数据结构与算法设计

周　可

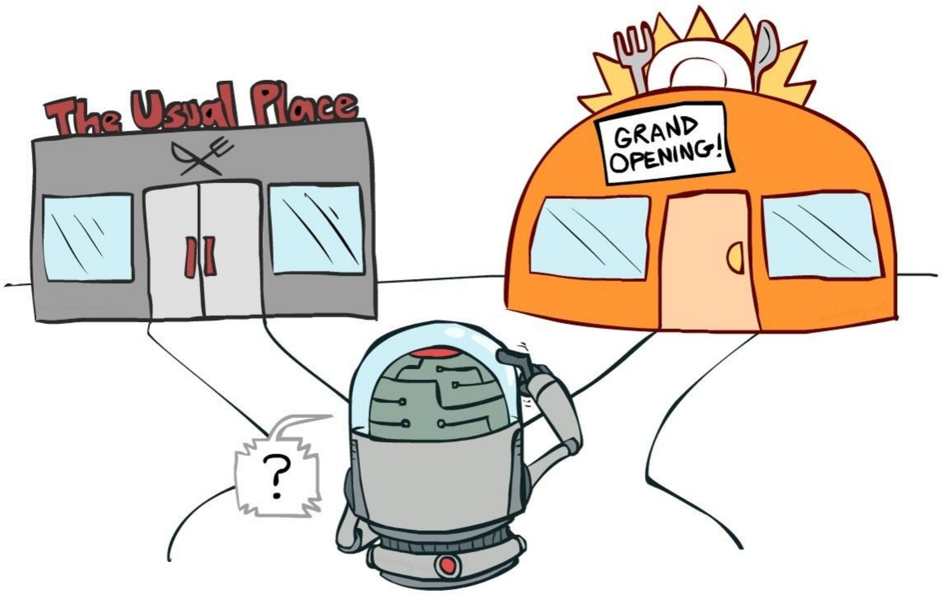
Mail：zhke@hust.edu.cn

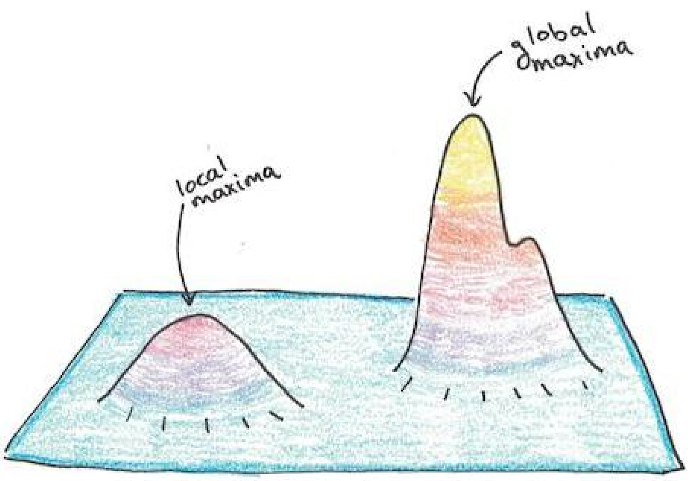
华中科技大学，武汉光电国家研究中心

1. Graph Representation
2. Graph Searching (BFS, DFS)
3. **Dijkstra Algorithm**
4. MST

课程回顾：

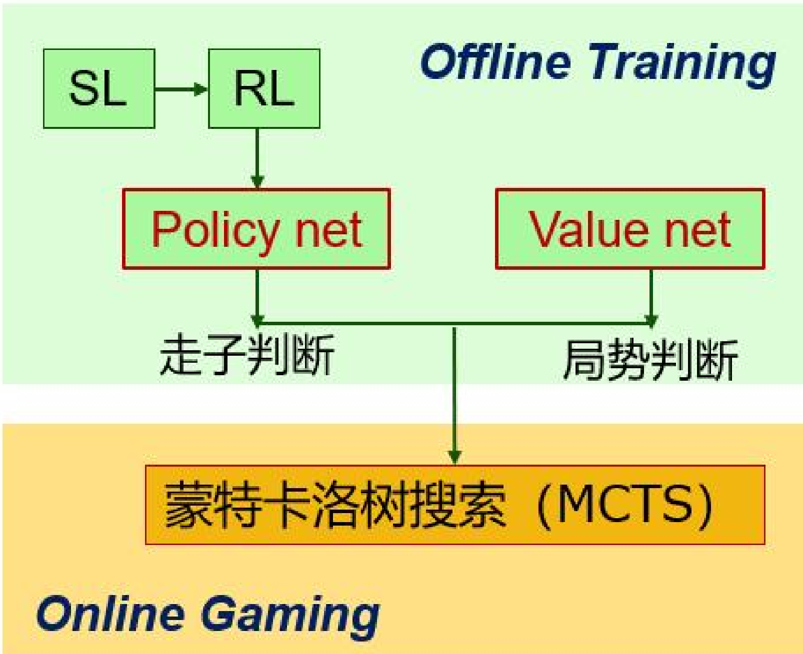
Exploration-Exploitation dilemma





Explore or Exploit: The Hidden Decision that Guides Your Life —— Scott H. Young

回顾：AI算法AlphaGo为何能够战胜人类?



AlphaGo工作原理示意图

任何完全信息博弈都是一种搜索。搜索复杂度取决于搜索空间的宽度和深度。

围棋：宽度约为250，深度约为150，总搜索空间约为250^{150}。

- Policy net（策略网络）：**减少搜索宽度**
- Value net（价值网络）：**减少搜索深度**

图注:

- SL（Supervised Learning，监督学习）：模仿人类
- RL（Reinforcement Learning，强化学习）：自我进化

Dijkstra算法

单源最短路径
(Single-Source Shortest Paths, SSSP)

生活中的最短路径问题

Q: Which path is the shortest?

最短路径问题的相关定义

- 已知：带权有向图 $G=(V,E)$

  权重函数 $w: E \to R$

  图中一条路径 $p=<v_0, v_1, \ldots, v_k>$ 的权重为：

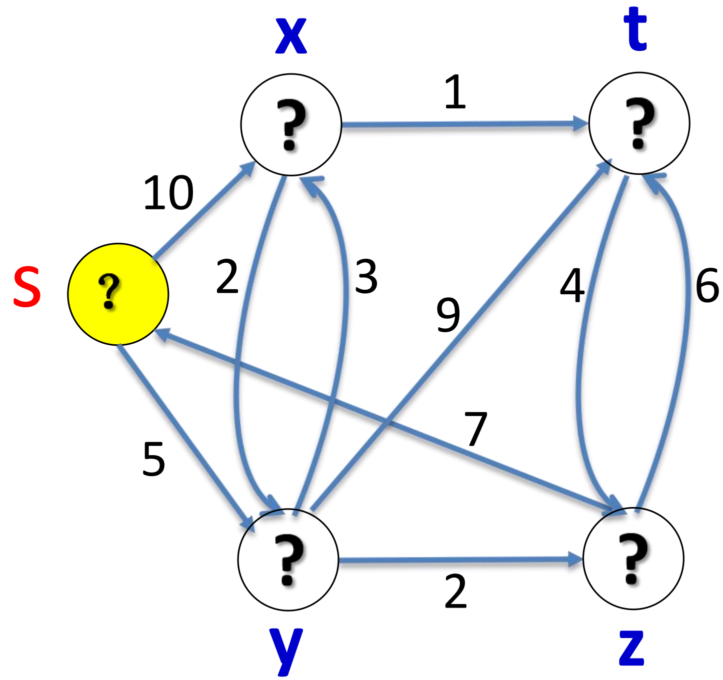
$$w(p)=\sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- 定义从结点 u到结点 v的**最短路径（权重）**为：

$$\partial(u,v)=\begin{cases}\min\{w(p): u \xrightarrow{p} v\} \\ \infty\end{cases}$$

单源最短路径（Single Source Shortest Path, SSSP）

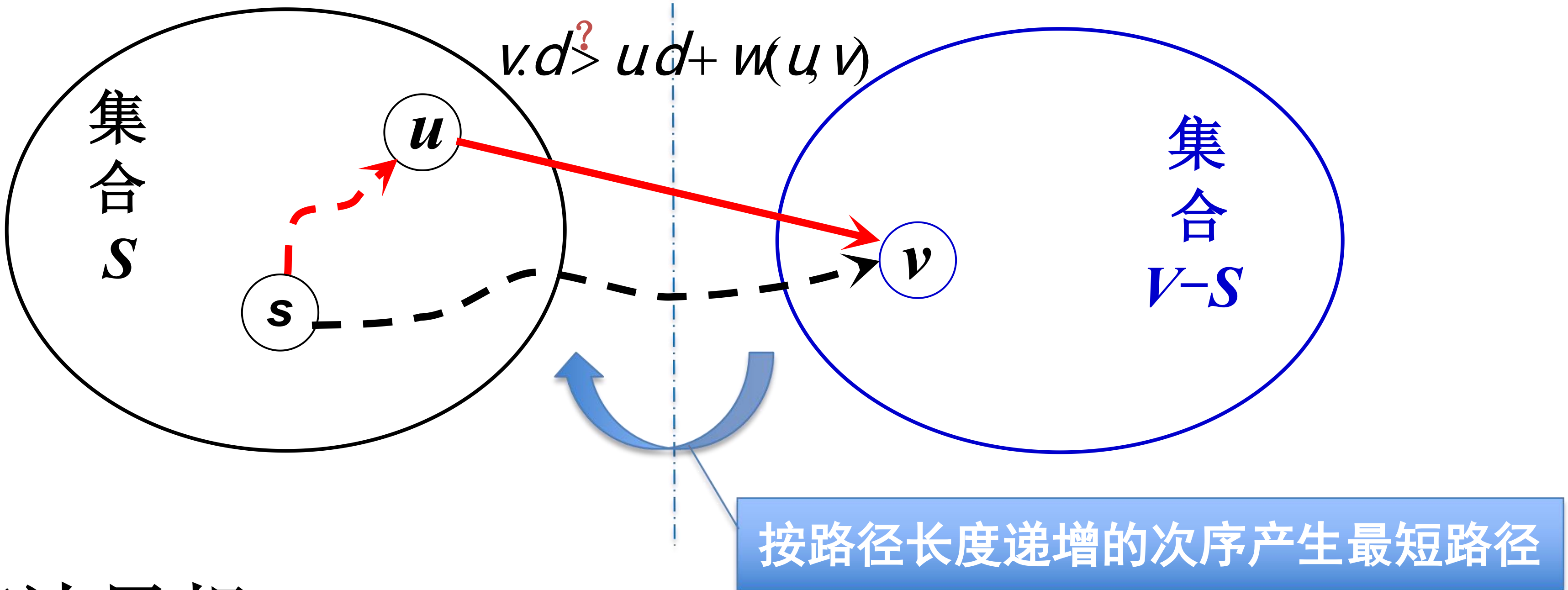
- 给定一个图 $G=(V,E)$，求从给定源结点 $s\in V$ 到每个结点 $v\in V$ 的最短路径。



