第7讲 时空关联规划

尹慧琳，yinhuilin@tongji.edu.cn
同济大学 电子与信息工程学院

第 7 讲 时空关联规划

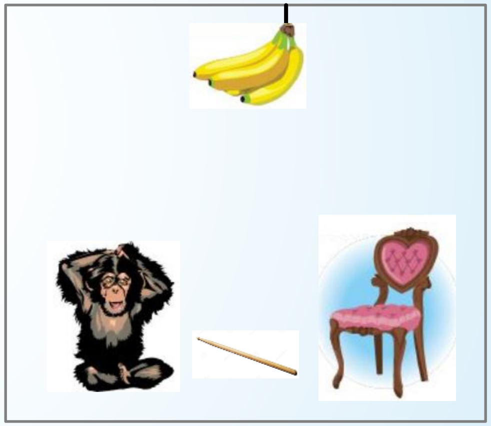
- 规划（Planning）是制定一套动作的计划来达到既定的目标。
- 时空关联规划，指的是与状态空间和时间相关联的规划，包含：
  - 空间规划（Space planning）
  - 时序规划（Temporal planning）
  - 调度（Scheduling）
  - 运动规划（Motion planning）。
- 其中，空间规划又可分为：
  - 经典规划（Classical planning）
  - 新经典规划（Neoclassical planning）

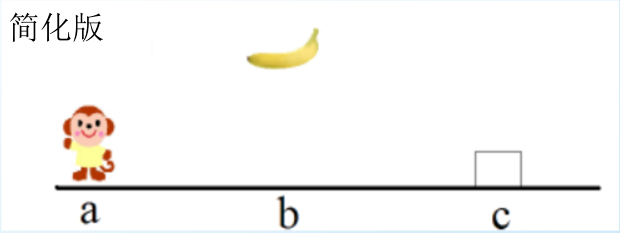
第 7 讲 时空关联规划

第7讲 时空关联规划

7.1 规划的基本概念

- 猴子与香蕉问题（Monkey and banana problem）
  - 目标：房间里有只猴子，天花板上悬挂着一串香蕉，猴子要摘香蕉。（但直接是够不着的）
  - 环境：房间里有一把椅子和一根棍子。（若猴子站在椅子上手拿棍子正好够得着香蕉）
  - 动作：猴子知道如何走动、搬东西、挥动棍子、摘下香蕉。
  - 问题：猴子的最佳动作顺序是什么





第7讲 时空关联规划

7.1 规划的基本概念

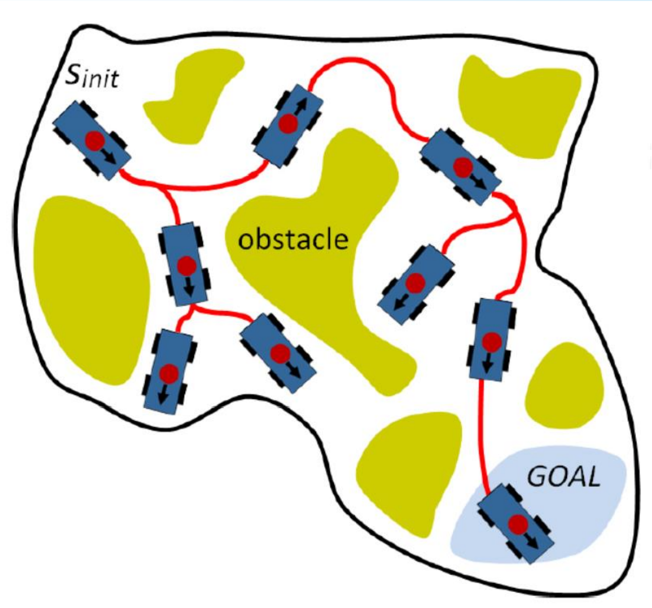
- 积木世界问题（Blocks world problem)
  - 目标：使积木按A、B、C的顺序叠放，如(b)
  - 环境：一组放在桌上的立方体形状的积木，积木能叠放，但只有一块积木能够直接放在另一块的上面
  - 动作：移动最上一块积木，放在桌子上或另一块积木的表面，每次只能移动一块积木
  - 问题：规划一个移动积木的最佳顺序，以达到目标

(a) 初始状态 (b) 目的状态

第7讲 时空关联规划

7.1 规划的基本概念

- 现实世界的规划问题举例：
  - 机器人运动规划（Motion planning）
  - 工厂作业调度（Job scheduling）
  - 航天器（Spacecraft）
  - 军事行动（Military campaigns）
  - ……



第7讲 时空关联规划

7.1 规划的基本概念

- 规划的概念

  指智能主体制定一套动作的计划来达到既定的目标。

  - （自动）规划 vs 人工智能
  - 规划 vs 搜索 vs 决策

- 规划的作用
  - 监控问题求解过程：简化搜索、解决目标矛盾、为差错补偿提供基础
  - 指导实际事业和工作：国家发展、企业规划、城市规划

- 规划的难度

特性	问题
动作	确定性的 vs 不确定性的 瞬时的 vs 有持续时间的 串行执行 vs 可并发执行
状态变量	连续 vs 离散
初始状态	有限 vs 无限
目标	到达指定的目标状态 最大化回报函数
智能体	单个 vs 多个 合作 vs 竞争

第7讲 时空关联规划

7.1 规划的基本概念

- 经典规划问题：
  - 完全可观测
  - 唯一已知初始状态
  - 静态环境
  - 确定性的动作
  - 每次仅一个动作
  - 单一智能体

原则	规划的分类
按方法	时空关联规划：空间规划、时序规划 非分层规划和分层规划 同步规划和异步规划 运动规划 多目标规划 多Agent规划 决策理论规划：马尔可夫决策
按实质	任务规划 路径规划 轨迹规划
按内容	国家战略规划、社会发展规划、重大项目规划、城市规划、环境规划……

第 7 讲 时空关联规划

第7讲 时空关联规划

7.2 经典规划问题描述

- 经典规划系统的定义
  - 状态转换系统：$\Sigma = (S, A, E, \gamma)$

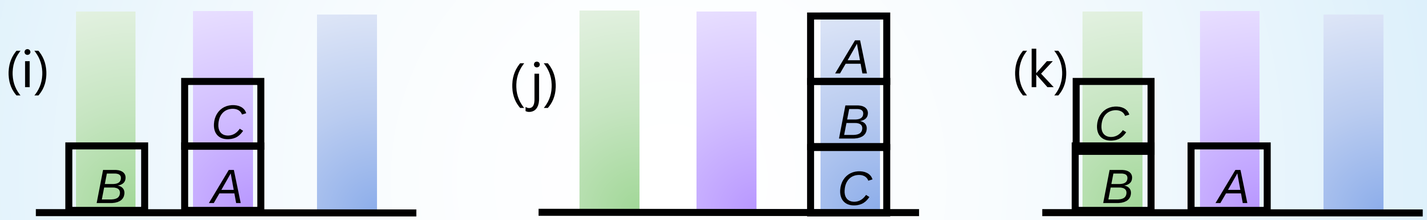
    其中，S是状态集，A是动作集，E是事件集，$\gamma = (s, a)$表示动作a作用于s，$a\epsilon A$，$s\epsilon S$

  - 受限转换系统：$\Sigma = (S, A, \gamma)$
    - 事件集E为空的状态转换系统
    - 满足：
      - 确定性（即每经过一个动作后会转换到另一个状态）
      - 静态（事件集为空集，只有动作集可影响系统状态）
      - 有限（S是有限的）
      - 完全可观测（状态S是完全可观测的）

第7讲 时空关联规划

7.2 经典规划问题描述

- 举例：积木世界（简化的汉诺塔问题）



受限状态转换系统

- 状态集描述：on(x,y), clear(x)　　x,y的集合为{A,B,C,T}
- 动作集描述：put-on(x,y)
- 转换描述：如$\gamma(s,a)$：s为状态(i)，a为put-on(C,B)

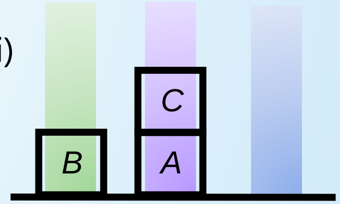
前提-效果

第7讲 时空关联规划

7.2 经典规划问题描述

- 经典规划问题的定义：三元组：$\mathcal{P}=(\sum, s_0, g)$
  - $\sum$ 是一个受限状态转换系统，
  - s_0 是初始状态，g 是目标状态。
  - 规划 $\mathcal{P}$ 的解是一个动作序列 $(a1, a2, \dots, ak)$，对应于一个状态转换序列（s1,s2,....sk），且满足$s1=\gamma(s0,a1)$, ...,$sk=\gamma(sk\text{-}1,ak)$，其中sk是一个目标状态。

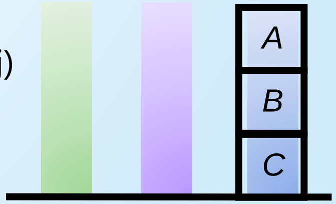
(i)



举例：积木问题定义

- $\sum$ —— 受限状态转换系统
- s_0 —— 初始状态(i) : on (B,T) ∧ on(C,A) ∧ on(A,T)
- g —— 目标状态(j) : on(A,B) ∧ on(B,C) ∧ on(C,T)

(j)



第 7 讲 时空关联规划

7.1 规划的基本概念

7.2 经典规划问题描述

7.3 经典规划问题求解

- 状态空间规划
- 计划空间规划

7.4 其他规划方法

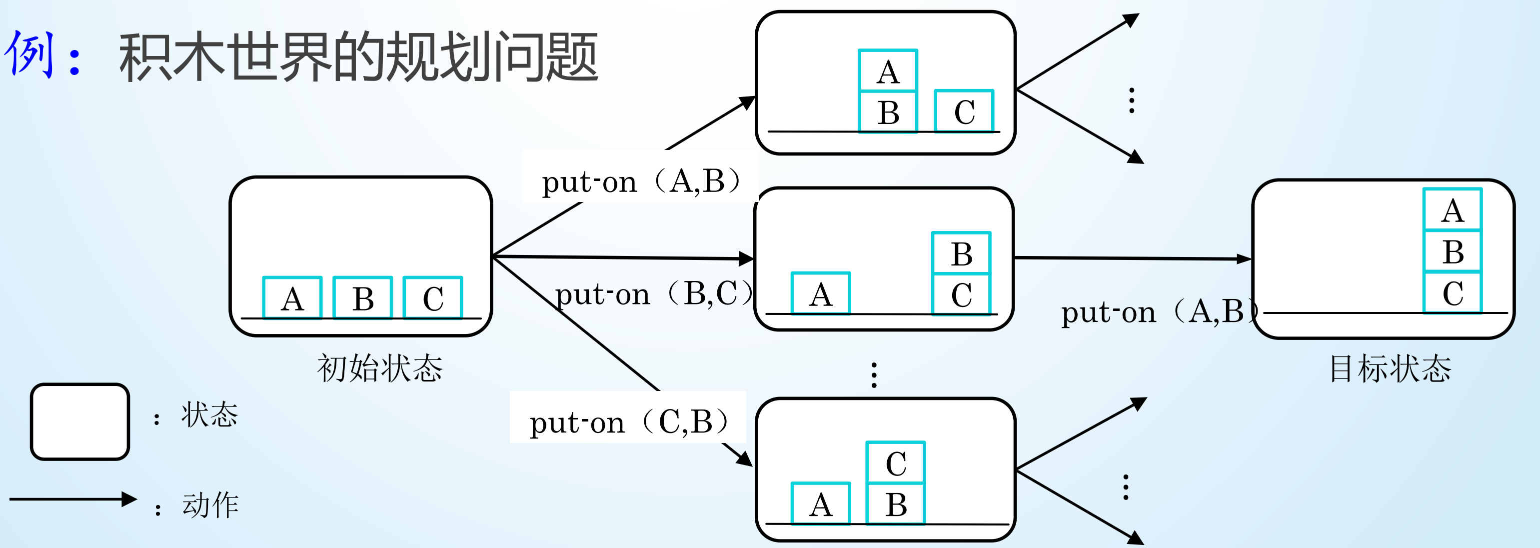
第7讲 时空关联规划

7.3 经典规划问题求解：状态空间规划

求解思路：将规划问题化作状态空间的搜索问题来解决

- 状态空间：用状态图表示
  - 图中每个节点对应于一个状态，
  - 每个边对应于一个状态转换，
  - 当前的计划对应于状态图中当前的路径。

举例：积木世界的规划问题



第7讲 时空关联规划

7.3 经典规划问题求解：状态空间规划

- 搜索算法
  - 前向搜索：从初始状态开始，寻找到一条达到目标状态的路径，该路径对应的动作序列就是状态空间规划的解
  - 后向搜索：从目标状态出发，逆向寻找一条到达初始状态的路径，再将该路径对应的动作序列反转过来就是解
  - 都满足可靠性soundness和完备性completeness

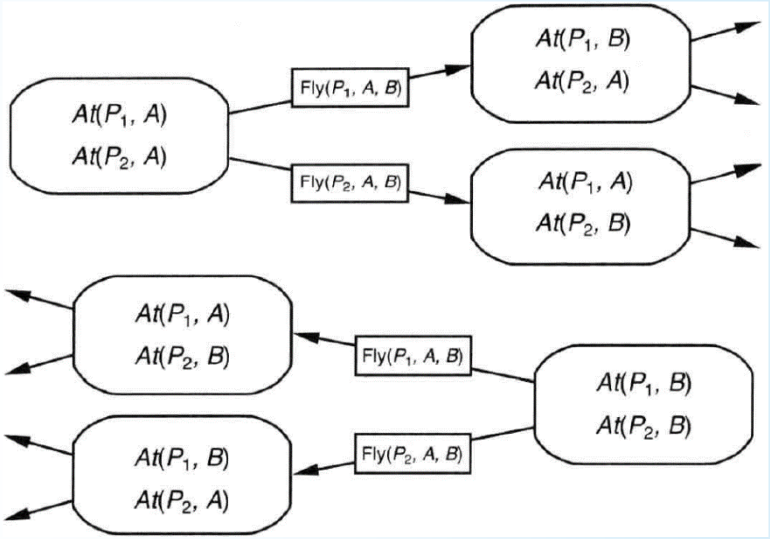
	前向搜索	后向搜索
导向	适应的（applicable）动作	相关的（relevant）动作
局限	容易搜索到无关动作 规划问题常有大的状态空间	需要能够从一个状态描述后退到前驱状态描述 需处理部分没有实例化的动作或状态

