算法分析与设计

Analysis and Design of Algorithm

Lesson 14

第五章小结

- 回溯法适用条件：多米诺性质
- 回溯法设计步骤
  1. 定义解向量和每个分量的取值范围
  2. 确定搜索策略：深度优先/宽度优先，...
  3. 确定每结点分支的约束条件，判断是否满足多米诺性质
  4. 确定存储搜索路径的数据结构
- 通过应用范例学习回溯法的设计策略
  - *n*后问题、0-1背包问题、旅行售货员问题
  - 装载问题
  - 图的着色问题

课程内容

NP完全性理论与近似算法

随机化算法

线性规划与网络流

递归分治

动态规划

贪心算法

回溯与分支限界

算法分析与问题的计算复杂性

算法高级理论

高级算法

基础算法

算法基础理论

第六章 分支限界法

学习要点

- 理解分支限界法的剪枝搜索策略
- 掌握分支限界法的算法框架
  - 队列式分支限界法
  - 优先队列式分支限界法
- 通过应用范例学习分支限界法的设计策略
  - 背包问题
  - 0-1背包问题
  - 非对称旅行商问题
  - 单源最短路径问题
  - 装载问题
  - 最大团问题

分支限界法概述

分支限界法与回溯法的区别

- **求解目标**：
  - 一般情况下，回溯法的求解目标是找出解空间树中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解（可以理解为更细粒度的回溯法）。
- **搜索策略**：
  - 回溯法更多地以深度优先的方式搜索解空间树，而分支限界法则更多地以广度优先或以最小耗费优先（函数优先）的方式搜索解空间树。
- **各自特点**：
  - 回溯法空间效率高；分支限界法往往更快。

分支限界法的基本思想

- 分支限界法常以**广度优先**或以**最小耗费（最大效益）优先**的方式搜索问题的解空间树。
- 对已处理的各结点根据限界函数估算目标函数的可能取值，从中选取使目标函数取得极值（极大/极小）的结点优先进行广度优先搜索➔不断调整搜索方向，尽快找到解。
- **特点：**限界函数常基于问题的目标函数，适用于求解最优化问题。

常见的分支限界法

- 队列式分支限界法
  - 按照队列先进先出原则选取下一个结点为扩展结点。
- 优先队列式分支限界法
  - 按照规定的结点费用最小原则选取下一个结点为扩展结点（常采用优先队列实现）。
- 栈式分支限界法
  - 按照栈后进先出原则选取下一个结点为扩展结点。

组合优化问题的分支限界法

组合优化问题

- 组合优化问题的相关概念
  - 目标函数（极大化或极小化）
  - 约束条件（解满足的条件）
  - 可行解：搜索空间满足约束条件的解
  - 最优解：使得目标函数达到极大（极小）的可行解

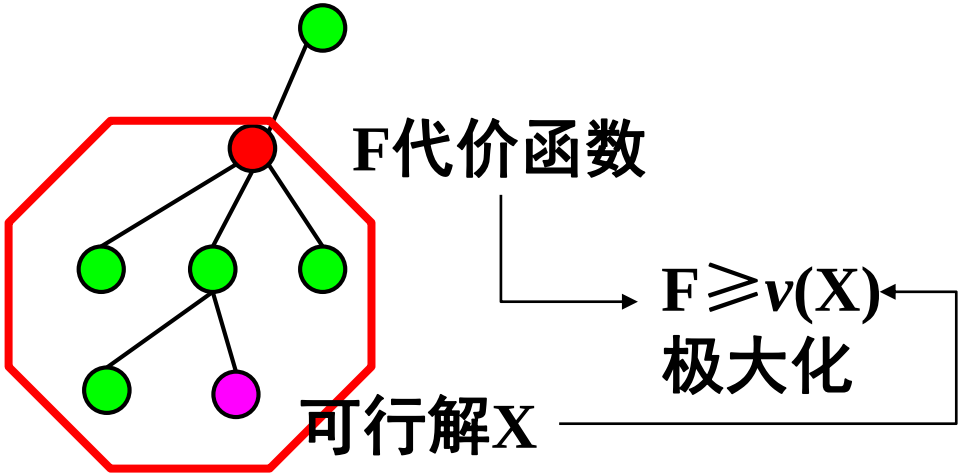
- 背包问题

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$s.t. \quad \begin{cases} 2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10 \\ x_i \in \mathbb{N}, i = 1,2,3,4 \end{cases}$$

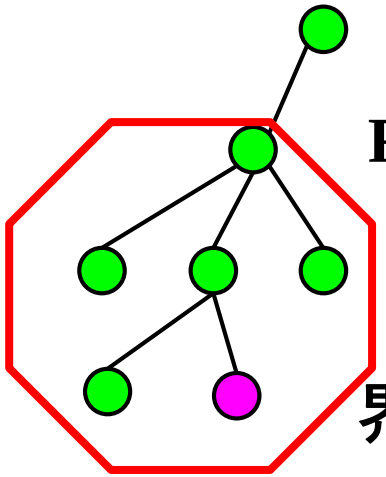
代价函数

- **计算位置：** 搜索树的结点
- **估值：** 极大化问题是以该点为根的子树所有可行解的值的上界（极小化问题则为下界）
- **性质：** 对极大化问题父节点代价不小于子结点的代价（极小化问题则相反）



界

- **含义：** 当前得到可行解的目标函数最大值（极小化问题则相反）
- **初值：** 极大化问题初值为0（极小化问题则为最大值）
- **更新：** 得到更好的可行解时



界 ⊆ 代价函数，即：F≥B

分支限界

- 停止分支回溯父节点的依据
  1. 不满足约束条件
  2. 对于极大化问题，代价函数值小于当前界（对于极小化问题是大于界）
- 界的更新
  - 对极大化问题，如果一个新的可行解的优化函数值大于（极小化问题为小于）当前的界，则把界更新为该可行解的值

实例

- **背包问题**

背包限重为10

物品i / 属性	1	2	3	4
价值v_i	1	3	5	9
重量w_i	2	3	4	7

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$s.t. \quad \begin{cases} 2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10 \\ x_i \in \mathbb{N}, i = 1,2,3,4 \end{cases}$$

代价函数的设定

- 对结点$\langle x_1, x_2, \ldots, x_k \rangle$，估计以该结点为根的子树中可行解的上界
- 按单位重量的价值v_i / w_i从大到小排序
- 代价函数=已装入价值+Δ
  - Δ：还可以继续装入最大价值的上界
  - Δ =背包剩余重量×v_{k+1} / w_{k+1}（可装）
  - Δ =0（不可装）

实例：背包问题

$$\max x_1 + 3x_2 + 5x_3 + 9x_4$$

$$s.t. \quad \begin{cases} 2x_1 + 3x_2 + 4x_3 + 7x_4 \le 10 \\ x_i \in \mathbb{N}, i = 1,2,3,4 \end{cases}$$

对**变元重新排序**使得 $\frac{v_i}{w_i} \ge \frac{v_{i+1}}{w_{i+1}}$

排序后

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$s.t. \quad \begin{cases} 7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10 \\ x_i \in \mathbb{N}, i = 1,2,3,4 \end{cases}$$

代价函数与分支策略

- 结点$\langle x_1, x_2, \ldots, x_k\rangle$的**代价函数F**

若对某个$j > k$有 $b - \sum_{i=1}^{k} w_i x_i \geq w_j$

$$F = \sum_{i=1}^{k} v_i x_i + (b - \sum_{i=1}^{k} w_i x_i)\frac{v_{k+1}}{w_{k+1}}$$

否则 $$F = \sum_{i=1}^{k} v_i x_i$$

分支策略——深度优先+代价函数优先

实例

$$\max 9x_1 + 5x_2 + 3x_3 + x_4$$

$$s.t. \quad \begin{cases} 7x_1 + 4x_2 + 3x_3 + 2x_4 \le 10 \\ x_i \in \mathbb{N}, i = 1,2,3,4 \end{cases}$$

代价函数计算的值

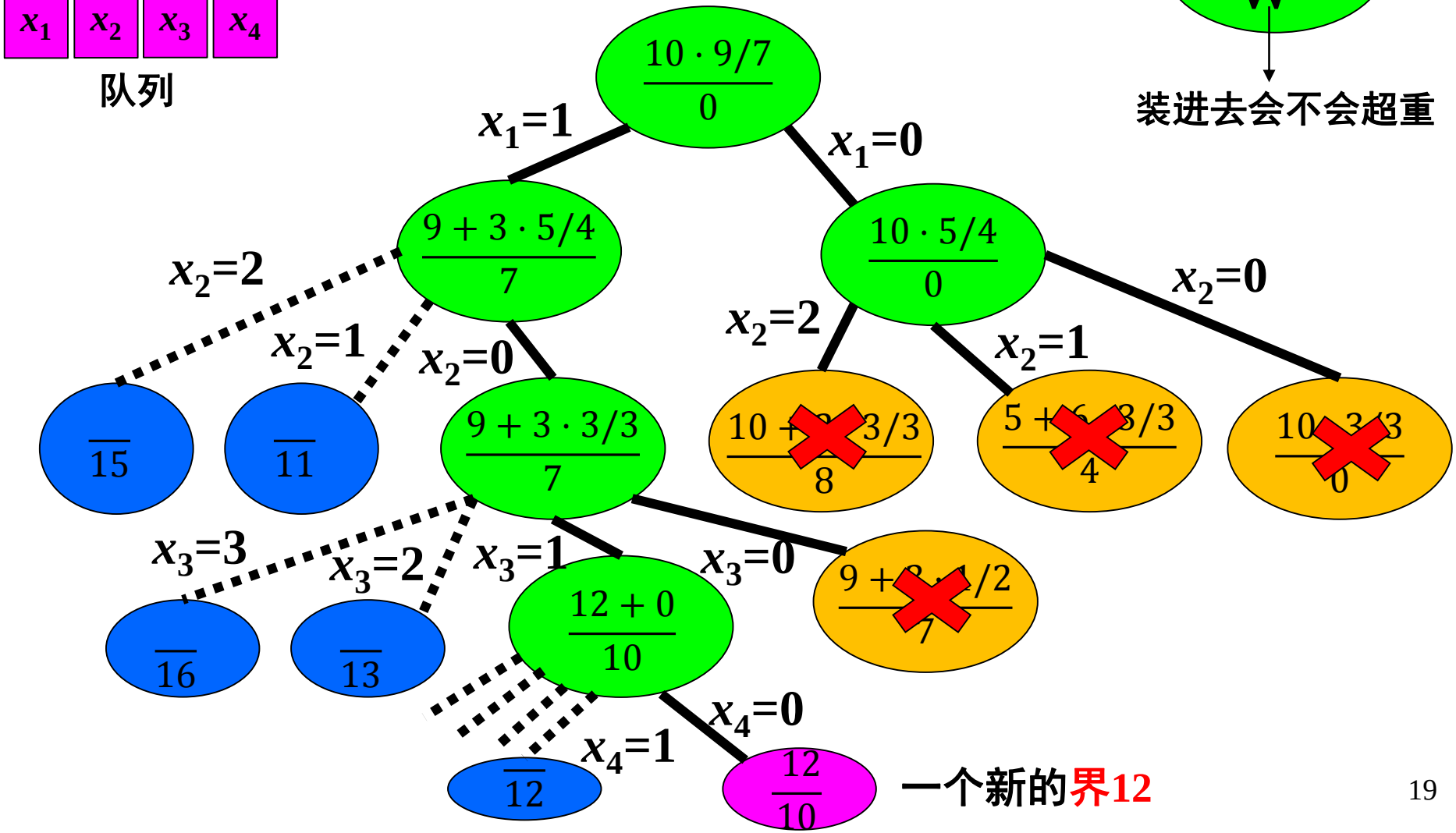
F

W

装进去会不会超重

x_1 x_2 x_3 x_4

队列



一个新的界12

0-1背包问题

- **实例**
  - 4种物品，重量w_i和价值v_i分别为
  - $v_1 = 1, v_2 = 3, v_3 = 5,\ v_4 = 10$
  - $w_1 = 2, w_2 = 3, w_3 = 6,\ w_4 = 7$
  - 背包重量限制为10
- **建模：**

最大化 $x_1 + 3x_2 + 5x_3 + 10x_4$

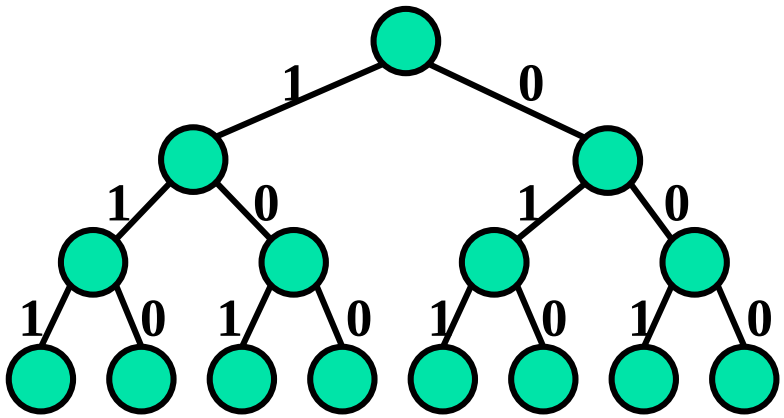
满足约束条件 $\begin{cases} 2x_1 + 3x_2 + 6x_3 + 7x_4 \le 10 \\ x_i \in \{0,1\},\ \ i = 1,2,3,4 \end{cases}$

0-1背包问题—代价函数

- 按v_i/w_i从大到小排序，$i = 1, 2, \cdots, n$
- 假设位于结点$\langle x_1, x_2, \cdots, x_k \rangle$
- 代价函数=已装入价值+Δ
  - Δ：还可继续装入最大价值的上界
  - Δ=背包剩余重量$\times\ v_{k+1}/w_{k+1}$（可装）
  - Δ=0（不可装）

0-1背包问题—分支限界法

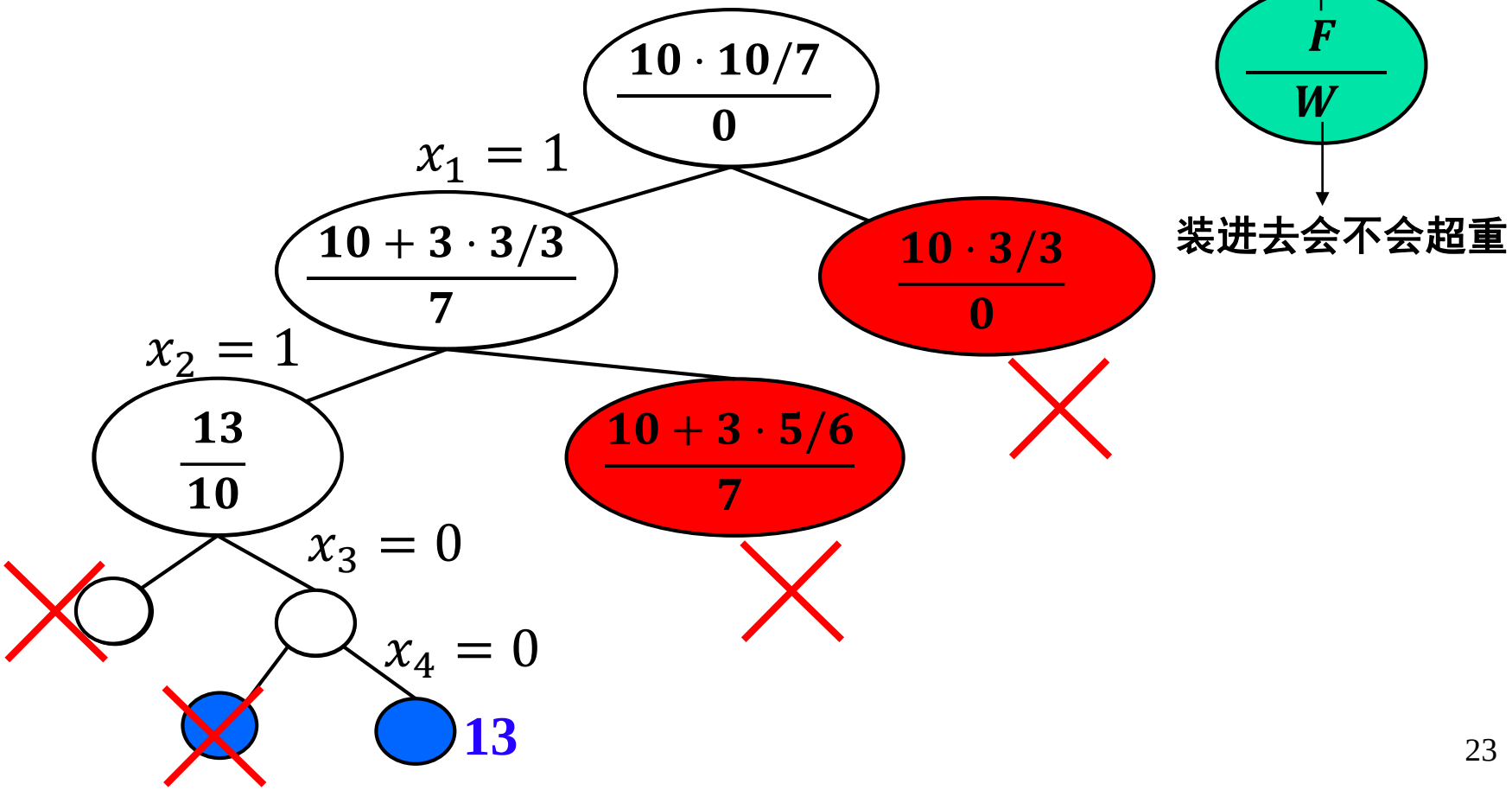
- 基本思想
  1. 将物品按v_i/w_i从大到小排序，确定解空间树
  2. 从空集∅和仅含空集∅的优先队列开始
  3. 选择计算节点队列中代价值最高的节点并扩展
  4. 若扩展出节点不被剪枝，将节点插入节点队列
  5. 反复2~3步，直到优先队列为空时为止
- 代价函数
  - 已装入价值+Δ
- 剪枝函数
  - 与回溯法相同



0-1背包问题—分支限界法

最大化 $10x_1 + 3x_2 + 5x_3 + x_4$

满足 $7x_1 + 3x_2 + 6x_3 + 2x_4 \leq 10; x_i \in \{0,1\}, \quad i = 1,2,3,4$



一个关于巡回演唱会的例子

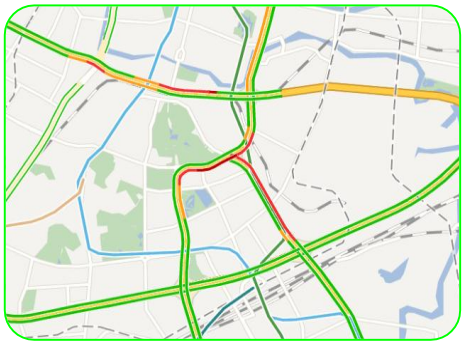
- 薛之谦2018演唱会
  - 地点：北京、上海、广州、深圳、南京、杭州、武汉、成都、重庆、雄安
  - 线路：从北京出发，跑遍各大城市，回到北京

