计算y=Ax (CSC存储格式)

- Uses sparse SAXPY
  (scalar * x + y)

计算量？若考虑x稀疏呢？



(类似于解Ux=b的两种回代算法)

矩阵填入元与图

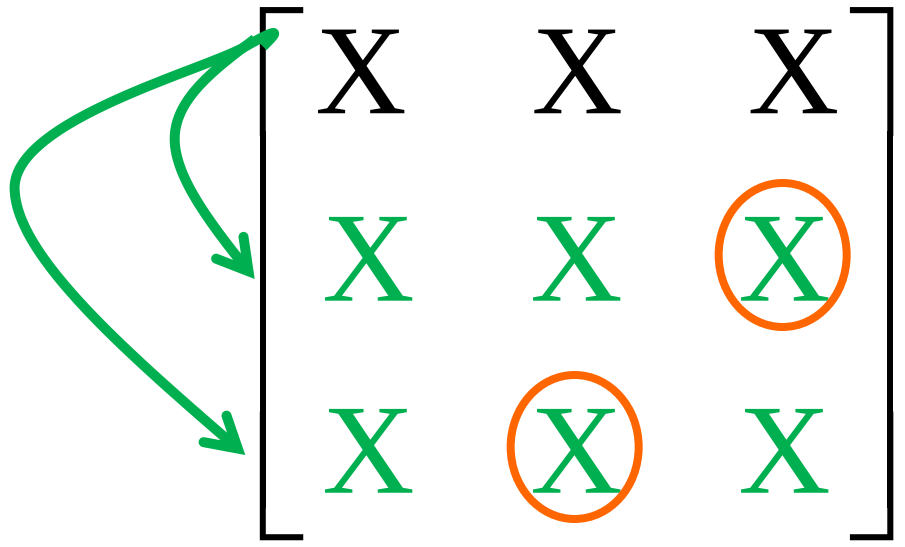
- 稀疏矩阵的填入元
- 稀疏矩阵的图表示

稀疏矩阵的“填入元” – 例1

系数矩阵的非零元结构

$$\begin{bmatrix} X & X & X \\ X & X & 0 \\ X & 0 & X \end{bmatrix}$$

一次高斯消去步骤后



X—非零元

X —填入元(Fill-in)

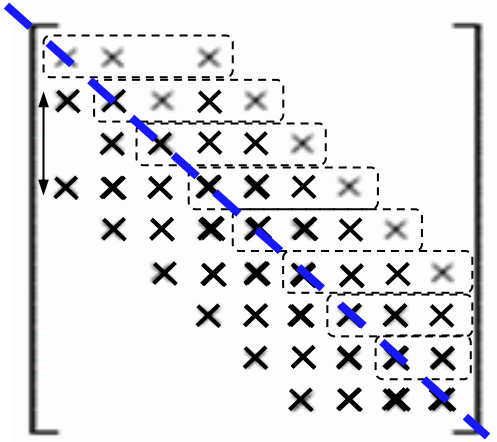
Cholesky分解的填入元情况是一样的!

稀疏矩阵的“填入元” – 更多例子

“填入元”的传播

$$\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ 0 & X & X & X \\ 0 & X & X & X \end{bmatrix}$$

条带状矩阵



仍在原始带宽内

若部分选主元，填入元使带宽增大(最多$2\beta+1$)

1.满足何条件时(i, j)处会发生fill-in?

2.证明: 对”结构对称”矩阵，发生fill-in的位置也是对称的

填入元给方程求解(LU分解)带来多少麻烦？！

稀疏阵的“图”表示

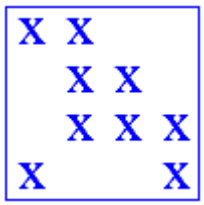
- Graph theory is a fundamental tool in sparse matrix techniques.

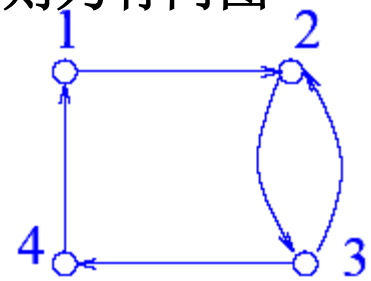
Graph $G = (V, E)$ of an $n \times n$ matrix A defined by

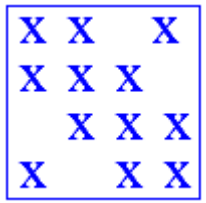
Vertices $V = \{1, 2, ..., N\}$.

Edges $E = \{(i, j) | a_{ij} \neq 0\}$. 或$a_{ji} \neq 0$ (一般不含矩阵对角元的信息)

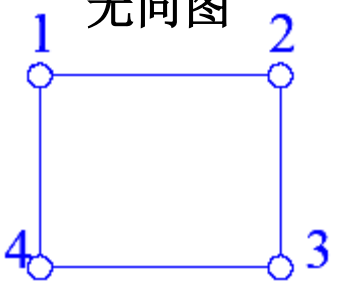
- Graph is undirected if matrix has symmetric structure: $a_{ij} \neq 0$ iff $a_{ji} \neq 0$. 否则为有向图







无向图



图的计算机表示：邻接矩阵 (无数值)

图(V, E)的一些概念

- 若$(u, v) \in E$： v与u相邻 adjacent to
- 对于有向图，(u, v)是从顶点u的出边，是到v的入边 (outgoing/incoming edge)
- $Adj(i) = \{j \mid j \text{ adjacent to } i\}$： i的邻点集合
- 顶点v的度(degree): 连到v的边数(有向图, 出度/入度)
- $|Adj(i)| = deg(i)$ 出度
- 若$V' \subseteq V$且$E' \subseteq E$， $G' = (V', E')$为$G = (V, E)$的子图
- 路径(path): 一组有序的点$w_0, w_1, \ldots, w_k$，满足$(w_i, w_{i+1}) \in E$, $i = 0, \ldots, k-1$. 路径的长度=边的数目.
- 封闭的路径为环(circle)，即满足$w_k = w_0$的路径
- 无环图acyclic：森林forest, 有向无环图DAG. …

稀疏下三角阵的Lsolve算法

- 求解稀疏下三角线性方程组Lsolve算法

稀疏下三角阵

- 没有选主元的问题，也没有fill-in(对角元≠0)
- 设用CSC存储格式，应使用“按列运算”算法

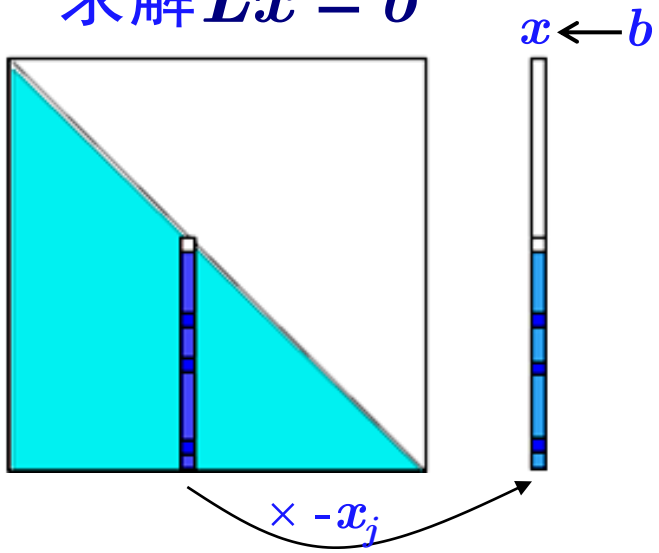
矩阵L稠密时的算法

$x = b$
for $j = 1{:}n$
  $x_j = x_j / l_{jj}$
  for $i = j{+}1{:}n$, and $l_{ij} \neq 0$
    $x_i = x_i - l_{ij} x_j$
  end
end

