データ構造とプログラミング技法（第４回）

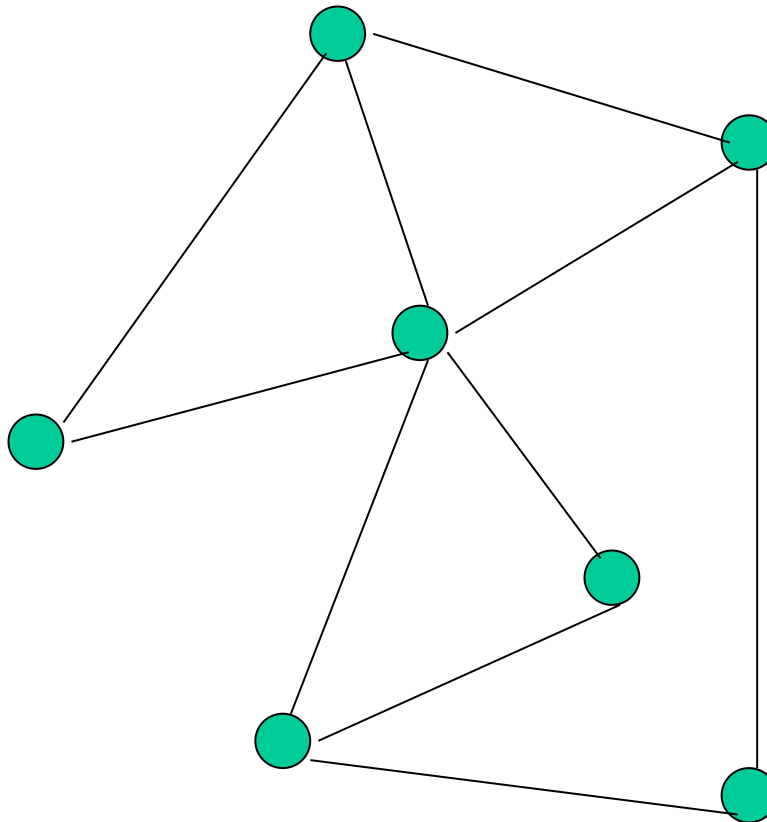
ーグラフ構造ー

グラフ構造

- グラフは以下の二つの集合によって定義される。

（１） 頂点の有限集合V

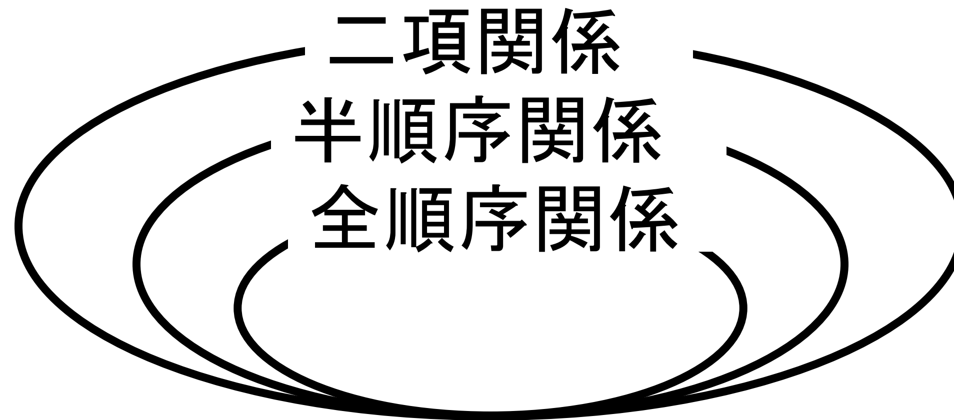
（２） 辺（2つの頂点の組）の有限集合E（⊆V×V）

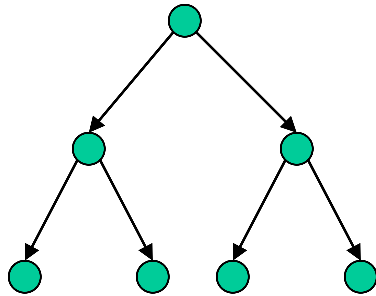


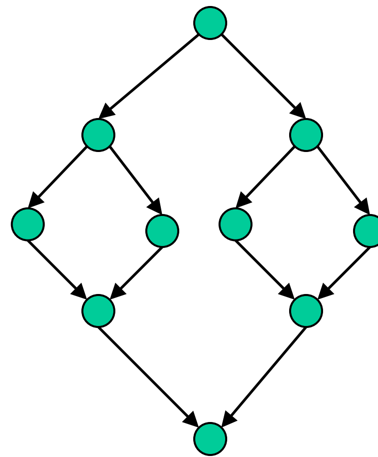
G=(V，E)

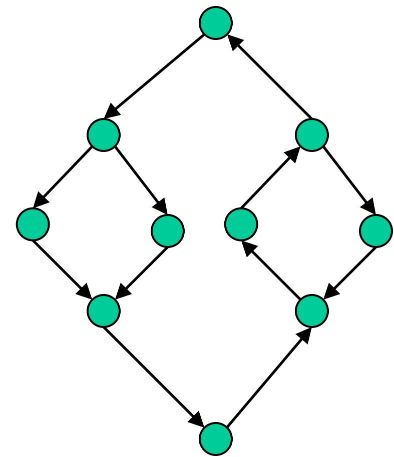
グラフ構造

- グラフは一般的な二項関係の表現ができる。

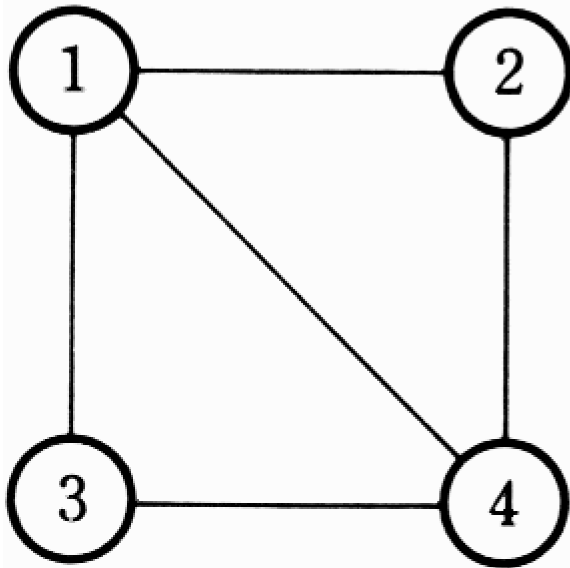


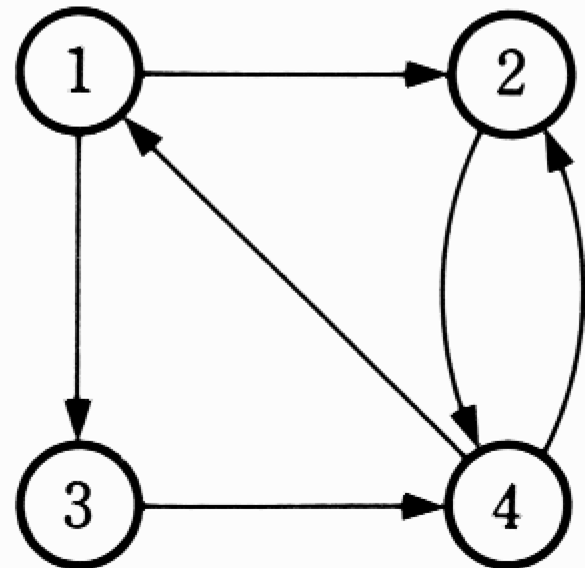






グラフの種類





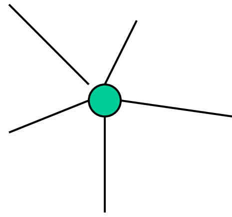
$G_1=(V_1, E_1)$
$V_1=\{1,2,3,4\}$
$E_1=\{(1,2),(1,3),$
$(1,4),(2,4),$
$(3,4)\}$
(a) 無向グラフ

$G_2=(V_2, E_2)$
$V_2=\{1,2,3,4\}$
$E_2=\{\langle 1,2\rangle,\langle 1,3\rangle,$
$\langle 2,4\rangle,\langle 4,2\rangle,$
$\langle 3,4\rangle,\langle 4,1\rangle\}$
(b) 有向グラフ

