Dynamic Programming

Endre Boros
26:711:653: Discrete Optimization

Optimality Principle

- Dynamic programming was formulated in many ways in many distinct areas of mathematics and operations research.
- It is another way of talking about mathematical recursion.
- In the context of sequential decision making Bellman (1952,1957) formulated it as the **Principle of Optimality**: *An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the (current) state resulting from the first decision.*

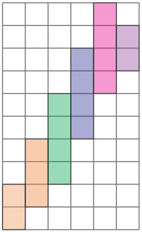
Optimality Principle

- Dynamic programming was formulated in many ways in many distinct areas of mathematics and operations research.
- It is another way of talking about mathematical recursion.
- In the context of sequential decision making Bellman (1952,1957) formulated it as the **Principle of Optimality**: *An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the (current) state resulting from the first decision.*

Optimality Principle

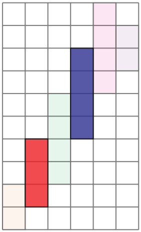
- Dynamic programming was formulated in many ways in many distinct areas of mathematics and operations research.
- It is another way of talking about mathematical recursion.
- In the context of sequential decision making Bellman (1952,1957) formulated it as the **Principle of Optimality**: *An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the (current) state resulting from the first decision.*

A scheduling example



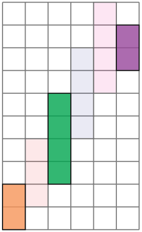
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



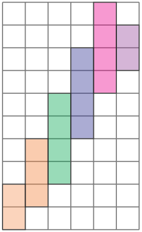
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



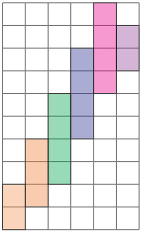
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



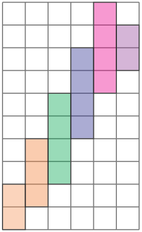
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



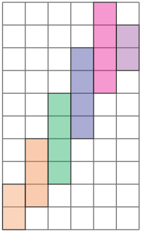
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



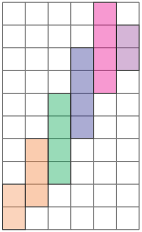
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i)} = \max\{\mathbf{c_i} + \mathbf{v(k(i))}; \mathbf{v(i+1)}\}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



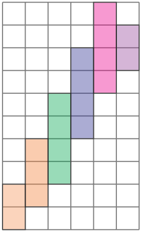
- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i)} = \max\{\mathbf{c_i} + \mathbf{v(k(i))}; \mathbf{v(i+1)}\}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

A scheduling example



- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- How to find the optimal subset of jobs?

A scheduling example



- Given n jobs: start time s_i, processing time p_i, profit c_i, $i = 1, ..., n$.
- Find the maximum profit subset that can be processed on a single processor without overlap.
- Assume that $s_1 \leq s_2 \leq \cdots \leq s_n$.
- Let $J(i) = \{i, i+1, ..., n\}$ and $v(i)$ is the optimum value for the problem when $J(i)$ is the input, $i = 1, ..., n$.
- Let $k(i)$ be the smallest j such that $s_j \geq s_i + p_i$.
- $\mathbf{v(i) = \max\{c_i + v(k(i)); v(i+1)\}}$
- Compute first $v(n)$, then $v(n-1)$, ..., and last $v(1)$
- $v(1)$ is our optimum value!
- **How to find the optimal subset of jobs?**

Scheduling Example Cont'd

```
1: procedure DPScheduling((s_i, p_i, c_i), i = 1, ..., n)
2:     Sort jobs by start times: s_1 ≤ s_2 ≤ ··· ≤ s_n
3:     Compute k(i) = min{k | s_k ≥ s_i + p_i} ∪ {n + 1} for all i.
4:     Set v(n + 1) = 0.
5:     for ℓ = n → 1 do
6:         Set v(ℓ) = max{c_ℓ + v(k(ℓ)), v(ℓ + 1)}.
7:     end for
8:     return v(1)
9: end procedure
```

- Step 2: $O(n \log n)$
- Step 3: $O(n \log n)$
- Steps 5-7: n times $O(1)$
- In total: $O(n \log n)$.

Scheduling Example Cont'd

1: **procedure** DPScheduling($(s_i, p_i, c_i),\ i = 1, ..., n$)
2: Sort jobs by start times: $s_1 \leq s_2 \leq \cdots \leq s_n$
3: Compute $k(i) = \min\{k \mid s_k \geq s_i + p_i\} \cup \{n+1\}$ for all i.
4: Set $v(n+1) = 0$.
5: **for** $\ell = n \to 1$ **do**
6: Set $v(\ell) = \max\{c_\ell + v(k(\ell)),\ v(\ell+1)\}$.
7: **end for**
8: **return** $v(1)$
9: **end procedure**

- Step 2: $O(n \log n)$
- Step 3: $O(n \log n)$
- Steps 5-7: n times $O(1)$
- In total: $O(n \log n)$.

Scheduling Example Cont'd

```
1: procedure DPScheduling((s_i, p_i, c_i), i = 1, ..., n)
2:     Sort jobs by start times: s_1 ≤ s_2 ≤ ··· ≤ s_n
3:     Compute k(i) = min{k | s_k ≥ s_i + p_i} ∪ {n + 1} for all i.
4:     Set v(n + 1) = 0.
5:     for ℓ = n → 1 do
6:         Set v(ℓ) = max{c_ℓ + v(k(ℓ)), v(ℓ + 1)}.
7:     end for
8:     return v(1)
9: end procedure
```

- Step 2: $O(n \log n)$
- Step 3: $O(n \log n)$
- Steps 5-7: n times $O(1)$
- In total: $O(n \log n)$.

Scheduling Example Cont'd

1: **procedure** DPSCHEDULING$((s_i, p_i, c_i),\ i = 1, ..., n)$
2: Sort jobs by start times: $s_1 \leq s_2 \leq \cdots \leq s_n$
3: Compute $k(i) = \min\{k \mid s_k \geq s_i + p_i\} \cup \{n+1\}$ for all i.
4: Set $v(n+1) = 0$.
5: **for** $\ell = n \to 1$ **do**
6: Set $v(\ell) = \max\{c_\ell + v(k(\ell)),\ v(\ell + 1)\}$.
7: **end for**
8: **return** $v(1)$
9: **end procedure**

- Step 2: $O(n \log n)$
- Step 3: $O(n \log n)$
- Steps 5-7: n times $O(1)$
- In total: $O(n \log n)$.

Scheduling Example Cont'd

1: **procedure** DPSCHEDULING($(s_i, p_i, c_i),\ i = 1, ..., n$)
2: Sort jobs by start times: $s_1 \leq s_2 \leq \cdots \leq s_n$
3: Compute $k(i) = \min\{k \mid s_k \geq s_i + p_i\} \cup \{n+1\}$ for all i.
4: Set $v(n+1) = 0$.
5: **for** $\ell = n \to 1$ **do**
6: Set $v(\ell) \ = \ \max\{c_\ell + v(k(\ell)),\ v(\ell+1)\}$.
7: **end for**
8: **return** $v(1)$
9: **end procedure**

- Step 2: $O(n \log n)$
- Step 3: $O(n \log n)$
- Steps 5-7: n times $O(1)$
- In total: $O(n \log n)$.

