



# 离散数学

## Discrete Mathematics

### 第二十六讲：带权图与最短路

吴楠

南京大学计算机学院



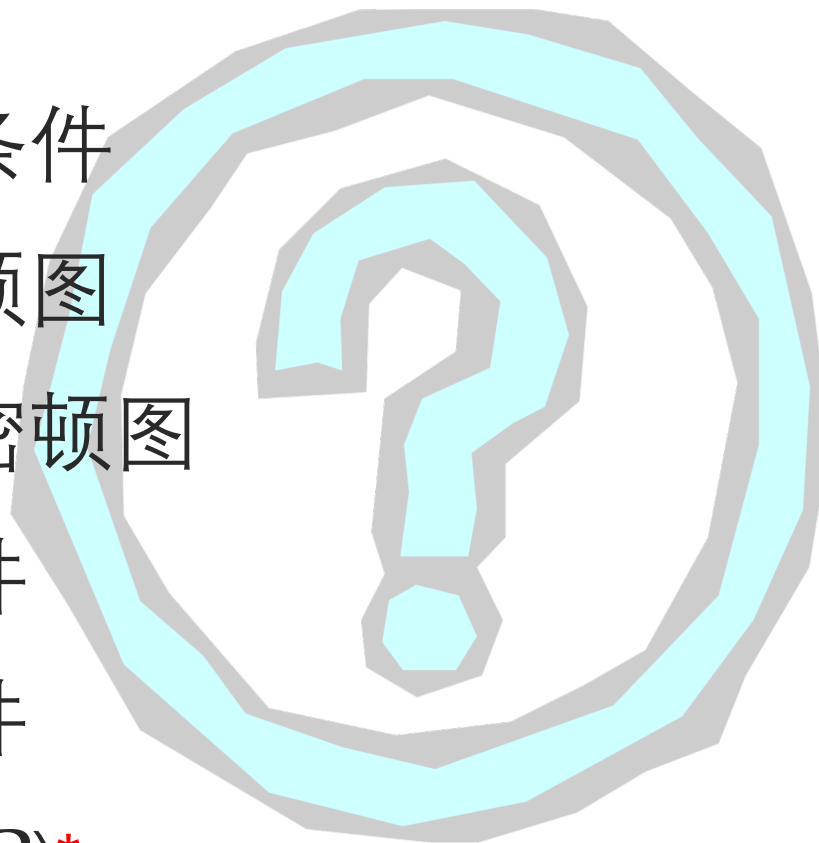
2025 年 5 月 27 日



# 前情提要



- 欧拉回路与欧拉图
- 欧拉图的充分必要条件
- 哈密顿回路与哈密顿图
- 哈密顿通路与半哈密顿图
- 哈密顿图的必要条件
- 哈密顿图的充分条件
- 旅行推销员问题(TSP)\*

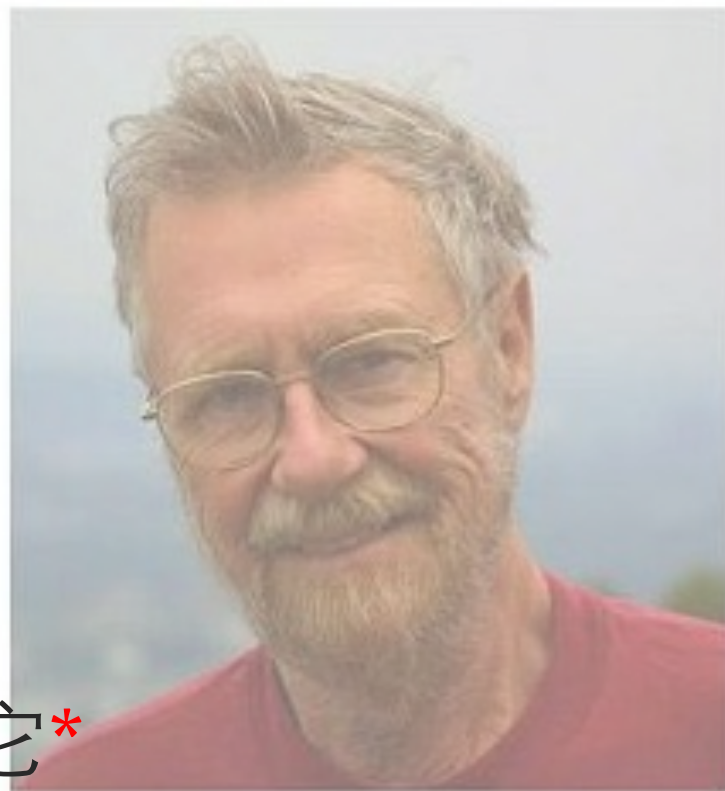




# 本讲主要内容



- 带权图
- 带权图的单源最短路
- 求最短路的算法思想
- Dijkstra算法
- Dijkstra算法的分析\*
- Dijkstra算法的应用及其它\*

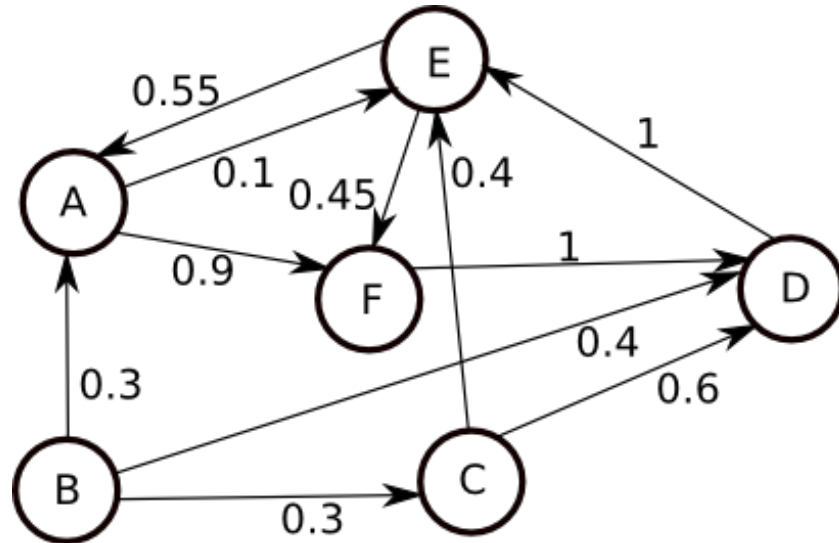
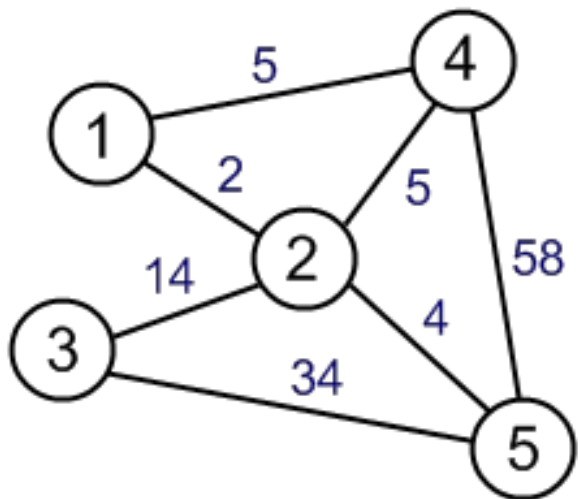




# 带权图 (weighted graph)



- **定义 (带权图)** : 三元组  $\langle V, E, W \rangle$ ,  $V$  代表 (有向或无向) 图的点集,  $E$  代表边集, 函数  $W: E \rightarrow \mathbb{Z}^+$  (或  $\mathbb{R}^+$ ) 为边的权重:  $W(e)$  表示边  $e$  的 **权**

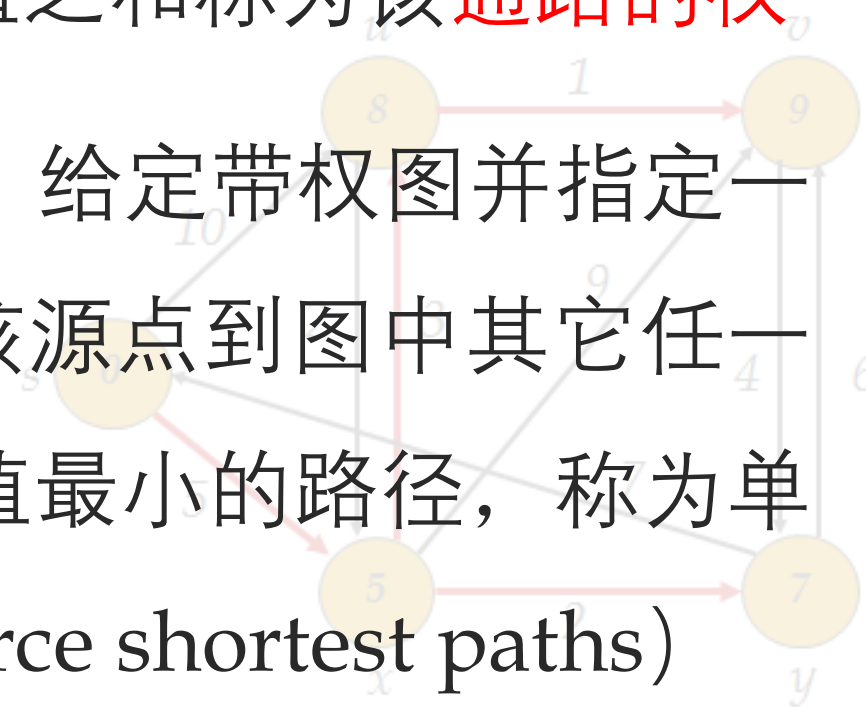




# 通路的权与单源最短路(SSSP)



- **定义（通路的权）**：一条通路上所有边（含重复出现的边）的权值之和称为该**通路的权**
- **定义（单源最短路）**：给定带权图并指定一个源点（source），该源点到图中其它任一顶点的**最短路**，即权值最小的路径，称为单源最短路（single-source shortest paths）



## A Note on Two Problems in Connexion with

By  
E. W. DIJKSTRA

We consider  $n$  points (nodes), some or all pairs of which are connected by a branch; the length of each branch is given. We restrict our attention to those graphs where at least one path exists between any two nodes. We shall consider two problems.