図 5.1 グラフ

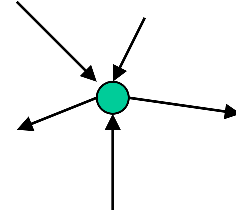
グラフの位相的側面

- 次数　５



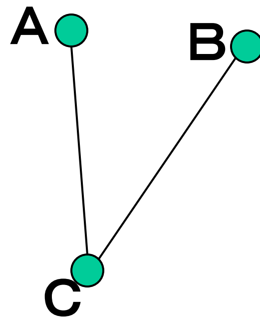
孤立点

- 出次数　２
- 入次数　３



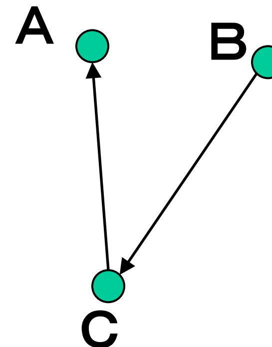
- 「隣接する」

ＡとＣ　ＢとＣ



- 「隣接する」

ＣはＡに　ＢはＣに



完全グラフ

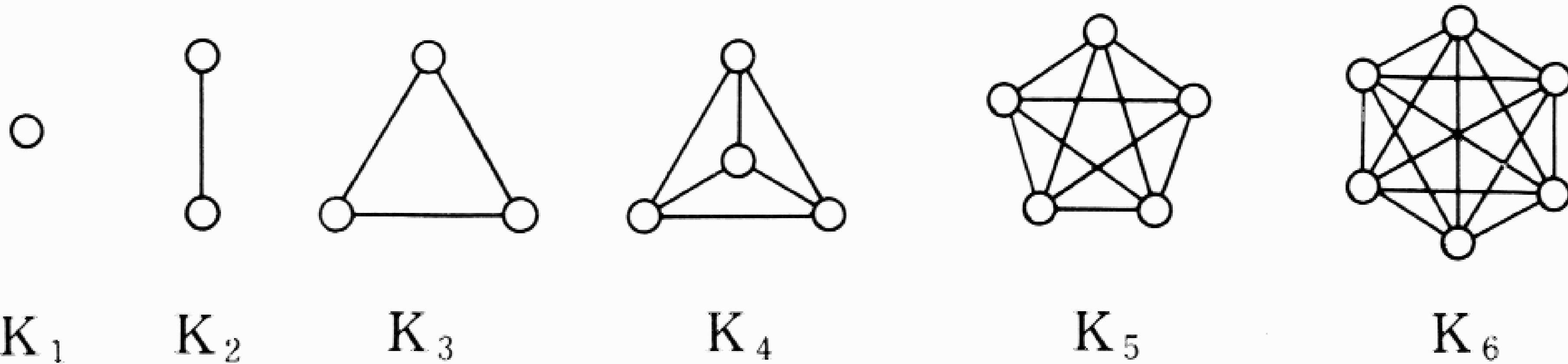
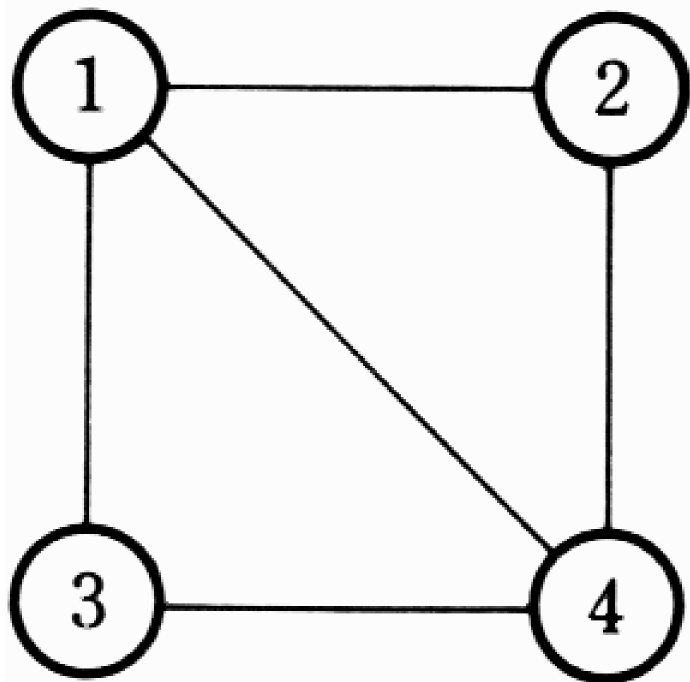


図 5.2 完全グラフ $\mathbf{K}_n$

任意の2頂点が隣接しているようなグラフ

部分グラフ



部分グラフ

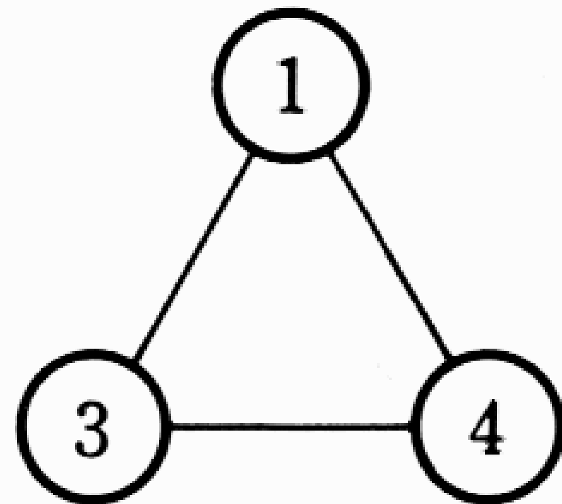
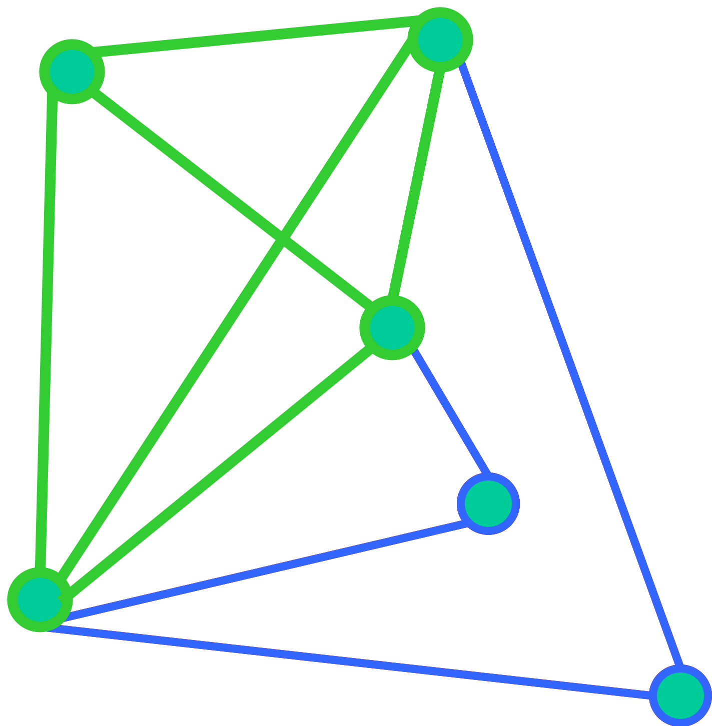
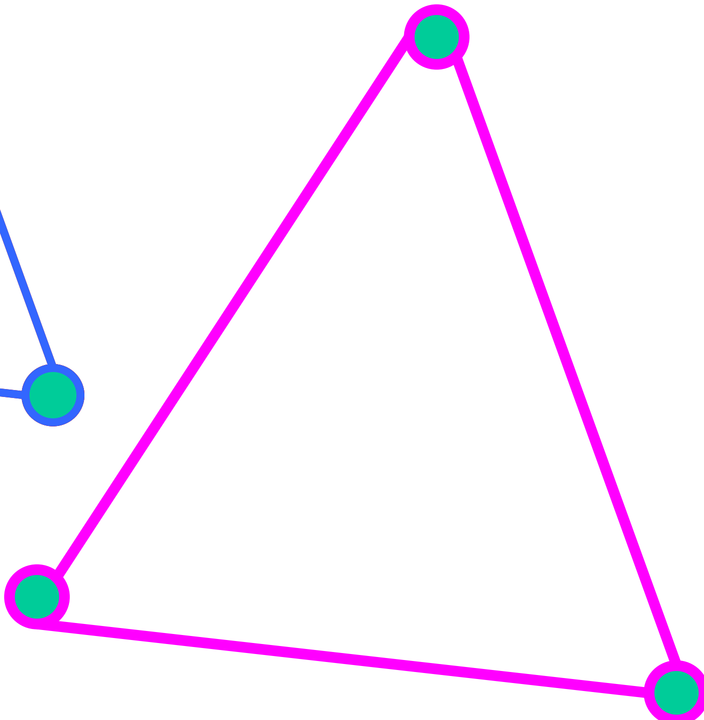


