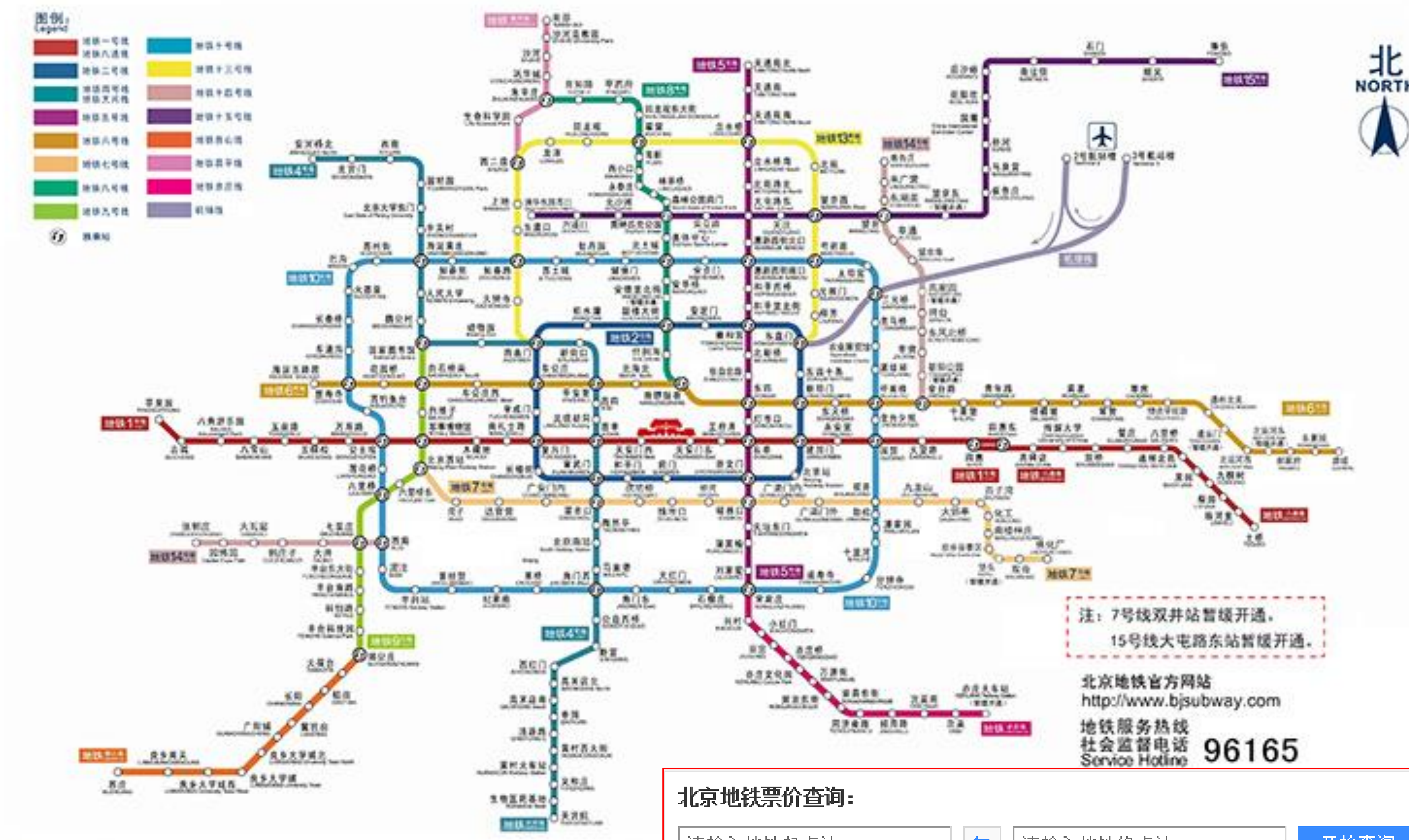
内容回顾

- 图的概念：度、路径、连通、连通分量、生成树
- 图的存储：邻接矩阵、邻接表
- 图的遍历：广度优先遍历、深度优先遍历
  - ✓ 设标志信息以防止顶点被重复访问
- 独立路径：基于遍历 + 允许顶点被重复访问 + 无环路

北京地铁线路图
Beijing Subway Map



北京地铁票价查询：

请输入地铁起点站 ⇆ 请输入地铁终点站 开始查询

详情：起步6公里内每人次3元，6-12公里每人次4元，12-32公里每10公里加1元，32公里以上每20公里加1元，票价不封顶。（北京市发改委）

问题6.1：北京地铁乘坐线路查询

- **编写程序实现北京地铁乘坐线路查询，输入为起始站和终点站名，输出为从起始到终点的换乘线路。要求给出：**
  - 乘坐站数最少的换乘方式（理论最快，通常用于计算票价）。如从西土城到北京西站：