**Problem 1.** Construct the tree of minimum total length between every pair of nodes. (A tree is a graph with one and only one path between every pair of nodes.)

In the course of the construction that we present here, the graph is subdivided into three sets:

I. the branches definitely assigned to the tree under construction (they form a subtree);

II. the branches from which the next branch to be added to the tree will be selected;

III. the remaining branches (rejected or not yet considered). The nodes are subdivided into two sets:

A. the nodes connected by the branches of set I;

B. the remaining nodes (one and only one path exists between any two of these nodes).

We start the construction by choosing an arbitrary node  $P$  of set A, and by placing all branches that connect  $P$  with set I in set I. From then onwards the construction proceeds repeatedly.

**Step 1.** The shortest branch of set II is selected and added to set I. As a result one node is transferred from set B to set A.

**Step 2.** Consider the branches leading from the node just transferred to set A, to the nodes that are still in set B. If the length of such a branch is shorter than the length of the shortest branch in set II, it replaces the corresponding branch in set II.

We then return to step 1 and repeat the process. The branches in set I form the tree required.

The solution given here is to be preferred to the solutions of KRUSKAL [1] and those given by H. LOBOS [2]. Their solutions all the — possibly  $\frac{1}{2}n(n-1)$  — branches are considered according to length. Even if the length of the branches is a function of the node coordinates, their methods demand that data be stored simultaneously. Our method only requires the six

## Top Algorithms/Data Structures/Concepts every computer science student should know



Coding Freak Follow  
Jun 27, 2018 · 1 min read



Top algorithms –

Insertion sort, Selection sort, Bubble sort

## DIJKSTRA( $G, w, s$ )

1 INITIALIZE-SINGLE-SOURCE( $G, s$ )

2  $S = \emptyset$

3  $Q = G.V$

4 while  $Q \neq \emptyset$

5  $u = \text{EXTRACT-MIN}(Q)$

6  $S = S \cup \{u\}$

7 for each vertex  $v \in G.Adj[u]$

8 RELAX( $u, v, w$ )

Disjoint-Set Data Structure (Union-Find Algorithm)

Kruskal's Algorithm for finding Minimum Spanning Tree

Single-Source Shortest Paths — Dijkstra's Algorithm

All-Pairs Shortest Paths — Floyd Warshall Algorithm

E. W. DIJKSTRA:

branches, viz. the branches in sets I and II and the branch step 2.

the path of minimum total length between two given nodes

if  $R$  is a node on the minimal path from  $P$  to  $Q$ , knowledge of the knowledge of the minimal path from  $P$  to  $R$ . In the minimal paths from  $P$  to the other nodes are constructed length until  $Q$  is reached.

the solution the nodes are subdivided into three sets:

which the path of minimum length from  $P$  is known; nodes in order of increasing minimum path length from node  $P$ ;

which the next node to be added to set A will be selected;

those nodes that are connected to at least one node of set A themselves;

the solution is subdivided into three sets:

arriving in the minimal paths from node  $P$  to the nodes

in which the next branch to be placed in set I will be

this set will lead to each node in set B; (rejected or not yet considered).

and all branches are in set III. We now proceed onwards repeatedly perform the following

selecting the node just transferred to set A

belongs to set B, we investigate whether there is a shorter path from  $P$  to  $R$  than the known path

in set II. If this is not so, branch  $r$  is rejected. If it results in a shorter connexion between  $P$  and  $R$ , the corresponding branch in set II is replaced by branch  $r$ .

belongs to set C, it is added to set B and the process is repeated.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.

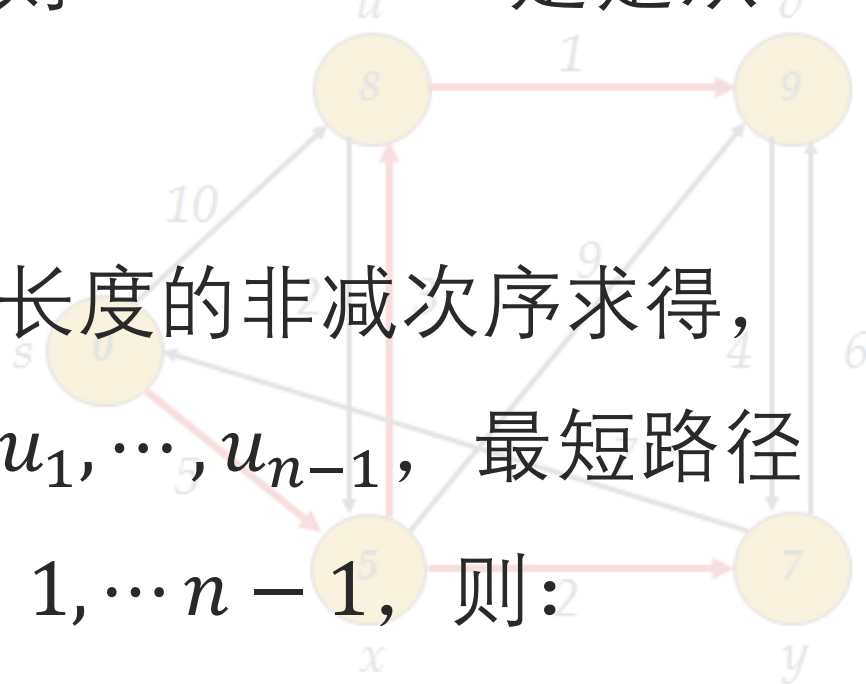
connected to node  $P$  in only one way. The node with minimum distance from  $P$  is transferred to set I and one from set II. In this sense the node with minimum distance from  $P$  is transferred to set I, and the corresponding branch is transferred to set I. We then return to step 1 and repeat the process until the solution has been found.



# 求单源最短路的算法思想



- **思路（最短子路径）**：若源点 $s$ 到顶点 $v$ 的最短路径为 $s - \dots - u - v$ ，则 $s - \dots - u$ 一定是从 $s$ 到 $u$ 的**最短路径**
- 设 $n - 1$ 条最短路径按其长度的非减次序求得，设其相应的端点分别为 $u_1, \dots, u_{n-1}$ ，最短路径的长度记为 $d(s, u_i)$ ， $i = 1, \dots, n - 1$ ，则：





# 求单源最短路的算法思想（续）



- **思路（Greedy Choice）**：假设前 $i$ 条最短路径已知，则第 $i + 1$ 条最短路径长度可以这样求得：

$$d(s, u_{i+1}) = \min\{d(s, u_j) + W(u_j, u_{i+1}) \mid j = 1, \dots, i\}$$

A Note on Two Problems in Connexion with Graphs

- 根据以上思路可以保证求得SSSP
- 1959年，荷兰科学家Dijkstra提出基于上述思路的SSSP求解算法，史称“Dijkstra算法”



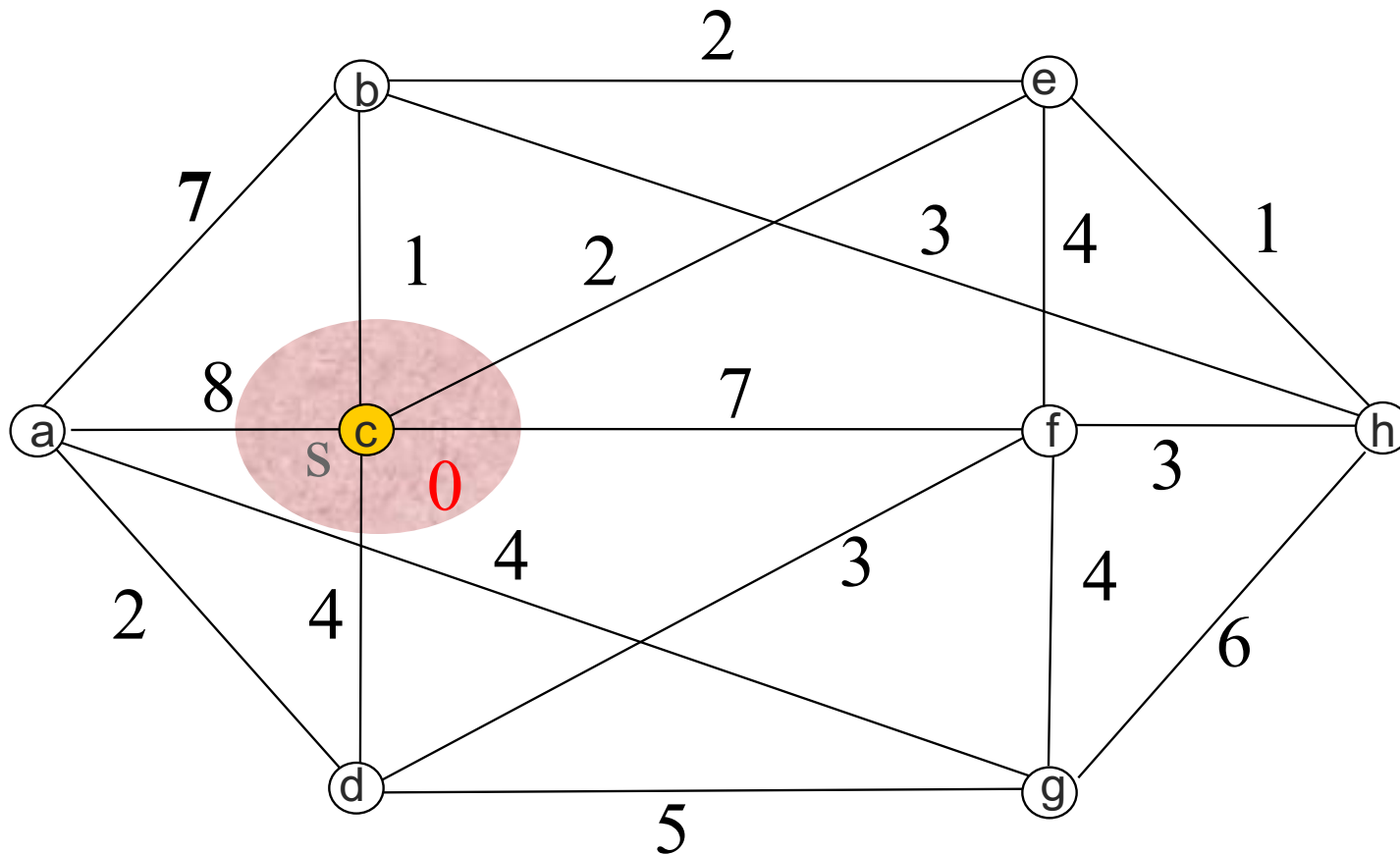
# 求单源最短路Dijkstra算法



- **输入：**  $n$ 阶连通带权图 $G$ ，指定顶点 $s \in V_G$
- **输出：** 每个顶点 $v$ 的标注 $\langle L(v), u \rangle$ ，其中：
  - $L(v)$ 即为从源点 $s$ 到各目标顶点 $v$ 的所有路径中（目前求得的）最短路径的长度
  - $u$ 为该路径上到达 $v$ 之前的顶点标识
- **算法过程演示**（ $S_i$ 为当前已求得单源最短路的点集， $u_i$ 为当前生成的第 $i$ 条最短路对应的目标顶点）

