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

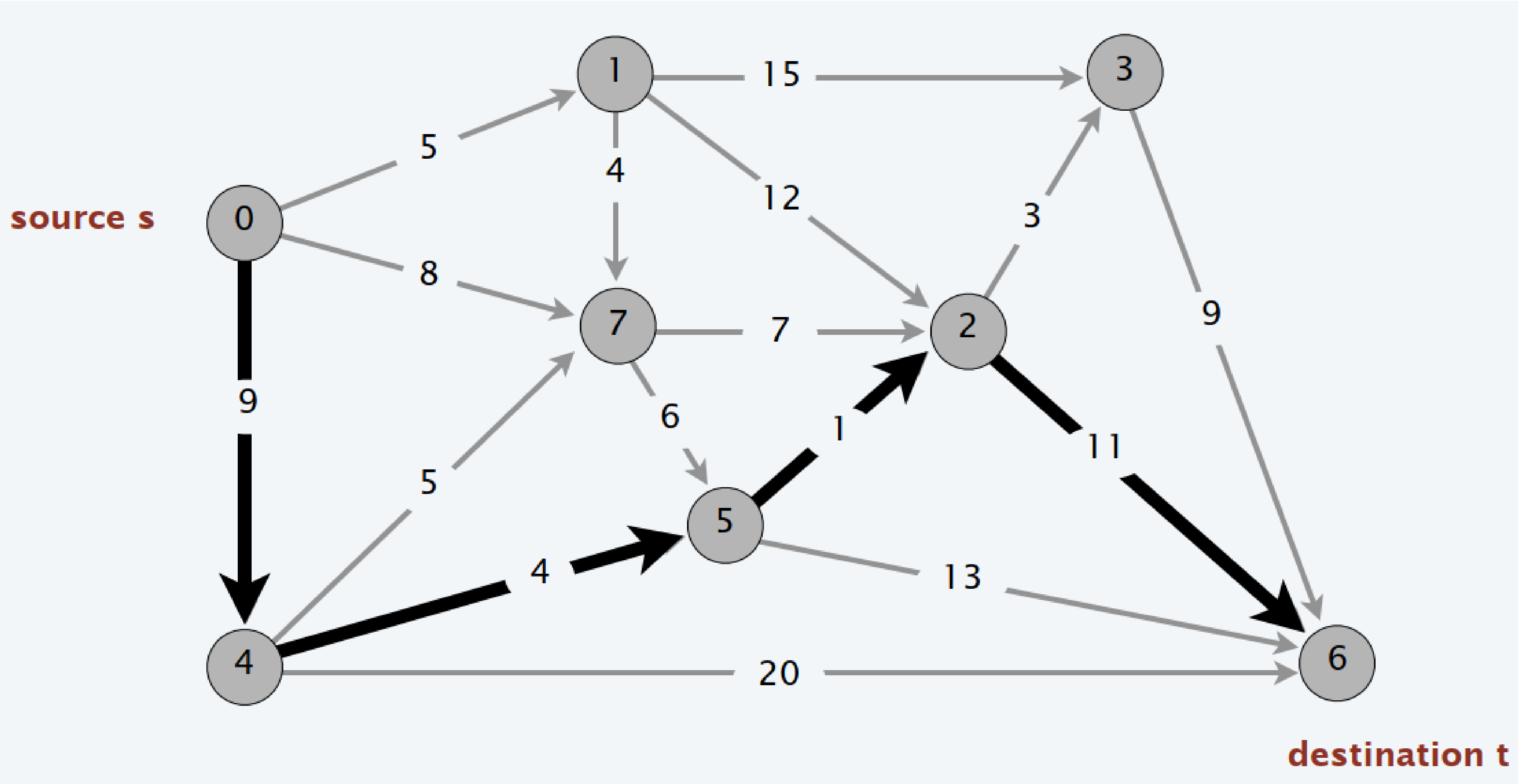
Shortest path network.

- 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

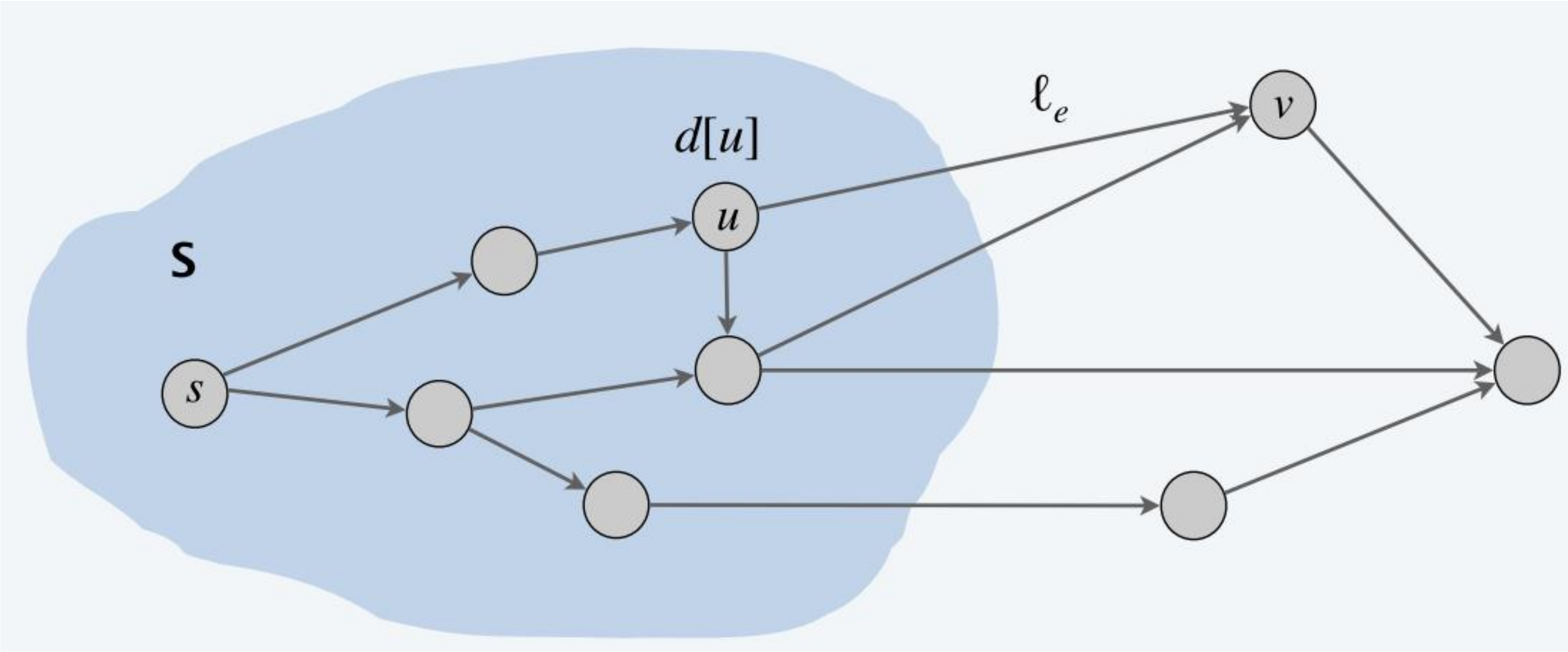
- [continue]
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e=(u,v):u\in S} d[u] + l_e$$