    西土城-10-黄庄-4-国家图书馆-9-北京西站

  - 换乘次数最少的换乘方式（最方便）。如从从西土城到北京西站：

    西土城-10-六里桥-9-北京西站

最短路径问题

6.4 最短路径问题

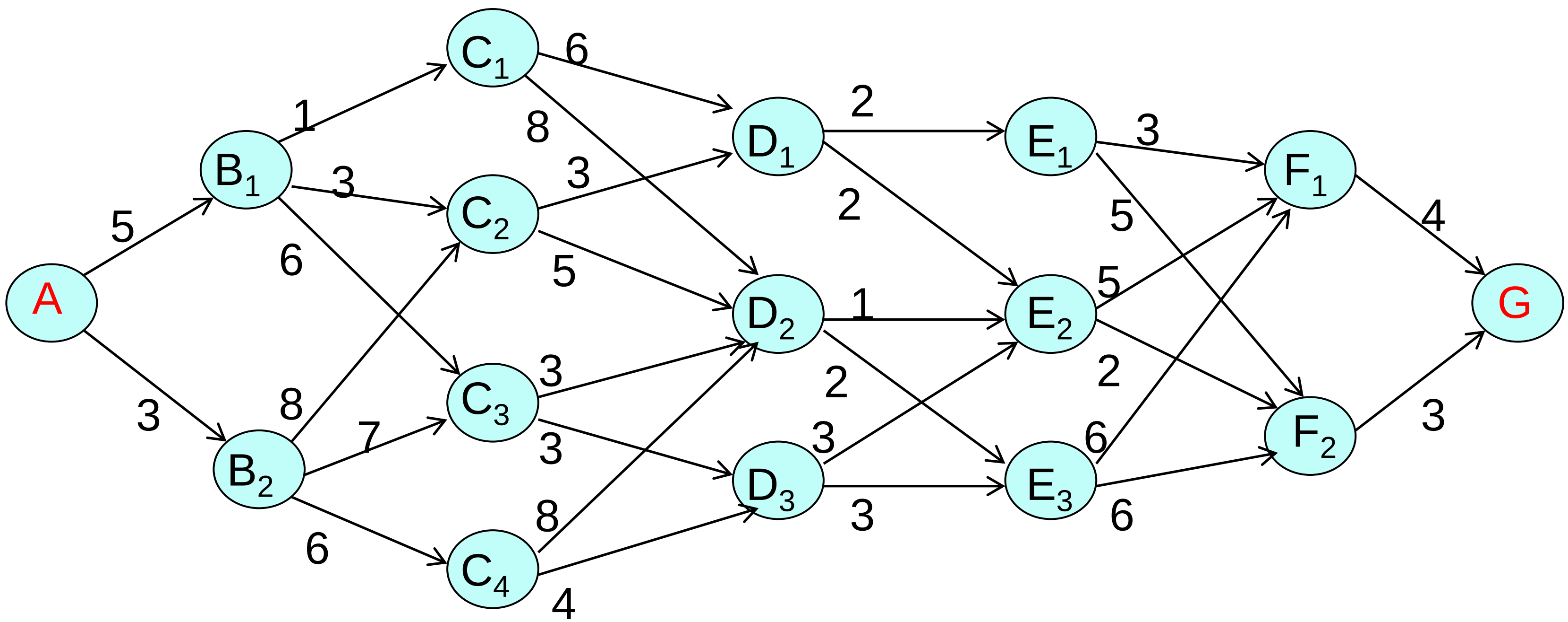
不带权的图　　路径上所经过的边的数目。

带权的图　　路径上所经过的边上的权值之和。

源　点： 如A

目的点： 如G

1. 单源点最短路径；
2. 每对顶点之间的最短路径；
3. 求图中第1短、第2短、…的最短路径。

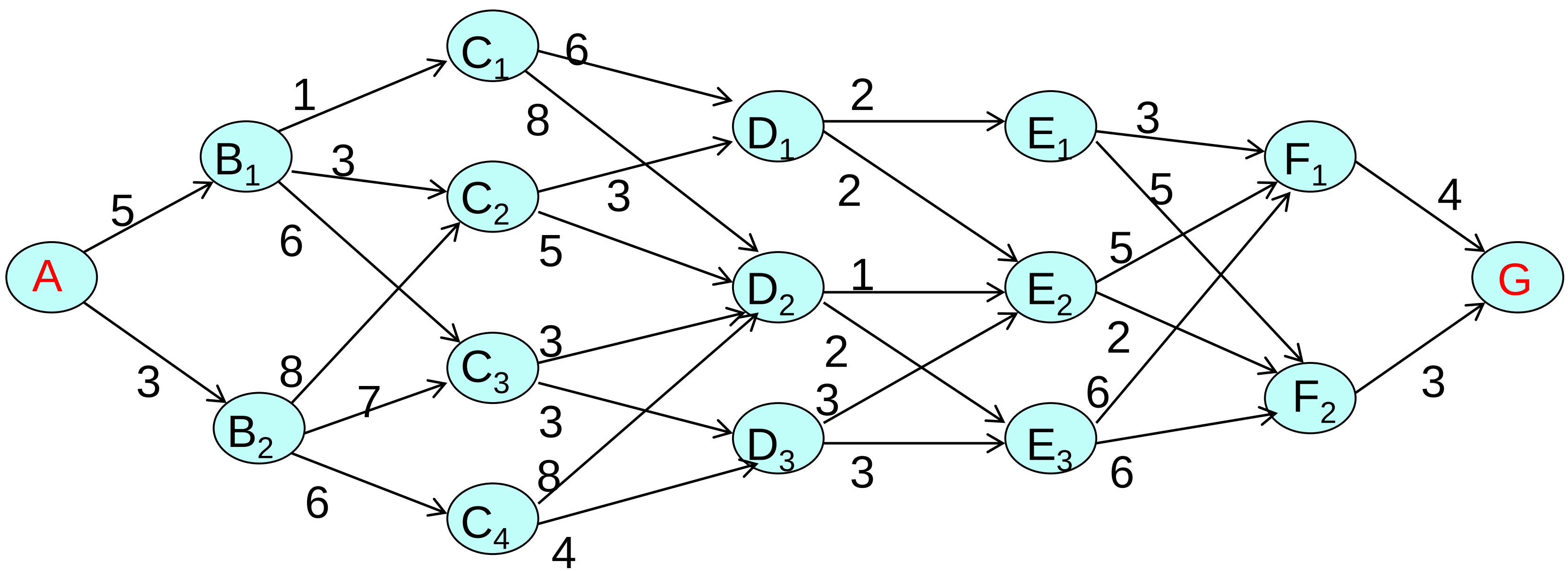


单源最短路径的求解方法

注意：遍历中需允许顶点被重复访问，同时要避免出现环路。

1. 基于深度优先遍历的方法
2. 基于广度优先遍历的方法
3. 基于动态规划的方法* 如 dist(A) = min(5+dist(B_1), 3+dist(B_2))
4. 基于贪心策略的方法：迪杰斯特拉算法

   可同时计算出源点至每个顶点的最短路径



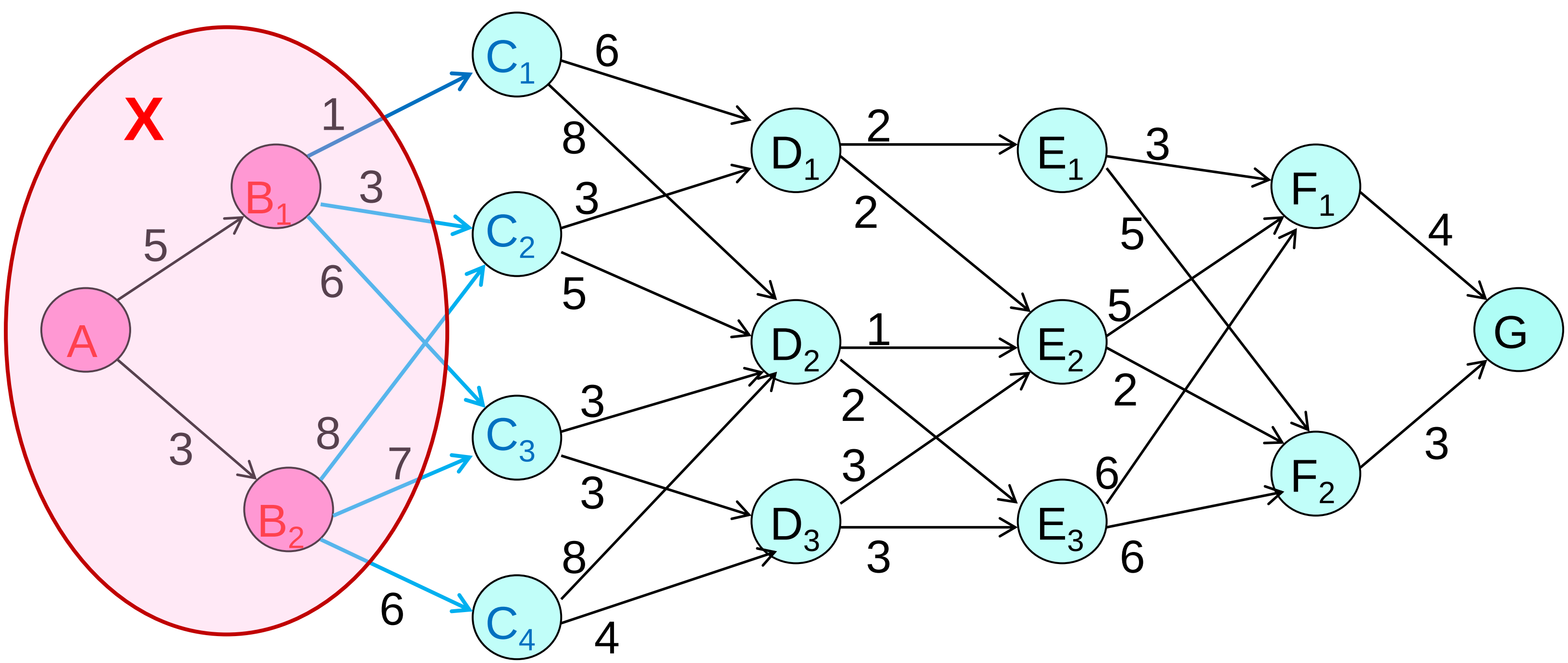
Dijkstra算法将顶点V分为两个集合X和Y=V-X.

X包含至源点的距离已经确定的顶点，初始X={源点}

Y是未确定最短路径的顶点集合

dist[v]，它是v只经过X中顶点至源点的最短路径长度

如 dist[c2] = min{3+5, 3+8} = min{8, 11} = 8

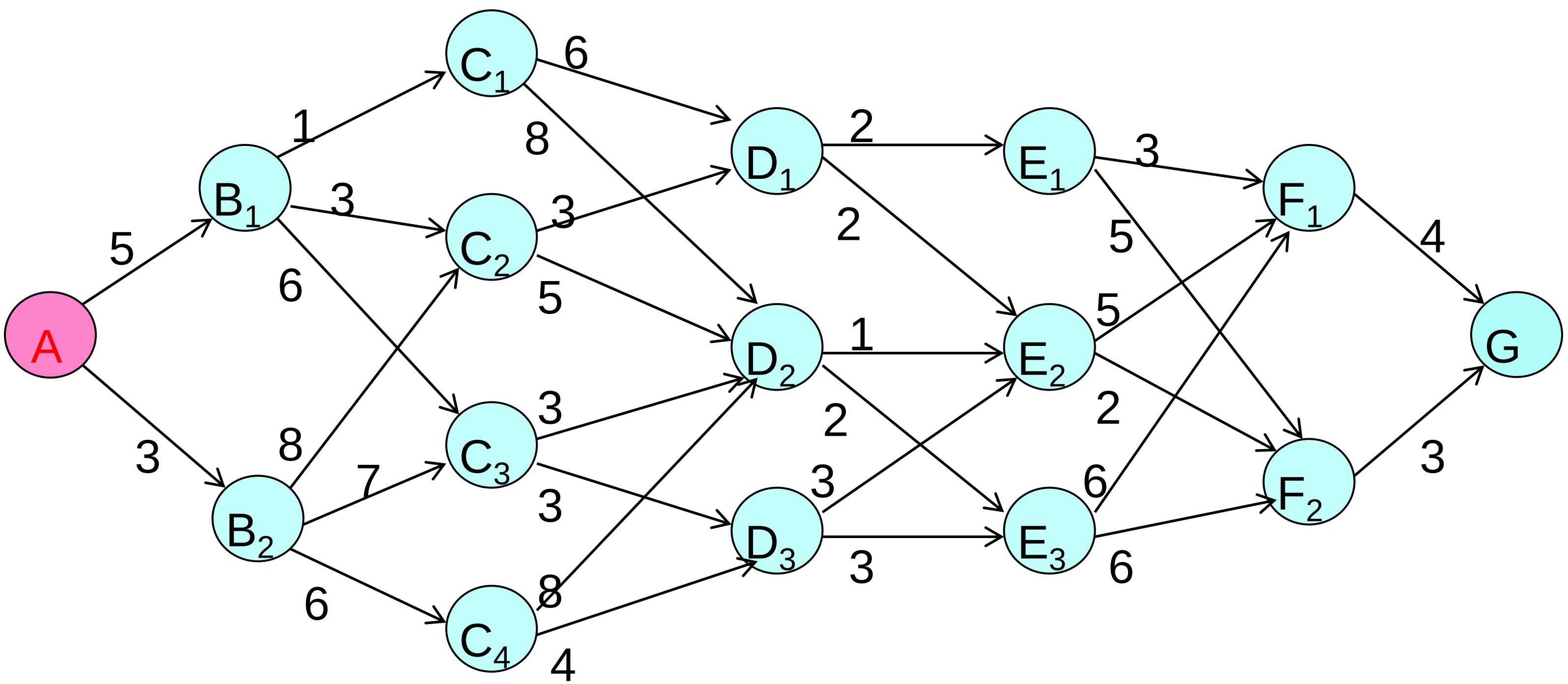


Dijkstra算法(简化描述)

