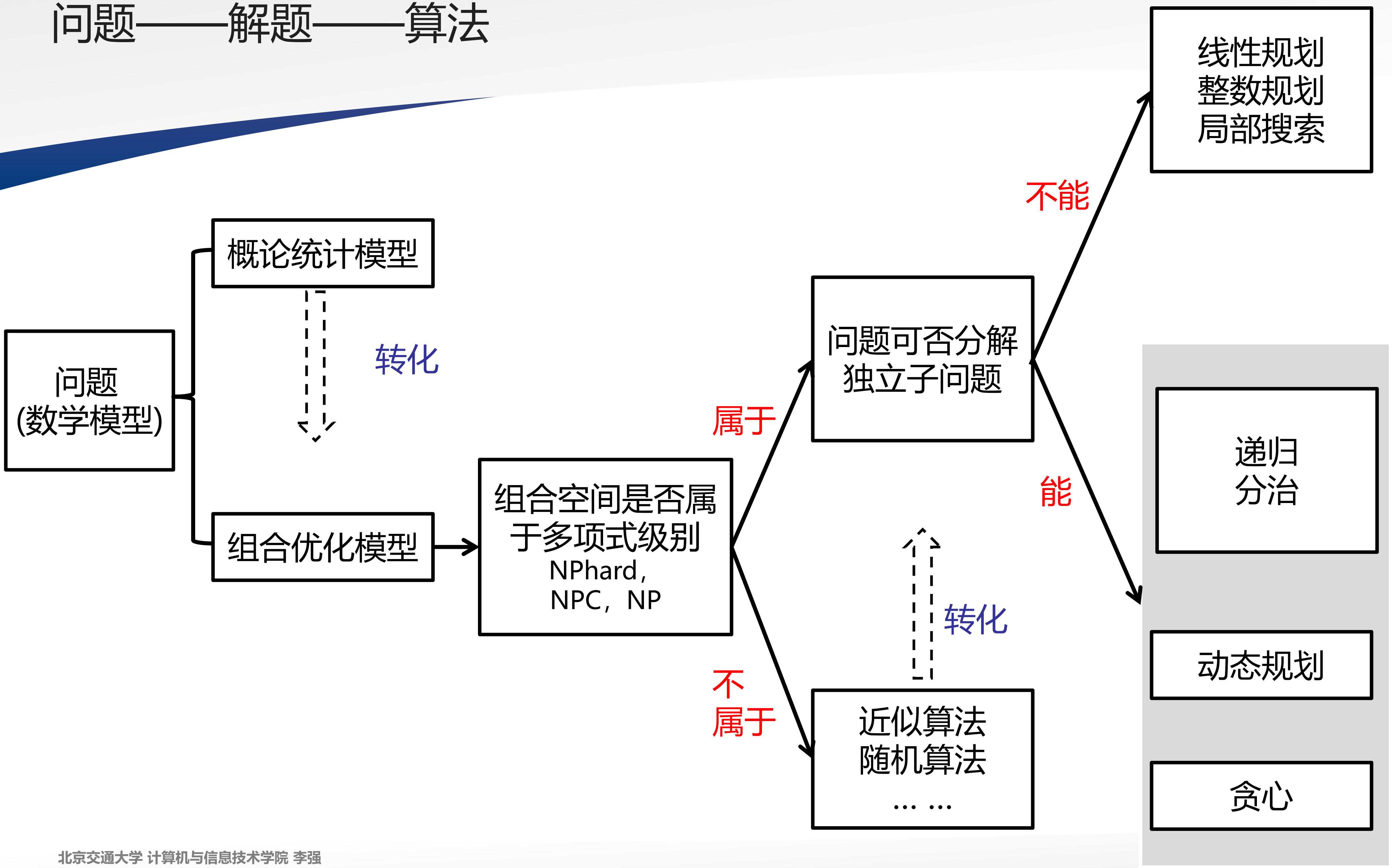
算法设计与分析
贪心

李强

计算机与信息技术学院

liqiang@bjtu.edu.cn

问题——解题——算法



线性规划
整数规划
局部搜索
不能
概论统计模型
转化
问题
(数学模型)
问题可否分解
独立子问题
属于
递归
分治
能
组合空间是否属于多项式级别
NPhard，
NPC，NP
组合优化模型
转化
动态规划
不
属于
近似算法
随机算法
… …
贪心

目录

贪心算法概述

贪心算法(Greedy Algorithm)

顾名思义，贪心算法总是作出在当前阶段看来最好的选择。也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择，是一种“ 短视 ”行为。

- 希望贪心算法得到的最终结果也是整体最优的。

不能对所有问题都得到整体最优解，但对许多问题能产生整体最优解。

- 在一些情况下，即使不能得到整体最优解，其最终结果却是最优解的很好 近似

Greedy algorithms build up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.

GA和DP的相似性

1. 都是为了解决组合优化问题
2. 存在Optimal substructure，贪心算法和动态规划，都需要找到原问题的最优子结构，即分解成独立子问题，且原问题的最优解包含（←→）了子问题的最优解
3. 在每一个贪心解法后面，总是存在一个比较麻烦的动态规划解法

--CRLS

GA和DP的差异性

1. 动态规划，需要枚举所有的独立子问题，从而得到最优解，再枚举完所有子问题前，不能确定得到最优解
2. 贪心算法，面对所有独立子问题，不去枚举所有的子问题的解，而是根据一个原则，选择其中一个子问题，而放弃其他的子问题的求解过程

Knapsack Problem背包问题

一个旅行者随身携带一个背包。可以放入背包的物品有一个旅行者随身携带一个背包。可以放入背包的物品有n种，每种物品的重量和价值分别为种，每种物品的重量和价值分别为w i , v i 。如果背包的最大承重限制是C，每种物品可以放多个。怎么样选择放入背包的物品使得背包所装物品，每种物品可以放多个。怎么样选择放入背包的物品使得背包所装物品 价值最大 ？

C = 10

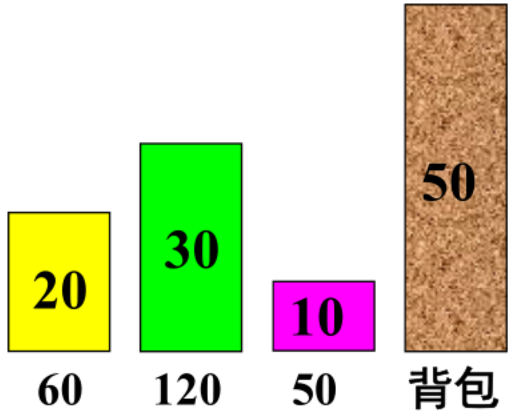
Item	Weight	Value
1	6	$30
2	3	$14
3	4	$16
4	2	$9

至少有三种看似合理的贪心策略：

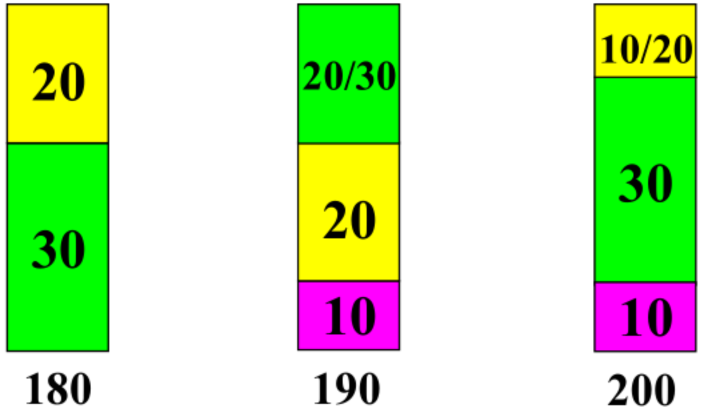
1. 选择 价值最大的物品，因为这可以尽可能快地增加背包的总价值。然而，虽然每一步选择获得了背包价值的极大增长，但背包容量却可能消耗得太快，使得装入背包的物品个数减少
2. 选择 重量最轻的物品，因为这可以装入尽可能多的物品，从而增加背包的总价值。但是，虽然每一步选择使背包的容量消耗得慢了，但背包的价值却没能保证迅速增长
3. 选择 单位重量价值最大的物品，在背包价值增长和背包容量消耗两者之间寻找平衡

一般背包问题实例

有 有3 个物品，其重量分别是{20, 30, 10}，价值分别为，价值分别为{60, 120, 50}，背包的容量为50



三个物品及背包

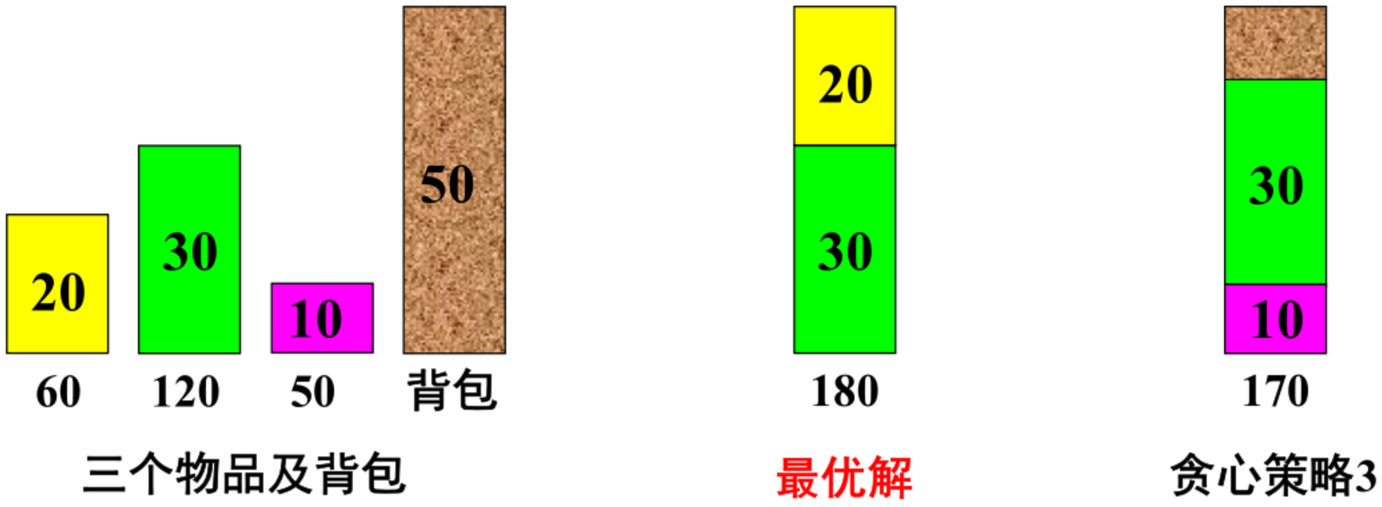


贪心策略1 贪心策略2 贪心策略3

策略3 能到达最优解吗？

0-1 背包问题实例

无法保证最终能将背包装满，部分闲置的背包空间使每千克背包空间的价值降低了



贪心算法的特点

贪心优势：算法简单，时空复杂度低

设计要素

- 依据某种“短视”的贪心选择性质判断，性质好坏决定算法的成败
- 贪心法 必须进行正确性证明
- 求解过程是多步判断过程，最终的判断序列对应于问题的最优解

最小生成树
Minimum spanning trees

无向连通带权图

G=(V,E,W)