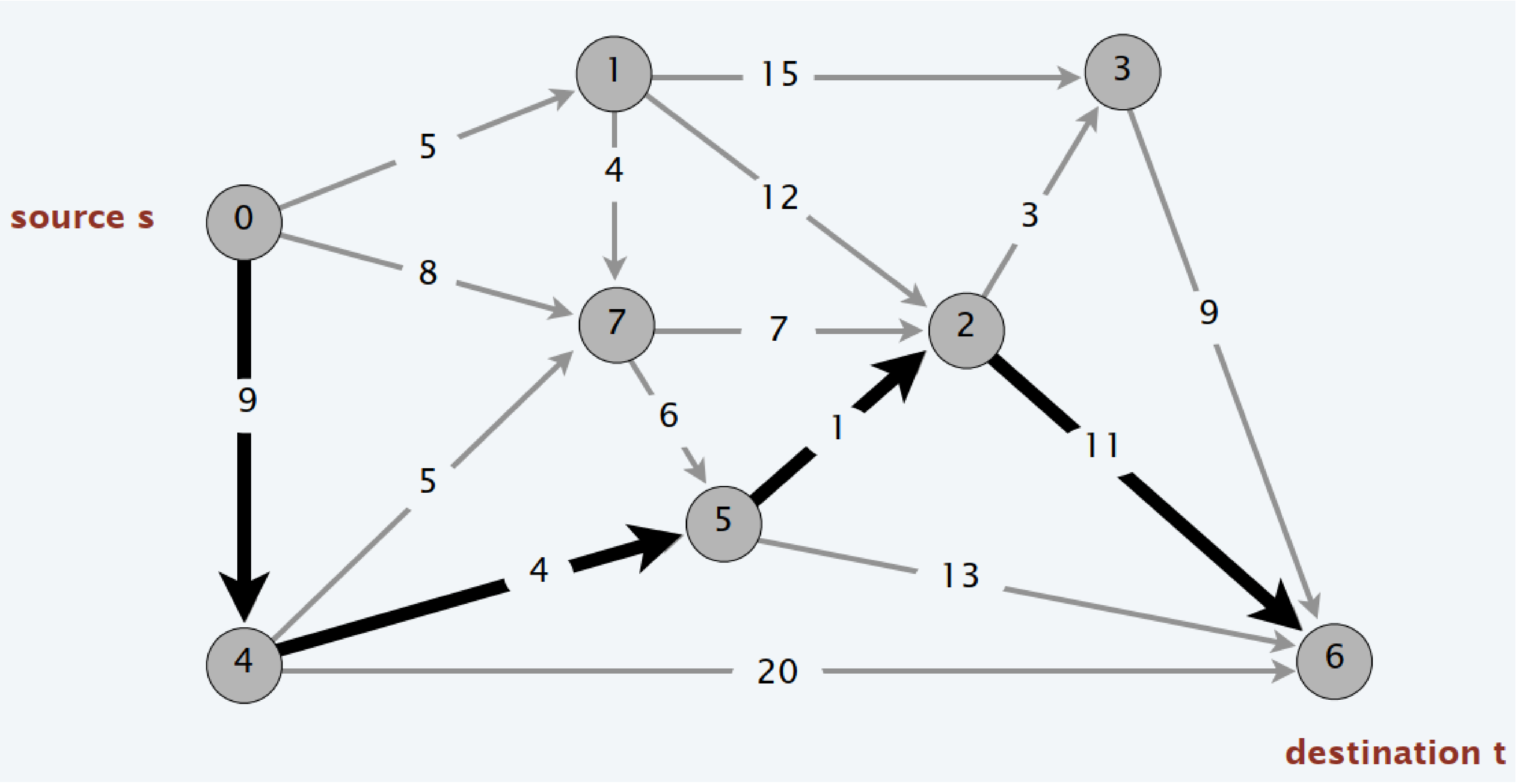
Add v to S, and set $d[v] = \pi(v)$

- 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 $d[s] = 0$.

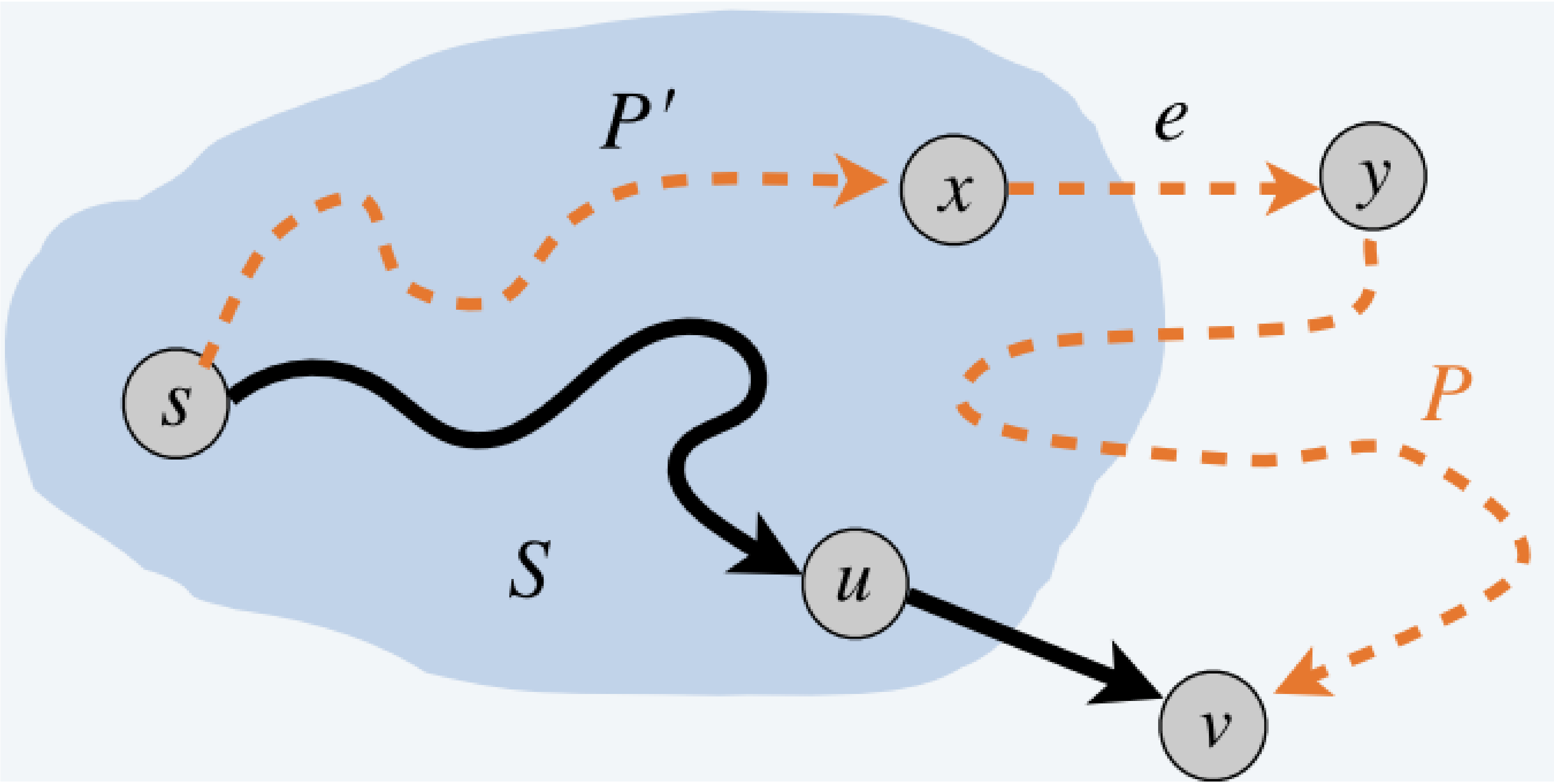
4.4 Shortest Paths in a Graph

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)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
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

