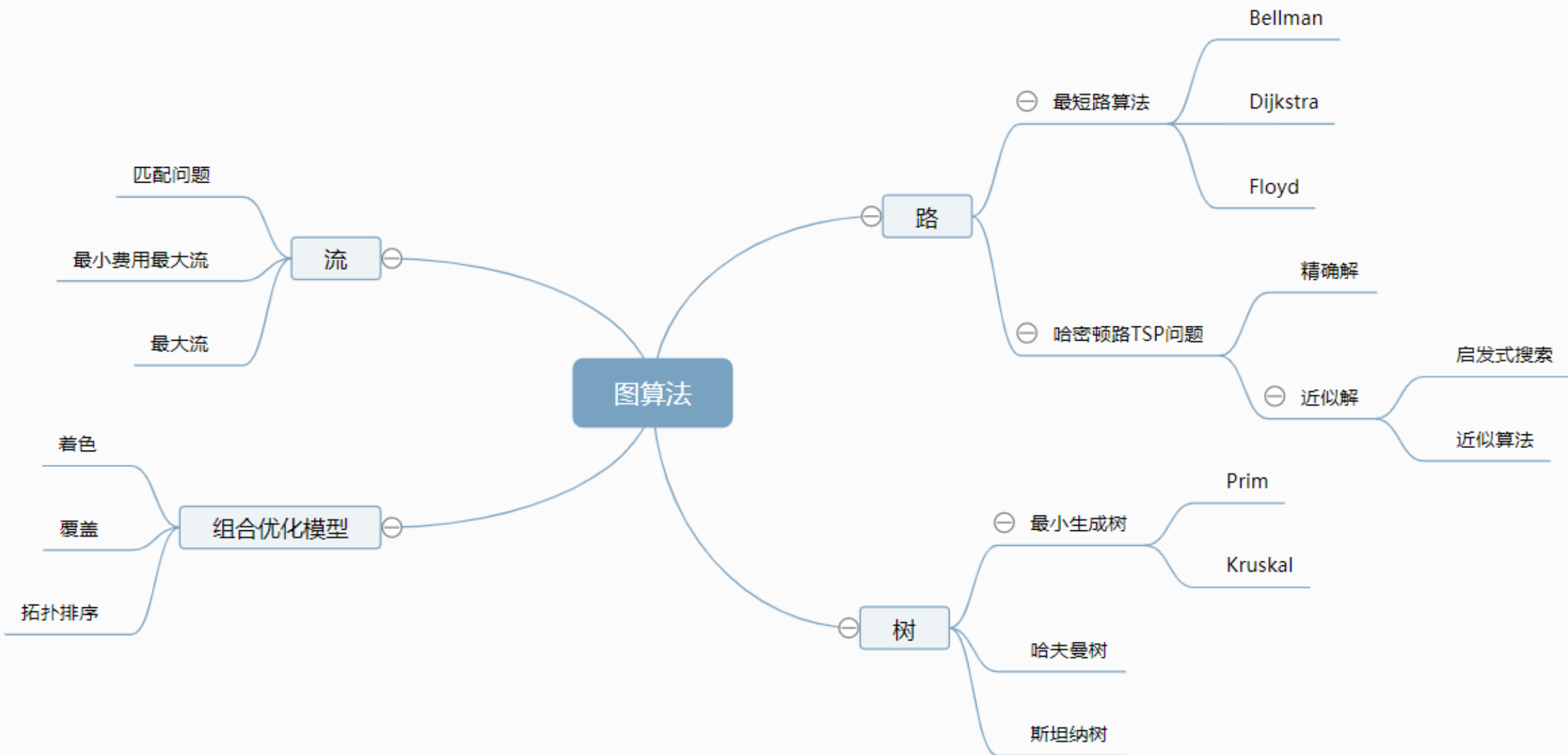




图算法及程序实现

2018级 IEEE试点班 张嘉乐

本讲主要内容

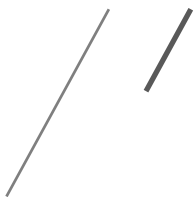


本讲基础：程序设计

- 知道算法的概念：一系列算术运算、逻辑判断过程
- 知道计算机能操作的东西：变量、数组(矩阵)、结构体
- 无基础可以听懂本讲，但不能代码实现。



图算法

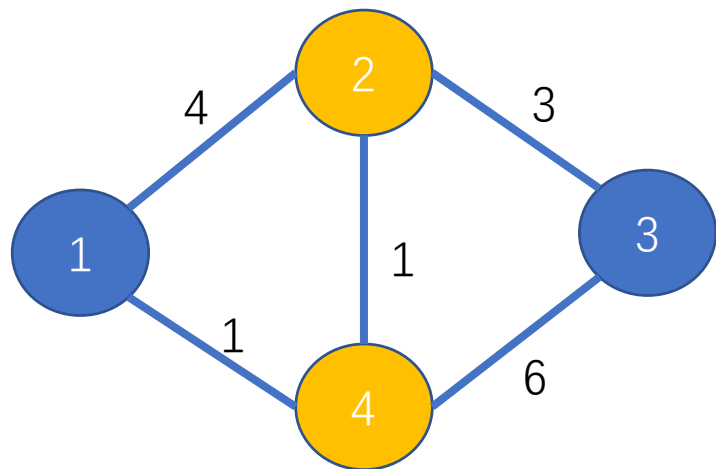




图的概念与含义

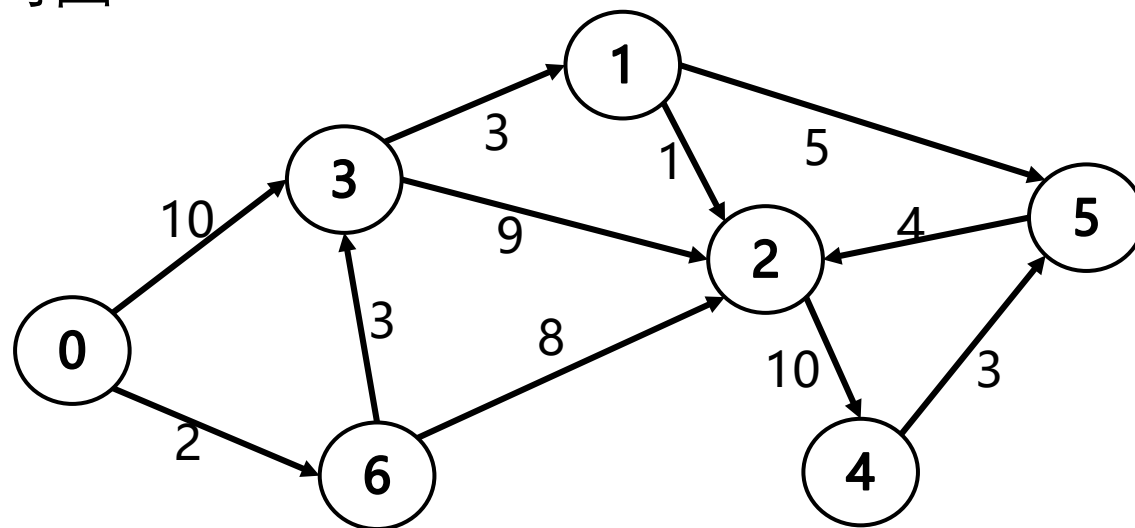
图的数学抽象

例1: 无向图



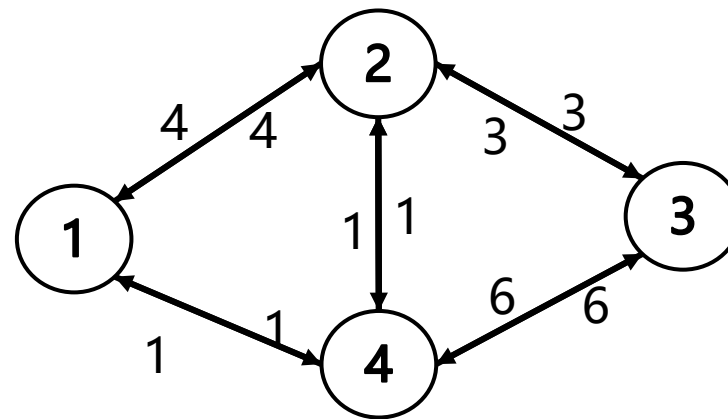
$G=(V,E)$, 4个点5条边
点集 $V=\{1,2,3,4\}$,
边集 $E=\{(1,2,4), (1,4,1), (2,3,3), (2,4,1), (3,4,6)\}$

例2: 有向图



$G=(V,E)$, 7个点11条边
点集 $V=\{0,1,2,3,4,5,6\}$,
边集 $E=\{(0,3,10), (0,6,2), (6,2,8), \dots\}$

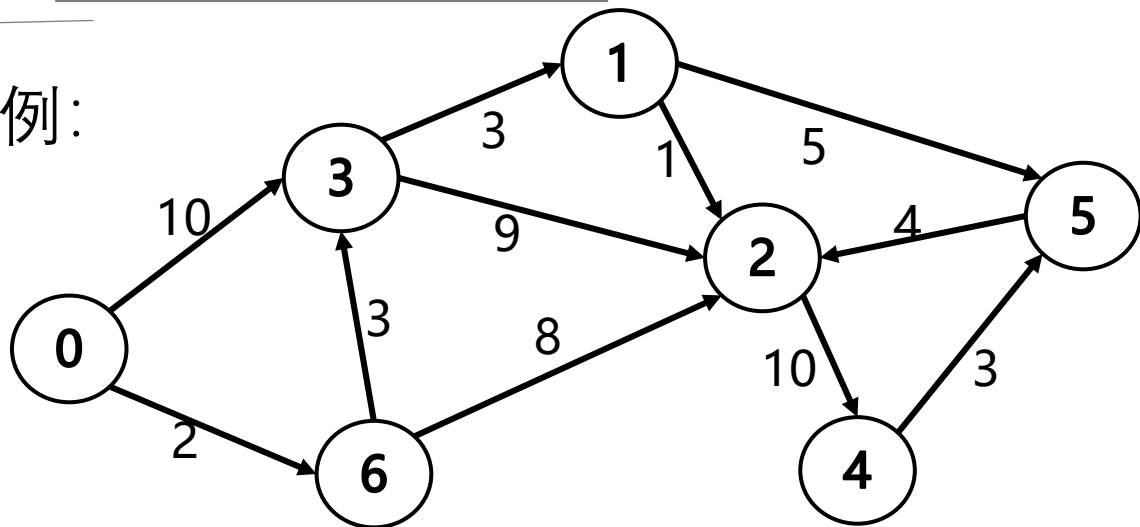
例3: 用有向图表示无向图



$G=(V,E)$, 5个点10条边

图的含义举例

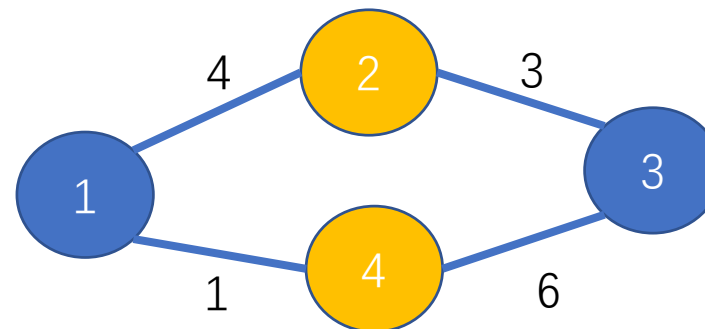
例:



1. 点表示地点, 边权重表示路线长度
 - 最短路
 - 最小生成树
 - 最短哈密顿回路(TSP问题)
2. 点表示地点, 边权表示水管流速
 - 最大网络流
 - 最小费用最大流

3. 点表示工作, 边权表示完成时间
 - 拓扑排序
 - 统筹安排

4. 点表示人, 边表示认识关系
 - 社交关系分析聚类




5. 蓝点是工人, 黄点是工作, 边权是任务收益
 - 最大匹配
 - 稳定匹配



P1

图算法1：路

- 图搜索
 - 深度优先搜索(DFS)
 - 广度优先搜索(BFS)
 - 一致代价搜索 (UCS, 即单源Dijkstra 算法)
 - 单源Bellman-Ford最短路 (允许负权)
 - 多源Floyd算法
- 

理解什么是算法——从图搜索说起

图1

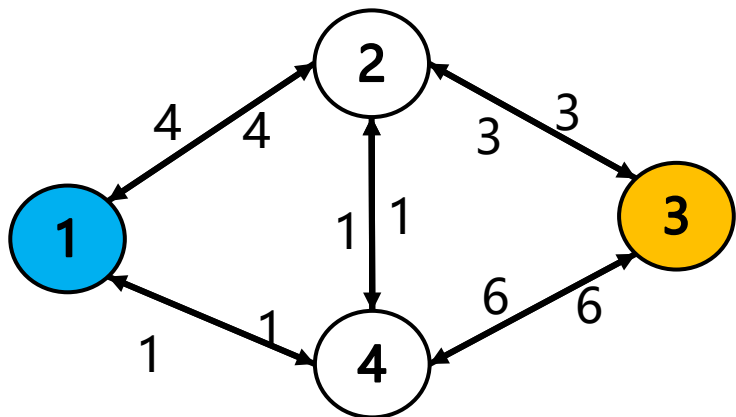
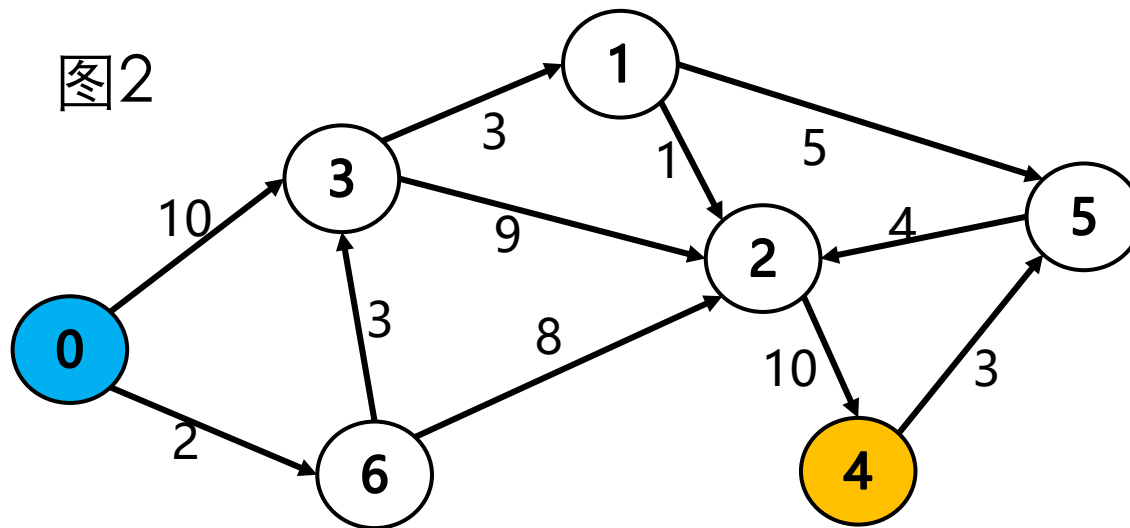


图2



问题1：找出一条从蓝点到黄点的路径

问题2：蓝点到黄点共有多少条简单路径？

问题3：蓝点到黄点的最短路径长度是多少？

理解什么是算法——从图搜索说起

图1

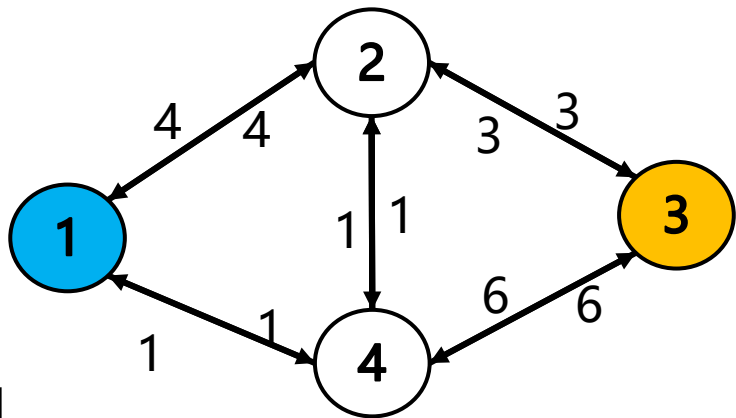


图2

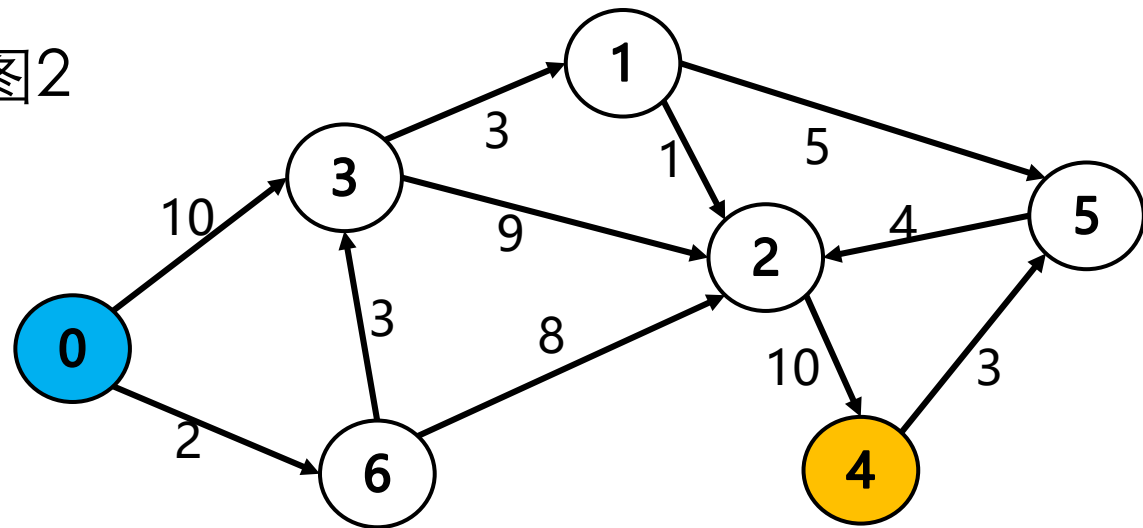


图1

问题1: 找出一条从蓝点到黄点的路径
比如1->2->3, 长度为4+3=7

问题2: 蓝点到黄点共有多少条简单路径? 4条
1->2->3, 长度7; 1->2->4->3, 长度11;
1->4->3, 长度7; 1->4->2->3, 长度5

问题3: 最短路径长度是5

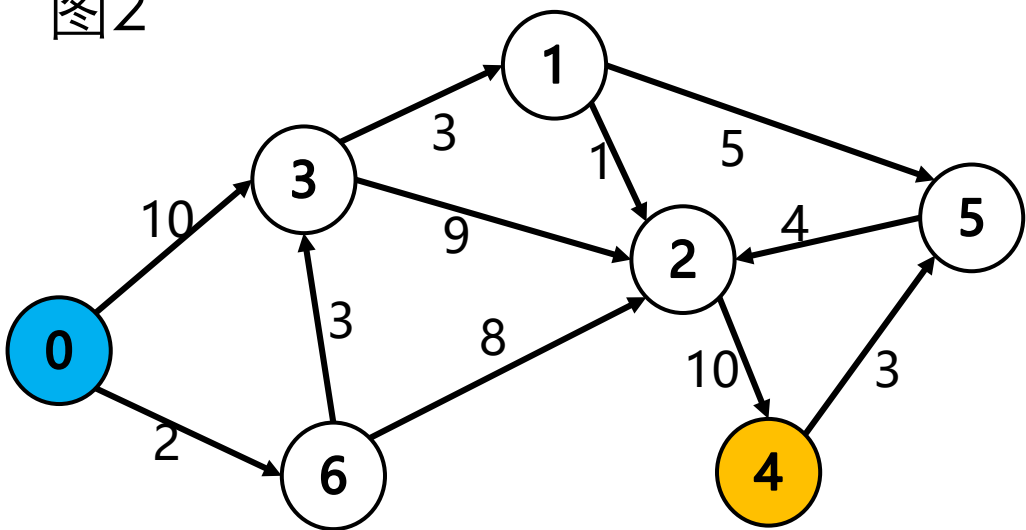
请问是怎么找的?
图2呢?

图2共有7条路径

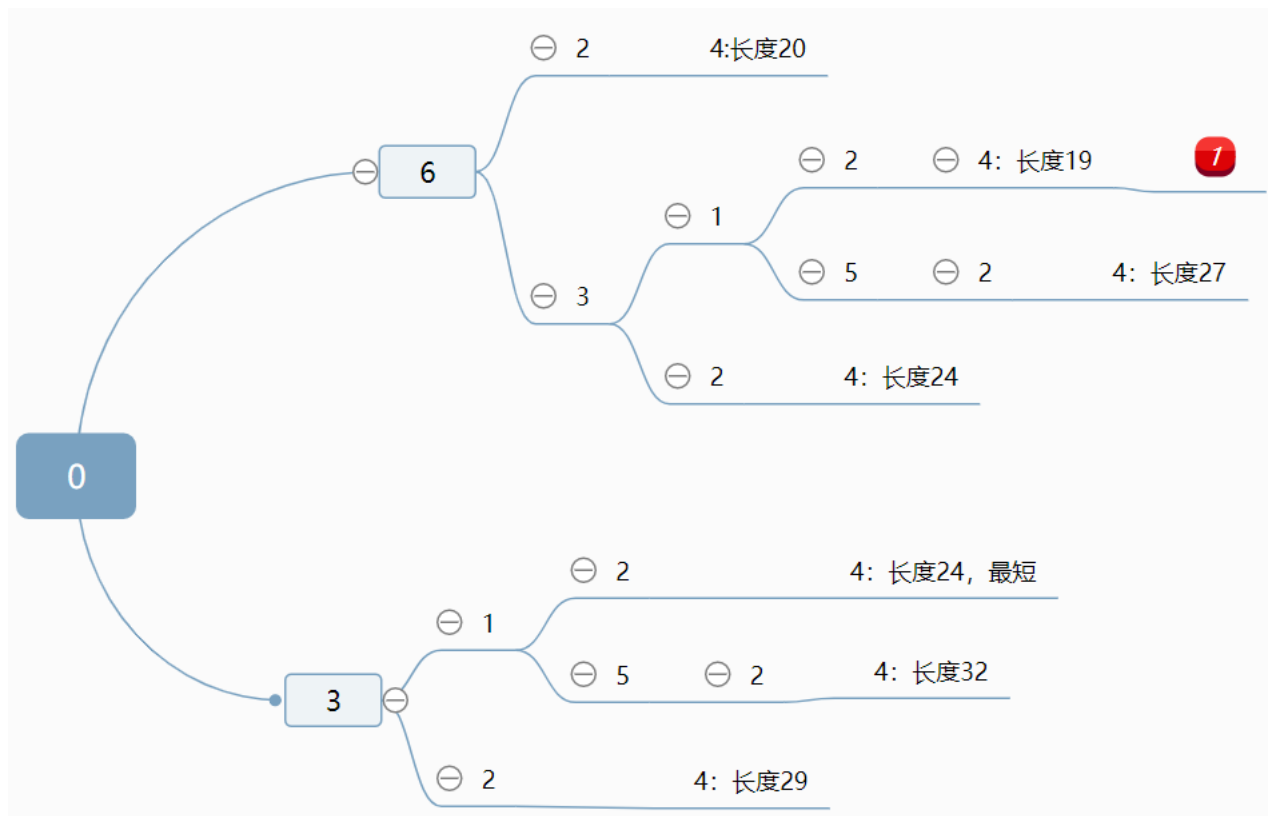
理解什么是算法——从图搜索说起

直接枚举，搜索树如下：

图2



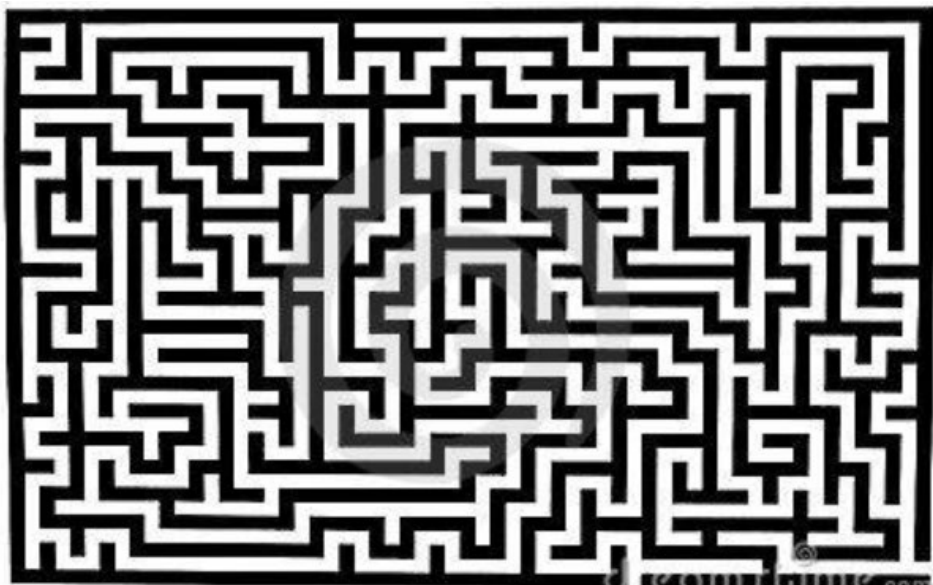
如何找齐7条路径?
如何找到最短路径?



- 1.可以找到路径：算法是解决问题的一系列步骤
- 2.有重复劳动：算法需要考虑复杂性

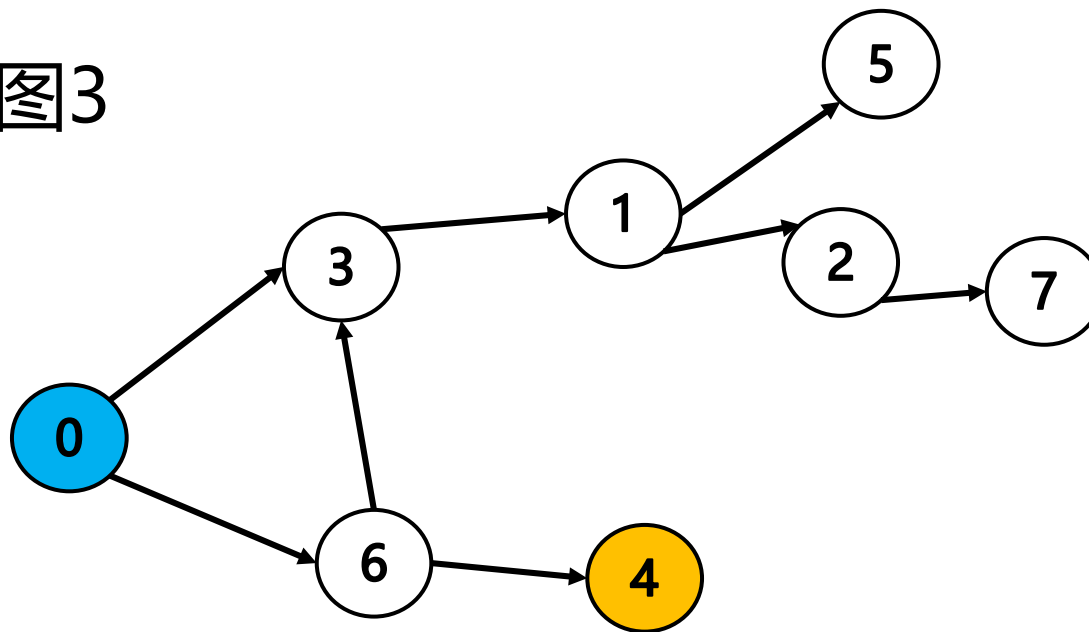
图搜索——深度优先搜索(Depth-First-Search, DFS)

找一条路径也不一定就很容易



必须按一定规则进行搜索

图3



用这个图来说明搜索的想法

0->3->1->5 回退到1

2->7 回退到2,1,3,0

6->4

图搜索——深度优先搜索(Depth-First-Search, DFS)

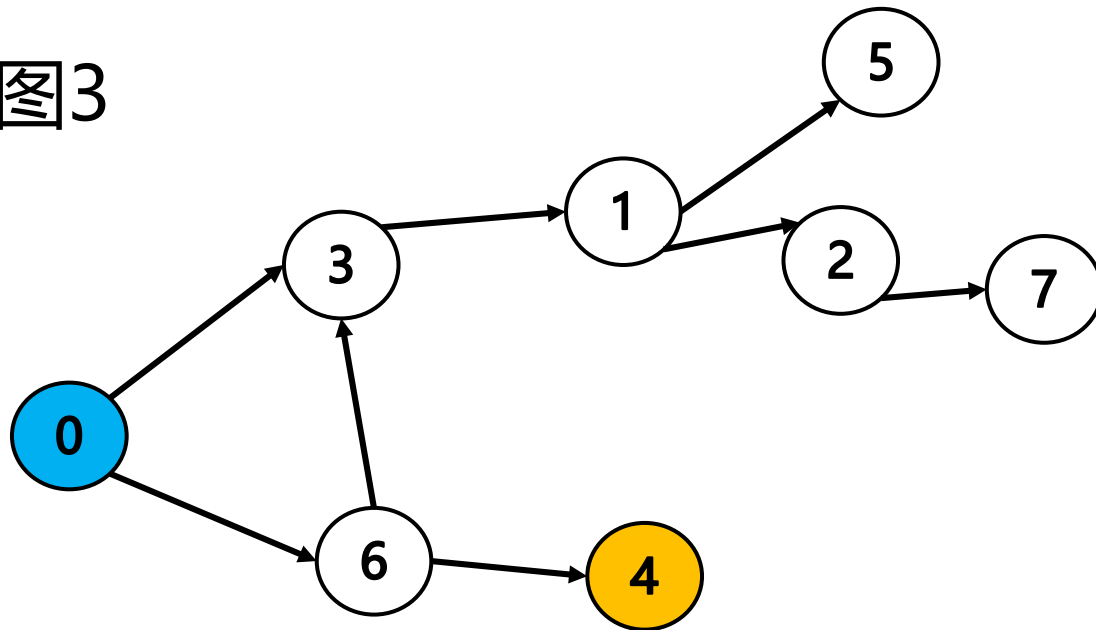
如何返回路径?

搜索过程中在每个节点上标注其前一个节点, 如 $pre[4]=6, pre[6]=0$.

