图算法篇：单源最短路径问题之Dijkstra算法

童咏昕

北京航空航天大学
计算机学院

中国大学MOOC北航《算法设计与分析》

提纲

问题背景

算法思想

算法实例

算法分析

算法性质

问题背景

- 从知春路到其他站点，如何安排路线?

问题背景

- 从知春路到其他站点，如何安排路线？

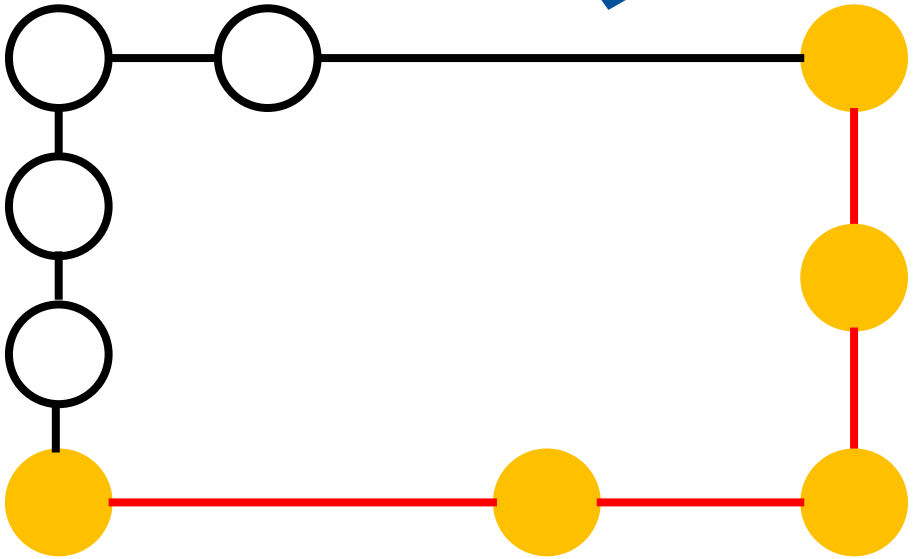
经停最少

问题背景

- 从知春路到其他站点，如何安排路线？

经停最少

无权图

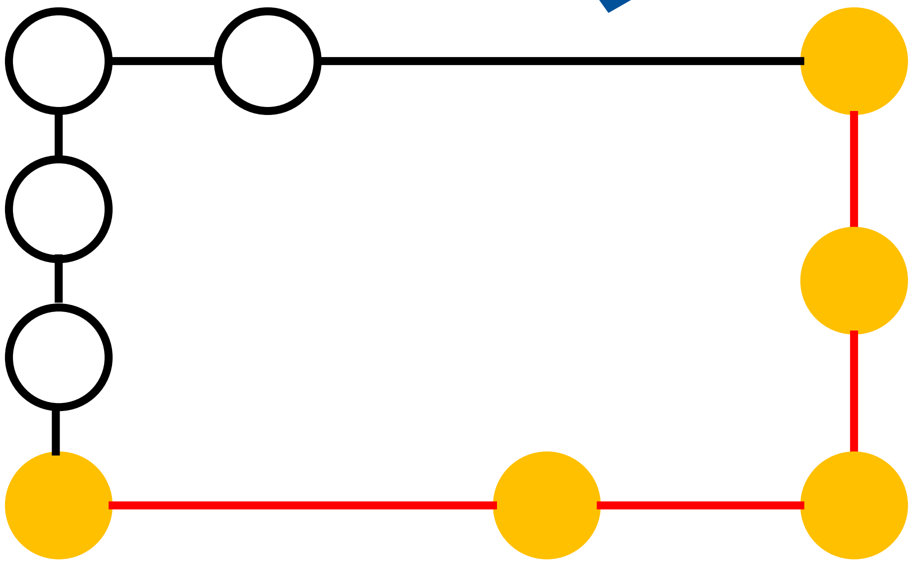


问题背景

- 从知春路到其他站点，如何安排路线？

经停最少

无权图



使用广度优先搜索求最短路径

问题背景

- 从知春路到其他站点，如何安排路线？

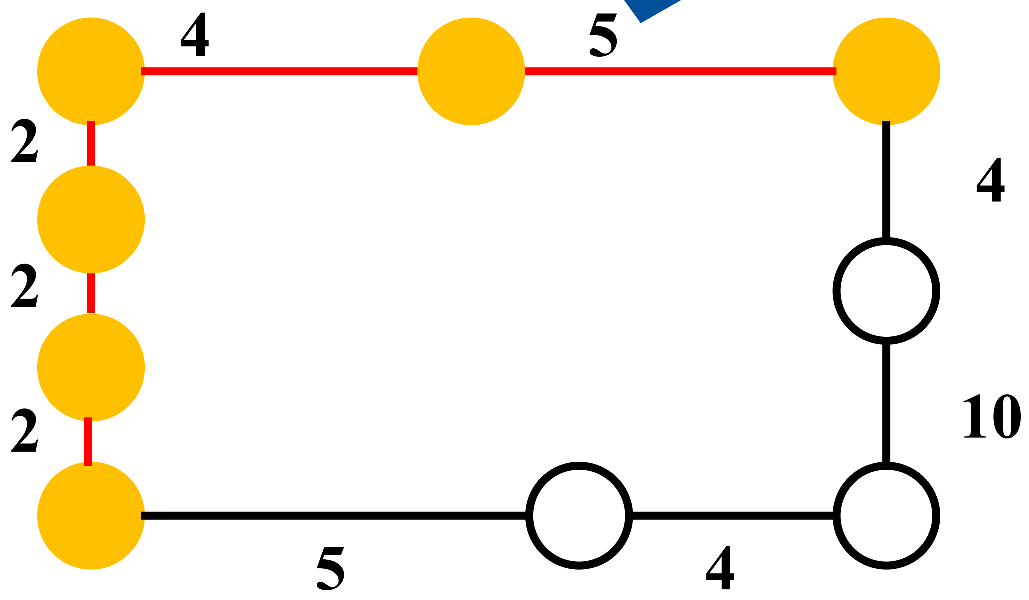
时间最短

问题背景

- 从知春路到其他站点，如何安排路线?

时间最短

带权图

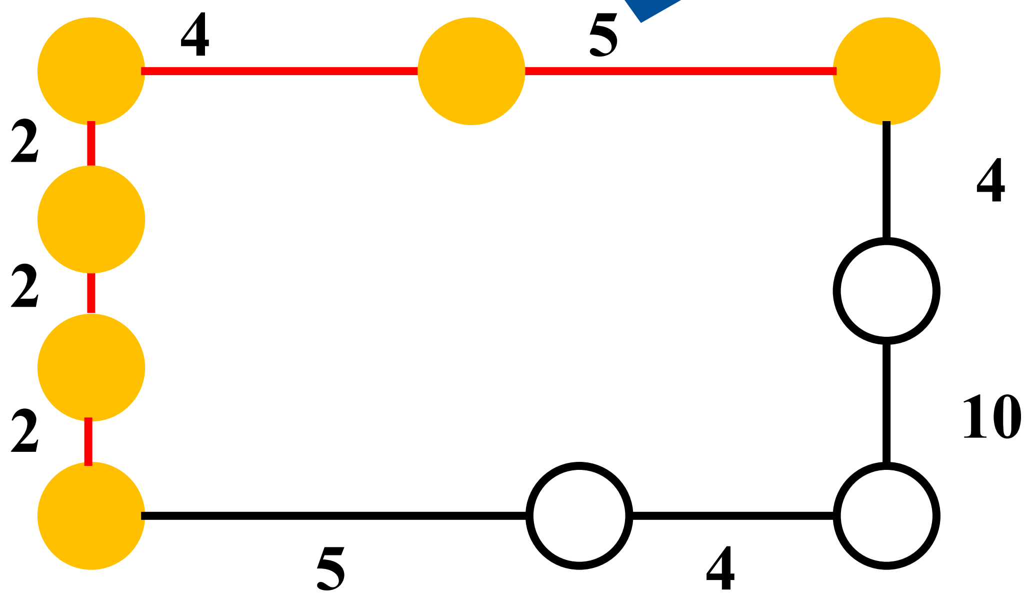


问题背景

- 从知春路到其他站点，如何安排路线？

时间最短

带权图



问题：如何计算**带权图**中源点到所有其他顶点的最短路径？

问题定义

单源最短路径问题（边权为正）

Single Source Shortest Paths Problem with Positive Weights

输入

- 带权图$G=<V,E,W>$，其中$w(u,v)\geq 0$（图中所有边权为正）, $(u,v)\in E$
- 源点编号s

问题定义

单源最短路径问题（边权为正）

Single Source Shortest Paths Problem with Positive Weights

输入

- 带权图$G =< V,E,W >$，其中$w(u,v) \geq 0$（图中所有边权为正），$(u,v) \in E$
- 源点编号s

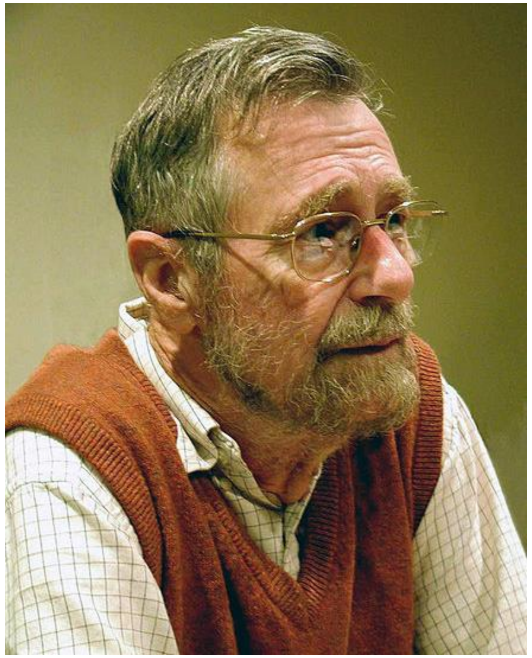
输出

- 源点s到所有其他顶点t的**最短距离**$\delta(s,t)$和**最短路径**$< s, \ldots, t >$

提纲

- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

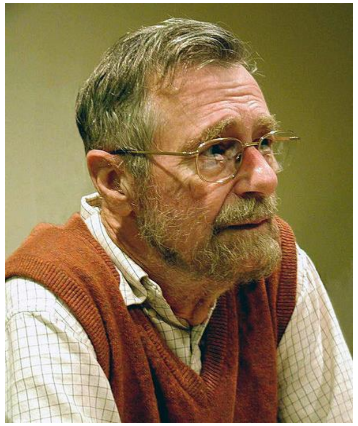
算法思想



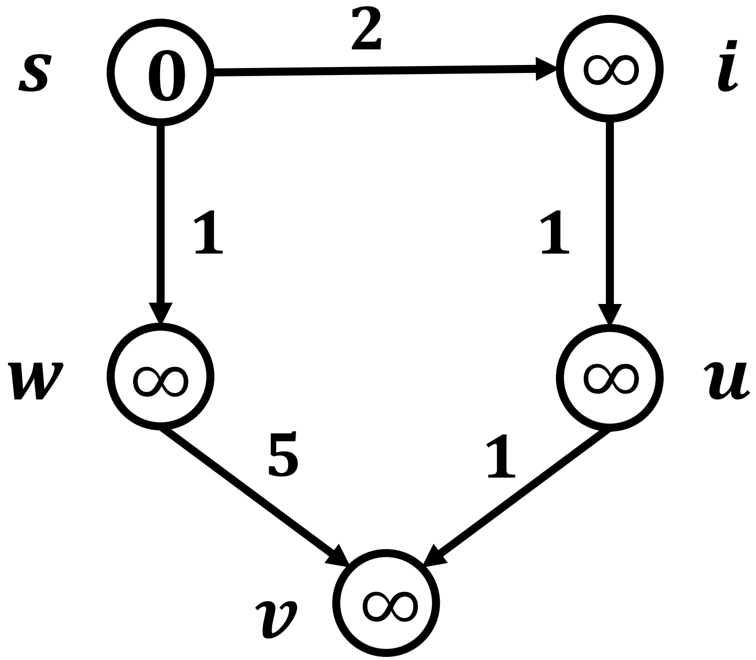
Edsger W. Dijkstra
1972, Netherlands
ALGOL之父
提出单源最短路径Dijkstra算法

算法思想

- 辅助数组
  - $dist$表示距离上界（估计距离）
    - 源点s，$dist[s]=0$；其他顶点u，$dist[u]$初始化为∞



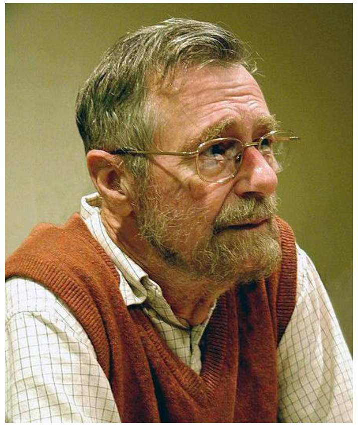
Edsger W. Dijkstra



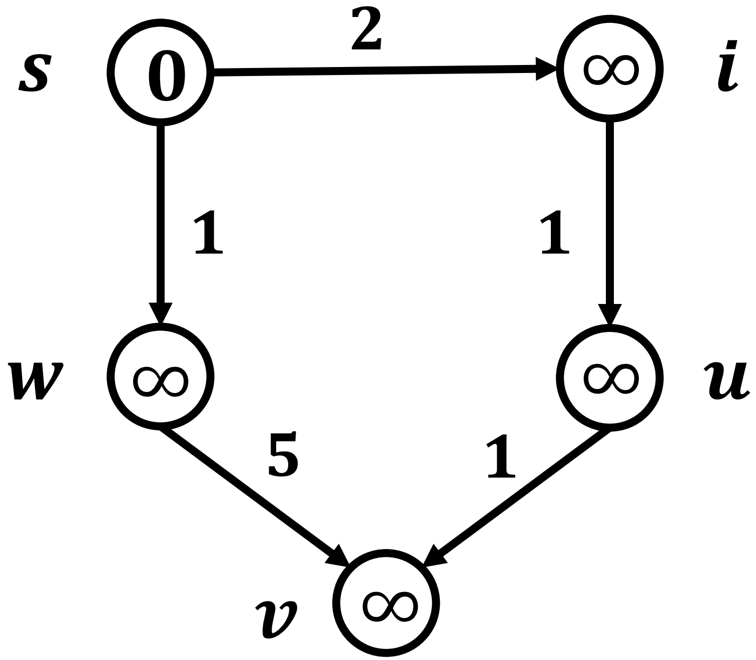
V	s	i	w	u	v
$dist$	0	∞	∞	∞	∞

算法思想

- 辅助数组
  - $dist$表示距离上界（估计距离）
    - 源点s，$dist[s] = 0$；其他顶点u，$dist[u]$初始化为∞
    - $dist[u]$：源点s到顶点u的距离上界，$\delta(s,u) \leq dist[u]$



Edsger W. Dijkstra

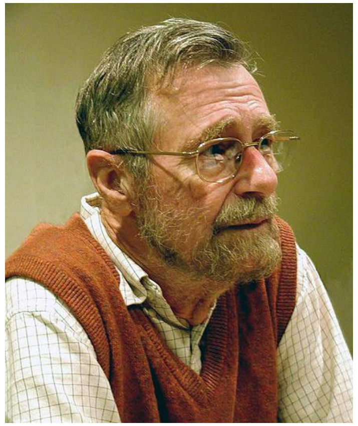


V	s	i	w	u	v
$dist$	0	∞	∞	∞	∞
δ	0	2	1	3	4

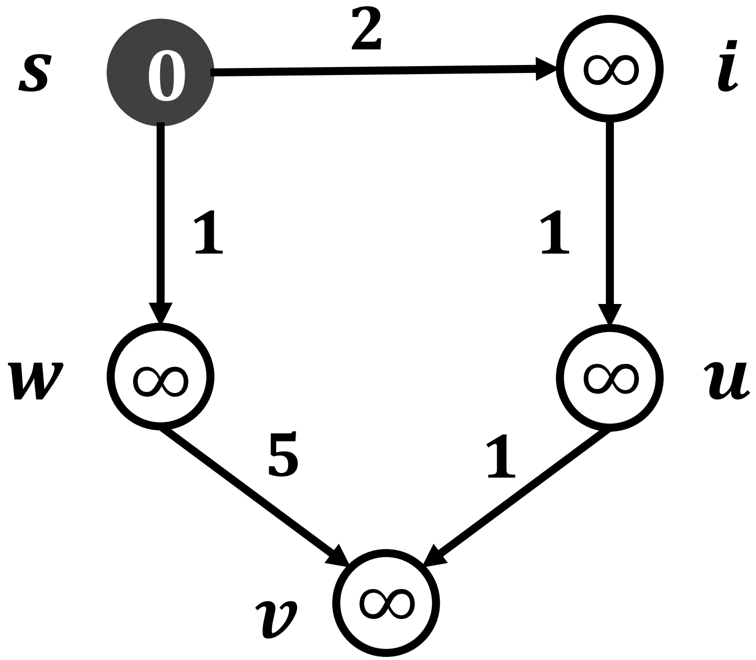
真实最短距离

算法思想

- 辅助数组
  - $dist$表示距离上界（估计距离）
    - 源点s，$dist[s]=0$；其他顶点u，$dist[u]$初始化为∞
    - $dist[u]$：源点s到顶点u的距离上界，$\delta(s,u)\leq dist[u]$
  - $color$表示顶点状态
    - 黑色：到顶点u最短路已被确定