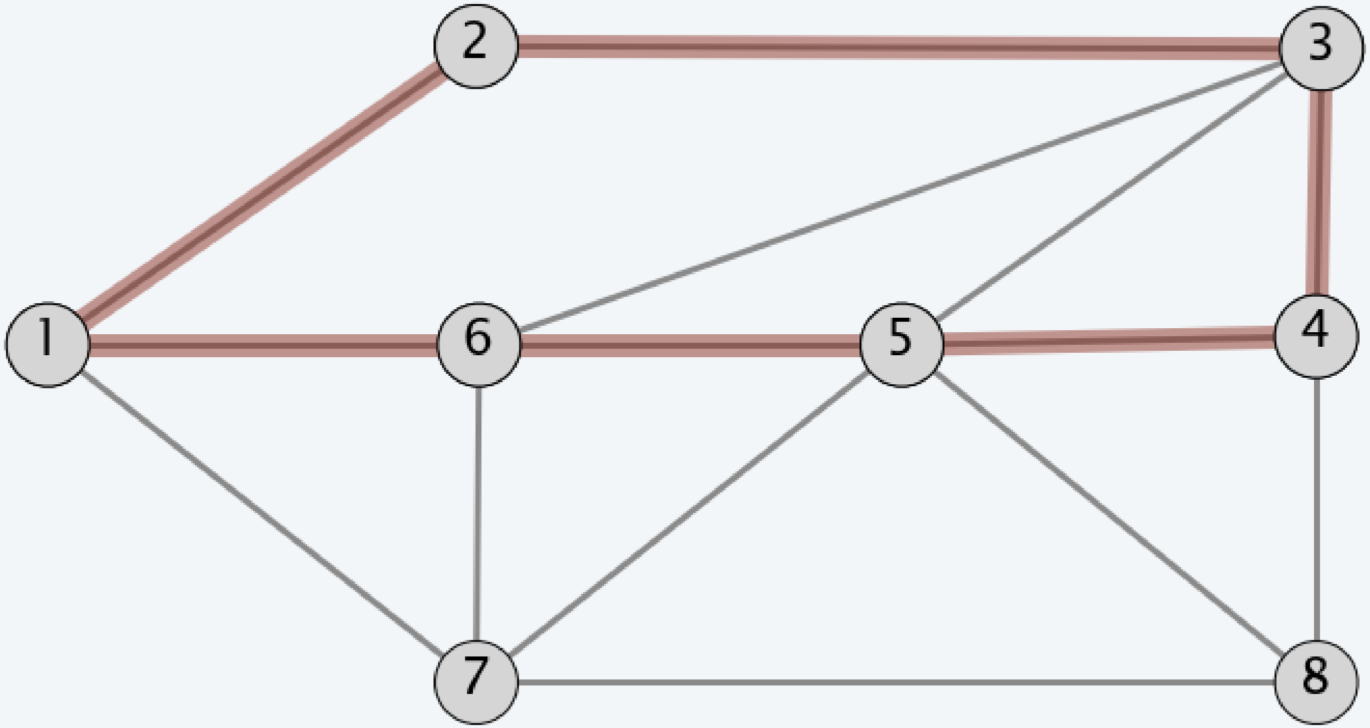
4.8 Huffman Codes

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



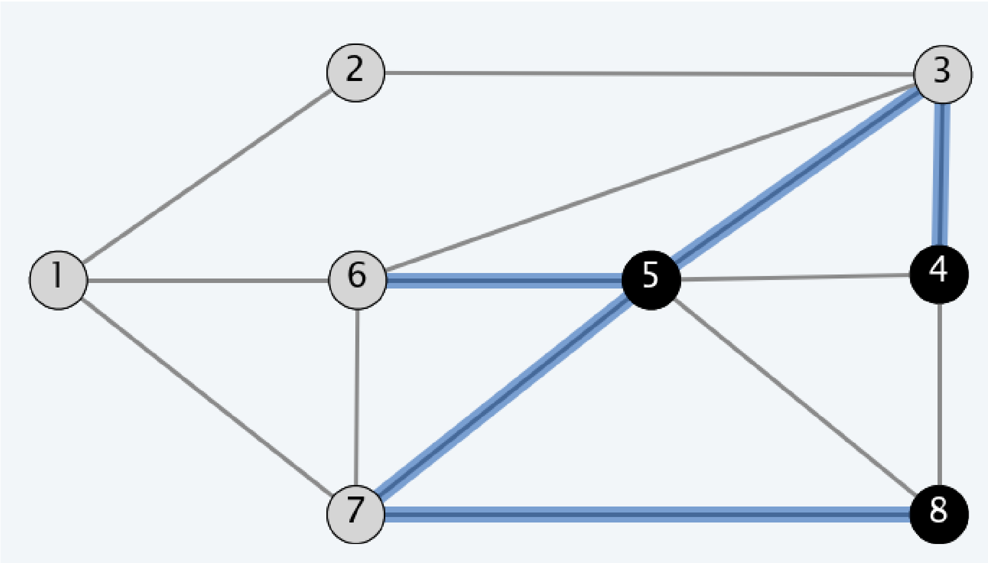
path P = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6) }

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)

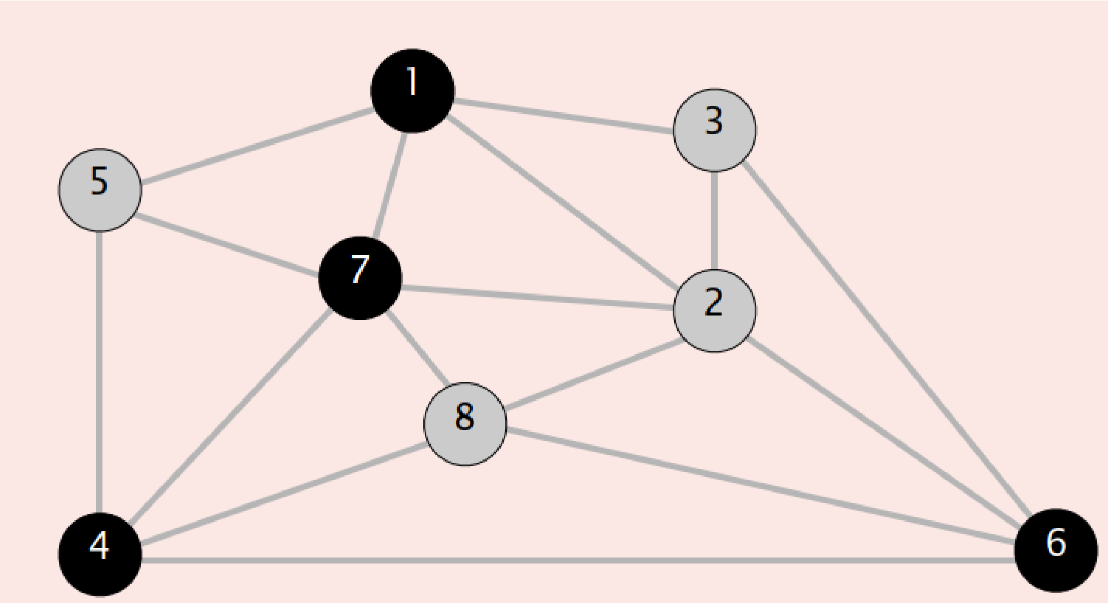
Consider the cut S = { 1, 4, 6, 7 }.
Which edge is in the cutset of S?