若L稀疏呢？

遍历CSC格式的矩阵第j列

求解$Lx = b$



浮点乘法次数为nnz(L)

没考虑b可能稀疏的情况，极端例子为 $b = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$

稀疏下三角阵

- 不失一般性，设L为单位下三角阵，考虑b可能稀疏
- b稀疏意味着x_j可能为0

$x = b$

for $j = 1{:}n$

If $x_j \neq 0$ (单位下三角)

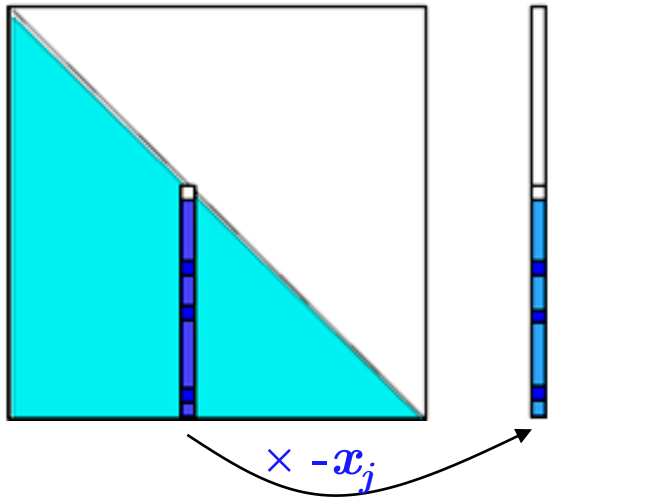
解出了x_j for $i = j{+}1{:}n$, and $l_{ij} \neq 0$

还未解出的 $x_i = x_i - l_{ij}x_j$

end

end

求解$Lx = b$



增加n次判断，避免0的运算

(可能判断太多，看极端例子 $b = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$)

for each $j \in \mathcal{X}$ do

按什么顺序取j？

若$l_{ij} \neq 0$，在$\mathcal{X}$中须先取j，再取i

若先知道x的非零元分布$\mathcal{X}$？

稀疏下三角阵(右端项稀疏) — 求x

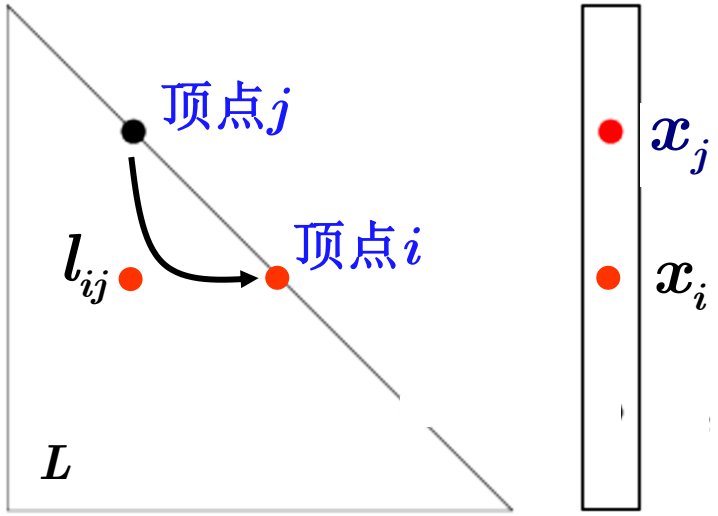
- 什么条件使 $x_i \neq 0$？ 解 $Lx = b$

$G(L)$是个DAG!
(边总是从编号小的点指向大的)

集合x中顶点顺序须按图的拓扑顺序!

$$x_i = b_i - \sum_{j=1}^{i-1} l_{ij} x_j$$

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$
- let $G(L)$ have an edge $j \rightarrow i$ if $l_{ij} \neq 0$
- let $\mathcal{B} = \{i \mid b_i \neq 0\}$ and $\mathcal{X} = \{i \mid x_i \neq 0\}$
- then $\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$

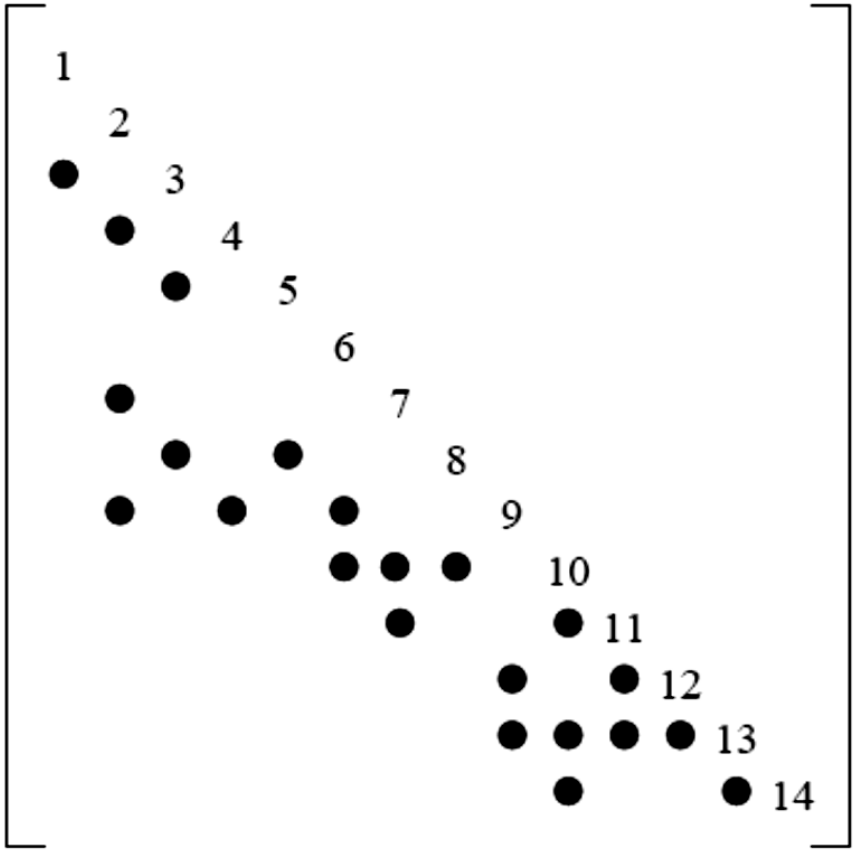


适合CSC格式, 便于找邻点

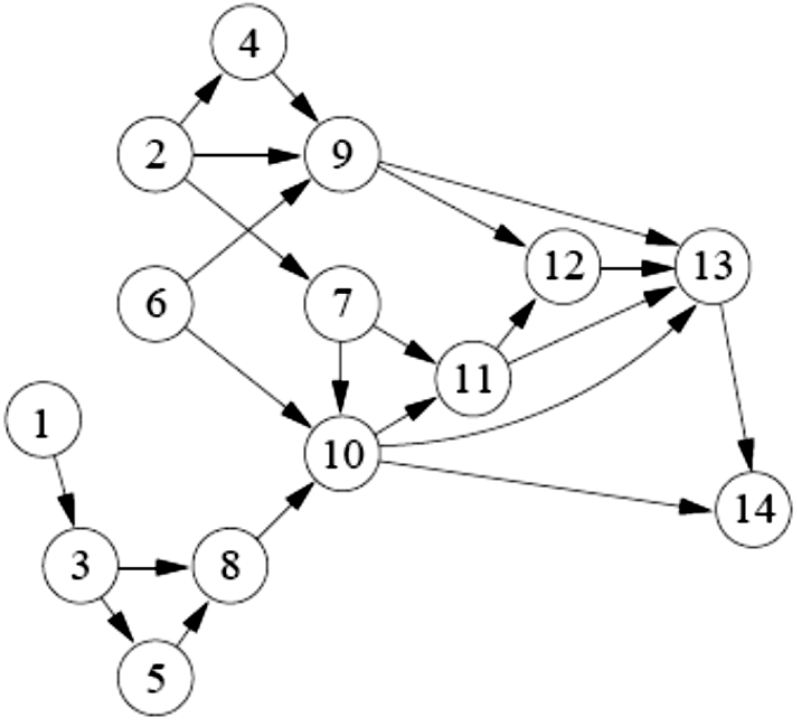
集合$\mathcal{B}$的到达集：从集合$\mathcal{B}$对应点出发的路径上的点