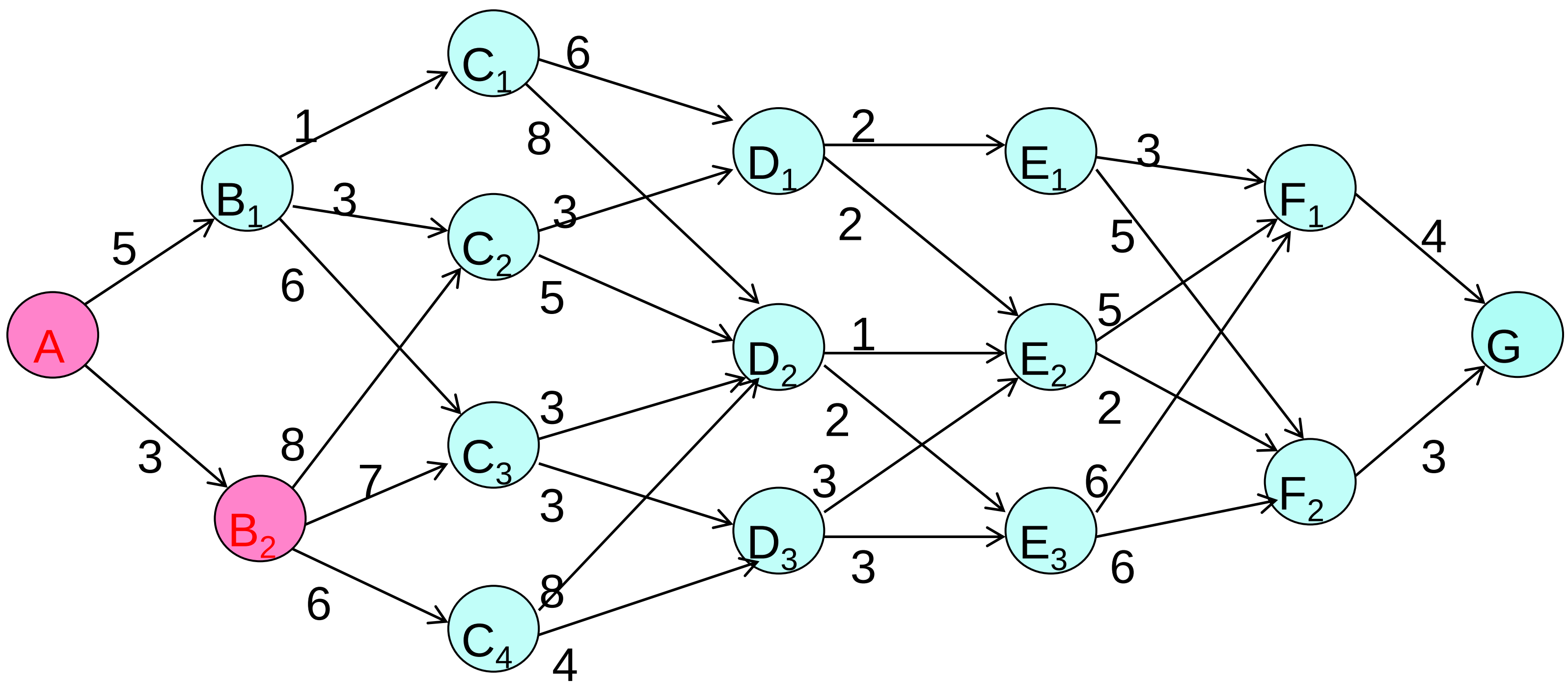
- 初始时，将顶点V分为两个集合X和Y=V-X.
  - X包含至源点v1的距离已经确定的顶点，初始X={v1}
  - Y中每个顶点**v**有标记dist[v]， 它是只经过X中顶点后到达v的最短路径长度
- **依次在Y中选择一个至源点距离最短的顶点*v*（即dist[*v*]最小），并将该顶点移到X中**
  - 一旦顶点*v*∈*Y*移到*X*中，则**更新***v*相邻的每个顶点*w*∈*Y* 的距离dist[*w*]
- **在算法结束时，对于每个顶点*v*∈V，路径长度dist[*v*]为源点到该顶点的距离**

注：Dijkstra算法采用了贪心(greedy)策略，是贪心算法。其正确性需严格证明。

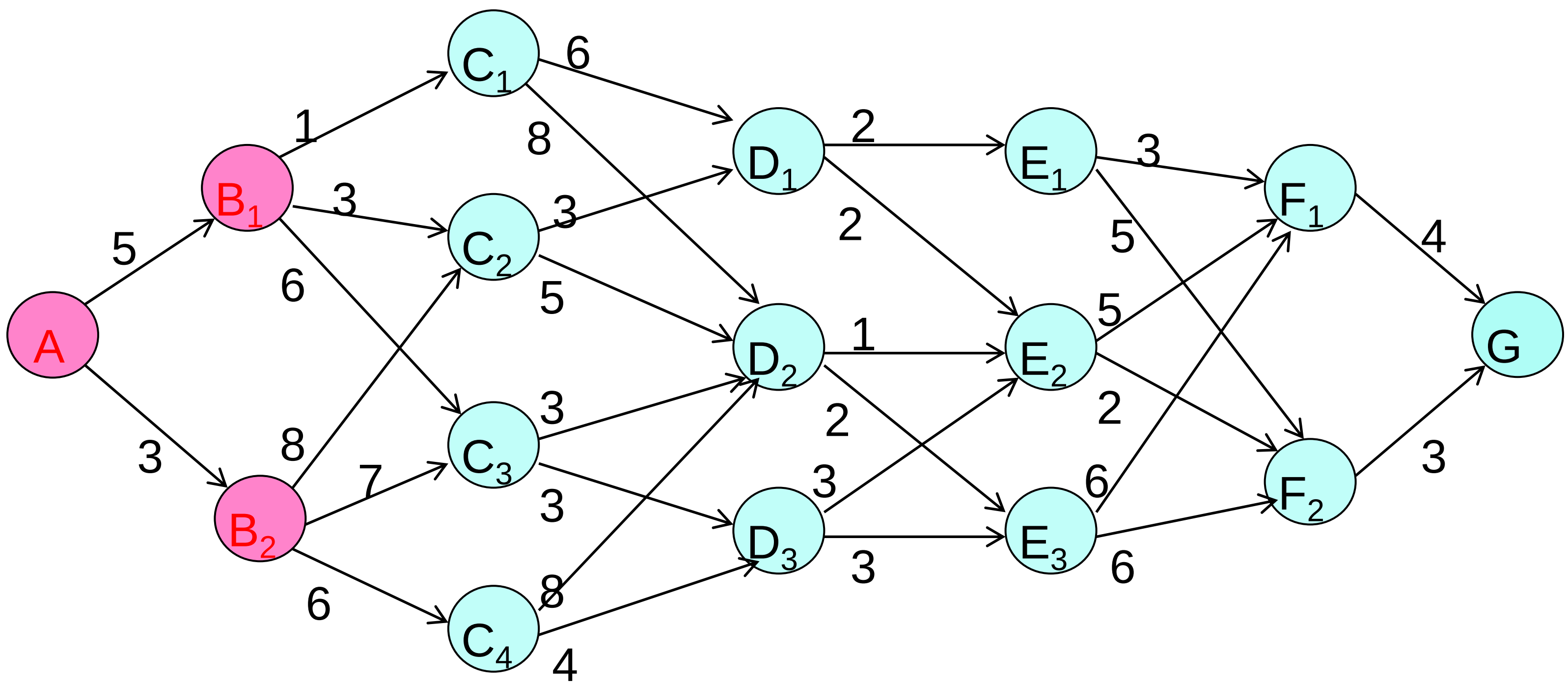
	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
dist	0	5	3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞



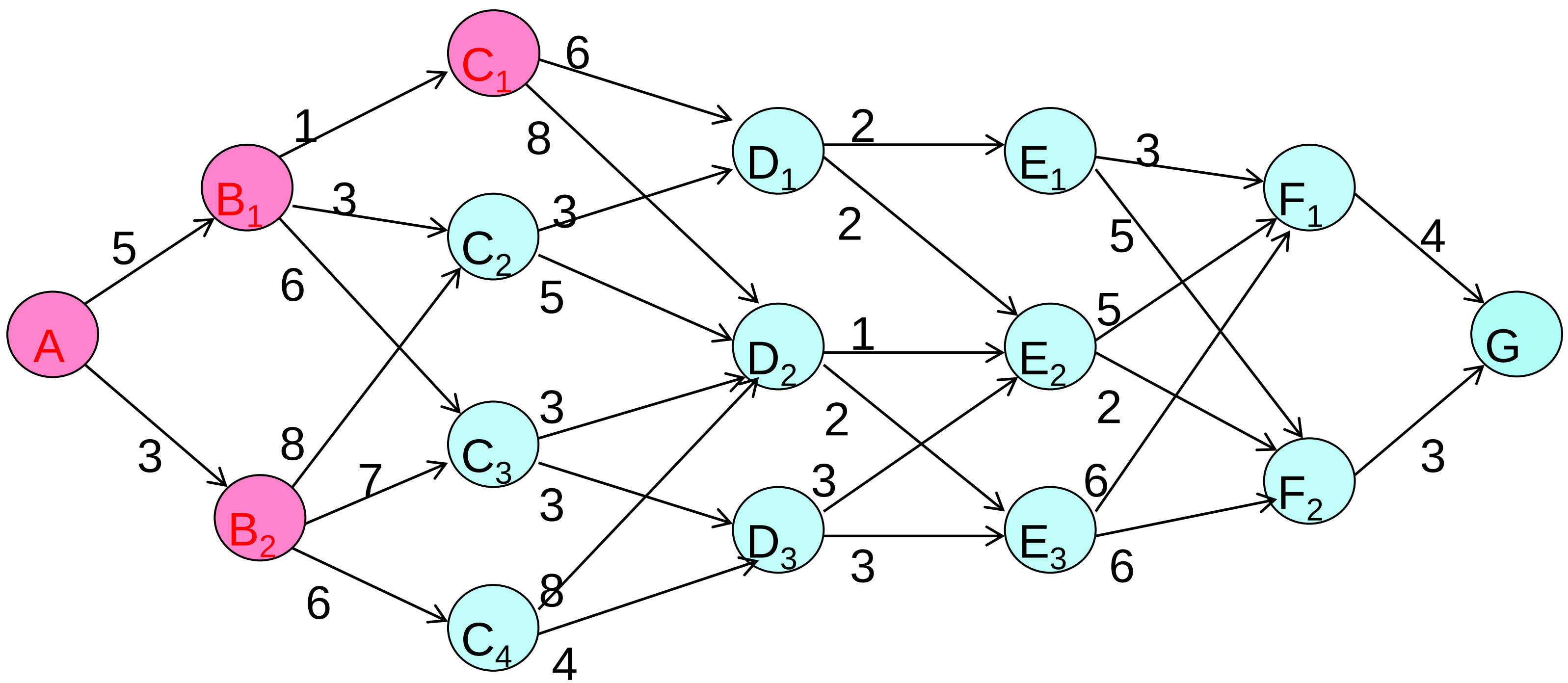
	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
dist	0	5	3	∞	11	10	9	∞	∞	∞	∞	∞	∞	∞	∞	∞