- 目标：考虑机票价格，确定票价最少的线路

票价	北京	上海	广州	……
北京	0	500	600	……
上海	100	0	800	……
广州	1000	200	0	……
……	……	……	……	……

非对称旅行商问题

- **问题定义**
  - 城市集合：$C = \{c_1, c_2, \cdots c_n\}$
  - 城市距离：$d(c_i, c_j)$
  - 距离不对称：$d(c_i, c_j) \neq d(c_j, c_i)$
- **目标：求遍历所有城市（不重复）的最短路径**



道路拥堵情况下
的送快递问题

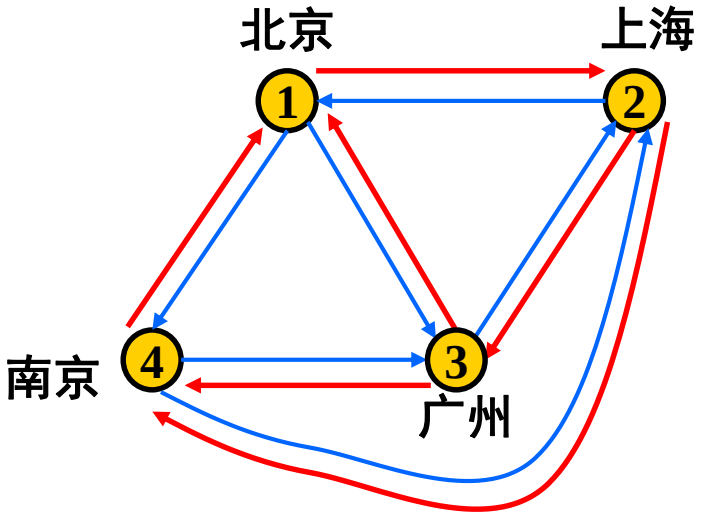
考虑城市单行线
的送快递问题

全国巡回演唱会
的路线安排问题

非对称旅行商问题的解空间

- 实例

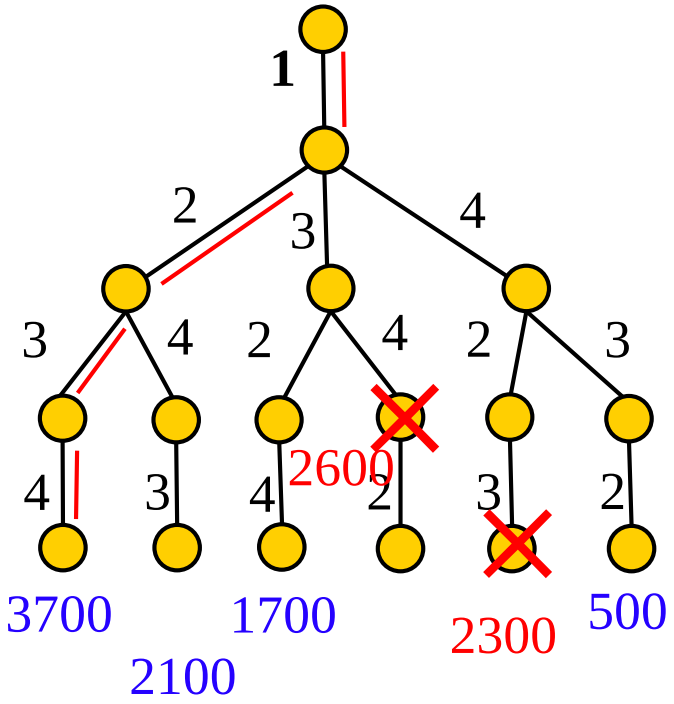
票价	北京	上海	广州	南京
北京	0	500	600	100
上海	100	0	800	500
广州	1000	200	0	2000
南京	400	400	100	0



- 最优解
  - 解的表示：⟨1,4,3,2⟩
  - 路线：北京→南京→广州→上海→北京
  - 总票价：100+100+200+100=500

非对称旅行商问题的解空间树

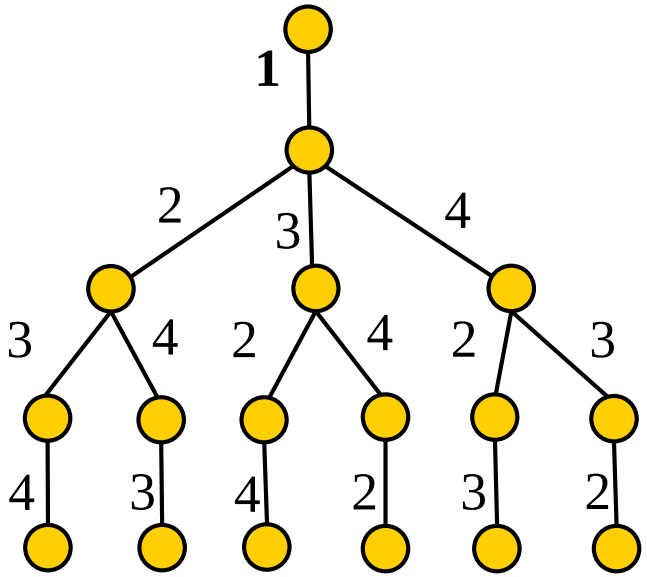
	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



- 回溯法
  - 深度优先遍历解空间树
  - 剪枝函数：比较当前解与当前最优解

非对称旅行商问题的解

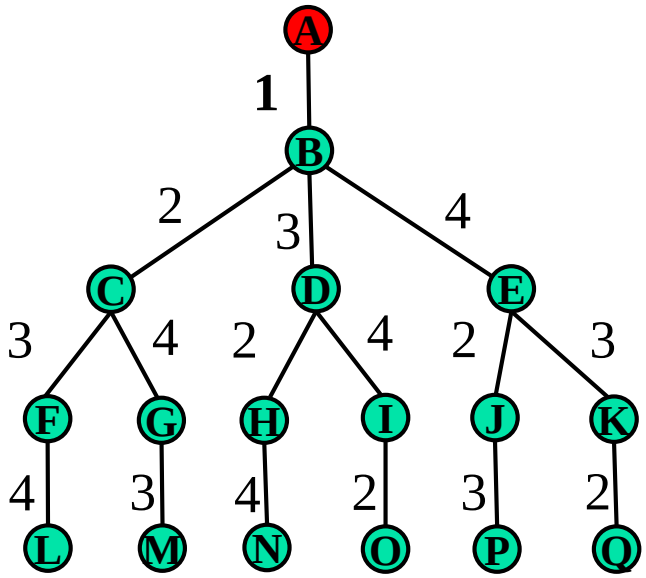
	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



- 如何尽快寻找最优解
  - 不用深度优先搜索
  - 优先访问更靠近最优解的节点
  - 设置代价函数计算优先级
  - 利用优先级队列管理节点
- 如何设计代价函数
  - 当前解的值 + 未来的最优解估计值

非对称旅行商问题的解

	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

红色：优先级队列中的节点
绿色：未被访问的节点
白色：已经完成访问的节点

节点	当前值	未来最优值	总代价
A	0	500	500

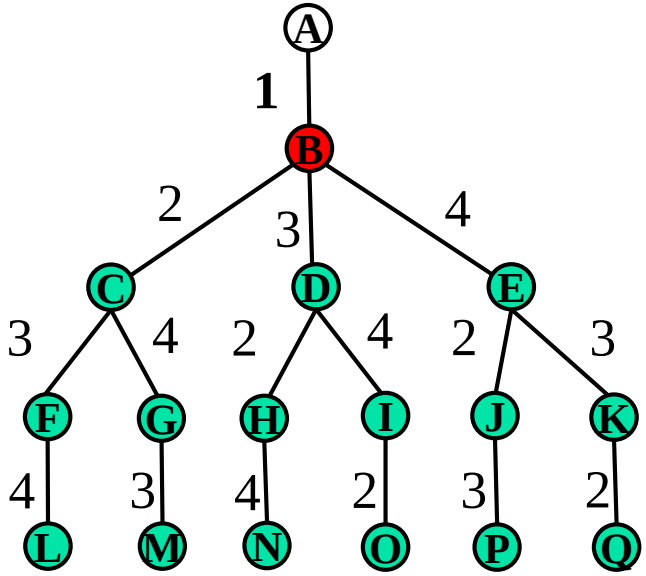
查表

未来最优解估计值=
每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

（100+200+100+100）+ 0

非对称旅行商问题的解

	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

红色：优先级队列中的节点
绿色：未被访问的节点
白色：已经完成访问的节点

节点	当前值	未来最优值	总代价
B	0	500	500

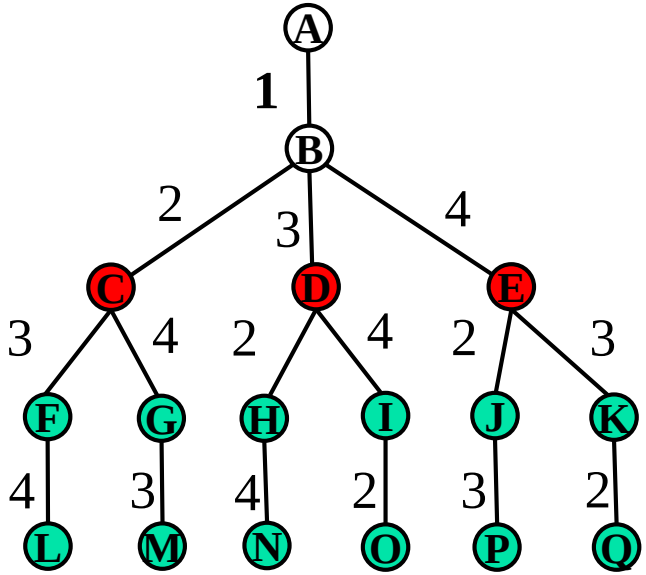
查表

未来最优解估计值=
每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

（100+200+100）+100

非对称旅行商问题的解

	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

红色：优先级队列中的节点
绿色：未被访问的节点
白色：已经完成访问的节点

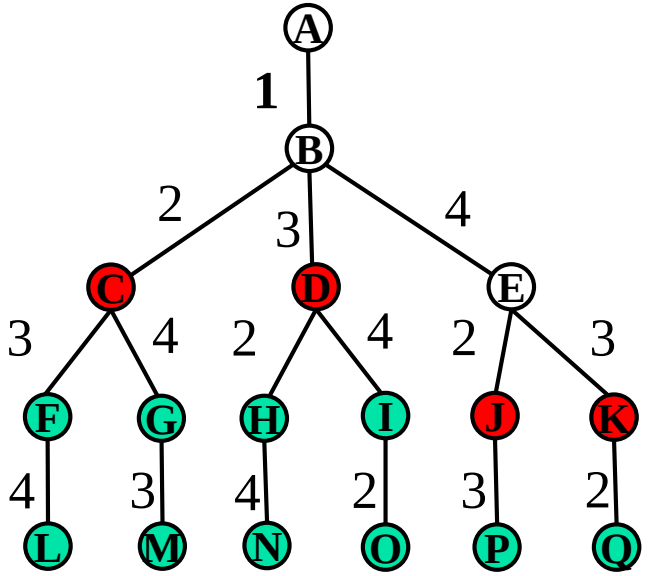
节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
E	100	400	500

查表

未来最优解估计值=
每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解

	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

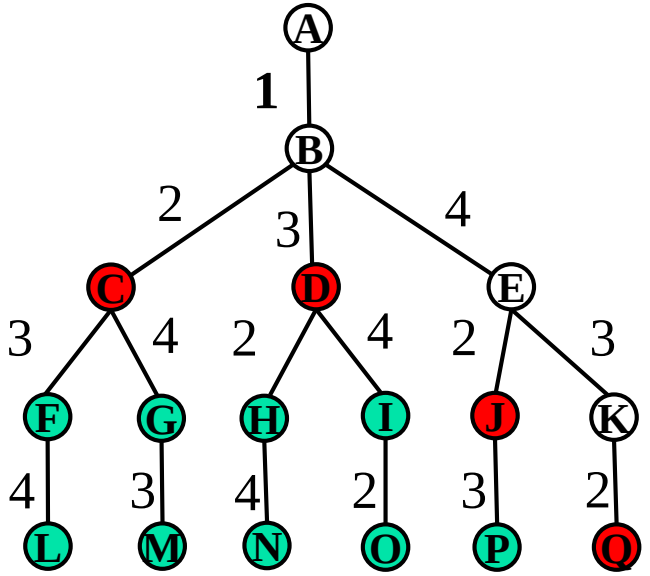
红色：优先级队列中的节点
绿色：未被访问的节点
白色：已经完成访问的节点

节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
J	500	300	800
K	200	300	500

未来最优解估计值=
每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解

	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

红色：优先级队列中的节点
绿色：未被访问的节点
白色：已经完成访问的节点

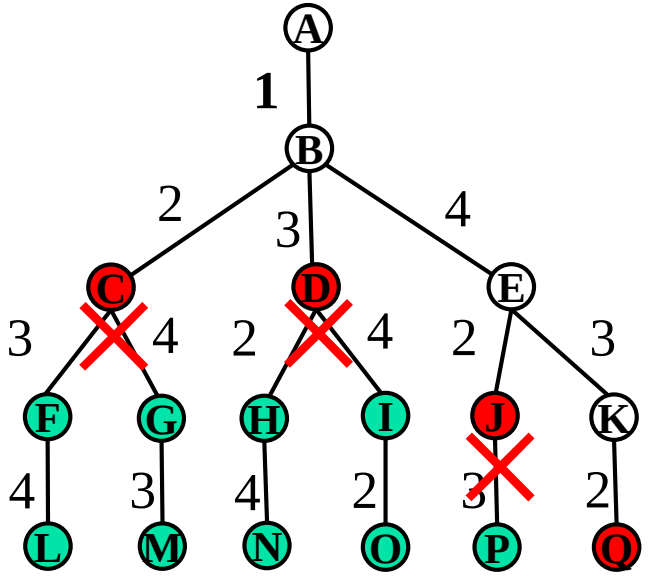
节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
J	500	300	800
Q	400	100	500

当前最优

未来最优解估计值=
每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解

	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

红色：优先级队列中的节点
绿色：未被访问的节点
白色：已经完成访问的节点

节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
J	500	300	800
Q	400	100	500