Dijkstra算法

S: 到源点的最短路径已确定

V-S: 到源点的最短路径未确定



算法思想：

重复从节点集***V-S***中选择最短路径估计最小的节点***u***，将***u***加入到集合***S***，然后对所有从***u***发出的边进行松弛（**Relax**）。

算法描述及示例：

DIJKSTRA (G, w, s)

INITIALIZAE-SINGLE-SOURCE (G, s)

$S = \varnothing$

$Q = G.V$

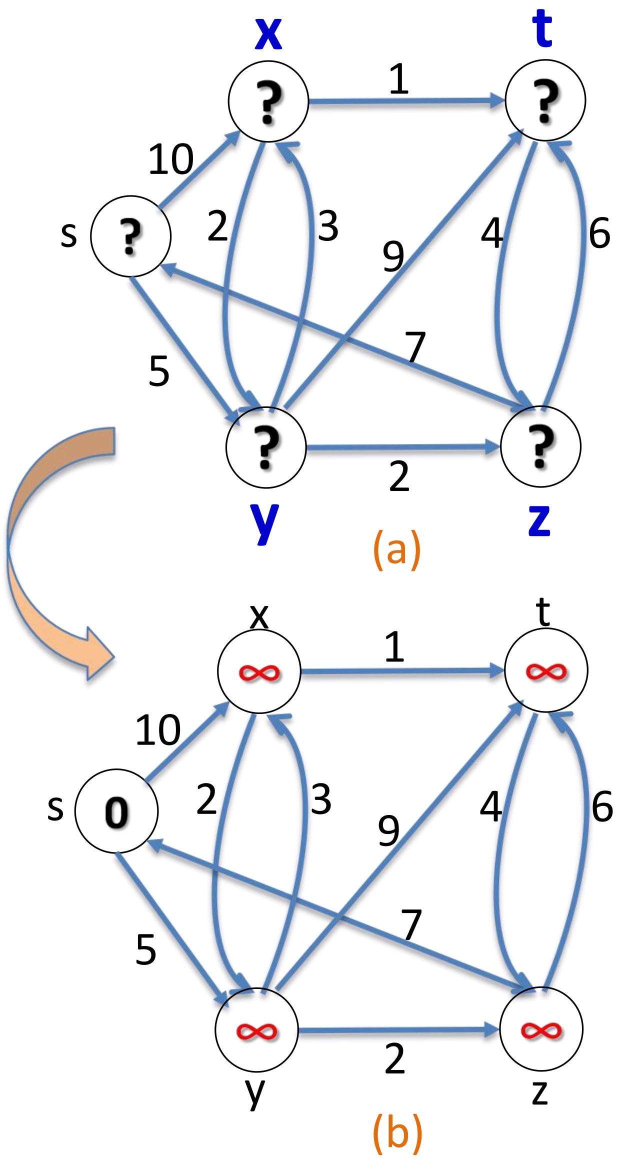
while $Q \neq \varnothing$

u = EXTRACT-MIN (Q)

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

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

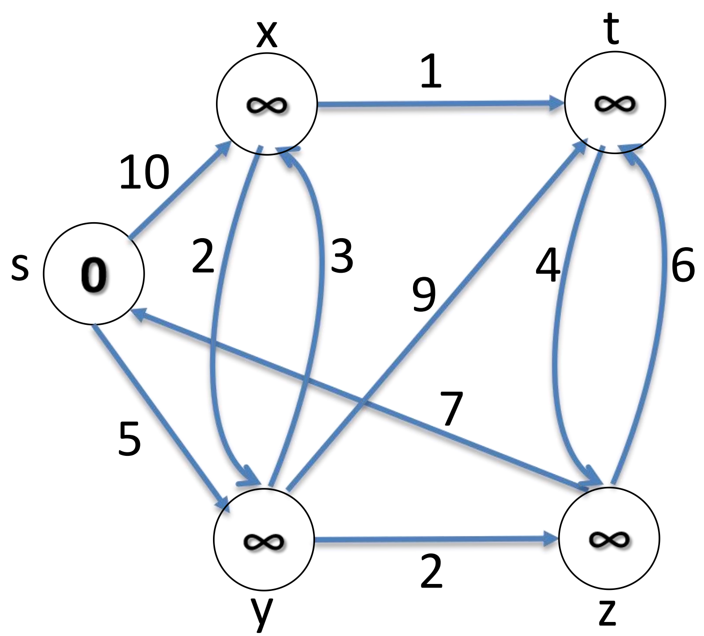
RELAX (u, v, w)



算法描述及示例：

DIJKSTRA (G, w, s)
    INITIALIZAE-SINGLE-SOURCE (G, s)
    $S = \varnothing$
    $Q = G.V$
    while $Q \neq \varnothing$
        u = EXTRACT-MIN (Q)
        $S = S \cup \{u\}$

        for each vertex $v \in G.Adj[u]$
            RELAX (u, v, w)



(c)

算法描述及示例：

DIJKSTRA (G, w, s)

INITIALIZAE-SINGLE-SOURCE (G, s)

$S = \varnothing$

$Q = G.V$

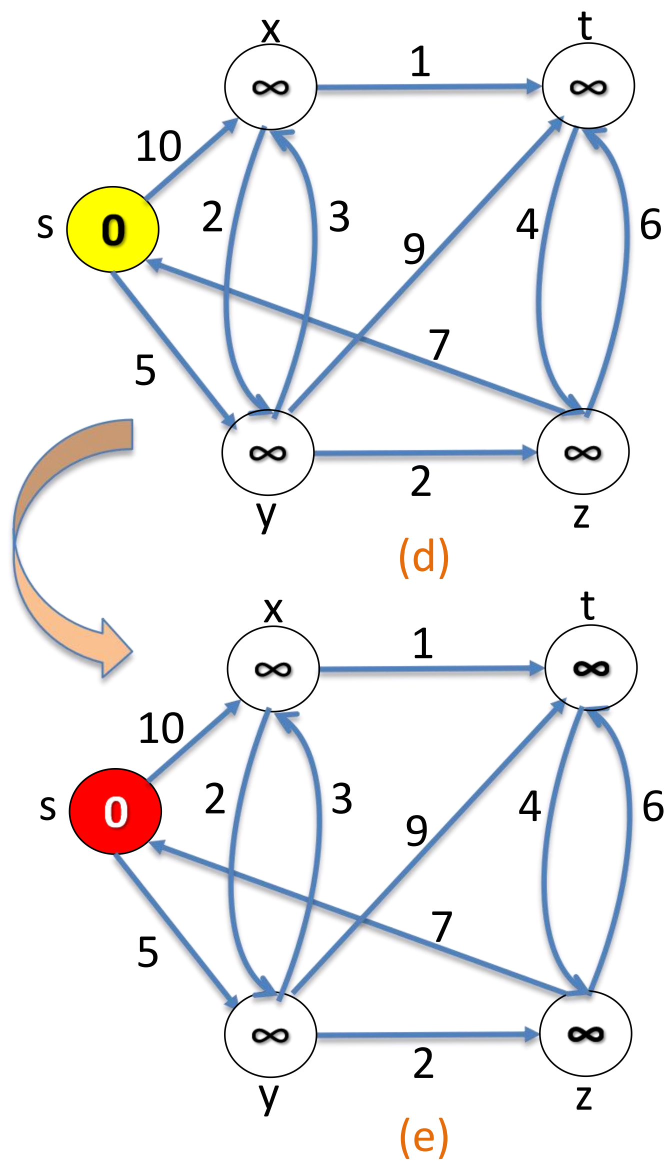
while $Q \neq \varnothing$

$u =$ EXTRACT-MIN (Q)

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

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

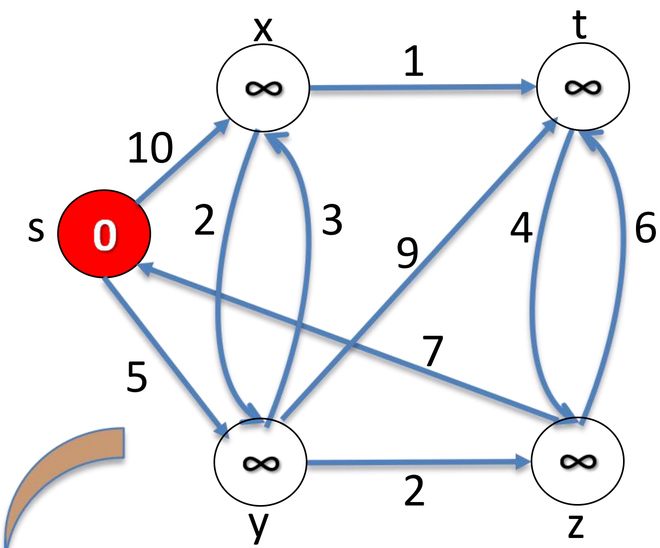
RELAX (u, v, w)