第7讲 时空关联规划

7.3 经典规划问题求解：状态空间规划

- 举例：飞机载货
  - 状态描述：$At(P_i,x)$
  - 动作描述：$Fly(P_i,x,y)$
  - 动作转换描述

前向和后向都不高效



- 规划的启发式

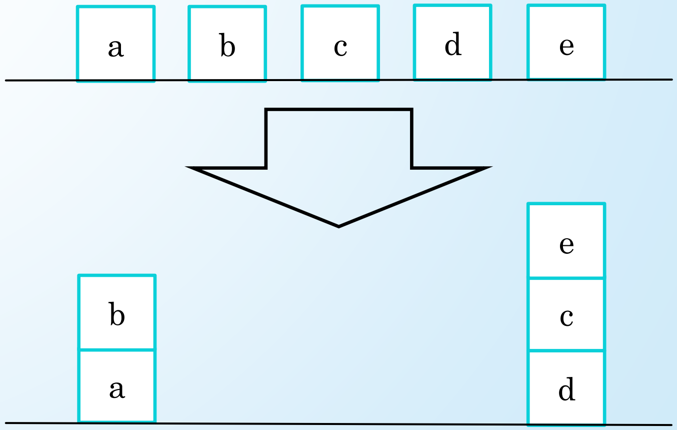
将规划搜索问题视为一个图，节点表示状态、边为动作，寻找一条连接初始状态至某个目标状态的路径。

  - 状态抽象——将多个节点组合到一起，将状态空间抽象为更少的状态、更便于搜索

第7讲 时空关联规划

7.3 经典规划问题求解：计划空间规划

- 计划空间规划
  - 节点表示一个部分指定计划（partially specified plan）
  - 边表示一个计划细化操作（plan refinement operation）
  - 计划从初始节点开始，此时该计划为空。其搜索旨在找到一个最终节点，它包含能达到所需目标的解法计划（solution plan）
- 偏序规划：
  - 全序关系 vs 偏序关系
  - 偏序规划：利用目标和动作的结构，使动作的搜索更加有效

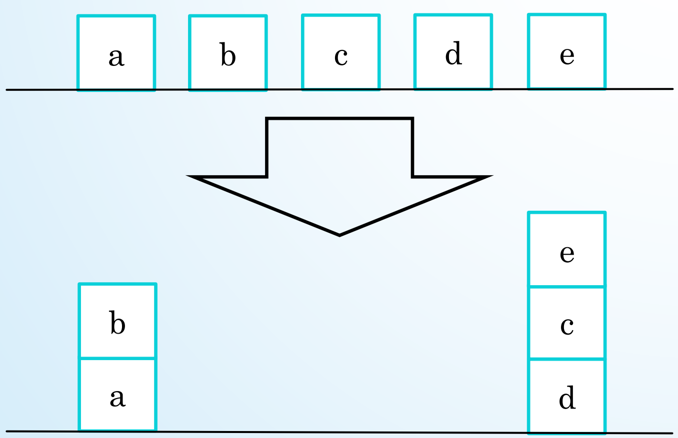


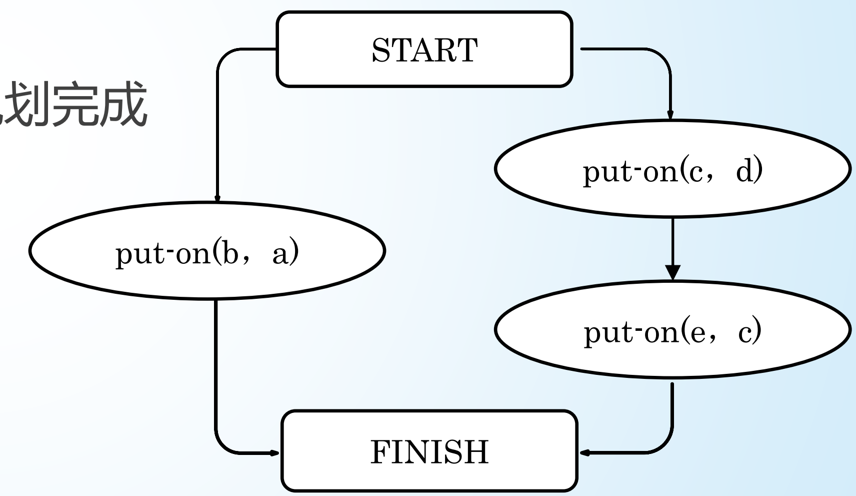
第7讲 时空关联规划

7.3 经典规划问题求解：计划空间规划

- 偏序规划的过程：
  - 初始化规划为 {start, finish}，添加序约束 start ≺ finish
  - 不断添加动作和序约束关系
  - 同时检测和消解冲突，直至规划完成

    start ≺ … ≺… ≺ finish





偏序规划

[put-on(b，a)，put-on(c，d)，put-on(e，c)]
[put-on(c，d)，put-on(b，a)，put-on(e，c)]
[put-on(c，d)，put-on(e，c)，put-on(b，a)]

全序规划

第7讲 时空关联规划

7.3 经典规划问题求解：计划空间规划

- 计划空间规划使用比动作序列更为通用的规划结构。包含两个独立的操作：
  - 选择动作
  - 对所选动作进行排序以达到目标

一个计划被定义为一组计划操作符以及排序约束和绑定约束，它可能对应于一个或多个动作序列。

第7讲 时空关联规划

7.3 经典规划问题求解：计划空间规划

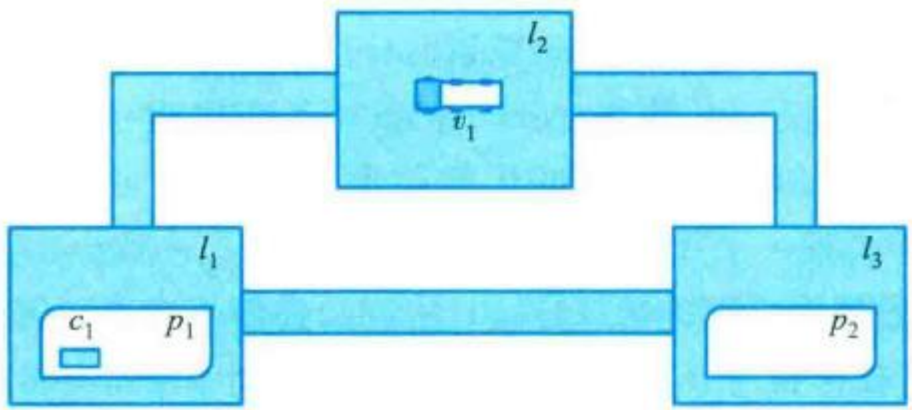
- 部分计划定义为一个四元组 $\pi = (A, \prec, B, L)$ ，其中：
  - $A = \{a_1, \dots, a_k\}$是部分实例化操作子（部分动作）的集合。
  - $\prec$是A中动作顺序约束的集合。形式为$(a_i \prec a_j)$，表示动作a_i先于动作a_j。
  - B是动作参数绑定约束的集合。形式为$x = y$，$x \neq y$，或$x \in D_x$，其中D_x表示x域的子集。
  - L是因果关联的集合。形式为$(a_i \xrightarrow{p} a_j)$，表示命题$p$是动作$a_i$的效果和动作$a_j$的前提，并且对于动作$a_i$和$a_j$的参数的绑定约束存在于$B$中。
- 四个细化操作
  - 添加动作（actions）
  - 添加顺序约束（ordering constraints）
  - 添加因果关联（causal links）
  - 添加可变绑定约束（variable binding constraints）

第7讲 时空关联规划

7.3 经典规划问题求解：计划空间规划

生成 部分计划 例题 P196 例7. 5

例 7.5 如图 7.2 所示，有三个地点、两个货堆、一个集装箱、一辆可以自动装卸集装箱的无人搬运车。用无人搬运车 v_1 将集装箱 c_1 从地点 l_1 的货堆 p_1 处运到地点 $l_2$3的货堆 p_2 处。在初始状态下，无人搬运车 v_1 位于地点 $l_3$2 所有的地点都是相邻可到达的。



第 7 讲 时空关联规划

7.1 规划的基本概念

7.2 经典规划问题描述

7.3 经典规划问题求解

7.4 其他规划方法

- 分层规划
- 新经典规划
- 路径规划 Dijkstra, A*

第7讲 时空关联规划

7.4 方法：分层规划

- 经典规划 vs 分层规划
  - 经典规划基于固定的原子动作——数量大、关系复杂
  - 分层规划引入抽象动作——将低层、原子动作抽象为高层、抽象任务
- 基元动作 vs 高层动作
  - 基元动作（primitive action）：即经典规划中的原子动作，具有标准的前提-效果模式

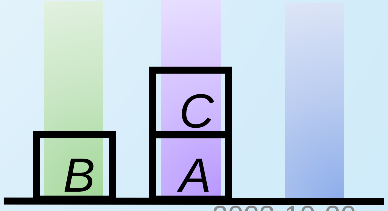
  例如：put-on(B,C)　　- 前提 clear(B) ∧clear(C)　　- 效果 on(B,C)

  - 高层动作HLA（high-level action）相对的

  例如：top(A): put-on(C,B), put-on(A,C);　或 put-on(C,B)

    - 每个HLA有一个或多个可能的细化
    - 每个动作可以是一个HLA，或一个基元动作

    put-on(C,B): unstack(C), load(C,B)



第7讲 时空关联规划

7.4 方法：新经典规划

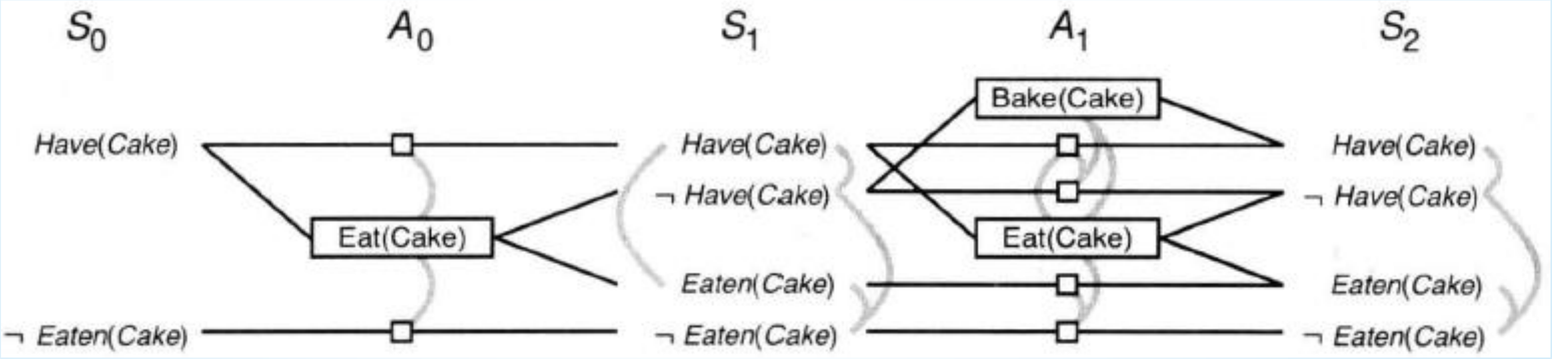
- 经典规划 ——搜索空间的每个节点是部分计划
  - 状态空间规划：动作序列
  - 计划空间规划：部分动作集合
- 新经典规划
  - 搜索空间的每个节点是若干个动作集合序列

    如: ({a1,a2},{a3,a4},{a5,a6,a7})
  - 将经典规划形式与自适应推理引擎相结合
- 新经典规划的主要方法
  - 规划图技法（Planning Graph Techniques）
  - 命题可满足性技法（Propositional satisfiability techniques）
  - 逻辑演绎技法（Logical deduction techniques）
  - 约束满足技法（Constraint satisfaction techniques）。

第7讲 时空关联规划

7.4 方法：新经典规划

- 规划图（Planning Graph）
  - 规划图是一个有向分层图，边只允许连接相邻层的节点
  - 作为状态空间规划和计划空间规划的一种折中，使得搜索空间可以更有效的组织和约束



- 规划图技法的二个重要概念
  - 可达性分析：用于判断是否可以从某个给定状态到达另一状态
  - 析取细化：通过求解器的析取来解决一个或多个瑕疵（flaw）

第7讲 时空关联规划

7.4 方法：新经典规划

可达性树 例题 P203 例7.6

(a)

move(*v*, *l*, *l*′) // automated guided wehicle *v* at location *l* moves to *l*′
    precond: at(*v*, *l*), adjacent(*l*, *l*′)
    effects: at(*v*, *l*′)
load(*c*, *v*, *l*) // wehicle *v* loads container *c* at location *l*
    precond: at(*v*, *l*), in(*c*, *l*), unloaded(*v*)
    effects: loaded(*v*, *c*), ¬in(*c*, *l*), ¬unloaded(*v*)
unload(*c*, *v*, *l*) // wehicle *v* unloads container *c* at location *l*
    precond: at(*v*, *l*), loaded(*v*, *c*)
    effects: unloaded(*v*), in(*v*, *l*), ¬loaded(*v*, *c*)

(b)

