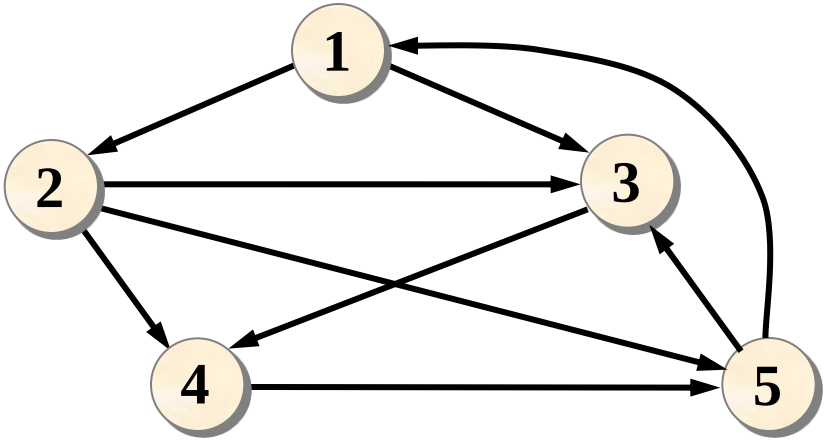
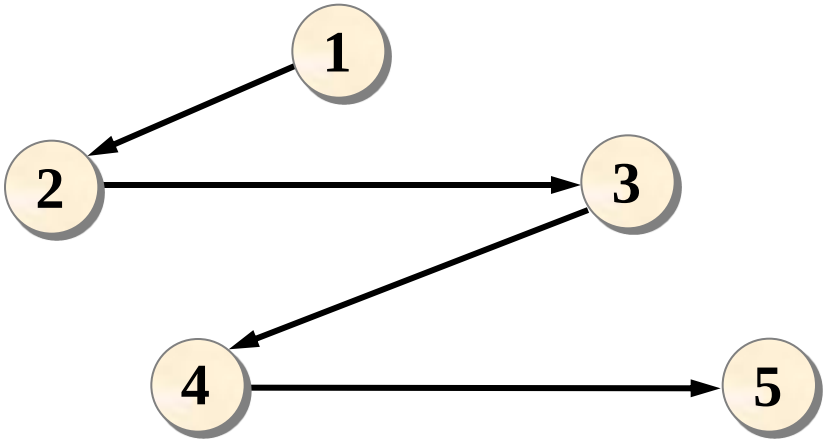
Chapter 8 图