	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
dist	0	5	3	6	8	10	9	∞	∞	∞	∞	∞	∞	∞	∞	∞



	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
dist	0	5	3	6	8	10	9	12	14	∞	∞	∞	∞	∞	∞	∞

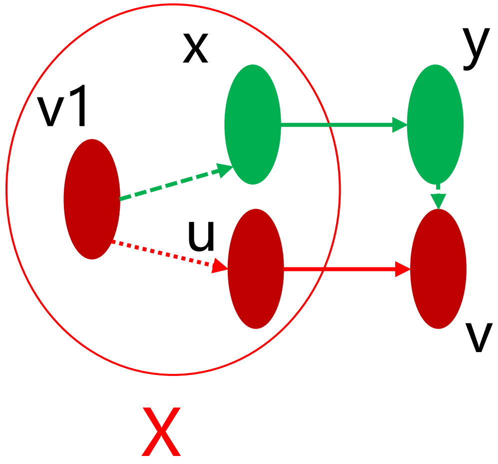


Dijkstra算法正确性证明（归纳+反证）*

- **初始时，将顶点V分为两个集合X和Y=V-X.**
  - X包含至源点v1的距离已经确定的顶点， 初始X={v1}
  - Y中每个顶点 *v* 有标记dist[v]，它是只经过X中顶点后到达v的最短路径长度
- **依次在Y中选择一个至源点距离最短的顶点 *v*.**

Key：令**v在X中的邻接点为u，**证明边(u, v)在v1到v的最短路径上

假设边(u, v)不在v1到v的最短路径上，则必存在其他更短的路径v1,…,x,y, …, v. 按算法原理，则v1, …, x, y的长度一定不小于v1, …, u, v的长度，与假设矛盾。



Dijkstra算法(C语言)

最短路径数组Spath含义：Spath[v]表示顶点v在最短路径上的直接前驱顶点。
假设某最短路径由顶点v1, v2, v3组成，则有：Spath[v3]==v2，Spath[v2]==v1。

int Weigths[VNUM][VNUM]为邻接矩阵，表示顶点间权重/距离

//设v_1为源顶点，定义如下辅助数据结构：

int **wfound**[VNUM]; //表示顶点是否已确定最短路径(0未确定,1已确定)

int **dist**[VNUM]; //表示顶点到起始点的最短距离

int **Spath**[VNUM]={0}; //表示最短路径，只记前驱结点

0.初始化: wfound[v1] = 1;
dist[v1]=0; dist[i] = Weigths[v1][i];

重复n–1次

1.**挑选**：查找与v1间距离最小且没有确定最短路径的顶点v, 即在dist数组中查找权重最小且没有确定最短路径的顶点。

2.**标记**：标记v为已找到最短路径的顶点

3.**更新**：对于图G中每个从顶点v1到其最短路径还未找到，且存在边(v,w)，如果从v1通过v到w的路径权值小于它当前的权值，则更新w的权值为：v的权值+边(v,w)的权值，即：

dist[w] = dist[v]+Weights[v][w]

```
Dijkstra( int  v0){
    int i, j, v, minDist;
    char  wfound[VNUM] = { 0 }; //标记从v1到相应顶点是否找到最短路径，0未找到，1找到
    //初始化
    for(i=0; i<VNUM; i++) { dist[i] = Weights[v1][i]; Spath[i] = v1; }
    dist [v1] = 0;
    wfound [v1] = 1;
    //增量更新
    for(i=0; i< VNUM-1; i++) {//迭代VNUM-1次
       minDist =   INFINITY;
       for(j=0; j < VNUM;  j++)   //1.挑选一个新的结点v
          if( !wfound[j] && ( dist[j] < minDist) ) {
             v = j;          minDist = dist[v];
          }
       wfound[v] = 1;              //2.标记该顶点为已找到最短路径
       for(j =0; j<VNUM; j++)   //3.更新v的邻居结点的最短路径及长度
          if( !wfound[j]  &&  (minDist + Weights[v][j] < dist[j] )) {
             dist[j] = minDist + Weights[v][j];  //更新邻居顶点的距离
             Spath[j] = v;                              //记录其前驱顶点，以构成路径
          }
    }
}
```

时间复杂度：$O(|V|^2)$

小根堆优化：$O(|V|lg|V|+|E|)$

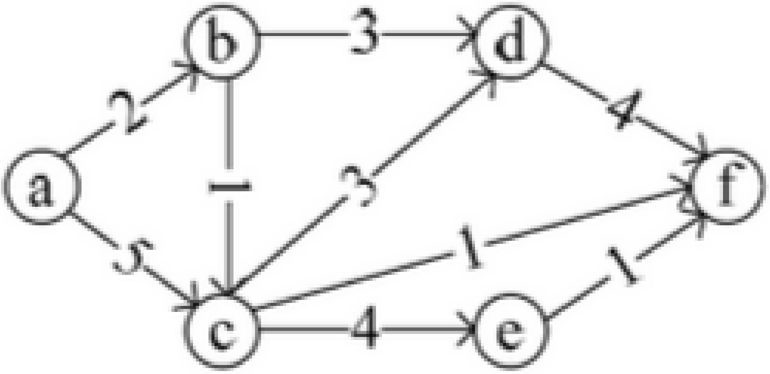
一有向带权图如下图所示，若采用迪杰斯特拉（Dijkstra）算法求源点a到其他各顶点的最短路径，得到的第一条最短路径的目标顶点是b，后续得到的其余各最短路径的目标顶点依次是________。

A. c, d, e, f

B. c, e, d, f

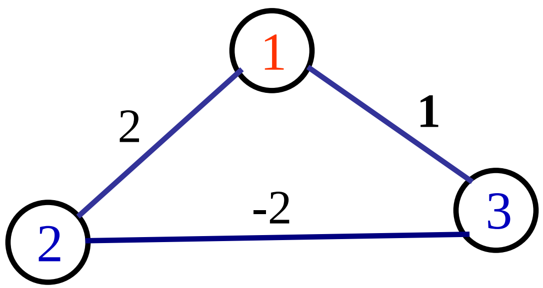
C. c, f, e, d

D. c, f, d, e



Dijkstra算法的局限性*

对于带有负权值的图？



考虑从1-3的最短路径：

- 1->3：路径长度1，Dijkstra输出结果
- 1->2->3：路径长度0，更短！

- 问题所在：Dijkstra算法要求已经确定的最短路径在后续算法执行时不会被修改。即路径长度要单调递增。
- 解决方法：re-weighting，每条边的权值都增加2？
- Bellman-Ford algorithm (无负回路)

- 如果存在负回路，则最短路径可以任意负下去！

问题6.1：北京地铁乘坐线路查询

文件bgstations.txt为数据文件，包含了北京地铁所有线路及车站信息。其格式如下：

12
1 23
苹果园 0
古城 0
…
公主坟 1
…
四惠东 1
2
2 19
西直门 1
积水潭 0
…
西直门

说明：表明目前北京地铁共开通12条线，其中1号线有23个车站，分别 为苹果园，非换乘站；…； 公主坟，换乘站…。2线共有19个站，分别为西直门，换乘站，…。

输入：

起始站：西土城

目的站：北京西站

输出： ***西土城**-10(1)-**知春路**-13(2)-**西直门**-4(2)-**国家图书馆**-9(4)-**北京西站***

(原以为为：西土城-10(3)-黄庄-4(3)-国家图书馆-9(4)-北京西站)

问题6.1：问题分析与算法设计

主算法流程（片段）

↓

初始化地铁线路图（initMap()函数）

↓

分别读入起始站和目的站

↓

按照Dijkstra算法在图中查找最短路径（Dijkstra()函数）

↓

依据最短路径按按照格式要求输出换乘路径（printPath()函数）

↓

1. 依次读入每条地铁线路的站名信息
   - 1.1 将站信息加入到站信息数组中
     注：站名是唯一的，每个站在该数组中的下标即为图的顶点编号
   - 1.2 将每条线路的当前站和其前序站构成的边(v1,v2)加入到图顶点权重数组中
     注：在权重数组中权重信息包括两站间的站数（缺省为1），以及所属线路