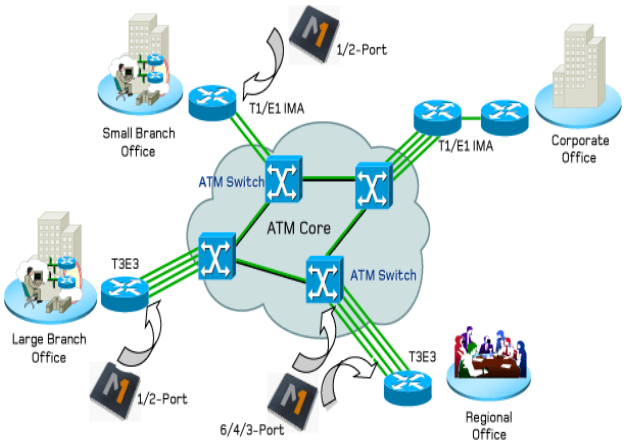
当前最优

未来最优解估计值=
每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

最短路径问题（回顾）

- 最短路径问题是图论研究中的一个经典问题，旨在寻找图（由结点和路径组成的）中两结点之间的最短路径。

地图导航

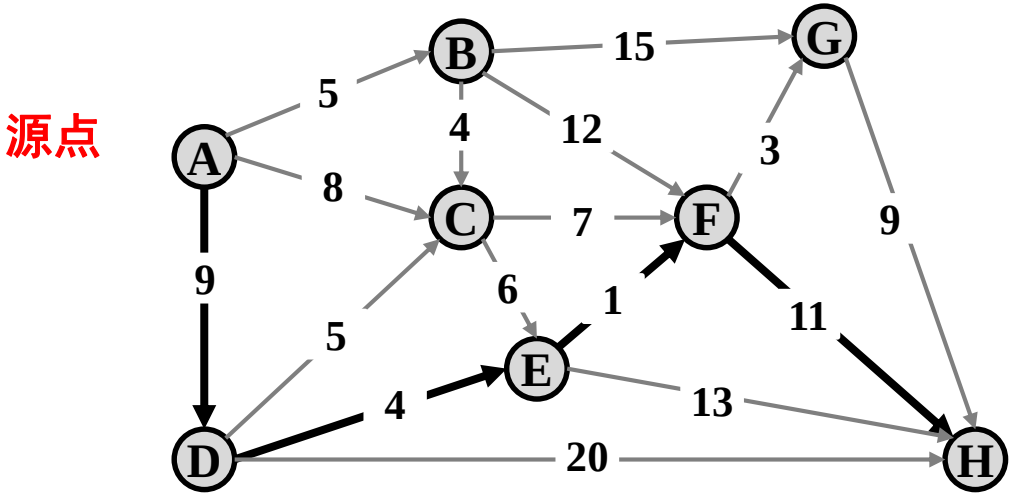


网络路由

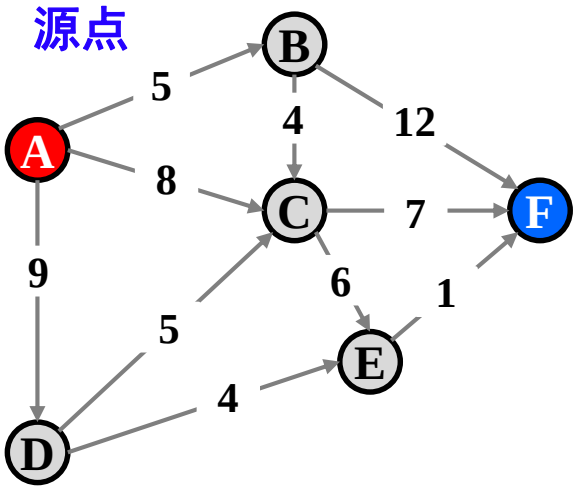
极速外卖

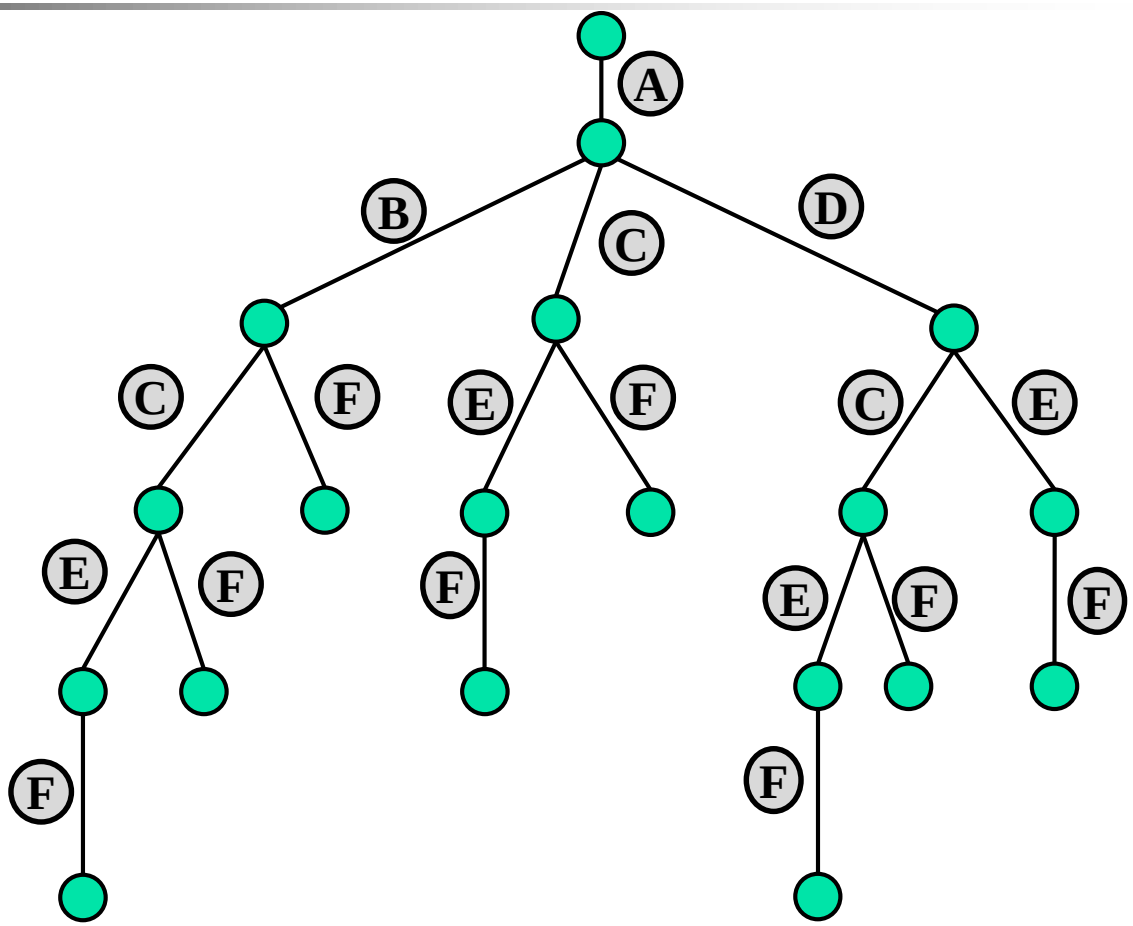
单源最短路径（回顾）

- 给定带权有向图$G=(V, E)$，其中每条边的权是非负实数。给定V中的一个顶点作为源点，求源点到所有其他各顶点的最短路长度。

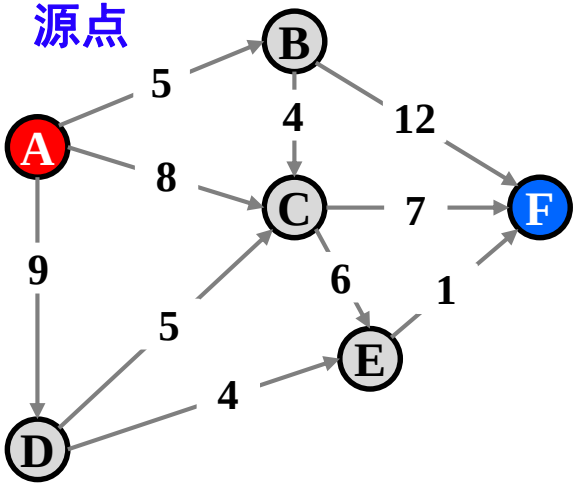


计算距离(A,F)的解空间树



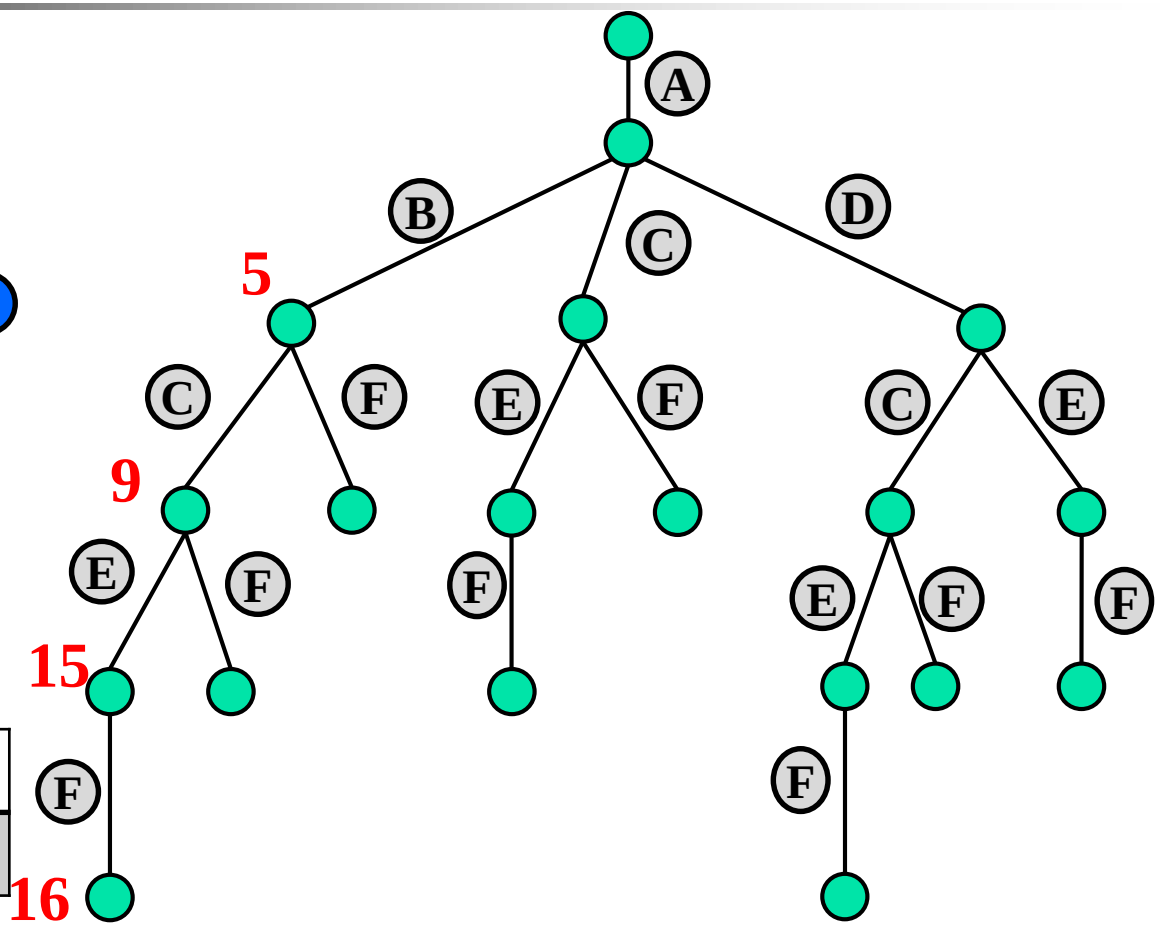


计算距离(A,F)—回溯法

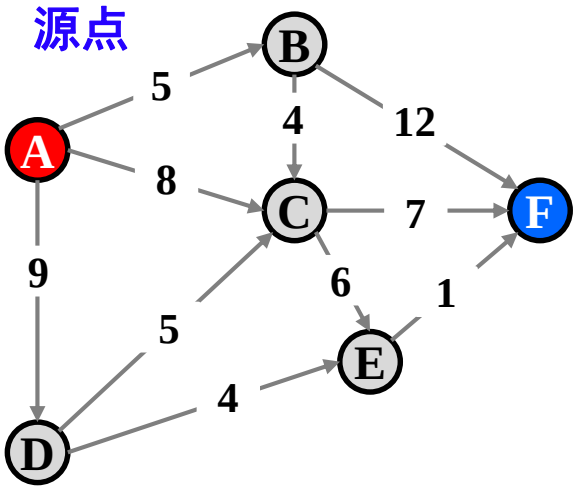


当前最短距离

	B	C	D	E	F
dist	5	9		15	16

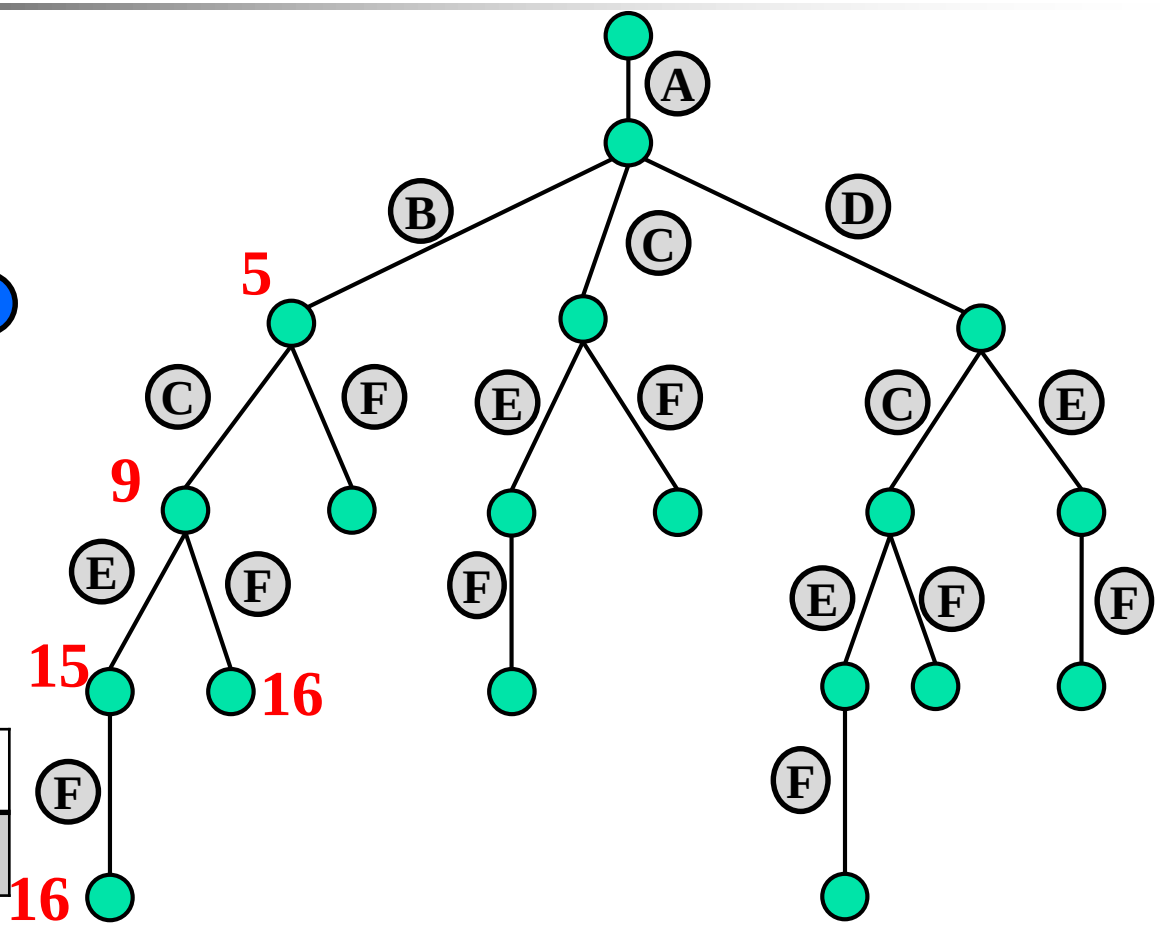


计算距离(A,F)—回溯法

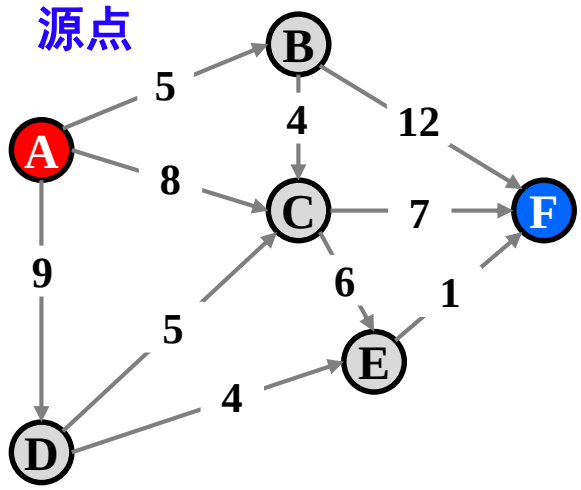


当前最短距离

	B	C	D	E	F
dist	5	9		15	16

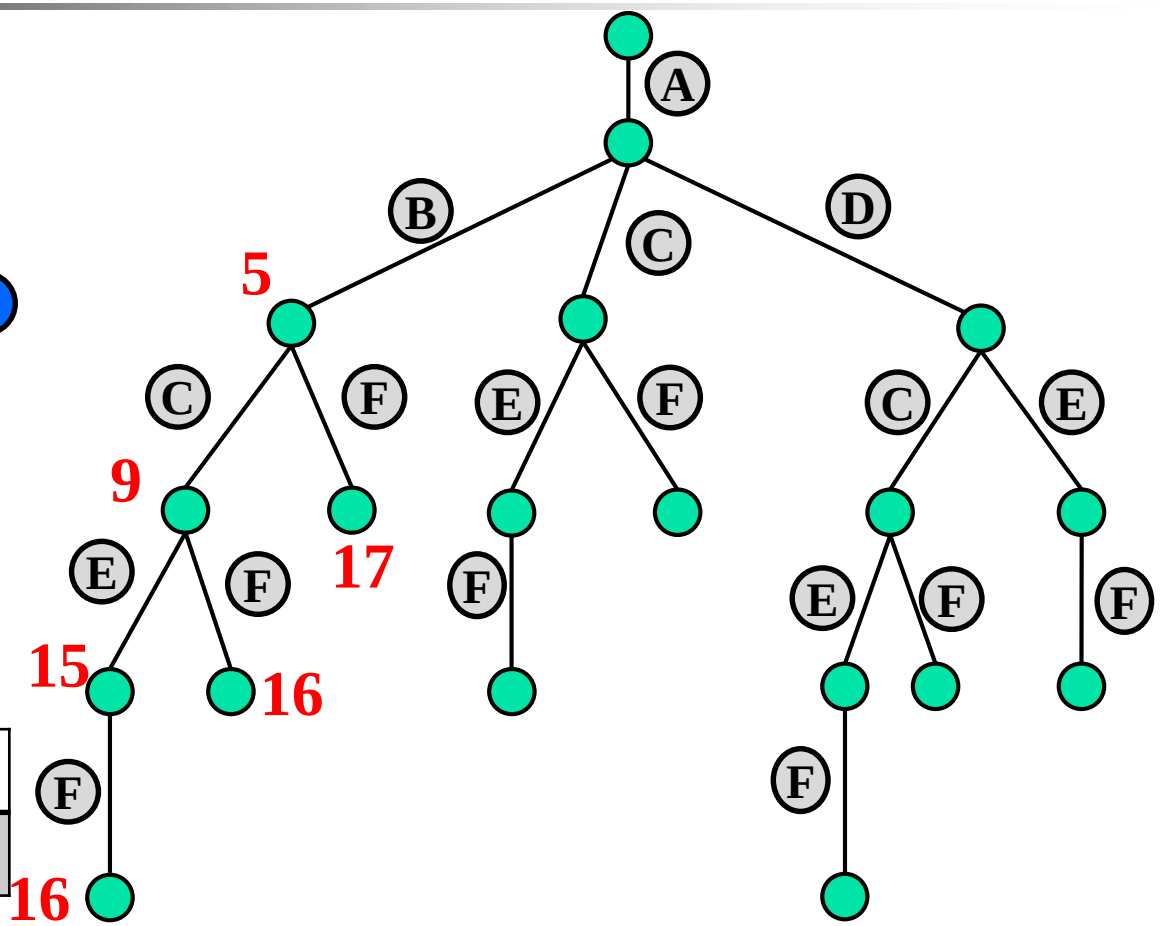


计算距离(A,F)—回溯法

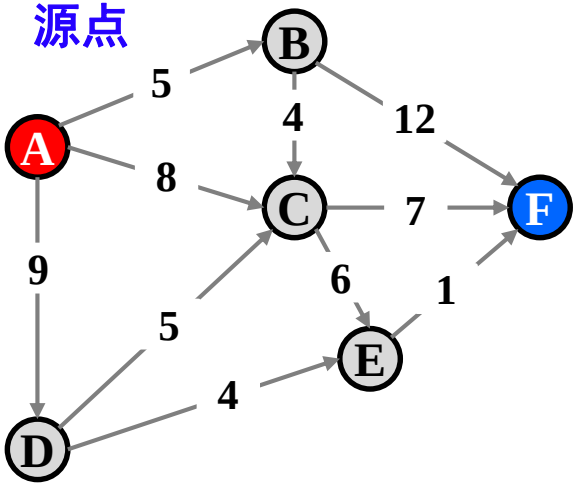


当前最短距离

	B	C	D	E	F
dist	5	9		15	16