图 7.3 简化的无人搬运车问题及其操作

第7讲 时空关联规划

7.4 方法：Dijkstra算法

- 算法流程

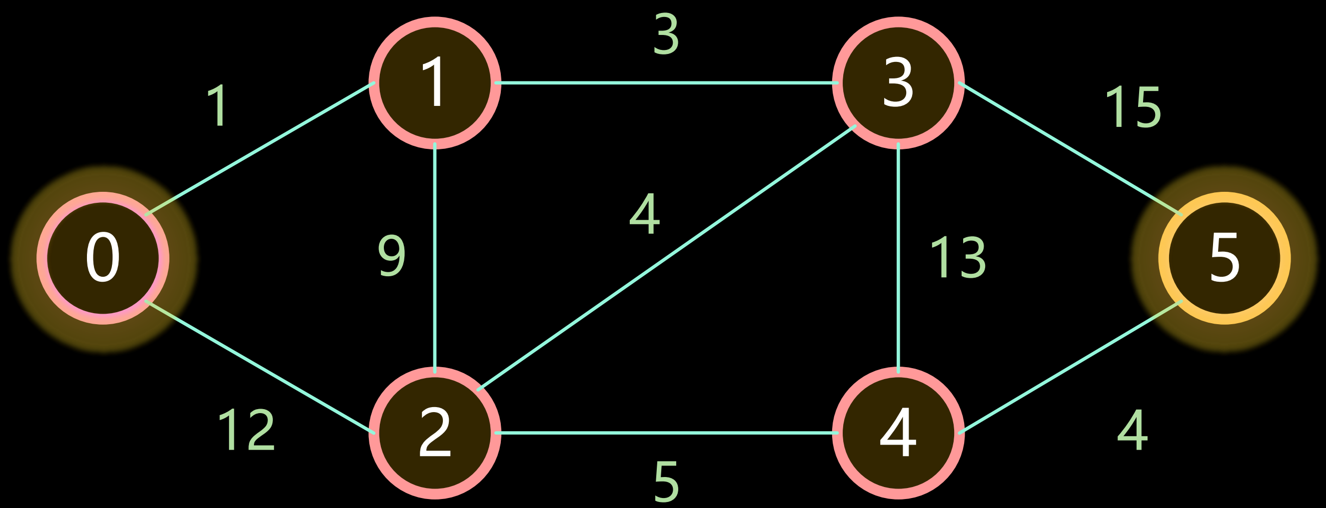
1. 指定一个起点D(即从节点D开始计算)。
2. 初始化两个数组S和U。S表示已求出最短路径的节点数组，U表示还未求出最短路径的节点数组。
3. 初始时，数组S中只有节点D；数组U中是除起点D之外的节点，并且数组U中记录各节点到起点D的距离。如果节点与起点D不相邻，距离为无穷大。
4. 每次从U中搜索距离最近的节点移入S中，并更新邻近节点的距离（仅在U中扩展，不包含已在S中的节点）。
5. 重复第4步操作，若U为空，或指定目标节点已在S中，则算法结束

第7讲 时空关联规划

7.4 方法：Dijkstra算法

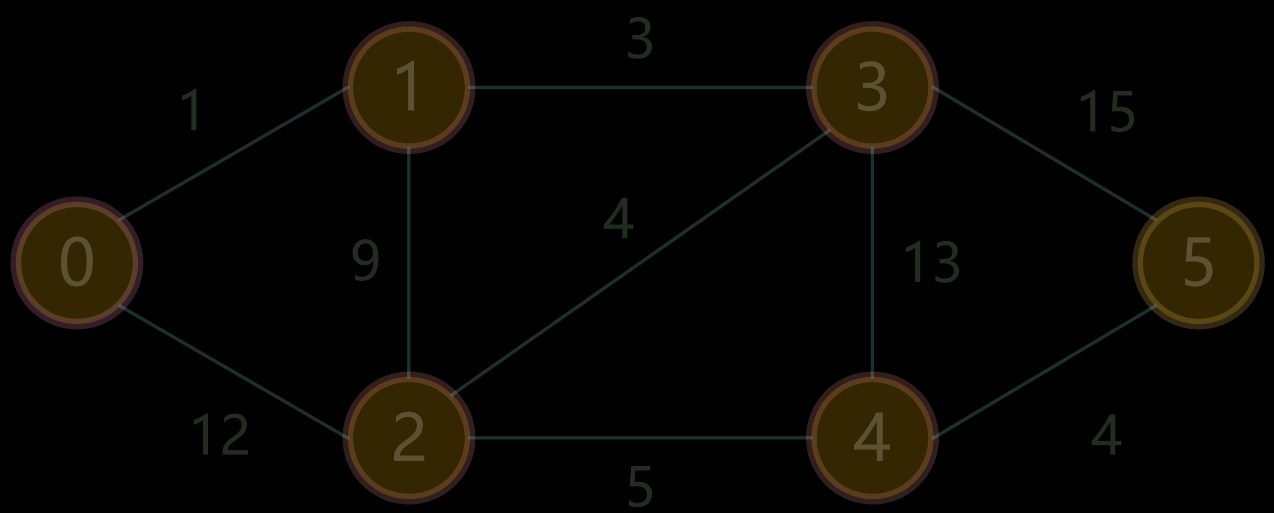
- 动画演示

问题：找出节点0到节点5的最短路径



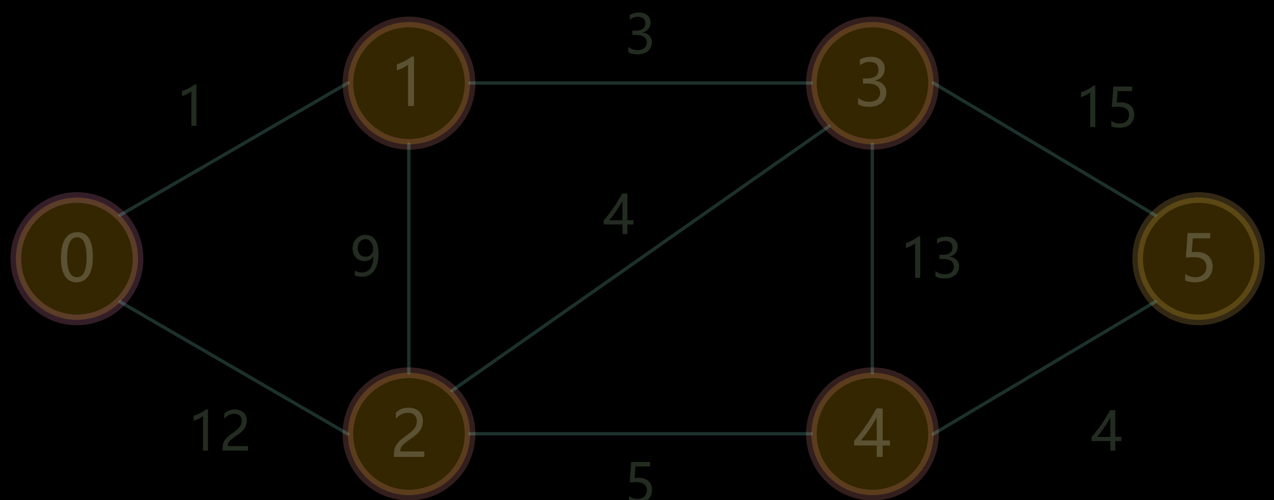
Index	0	1	2	3	4	5
Sign						
Parent						
Dis.						

问题：找出节点0到节点5的最短路径



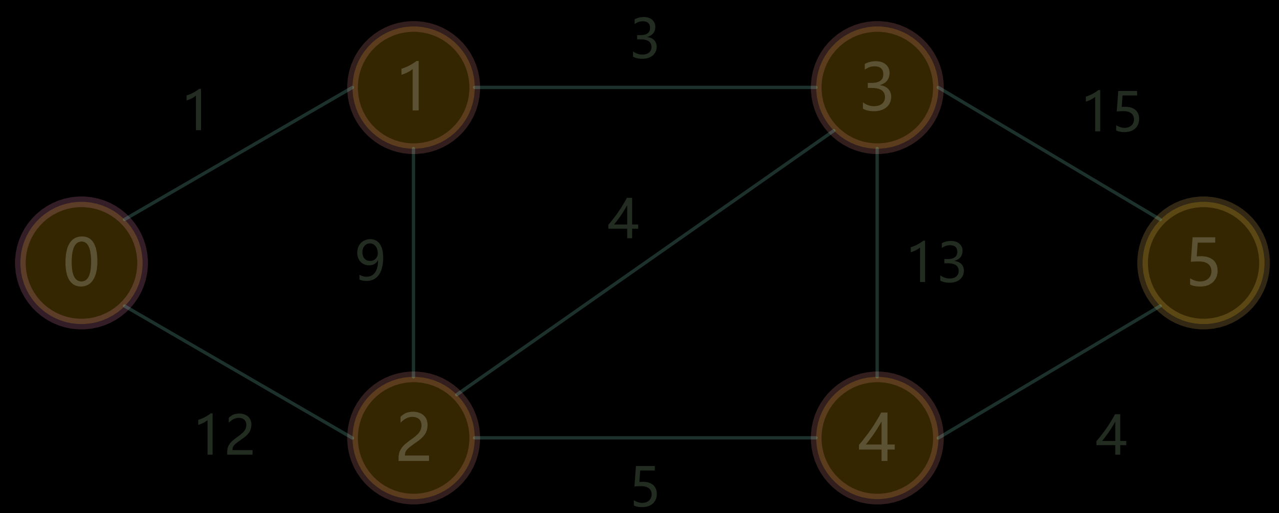
Index	0	1	2	3	4	5
Sign	×	×	×	×	×	×
Parent						
Dis.	+∞	+∞	+∞	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



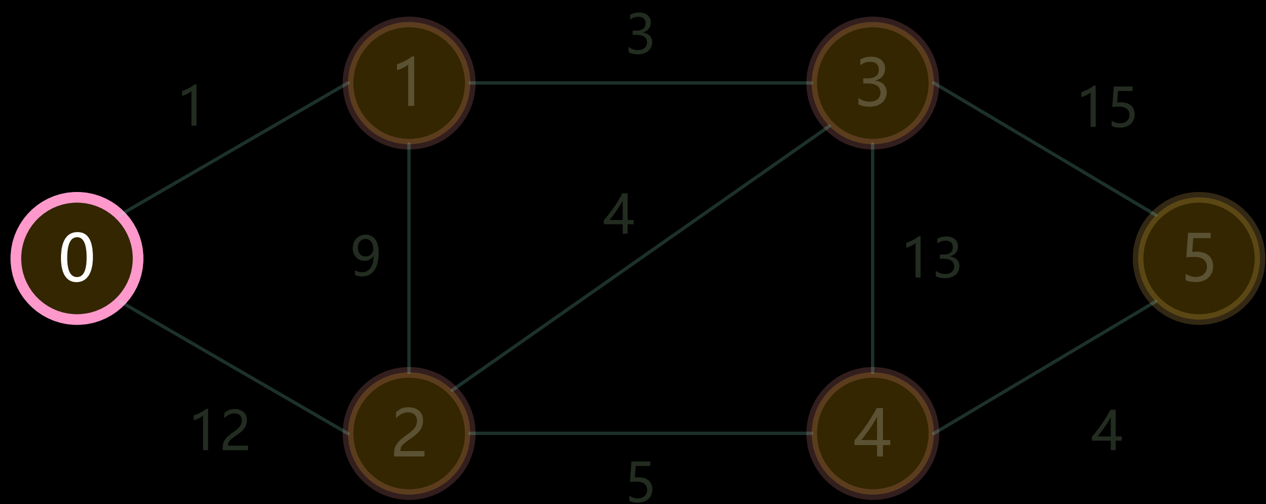
Index	0	1	2	3	4	5
Sign	×	×	×	×	×	×
Parent						
Dis.	0	+∞	+∞	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



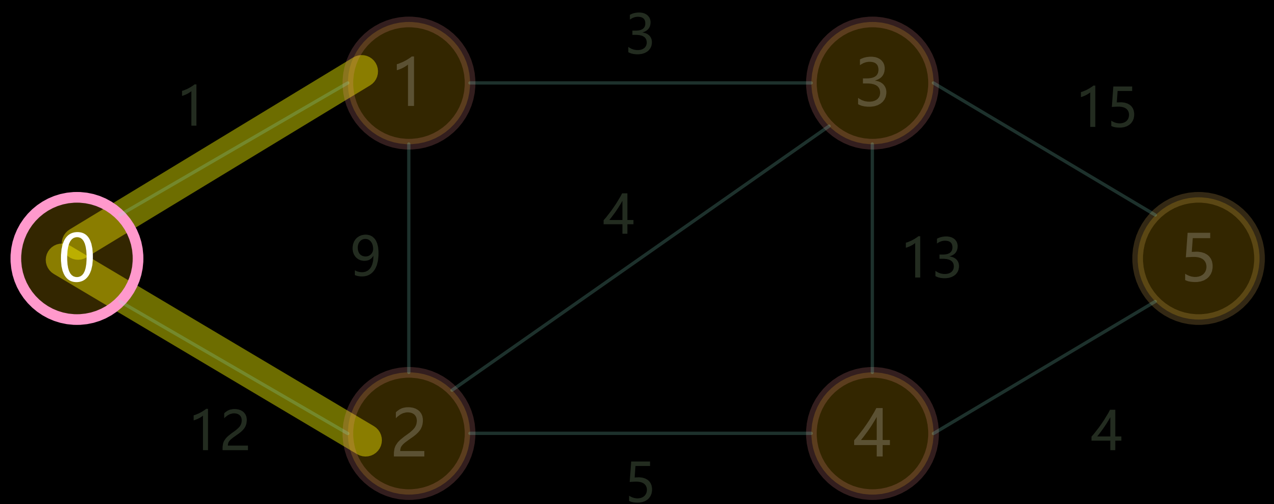
Index	0	1	2	3	4	5
Sign	×	×	×	×	×	×
Parent						
Dis.	$\boxed{0}$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