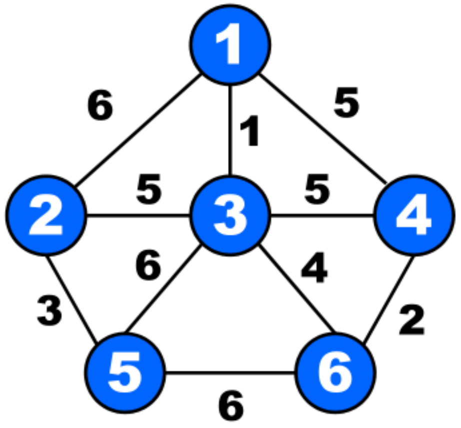
其中w(e) W 是边e 的权值

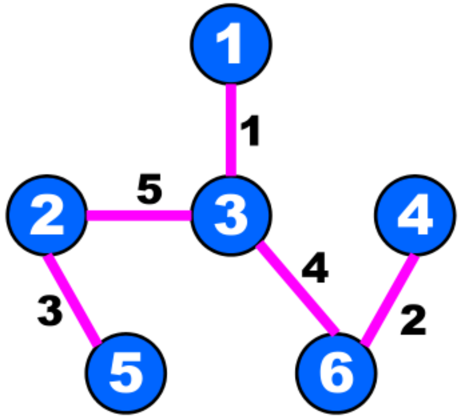
G 的一棵生成树T 是包含了G所有顶点的树，树中各边的权值之和所有顶点的树，树中各边的权值之和W(T) 称为树的 权，具有最小权的生成树称为G的 的 最小生成树

例子

G=(V, E, W), V={1,2,3,4,5,6}, W 如图所示。

E={{1,2},{1,3},{1,4},{2,3},{2,5},{3,4},{3,5},{3,6},{4,6},{5,6}}





求最小生成树

给定连通带权图G=(V,E,W), W(e) W 是边e 的权。求G 的一棵最小生成树

贪心法

- Prim 算法
- Kruskal 算法

生成树在网络中有着重要应用

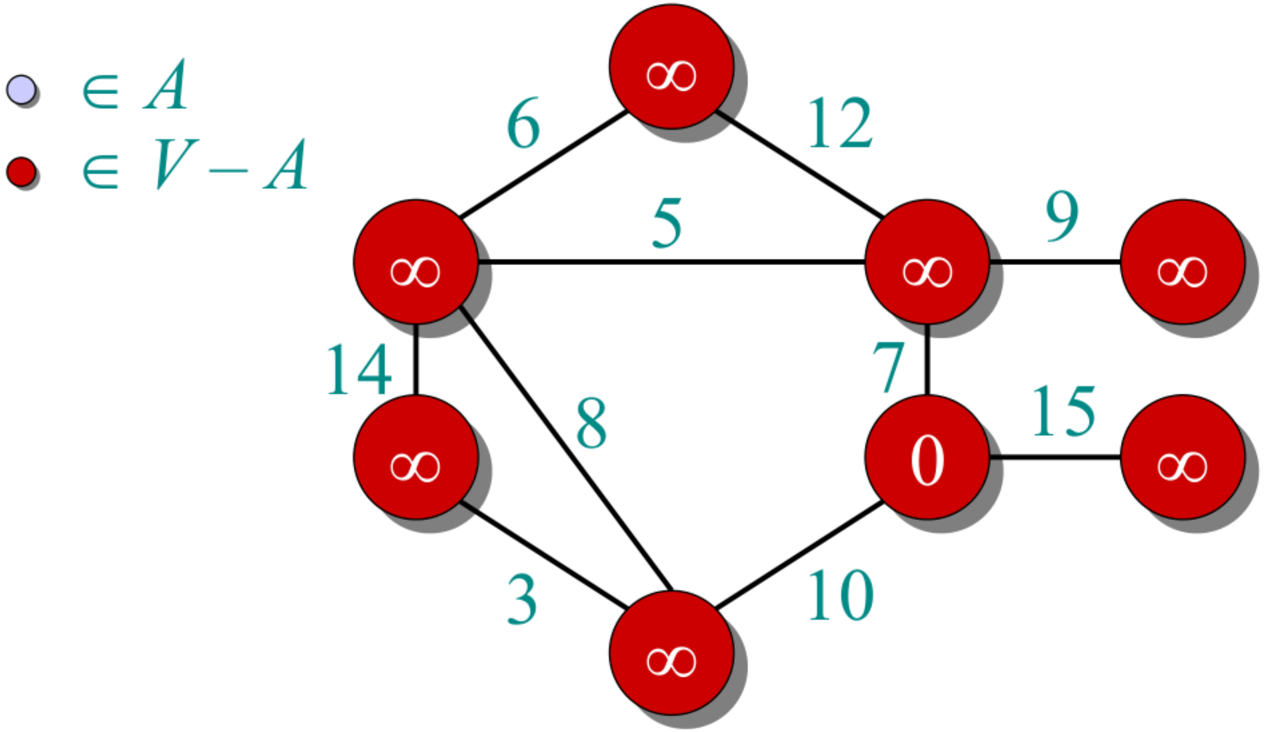
Prim

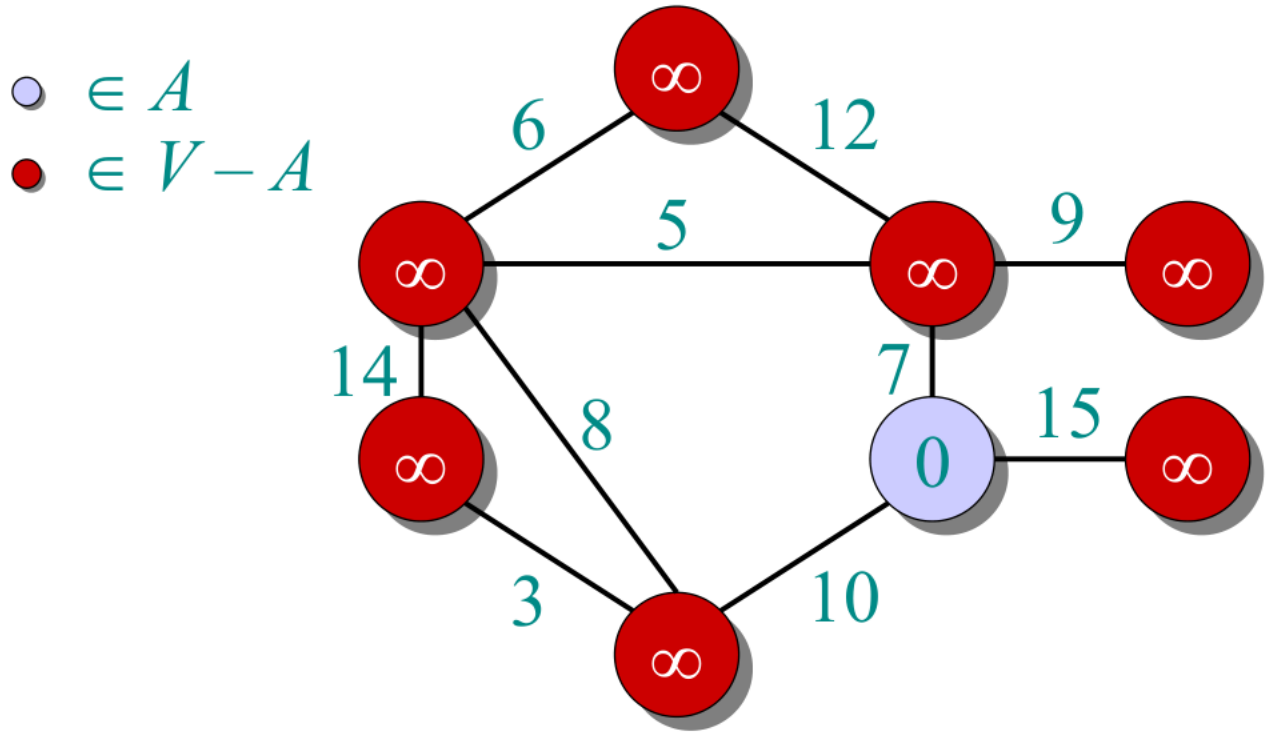
A 代表最小生成树的节点；V-A图中的节点减去最小生成树的节点

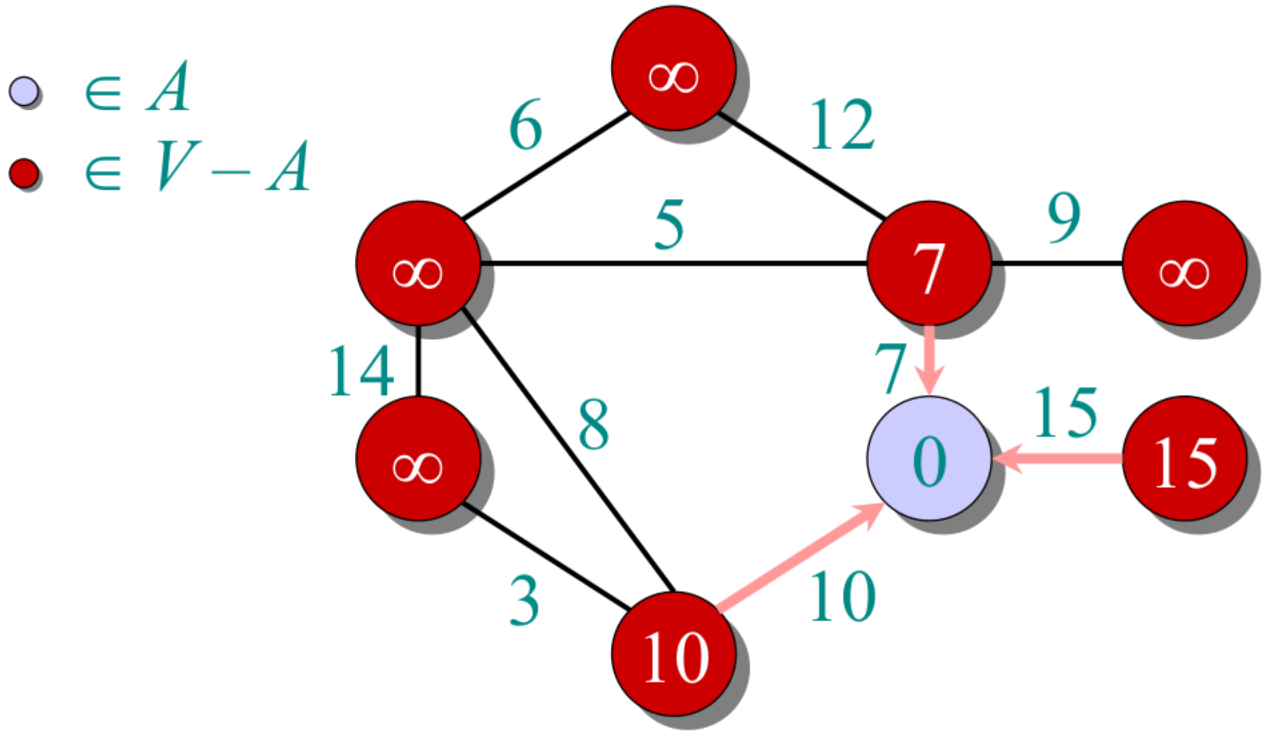
选择连接A 与V-A 集合的最短边e={i, j}，其中i∈S,j∈V-S。将将e 加入树T
直到V-A为空