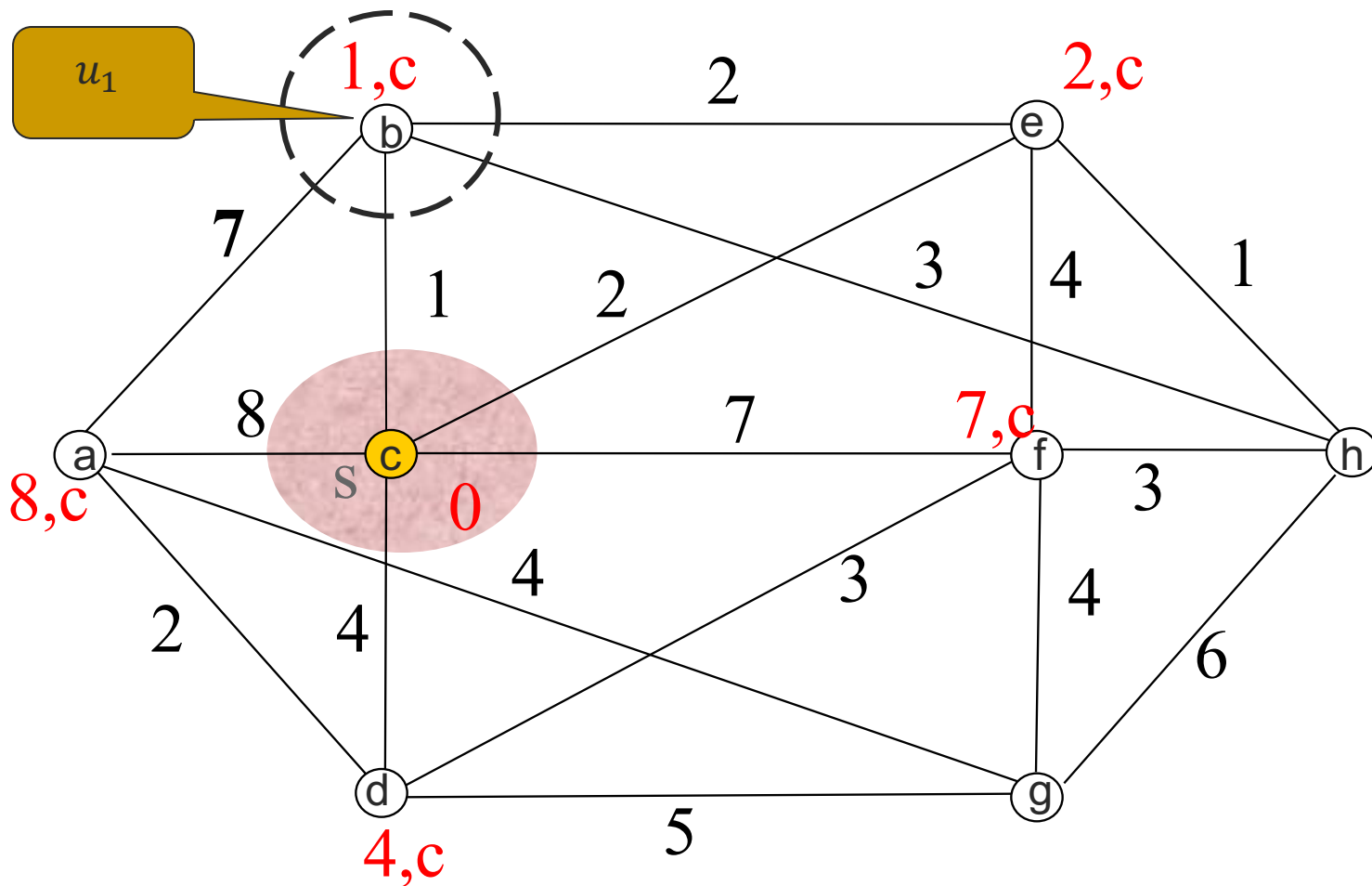


# Dijkstra算法过程演示



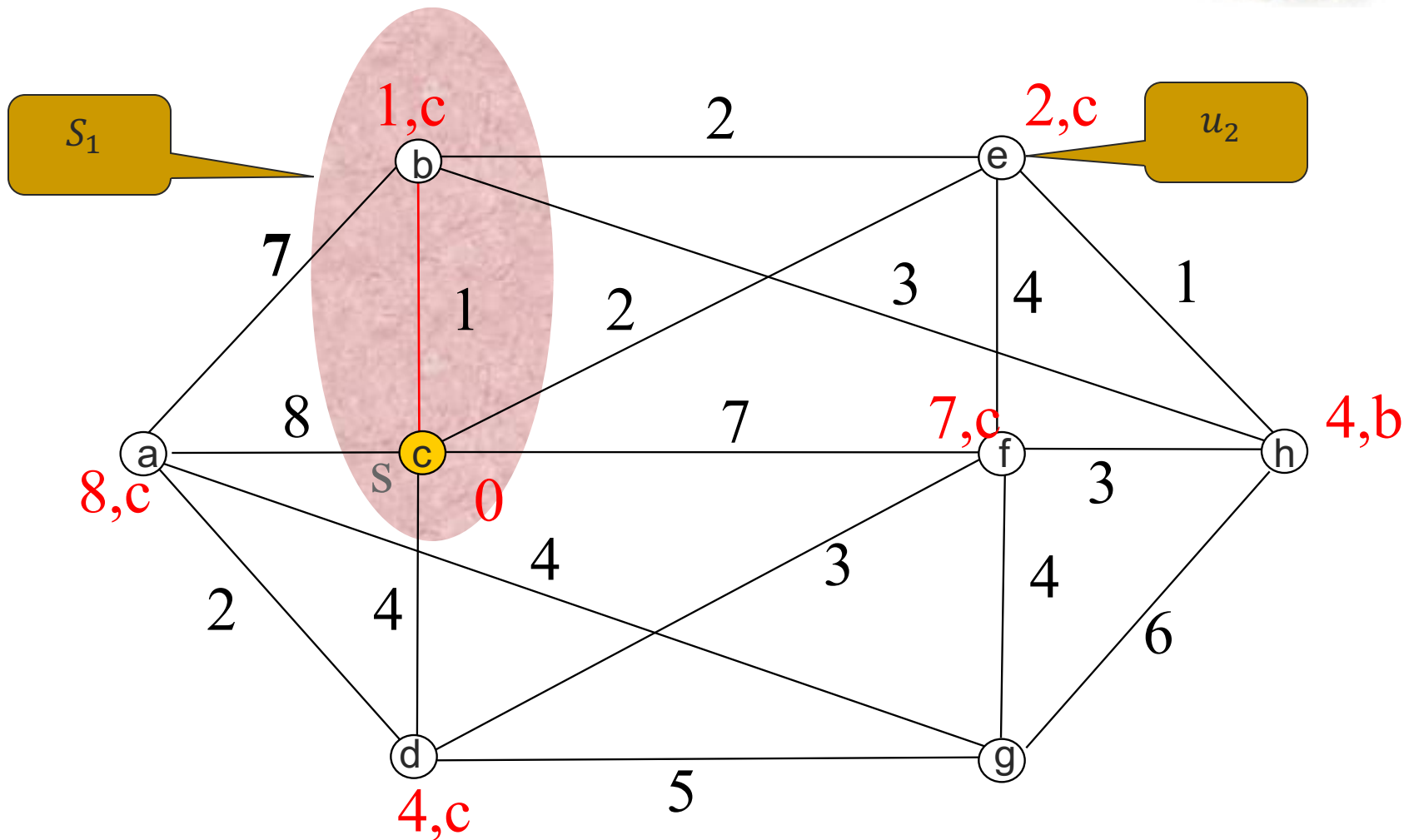


# Dijkstra算法过程演示



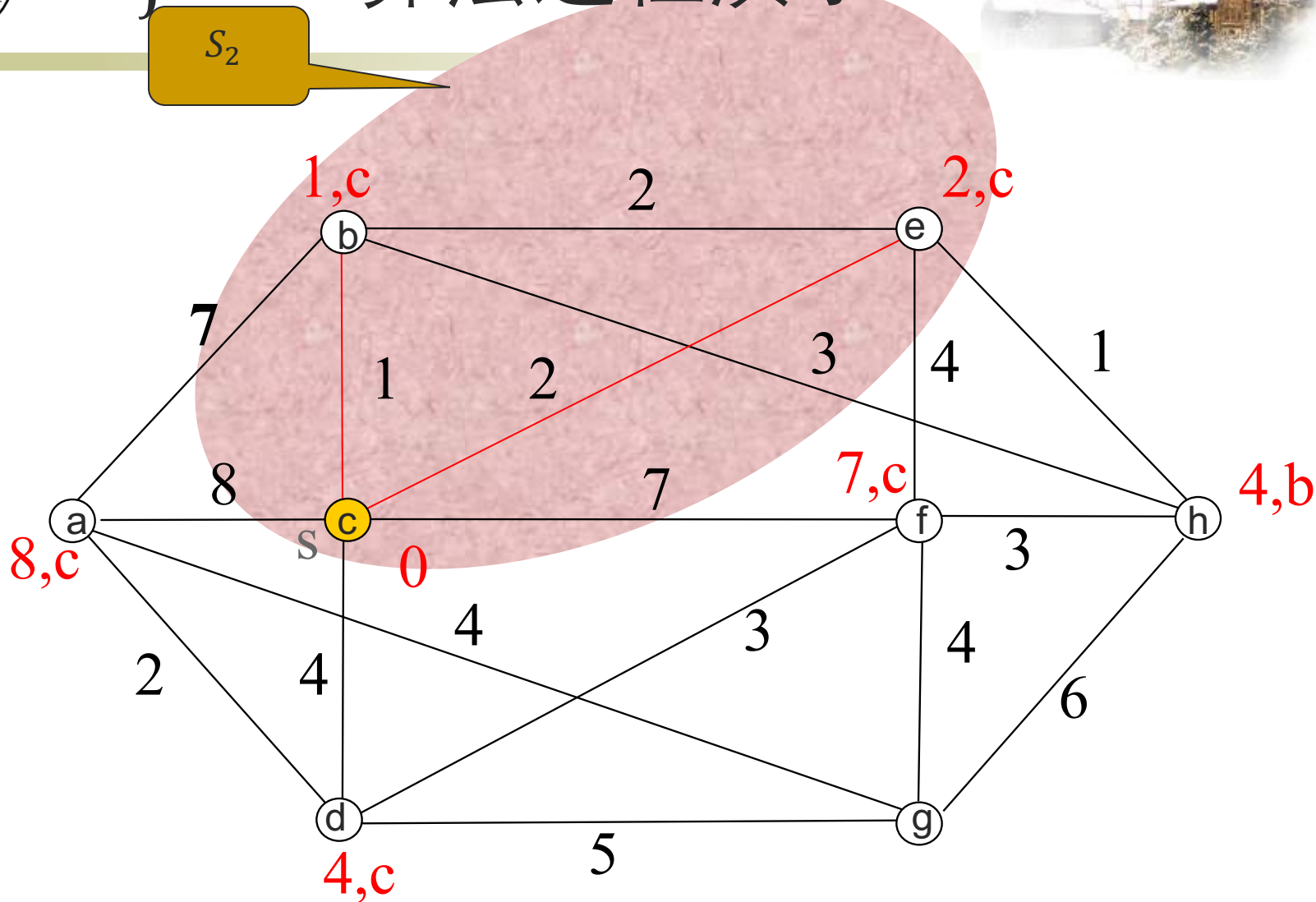


