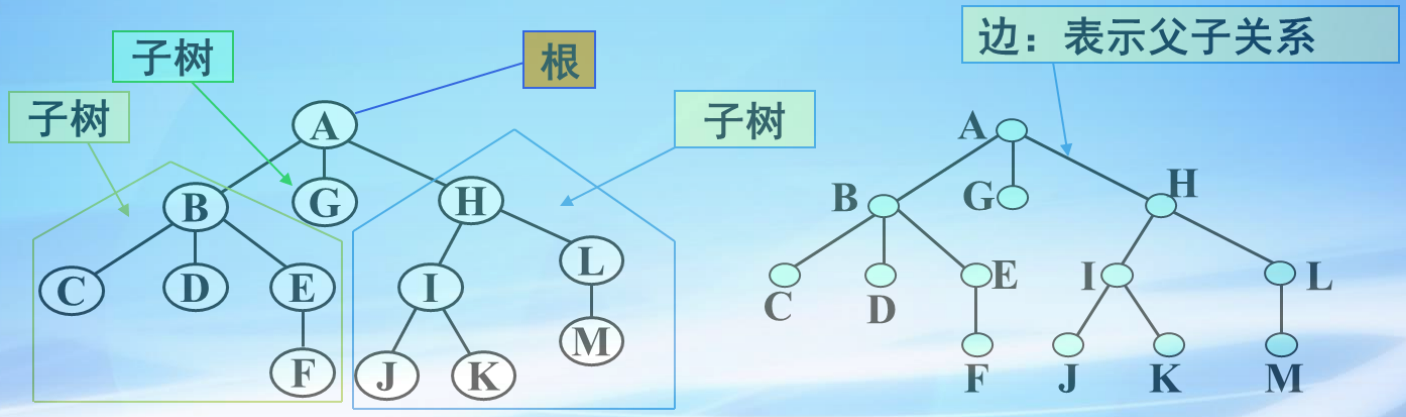
《数据结构》

最小生成树：Prim算法



最小生成树：Prim算法

教学目标和要求

1.能够运用Prim算法思想图解MST的过程

2.能够编程实现Prim算法

3.理解Prim算法与Kruskal算法的区别

Prim算法的基本思想

采用子树延伸法，每次选择最小边的顶点及边加入

将顶点分成两类：
生长点——已经在生成树上的顶点
非生长点——未长到生成树上的顶点

使用待选边表
每个非生长点在待选边表中有一条待选边，一端连着非生长点，另一端连着生长点

Prim算法描述

Prim算法实现步骤

步骤1）构造初始待选边表。

任选一个顶点v作为初始生长点，对其余每个非生长点w（共n-1个），将边（v，w）加进待选边表；如果边（v，w）不存在，则认为边（v，w）的长度是∞

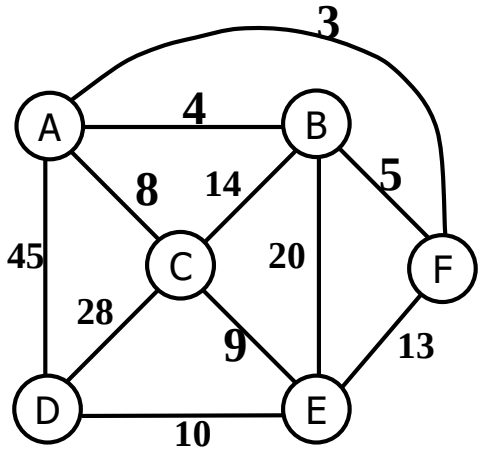
选边表：(v,u,cost)

Prim算法描述

Prim算法实现步骤

步骤1）构造初始待选边表。

任选一个顶点v作为初始生长点，对其余每个非生长点w（共n-1个），将边（w，v）加进待选边表；如果边（w，v）不存在，则认为边（w，v）的长度是∞



非生长点

初始待选边表
(以B为初始生长点)