DFS作为算法又称回溯法

- 不一定找到最短路
- 有路也不一定找得到

图3



用这个图来说明搜索的想法

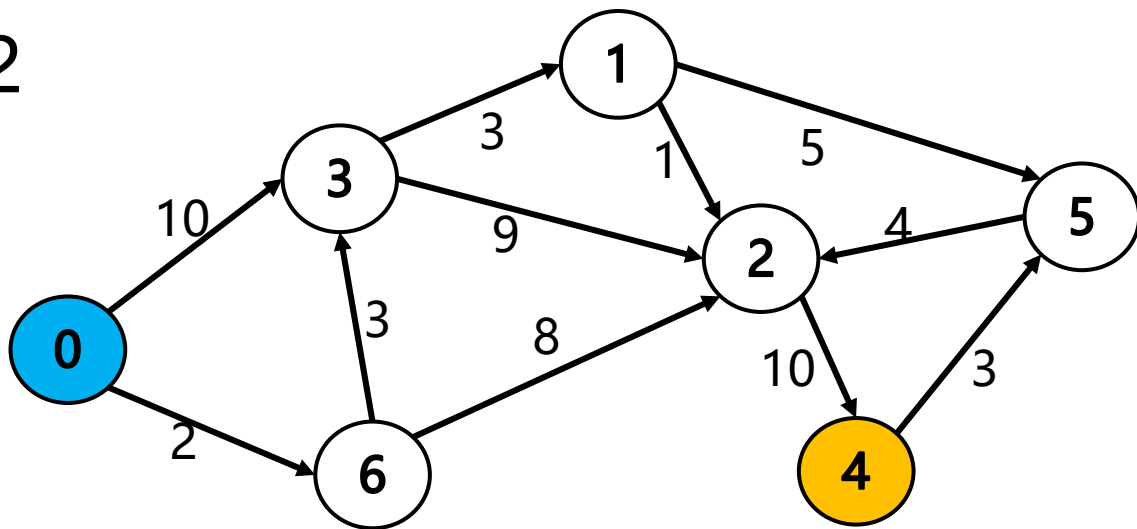
0->3->1->5 回退到1

2->7 回退到2,1,3,0

6->4

图搜索——深度优先搜索(Depth-First-Search, DFS)

图2



从0开始, 扩展3, $pre[3]=0$

扩展1, $pre[1]=3$

扩展5, $pre[5]=1$

扩展2, $pre[2]=5$

扩展4, $pre[4]=2$

路径长度32

算法流程:

- 从起点开始, 不断向前扩展
- 若遇到尽头则回退, 扩展下一个

被扩展过节点需要标注, 避免重复扩展

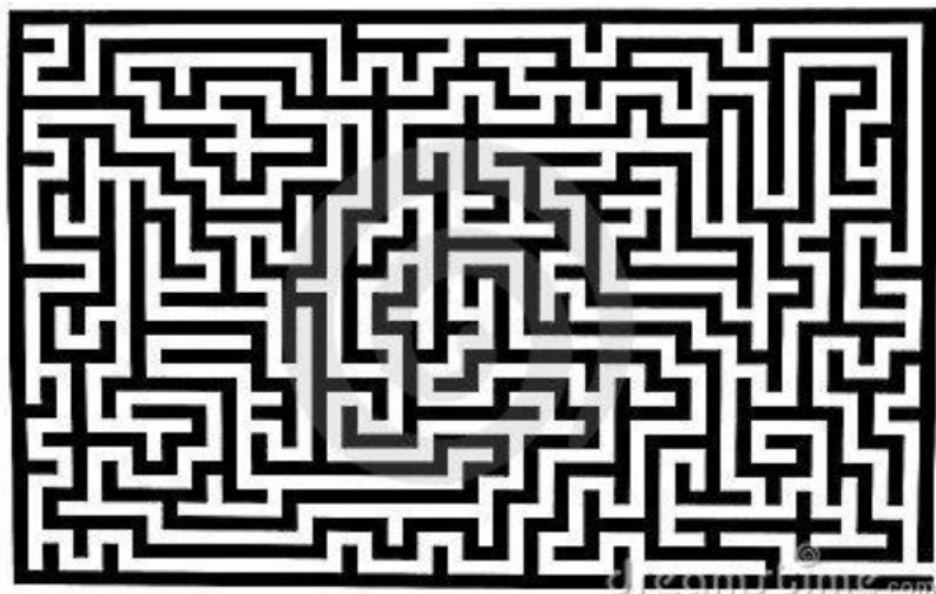
图搜索——深度优先搜索(Depth-First-Search, DFS)

1. 不一定找到最短路
2. 有路也不一定找得到



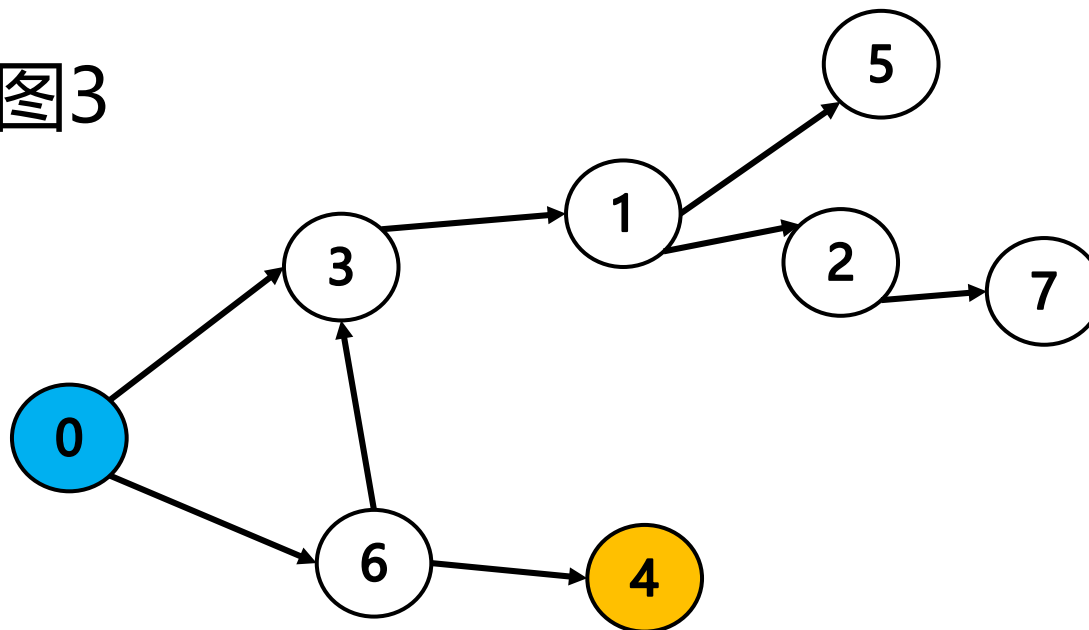
图搜索——广度优先搜索(Breath-First-Search, BFS)

找一条路径也不一定就很容易



必须按一定规则进行搜索

图3



用这个图来说明搜索的想法

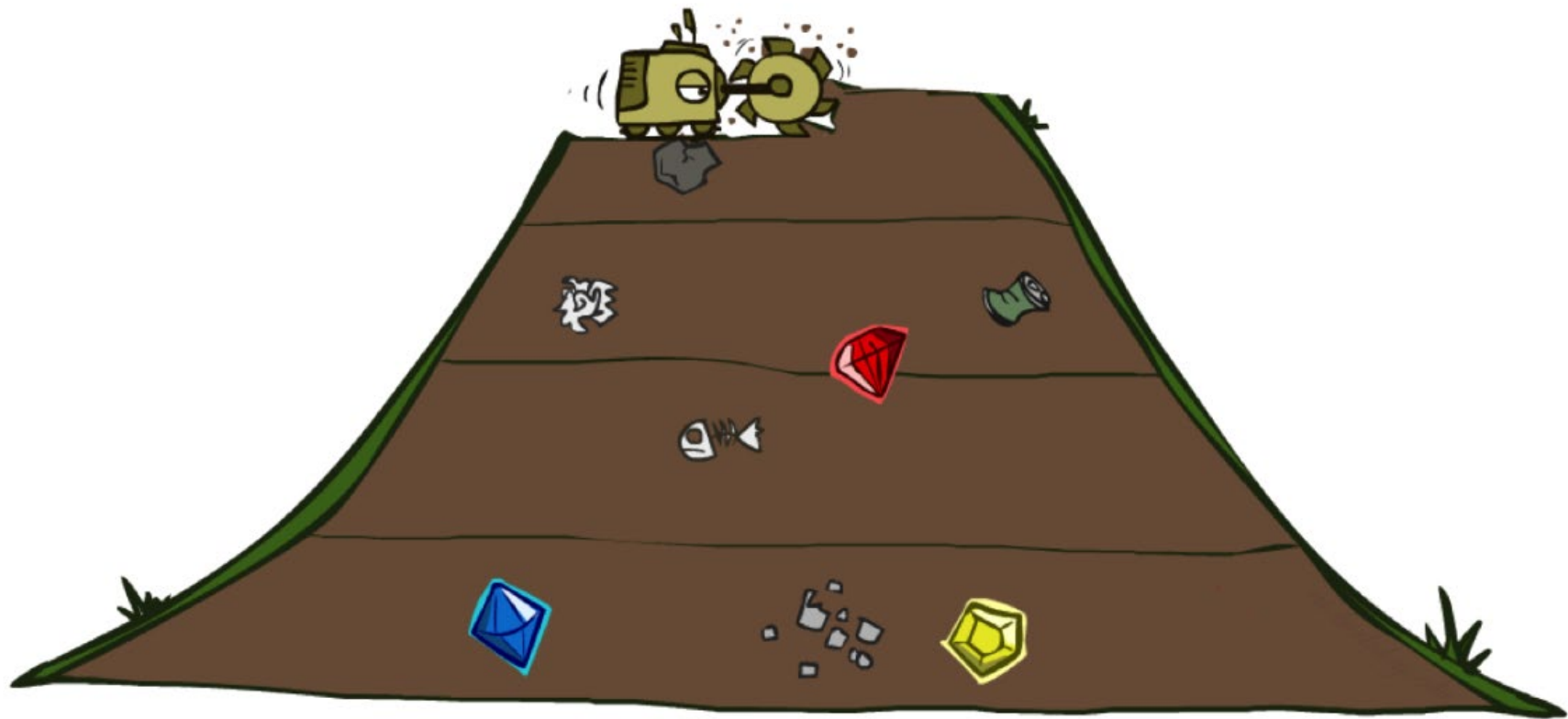
距离1的点: 3,6

距离2的点: 1,4。找到了, 结束。

返回路径方式同DFS。

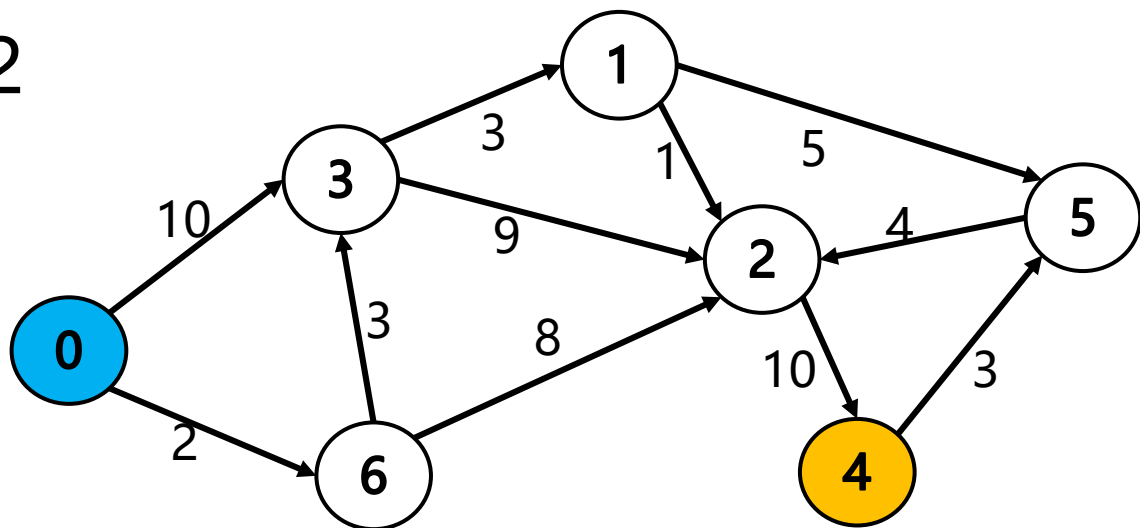
图搜索——广度优先搜索(Breath-First-Search, BFS)

1. 一定找到边数最少的路(边权值为1的话即最短路)
2. 有路一定找得到



图搜索——广度优先搜索(Breath-First-Search, BFS)

图2



算法流程:

- 从起点开始, 一层层向外扩展
- 被扩展节点加入队列
- 每次取出队列头, 向外扩展

从0开始, 扩展3,6。

取出3, 扩展1,2。

取出6, 无可扩展(注意)。

取出1, 扩展5。

取出2, 扩展4, 找到。

路径: 4←2←3←0。长度29, 非最短。

队列[3,6]

队列[6,1,2]

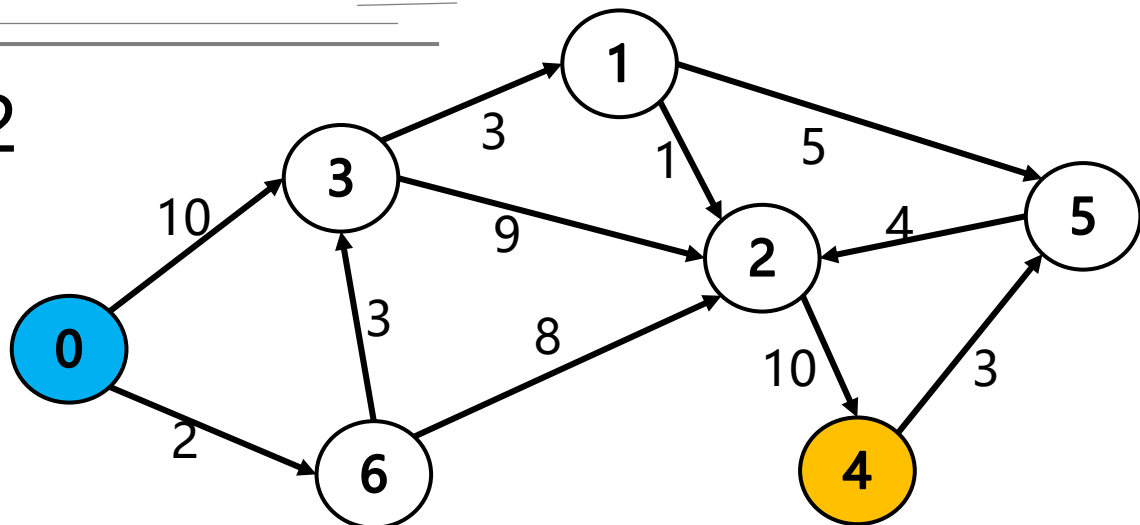
队列[1,2]

队列[2,5]

被扩展过节点需要标注, 避免重复扩展

图搜索——一致代价搜索(Uniform-Cost-Search, UCS)

图2



从0开始, 扩展3,6。 [3(10),6(2)]
取出6, 扩展2,3。 [3(5),2(10)]
取出3, 扩展1。 [2(10),1(8)]
取出1, 扩展2,5。 [2(9),5(13)]
取出2, 扩展4。 [4(19),5(13)]
取出5, 无可扩展。 (注意)
取出4, 找到。
路径: 4<-2<-1<-3<-6<-0。长度19, 最短。

算法流程:

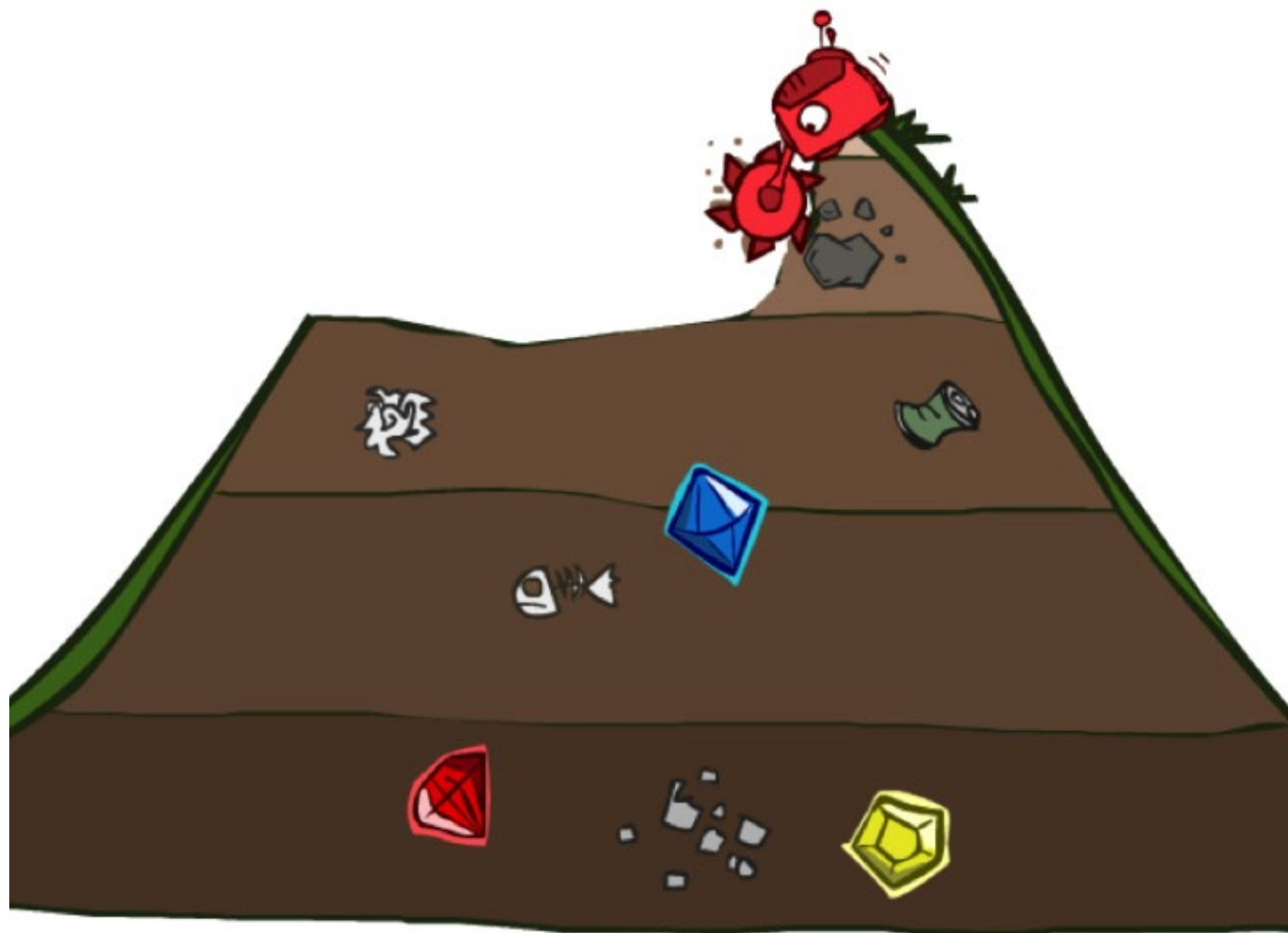
- 从起点开始, 一层层向外扩展
- 被扩展节点加入优先队列
- 每次取出优先队列中距离起点最小的点, 向外扩展

被**取出**的节点需要标注, 避免重复扩展

图搜索——一致代价搜索(Uniform-Cost-Search, UCS)

1. 一定找到最短路
2. 有路一定找得到

UCS在简单图里又称
Dijkstra 算法



最短路问题

单源最短路问题

从一个源点 w 到图中其他所有点的最短路.

- Dijkstra 算法, 要求正权图
- Bellman-Ford 算法, 可负权并可检测负环

多源最短路问题

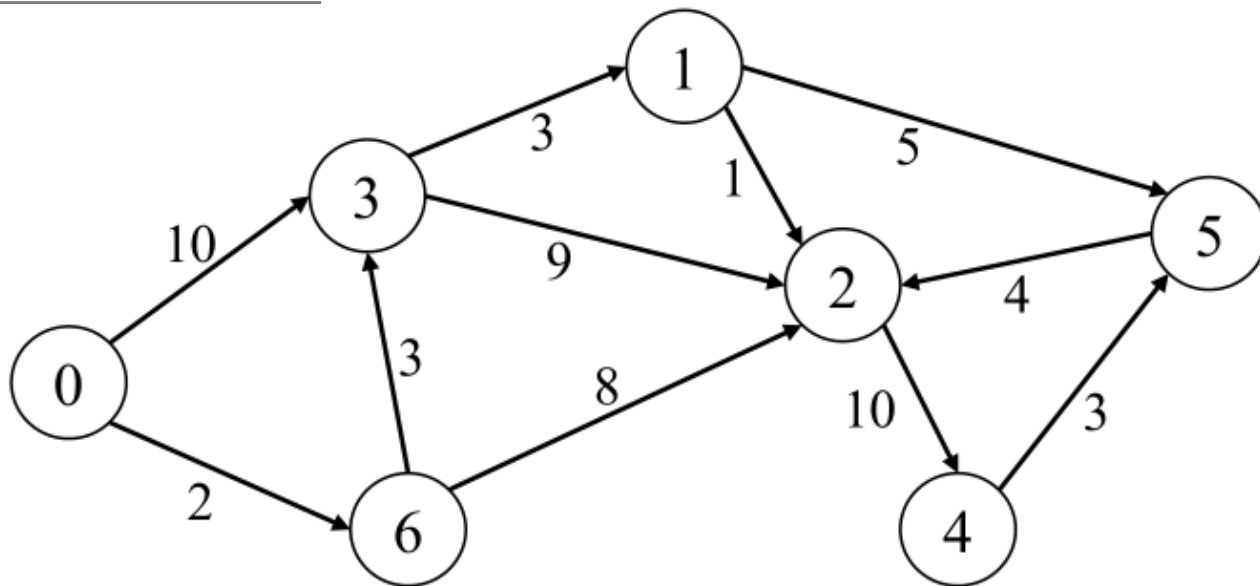
给出图中所有点两两间最短距离

- Floyd-Warshall 算法
- Johnson 算法

最短路——Dijkstra 算法

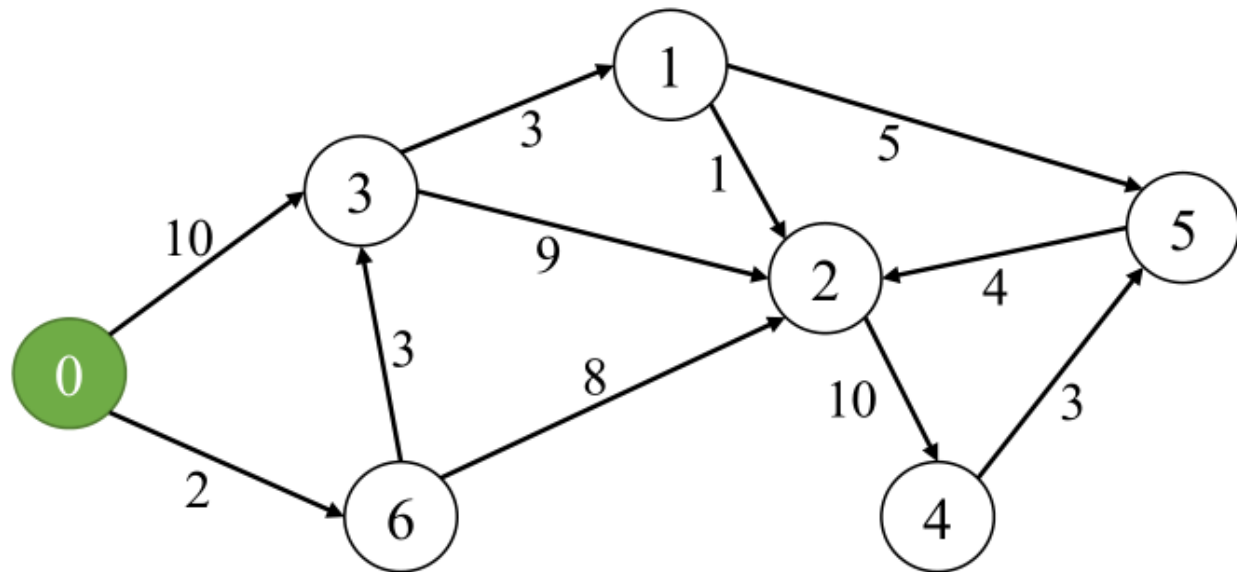
```
1 function Dijkstra(Graph, source):
2
3     create vertex set Q
4
5     for each vertex  $v$  in Graph:
6         dist[ $v$ ]  $\leftarrow$  INFINITY
7         prev[ $v$ ]  $\leftarrow$  UNDEFINED
8         add  $v$  to  $Q$ 
10    dist[source]  $\leftarrow$  0
11
12    while  $Q$  is not empty:
13         $u \leftarrow$  vertex in  $Q$  with min dist[ $u$ ]
14
15        remove  $u$  from  $Q$ 
16
17        for each neighbor  $v$  of  $u$ :           // only  $v$  that are still in  $Q$ 
18            alt  $\leftarrow$  dist[ $u$ ] + length( $u, v$ )
19            if alt < dist[ $v$ ]:
20                dist[ $v$ ]  $\leftarrow$  alt
21                prev[ $v$ ]  $\leftarrow$   $u$ 
22
23    return dist[], prev[]
```

最短路——Dijkstra 算法



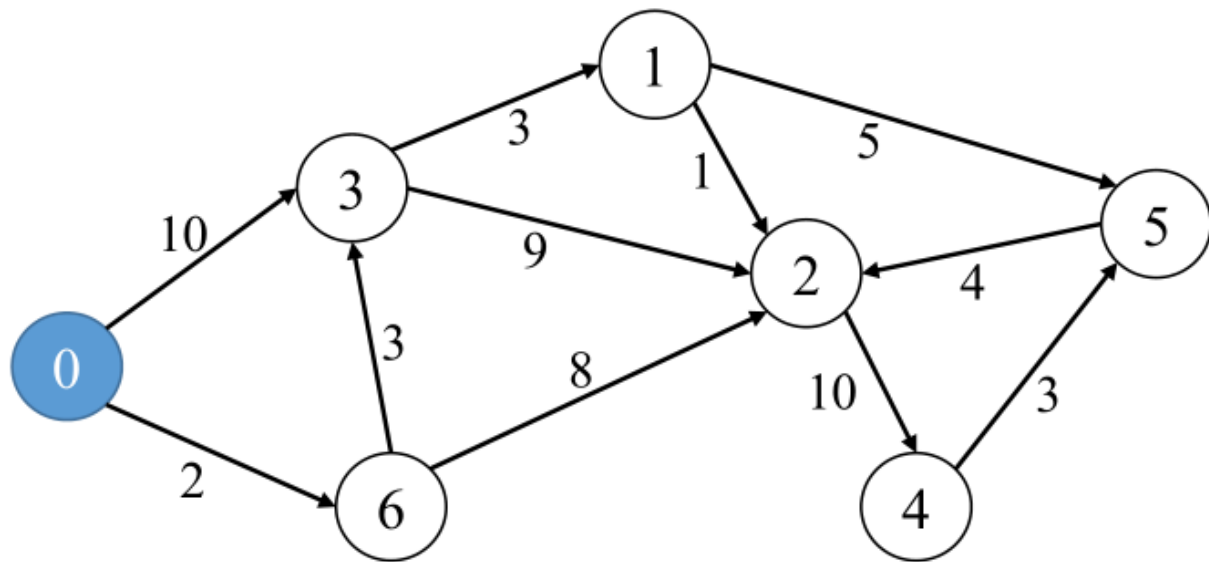
Node	0	1	2	3	4	5	6
d[·]	-	-	-	-	-	-	-
S	No	No	No	No	No	No	No
pre	-	-	-	-	-	-	-

最短路——Dijkstra 算法



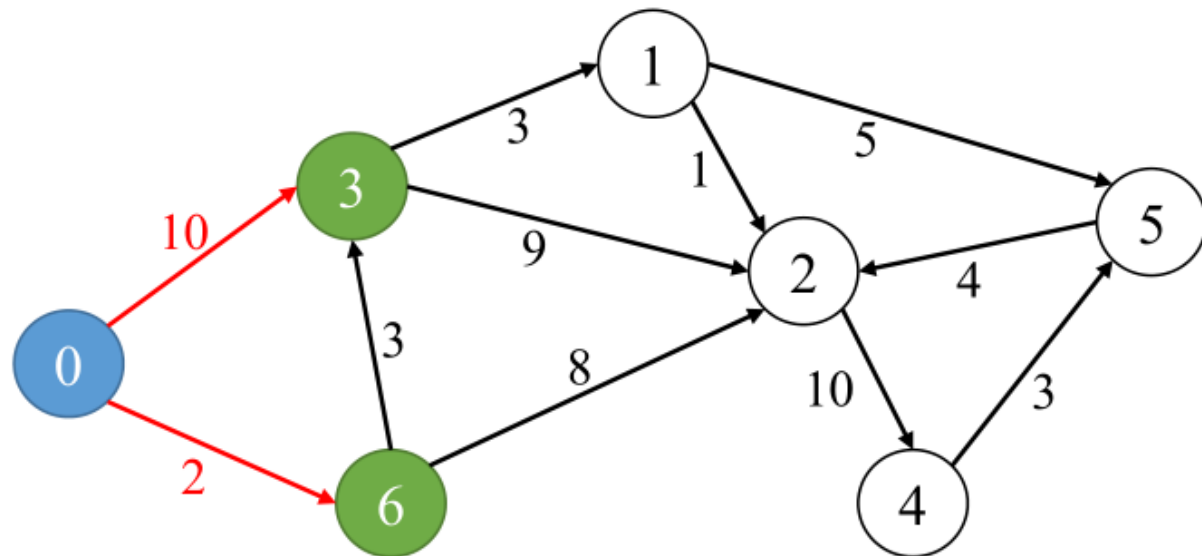
Node	0	1	2	3	4	5	6
d[·]	0	-	-	-	-	-	-
S	No	No	No	No	No	No	No
pre	0	-	-	-	-	-	-

最短路——Dijkstra 算法



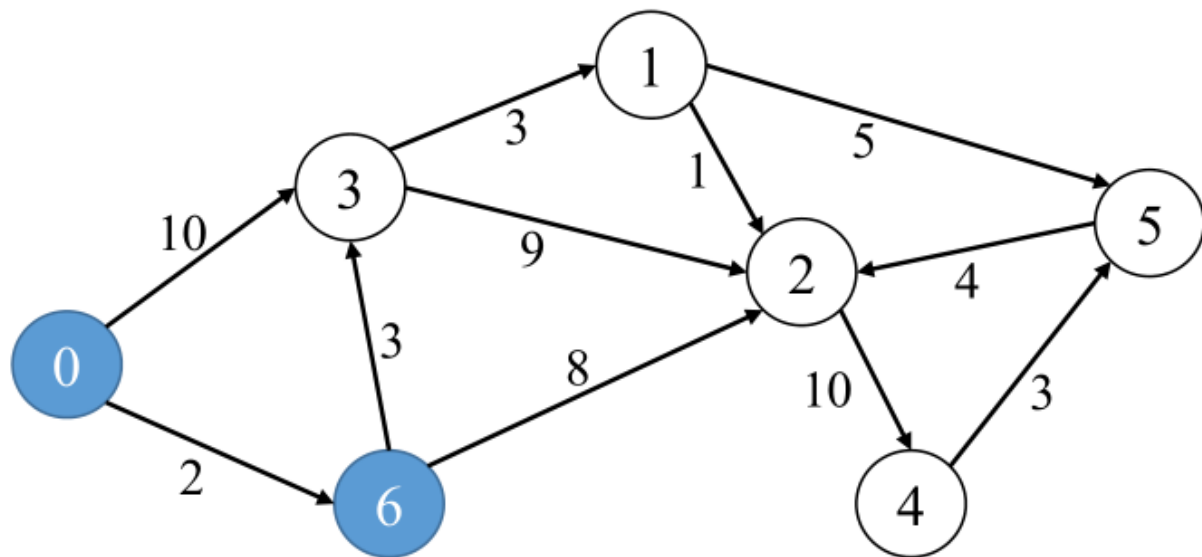
Node	0	1	2	3	4	5	6
d[·]	0	-	-	-	-	-	-
S	Yes	No	No	No	No	No	No
pre	0	-	-	-	-	-	-