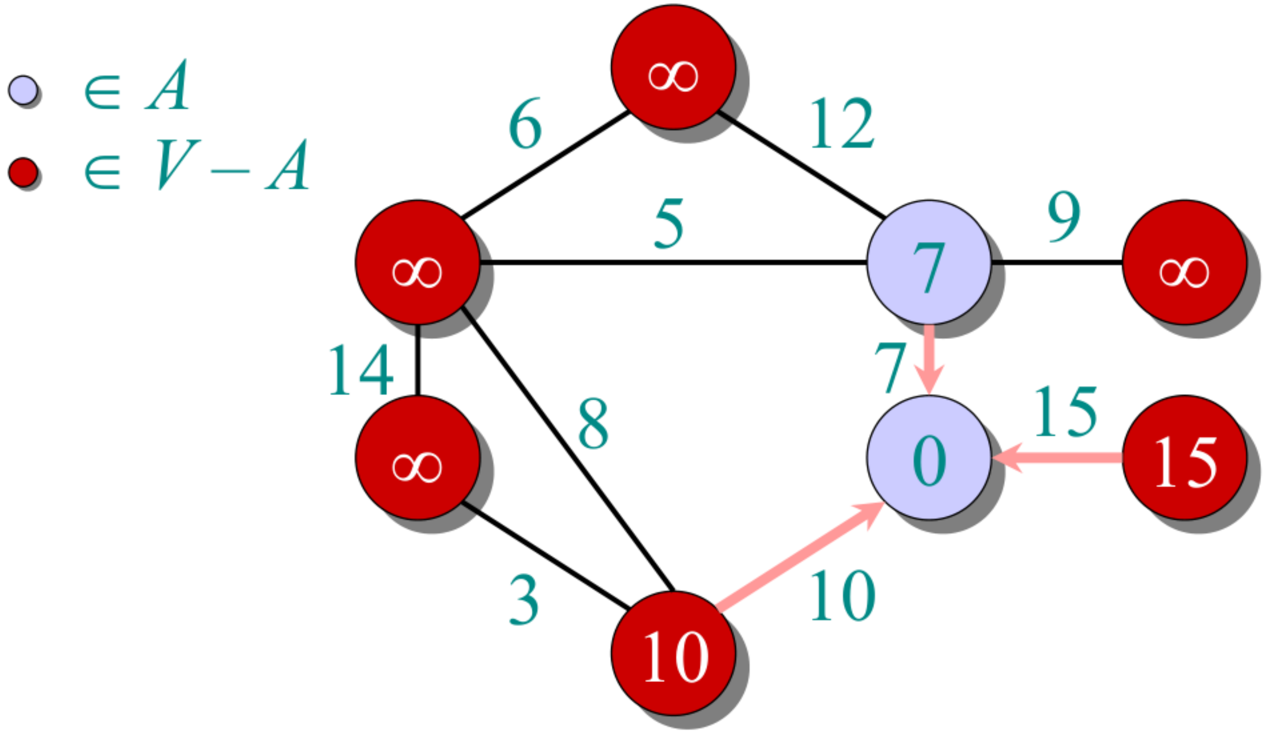
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $key[v] \leftarrow w(u, v)$ ▷ DECREASE-KEY
          $\pi[v] \leftarrow u$