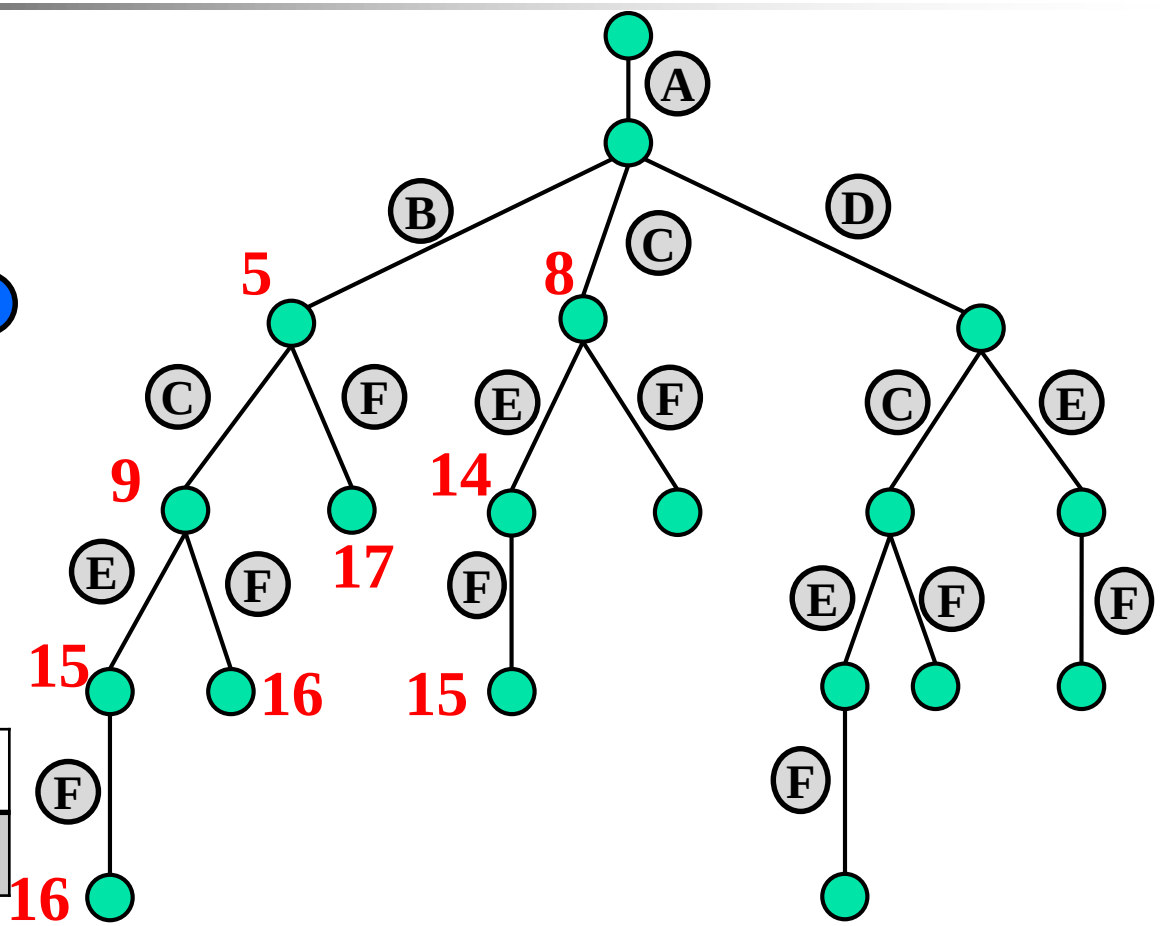


计算距离(A,F)—回溯法

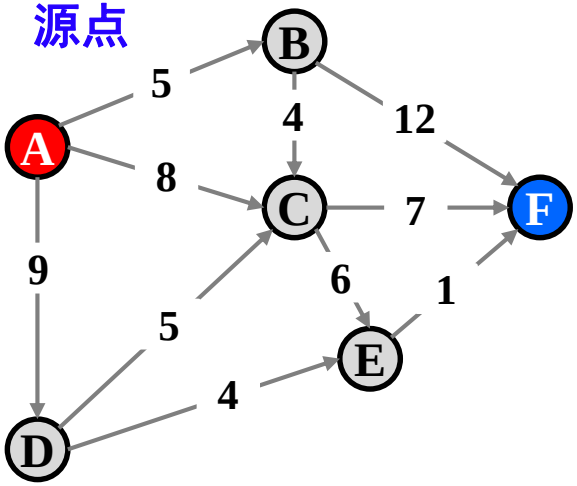


当前最短距离

	B	C	D	E	F
dist	5	8		14	15

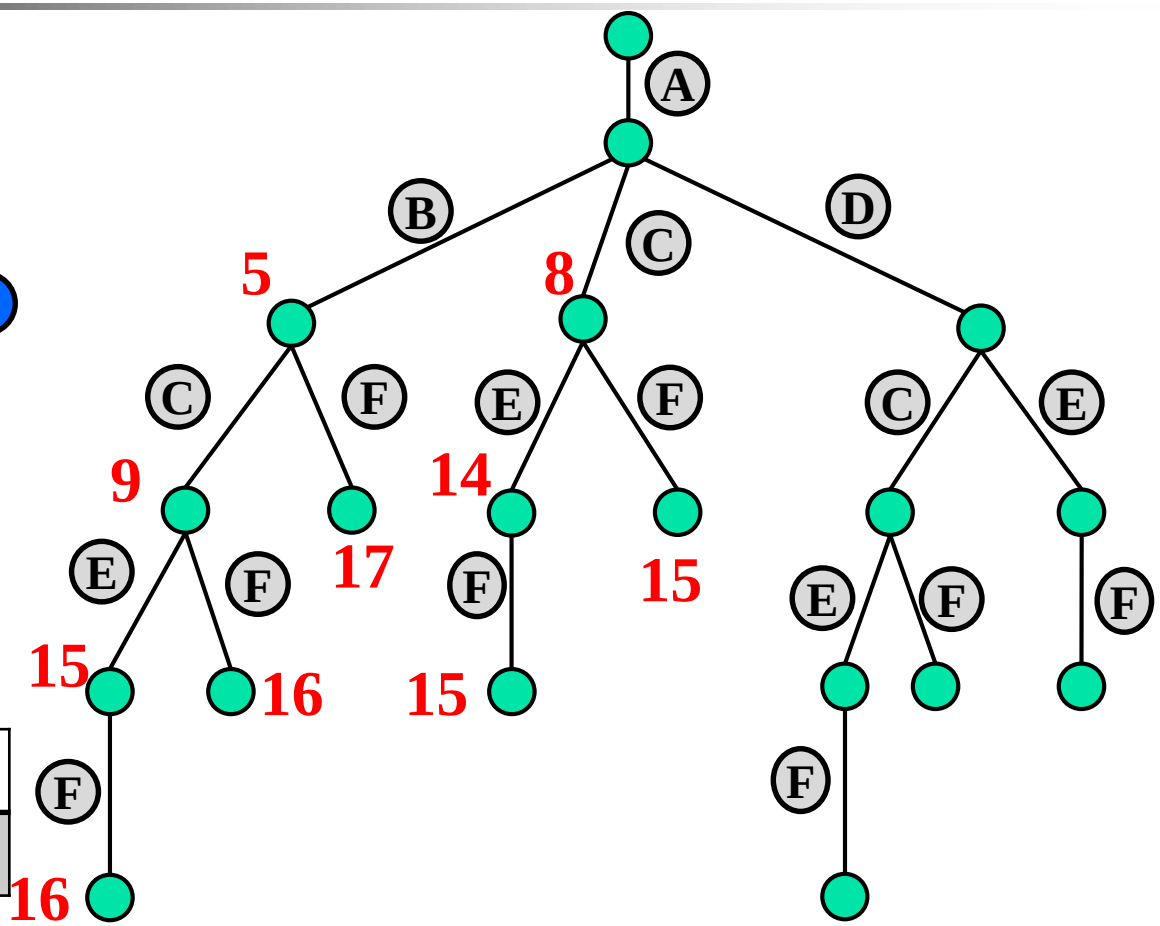


计算距离(A,F)—回溯法

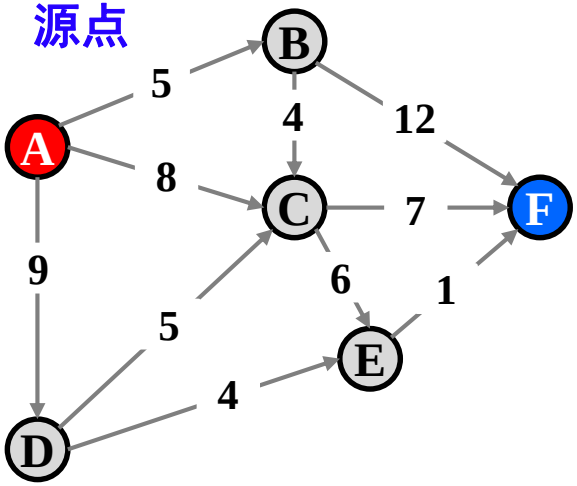


当前最短距离

	B	C	D	E	F
dist	5	8		14	15

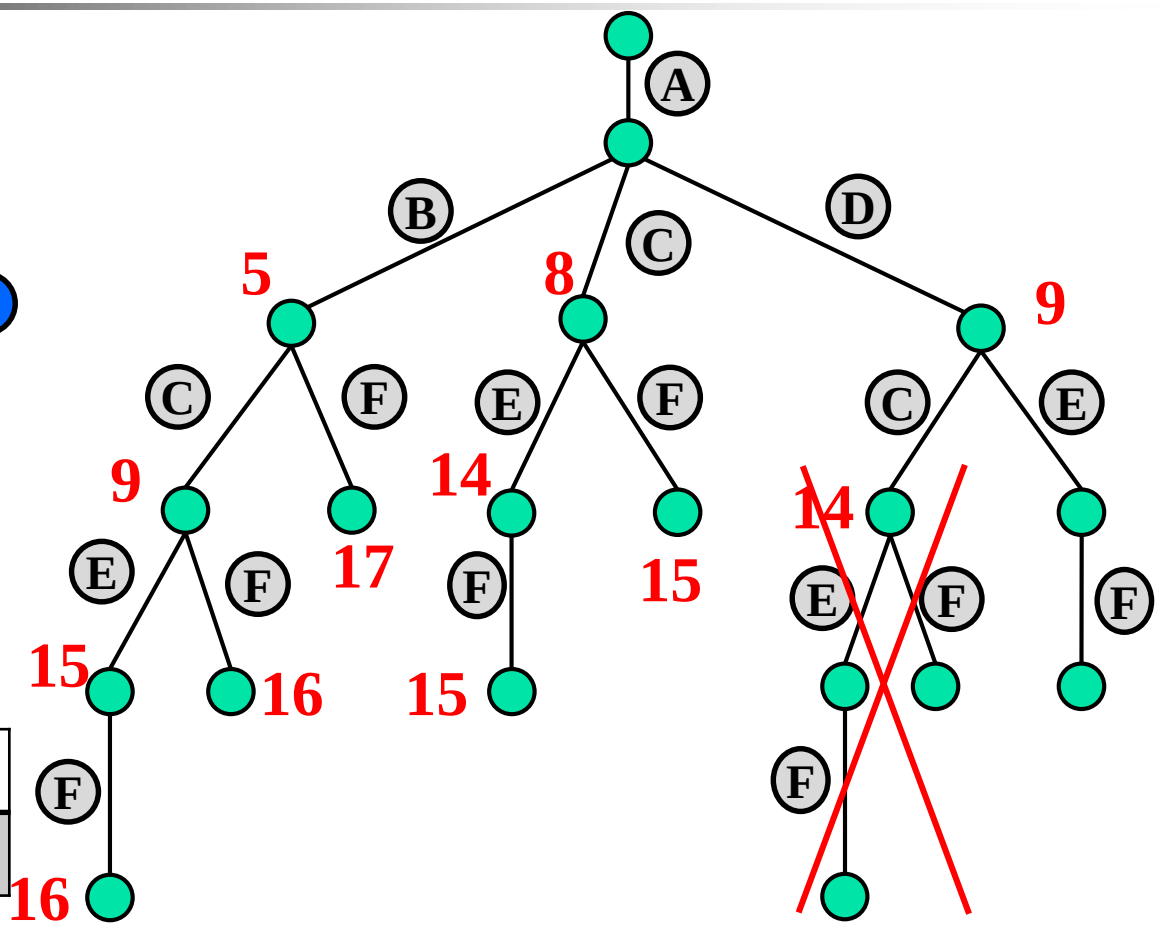


计算距离(A,F)—回溯法

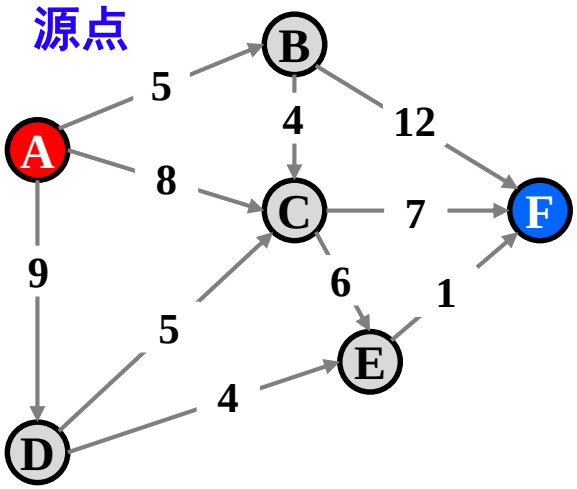


当前最短距离

	B	C	D	E	F
dist	5	8	9	14	15

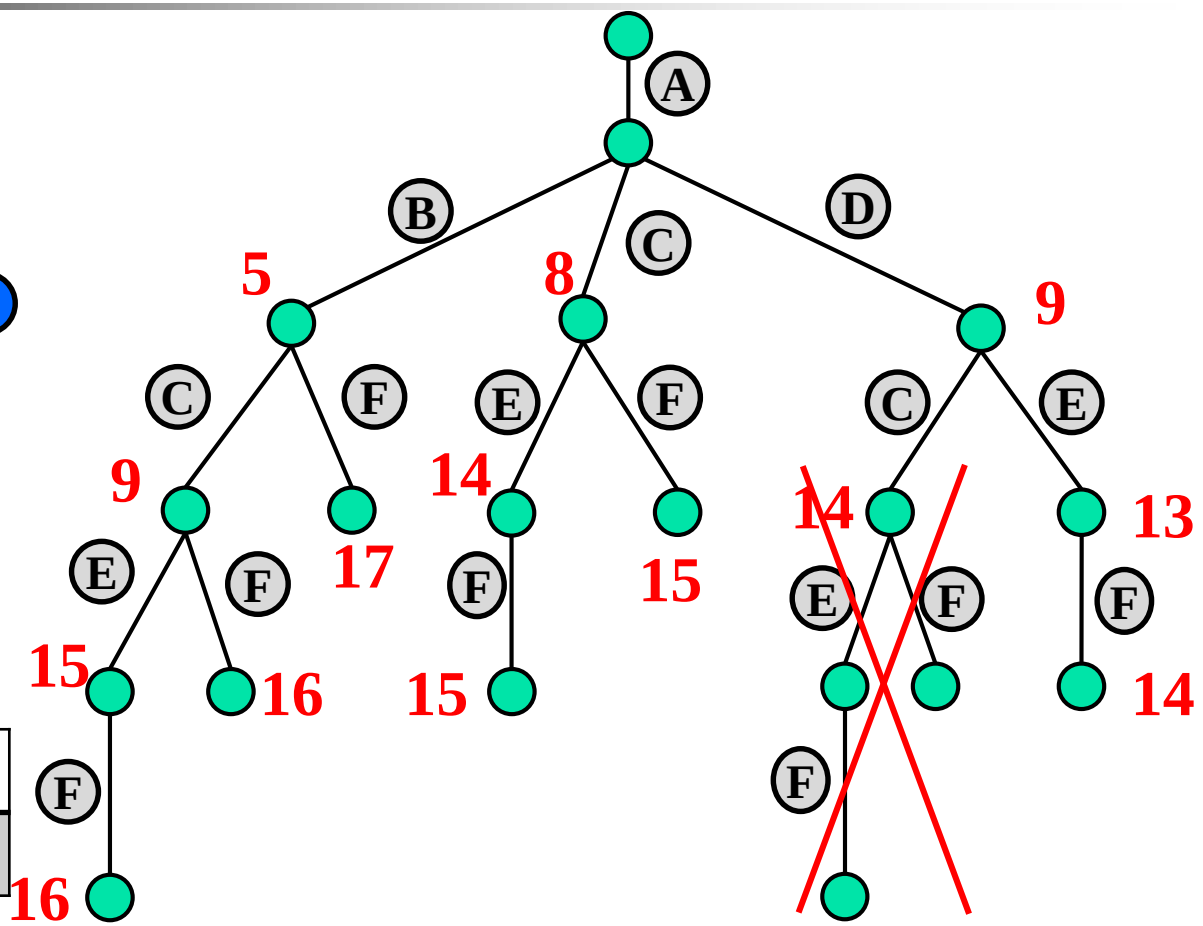


计算距离(A,F)—回溯法



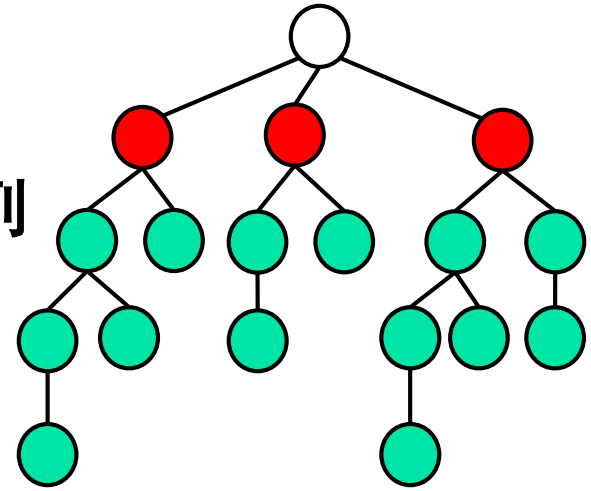
当前最短距离

	B	C	D	E	F
dist	5	8	9	13	14

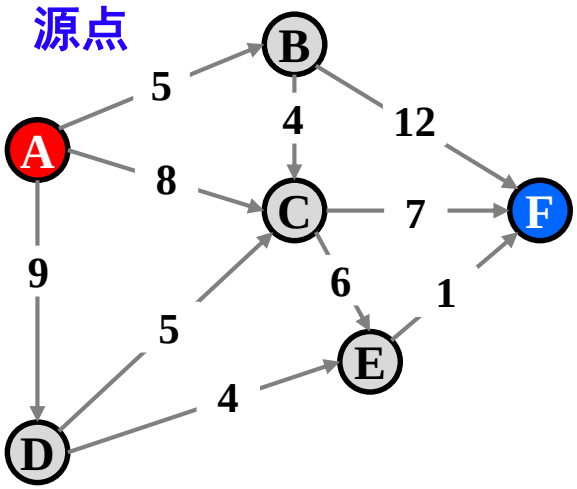


计算距离(A,F)—分支限界法

- 基本思想
  - 从源顶点s和仅含s的优先队列开始
  - 节点队列中选择一节点，并扩展
  - 若节点不被剪枝，将节点插入节点队列
  - 反复2~3步，直到优先队列为空时为止
- 剪枝函数
  - 节点*i*代表顶点*v*的一个距离*dist*(*v*)
  - 若*dist*(*v*)≥ 源点到终点最短距离，则剪枝
  - 若*dist*(*v*)≥ *dist_best*(*v*)，则剪枝
- 节点选择方法
  - 先入先出（FIFO队列）
  - 当前路长最短（优先级队列）