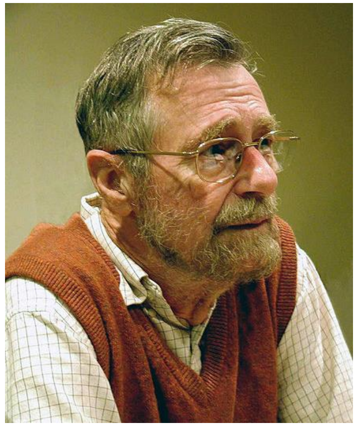
Edsger W. Dijkstra

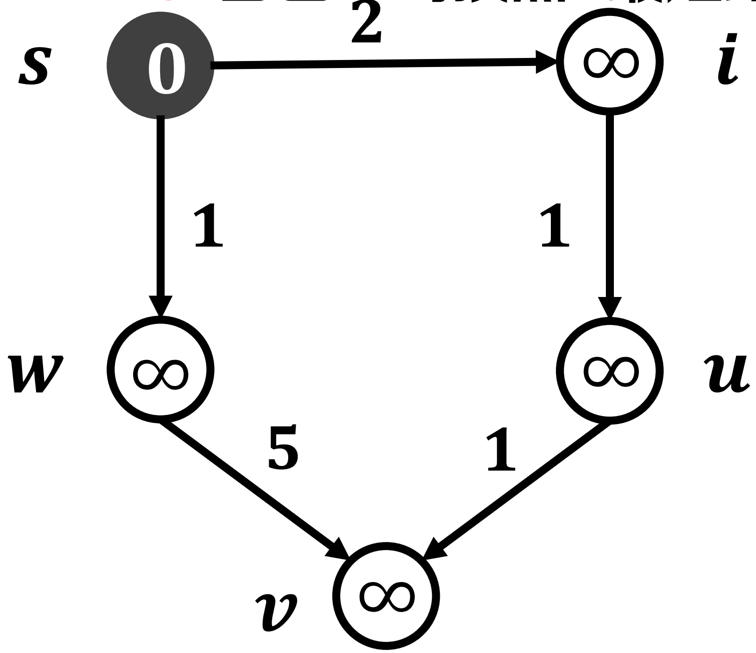


V	s	i	w	u	v
$dist$	0	∞	∞	∞	∞
δ	0	2	1	3	4

算法思想

- 辅助数组
  - $dist$表示距离上界（估计距离）
    - 源点s，$dist[s]=0$；其他顶点u，$dist[u]$初始化为∞
    - $dist[u]$：源点s到顶点u的距离上界，$\delta(s,u)\leq dist[u]$
  - $color$表示顶点状态
    - 黑色：到顶点u最短路已被确定
    - 白色：到顶点u最短路尚未确定

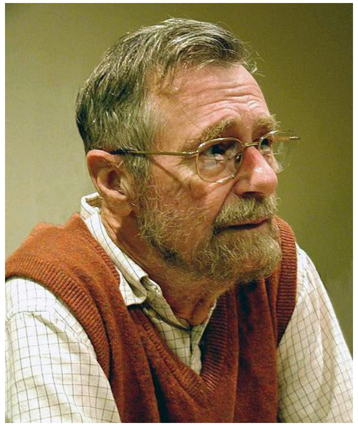




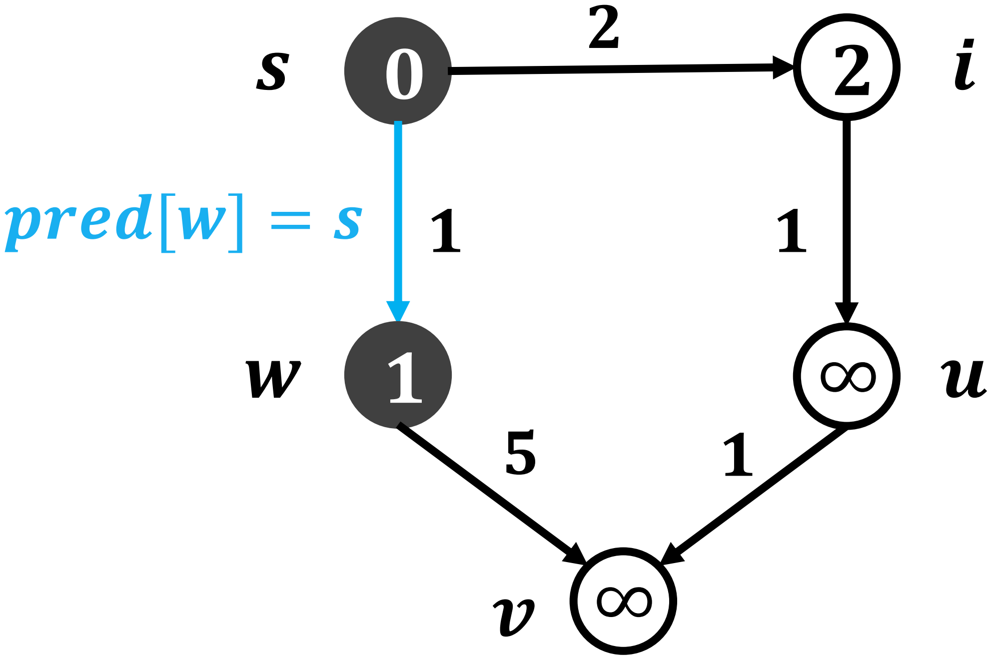
V	s	i	w	u	v
$dist$	0	∞	∞	∞	∞
δ	0	2	1	3	4

算法思想

- 辅助数组
  - $dist$表示距离上界（估计距离）
    - 源点s，$dist[s] = 0$；其他顶点u，$dist[u]$初始化为∞
    - $dist[u]$：源点s到顶点u的距离上界，$\delta(s, u) \leq dist[u]$
  - $color$表示顶点状态
    - 黑色：到顶点u最短路已被确定
    - 白色：到顶点u最短路尚未确定
  - $pred$表示前驱顶点
    - $(pred[u], u)$为最短路径上的边

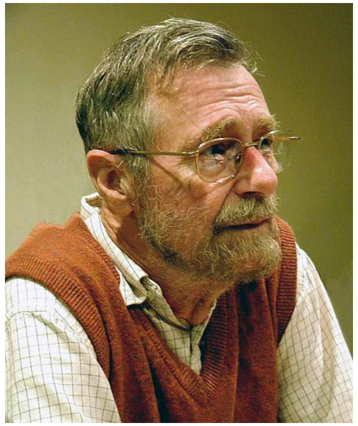


Edsger W. Dijkstra

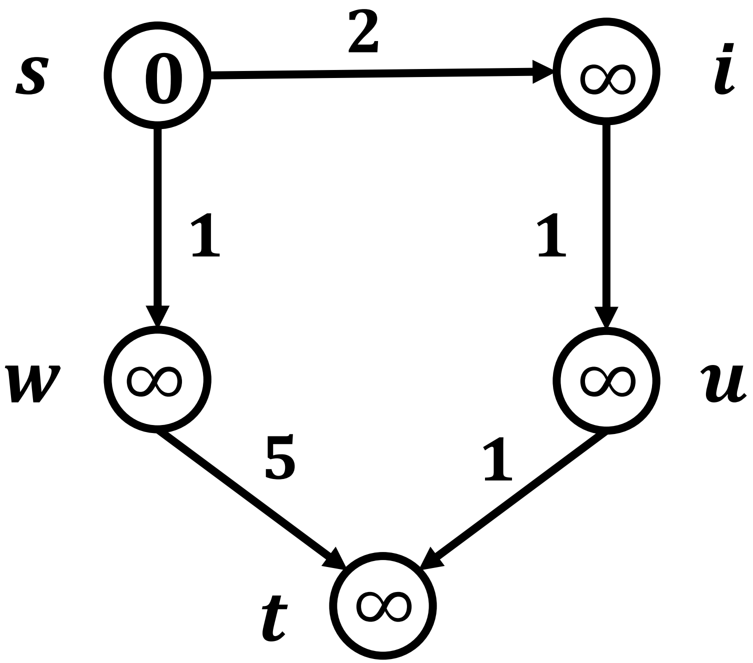


算法思想

- 核心思想
  - 步骤1：新建空的黑色顶点集V_A



Edsger W. Dijkstra

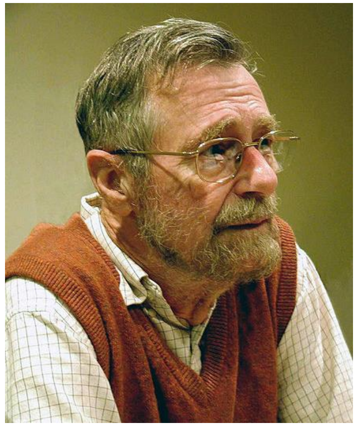


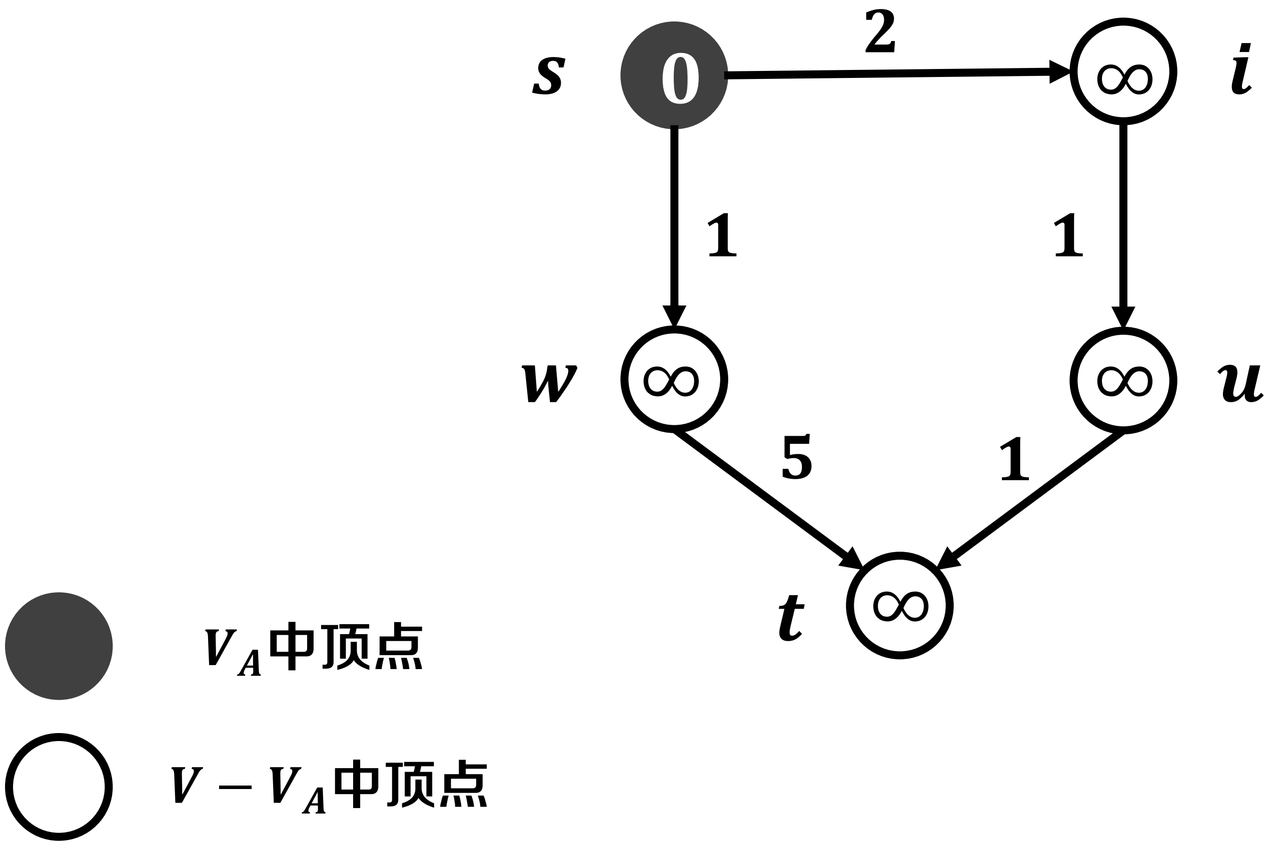
V_A中顶点

$V - V_A$中顶点

算法思想

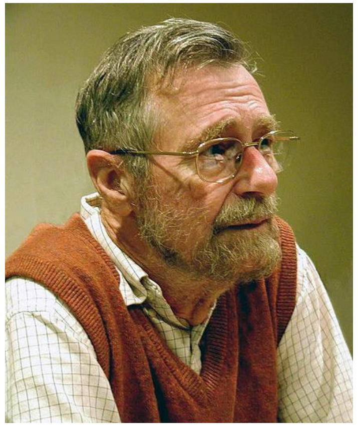
- 核心思想
  - 步骤1：新建空的黑色顶点集V_A
  - 步骤2：选择一个白色顶点变为黑色（到该顶点最短路被确定）


Edsger W. Dijkstra

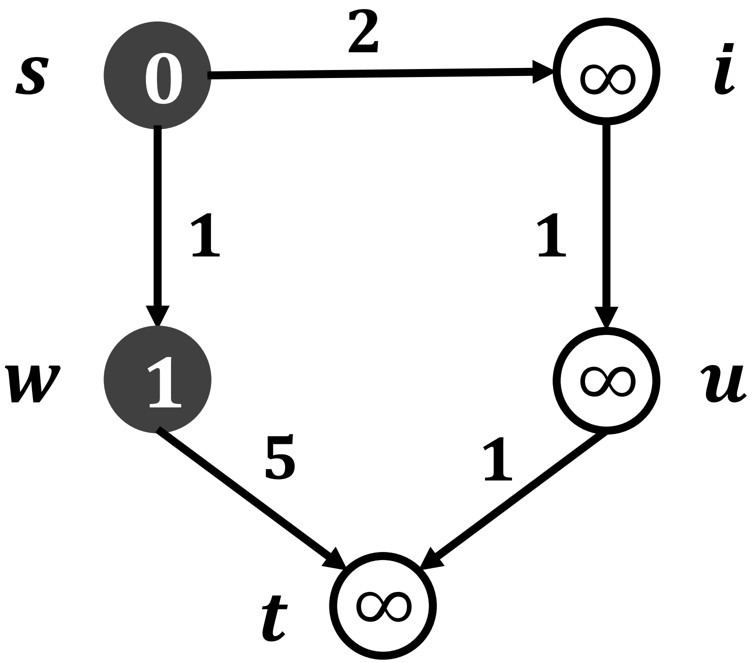


算法思想

- 核心思想
  - 步骤1：新建空的黑色顶点集V_A
  - 步骤2：选择一个白色顶点变为黑色（到该顶点最短路被确定）
  - 步骤3：重复步骤2



Edsger W. Dijkstra

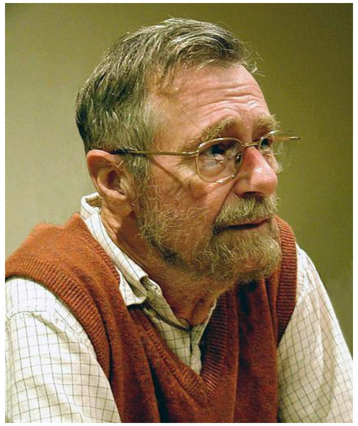


V_A中顶点

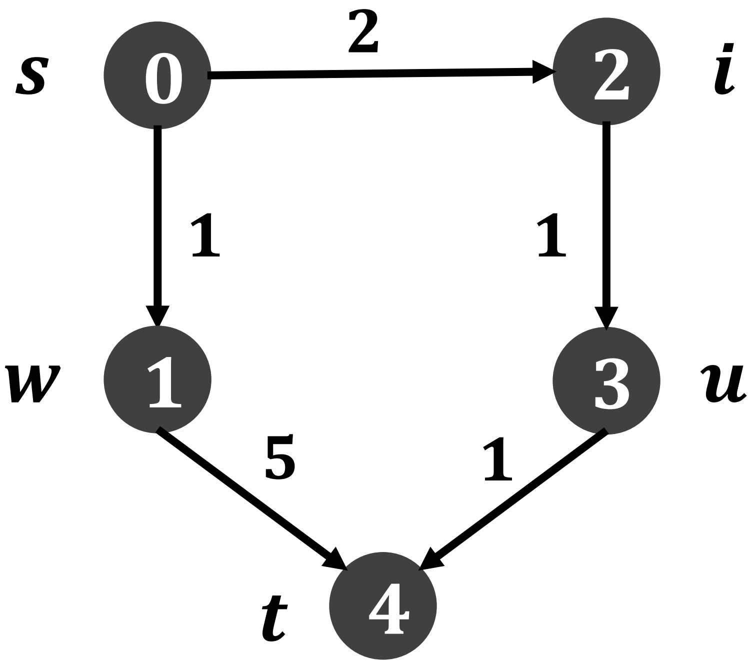
$V - V_A$中顶点

算法思想

- 核心思想
  - 步骤1：新建空的黑色顶点集V_A
  - 步骤2：选择一个白色顶点变为黑色（到该顶点最短路被确定）
  - 步骤3：重复步骤2，直至所有顶点均为黑色



Edsger W. Dijkstra

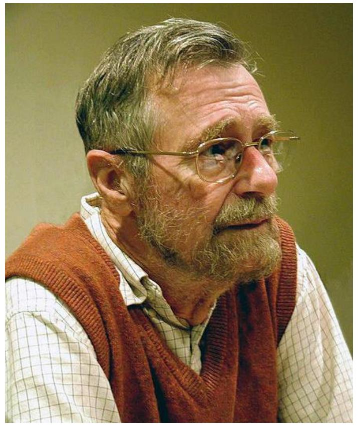


V_A中顶点

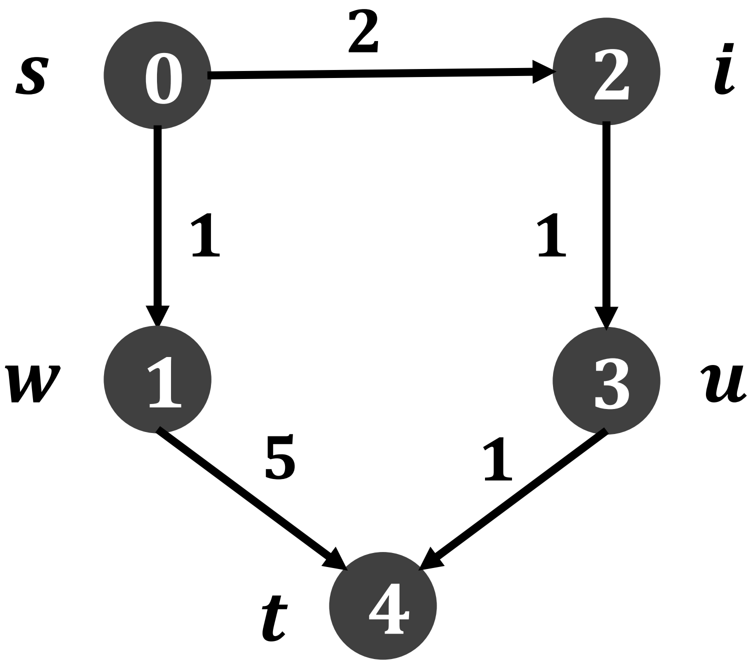
$V - V_A$中顶点

算法思想

- 核心思想
  - 步骤1：新建空的黑色顶点集V_A
  - 步骤2：选择一个白色顶点变为黑色（到该顶点最短路被确定）
  - 步骤3：重复步骤2，直至所有顶点均为黑色



