Algorithms Design
Chap04-Greedy Algorithms

College of Computer Science

Nankai University

Tianjin, P.R.China

Chap04-Greedy Algorithms Outline

4.1 Interval Scheduling and Interval Partitioning

4.2 Scheduling to Minimize Lateness

4.3 Optimal Caching

4.4 Shortest Paths in a Graph

4.5 Minimum Spanning Tree

4.7 Clustering

4.8 Huffman Codes

4.4 Shortest Paths in a Graph

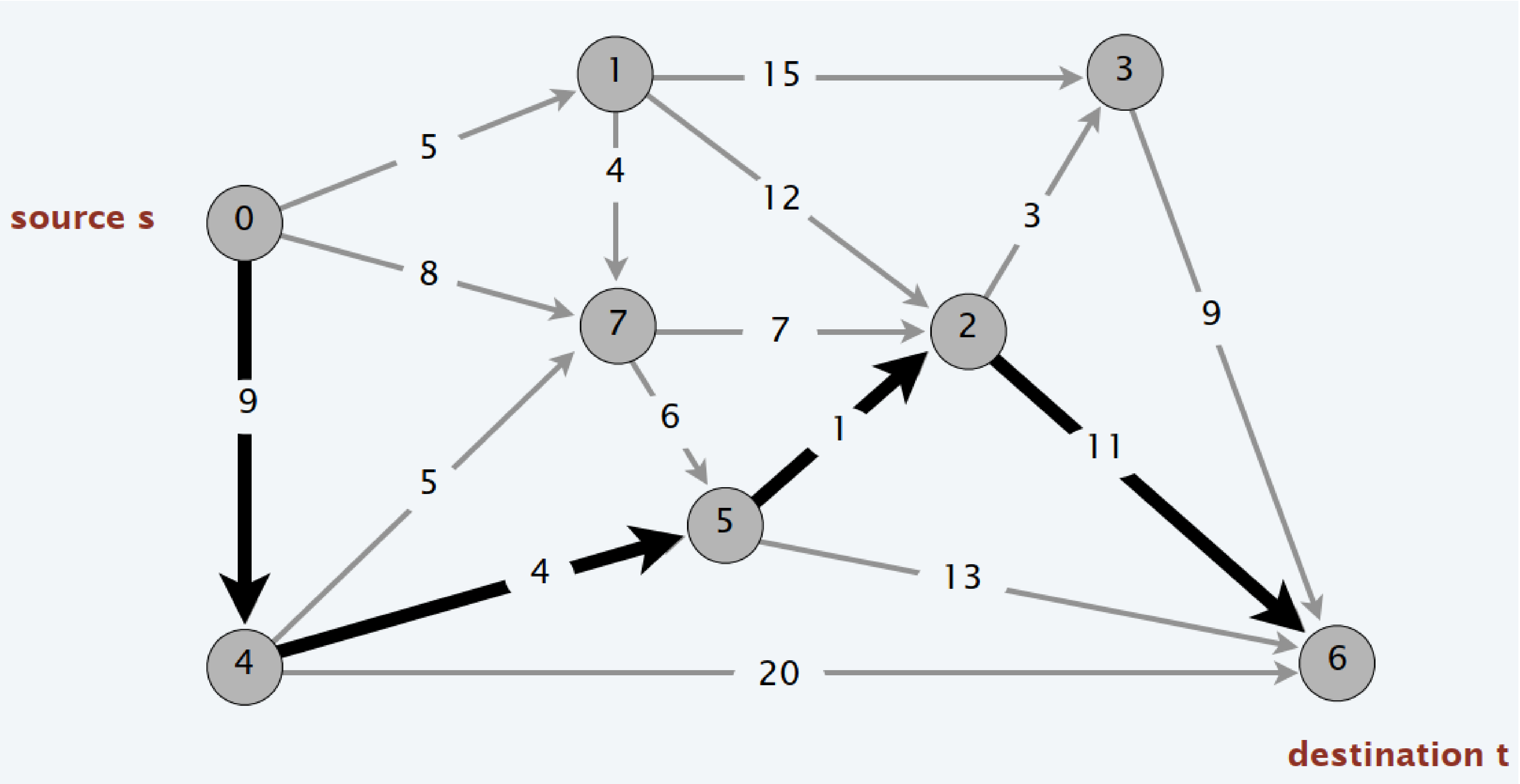
Shortest path network.

- Directed graph $G = (V, E)$.
- Source $s \in V$, destination $t \in V$.
- Length l_e = length of edge e.($l_e \geq 0$)

Goal

- find a shortest path from s to t.
- Cost of path = sum of edge costs in path

4.4 Shortest Paths in a Graph



4.4 Shortest Paths in a Graph

Suppose that you change the length of every edge of G as follows. For which is every shortest path in G a shortest path in G'?

A. Add 17.

B. Multiply by 17.

C. Either A or B.

D. Neither A nor B

4.4 Shortest Paths in a Graph

Shortest path applications

- Map routing.
- Robot navigation.
- Texture mapping.
- Urban traffic planning.
- Network routing protocols (OSPF, BGP, RIP).
- …

4.4 Shortest Paths in a Graph

Which variant in car GPS?