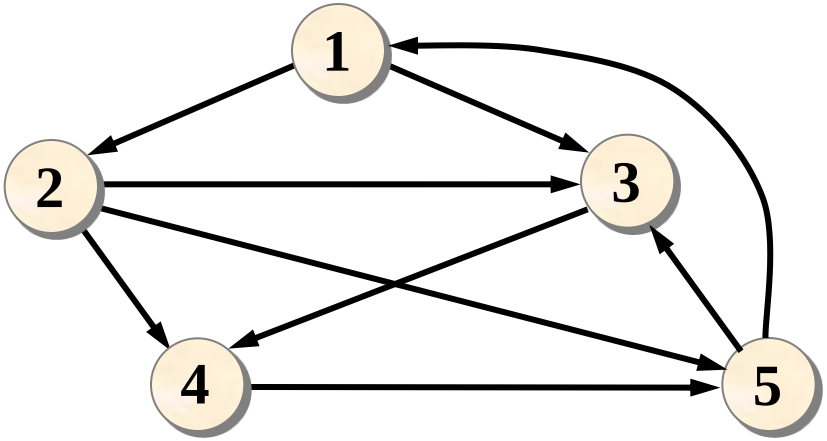
书P392：8.10



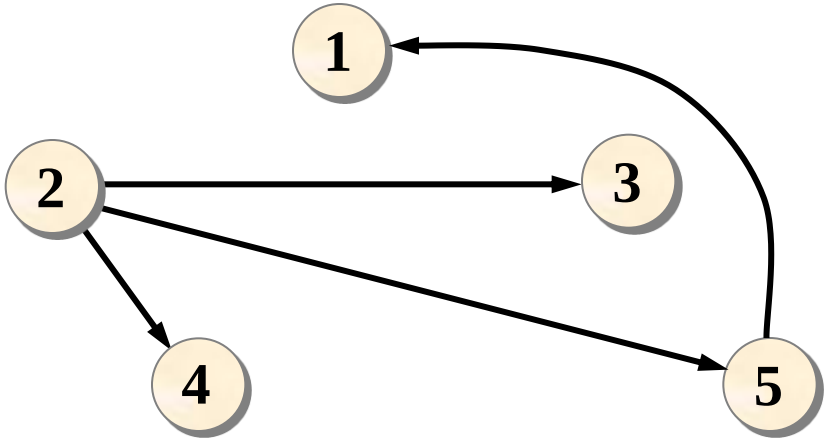
（1）从顶点1出发的深度优先生成树（不唯一）：



书P392：8.10

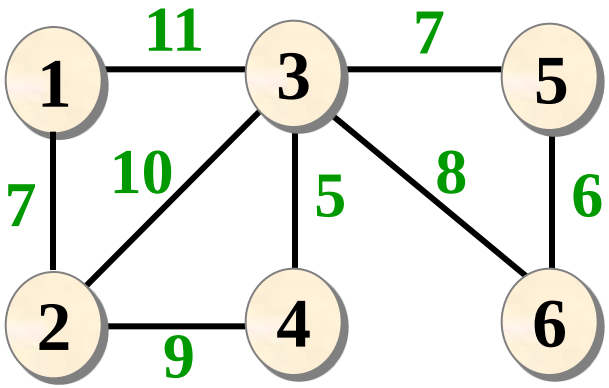


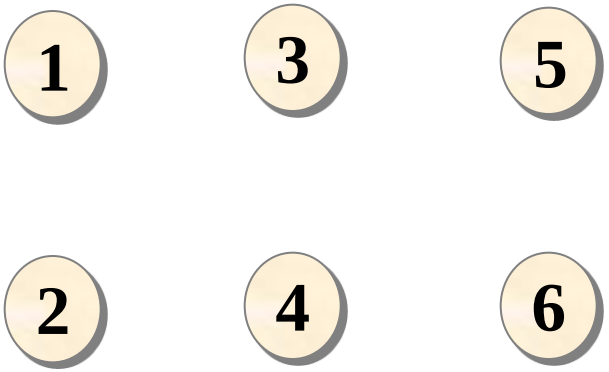
（2）从顶点2出发的广度优先生成树：



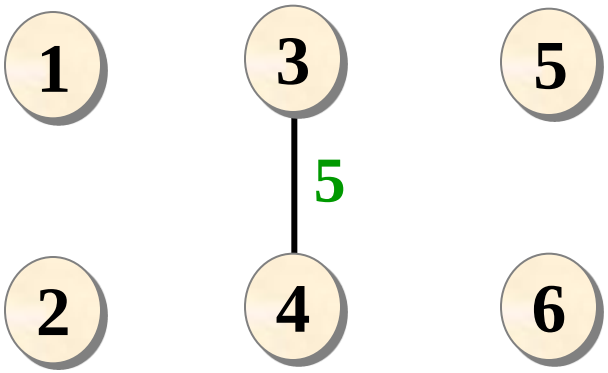
书P395：8.21

Kruskal算法构造最小生成树过程：

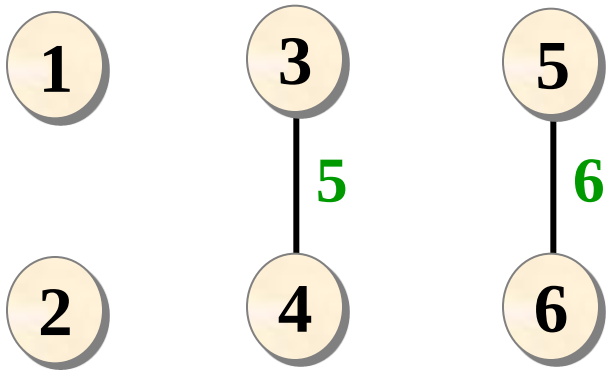