# Dijkstra算法过程演示



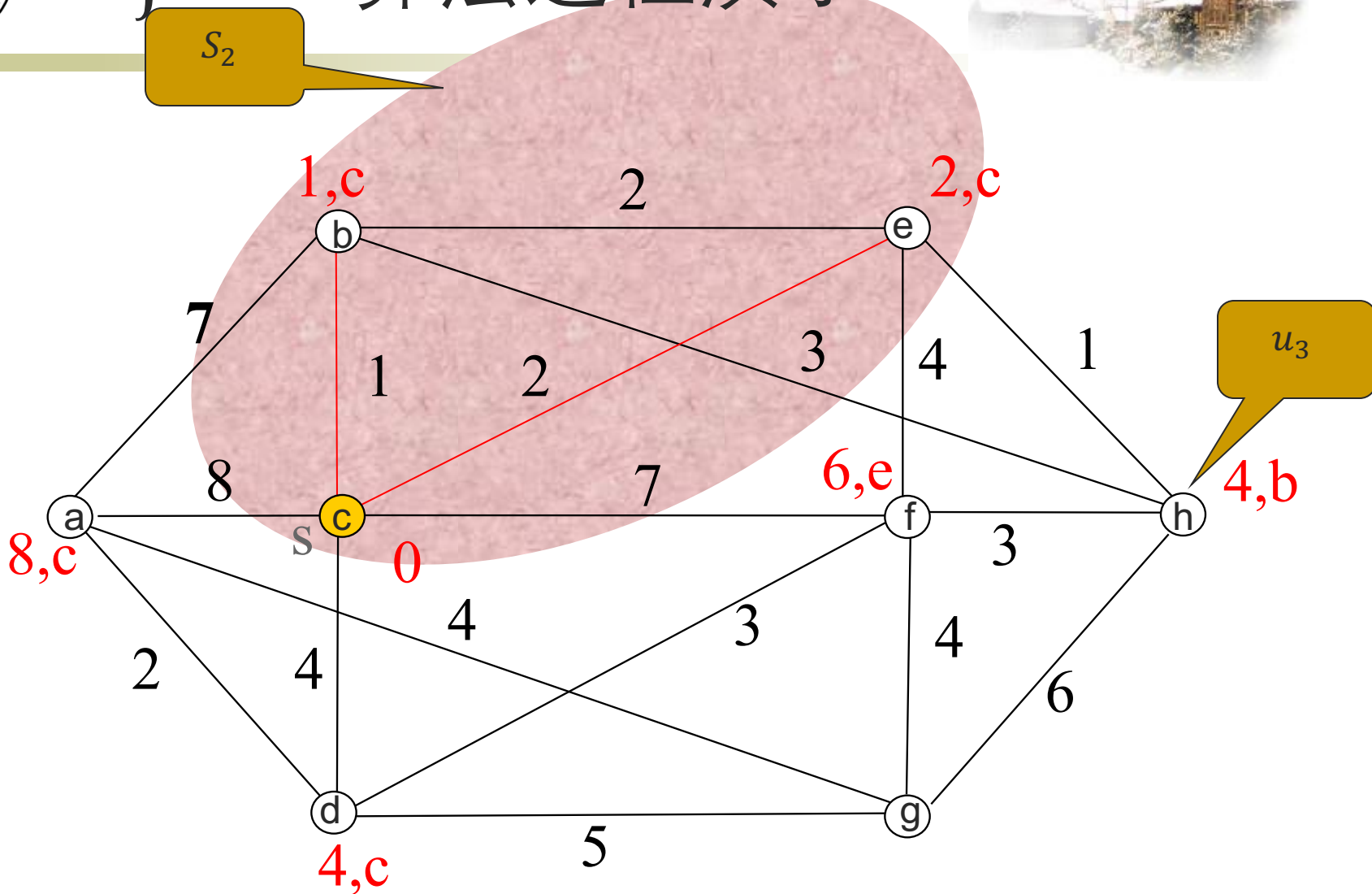


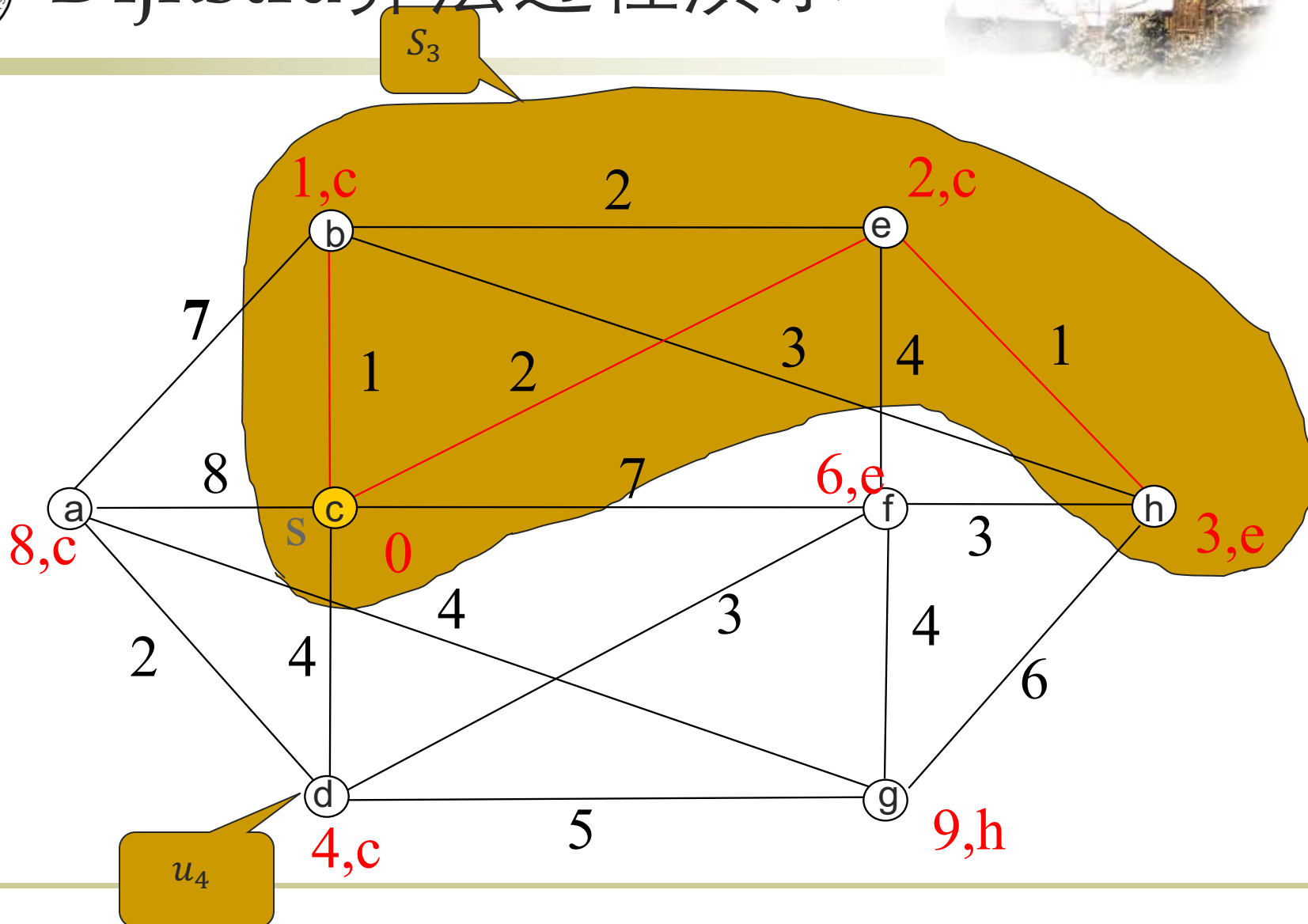
# Dijkstra算法过程演示





# Dijkstra算法过程演示

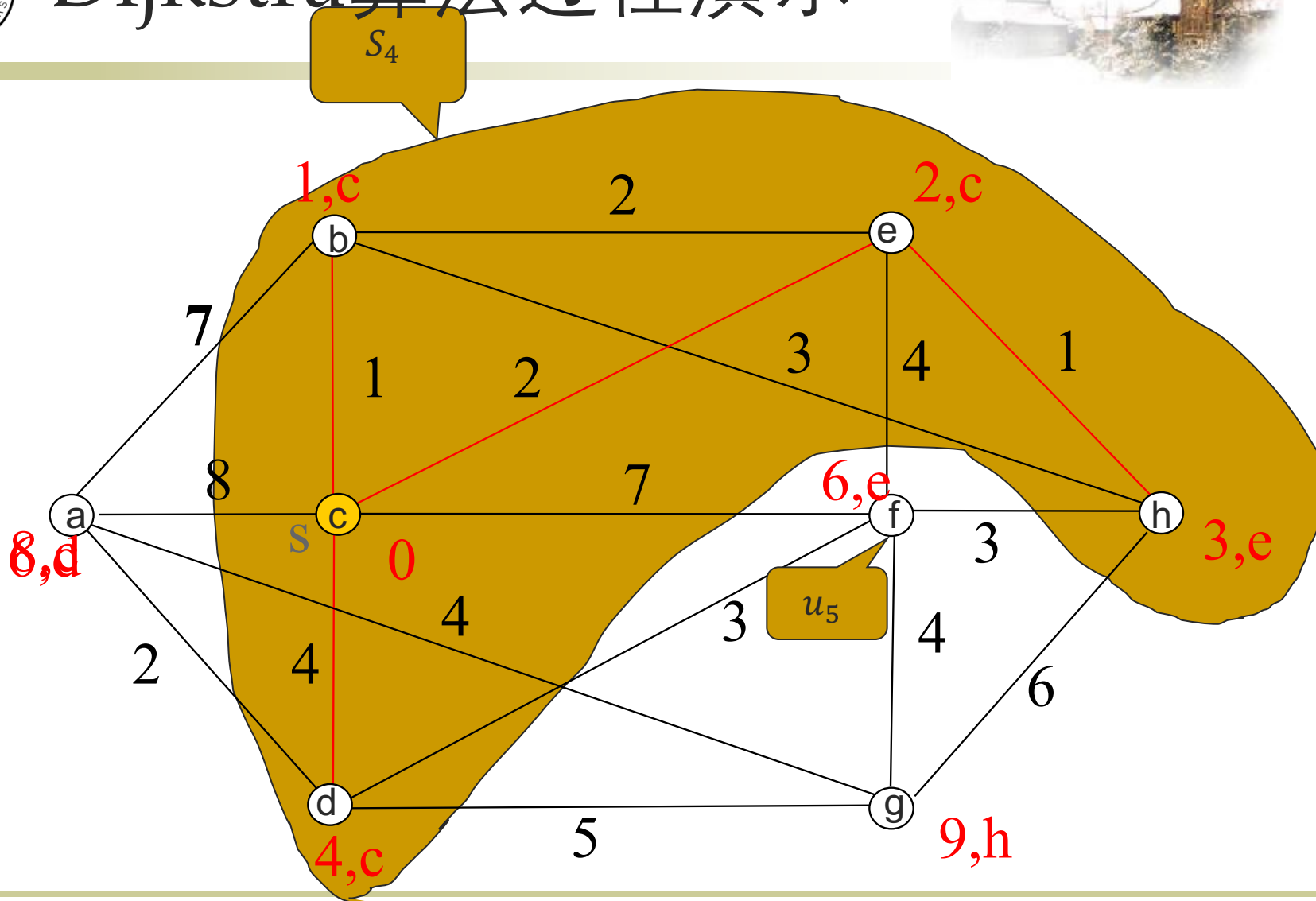