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

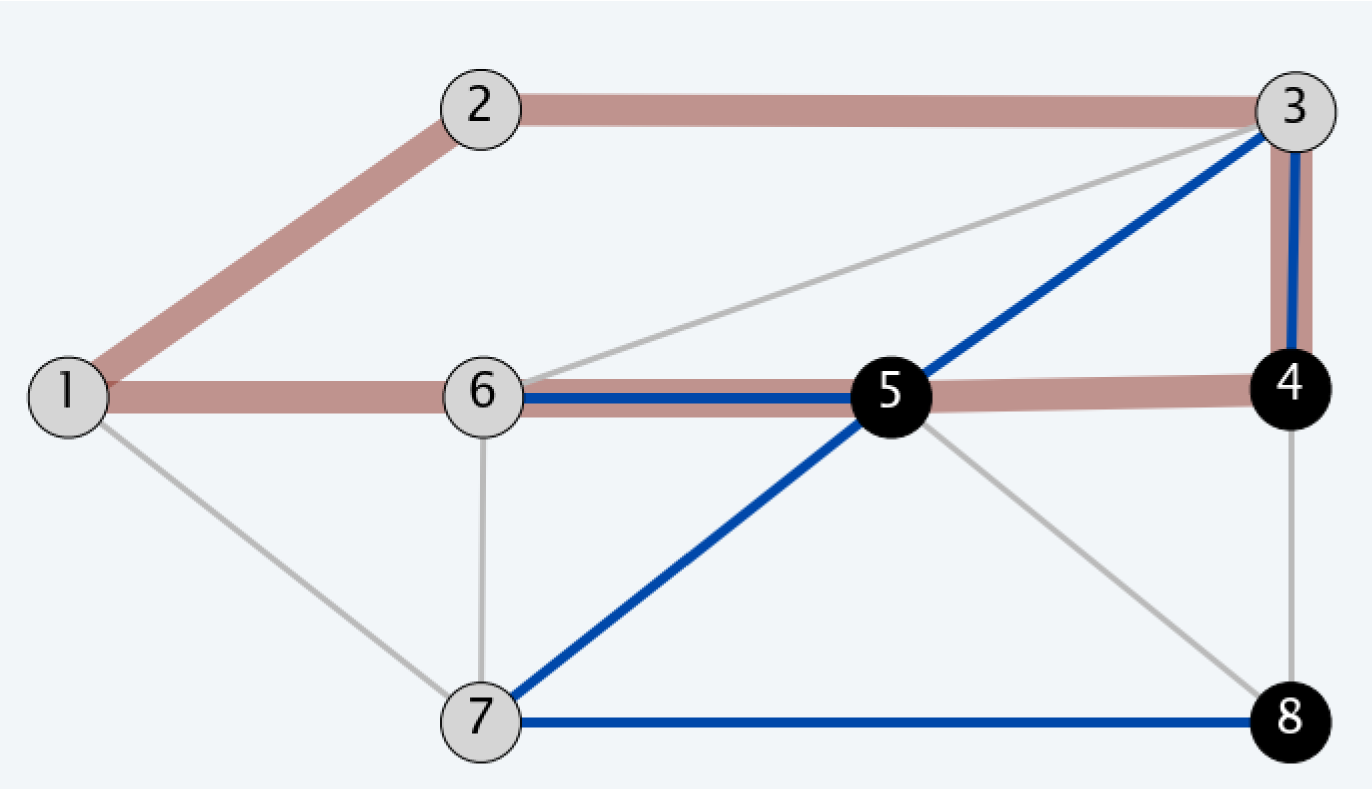
B. 2

C. not 1

D. an even number

4.5 Minimum Spanning Tree

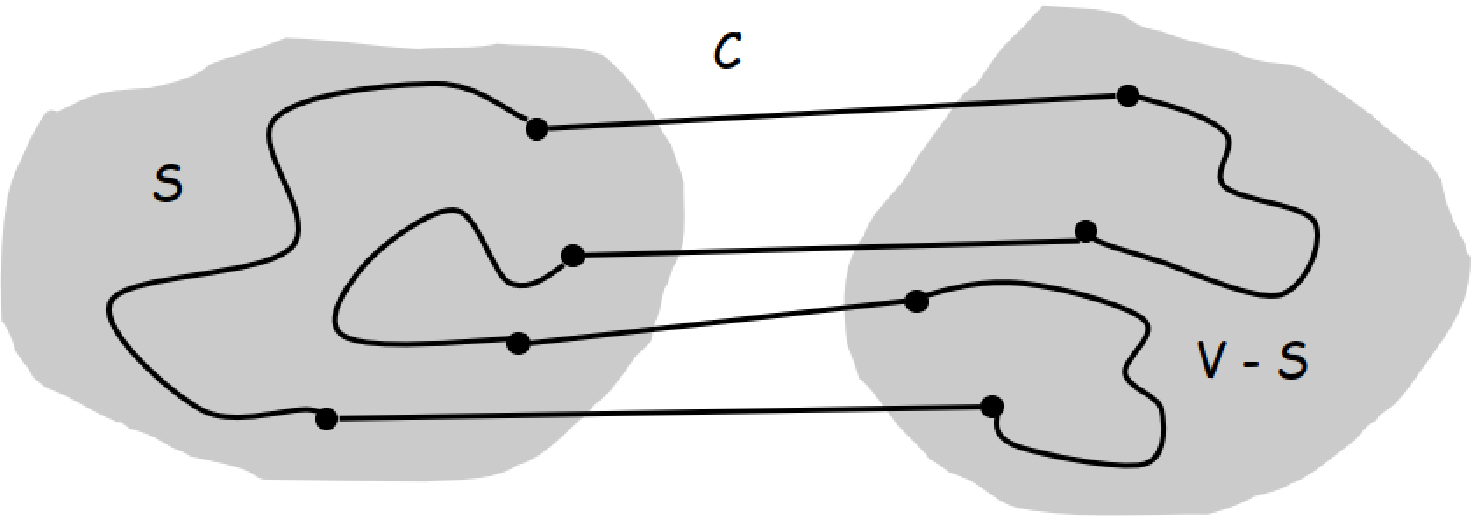
Proposition. A cycle and a cutset intersect in an even number of edges.



4.5 Minimum Spanning Tree

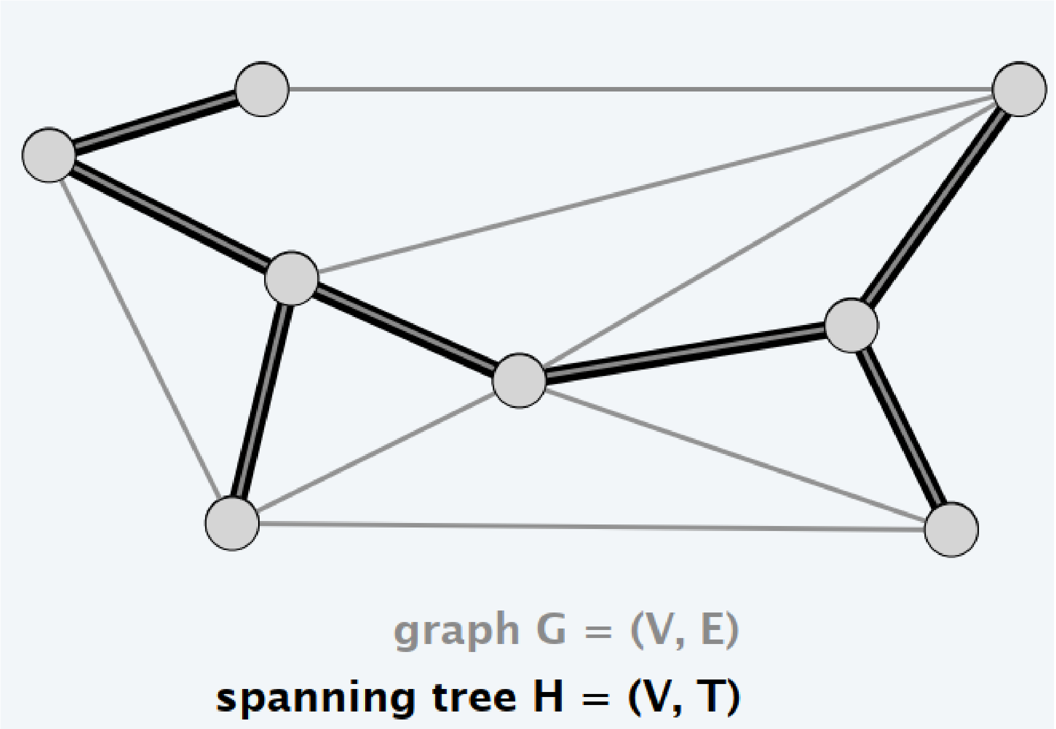
Proposition. A cycle and a cutset intersect in an even number of edges.

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.