Prim例子

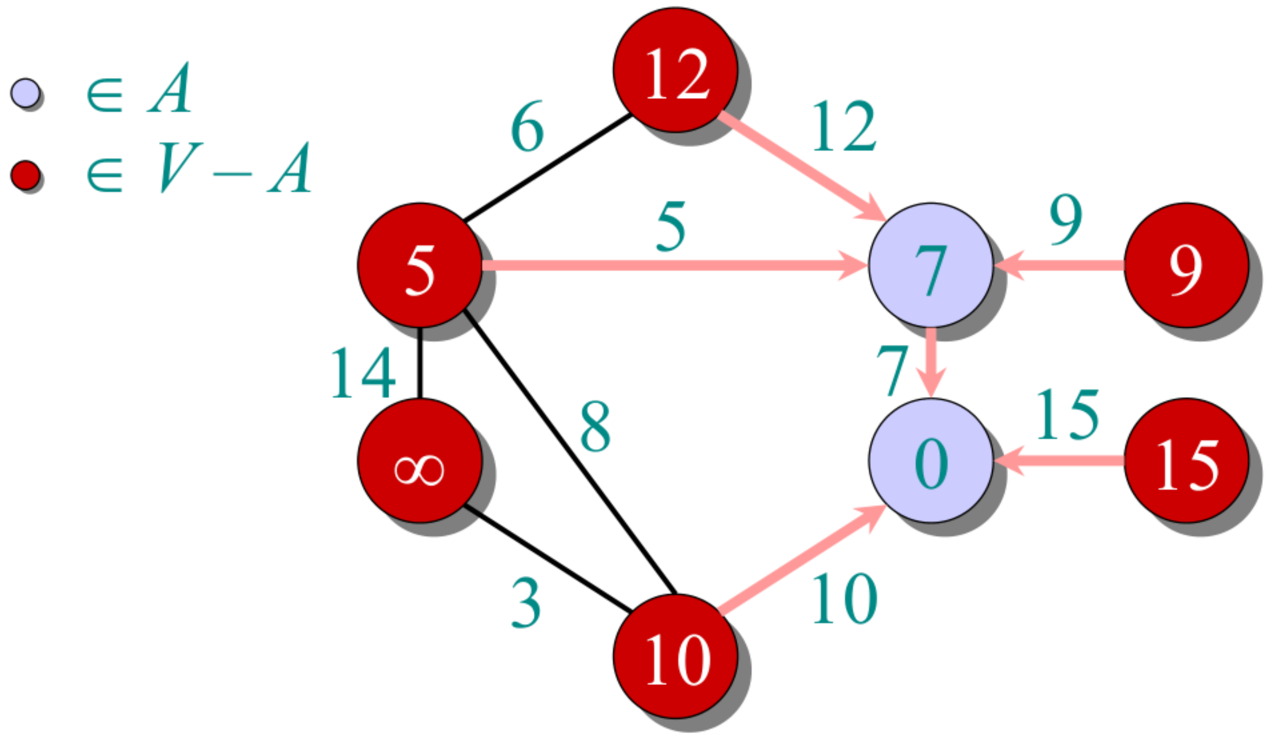



$\in A$
$\in V-A$
6
12
5
9
14
7
8
15
3
10
0
∞

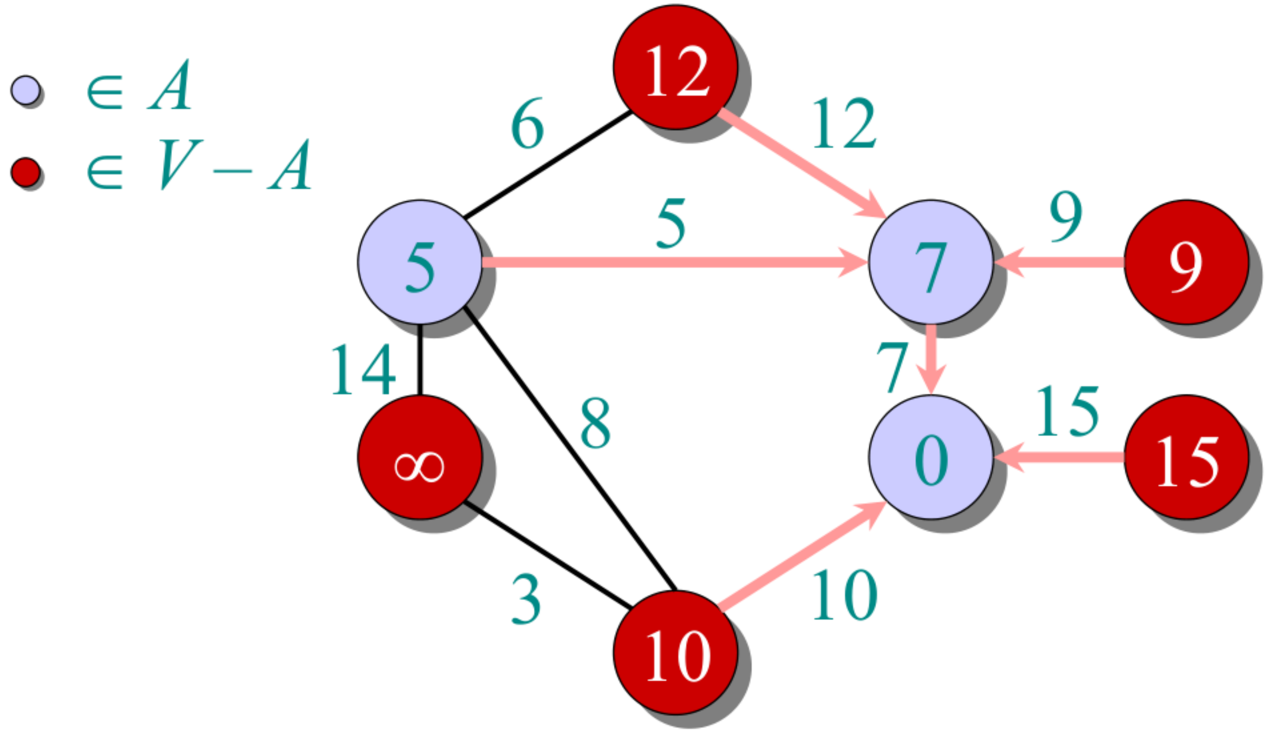

$\in A$
$\in V - A$
∞
6
12
5
∞
7
9
∞
14
7
8
15
∞
0
15
3
10
10

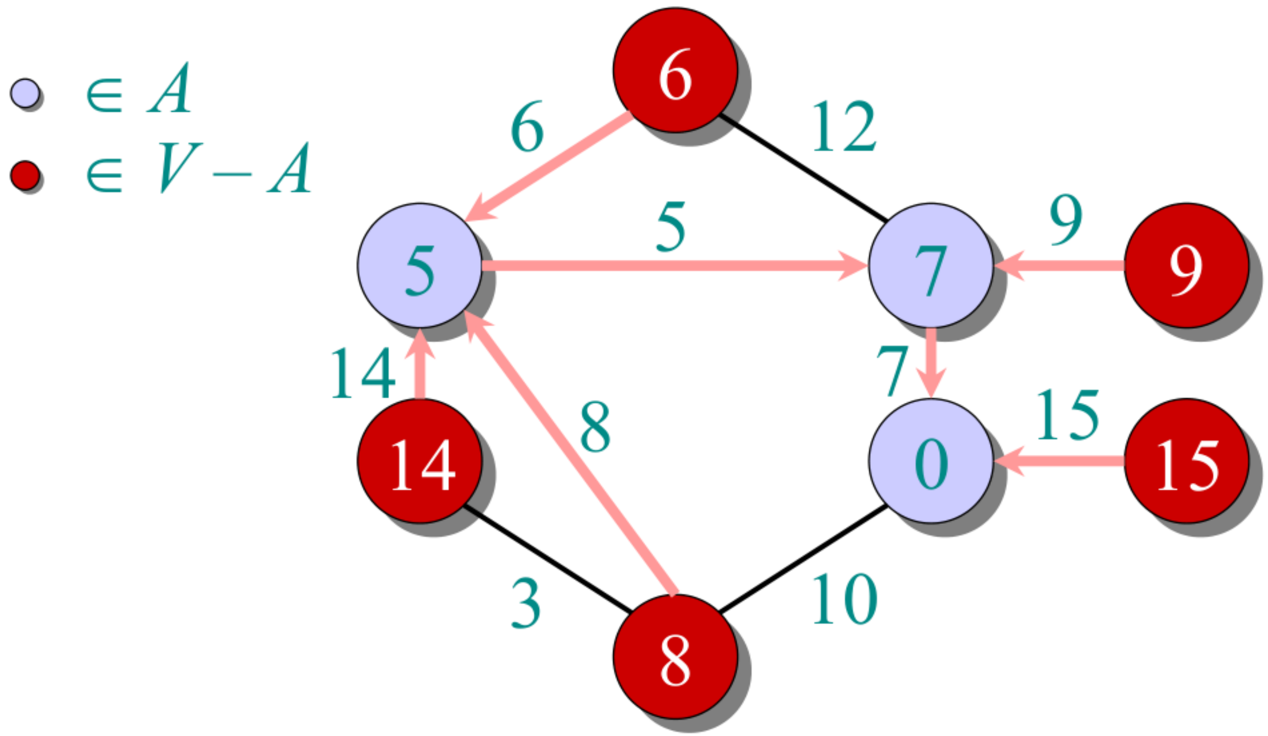


$\in A$
$\in V - A$
∞
6
12
5
∞
7
9
∞
14
7
8
15
∞
0
15
3
10
10

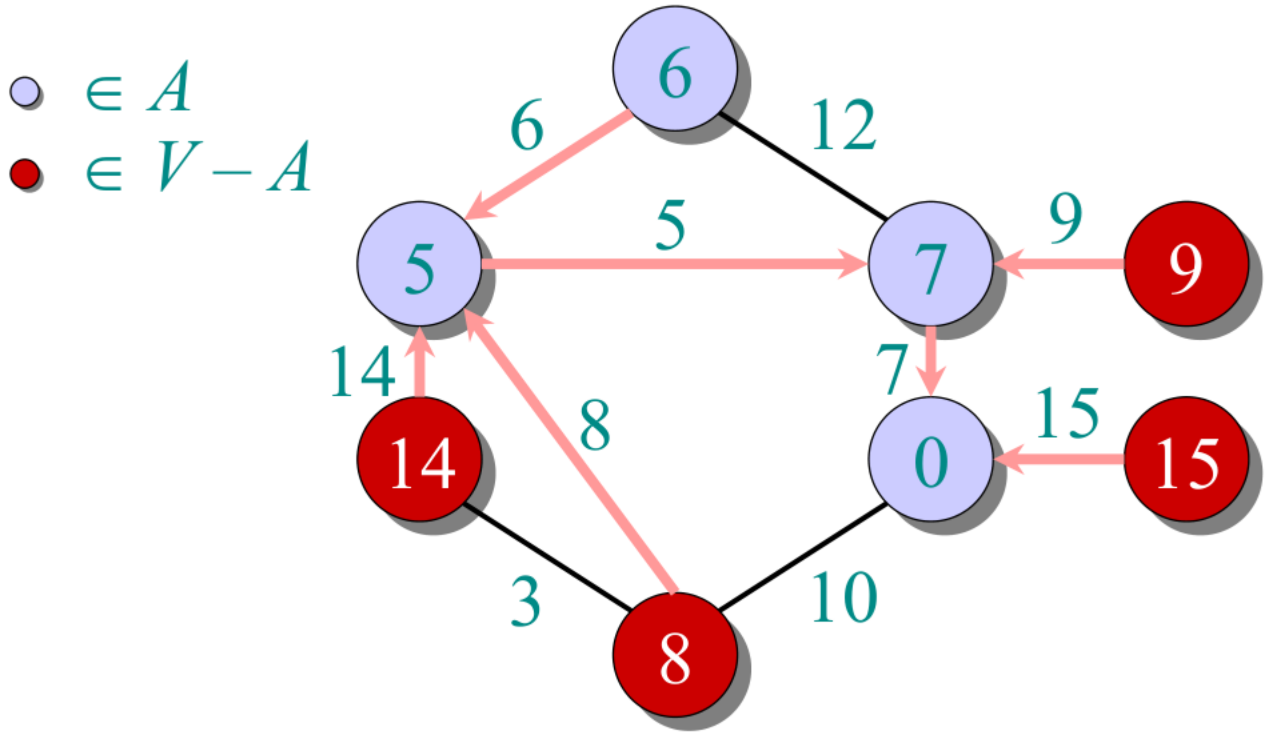


$\in A$
$\in V - A$
12
6
12
5
5
7
9
9
14
7
8
∞
0
15
15
3
10
10

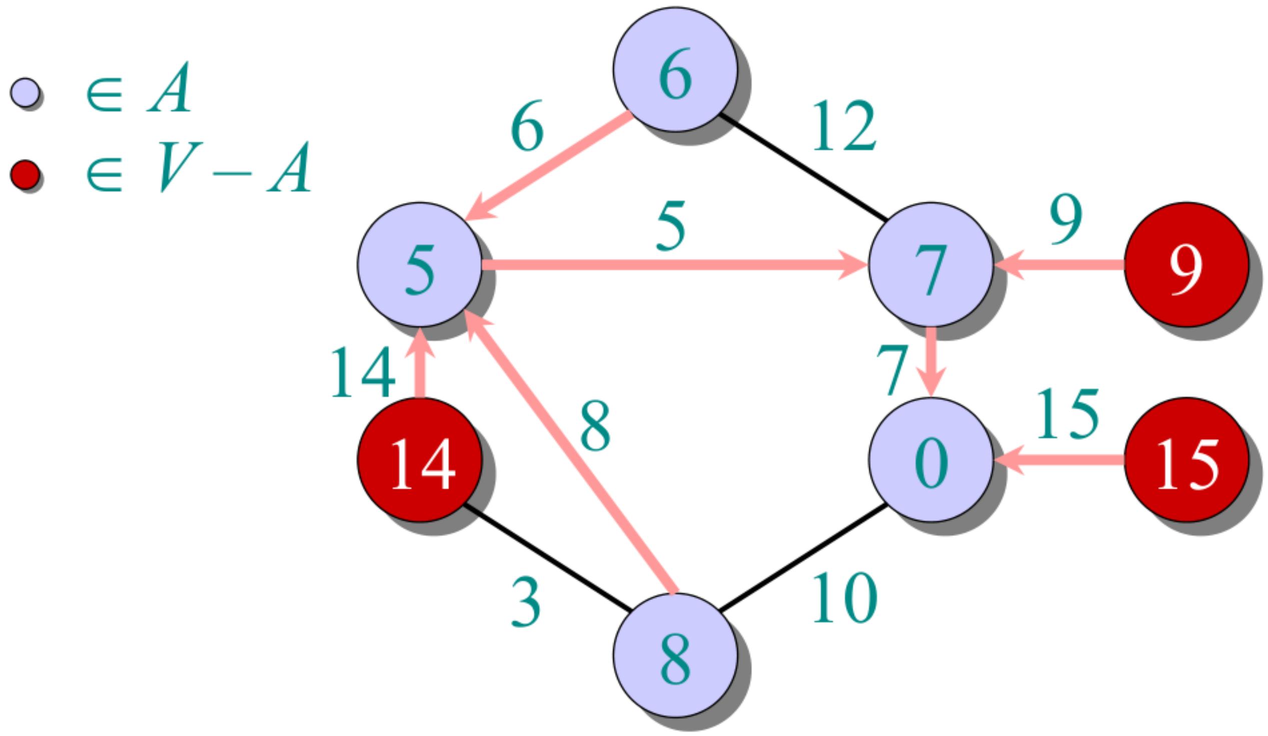

∈ A
∈ V − A
12
6
12
5
5
7
9
9
14
7
8
∞
0
15
15
3
10
10



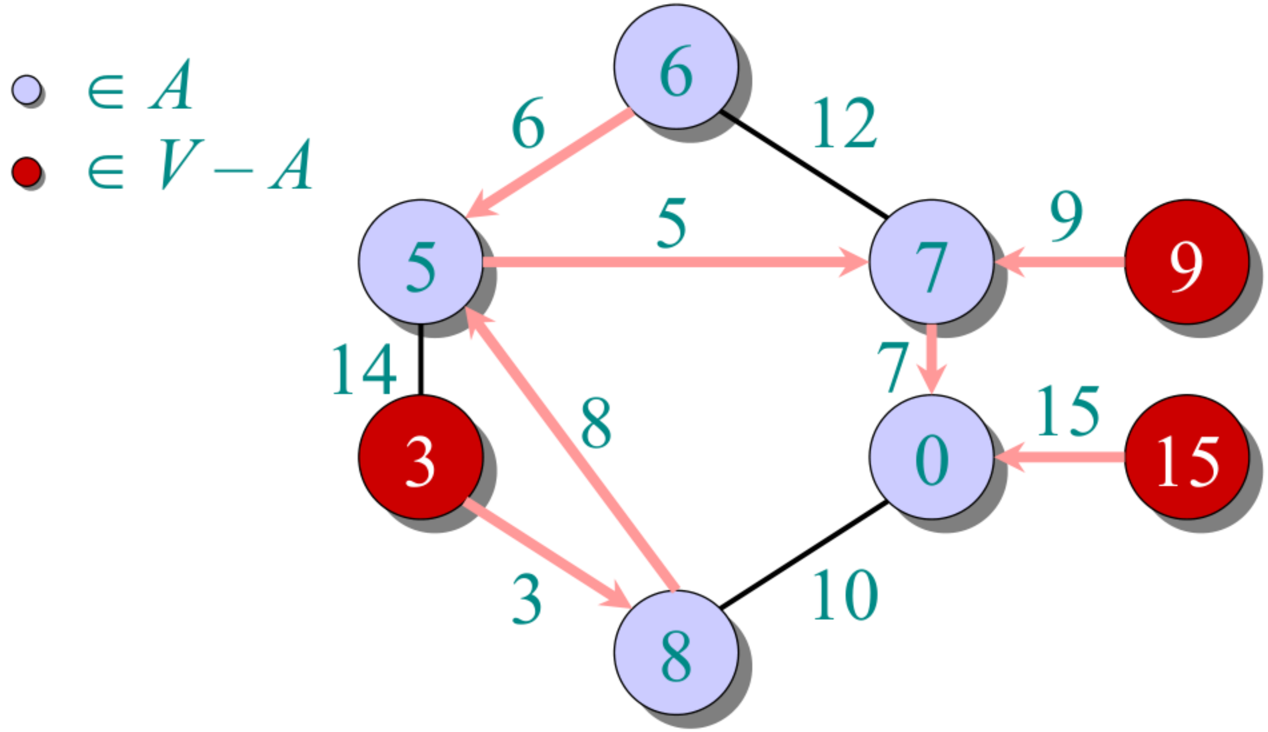
$\in A$
$\in V - A$
6
12
5
9
14
8
7
15
3
10
6
5
7
9
14
0
15
8



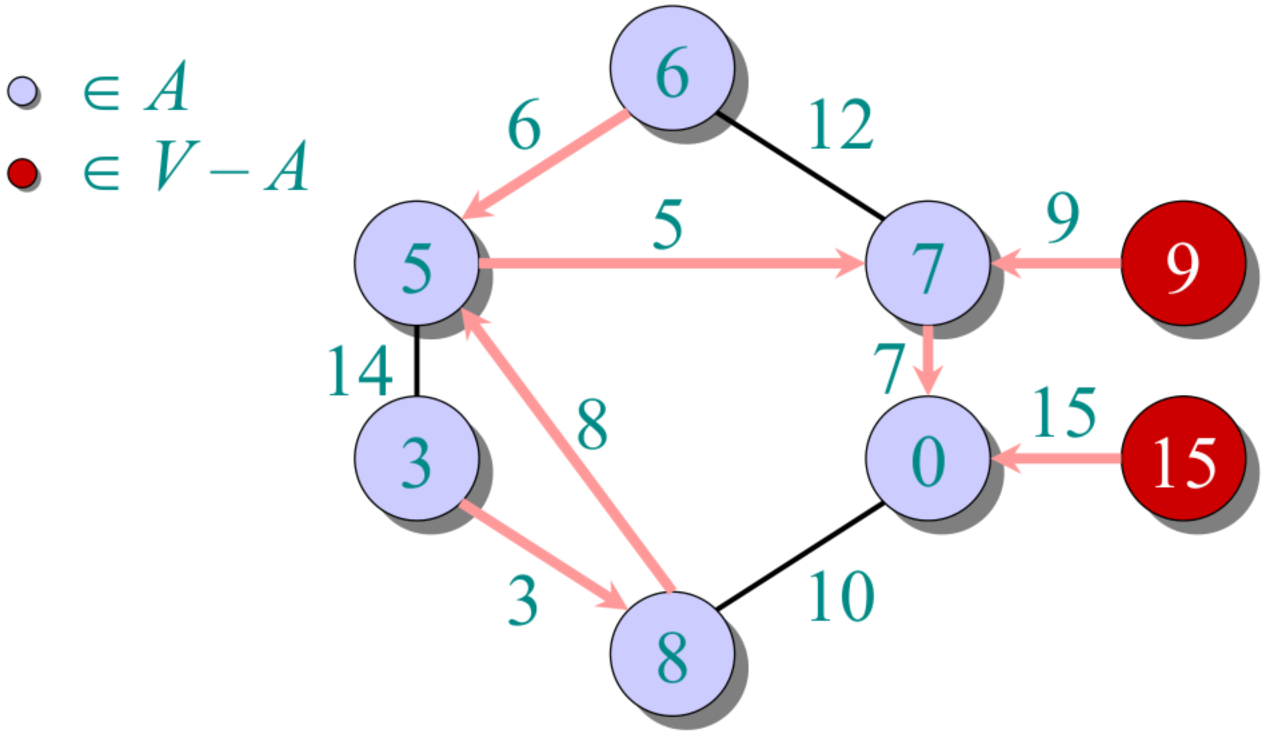
$\in A$
$\in V - A$
6
6
12
5
5
7
9
9
14
7
8
14
0
15
15
3
8
10