A. Single source: from one node s to every other node.

B. Single target: from every node to one node t.

C. Source–target: from one node s to another node t.

D. All pairs: between all pairs of nodes.

4.4 Shortest Paths in a Graph

Dijkstra′s algorithm

- single-source shortest paths problem

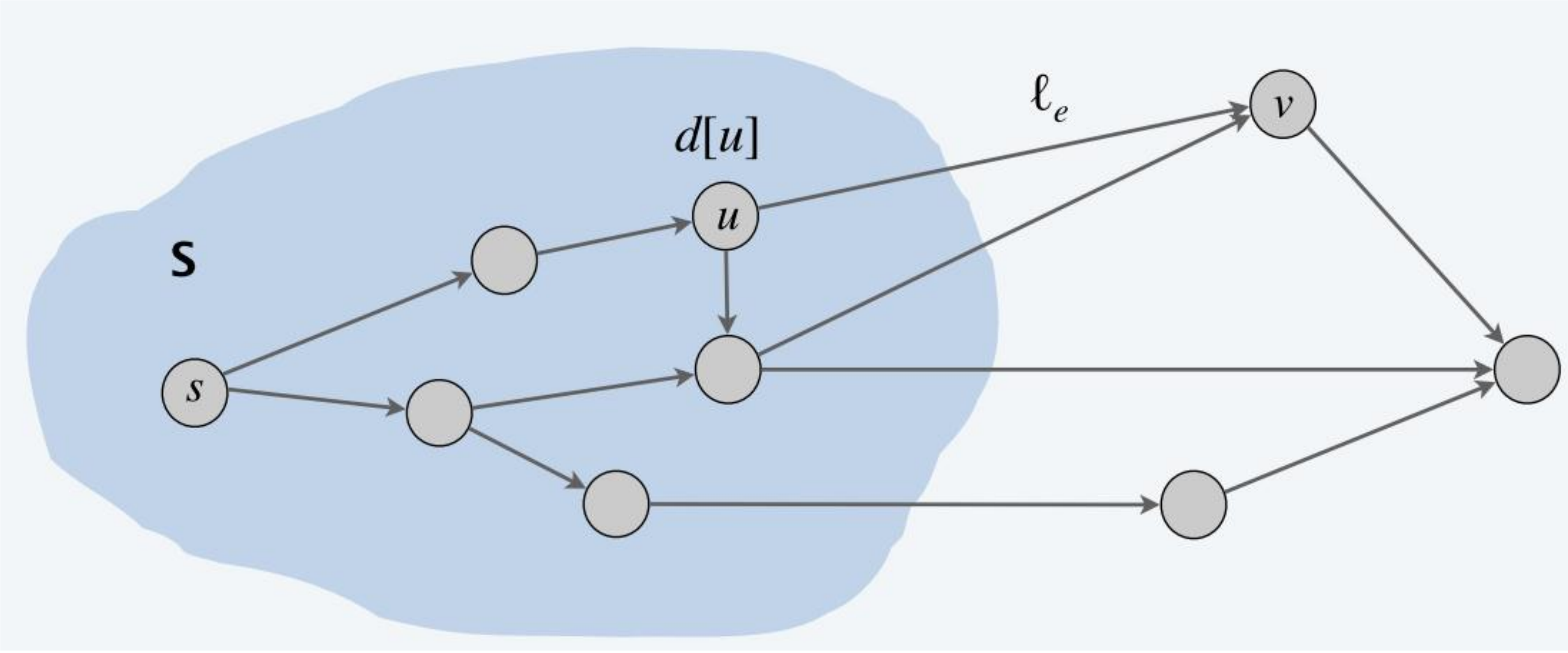
Greedy approach

- Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d[u]$ from s to u.
- Initialize $S = \{s\}$, $d[s] = 0$.

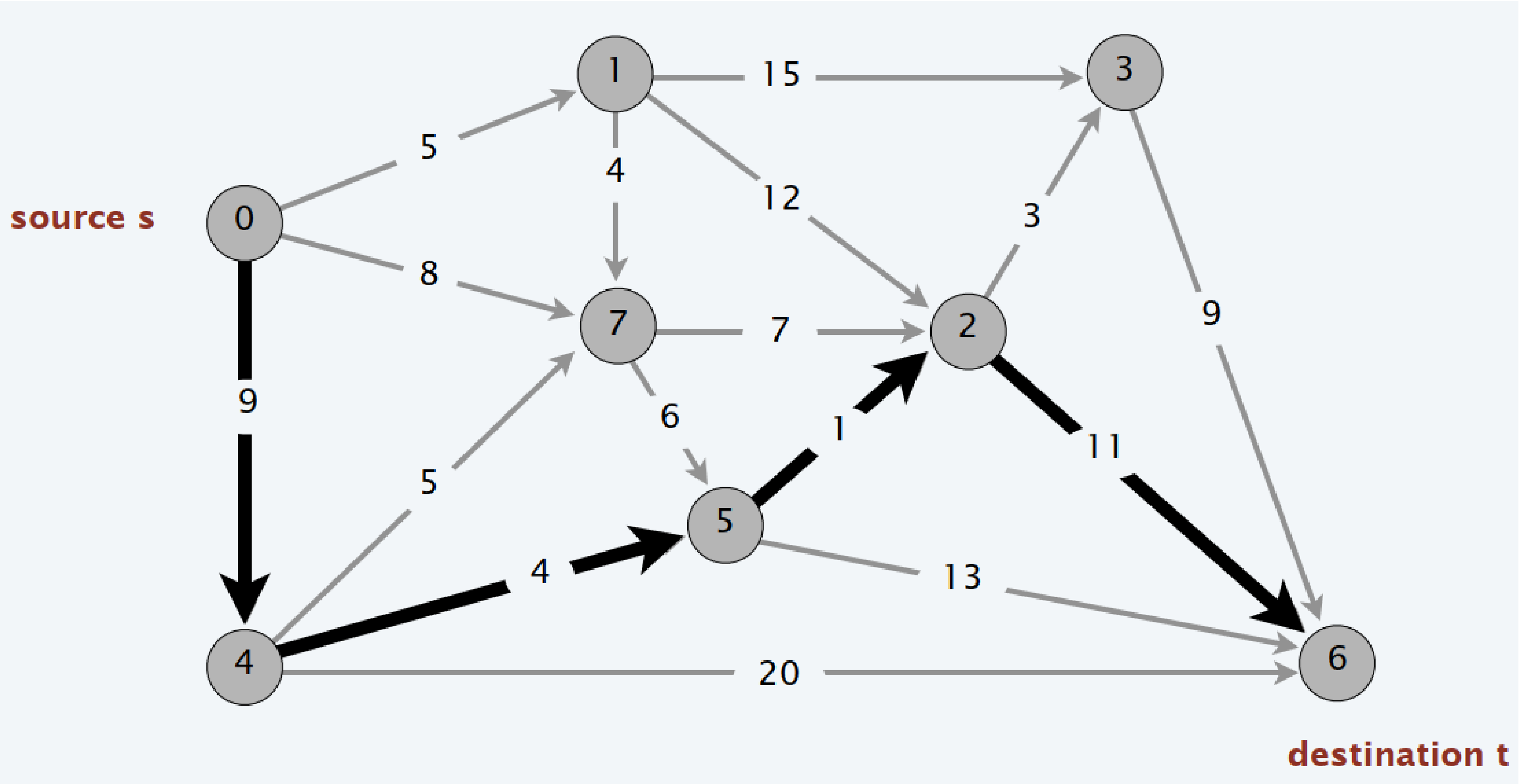
4.4 Shortest Paths in a Graph

- [continue]
- Repeatedly choose unexplored node $v \notin S$ which minimizes
$$\pi(v) = \min_{e=(u,v):u \in S} d[u] + l_e$$
Add v to S, and set $d[v] = \pi(v)$
- To recover path, set pred[v] ← e that achieves min.