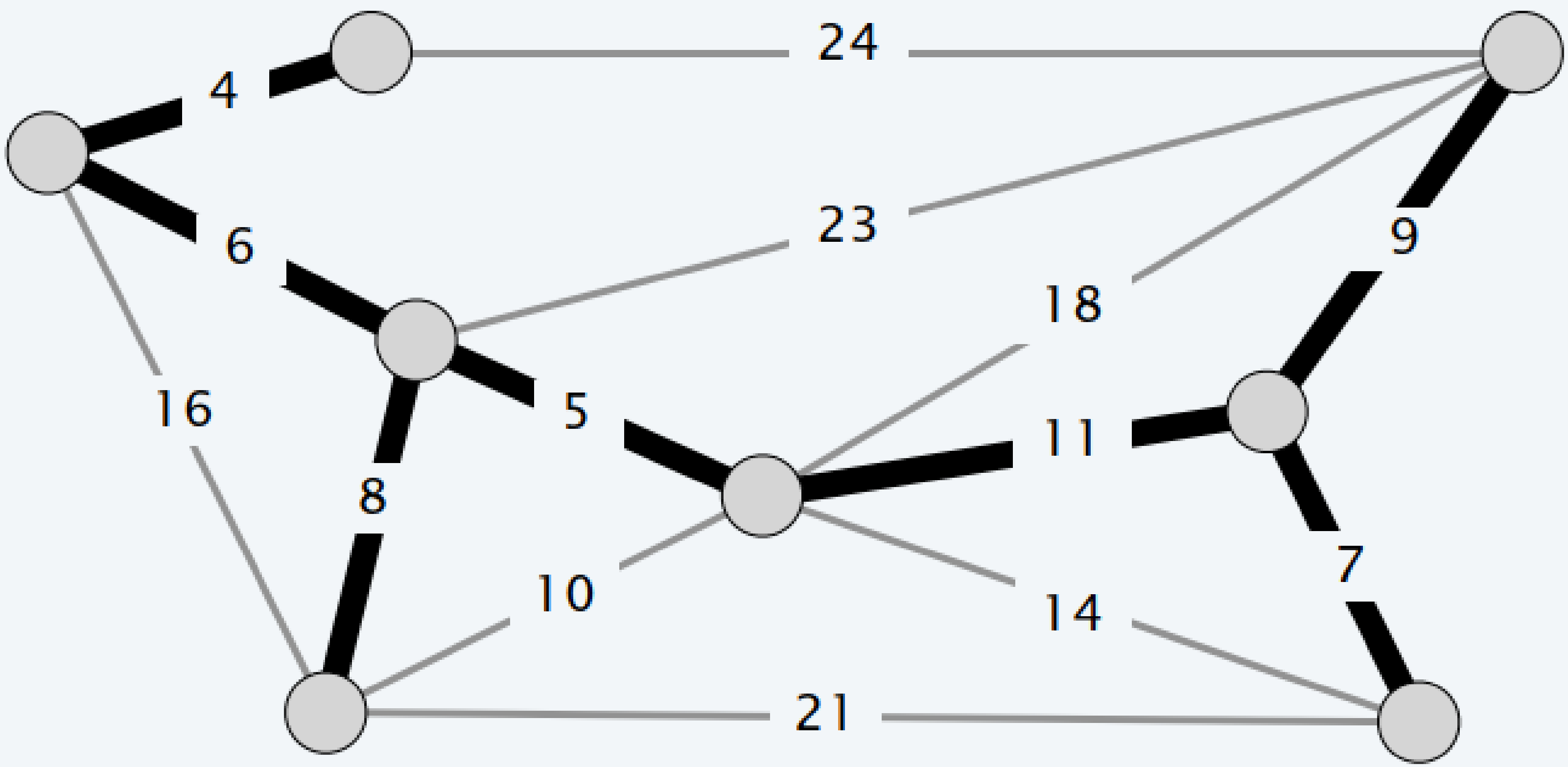
4.5 Minimum Spanning Tree

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

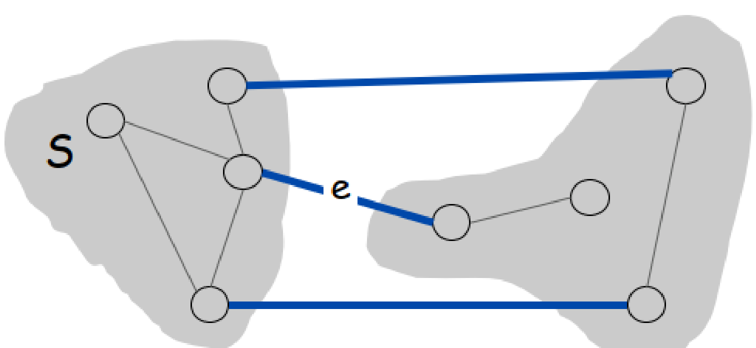
4.5 Minimum Spanning Tree

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 in the cutset of S. Then the MST contains e.



e is in the MST

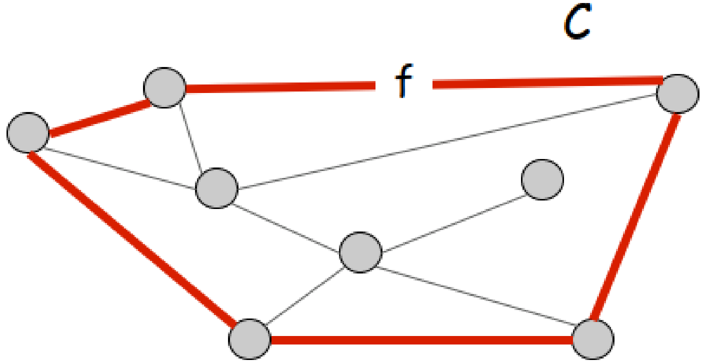
4.5 Minimum Spanning Tree

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.

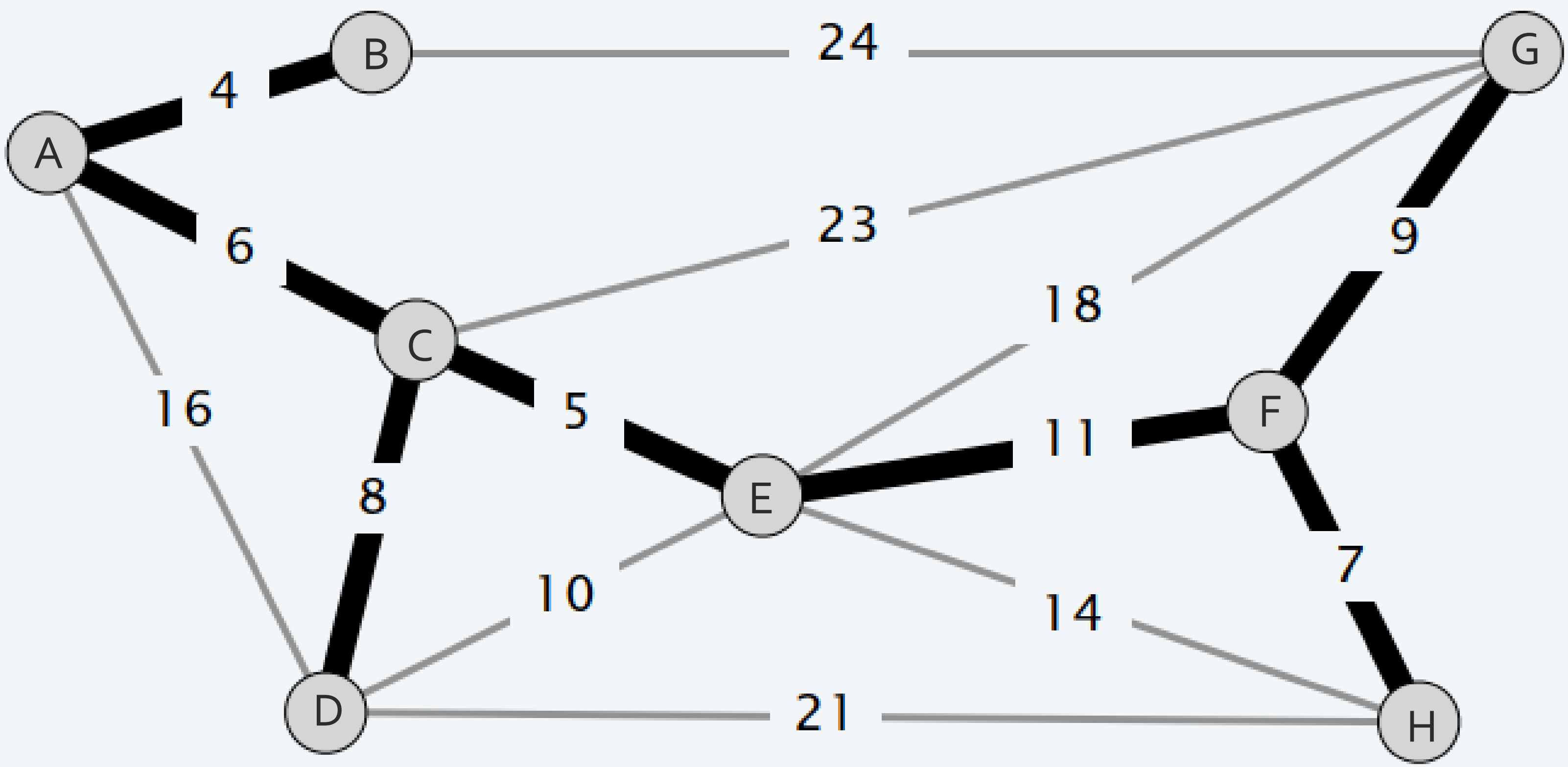
- Suppose f belongs to T^*, and let's see what happens.
- Deleting f from T^* creates a cut S in T^*.
- Edge f is both in the cycle C and in the cutset D corresponding to $S \Rightarrow$ there exists another edge, say e, that is in both C and D.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.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

4.5 Minimum Spanning Tree

Prim's algorithm. [Jarn ḱ 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 of S.
  - Add the other endpoint to S.