Edsger W. Dijkstra



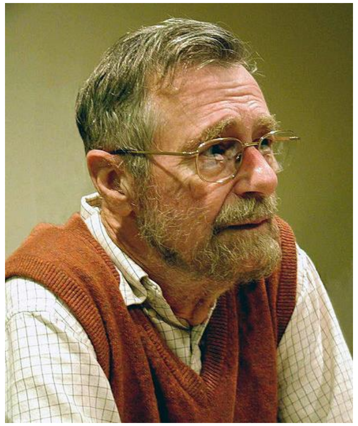
问题：选择哪个白色顶点变为黑色？

● V_A中顶点

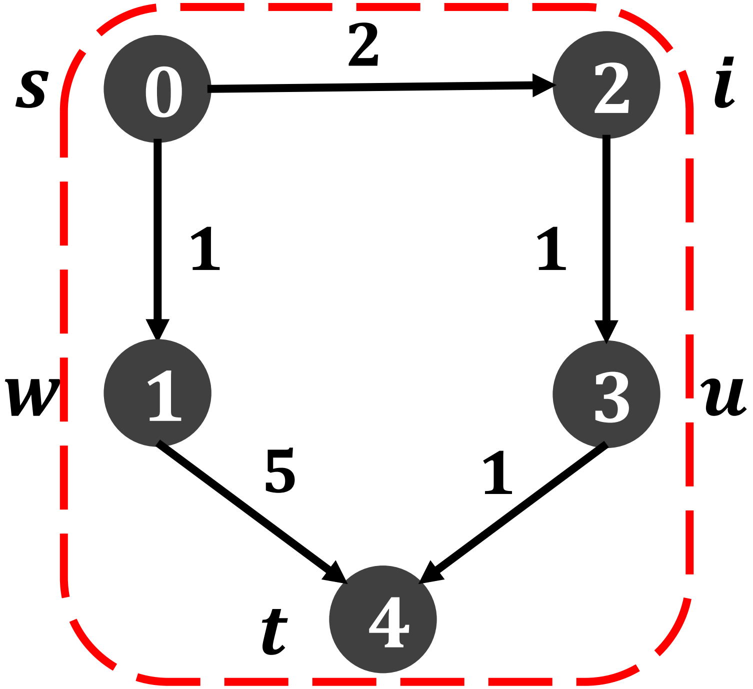
○ $V - V_A$中顶点

算法思想

- 核心思想
  - 步骤1：新建空的黑色顶点集V_A
  - 步骤2：选择一个白色顶点变为黑色（到该顶点最短路被确定）
  - 步骤3：重复步骤2，直至所有顶点均为黑色



Edsger W. Dijkstra



问题：选择哪个白色顶点变为黑色？

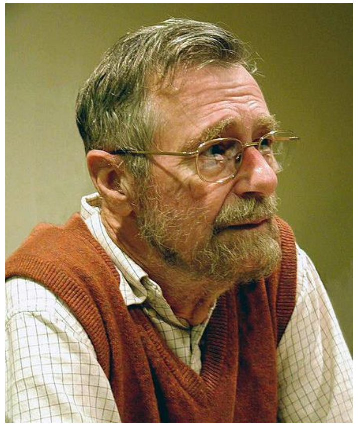
问题：如何更新每顶点的估计距离？

V_A中顶点

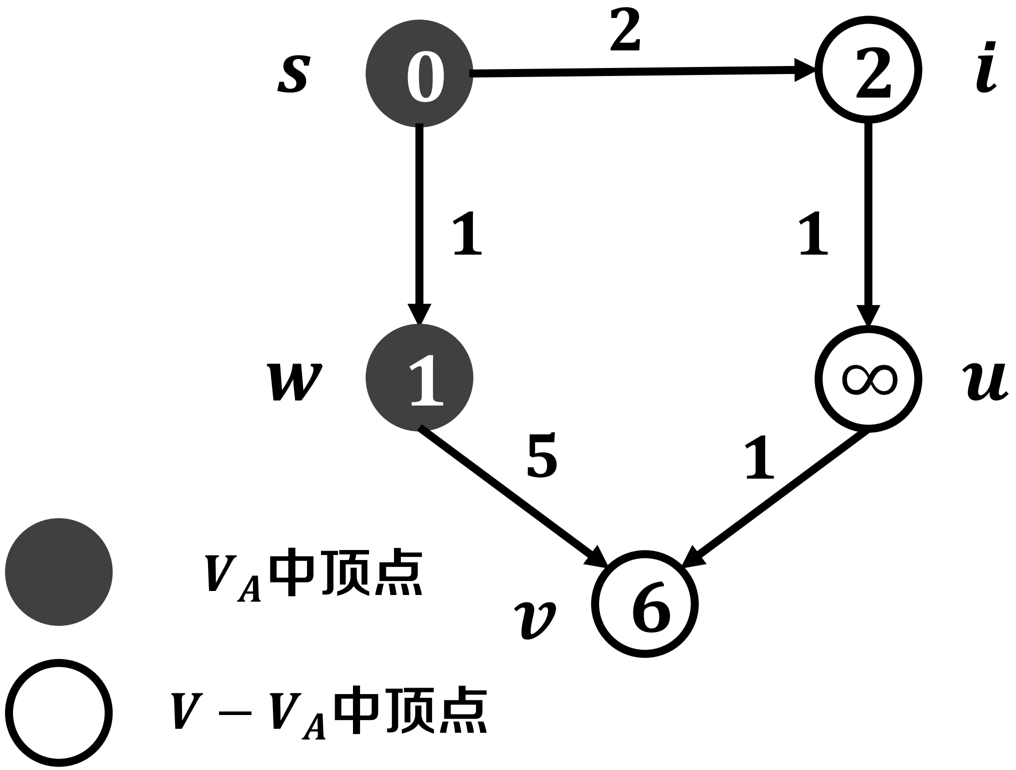
$V-V_A$中顶点

算法思想

- **问题1：选择哪个白色顶点变为黑色？采用贪心策略**

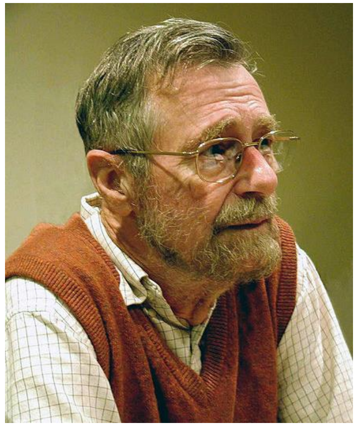


Edsger W. Dijkstra

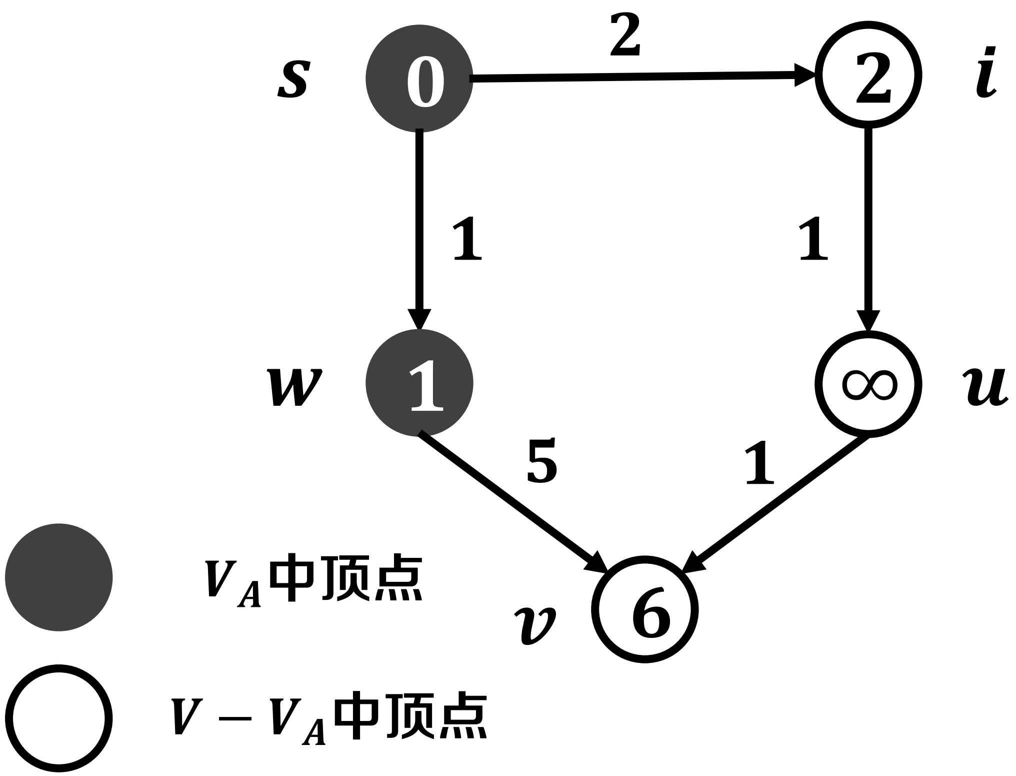


算法思想

- 问题1：选择哪个白色顶点变为黑色？采用贪心策略
  - 对每个白色顶点$y \in V - V_A$，都有一个估计距离$dist[y]$



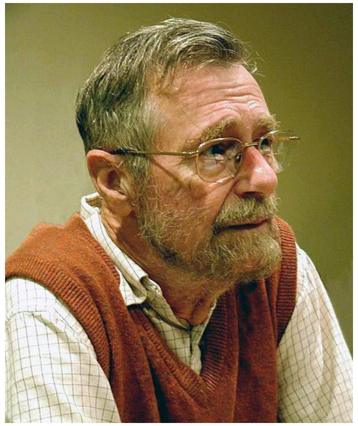
Edsger W. Dijkstra



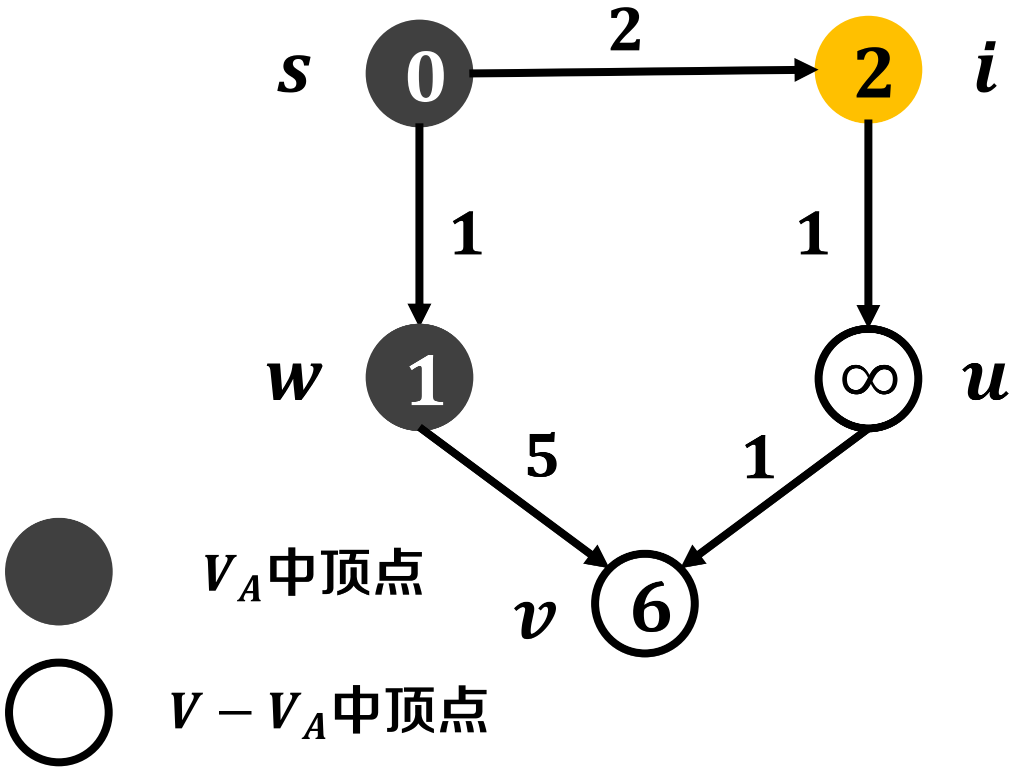
V	s	i	w	u	v
$dist$	0	2	1	∞	6
δ	0	2	1	3	4

算法思想

- 问题1：选择哪个白色顶点变为黑色？采用贪心策略
  - 对每个白色顶点$y \in V - V_A$，都有一个估计距离$dist[y]$
  - 选择估计距离最小的顶点v，$dist[v] \le dist[y]$，$v, y \in V - V_A$



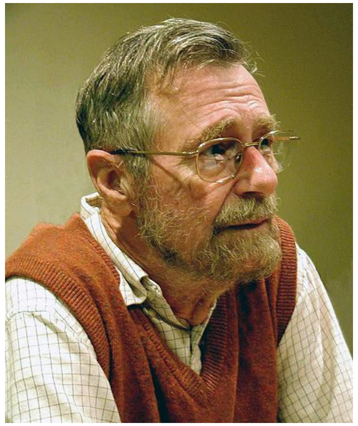
Edsger W. Dijkstra



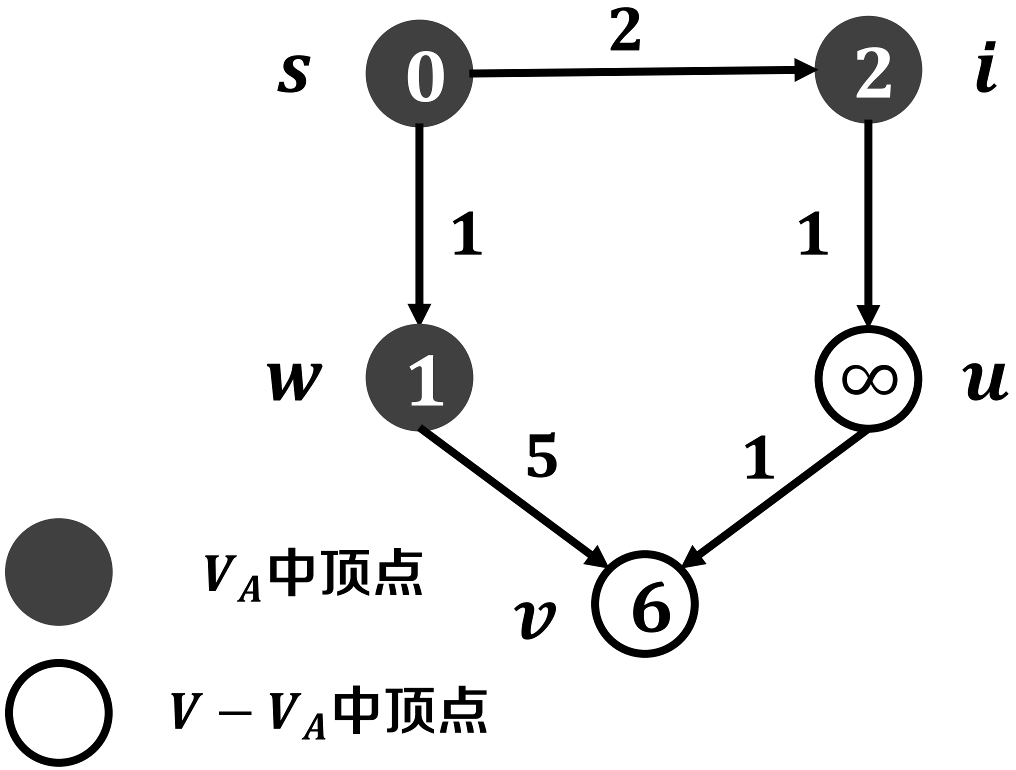
V	s	i	w	u	v
$dist$	0	2	1	∞	6
δ	0	2	1	3	4

算法思想

- 问题1：选择哪个白色顶点变为黑色？采用贪心策略
  - 对每个白色顶点$y \in V - V_A$，都有一个估计距离$dist[y]$
  - 选择估计距离最小的顶点v，$dist[v] \leq dist[y]$，$v, y \in V - V_A$