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**		
2	1	3	**3**		
3	2	4	**4**		
4	4	4	**5**		
5	6	4	**6**		
6	7	2	**2**		
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**		
3	2	4	**4**		
4	4	4	**5**		
5	6	4	**6**		
6	7	2	**2**		
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**		
4	4	4	**5**		
5	6	4	**6**		
6	7	2	**2**		
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	$\mathbf{1}$	3	
2	1	3	$\mathbf{3}$	4	
3	2	4	$\mathbf{4}$	5	
4	4	4	$\mathbf{5}$		
5	6	4	$\mathbf{6}$		
6	7	2	$\mathbf{2}$		
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**		
6	7	2	**2**		
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**	7	
6	7	2	**2**		
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**	7	
6	7	2	**2**	7	
7	–	–	–		

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**	7	
6	7	2	**2**	7	
7	–	–	–	–	

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**	7	
6	7	2	**2**	7	
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**	7	
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	
2	1	3	**3**	4	**10**
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	1	3	**11**
2	1	3	**3**	4	**10**
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	**11**
2	1	3	**3**	4	**10**
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	**11**
2	1	3	**3**	4	**10**
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	**11**
2	1	3	**3**	4	**10**
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

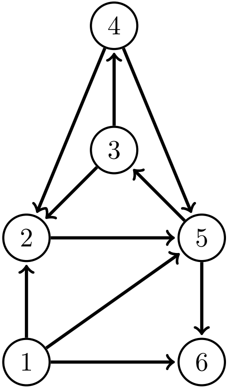
- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

Scheduling Example Cont'd

i	s_i	p_i	c_i	$k(i)$	$v(i)$
1	0	2	**1**	3	**11**
2	1	3	**3**	4	**10**
3	2	4	**4**	5	**10**
4	4	4	**5**	7	**6**
5	6	4	**6**	7	**6**
6	7	2	**2**	7	**2**
7	–	–	–	–	**0**

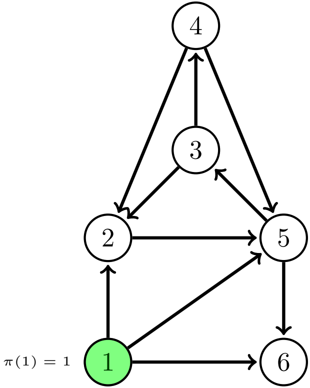
- **How to find the optimal set of jobs?**
- Feasible job sets **independent** $\rightsquigarrow$ independence system $(J, \mathcal{F})$!
- What is $q(J, \mathcal{F})$ for this example?
- What is it in the worst case?

A reachability example



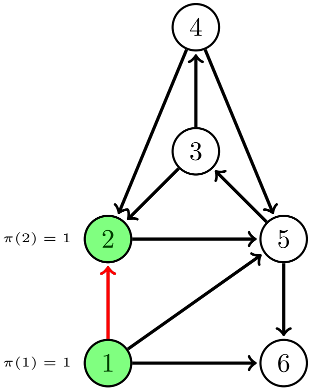
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



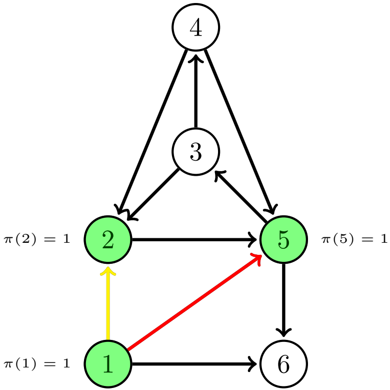
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



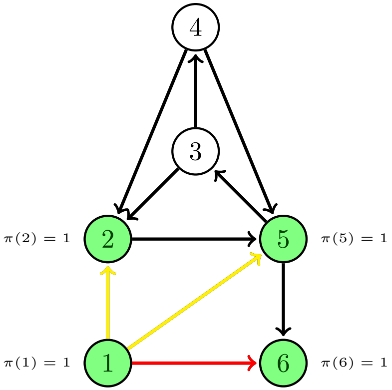
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



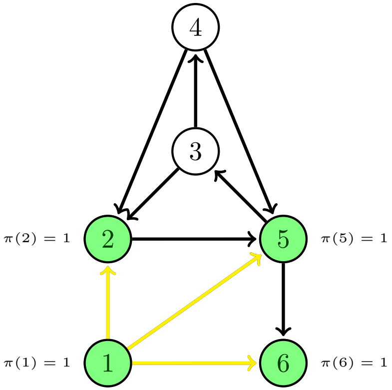
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



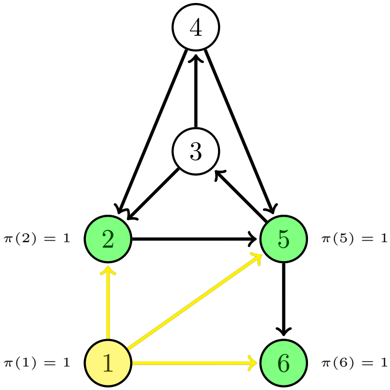
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \rightarrow v$ path.

A reachability example



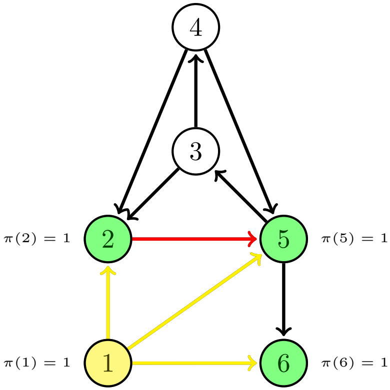
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



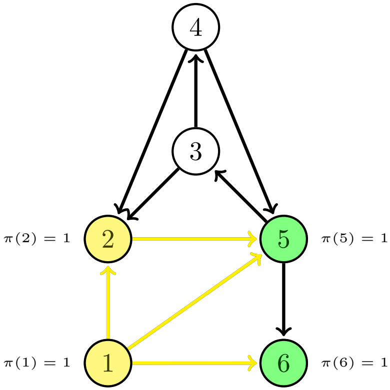
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



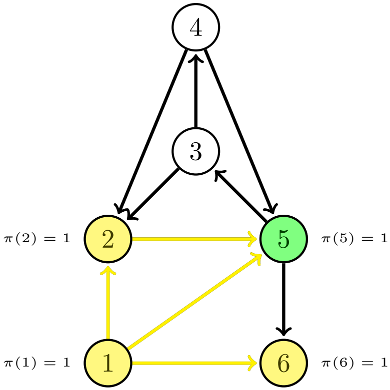
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



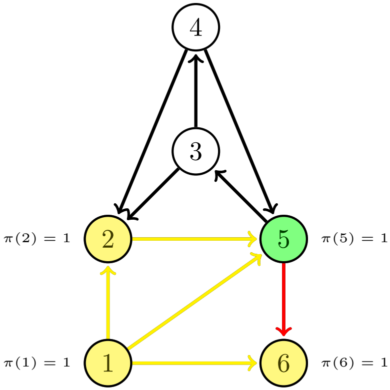
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



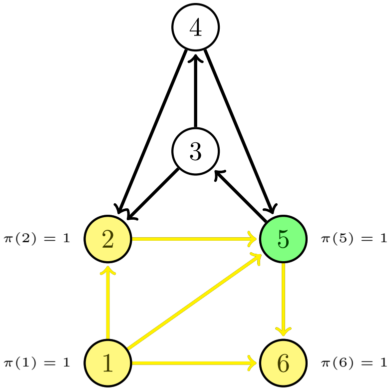
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