4.5 Minimum Spanning Tree



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

4.5 Minimum Spanning Tree

Prim's Algorithm:(n=8,m=13)

0:S={A},T={}

1:S={A,B},T={A-B}

2:S ={A,B,C},T={A-B,A-C}

3:S ={A,B,C,E},T={A-B,A-C,C-E}

4:S ={A,B,C,E,D},T={A-B,A-C,C-E,C-D}

4.5 Minimum Spanning Tree

Prim Algorithm:(*TBC*)

5:S ={A,B,C,E,D,F},T={A-B,A-C,C-E,C-D,E-F}

6:S ={A,B,C,E,D,F,H},T={A-B,A-C,C-E,C-D,E-F,F-H}

7:S ={A,B,C,E,D,F,H,G},T={A-B,A-C,C-E,C-D,E-F,F-H,F-G}

Start with a root node and grow greedily a tree outward.

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$(pq, v, \pi[v])$.

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

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).

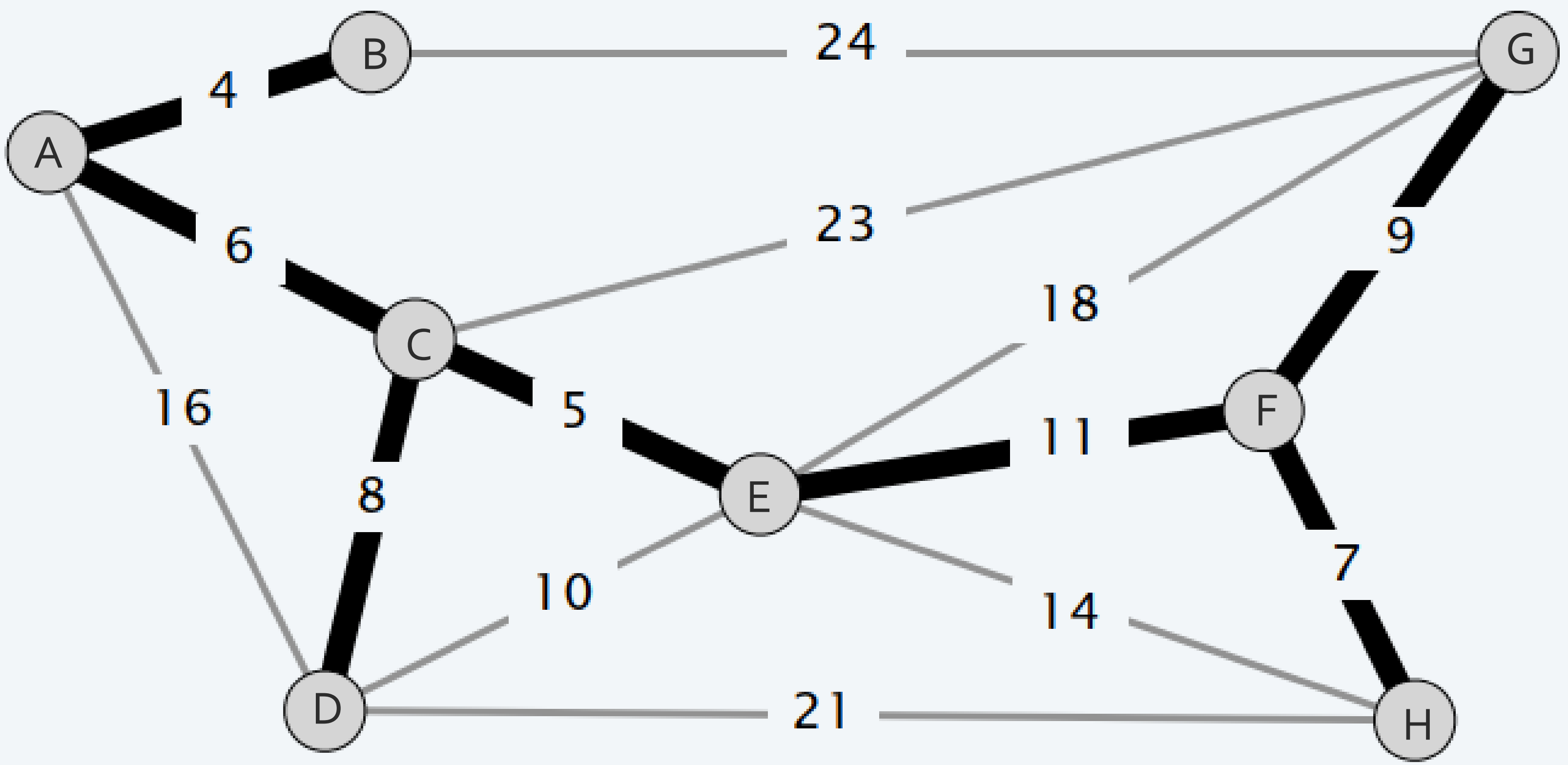
            $\pi[v] \leftarrow c_e$; $pred[v] \leftarrow e$.

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



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

4.5 Minimum Spanning Tree

Kruskal's algorithm:(n=8,m=13)

e=(A,B)[4],T={A-B}

e=(C,E)[5],T={A-B,C-E}

e=(A,C)[6],T={A-B,C-E,A-C}

e=(F,H)[7],T ={A-B,C-E,A-C,F-H}

e=(C,D)[8],T ={A-B,C-E,A-C,F-H,C-D,}

4.5 Minimum Spanning Tree

e=(F,G)[9],T ={A-B,C-E,A-C,F-H,C-D,F-G}

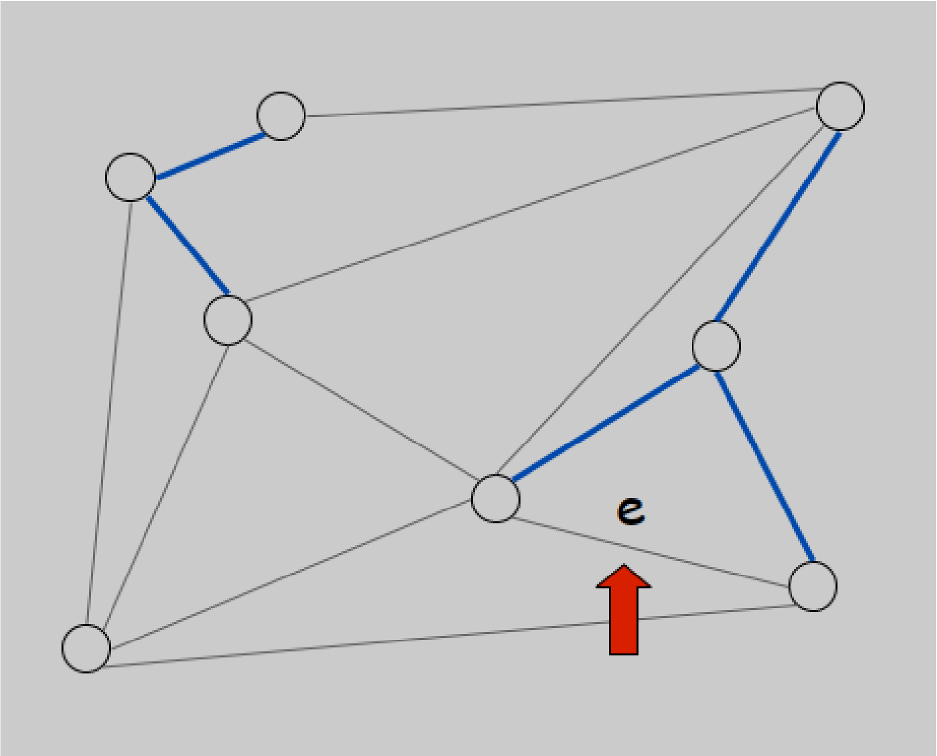
e=(D,E)[10],A cycle exists. discard(D,E)