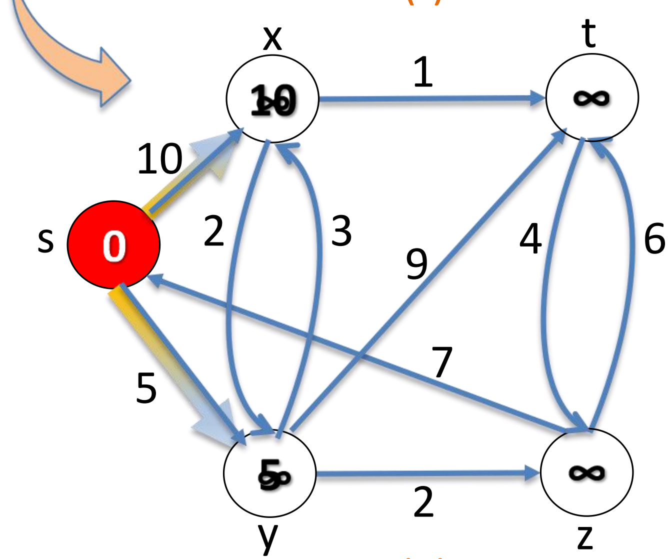
算法描述及示例：

DIJKSTRA (G, w, s)
    INITIALIZAE-SINGLE-SOURCE (G, s)
    $S = \varnothing$
    $Q = G.V$
    while $Q \neq \varnothing$
        u = EXTRACT-MIN (Q)
        $S = S \cup \{u\}$

        for each vertex $v \in G.Adj[u]$
            RELAX (u, v, w)



(f)



(g)

算法描述及示例：

DIJKSTRA (G, w, s)

    INITIALIZAE-SINGLE-SOURCE (G, s)

    S = ∅

    Q = G. V

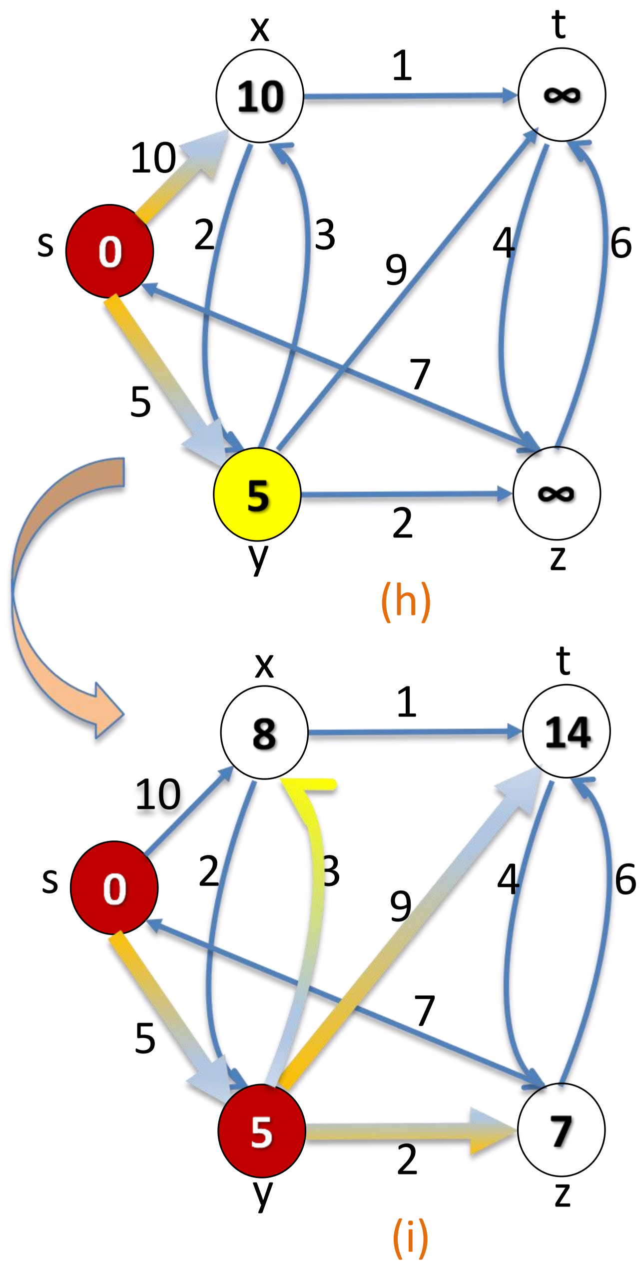
    while Q ≠ ∅

        u = EXTRACT-MIN (Q)

        S = S ∪ {u}

        for each vertex v∈G. Adj[u]

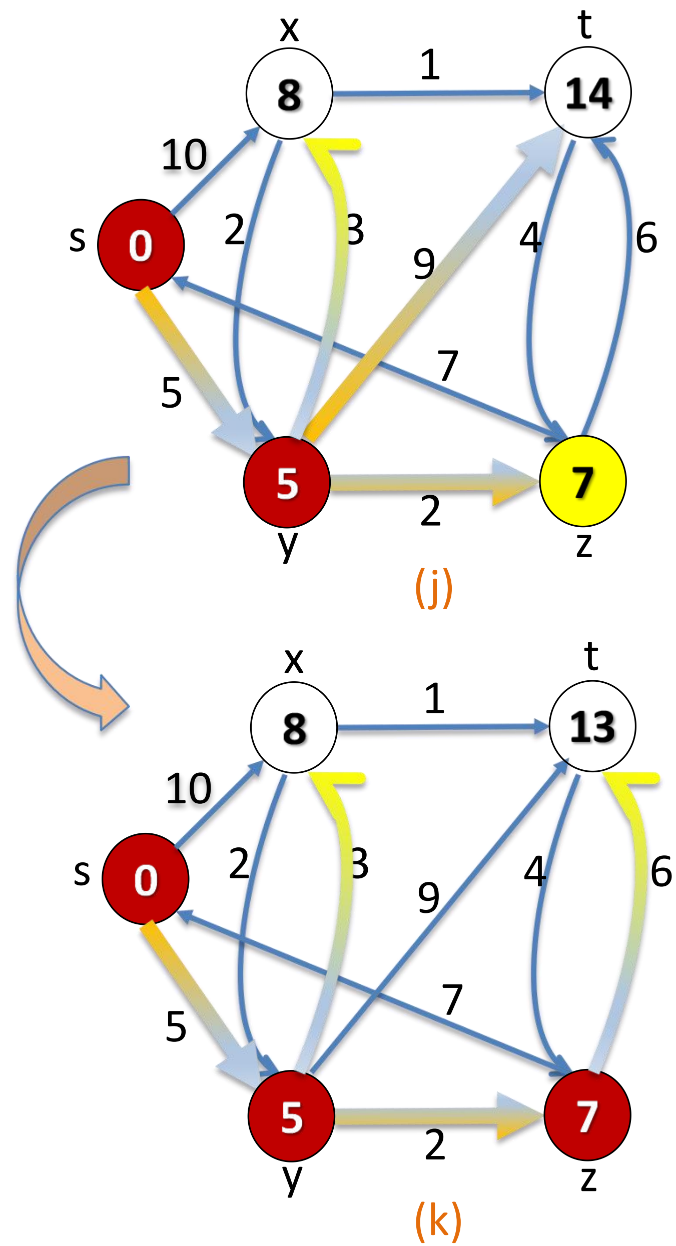
            RELAX (u, v, w)



算法描述及示例：

DIJKSTRA (G, w, s)
    INITIALIZAE-SINGLE-SOURCE (G, s)
    S = ∅
    Q = G. V
    while Q ≠ ∅
        u = EXTRACT-MIN (Q)
        S = S ∪ {u}

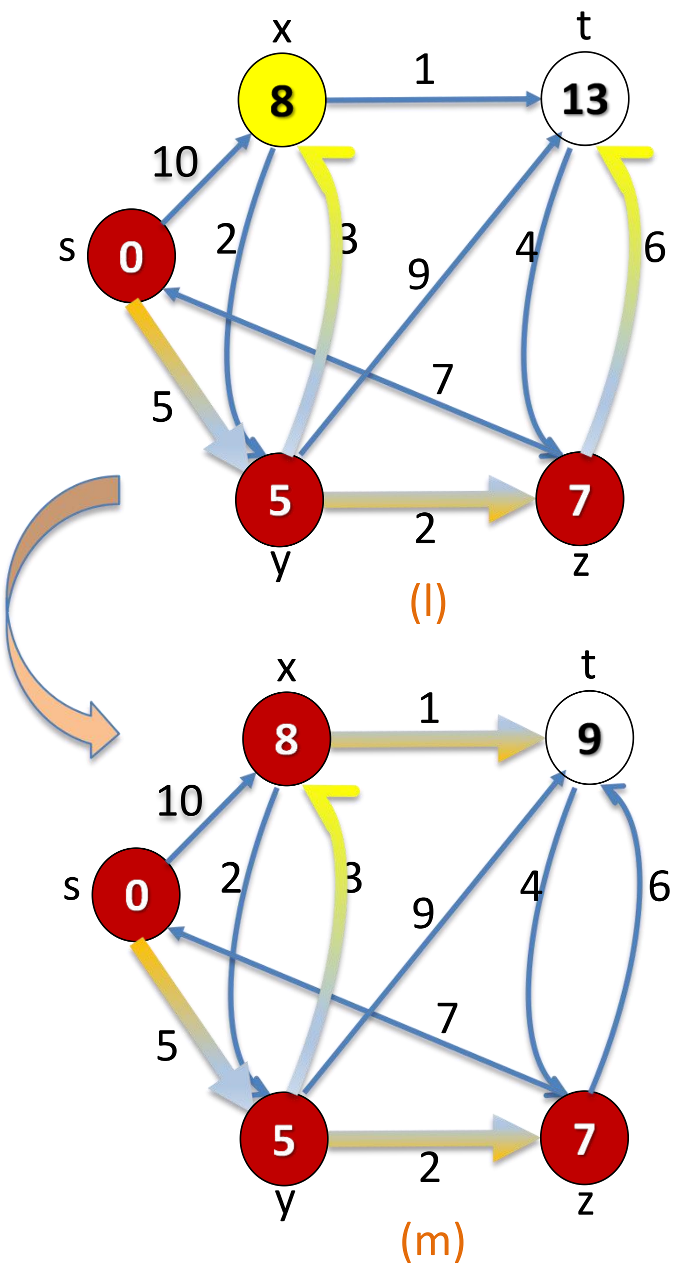
        for each vertex v∈G. Adj[u]
            RELAX (u, v, w)