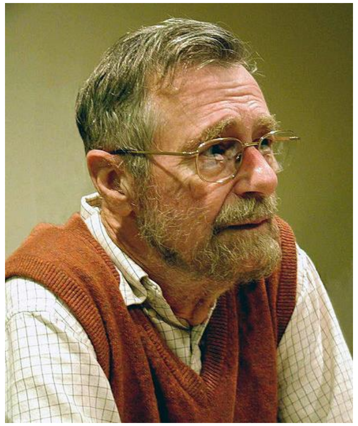
Edsger W. Dijkstra



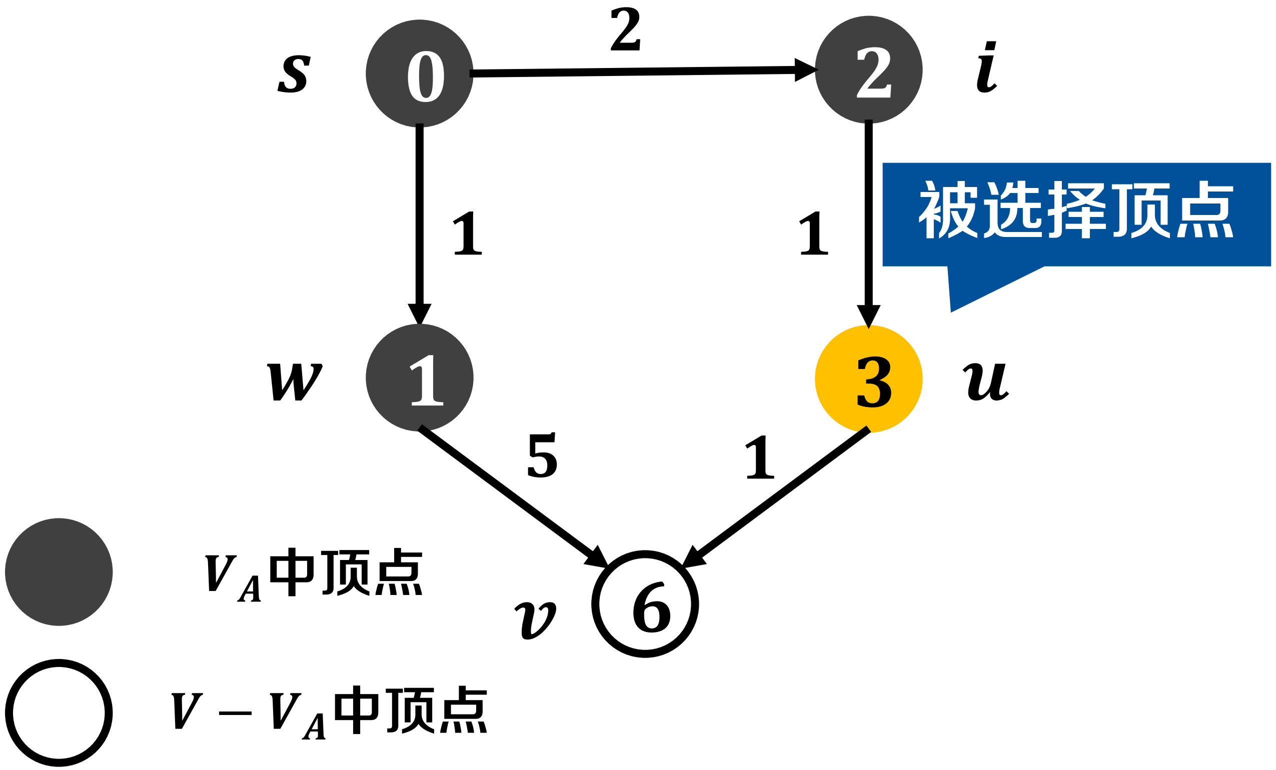
V	s	i	w	u	v
$dist$	0	2	1	∞	6
δ	0	2	1	3	4

算法思想

- 问题2：如何更新每顶点的估计距离？

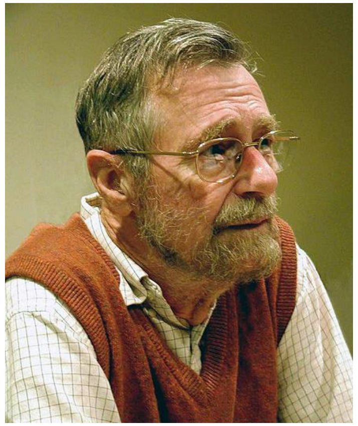


Edsger W. Dijkstra

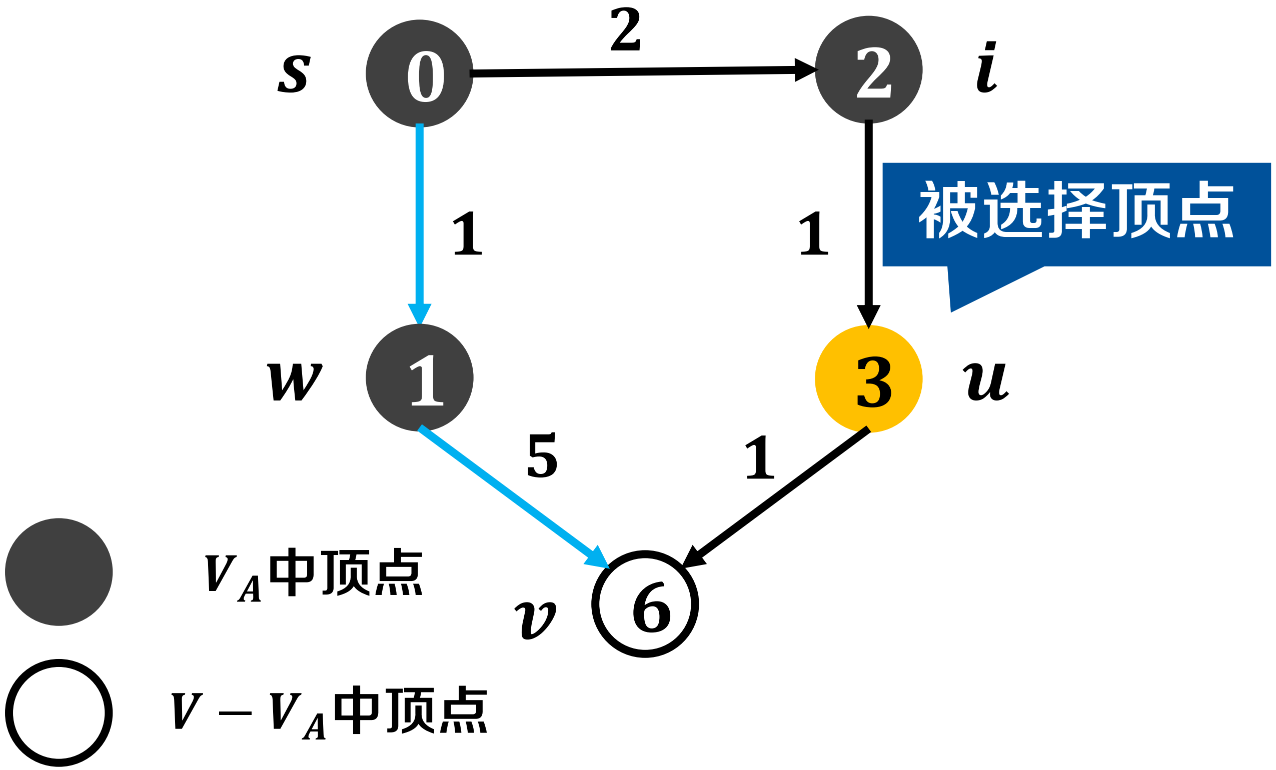


算法思想

- 问题2：如何更新每顶点的估计距离？
  - 当前到顶点v的最短路径：$<s, w, v>$，距离为$dist[v]$

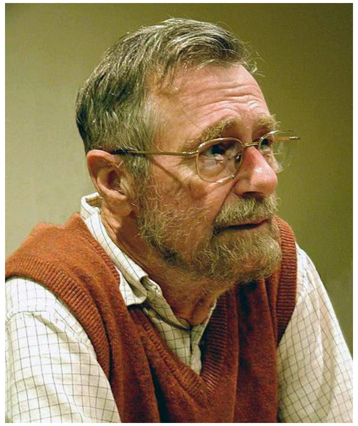


Edsger W. Dijkstra

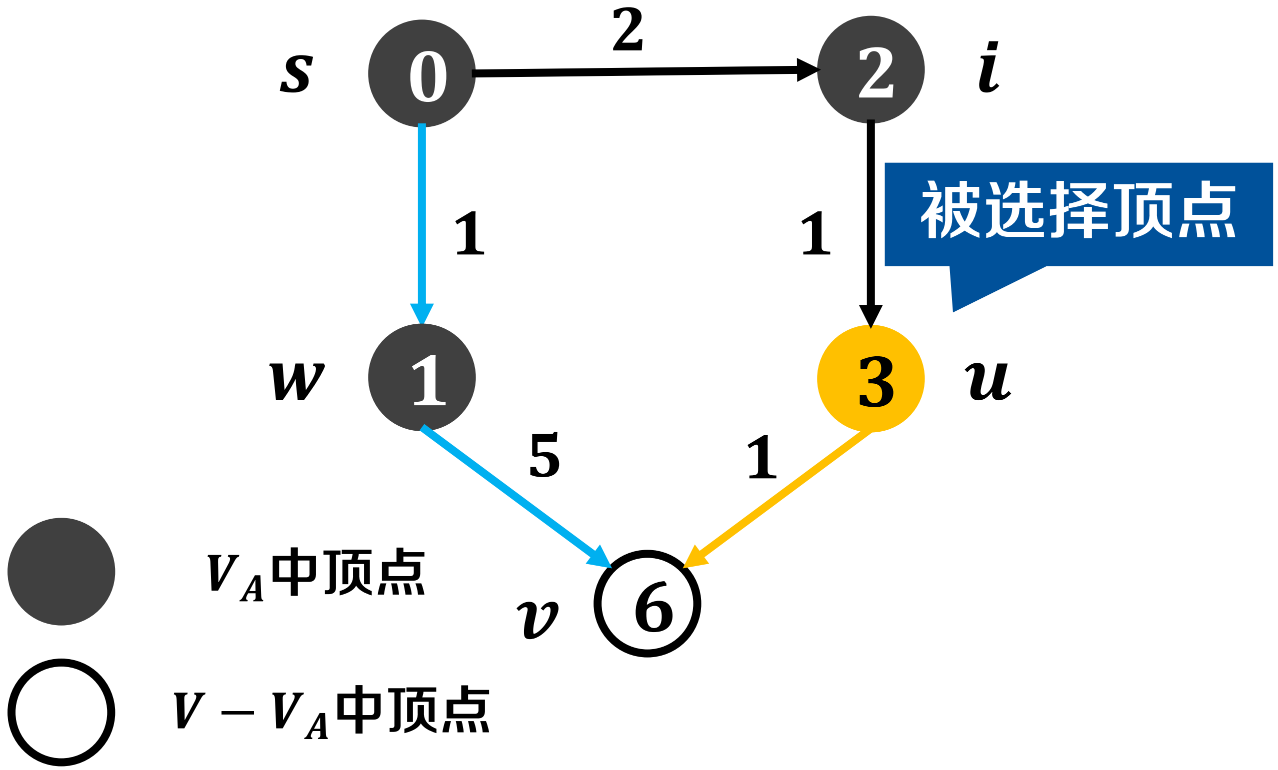


算法思想

- 问题2：如何更新每顶点的估计距离？
  - 当前到顶点v的最短路径：$< s, w, v >$，距离为$dist[v]$
  - 通过顶点u的新路径：$< s, \dots, u, v >$，距离为$dist[u] + w(u, v)$

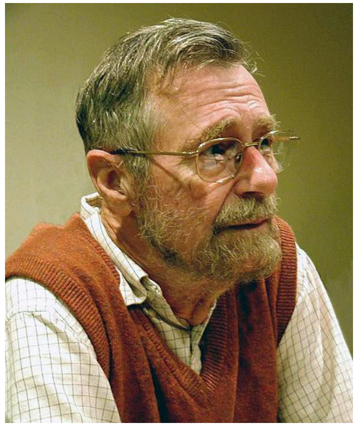


Edsger W. Dijkstra

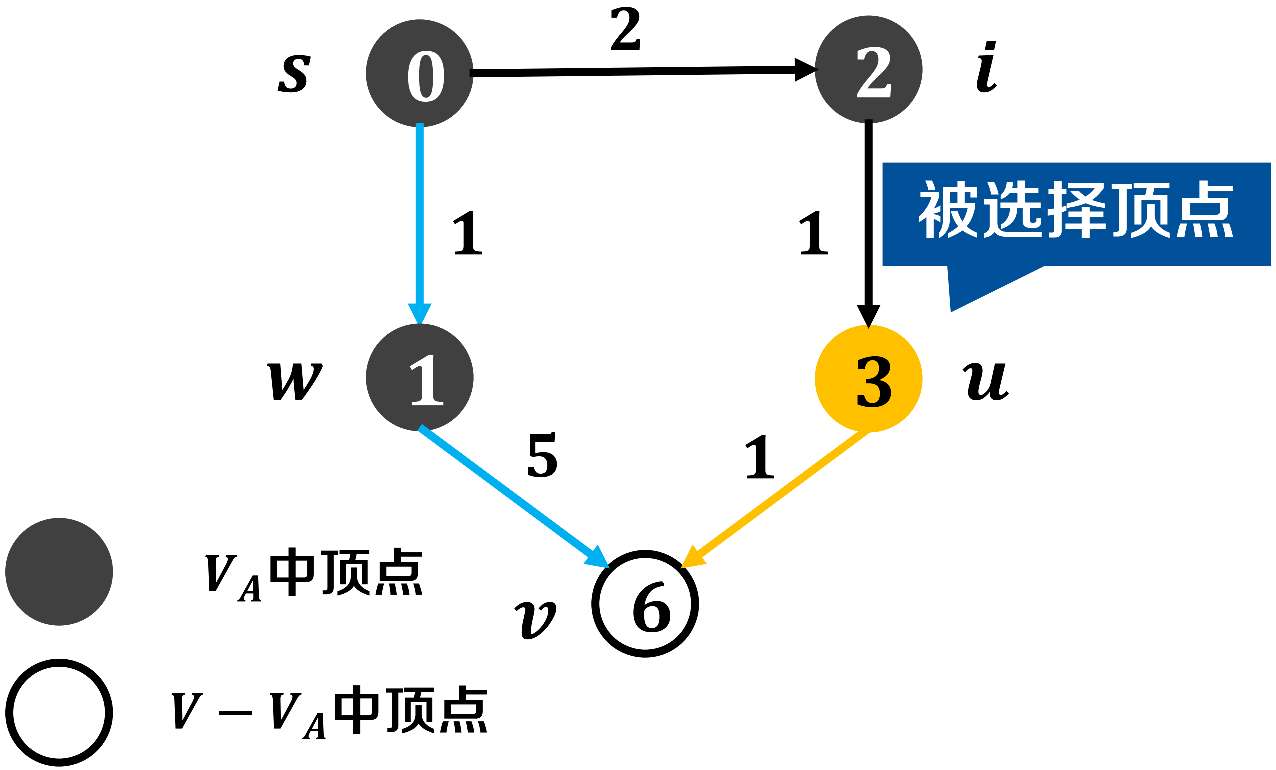


算法思想

- 问题2：如何更新每顶点的估计距离？
  - 当前到顶点v的最短路径：$< s, w, v >$，距离为$dist[v]$
  - 通过顶点u的新路径：$< s, \dots, u, v >$，距离为$dist[u] + w(u, v)$
  - 如果新路径更短$(dist[u] + w(u, v) < dist[v])$
    - 更新$dist[v]$：$dist[v] = dist[u] + w(u, v)$



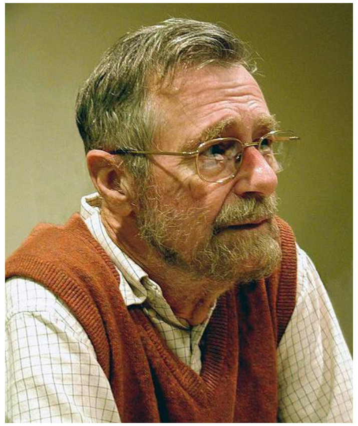
Edsger W. Dijkstra



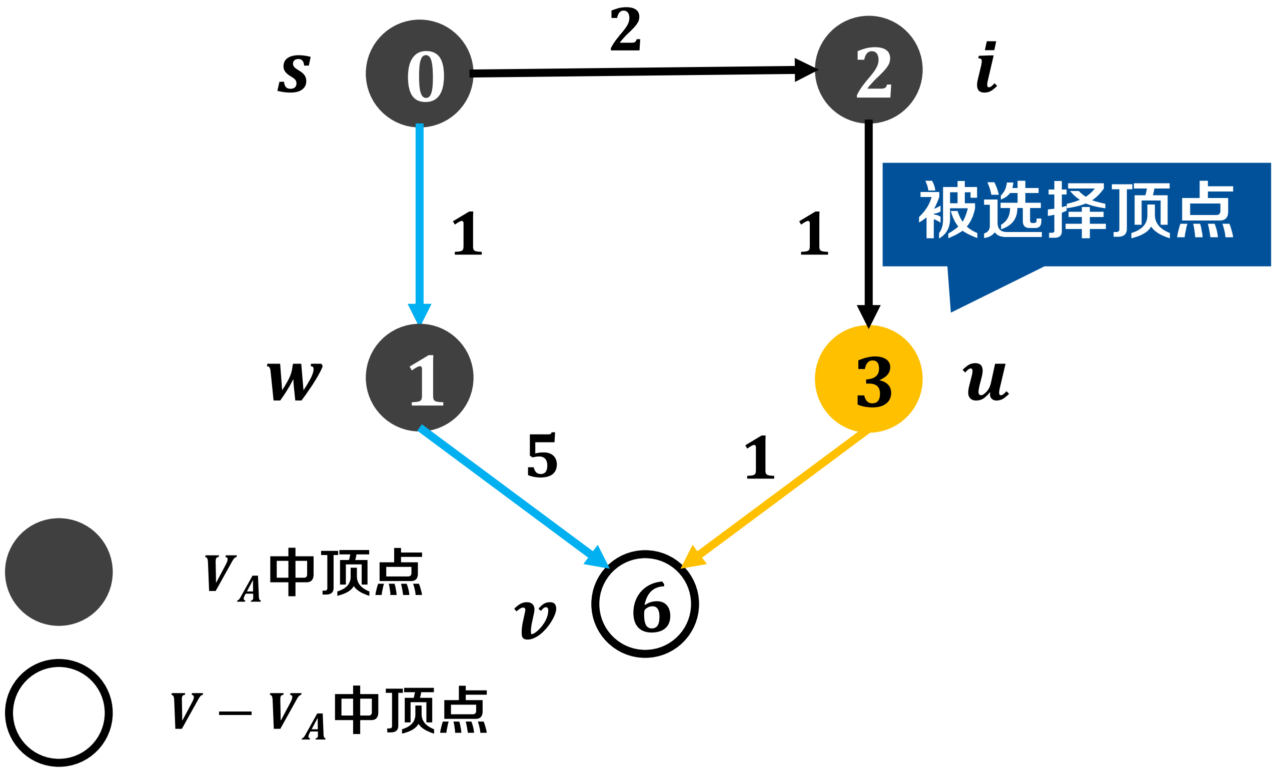
算法思想

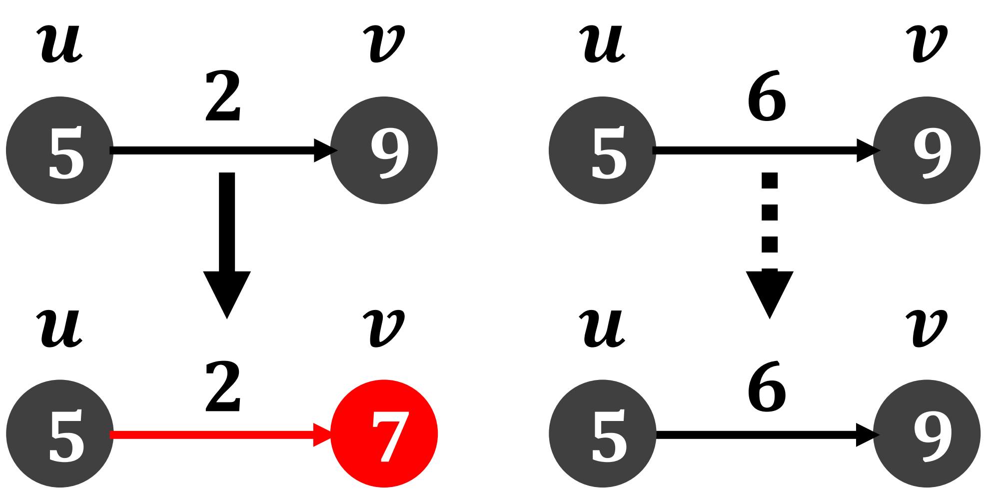
- 问题2：如何更新每顶点的估计距离？
  - 当前到顶点v的最短路径：$< s, w, v >$，距离为$dist[v]$
  - 通过顶点u的新路径：$< s, \ldots, u, v >$，距离为$dist[u] + w(u, v)$
  - 如果新路径更短$(dist[u] + w(u, v) < dist[v])$
    - 更新$dist[v]$：$dist[v] = dist[u] + w(u, v)$