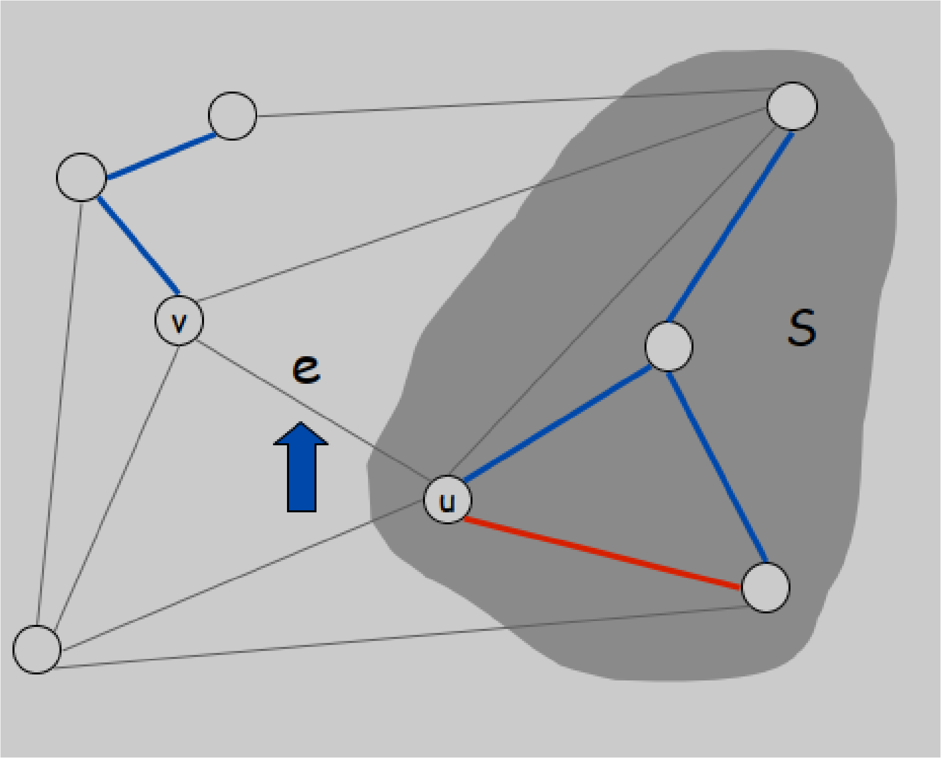
e=(E,F)[11],T ={A-B,C-E,A-C,F-H,C-D,E-F,}

In brief, start without any edges at all and build a SMT by inserting edges in the ascending order of cost.

