Ch18. The Greedy Methods

SNU Internet DataBase Lab.

BIRD'S-EYE VIEW

- Enter the world of algorithm-design methods
- In the remainder of this book, we study the methods for the design of good algorithms
- Basic algorithm methods (Ch18~22)
  - Greedy method
  - Divide and conquer
  - Dynamic Programming
  - Backtracking
  - Branch and bound
- Other classes of algorithms
  - Amortized algorithm method
  - Genetic algorithm method
  - Parallel algorithm method

SNU Internet DataBase Lab.

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

Optimization Problem

- Many problems in chapter 18—22 are optimization problems

- Optimization problem
  - A problem in which the optimization function is to be optimized (usually minimized or maximized) subject to some constraints

- A feasible solution
  - a solution that satisfies the constraints
- An optimal solution
  - a feasible solution for which the optimization function has the best possible value
  - In general, finding an optimal solution is computationally hard

SNU Internet DataBase Lab.

Examples of Optimization Problem

- **Machine Scheduling: Find a schedule that minimizes the finish time**
  - **optimization function: finish time**
  - **constraints**
    - **each job is scheduled continuously on a single machine for its processing time**
    - **no machine processes more than one job at a time**

- **Bin Packing: Pack items into bins using the fewest number of bins**
  - **optimization function: number of bins**
  - **constraints**
    - **each item is packed into a single bin**
    - **the capacity of no bin is exceeded**

- **Minimum Cost Spanning Tree: Find a spanning tree that has minimum cost**
  - **optimization function: sum of edge costs**
  - **constraints**
    - **must select n-1 edges of the given n vertex graph**
    - **the selected edges must form a tree**

Various Attack Strategies for Optimization

- Greedy method
- Divide and Conquer
- Dynamic Programming
- Backtracking
- Branch and Bound

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

The Greedy Method

- Solve a problem by making a sequence of decisions
- Decisions are made one by one in some order
- Each decision is made using a greedy criterion
  - At each stage we make a decision that appears to be the best at the time
- A decision, once made, is (usually) not changed later

SNU Internet DataBase Lab.

Machine Scheduling (1)

- Assign tasks to machines
  - **Given n tasks & an infinite supply of machines**
  - **A feasible assignment is that no machine is assigned two overlapping tasks**
  - **An optimal assignment is a feasible assignment that utilizes the fewest # of machines**

- Suppose we have the following tasks

task	a	b	c	d	e	f	g
start	0	3	4	9	7	1	6
finish	2	7	7	11	10	5	8

(a) Seven tasks

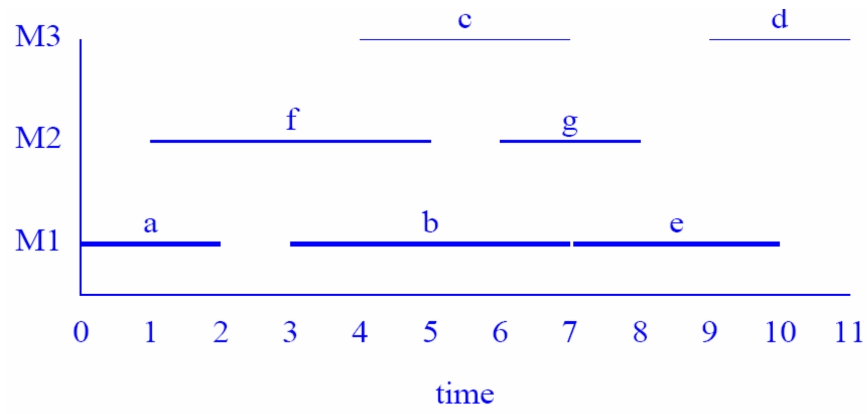
- A feasible assignment is to use 7 machines, but it is not an optimal assignment
  - because other assignments can use fewer machines
  - e.g. we can assign tasks a, b, and d to the same machine, reducing the # of utilized machines to 5

Machine Scheduling (2)

- A greedy way to obtain an optimal task assignment
  - Assign the tasks in stages
    - **one task per stage in nondecreasing order of the task start times**
    - **E.g. task at the starting time 0, task at the starting time 1, etc**
  - For machine selection
    - **If an old machine becomes available by the start time of the task to be assigned, assign the task to this machine**
    - **If not, assign it to a new machine**
- The tasks in the (a) can be ordered by start times: a, f, b, c, g, e, d
  - Then, only 3 machines are needed

task	a	b	c	d	e	f	g
start	0	3	4	9	7	1	6
finish	2	7	7	11	10	5	8

(a) Seven tasks



(b) Schedule

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

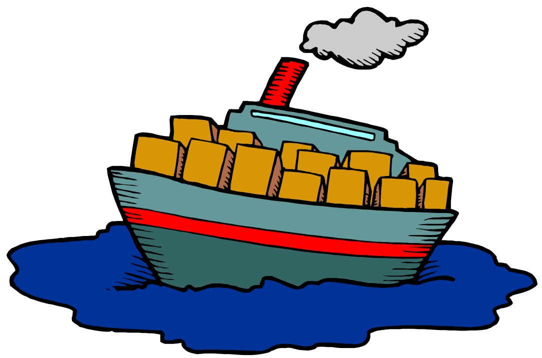
SNU Internet DataBase Lab.

The Original Container-Loading Problem

- Problem Definition
  - Loading a large ship with containers
  - Different containers have different sizes
  - Different containers have different weights
- Goal ➔ To load the ship with the maximum # of containers
- Complexity Analysis
  - Container-loading problem is a kind of bin packing problem
  - The bin packing problem is known to be a combinational NP-hard problem
- Solution
  - Since it is NP-hard, the most efficient known algorithms use heuristics to accomplish good results
    - Which may not be the optimal solution
  - Here, we use greedy heuristics and relax the original problem
    - Which guarantees the optimal solution under a special condition

"Relaxed" Container Loading (1)

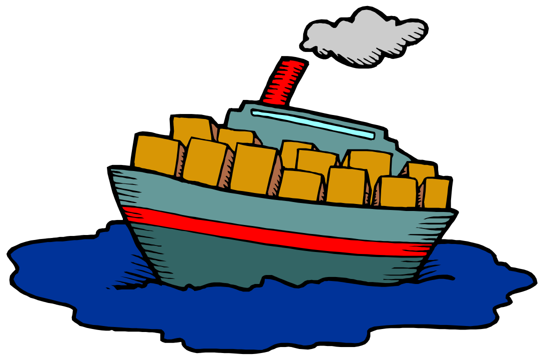
- Problem: Load as many containers as possible without sinking the ship!
  - The ship has the capacity c
  - There are m containers available for loading
  - The weight of container i is w_i
    - Each weight is a positive number
  - The volume of container is fixed
- Constraint: Sum of container weights $<$ c



SNU Internet DataBase Lab.