根据稀疏矩阵L的图求x (x的非零元位置编号集合)

Lower triangular matrix L

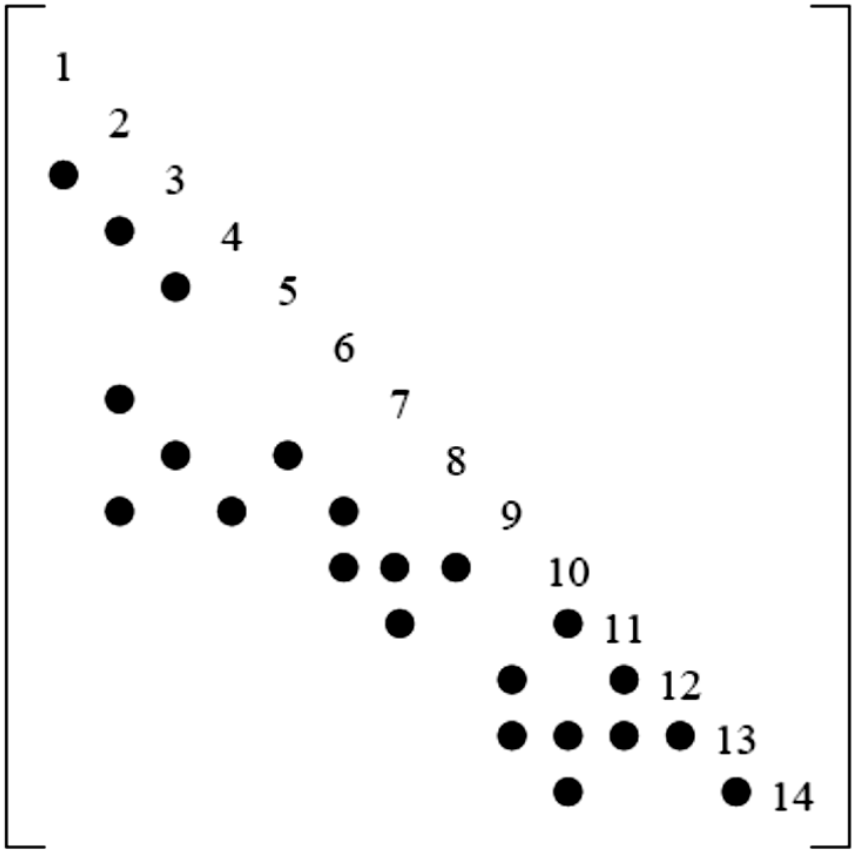


Graph G_L



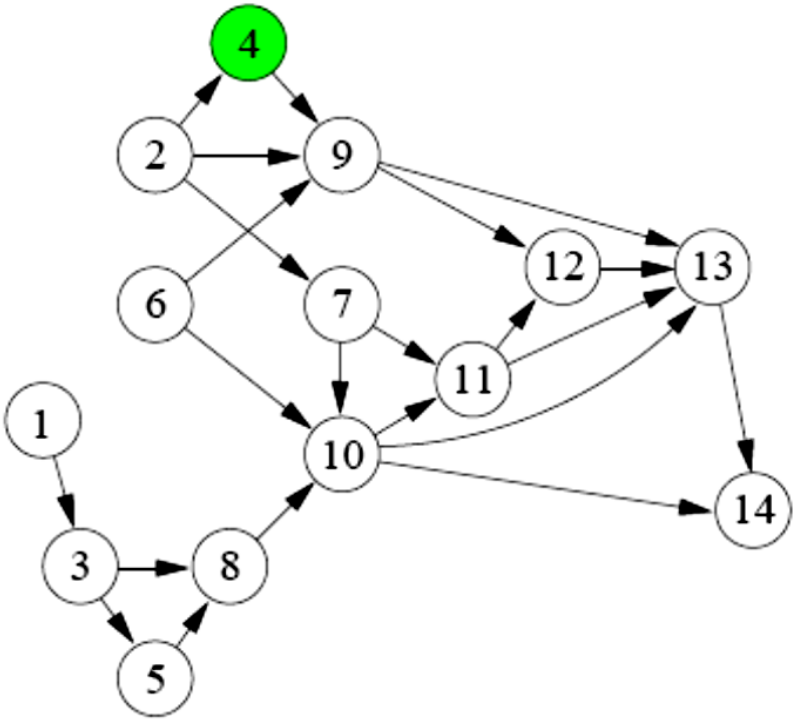
根据稀疏矩阵L的图求x (x的非零元位置编号集合)