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
      (u,v) = eᵢ
      if (u and v are in different sets) {
         T ← T ∪ {eᵢ}
         merge the sets containing u and v
      }
   return T
}
```

are u and v in different connected components?

merge two components

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

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

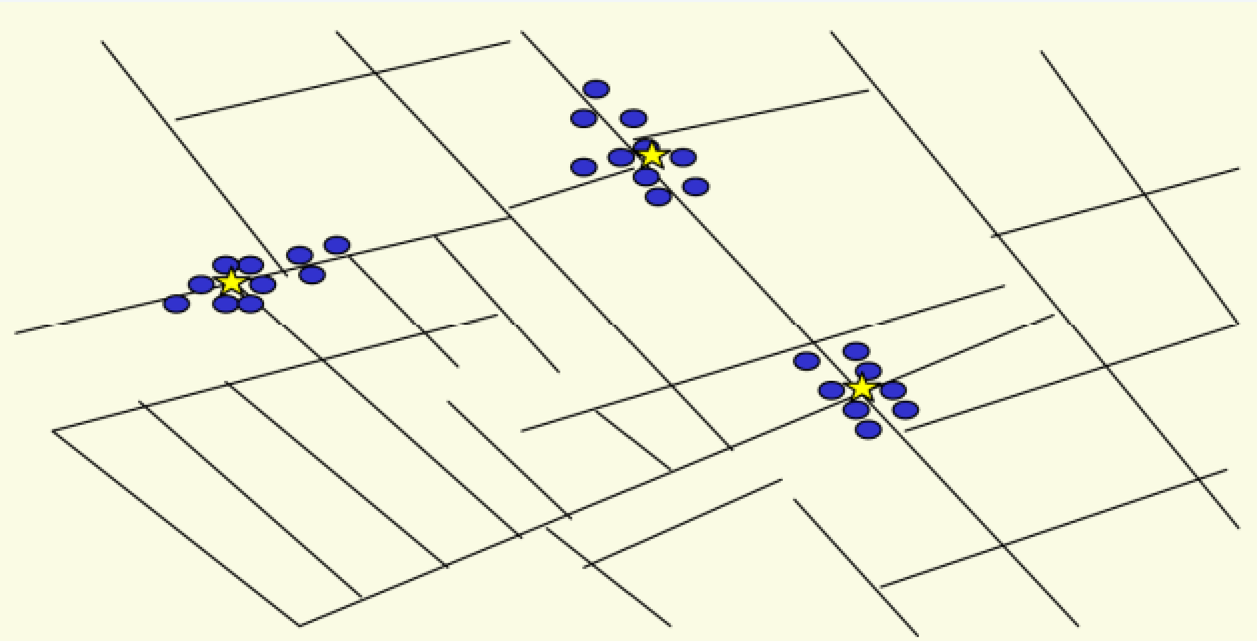
4.5 Minimum Spanning Tree

4.7 Clustering

4.8 Huffman Codes

4.7 Clustering

Goal. Given a set U of n objects labeled $p_1, \cdots, p_n$, partition into clusters so that objects in different clusters are far apart.



4.7 Clustering

Applications

- Routing in mobile ad-hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases
- Identify patterns in gene expression

4.7 Clustering

k-clustering.

- Divide objects into k non-empty groups.

Distance function.

- Numeric value specifying “closeness” of two objects.
- Assume it satisfies several properties.
  - $d(p_i, p_j) = 0$ iff $p_i = p_j$ [identity]
  - $d(p_i, p_j) \geq 0$ [non-negativity]
  - $d(p_i, p_j) = d(p_j, p_i)$ [symmetry]

4.7 Clustering

Single-linkage k-clustering algorithm

- Form a graph on the node set U, corresponding to n clusters.
- Repeat $n - k$ times until there are exactly k clusters.
  - Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.

Key observation.

- This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components)

4.7 Clustering

Theorem. Let C^* denote the clustering $C_1^*, \cdots, C_k^*$ formed by deleting the $k-1$ longest edges of an MST. Then, C^* is a k-clustering of max spacing.

Pf.

- Let C denote any other clustering $C_1, \cdots, C_k$.
- Let p_i and p_j be in the same cluster in C^*, say C_r^* , but different clusters in C, say C_s and C_t.
- Some edge (p, q) on $p_i - p_j$ path in C_r^* spans two different clusters in C.
- Spacing of $C^* =$ length d^* of the $(k-1)^{st}$ longest edge in MST.
- Edge (p, q) has length $\leq d^*$ since it was added by Kruskal.
- Spacing of C is $\leq d^*$ since p and q are in different clusters.

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

4.8 Huffman Codes

Q. Given a text that uses 32 symbols (26 different letters, space, and some punctuation characters), how can we encode this text in bits?

A. We can encode 2^5 different symbols using a fixed length of 5 bits per symbol. This is called fixed length encoding.

Q. Some symbols (i.e., e, t, a, o, i, n) are used far more often than others. How can we use this to reduce our encoding?

A. Encode these characters with fewer bits, and the others with more bits.

4.8 Huffman Codes

Q. How do we know when the next symbol begins?

A. Use a separation symbol (like the pause in Morse), or make sure that there is no ambiguity by ensuring that no code is a prefix of another one.

E.g., c(a)=01, c(b)=010,c(e)=1

What is 0101?

4.8 Huffman Codes

Definition. A prefix code for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.

E.g.,

- $c(a) = 11$
- $c(e) = 01$
- $c(k) = 001$
- $c(l) = 10$
- $c(u) = 000$

Q. What is the meaning of 1001000001 ?

4.8 Huffman Codes

Definition. The average bits per letter of a prefix code c is the sum over all symbols of its frequency times the number of bits of its encoding:

$$ABL(c) = \sum_{x \in S} f_c |c(x)|$$

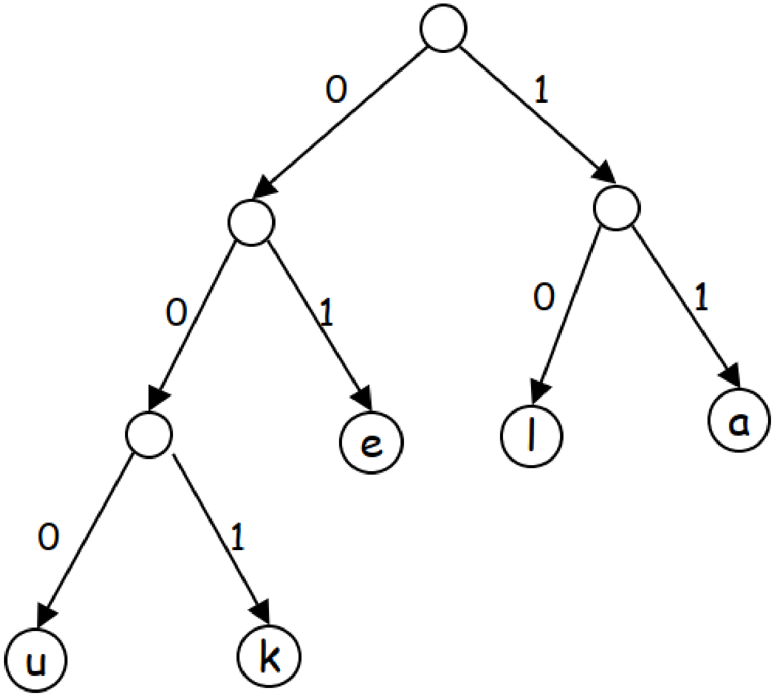
We would like to find a prefix code that is has the lowest possible average bits per letter.

4.8 Huffman Codes

Representing Prefix Codes using Binary Trees

E.g.,

- $c(a) = 11$
- $c(e) = 01$
- $c(k) = 001$
- $c(l) = 10$
- $c(u) = 000$



4.8 Huffman Codes

Q. How does the tree of a prefix code look?

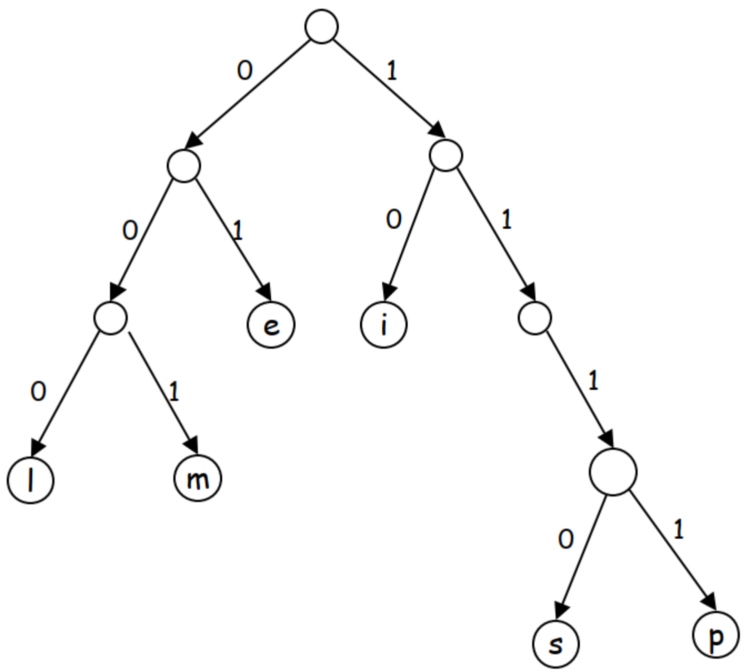
A. Only the leaves have a label.

Pf. An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.

4.8 Huffman Codes

Q. How can this prefix code be made more efficient?

A. Change encoding of p and s to a shorter one. This tree is now full.



4.8 Huffman Codes

Definition. A tree is full if every node that is not a leaf has two children.

Claim. The binary tree corresponding to the optimal prefix code is full.

Pf. [by contradiction]

- Suppose T is binary tree of optimal prefix code and is not full.
- This means there is a node u with only one child v.
- Case 1: u is the root; delete u and use v as the root

4.8 Huffman Codes

Pf. [TBC]

- Case 2: u is not the root
  - let w be the parent of u
  - delete u and make v be a child of w in place of u
- In both cases the number of bits needed to encode any leaf in the subtree of v is decreased. The rest of the tree is not affected.
- Clearly this new tree T' has a smaller ABL than T. Contradiction.

4.8 Huffman Codes

Q. Where in the tree of an optimal prefix code should letters be placed with a high frequency?

A. Near the top.

Greedy choice.

- Create tree top-down, split S into two sets S_1 and S_2 with (almost) equal frequencies.
- Recursively build tree for S_1 and S_2.

4.8 Huffman Codes

Optimal Prefix Codes: Huffman Encoding

Observation 1. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

Observation 2. For $n > 1$, the lowest level always contains at least two leaves.

Observation 3. The order in which items appear in a level does not matter.

4.8 Huffman Codes

Claim. There is an optimal prefix code with tree T^* where the two lowest-frequency letters are assigned to leaves that are siblings in T^*.

Greedy Choice. [Huffman, 1952]

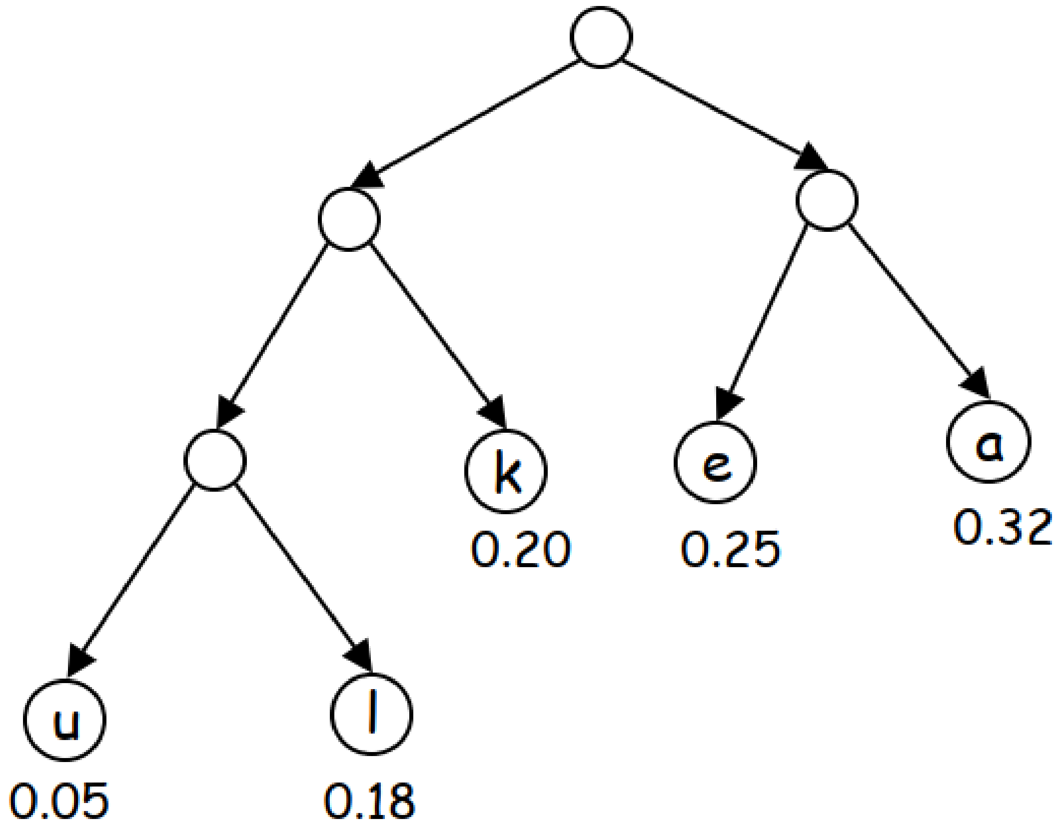
- Create tree bottom-up.
- Make two leaves for two lowest-frequency letters y and z.
- Recursively build tree for the rest using a meta-letter for yz.

4.8 Huffman Codes

```
Huffman(S) {
   if |S|=2 {
      return tree with root and 2 leaves
   } else {
      let y and z be lowest-frequency letters in S
      S′ = S
      remove y and z from S′
      insert new letter ω in S′ with fω=fy+fz
      T′ = Huffman(S′)
      T = add two children y and z to leaf ω from T′
      return T
   }
}
```

4.8 Huffman Codes

f_a=0.32, f_e=0.25, f_k=0.20, f_l=0.18, f_u=0.05



Thanks for Listening

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