问题：找出节点0到节点5的最短路径



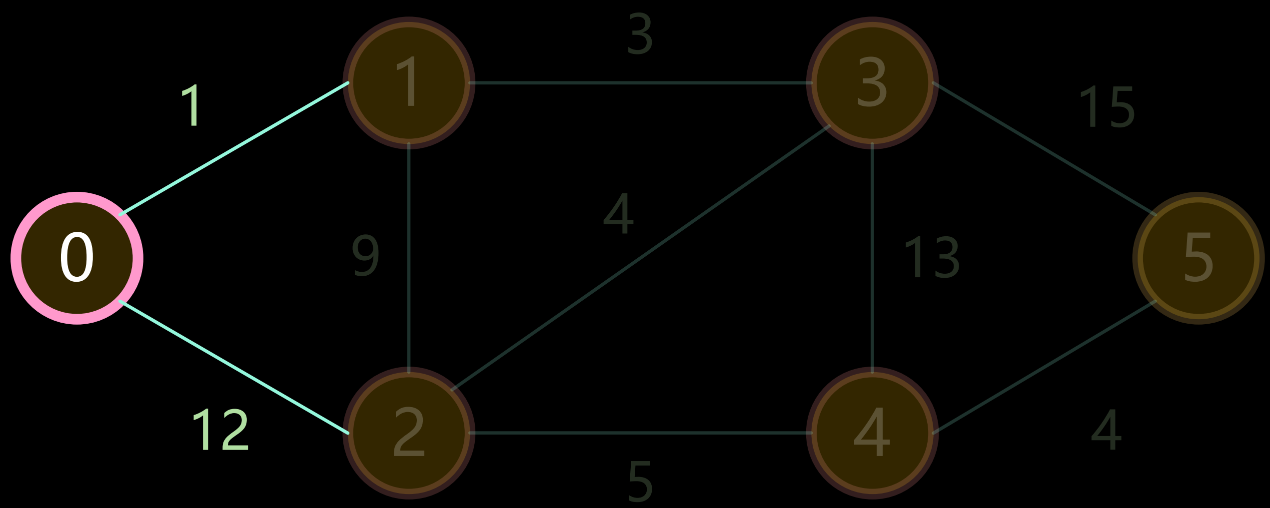
Index	0	1	2	3	4	5
Sign	√	×	×	×	×	×
Parent						
Dis.	0	+∞	+∞	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



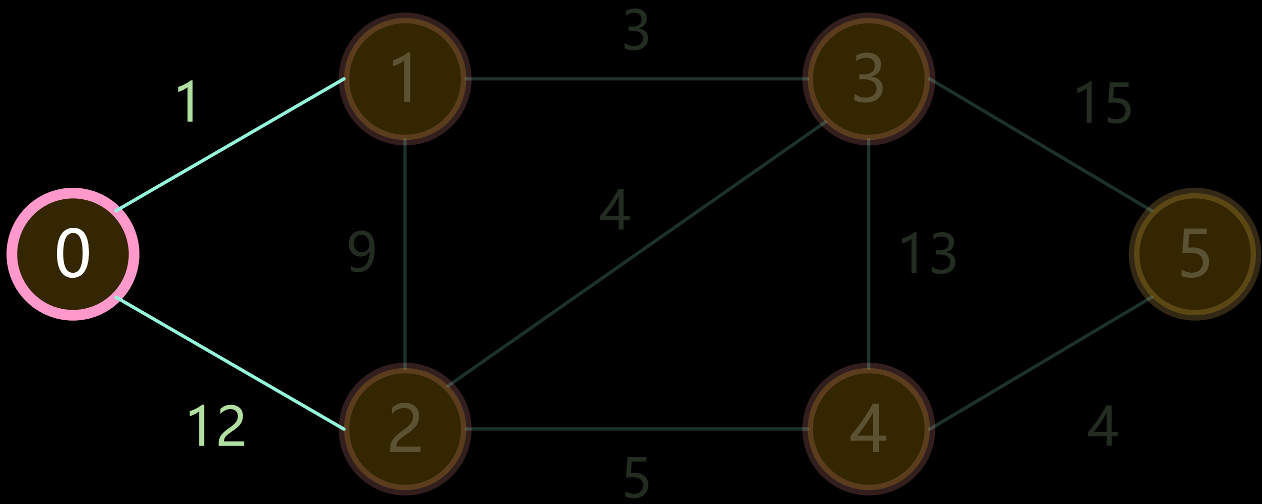
Index	0	1	2	3	4	5
Sign	√	×	×	×	×	×
Parent						
Dis.	0	+∞	+∞	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



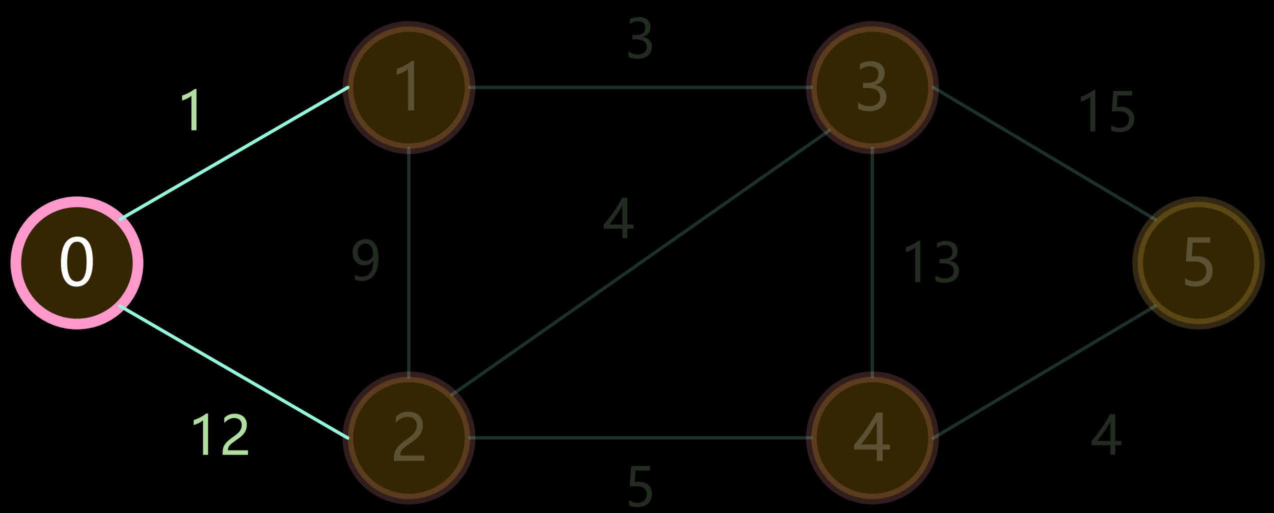
Index	0	1	2	3	4	5
Sign	√	×	×	×	×	×
Parent		0	0			
Dis.	0	1	12	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



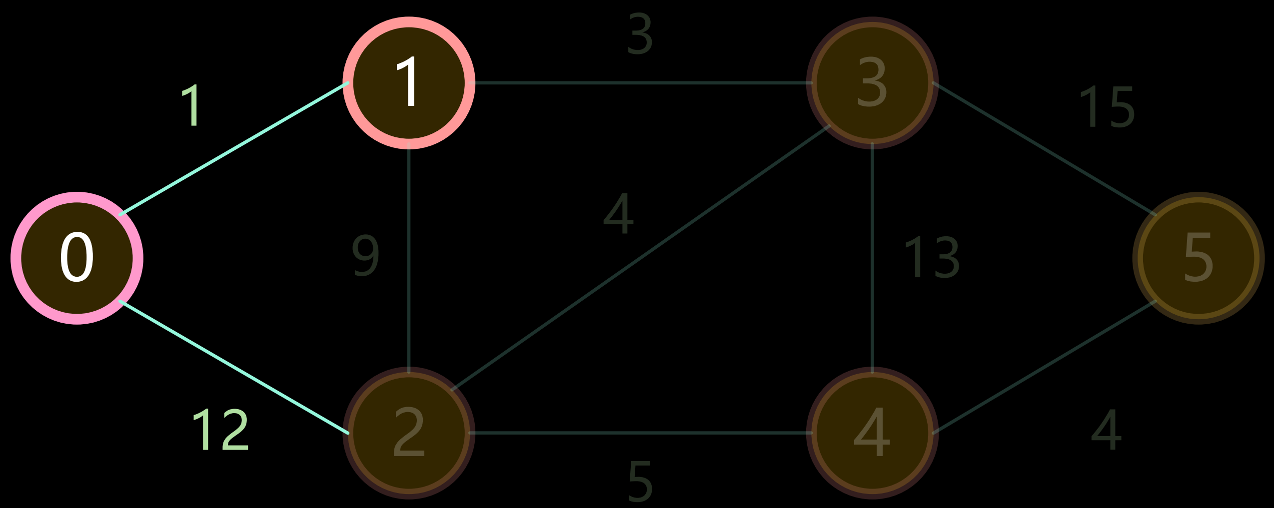
Index	0	1	2	3	4	5
Sign	√	×	×	×	×	×
Parent		0	0			
Dis.	0	1	12	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



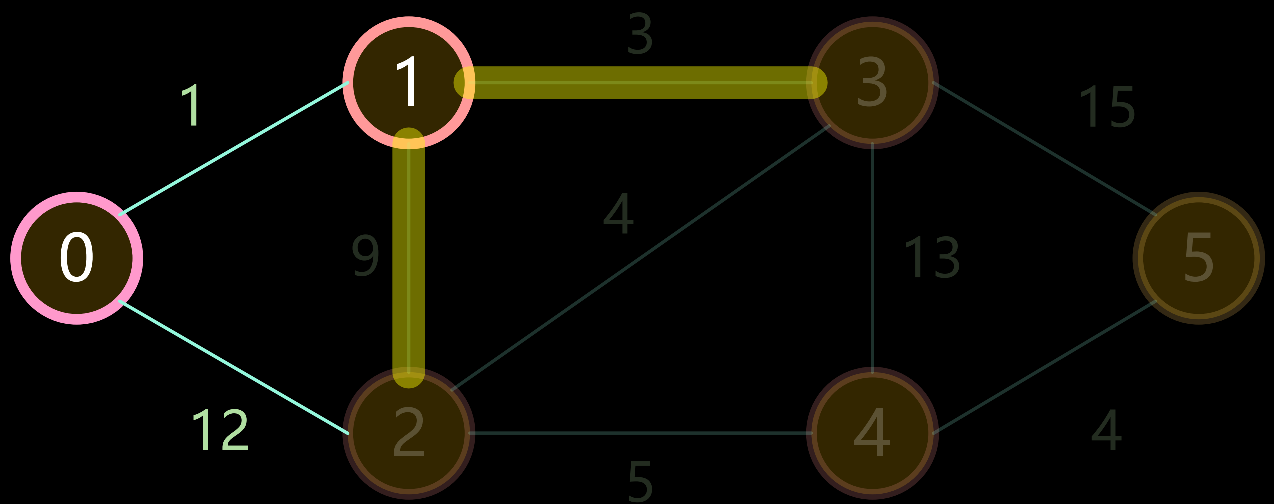
Index	0	1	2	3	4	5
Sign	√	×	×	×	×	×
Parent		0	0			
Dis.	0	1	12	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



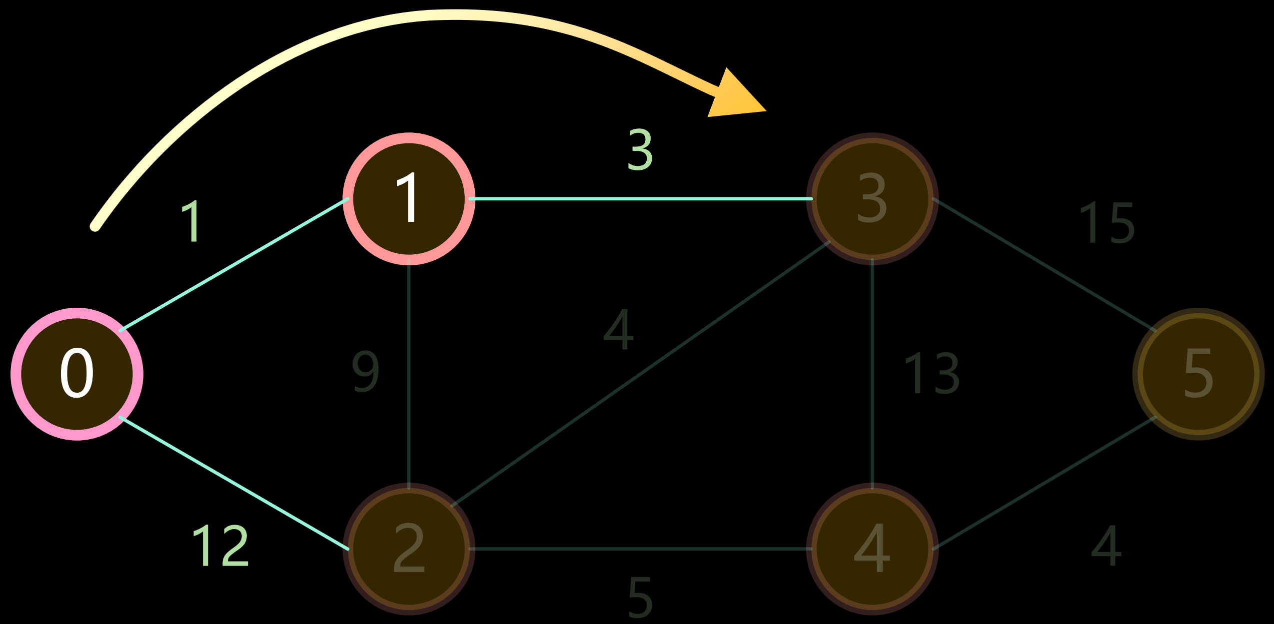
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	0			
Dis.	0	1	12	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