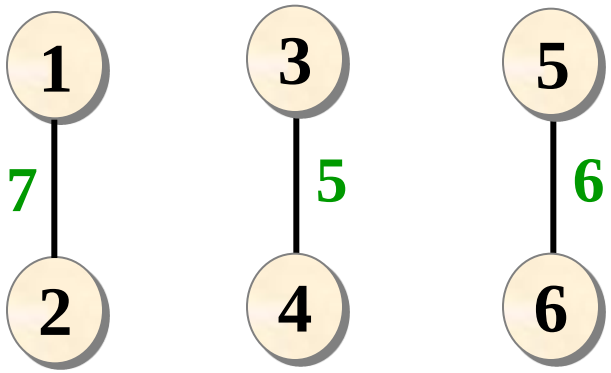
（a）



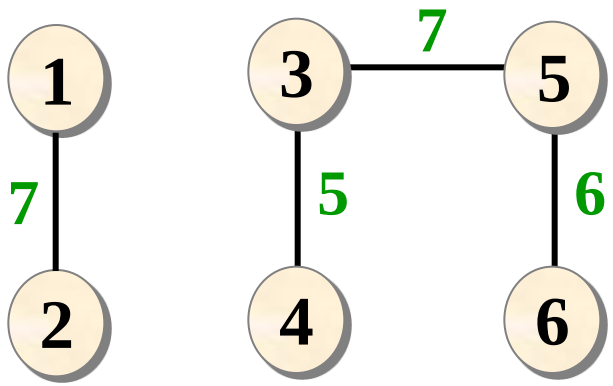
（b）



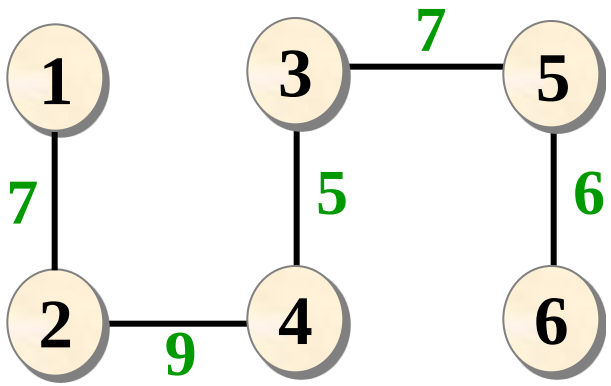
（c）



（d）



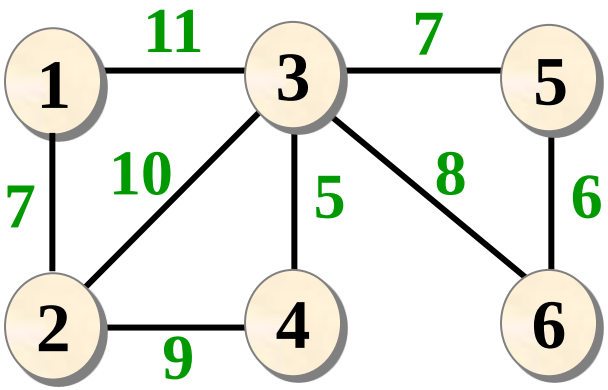
（e）



（f）

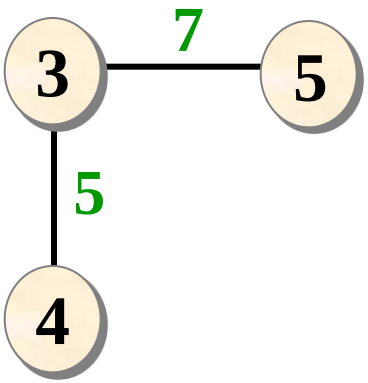
书P395：8.21

Prim算法（从顶点3出发）构造最小生成树过程：

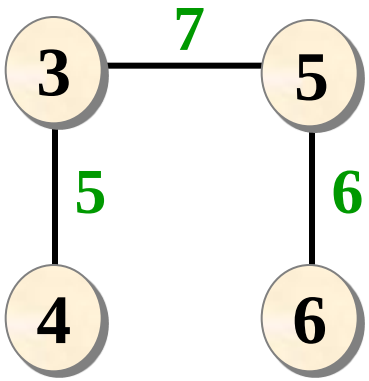


（a）