最短路——Dijkstra 算法



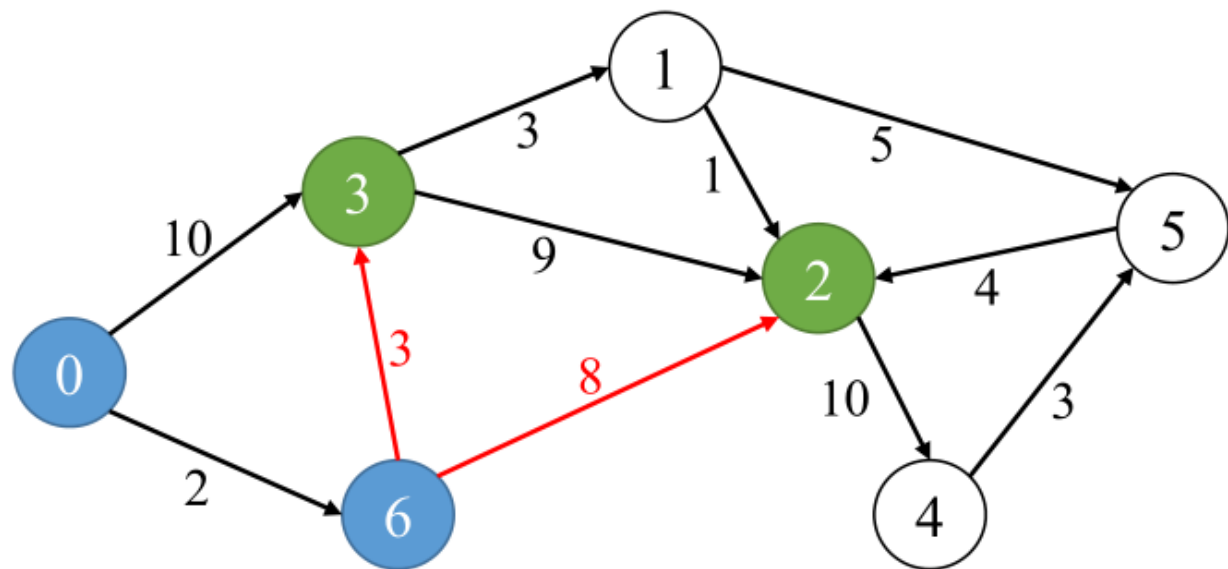
Node	0	1	2	3	4	5	6
d[·]	0	-	-	10	-	-	2
S	Yes	No	No	No	No	No	No
pre	0	-	-	0	-	-	0

最短路——Dijkstra 算法



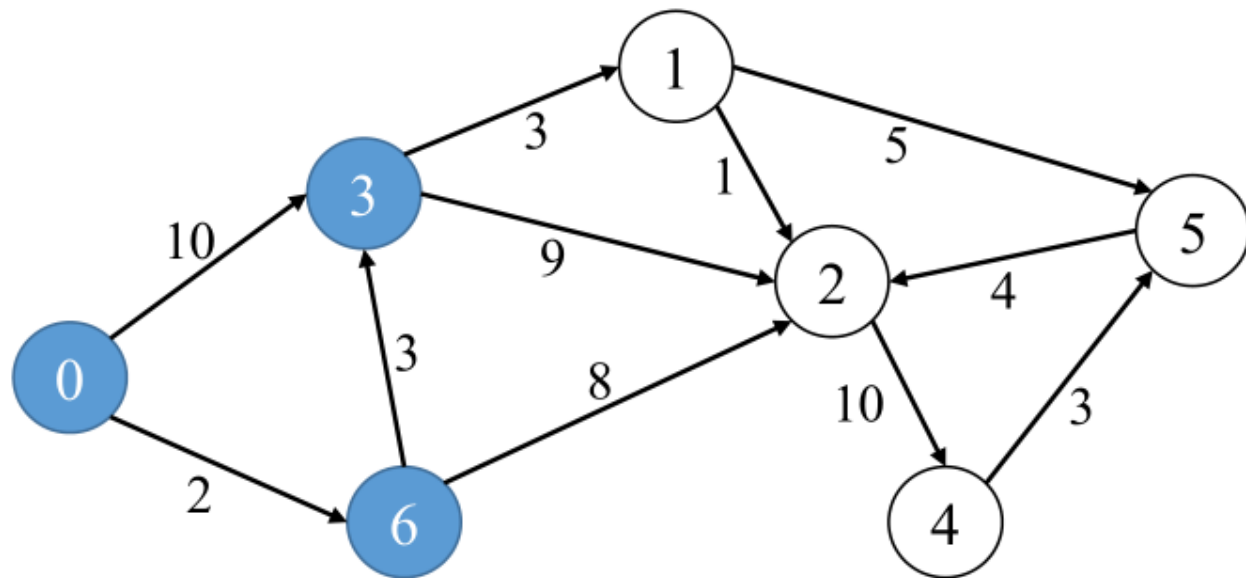
Node	0	1	2	3	4	5	6
d[·]	0	-	-	10	-	-	2
S	Yes	No	No	No	No	No	Yes
pre	0	-	-	0	-	-	0

最短路——Dijkstra 算法



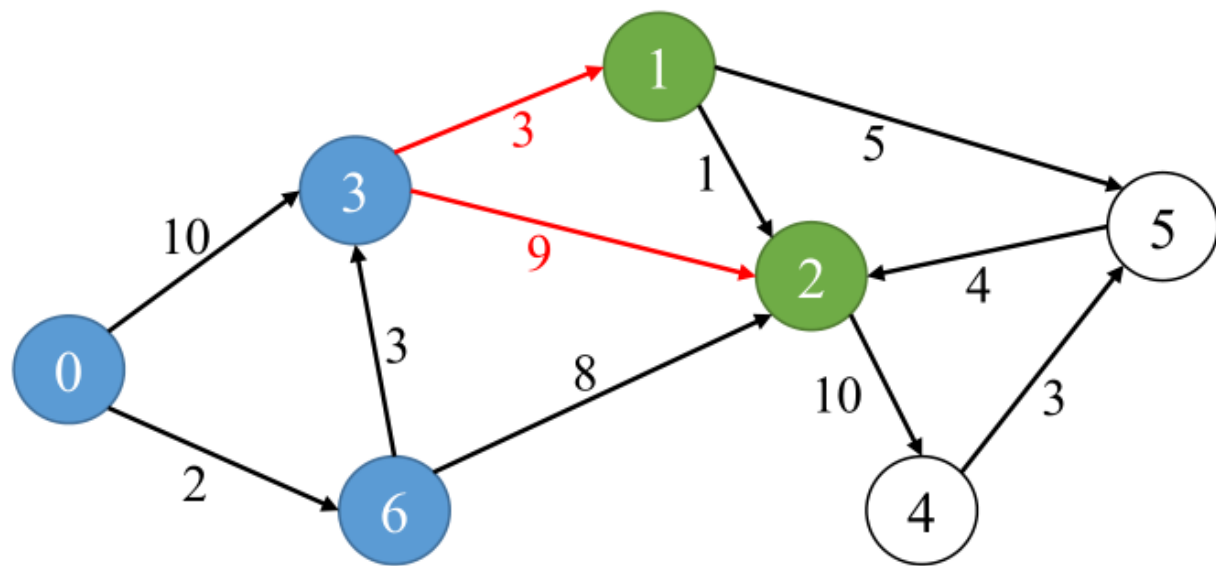
Node	0	1	2	3	4	5	6
d[·]	0	-	10	5	-	-	2
S	Yes	No	No	No	No	No	Yes
pre	0	-	6	6	-	-	0

最短路——Dijkstra 算法



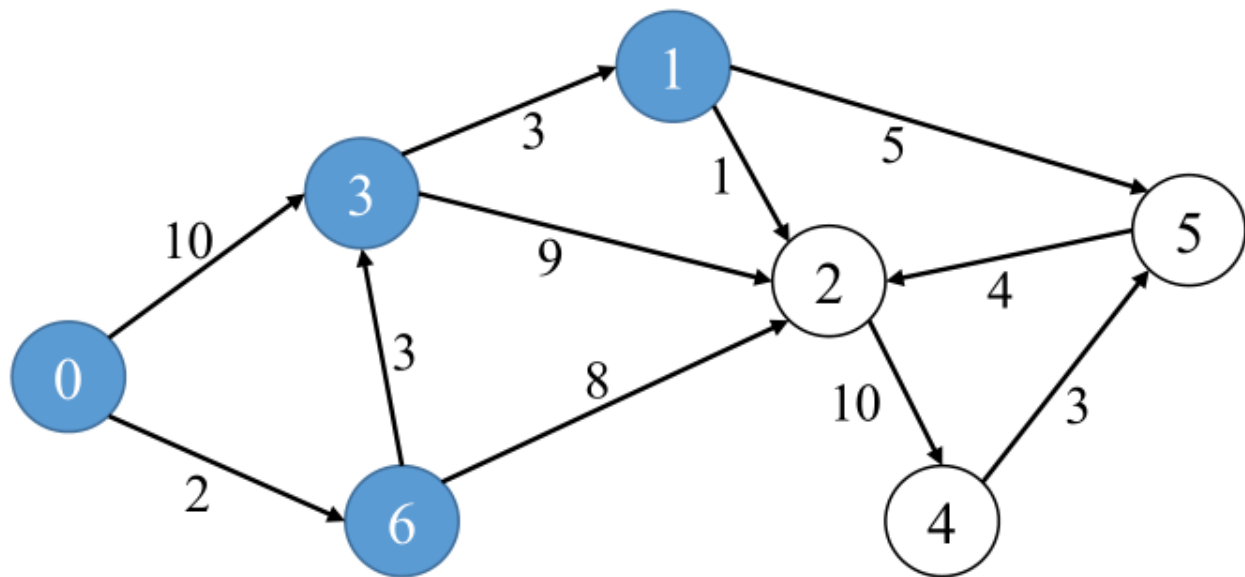
Node	0	1	2	3	4	5	6
d[·]	0	-	10	5	-	-	2
S	Yes	No	No	Yes	No	No	Yes
pre	0	-	6	6	-	-	0

最短路——Dijkstra 算法



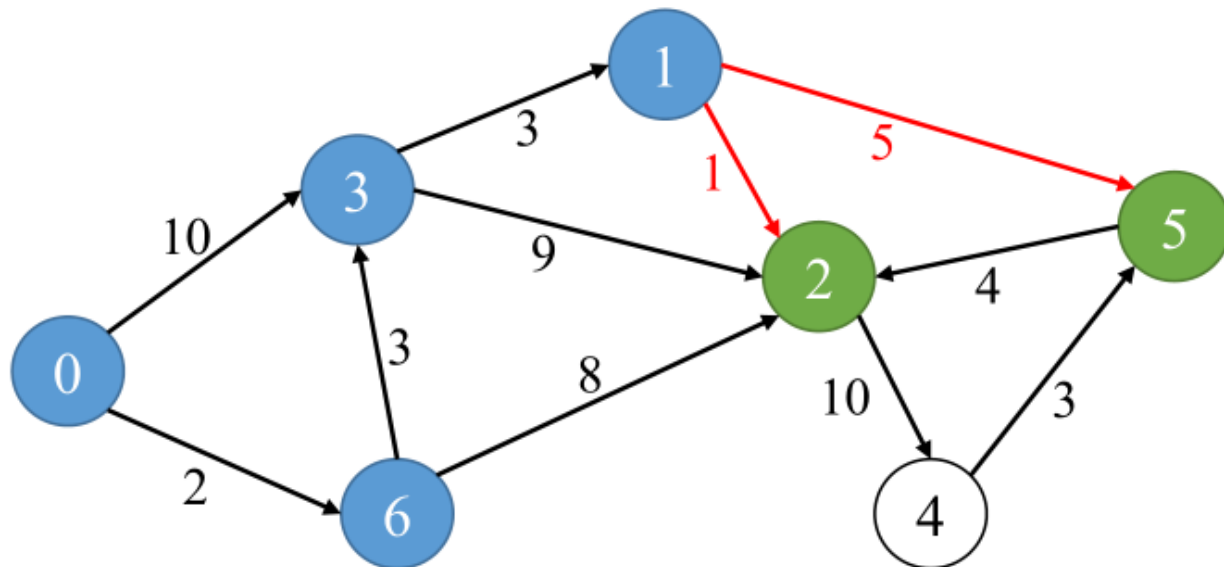
Node	0	1	2	3	4	5	6
d[·]	0	8	10	5	-	-	2
S	Yes	No	No	Yes	No	No	Yes
pre	0	3	6	6	-	-	0

最短路——Dijkstra 算法



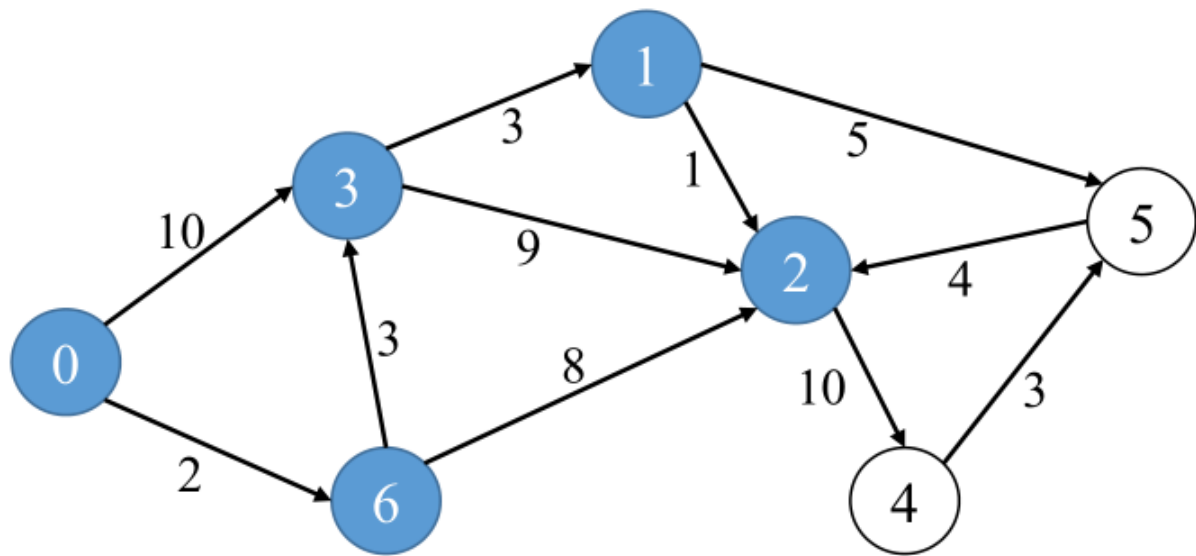
Node	0	1	2	3	4	5	6
d[·]	0	8	10	5	-	-	2
S	Yes	Yes	No	Yes	No	No	Yes
pre	0	3	6	6	-	-	0

最短路——Dijkstra 算法



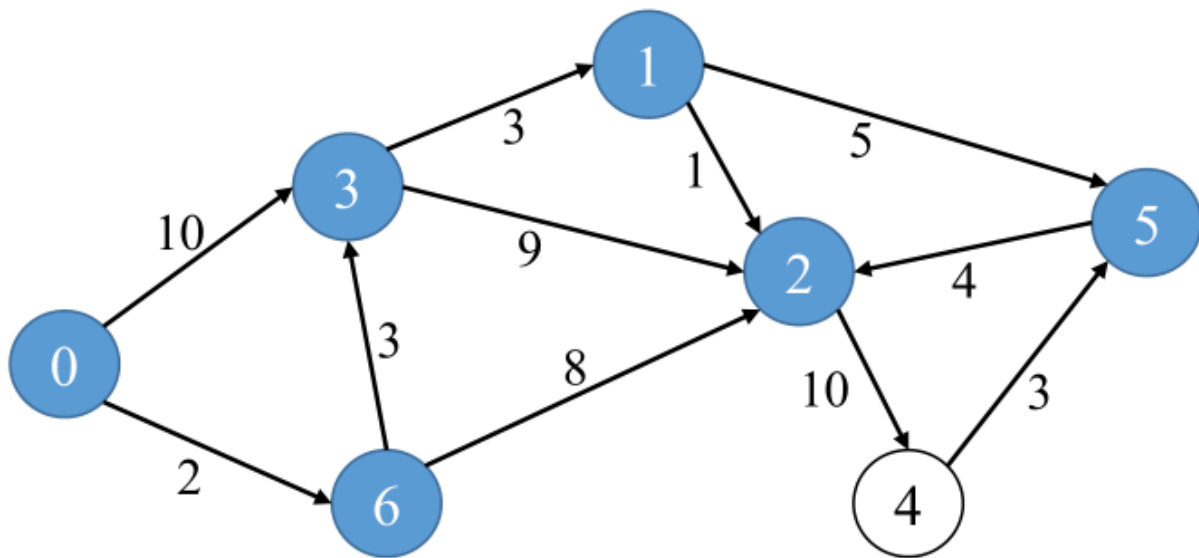
Node	0	1	2	3	4	5	6
d[·]	0	8	9	5	-	13	2
S	Yes	Yes	No	Yes	No	No	Yes
pre	0	3	1	6	-	1	0

最短路——Dijkstra 算法



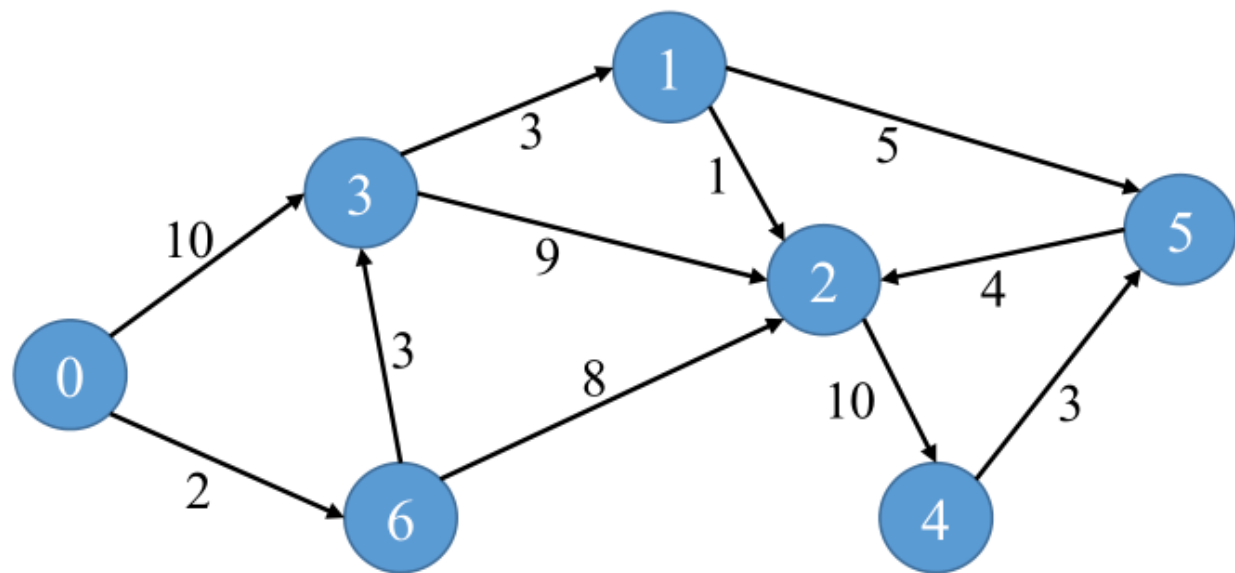
Node	0	1	2	3	4	5	6
d[·]	0	8	9	5	-	13	2
S	Yes	Yes	Yes	Yes	No	No	Yes
pre	0	3	1	6	-	1	0

最短路——Dijkstra 算法



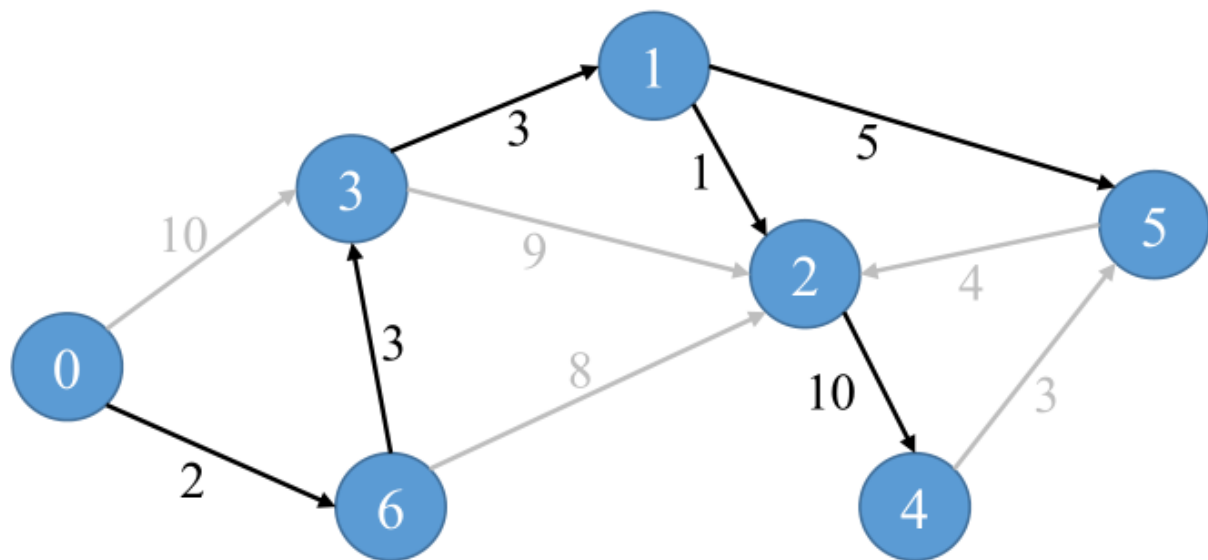
Node	0	1	2	3	4	5	6
d[·]	0	8	9	5	19	13	2
S	Yes	Yes	Yes	Yes	No	Yes	Yes
pre	0	3	1	6	2	1	0

最短路——Dijkstra 算法



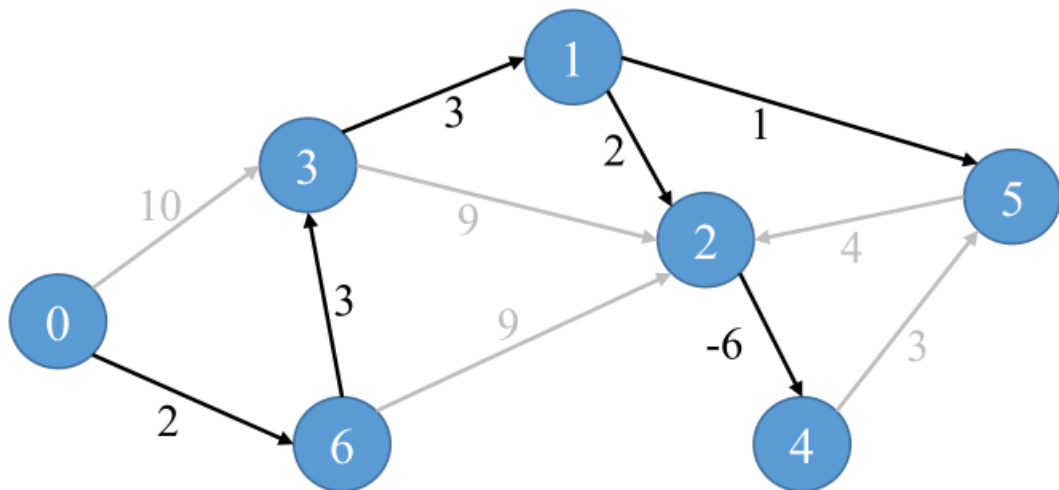
Node	0	1	2	3	4	5	6
d[·]	0	8	9	5	19	13	2
S	Yes	Yes	Yes	Yes	Yes	Yes	Yes
pre	0	3	1	6	2	1	0

最短路——Dijkstra 算法



Node	0	1	2	3	4	5	6
d[·]	0	8	9	5	19	13	2
S	Yes	Yes	Yes	Yes	Yes	Yes	Yes
pre	0	3	1	6	2	1	0

最短路——Dijkstra 算法：不允许负权



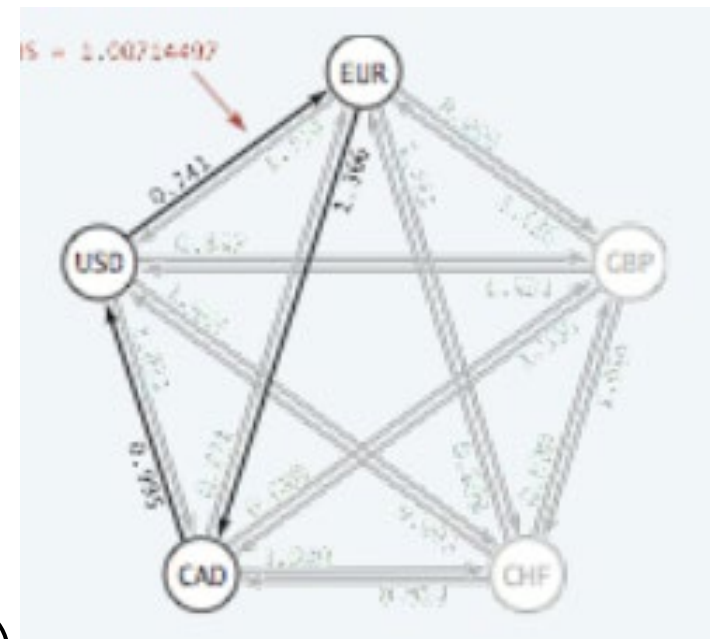
Node	0	1	2	3	4	5	6
d[·]	0	8	10	5	4	9	2
S	Yes	Yes	Yes	Yes	Yes	Yes	Yes
pre	0	3	1	6	2	1	0

Wrong Answer

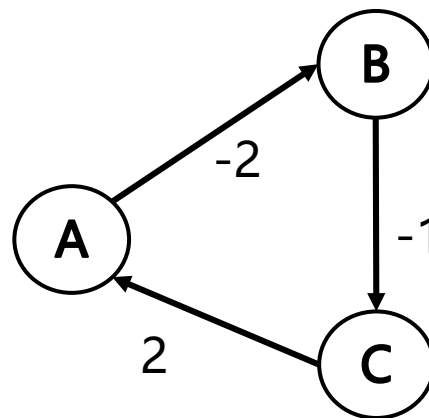
最短路——负权的意义举例

负环检测的应用举例：防止金融风险

右图是各种货币汇率构成的完全图，两点间最短路长度代表什么？出现负环意味着什么？



兑换方式	汇率	$\lg(\text{汇率})$
A->B	1:100	-2
B->C	1:10	-1
C->A	1:0.01	2



比如，1块钱换一圈变10块

单源最短路——Bellman-Ford 算法

美国应用数学家Richard Bellman, 1958 年。

此外Lester Ford在1956年发表。

算法又称Bellman-Ford算法。

其实Edward F. Moore在1957年也发表,

算法也称Bellman-Ford-Moore算法。

可以处理负权图, 检测是否含有负环

单源最短路——Bellman-Ford 算法

循环 $|V|$ 次：
遍历每条边 (u,v) ：
 $d[v] = \min(d[v], d[u] + w(u,v))$

复杂度 $O(|V||E|)$

很好记，但如何理解？
动态规划，压缩了一个维度以节省空间

```
function BellmanFord(list vertices, list edges, vertex source) is
    ::distance[], predecessor[]

    // This implementation takes in a graph, represented as
    // lists of vertices and edges, and fills two arrays
    // (distance and predecessor) about the shortest path
    // from the source to each vertex

    // Step 1: initialize graph
    for each vertex v in vertices do
        distance[v] := inf           // Initialize the distance to all vertices to infinity
        predecessor[v] := null       // And having a null predecessor

    distance[source] := 0           // The distance from the source to itself is, of course, zero

    // Step 2: relax edges repeatedly
    for i from 1 to size(vertices)-1 do //just |V|-1 repetitions; i is never referenced
        for each edge (u, v) with weight w in edges do
            if distance[u] + w < distance[v] then
                distance[v] := distance[u] + w
                predecessor[v] := u

    // Step 3: check for negative-weight cycles
    for each edge (u, v) with weight w in edges do
        if distance[u] + w < distance[v] then
            error "Graph contains a negative-weight cycle"

    return distance[], predecessor[]
```


单源最短路——Bellman-Ford 算法

循环 $|V|$ 次:

遍历每条边 (u,v) :

$d[v] = \min(d[v],$
 $d[u] + w(u,v))$

```
for i from 1 to size(vertices)-1 do //just |V|-1 repetitions; i is never referenced
  for each edge (u, v) with weight w in edges do
    if distance[u] + w < distance[v] then
      distance[v] := distance[u] + w
      predecessor[v] := u
```

算法理解：动态规划，压缩了一个维度以节省空间。

$dp[v][k]$ 以**某不少于 k 个**节点为中继，源点和点 v 最短距离

有递推式 $dp[v][k] = \min(dp[v][k],$

$\min_u dp[u][k-1] + w(u,v))$

第二个维度被压缩了，用 $d[]$ 这个一维数组就足够存储 $dp[][]$ 的内容

单源最短路——Bellman-Ford 算法

循环 $|V|$ 次:

遍历每条边 (u,v) :

$d[v] = \min(d[v],$
 $d[u] + w(u,v))$

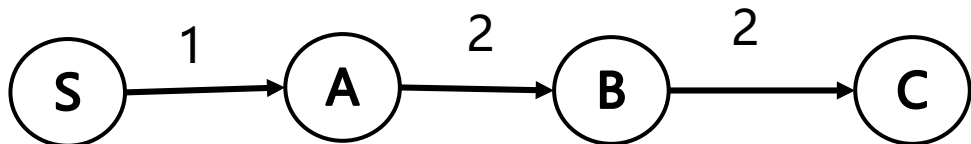
```
for i from 1 to size(vertices)-1 do //just |V|-1 repetitions; i is never referenced
  for each edge (u, v) with weight w in edges do
    if distance[u] + w < distance[v] then
      distance[v] := distance[u] + w
      predecessor[v] := u
```

$dp[j][k]$ 是以**某不少于k个**节点为中继，源点和点j最短距离。 ✓

$dp[j][k]$ 是以k个节点为中继，源点和点j最短距离。 ✗

$dp[j][k]$ **不大于**以某k个节点为中继，源点和点j最短距离。 ✓

例子：遍历顺序得当的话一轮之后就有 $d[C]=5$.



多源最短路——Floyd算法

算法很好记但如何理解?