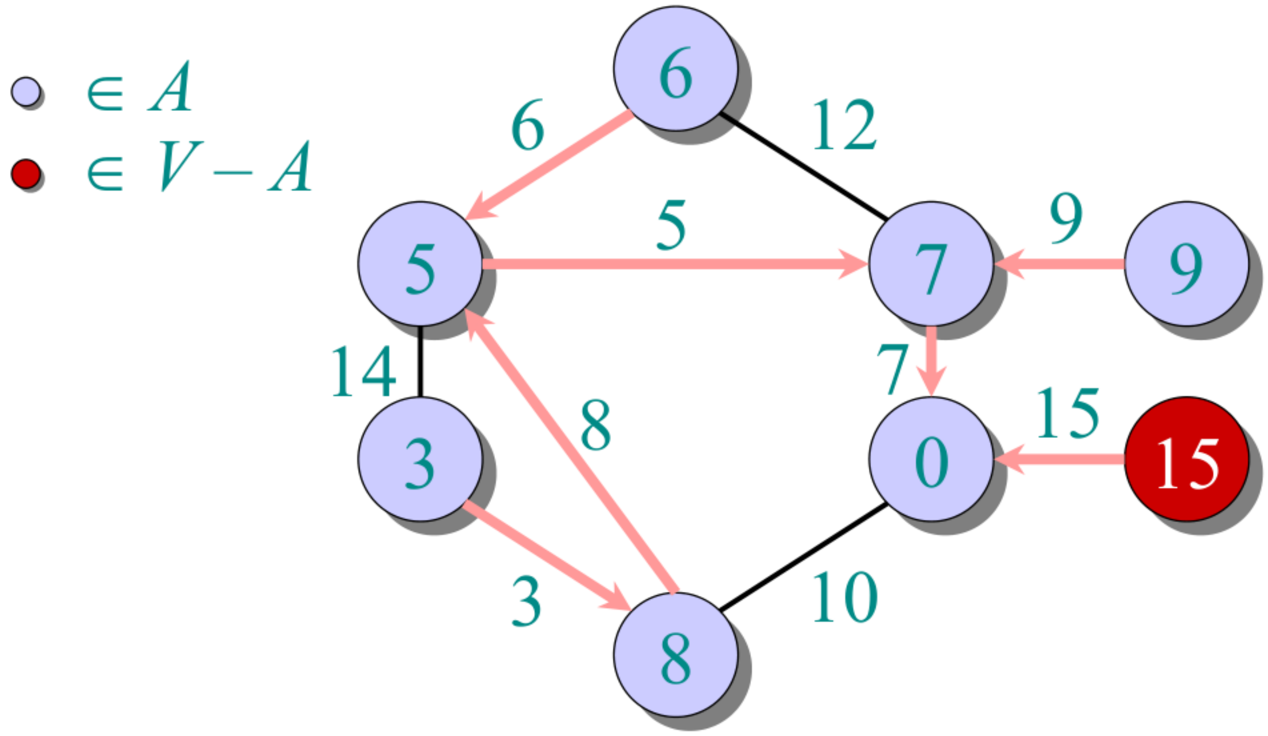
$\in A$
$\in V - A$
6
6
12
5
5
7
9
9
14
7
8
14
0
15
15
3
10
8



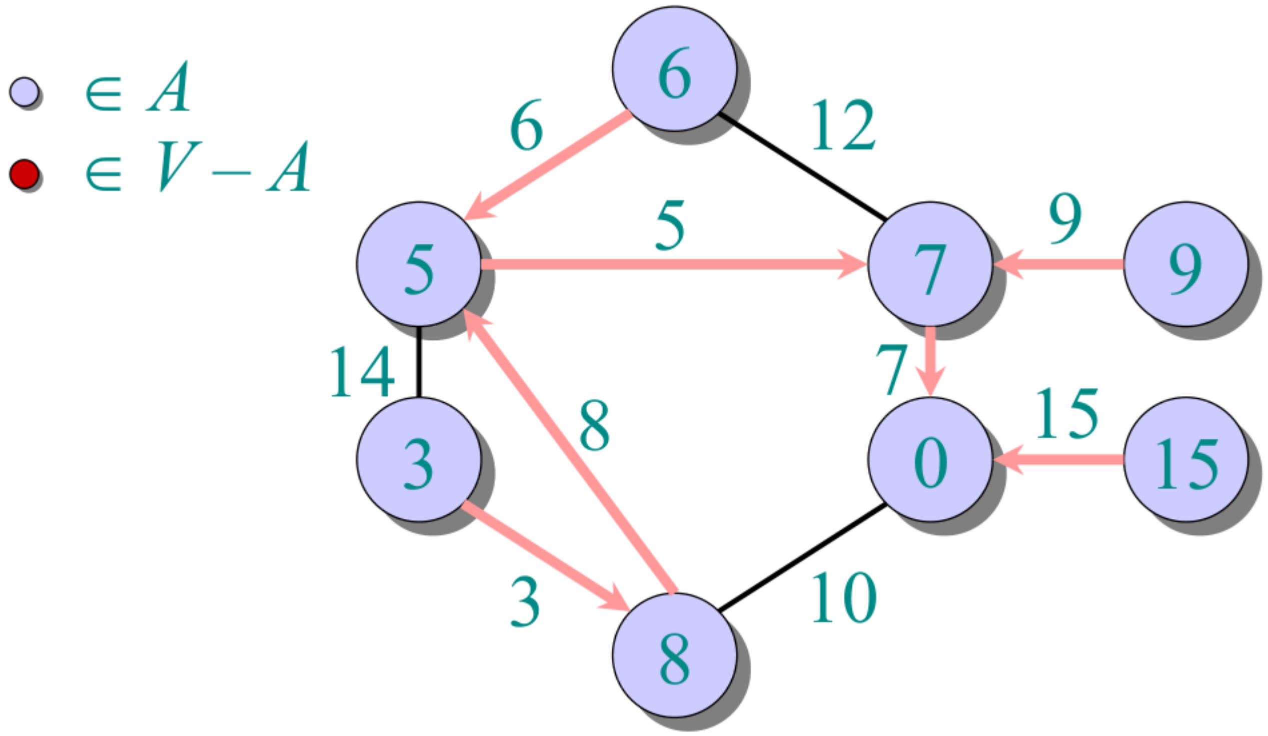
$\in A$
$\in V - A$
6
12
6
5
9
5
7
9
14
7
3
8
15
0
15
3
10
8



∈ A
∈ V − A
6
12
6
5
5
7
9
9
14
7
8
3
0
15
15
3
10
8



$\in A$
$\in V - A$
6
12
6
5
5
7
9
9
14
7
15
8
3
0
15
3
10
8



$\in A$
$\in V - A$
6
6
12
5
5
7
9
9
14
7
8
3
0
15
15
3
10
8

Kruskal 算法

- 给定无向带权图$G=(V,E)$，$V=\{1,2,\dots,n\}$。设最小生成树$MST\ =(V,\ TE)$，该树的初始状态为**只有n个顶点而无边**的非连通图$T\ =\ (V,\{\}\ )$，　Kruskal算法将这n个顶点看成是n个**孤立的连通分支**；

- **贪心选择**：在边集E中选择**权值最小**的边(i,j)，如果将边(i,j)加入TE中**不产生回路**，则将边(i,j)**加入**TE，即用边(i,j)将T中的两个连通分支合并成一个联通分支；否则**舍弃**它。循环此过程，直至所有顶点都在一个联通分支

Kruskal算法俗称**避环法**，该算法的实现关键是在加入边时**避免出现环路**。那么，怎么判断加入某条边后图T中不会出现回路？

```
procedure kruskal(G,w)

for all u ∈ V :
        makeset(u)

X = {}
Sort the edges E by weight
for all edges {u,v} ∈ E, in increasing order of weight:
        if find(u) != find(v):
                add edge {u,v} to X
                union(u,v)
```

procedure makeset(x)
$\pi(x) = x$
rank(x) = 0

function find(x)
while x != $\pi(x)$: x = $\pi(x)$
return x

procedure union(x,y)
r_x = find(x)
r_y = find(y)
if $r_x = r_x$: return
if rank(r_x) > rank(r_y):

    $\pi(r_y) = r_x$

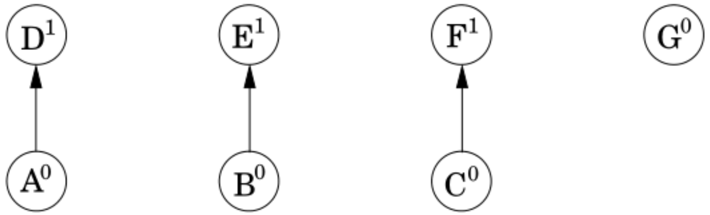
else:

    $\pi(r_x) = r_y$
    if rank(r_x) = rank(r_y) :

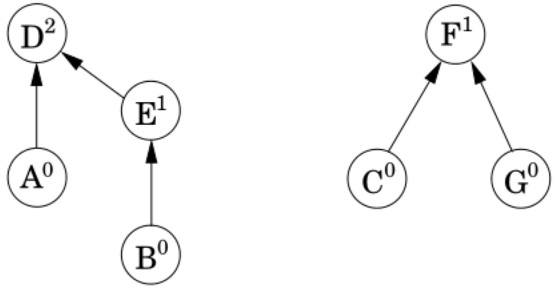
        rank(r_y) = rank(r_y) + 1

After makeset(A), makeset(B), …, makeset(G):

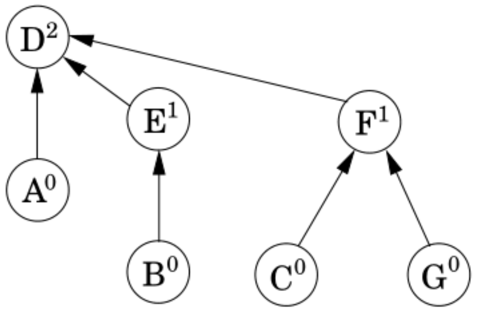
After union(A, D), union(B, E), union(C, F):



After $\texttt{union}(C, G), \texttt{union}(E, A)$:

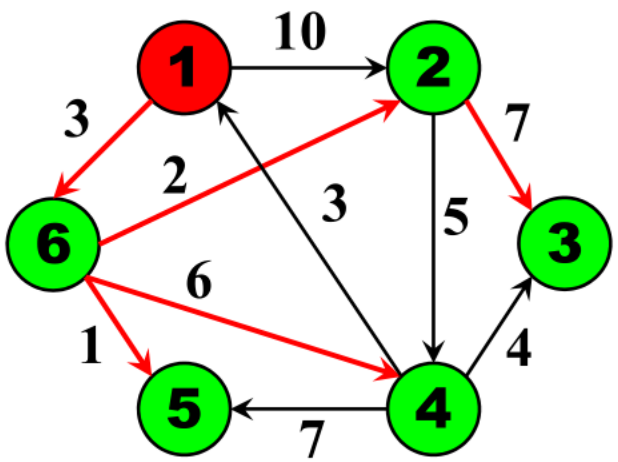


After $\texttt{union}(B, G)$:



最短路径
Shortest paths

问题描述：给定带权有向图G = (V, E)，其中每条边的权是非负实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路径长度，假设从源可以到达任何一个顶点。这里路径的长度是指路径上各边权之和。这个问题通常称为*单源*最短路径问题。



源点：1

1→6→2: short[2]=5

1→6→2→3 : short[3]=12

1→6→4 : short[4]=9

1→6→5 : short[5]=4

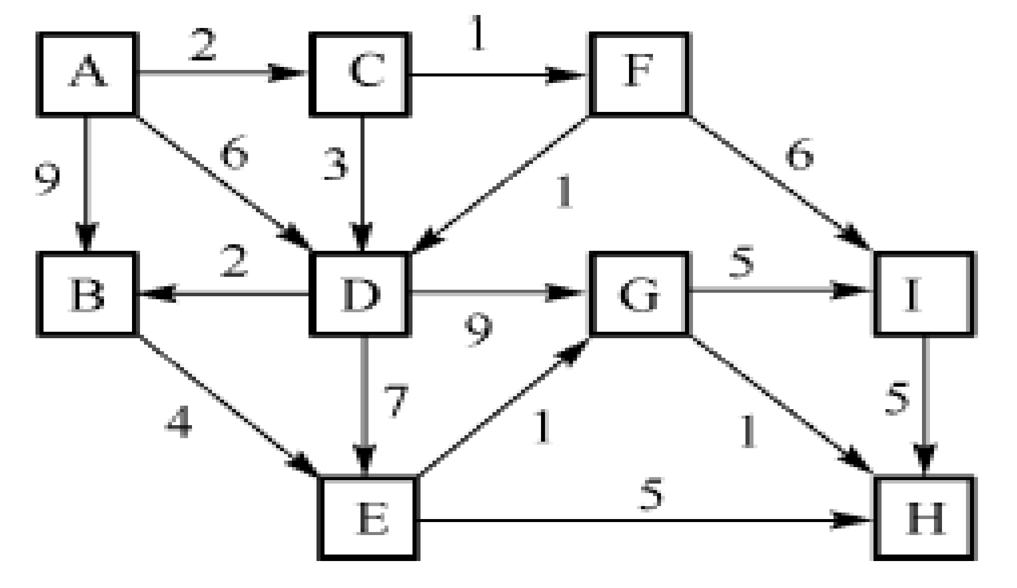
1→6 : short[6]=3

算法Dijkstra

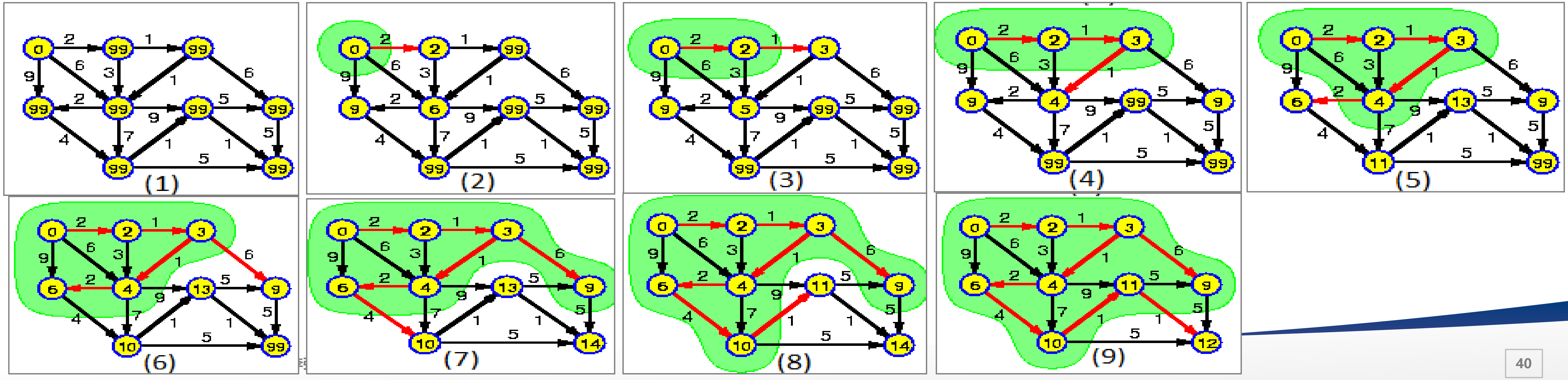
```
procedure dijkstra(G, l, s)
Input:    Graph G = (V, E), directed or undirected;
          positive edge lengths {l_e : e ∈ E}; vertex s ∈ V
Output:   For all vertices u reachable from s, dist(u) is set
          to the distance from s to u.

for all u ∈ V:
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0

H = makequeue(V)   (using dist-values as keys)
while H is not empty:
    u = deletemin(H)
    for all edges (u, v) ∈ E:
        if dist(v) > dist(u) + l(u, v):
            dist(v) = dist(u) + l(u, v)
            prev(v) = u
            decreasekey(H, v)
```

dist(v) =
min{dist(v),dist(u) + l(u,v)}

例子



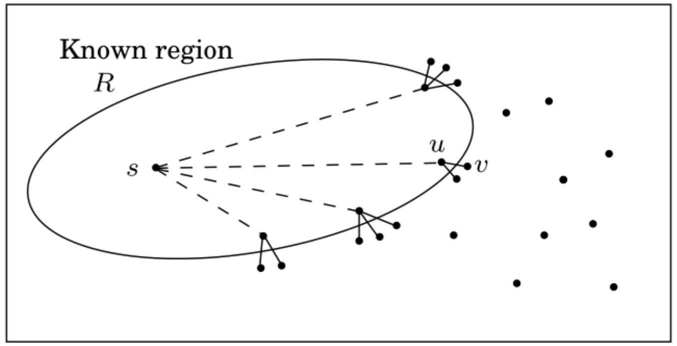
0	{}		{0, 99, 99, 99, 99, 99, 99, 99, 99}
1	{A}	A	{0, 9, 2, 6, 99, 99, 99, 99, 99}
2	{A, C}	C	{0, 9, 2, 5, 99, 3, 99, 99, 99}
3	{A,C,F}	F	{0, 9, 2, 4, 99, 3, 99, 99, 9}
4	{A,C,F,D}	D	{0, 6, 2, 4, 11, 3, 13, 99, 9}
5	{A,C,F,D,B}	B	{0, 6, 2, 4, 10, 3, 13, 99, 9}
6	{A,C,F,D,B,I}	I	{0, 6, 2, 4, 10, 3, 13, 14, 9}
7	{A,C,F,D,B,I,E}	E	{0, 6, 2, 4, 10, 3, 11, 14, 9}
8	{A,C,F,D,B,I,E,G}	G	{0, 6, 2, 4, 10, 3, 11, 12, 9}



如果边为负，怎么办

如果存在负边

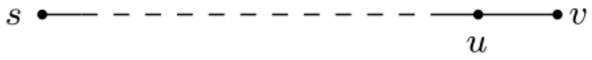
算法Dijkstra 不能工作，基于Cut 原理



如果存在负边

dist(v) = min{dist(v),dist(u) + l(u,v)}

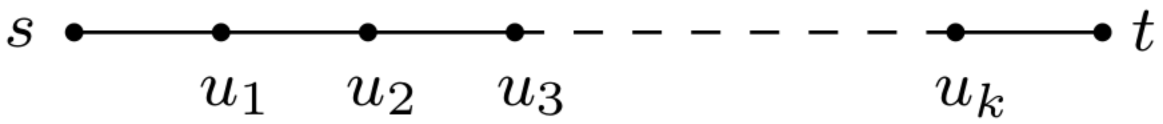
这条规则是否工作？



假设s到v的最短路径经过u，那么u一定在known region之中

(s,u) + l(u,v)　在known region遍历u，

如果边为负，怎么办



无论存在负边，还是不存在

源点s到目的节点t的路径，最长，不会超过|V | − 1边？

如果超过，必然会存在cycle

dist(v) = min{dist(v),dist(u) + l(u,v)}
只要更新|V | − 1次，
即使存在负边，也没有关系

Bellman-Ford algorithm

procedure shortest-paths(G,l,s)

for all $u \in V$:

    dist(u) = ∞

    prev(u) = nil

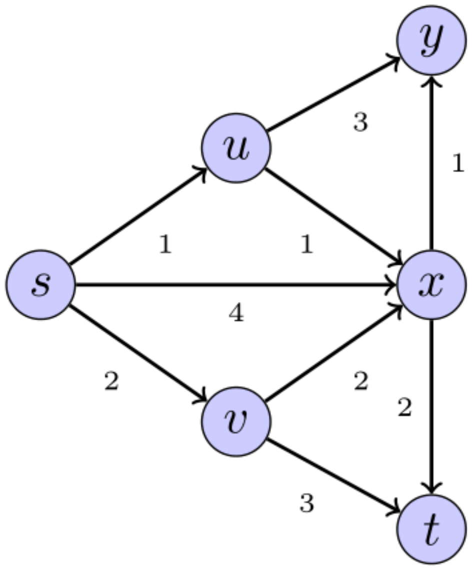
dist(s) = 0

repeat $|V| - 1$ times:

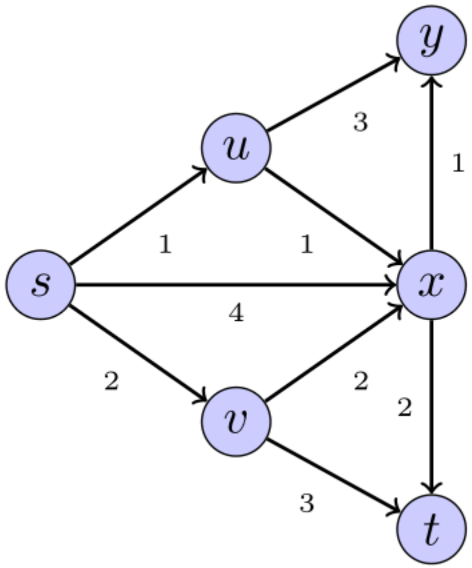
    for all edge $(u,v) \in E$:

        dist(v) = min{dist(v), dist(u) + l(u,v)}

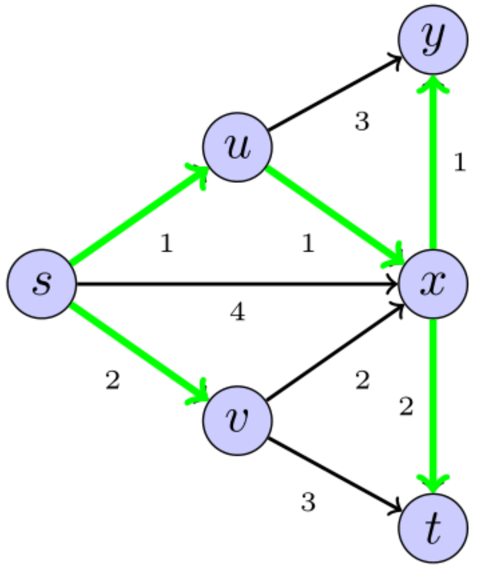
如果存在cycle 为负，怎么办？



	k=0	1	2	3	4	5
s	0	0	0	0	0	0
u	–	1				
v	–	2				
x	–	4				
y	–	–				
t	–	–				

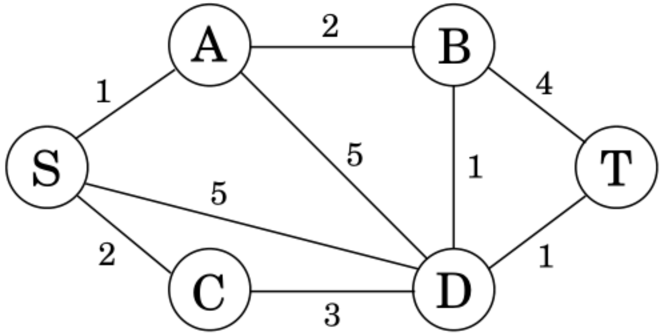


	k=0	1	2	3	4	5
s	0	0	0	0	0	0
u	–	1	1			
v	–	2	2			
x	–	4	2			
y	–	–	4			
t	–	–	5			



	k=0	1	2	3	4	5
s	0	0	0	0	0	0
u	–	1	1	1	1	1
v	–	2	2	2	2	2
x	–	4	2	2	2	2
y	–	–	4	3	3	3
t	–	–	5	4	4	4

最短可靠路径



S 到 T的所有路径：

S-A-B-T

S-C-D-T

S-D-T

S-A-D-T

霍夫曼编码
Huffman encoding

二元前缀码是广泛用于数据 文件压缩的编码方法，其使用字符在文件中出现的频率表来建立一个用的编码方法，其使用字符在文件中出现的频率表来建立一个用0,1串表示各个字符的最优表示方式

Google

Draco 3D compressor

facebook

Zstandard

pied piper

最优前缀码

二元前缀码

- 用 用0-1字符串作为代码表示字符，要求任何字符的代码都不能作为其他字符代码的前缀
- 非前缀码的例子

  a: 001, b: 00, c:010, d:01
- 解码的歧义，例如字符串0100001

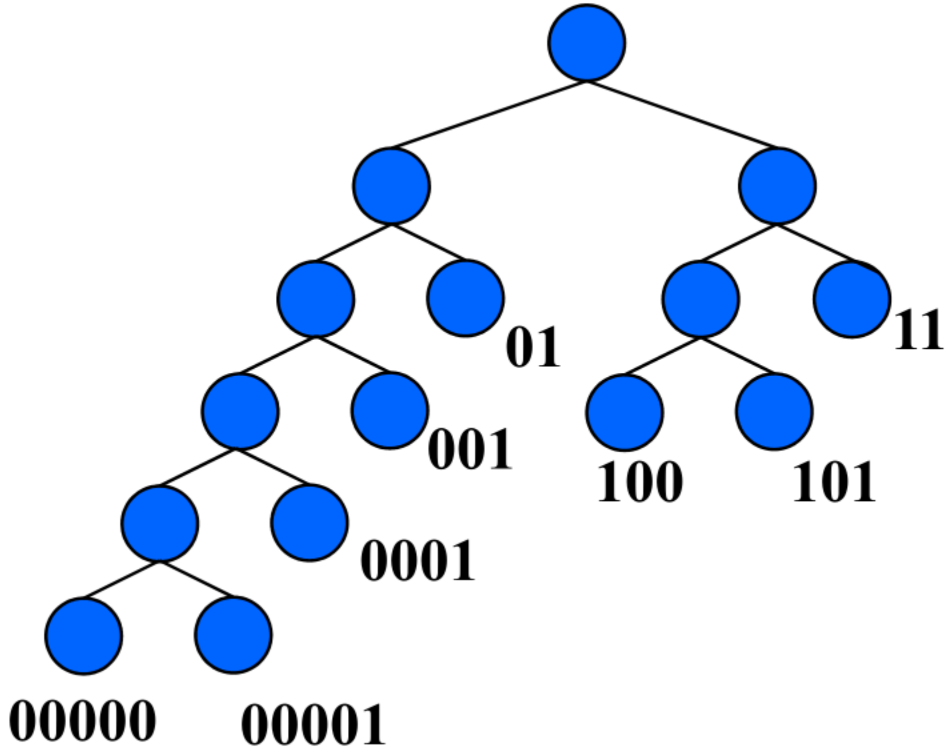
  解码1： 01, 00, 001 d, b, a

  解码2： 010, 00, 01 c, b, d

前缀码：
{00000, 00001, 0001, 001, 01, 100, 101, 11}

构造树：

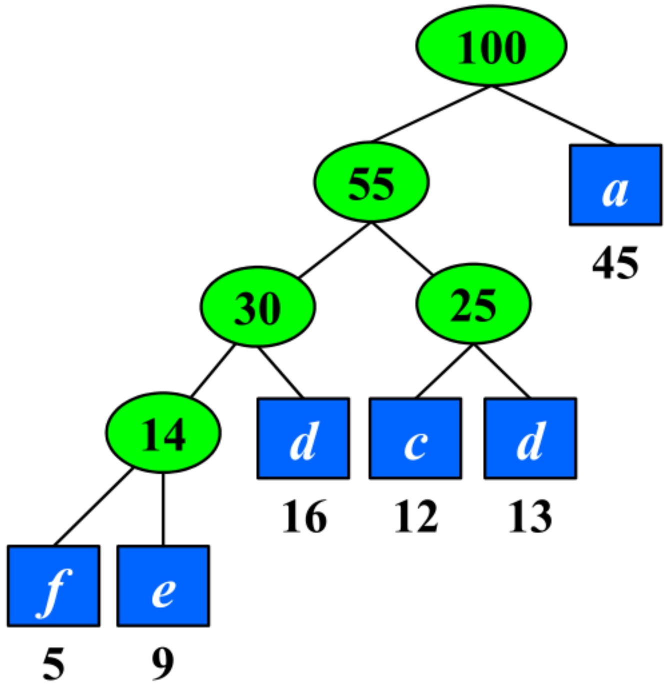
0- 左子树
1- 右子树
码对应一片树叶
最大位数为树深



给定字符集C={x 1 ,x 2 ,...,x n } 和每个字符的频率f(x i), i=1,2,...,n 。求关于C 的一个 最优前缀码（平均传输位数最小）

输入： a:45, b:13, c:12, d:16, e:9, f:5

- f → 0000
- e → 0001
- d → 001
- c → 010
- b → 011
- a → 1

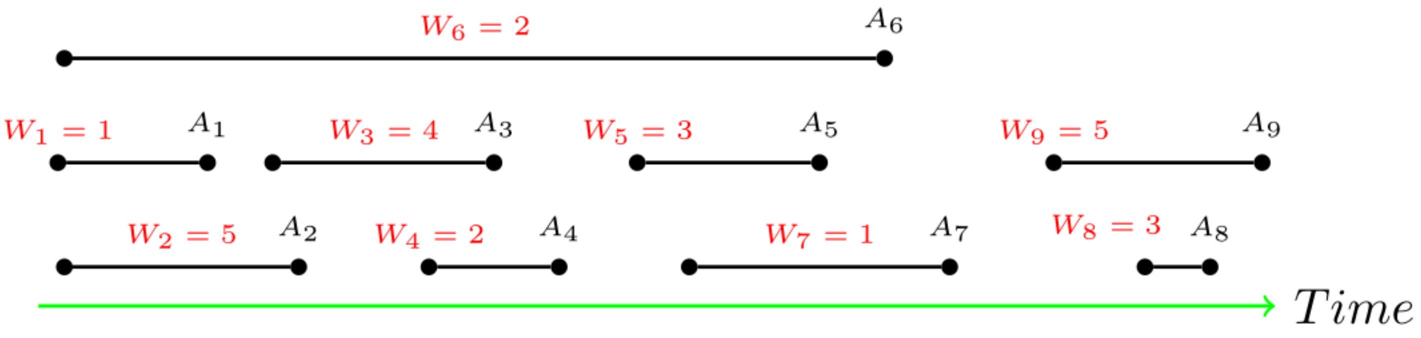


```
Huffman(C)
C={x 1 ,x 2 ,...,x n }, f(x i ), i=1,2,...,n
        n ← |C|
        Q ← C
         for i ← 1 to n-1:
                z ← Allocate-Node()
                 z.left ← Q 中最小元
                 z.right ← Q 中最小元
                f(z)← f(x)+f(y)
                Insert(Q, z)
        return Q
```

活动安排问题
Interval Scheduling

问题描述：假设某社团某一天要组织n个活动$A=\{1,2,\cdots,n\}$，其中每个活动都要求使用同一礼堂，而且在同一时间内只有一个活动能使用这个礼堂。每个活动i都有一个要求使用礼堂的起始时间s_i和结束时间f_i，且有$s_i<f_i$。如果选择了活动i，则它在半开时间区间$[s_i, f_i)$内占用资源。**若区间$[\boldsymbol{s_i}, \boldsymbol{f_i})$与区间$[\boldsymbol{s_j}, \boldsymbol{f_j})$不相交，则称活动i与活动j是相容的。**现在给定$n$个活动的开始时间和结束时间，请设计一个活动安排方案，使得安排的相容活动数目最多。

A general form



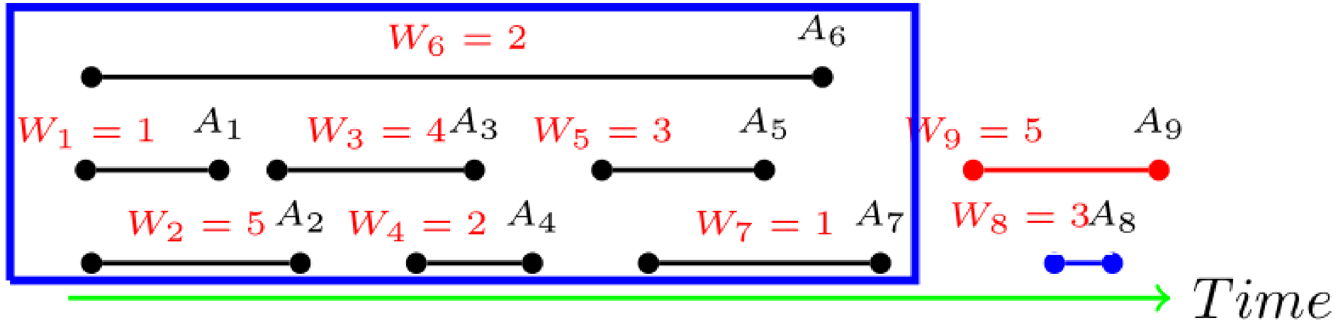
选择: S 1 = {A 1 ,A 3 ,A 5 ,A 8 } | S 2 = {A 6 ,A 9 }