计算距离(A,F)—FIFO队列

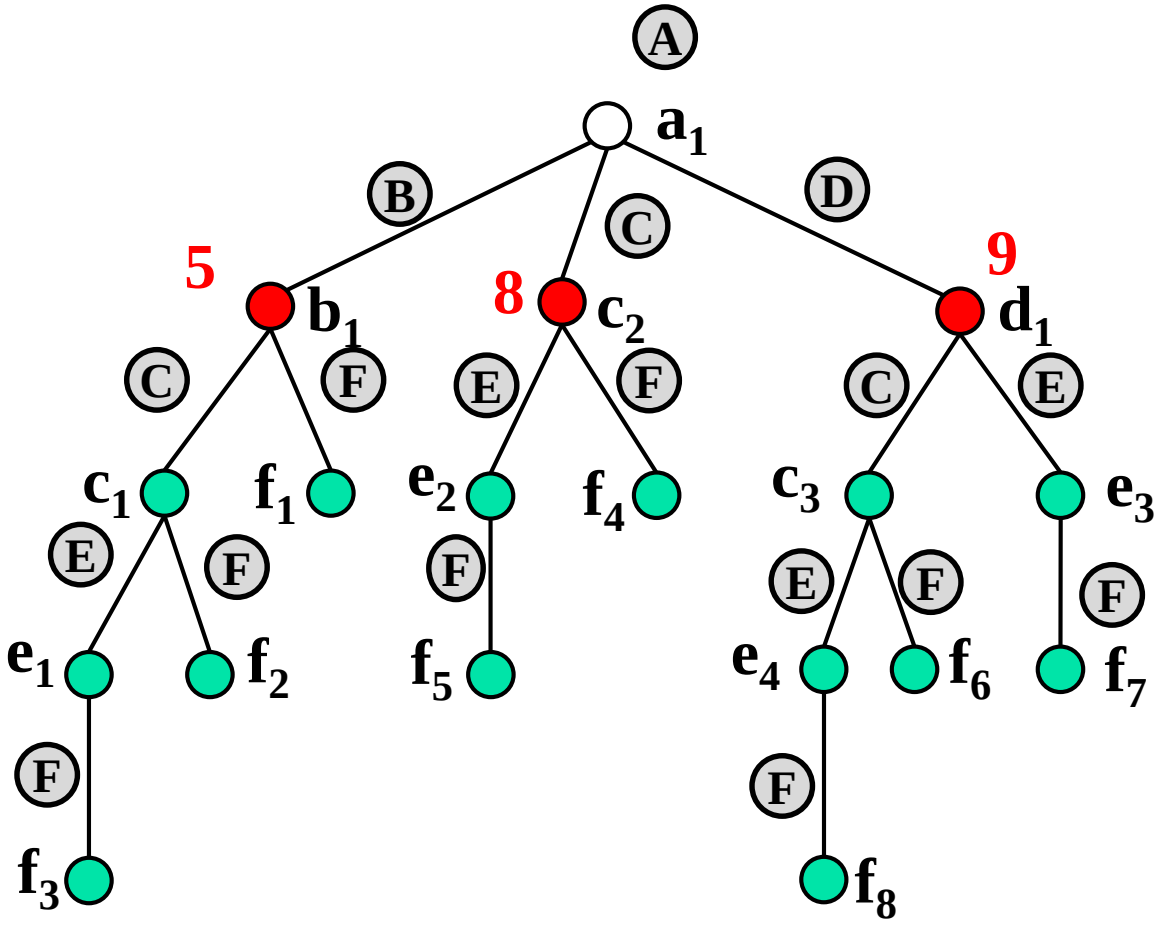


FIFO队列

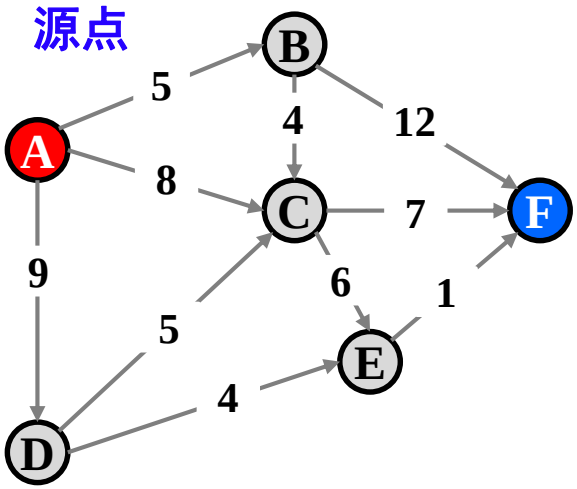
	b_1	c_2	d_1		
dist	5	8	9		

当前最短距离

	B	C	D	E	F
dist	5	8	9		



计算距离(A,F)—FIFO队列

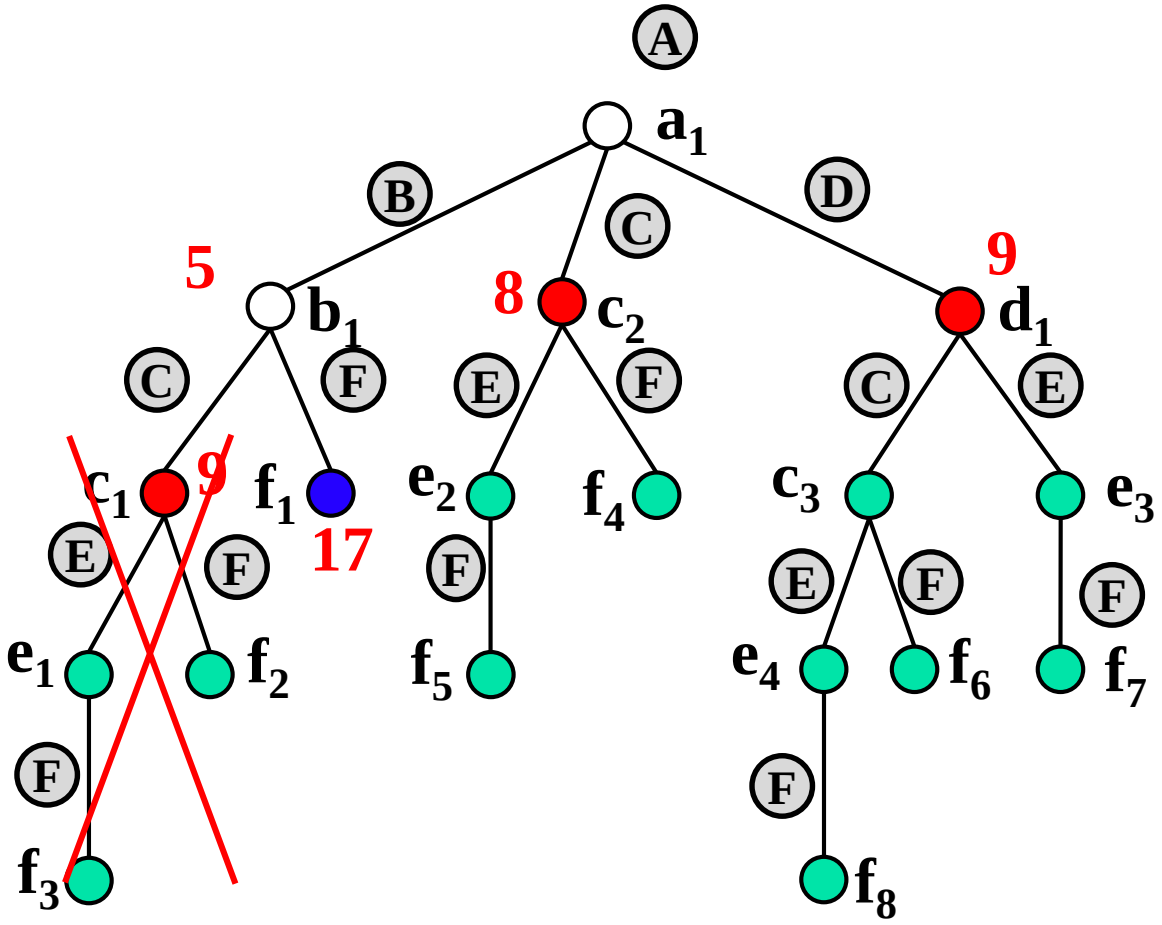


FIFO队列

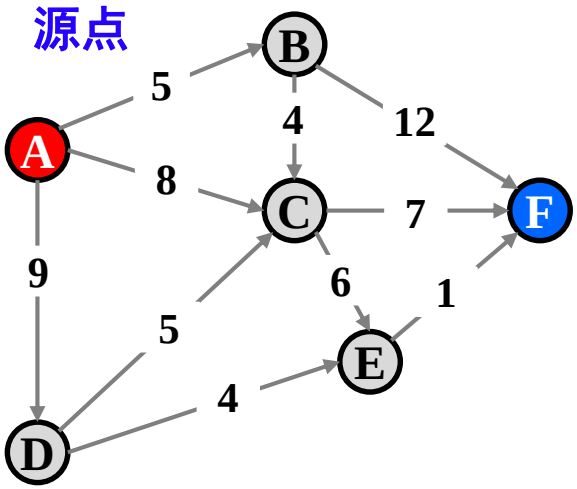
	c_2	d_1			
dist	8	9			

当前最短距离

	B	C	D	E	F
dist	5	8	9		17



计算距离(A,F)—FIFO队列

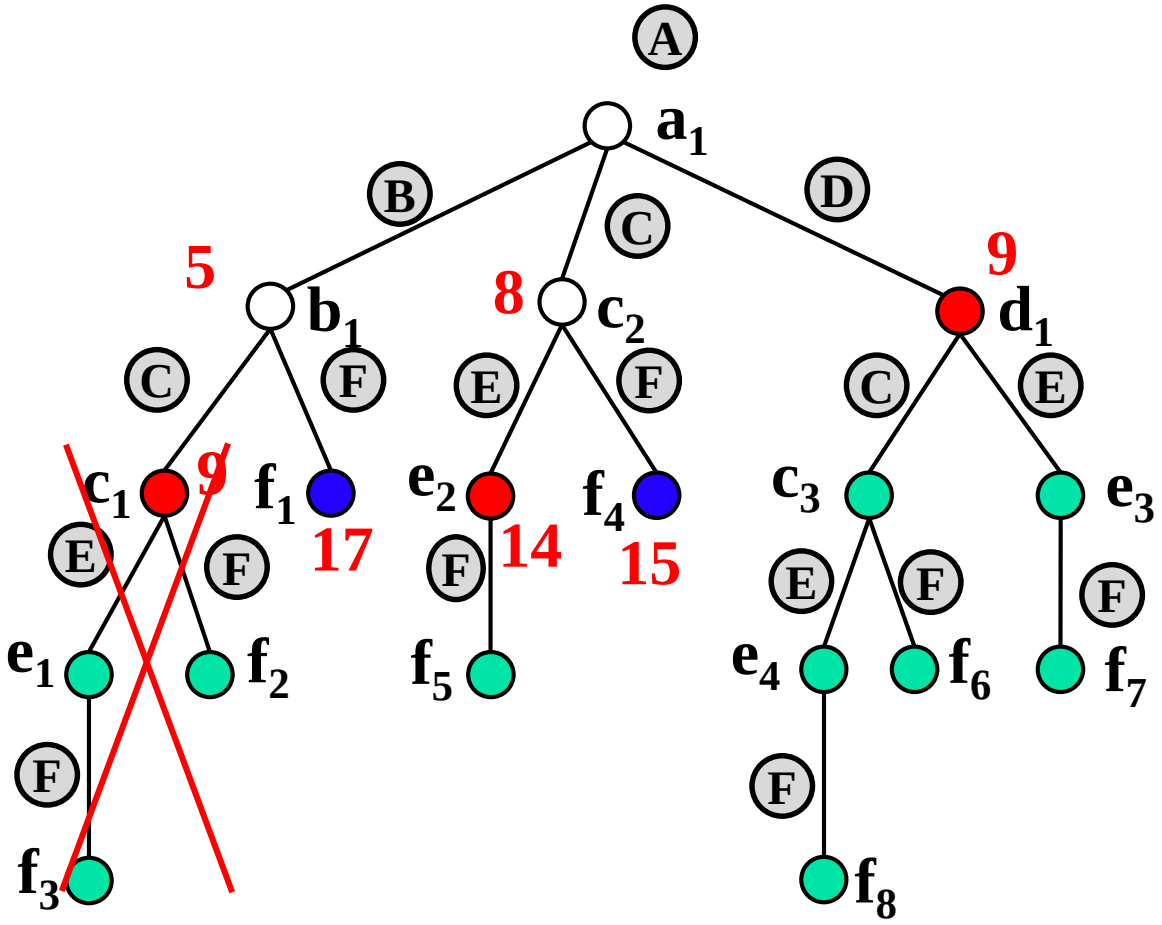


FIFO队列

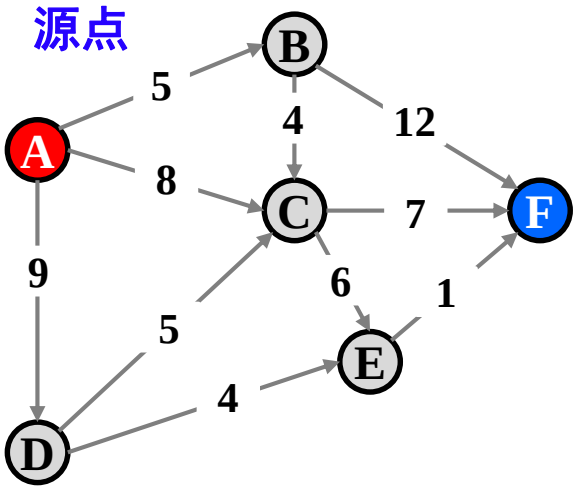
	d_1	e_2			
dist	9	14			

当前最短距离

	B	C	D	E	F
dist	5	8	9	14	15



计算距离(A,F)—FIFO队列

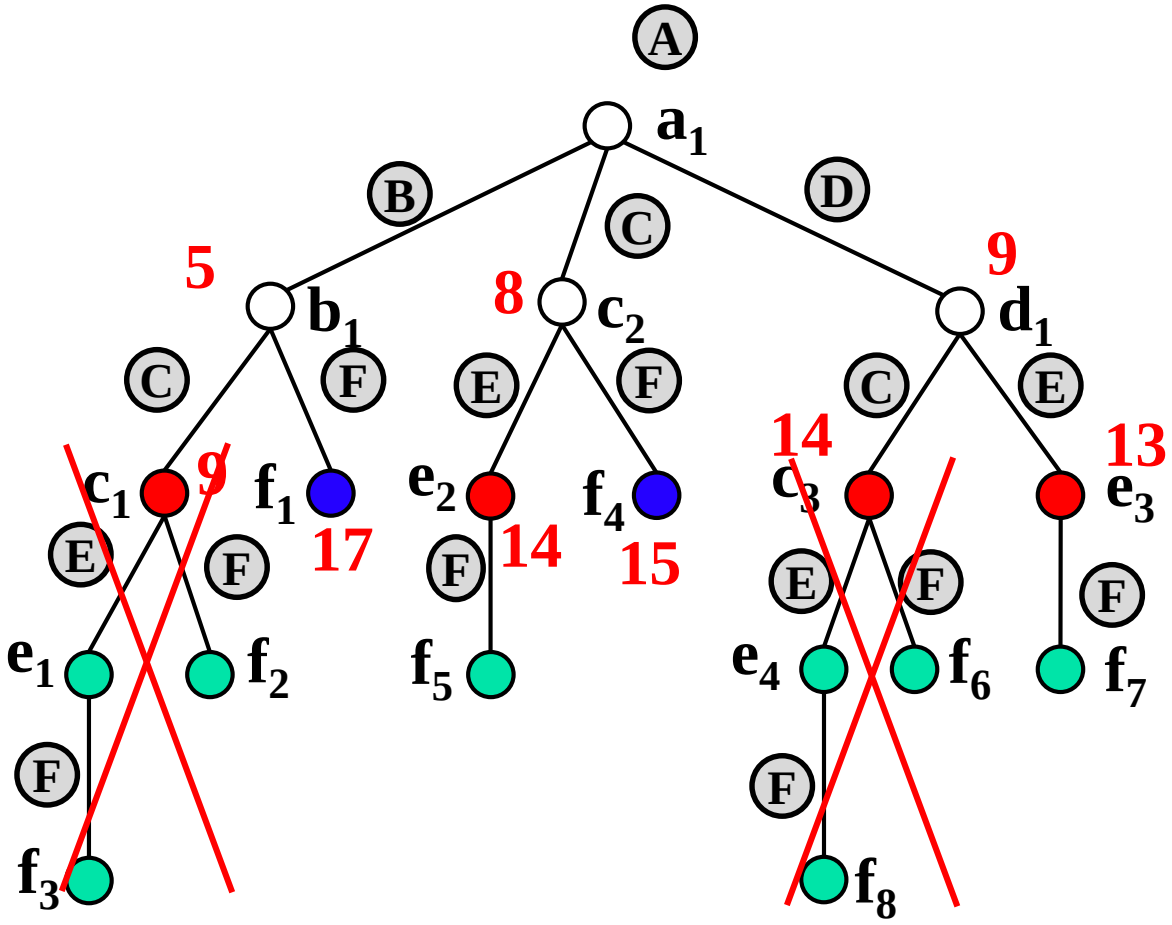


FIFO队列

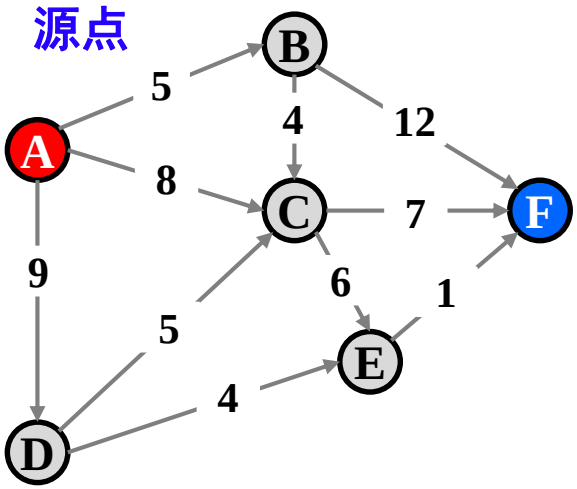
	e_2	e_3			
dist	14	13			

当前最短距离

	B	C	D	E	F
dist	5	8	9	13	15



计算距离(A,F)—FIFO队列

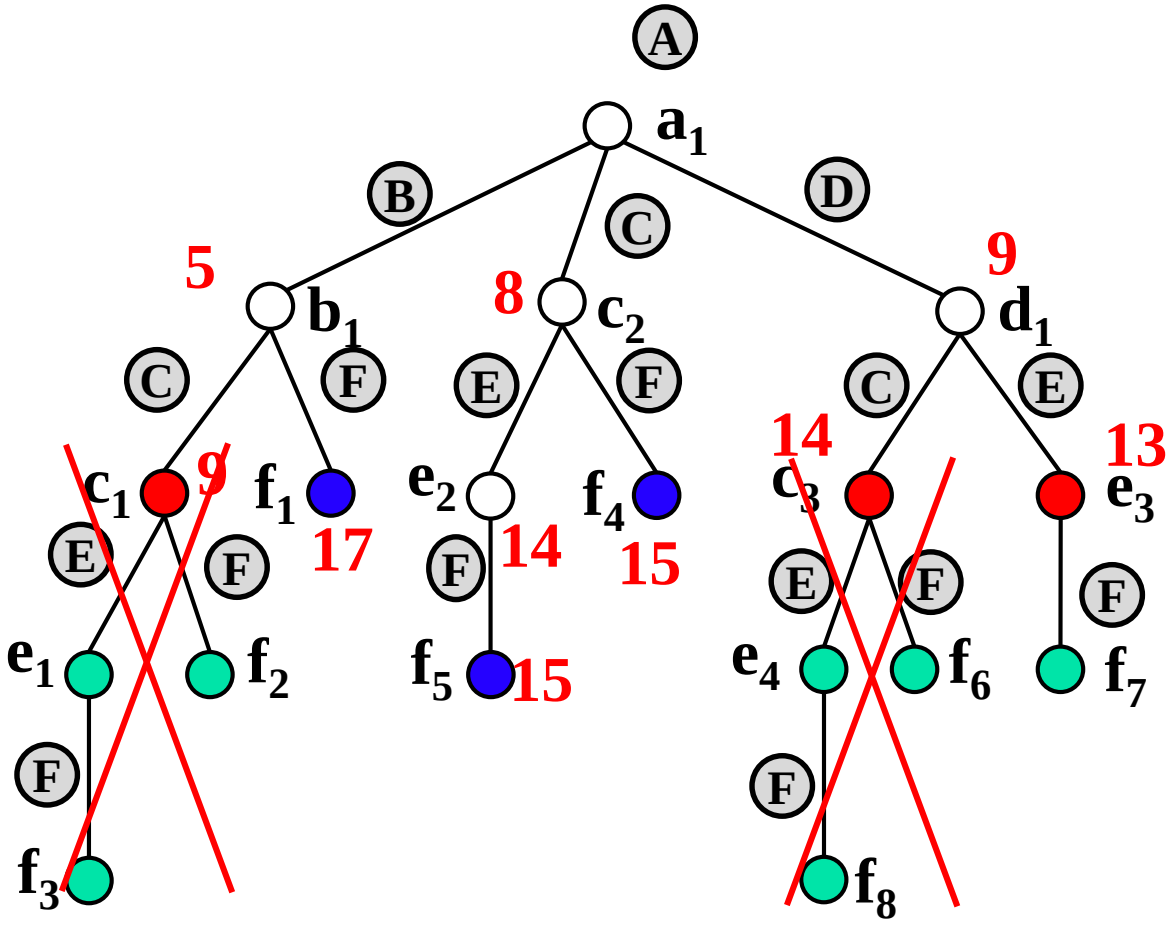


FIFO队列

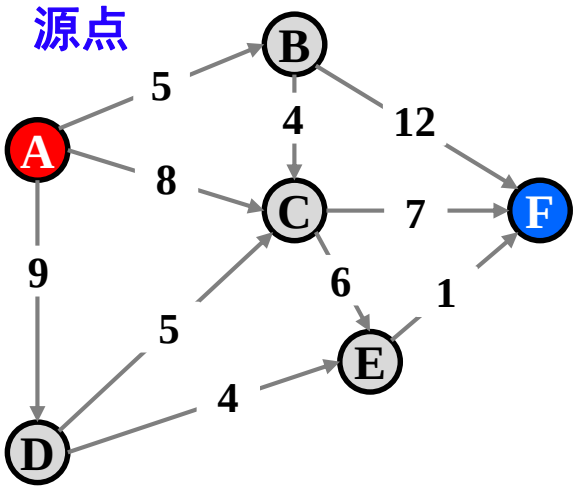
	e_3				
dist	13				

当前最短距离

	B	C	D	E	F
dist	5	8	9	13	15



计算距离(A,F)—FIFO队列

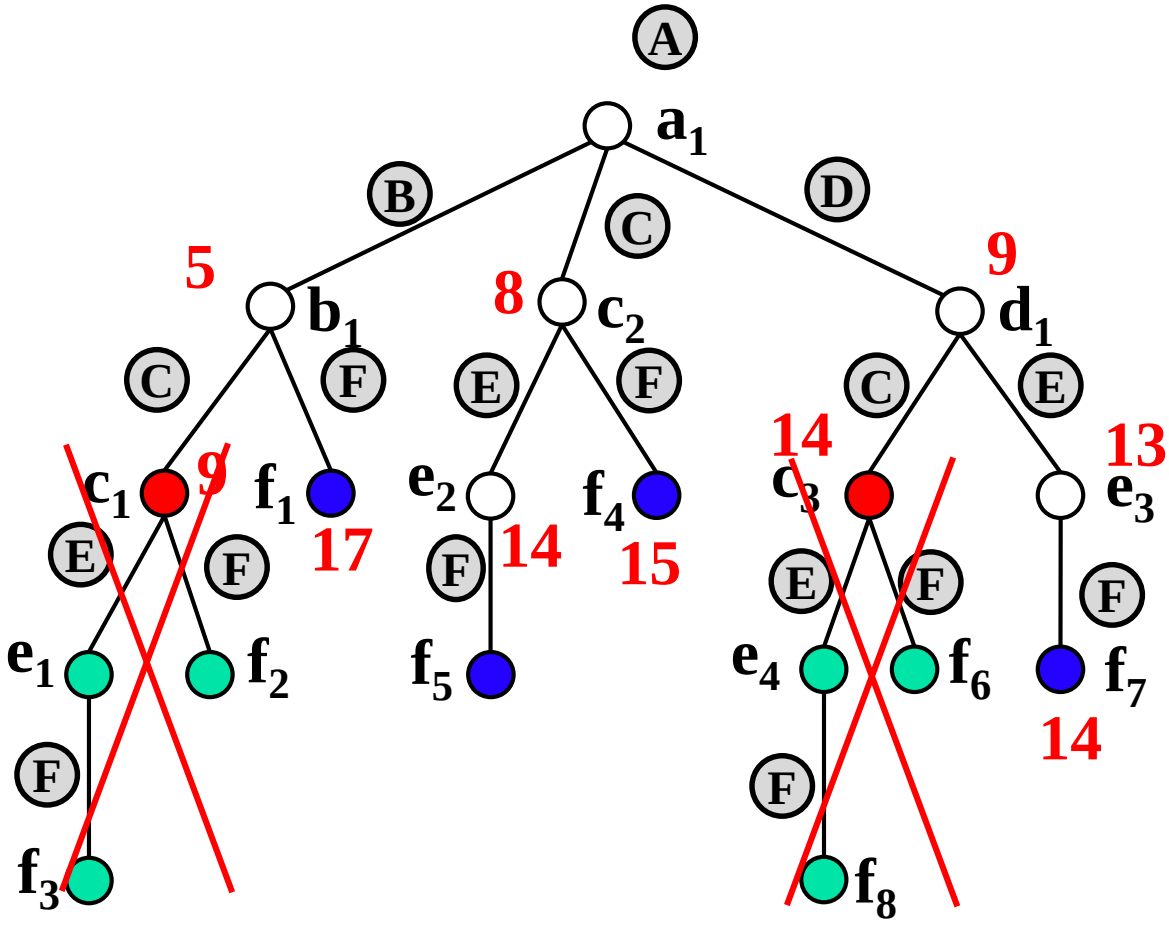


FIFO队列

dist					

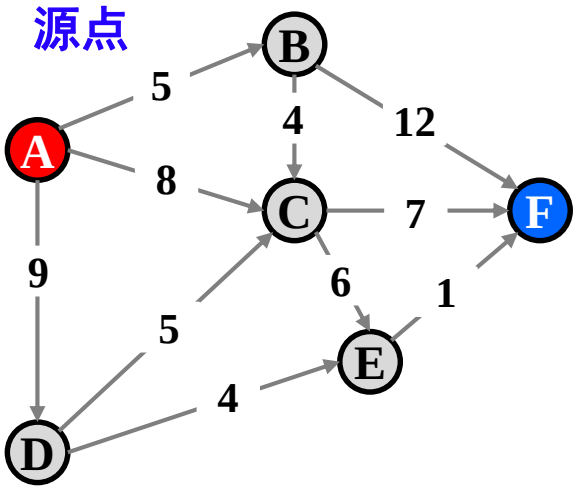
当前最短距离