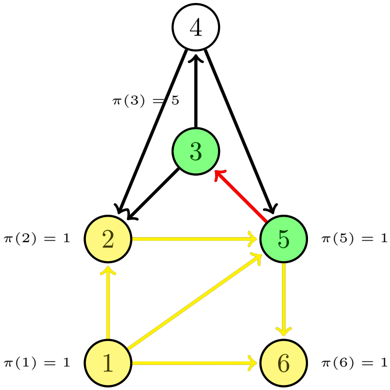
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



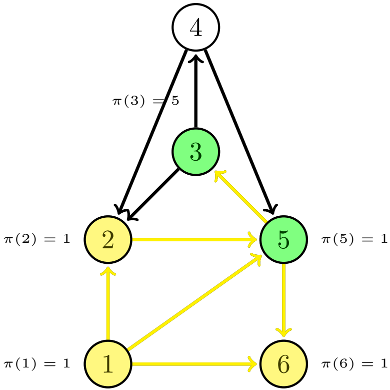
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



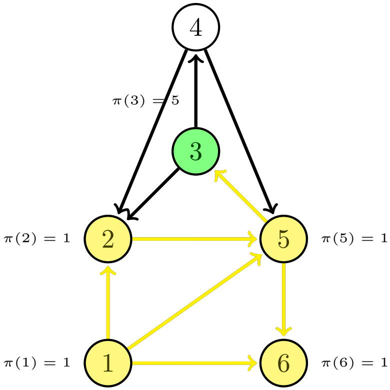
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



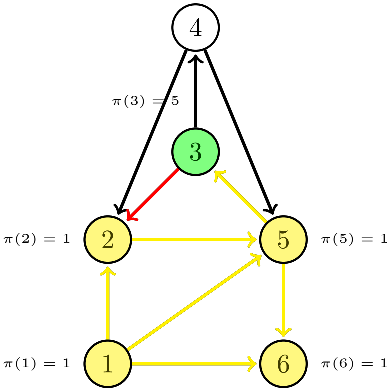
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



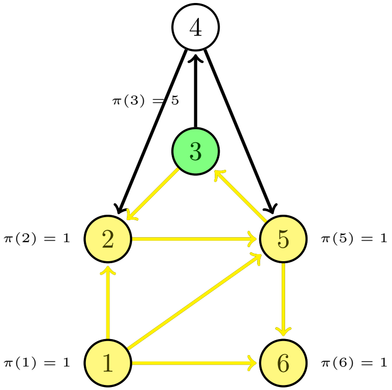
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



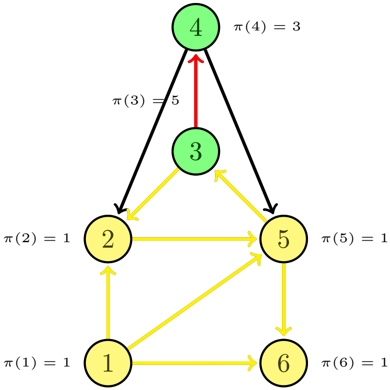
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



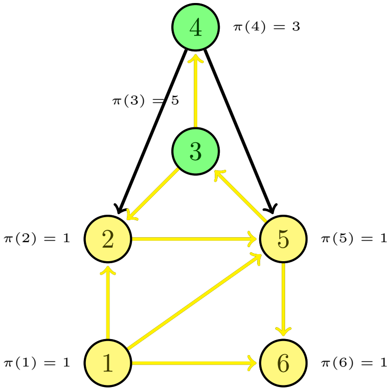
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



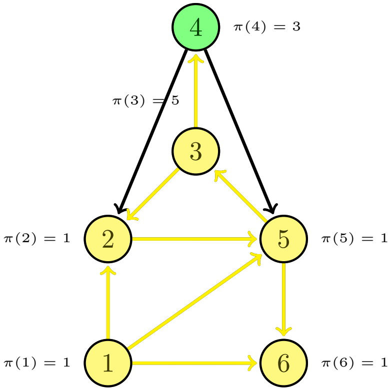
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



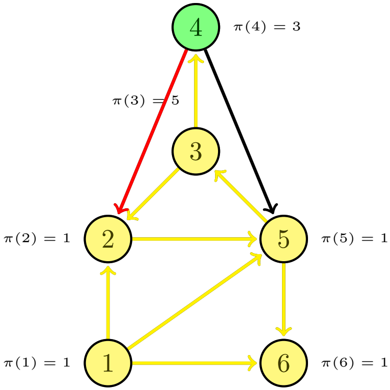
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



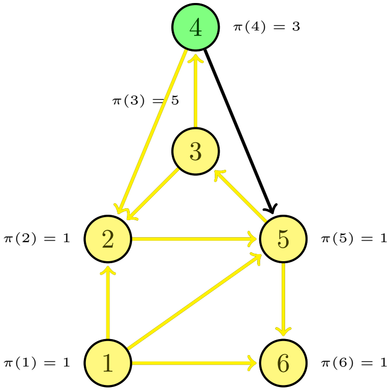
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



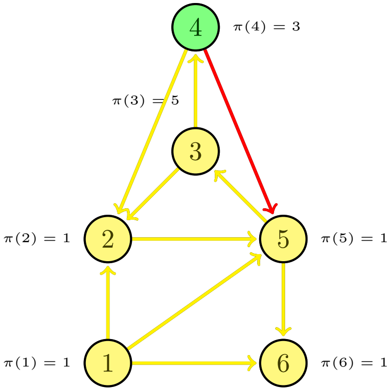
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



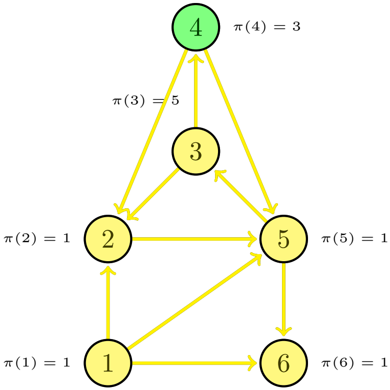
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



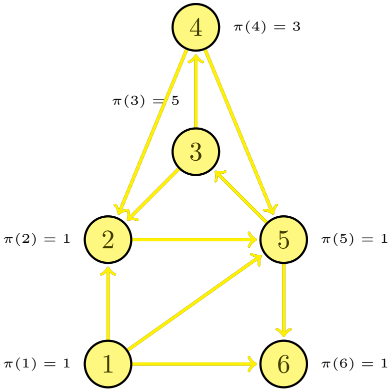
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



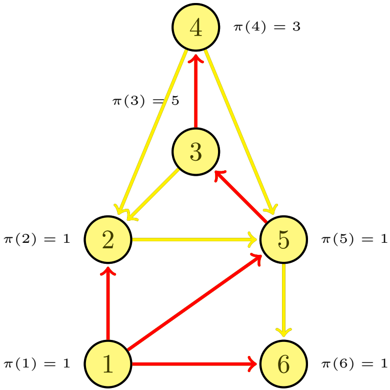
- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example



- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

A reachability example

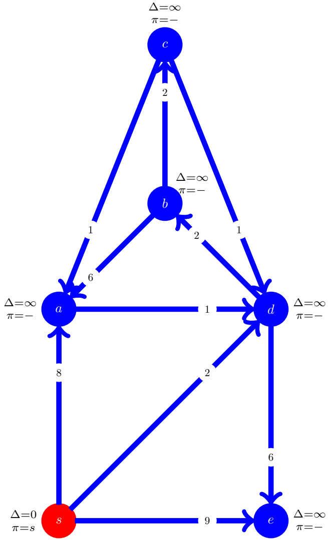


- Start with the source: $s = 1$.
- When $N^+(s)$ is processed move to another already reached node.
- When all nodes are processed, label $\pi(v)$ is the predecessor on the $s \to v$ path.