松弛操作



Edsger W. Dijkstra



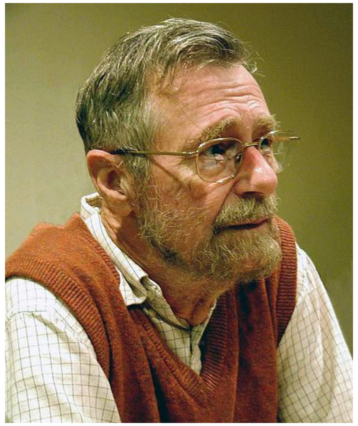


松弛成功　　松弛失败

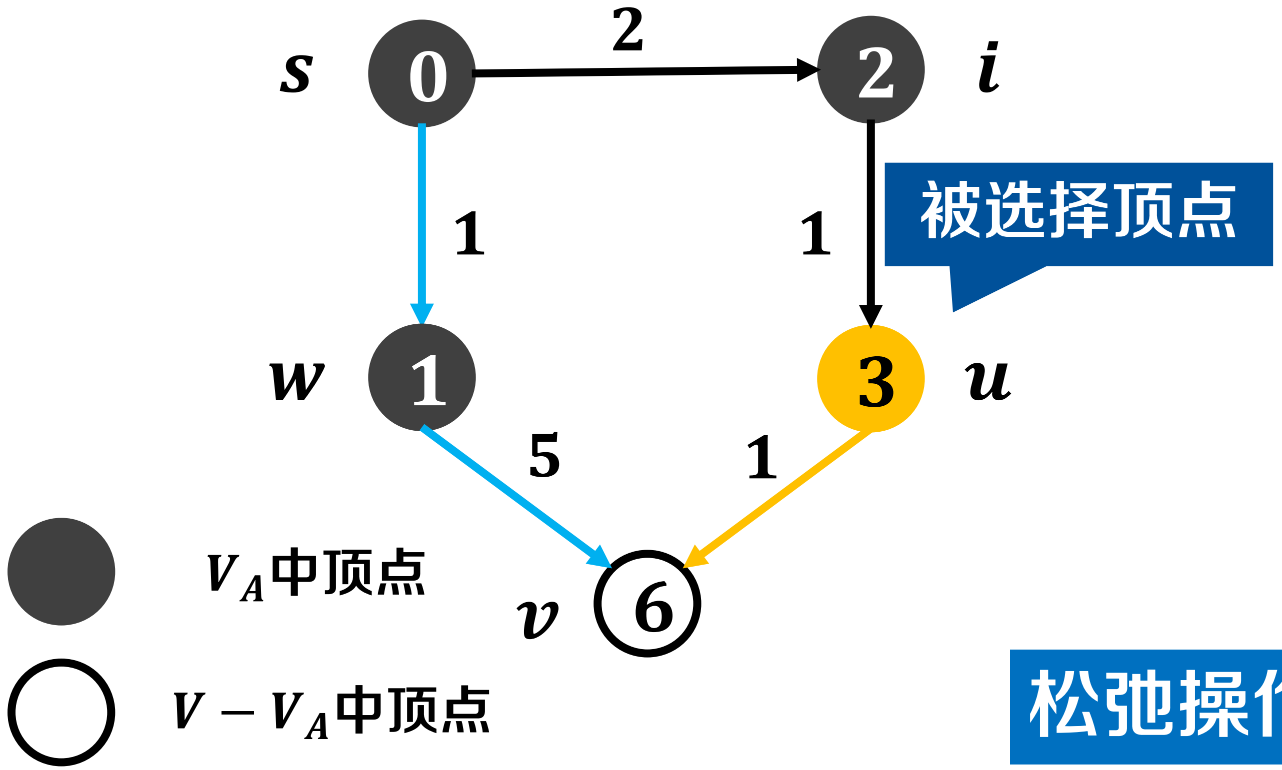
算法思想

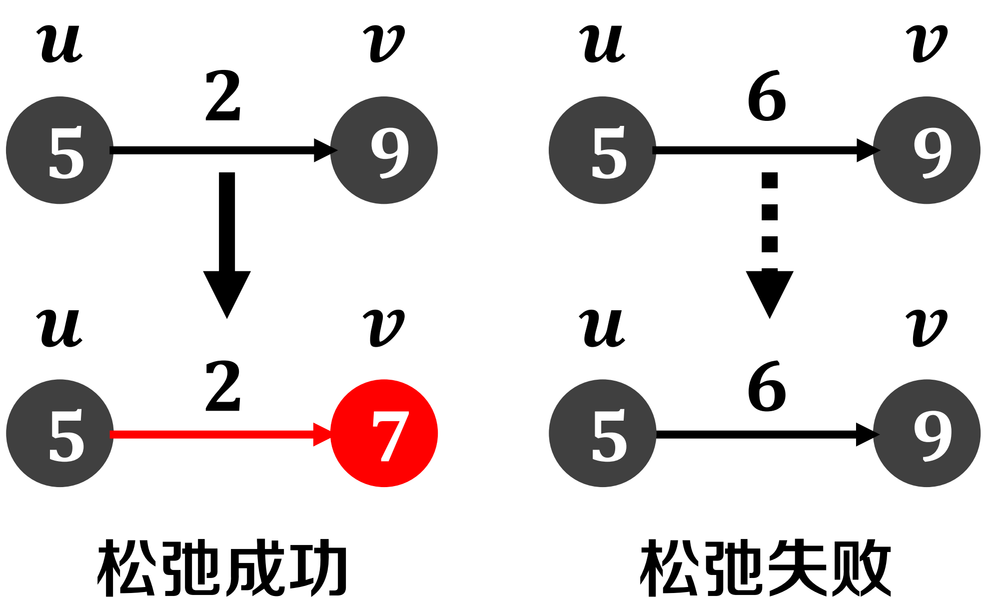
- 问题2：如何更新每顶点的估计距离？
  - 当前到顶点v的最短路径：$<s, w, v>$，距离为$dist[v]$
  - 通过顶点u的新路径：$<s, \dots, u, v>$，距离为$dist[u] + w(u, v)$
  - 如果新路径更短$(dist[u] + w(u, v) < dist[v])$
    - 更新$dist[v]$：$dist[v] = dist[u] + w(u, v)$