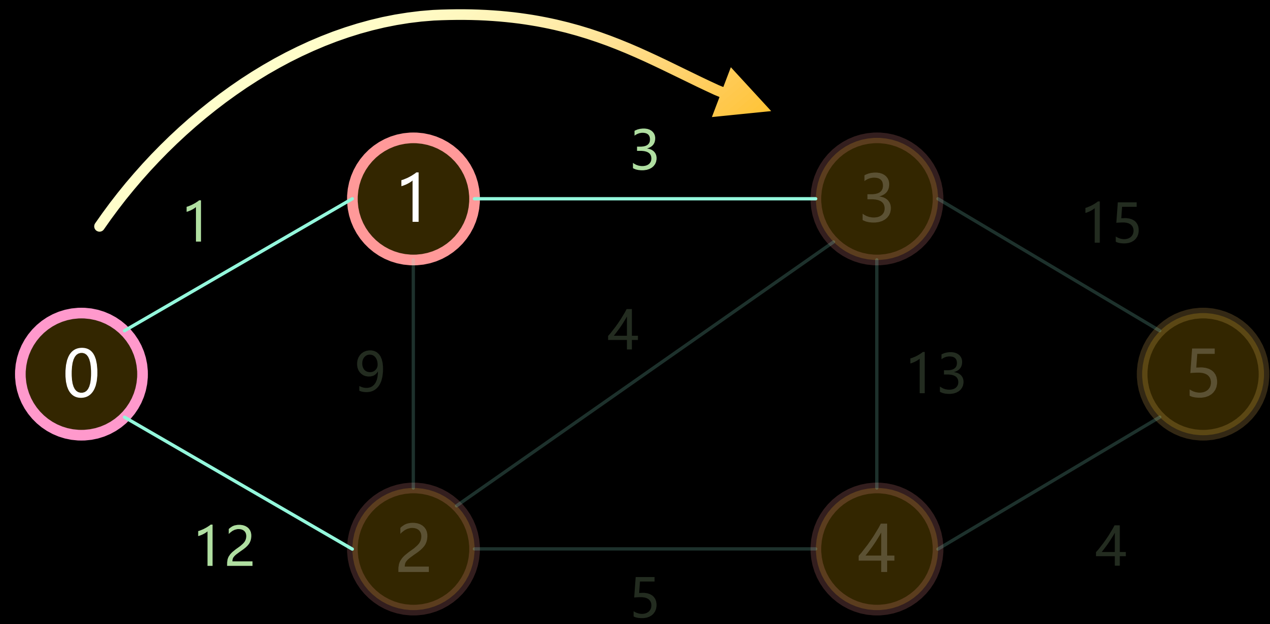
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	0			
Dis.	0	1	12	+∞	+∞	+∞

问题：找出节点0到节点5的最短路径



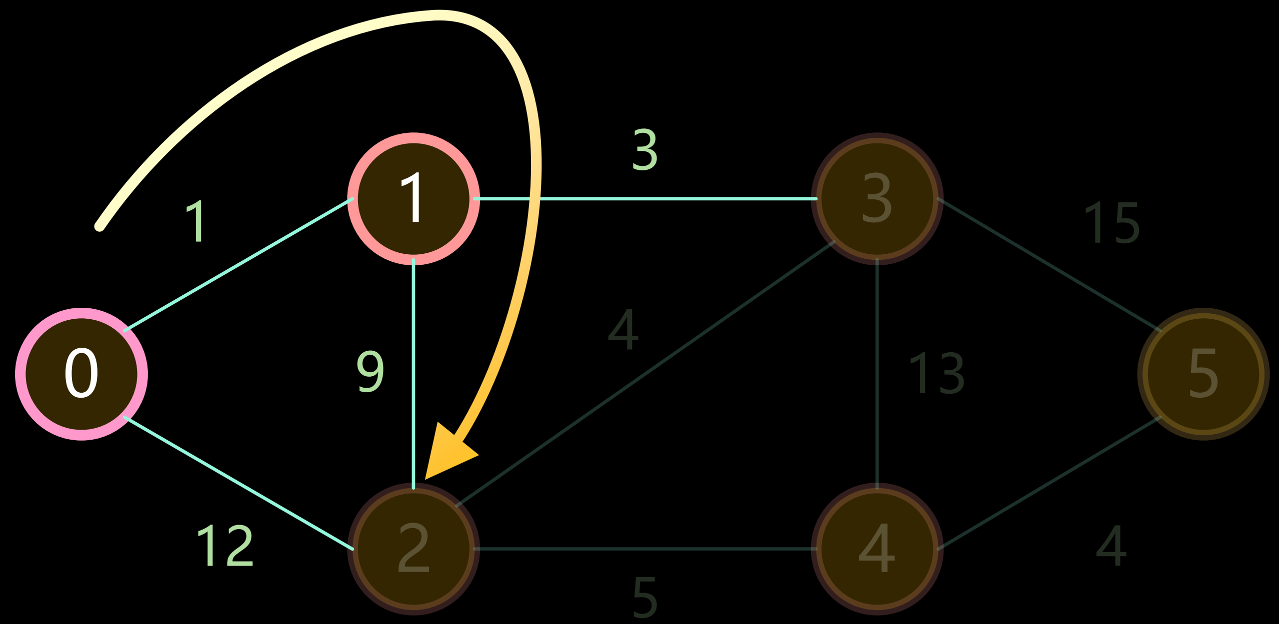
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	0			
Dis.	0	1	12	+∞ 4	+∞	+∞

问题：找出节点0到节点5的最短路径



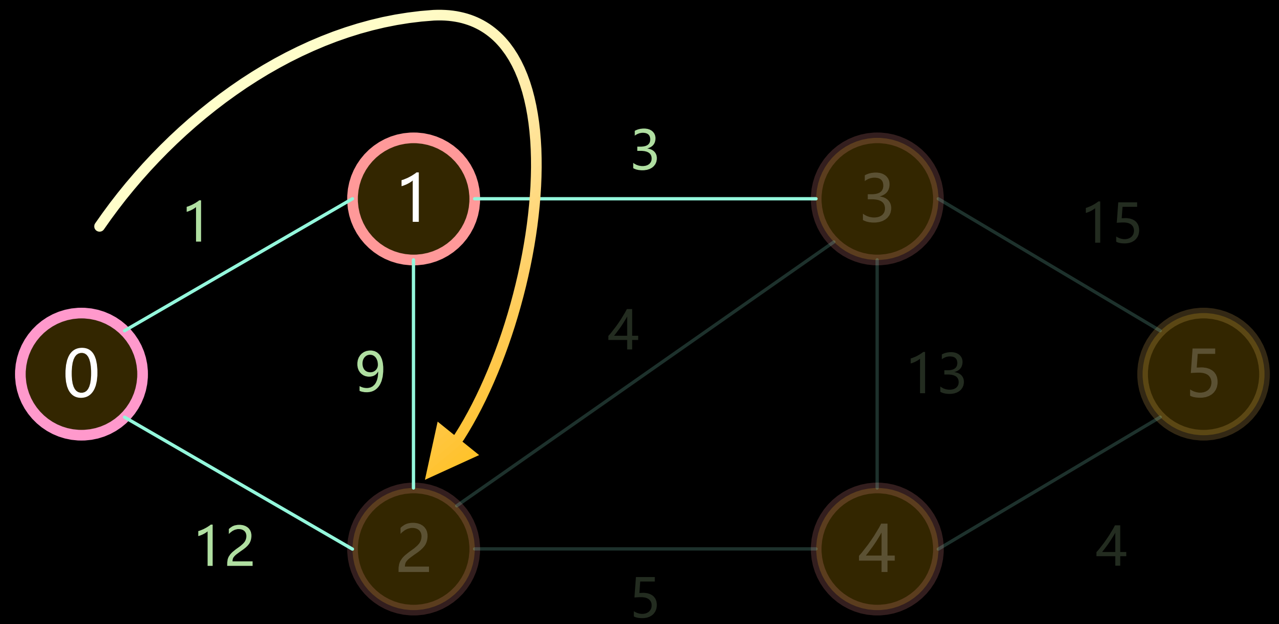
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	0	1		
Dis.	0	1	12	4	+∞	+∞

问题：找出节点0到节点5的最短路径



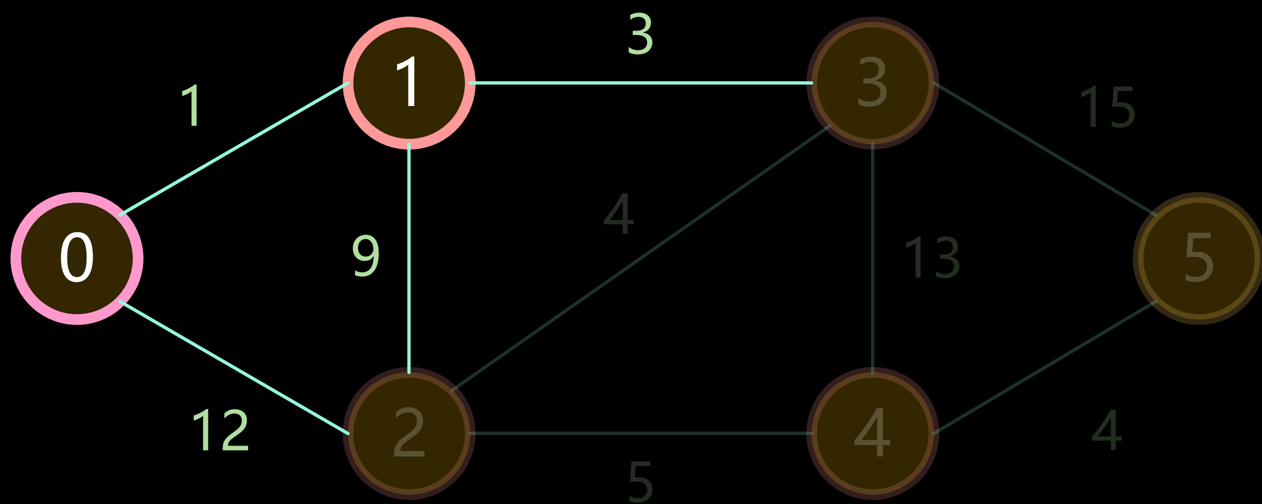
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	0	1		
Dis.	0	1	12 10	4	+∞	+∞

问题：找出节点0到节点5的最短路径



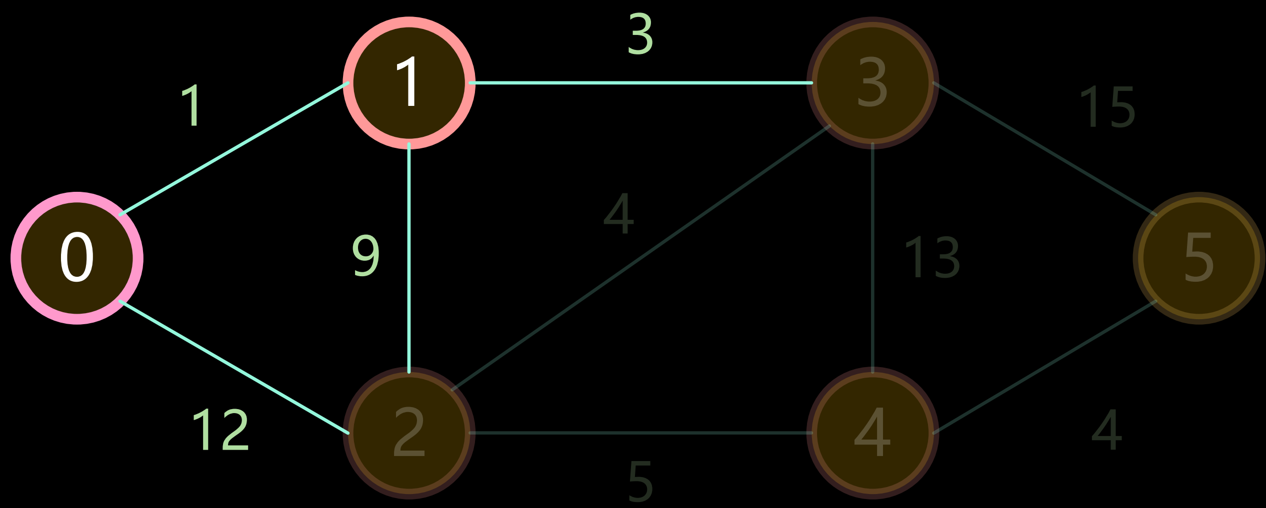
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	1	1		
Dis.	0	1	10	4	+∞	+∞

问题：找出节点0到节点5的最短路径



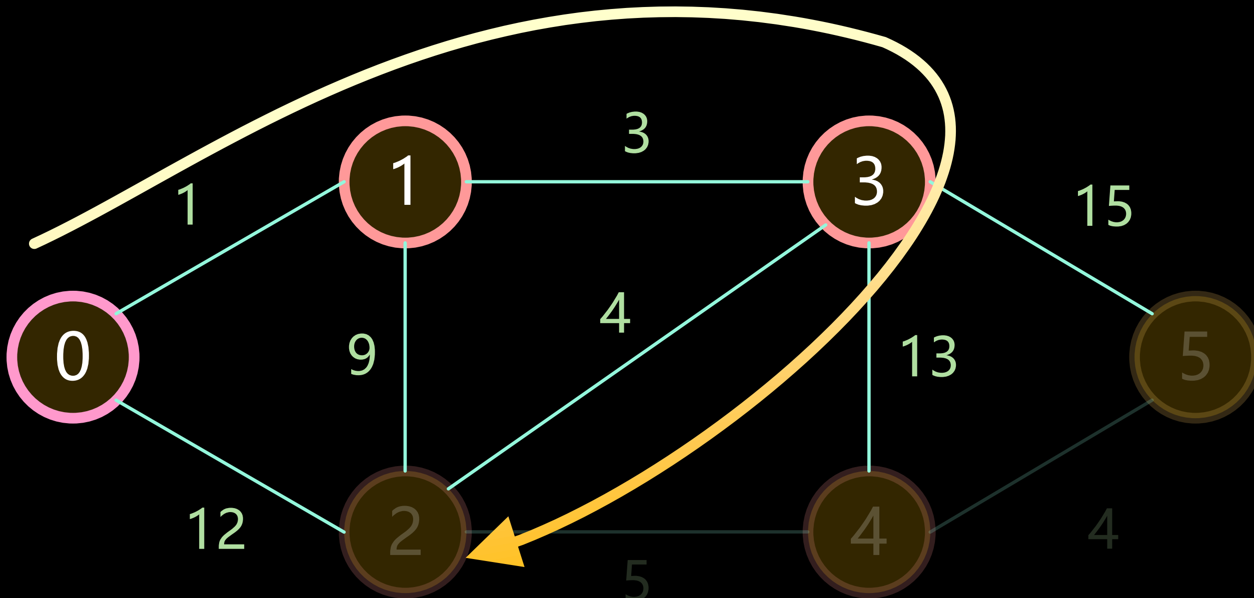
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	1	1		
Dis.	0	1	10	4	+∞	+∞