Reachability: the labeling algorithm

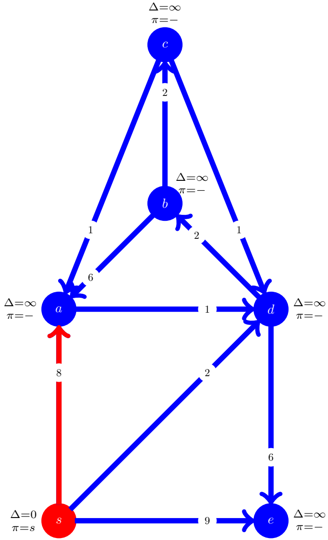
procedure LABELING ALGORITHM($G = (V, A)$ and $s \in V$)
    Set $L = \emptyset$.
    Set $T = \{s\}$.
    Set $\pi(v) = nil$ for all $v \in V \setminus \{s\}$, and set $\pi(s) = s$.
    while $T \neq \emptyset$ **do**
        Let $u \in T$ and set $T = T \setminus \{u\}$.
        for $v \in N^+(u)$ **do**
            if $\pi(v) = nil$ **then**
                Set $\pi(v) = u$ and $T = T \cup \{v\}$.
            end if
        end for
        Set $L = L \cup \{u\}$.
    end while
    return L
end procedure

A shortest path example: Dijkstra's algorithm



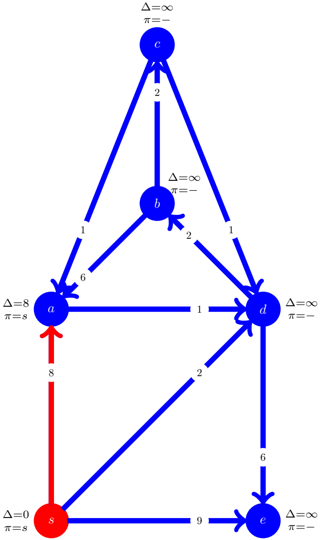
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



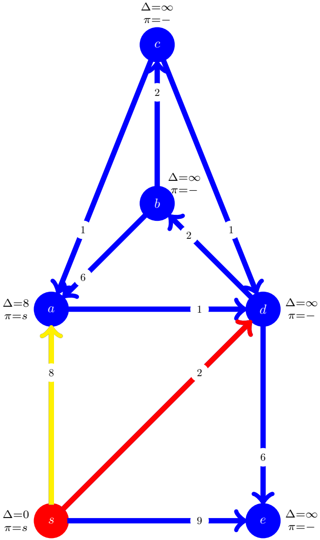
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



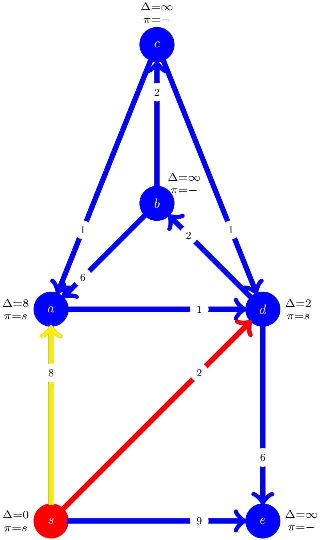
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



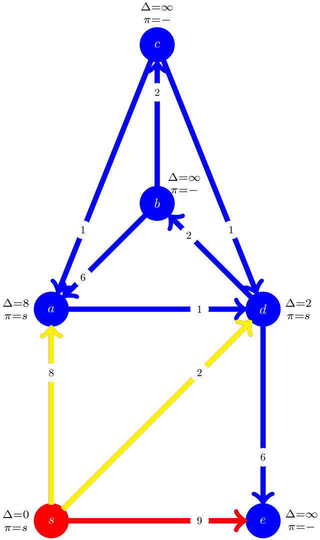
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



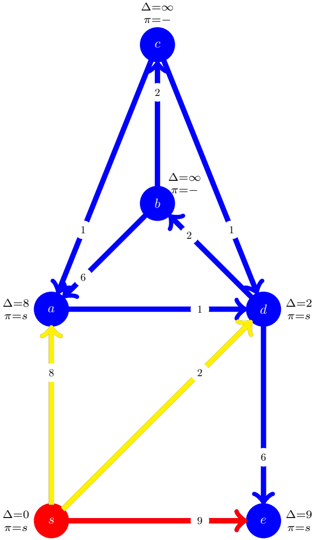
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



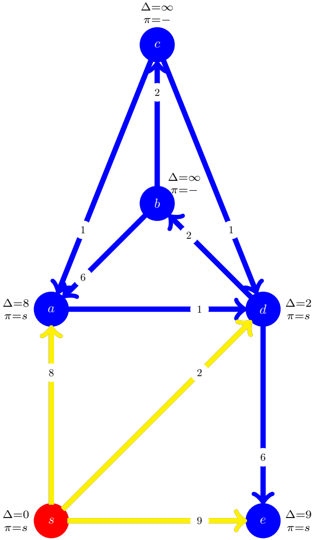
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



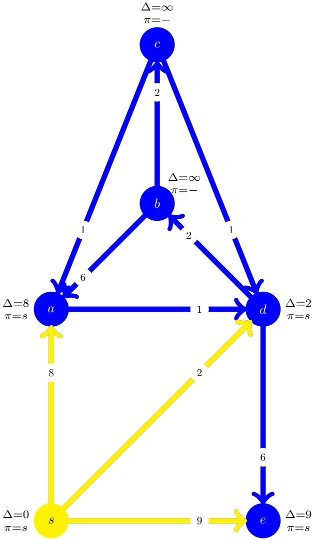
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



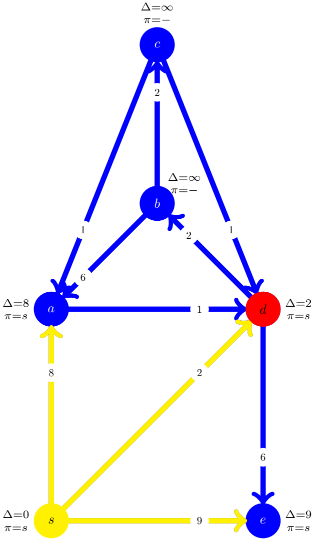
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



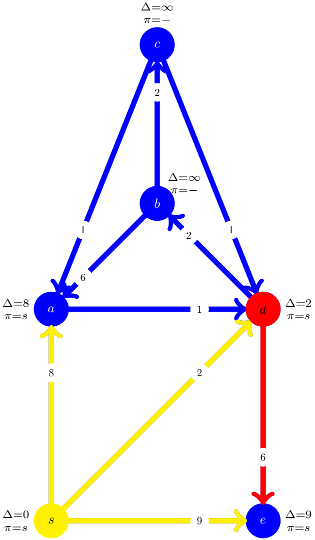
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



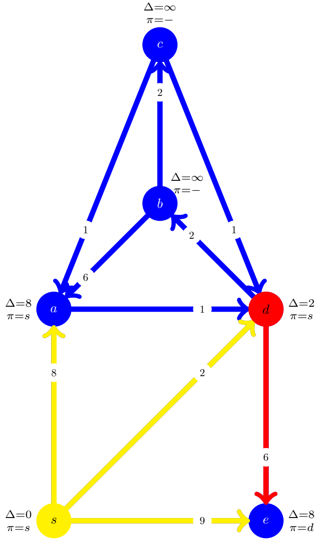
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