算法描述及示例：

DIJKSTRA (G, w, s)
    INITIALIZAE-SINGLE-SOURCE (G, s)
    S = ∅
    Q = G. V
    while Q ≠ ∅
        u = EXTRACT-MIN (Q)
        S = S ∪ {u}

        for each vertex v∈G. Adj[u]
            RELAX (u, v, w)



算法描述及示例：

DIJKSTRA (G, w, s)

  INITIALIZAE-SINGLE-SOURCE (G, s)

  S = ∅

  Q = G. V

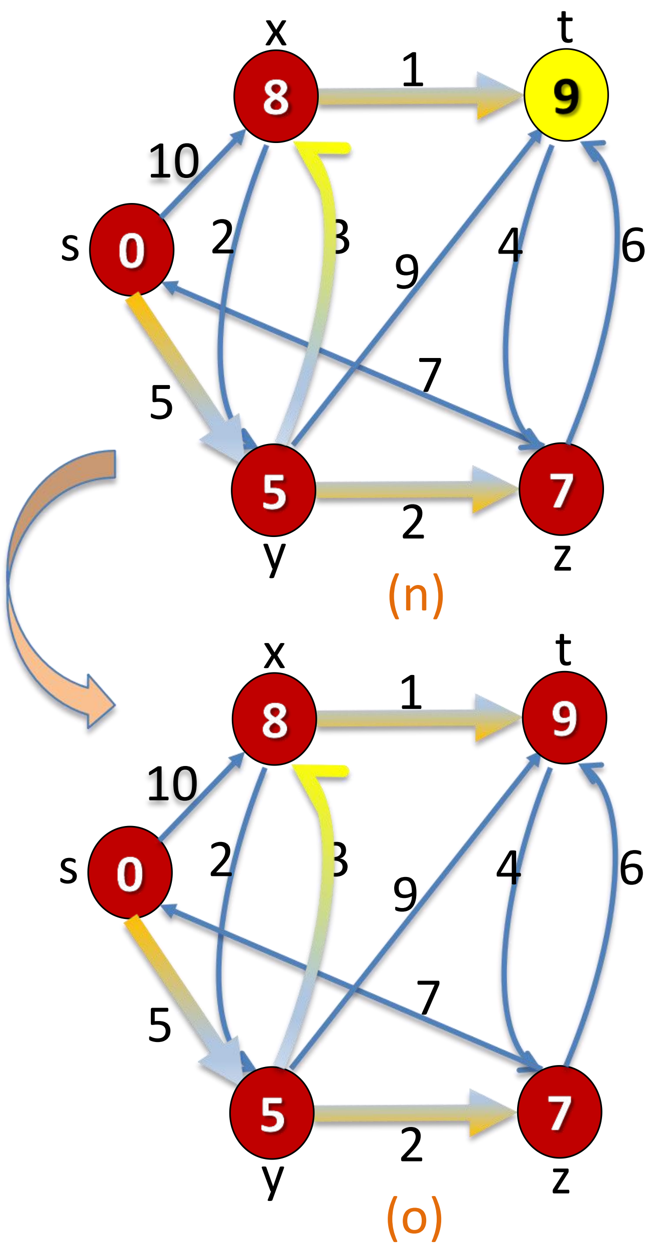
  while Q ≠ ∅

    u = EXTRACT-MIN (Q)

    S = S ∪ {u}

    for each vertex v∈G. Adj[u]

      RELAX (u, v, w)



算法复杂度分析：

DIJKSTRA (G, w, s)

(1) INITIALIZAE-SINGLE-SOURCE (G, s)

(2) $S = \varnothing$

(3) <u>Q = G. V</u>

(4) **while** $Q \neq \varnothing$

(5) <u>u = EXTRACT-MIN (Q)</u>

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

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

(8) <u>RELAX (u, v, w)</u>

三种队列操作：

INSERT: (3) → O(V)

EXTRACT-MIN: (5) → $O(V^2)$

DECREASE-KEY: (8) → O(E)

$\sum = O(V^2)$

Dijkstra算法总结：

包含DP思想

$u.d+w(u,v) < v.d \Rightarrow v.d = u.d+w(u,v)$

贪心选择

选最短路径对应的节点u → 对于$v \in G.Adj[u]$做松弛操作

Next Round

贪心策略的“形”，包裹了动态规划算法保证正确的“魂”。

生活中的最短路径问题

The shortest path is shown as above.

小结

- Dijkstra算法使用贪心策略
- 最短路径具有“最优子结构”性质
- 非常简单的贪心策略的“形”，包裹了动态规划算法保证正确的“魂”。

1. Graph Representation
2. Graph Searching (BFS, DFS)
3. Dijkstra Algorithm
4. **MST**

Minimum spanning trees

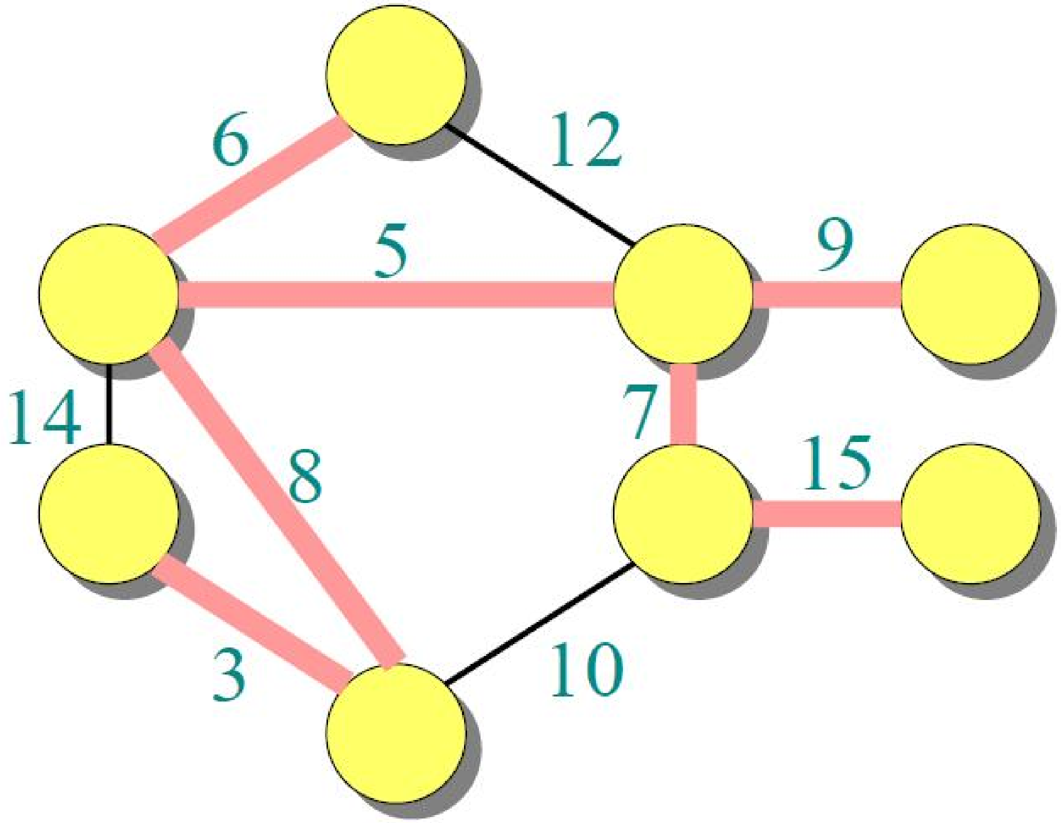
Input: A connected, undirected graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbb{R}$.

Output: A ***spanning tree*** T — a tree that connects all vertices $w(T) = \sum_{(u,v)\in T} w(u,v)$.

A ***spanning tree*** T — a tree that connects all vertices — of minimum weight:

$$w(T) = Min \sum_{(u,v)\in T} w(u,v).$$

Example of MST



生成树的性质

设G是n阶连通图，那么

(1) T是G的生成树当且仅当T无圈且有n-1条边

(2) 如果T是G的生成树，$e \notin T$，那么$T \cup \{e\}$含有一个圈C（回路）

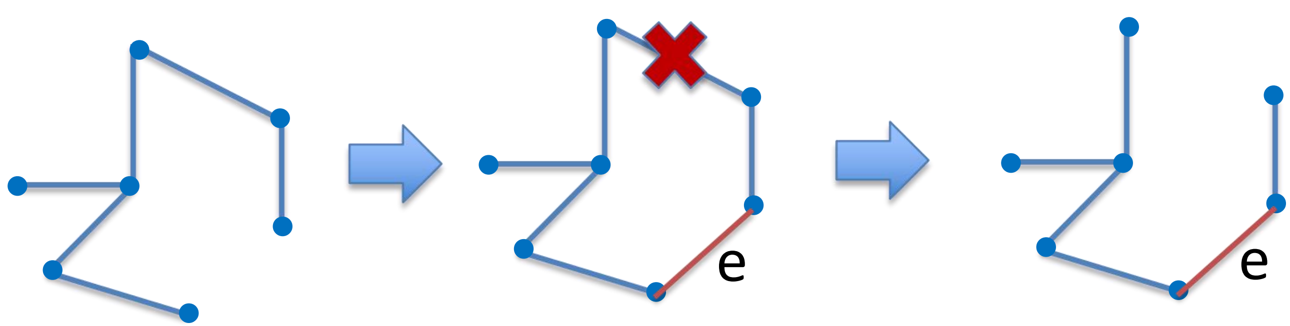
生成树的性质

设G是n阶连通图，那么

(1) T是G的生成树当且仅当T无圈且有n-1条边

(2) 如果T是G的生成树，$e \notin T$，那么$T \cup \{e\}$含有一个圈C（回路）

(3) 去掉圈C的任意一条边，就得到G的另外一棵生成树T'。



生成树性质的应用

- 算法步骤：选择边
- 约束条件：不形成回路
- 截止条件：边数达到n-1

改进生成树 T 的方法：

在T中加一条非树边e'，形成回路C，在C中去掉一条树边e，形成一棵新的生成树T'