生长点

B	B	B	B	B
A	C	D	E	F
4	14	∞	20	5
0	1	2	3	4

边长

Prim算法描述

Prim算法实现步骤

步骤2）循环n-2遍，使非生长点个数k从n-1变到1。

（1）选择树边

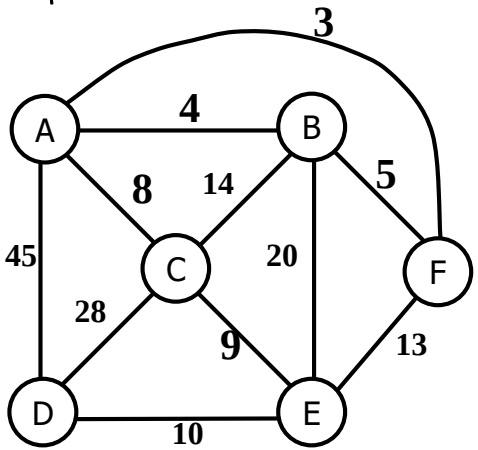
从待选边表中选出一条最短的待选边（v，u）（这里u是非生长点，v是生长点），将（v，u）从待选边表移入生成树的边集，并且将u作为新选出的生长点

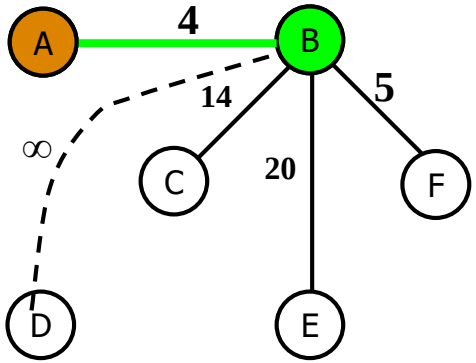
（2）修改待选边

对剩下的每个非生长点w，比较待选边（w，x）与边（w，u）的长度（这里x是原有的生长点，u是新选出的生长点）；如果（w，u）短于边（w，x），则用（w，u）代替（w，x），作为w的待选边；否则，什么也不做

Prim算法运行示例

Prim算法示例（1）





以A为生长点替换待选边

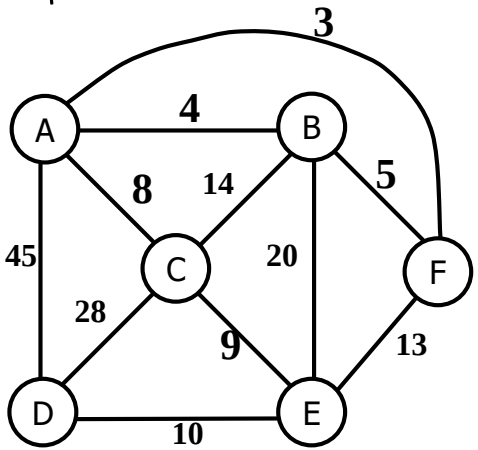
初始待选边表(以B为初始生长点)

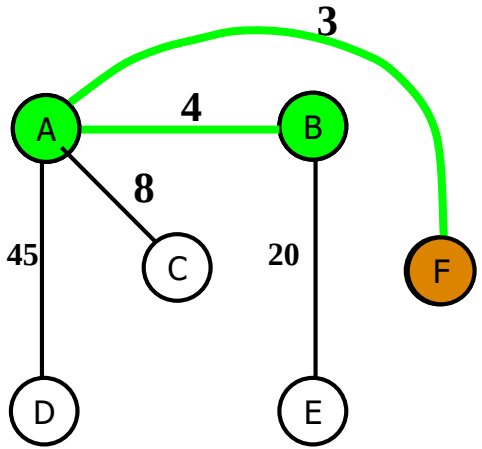
B	B	B	B	B
A	C	D	E	F
4	14	∞	20	5
0	1	2	3	4