该方法的优点是简单，缺点是所有站都在图中，性能低。

问题6.1：数据结构设计

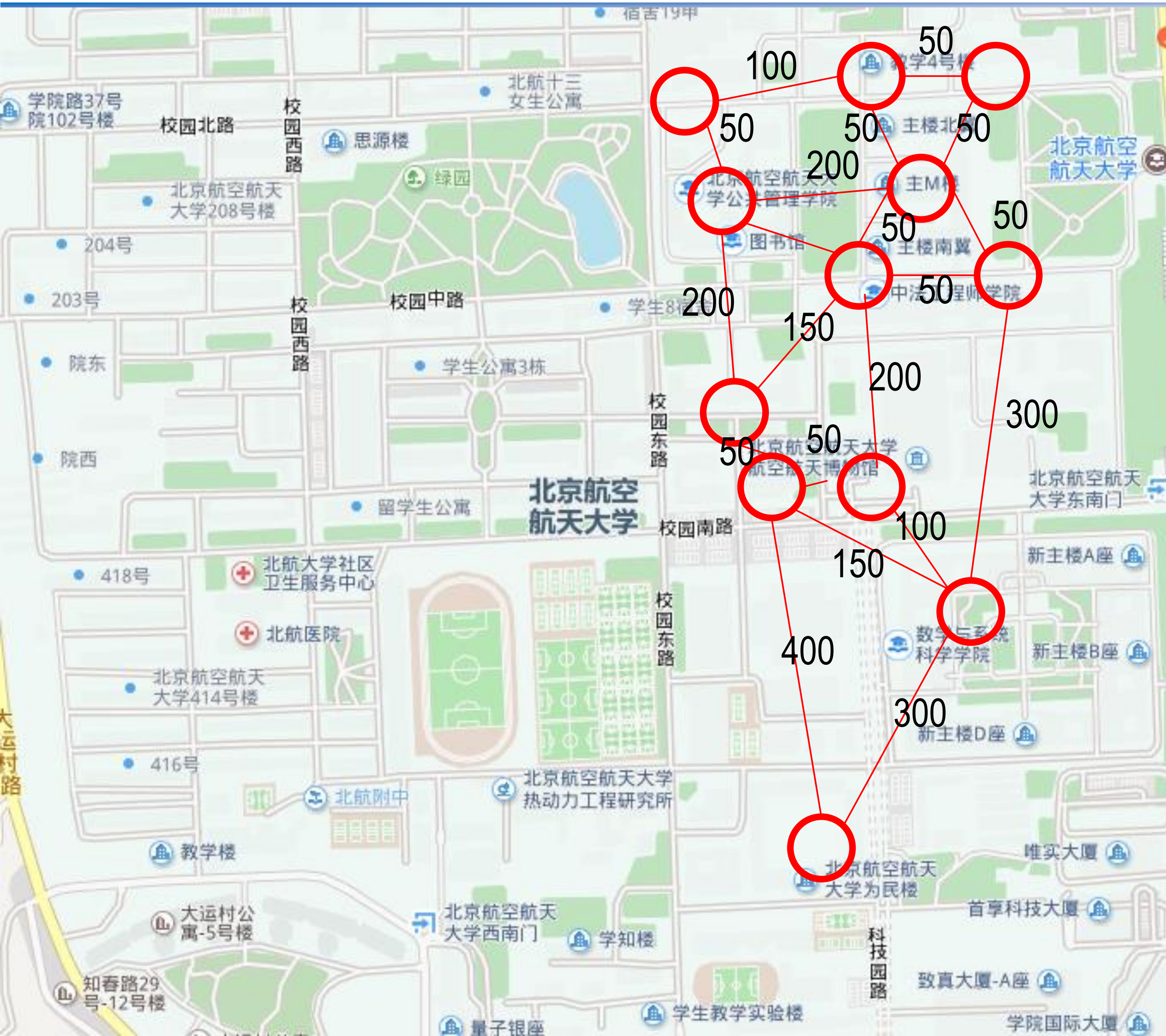
```
#define MAXNUM 512           //地铁最大站数
#define MAXLEN 16            //地铁站名的最大长度
#define INFINITY  32767

struct  station { //车站信息
    char  sname[MAXLEN];     //车站名
    int ischange;            //是否为换乘站，0-否，1-换乘
};
struct weight {
    int wei;                 //两个站间的权重，即相差站数，缺省为1
    int lno;                 //两个顶点所在的线号
};
struct station BGvertex[MAXNUM];               //地铁网络图顶点数组
struct weight BGweights[MAXNUM][MAXNUM];   //网络图权重数组，邻接矩阵
int Vnum = 0;          //实际地铁总站数
```