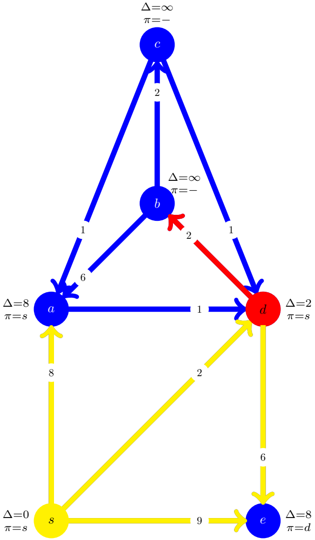
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



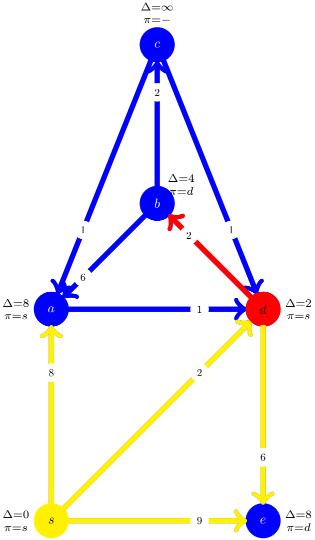
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



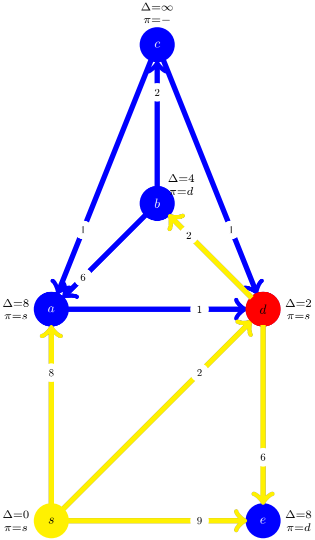
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



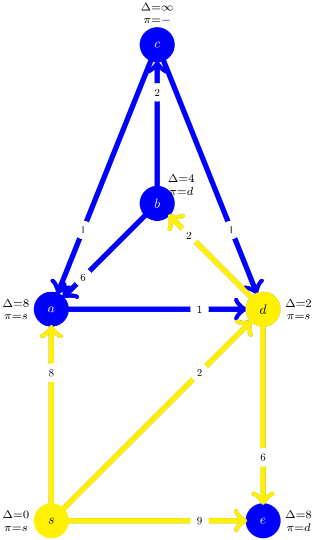
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



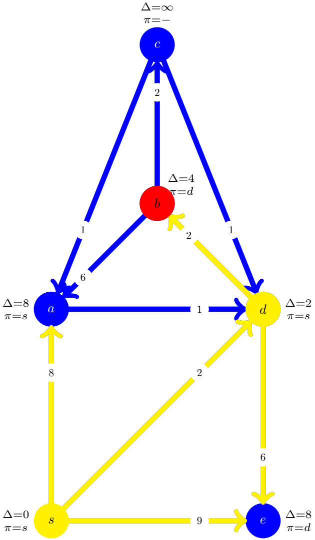
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



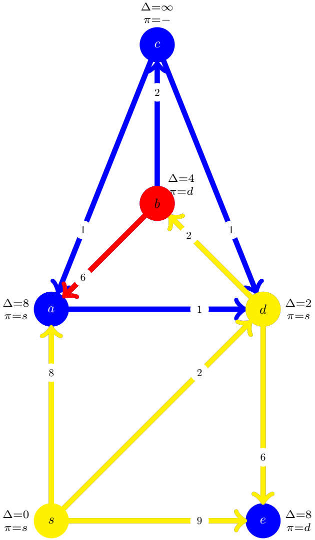
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



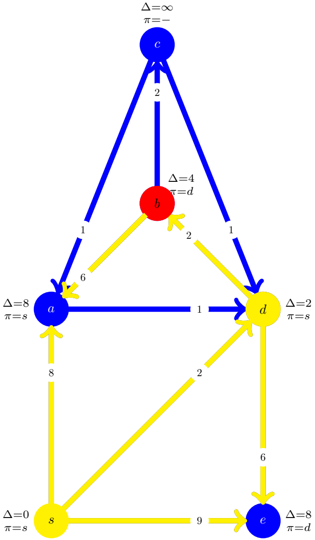
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



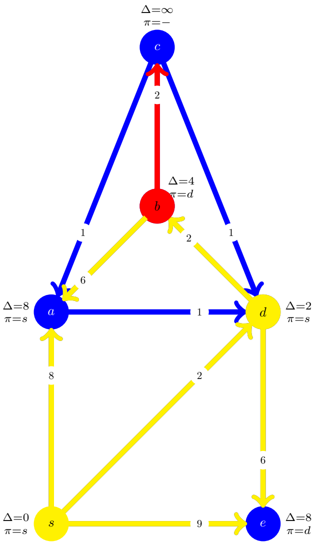
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



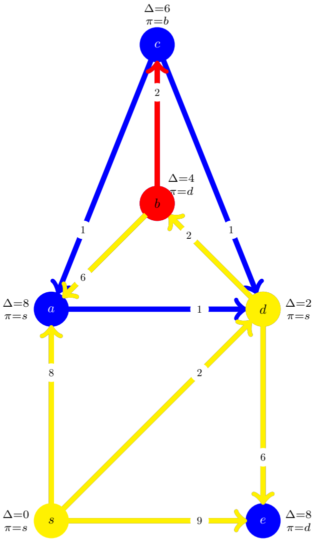
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



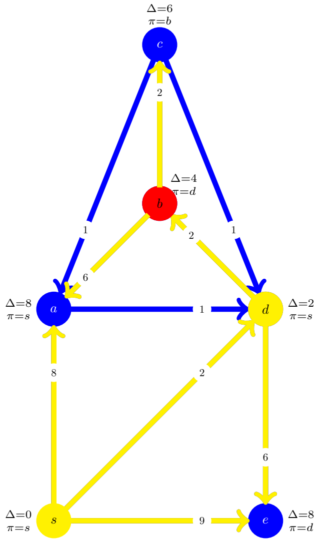
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



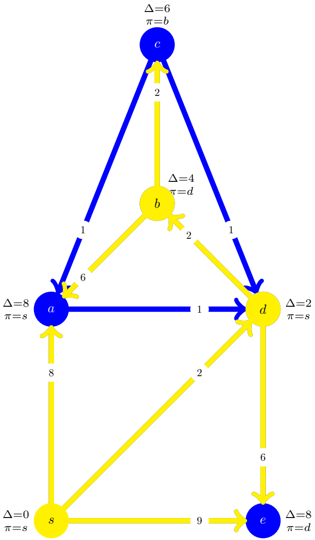
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



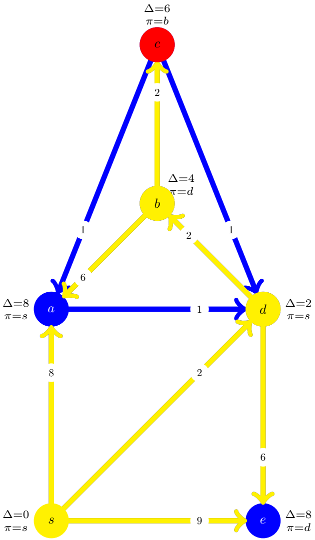
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