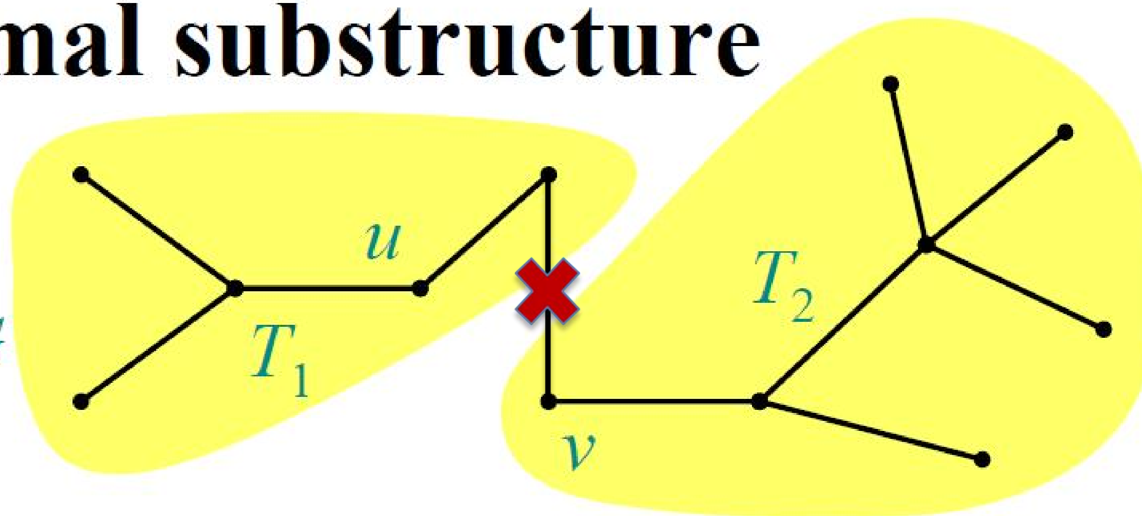
$W(T')-W(T) = W(e')-W(e)$

若$W(e') \leq W(e)$，则$W(T') \leq W(T)$

Optimal substructure

MST T:

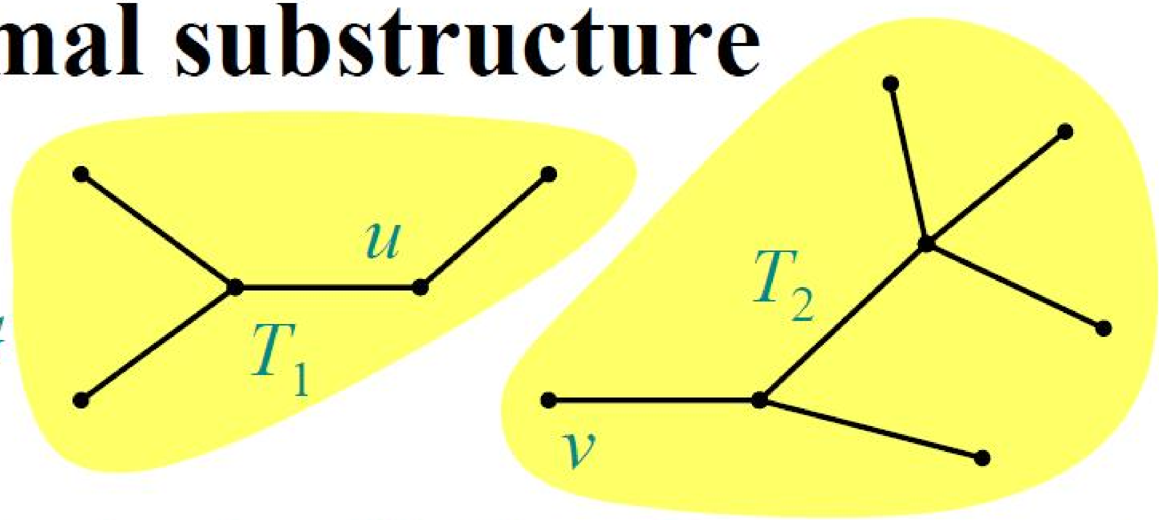
(Other edges of G are not shown.)



Optimal substructure

MST T:

(Other edges of G are not shown.)



Remove any edge $(u, v) \in T$. Then, T is partitioned into two subtrees T_1 and T_2.

Theorem. The subtree T_1 is an MST of $G_1 = (V_1, E_1)$, the subgraph of G ***induced*** by the vertices of T_1:

$$V_1 = \text{vertices of } T_1,$$
$$E_1 = \{ (x, y) \in E : x, y \in V_1 \}.$$

Similarly for T_2.

Proof of optimal substructure

Proof. Cut and paste:

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

If T_1' were a lower-weight spanning tree than T_1 for G_1, then $T' = \{(u, v)\} \cup T_1' \cup T_2$ would be a lower-weight spanning tree than T for G. $\square$

Do we also have overlapping subproblems?

- Yes.

Great, then dynamic programming may work!

- Yes, but MST exhibits another powerful property which leads to an even more efficient algorithm.

Hallmark for "greedy" algorithms

Greedy-choice property

A locally optimal choice is globally optimal.

Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.

Generic algorithm for MST

Generic-MST(G,w)

1 A=Φ

2 **While** A does not form a MST

3 find an edge(u,v) that is safe for A

4 A=A∪{(u,v)}

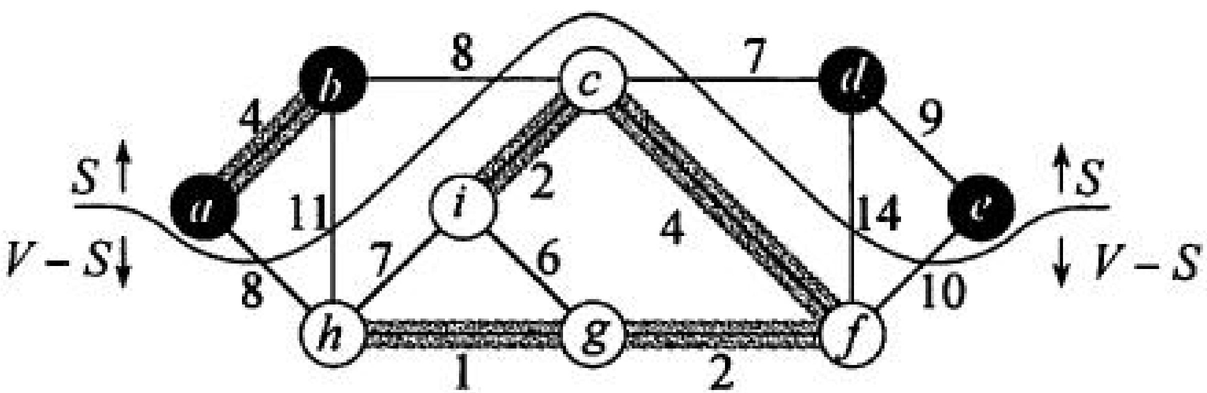
5 return A

循环不变式：每遍循环之前，A是某棵MST的子集。

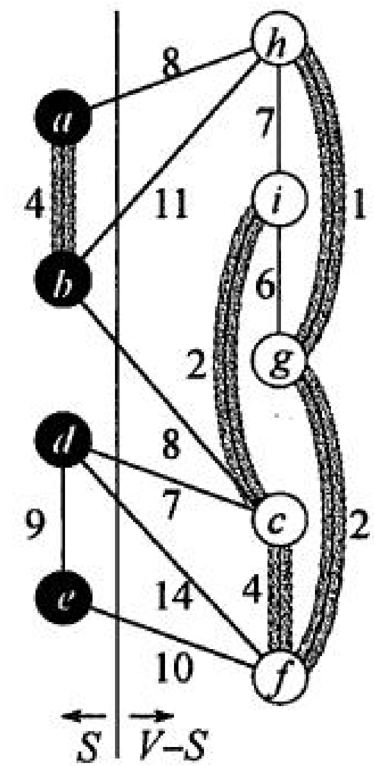
Q: How to find the safe edge?

Some notions

无向图G＝(V,E)的一个切割(S, V−S)是集合V的一个划分。如果一条边(u, v)∈E的一个端点位于集合S，另一个端点位于集合V−S，则称该边**横跨切割**(S, V−S)。在横跨切割的所有边中，权重最小的边称为**轻量级边**。



(a)



(b)

Q: How to find the safe edge?

定理： 设G=(V,E)是一个在边E上定义了实数值权重函数w的连通无向图。设集合A为E的一个子集，且A包括在图G的某棵最小生成树中，设(S, V-S)是图中**尊重**集合A的任意一个切割，又设(u, v)是横跨切割(S, V-S)的一条轻量级边。那么边(u, v)对于集合A是**安全**的。

Theorem. Let T be the MST of $G = (V, E)$, and let $A \subseteq V$. Suppose that $(u, v) \in E$ is the least-weight edge connecting A to $V - A$. Then, $(u, v) \in T$.