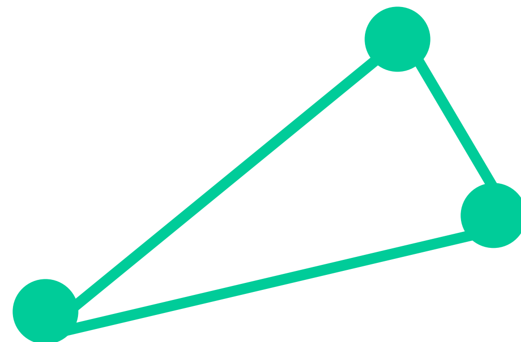
図 5.3　G_1の部分グラフ

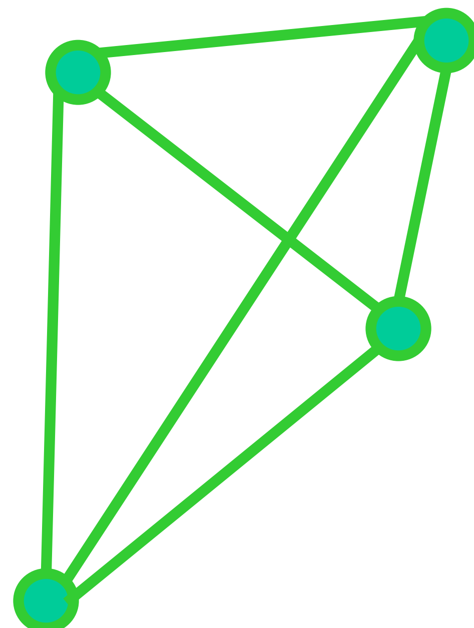
クリーク

完全グラフである**部分グラフ**

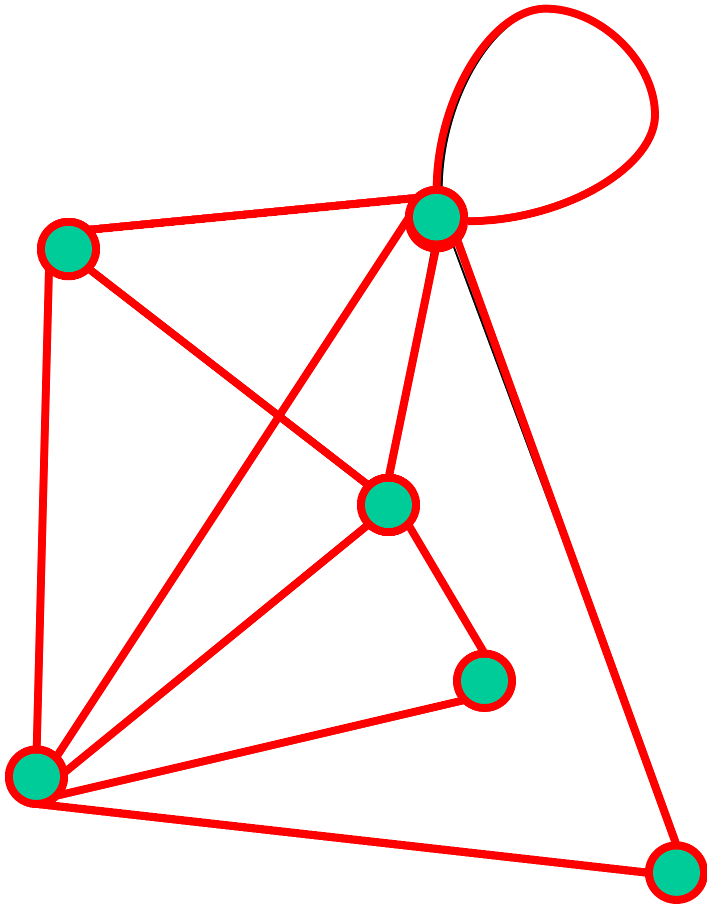


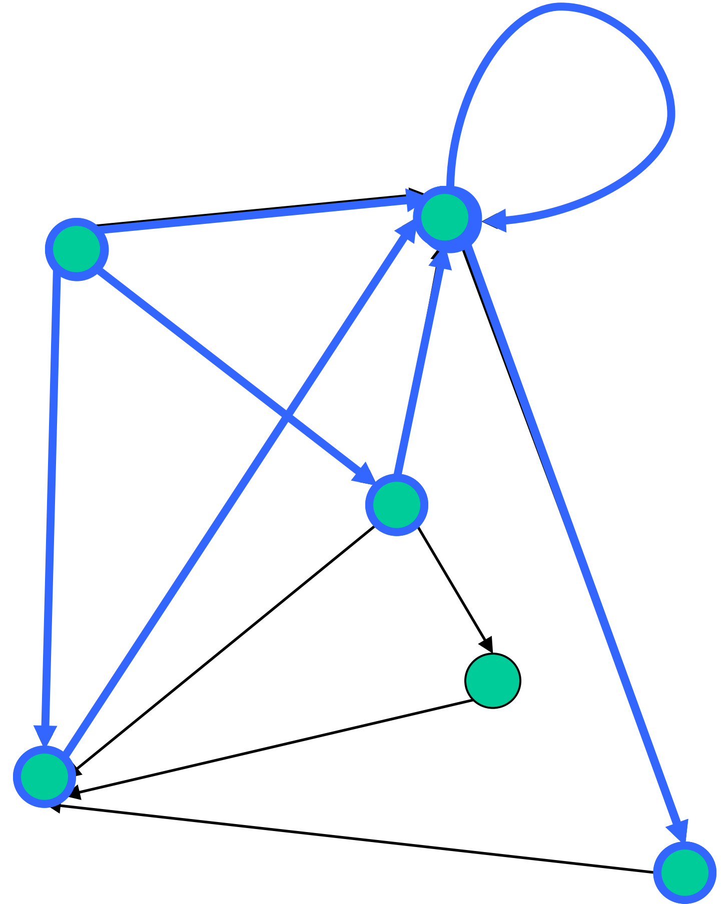




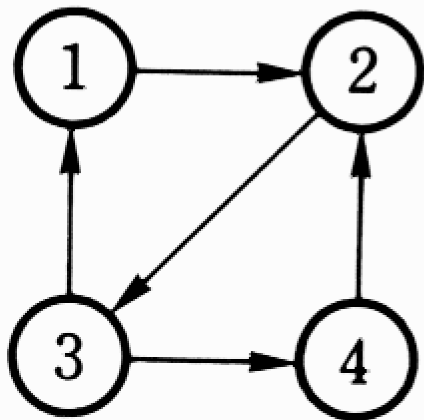


路／閉路／距離

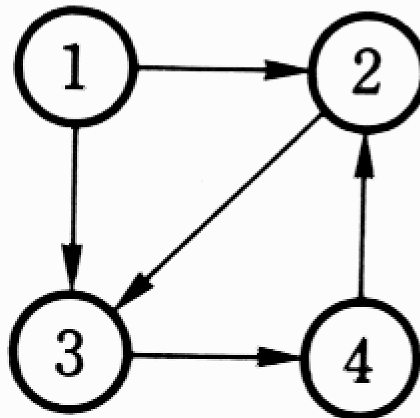




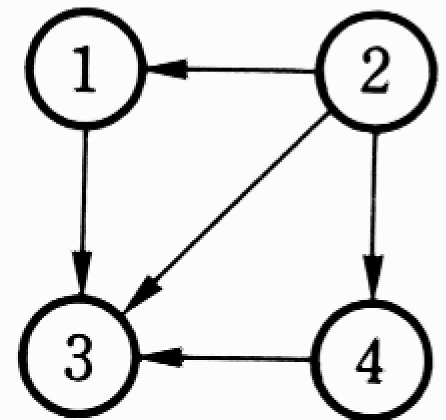
グラフの連結性



強連結



連　結



弱連結

図 5.4　有向グラフの連結性

グラフの直径／離心数／中心／半径

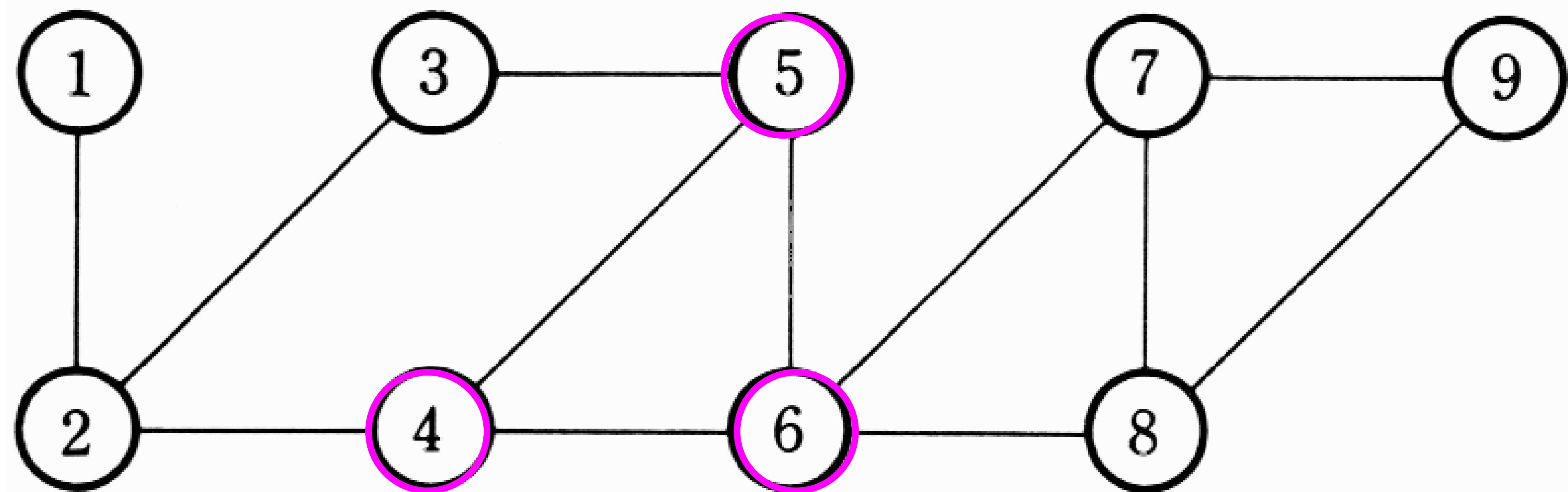
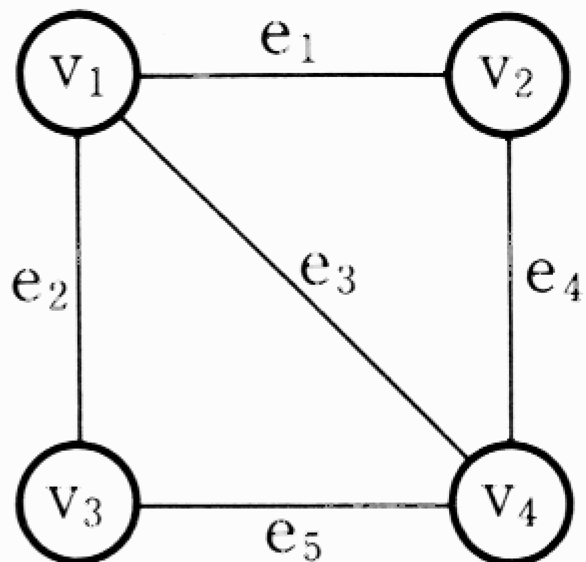
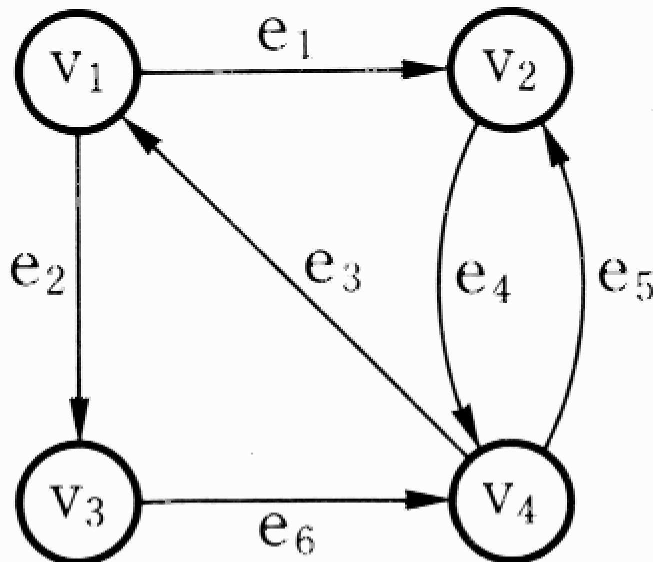