4.4 Shortest Paths in a Graph



4.4 Shortest Paths in a Graph



4.4 Shortest Paths in a Graph

$S=\{0\}, d[0]=0$

$\pi[1]=5$, $\pi[7]= 8$, $\pi[4]=9$, $S=\{0,1\}$, $d[1]=5$, $pred[1]=0$

$\pi[7]=\min(8,9)$, $\pi[3]=15+5=20$, $\pi[2]= 17$, $|\pi[4]=9$, $S=\{0,1,7\}$, $d[7]=8$, $pred[7]=0$

$\pi[5]= 8+6=14$, $\pi[2]=\min(8+7, 5+12)$, $|\pi[4]=9$, $\pi[3]=20$, $S=\{0,1,4,7\}$, $d[4]=9$, $pred[4]=0$

$\pi[5]=\min(9+4,14)$, $\pi[6] = 9+20 = 29$, $|\pi[2]=15$, $\pi[3]=20$, $S=\{0,1,4,5,7\}$, $d[5]=13$, $pred[5]=4$

4.4 Shortest Paths in a Graph

$\pi[2]$=min(13+1,8+7), $\pi[6]$=min(13+13,29) ,| $\pi[3]$=20, S={0,1,2,4,5,7}, $d[2]=14$,pred[2]=5

$\pi[3]$=min(20, 14+3), $\pi[6]$=min(26, 14+11) , S={0,1,2,3,4,5,7}, $d[3]=17$, pred[3]=2

$\pi[6]$=min(17+9, 14+11)=25, S={0,1,2,3,4,5,6,7}, d[6]=25, pred[6]=2

4.4 Shortest Paths in a Graph

	V1	V2	V3	V4	V5	V6	V7	S
R1	5=5 (v_0)	∞	∞	9 (v_0)	∞	∞	8 (v_0)	{0,1}
R2		5+12=17 (v_1)	5+15=20 (v_1)	9 (v_0)	∞	∞	min(9,8) =8(v_0)	{0,1,7}
R3		min(15,17) =15(v_7)	20 (v_1)	9=9(v_0)	8+6=14 (v_7)	∞		{0,1,4,7 }
R4		15 (v_7)	20 (v_1)		min(13, 14)=13 (v_4)	20+9=29 (v_4)		{0,1,4,5, 7}
R5		min(14,15) =14(v_5)	20 (v_1)			min(26,29) =26(v_5)		{0,1,2,4, 5,7}
R6			min(17,20)=17(v_2)			min(25,26) =25(v_2)		{0,1,2,3, 4,5,7}
R7						min(26,25) =25(v_2)		{0,1,2,3, 4,5,6,7}

4.4 Shortest Paths in a Graph

Invariant. For each node $u \in S$, $d[u]$ is the length of the shortest s-u path.

Pf. (by induction on $|S|$)

Base case: $|S| = 1$ is easy since $S = \{s\}$ and $d[s] = 0$.

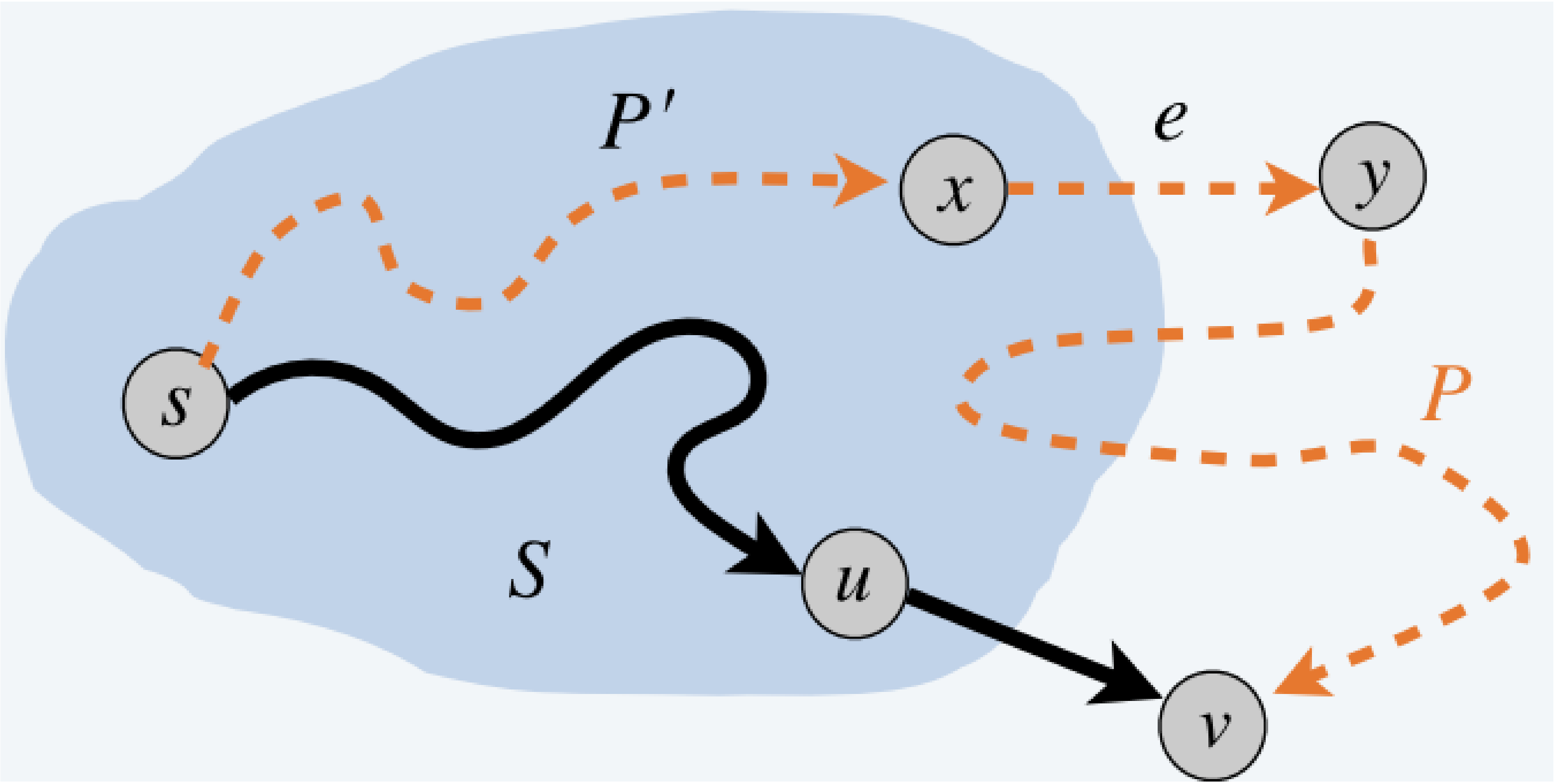
4.4 Shortest Paths in a Graph

Inductive hypothesis:

Assume true for $|S| = k \geq 1$.

- Let v be next node added to S, and let u-v be the chosen edge.
- The shortest s-u path plus (u, v) is an s-v path of length $\pi(v)$.
- Consider any s-v path P. We'll see that it's no shorter than $\pi(v)$.
- Let x-y be the first edge in P that leaves S, and let P' be the subpath to x.
- P is already too long as soon as it leaves S.

4.4 Shortest Paths in a Graph



4.4 Shortest Paths in a Graph

Efficient implementation

- For each unexplored node $v \notin S$: explicitly maintain $\pi[v]$ instead of computing directly from definition
$$\pi(v) = \min_{e=(u,v):u \in S} d[u] + l_e$$
- The set of unexplored nodes can only decrease because set S increases.
- Specifically, it suffices to update:
$$\pi[v] = \min(\pi[v], \pi[u] + l_e)$$

4.4 Shortest Paths in a Graph

Efficient implementation

Use a min-oriented priority queue (PQ) to choose an unexplored node that minimizes $\pi[v]$.