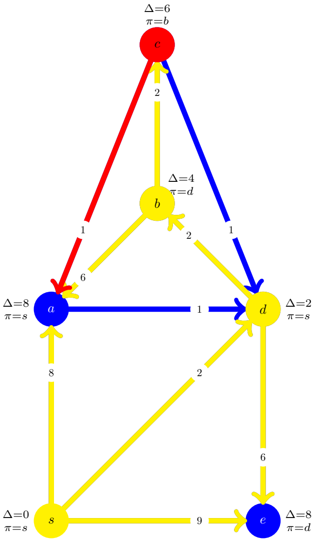
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



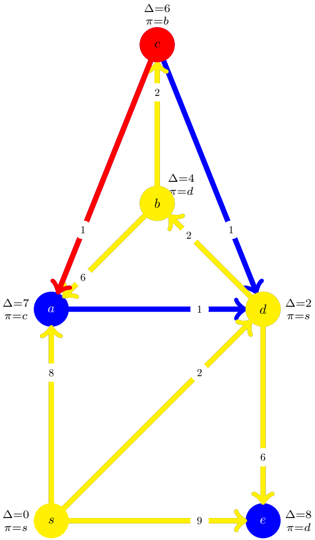
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



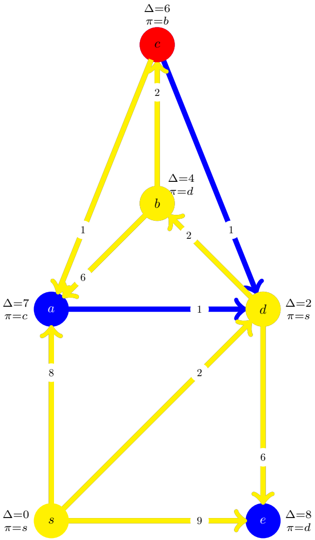
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



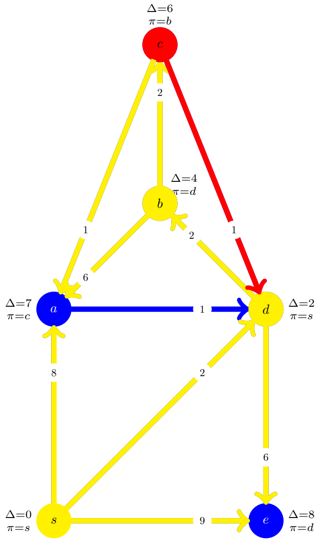
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



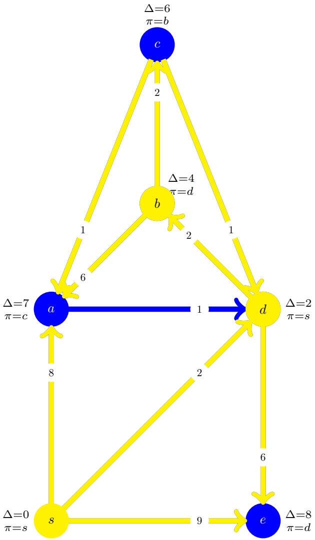
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



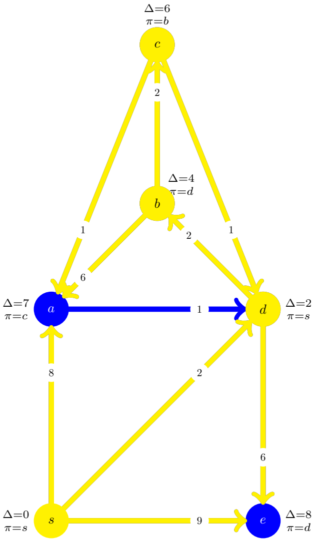
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



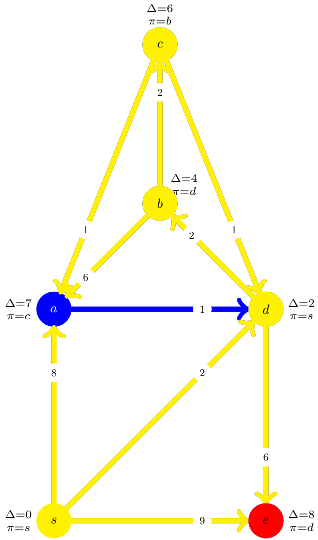
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



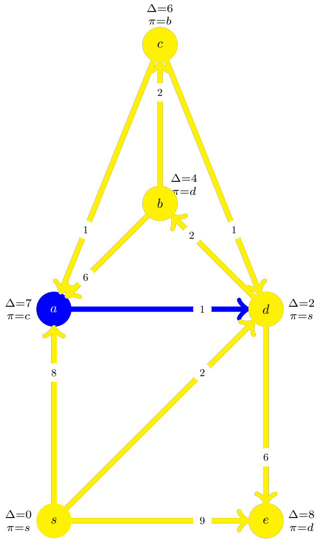
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



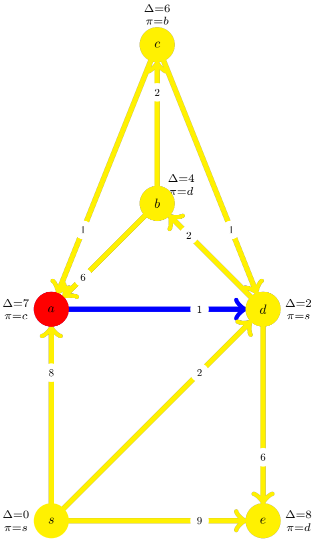
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



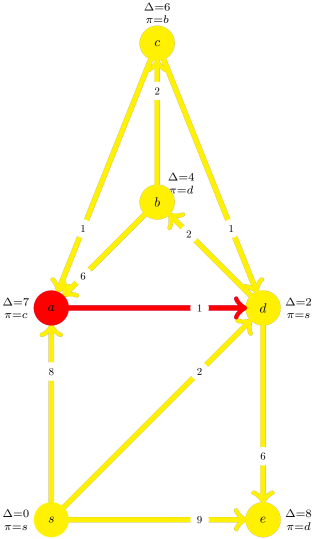
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



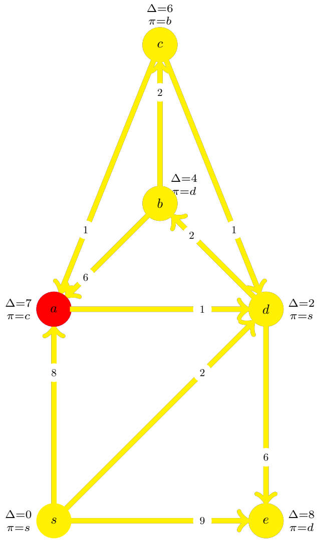
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



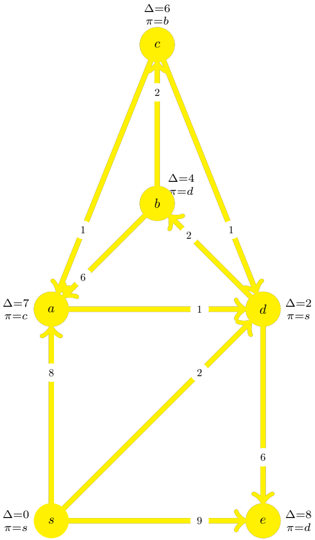
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



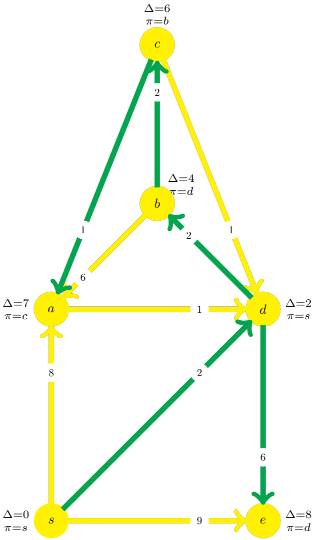
- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

A shortest path example: Dijkstra's algorithm



- We start assigning $\Delta(s) = 0$ and $\pi(s) = s$ to the source node.
- Next we try all outgoing arcs from s and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- Next we find the smallest Δ value among non-yellow nodes.
- We try again all outgoing arcs and update labels ...
- ...
- The π labels describe a shortest path tree rooted at s, and the Δ labels are the lengths of the shortest paths.