"Relaxed" Container Loading (2)

- Greedy Solutions
  - Load containers in increasing order of weight until we get to a container that doesn't fit
  - Does this greedy algorithm always load the maximum # of containers?
    - Yes, This is optimal solution!
    - May be proved by using a proof by induction (see text)



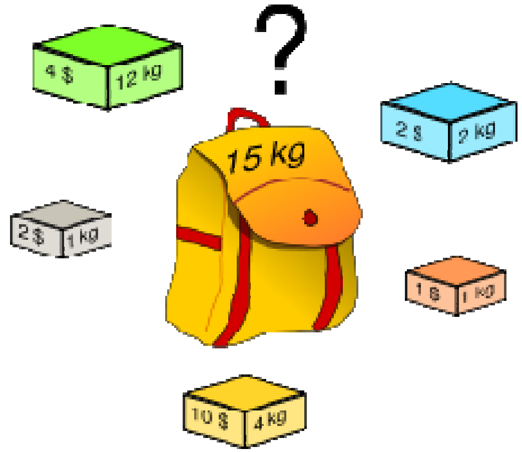
SNU Internet DataBase Lab.

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

The Original Knapsack Problem (1)

- Problem Definition
  - Want to carry essential items in one bag
  - Given a set of items, each has
    - A cost (i.e., 12kg)
    - A value (i.e., 4$)



- Goal
  - To determine the # of each item to include in a collection so that
    - The total cost is less than some given cost
    - And the total value is as large as possible

The Original Knapsack Problem (2)

- Three Types
  - 0/1 Knapsack Problem
    - restricts the number of each kind of item to zero or one
  - Bounded Knapsack Problem
    - restricts the number of each item to a specific value
  - Unbounded Knapsack Problem
    - places no bounds on the number of each item
- Complexity Analysis
  - The general knapsack problem is known to be NP-hard
    - No polynomial-time algorithm is known for this problem
  - Here, we use greedy heuristics which cannot guarantee the optimal solution

SNU Internet DataBase Lab.

0/1 Knapsack Problem (1)

- Problem: Hiker wishes to take n items on a trip
  - The weight of item i is w_i & items are all different (0/1 Knapsack Problem)
  - The items are to be carried in a knapsack whose weight capacity is c
    - When sum of item weights $\leq c$, all n items can be carried in the knapsack
    - When sum of item weights $> c$, some items must be left behind
- Which items should be taken/left?

0/1 Knapsack Problem (2)

- Hiker assigns a profit p_i to item i
  - All weights and profits are positive numbers
- Hiker wants to select a subset of the n items to take
  - The weight of the subset should not exceed the capacity of the knapsack (constraint)
  - Cannot select a fraction of an item (constraint)
  - The profit of the subset is the sum of the profits of the selected items (optimization function)
  - The profit of the selected subset should be maximum (optimization criterion)

- Let $x_i = 1$ when item i is selected and $x_i = 0$ when item i is not selected
  - Because this is a 0/1 Knapsack Problem, you can choose the item or not

$$\text{maximize} \sum_{i=1}^{n} p_i x_i \qquad \text{subject to} \sum_{i=1}^{n} w_i x_i \le c$$

SNU Internet DataBase Lab.

Greedy Attempts for 0/1 Knapsack (1)

- Some heuristics can be applied
  - Greedy attempt on capacity utilization
    - Greedy criterion: select items in increasing order of weight
    - When $n = 2$, $c = 7$, $w = [3, 6]$, $p = [2, 10]$,
      if only item 1 is selected ➔ profit of selection is 2 ➔ not best selection!
  - Greedy attempt on profit earned
    - Greedy criterion: select items in decreasing order of profit
    - When $n = 3$, $c = 7$, $w = [7, 3, 2]$, $p = [10, 8, 6]$,
      if only item 1 is selected ➔ profit of selection is 10 ➔ not best selection!

SNU Internet DataBase Lab.

Greedy Attempts for 0/1 Knapsack (2)

- Greedy attempt on profit density (p/w)
  - Greedy criterion: select items in decreasing order of profit density
  - When n = 2, c = 7, w = [1, 7], p = [10, 20],
    if only item 1 is selected ➔ profit of selection is 10 ➔ not best selection!

- Another greedy attempt on profit density (p/w)
  - Works when selecting a fraction of an item is permitted
  - Greedy criterion: select items in decreasing order of profit density, and if next item doesn't fit, take a fraction so as to fill knapsack
  - When n = 2, c = 7, w = [1, 7], p = [10, 20],
    item 1 and 6/7 of item 2 are selected
  - But this solution is not allowed in 0/1 Knapsack

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

SNU Internet DataBase Lab.

Topological Sorting

- A precedence relation exists between certain pairs of tasks
- The set of tasks together with the precedence may be represented as a digraph
  - A task digraph or an activity on vertex (AOV) network
- Topological sorting constructs a topological order from a task digraph
- We tarverse the graph using the greedy criterion:
  - Select any one among vertices having no incoming edge
  - Put the node into the solution & Remove the node and its outgoing edges from the graph
  - Repeat the above steps until no nodes remain

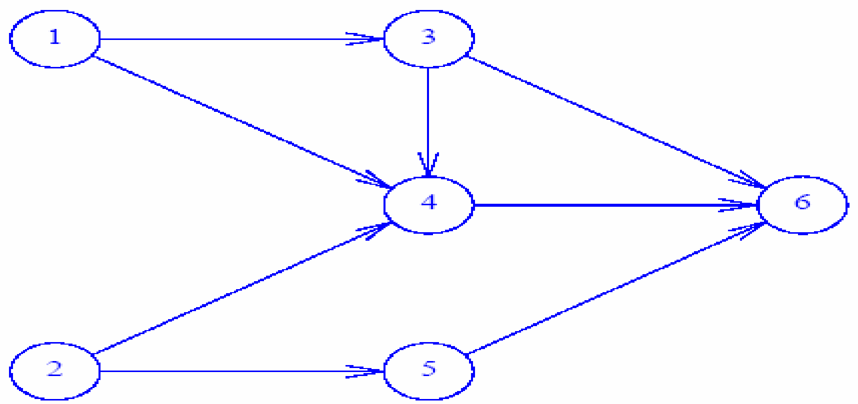


Figure 18.4 A task digraph

SNU Internet DataBase Lab.

Pseudo Code for Topological Sorting

```
Let n be the number of vertices in the digraph.
Let theOrder be an empty sequence.
while (true)
{
  Let w be any vertex that has no incoming edge
   (v,w) such that v is not in theOrder.
  if there is no such w, break.
  Add w to the end of theOrder.
}
if (theOrder has fewer than n vertices)
  the algorithm fails.
else
  theOrder is a topological sequence.
```

Figure 18.5 Topological sorting

Greedy Criterion

- Optimal Solution
  - The greedy method can produce the optimal solution which has linear running time
- Complexity Analysis
  - Looking at the while loop in Fig 18.5, it depends on the data structure
    - O(n^2) if we use an adjacency-matrix representation
    - O(n+e) if we use a linked-adjacency-list representation

SNU Internet DataBase Lab.