复杂度 $O(|V|^3)$

```
let dist be a  $|V| \times |V|$  array of minimum distances initialized to  $\infty$  (infinity)
for each edge  $(u, v)$  do
    dist[u][v]  $\leftarrow$   $w(u, v)$  // The weight of the edge  $(u, v)$ 
for each vertex  $v$  do
    dist[v][v]  $\leftarrow$  0
for  $k$  from 1 to  $|V|$ 
    for  $i$  from 1 to  $|V|$ 
        for  $j$  from 1 to  $|V|$ 
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j]  $\leftarrow$  dist[i][k] + dist[k][j]
            end if
```

理解：动态规划，压缩了一个维度以节省空间

多源最短路——Floyd算法

理解：动态规划，压缩了一个维度以节省空间

$dp[i][j][k]$ 表示以前 k 个节点为中继，点 i 和点 j 最短距离

有递推式 $dp[i][j][k]=\min(dp[i][j][k],$

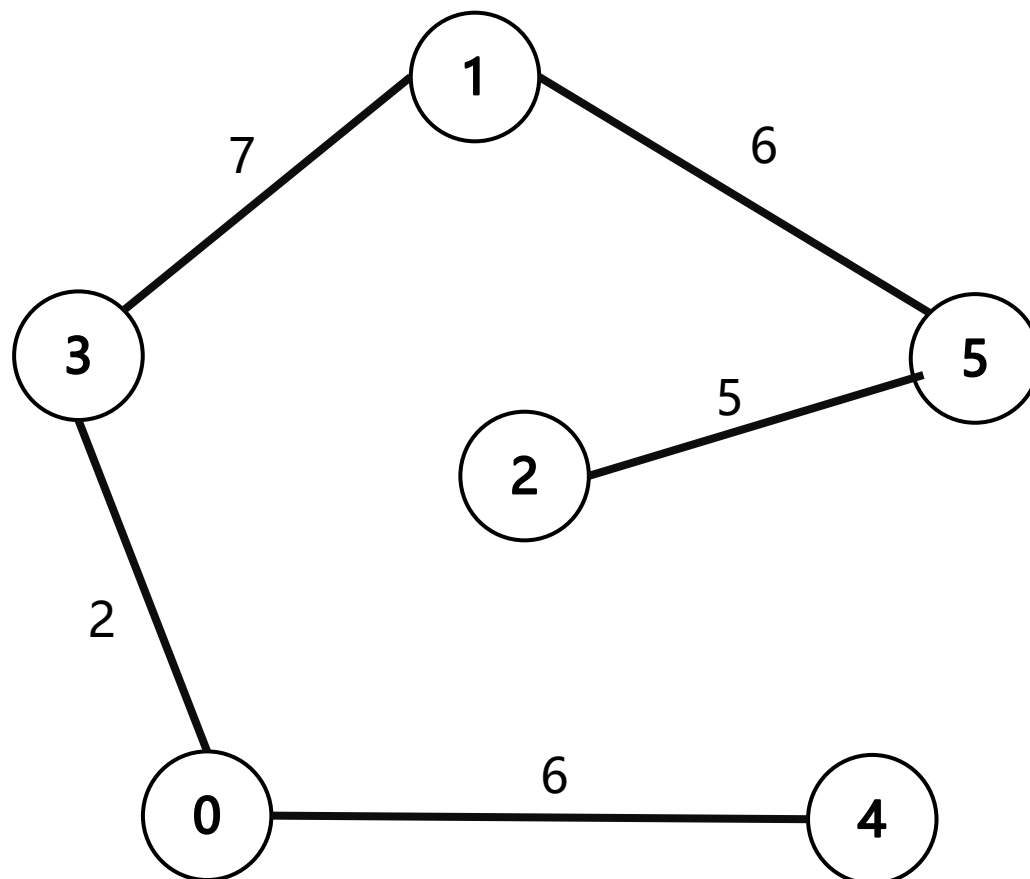
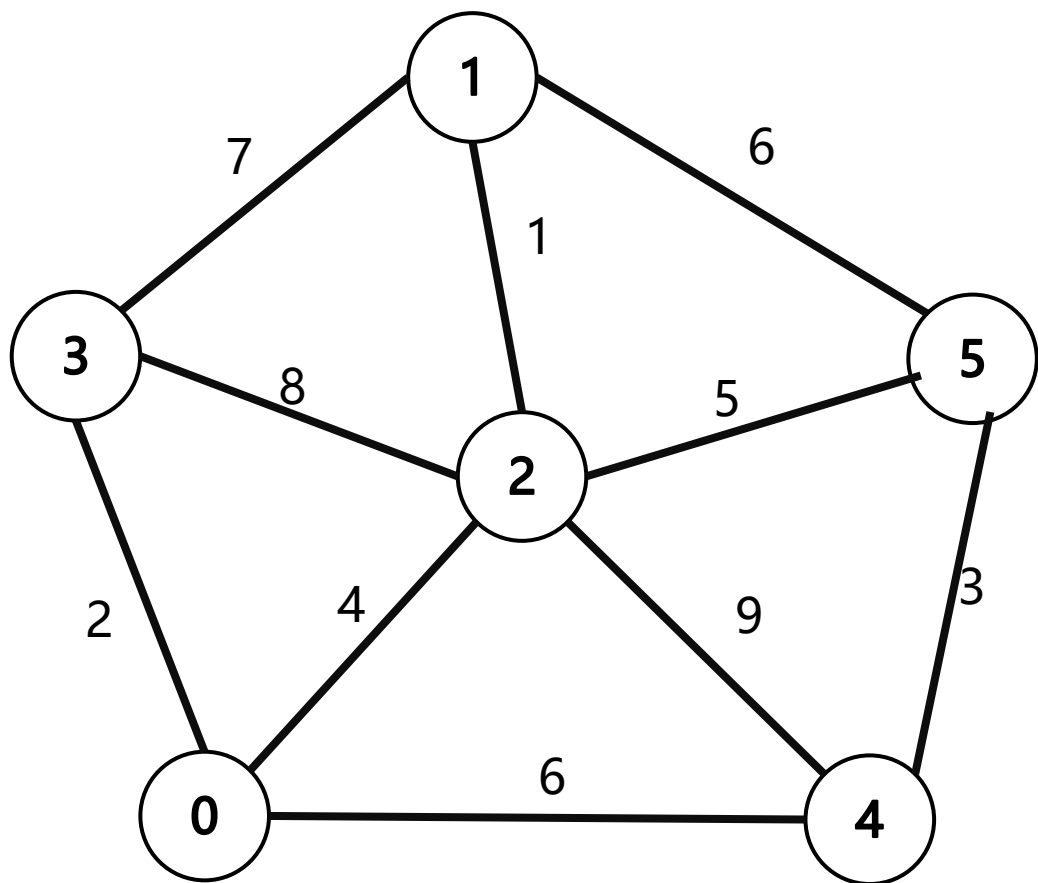
$dp[i][k][k-1]+dp[k][j][k-1])$

```
for k from 1 to |V|
  for i from 1 to |V|
    for j from 1 to |V|
      if dist[i][j] > dist[i][k] + dist[k][j]
        dist[i][j] ← dist[i][k] + dist[k][j]
      end if
```

哈密顿路——旅行商问题(Traveling Salesman Problem, TSP)

哈密顿回路：一条遍历所有节点的回路

TSP：找总长度最小的哈密顿回路，NPH问题



哈密顿路——TSP的求解算法

暴力搜索(如DFS), $O(n!)$
动态规划算法: $O(n^2 2^n)$

局部搜索, 启发式搜索
最小生成树近似解

哈密顿路——TSP的01规划求解

设城市的个数为 n , d_{ij} 是两个城市 i 和 j 之间的距离, $x_{ij} = 0$ 或 1 (1表示走过城市 i 到城市 j 的路, 0表示没有选择走这条路)。则有

$$\min \sum_{i \neq j} d_{ij} x_{ij},$$

复杂度还是指数


有规划求解器

$$s.t. \begin{cases} \sum_{j=1}^n x_{ij} = 1, i = 1, 2, \dots, n, (\text{每个点只有一条边出去}) \\ \sum_{i=1}^n x_{ij} = 1, i = 1, 2, \dots, n, (\text{每个点只有一条边进去}) \\ \sum_{i,j \in s} x_{ij} \leq |s| - 1, 2 \leq |s| \leq n - 1, s \subset \{1, 2, \dots, n\}, \\ (\text{除起点和终点外, 各边不构成圈}) \\ x_{ij} \in \{0, 1\}, i, j = 1, 2, \dots, n, i \neq j. \end{cases}$$



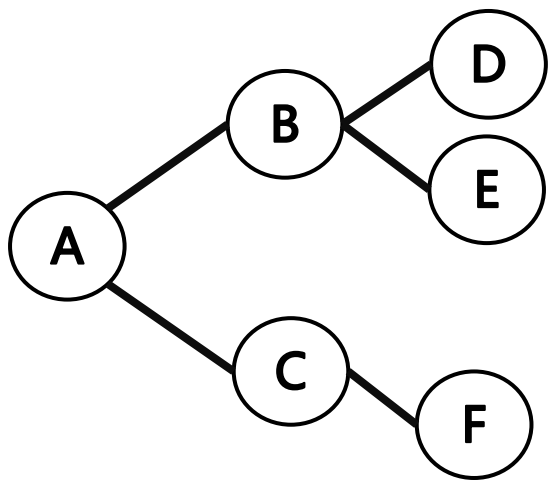
P2

图算法2：树

- 
- 树与生成树
 - 最小生成树问题
 - 斯坦纳树
 - 哈夫曼树

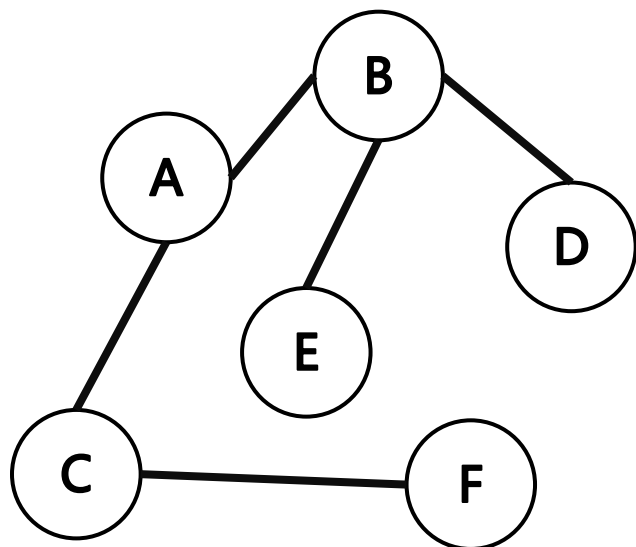
树的概念：连通无环图

典型的树

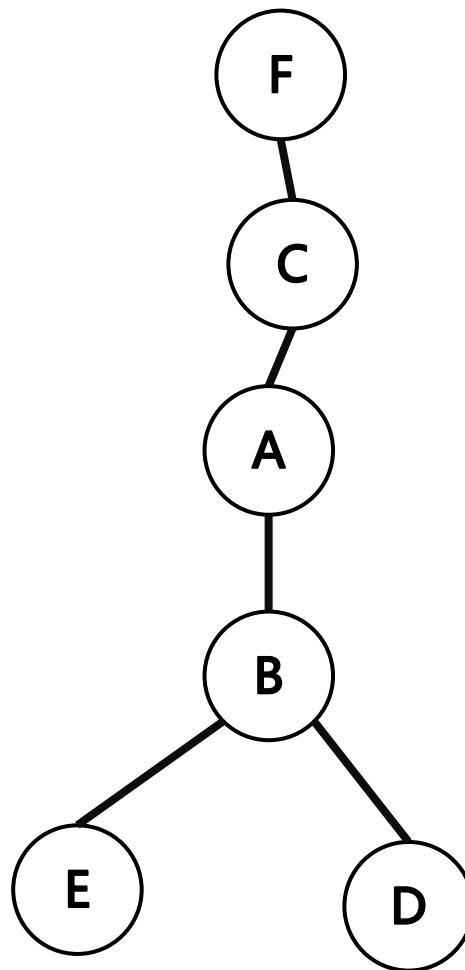


根, 叶子
父、子节点

还是树



换个根也不妨



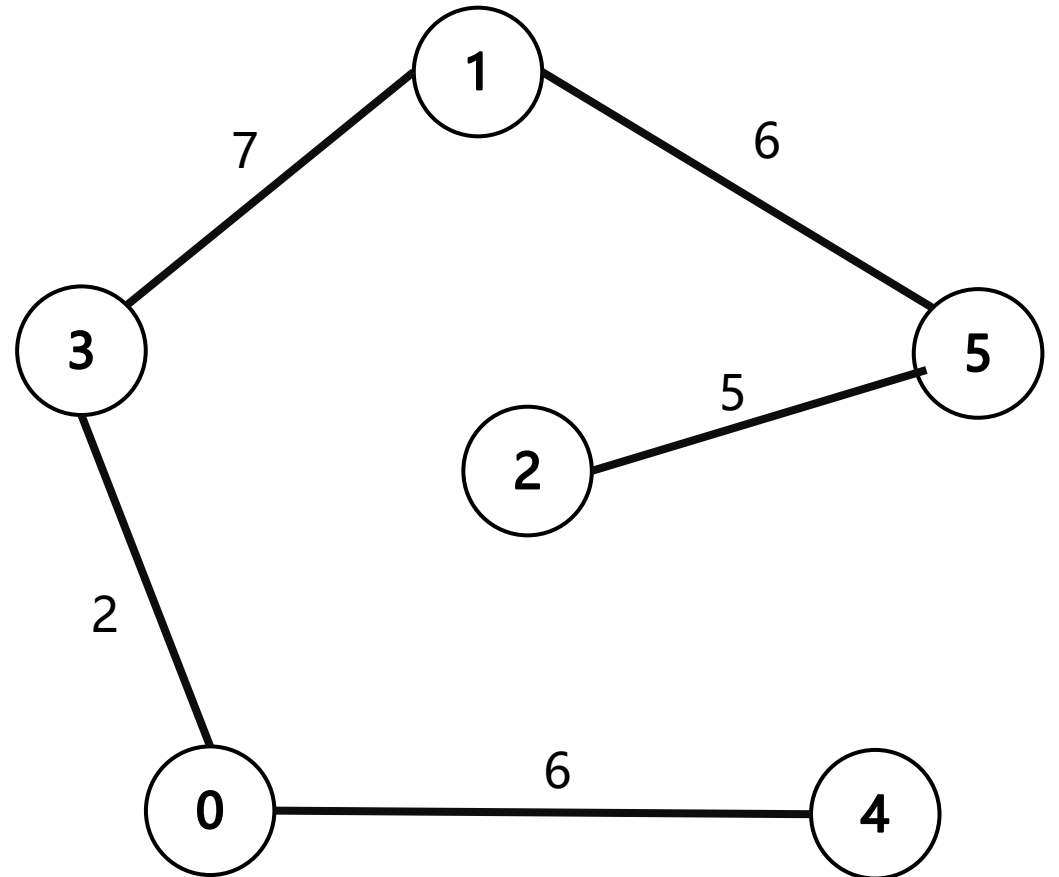
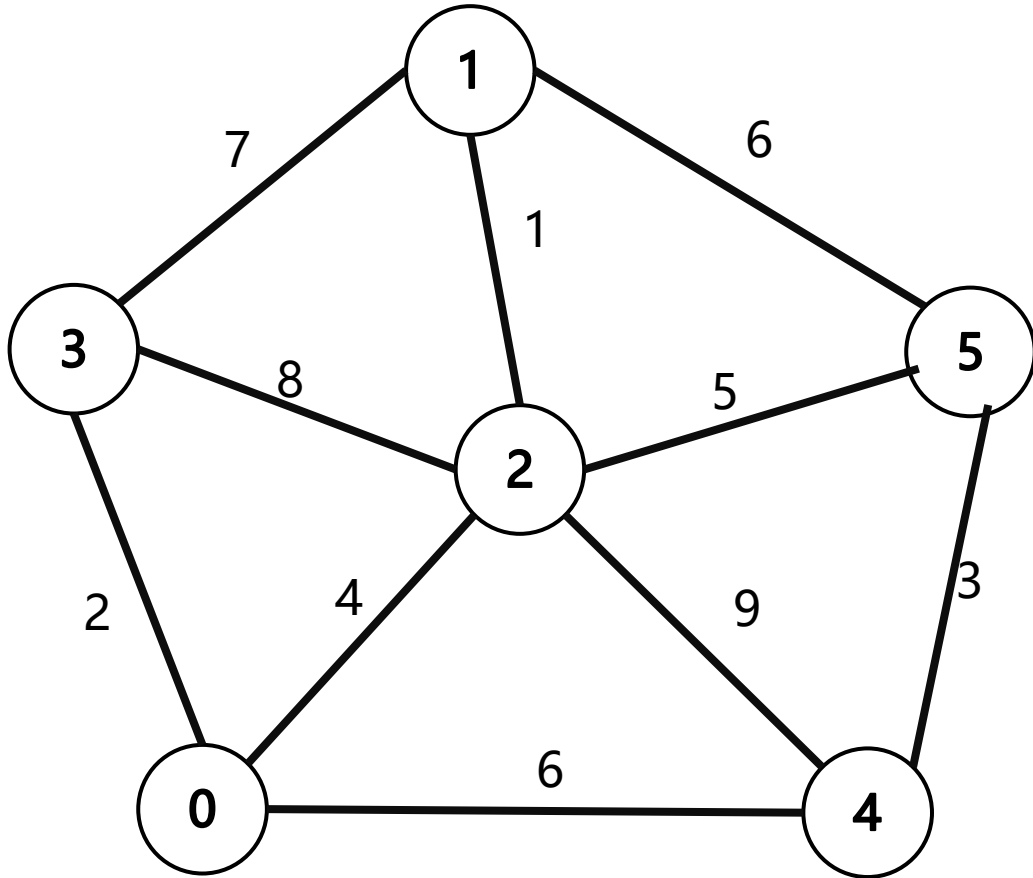
核心属性

1. 连通
2. 无环

图的生成树

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

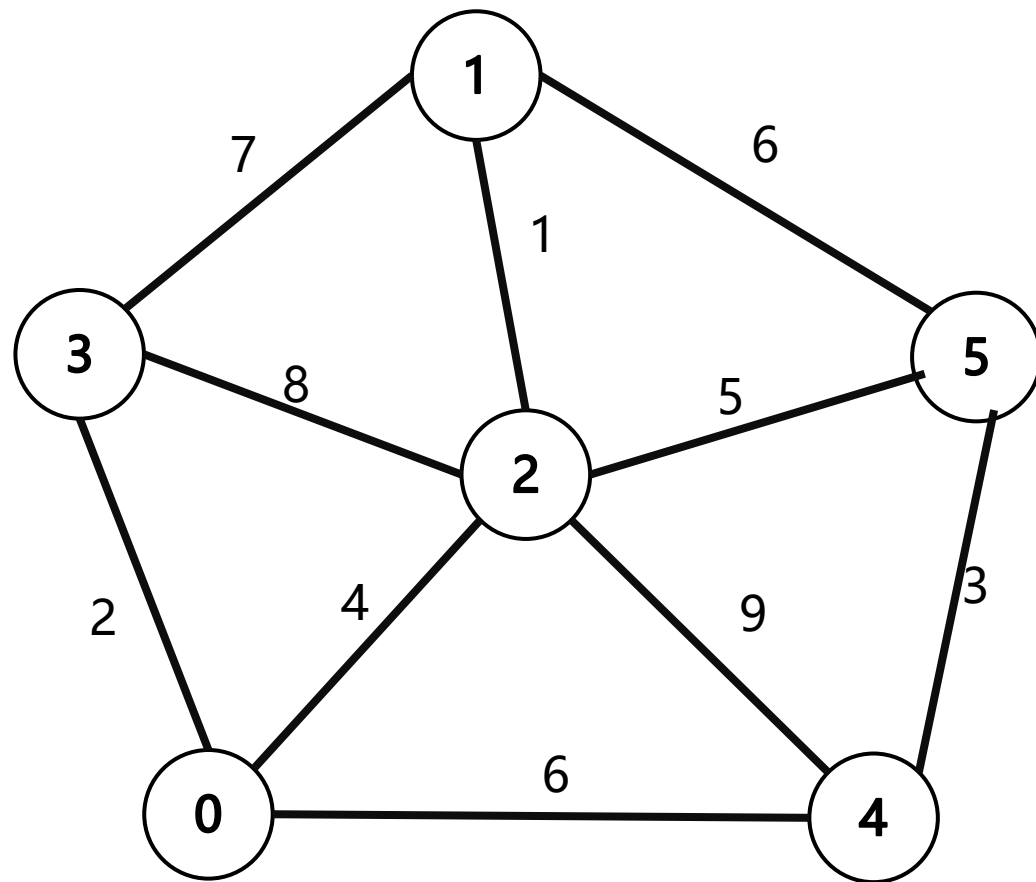


最小生成树问题

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

例：右图表示一个城市的供水管道网，权值表示线路长度，要求使得每个地点通水，求**最短**保留的管道总长度



尝试枚举：有多少种生成树？

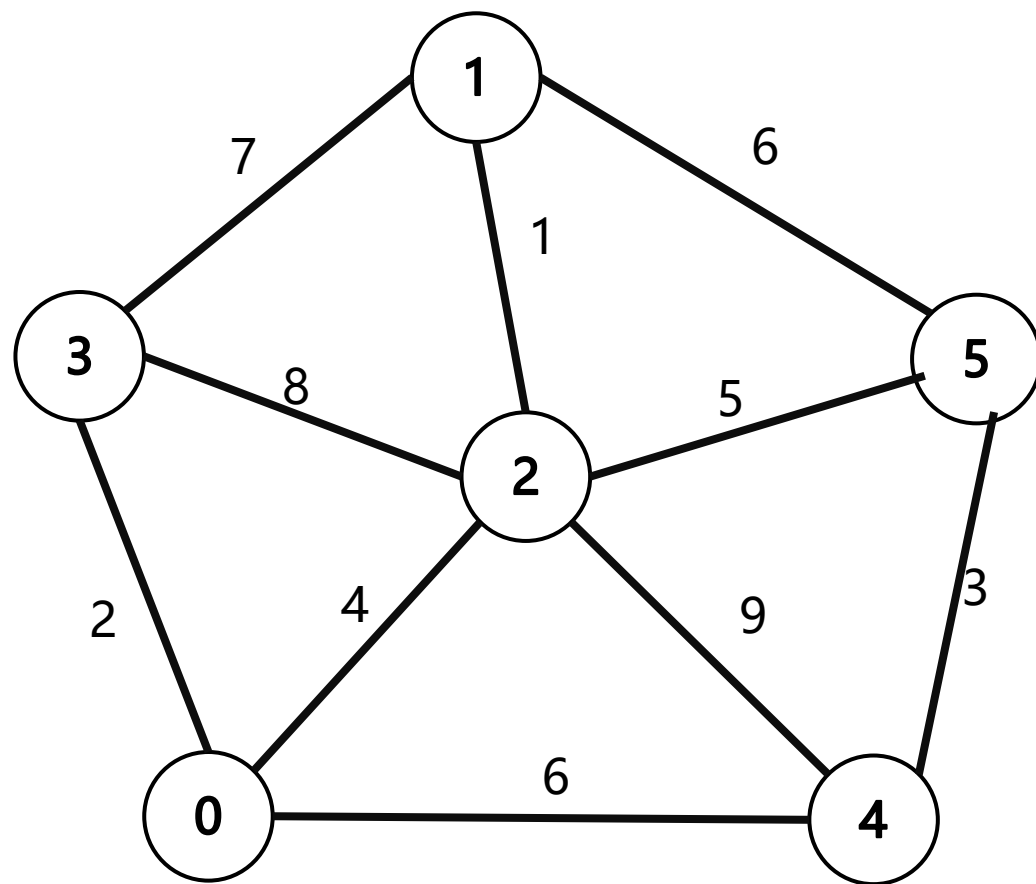
尝试贪心：如以下三种方法

最小生成树——Prim算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法1：从一个点开始，用最小权的边去连其他点。

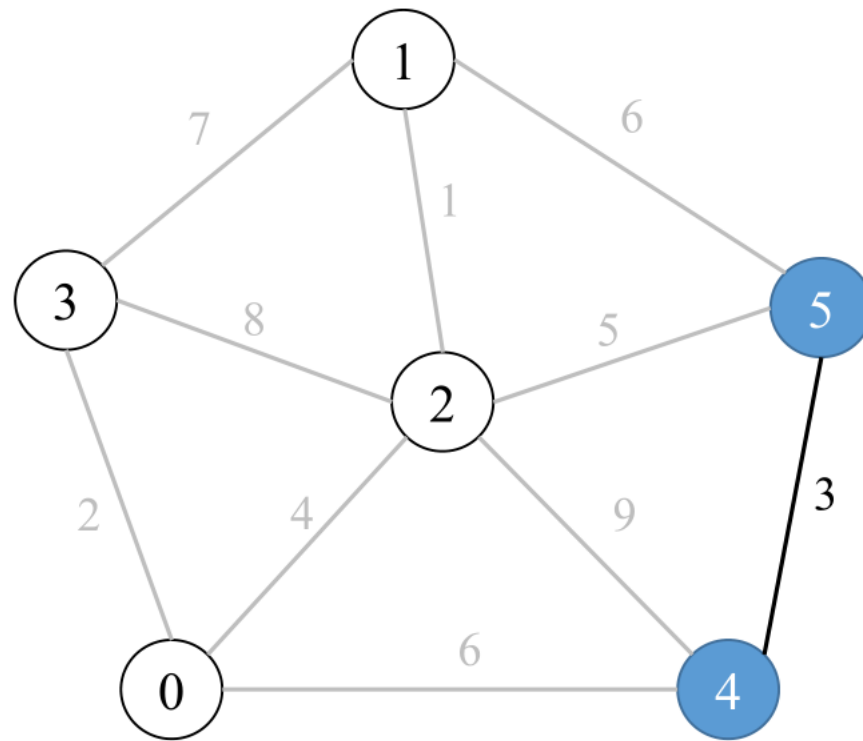
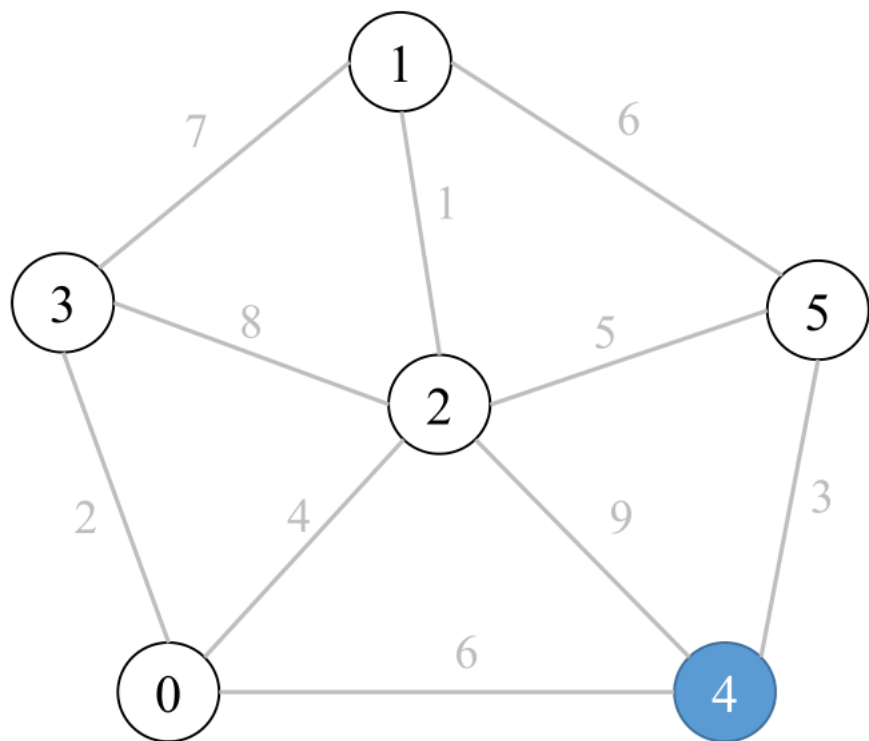


最小生成树——Prim算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法1：从一个点开始，用最小权的边去连其他点。

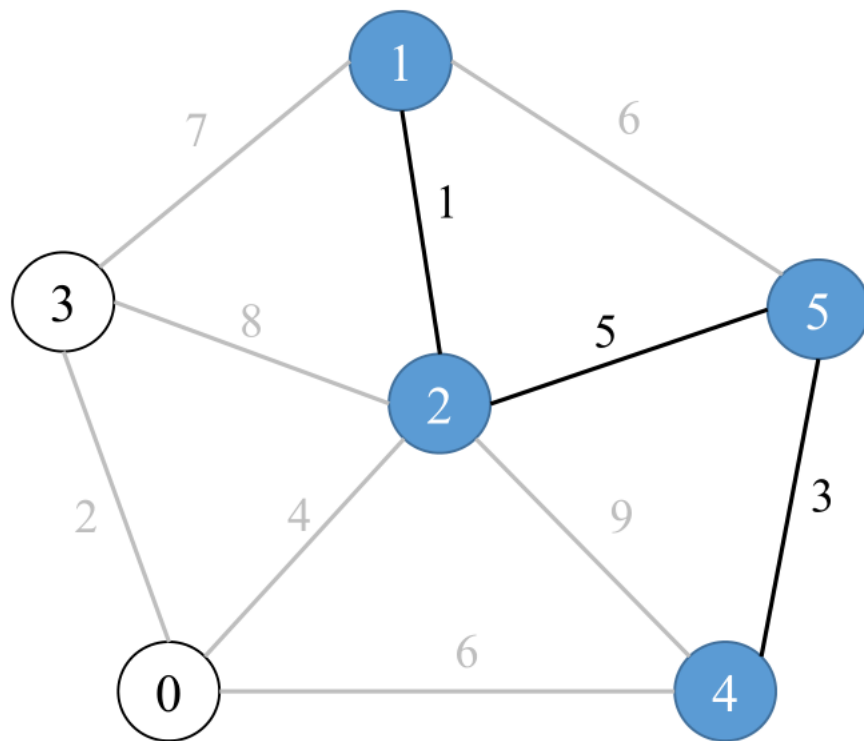
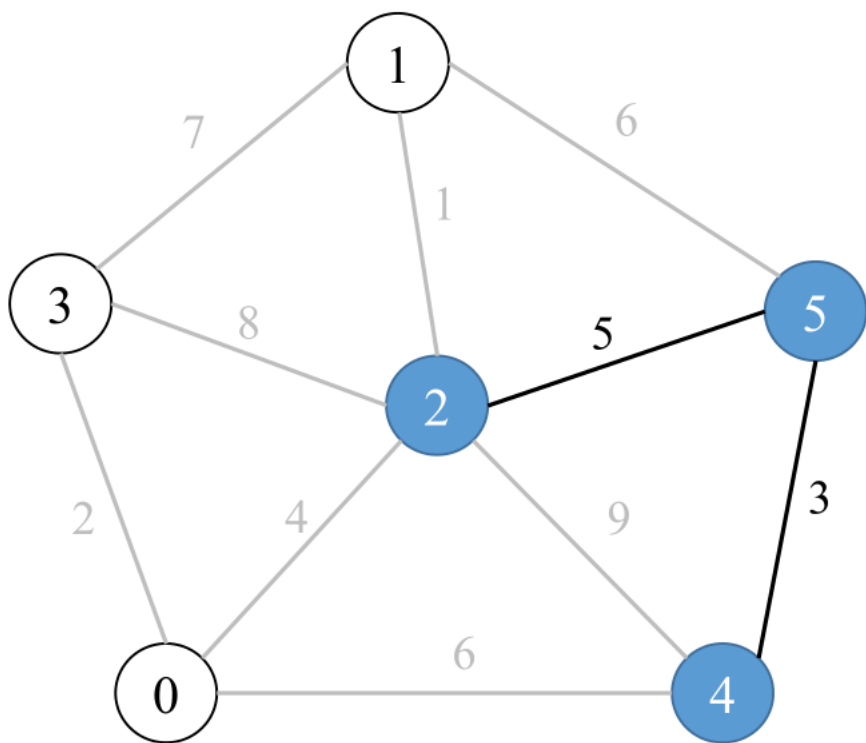


最小生成树——Prim算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法1：从一个点开始，用最小权的边去连其他点。