	B	C	D	E	F
dist	5	8	9	13	14



当FIFO队列空时，
停止分支限界法

计算距离(A,F)—优先级队列

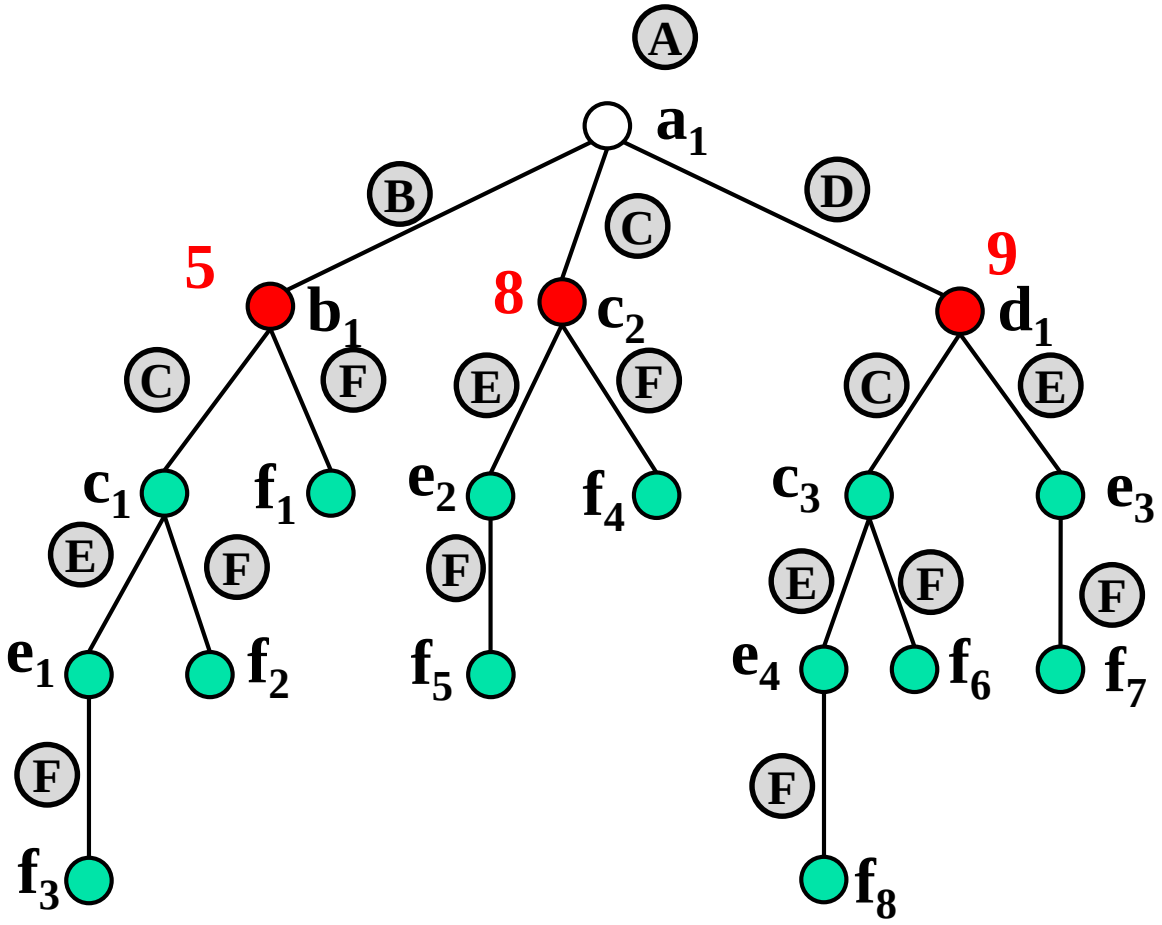


优先级队列

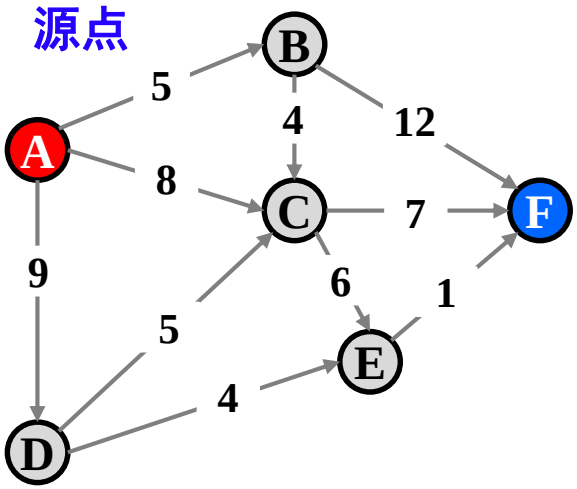
	b_1	c_2	d_1		
dist	5	8	9		

当前最短距离

	B	C	D	E	F
dist	5	8	9		



计算距离(A,F)—优先级队列

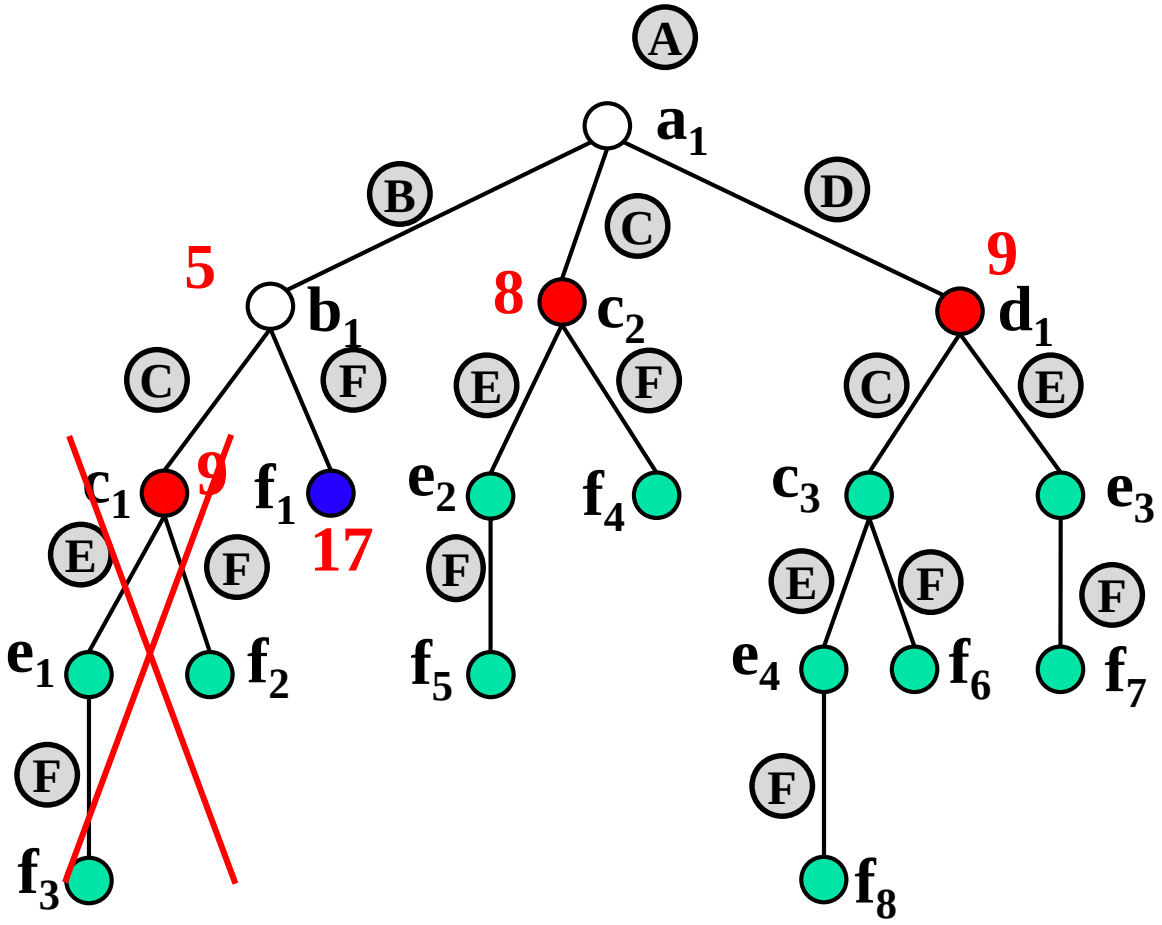


优先级队列

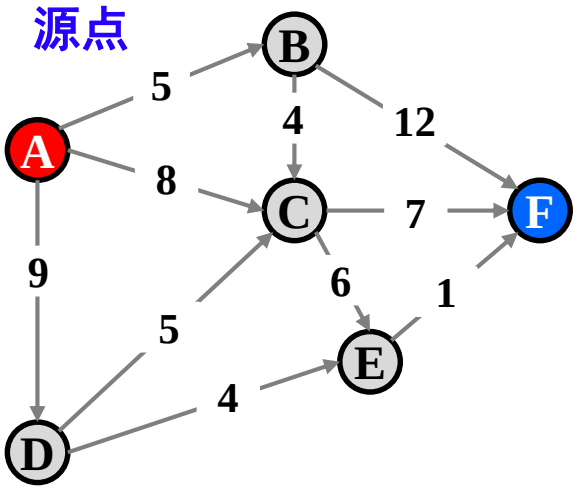
	c_2	d_1			
dist	8	9			

当前最短距离

	B	C	D	E	F
dist	5	8	9		17



计算距离(A,F)—优先级队列

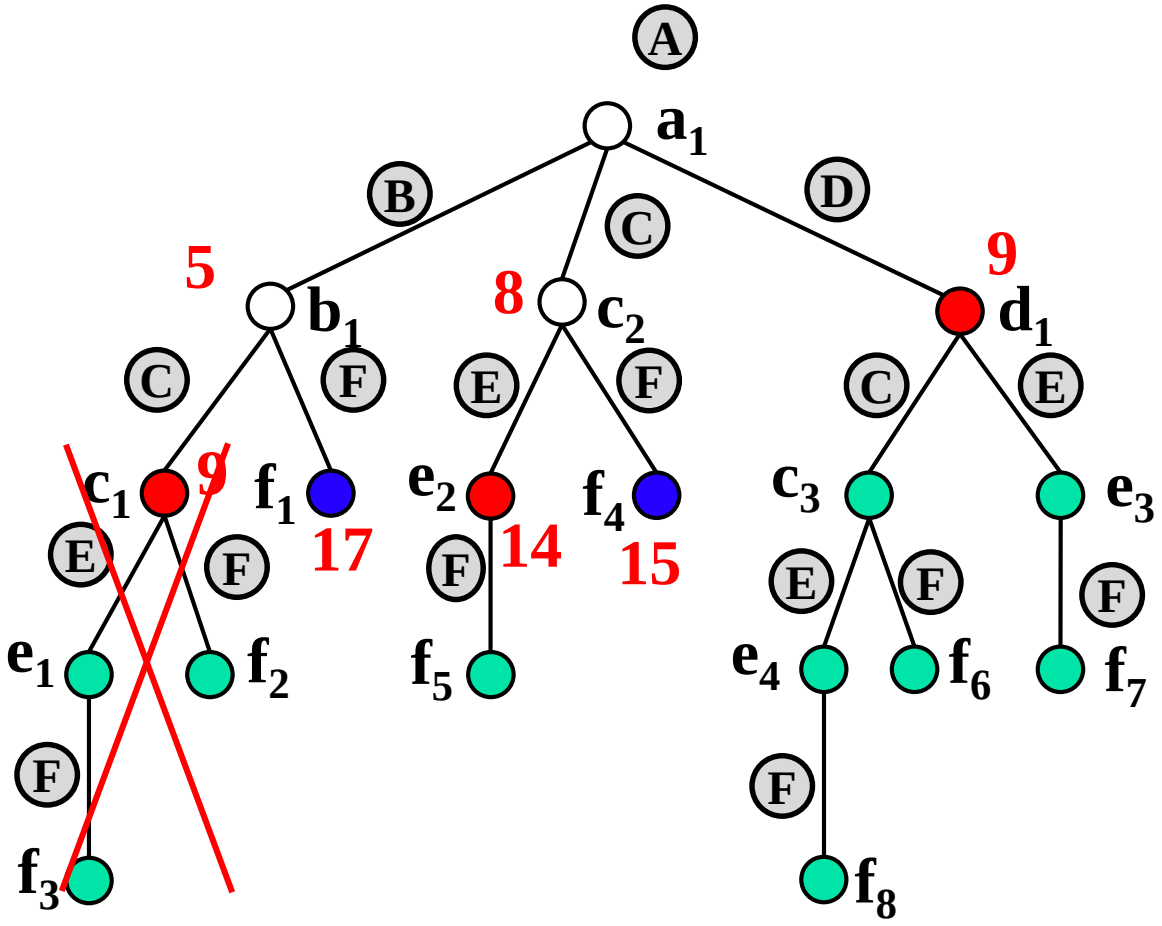


优先级队列

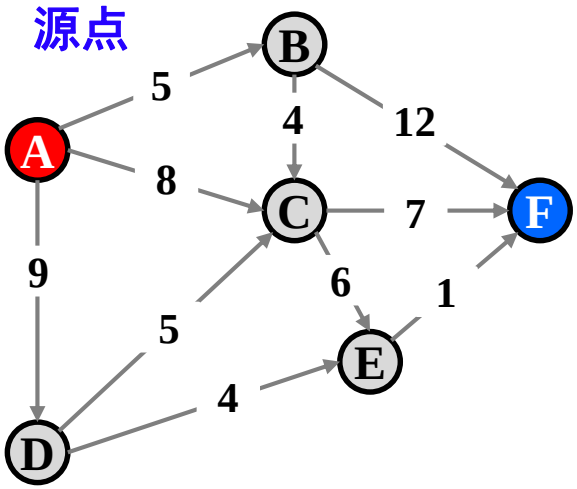
	d_1	e_2			
dist	9	14			

当前最短距离

	B	C	D	E	F
dist	5	8	9	14	15



计算距离(A,F)—优先级队列

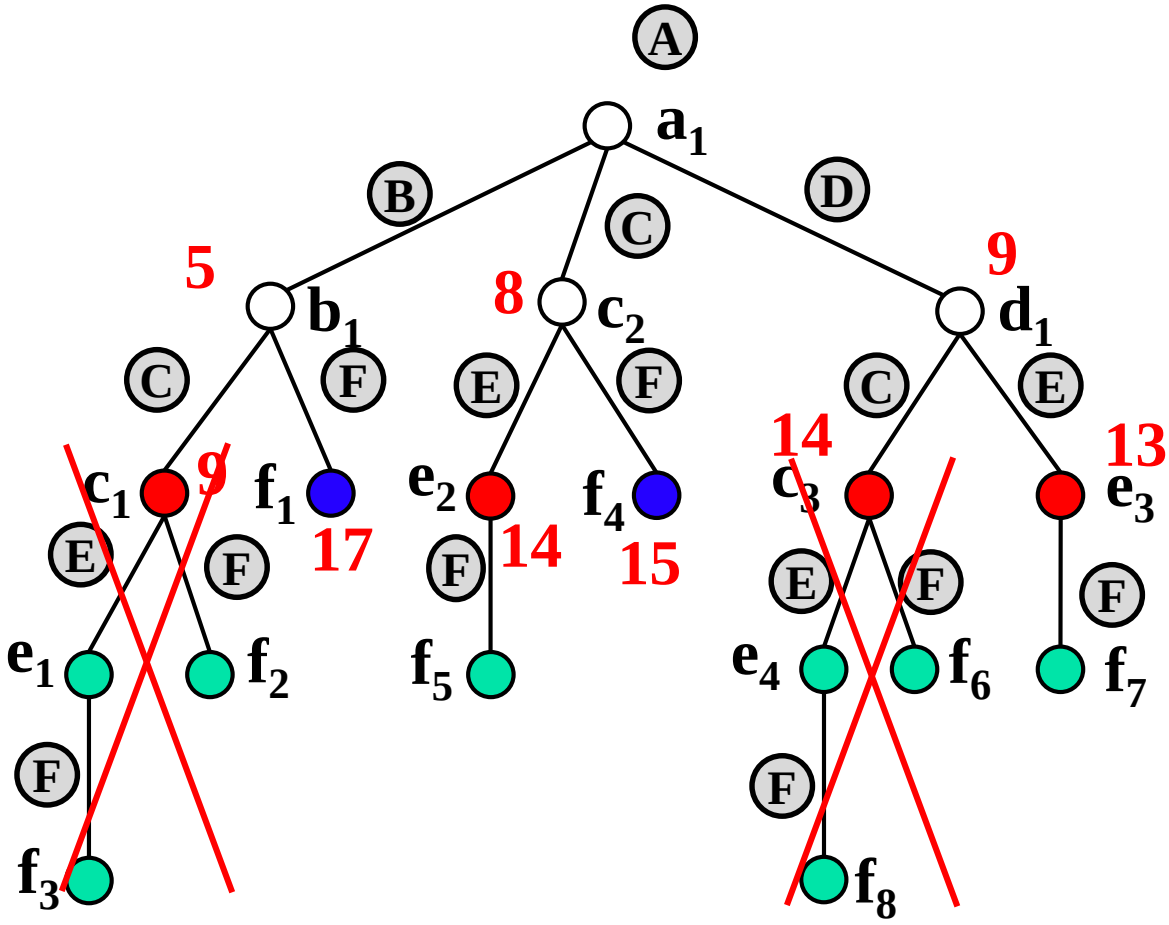


优先级队列

	e_3	e_2			
dist	13	14			

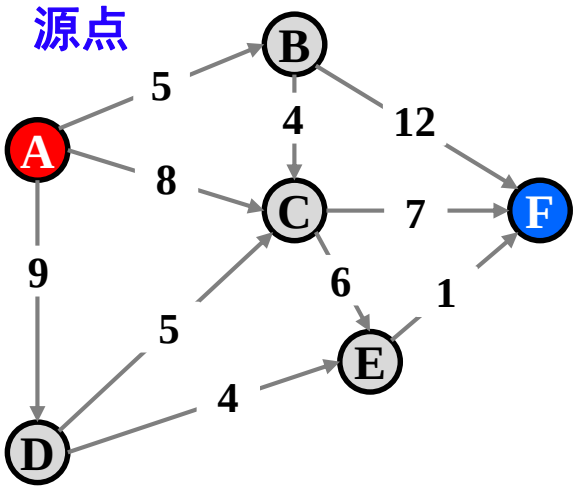
当前最短距离

	B	C	D	E	F
dist	5	8	9	13	15



与FIFO队列区别在于先扩展e_3

计算距离(A,F)—优先级队列

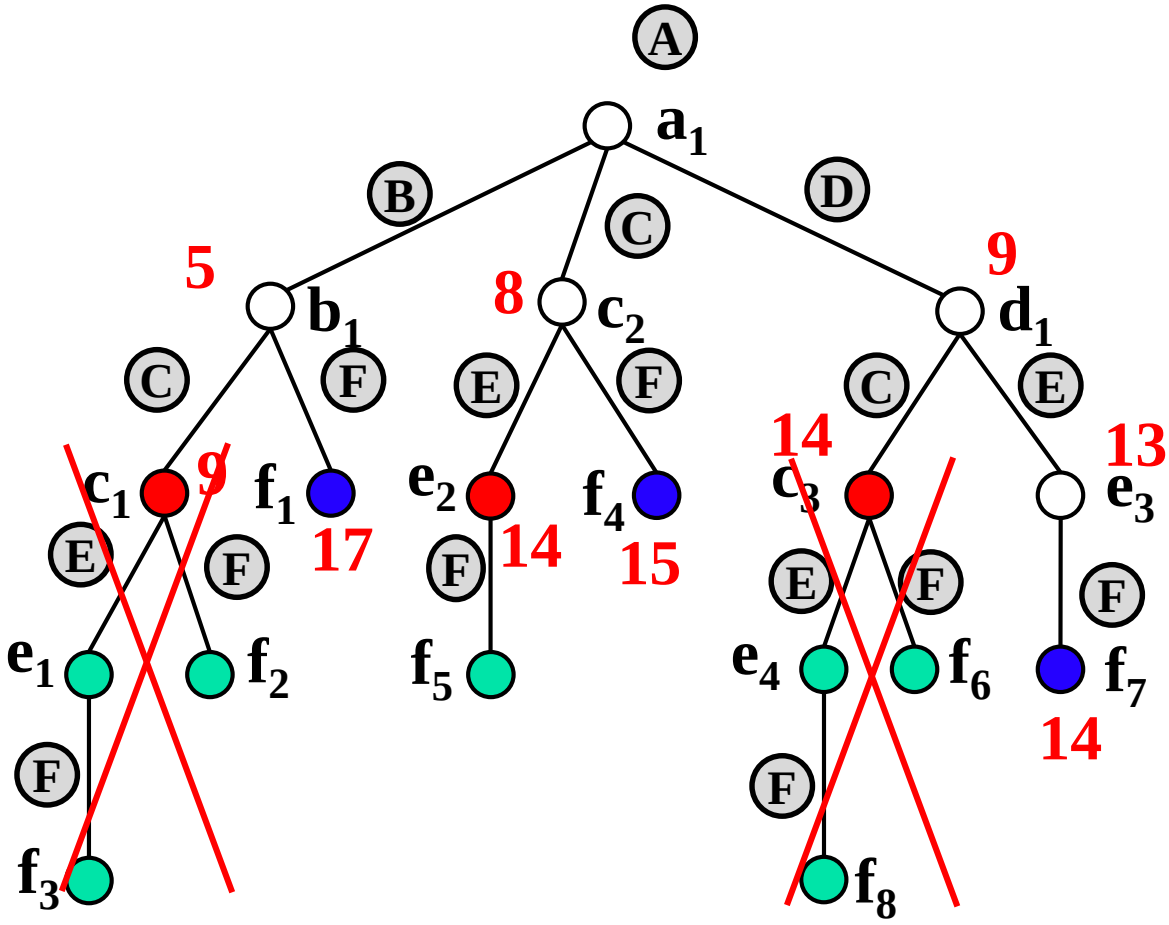


优先级队列

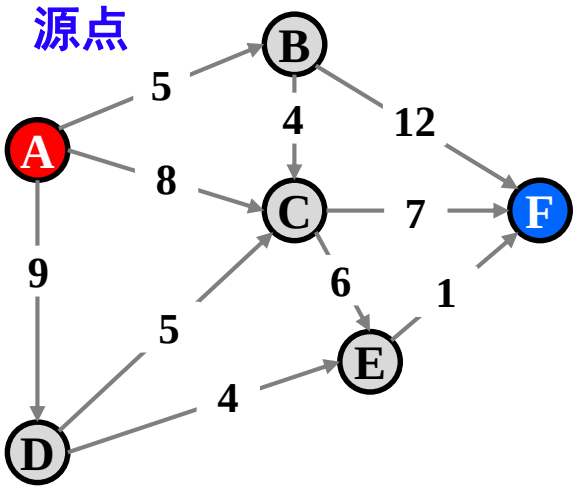
	e_2				
dist	14				

当前最短距离

	B	C	D	E	F
dist	5	8	9	13	14



计算距离(A,F)—优先级队列

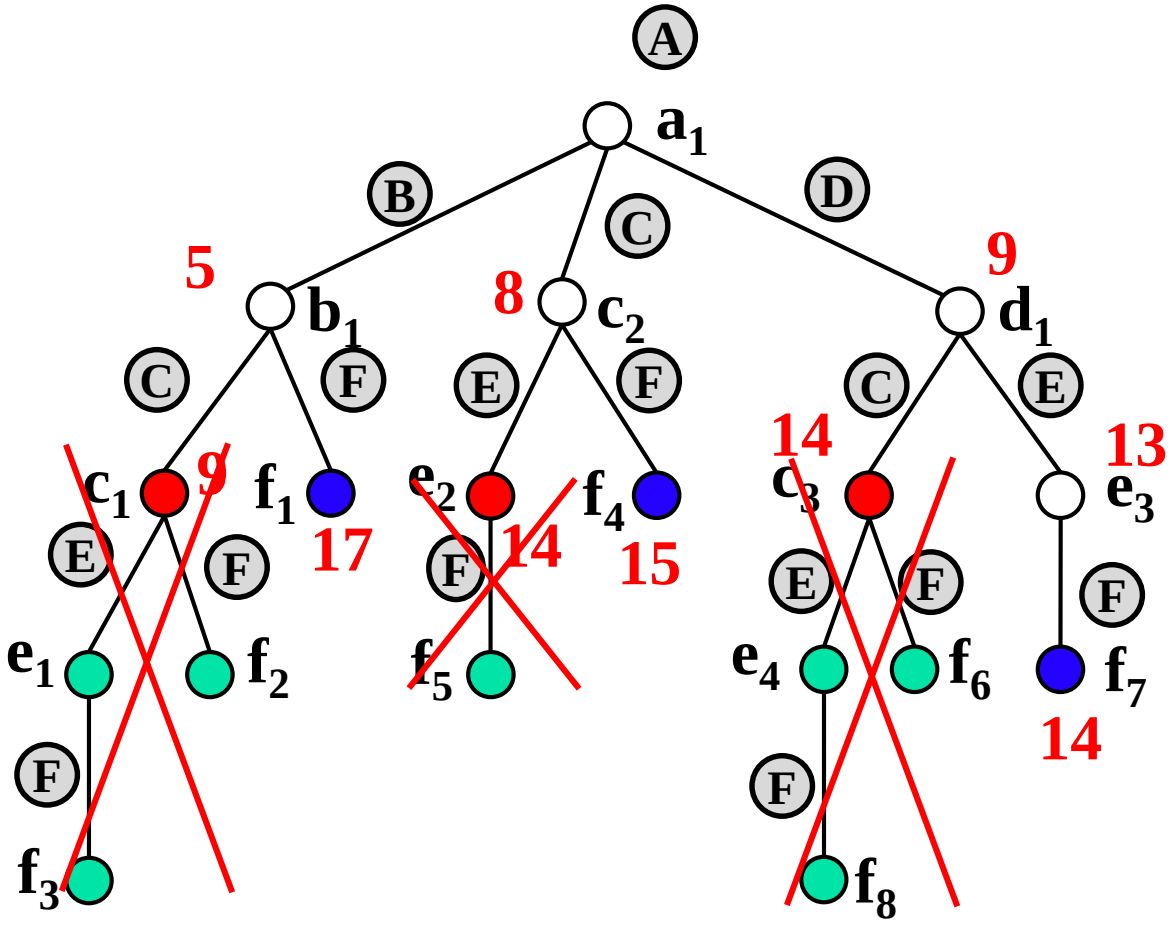