Topological Sorting Example

- Results of Topological Sorting
  - Possible topological orders
    - 1 → 2 →3 → 4 → 5 → 6
    - 1 → 3 →2 → 4 → 5 → 6
    - 2 → 1 →5 → 3 → 4 → 6
    - ….
  - Impossible topological orders
    - 1 → 4 → 2 → 3 → 5 → 6
      - Because (for example) task 4 precedes task 3 in this sequence

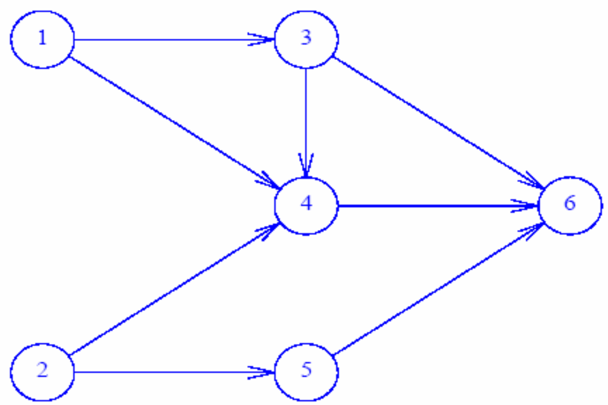


Figure 18.4 A task digraph

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

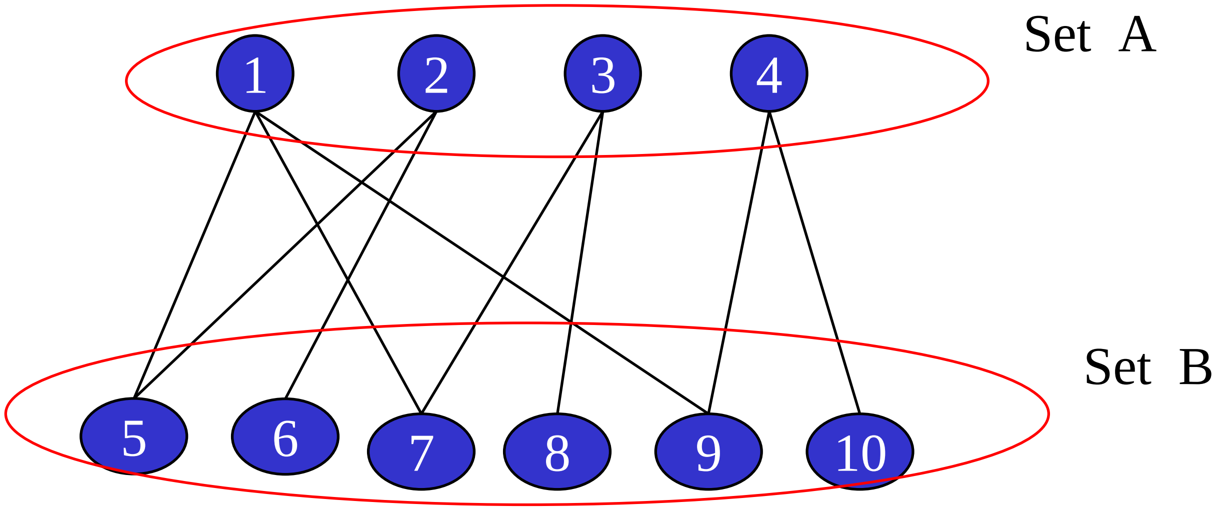
SNU Internet DataBase Lab.

The Original Set Cover Problem

- Problem Definition
  - Given several sets as input
  - The sets may have some elements in common
- Goal
  - To select a minimum number of these sets so that the sets you have picked contain all the elements that are contained in any of the sets in the input
- Example: A (a1, a3), B(a1, a4, a5), C(a2, a5), D(a2, a4, a5), E(a3, a5)
  - Minimum cover: A (a1, a3), D(a2, a4, a5)
- Complexity Analysis
  - The set cover problem is known to be NP-hard
- Bipartite-cover problem is a kind of the set cover problem

Bipartite Graph

- A bipartite graph
  - an undirected graph in which the n vertices may be partitioned into two sets A and B so that no edge in the graph connects two vertices that are in the same set
- A subset *A'* of the node set *A* is said to cover the node set *B* (or simply, *A'* is a cover) iff every vertex in B is connected to at least one vertex of A'



Bipartite Cover Problem

- Find a minimum cover in a bipartite graph!
- Ex: 17-vertex bipartite graph
  - A = {1, 2, 3, 16, 17} B = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
  - The subset A' = {1, 2, 3, 17} covers the set B (size = 4)
  - The subset A' = {1, 16, 17} also covers the set B (size = 3)
  - Therefore, A' = {1, 16, 17} is a minimum cover of B

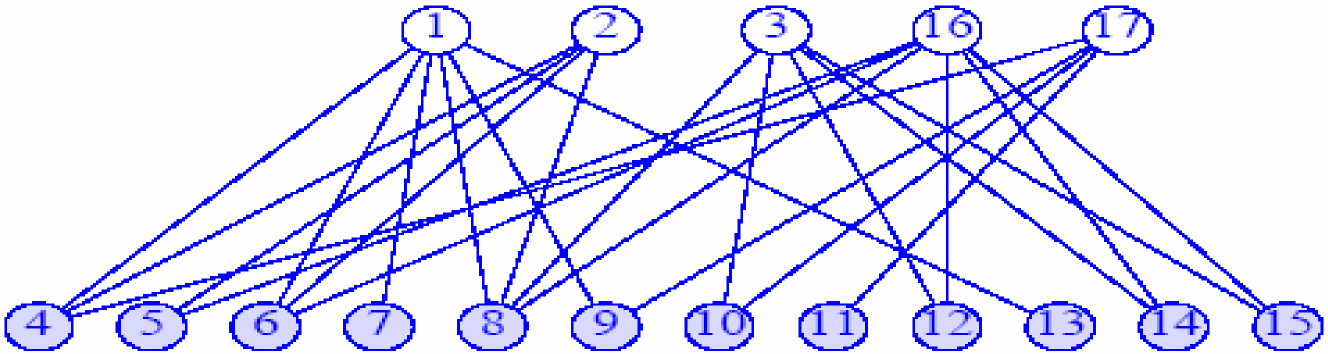


Figure 18.6 Figure for Example 18.10

SNU Internet DataBase Lab.

A Greedy Heuristic for Bipartite Cover (1)

- Bipartite-cover problems are NP-hard
- A greedy method to develop a fast heuristic
  - Construct the cover *A'* in stages
  - Select a vertex of *A* using the greedy criterion:
    - Select a vertex of A that covers the largest # of uncovered vertices of B

- Pseudo Code for Bipartite Cover

```
A' = φ
while (more vertices can be covered)
   Add the vertex that covers the largest number of
   uncovered vertices to A'.
if (some vertices are uncovered) fail.
else a cover has been found.
```

Greedy Criterion

Figure 18.7 High-level statement of greedy covering heuristic

SNU Internet DataBase Lab.