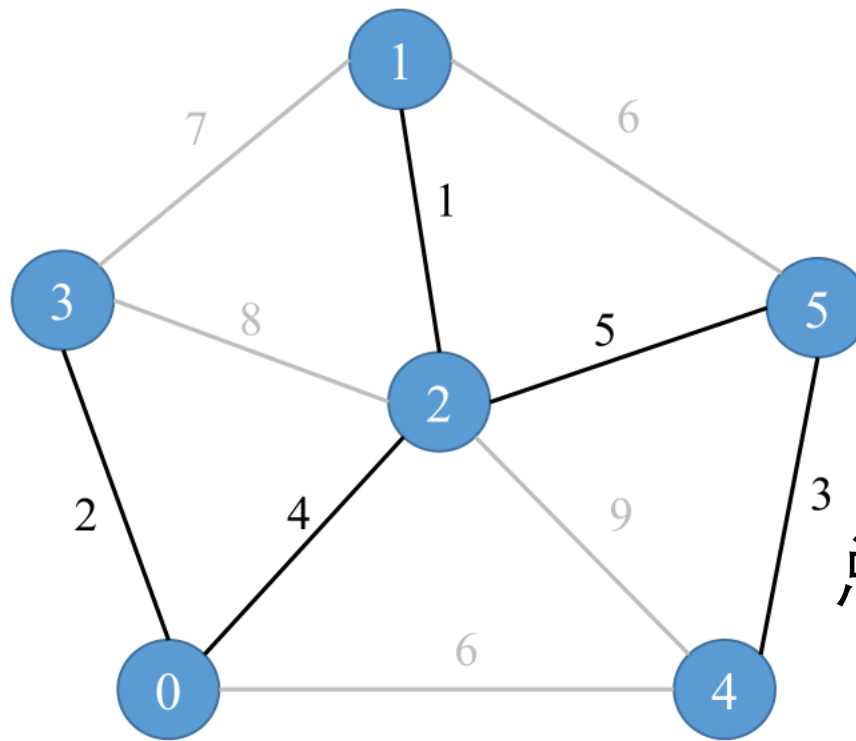
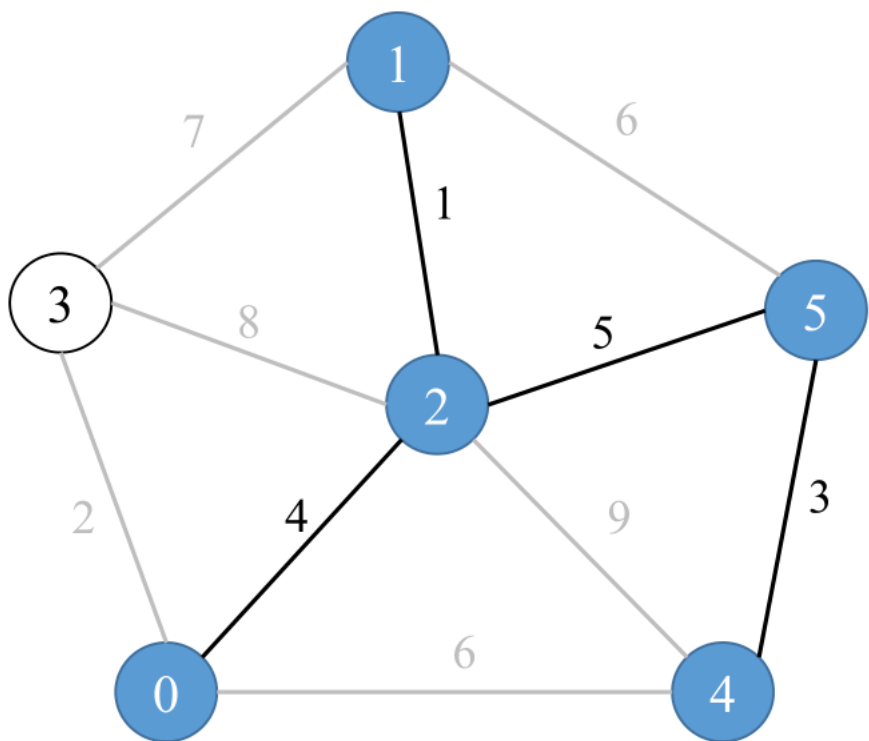


最小生成树——Prim算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法1：从一个点开始，用最小权的边去连其他点。



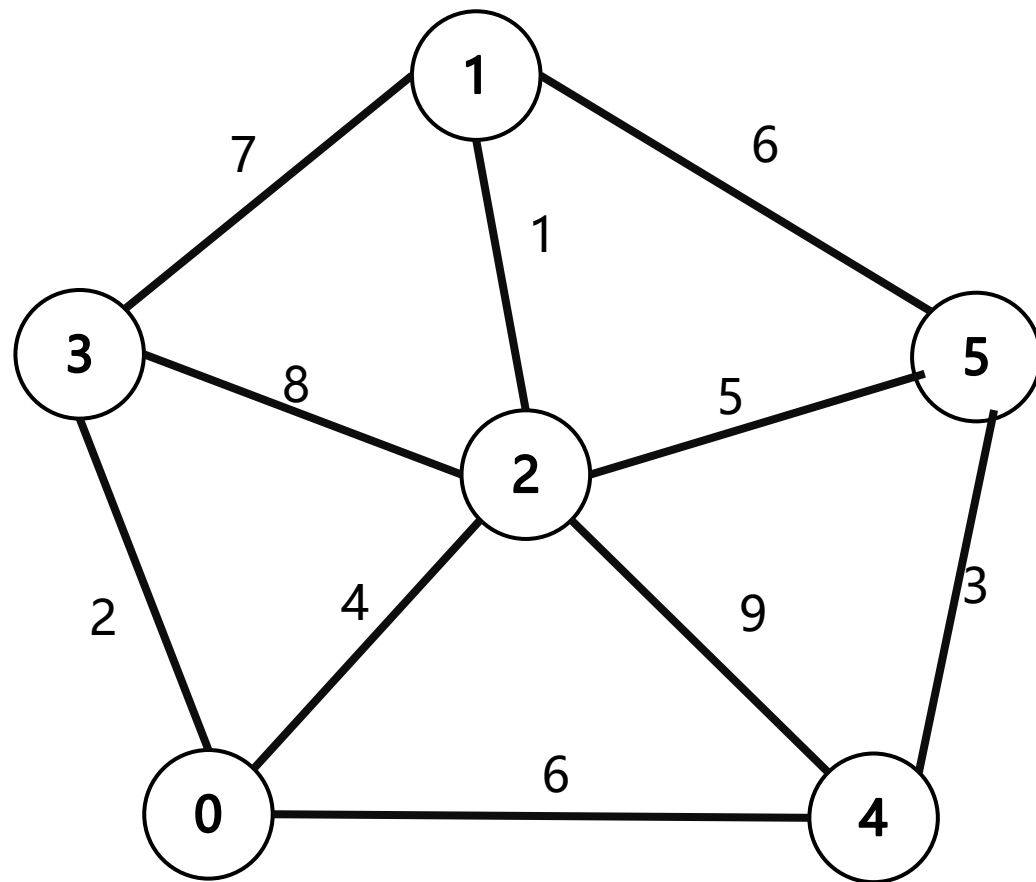
总长度15

最小生成树——Kruskal算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法2：从零开始，不断加最小边直到 $n-1$ 条，只要不成环

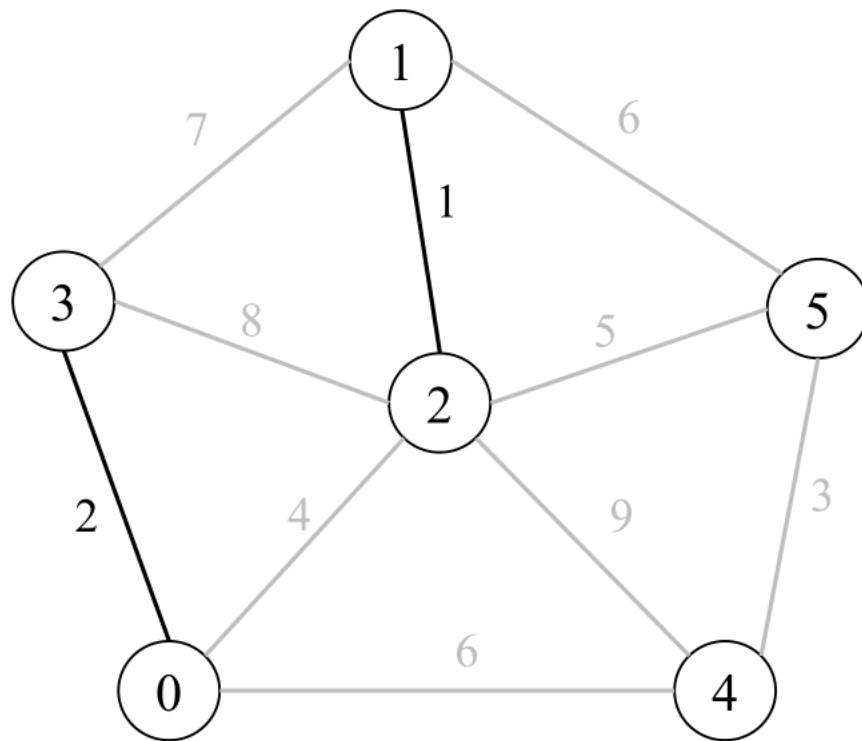
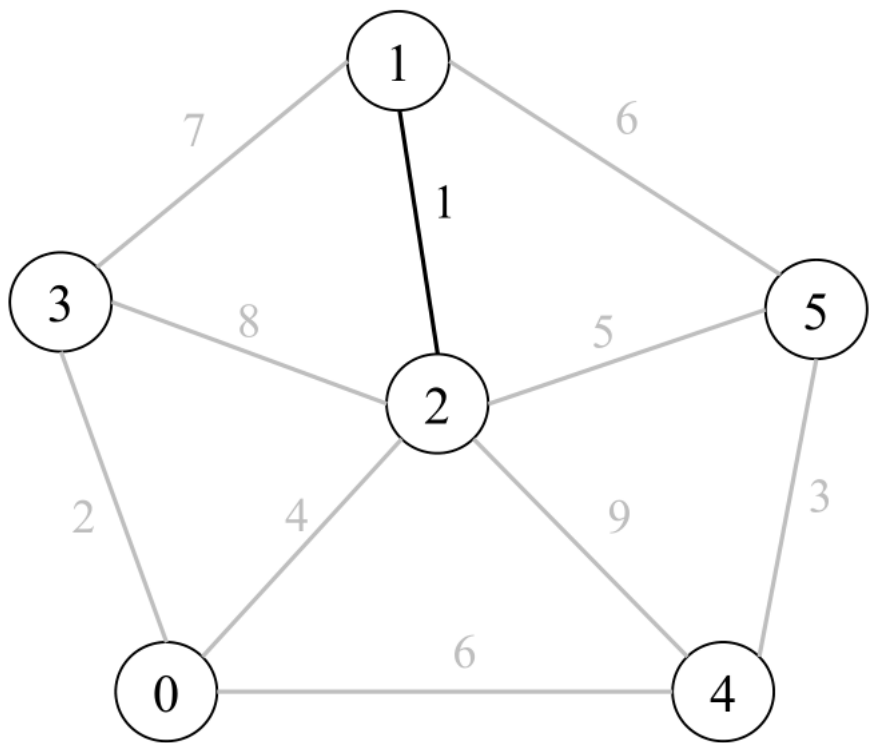


最小生成树——Kruskal算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法2：从零开始，不断加最小边直到 $n-1$ 条，只要不成环

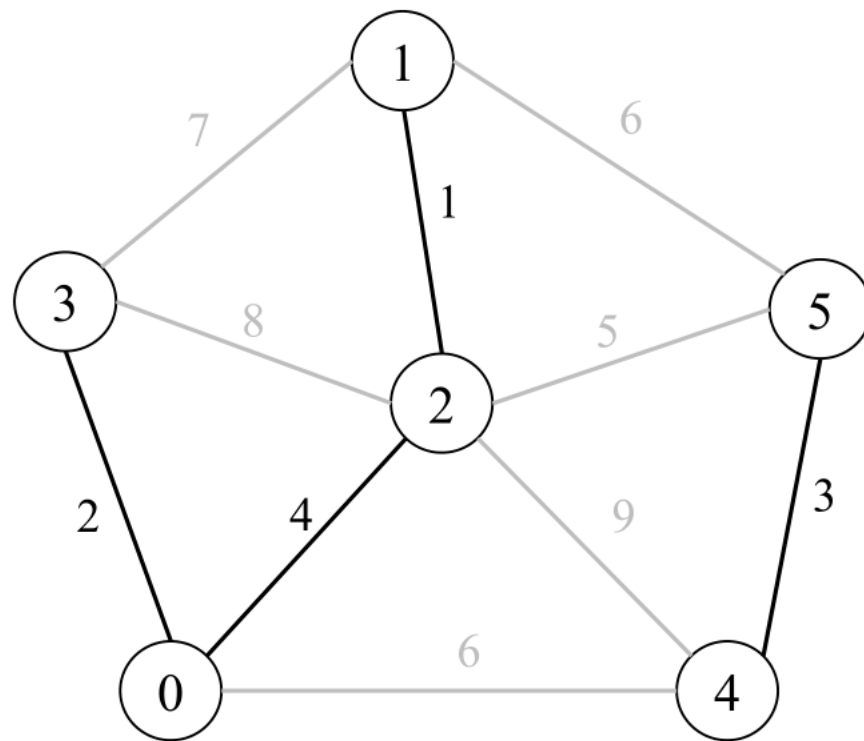
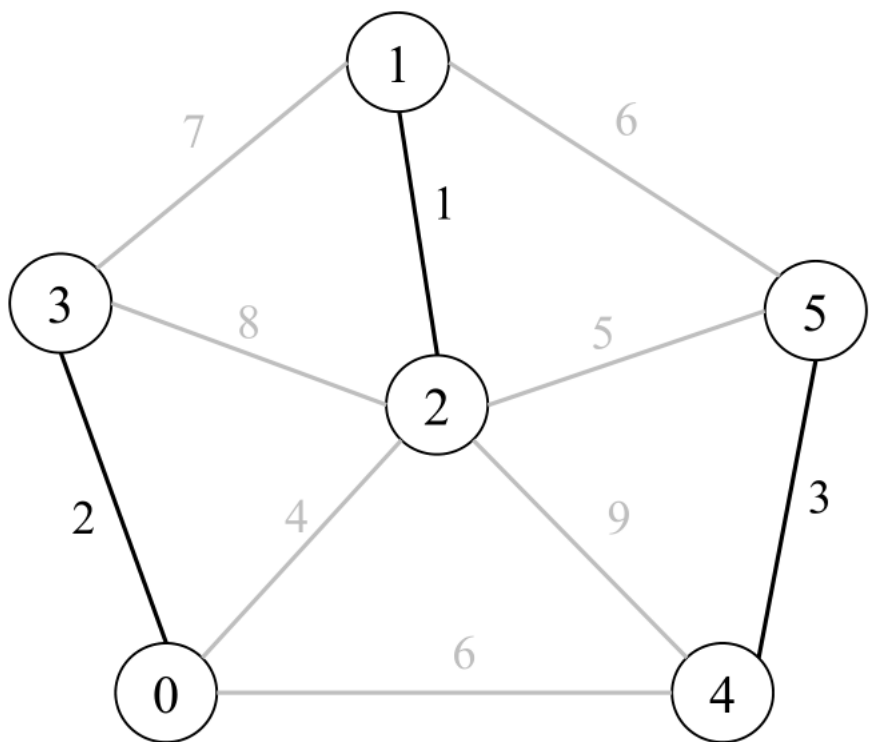


最小生成树——Kruskal算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法2：从零开始，不断加最小边直到 $n-1$ 条，只要不成环

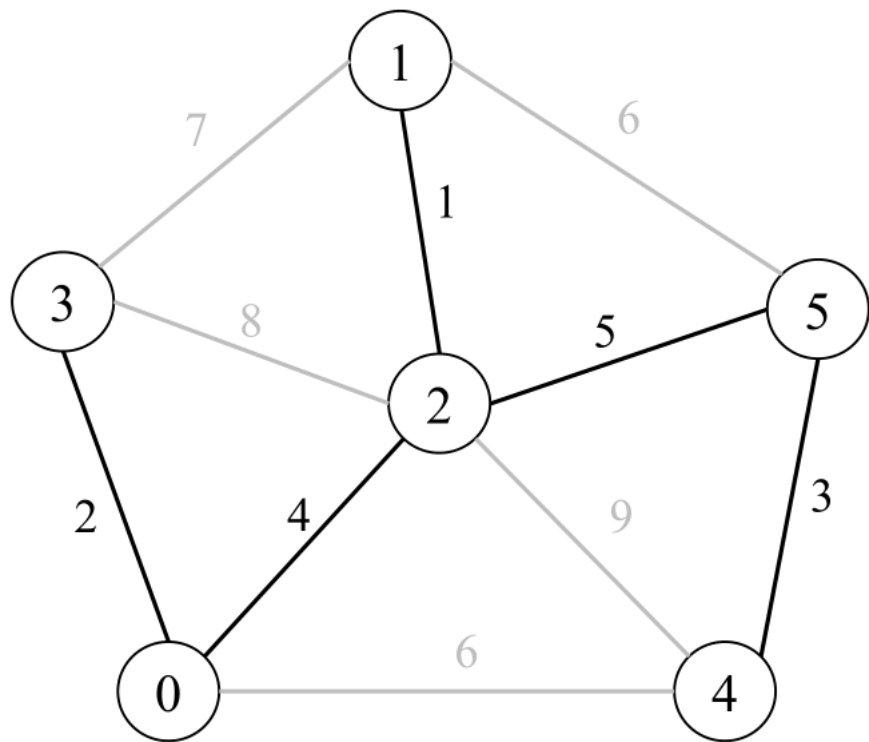


最小生成树——Kruskal算法

图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法2：从零开始，不断加最小边直到 $n-1$ 条，只要不成环



总长度15

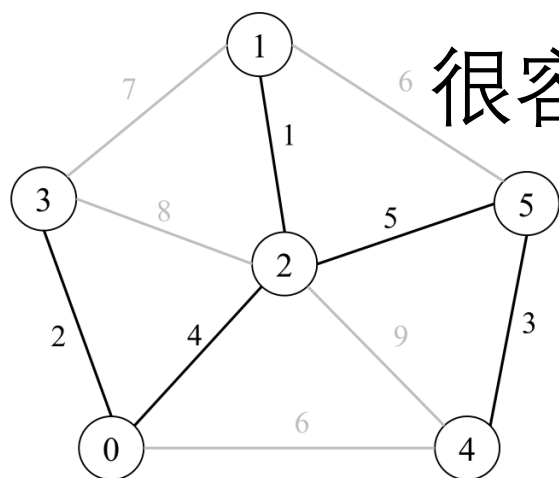
这里最后的生成树和Prim一样，
但这不一定

最小生成树——逆删除算法

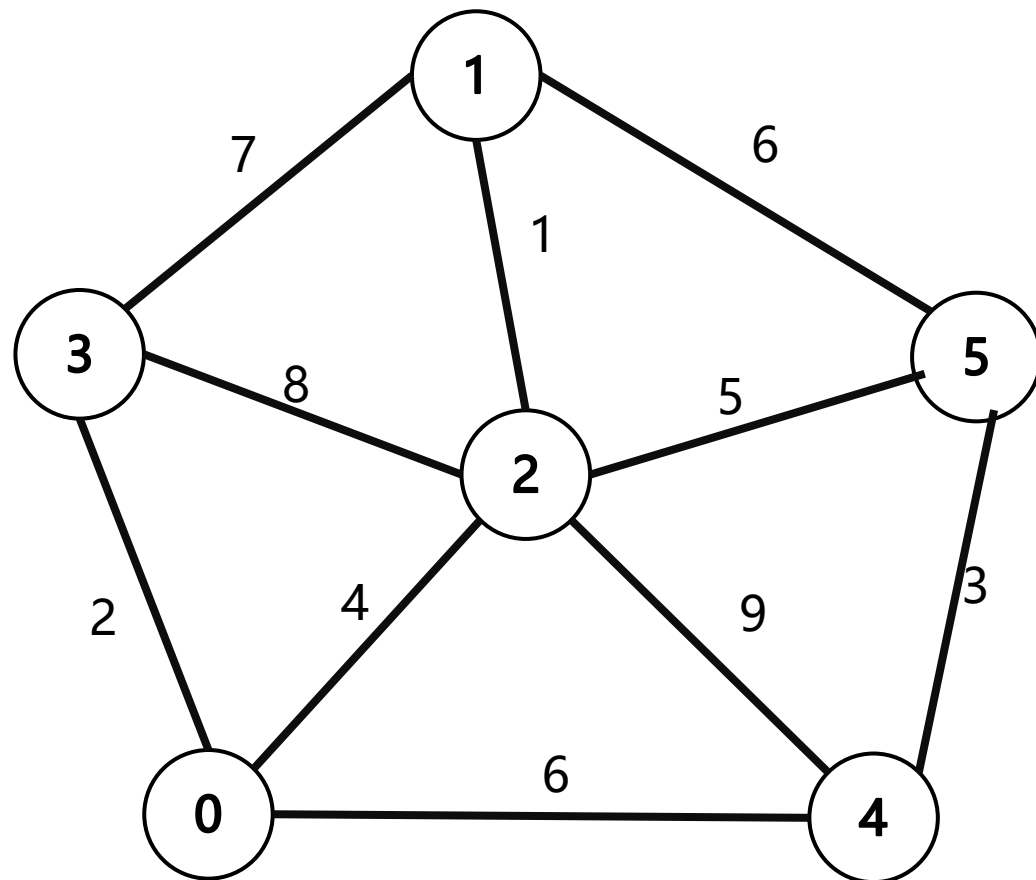
图的生成树：保留图中 n 个点和 $n-1$ 条边子图

最小生成树问题：要求 $n-1$ 条边权值和最小

贪心法3：从原图开始，不断删
最大边直到 $n-1$ 条，只要图连通



很容易得到同样结果



最小生成树应用——TSP的2近似算法

(1) 选择G的一个顶点r作为根节点(出发/结束点)

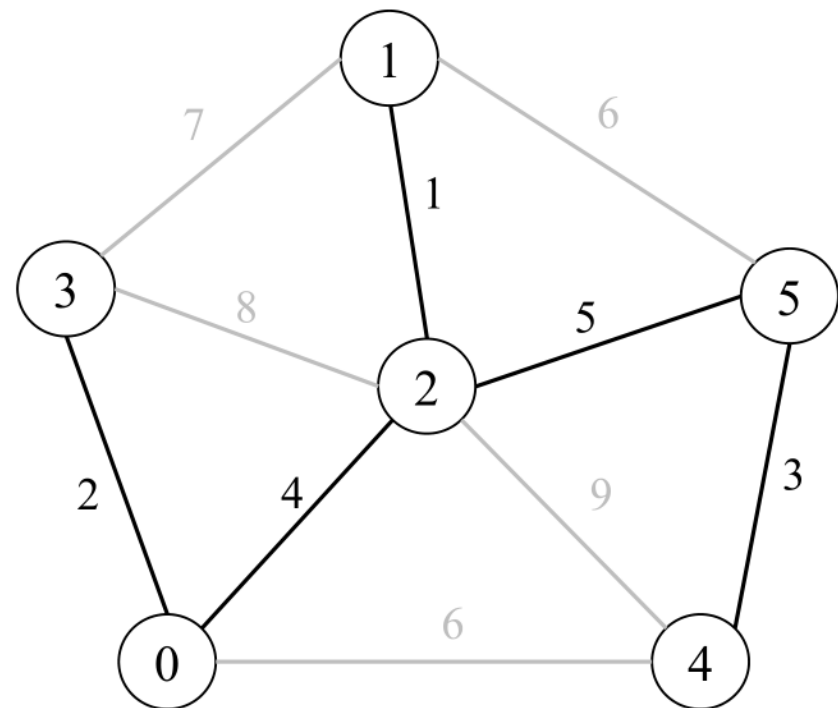
(2) 用Prim算法找一棵以r为根的最小生成树T

(3) 前序遍历T, 得到遍历顺序组成的顶点表L

(4) 将r加到顶点表L的末尾, 按L中顶点的次序组成哈密顿回路H

距离满足三角不等式时有近似比2

前序遍历: 2,1,5,4,0,3
结果2->1->5->4->0->3



斯坦纳树问题

费马点问题：平原上的三个城间要建一个公用供应站，选址要求使供应站到三个城镇的输送管道的总长最短。如何去寻找合适地点？

假如是加权和呢？

历史故事：1967年前，贝尔公司按照链接各分部的最小生成树的长度来收费。

一家航空公司申请要求增加服务点，在Steiner顶点上。这使得贝尔公司不仅要拉新线，增加服务网点，而且还要减少收费。

斯坦纳树求解

这是NPC问题

普遍的求解思想： 动态规划

具体看求解器

Steiner Tree介绍网址 <http://www.cs.sunysb.edu/~algorithm/files/steiner-tree.shtml>

Geosteiner网址 <http://ganley.org/steiner/> FLUTE网

址 <http://home.eng.iastate.edu/~cnchu/flute.html> Goblin网

址 <http://www.math.uni-augsburg.de/~fremuth/goblin.html> PHYLIP网

址 <http://evolution.genetics.washington.edu/phylip.html> Salwoe网

址 <http://www.cs.sunysb.edu/~algorithm/implement/salowe/distrib/>

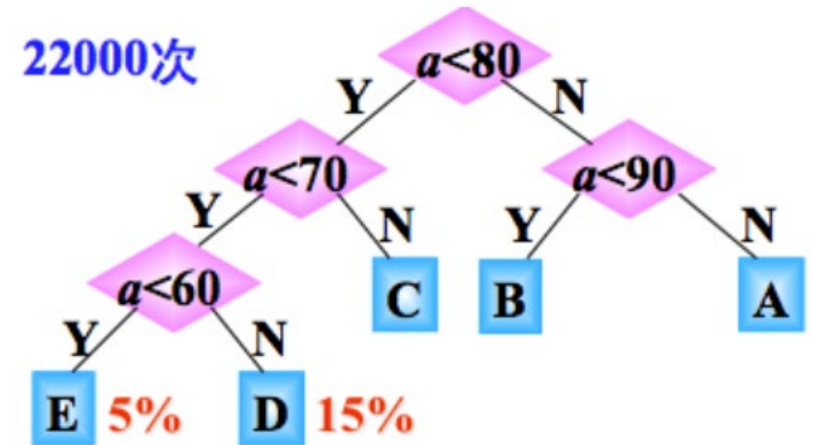
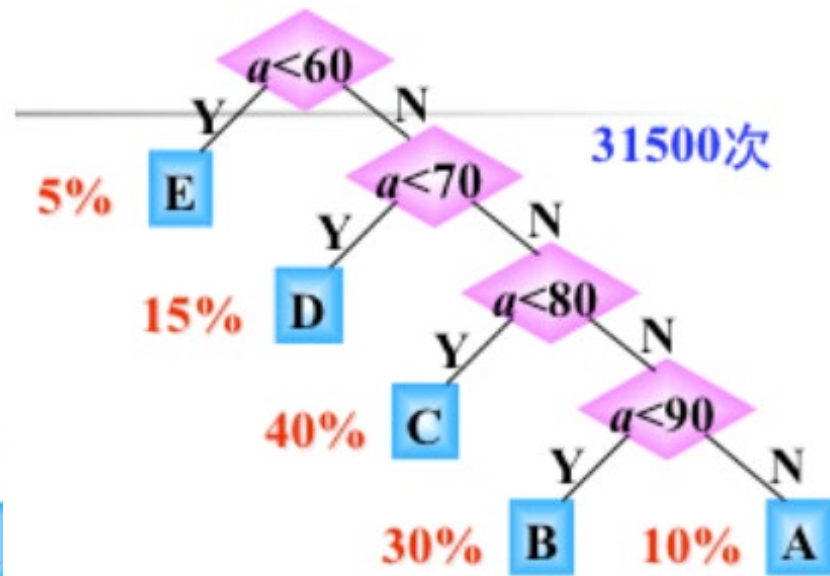
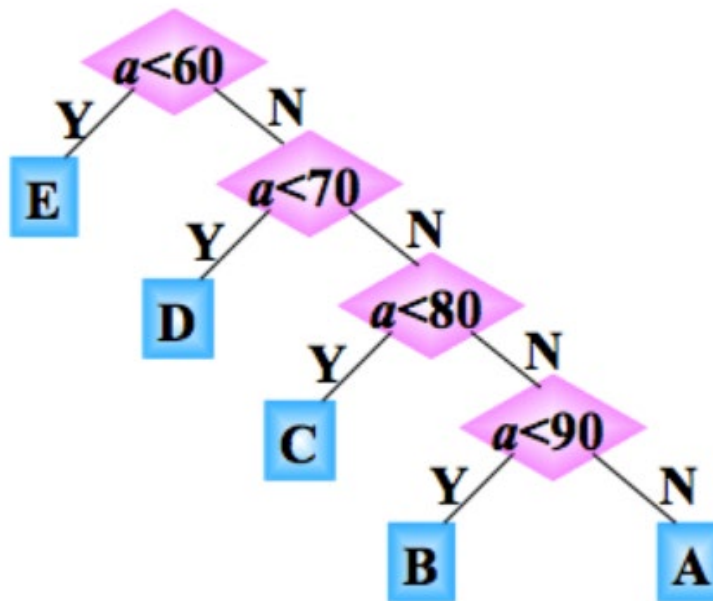
哈夫曼树

哈夫曼树 (Huffman Tree)，又称最优二叉树，是一类带权路径长度最短的树。

例子：成绩评级程序

若10000条成绩

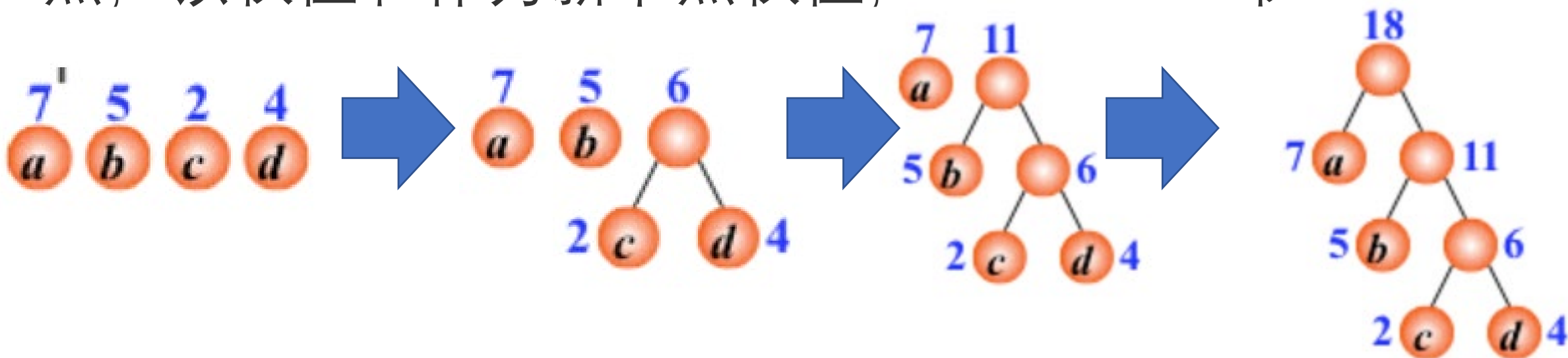
优化一下



哈夫曼树构造算法

哈夫曼树定义： $WPL = \sum_{i=1}^n w_i l_i$ 最小的树。 w_i 为节点权重， l_i 为节点深度

构造：不断把权值最小的两个节点组成成新的根节点，以权值和作为新节点权值，直到构成一棵树



哈夫曼树应用

哈夫曼树定义： $WPL = \sum_{i=1}^n w_i l_i$
树。 w_i 为节点权重， l_i 为节点深度

最小的

1. 程序逻辑优化
2. 最优信源编码（香农定理）



P3



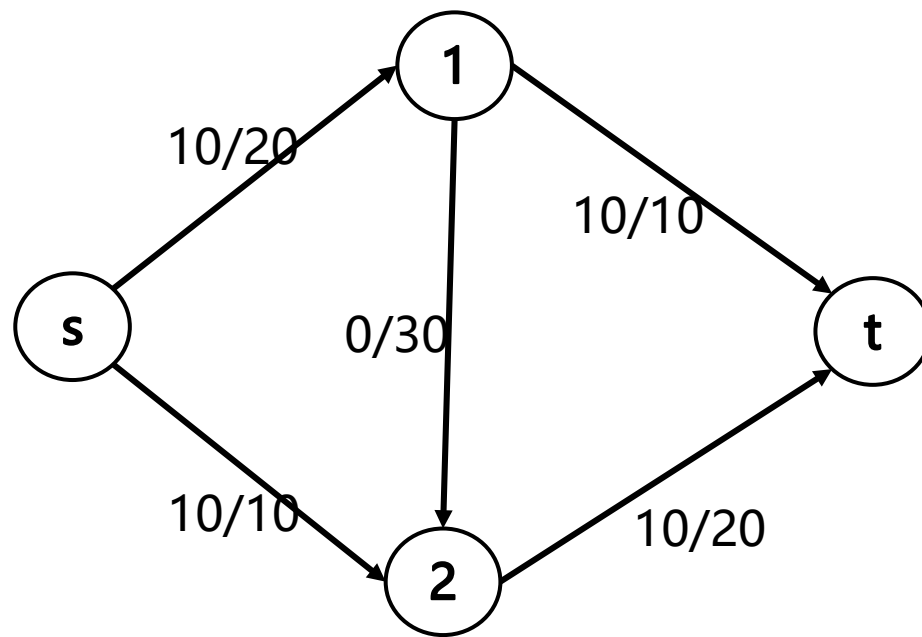
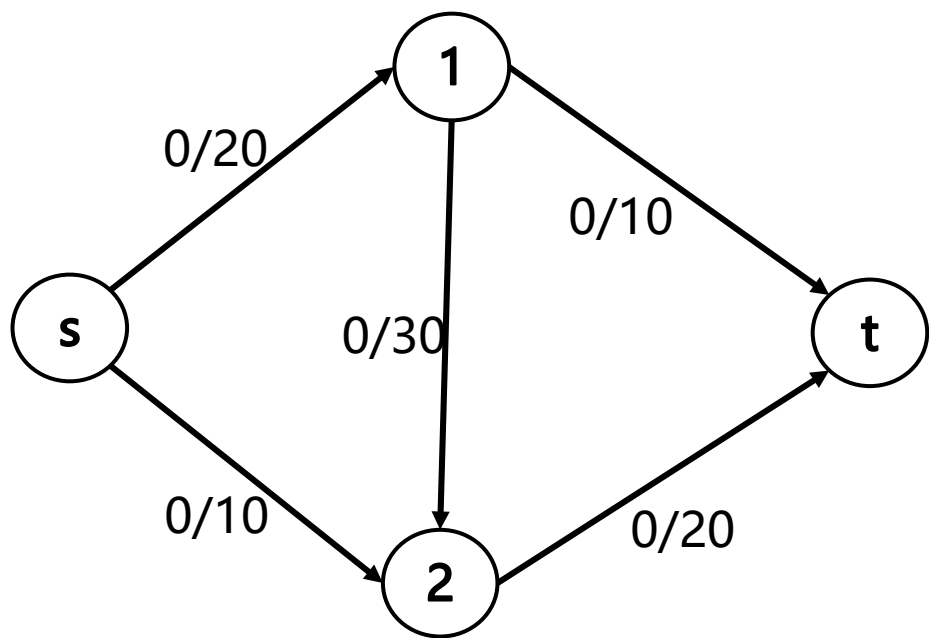
图算法3：流