Lower triangular matrix L



If $\mathcal{B} = \{4\}$

Graph G_L

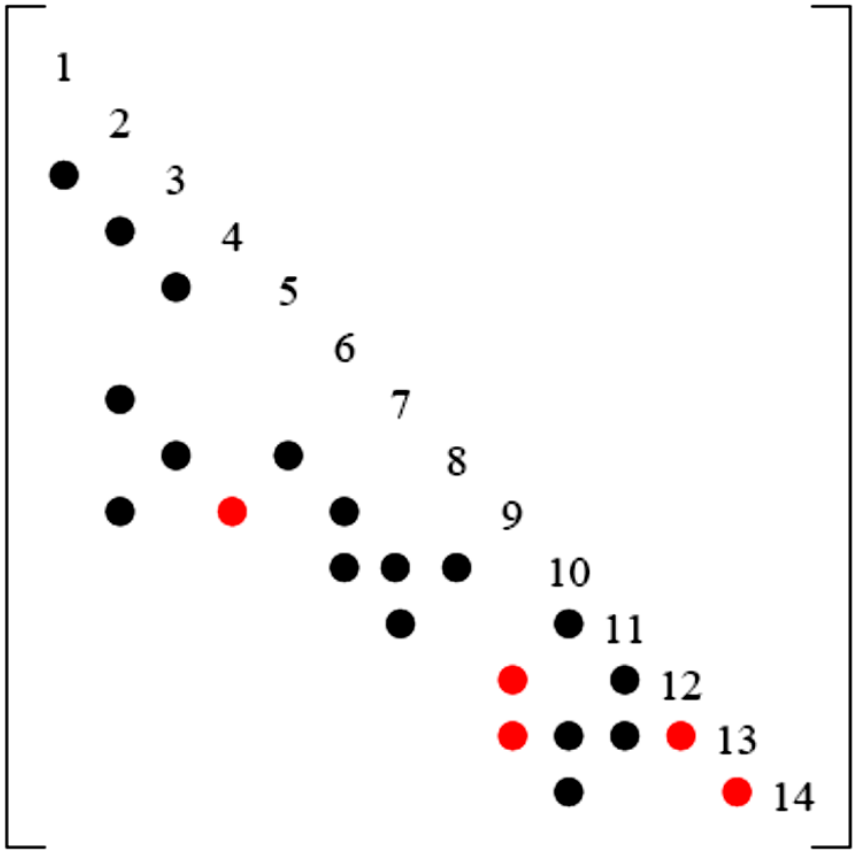


$\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$

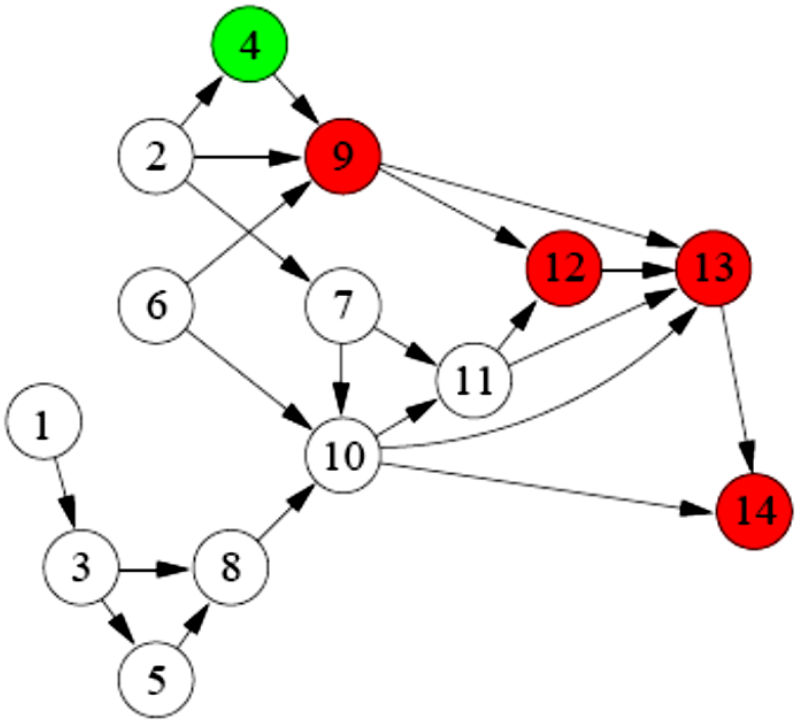
从4出发进行遍历(稀疏矩阵的列)

根据稀疏矩阵L的图求$\mathcal{X}$ (x的非零元位置编号集合)

Lower triangular matrix L



Graph G_L



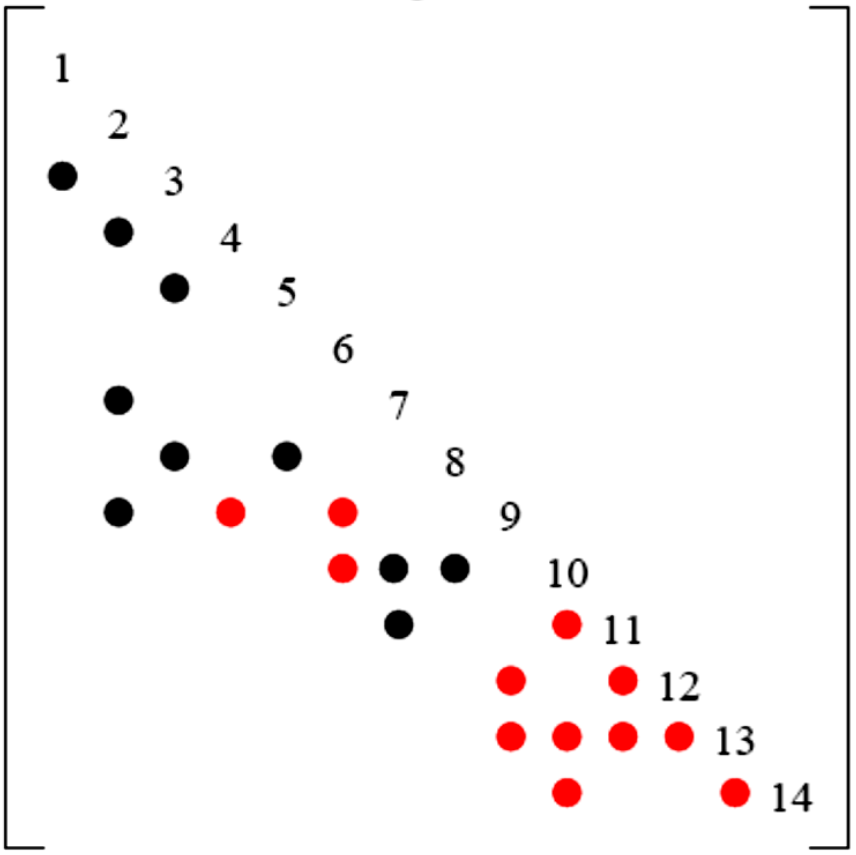
$\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$

If $\mathcal{B} = \{4\}$
then $\mathcal{X} = \{4, 9, 12, 13, 14\}$

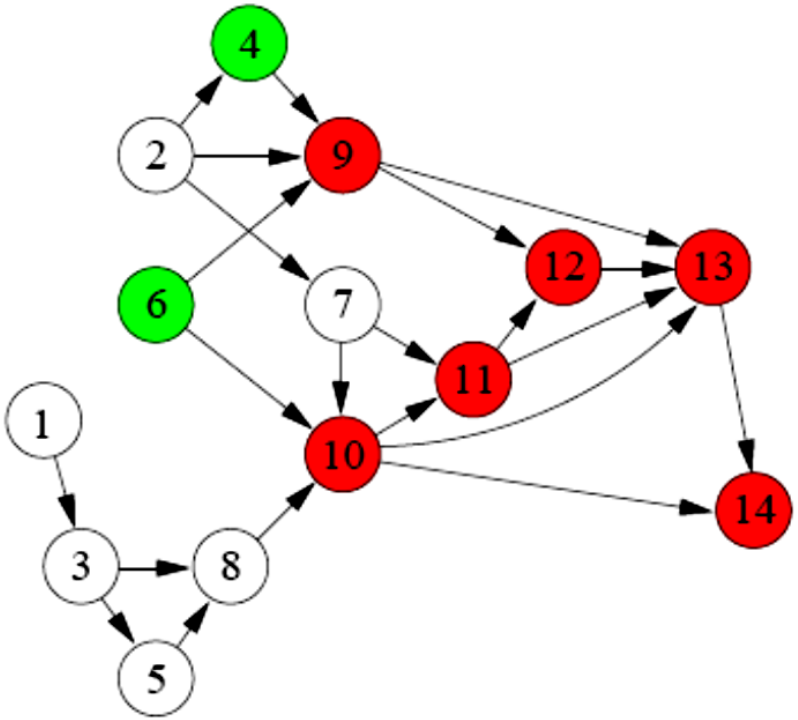
从4出发进行遍历(稀疏矩阵的列)
怎么使$\mathcal{X}$中元素按拓扑顺序?
深度优先+后序+栈

根据稀疏矩阵L的图求$\mathcal{X}$ (x的非零元位置编号集合)

Lower triangular matrix L



Graph G_L



If $\mathcal{B} = \{4, 6\}$ ← 处理顺序可任意

then $\mathcal{X} = \{6, 10, 11, 4, 9, 12, 13, 14\}$

深度优先+后序+栈

出栈顺序即$\mathcal{X}$的拓扑顺序

解稀疏下三角方程的Lsolve算法

function x = lsolve(L,b)　　求解单位下三角方程 $Lx = b$

$\mathcal{X} = \text{Reach}(L, \mathcal{B})$

x = b

for each j in $\mathcal{X}$　　(按节点的拓扑顺序)

x(j+1:n) = x(j+1:n) - L(j+1:n,j) * x(j)

function $\mathcal{X} = \text{Reach}(\text{L}, \mathcal{B})$

for each i in $\mathcal{B}$ do

if (node i is unmarked) dfs(i)

function dfs(j)　复杂度为O($|\mathcal{B}|$+e), e为边数

mark node j

for each i in $\mathcal{L}_j$ do　　j的邻点集合

if (node i is unmarked) dfs(i)

push j onto stack for $\mathcal{X}$　　(DFS后序遍历入栈)