## Dijkstra算法过程演示

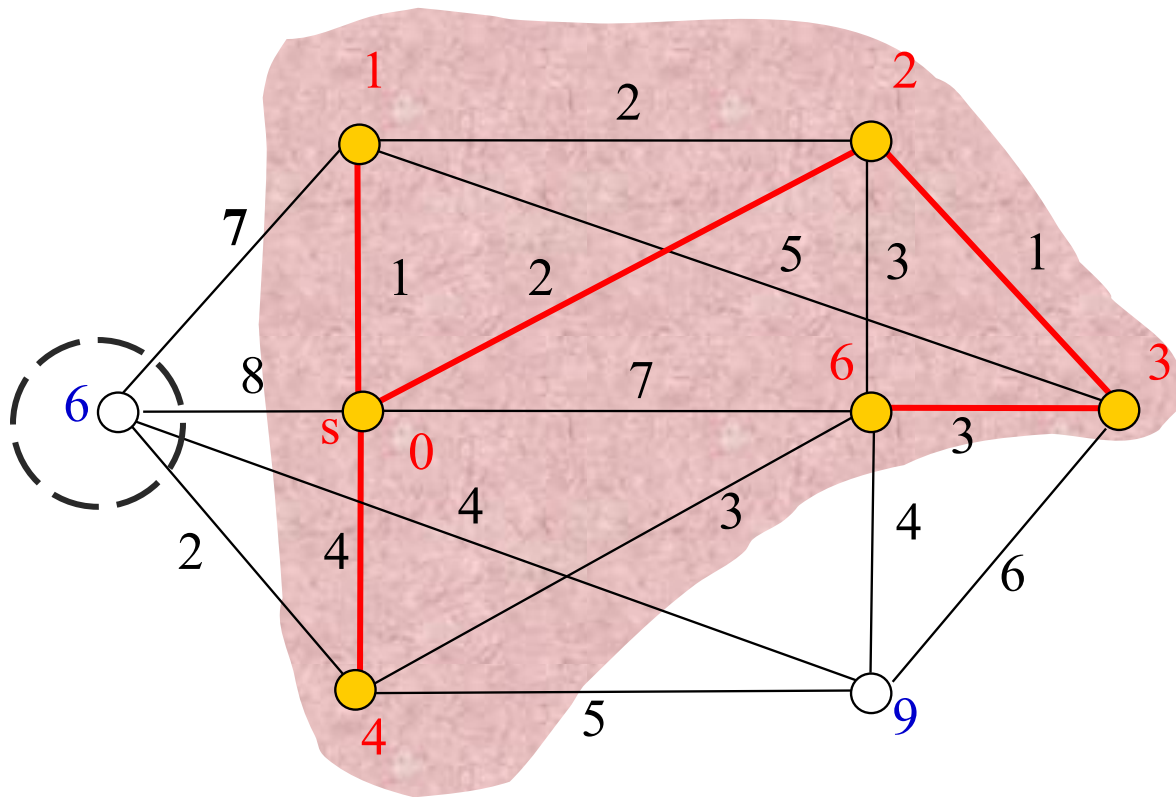


# Dijkstra算法过程演示



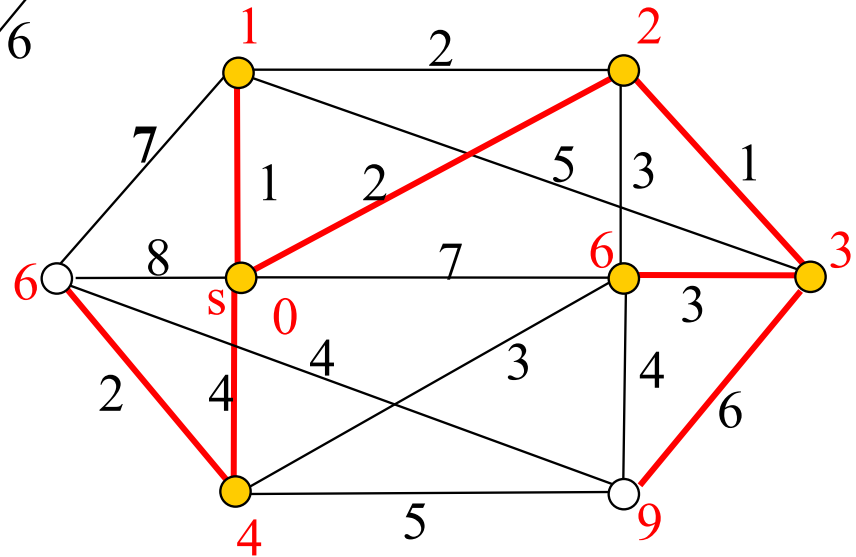
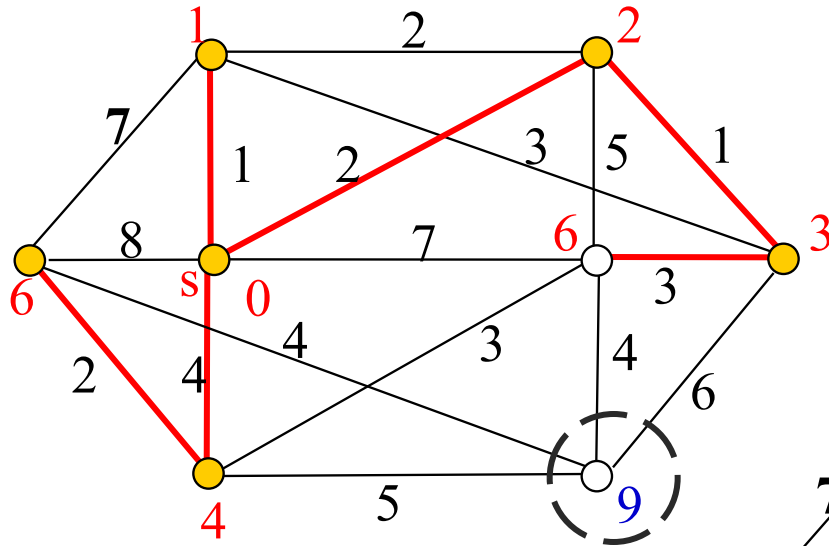