优先级队列

dist					

当前最短距离

	B	C	D	E	F
dist	5	8	9	13	14



当优先级队列空时，
停止分支限界法

最短路径问题—算法的比较

遍历解空间树的方法，如何选择下一个节点

- Dijakstra算法
  - 当前步的最优，复杂度为O(n^2)
  - 不适合设计分布式算法
- 回溯法
  - 当前路径的下一跳(深度优先搜索)
- 分支限界算法
  - 当前图中跳数的最优(FIFO队列，广度优先搜索)
  - 当前距离的最优(优先级队列，优先级搜索)

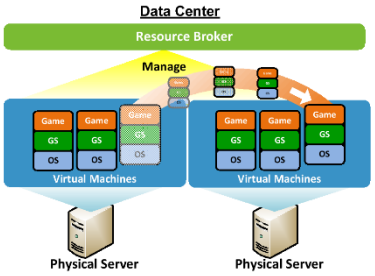
➔相较于Dijakstra算法，分支限界算法更适合设计分布式的单源最短路径算法

装载问题（回顾）

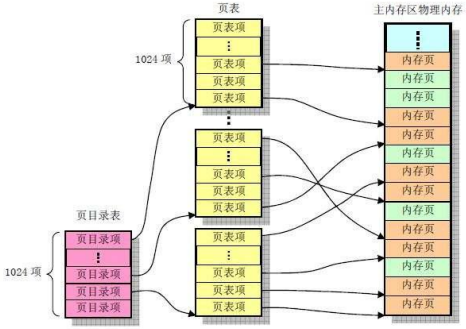
- 有一批共n个集装箱要装上2艘载重量分别为c_1和c_2的轮船，其中集装箱i的重量为w_i，且

$$w_1 + w_2 + \cdots + w_n \le c_1 + c_2$$

- 装载问题要求确定是否有一个合理的装载方案可将这个集装箱装上这2艘轮船。如果有，找出一种装载方案。



云计算虚拟机调度



操作系统内存管理

物料最优剪裁

集装箱最优装载

装载问题的求解思路（回顾）

- **输入：** 集装箱重量W，轮船载重c_1, c_2
  - 首先将第一艘轮船尽可能装满；
  - 将剩余的集装箱装上第二艘轮船。
  - 将第一艘轮船尽可能装满⇔选取全体集装箱的一个子集，使该子集中集装箱重量之和与c_1最接近。
- **实例：**
  - $W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$
  - 最优解$\langle 1,0,1,0,0,1,0 \rangle$

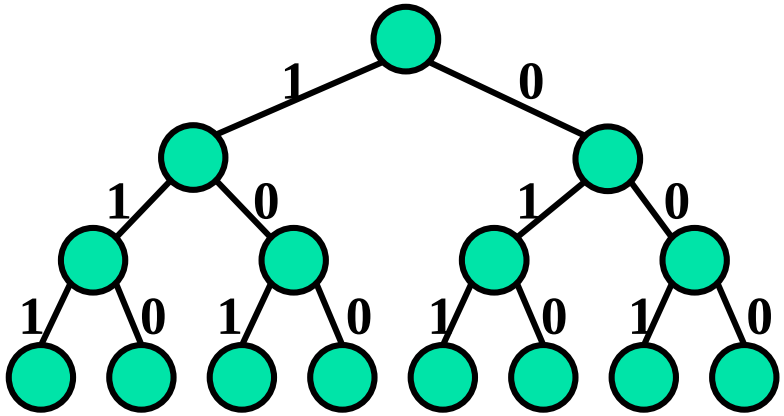
$$\max \sum_{i=1}^{n} w_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c_1$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