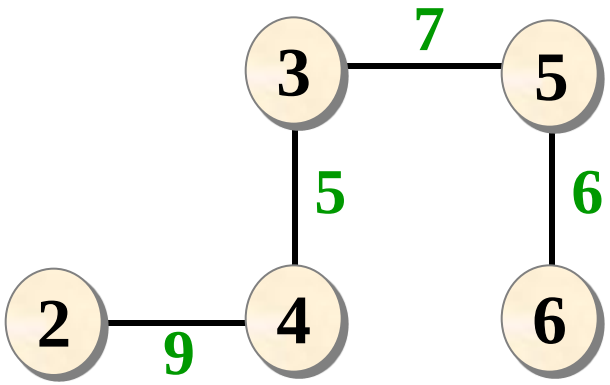
（b）



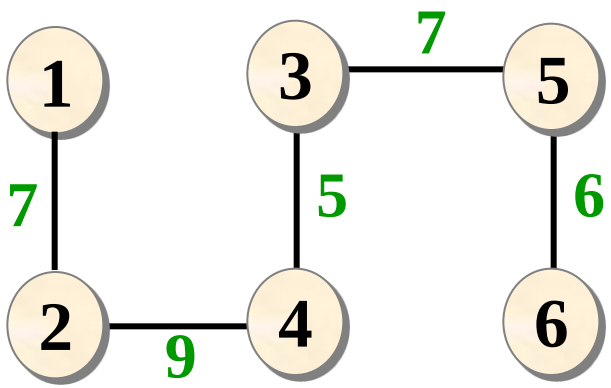
（c）



（d）



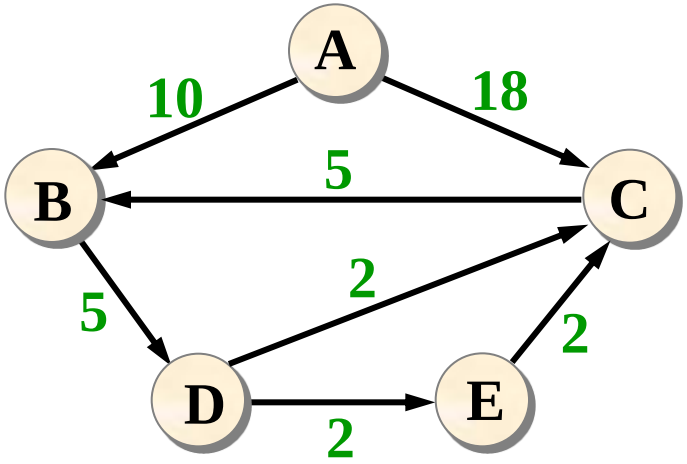
（e）



（f）

书P395： 8.24

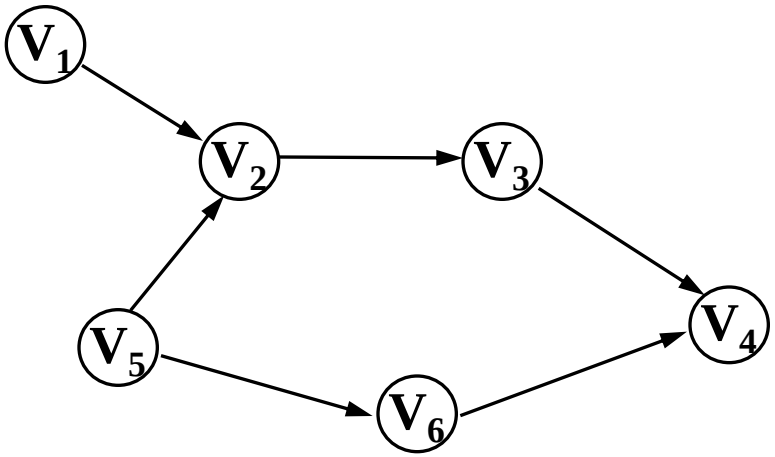
Dijkstra算法求解单源最短路径



终点	从源点A到各终点的dist值及最短路径			
B	10 〈A, B〉			
C	18 〈A, C〉	18	17 〈A, B, D, C〉	
D	∞	15 〈A, B, D〉		
E	∞	∞	17 〈A, B, D, E〉	17 〈A, B, D, E〉
V_u	B	D	C	E