# Dijkstra算法过程演示



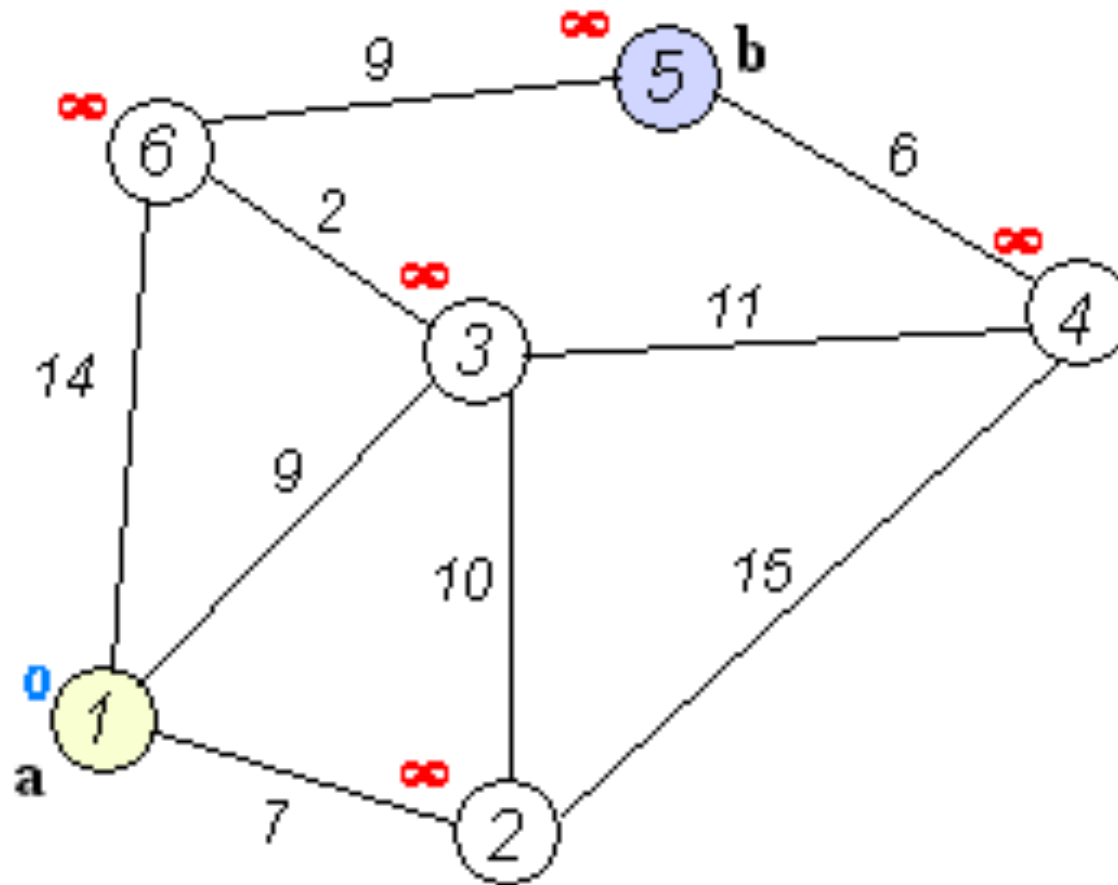


# Dijkstra算法过程演示





# Dijkstra算法过程演示（动态）





# Dijkstra算法描述



- **Step 1.** 初始化:  $i = 0$ ,  $S_0 = \{s\}$ ,  $L(s) = 0$ , 对其它一切  $v \in V_G$ , 将  $L(v)$  置为  $\infty$ 。若  $n = 1$ , **结束**
- **Step 2.**  $\forall v \in S_i' = V_G - S_i$ , 比较  $L(v)$  和  $L(u_i) + W(u_i, v)$  的值 ( $u_i \in S_i$ ), 如果  $L(u_i) + W(u_i, v) < L(v)$ , 则将  $v$  的标注更新为  $\langle L(u_i) + W(u_i, v), u_i \rangle$ , 即:  
$$L(v) = \min\{L(v), \min_{u \in S_i} \{L(u) + W(u, v)\}\}$$
- **Step 3.** 对所有  $S_i'$  中的顶点, 找出具有最小  $L(v)$  的顶点  $v$  作为  $u_{i+1}$
- **Step 4.**  $S_{i+1} = S_i \cup \{u_{i+1}\}$
- **Step 5.**  $i = i + 1$ ; 若  $i = n - 1$ , 算法终止; 否则转到 **Step 2.**



# Dijkstra算法的伪代码实现\*

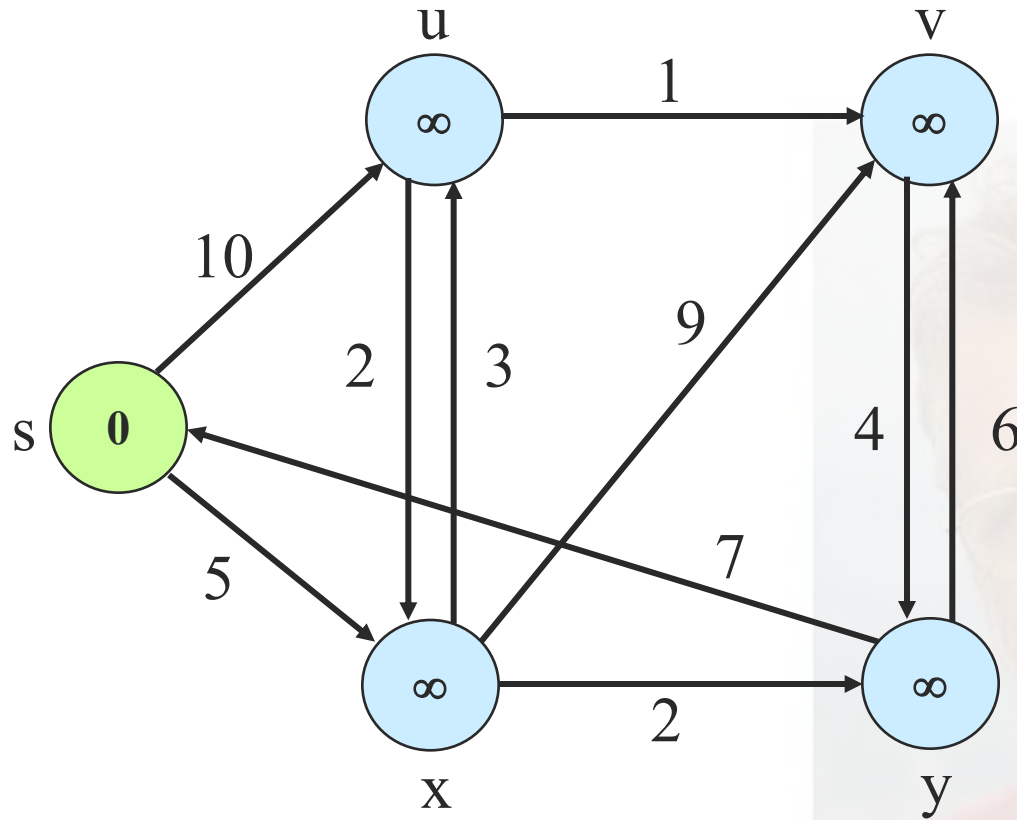


```
1  function Dijkstra(G, w, s)
2      for each vertex v in V[G]           // 初始化
3          d[v] := infinity                // 将各点的已知最短距离先设成无穷大
4          previous[v] := undefined        // 各点的已知最短路径上的前趋都未知
5      d[s] := 0                           // 因为出发点到出发点间不需移动任何距离, 所以可以直接将s到s的最小距离设为0
6      S := empty set
7      Q := set of all vertices             $O(V)$ 
8      while Q is not an empty set         // Dijkstra 算法主体
9          u := Extract_Min(Q)              $O(V)$ 
10         S.append(u)
11         for each edge outgoing from u as (u,v)  $O(E)$ 
12             if d[v] > d[u] + w(u,v)     // 拓展边 (u,v) 。w(u,v)为从u到v的路径长度。
13                 d[v] := d[u] + w(u,v)    $O(1)$  // 更新路径长度到更小的那个和值。
14                 previous[v] := u         // 纪录前趋顶点
```