A Greedy Heuristic for Bipartite Cover (2)

- Initial condition
  - V1 & V16 covers six
  - V3 covers five
  - V2 & V17 covers four

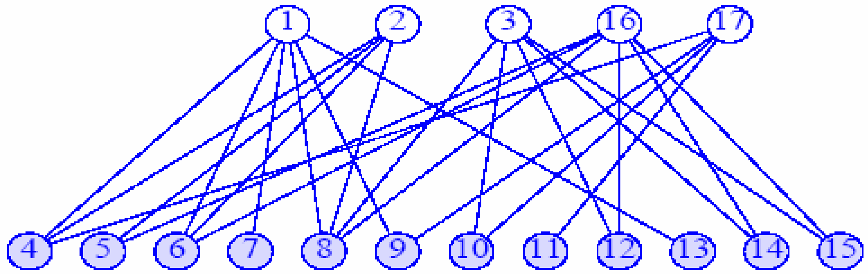


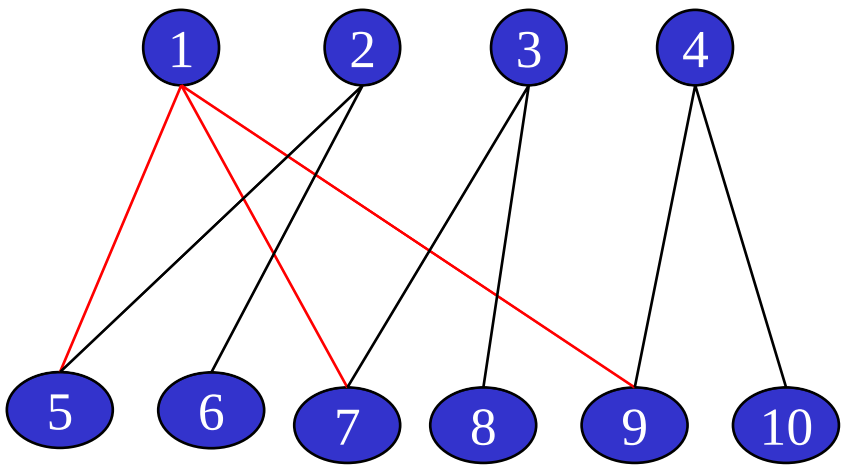
Figure 18.6 Figure for Example 18.10

- 1st stage: Among (V1, V16), suppose we first add V16 to A',
  - it covers {V5, V6, V8, V12, V14, V15} & doesn't cover {V4, V7, V9, V10, V11, V13}
- 2nd stage: Among remainders (V1, V3, V2, V17)
  - choose V1 because it covers four of theses uncovered vertices ({V4, V7, V9, V13})
    - V1 is added to A' and {V10, V11} remain uncovered
- 3rd stage: Among remainders (V3, V2, and V17)
  - V17 covers two of theses uncovered vertices, so we add V17 to A'
- Now no uncovered vertices remain ➔ A' = {V1, V16, V17}

SNU Internet DataBase Lab.

A Greedy Heuristic for Bipartite Cover (3)

- But, this greedy heuristic cannot guarantee the optimal solution
  - If we use the greedy heuristic in the below example,
    - V1 will be added to A'
    - Then V2, V3, and V4 will be added to A'
    - Then A' = {V1, V2, V3, V4}
  - But the optimal solution is {V2, V3, V4}



SNU Internet DataBase Lab.

Table of Contents

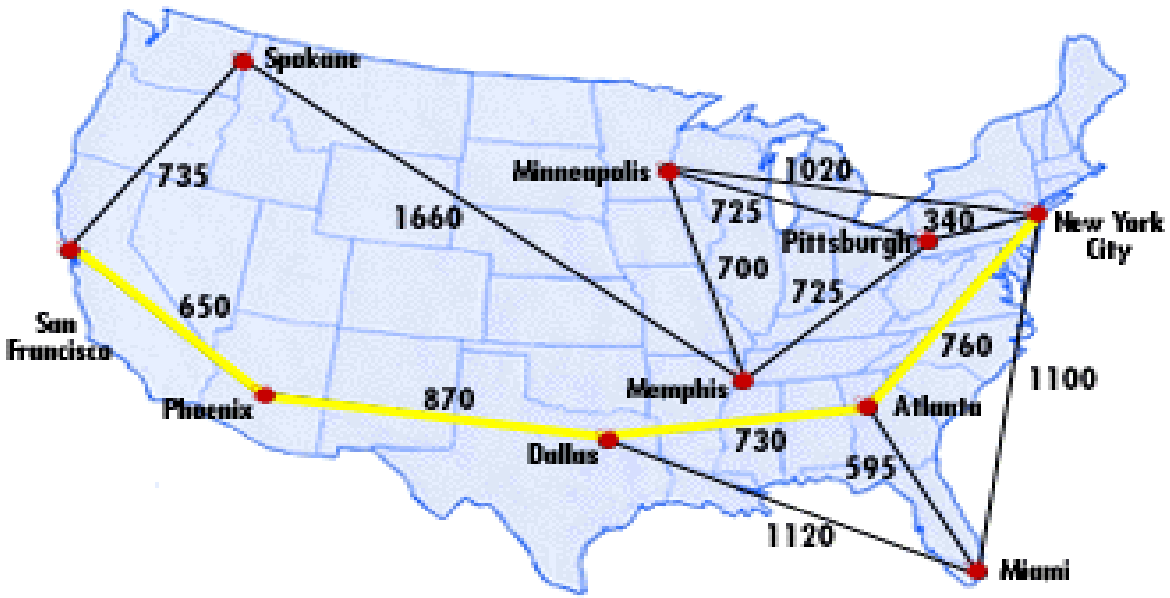
- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

The Shortest Path Problem

- Path length is sum of weights of edges on path in directed weighted graph
  - The vertex at which the path begins is the source vertex
  - The vertex at which the path ends is the destination vertex
- Goal
  - To find a path between two vertices such that the sum of the weights of its constituent edges is minimized
- Complexity Analysis
  - The shortest path problem can be computed in polynomial time
  - But, some varied versions, such as Traveling Salesman Problem, are known to be NP-complete