计算复杂度由O(n+f)降为$O(|\mathcal{B}|+f)$, f为实际浮点运算次数(不含与0运算), 可能比n, nnz(L)小得多

Lsolve算法的原理程序

```
function x= lsolve(L, b)
% L, b为稀疏矩阵, 向量
n= size(L,1);
marked= zeros(n, 1);   //节点标记
stack= zeros(n, 1);    //栈
ptr_s= 0;              //栈顶指针
bb= find(b);           //b中非零元位置
reach(bb);             //结果存于stack
x= zeros(n, 1);
x(bb)= b(bb);
for i=ptr_s:-1:1,      //按出栈顺序
   j=stack(i);
   x(j+1:n)=x(j+1:n)-L(j+1:n, j)*x(j);
end
```

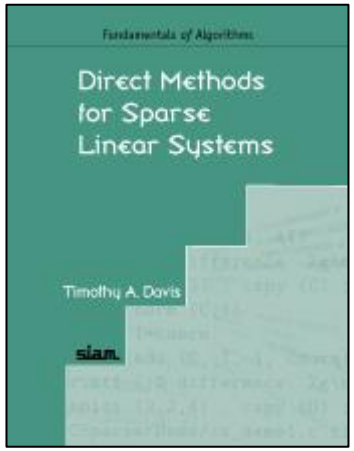
实际编程要解除递归!

```
function reach(bb)
for i=1:length(bb),      //bb为向量
   ii= bb(i);
   if ~marked(ii),
      dfs(ii);           //深度优先遍历
   end
end
```

```
function dfs(j)
marked(j)=1;
adj=find(L(:, j));       //找邻节点
for i=1:length(adj),
   if ~marked(adj(i)),
      dfs(adj(i));       //递归调用
   end
end
ptr_s= ptr_s+1;          //入栈
stack(ptr_s)=j;
```

解稀疏下三角方程的Lsolve算法

- C语言程序见T. Davis的Csparse程序包
  - http://www.cise.ufl.edu/research/sparse/CSparse/
  - *Direct Methods for Sparse Linear Systems*, Chap. 3
  - cs_spsolve.c, cs_dfs.c (非递归的DFS遍历)
- 小结
  - 解稀疏下三角方程: $\boldsymbol{Lx = b}$
  - 最低计算复杂度:$O(|\mathcal{B}|+f)$, $|\mathcal{B}|$为向量b中非零元数目, f为排除与0运算的浮点运算次数
  - 对一般稀疏矩阵进行Cholesky分解、LU分解，可转化为多次求解稀疏下三角方程： Lsolve算法特别有用
  - *(算法中的额外开销被摊薄)*



稀疏矩阵LU分解与矩阵重排序

- LU分解的Left-looking/GPLU算法
  减少填入元的重排序技术
  同时进行重排序与数值选主元
  Matlab相关命令与UMFPACK

一般稀疏阵的LU分解

- LU分解的Left-looking算法 (先考虑不选主元)

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & l_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & u_{12} & U_{13} \\ & u_{22} & u_{23} \\ & & U_{33} \end{bmatrix} = \begin{bmatrix} A_{11} & a_{12} & A_{13} \\ a_{21} & a_{22} & a_{23} \\ A_{31} & a_{32} & A_{33} \end{bmatrix}$$

  - 根据L, U的前k-1列算它们的第k列

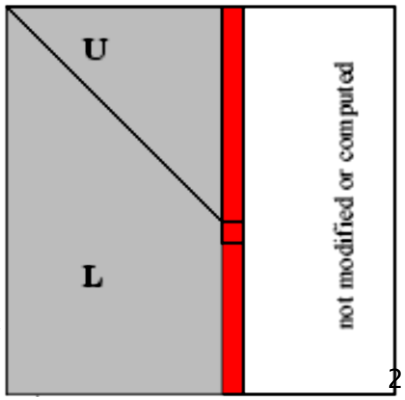
$$\begin{cases} L_{11}u_{12} = a_{12} \\ l_{21}u_{12} + u_{22} = a_{22} \\ L_{31}u_{12} + \underline{l_{32}u_{22}} = a_{32} \end{cases}$$

写成一个下三角方程

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \underline{x_3} \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$

  - 则 $u_{12} = x_1,\ u_{22} = x_2,\ l_{32} = x_3 / u_{22}$
  - 注意: 算第1列时只求x_2, x_3; x_3是第k列消去前的值
  - 利用Lsolve算法解一系列方程

确定解非零元分布 $(G_{k-1}, \mathcal{A}_k) \longrightarrow \begin{matrix} \mathcal{U}^{(k)} \\ \mathcal{L}^{(k)} \end{matrix} \longrightarrow G_k$



一般稀疏阵的选主元LU分解

- GPLU算法(Gilbert/Peierls LU)

$$\begin{cases} L_{11}u_{12} = a_{12} \\ l_{21}u_{12} + u_{22} = a_{22} \\ L_{31}u_{12} + l_{32}u_{22} = a_{32} \end{cases}$$

写成一个下三角方程

$$\begin{bmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$

即第k步消去前的$A^{(k)}(:,k)$

- 则, $u_{12} = x_1, u_{22} = x_2, l_{32} = x_3 / u_{22}$
- 选主元: 保证l_{32}中元素≤1, 即x_3中元素≤ x_2
- 原理算法：Lu_left

对$A^{(k)}$选主元可通过操作$\begin{bmatrix} x_2 \\ x_3 \end{bmatrix}$实现!

LU=PA

```
L= speyes(n); U= speye(n); P= (1:n)';
For k= 1: n
   x= L \ A(P, k);          //Lsolve算法(符号分解, 数值分解)
   find pivot position i from x(k:n);
   swap rows i and k of P, x and L (前k-1列);
   U(1:k, k)= x(1:k);       //U的第k列
   L(k+1:n, k)= x(k+1:n)/U(k,k)   //L的第k列
end
```

(可证明其正确性)

计算复杂度为 $O(n+nnz(A)+f)$

减少矩阵填入元的重排序

- 分解过程中的填入(fill-in)
  - 对计算量f的影响很大
  - 可通过矩阵的重排序减小fill-in
  - 行/列都可以重排序，对称地重排

GPLU分解算法：

$O(n+nnz(A)+f)$

排除与0运算的浮点运算次数

$Ax=b$

$PAP^Ty=Pb$

$$\begin{bmatrix} X & X & 0 \\ X & X & X \\ 0 & X & X \end{bmatrix}$$

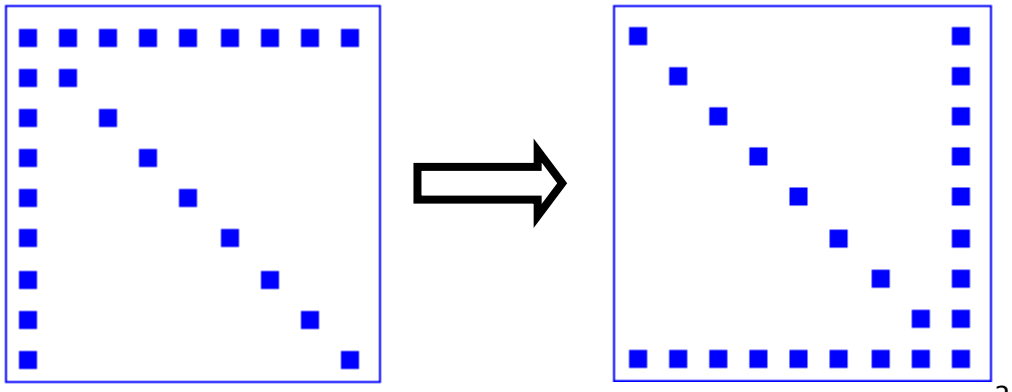
交换1,2行/列

$$\begin{bmatrix} X & X & X \\ X & X & 0 \\ X & 0 & X \end{bmatrix} \Rightarrow \begin{bmatrix} X & X & X \\ X & X & X \\ X & X & X \end{bmatrix}$$

Fill-ins

No fill-in!