最大流问题

问题定义：给定一个有向图，边有容量，求从s到t的最大流量。

实际含义：交通运输，人群移动等



一个例子，总流量20。
是最大吗？

最大流最小割问题

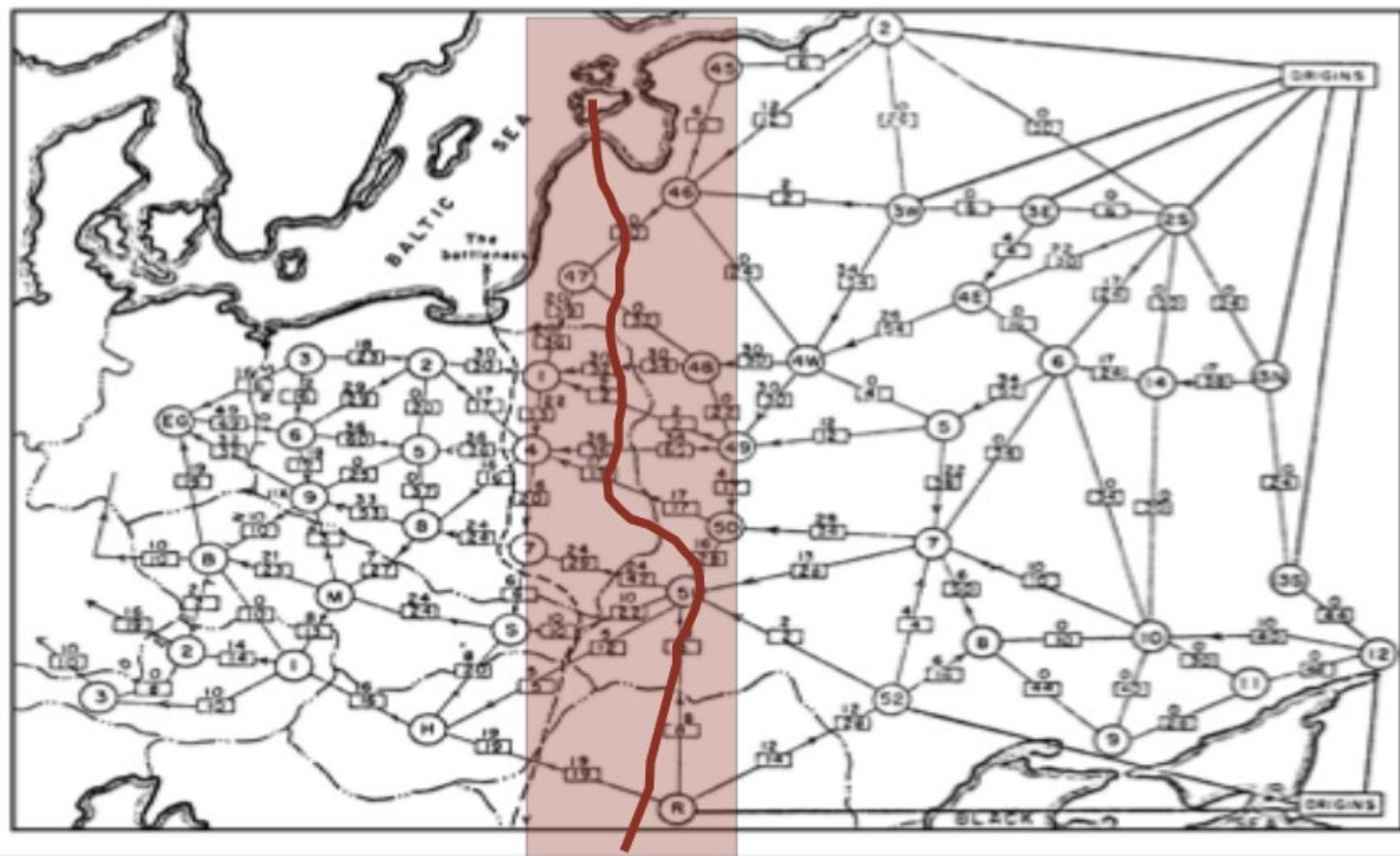


Figure 2

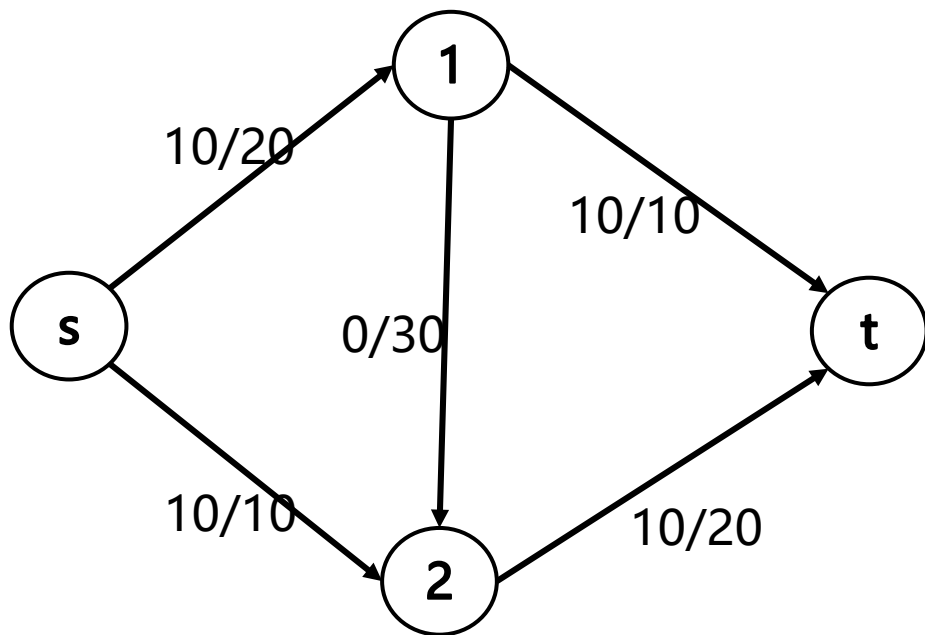
From Harris and Ross [1955]: Schematic diagram of the railway network of the Western Soviet Union and Eastern European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as 'The bottleneck'.