図 5.6 連結（無向）グラフ

グラフの物理データ構造：隣接行列

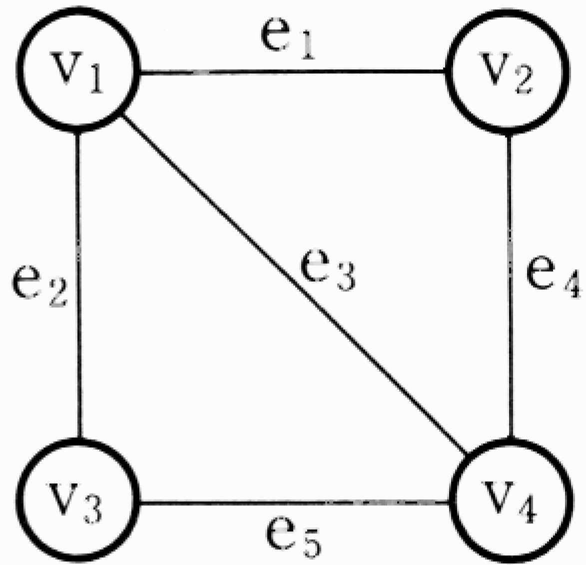




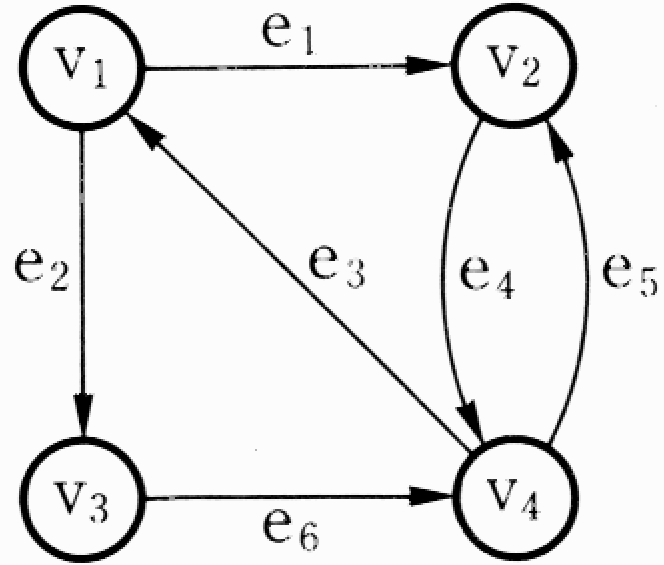
$$A_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

図 5.8 隣接行列

接続行列



(a) 無辺グラフ G_1



(b) 有向グラフ G_2

辺

端点

$$B_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \quad B_2 = \begin{bmatrix} -1 & -1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 1 & -1 & 1 \end{bmatrix}$$

図 5.9 接続行列

リンク配置：直交リスト

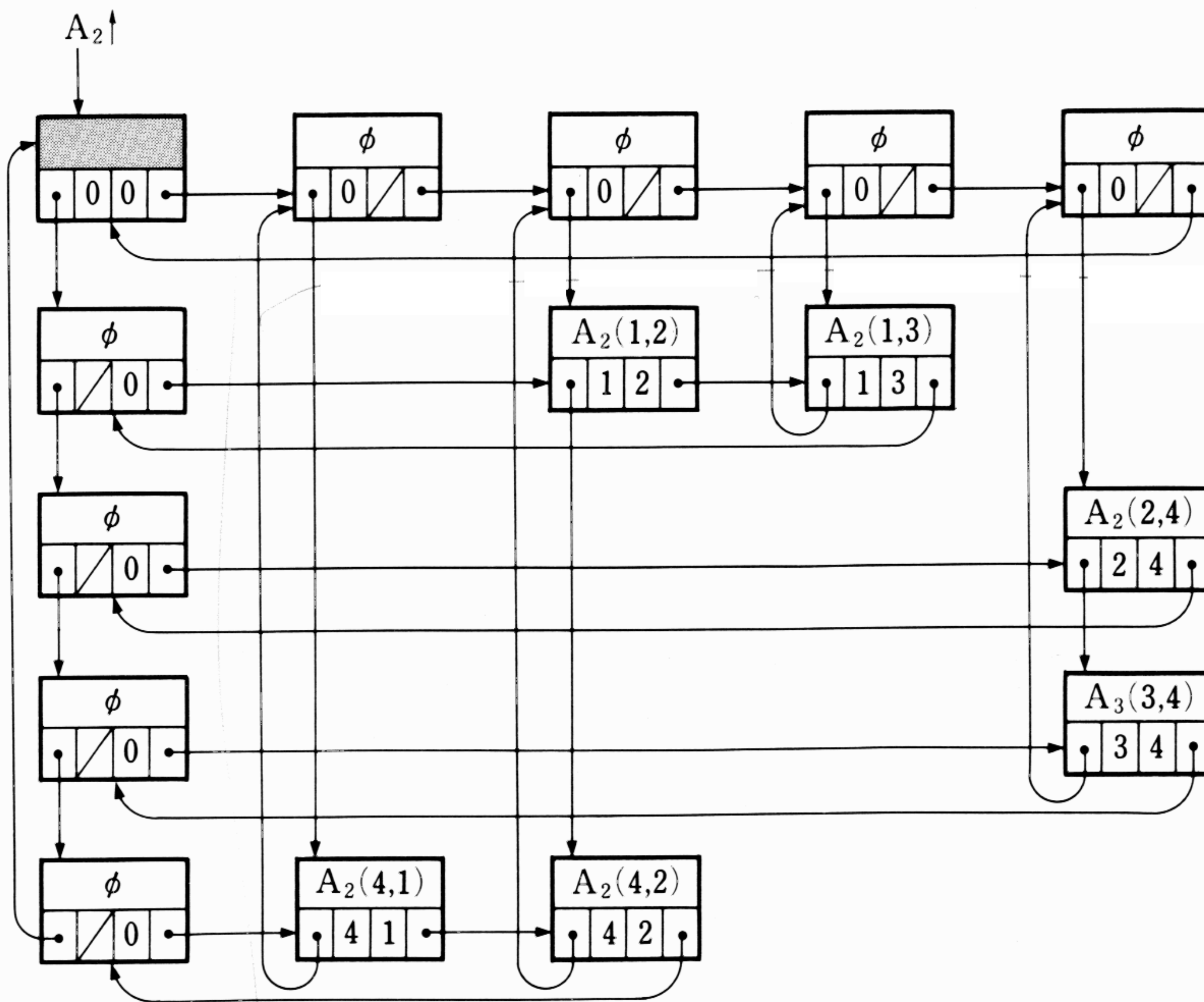
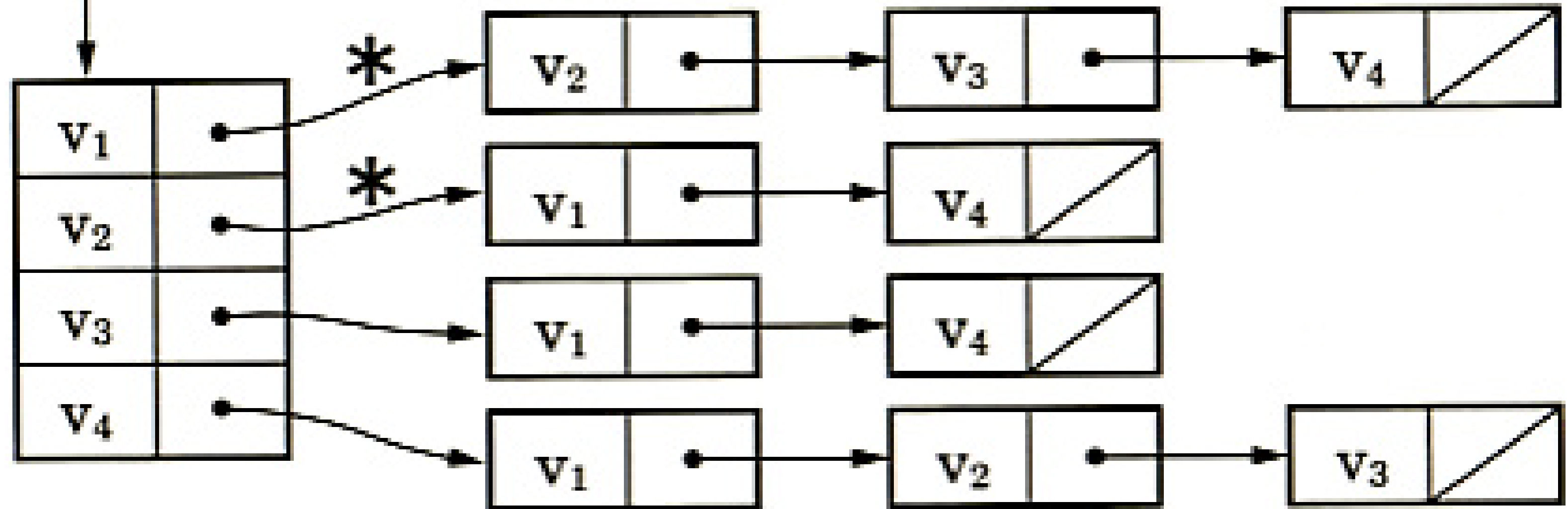