问题：找出节点0到节点5的最短路径



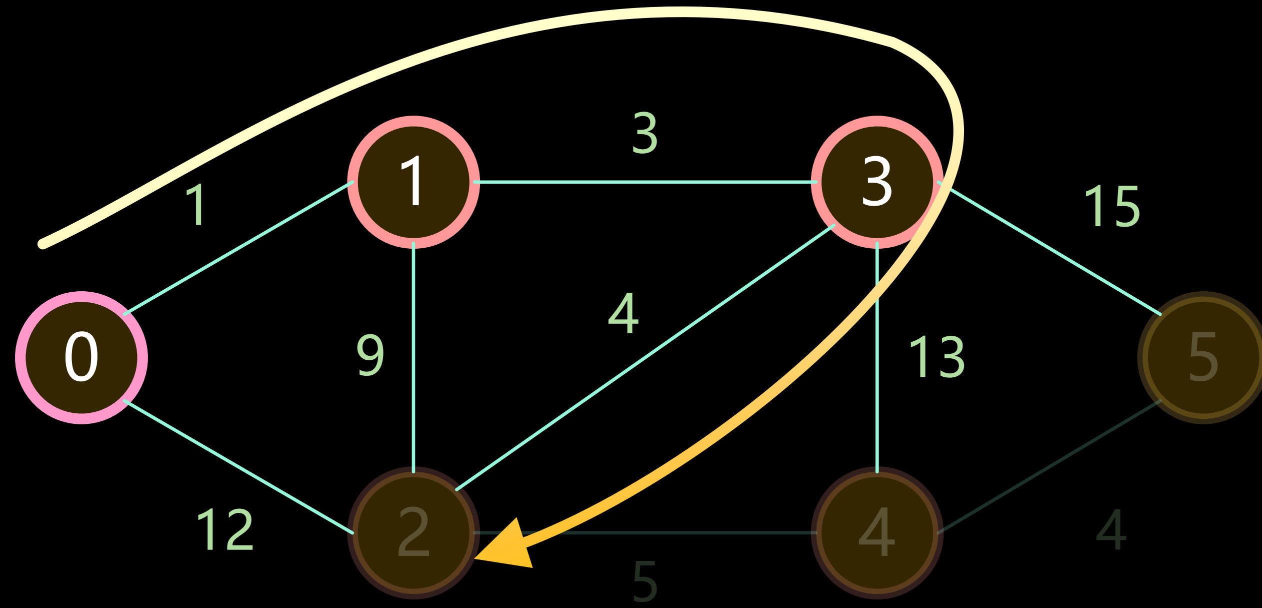
Index	0	1	2	3	4	5
Sign	√	√	×	×	×	×
Parent		0	1	1		
Dis.	0	1	10	4	+∞	+∞

问题：找出节点0到节点5的最短路径



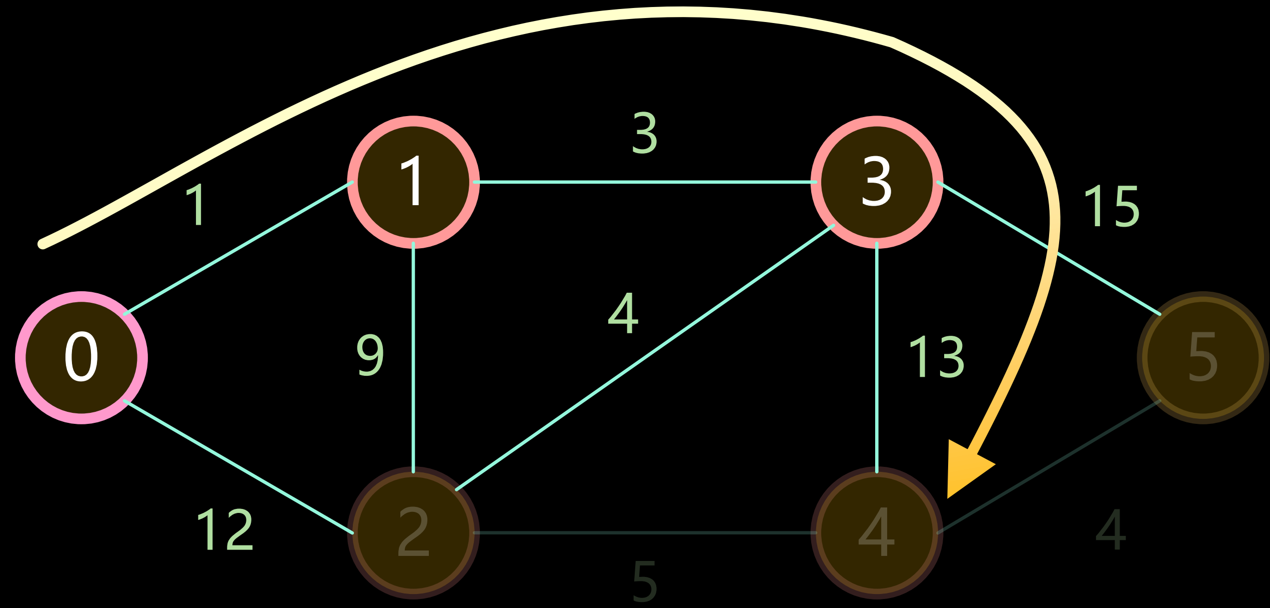
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	1	1		
Dis.	0	1	10	4	+∞	+∞

问题：找出节点0到节点5的最短路径



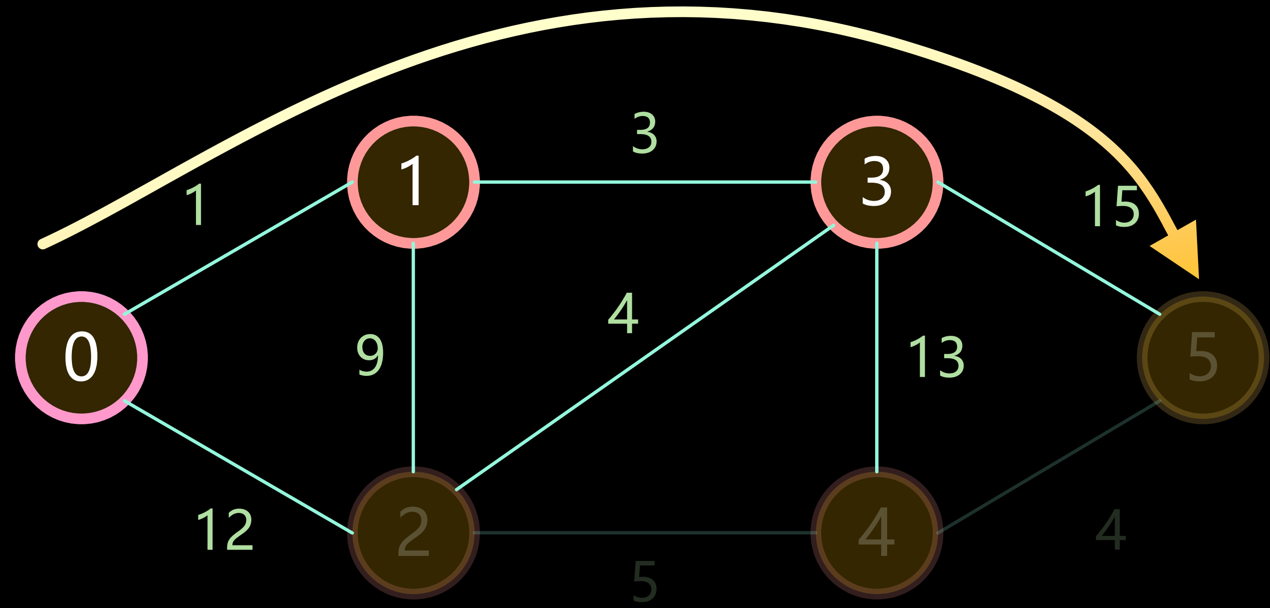
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	1	1		
Dis.	0	1	10 8	4	+∞	+∞

问题：找出节点0到节点5的最短路径



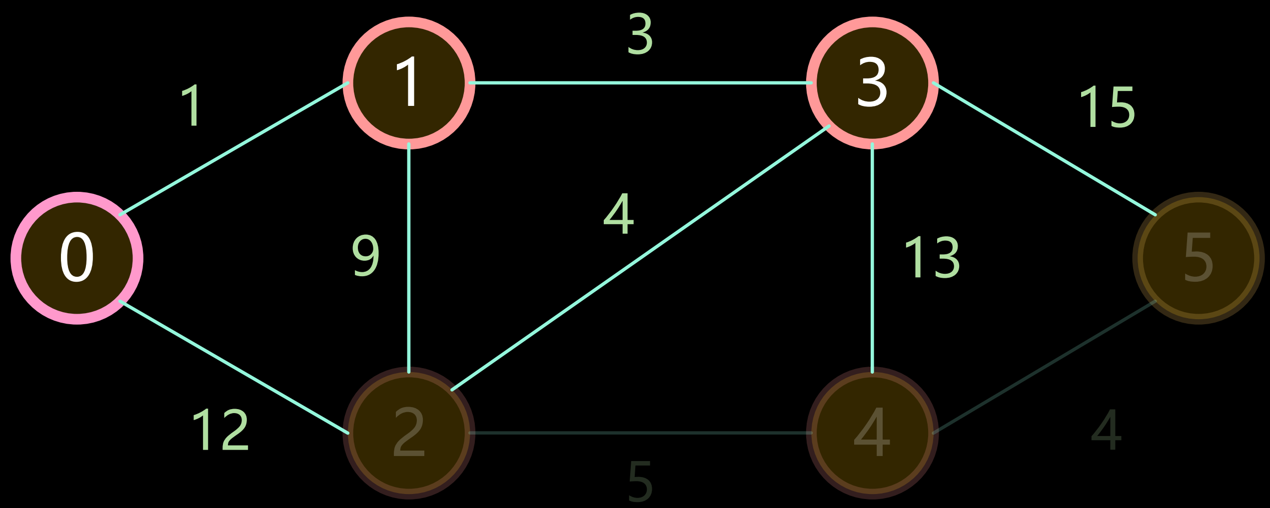
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	1	1		
Dis.	0	1	10 8	4	+∞ 17	+∞

问题：找出节点0到节点5的最短路径



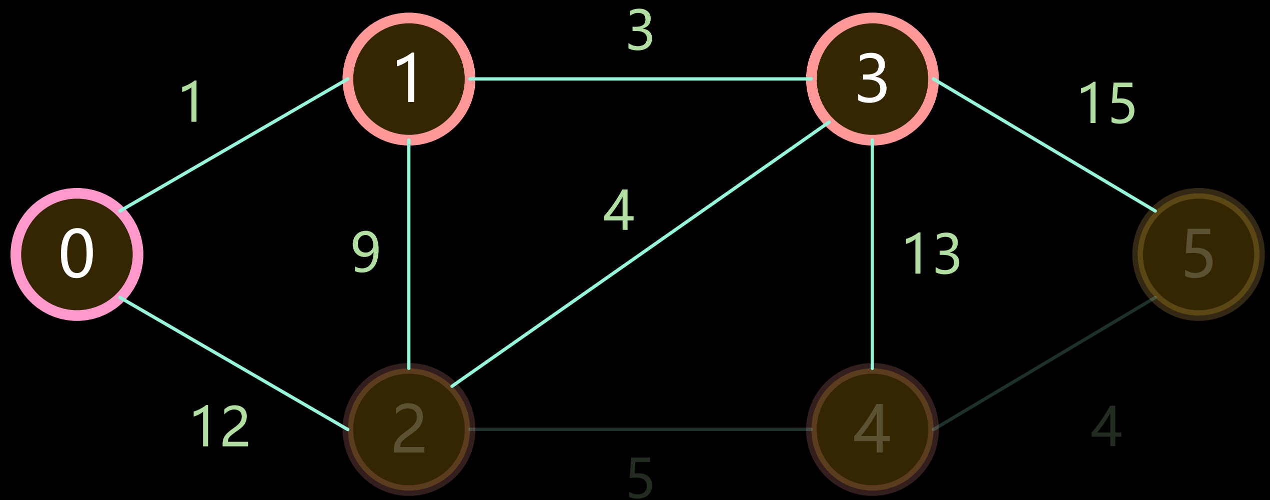
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	1	1		
Dis.	0	1	10 8	4	+∞ 17	+∞ 19

问题：找出节点0到节点5的最短路径



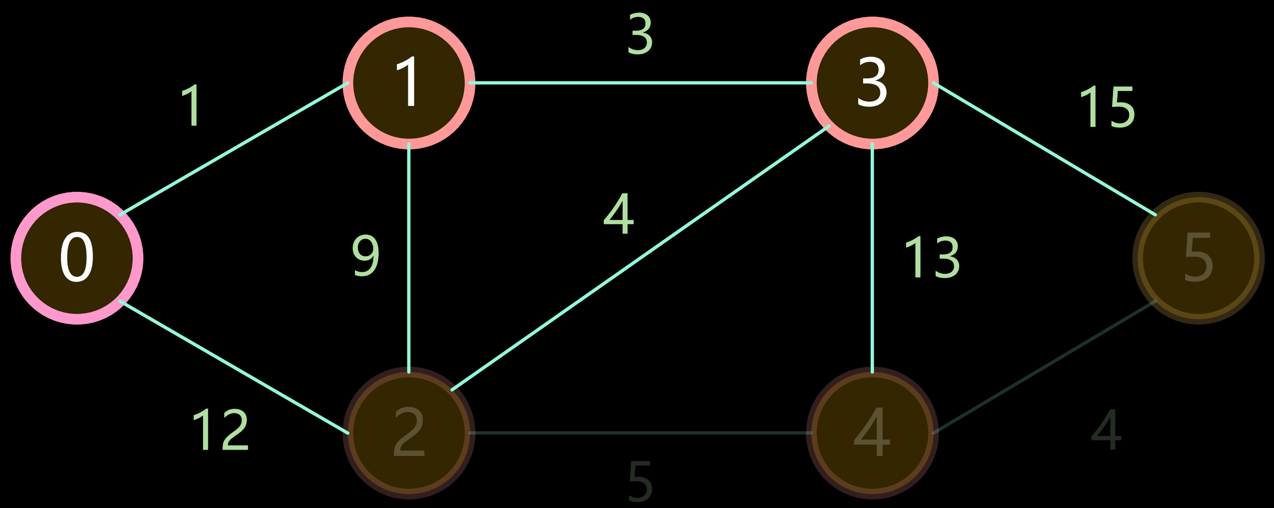
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	3	1	3	3
Dis.	0	1	8	4	17	19