50年代的苏联铁路网，东欧部分最大流163,000吨，红线标出的是瓶颈，也就是最小割。

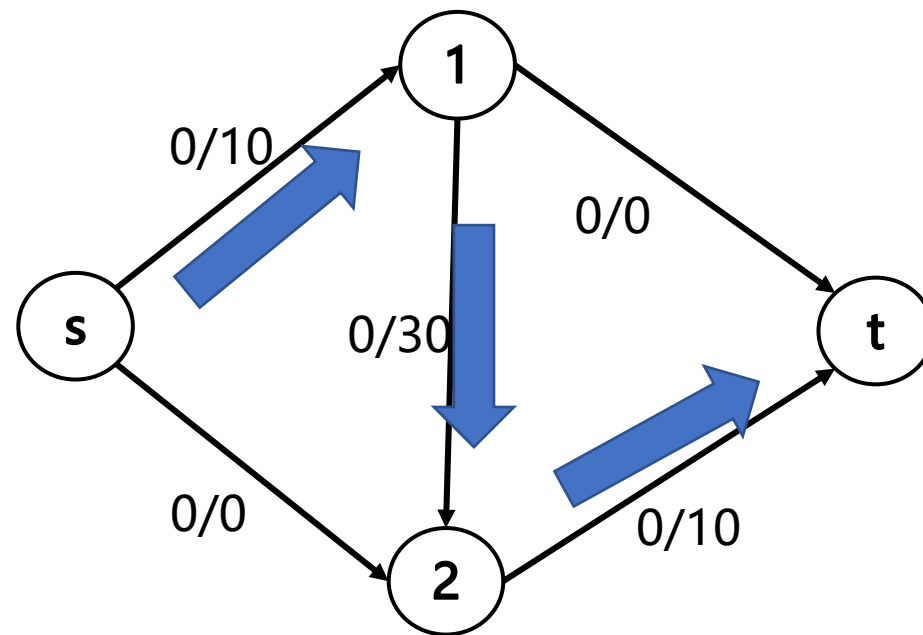
如果冷战变成热战，就切这里。

最大流问题

问题：给定一个有向图，边有容量，求从s到t的最大流量。



一个例子，总流量20。
是最大吗？



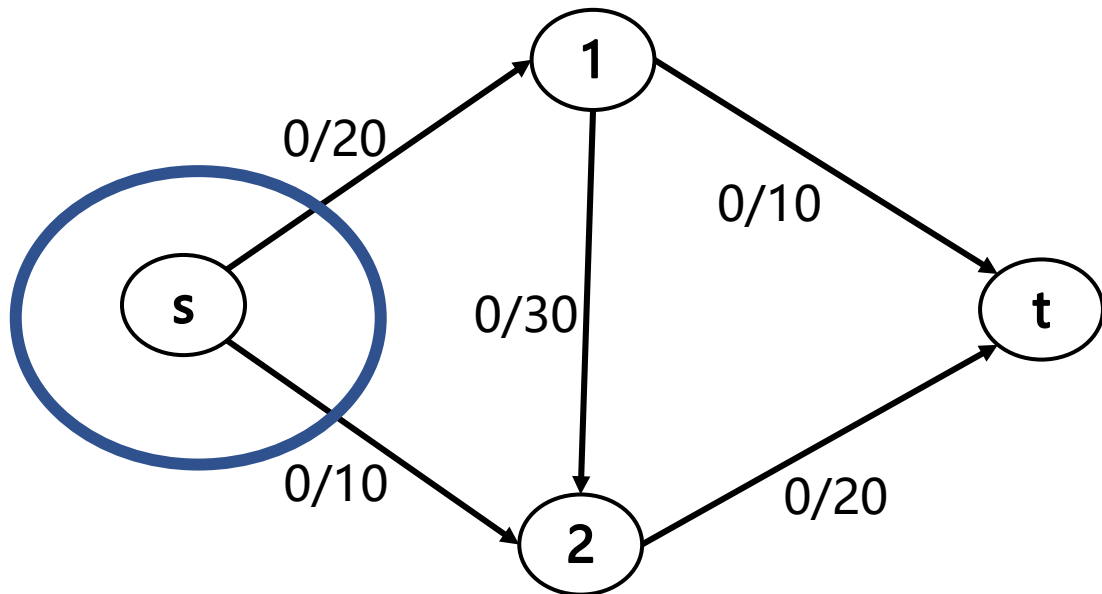
残余的容量网络里，还可以
加10的流量，到达30。
30一定是最大流，s总输入

最大流=最小割 原理

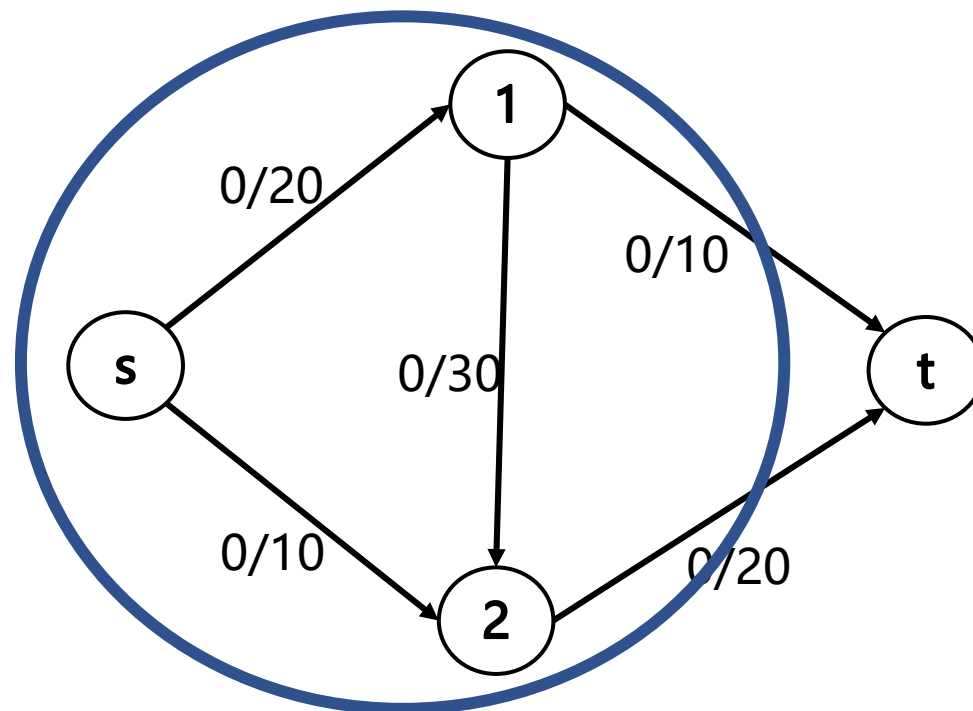
最大流不超过最小割容量：瓶颈

割：将点分为各自连通的两部分，一部分包括s，另一部分包括t，点集称为割，边集称为割集。

割的容量：割集的输出总量



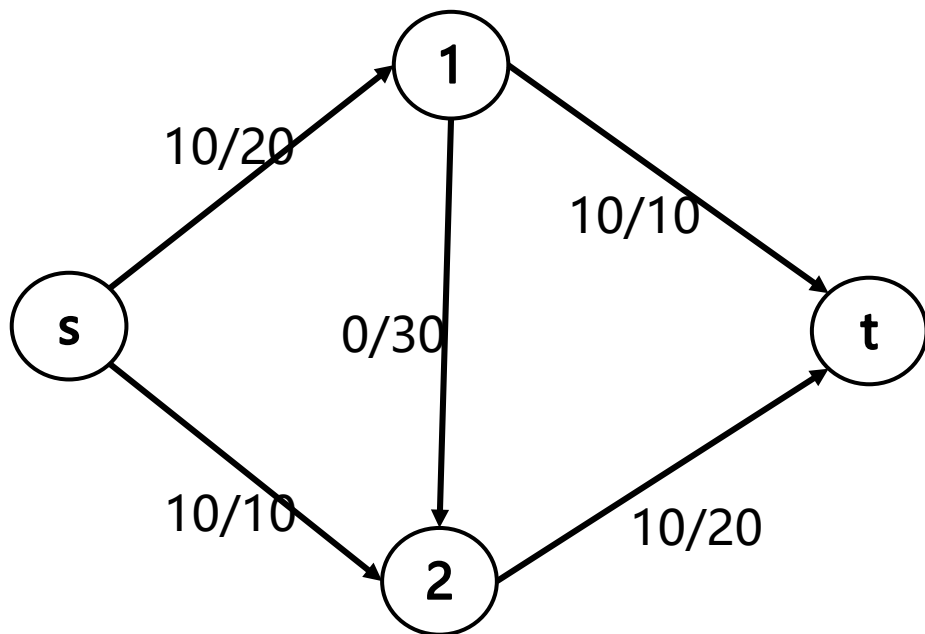
割 $(\{s\}, \{1, 2, t\})$ 的容量为30



割 $(\{s, 1, 2\}, \{t\})$ 的容量为30

最大流求解：增广

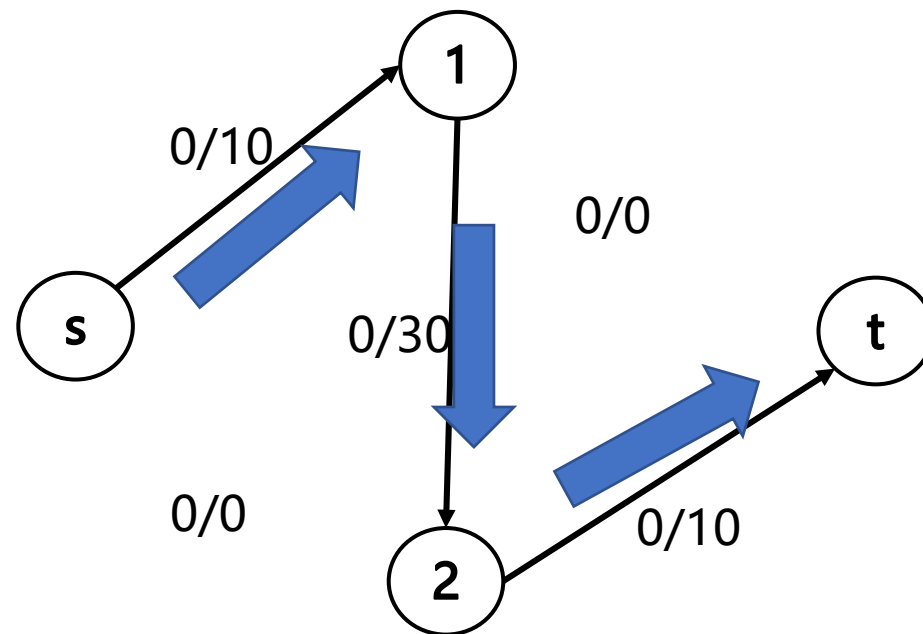
问题：给定一个有向图，边有容量，求从s到t的最大流量。



一个例子，总流量20。

是最大吗？

答：可以**增广**，不是最大

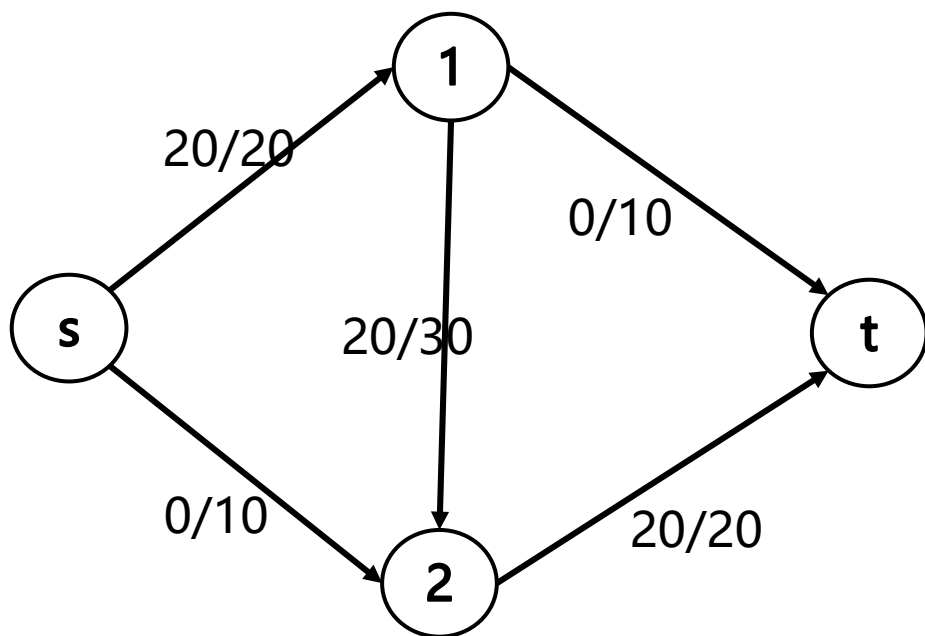


残余的容量网络里，还可以加10的流量，到达30。

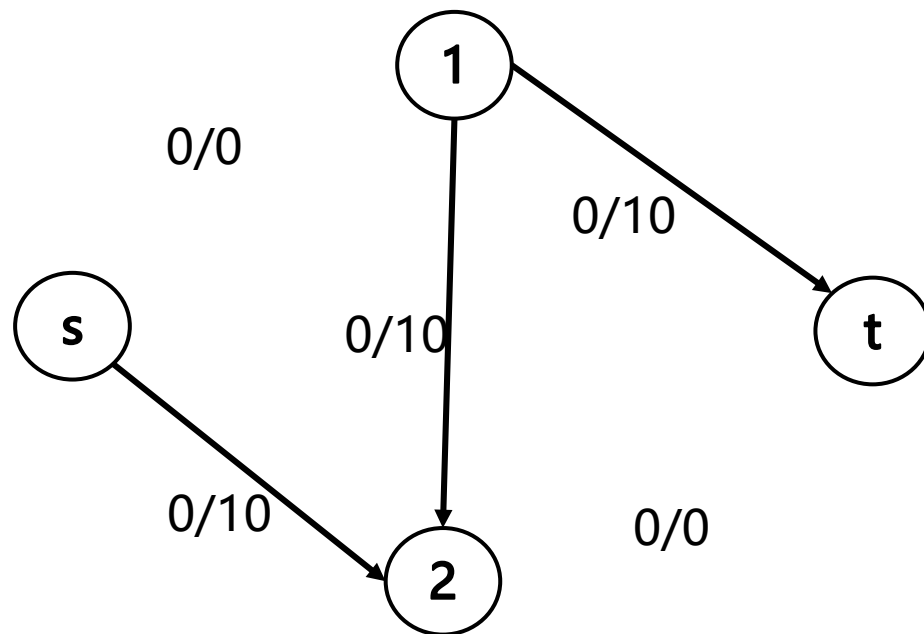
30一定是最大流，s总输入

最大流求解：增广

问题：给定一个有向图，边有容量，求从s到t的最大流量。



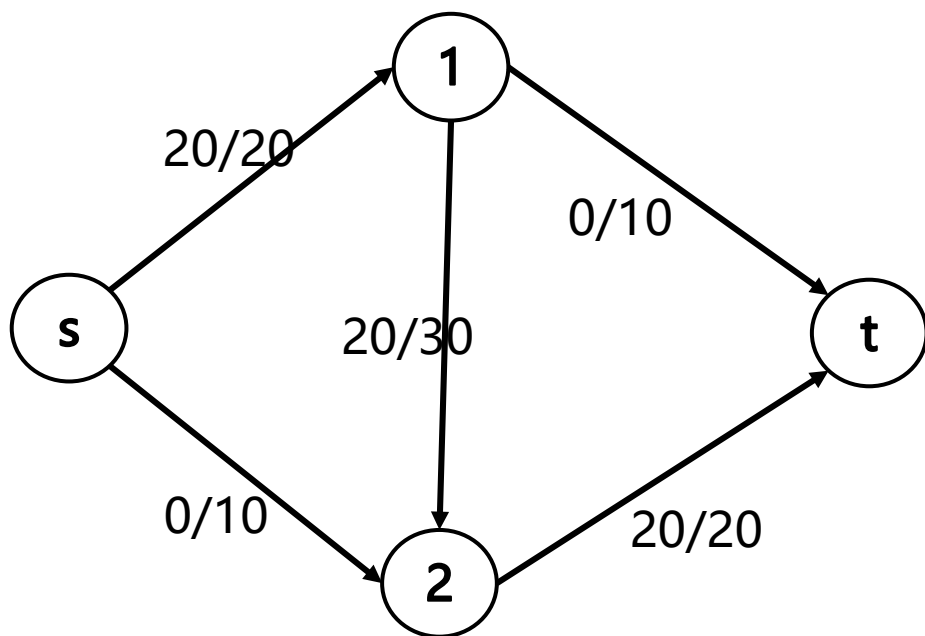
另一个例子，总流量20。
是最大吗？
答：可以增广？



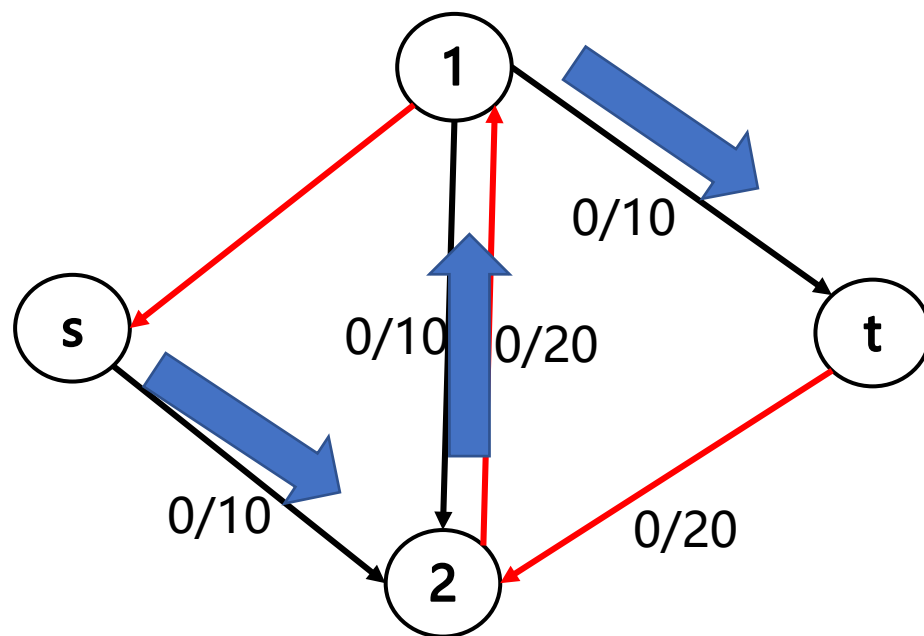
似乎不能增广了，
怎么回事？

最大流求解：增广

问题：给定一个有向图，边有容量，求从s到t的最大流量。



另一个例子，总流量20。
是最大吗？
答：可以增广？



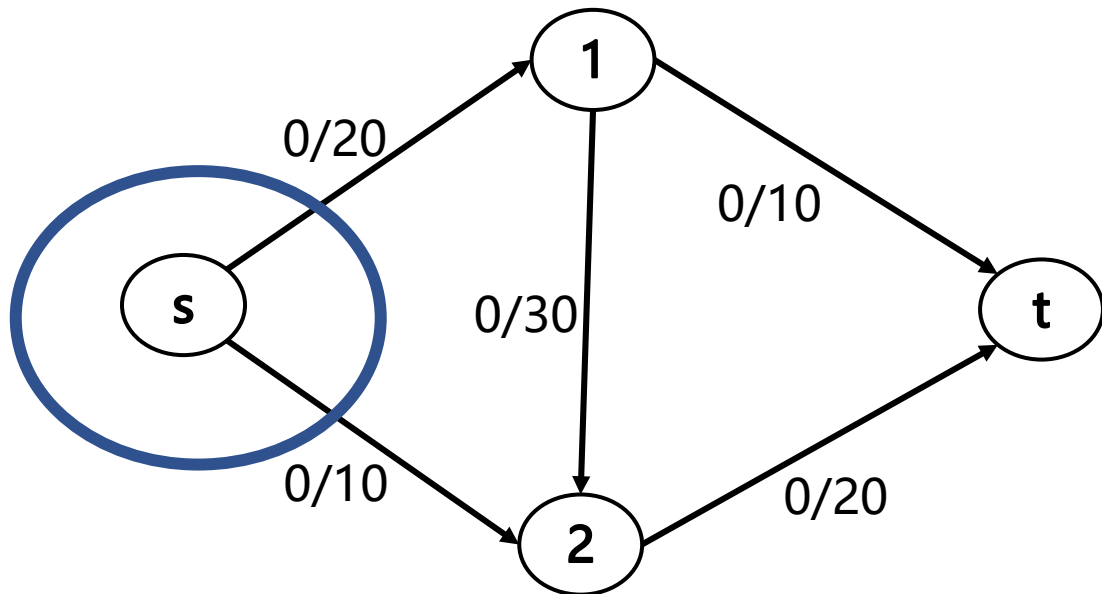
真正的残量网络，有反向边
有增广路s->2->1->t
总流量30

最大流=最小割 原理

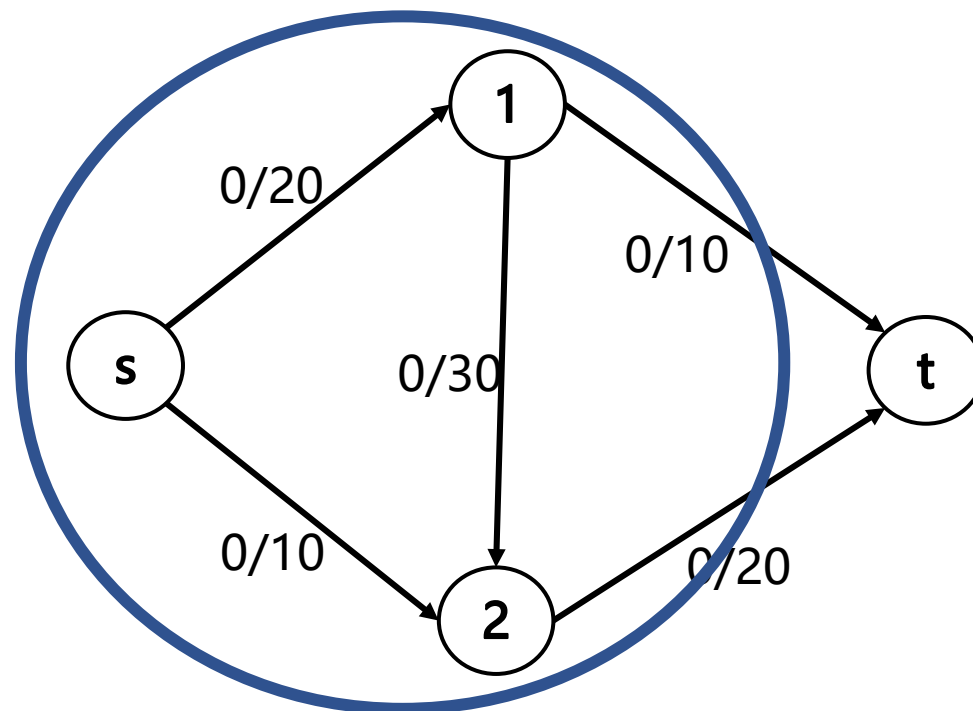
最大流不小于**最小**割容量：否则可增广

割：将点分为各自连通的两部分，一部分包括s，另一部分包括t，点集称为割，边集称为割集。

割的容量：割集的输出总量



割 $(\{s\}, \{1, 2, t\})$ 的容量为30



割 $(\{s, 1, 2\}, \{t\})$ 的容量为30

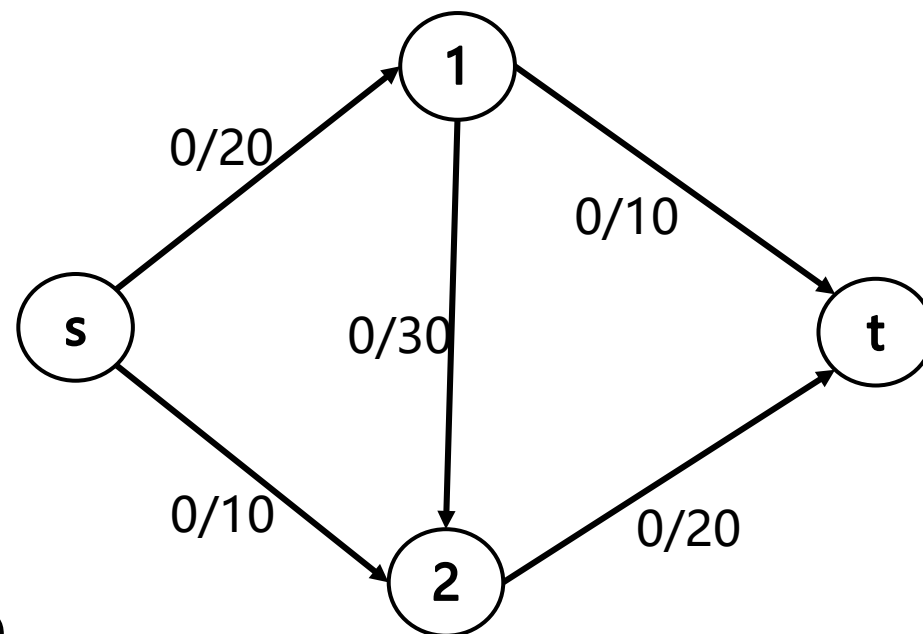
最大流求解：增广路算法

问题：给定一个有向图，边有容量，求从s到t的最大流量。

Ford–Fulkerson算法：

```
While(有增广路){  
    在残量网络中找增广路  
    更新残量网络  
}
```

其BFS实现称为：Edmonds–Karp
算法



最大流求解：增广路算法

问题：给定一个有向图，边有容量，求从s到t的最大流量。

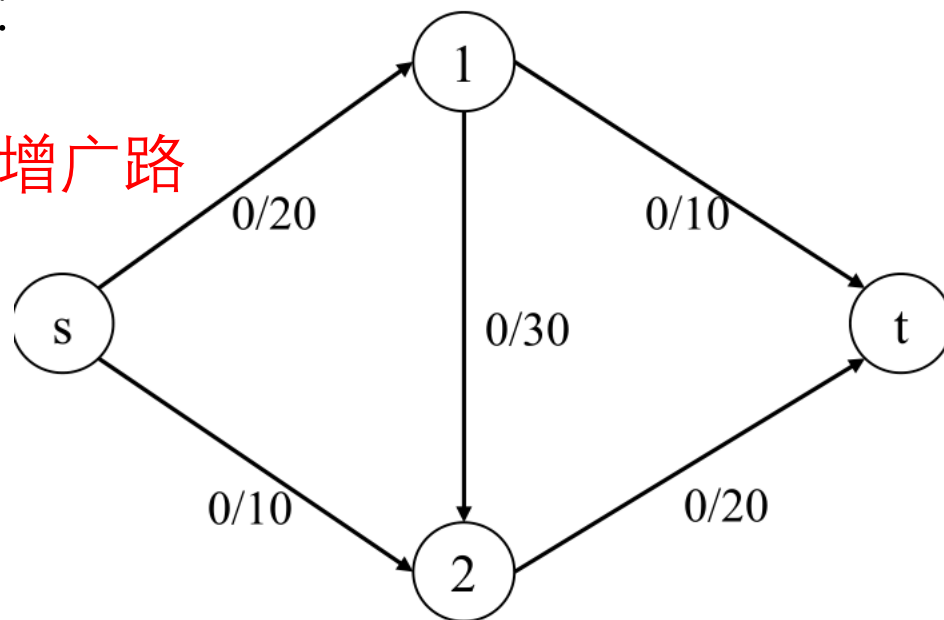
Ford-Fulkerson算法：

While(有增广路){

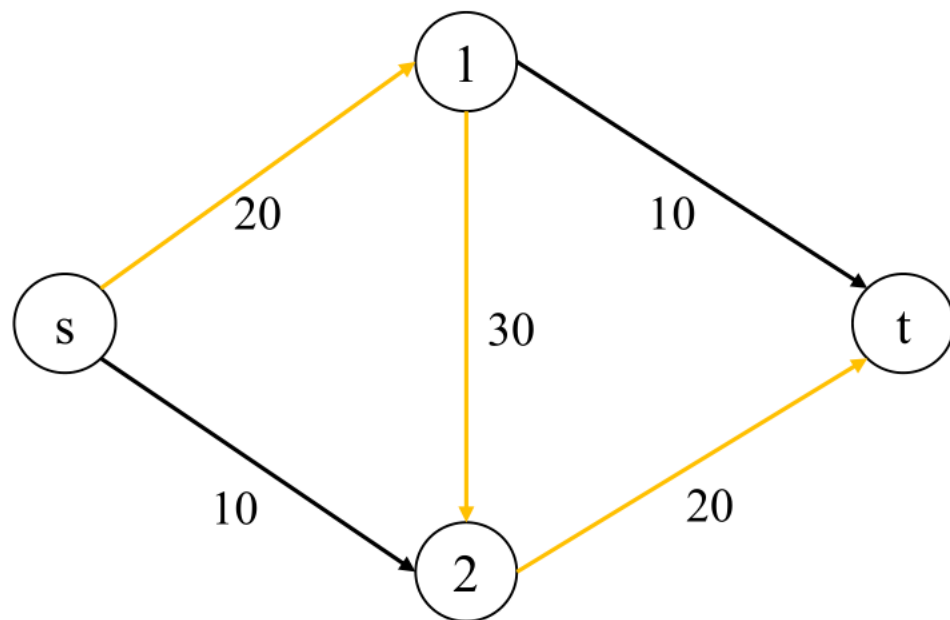
在残量网络中找增广路

更新残量网络

}



$$|f| = 0$$



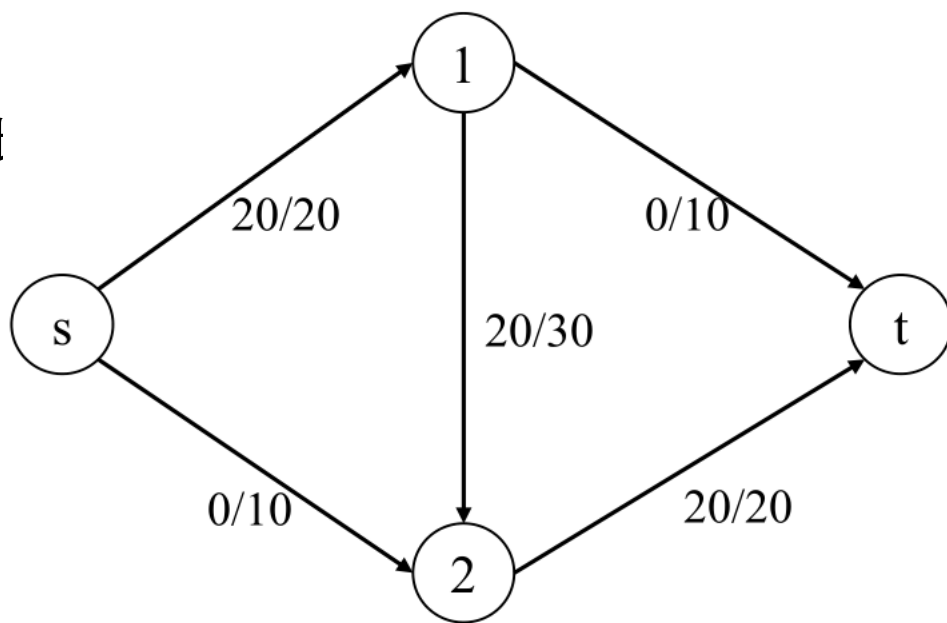
$$G_f$$

$$c_f(u, v) = c(u, v) - f(u, v)$$

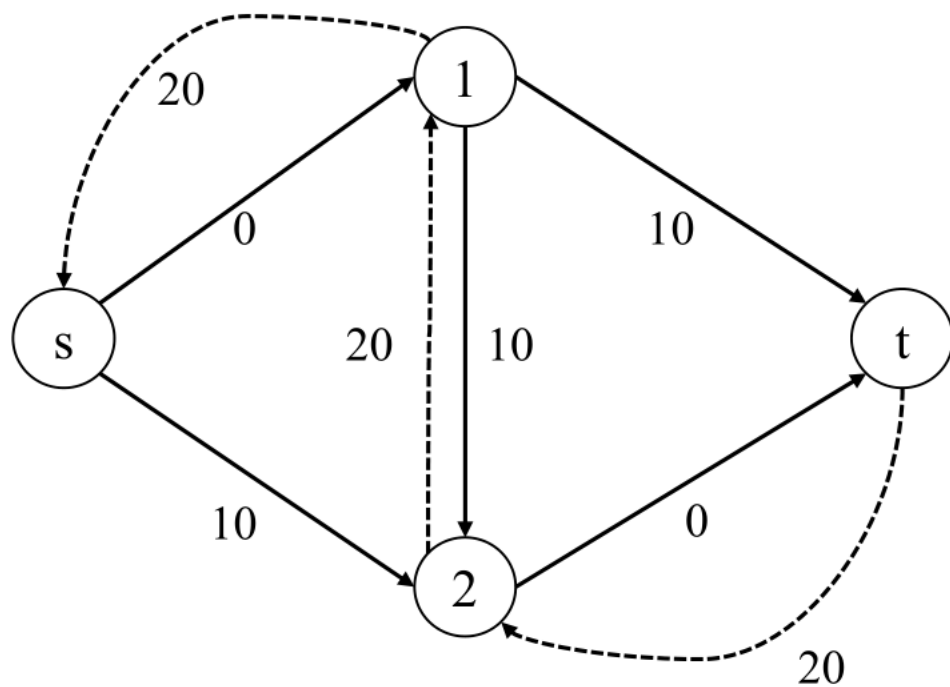
最大流求解：增广路算法

问题：给定一个有向图，边有容量，求从s到t的最大流量。

Ford-Fulkerson算法：
While(有增广路){
 在残量网络中找t
 更新残量网络
}



$$|f| = 20$$



$$G_f$$

$$c_f(u, v) = c(u, v) - f(u, v)$$

最大流求解：增广路算法

问题：给定一个有向图，边有容量，求从s到t的最大流量。

Ford-Fulkerson算法

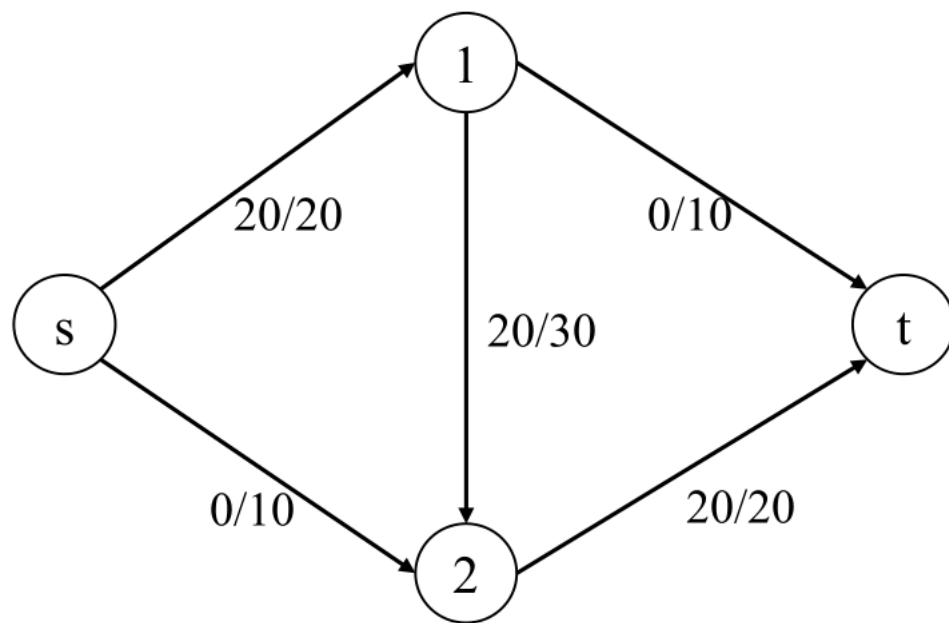
While(有增广路){

在残量网络中

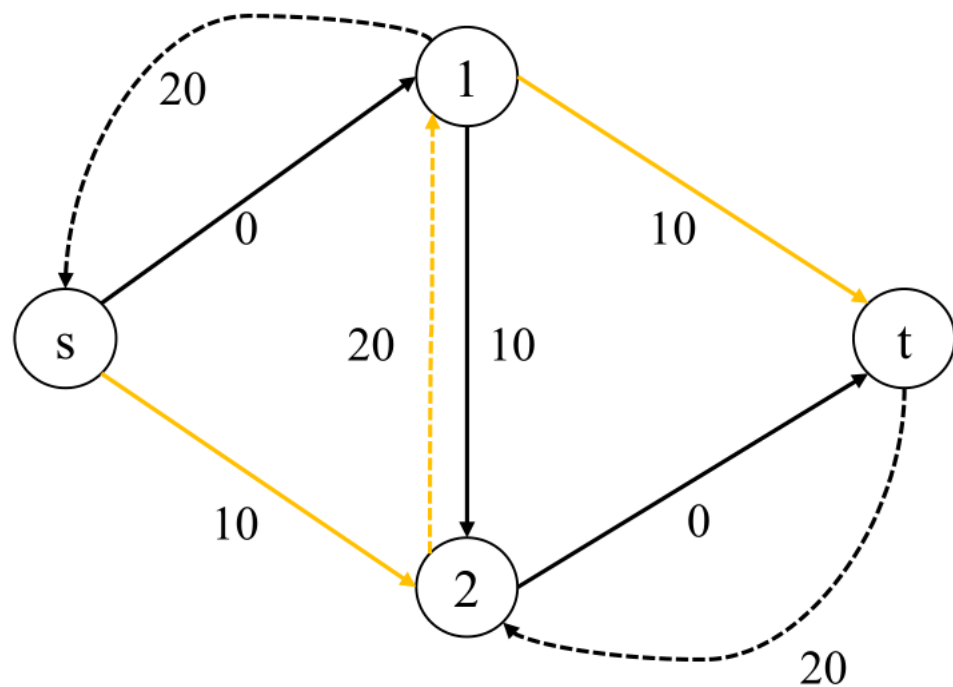
找增广路

更新残量网络

}



$$|f| = 20$$



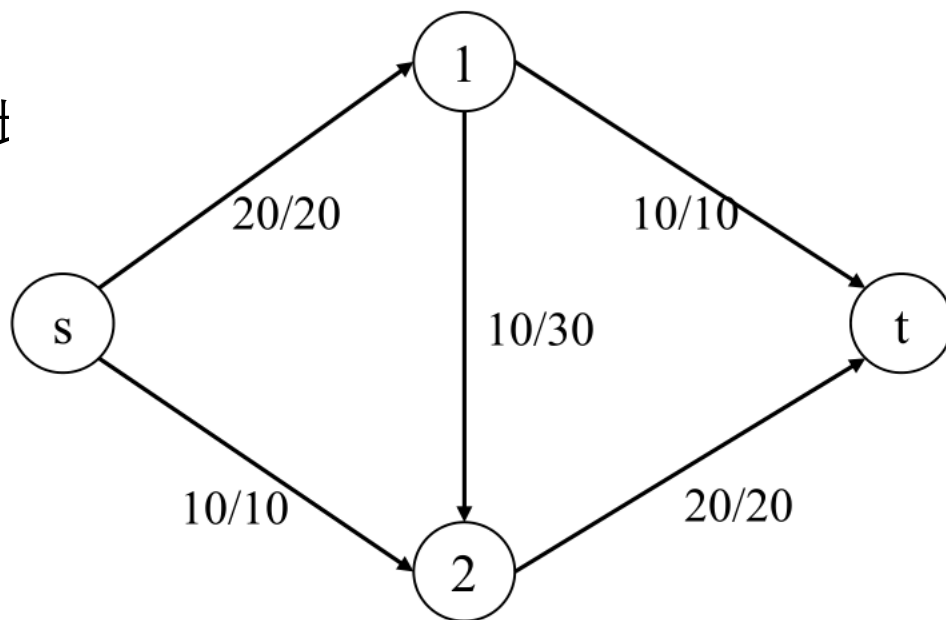
G_f

$$c_f(u, v) = c(u, v) - f(u, v)$$

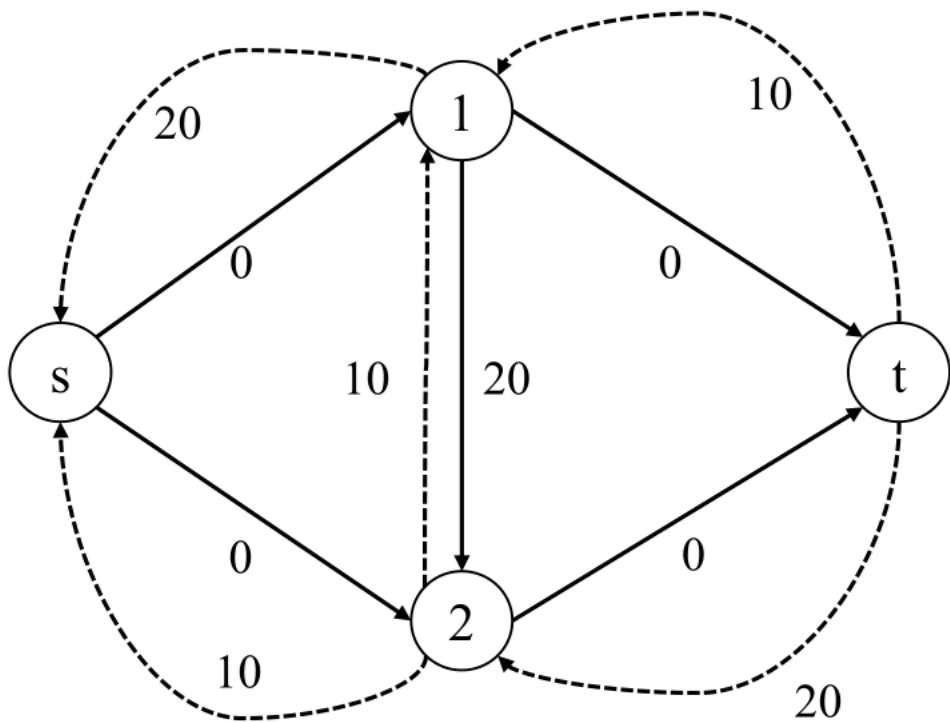
最大流求解：增广路算法

问题：给定一个有向图，边有容量，求从s到t的最大流量。

Ford-Fulkerson算法：
While(有增广路){
 在残量网络中找t
 更新残量网络
}



$|f| = 30$



G_f

$$c_f(u, v) = c(u, v) - f(u, v)$$

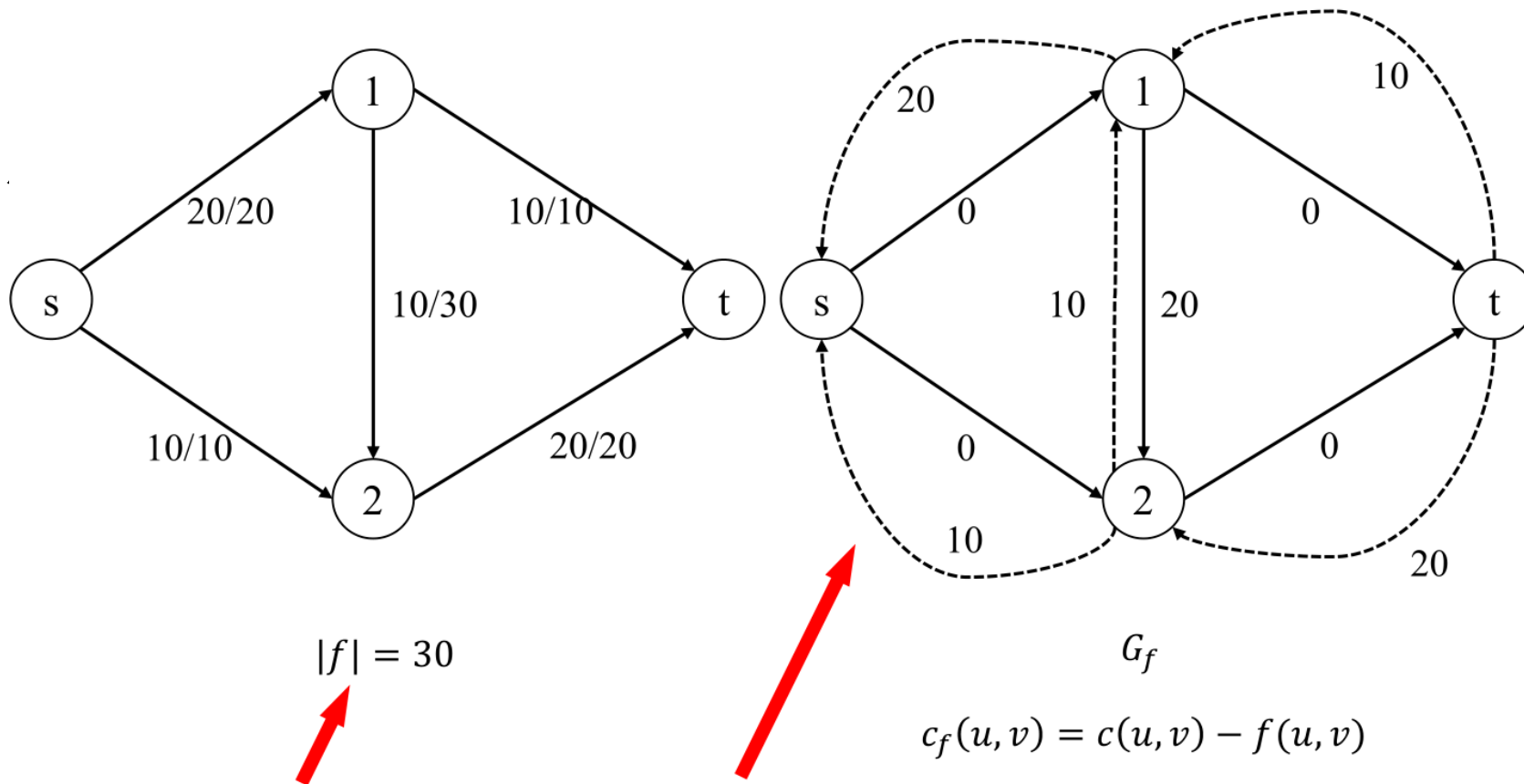
最大流求解：增广路算法

问题：给定一个有向图，边有容量，求从s到t的最大流量。

Ford-Fulkerson算法：

```
While(有增广路){  
    在残量网络中找增  
    更新残量网络  
}
```

返回最大流值30
具体路径也有记录



Maximum Flow Value

No Augmenting Path

最大流问题的研究

问题：给定一个有向图，边有容量，求从s到t的最大流量。
关键点：增广路如何找

Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS

University of Waterloo, Waterloo, Ontario, Canada

AND

RICHARD M. KARP

University of California, Berkeley, California

ABSTRACT. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

Edmonds-Karp 1972 (USA)

Dokl. Akad. Nauk SSSR
Tom 194 (1970), No. 4

Soviet Math. Dokl.
Vol. 11 (1970), No. 5

ALGORITHM FOR SOLUTION OF A PROBLEM OF MAXIMUM FLOW IN A NETWORK WITH POWER ESTIMATION

UDC 518.5

E. A. DINIC

Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this algorithm requires preliminary rounding off of the initial data, i.e. only an approximate solution of the problem is possible. In this connection the rapidity of convergence of the algorithm is inversely proportional to the relative precision.

Dinic 1970 (Soviet Union)

最大流问题的研究

关键点：增广路如何找

增广复杂度

总复杂度

	method	# augmentations	running time
逐一增流	augmenting path	$n C$	$O(m n C)$
	fattest augmenting path	$m \log (mC)$	$O(m^2 \log n \log (mC))$
对流量二分	capacity scaling	$m \log C$	$O(m^2 \log C)$
	improved capacity scaling	$m \log C$	$O(m n \log C)$
EK, 用BFS	shortest augmenting path	$m n$	$O(m^2 n)$
	improved shortest augmenting path	$m n$	$O(m n^2)$
	dynamic trees	$m n$	$O(m n \log n)$

最大流应用1：最小费用最大流

问题：给定一个有向图，边有容量，有单位运费，求从s到t的最大流量，并选择运费最小的路径。

Ford-Fulkerson算法：

While(有增广路){

 在残量网络中对**运费**用**最短路**

 算法找增广路

 更新残量网络

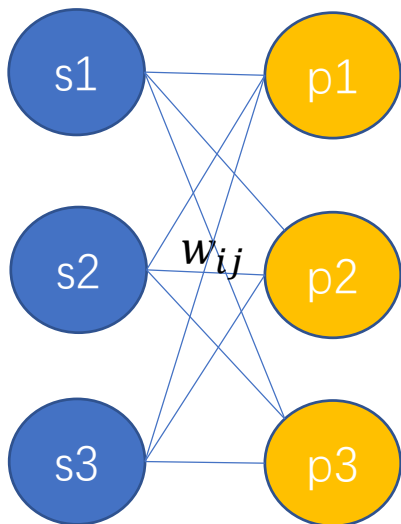
}

一般用Bellman-Ford算法找最短路

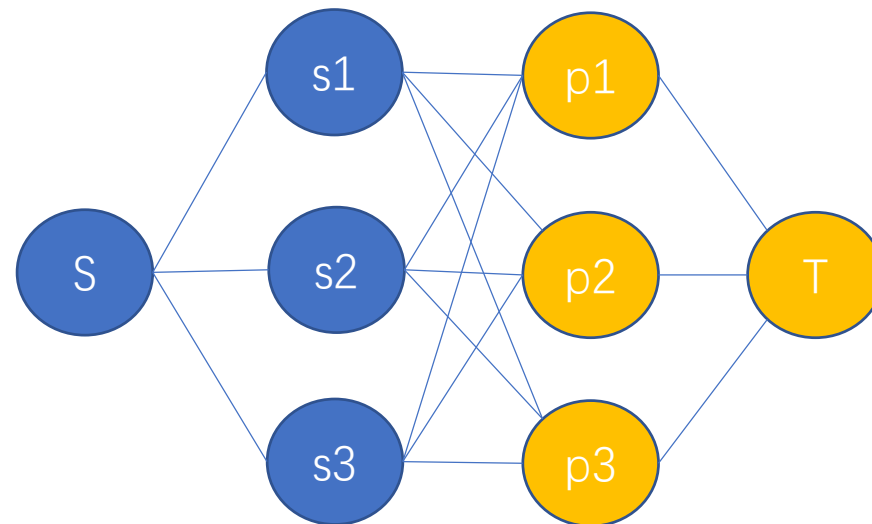
最大流应用2：最大匹配问题

问题：给定一个二分图，寻找一个最大的匹配。

例：蓝点表示工人，黄点表示工作
权重表示匹配收益，求收益总和最大的匹配



用网络流解最大权匹配

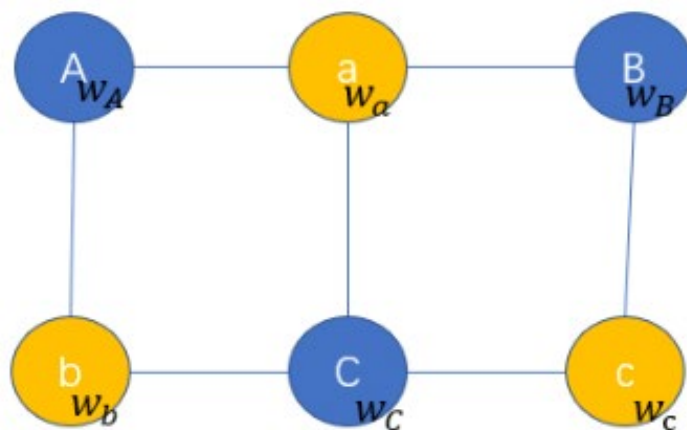


$$\begin{aligned} cap_{ij} &= w_{ij} \\ cap_{Si} &= cap_{jT} = \infty \end{aligned}$$

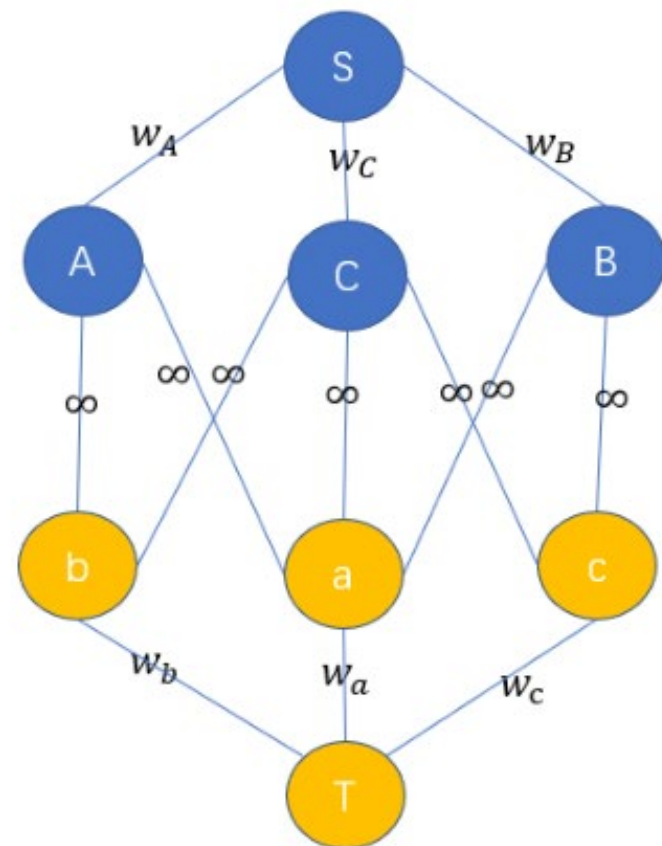
最大流应用3：最小割问题

问题：给一个 $m \times n$ 格子，每个格子有一定数量宝箱，要求不得同时选取相邻格子，求选得宝箱的最大数量。

解：先转化成二分图最小割问题，用最大流算法求解



$G = (\{A, B, C\}, \{a, b, c\}, E)$



结果为：
总和 - 最小割

最小生成树+匹配应用——TSP的 $\frac{3}{2}$ 近似算法

- 1) Find a minimum spanning tree T of G .
- 2) Let E' be the set of vertices whose degree is odd in T . By the handshaking lemma, E' has an even number of vertices.
- 3) Find a **minimum-weight perfect matching** M in the induced subgraph given by the vertices from E' .
- 4) Combine the edges of M and T to form a connected multigraph G' in which every vertex has even degree.
- 5) Form a Eulerian circuit in G' and make the circuit found in previous step into a Hamiltonian circuit by removing the repeated vertices from the circuit, which is called shortcutting.