收益: B(S 1) = 1 + 4 + 3 + 3 = 11 | B(S 2) = 2 + 5 = 7

子问题集合

假设A1, A2, A3, …, An 个活动

在{A1, A2, A3, …}，考虑An是否被选择：
1. An被选择，那么活动安排的策略S，是从An的开始时间前，选择

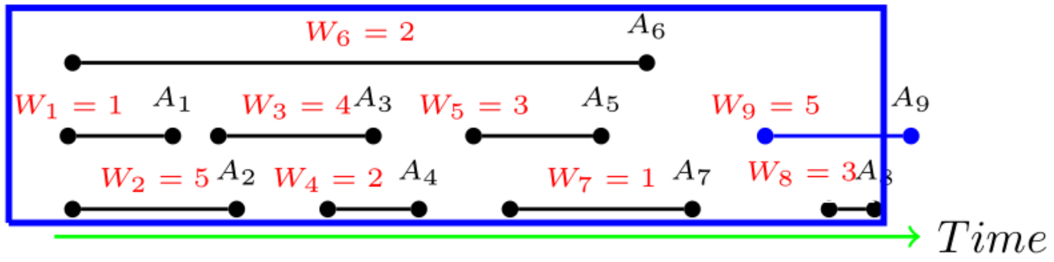


子问题集合

假设A1, A2, A3, …, An 个活得

在{A1, A2, A3, …}，考虑An是否被选择：

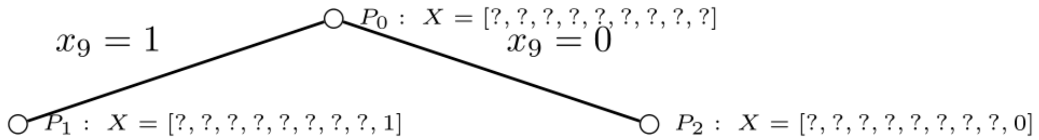
1. An被选择，那么活动安排的策略S，是从An的开始时间前，选择
2. An没有被选择，那么活动安排策略S，是从A1 ,A2 ,...,A n−1，选择



子问题分解

$$OPT(i) = \min\{OPT(pre(i))+W_i, OPT(i-1)\}$$

pre(i)是从An的开始时间前的活动集合



所有的获得Ai 按照结束时间Fi，进行升序排列

OPT(1) → OPT(n)

简化

问题描述：假设某社团某一天要组织n个活动$A=\{1,2,\cdots,n\}$，其中每个活动都要求使用同一礼堂，而且在同一时间内只有一个活动能使用这个礼堂。每个活动i都有一个要求使用礼堂的起始时间s_i和结束时间f_i，且有$s_i<f_i$。如果选择了活动i，则它在半开时间区间$[s_i, f_i)$内占用资源。**若区间$[s_i, f_i)$与区间$[s_j, f_j)$不相交，则称活动i与活动j是相容的**。现在给定n个活动的开始时间和结束时间，请设计一个活动安排方案，使得安排的相容活动数目最多。

策略一：选择具有最早开始时间，而且不与已安排的活动冲突的活动。

A = {(0, 10), (2, 5), (7, 9)}

策略二：选择具有最短使用时间，而且不与已安排的活动冲突的活动。

E = {(0, 4), (3, 5), (4, 9)}

策略三：选择具有最早结束时间，而且不与已安排的活动冲突的活动。

证明

策略三：选择具有最早结束时间，而且不与已安排的活动冲突的活动。

证明（数学归纳法）：

预处理 将集合S中的活动按照**结束时间递增顺序排列**，即记为$S = \{1,2,\cdots,n\}$；假设$A = \{j_1, j_2, \cdots, j_m\}$ 是S的一个最优解，其中活动也按照**结束时间递增顺序排列。**

1）基础步 $k = 1$时选择活动为$\{1\}$，需要证明存在一个最优解包含了活动$\{1\}$

- $j_1 = 1$，即最优解包含活动1
- $j_1 \neq 1$，则用活动1替换最优解A中的j_1，得到活动集合A'，既有：$A' = \{A - \{j_1\}\} \cup \{1\}$

A'是相容活动集合，也是一个最优的活动安排

策略三：选择具有最早结束时间，而且不与已安排的活动冲突的活动。

证明（数学归纳法）：

2）归纳步　假设对于正整数k，命题正确。令$\{i_1=1, i_2, \cdots, i_k\}$ 是GreedySelector算法前k步顺序选择的活动，那么存在一个最优解：$A=\{i_1=1, i_2, \cdots, i_k\} \cup B$

假设S' 是S中与$\{i_1=1, i_2, \cdots, i_k\}$相容的活动，即$S'=\{j \mid s[j] \geq f[i_k],\ j \in S\}$

那么B是S'的一个最优解。

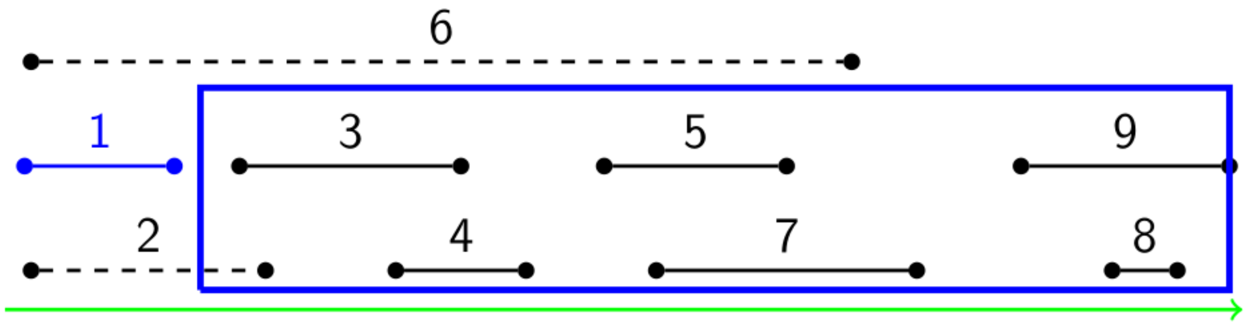
对于子问题S' **应用基础步的结论**：即S' 的结束时间最早的活动i_{k+1}包含在某一个最优解B′中，则可以构造原问题的一个最优解：

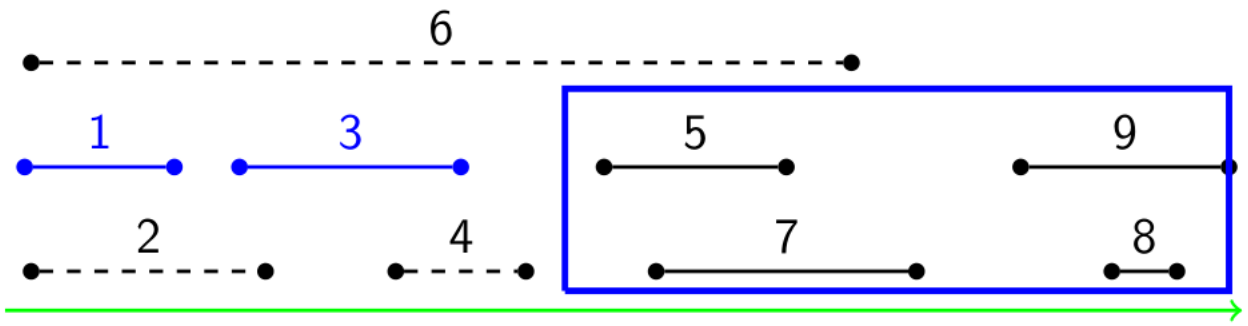
$$A'=\{i_1=1, i_2, \cdots, i_k\} \cup B'=\{\boldsymbol{i_1=1, i_2, \cdots, i_k, i_{k+1}}\} \cup \{B'-\{i_{k+1}\}\}$$

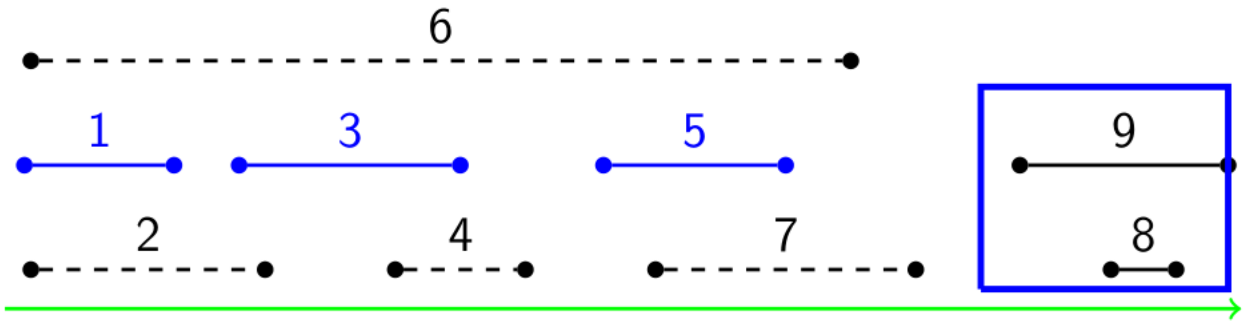
Interval Scheduling Greedy(n)

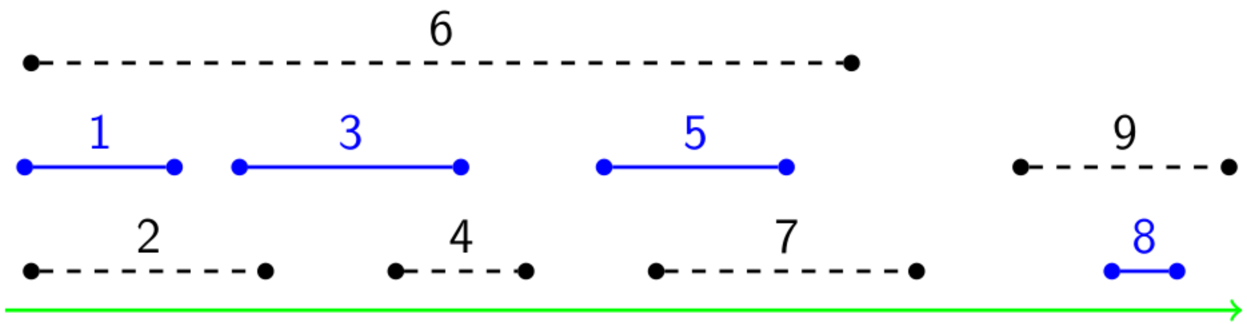
Require: All A_i have been sorted in the increasing order of F_i .

```
prev = −∞;
for i = 1 to n do:
   if S i ≥ prev then:
        Select activity A i ;
        prev = F i ;
```

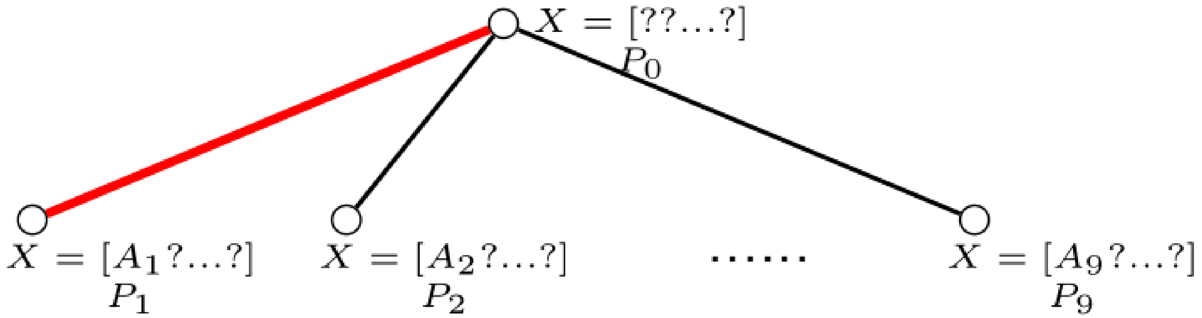

6
1
3
5
9
2
4
7
8


6
1
3
5
9
2
4
7
8


6
1
3
5
9
2
4
7
8


6
1
3
5
9
2
4
7
8

$x_1 = A_1/A_2/.../A_9$



决策

在x1节点，并不分析P1，P2，P3，----，P9哪一个最优

而是直接选择P1，基于贪心准则

李强