B	A	A	B	A
A	C	D	E	F
4	8	45	20	3
0	1	2	3	4

Prim算法运行示例

Prim算法示例（2）





以F为生长点替换待选边

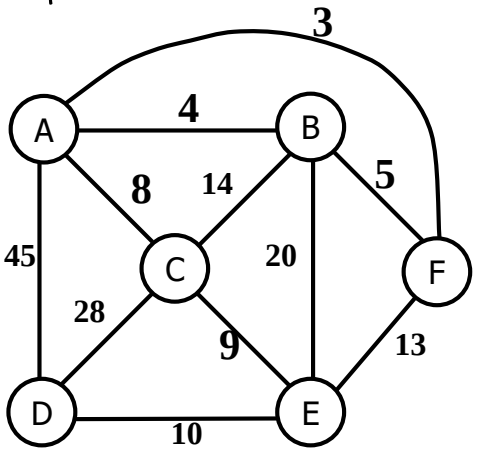
初始待选边表(以B为初始生长点)

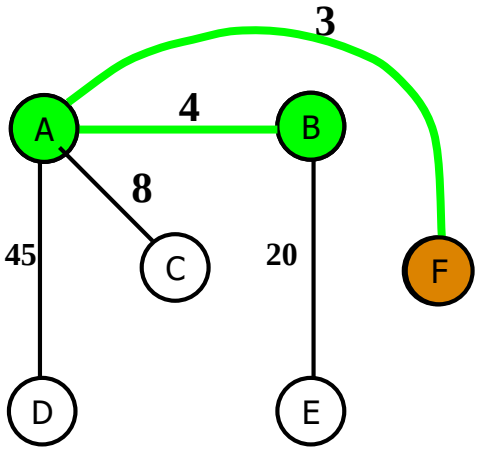
B	A	A	B	A
A	C	D	E	F
4	8	45	20	3
0	1	2	3	4

B	A	A	B	A
A	F	D	E	C
4	3	45	20	8
0	1	2	3	4

Prim算法运行示例

Prim算法示例（2）





以F为生长点替换待选边

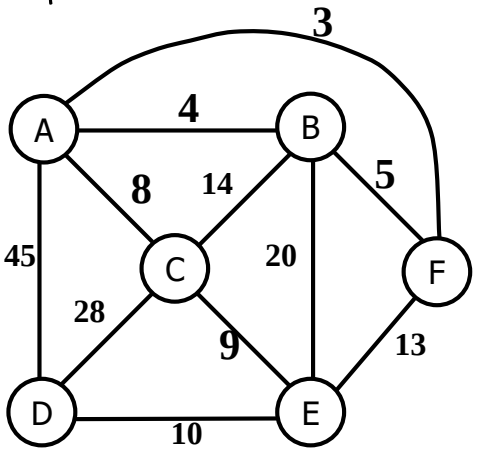
初始待选边表(以B为初始生长点)

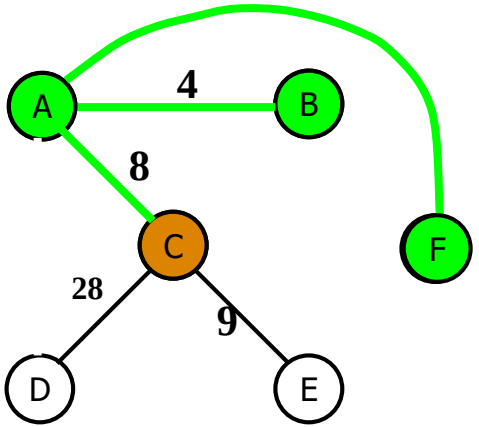
B	A	A	B	A
A	C	D	E	F
4	8	45	20	3
0	1	2	3	4

B	A	A	F	A
A	F	D	E	C
4	3	45	13	8
0	1	2	3	4

Prim算法运行示例

Prim算法示例（3）





初始待选边表(以B为初始生长点)

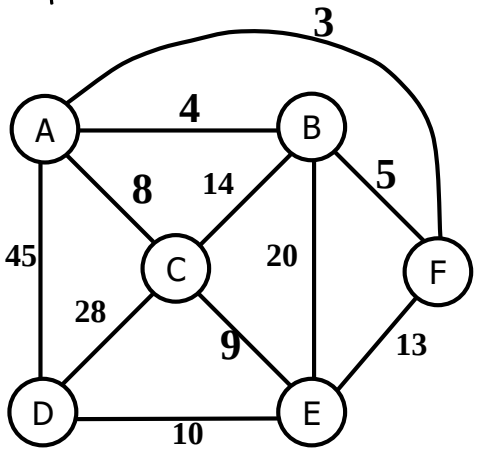
B	A	A	F	A
A	F	D	E	C
4	3	45	13	8
0	1	2	3	4