在实际应用时，可事先计算出地铁所有站间的最短路径，这样每次查询时不用重新计算，直接显示就行。

延伸阅读*：请同学自学有关所有点对之间的最短路径问题，了解弗洛伊德（Floyd）算法的原理及C实现。

弗洛伊德（Floyd）算法基于动态规划思想，其算法复杂度为O(n^3)。

6.5 最小生成树

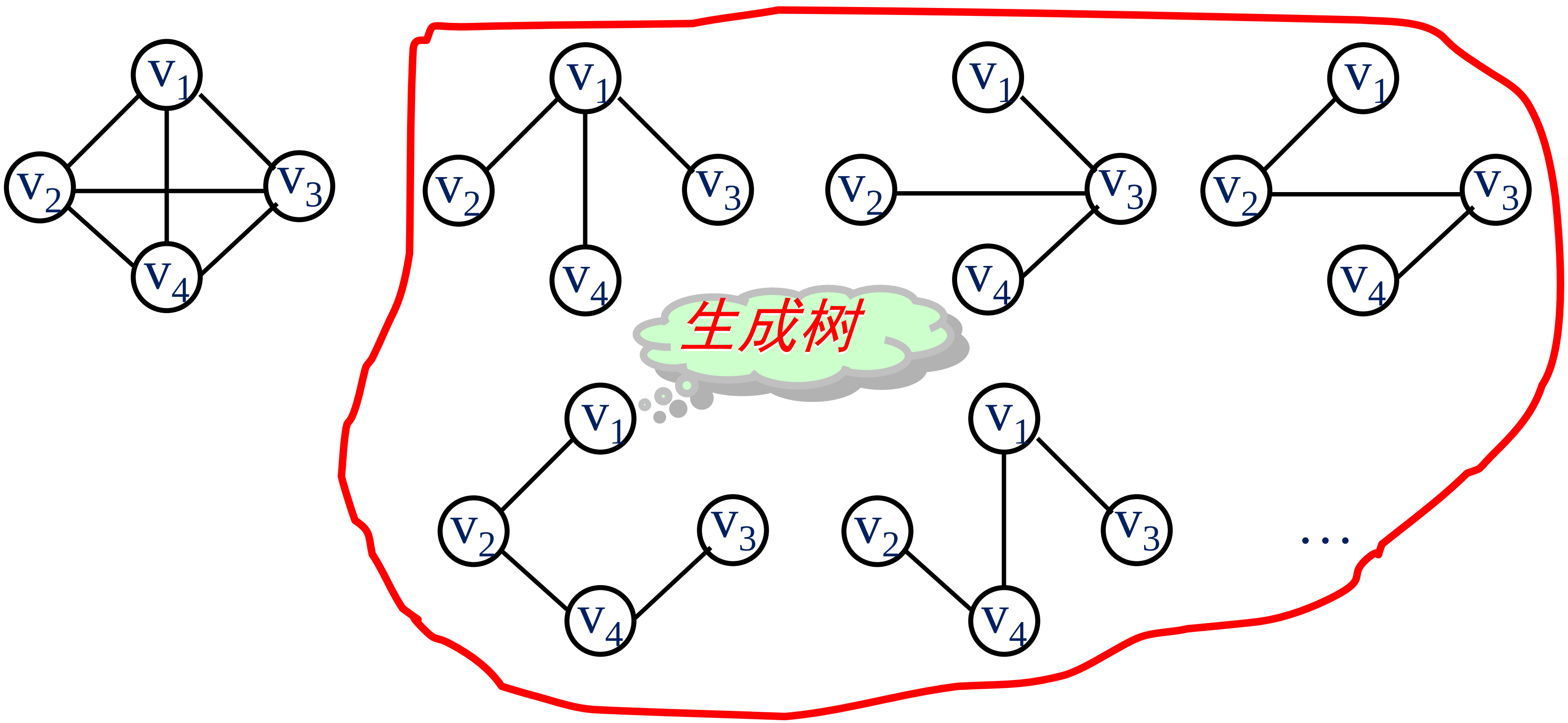


示例：北航网络中心要给北航主要办公楼间铺设光缆以构建通讯网络。

问题：如何以**最小的花费**完成网络铺设？

一. 最小生成树的相关概念

1. 生成树：给定连通图G(V, E)，包含着连通图的全部n个顶点的极小连通子图（包含其n-1条边、无回路）。



生成树性质（回顾）

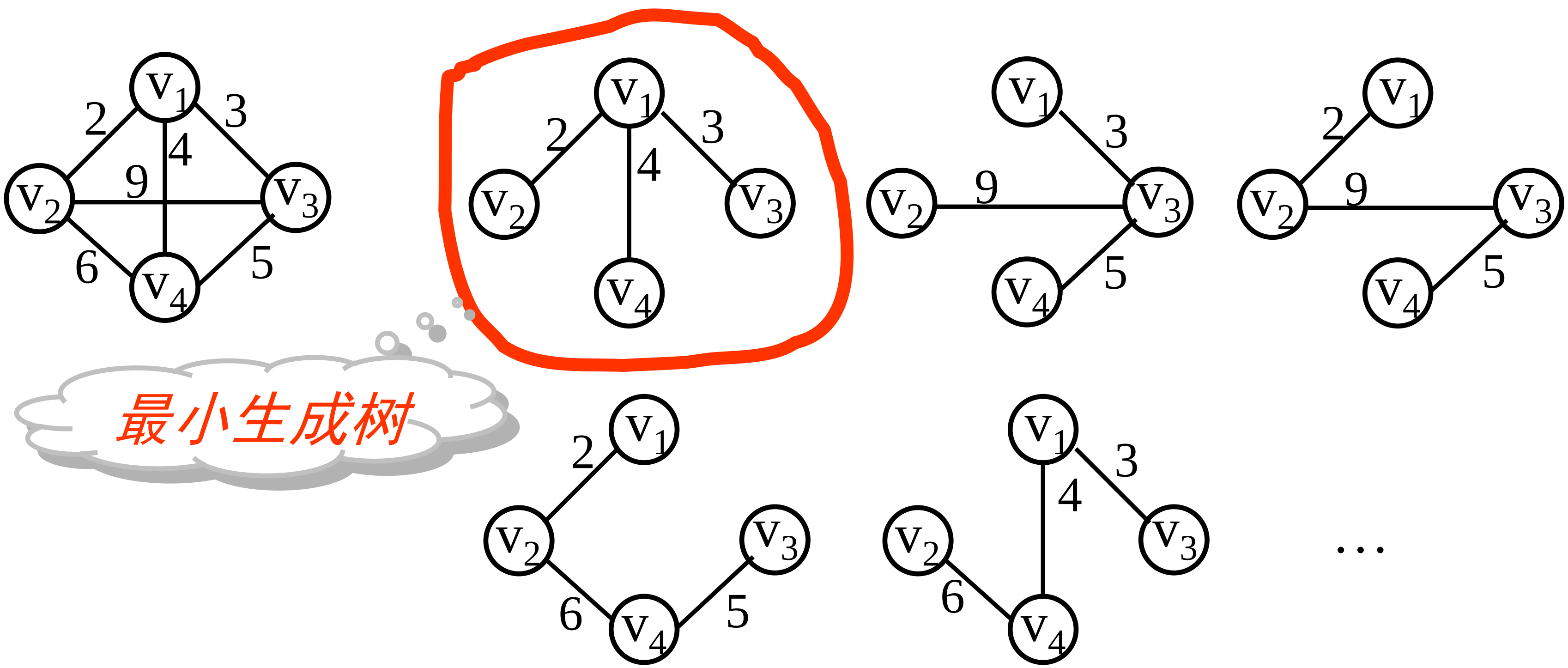
包含具有n个顶点的连通图G的全部n个顶点，仅包含其n-1条边的极小连通子图称为G的一个生成树。

性质：

1. 包含n个顶点的图：连通且有n-1条边
   <=>无回路且有n-1条边
   <=>无回路且连通
   <=>是一棵树
2. 如果n个顶点的图中只有少于n-1条边，图将不连通
3. 如果n个顶点的图中有多于n-1条边，图将有环（回路）
4. 一般情况下，生成树不唯一

一. 最小生成树的相关概念

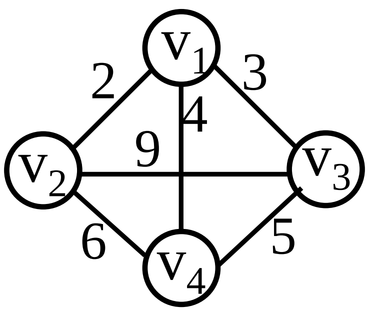
2. 带权连通图中，总的权值之和**最小**的生成树称为**最小生成树**，也称**最小代价生成树**或**最小花费生成树**。

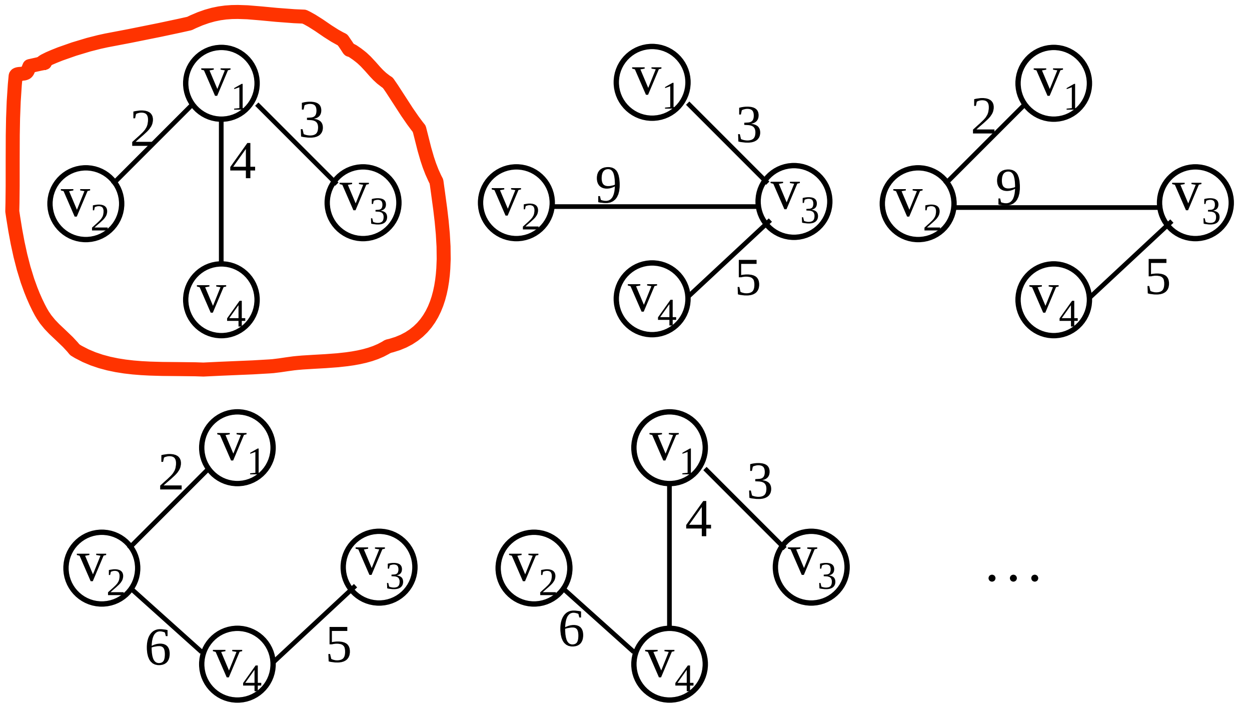


最小生成树可能不唯一

二. 求解最小生成树

最基本的办法：穷举法



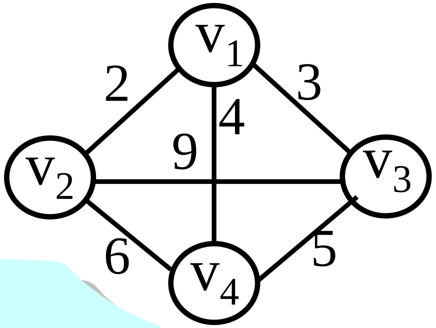


$$C_{|E|}^{|V|-1}$$

二. 求解最小生成树

增量法：从空树开始，依次从图G中**挑选**一条边/顶点加入生成树，逐步构造出完整的生成树。

- 普里姆(Prim)法
- 克鲁斯卡尔(Kruskal)法



构造最小生成树的基本原则:

- 只能利用图中的边来构造最小生成树；
- 只能使用图中的n-1条边来连接图中的n个顶点；
- 选择不使子图产生回路的边加入生成树。

1. 普里姆(Prim)算法

设G=(V, GE)为具有n个顶点的带权连通图;

设T=(U, TE)为生成的最小生成树。

初始时, U={v0}，v0∈V；TE=Φ。

一条边
及端点

令G’=(V’, GE’) = (V-U, GE-TE)，为图G去掉生成树T的剩余(残图)。

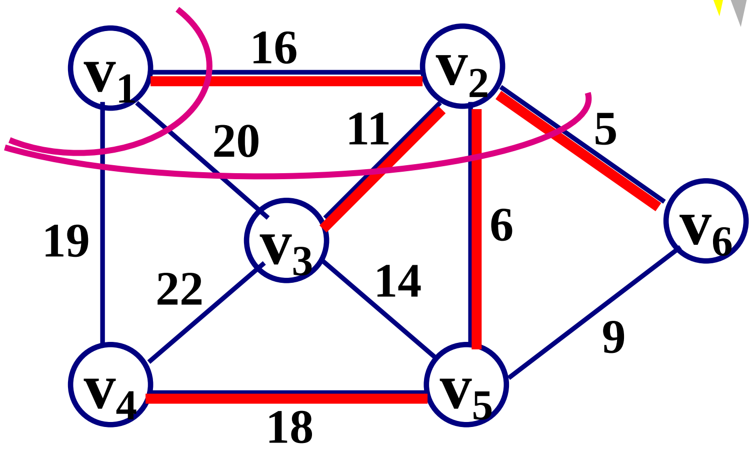
初始时**V’=V-{v0}, GE’=GE**

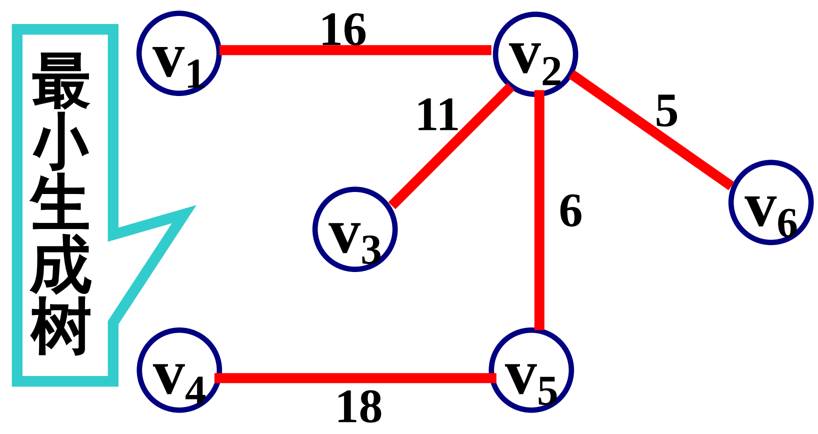
如何挑选边/顶点?