4.4 Shortest Paths in a Graph

Efficient Implementation.

- Algorithm maintains $\pi[v]$ for each node v.
- Priority Queue(PQ) stores unexplored nodes, using $\pi[]$ as priorities.
- Once u is deleted from the PQ, $\pi[u] =$ length of a shortest s-u path.

DIJKSTRA (V, E, ℓ, s)

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty$, $pred[v] \leftarrow null$; $\pi[s] \leftarrow 0$.

Create an empty priority queue pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$.

WHILE (IS-NOT-EMPTY(pq))

    $u \leftarrow$ DEL-MIN(pq).

    FOREACH edge $e = (u, v) \in E$ leaving u:

        IF $(\pi[v] > \pi[u] + \ell_e)$

            DECREASE-KEY$(pq, v, \pi[u] + \ell_e)$.

            $\pi[v] \leftarrow \pi[u] + \ell_e$; $pred[v] \leftarrow e$.

4.4 Shortest Paths in a Graph

Performance. n INSERT, n DELETE-MIN, $\leq m$ DECREASE-KEY.

priority queue	INSERT	DELETE-MIN	DECREASE-KEY	total
node-indexed array (A[i] = priority of i)	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman-Tarjan 1984)	$O(1)$	$O(\log n)^{\dagger}$	$O(1)^{\dagger}$	$O(m + n \log n)$
integer priority queue (Thorup 2004)	$O(1)$	$O(\log \log n)$	$O(1)$	$O(m + n \log \log n)$

4.4 Shortest Paths in a Graph

Dijkstra's algorithm and proof extend to several related problems

- Shortest paths in undirected graphs: $\pi[v] \leq \pi[u] + \ell(u, v)$.
- Maximum capacity paths: $\pi[v] \geq \min\big(\pi[u], c(u, v)\big)$.
- Maximum reliability paths: $\pi[v] \geq \pi[u] \times \gamma(u, v)$.
- …

Chap04-Greedy Algorithms Outline

4.1 Interval Scheduling and Interval Partitioning

4.2 Scheduling to Minimize Lateness

4.3 Optimal Caching

4.4 Shortest Paths in a Graph

4.5 Minimum Spanning Tree

4.7 Clustering

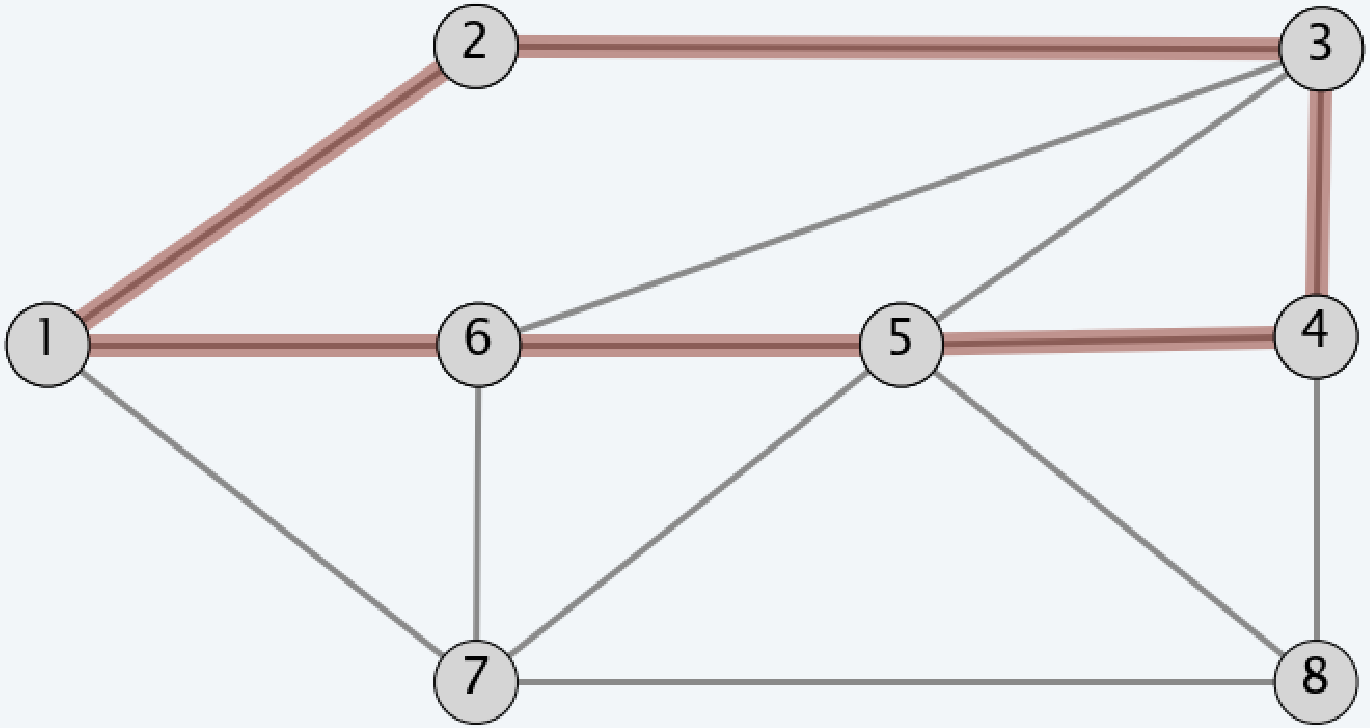
4.8 Huffman Codes

4.5 Minimum Spanning Tree

Def.(4.5.2) A path is a sequence of edges which connects a sequence of nodes.

Def.(4.5.3) A cycle is a path with no repeated nodes or edges other than the starting and ending nodes.