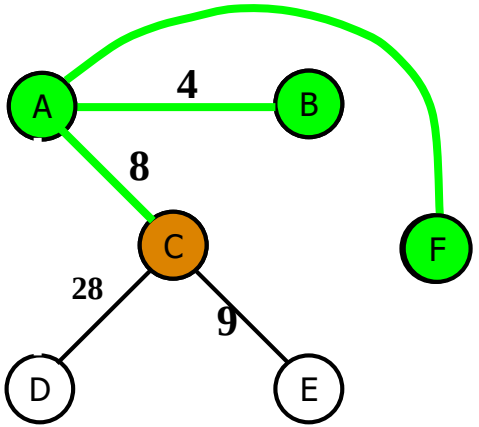
以C为生长点替换待选边

B	A	A	F	A
A	F	C	E	D
4	3	8	13	45
0	1	2	3	4

Prim算法运行示例

Prim算法示例（3）





初始待选边表(以B为初始生长点)

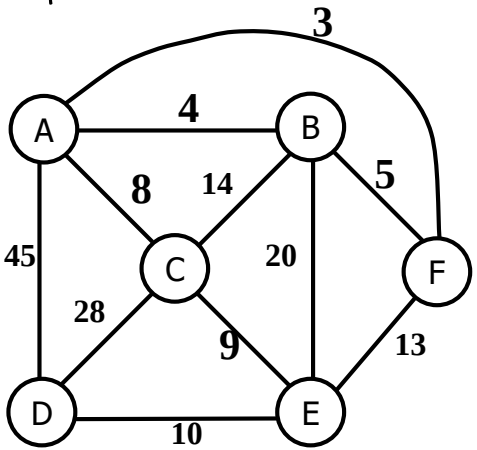
B	A	A	F	A
A	F	D	E	C
4	3	45	13	8
0	1	2	3	4

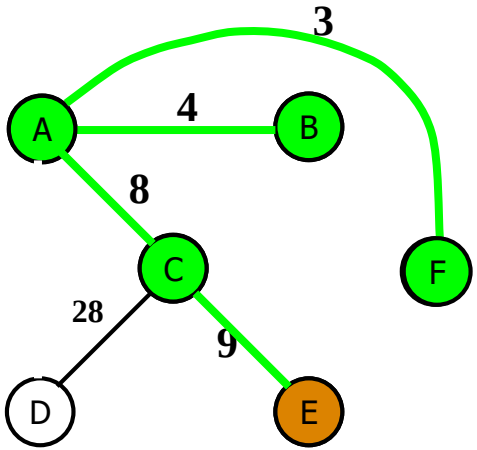
以C为生长点替换待选边

B	A	A	C	C
A	F	C	E	D
4	3	8	9	28
0	1	2	3	4

Prim算法运行示例

Prim算法示例（4）





以E为生长点替换待选边

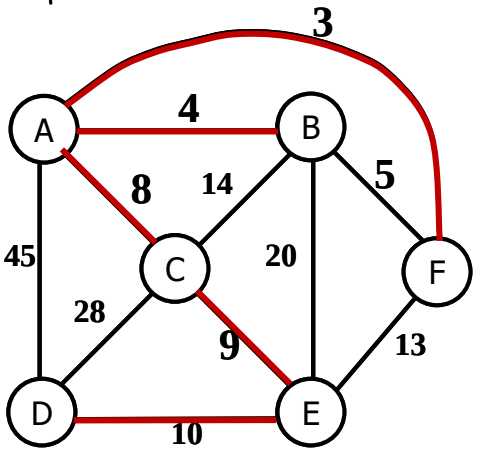
初始待选边表(以B为初始生长点)