図 5.11 直交リストによる A_2 の表現

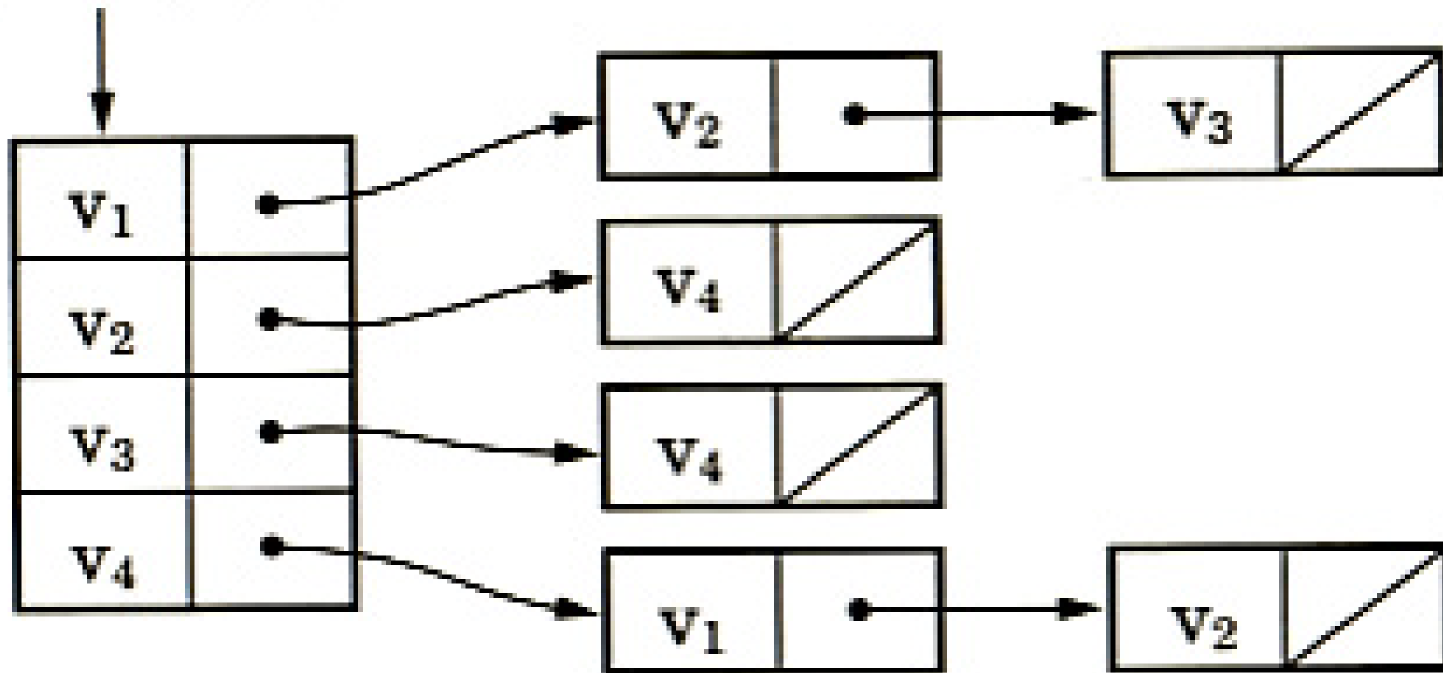
隣接リスト(無向グラフの場合）



（a）無向グラフ G_1

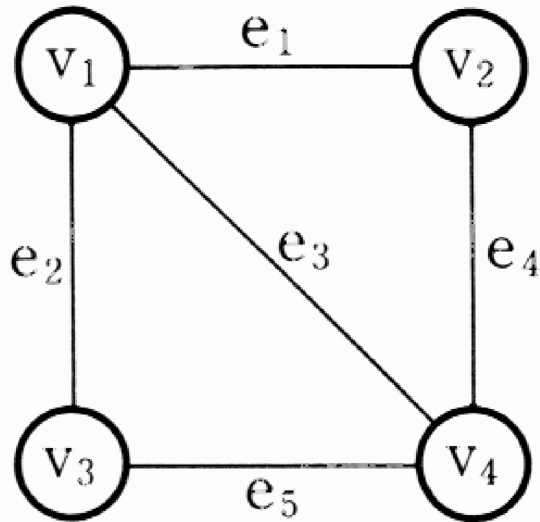
（ａ） 無向グラフ G_1

隣接リスト(有向グラフの場合）



（b）有向グラフ G_2

隣接多重リスト



（a） 無向グラフ G_1

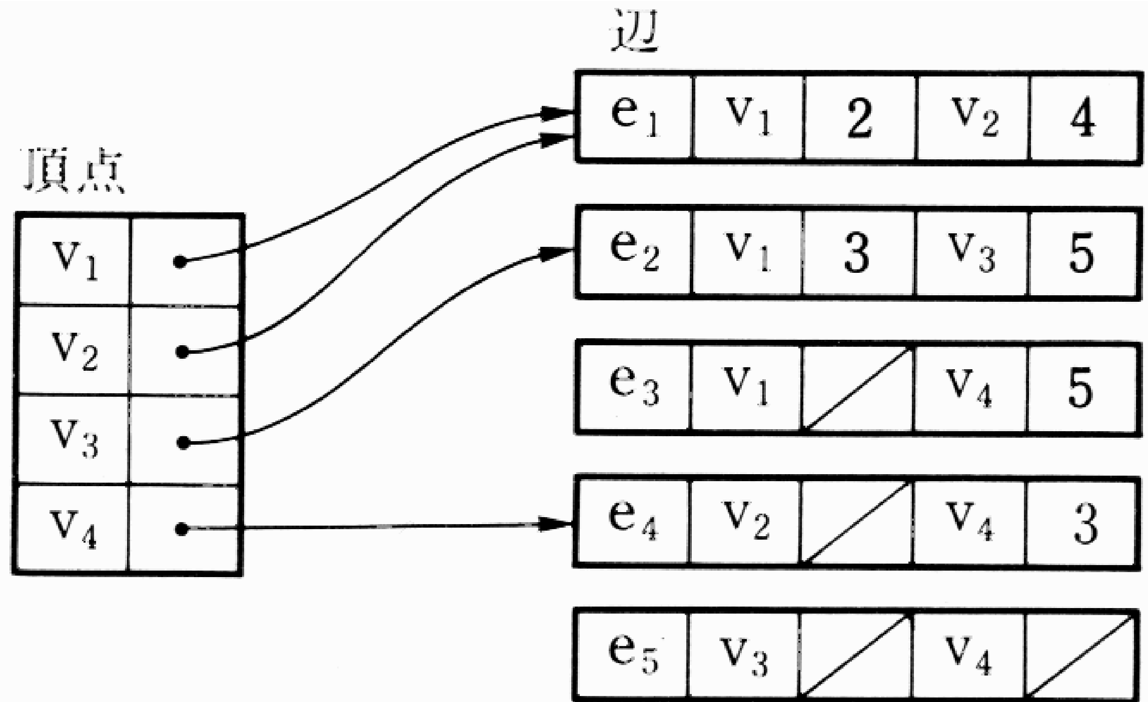
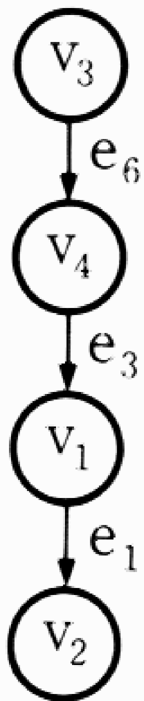
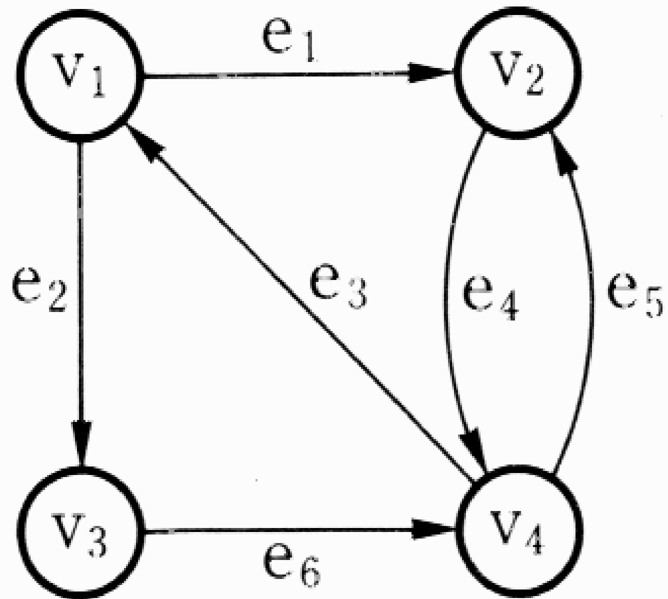