Generic algorithm for MST

Generic-MST(G,w)

1 A=Φ

2 **While** A does not form a MST

3 find an edge(u,v) that is safe for A

4 A=A∪{(u,v)}

5 return A

循环不变式：每遍循环之前，A是某棵MST的子集。

Minimum spanning trees

- Prim algorithm
- Kruskal algorithm

MST property

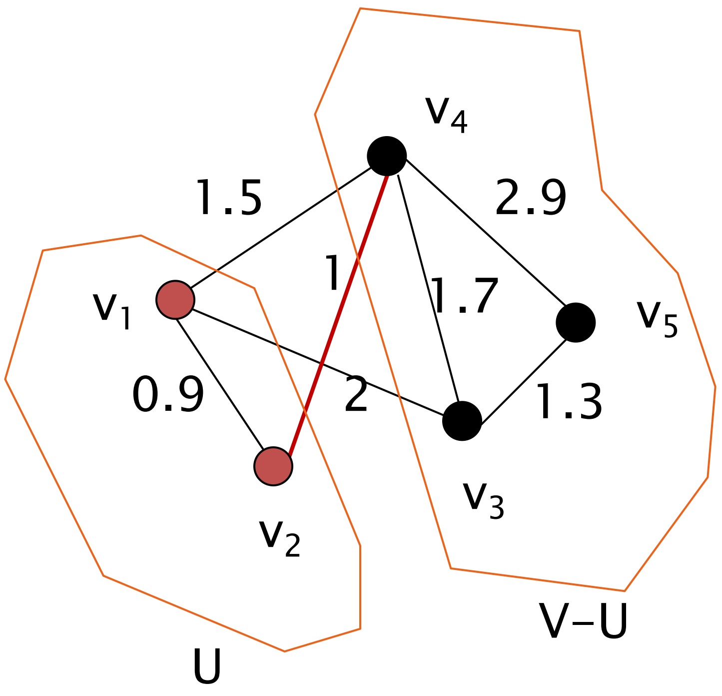
To obtain MST, we must exploit the useful property.

MST-Property:

The lightest edge e between

U and V-U must be in some

Minimum spanning tree of G.



How do we prove this?

Generic algorithm for MST

Generic-MST(G,w)

1 A=Φ

2 **While** A does not form a MST

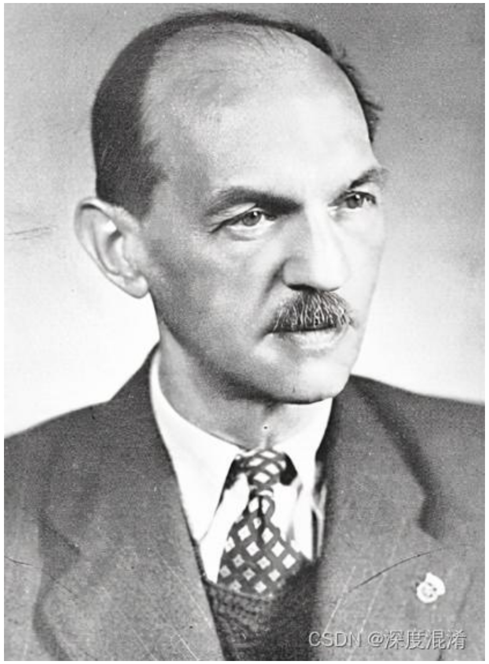
3 find an edge(u,v) that is safe for A

4 A=A∪{(u,v)}

5 return A

循环不变式：每遍循环之前，A是某棵MST的子集。

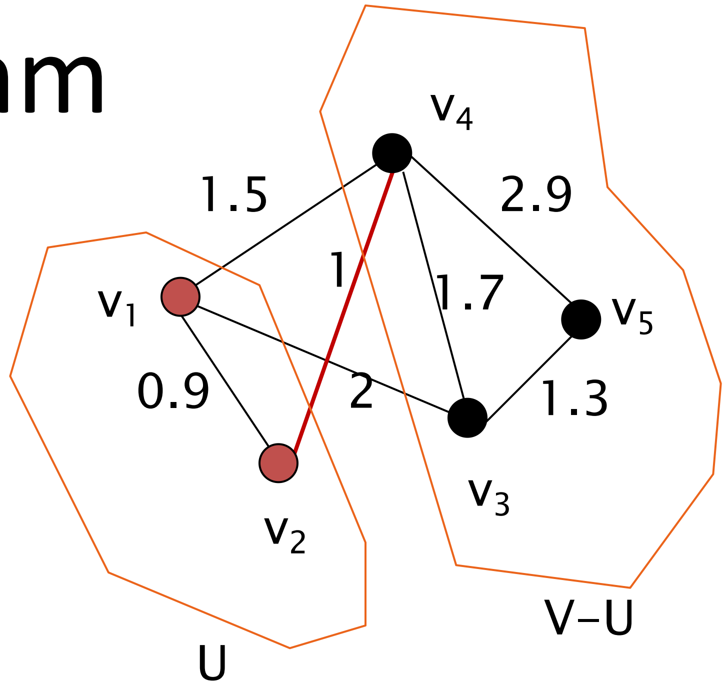
Prim’s algorithm



Vojtěch Jarník

Prim算法（普里姆算法），是1930年捷克数学家沃伊捷赫·亚尔尼克（Vojtěch Jarník）最早设计；1957年，由美国计算机科学家罗伯特·普里姆（Robert C. Prim）独立实现。

Prim's algorithm

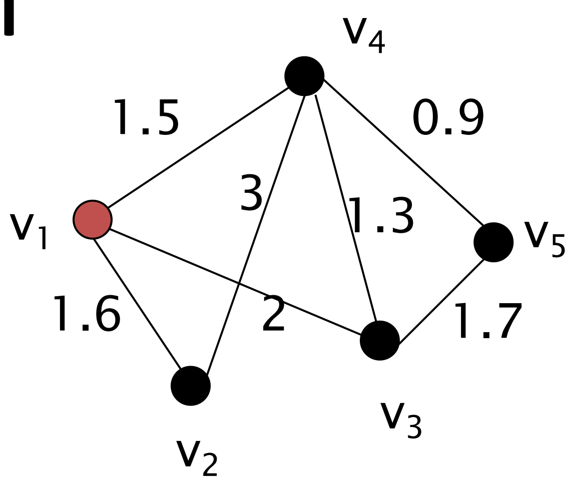


Let G=(V,E)=({1,2,…,n},E)

Prim(G,T) //T will be the output

1. T ← Ø
2. U ← {1}
3. While U ≠ V
4. let (u,v) be the lightest edge with u∈U and v∈V-U
5. T ← T \union {(u,v)}
6. U ← U \union {v}

Prim's algorithm

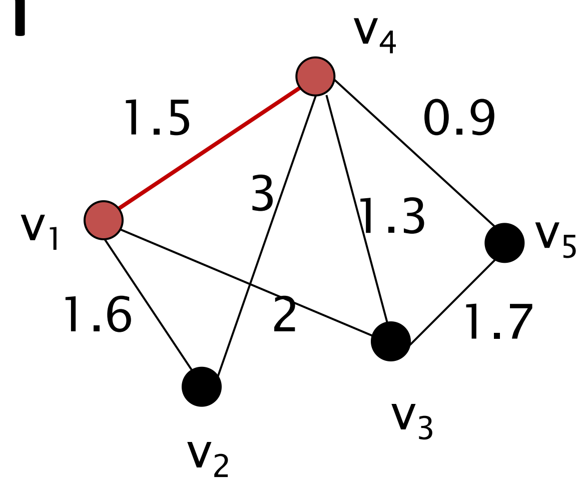


Let G=(V,E)=({1,2,…,n},E)

Prim(G,T) //T will be the output

1. T ← Ø
2. U ← {1}
3. While U ≠ V
4. let (u,v) be the lightest edge with u∈U and v∈V-U
5. T ← T \union {(u,v)}
6. U ← U \union {v}

Prim’s algorithm

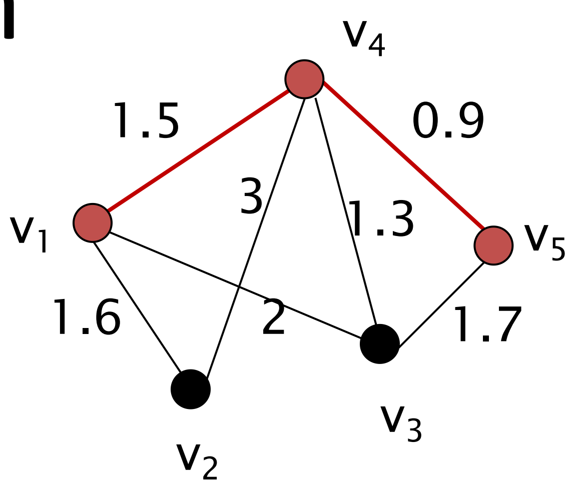


Let G=(V,E)=({1,2,…,n},E)

Prim(G,T) //T will be the output

1. $T \leftarrow \emptyset$
2. $U \leftarrow \{1\}$
3. While $U \neq V$
4. let (u,v) be the lightest edge with $u \in U$ and $v \in V-U$
5. $T \leftarrow$ T \union {(u,v)}
6. $U \leftarrow$ U \union {v}

Prim's algorithm

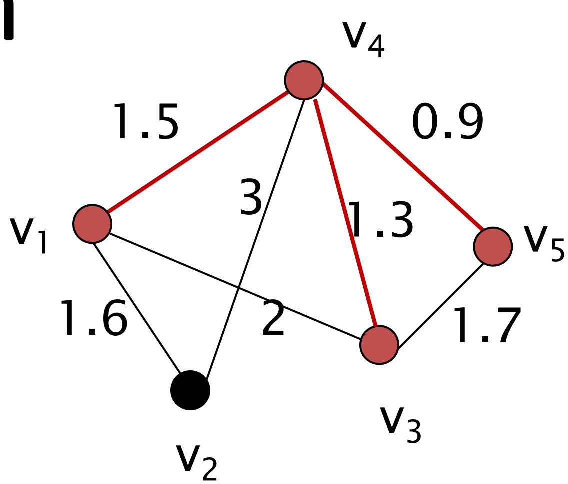


Let G=(V,E)=({1,2,…,n},E)