S={A}	{A,B}	{A,B,D}	{A,B,D,C}	{A,B,D,C,E}

补充作业1：

列出下图中全部可能的拓扑有序序列，并指出应用拓扑排序算法求得的是哪一个（注意，应先确定其存储结构）。



拓扑有序序列为（7种）：

$V_1, V_5, V_2, V_3, V_6, V_4$　　$V_1, V_5, V_2, V_6, V_3, V_4$

$V_1, V_5, V_6, V_2, V_3, V_4$　　$V_5, V_1, V_2, V_3, V_6, V_4$

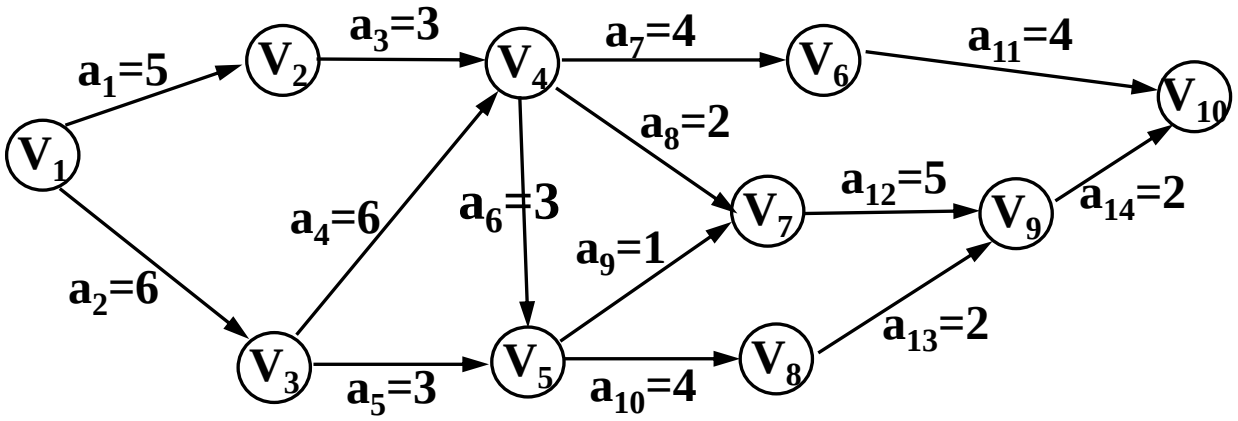
$V_5, V_1, V_2, V_6, V_3, V_4$　　$V_5, V_1, V_6, V_2, V_3, V_4$