図 5.14 隣接多重リストによる G_1 の表現

グラフの縦型探索



(b1)



(b) 有向グラフ G_2

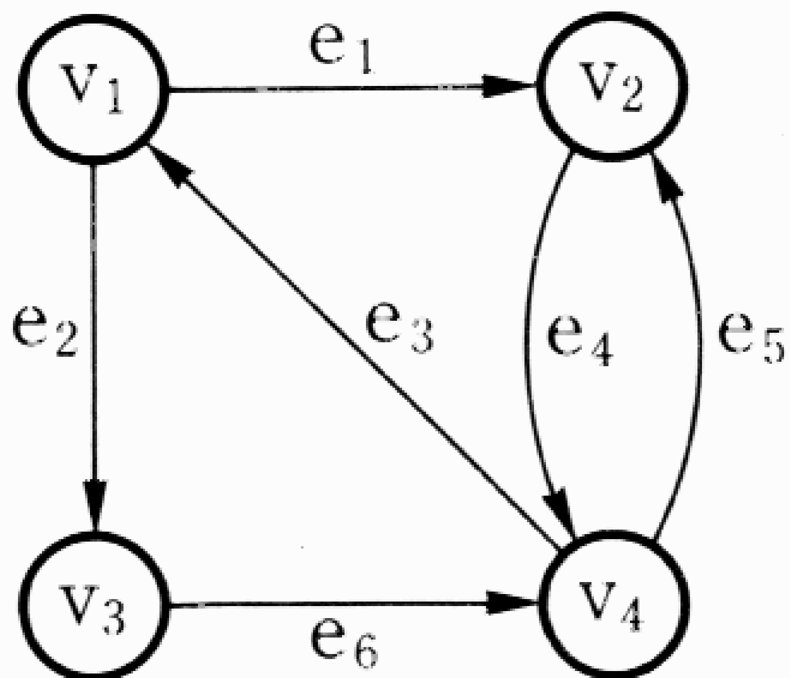
```
vo
{

}

ma
{

}
```

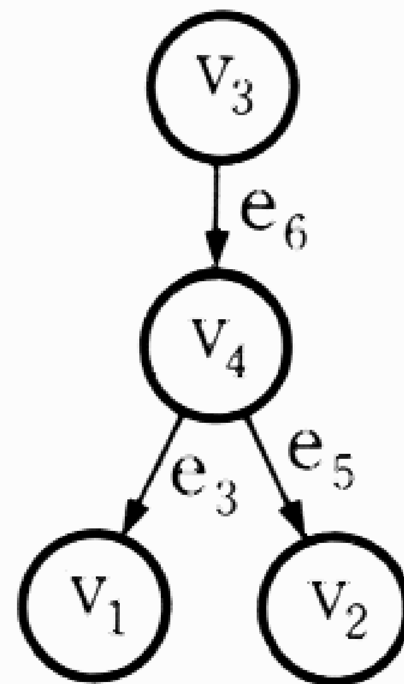
グラフの横型探索（課題）

参考：迷路の探索アルゴリズム

```
list:={'S'};
while list≠φ do
    [ get-and-delete(list,n);
      if n='G' then return solved
                   else [ expand(n,P);
                          foreach m ∈P do
   add(list,m)
    ];
return(no-solution)
```

グラフの横型探索の結果





強連結成分を求める

```
#define   N   100
int graph[N][N],t[N],s;

void strong_connect()
 {
    step1: depth_first_call(0);       // 結果はt[N]に格納される
                                /* ただしdepth_first(v)を終わるとき頂点vに
                                   通し番号s++をつける */
    step2: transpose();
    step3: depth_first_call(1);       // 結果はt[N]に再度格納される
                                /* ただしdepth_first(v)のvは未訪問頂点のう
                                   ち最大の番号を持つものを優先的に選ぶ */
}
```