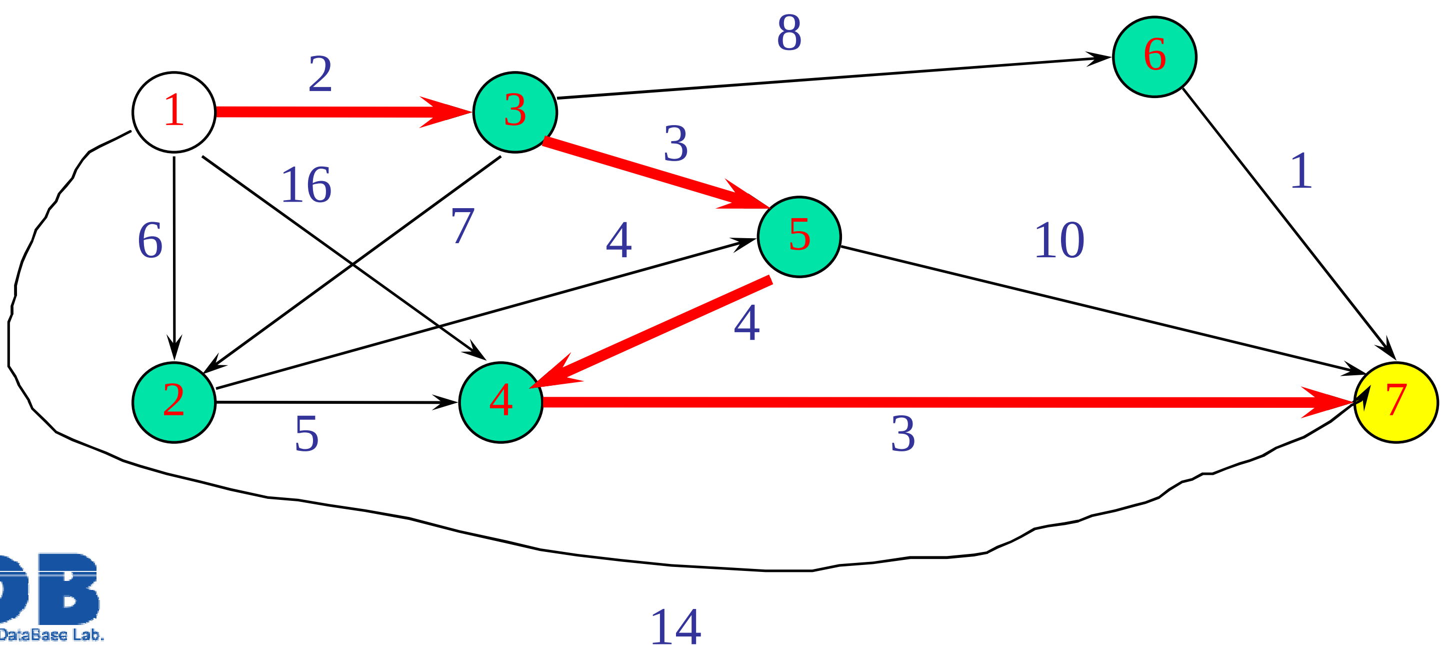
Types of The Shortest Path Problem

- Three types
  - Single-source single-destination shortest path
  - Single-source all-destinations shortest path
  - All pairs (every vertex is a source and destination) shortest path



Single-Source Single-Destination Shorted Path

- Possible greedy algorithm
  - Leave the source vertex using the cheapest edge
  - Leave the current vertex using the cheapest edge to the next vertex
  - Continue until destination is reached
- Try Shortest 1 to 7 Path by this Greedy Algorithm
  - the algorithm does not guarantee the optimal solution

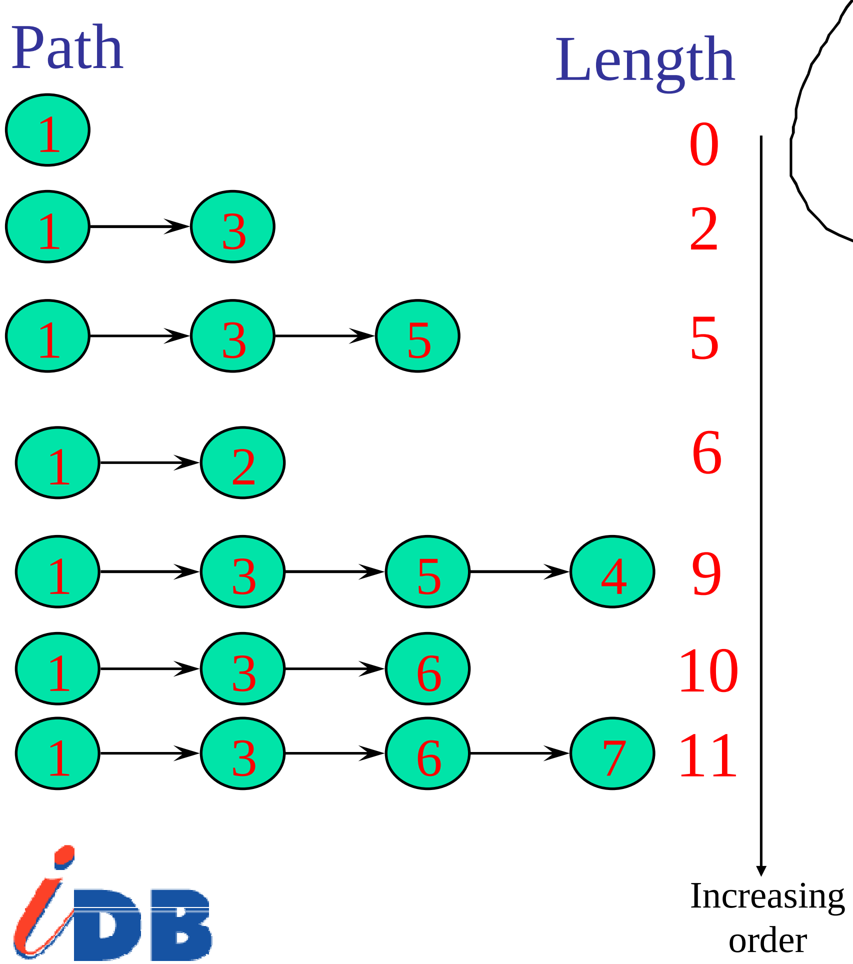


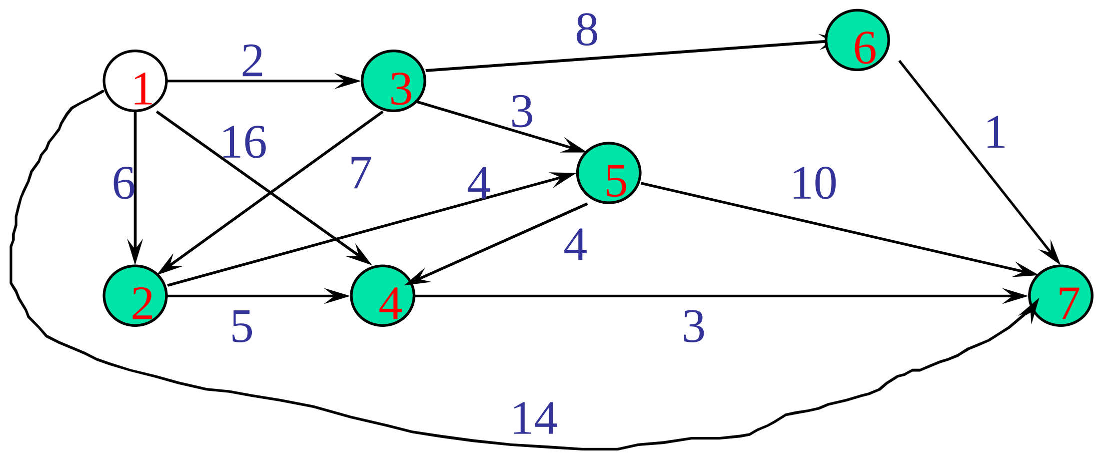
Greedy Single-Source All-Destinations Shortest Path (1)

- Problem: Generating the shortest paths in increasing order of length from one source to multiple destinations

- Greedy Solution
  - Given n vertices, First shortest path is from the source vertex to itself
    - The length of this path is 0
  - Generate up to n paths (including path from source to itself) by the greedy criteria
    - from the vertices to which a shortest path has not been generated, select one that results in the least path length
    - Construct up to n paths in order of increasing length

Greedy Single-Source All-Destinations Shortest Path (2)





- Each path (other than first) is a one edge extension of a previous path
- Next shortest path is the shortest one edge extension of an already generated shortest path

이전에 이미 생성된 shortest path들 중에서 one edge extension 했을 때 length가 가장 작게 증가하는 edge를 선택 → increasing order 보장할 수 있음!