贪心策略

挑选在连接U和V-U的所有边里权值最小的边

例





G’:

V’ = { }

$$GE' = \left\{ \begin{array}{l} (v_1, v_3)_{20} \\ (v_1, v_4)_{19}, \\ (v_3, v_4)_{22}, \quad (v_3, v_5) \end{array} \right\}$$

T: **总权值 = 56**

$U = \{ v_1, v_2, v_6, v_5, v_3, v_4 \}$

$$TE = \{ (v_1, v_2)_{16} \quad (v_2, v_6)_5 \quad (v_2, v_5)_6 \quad (v_2, v_3)_{11} \quad (v_4, v_5)_{18} \}$$

算法关键：如何高效地依次从GE’的众多边中挑选满足要求的一条边/点

1. 普里姆(Prim)算法

设G=(V, GE)为具有n个顶点的带权连通图;
设T=(U, TE)为生成的最小生成树。

初始时，U={v0}，v0∈V；TE=Φ。

令G’=(V’, GE’) = (V-U, GE-TE)，为图G去掉生成树T的剩余(残图)。初始时V’=V-{v0}, GE’=GE

重复 n–1 次

1. 选择：从连接残图G’(顶点集V’)和最小生成树T(顶点集U)的所有边中选择一条权值最小的边/顶点
2. 添加：将该边及其端点加入生成树T
3. 更新：更新邻接顶点的最短边信息

任何时刻T都是部分顶点的最小生成树
最终为G的最小生成树

贪心策略

1. 普里姆(Prim)算法——关键数据结构

int weights[VNUM][VNUM]：存放图G中边的权值。

针对残图(集合V')中的每个顶点v，记录它到最小生成树（集合U）的最短边信息：

int dest[VNUM]: **dest**[v]记录最短边的另一个端点

int minweight[VNUM]： **minweight[v]**存放最短边的权值；值为0表示v已经加入生成树

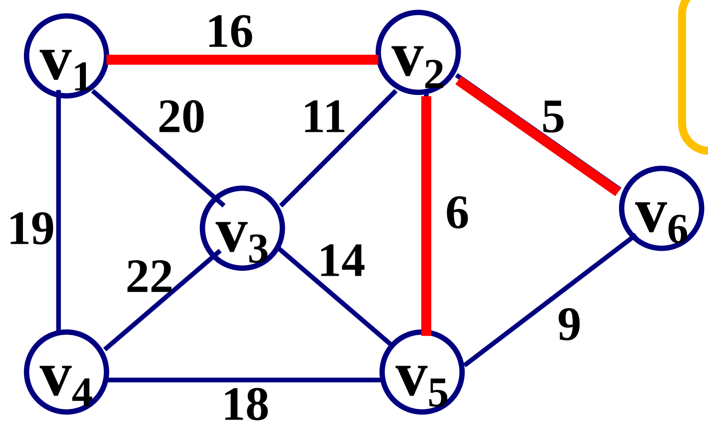
Q: Dijkstra算法有几个辅助数组，分别记录什么信息？

~~int wfound[VNUM]; //表示顶点是否已确定最短路径~~

int **dist**[VNUM]; //表示顶点到起始点的最短距离

int **Spath**[VNUM]; //表示最短路径，只记前驱结点

1. 普里姆(Prim)算法——关键数据结构



v1加入树
T(U, TE)

src	dest	minWeight
v1	-	0
v2	-	∞
v3	-	∞
v4	-	∞
v5	-	∞
v6	-	∞

每次添加新顶点后更新数据表

src	dest	minWeight
v1	-	0
v2	v1	16
v3	v1	20
v4	v1	19
v5	-	∞
v6	-	∞

src	dest	minWeight
v1	-	0
v2	v1	0
v3	v2	11
v4	v1	19
v5	v2	6
v6	v2	5

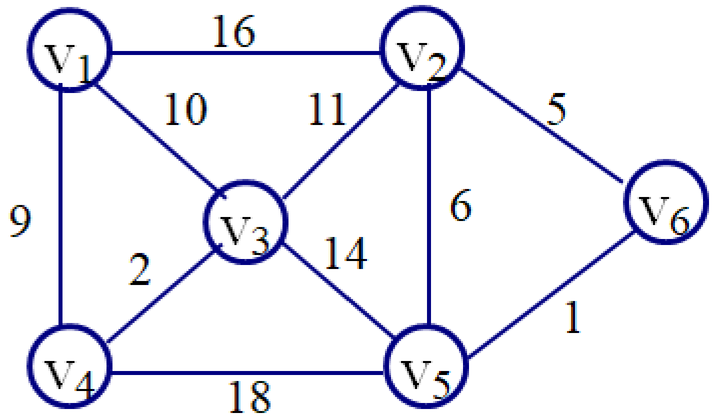
src	dest	minWeight
v1	-	0
v2	v1	0
v3	v2	11
v4	v1	19
v5	v2	6
v6	v2	0

```
#define MAXVER  512
#define INFINITY   32767
void Prim(int  weights[][ MAXVER], int n, int src, int dest[ ])
{ //weights为权重数组、n为顶点个数、src为最小树的第一个顶点、 dest为最小生成树边
    int minweight [MAXVER], min;
    int i, j, k;
    for(i=0; i<n; i++){  //初始化相关数组
        minweight[i][0] = weights[src][i];  //将src顶点与之有边的权值存入数组
        dest[i]  = src;  //初始化第一个顶点为src
    }
    minweight[src]  = 0;   //将第一个顶点src顶点加入生成树
    for(i=1; i< n; i++){
        min = INFINITY;
        for(j=0, k=0;  j<n; j++)          //1.选择：查找满足要求的一条边，保存在(j, k)
            if(minweight[j] !=0 && minweight[j] < min) {  //在数组中找最小值，其下标为k
                min = minweigth[j];  k = j;
            }
        minweight[k] = 0;                 //2. 添加：将该边的端点k加入最小生成树
        for(j=0;  j<n; j++)               //3. 更新：修改邻接点的最短边信息
            if(minweight[j] != 0; && weights[k][j] < minweight[j] ) {
                minweight[j] = weights[k][j];    //将小于当前权值的边(k,j)权值加入数组中
                dest[j] = k;   //将边(j,k)信息存入边数组中
            }
    }
}
```

时间复杂度：$O(|V|^2)$

堆优化：$O(|E|\log|V|)$

对于上图所示的无向连通图，若采用普里姆（Prim）算法求其最小生成树，假设第一个选择加入最小生成树的顶点为V2，则最后一条加入最小生成树的边的权值为______。



2. 克鲁斯卡尔(Kruskal)算法

贪心策略

G=(V, GE)， T=(U, TE)，初始时， U=V, TE=Φ

VS Prim: Prim仅从连接残图G'的顶点集V'和生成树T的顶点集U的所有边中选；
Kruskal从所有边中选（因此过程中T可能不是生成树）

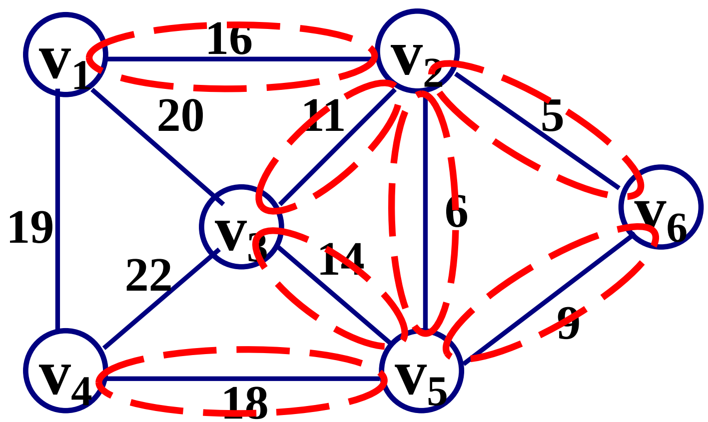
① 选择：从G中选择一条之前未选择过的、权值最小的边e（注：可以先对所有边按权值升序排序）

② 更新：若边e使得T产生回路，则本次选择无效，放弃e；否则将e加入T

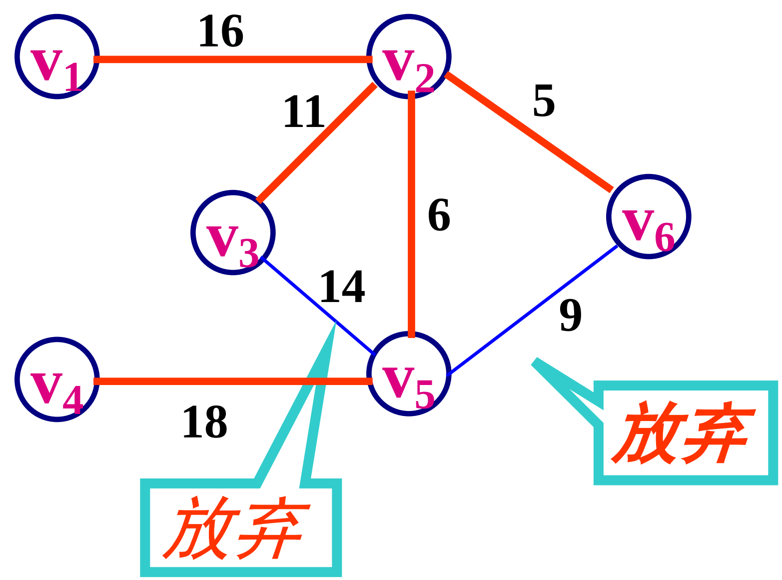
③ 重复上述选择过程直到TE中包含了G的n-1条边，此时的T为G的最小生成树。

例

G (连通图)



T (生成树)



Q: 如何快速判断是否存在环路?

用1个标志信息表示顶点已加入生成树?