4.5 Minimum Spanning Tree



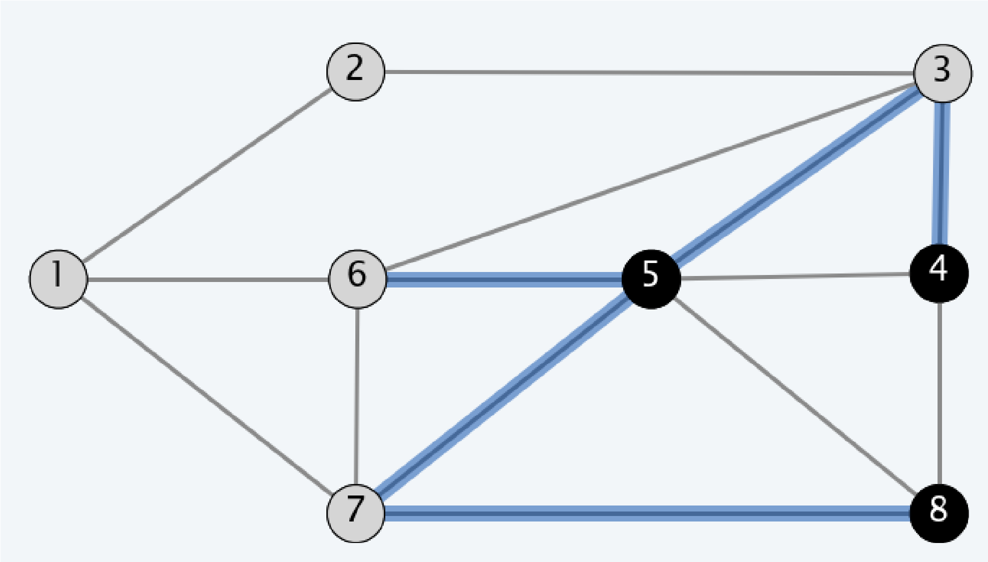
path P = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6) }

cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

4.5 Minimum Spanning Tree

Def. A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

Def. The cutset of a cut S is the set of edges with exactly one endpoint in S.



Quiz(4.5.1)

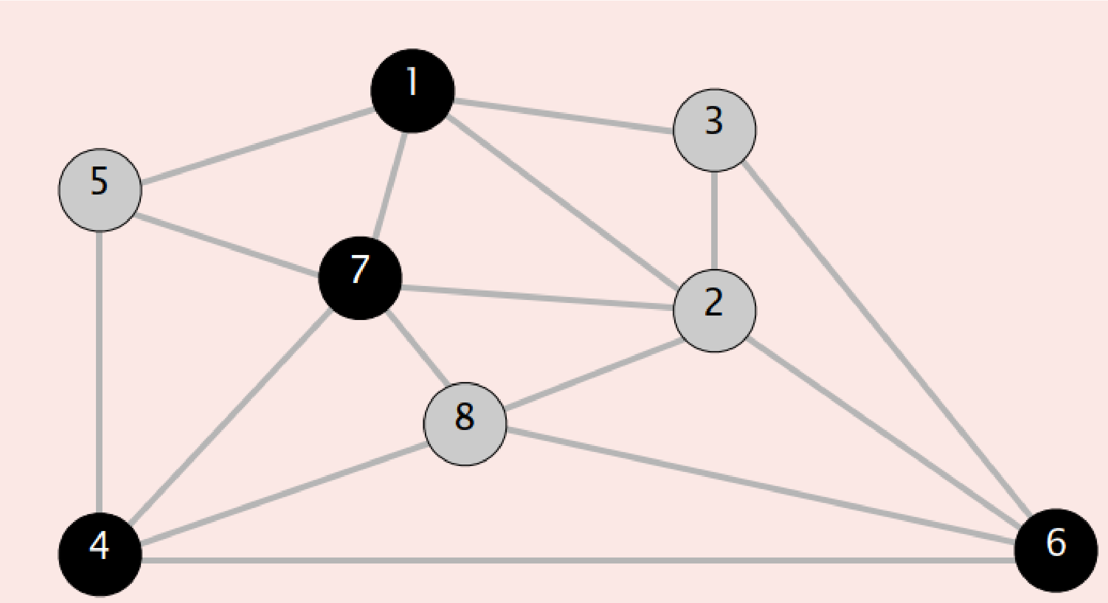
Consider the cut S = { 1, 4, 6, 7 }.
Which edge is in the cutset of S?

A. S is not a cut (not connected)

B. 1–7

C. 5–7

D. 2–3



Quiz(4.5.2)

Let C be a cycle and let D be a cutset. How many edges do C and D have in common? Choose the best answer.