松弛操作



Edsger W. Dijkstra





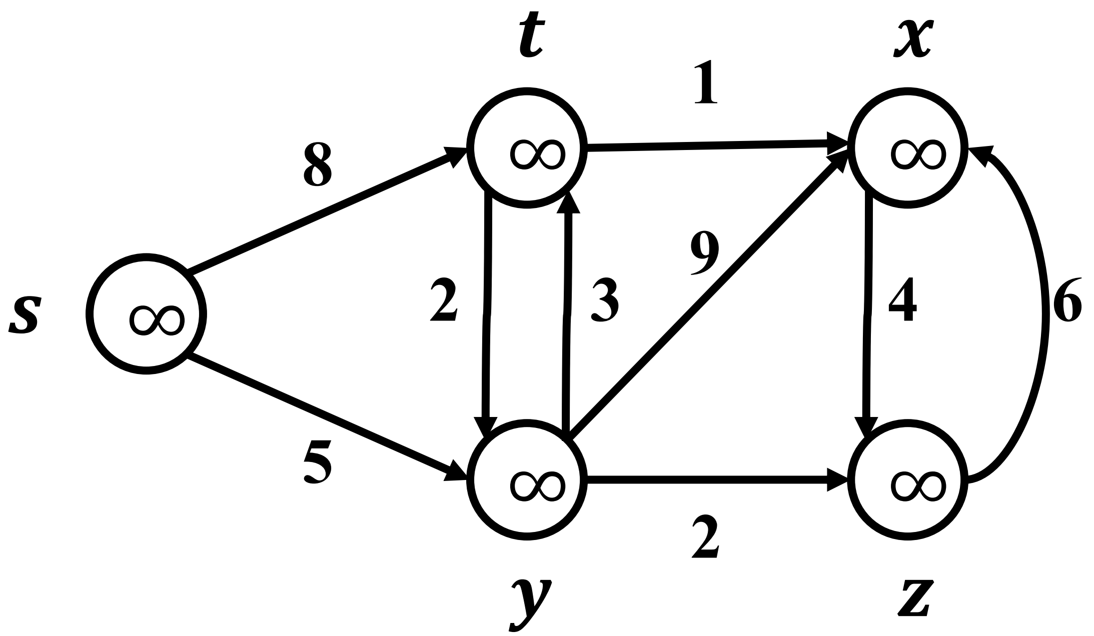
松弛操作的效果：$dist[v] \le dist[u] + w(u, v)$

提纲

- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

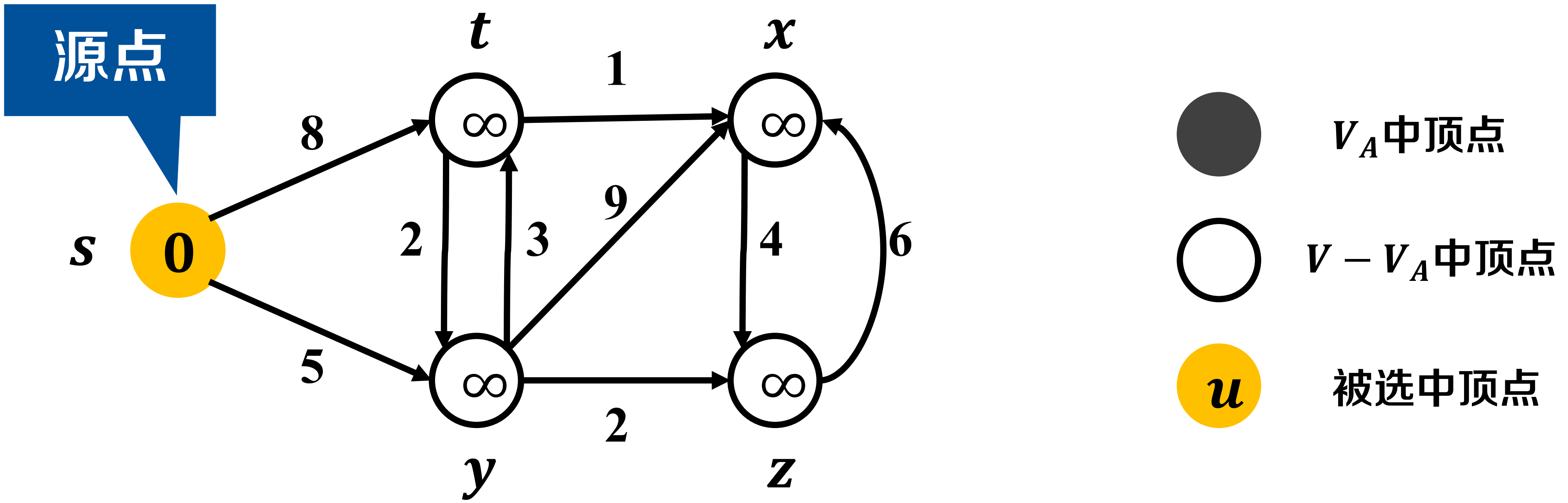
算法实例

V	s	t	x	y	z
$color$	W	W	W	W	W
$pred$	N	N	N	N	N
$dist$	∞	∞	∞	∞	∞



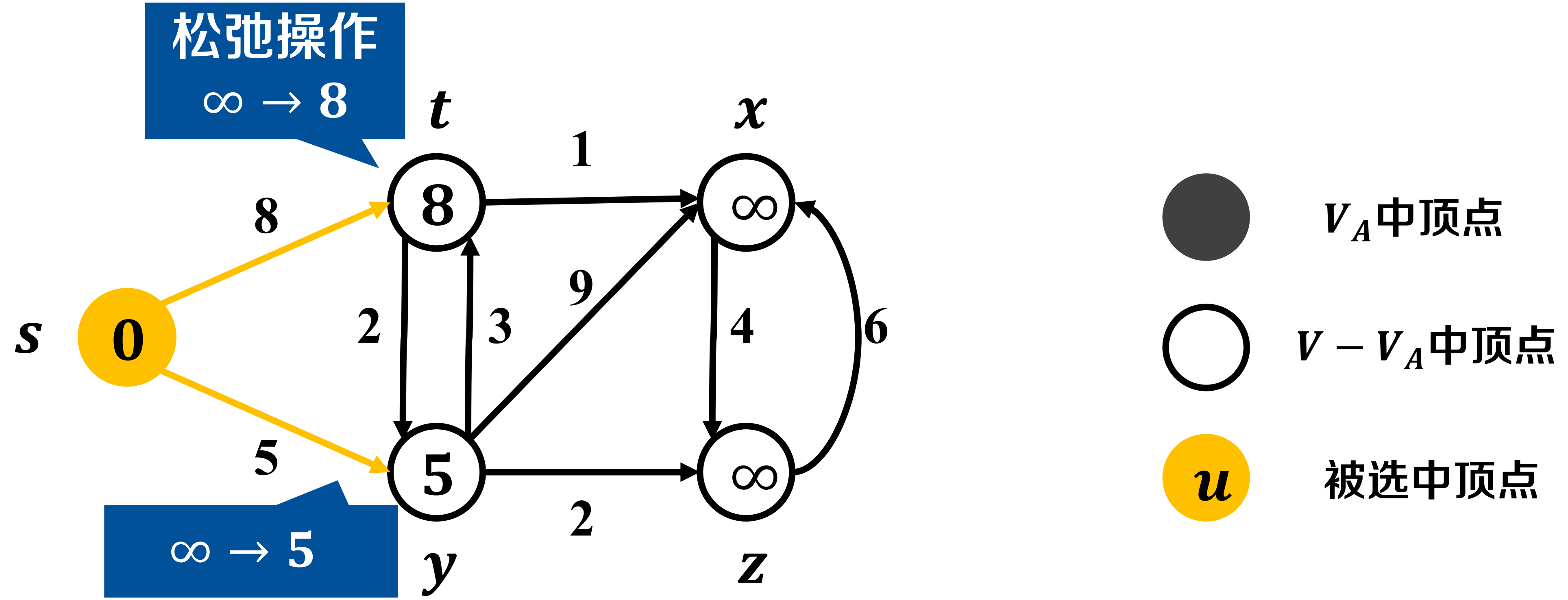
算法实例

V	s	t	x	y	z
color	W	W	W	W	W
pred	N	N	N	N	N
dist	0	∞	∞	∞	∞



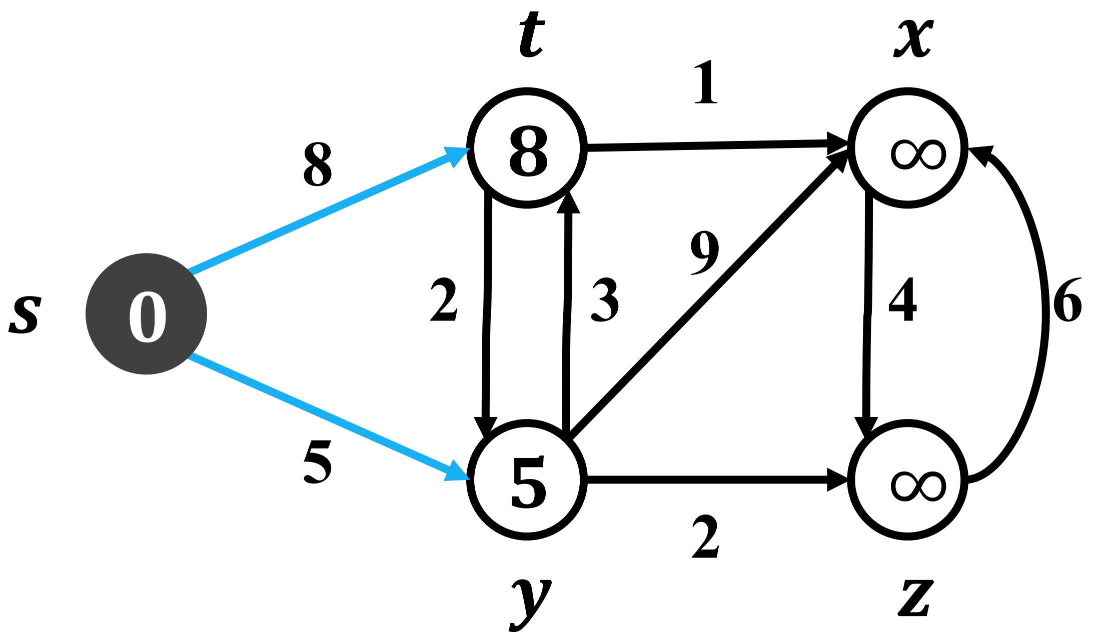
算法实例

V	s	t	x	y	z
$color$	W	W	W	W	W
$pred$	N	s	N	s	N
$dist$	0	8	∞	5	∞



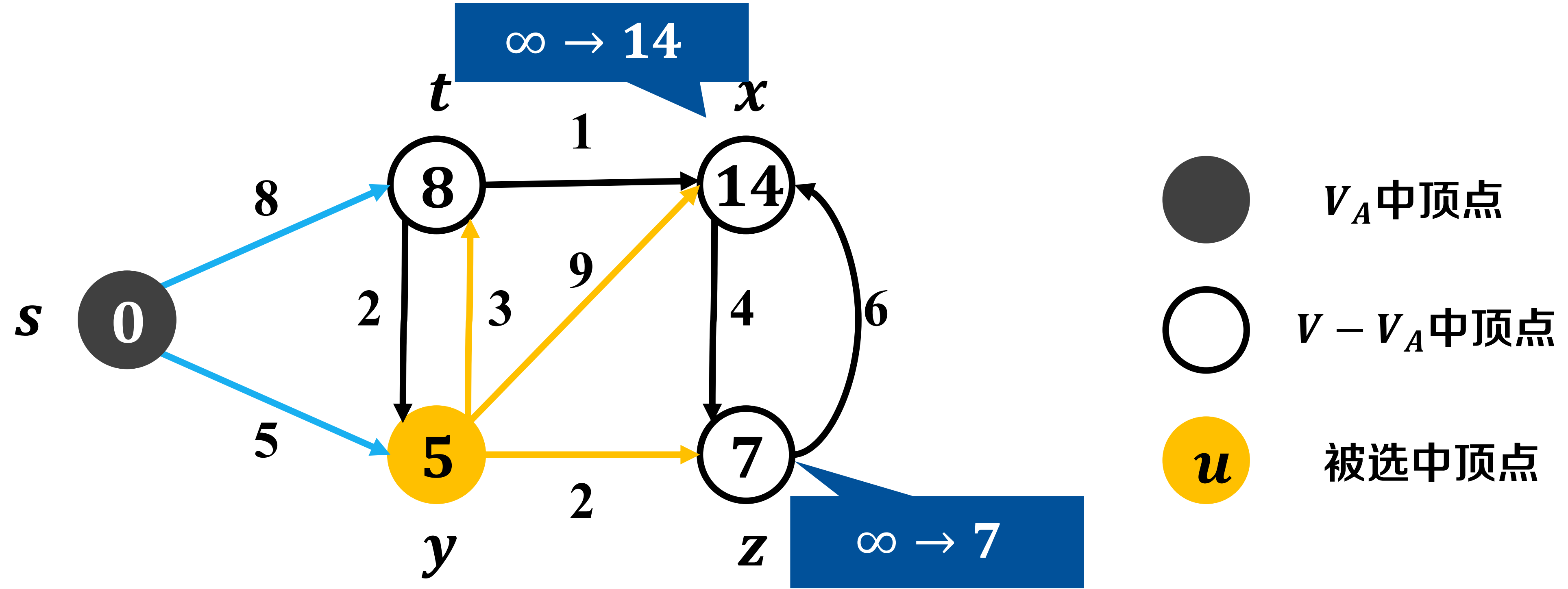
算法实例

V	s	t	x	y	z
color	B	W	W	W	W
pred	N	s	N	s	N
dist	0	8	∞	5	∞



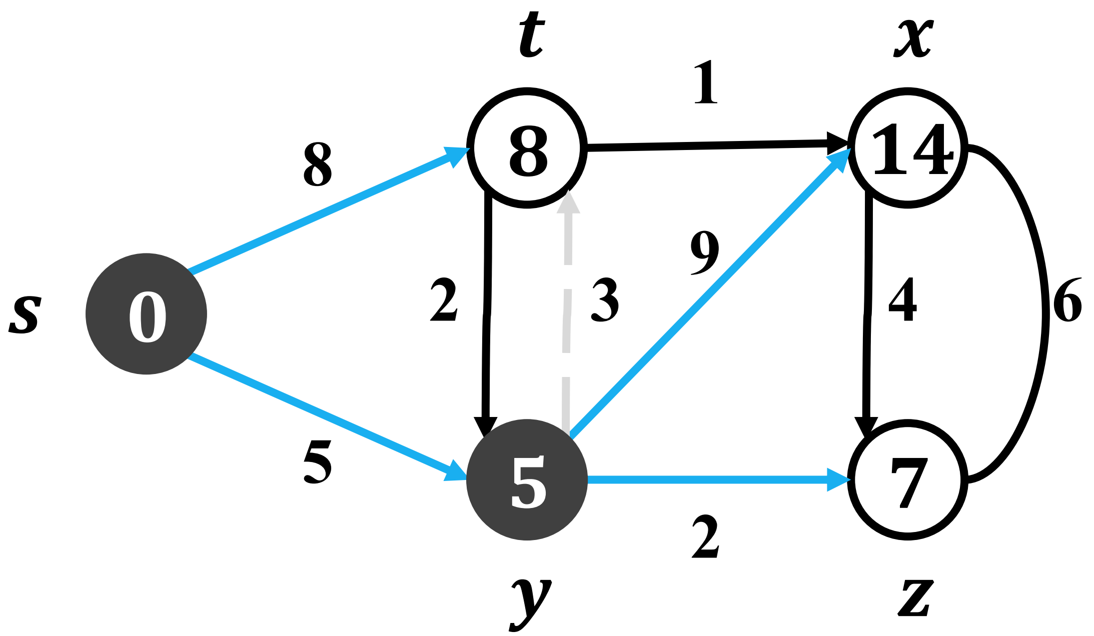
算法实例

V	s	t	x	y	z
$color$	B	W	W	W	W
$pred$	N	s	y	s	y
$dist$	0	8	14	5	7



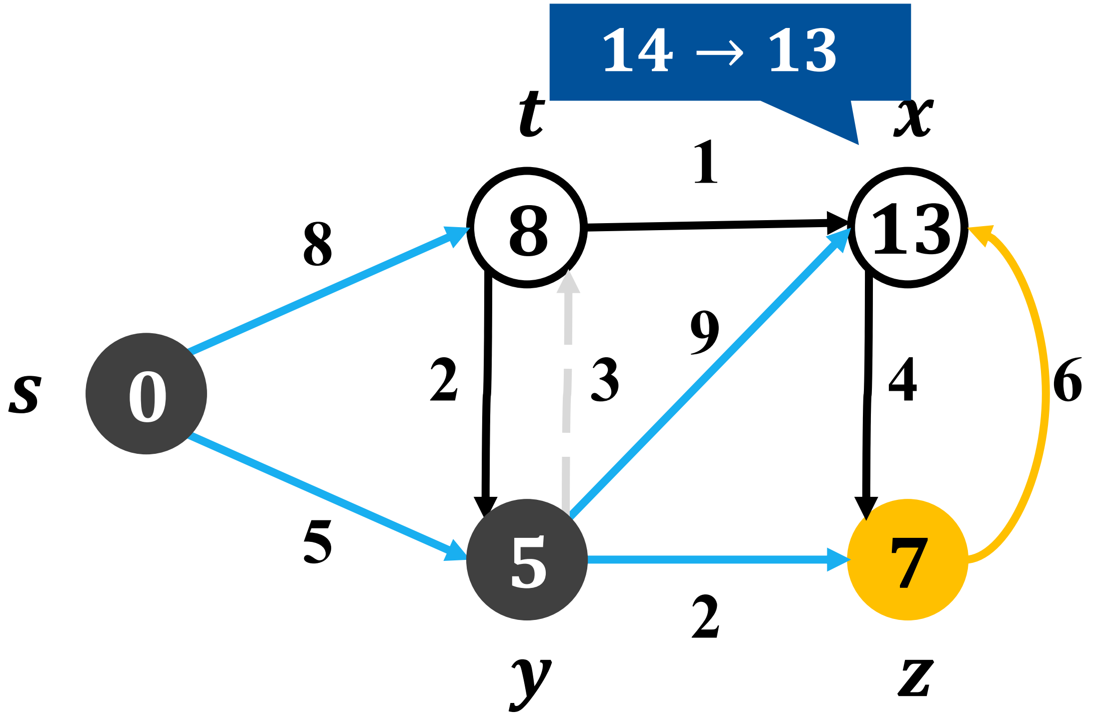
算法实例

V	s	t	x	y	z
color	B	W	W	B	W
pred	N	s	y	s	y
dist	0	8	14	5	7



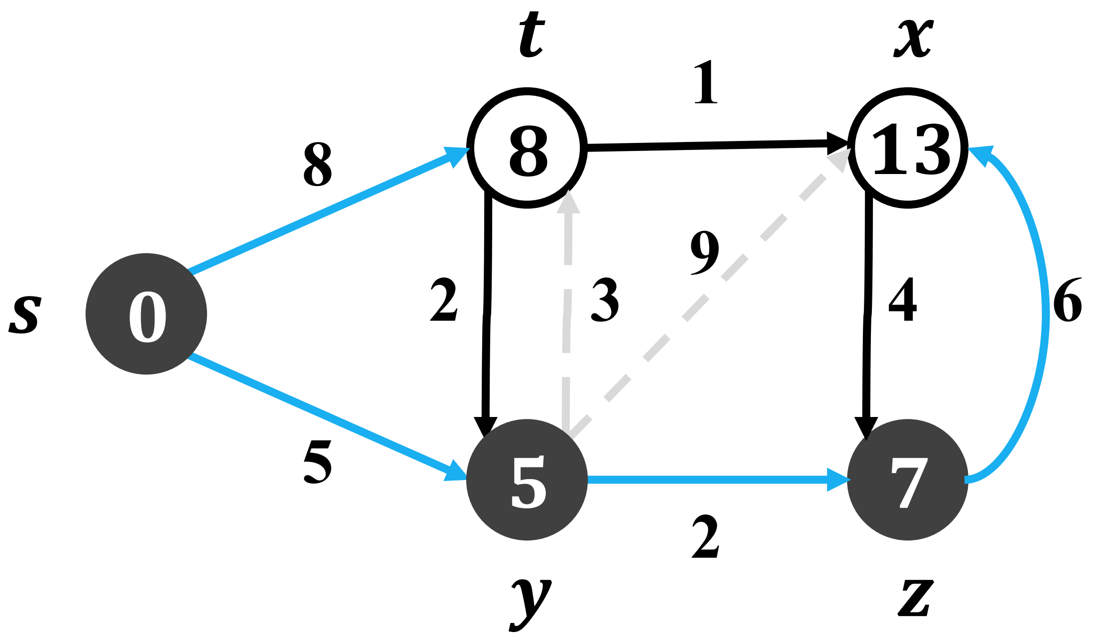
算法实例

V	s	t	x	y	z
color	B	W	W	B	W
pred	N	s	z	s	y
dist	0	8	13	5	7



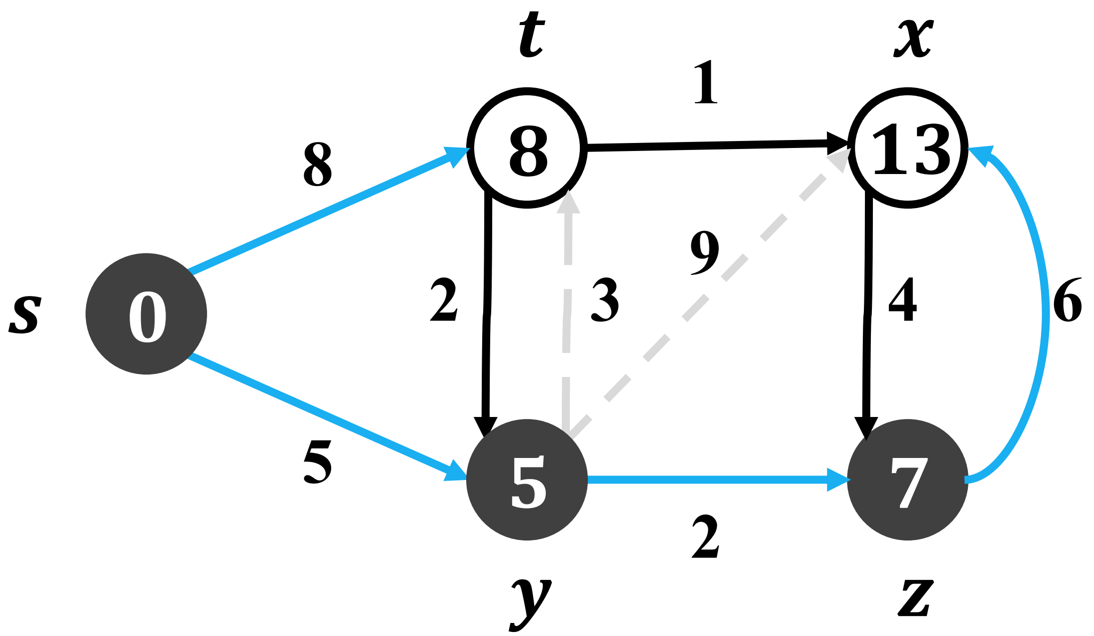
算法实例

V	s	t	x	y	z
color	B	W	W	B	B
pred	N	s	z	s	y
dist	0	8	13	5	7



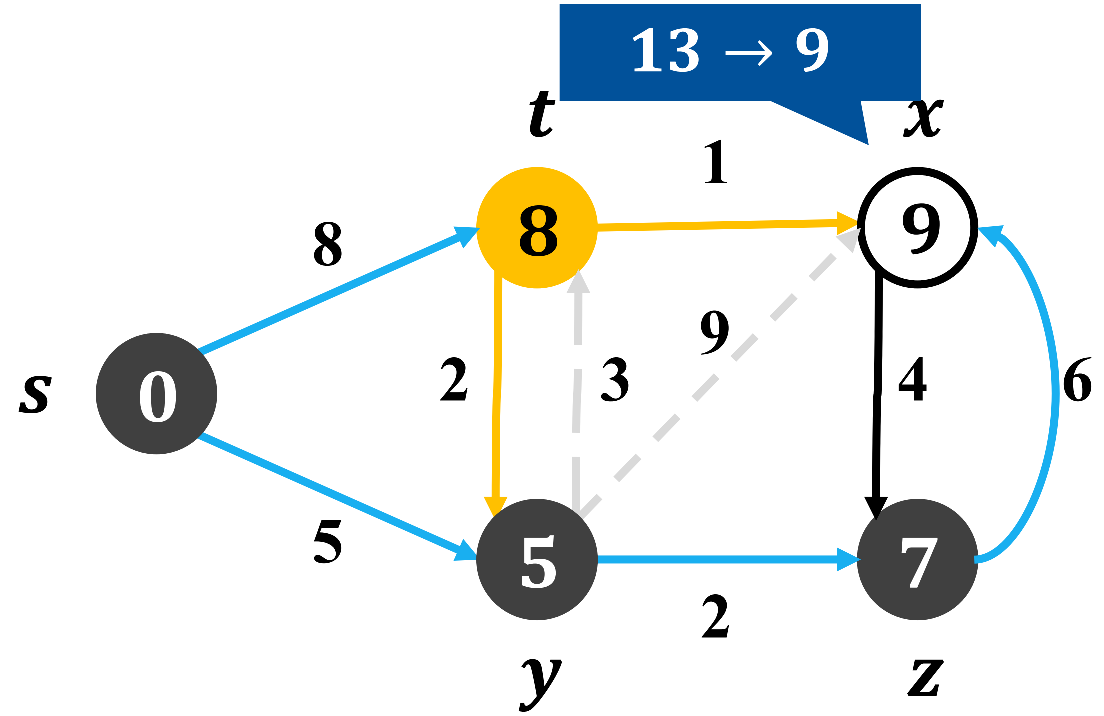
算法实例

V	s	t	x	y	z
color	B	W	W	B	B
pred	N	s	z	s	y
dist	0	8	13	5	7



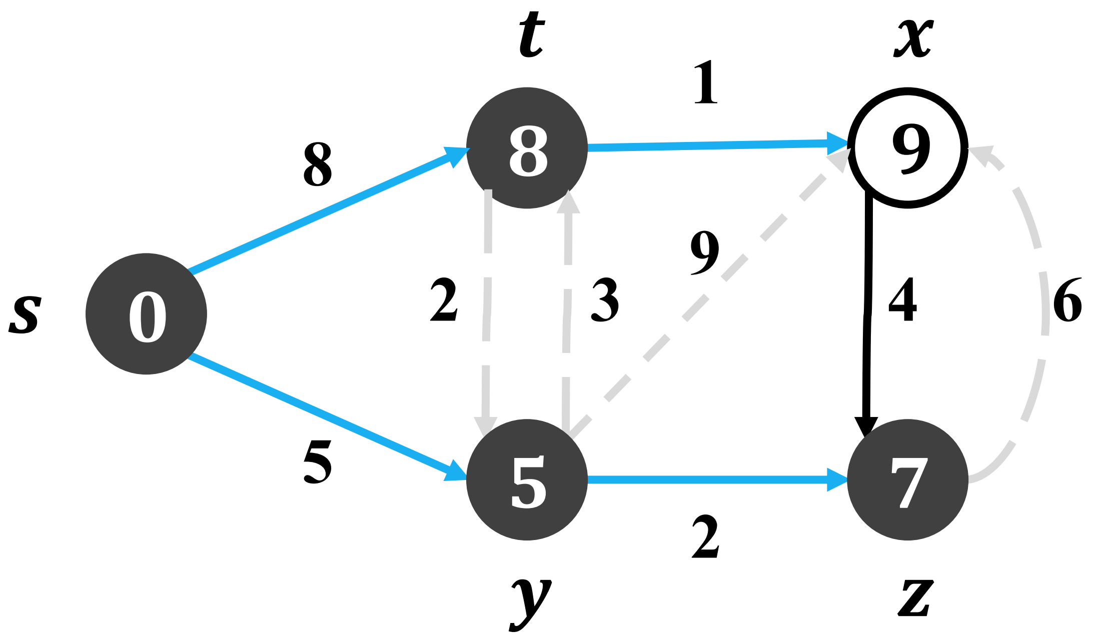
算法实例

V	s	t	x	y	z
color	B	W	W	B	B
pred	N	s	t	s	y
dist	0	8	9	5	7



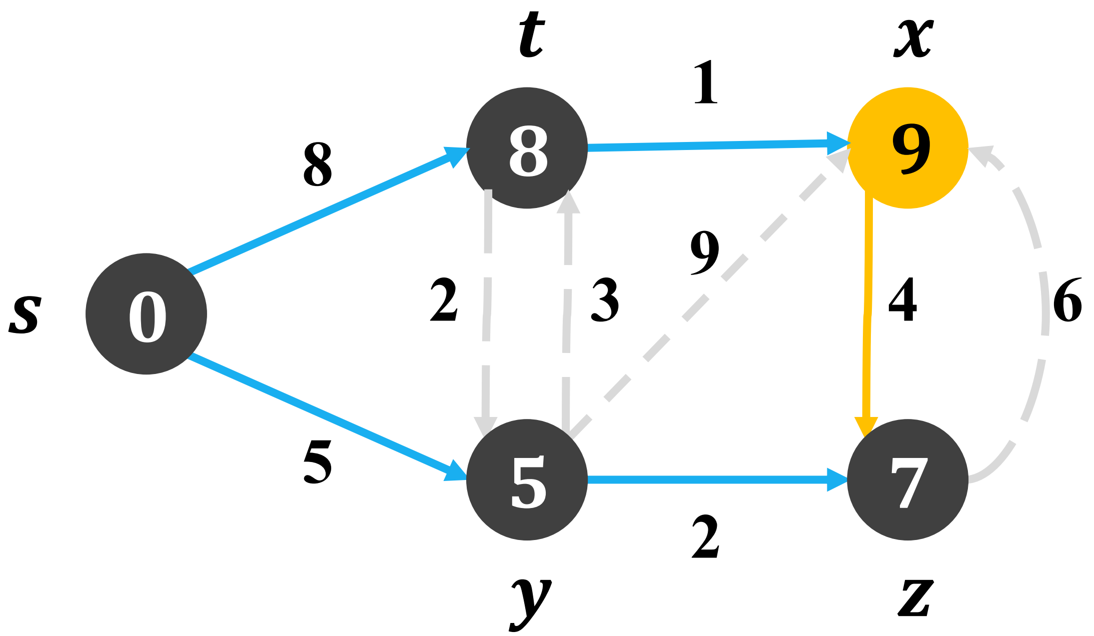
算法实例

V	s	t	x	y	z
$color$	B	B	W	B	B
$pred$	N	s	t	s	y
$dist$	0	8	9	5	7



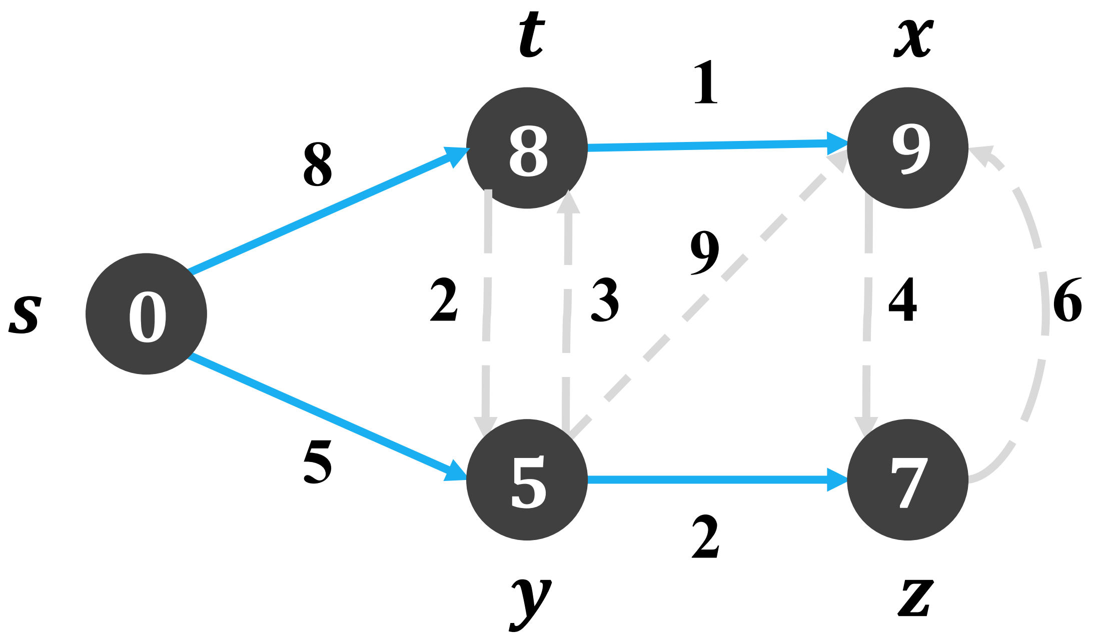
算法实例

V	s	t	x	y	z
color	B	B	W	B	B
pred	N	s	t	s	y
dist	0	8	9	5	7



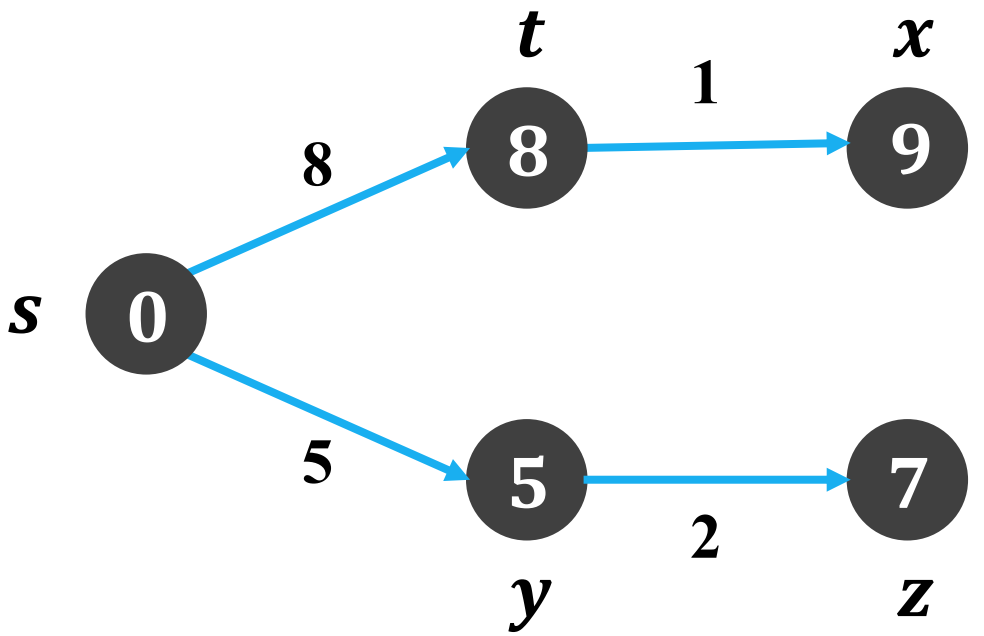
算法实例

V	s	t	x	y	z
color	B	B	B	B	B
pred	N	s	t	s	y
dist	0	8	9	5	7



算法实例

V	s	t	x	y	z
color	B	B	B	B	B
pred	N	s	t	s	y
dist	0	8	9	5	7



提纲

- 问题背景
- 算法思想
- 算法实例
- **算法分析**
- 算法性质

伪代码

- **Dijkstra(G, s)**

输入: 图$G =< V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$
//初始化
for $u \in V$ **do**
  $color[u] \leftarrow WHITE$
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$

初始化辅助数组

伪代码

- **Dijkstra(G, s)**

输入: 图$G = < V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$
//初始化
for $u \in V$ **do**
  $color[u] \leftarrow WHITE$
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$

源点到自身距离为0

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

依次计算源点到各顶点的最短路

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

***minDist*记录最小估计距离**
***rec*记录距源点最近的白色顶点**

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

选择距源点最近的白色顶点

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

对rec出发的边进行松弛

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

松弛操作

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

记录前驱顶点

伪代码

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

标记处理完成

时间复杂度分析

- **Dijkstra(G, s)**

输入: 图$G=<V,E,W>$, 源点s
输出: 单源最短路径P
新建一维数组$color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$
//初始化
for $u \in V$ **do**
    $color[u] \leftarrow WHITE$
    $dist[u] \leftarrow \infty$
    $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$

$\boldsymbol{O(|V|)}$

时间复杂度分析

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

时间复杂度分析

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

$O(\deg(u))$

时间复杂度分析

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

$O(|V| \cdot |V|) = O(|V|^2)$

$O(\deg(u))$

时间复杂度分析

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|)$

$O(|V| \cdot |V|) = O(|V|^2)$

$O(\deg(u))$

$\sum_{u \in V} \deg(u) = 2|E|$

时间复杂度分析

- **Dijkstra(G, s)**

```
//执行单源最短路径算法
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

$O(|V|^2)$

时间复杂度分析

- **Dijkstra(G, s)**

//执行单源最短路径算法