引用： N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, Tech. Rep., 1976.



P4

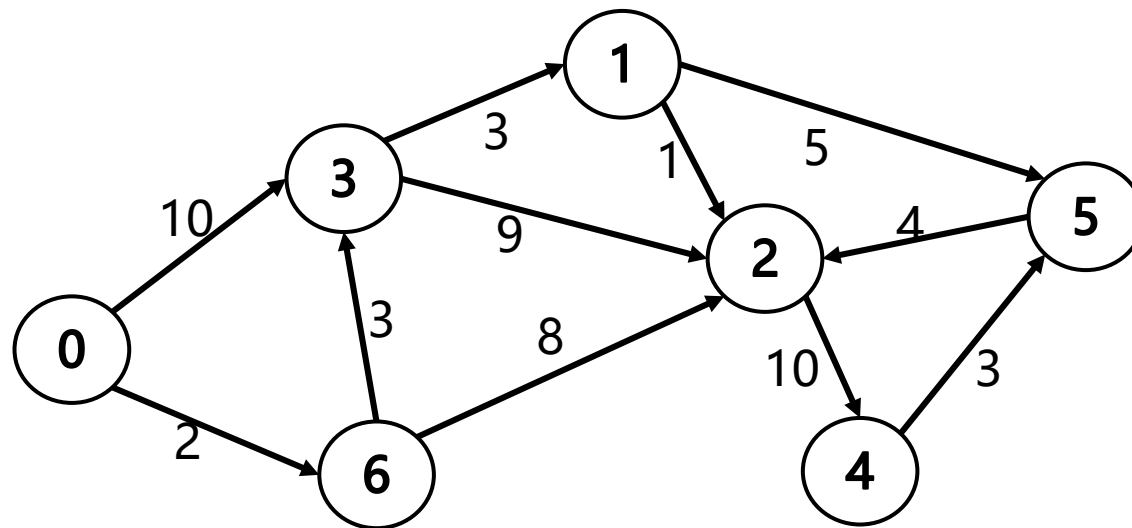


组合优化图论模型

拓扑排序
着色问题
覆盖问题

拓扑排序

顶点：工程项目
有向边：依赖关系
边权：完成时间



求一个合理的顺序（拓扑排序）

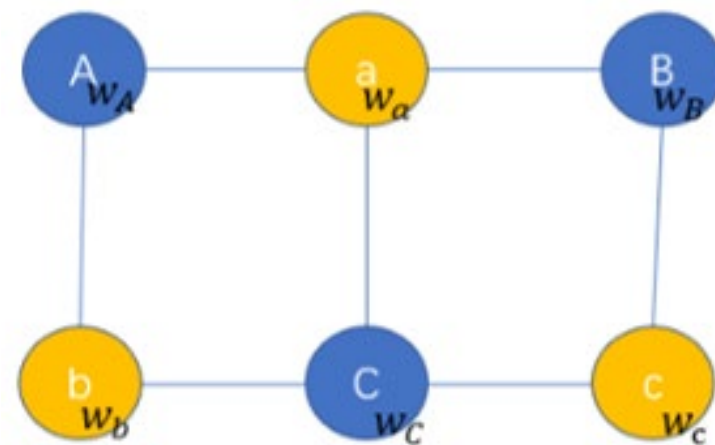
和完成时间（关键路径）

着色问题

问题：在学生选课不冲突的情况下，如何制订一张课时数尽可能少的课表

模型：点表示课程，连线表示当且仅当有某个学生同时选了这两门课程

转化后的问题：问题转化为求该图的点色数

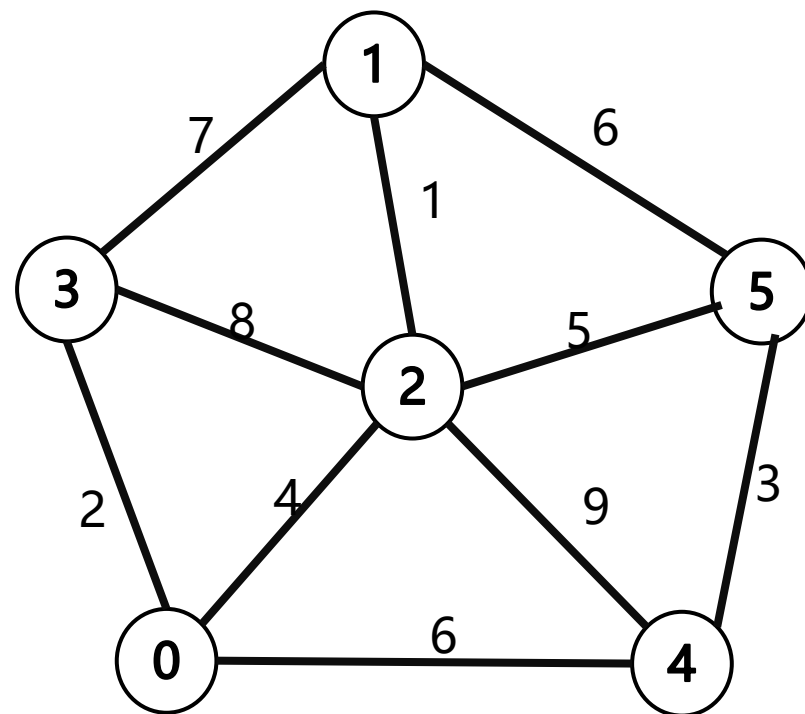


覆盖问题

问题：选择一个居民代表团使得认识所有居民

模型：点表示人，连线表示互相认识

转化后的问题：选一个点集使得所有点在点集中或和点集中的点相邻



图算法小结

- 图搜索： 深度优先搜索， 广度优先搜索， 一致代价搜索
- 最短路径算法
 - 单源 Dijkstra (正权图)、 Bellman-Ford (SPFA)
 - 多源 Floyd-Warshall
- 最小生成树 Kruskal、 Prim
- 最大网络流 Ford-Fulkerson

拓展问题： 中国邮路问题(欧拉图)、 旅行商问题 (哈密顿图)

- 斯坦纳树、 哈夫曼树
- 网络流应用
- 拓扑排序， 图着色、 覆盖.....

赛题举例

美赛2020D

Teaming Strategies

——从传球网络分析团队合作规律

- 1.以球员为点，传球为连边，建立传球关系图(网络)
- 2.定义一些指标，量化评价这一网络
- 3.根据前面的分析给教练提建议
- 4.推广到其他团队合作

美赛2019D

Time to leave the Louvre

——逃离卢浮宫的路线、指挥方法设计

根据地图设计疏散方案，思路提示有：

- 找一下疏散“瓶颈”，就是会拥堵的位置
- 考虑游客的语言、年龄等不同
- 考虑如何开放秘密出口
- 科技手段



2



图算法程序实现



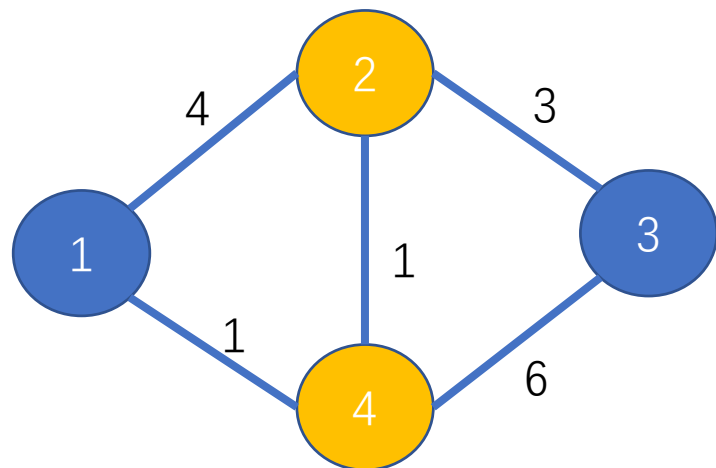


图在计算机中的表示

邻接矩阵
邻接表

图的表示

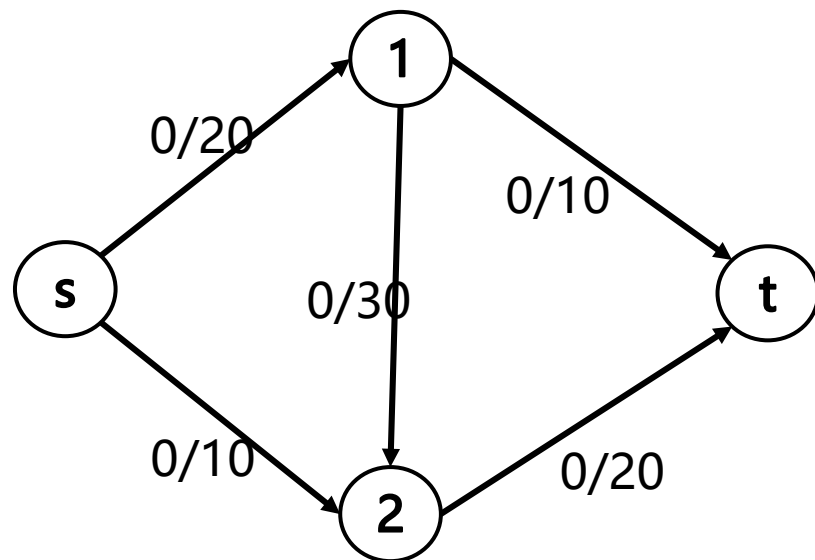
邻接矩阵



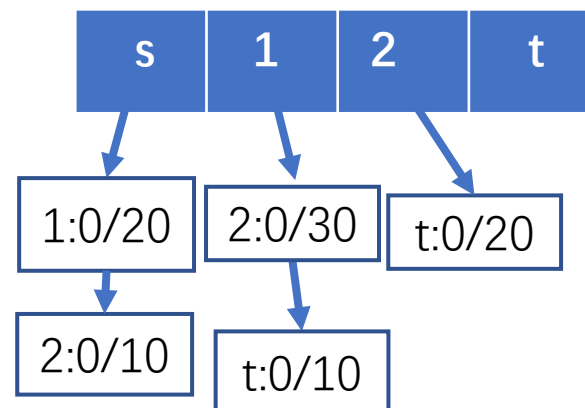
比较简单，矩阵实现

0	2	5	1
2	0	3	1
5	3	0	4
1	1	4	0

邻接表



功能较强，访问效率较高



链表数组实现

C++的vector<vector<类型>>实现



P1



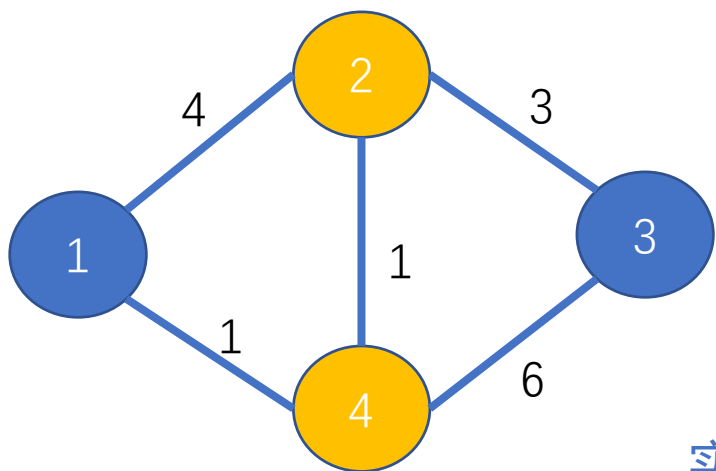
路树流MATLAB快餐



图算法实现

1.点表示地点，边权重表示路线长度
1.1 最短路

例:



$G=(V,E)$

点集 $V=\{1,2,3,4\}$,

边集 $E=\{(1,2,4), (1,4,1), (2,4,1), (2,3,3), (3,4,6)\}$

理论上我们要: Dijkstra、Bellman-Ford (SPFA)、

Floyd

实际上: `graphallshortestpaths(W);`

要路径的话: `graphshortestpath(W, 1, 3);`

0	2	5	1
2	0	3	1
5	3	0	4
1	1	4	0

1.2 最小生成树

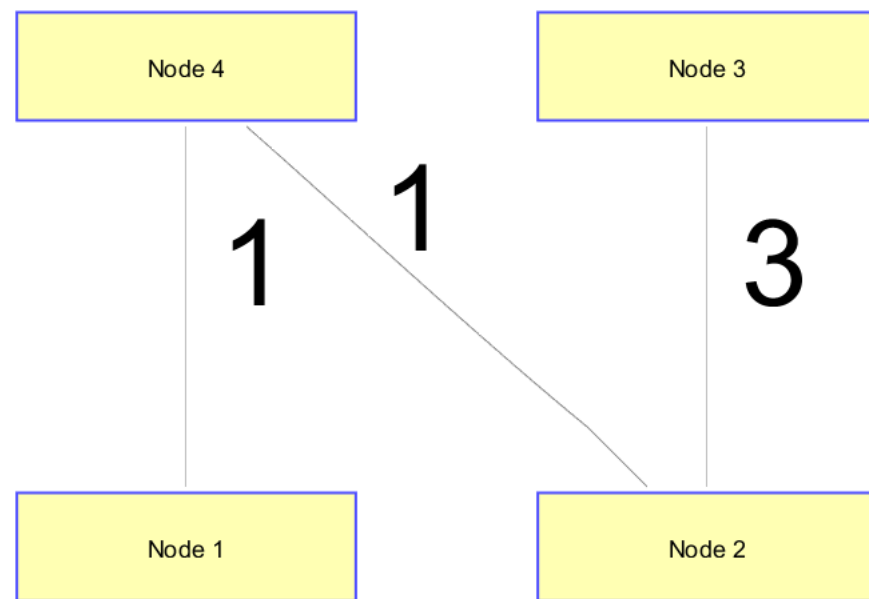
理论上我们要: Kruskal、Prim

实际上: `graphminspantree(W);`

1.3 TSP 问题

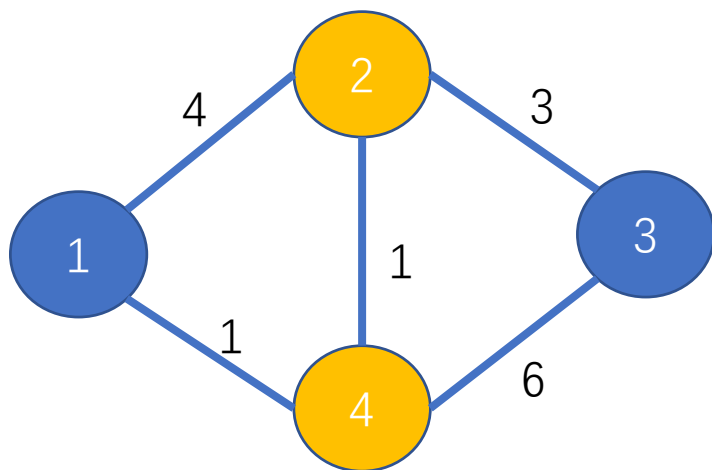
小规模可以动态规划

大规模用最小生成树得近似解



图算法实现

例:



$G=(V,E)$
点集 $V=\{1,2,3\}$,
边集 $E=\{(1,2,4), (1,4,1), (2,4,1), (2,3,3), (3,4,6)\}$

2.点表示地点, 边权表示水管容量: 网络流

理论上我们要: Ford-Fulkerson(Edmonds-Karp, Dinic)

实际上: `graphmaxflow(W, 1, 3);` 或 线性规划

最大流为5:

FlowMatrix =

(1, 2)	4.0000
(2, 3)	3.0000
(4, 3)	2.0000
(1, 4)	1.0000
(2, 4)	1.0000

设边流量分别为 $x_{12}, x_{14}, x_{24}, x_{23}, x_{43}$

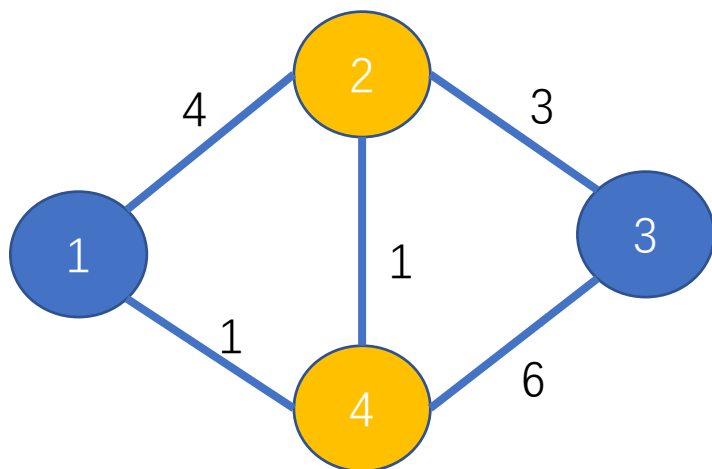
$$\begin{aligned} \max f(x) &= x_{23} + x_{43} \\ \text{s.t.} \quad &x_{12} - x_{24} - x_{23} = 0 \\ &x_{14} + x_{24} - x_{43} = 0 \\ &x_{ij} \leq \text{cap}_{ij} (i = 1, 2, 3, 4) \end{aligned}$$

网络流模型延伸(均可线规):

- 匹配问题
- 最小费用流
- 最小割问题

图算法实现

例:



$G=(V,E)$
点集 $V=\{1,2,3\}$,
边集 $E=\{(1,2,4), (1,4,1), (2,4,1), (2,3,3), (3,4,6)\}$

最短路、最小生成树、网络流MATLAB代码

```
vi=[1,1,2,2,4];  
vj=[2,4,4,3,3];  
weight=[4,1,1,3,6];  
W=sparse (vi,vj,weight,4,4);  
graph= full(W);%邻接矩阵  
[MaxFlow, FlowMatrix, Cut] = graphmaxflow(W, 1, 3);  
%最大流  
W=W+W';  
[dist,path,pred]=graphshortestpath(W,1,3); %单源最短路  
dist2= graphallshortestpaths(W); %多源最短路  
[Tree, pred] = graphminspantree(W); %最小生成树  
view(biograph(Tree, [], 'ShowArrows','off', 'ShowWeights', 'on', 'EdgeFontSize', 45))
```



P2



前文算例C++演示

最短路算法

Dijkstra's Algorithm

node	dist	path
0	0	0
1	8	0 → 6 → 3 → 1
2	9	0 → 6 → 3 → 1 → 2
3	5	0 → 6 → 3
4	19	0 → 6 → 3 → 1 → 2 → 4
5	13	0 → 6 → 3 → 1 → 5
6	2	0 → 6

Bellman-Ford Algorithm

node	dist	path
0	0	0
1	8	0 → 6 → 3 → 1
2	9	0 → 6 → 3 → 1 → 2
3	5	0 → 6 → 3
4	19	0 → 6 → 3 → 1 → 2 → 4
5	13	0 → 6 → 3 → 1 → 5
6	2	0 → 6

Floyd-Warshall Algorithm

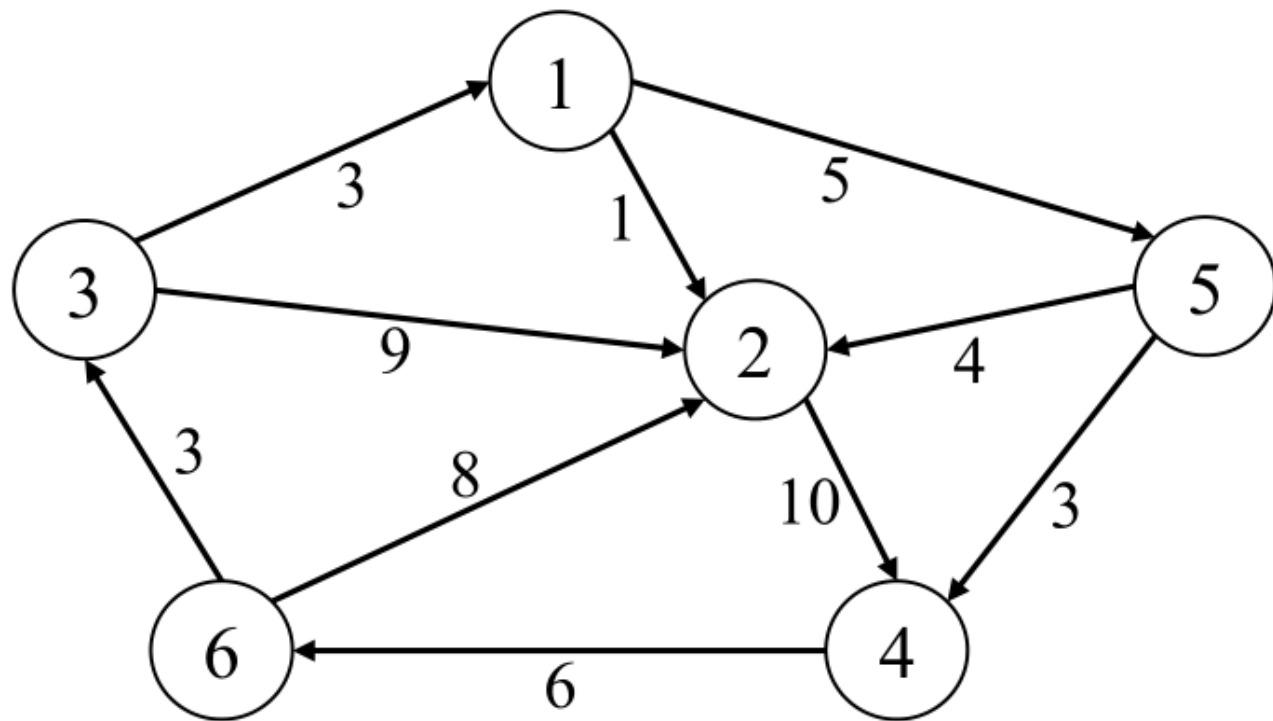
dist:

0	1	17	8	5	14
22	0	19	10	27	16
3	4	0	11	8	17
12	13	9	0	17	6
15	4	12	3	0	9
6	7	3	14	11	0

prev:

0	0	5	4	0	3
2	0	5	1	0	3
2	0	0	4	0	3
2	0	5	0	0	3
2	4	5	4	0	3
2	0	5	4	0	0

最短路算法



Floyd-Warshall Algorithm

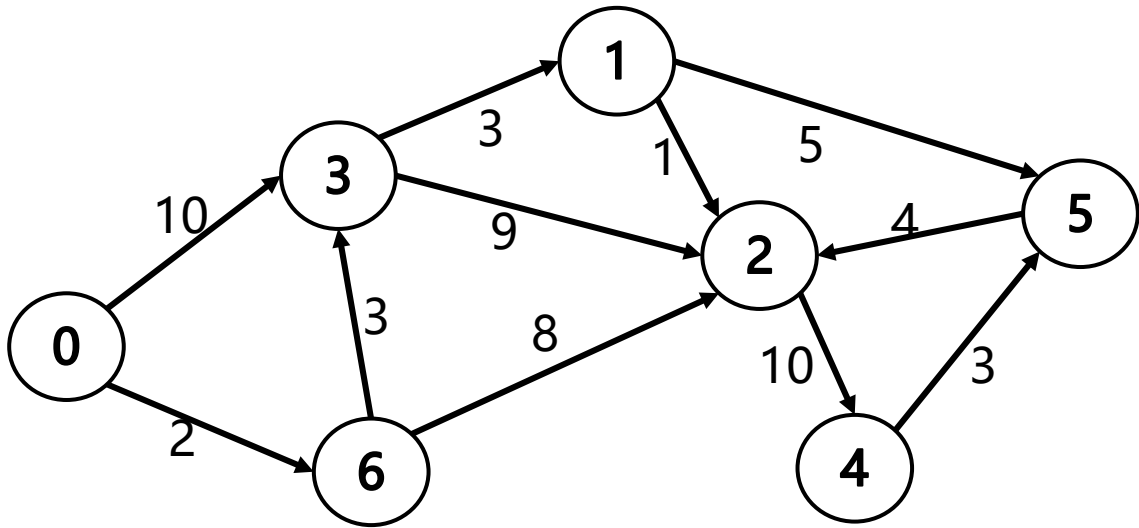
dist:

```
0 1 17 8 5 14
22 0 19 10 27 16
3 4 0 11 8 17
12 13 9 0 17 6
15 4 12 3 0 9
6 7 3 14 11 0
```

prev:

```
0 0 5 4 0 3
2 0 5 1 0 3
2 0 0 4 0 3
2 0 5 0 0 3
2 4 5 4 0 3
2 0 5 4 0 0
```

最短路算法



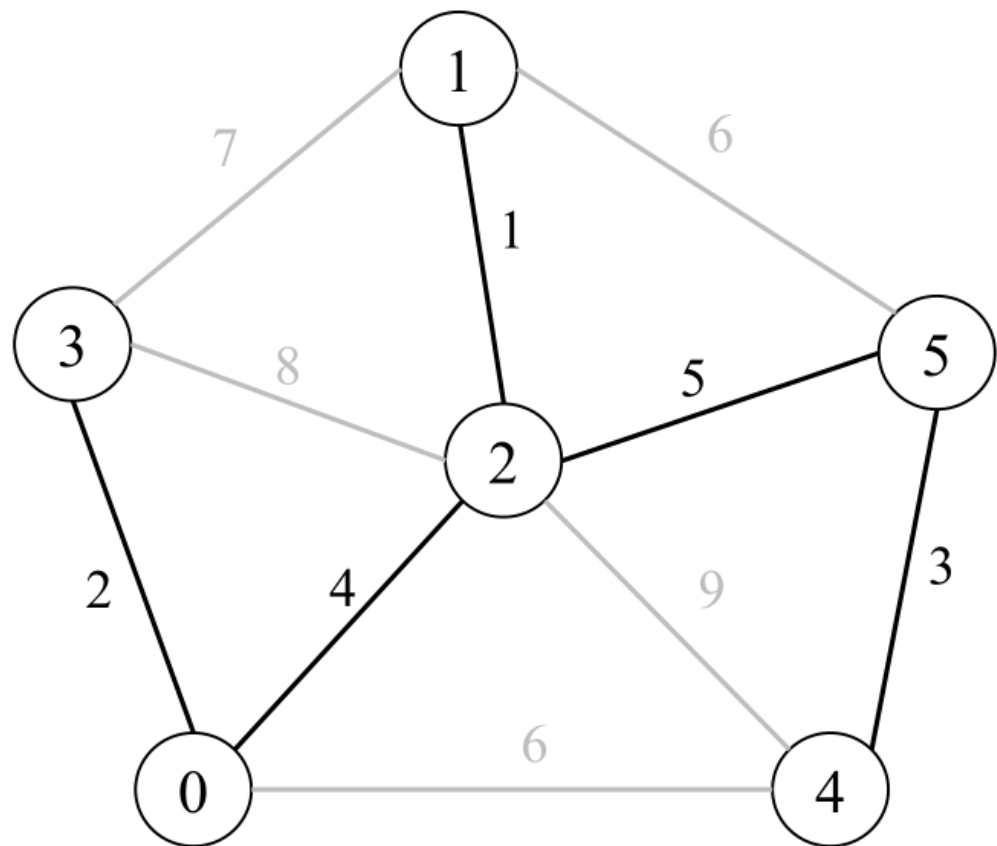
Dijkstra's Algorithm

node	dist	path
0	0	0
1	8	0 -> 6 -> 3 -> 1
2	9	0 -> 6 -> 3 -> 1 -> 2
3	5	0 -> 6 -> 3
4	19	0 -> 6 -> 3 -> 1 -> 2 -> 4
5	13	0 -> 6 -> 3 -> 1 -> 5
6	2	0 -> 6

Bellman-Ford Algorithm

node	dist	path
0	0	0
1	8	0 -> 6 -> 3 -> 1
2	9	0 -> 6 -> 3 -> 1 -> 2
3	5	0 -> 6 -> 3
4	19	0 -> 6 -> 3 -> 1 -> 2 -> 4
5	13	0 -> 6 -> 3 -> 1 -> 5
6	2	0 -> 6

最小生成树



Prim's Algorithm

0: 3 2

1: 2

2: 5

3:

4: 5

5:

weight: 15

Kruskal's Algorithm

0: 2 3

1: 2

2: 5

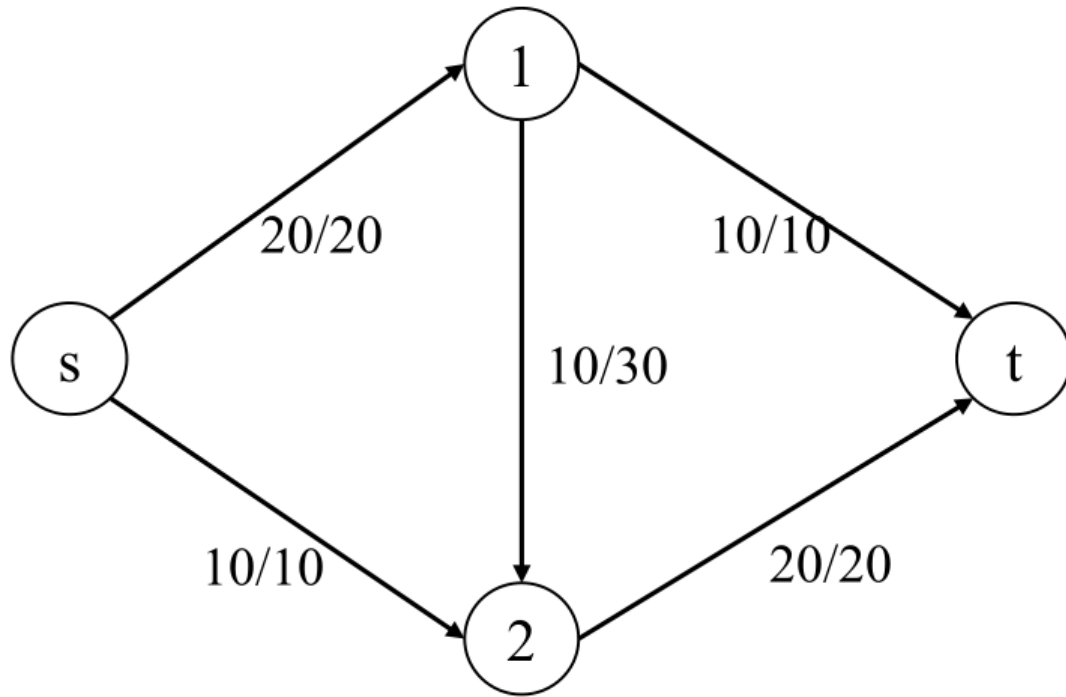
3:

4: 5

5:

weight: 15

网络流



$$|f| = 30$$

```
Ford-Fulkerson Algorithm  
0: 2 (10) 1 (20)  
1: 2 (10) 3 (10)  
2: 3 (20)  
3:  
flow value: 30
```



P3



习题推荐

习题

- (空) :上海交大评测平台<http://acm.sjtu.edu.cn/OnlineJudge/>
- POJ: 北大评测平台<http://poj.org/>
- HDU: 杭电评测平台<http://acm.hdu.edu.cn/>

- 图搜索: 1243, 广搜1281,1564, 1521, 深搜1606
- 最短路: 1635, 1565 (最小费用最短路), POJ2387
- 最小生成树: 1234(Prim),POJ1251(Prim),POJ1287 (Kruskal)
- 网络流: HDU1532, POJ3281,POJ1459
- 拓扑排序: HDU1285, HDU4109



THANKS FOR LISTENING

- 参考资料:
1. 0201-范佳豪-图算法
 2. 上海交通大学CS222算法课课件
 3. 加州大学伯克利分校CS188人工智能课件
 4. 范佳豪、曾锴鹏代码