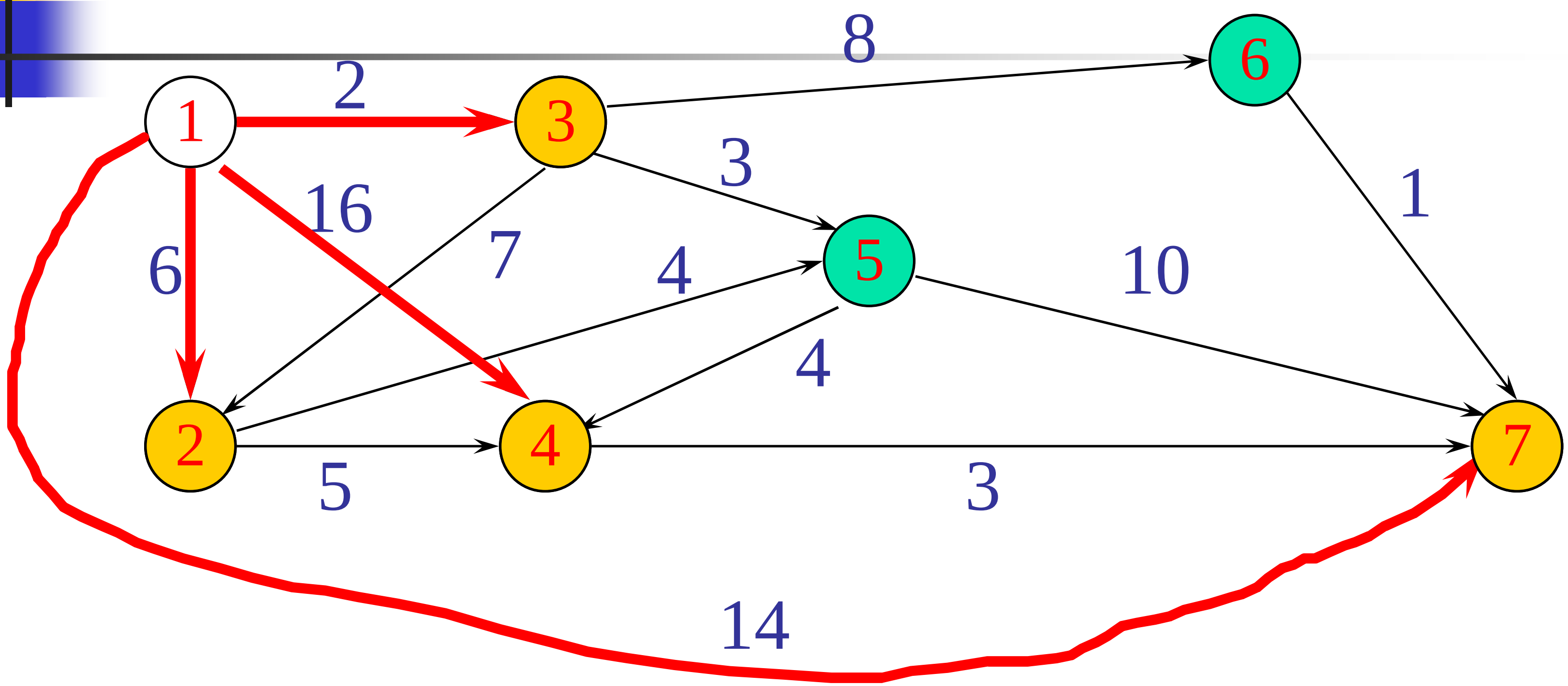
SNU Internet DataBase Lab.

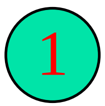
Greedy Single-Source All-Destinations Shortest Path (3)

- Data Structures
  - Let d(i) (distanceFromSource(i)) be the length of a shortest one edge extension of an already generated shortest path, the one edge extension ends at vertex i
  - The next shortest path is to an as yet unreached vertex for which the d() value is least
  - Let p(i) (predecessor(i)) be the vertex just before vertex i on the shortest one edge extension to i
- Complexity Analysis: O(n^2)
  - Any shortest path algorithm must examine each edge in the graph at least once, since any of the edges can be in a shortest path
  - So the minimum possible time for such an algorithm would be O(e)
  - Since cost-adjacency matrices were used to represent the digraph, it takes O(n^2)

SNU Internet DataBase Lab.

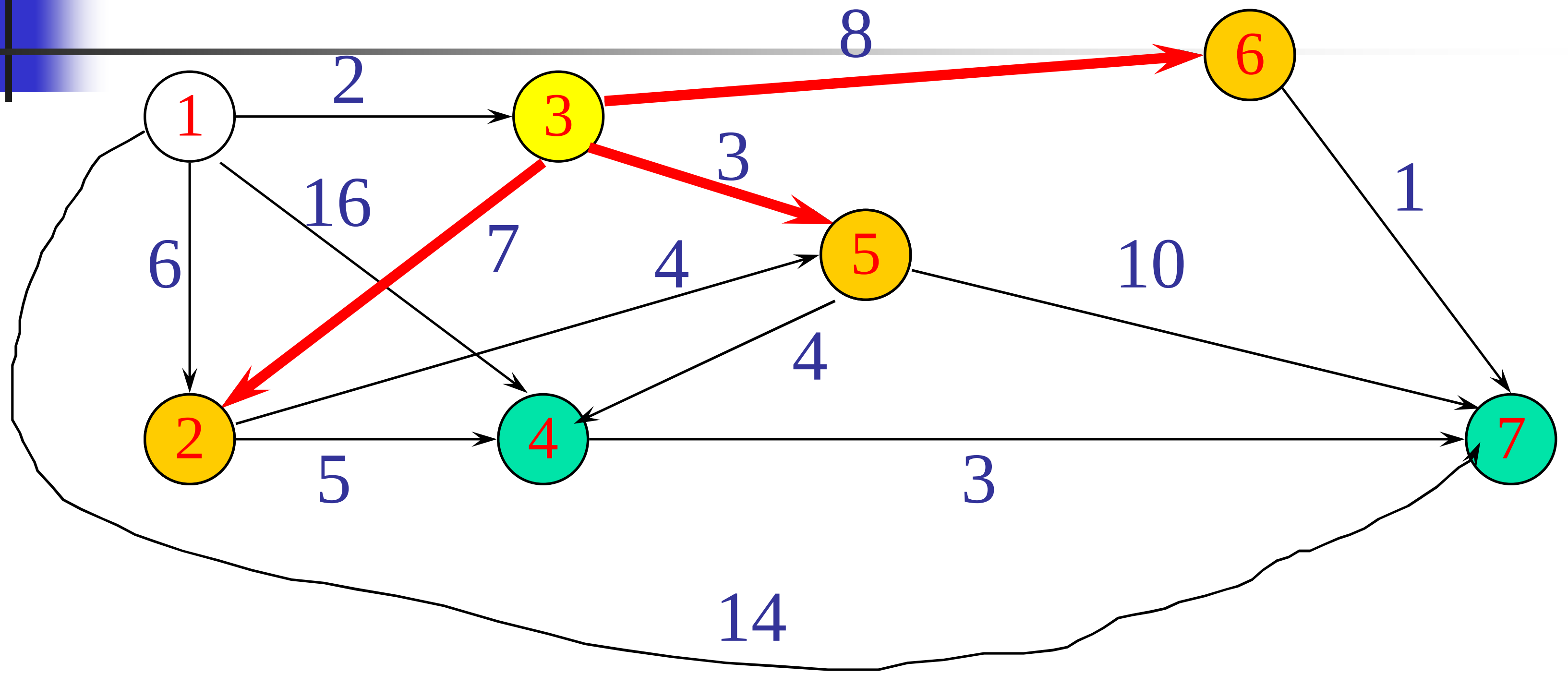
Greedy Single Source All Destinations: Example (1)