怎么做对称重排序，使得没有fill-in？



减少矩阵填入元的重排序

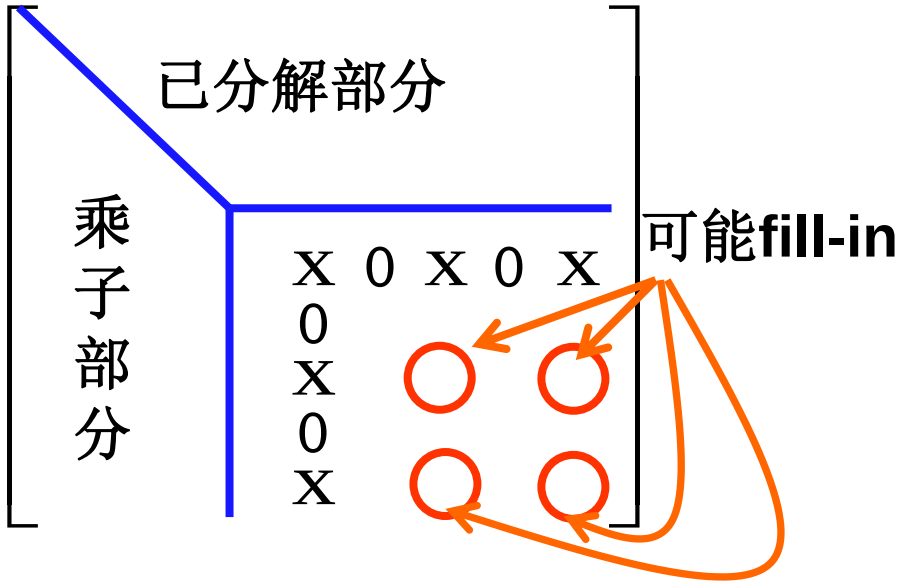
- 三种矩阵重排序

$$\begin{cases} PAx = Pb & \text{行重排} \\ APy = b & \text{列重排} \\ PAQy = Pb & \text{(若 } Q=P^T\text{, 为对称重排, 保对称正定)} \end{cases}$$

- 主要问题
  - 怎样重排序使矩阵做LU/Cholesky分解时填入元最少？
  - 1.重排序与数值选主元可能矛盾
  - 2.即便不考虑选主元(如: SPD阵、对称重排)， NP-hard问题!
- Markowitz重排序算法 (不选主元的情况)
- 其他启发式重排序算法
- 需要选主元时如何处理?

减少矩阵填入元的重排序

- Markowitz算法
  - (1957), 定义填入元上限: Markowitz积
  - 对未分解部分,计算任一非零元素的Markowitz积=
    (当前行非零元数 −1)×
    (当前列非零元数 −1)
  - 将元素交换到主元位置后,其Markowitz积不变
  - Markowitz积最小的为主元
  - 例子:
    列重排LU分解



For $k = 1$ to n-1

在a_{kl} ($l \geq k$)中寻找Markowitz积最小的那项a_{kj}

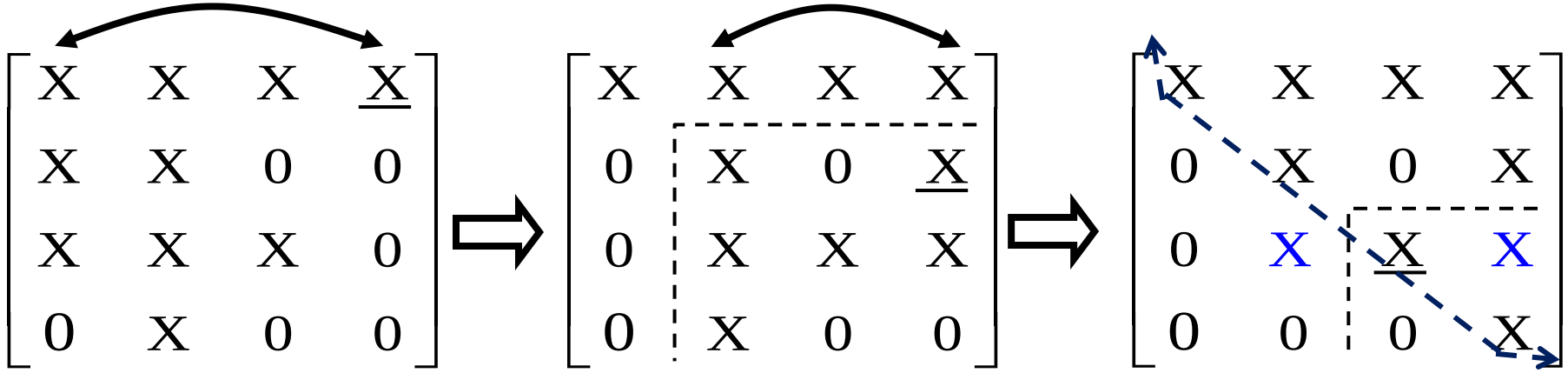
若$j \neq k$, 交换A的k, j列

用新的第k行执行消元操作

End

减少矩阵填入元的重排序

- Markowitz算法 (执行列重排的例子)



若不重排, 5 fill-in's

重排后, 0 fill-in !

- 类似地，可得行重排、行列重排、对称重排算法
- 对于对称阵, 对称重排仅在对角元上选
- 对角元Markowitz积=图中顶点的度的平方
- Markowitz算法不断找度最小的顶点, 即最小度(MD)排序

注意：近似、局部最优, 无法保证全局最优!

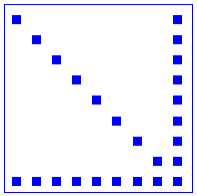
Permutations for sparsity

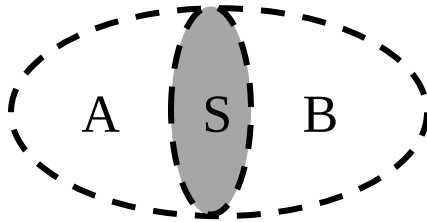
"I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were 'sparse' in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care"

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics

Courtesy of J. Gilbert at UCSB

减少矩阵填入元的重排序

- 其他的启发式重排序
  - 嵌套割集(Nested dissection): 矩阵图的割集
  -  找矩阵图的割集, 排最后; 如此反复
  - 受 的启发; 对非常大的例子往往效果好.
  - 近似最小度(Appr. min. degree, AMD)
  - Markowitz算法效率较低; T. Davis等人提出改进
  - 理论上次优; 但对中规模问题效果较好
  - 带宽缩小排序(Reverse Cuthill-McKee, Sloan, . . .)
  - 目标: 非零元向对角线集中; often wins for “long, thin” problems



$$\begin{bmatrix} A & O & X \\ O & B & X \\ X & X & S \end{bmatrix}$$

The best *general-purpose* orderings are ND/MD hybrids

减少矩阵填入元的重排序

- Matlab中的fill-reducing重排序 (amd_demo.m)
  - 对称近似最小度(Symmetric AMD)
  - amd比symamd速度快