$V_5, V_6, V_1, V_2, V_3, V_4$ （算法求得，堆栈）

若用队列存储入度为0的顶点，则算法求得的是$V_1, V_5, V_2, V_6, V_3, V_4$

补充作业2：

求关键路径。



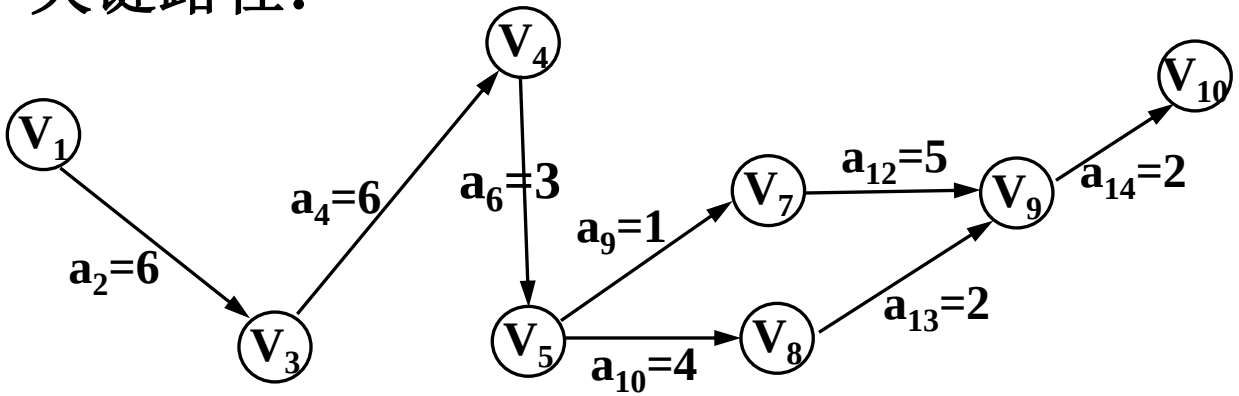
顶点	ve	vl
v_1	0	0
v_2	5	9
v_3	6	6
v_4	12	12
v_5	15	15
v_6	16	19
v_7	16	16
v_8	19	19
v_9	21	21
v_{10}	23	23

活动	e	l	l-e
a_1	0	4	4
a_2	0	0	0
a_3	5	9	4
a_4	6	6	0
a_5	6	12	6
a_6	12	12	0
a_7	12	15	3
a_8	12	14	2
a_9	15	15	0
a_{10}	15	15	0
a_{11}	16	19	3
a_{12}	16	16	0
a_{13}	19	19	0
a_{14}	21	21	0

答：① 完成该工程的最短时间是23。

② 关键活动： $a_2, a_4, a_6, a_9, a_{10}, a_{12}, a_{13}, a_{14}$

关键路径：



提高活动a_2, a_4, a_6, a_{14}或同时提高a_9（或a_{12}）与a_{10}（或a_{13}）的进度可缩短整个工程的工期。

补充题：在带权有向图G中计算顶点v的入度和出度

```
template <class T, class E>
int Graphlnk<T, E>::InDegree(int v) {
//计算顶点v 的入度，若顶点不存在，则函数返回-1
   if (v != -1) {                  //顶点v存在
      int id=0;  Edge<T, E> *p;
      for (int i = 0; i < numVertices; i++ ) {
          if ( i == v)  continue;
          p = NodeTable[i].adj;  //对应边链表第一个边结点
          while (p != NULL) {   //统计邻接点域为v的边结点个数
              if ( p->dest == v )  { id++;  break; }
              p = p->link;  }
      }
      return id;
   }
   return -1;              //顶点v不存在
}
```

```
template <class T, class E>
int Graphlnk<T, E>::OutDegree(int v) {
//计算顶点v 的出度，若顶点不存在，则函数返回-1
    if (v != -1) {                    //顶点v存在
        int od=0;
        Edge<T, E> *p = NodeTable[v].adj;
        while (p != NULL) {      //统计第v个边链表的结点个数
              od++;
              p = p->link;
        }
        return od;
    }
    return -1;            //顶点v不存在
}
```

补充题：输出图G中从顶点vi 到顶点 vj 的所有简单路径

```
template <class T, class E>
void DFSearch (Graph<T, E>& G, int vi, int vj, bool visited[],
     int path[], int k) {
//用深度优先遍历在图G中寻找从vi到vj的简单路径，数组path[]记录
//路径上顶点序列，k是path[]中当前可存放位置，初次调用时k为0
   visited[vi]=true;   path[k]=vi;       //访问顶点vi并加入路径
   int i, w = G.getFirstNeighbor (vi);
   while (w!=-1) {
       if (!visited[w])
           if (w==vj) {                          //当此顶点为路径的终点
               for (i=0; i<=k; i++) cout<<path[i];
               cout<<w<<endl;              //输出一条路径
           }
           else DFSearch(G, w, vj, visited, path, k+1);
       w = G.getNextNeighbor (vi, w);
   }
   visited[vi] = false;        //将不在此路径中的顶点重置为未访问
}
```