A. 0

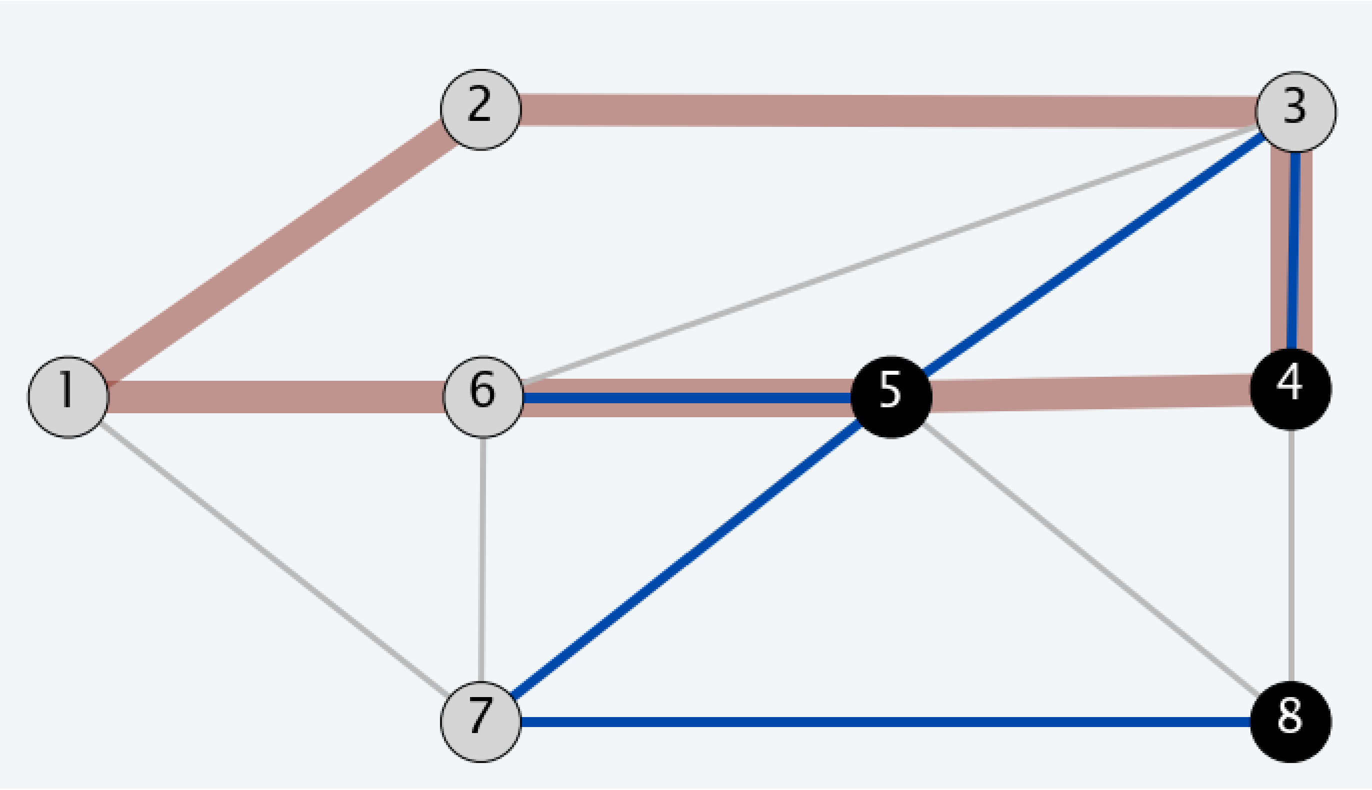
B. 2

C. not 1

D. an even number

4.5 Minimum Spanning Tree

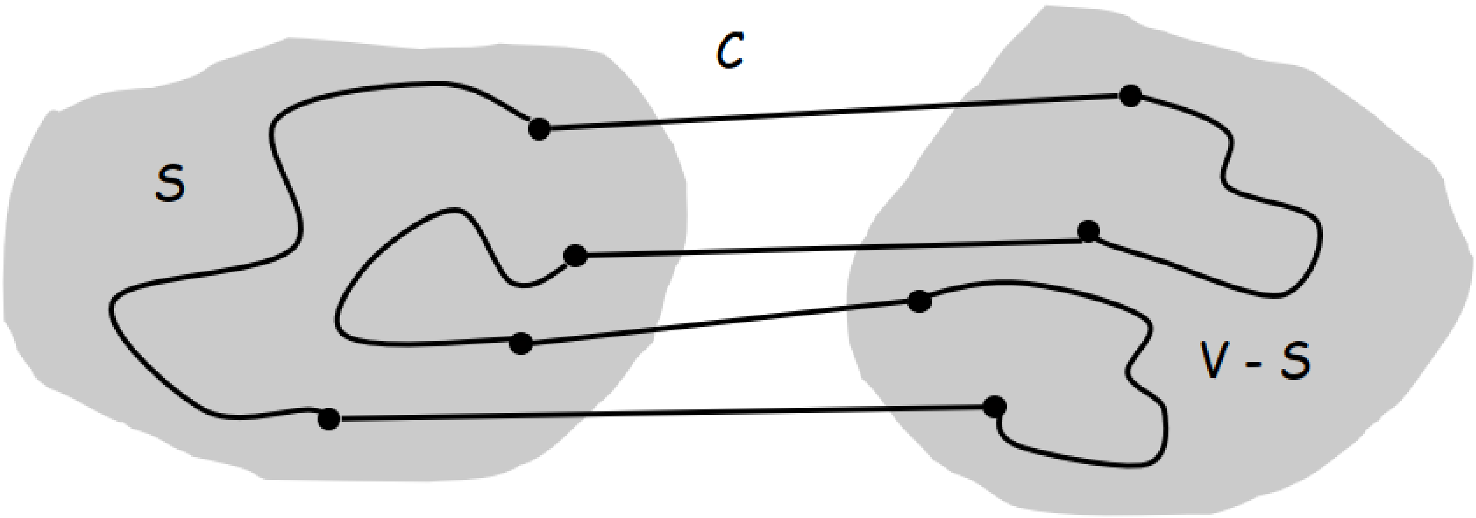
Proposition. A cycle and a cutset intersect in an even number of edges.



4.5 Minimum Spanning Tree

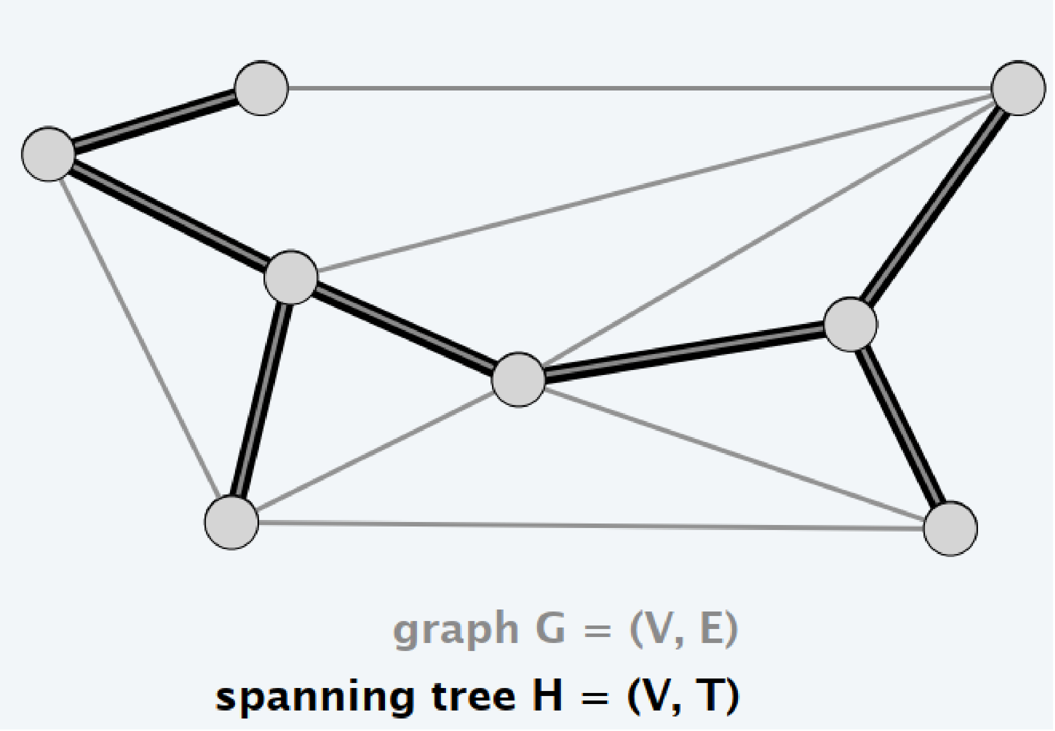
Proposition. A cycle and a cutset intersect in an even number of edges.

Pf. [by picture]



4.5 Minimum Spanning Tree

Def. Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$. H is a **spanning tree** of G if H is both acyclic and connected.



4.5 Minimum Spanning Tree

Proposition.

Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$.

Then, the following are equivalent:

- H is a spanning tree of G.
- H is acyclic and connected.
- H is connected and has $|V| - 1$ edges.
- H is acyclic and has $|V| - 1$ edges.
- H is minimally connected: removal of any edge disconnects it.
- H is maximally acyclic: addition of any edge creates a cycle.