```
for i ← 1 to |V| do
    minDist ← ∞
    rec ← 0
    for j ← 1 to |V| do
        if color[j] ≠ BLACK and dist[j] < minDist then
            minDist ← dist[j]
            rec ← j
        end
    end
    for u ∈ G.Adj[rec] do
        if dist[rec] + w(rec, u) < dist[u] then
            dist[u] ← dist[rec] + w(rec, u)
            pred[u] ← rec
        end
    end
    color[rec] ← BLACK
end
```

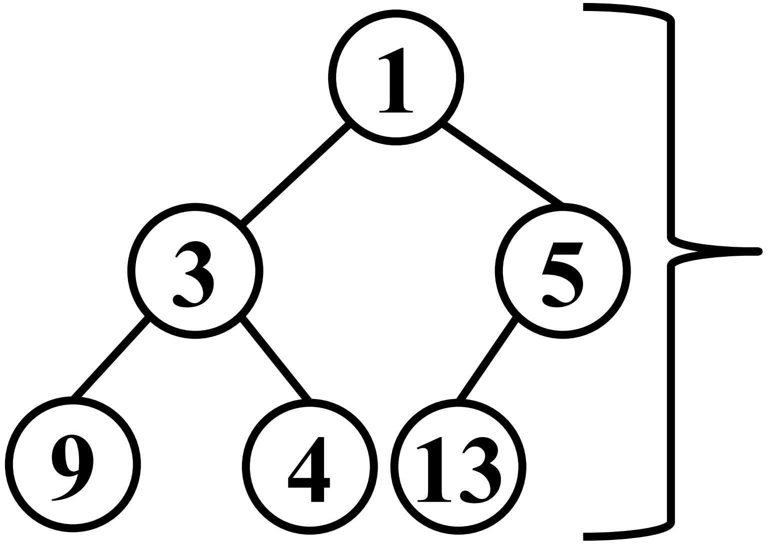
→ $O(\log|V|)$

使用优先队列，加速查询

数据结构：优先队列

优先队列

- 队列中每个元素有一个关键字，依据**关键字大小**离开队列
- 通过二叉堆来实现优先队列
  - $Q.Insert()$ 时间复杂度$O(\log n)$
  - $Q.ExtractMin()$ 时间复杂度$O(\log n)$
  - $Q.DecreaseKey()$ 时间复杂度$O(\log n)$



$O(\log n)$层

1	3	5	9	4	13

优先队列Q

伪代码

- **Dijkstra-PriQueue(G, s)**

```
输入: 图G =< V, E, W >, 源点s
输出: 单源最短路径P
新建一维数组color[1..|V|], dist[1..|V|], pred[1..|V|]
新建空优先队列Q
//初始化
for u ∈ V do
    color[u] ← WHITE
    dist[u] ← ∞
    pred[u] ← NULL
end
dist[s] ← 0
Q.Insert(V, dist)
```

初始化辅助数组

伪代码

- **Dijkstra-PriQueue(G, s)**

输入: 图$G =< V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$
新建空优先队列Q
//初始化
for $u \in V$ do
  $color[u] \leftarrow WHITE$
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$
$Q.Insert(V, dist)$

初始化源点和优先队列

伪代码

- **Dijkstra-PriQueue(G, s)**

//执行单源最短路径算法

依次计算到各点最短路

```
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

伪代码

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

选择最小估计距离的白色顶点

伪代码

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

对u出发的边进行松弛

伪代码

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

松弛操作，更新距离上界

伪代码

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

记录前驱顶点

伪代码

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

更新优先队列中关键字

伪代码

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

标记顶点计算完成

时间复杂度分析

- **Dijkstra-PriQueue(G, s)**

输入: 图$G =< V, E, W >$, 源点s
输出: 单源最短路径P
新建一维数组$color[1..|V|]$, $dist[1..|V|]$, $pred[1..|V|]$
新建空优先队列Q
//初始化
for $u \in V$ **do**
  $color[u] \leftarrow WHITE$
  $dist[u] \leftarrow \infty$
  $pred[u] \leftarrow NULL$
end
$dist[s] \leftarrow 0$
$Q.Insert(V, dist)$

$O(|V|)$

时间复杂度分析

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()                                   O(log|V|)
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))                  O(log|V|)
        end
    end
    color[v] ← BLACK
end
```

时间复杂度分析

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()                                  O(log|V|)
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))   O(log|V|)     O(deg(u) · log|V|)
        end
    end
    color[v] ← BLACK
end
```

时间复杂度分析

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

$O(|V|\log|V|)$

$O(\log|V|)$

$O(\deg(u) \cdot \log|V|)$

时间复杂度分析

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

$O(\log|V|)$

$O(\deg(u) \cdot \log|V|)$

$O(|V|\log|V|)$

$O(|E|\log|V|)$

$$\sum_{u\in V} \deg(u) = 2|E|$$

时间复杂度分析

- **Dijkstra-PriQueue(G, s)**

```
//执行单源最短路径算法
while 优先队列Q非空 do
    v ← Q.ExtractMin()
    for u ∈ G.adj[v] do
        if dist[v] + w(v, u) < dist[u] then
            dist[u] ← dist[v] + w(v, u)
            pred[u] ← v
            Q.DecreaseKey((u, dist[u]))
        end
    end
    color[v] ← BLACK
end
```

时间复杂度$O(|E| \cdot \log|V|)$

提纲

问题背景

算法思想

算法实例

算法分析

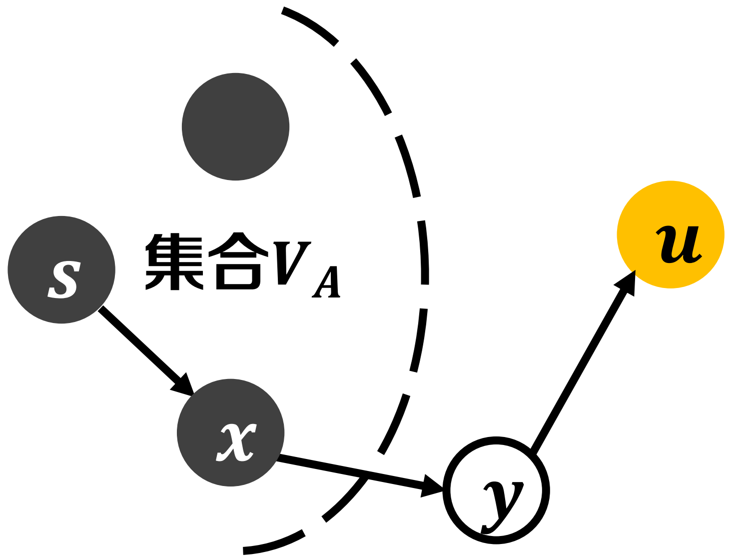
算法性质

正确性证明

- **定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s, u)$**

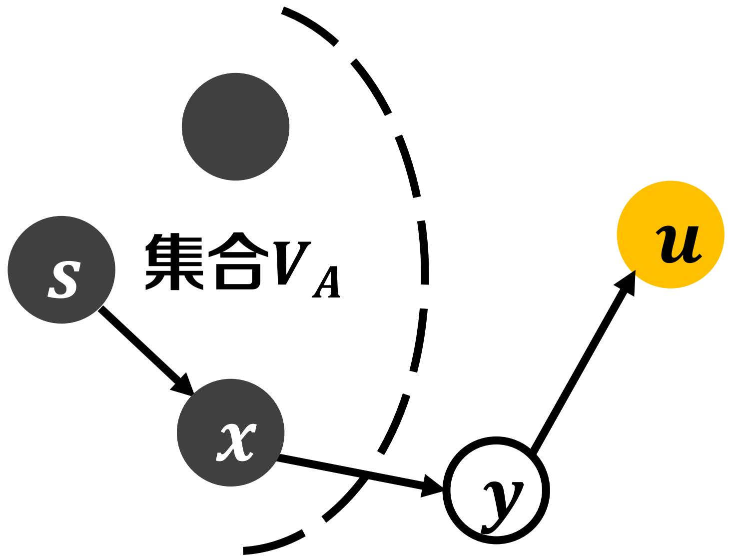
正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s, u)$
- 证明：
  - 采用反证法，假设Dijkstra算法将顶点u添加到V_A时，$dist[u] \neq \delta(s, u)$