図5·22　強連結成分を求める手続き

強連結成分への分解

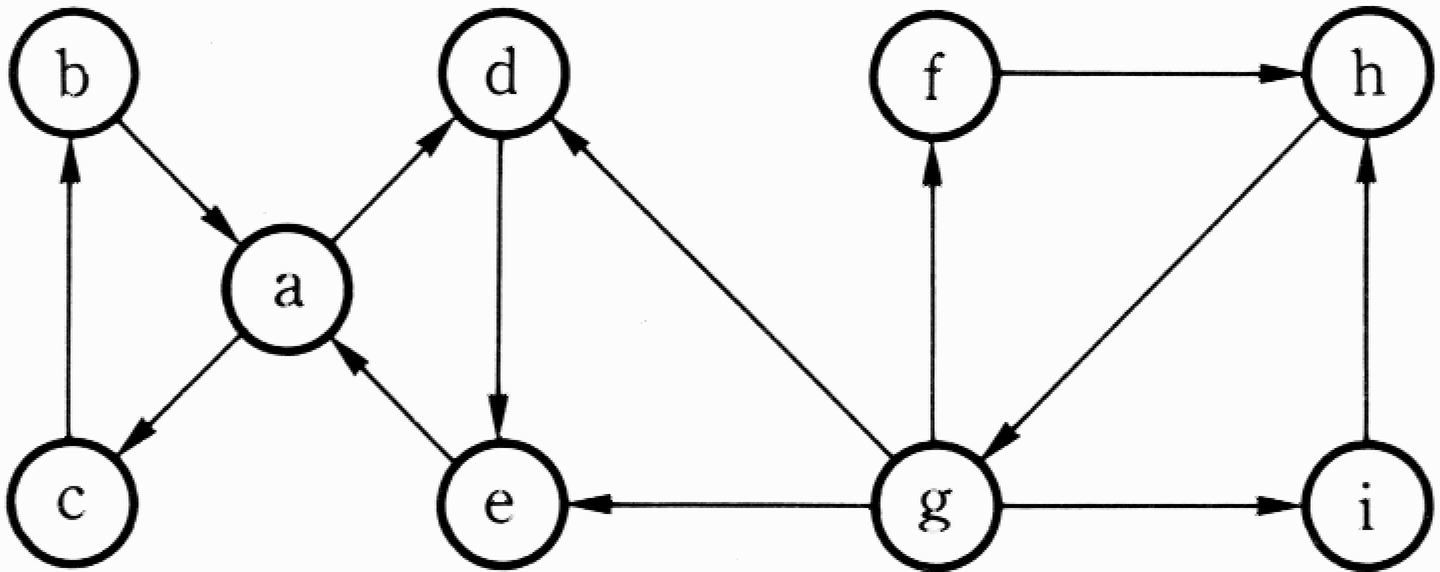


図 5.21　有向グラフ G

STEP1：縦型探索

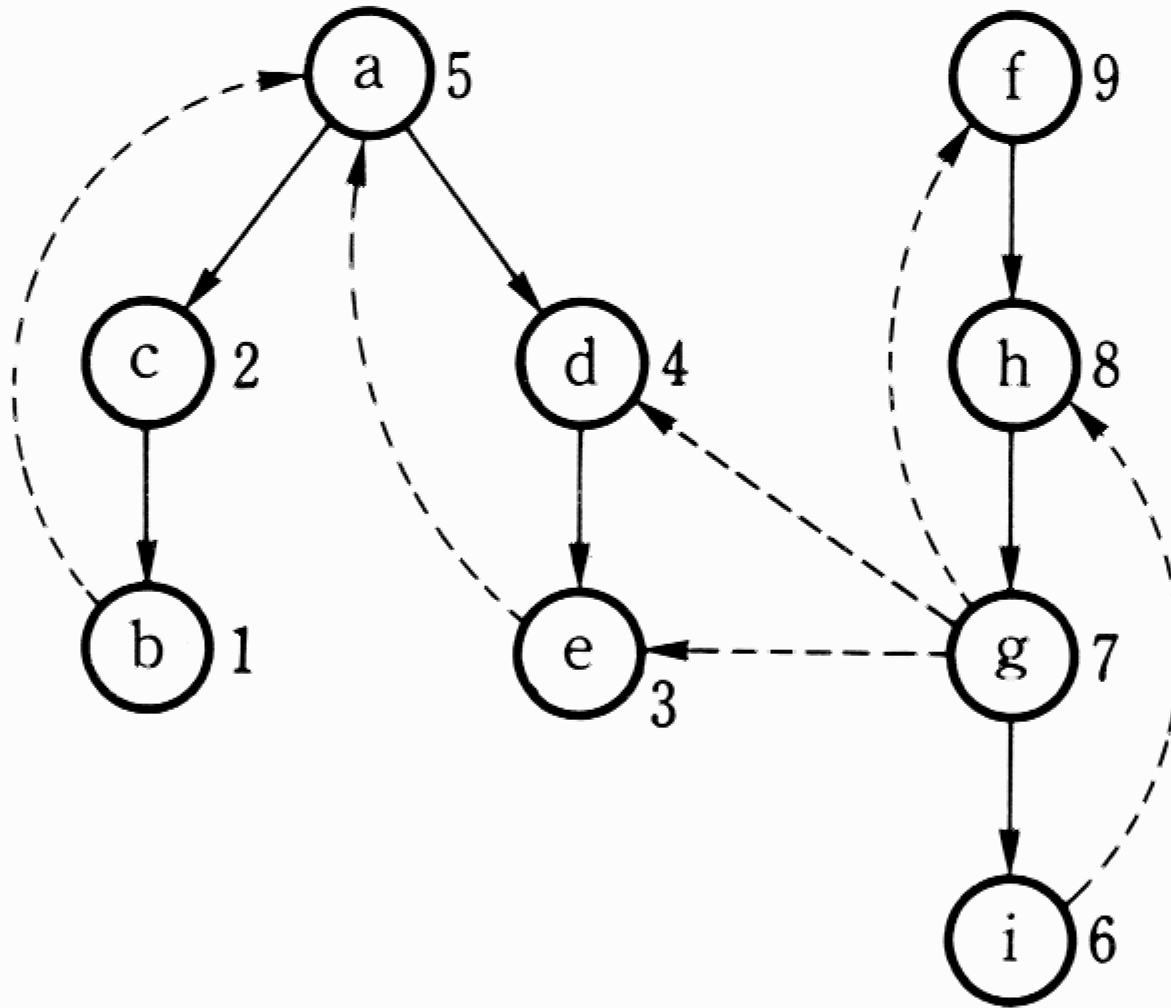


図 5.23　G の縦型探索森（step 1 の結果）

STEP2：辺の向きの反転

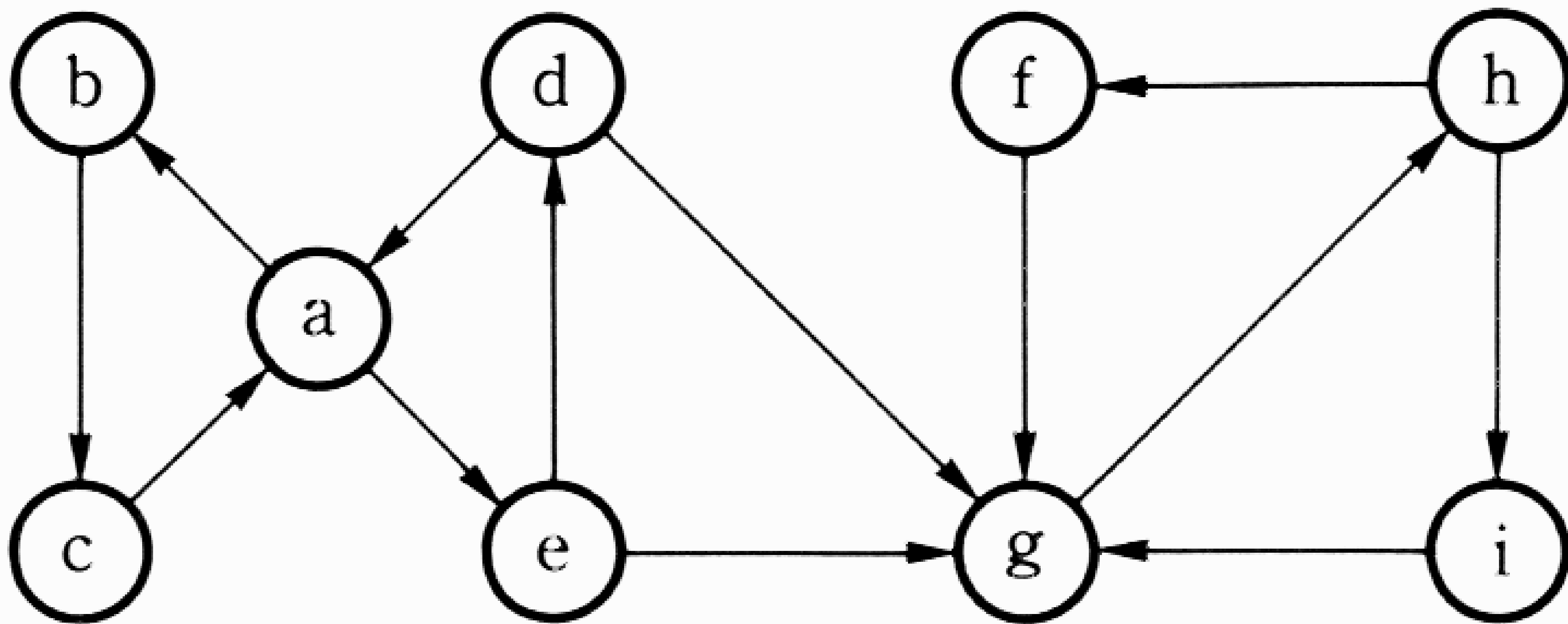


図 5.24　有向グラフ G^T（step 2 の結果）

STEP3：辺を反転したグラフの縦型探索

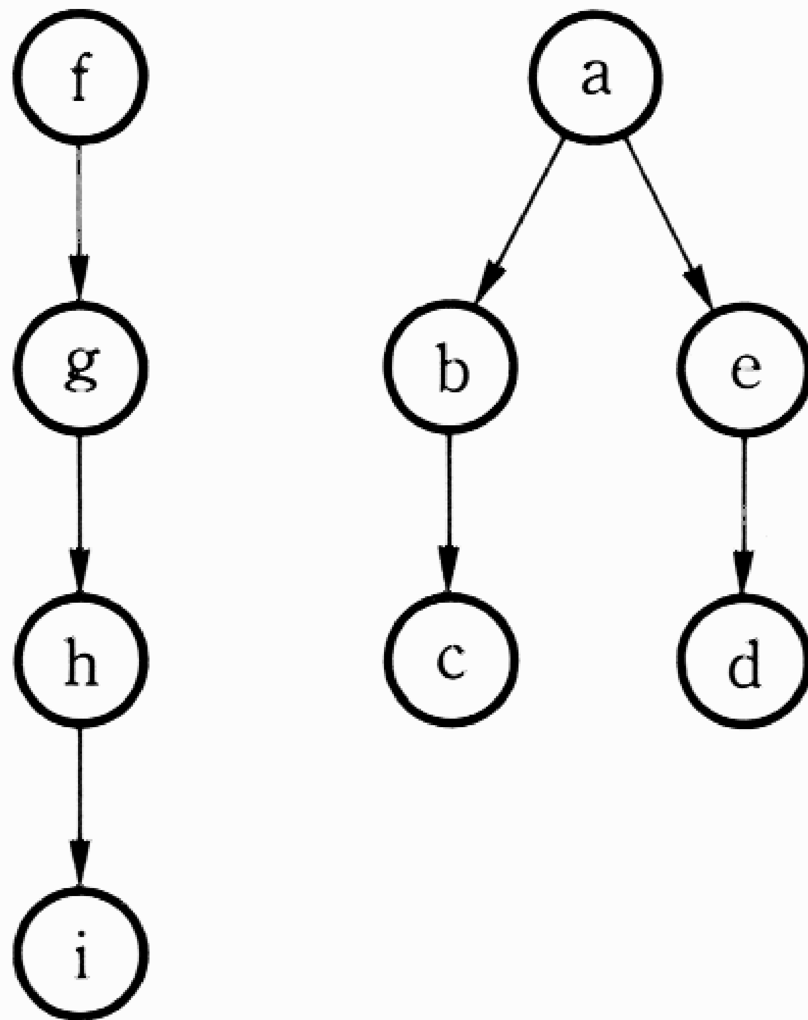


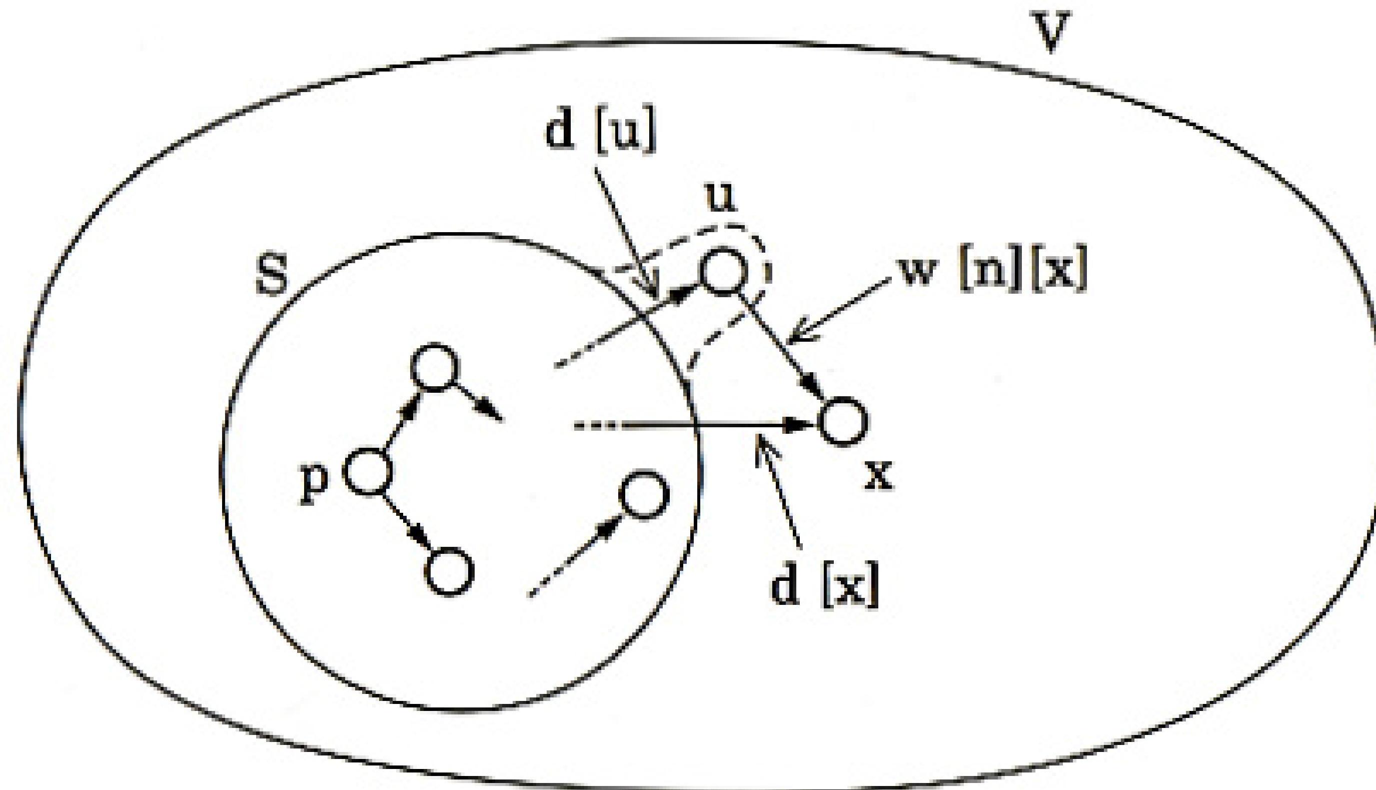
図 5.25　G^T の縦型探索森 F

Dijkstraのアルゴリズム

```
void Dijkstra(int p)
{   int i,u,x,k;
    add(p,S);                                        ①
    for (i=0; i<N; i++) d[i] = w[p][i];              ②
    while (remain()) {                   }
        u = select_min();                }③
        add(u,S);                        }
        for (x=0; x<N; x++) {
          if (member(u,x)) {                                      }
               if (k = d[u]+w[u][x] < d[x]) d[x] = k;}            }④
        }
    }
}
```

図 5·26　頂点 p からの最短距離を求める Dijkstra アルゴリズム

Dijkstraのアルゴリズム



d[u]+w[u][x]<d[x] なら，x に至るより短い路が見つかった
（w[u][x] は辺 <u, x> の重み）

図 5·27　u は V−S 中の最小の d [u] を持つ頂点

Floydのアルゴリズム

```
void Floyd()
{   int i,j,k,can;
    for (i=0; i<N; i++)
        for (j=0; j<N; j++)    {d[i][j] = w[i][j]; p[i][j] = i;} }①