Prim(G,T) //T will be the output

1. T ← Ø
2. U ← {1}
3. While U ≠ V
4. let (u,v) be the lightest edge with u∈U and v∈V-U
5. T ← T \union {(u,v)}
6. U ← U \union {v}

Prim’s algorithm

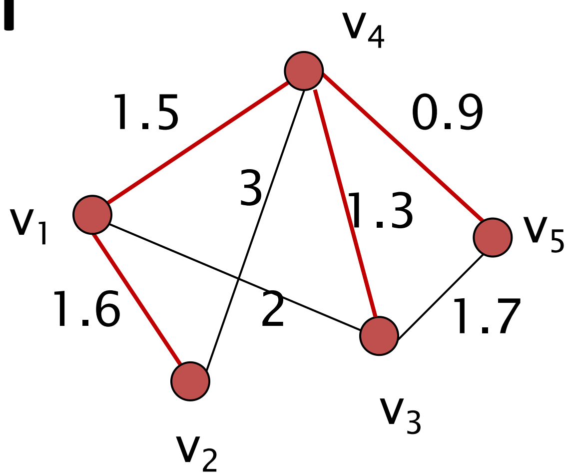


Let G=(V,E)=({1,2,…,n},E)

Prim(G,T) //T will be the output

1. $T \leftarrow \emptyset$
2. $U \leftarrow \{1\}$
3. While $U \neq V$
4. let (u,v) be the lightest edge with $u \in U$ and $v \in V-U$
5. $T \leftarrow T$ \union {(u,v)}
6. $U \leftarrow U$ \union {v}

Prim's algorithm



Let G=(V,E)=({1,2,…,n},E)

Prim(G,T) //T will be the output

1. T $\leftarrow$ Ø
2. U $\leftarrow$ {1}
3. While U $\neq$ V
4. let (u,v) be the lightest edge with u$\in$U and v$\in$V-U
5. T $\leftarrow$ T \union {(u,v)}
6. U $\leftarrow$ U \union {v}

Kruskal

艾兹格·W·迪科斯彻（Edsger Wybe Dijkstra，1930年5月11日~2002年8月6日），生于荷兰鹿特丹，计算机科学家，毕业就职于荷兰莱顿大学，早年钻研物理及数学，而后转为计算学。曾在1972年获得过素有计算机科学界的诺贝尔奖之称的图灵奖，之后，他还获得过1974年AFIPS Harry Goode Memorial Award、1989年ACM SIGCSE计算机科学教育教学杰出贡献奖、以及2002年ACM PODC最具影响力论文奖。
2002年8月6日，与癌症抗争多年后，在荷兰Nuenen自己的家中去世，享年72岁。

主要成就

1.提出“goto有害论”;

2.提出信号量和PV原语;

3.解决了“哲学家就餐”问题;

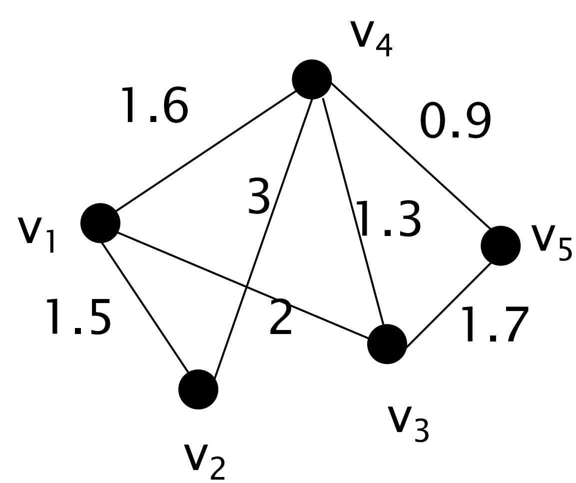
4.Dijkstra最短路径算法和银行家算法的创造者;

5.第一个Algol 60编译器的设计者和实现者;

6.THE操作系统的设计者和开发者;

Kruskal's algorithm

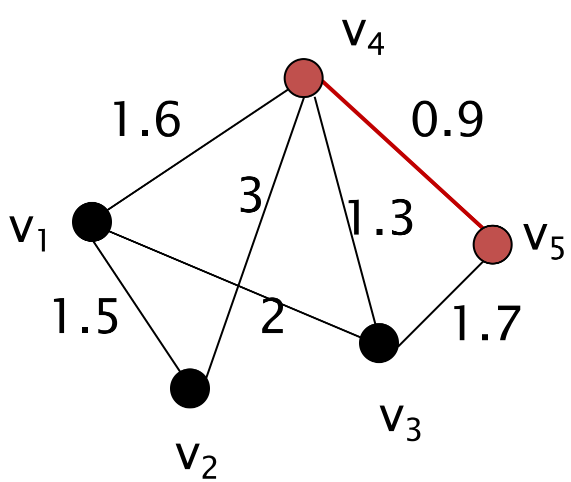
```
Kruskal(G,w)
A ←Ø //A is a forest
For each vertex v in V[G]
    do MAKE-SET(v)
Sort edges of E in nondecreasing order by weight w
For each edge (u,v) in E (taken in nondecreasing order by weight)
    do if FIND-SET(u) ≠ FIND-SET(v)
        then A ← A U {(u,v)}
            UNION(u,v)
Return A
```



Kruskal's algorithm

Kruskal(G,w)

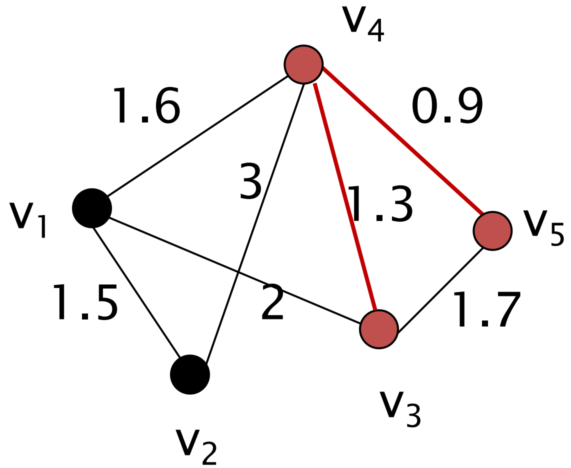
1. A ←Ø //A is a forest
2. For each vertex v in V[G]
3. do MAKE-SET(v)
4. Sort edges of E in nondecreasing order by weight w
5. For each edge (u,v) in E (taken in nondecreasing order by weight)
6. do if FIND-SET(u) ≠ FIND-SET(v)
7. then A ← A U {(u,v)}
8. UNION(u,v)
9. Return A



Kruskal's algorithm

Kruskal(G,w)

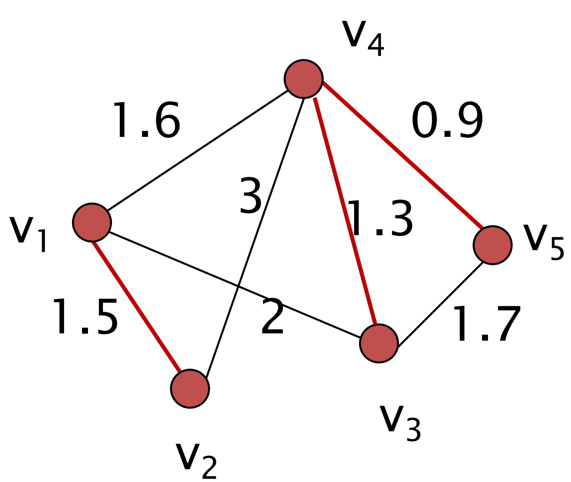
1. A ←Ø //A is a forest
2. For each vertex v in V[G]
3. do MAKE-SET(v)
4. Sort edges of E in nondecreasing order by weight w
5. For each edge (u,v) in E (taken in nondecreasing order by weight)
6. do if FIND-SET(u) ≠ FIND-SET(v)
7. then A ← A U {(u,v)}
8. UNION(u,v)
9. Return A



Kruskal's algorithm

Kruskal(G,w)

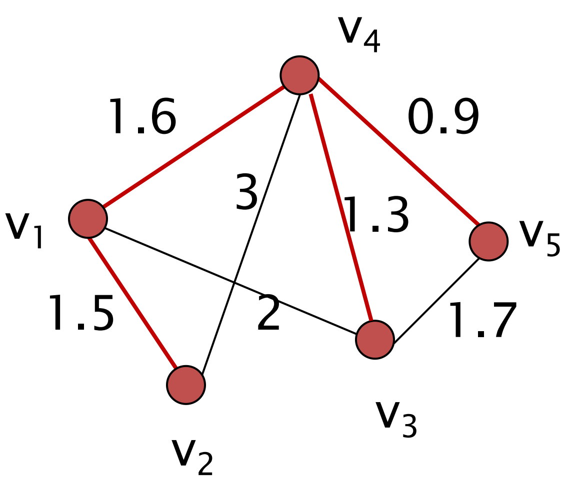
1. A ←Ø //A is a forest
2. For each vertex v in V[G]
3.   do MAKE-SET(v)
4. Sort edges of E in nondecreasing order by weight w
5. For each edge (u,v) in E (taken in nondecreasing order by weight)
6.   do if FIND-SET(u) ≠ FIND-SET(v)
7.   then A ← A U {(u,v)}
8.   UNION(u,v)
9. Return A



Kruskal's algorithm

Kruskal(G,w)

1. A ← Ø //A is a forest
2. For each vertex v in V[G]
3. do MAKE-SET(v)
4. Sort edges of E in nondecreasing order by weight w
5. For each edge (u,v) in E (taken in nondecreasing order by weight)
6. do if FIND-SET(u) ≠ FIND-SET(v)
7. then A ← A U {(u,v)}
8. UNION(u,v)
9. Return A