Dijkstra's shortest path algorithm (1956,59)

procedure DIJKSTRA'S ALGORITHM($G = (V, A)$, $\ell : A \to \mathbb{R}$, $s \in V$)
  Set $\Delta(v) = +\infty$ and $\pi(v) = nil$ for all $v \neq s$.
  Set $\Delta(s) = 0$, $\pi(s) = s$, $S = \emptyset$, and $L = \{s\}$.
  while $L \neq \emptyset$ **do**
    Let $u \in L$ be a vertex with the smallest $\Delta(u)$ value; $L = L \setminus \{u\}$.
    for $v \in N^+(u)$ **do**
      if $\Delta(v) = +\infty$ **then**
        Set $L = L \cup \{v\}$.
        Set $\Delta(v) = \Delta(u) + \ell(u, v)$ and $\pi(v) = u$.
      else
        if $\Delta(v) > \Delta(u) + \ell(u, v)$ **then**
          Set $\Delta(v) = \Delta(u) + \ell(u, v)$ and $\pi(v) = u$.
        end if
      end if
    end for
    Set $S = S \cup \{u\}$.
  end while
  return $\Delta(v)$ and $\pi(v)$ for $v \in V$.
end procedure

Dijkstra's shortest path algorithm (1956,59)

Theorem 1

Dijkstra's algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$ upon termination.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.

- At any given time during the algorithm $\Delta(v)$ is the length of a minimum length $s \to v$ path, if

$$\Delta(v) \leq \min_{u \in L} \Delta(u).$$

- Disjkstra's algorithm works as long as the directed graph is **free of negative cycles**!
- Finding the shortest $s \to v$ path in a graph that has negative cycles is **NP-hard**!
- Dynamic programming principle (recursion)

$$\rho(s, v) \; = \; \min_{w \in N^-(v)} \rho(s, w) + \ell(w, v).$$

- This is a proper recursion for **acyclic** graphs! Otherwise we need Dijkstra's "**trick**:" choose the smallest Δ value ...

Dijkstra's shortest path algorithm (1956,59)

Theorem 1

Dijkstra's algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$ upon termination.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.

- At any given time during the algorithm $\Delta(v)$ is the length of a minimum length $s \to v$ path, if

$$\Delta(v) \leq \min_{u \in L} \Delta(u).$$

- Disjkstra's algorithm works as long as the directed graph is **free of negative cycles**!
- Finding the shortest $s \to v$ path in a graph that has negative cycles is **NP-hard**!
- Dynamic programming principle (recursion)

$$\rho(s, v) \;=\; \min_{w \in N^-(v)} \rho(s, w) + \ell(w, v).$$

- This is a proper recursion for **acyclic** graphs! Otherwise we need Dijkstra's "**trick**:" choose the smallest Δ value ...

Dijkstra's shortest path algorithm (1956,59)

Theorem 1

Dijkstra's algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$ upon termination.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.

- At any given time during the algorithm $\Delta(v)$ is the length of a minimum length $s \to v$ path, if

$$\Delta(v) \leq \min_{u \in L} \Delta(u).$$

- Disjkstra's algorithm works as long as the directed graph is **free of negative cycles**!
- Finding the shortest $s \to v$ path in a graph that has negative cycles is **NP-hard**!
- Dynamic programming principle (recursion)

$$\rho(s, v) = \min_{w \in N^-(v)} \rho(s, w) + \ell(w, v).$$

- This is a proper recursion for **acyclic** graphs! Otherwise we need Dijkstra's "**trick**:" choose the smallest Δ value ...

Dijkstra's shortest path algorithm (1956,59)

Theorem 1

Dijkstra's algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$ upon termination.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.

- At any given time during the algorithm $\Delta(v)$ is the length of a minimum length $s \to v$ path, if
$$\Delta(v) \leq \min_{u \in L} \Delta(u).$$
- Disjkstra's algorithm works as long as the directed graph is **free of negative cycles**!
- Finding the shortest $s \to v$ path in a graph that has negative cycles is **NP-hard**!
- Dynamic programming principle (recursion)
$$\rho(s, v) \; = \; \min_{w \in N^-(v)} \rho(s, w) + \ell(w, v).$$
- This is a proper recursion for **acyclic** graphs! Otherwise we need Dijkstra's "**trick**:" choose the smallest Δ value ...

Dijkstra's shortest path algorithm (1956,59)

Theorem 1

Dijkstra's algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$ upon termination.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.

- At any given time during the algorithm $\Delta(v)$ is the length of a minimum length $s \to v$ path, if

$$\Delta(v) \le \min_{u \in L} \Delta(u).$$

- Disjkstra's algorithm works as long as the directed graph is **free of negative cycles**!
- Finding the shortest $s \to v$ path in a graph that has negative cycles is **NP-hard**!
- Dynamic programming principle (recursion)

$$\rho(s, v) \;=\; \min_{w \in N^-(v)} \rho(s, w) + \ell(w, v).$$

- This is a proper recursion for **acyclic** graphs! Otherwise we need Dijkstra's "**trick**:" choose the smallest Δ value ...

Dijkstra's shortest path algorithm (1956,59)

Theorem 1

Dijkstra's algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$ upon termination.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.

- At any given time during the algorithm $\Delta(v)$ is the length of a minimum length $s \to v$ path, if

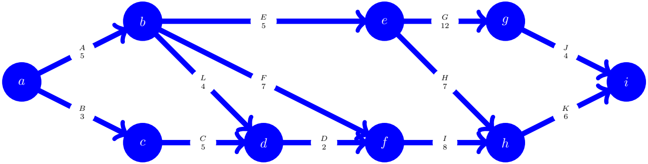
$$\Delta(v) \leq \min_{u \in L} \Delta(u).$$

- Disjkstra's algorithm works as long as the directed graph is **free of negative cycles**!
- Finding the shortest $s \to v$ path in a graph that has negative cycles is **NP-hard**!
- Dynamic programming principle (recursion)

$$\rho(s, v) \;=\; \min_{w \in N^-(v)} \rho(s, w) + \ell(w, v).$$

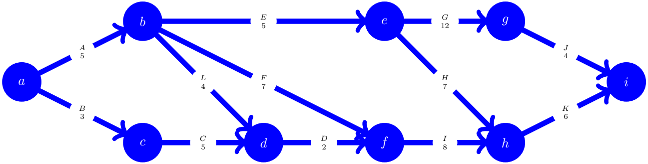
- This is a proper recursion for **acyclic** graphs! Otherwise we need Dijkstra's "**trick**:" choose the smallest Δ value ...