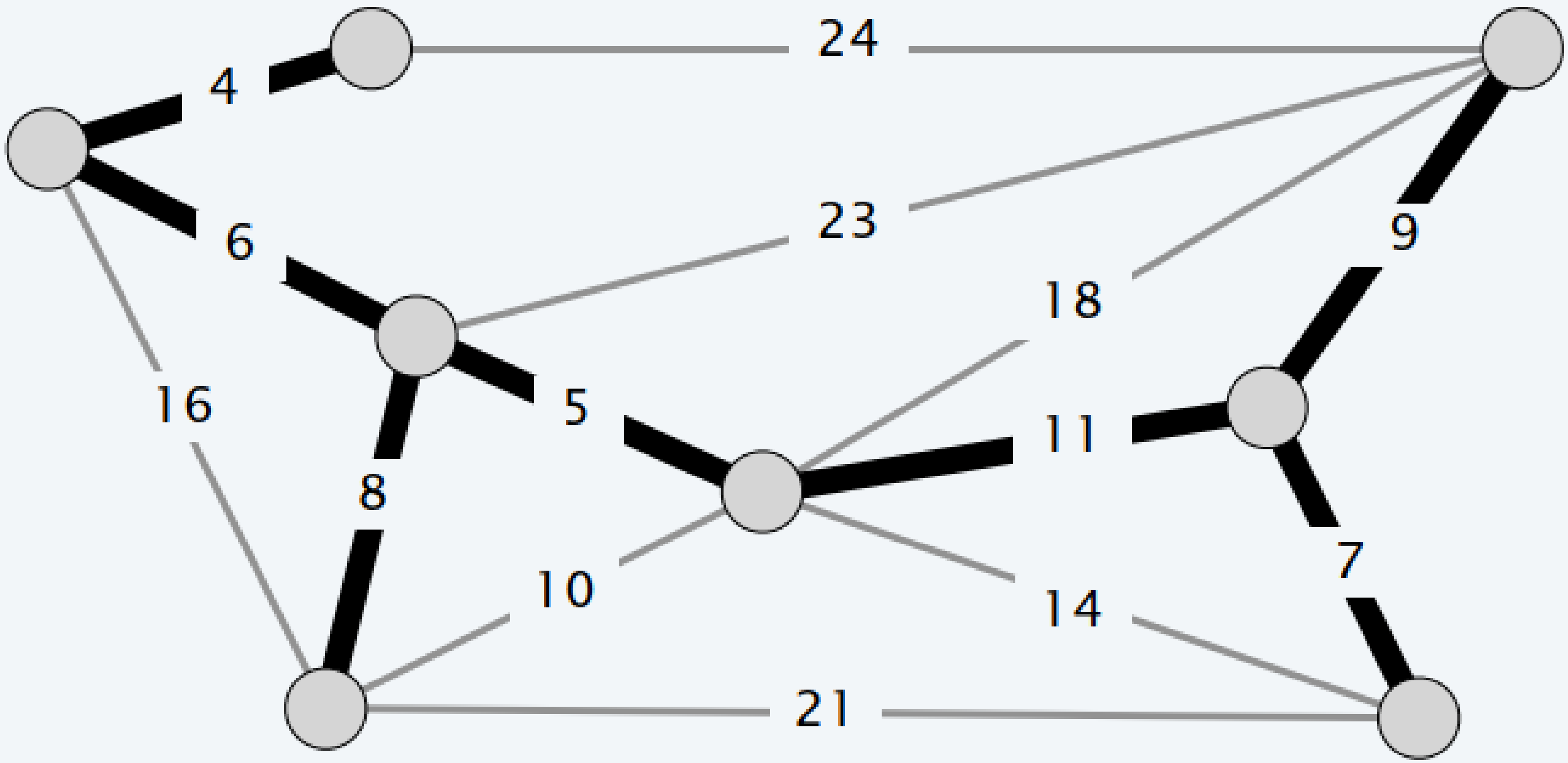
4.5 Minimum Spanning Tree

Minimum Spanning Tree(MST).

Def. Given a connected, undirected graph $G = (V, E)$ with edge weights c_e, a MST is a spanning tree of G such that the sum of edge weights of tree is minimized.

- Note that the set of edges of a MST is a subset of the edges E.

4.5 Minimum Spanning Tree



MST cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

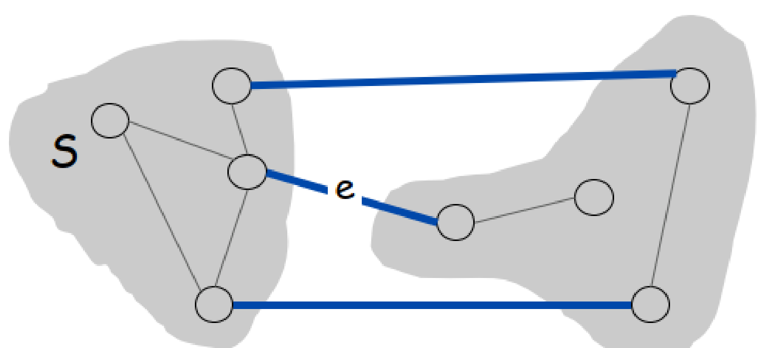
4.5 Minimum Spanning Tree

Simplifying assumption.

- All edge costs c_e are distinct.

Cut property.

- Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.



e is in the MST

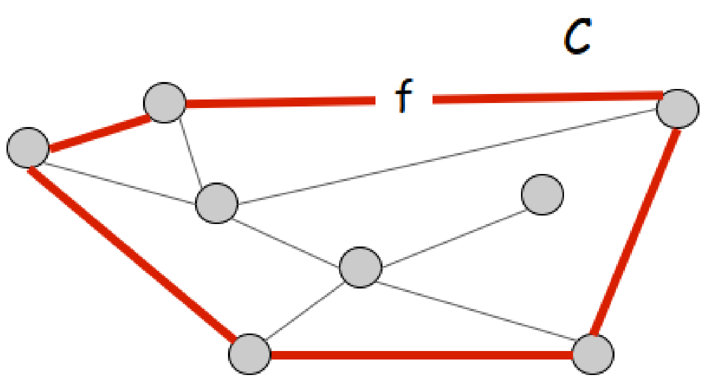
4.5 Minimum Spanning Tree

Simplifying assumption.

- All edge costs c_e are distinct.

Cycle property.

- Let C be any cycle, and let f be the max cost edge belonging to C. Then the MST does not contain f.



f is not in the MST

4.5 Minimum Spanning Tree

Cut property. Let S be any subset of nodes, and let e be the min cost edge with exactly one endpoint in S. Then the MST contains e.

Pf.

- Suppose e does not belong to T^*, and let's see what happens.
- Adding e to T^* creates a cycle C in T^*.
- Edge e is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say f, that is in both C and D.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, cost(T') < cost(T^*).
- This is a contradiction.

4.5 Minimum Spanning Tree

Cycle property. Let C be any cycle, and let f be the max cost edge belonging to C. Then the MST does not contain f.

Pf.

- Suppose f belongs toT^*, and let's see what happens.
- Deleting f from T^* creates a cut S in T^*.
- Edge f is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say e, that is in both C and D.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, cost(T') < cost(T^*).
- This is a contradiction.

4.5 Minimum Spanning Tree

Prim's algorithm. [Jarn ík 1930, Dijkstra 1957, Prim 1959]

- Initialize $S = \{s\}$ for any node s, $T = \emptyset$.
- Repeat $n - 1$ times:
  - Add to T a min-cost edge in the cutset corresponding to S.
  - Add the other endpoint to S.