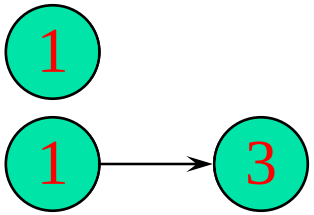




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	$\boxed{2}$	16	-	-	14
p	-	1	1	1	-	-	1

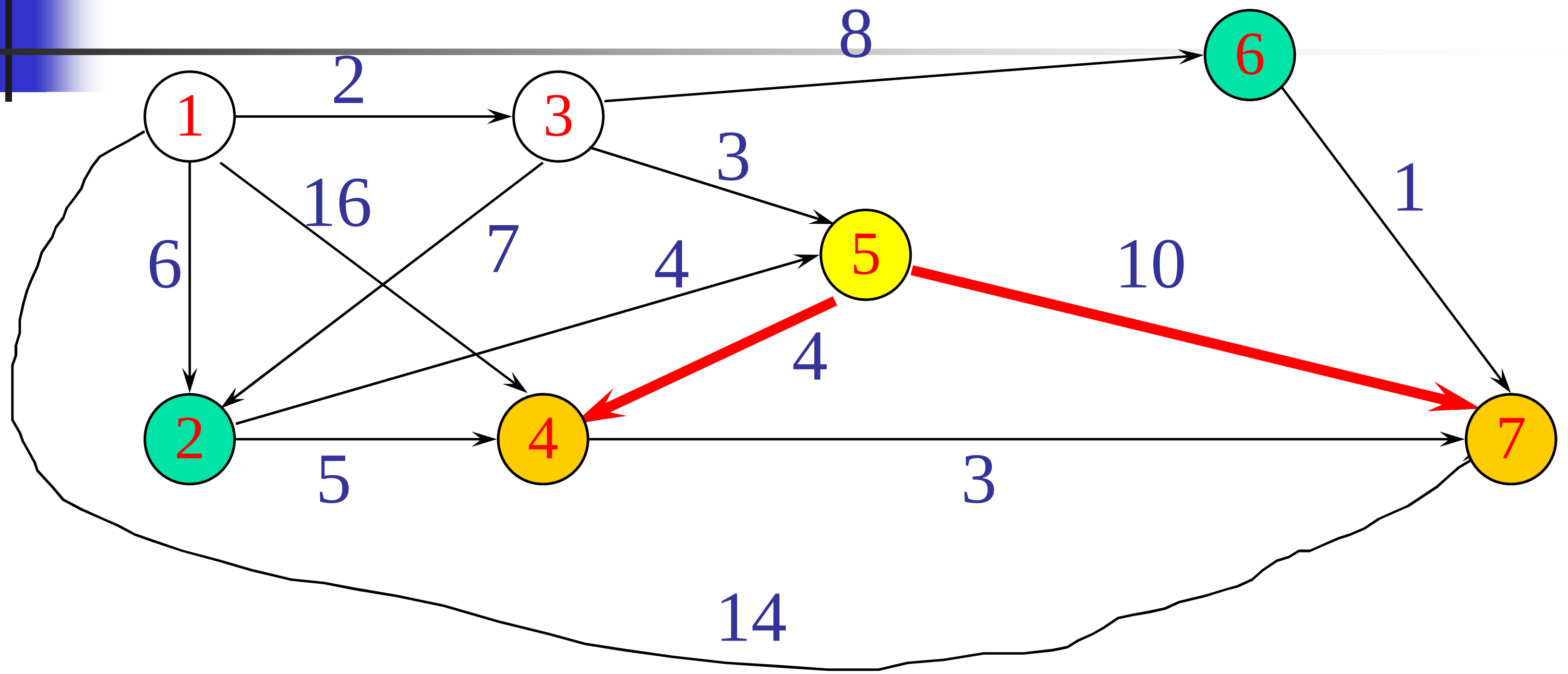
Greedy Single Source All Destinations : Example (2)