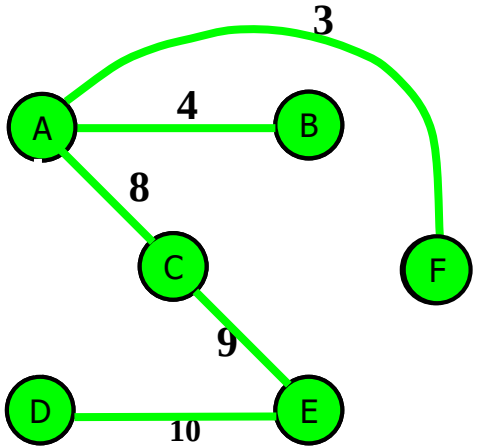
B	A	A	C	C
A	F	C	E	D
4	3	8	9	28
0	1	2	3	4

B	A	A	C	E
A	F	C	E	D
4	3	8	9	10
0	1	2	3	4

Prim算法运行示例

Prim算法示例（4）





初始待选边表(以B为初始生长点)

B	A	A	C	E
A	F	C	E	D
4	3	8	9	10
0	1	2	3	4

最后的顶点D作为生长点，边作为树边

B	A	A	C	E
A	F	C	E	D
4	3	8	9	10
0	1	2	3	4

如何编程实现?

- 如何表示树边和待选边?

树边与待选边本质上时一样的，因此，定义边结点结构**edge**，包括生长点、非生长点和边长度三个域。

- 如何存储树边和待选边?

可以定义长度为**n-1**的表结构，二者共享表空间。

- 如何存储图?

为便于快速获取边长，图的存储结构为邻接数组。

回答了这些问题，就可以写出具体的代码了。

Prim算法的实现

Prim算法实现方法

```
typedef  struct  edge_node
  { int  incr_vert ,vertex;  //生长点，非生长点域
    int  cost;          //边长度域
  }edge;  //待选边类型名（三元组）
void  Prim2(int c[M][M],int n) //假定顶点0作为初始生长点
 { int v, i, j, k;          edge  t, wait[M-1]; //存储待选边表的数组
   for (v=0;v<n-1;v++)              //以顶点0作为初始生长点建立初始待选边表
   {  wait[v].incr_vert=0;          //生长点
      wait[v].vertex=v+1;           //非生长点
      wait[v].cost=c[0][v+1];
   }
    //接下页
```

Prim算法的实现

Prim算法实现方法

```
//接上页
for(i=0;i<n-2;i++) {
    k=i;  //找最短的待选边（简单选择）
    for(j=i+1;j<n-1;j++) if(wait[j].cost<wait[k].cost)k=j;
    t=wait[k];  wait[k]=wait[i];  wait[i]=t;          //得到一条树边，换到数组前部
    v=wait[i].vertex;                                 //v作为新的生长点
    for(j=i+1;j<n-1;j++)                              //修改待选边表
      if(wait[j].cost>c[v][wai[j].vertex]) {  wait[j].cost=c[v][wait[j].vertex];
                                              wait[j].incr_vert=v;  }
  }
 printf("  最小生成树的边集为：\n ");
 for(i=0;i<n-1;i++)
    printf("(%d, %d, %d)  ",wait[i].incr_vert,wait[i].vertex,wait[i].cost);
 printf("\n");
}
```

Prim算法和Kruskal算法的比较

设计思想—贪心法

都是选短边，但选法不同
Kruskal算法从全图中选短边，Prim算法从待选边表中选短边

直观性

Kruskal采用子树合并法
Prim算采用子树延伸法

实现难易程度

Kruskal算法需要判断回路（实现困难些）
Prim算法不需要判断回路

Prim算法和Kruskal算法的比较

时间复杂性

Kruskal算法执行时间主要花费在判断回路，所需时间不超过O(mlogm)，m是边数

Prim算法执行时间主要花费在n-2次修改待选边表，其时间耗费用量是$O(n^2)$阶的，n是顶点数

Kruskal算法适用于**顶点数较多，而边数较少**的情况

Prim算法适用于**顶点数较少，而边数较多**的情况

为什么Prim和Kruskal算法能取到MST？

MST的性质

设G=(V,E)是无向连通加权图。U是V的非空子集，（u，v）的权最小，u∈U，v∈V-U，则必存在一棵包含（u，v）的MST

反证！

The End, Thank You!