4.5 Minimum Spanning Tree

Theorem. Prim's algorithm can be implemented to run in $O(m \log n)$ time.

Pf. Implementation almost identical to Dijkstra's algorithm.

PRIM (V, E, c)

$S \leftarrow \varnothing,\ T \leftarrow \varnothing.$

$s \leftarrow$ any node in V.

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty, pred[v] \leftarrow null;\ \pi[s] \leftarrow 0.$

Create an empty priority queue pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$.

WHILE (IS-NOT-EMPTY(pq))

  $u \leftarrow$ DEL-MIN(pq).

  $S \leftarrow S \cup \{ u \},\ T \leftarrow T \cup \{ pred[u] \}.$

  FOREACH edge $e = (u, v) \in E$ with $v \notin S$:

    IF $(c_e < \pi[v])$

      DECREASE-KEY(pq, v, c_e).

      $\pi[v] \leftarrow c_e;\ pred[v] \leftarrow e.$

$\pi[v]$ = cost of cheapest known edge between v and S

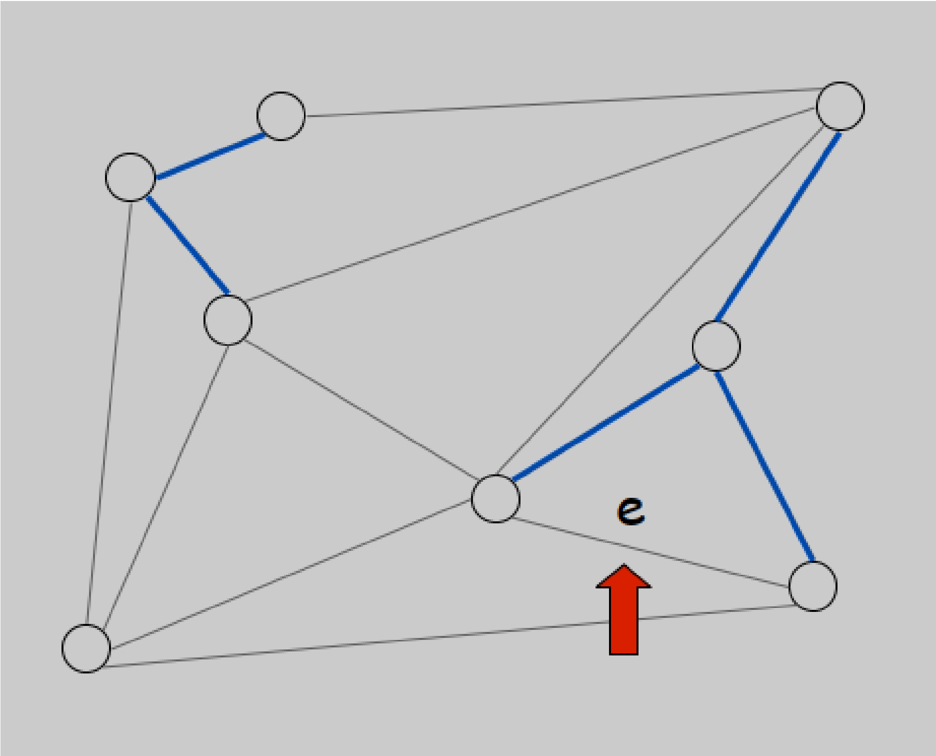
4.5 Minimum Spanning Tree

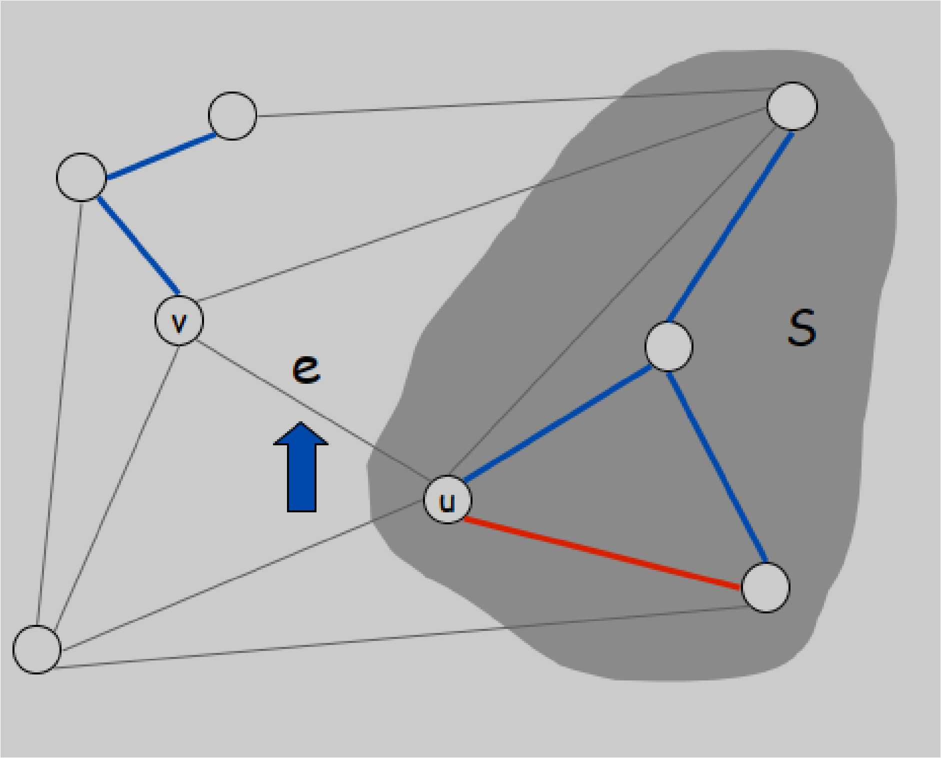
Kruskal's algorithm. [Kruskal, 1956]

- Sort edges in ascending order of cost
- Repeat m times:
  - Select the min-cost edge e so far
  - If adding e to T creates a cycle, discard e according to cycle property.
  - Otherwise, insert $e = (u, v)$ into T according to cut property where S = set of nodes in u's connected component.

4.5 Minimum Spanning Tree

E.g.





4.5 Minimum Spanning Tree

Theorem. Kruskal's algorithm can be implemented to run in $O(m \log m)$ time.

- Sort edges by cost.
- Use union–find data structure to dynamically maintain connected components.

4.5 Minimum Spanning Tree

```
Kruskal(G, c) {
   Sort edges weights so that c₁ ≤ c₂ ≤ ... ≤ cₘ.
   T ← φ

   foreach (u ∈ V) make a set containing singleton u

   for i = 1 to m        are u and v in different connected components?
      (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         merge the sets containing u and v
      }                  merge two components
   return T
}
```

4.5 Minimum Spanning Tree

Reverse-delete algorithm

- Start with all edges in T and consider them in descending order of cost
- Delete edge from T unless it would disconnect T

Thanks for Listening

College of Computer Science
Nankai University
Tianjin, P.R.China