比chol(A)的结果更稀疏

```
>> p = amd(A);
>> Lp = chol(A(p, p), 'lower');
```

  - 非对称近似最小度(Nonsymmetric AMD)
  - colamd (colamd_demo.m) 列排序，适合LU分解, QR分解

```
>> p = colamd(A);
>> [L, U, P] = lu(A(:,p));
```

  - 带宽缩小排序(Reverse Cuthill-McKee)
  - symrcm (rcm_amd.m)

```
>> p = symrcm(A);
>> Lp = chol(A(p, p), 'lower');
```

Matlab中没有ND排序;

pp.6的那个问题

同时考虑重排序与数值选主元

- 选主元的LU分解
  - 进行行、列的重排序(类似全选主元)
  - 交换行列时同时考虑: 数值主元、fill-in reduction
  - 为了控制填入元，放松数值主元的要求
  - 原始要求: $a_{kk}^{(k)} \geq a_{ik}^{(k)},\ i > k$
  - **阈值选主元**: $a_{kk}^{(k)} \geq \mu a_{ik}^{(k)},\ i > k$, 例如取 $\mu = 0.1$

$$PAQ = LU$$

- UMFPACK算法
  - 对于非对称矩阵: [L, U, P, Q]=lu(A)
  - LU分解的Right-looking算法
  - Unsymmetric mutilfrontal技术、一种与分解结合的重排序
- “\”对稀疏矩阵的处理:
  - 用AMD排序，见spparms
  - CHOLMOD (对称正定阵)
  - UMFPACK (其他矩阵)

参考资料

- T. A. Davis, *Direct Methods for Sparse Linear Systems*, SIAM, Philadelphia, Sep. 2006.
- T. A. Davis, **(SuiteSparse)** **(CSparse)** **http://www.cise.ufl.edu/~davis/**
- John R. Gilbert, **http://www.cs.ucsb.edu/~gilbert/**
- Y. Saad's Course: *"Sparse Matrix Computations"*, CSE Department, University of Minnesota, USA.
- Florida collection

"http://www.cise.ufl.edu/research/sparse/matrices/"

- SUPERLU web-page

"http://crd.lbl.gov/~xiaoye/SuperLU"

对称正定阵的Cholesky分解

- Up-looking算法
- 消去树
- 小结

非结构化的稀疏对称正定阵

- 将Cholesky分解转化为求解一系列Lx=b

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & & \vdots \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & l_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

第3种算法

$L_{k-1}x = a_k(1{:}k\text{-}1)$，$L_{k-1}$为L的k-1阶顺序主子阵 **Up-looking方法**

- Up-looking算法

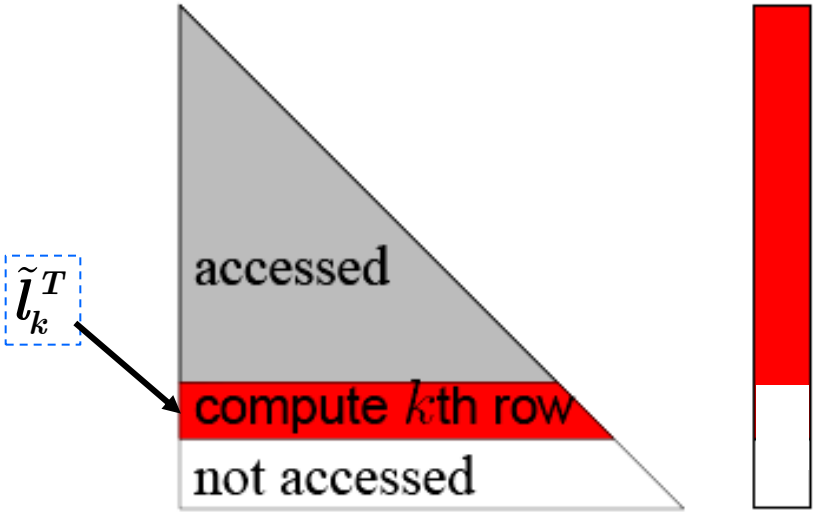
for k = 1:n

solve $L_{k-1}\tilde{l}_k = a_k(1{:}k\text{-}1)$ for $\tilde{l}_k$

$l_{kk} = \sqrt{a_{kk} - \tilde{l}_k^T \tilde{l}_k}$

end

(k=1时不执行)



若A稀疏，能否容易地知道$\tilde{l}_k$的，及L的非零元分布？

非结构化的稀疏对称正定阵

- 利用Lsolve算法进行分解 $A = LL^T$
  - 计算L第k行的非对角元: solve $L_{k-1}\tilde{l}_k = a_k(1:k-1)$
    - G_{k-1} : 矩阵L_{k-1}的图
    - $\mathcal{A}_k$: a_k 向量上三角部分中的非零元位置
    - $\tilde{\mathcal{L}}_k$: L 第k行(除对角元)的非零元位置, 则 $\tilde{\mathcal{L}}_k = \mathrm{Reach}_{G_{k-1}}(\mathcal{A}_k)$
  - 1.符号分解: 逐行得到L的非零元分布 (一些算法需要)
    - $(G_1, \mathcal{A}_2) \rightarrow \tilde{\mathcal{L}}_2 \rightarrow G_2$, $(G_2, \mathcal{A}_3) \rightarrow \tilde{\mathcal{L}}_3 \rightarrow G_3$, ……
  - 2.数值分解: 根据Lsolve、Up-looking算法具体计算L矩阵
- 使用消去树降低符号分解的复杂度
  - 每次DFS后序遍历得$\tilde{\mathcal{L}}_k$: $O(|\tilde{\mathcal{L}}_k| + e)$ (e为遍历的边数)
  - 剪掉L图G_{k-1}的一些边得到树$\mathcal{T}$, DFS复杂度降到$O(|\tilde{\mathcal{L}}_k|)$

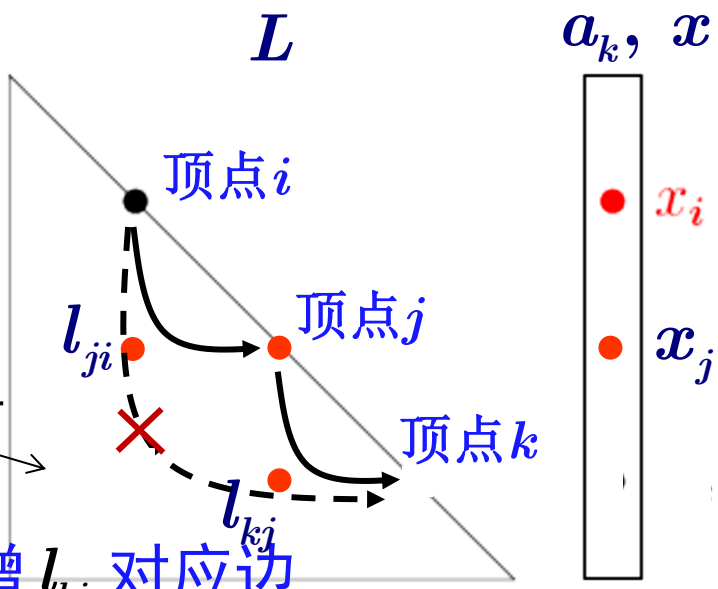
稀疏Cholesky分解中的符号分解

- 消去树(elimination tree) $G_k \to \mathcal{T}$
  - 符号分解: $(G_{k-1}, \mathcal{A}_k) \to \tilde{\mathcal{L}}_k \to G_k$,　　解 $L_{k-1}x = a_k(1:k-1)$
  - 设 $a_k(i) \neq 0$, $l_{ji} \neq 0$　　(j是i的邻点)

    $\Rightarrow$ $j \in \tilde{\mathcal{L}}_k$ $\Rightarrow$ $l_{kj} \neq 0$
  - G_k中从顶点j到k有一条边

    $i \in \tilde{\mathcal{L}}_k$, $l_{ki} \neq 0$, G_k中顶点i到k有边
  - 对后续计算 $\mathrm{Reach}_{G_k}(\mathcal{A}_{k+1})$ 这条边没必要!