    for (k=0; k<N; k++)                           ②
        for (i=0; i<N; i++)
            for (j=0; j<N; j++) {
                can = d[i][k] + d[k][j];  ③
                if (can < d[i][j]) {
                    d[i][j] = can; p[i][j] = p[k][j];} }④
                }
}
```

図 5·29 すべての2頂点間の最短路と長さを求める**Floyd**アルゴリズム

Floydのアルゴリズム

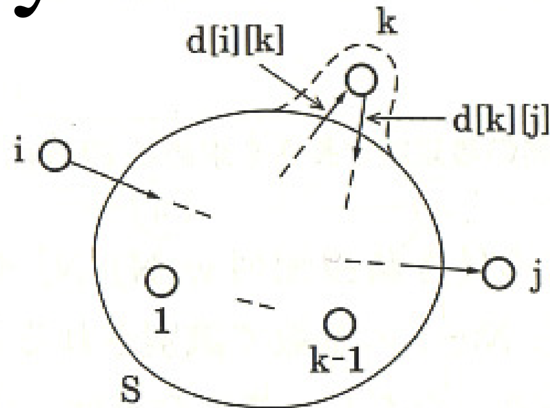


図 5·30　経由してもよい頂点として k を取り入れたときの最短路

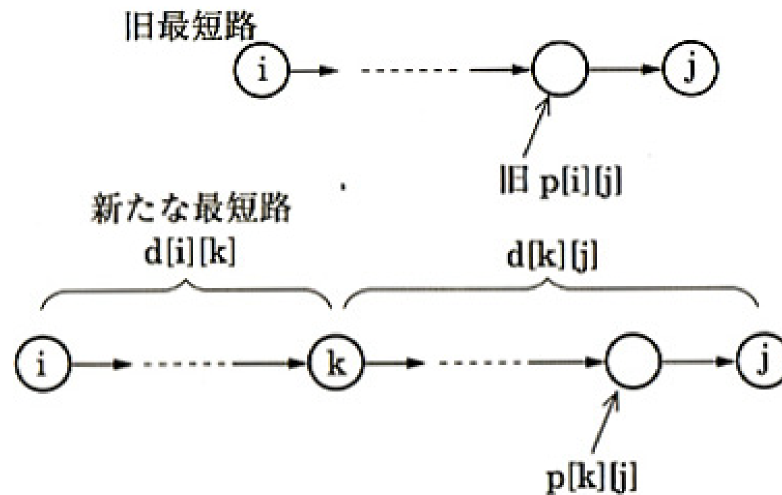


図 5·31　最短路の更新（**p[k][j]** が新たに **p[i][j]** に格納される）

Floydのアルゴリズム

```
void shortest_path(int m; int n)
{    int x;
     if (d[m][n] == M) printf("there is no path\n");
     else {x = n; push(x);
           while (x != m) {x = p[m][x]; push(x);}
           while (empty() != 1) {printf("%d =>",pop());}
           printf(" => END\n");
     }
}
```

図 5·32　最短路を出力する関数

グラフの応用：制約グラフ

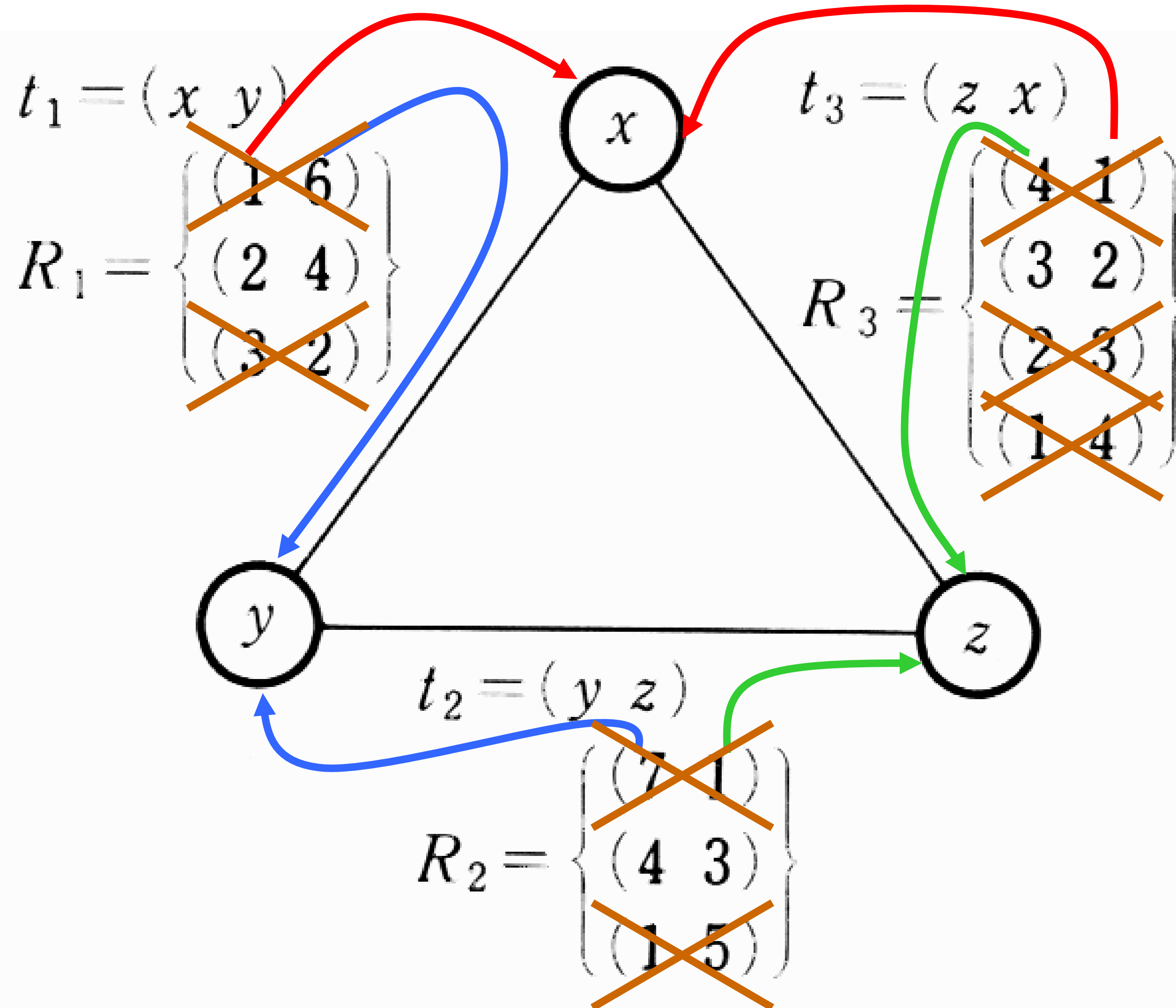


図 5.33　制約ネットワーク

グラフの応用:重みつき制約充足問題

（V,L,T,F)

例:

V={x,y,z}, L={a,b,c},

T={t_1=(x,y), t_2=(y,z) }

y \ z	a	b	c
a	11	9	13
b	12	8	7
c	8	15	11

(a) f_2に $\max f_1$ を加えた結果

z	$\max(f_1+f_2)$	y
a	12	b
b	15	c
c	13	a

(b) 変数 z の値と f_1+f_2 の最大値

図 5.36 変数 y, z に関する最適化(第2ステップの結果)