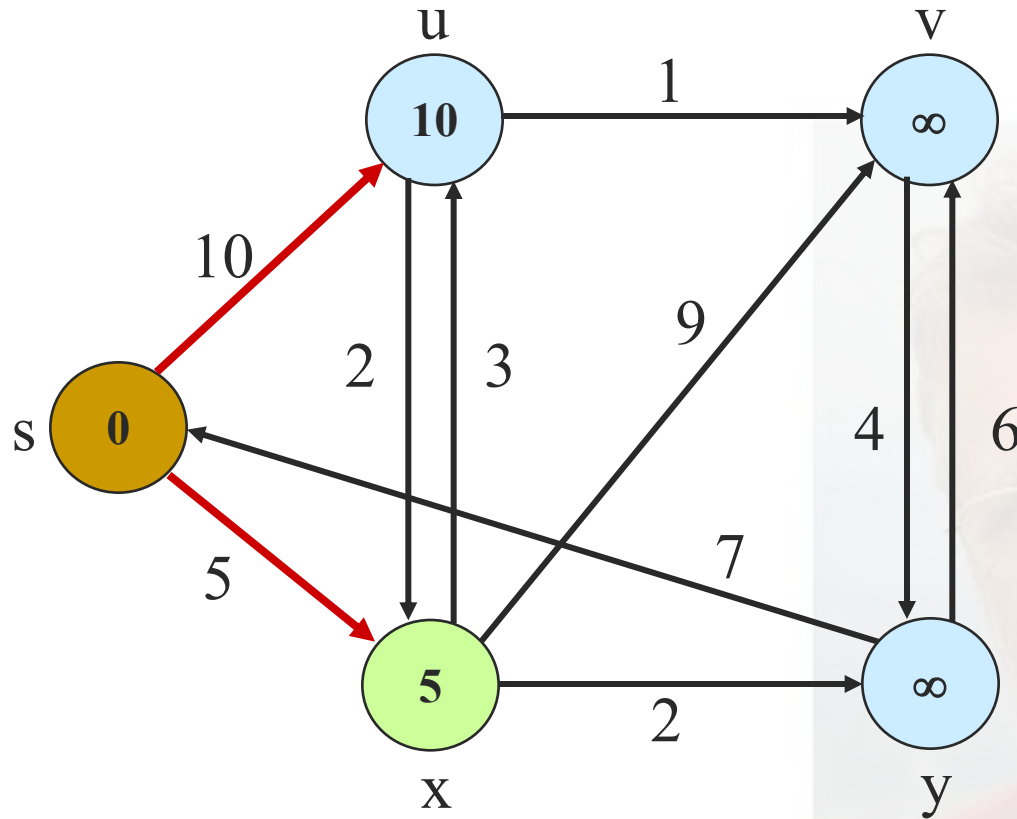


# 练习：试求以下有向图的 SSSP



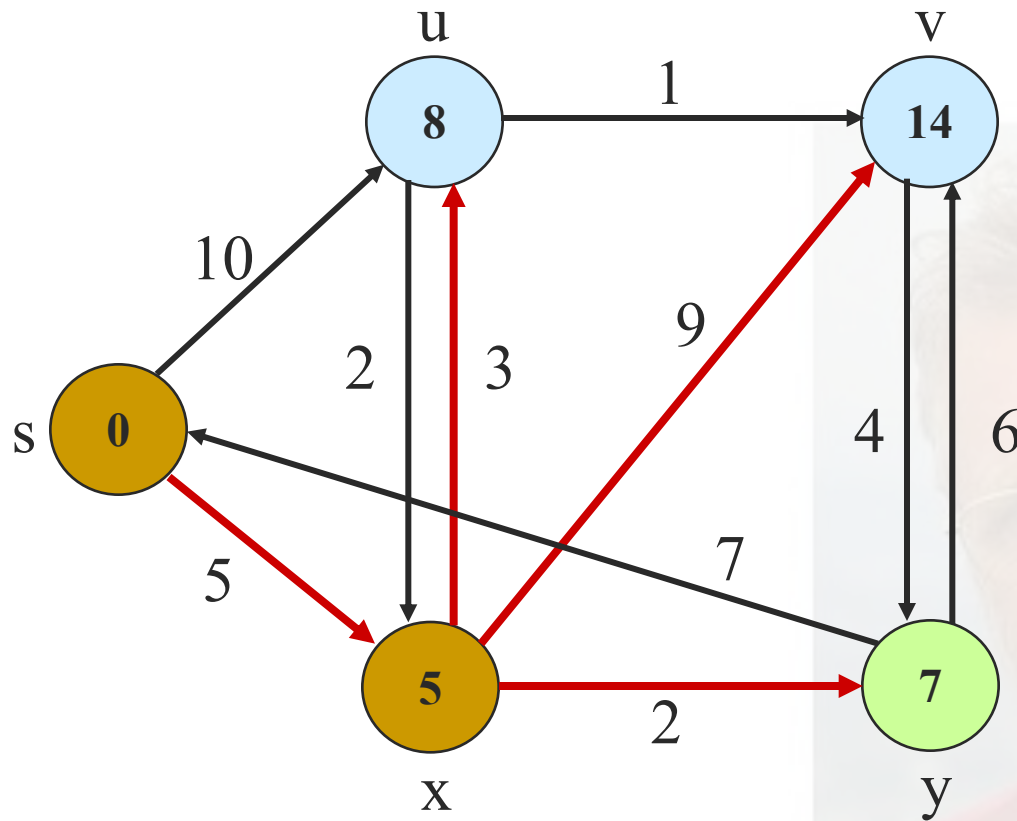


# 练习：试求以下有向图的 SSSP



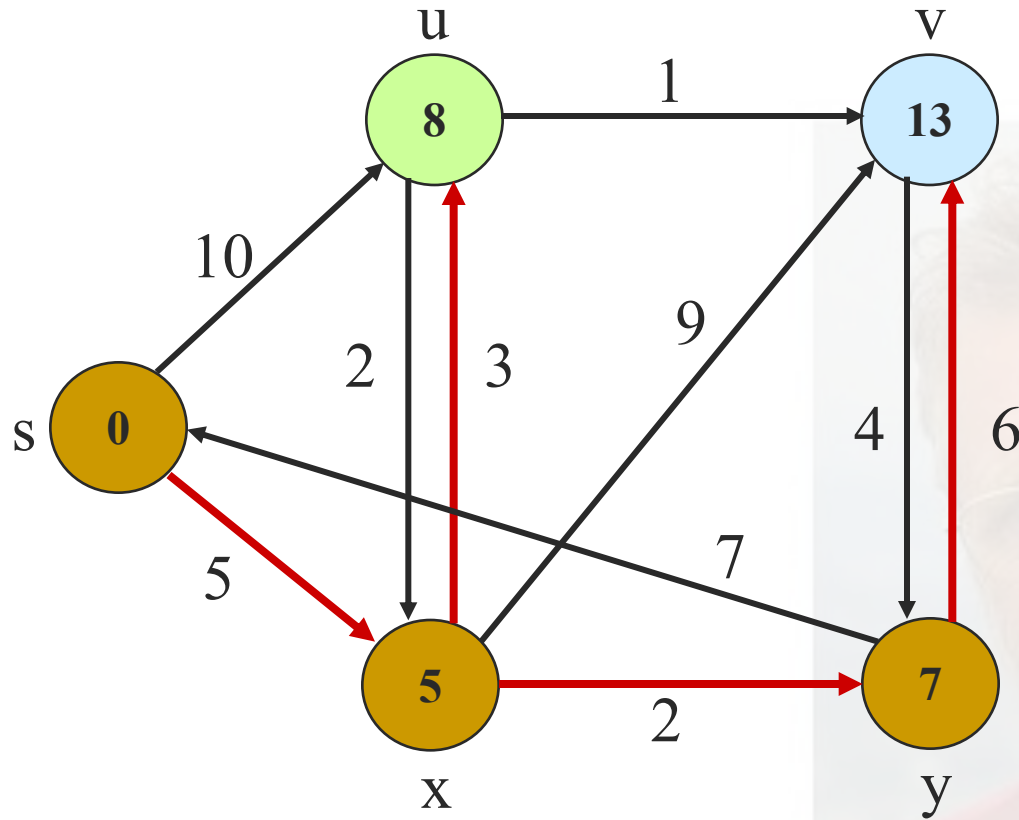


# 练习：试求以下有向图的 SSSP



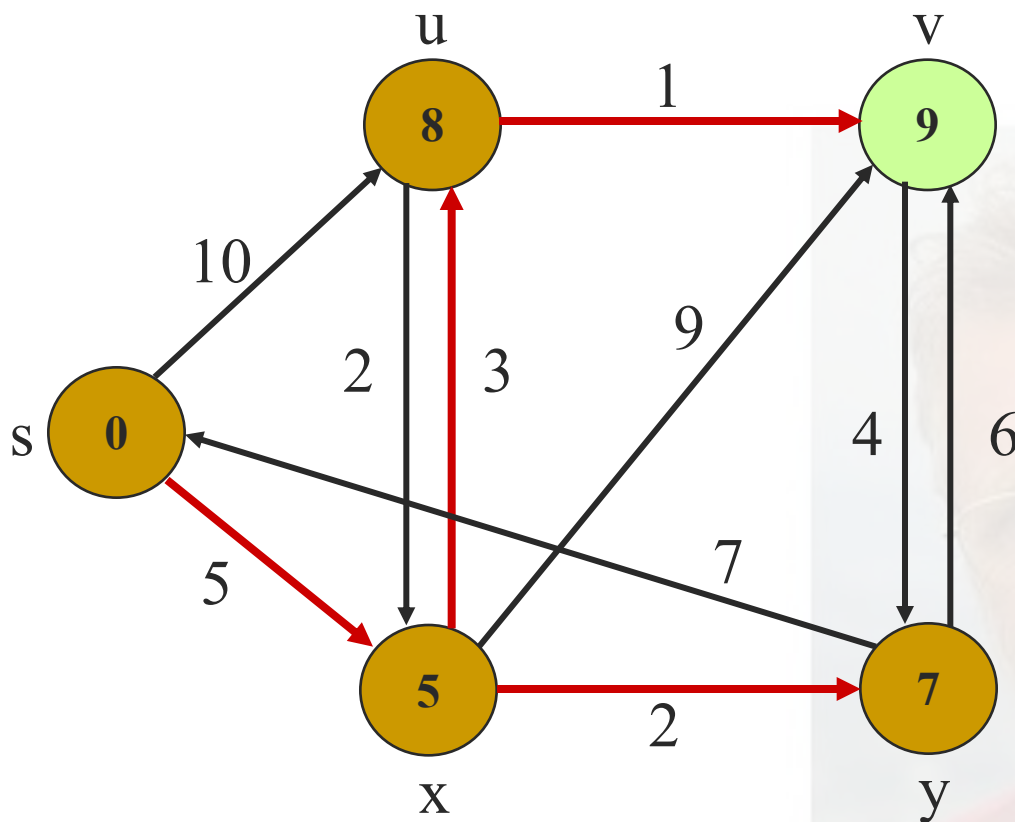


# 练习：试求以下有向图的 SSSP



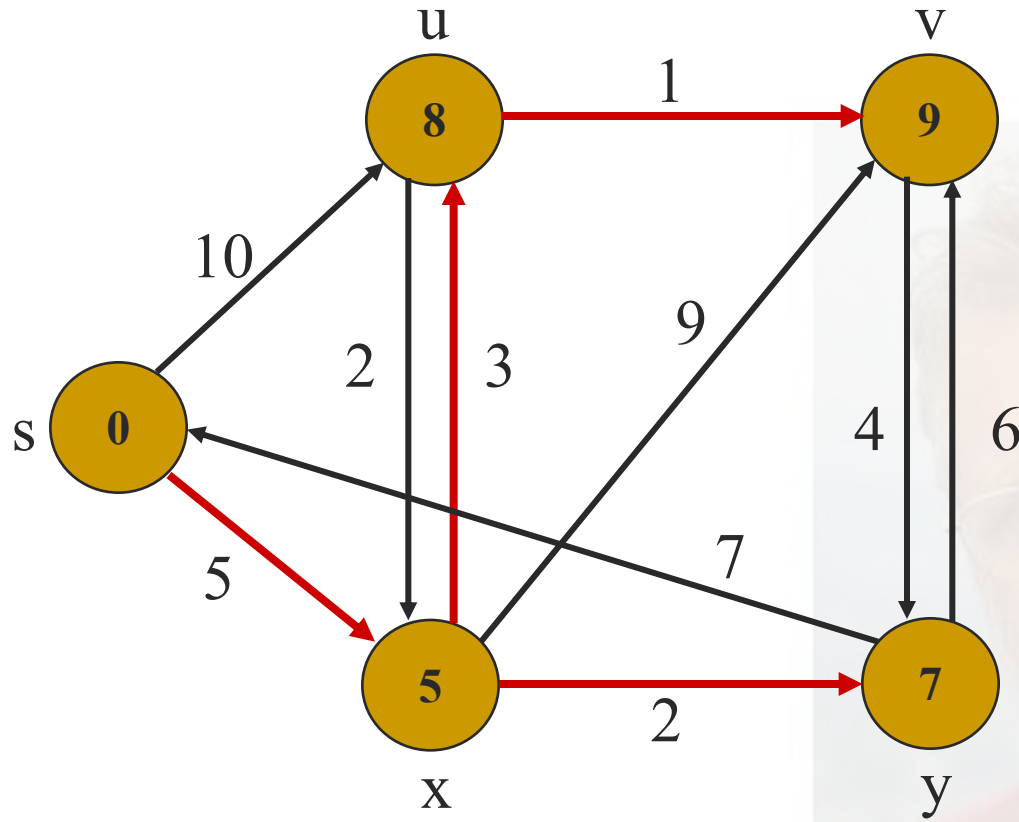


# 练习：试求以下有向图的 SSSP





# 练习：试求以下有向图的 SSSP





# Dijkstra算法的简单分析\*



- **算法的可终止性：** 计数控制
- **算法的正确性（留作思考）：** 需证明当算法终止时，
  - $L(v) = d(s, v)$  对一切  $v$  成立
  - 由标记中的诸  $u_i$  确定的路径是一条最短路径  
(这里  $d(s, v)$  是  $s$  到  $v$  的最短路径长度，即距离)
- **时间复杂度：**  $\Theta(|V|^2 + |E|) = \Theta(|V|^2)$  对  $|E| \in O(|V|^2)$ 
  - 可通过数据结构优化为  $O((|V| + |E|)\log|V|)$



# Dijkstra算法：Application and Beyond\*



## 应用最多的算法之一

AATGCCGTACGTAGGGTAATATATGACCA

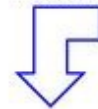
(Sequencing: Solexa, Illumina, etc...)

TGCCGT TAGGGT ATATAT  
AATGCT TACGTA ATGACC  
TTGCCG CGTAGG TAATAT  
GTACGT GTACTA  
AATGCC GGGTAA TGACCA  
GTAGGG TATGAC  
CTATAT

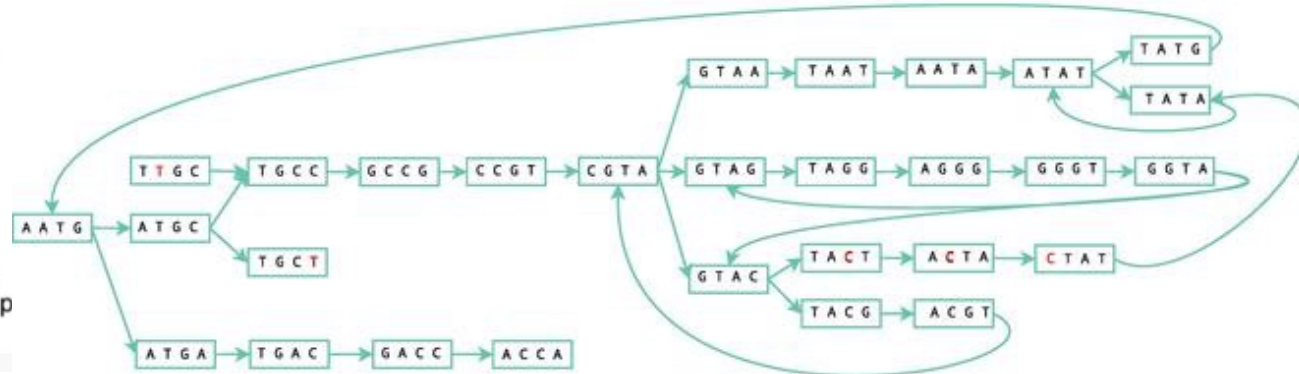
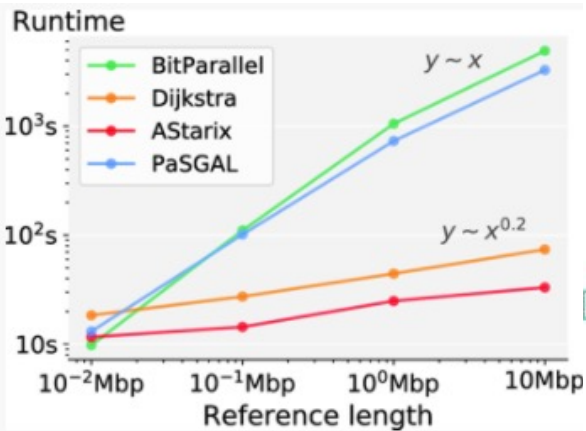