    算第k行非零元 →

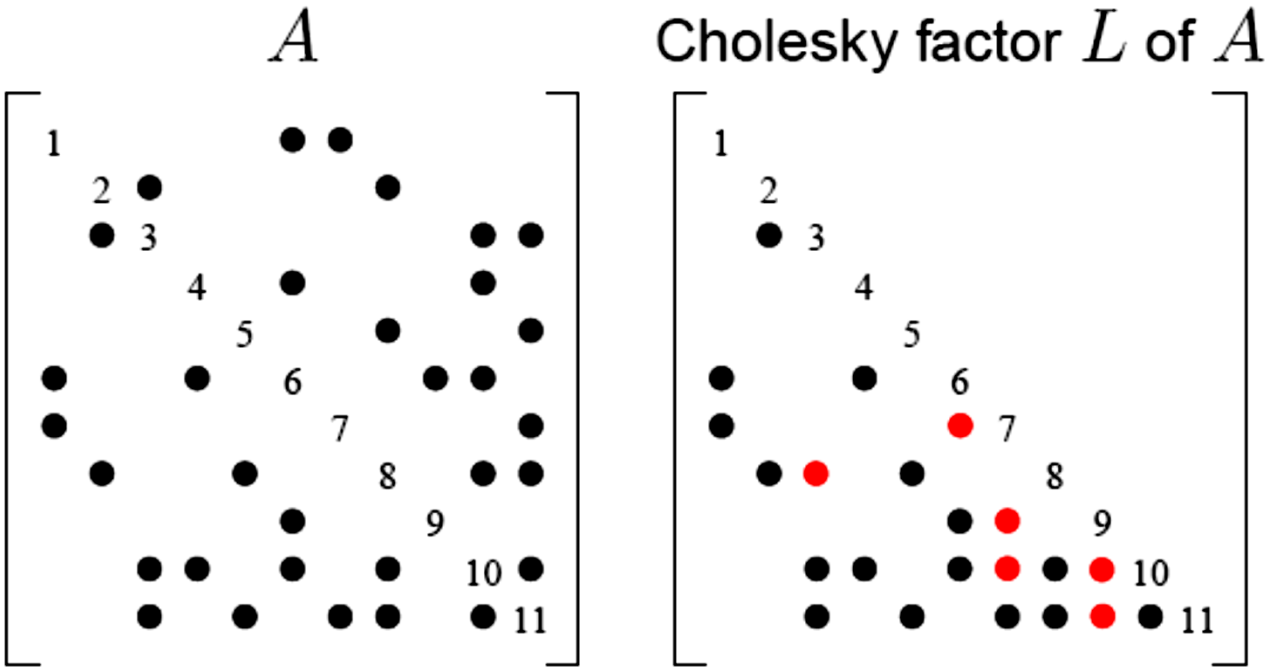


  - L第i列, 只要有$j<k$, $l_{ji} \neq 0$, 就不用增 l_{ki} 对应边
  - 删除这种多余边后, 图$G_k(k=2, \ldots)$中, 任何顶点i只有一条到j的出边, j是满足$l_{ji} \neq 0$ 的最小数.　　"消去树"$\mathcal{T}$ (j是i的双亲)

稀疏Cholesky分解中的符号分解

- 消去树(e-tree)—例子

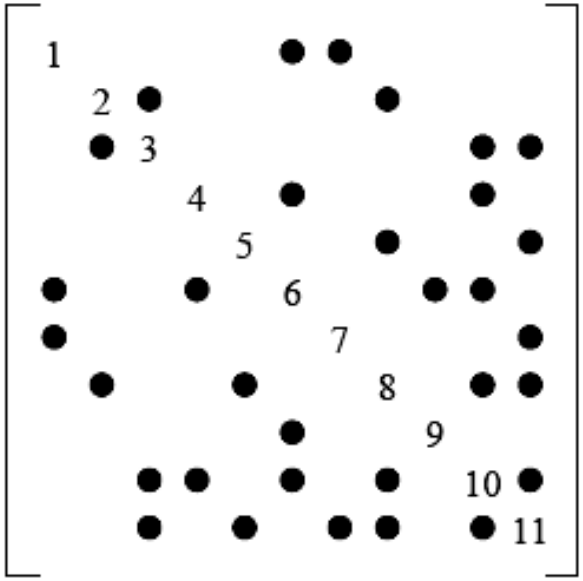
● 填入元



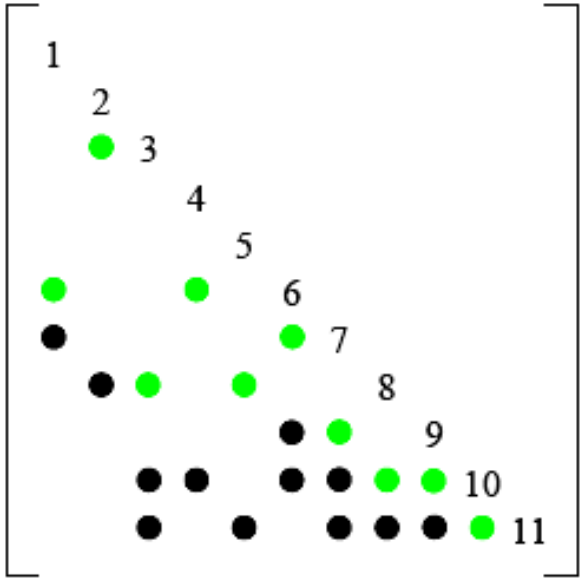
稀疏Cholesky分解中的符号分解

- 消去树(e-tree)—例子
  - 消去树中的边

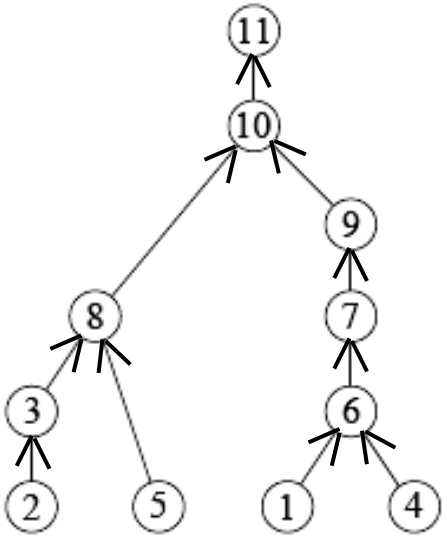
A



Cholesky factor L of A



elimination tree



通过构建“消去树”，$\tilde{\mathcal{L}}_k = \mathrm{Reach}_{\mathcal{T}_{k-1}}(\mathcal{A}_k)$ 的计算复杂度减小为$O(|\tilde{\mathcal{L}}_k|)$ (访问的边数目 = 顶点的数目！)

稀疏对称正定阵的分解

- 小结
  - 符号分解(Symbolic analysis):
  - 　依据原矩阵的非零元分布(pattern)，计算分解后因子矩阵的非零pattern, 方便后续执行的数值分解
  - 求解多个pattern一样的线性方程组时, 可以recycle SA
  - Up-looking算法: Lsolve算法 + 消去树 (排序, 计数, …)
  - Csparse程序包: cs_etree.c, cs_schol.c, cs_chol.c
- 更多算法与命令

Chap. 4

  - Left-looking, supernodal (更常用, 性能更高)
  - Matlab命令: etree, symbfact, chol

T. Davis的CHOLMOD包, 用 { left-looking ; up-looking }