PERT/CPM (US Navy, 1957)



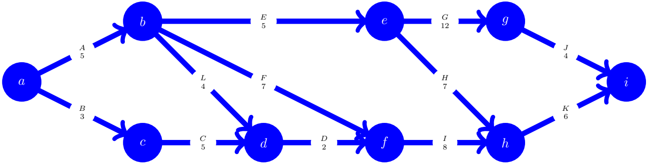
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



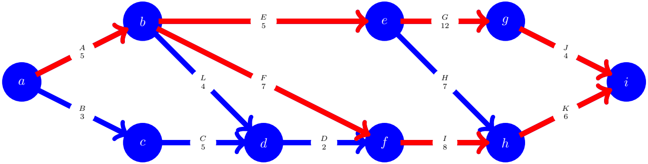
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



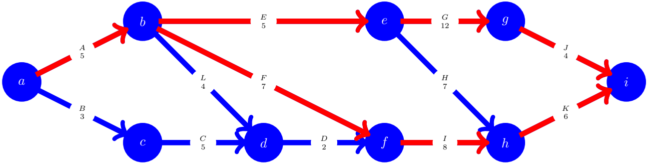
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



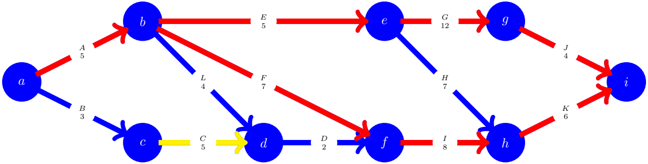
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



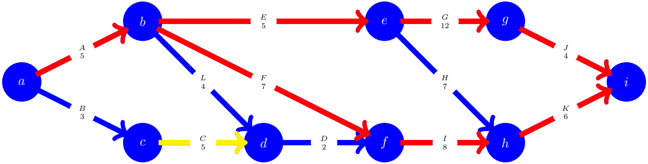
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



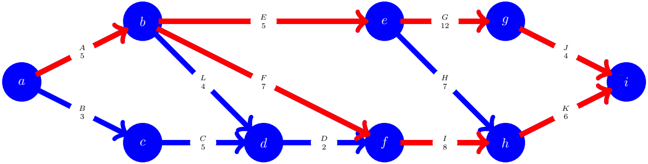
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:

PERT/CPM (US Navy, 1957)



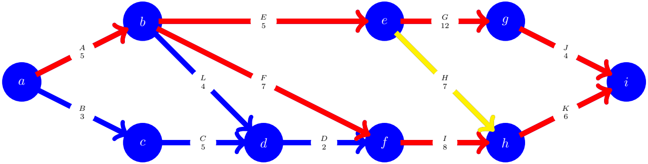
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



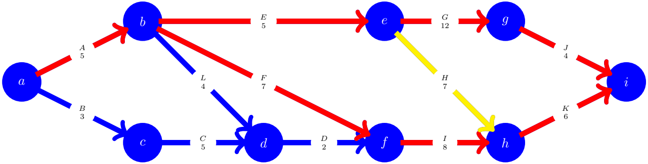
- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

PERT/CPM (US Navy, 1957)



- Program evaluation and review technique (1957)
- Critical path method
- Longest (critical) paths: $\Delta = 26$
- Task C:
  - Earliest start time: 3
  - Latest finish time: 10
- Task H:
  - Earliest start time: 10
  - Latest finish time: 20

Minimum mean cycle problem (Karp, 1973)

- Given a directed graph $G = (V, A)$ and $\ell : A \mapsto \mathbb{R}_+$, for a directed cycle $C \subseteq A$ let

$$\mu(C) \;=\; \frac{\displaystyle\sum_{e \in C} \ell(e)}{|C|}$$

- For $s \in V$ let $d_{=k}(s, v)$ denote the shortest $s \to v$ path using exactly k arcs.
- $d_{=k}(s, v)$ can be computed efficiently by dynamic programming:

$$d_{=k}(s, v) \;=\; \min_{u \in N^-(v)} d_{=k-1}(s, u) + \ell(u, v),$$

  for $k \geq 2$, where $N^-(v) = \{u \in V \mid (u, v) \in A\}$.
-

$$d_{=1}(s, v) = \begin{cases} \ell(s, v) & \text{if } (s, v) \in A, \\ \infty & \text{otherwise.} \end{cases}$$

Minimum mean cycle problem (Karp, 1973)

- Given a directed graph $G = (V, A)$ and $\ell : A \mapsto \mathbb{R}_+$, for a directed cycle $C \subseteq A$ let

$$\mu(C) = \frac{\sum\limits_{e \in C} \ell(e)}{|C|}$$

- For $s \in V$ let $d_{=k}(s, v)$ denote the shortest $s \to v$ path using exactly k arcs.
- $d_{=k}(s, v)$ can be computed efficiently by dynamic programming:

$$d_{=k}(s, v) = \min_{u \in N^-(v)} d_{=k-1}(s, u) + \ell(u, v),$$

for $k \geq 2$, where $N^-(v) = \{u \in V \mid (u, v) \in A\}$.

-

$$d_{=1}(s, v) = \begin{cases} \ell(s, v) & \textbf{if } (s, v) \in A, \\ \infty & \textbf{otherwise.} \end{cases}$$

Minimum mean cycle problem (Karp, 1973)

- Given a directed graph $G = (V, A)$ and $\ell : A \mapsto \mathbb{R}_+$, for a directed cycle $C \subseteq A$ let

$$\mu(C) = \frac{\sum_{e \in C} \ell(e)}{|C|}$$

- For $s \in V$ let $d_{=k}(s, v)$ denote the shortest $s \to v$ path using exactly k arcs.
- $d_{=k}(s, v)$ can be computed efficiently by dynamic programming:

$$d_{=k}(s, v) = \min_{u \in N^-(v)} d_{=k-1}(s, u) + \ell(u, v),$$

  for $k \geq 2$, where $N^-(v) = \{u \in V \mid (u, v) \in A\}$.

- $$d_{=1}(s, v) = \begin{cases} \ell(s, v) & \textbf{if } (s, v) \in A, \\ \infty & \textbf{otherwise.} \end{cases}$$

Minimum mean cycle problem (Karp, 1973)

- Given a directed graph $G = (V, A)$ and $\ell : A \mapsto \mathbb{R}_+$, for a directed cycle $C \subseteq A$ let

$$\mu(C) = \frac{\sum_{e \in C} \ell(e)}{|C|}$$

- For $s \in V$ let $d_{=k}(s, v)$ denote the shortest $s \to v$ path using exactly k arcs.
- $d_{=k}(s, v)$ can be computed efficiently by dynamic programming:

$$d_{=k}(s, v) = \min_{u \in N^-(v)} d_{=k-1}(s, u) + \ell(u, v),$$

  for $k \geq 2$, where $N^-(v) = \{u \in V \mid (u, v) \in A\}$.
-

$$d_{=1}(s, v) = \begin{cases} \ell(s, v) & \textbf{if } (s, v) \in A, \\ \infty & \textbf{otherwise.} \end{cases}$$

Minimum mean cycle problem cont'd

- **Theorem** (Karp, 1973):

$$\mu(G) = \min_{C \textbf{ is a cycle in } G} \mu(C) = \min_{v \in V} \max_{1 \le k \le n-1} \frac{d_{=n}(s,v) - d_{=k}(s,v)}{n-k},$$

where $s \in V$ is an arbitrary fixed vertex.

- $\mu(G)$ can be computed in $O(|V||A|)$ time (typically $|A| \ge |V|$.)

Minimum mean cycle problem cont'd

- **Theorem** (Karp, 1973):

$$\mu(G) = \min_{C \text{ is a cycle in } G} \mu(C) = \min_{v \in V} \max_{1 \le k \le n-1} \frac{d_{=n}(s,v) - d_{=k}(s,v)}{n-k},$$

where $s \in V$ is an arbitrary fixed vertex.
- $\mu(G)$ can be computed in $O(|V||A|)$ time (typically $|A| \geq |V|$.)