问题：找出节点0到节点5的最短路径



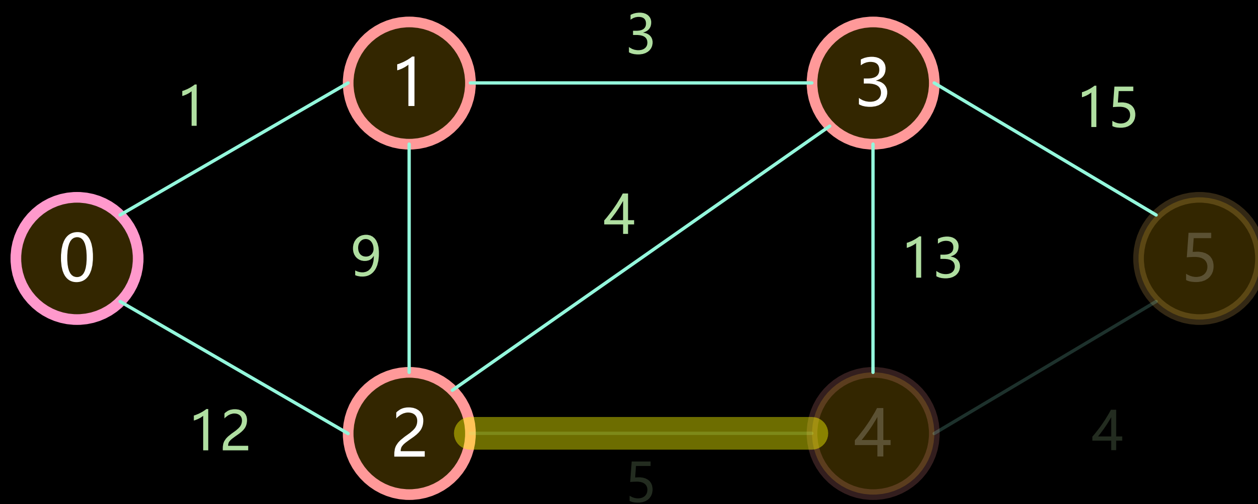
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	3	1	3	3
Dis.	0	1	8	4	17	19

问题：找出节点0到节点5的最短路径



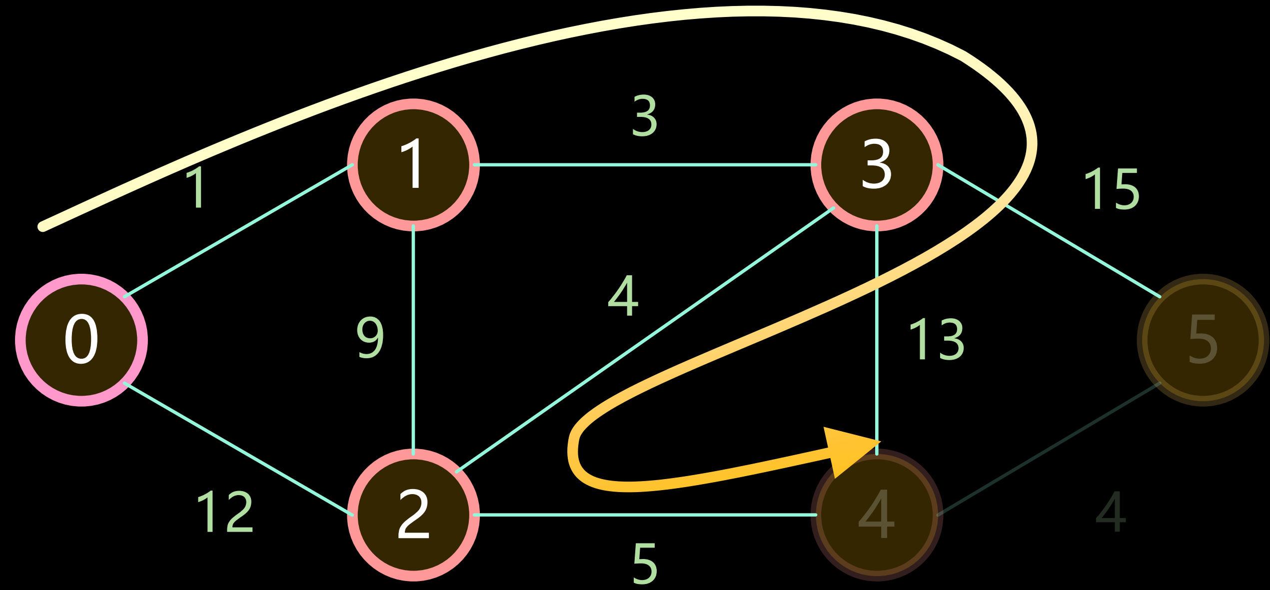
Index	0	1	2	3	4	5
Sign	√	√	×	√	×	×
Parent		0	3	1	3	3
Dis.	0	1	8	4	17	19

问题：找出节点0到节点5的最短路径



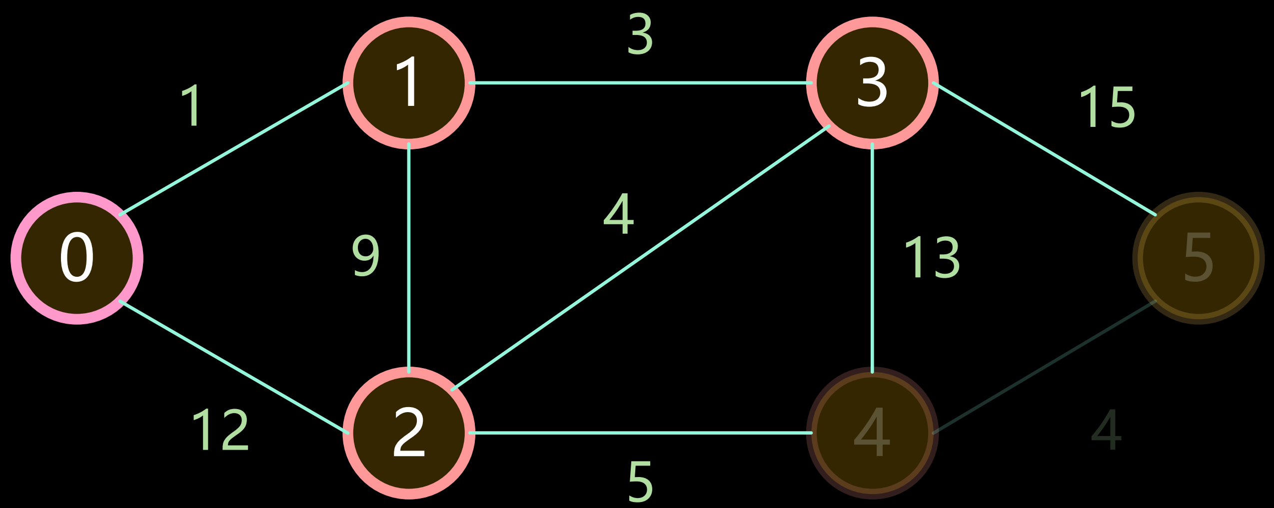
Index	0	1	2	3	4	5
Sign	√	√	√	√	×	×
Parent		0	3	1	3	3
Dis.	0	1	8	4	17	19

问题：找出节点0到节点5的最短路径



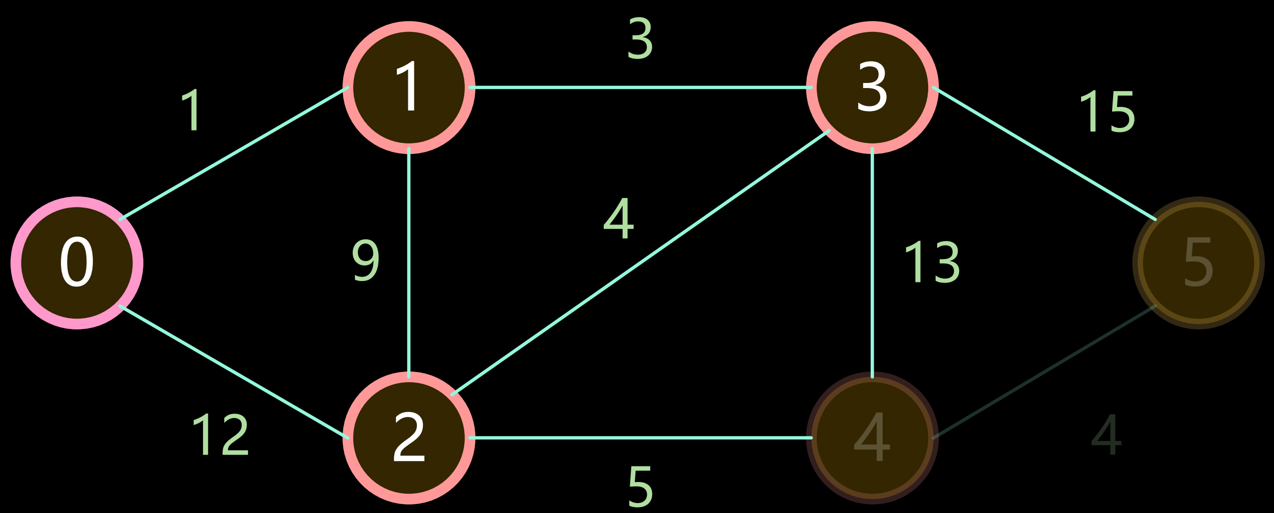
Index	0	1	2	3	4	5
Sign	√	√	√	√	×	×
Parent		0	3	1	3	3
Dis.	0	1	8	4	17 13	19

问题：找出节点0到节点5的最短路径



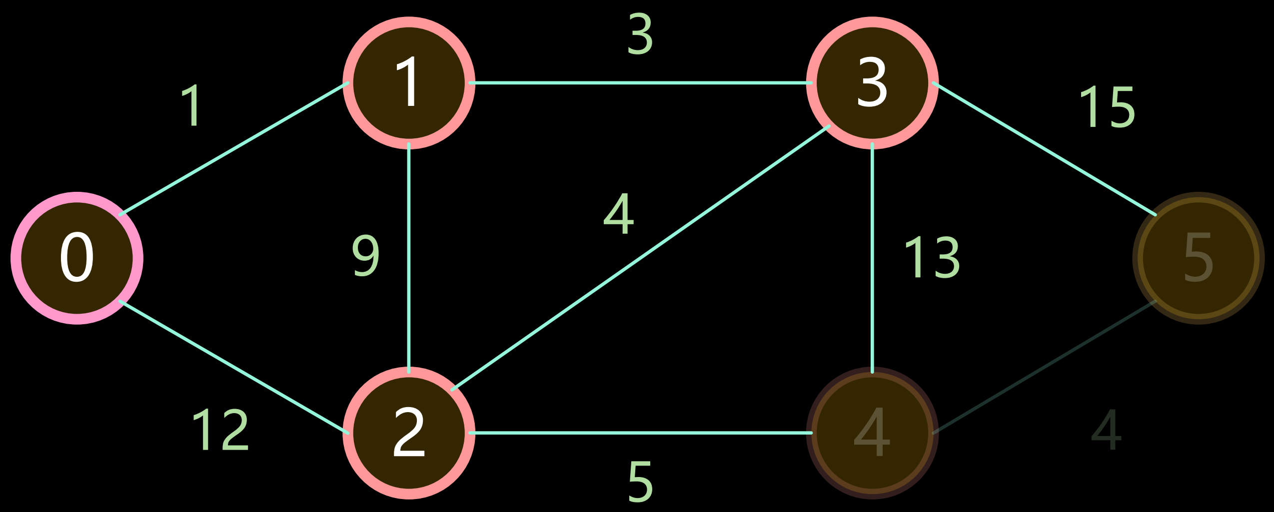
Index	0	1	2	3	4	5
Sign	√	√	√	√	×	×
Parent		0	3	1	2	3
Dis.	0	1	8	4	13	19

问题：找出节点0到节点5的最短路径



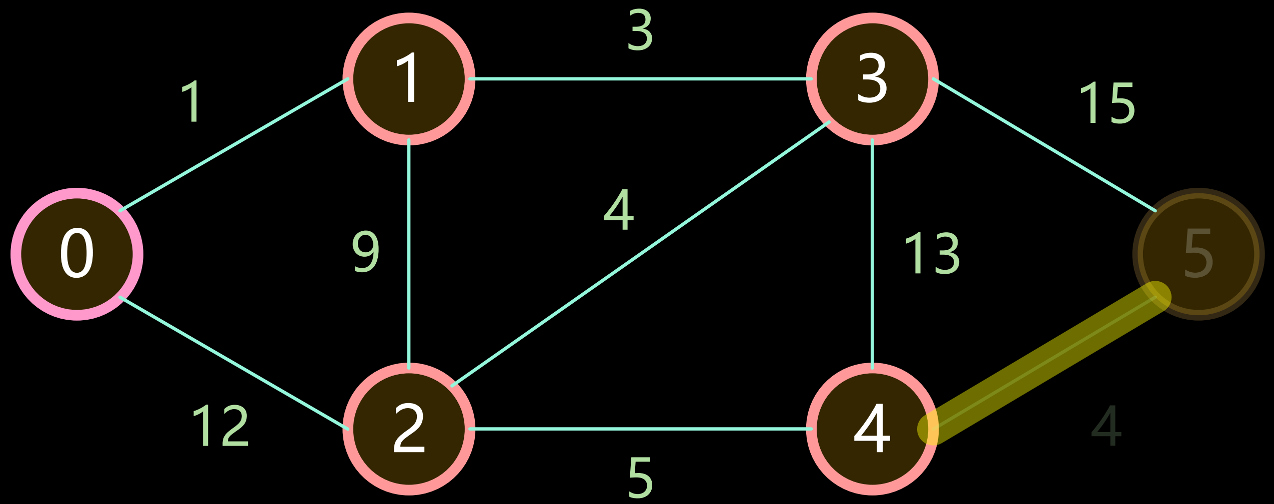
Index	0	1	2	3	4	5
Sign	√	√	√	√	×	×
Parent		0	3	1	2	3
Dis.	0	1	8	4	13	19