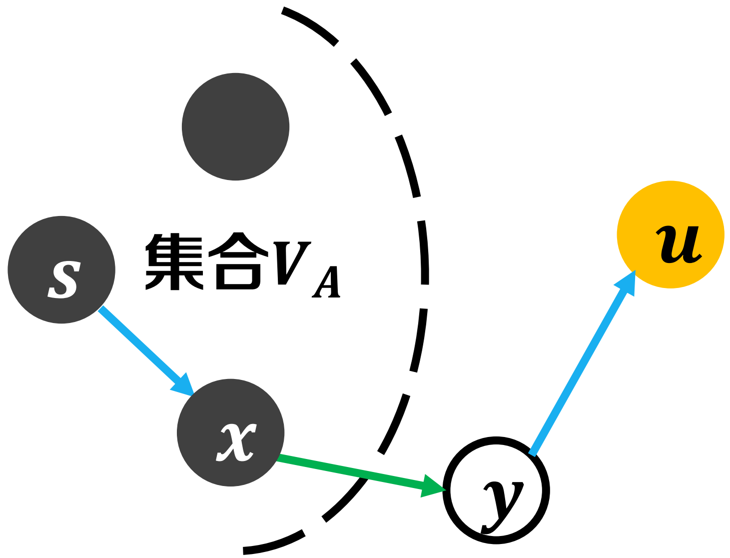
正确性证明

- **定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$**
- **证明：**
  - 采用反证法，假设Dijkstra算法将顶点u添加到V_A时，$dist[u] \neq \delta(s,u)$
  - 由于$dist[u]$作为$\delta(s,u)$的上界，故$dist[u] > \delta(s,u)$



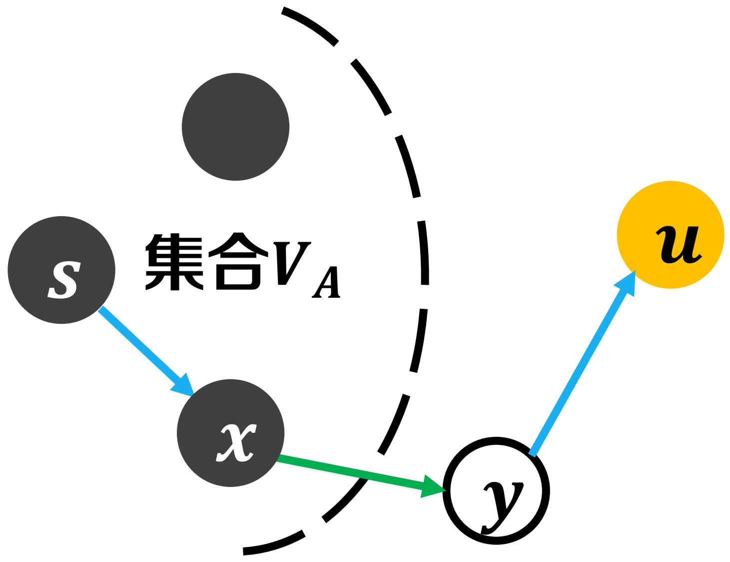
正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$
- 证明：
  - 采用反证法，假设Dijkstra算法将顶点u添加到V_A时，$dist[u] \neq \delta(s,u)$
  - 由于$dist[u]$作为$\delta(s,u)$的上界，故$dist[u] > \delta(s,u)$
  - 应存在一条长度为$\delta(s,u)$的从 s到u的最短路径，不妨设其为$< s, \dots, x, y, \dots, u >$，其中边$(x,y)$横跨$< V_A, V - V_A >$，$x \in V_A, y \in V - V_A$



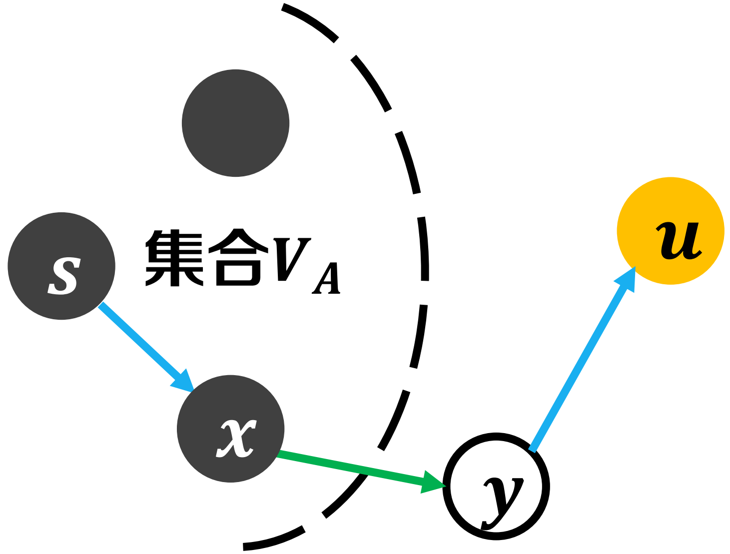
正确性证明

- **定理**：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s, u)$
- **证明**：
  - 采用反证法，假设Dijkstra算法将顶点u添加到V_A时，$dist[u] \neq \delta(s, u)$
  - 由于$dist[u]$作为$\delta(s, u)$的上界，故$dist[u] > \delta(s, u)$
  - 应存在一条长度为$\delta(s, u)$的从 s到u的最短路径，不妨设其为$< s, \dots, x, y, \dots, u >$，其中边$(x, y)$横跨$< V_A, V - V_A >$，$x \in V_A, y \in V - V_A$
  - 算法令V_A 中的顶点满足：$dist[x] = \delta(s, x), x \in V_A$



正确性证明

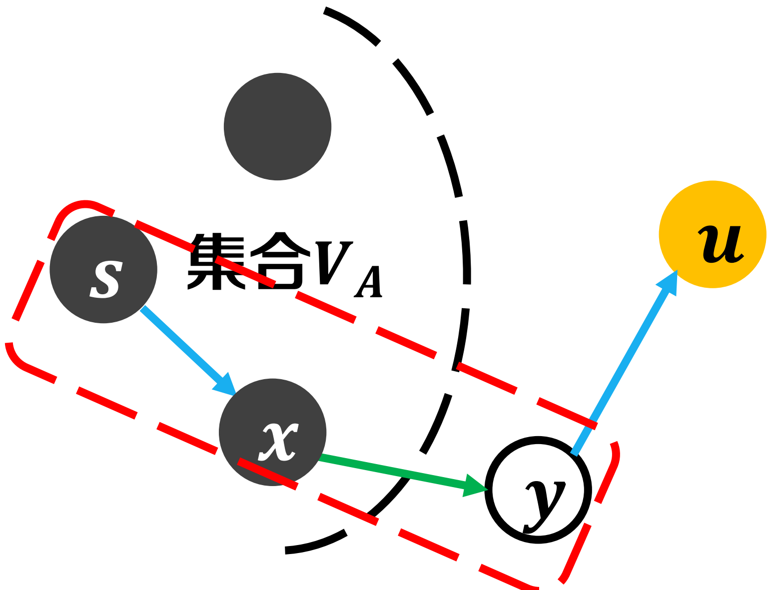
- **定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$**
- **证明：**
  - 采用反证法，假设Dijkstra算法将顶点u添加到V_A时，$dist[u] \neq \delta(s,u)$
  - 由于$dist[u]$作为$\delta(s,u)$的上界，故$dist[u] > \delta(s,u)$
  - 应存在一条长度为$\delta(s,u)$的从 s到u的最短路径，不妨设其为$< s, \ldots, x, y, \ldots, u >$，其中边$(x,y)$横跨$< V_A, V - V_A >$，$x \in V_A, y \in V - V_A$
  - 算法令V_A 中的顶点满足：$dist[x] = \delta(s,x), x \in V_A$



问题：$dist[y]$和$\delta(s,y)$具有何种关系？

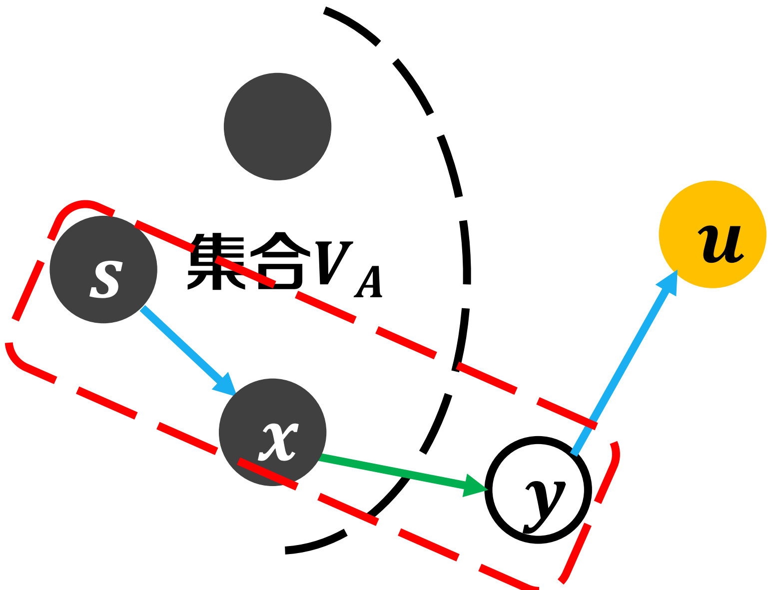
正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$
- 证明（接上页）：
  - $< s, \dots, x, y >$是最短路径$< s, \dots, x, y, \dots, u >$的子路径



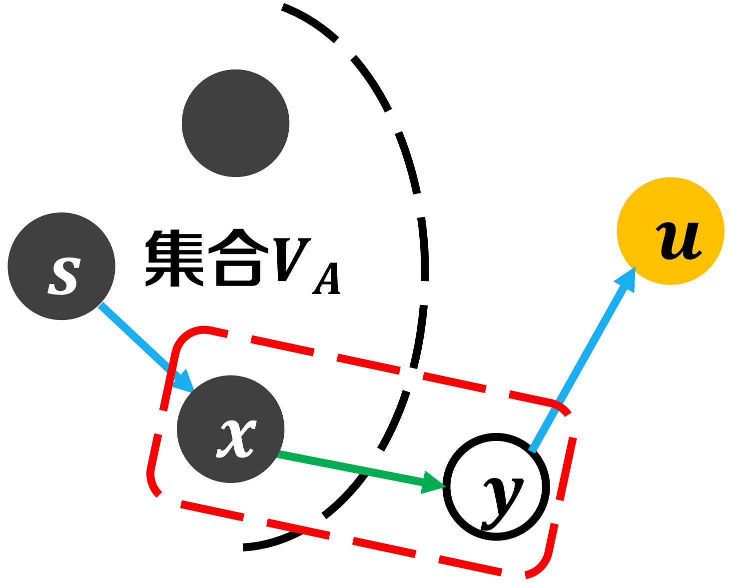
正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$
- 证明（接上页）：
  - $< s, \dots, x, y >$是最短路径$< s, \dots, x, y, \dots, u >$的子路径，故：
    - $\delta(s,y) = \delta(s,x) + w(x,y) = dist[x] + w(x,y)$ （公式1）



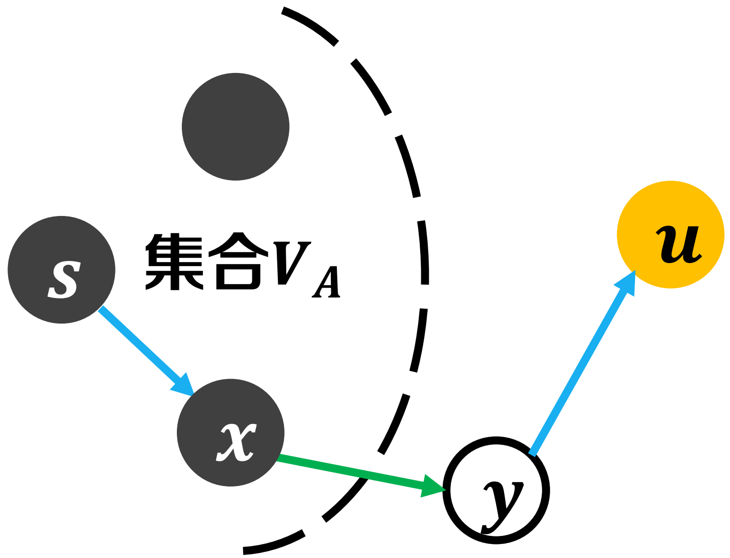
正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$
- 证明（接上页）：
  - $< s, \ldots, x, y >$是最短路径$< s, \ldots, x, y, \ldots, u >$的子路径，故：
    - $\delta(s,y) = \delta(s,x) + w(x,y) = dist[x] + w(x,y)$ （公式1）
  - 算法对顶点x出发的所有边（包括边(x,y)）已进行松弛操作，故：
    - $dist[y] \leq dist[x] + w(x,y)$ （公式2）



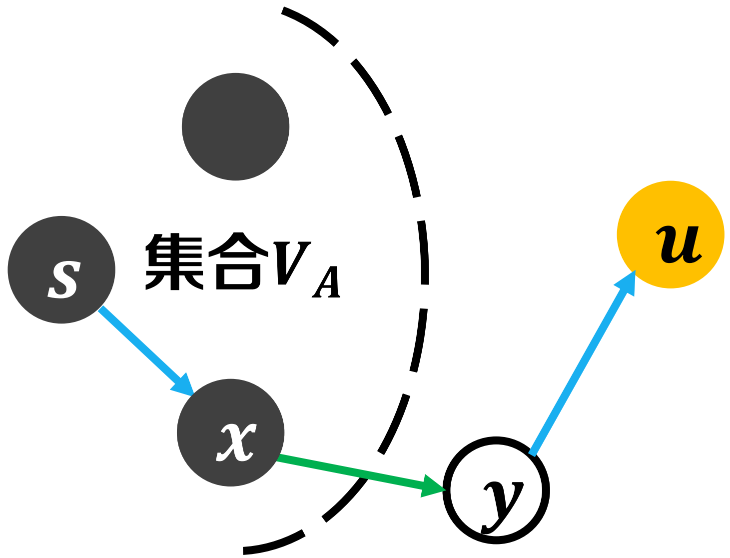
正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s, u)$
- 证明（接上页）：
  - $< s, \ldots, x, y >$是最短路径$< s, \ldots, x, y, \ldots, u >$的子路径，故：
    - $\delta(s, y) = \delta(s, x) + w(x, y) = dist[x] + w(x, y)$ （公式1）
  - 算法对顶点x出发的所有边（包括边(x, y)）已进行松弛操作，故：
    - $dist[y] \leq dist[x] + w(x, y)$（公式2）
  - 合并上述公式1和公式2，可得$dist[y] = \delta(s, y)$



正确性证明

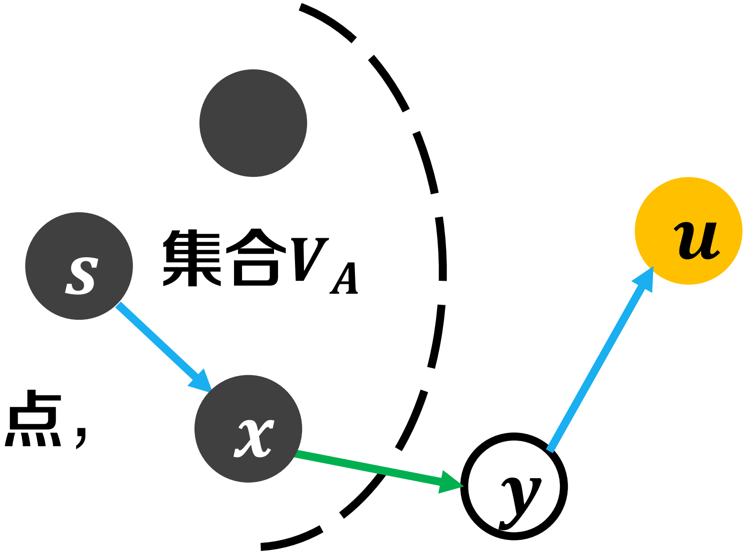
- **定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s, u)$**
- **证明（接上页）：**
  - $< s, \dots, x, y >$是最短路径$< s, \dots, x, y, \dots, u >$的子路径，故：
    - $\delta(s, y) = \delta(s, x) + w(x, y) = dist[x] + w(x, y)$ （公式1）
  - 算法对顶点x出发的所有边（包括边(x, y)）已进行松弛操作，故：
    - $dist[y] \le dist[x] + w(x, y)$ （公式2）
  - 合并上述公式1和公式2，可得$dist[y] = \delta(s, y)$
  - 最短路径$< s, \dots, x, y, \dots, u >$中， y出现在u之前，故：
    - $dist[u] > \delta(s, u) \ge \delta(s, y) = dist[y]$



正确性证明

- 定理：Dijkstra算法中，顶点u被添加到V_A时，$dist[u] = \delta(s,u)$
- 证明（接上页）：
  - $< s, \dots, x, y >$是最短路径$< s, \dots, x, y, \dots, u >$的子路径，故：
    - $\delta(s,y) = \delta(s,x) + w(x,y) = dist[x] + w(x,y)$ （公式1）
  - 算法对顶点x出发的所有边（包括边(x,y)）已进行松弛操作，故：
    - $dist[y] \le dist[x] + w(x,y)$ （公式2）
  - 合并上述公式1和公式2，可得$dist[y] = \delta(s,y)$
  - 最短路径$< s, \dots, x, y, \dots, u >$中， y出现在u之前，故：
    - $dist[u] > \delta(s,u) \ge \delta(s,y) = dist[y]$
  - $dist[u] > dist[y]$，$u \ne y$， u不应是下一个被添加顶点，

  故产生矛盾



小结

	广度优先搜索	Dijkstra算法
适用范围	无权图	带权图 （所有边权为正）
数据结构	队列	优先队列
运行时间	$O(\lvert V\rvert + \lvert E\rvert)$	$O(\lvert E\rvert \cdot \log\lvert V\rvert)$