A: 每个连通子图分别用1个标志

克鲁斯卡尔算法的时间效率主要取决于对边做排序，因而时间复杂度为：O(|E|log|E|)

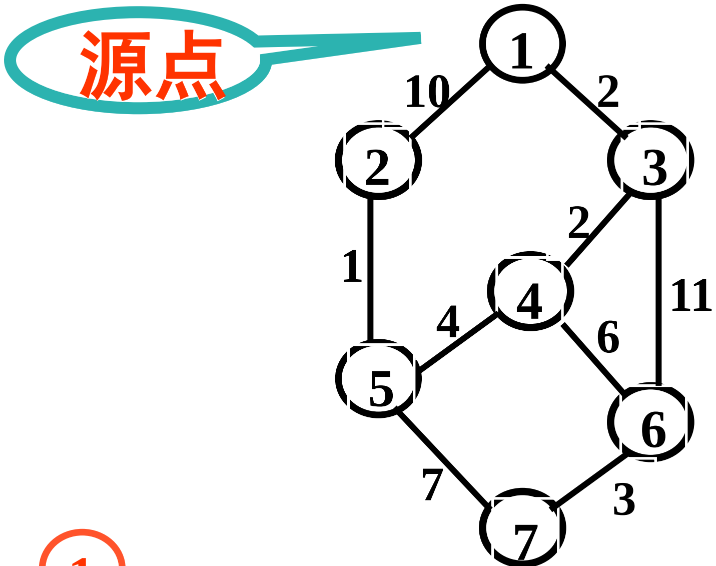
延伸阅读*：

上面我们介绍了*普里姆(Prim)算法*和*克鲁斯卡尔(Kruskal)算法*的基本原理，请同学们自学*Kruskal*算法正确性证明及C实现。

判断

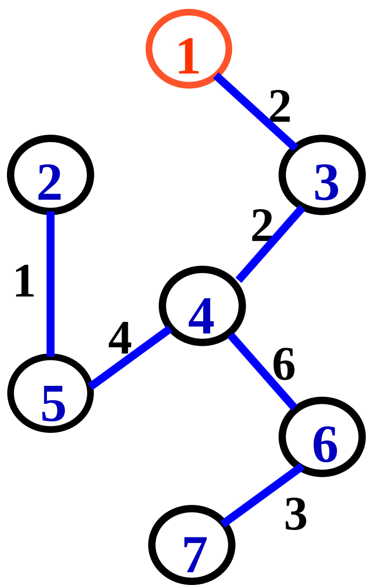
- √ 任意连通图中，假设没有相同权值的边存在，则权值最小的边一定是其最小生成树中的边。
- × 任意连通图中，假设没有相同权值的边存在，则权值最大的边一定不是其最小生成树中的边。
- √ 任意连通图中，假设没有相同权值的边存在，则与同一顶点相连的权值最小的边一定是其最小生成树中的边。
- × 采用克鲁斯卡尔算法求最小生成树的过程中，判断一条待加入的边是否形成回路，只需要判断该边的两个顶点是否都已经加入到集合U中。

最小代价生成树与最短路径生成树的关系

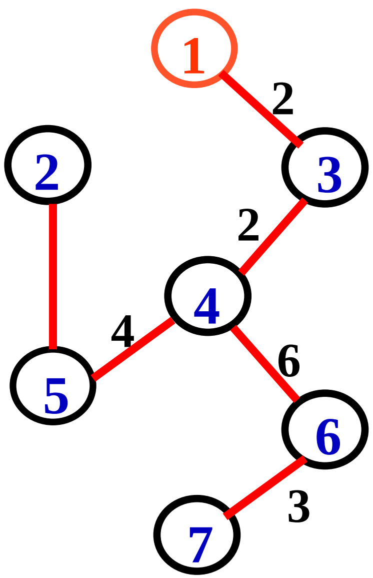


问题

最小代价生成树



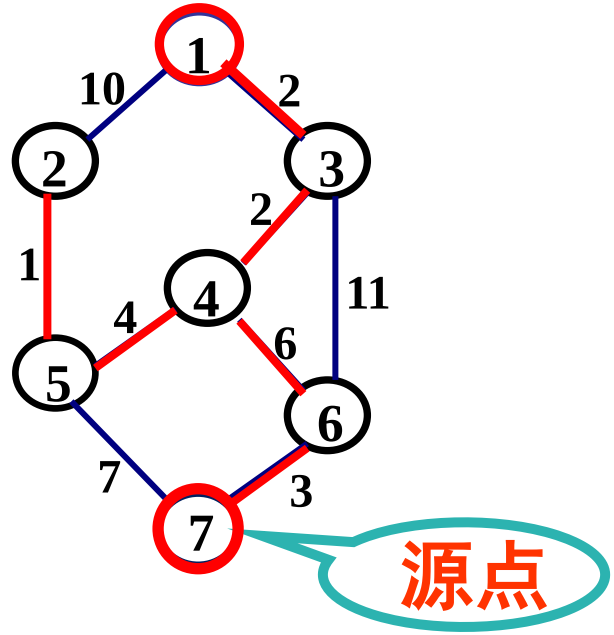
这个例子中两者相同



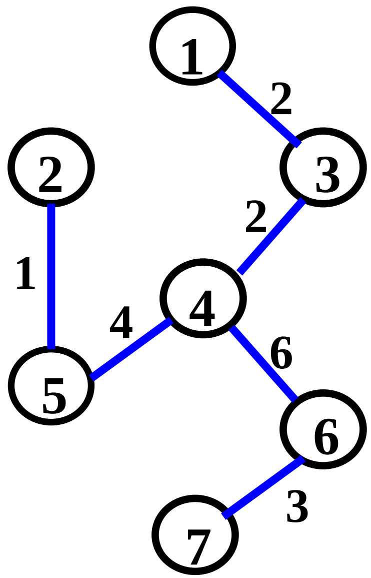
最短（路径）生成树

最小代价生成树与最短路径生成树的关系

对于给定的带权连通无向图，从某源点到图中各顶点的最短路径构成的生成树是否是该图的最小生成树 ?

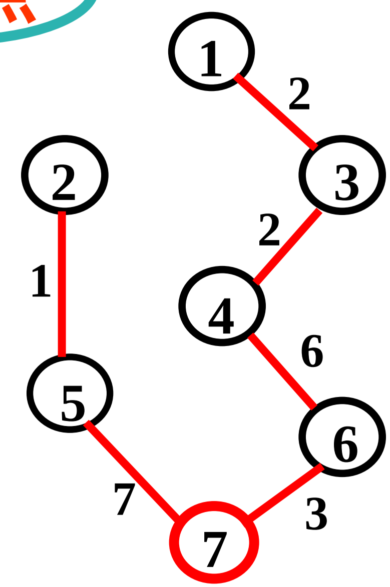


最小代价生成树



不相同！

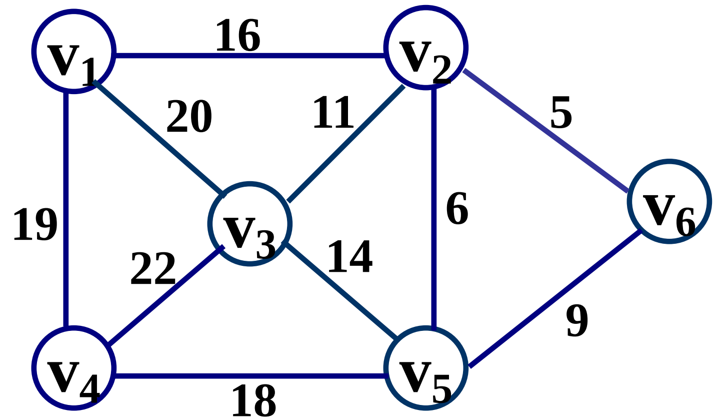
最短(路径)生成树



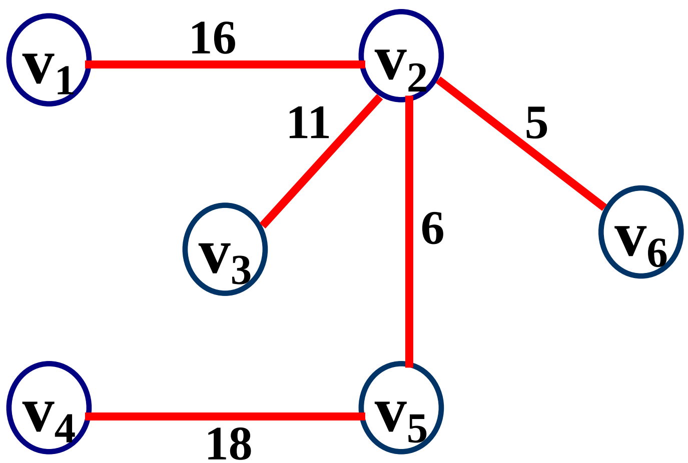
与“源点”无关！

与源点有关！

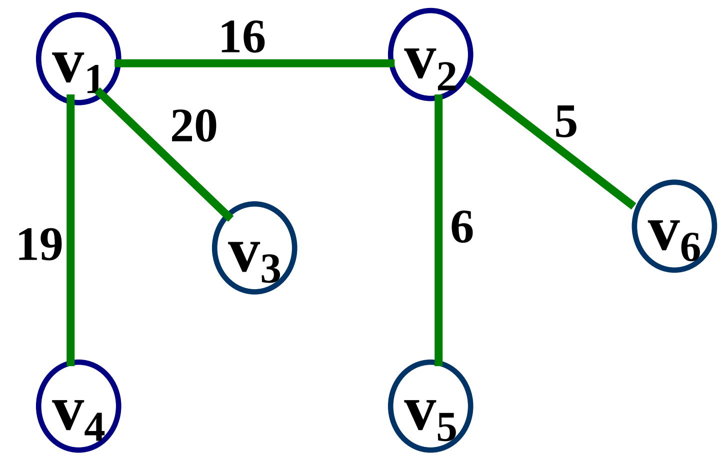
另一个例子



结论…



最小生成树



源点为v_1的最短路径