装载问题—分支限界法

- 基本思想（与0-1背包问题类似）
  - 从空集∅和仅含空集∅的优先队列开始
  - 选择计算节点队列中代价值最高的节点并扩展
  - 若扩展出节点不被剪枝，将节点插入节点队列
  - 反复2~3步，直到优先队列为空时为止
- 代价函数
  - 当前重量之和+未选中物品的重量之和
- 剪枝函数
  - 与回溯法相同



装载问题—构造最优解

- 回溯法
  - 采用了深度优先搜索
  - 只需要$O(|X|)$空间，$|X|$为解向量长度
- 分支限界法
  - 采用了广度优先搜索
  - 每个活的节点都需要保存解，空间开销大
  - 改进：用指针指向父节点，减少保存公共父节点的开销（书P169）

算法分析与设计

Analysis and Design of Algorithm

Lesson 15

分支限界法与回溯法的区别

- 搜索方式不同
  - 回溯法：深度优先
  - 分支限界法：FIFO队列式(广度优先)、优先级队列式
- 搜索目标不同
  - 回溯法：找所有解、可行解、最优解
  - 分支限界法：找最优解
- 搜索用到的函数不同
  - 回溯法：约束函数
  - 分支限界：约束函数、限界函数、优先级函数
- 实例：背包问题、非对称TSP问题、单源最短路径问题、装载问题、最大团问题、TSP问题

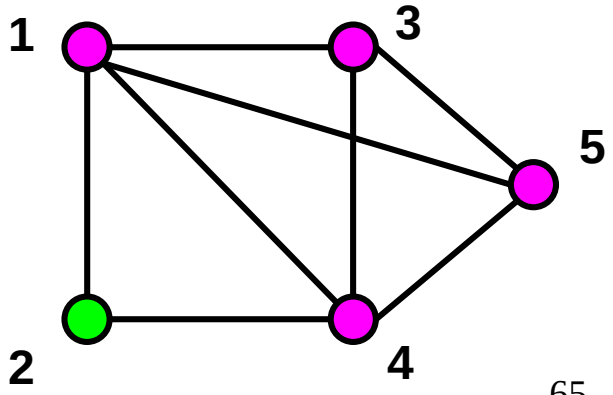
最大团问题

- **问题：** 无向图G=<V,E>，求G的最大团
- G的**子图**：G’=<V’,E’>, V’⊆V,E’⊆E
- G的**补图**：$\overline{G}$=<V,E’>,E’是E关于完全图边集的补集
- G中的**团**：G的完全子图
- G的**最大团**：顶点数最多的团

- **实例：**

团：{1,2,4}, {1,3,4}, {3,4,5}, {1,3,5}, {1,4,5}, {1, 3, 4, 5}

最大团：{1, 3, 4, 5}



独立集与团

- G的**点独立集**：G的顶点子集A，且

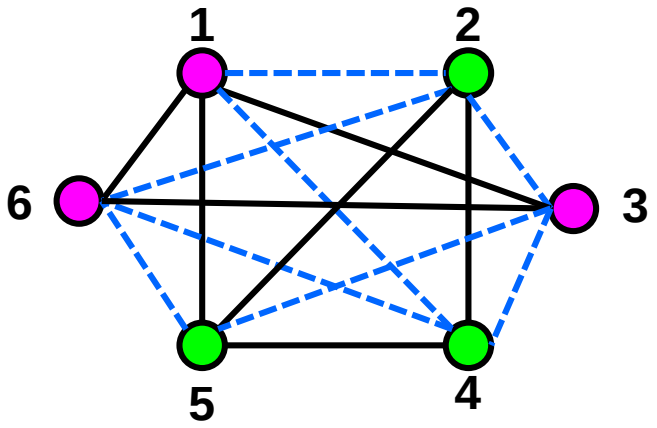
$$\forall u, v \in A, \{u, v\} \notin E$$

- **最大点独立集**：顶点最多的点独立集
- **命题**：U是G的最大团当且仅当U是$\overline{G}$的最大点独立集

G的最大团：

U={1, 3, 6}

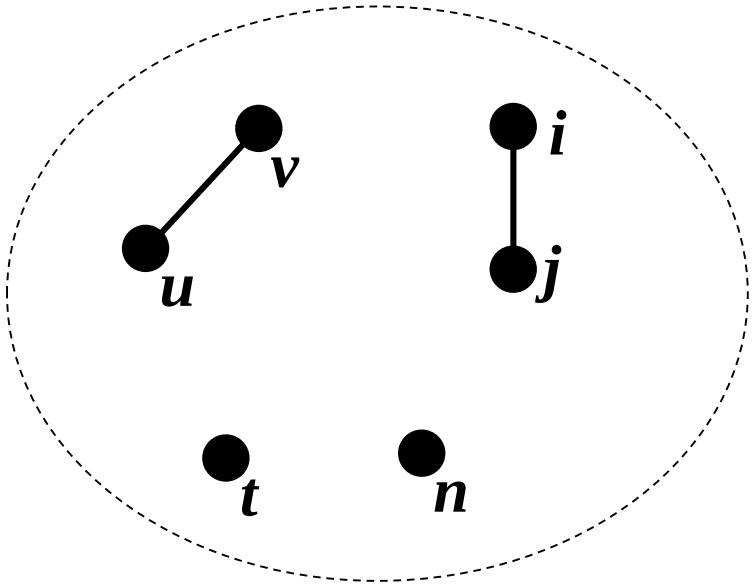
补图$\overline{G}$的最大点独立集



应用

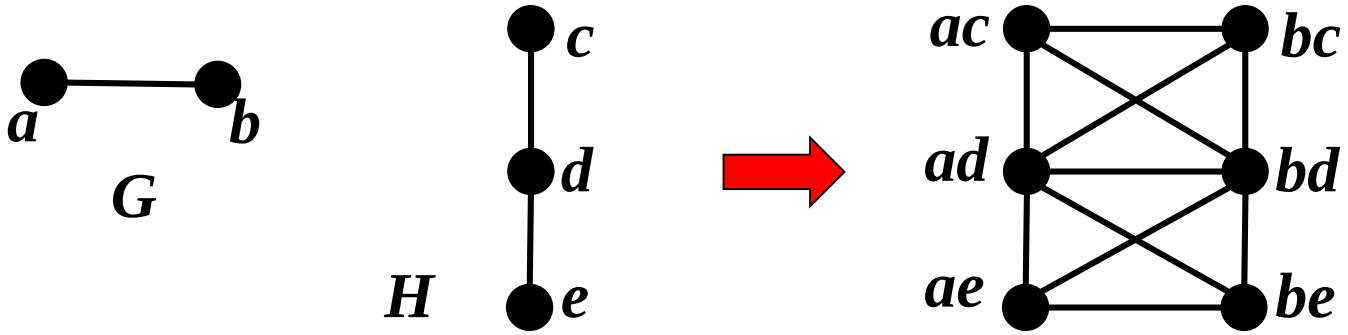
- 编码、故障诊断、计算机视觉、聚类分析、经济学、移动通信、VLSI电路设计。。。
- 例子：噪音使信道传输字符发生混淆
- 混淆图G=<V,E>, V为有穷字符集，{*u*,*v*}∈E⇔*u*和*v*易混淆

G=<V,E>



编码设计

*xy*与*uv*混淆⇔*x*与*u*混淆且*y*与*v*混淆∨*x*=*u*且*y*与*v*混淆∨*x*与*u*混淆且*y*=*v*



*G*与*H*的正规积

为减少噪音干扰，设计编码应该找到混淆图中的最大点独立集

最大团问题

- **问题：** 给定无向图G=<V,E>，其中顶点集V={1,2, ..., *n*}，边集为E，求G的最大团。
- **解：** $\langle x_1, x_2, \ldots, x_n\rangle$为0-1向量，$x_k$=1当且仅当顶点*k*属于最大团
- **穷举法：** 对每个顶点子集，检查是否构成团，即其中每对顶点之间是否都有边。有2^n个子集，至少需要指数时间。

分支限界算法设计

- 搜索数为子集树
- 结点$\langle x_1, x_2, \ldots, x_k\rangle$的含义：
  - 已检索顶点1, 2, ..., k，其中x_i=1表示顶点i在当前的团内
- 约束条件：该顶点与当前团内每个顶点都有边相连
- 界：当前已检索到的极大团的顶点数

代价函数

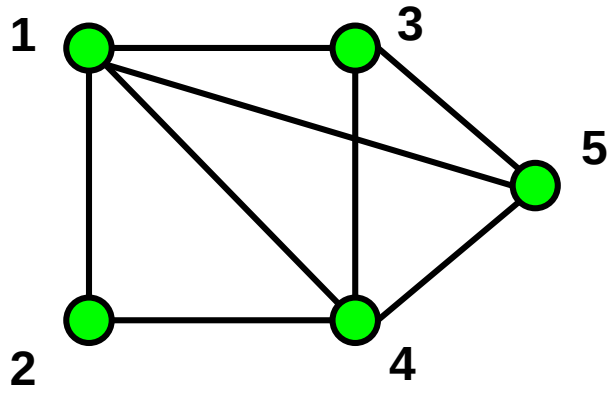
- **代价函数：** 目前的团可能扩张为极大团的顶点数上界

$$F=C_k+n-k$$

其中C_k为当前团的顶点数（初始为0），k为结点层数

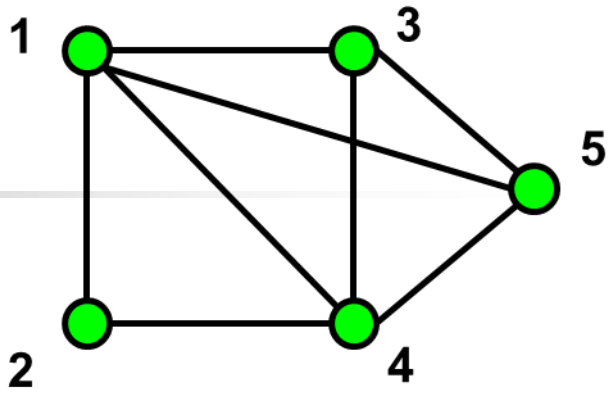
最坏情况下时间： $O(n2^n)$

实例



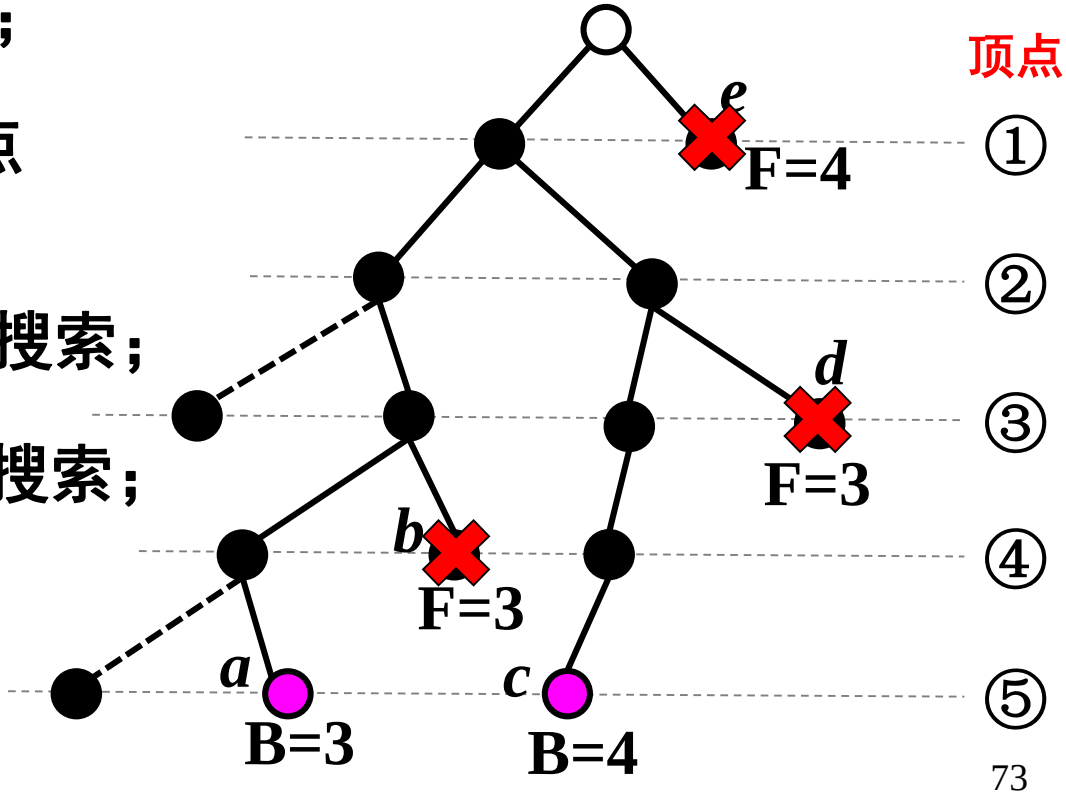
- 顶点编号顺序为1, 2, 3, 4, 5
- 对应x_1, x_2, x_3, x_4, x_5
- x_i=1当且仅当i在团内
- 分支规定左子树为1，右子树为0
- B为界，F为代价函数值

搜索树



- *a*: 极大团{1, 2, 4}，顶点数为3，界为B=3
- *b*: 代价函数值F=3，回溯；
- *c*: 极大团{1, 3, 4, 5}，顶点数为4，界为B=4
- *d*: 代价函数值F=3，不必搜索；
- *e*: 代价函数值F=3，不必搜索；

输出最大团{1, 3, 4, 5}
顶点数为4



TSP问题

输入： 城市集$C=\{c_1,c_2,\ldots,c_n\}$,距离$d(c_i,c_j)=d(c_j,c_i)$

解： $1,2,\ldots,n$的排列$k_1,k_2,\ldots,k_n$使得

$$\min\left\{\sum_{i=1}^{n-1} d(c_{k_i},c_{k_{i+1}})+d(c_{k_n},c_{k_1})\right\}$$

算法设计

- **解向量**为：$\langle 1, i_1, i_2, \ldots, i_{n-1} \rangle$，其中$i_1, i_2, \ldots, i_{n-1}$为$\{2, 3, \ldots, n\}$的排列
- 搜索空间为**排列树**，结点$\langle i_1, i_2, \ldots, i_k \rangle$表示得到 k步路线
- **约束条件**：令$O=\{i_1, i_2, \ldots, i_k\}$则$i_{k+1} \in \{2, 3, \ldots, n\}-O$，即每个结点只能访问一次

代价函数与界

- **界：** 当前得到的最短巡回路线长度
- **代价函数：** 设顶点c_i出发的最短边长度为l_i，d_j为选定巡回路线中第j段的长度

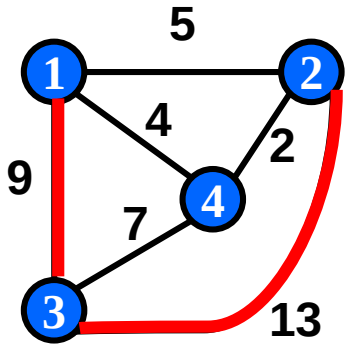
$$L = \sum_{j=1}^{k} d_j + l_{i_k} + \sum_{i_j \notin B} l_{i_j}$$

已走过路径长

剩余长度下界

代价函数

$$L=\sum_{j=1}^{k} d_j + l_{i_k} + \sum_{i_j \notin B} l_{i_j}$$

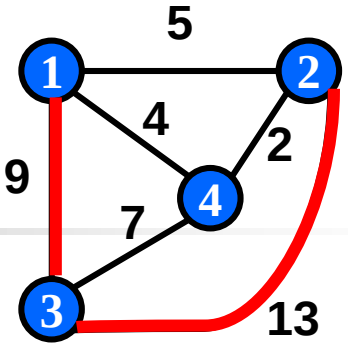


部分路线<1, 3, 2>

L=9+13+2+2=26

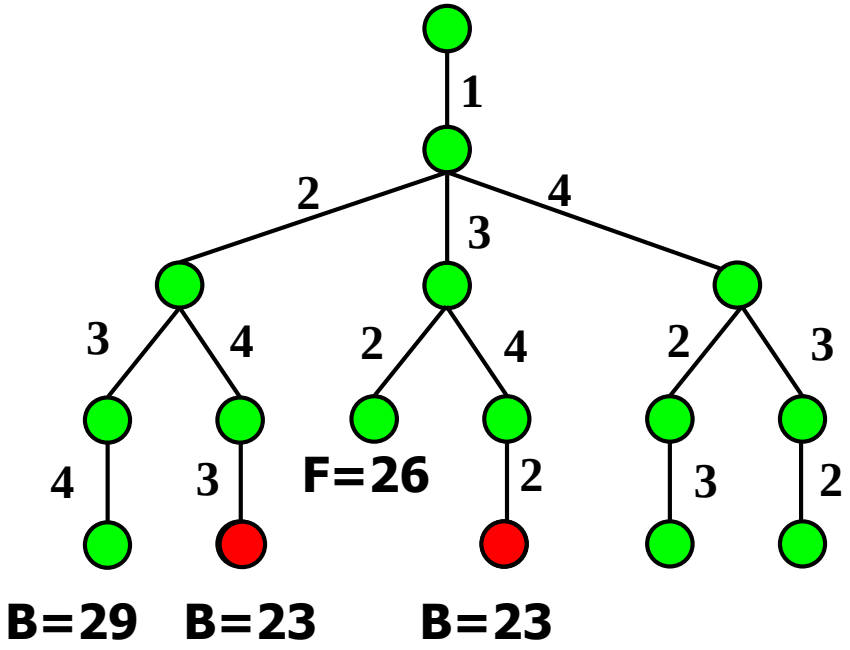
- **9+13为走过的路径长度**
- **后两项分别为从结点2及结点4出发的最短边长**

搜索树



深度优先遍历搜索树

- 第1个界：<1,2,3,4>, B=29
- 第2个界：<1,2,4,3>, B=23
- 结点<1,3,2>:代价函数值26>23,不再搜索,返回<1,3>,右子树向下
- 结点<1,3,4>,代价函数值9+7+2+2=20<23，继续，得到可行解<1,3,4,2>,长度23
- 回溯到结点<1>,沿<1,4>向下
- …



➔最优解：<1,2,4,3>或<1,3,4,2>,长度23

算法分析

- 搜索树的树叶个数：O((*n*-1)!)，每片树叶对应1条路径，每条路径有个*n*个结点
- 每个结点代价函数计算时间O(1)，每条路径的计算时间O(*n*)
- 最坏情况下算法的时间O(*n*!)

第六章小结

- 分支限界：一种与回溯法类似的算法
  - 将问题建模为解空间树
  - 通常用代价函数估算每个分支的最优值
  - 优先选择当前看来最好的分支
  - 搜索策略一般采用广度优先搜索
  - 搜索过程中剪枝
- 分支限界的剪枝函数
  - 不满足约束条件
  - 代价函数值不优于当前的界