Compute k-mer  
with k=4



Create Graph  
for the set of k-mers



AATA  
AATG (x2)  
ACCA  
ACGT (x2)  
ACTA  
AGGG (x2)  
ATAT (x4)  
ATGA (x2)  
ATGC (x2)  
CCGT  
CGTA (x2)  
CTAT  
GACC (x2)  
GCCG (x2)  
GGGT (x2)  
GGTA  
GTAA  
GTAC (x2)  
GTAG (x2)  
TAAT  
TACG (x2)  
TACT  
TAGG (x3)  
TATA (x2)  
TATG  
TGAC (x3)  
TGCC (x3)  
TGC  
TTGC





# Dijkstra算法：Application and Beyond\*



## ■ 高效而优雅（graceful）的算法

An  $\Omega(n^2 \log n)$  Lower Bound to the Shortest Paths Problem<sup>††</sup>

Andrew C. Yao  
Computer Science Department, Stanford University

David M. Avis  
Department of Operations Research, Stanford University

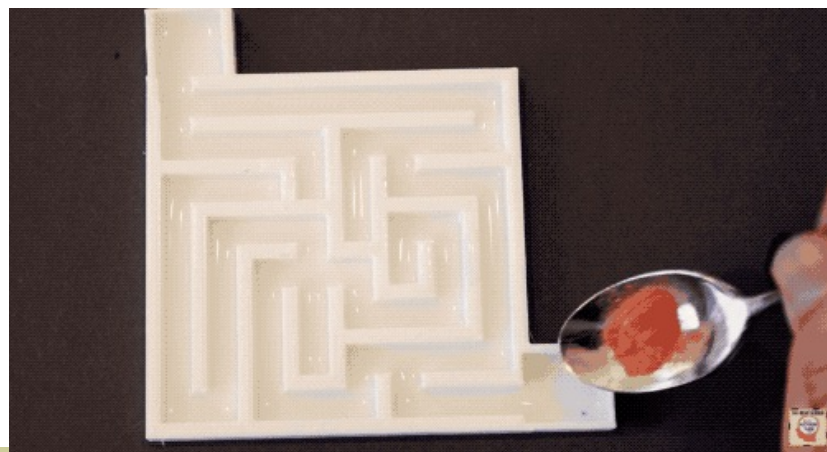
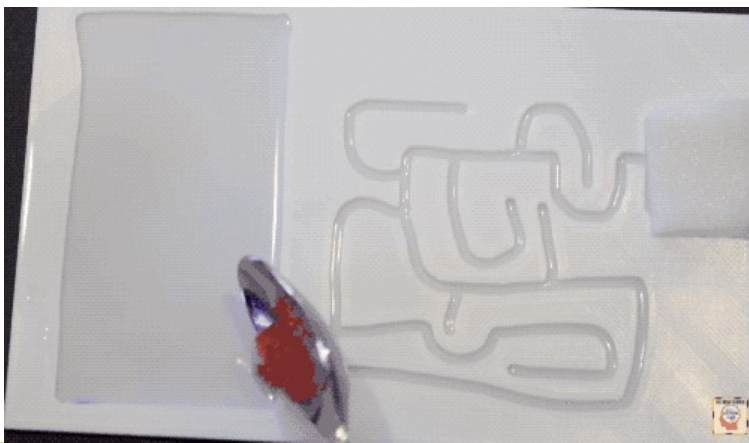
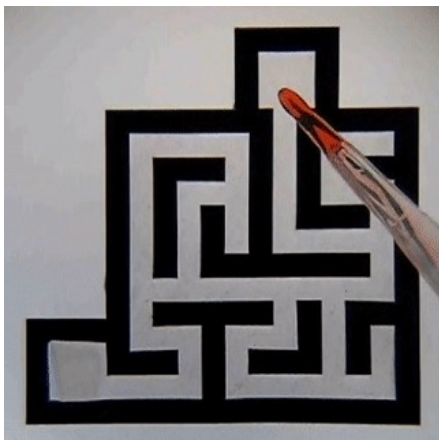
Ronald L. Rivest  
Laboratory for Computer Science, M.I.T.



# Dijkstra算法： Application and Beyond\*



## ■ Beyond Dijkstra Algorithm





# 本次课后作业



- 教材内容：[Rosen] 10.6.1, 10.6.2 节
- 课后习题：
  - Problem Set 26
- 提交时间：6月3日 10:00 前