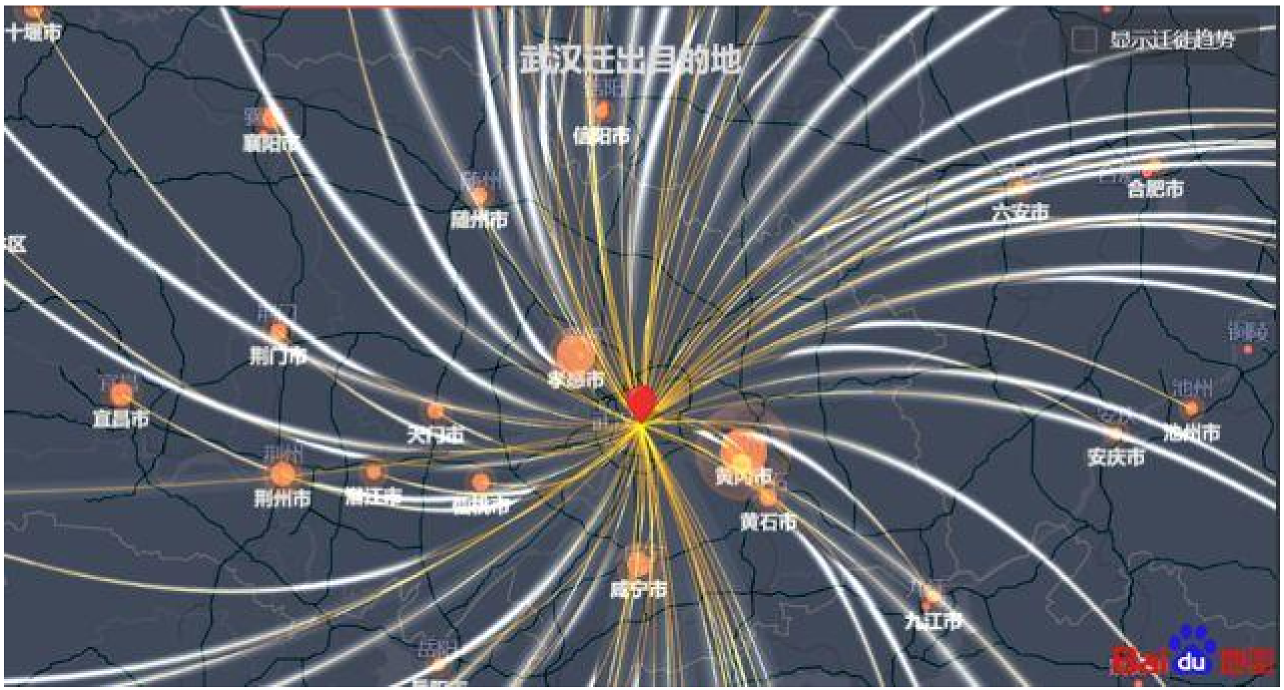
总结: 两种MST算法 (Prim/Kruskal)

Prim算法	Kruskal算法
按顶点构建生成树	按边构建生成树
构建中，总是一棵树	构建中，多棵树的森林
适用于稠密图（边多）	适用于稀疏图（边少）
时间复杂度O($\|V\|^2$)	时间复杂度O($\|E\|\log\|E\|$)
贪心策略	

思考：

- Dijkstra算法、Prim算法、Kruskal算法之间的异同，它们与图的搜索算法（BFS/DFS）的联系？

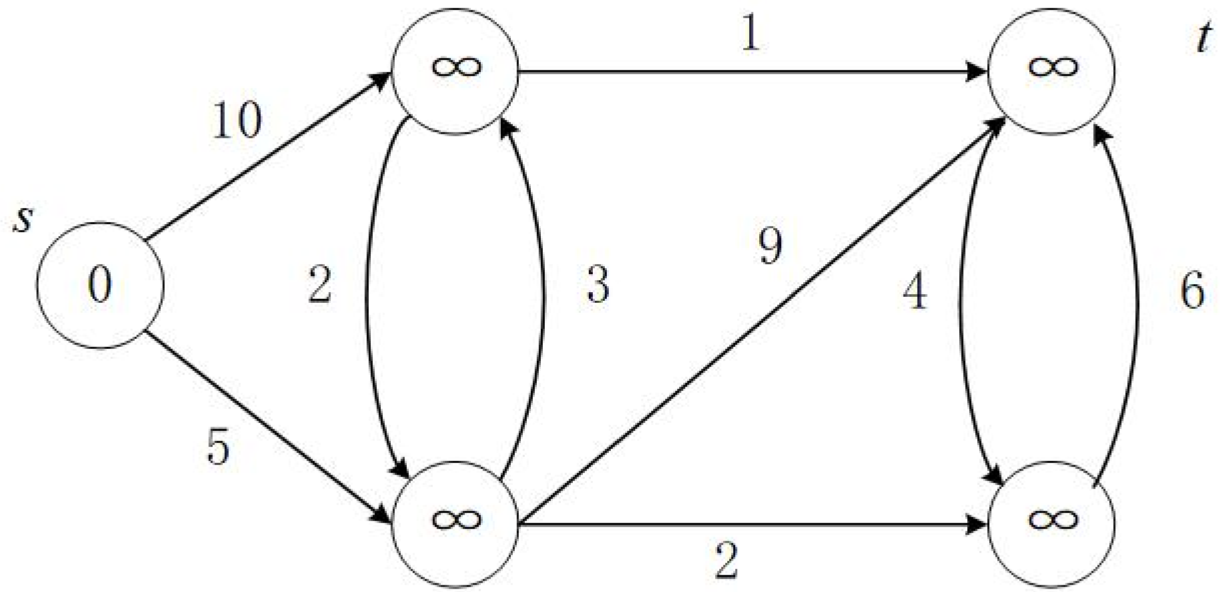
拓展：图论技术的使用——大数据算法下的全民抗疫



使用最小生成树模型对病毒感染进行流行病学调查

作业

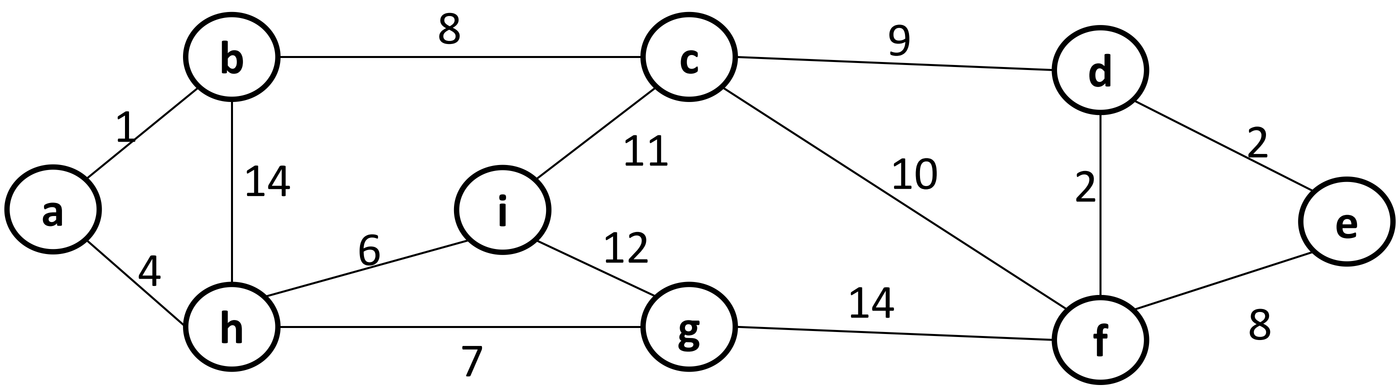
- 1. Please use the Dijkstra's algorithm to find the shortest path from *s* to *t* in the figure below, you should draw the process and mark the selected edges and nodes of each step in detail.

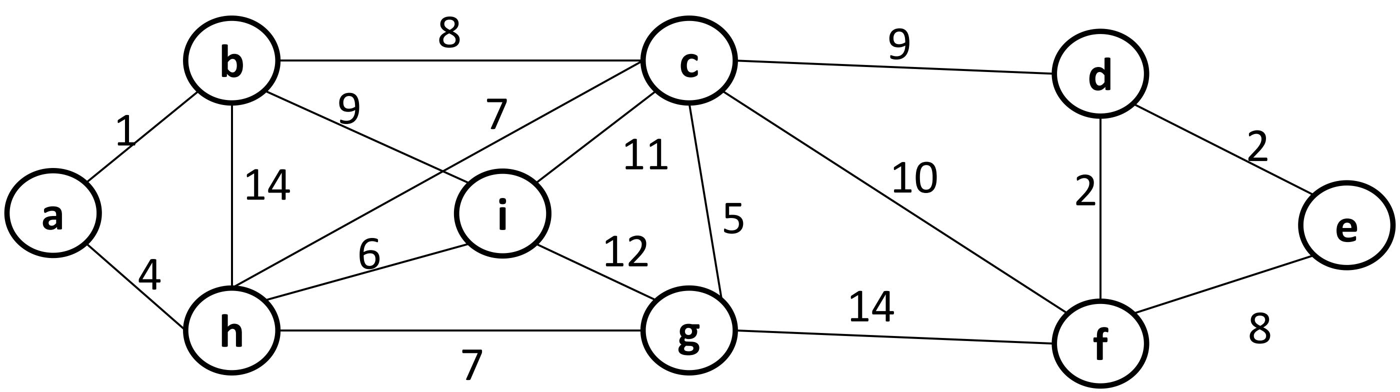


2：

（1）请分别使用Kruskal算法和Prim算法对下述两张图构造图的最小生成树，并画出构造流程；

（2）针对问题（1）的构造复杂程度，思考当图为稀疏图时，Kruskal算法和Prim算法中哪一个会更加适用，当图为稠密图时情况又如何，请给出你的分析。





Thank You!

Q&A