问题：找出节点0到节点5的最短路径



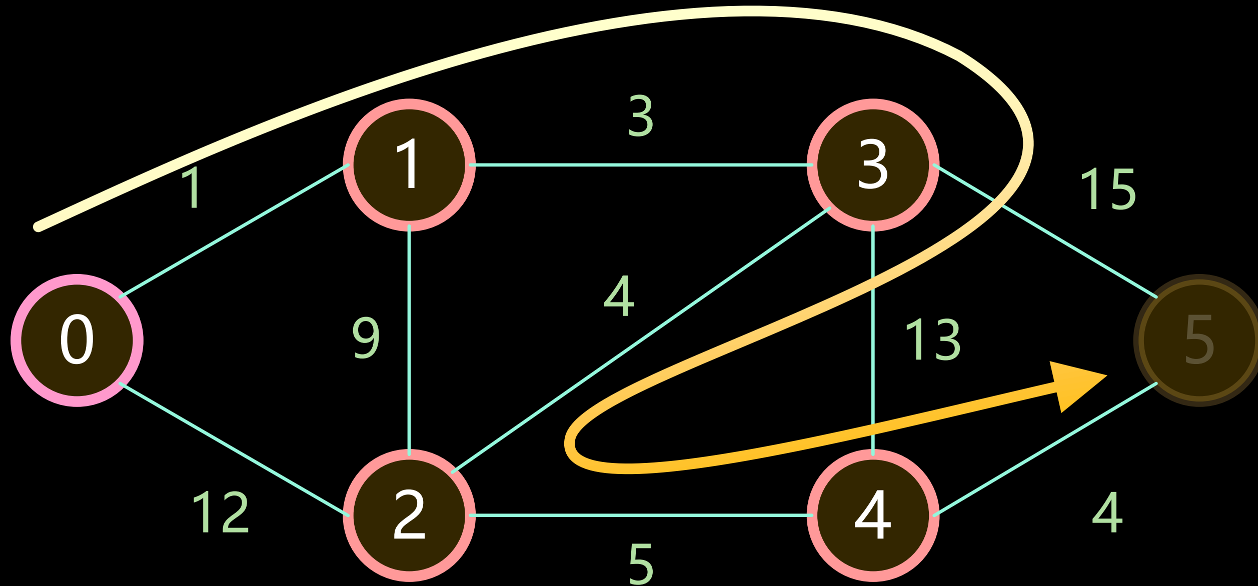
Index	0	1	2	3	4	5
Sign	√	√	√	√	×	×
Parent		0	3	1	2	3
Dis.	0	1	8	4	$\boxed{13}$	19

问题：找出节点0到节点5的最短路径



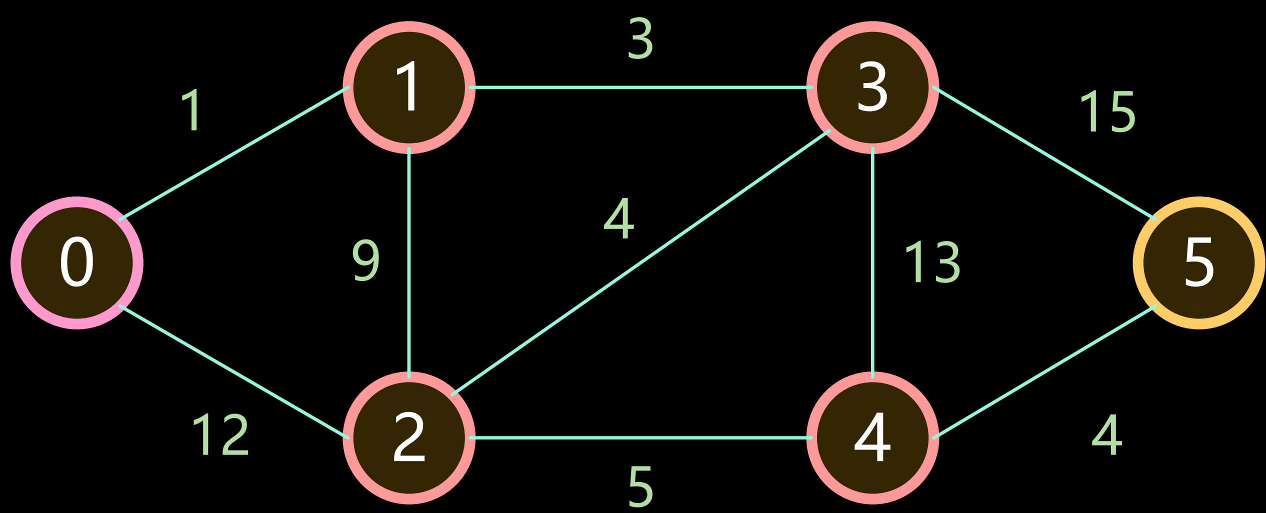
Index	0	1	2	3	4	5
Sign	√	√	√	√	√	×
Parent		0	3	1	2	3
Dis.	0	1	8	4	13	19

问题：找出节点0到节点5的最短路径



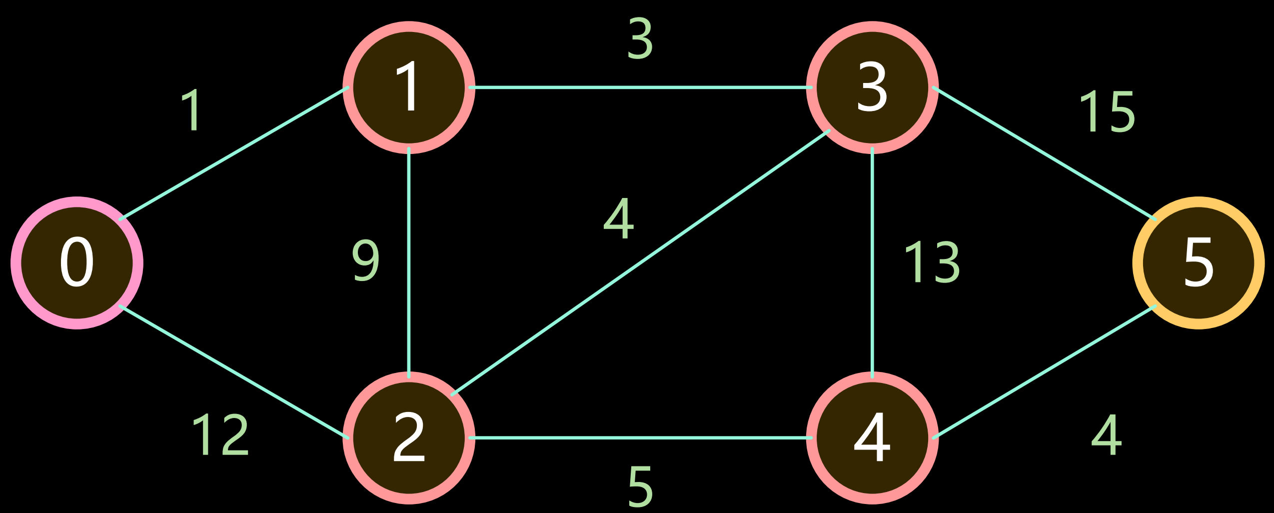
Index	0	1	2	3	4	5
Sign	√	√	√	√	√	×
Parent		0	3	1	2	4
Dis.	0	1	8	4	13	~~19~~ 17

问题：找出节点0到节点5的最短路径



Index	0	1	2	3	4	5
Sign	√	√	√	√	√	√
Parent		0	3	1	2	4
Dis.	0	1	8	4	13	17

问题：找出节点0到节点5的最短路径



Index	0	1	2	3	4	5
Sign	√	√	√	√	√	√
Parent		0	3	1	2	4
Dis.	0	1	8	4	13	17

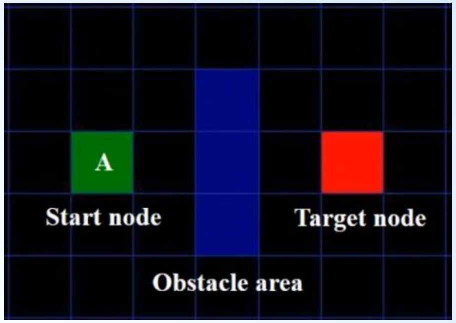
第7讲 时空关联规划

7.4 方法：A*算法

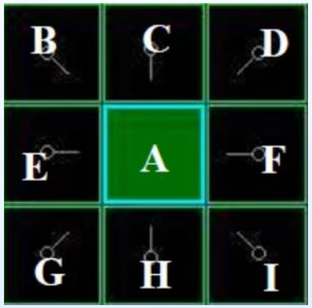
- 算法思想
- A^* 算法结合了贪心算法（深度优先）和Dijkstra算法（广度优先），是一种启发式搜索算法。
- 路径优劣评价公式为: $f(n) = g(n) + h(n)$
  - $f(n)$ 是从初始状态经由状态 n 到目标状态的代价估计。
  - $g(n)$ 是在状态空间中从初始状态到状态 n 的实际代价。
  - $h(n)$ 是从状态 n 到目标状态的最佳路径的估计代价。
  - 使用了两个状态表，分别称为openList表和closeList表。openList表由待考察的节点组成，closeList表由已经考察过的节点组成。

第7讲 时空关联规划

7.4 方法：A*算法

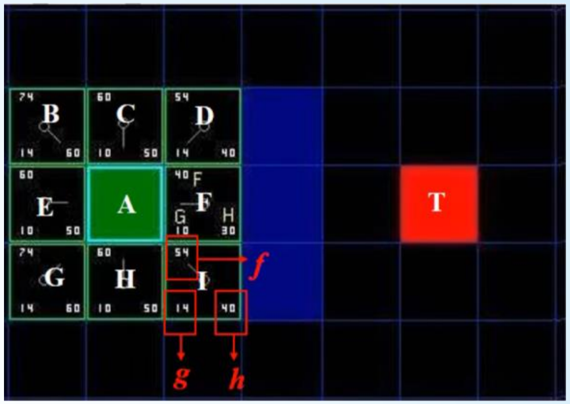


- 算法流程
- 预处理
  - 地图栅格化，并定义列表openList和closeList;(同Dijkstra算法中的U和S)
  - 初始时，定义 A 为父节点，节点 A 离自身距离为 0 ，路径完全确定，移入closeList中;
  - 父节点A周围共有8个节点，定义为子节点。将子节点放入 openList中，成为待考察对象。
  - 路径优劣判断依据是移动代价，单步移动代价采取Manhattan 计算方式，即把横向和纵向移动一个节点的代价定义为 10 。斜向移动代价参考等腰三角形计算斜边的方式距离为 14 。



第7讲 时空关联规划

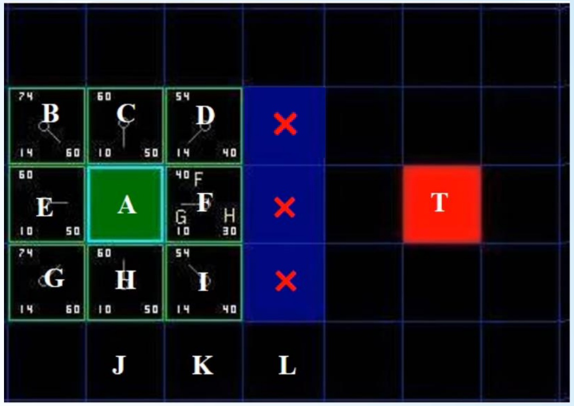
7.4 方法：A*算法



- 算法流程
- 开始搜索
  - 移动代价评价函数为： $f(n)=g(n)+h(n)$ 。 $f(n)$ 是从初始状态经由状态 n 到目标状态的代价估计， $g(n)$ 是在状态空间中从初始状态到状态 n 的实际代价， $h(n)$ 是从状态 n 到目标状态的最佳路径的估计代价。以节点I为例。
  - 首先考察 g ， 由于从 A 到该格子是斜向移动，单步移动距离为 14 ， 故 $g=14$.
  - 再考察估计代价 h 。估计指忽略剩下的路径是否包含有障碍物，完全按照Manhattan计算方式，计算只做横向或纵向移动的累积代价。以此类推，分别计算当前openList中余下的7个子节点的移动代价，挑选最小代价节点F， 移到closeList中。现在openList $=\{B,C,D,E,G,H,I\}$, closeList $=\{A,F\}$

第7讲 时空关联规划

7.4 方法：A*算法



- 算法流程
- 继续搜索
  - 从openList中选择 f值最小的节点I， 从 openList 里取出， 放到closeList中， 并把它作为新的父节点。
  - 检查所有与它相邻的子节点， 忽略障碍物不可走节点、忽略已经存在于closeList的节点; 如果方格不在openList 中， 则把它们加入到openList中。
  - 如果某个相邻的节点已经在 open list 中， 则检查这条路径是否更优， 判断经由当前节点 (我们选中的节点)到达那个节点是否具有更小的 G 值。如果没有， 不做任何操作。
  - 依次类推， 不断重复。一旦搜索到目标节点T， 完成路径搜索， 结束算法。

作业

- 用Dijkstra方法求下图中从A到D的最短路径