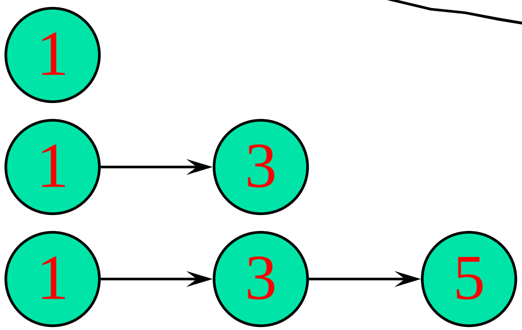




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	16	5	10	14
p	-	1	1	1	3	3	1

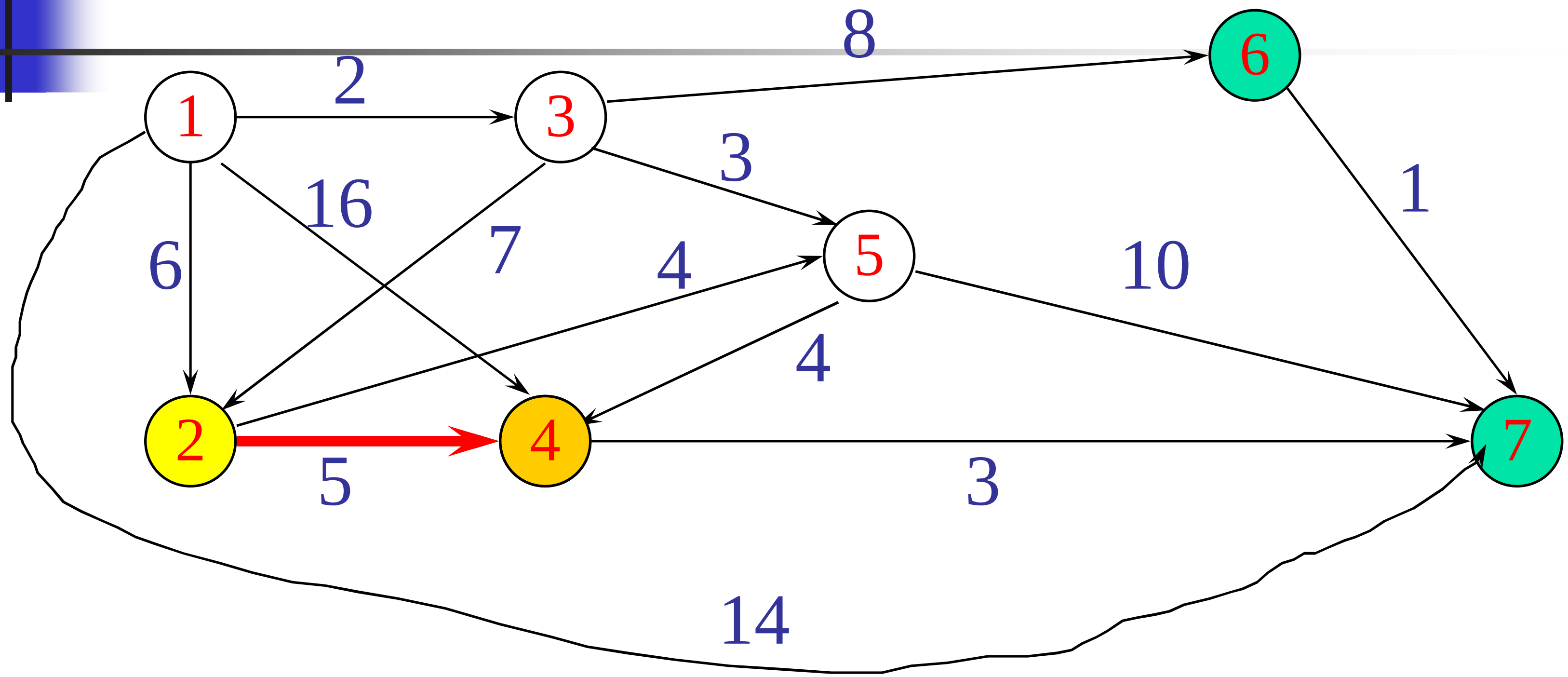
Greedy Single Source All Destinations : Example (3)

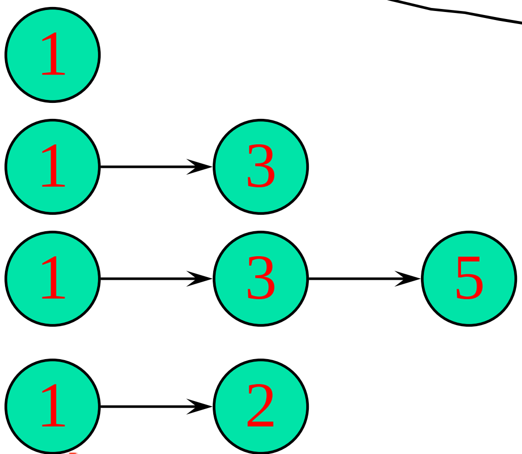




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	14
p	-	1	1	5	3	3	1

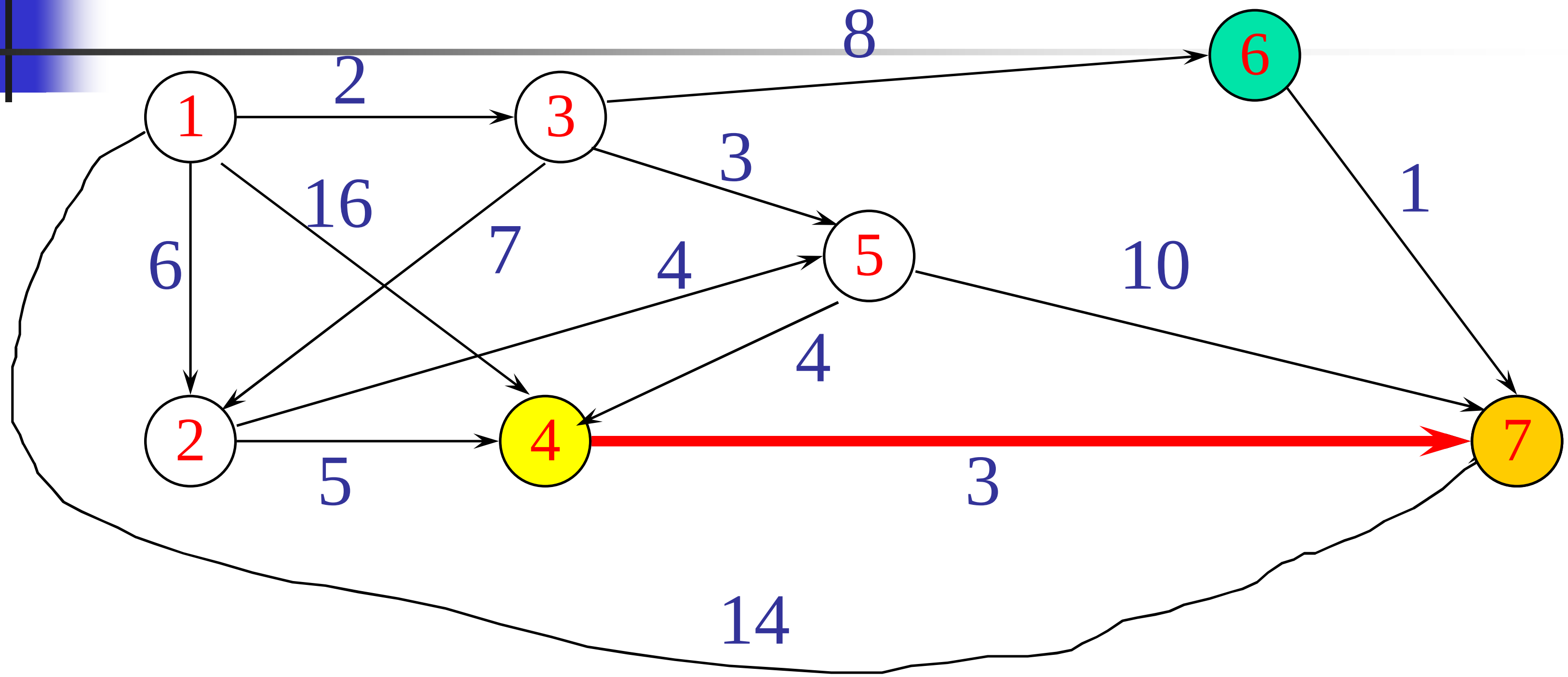
Greedy Single Source All Destinations : Example (4)

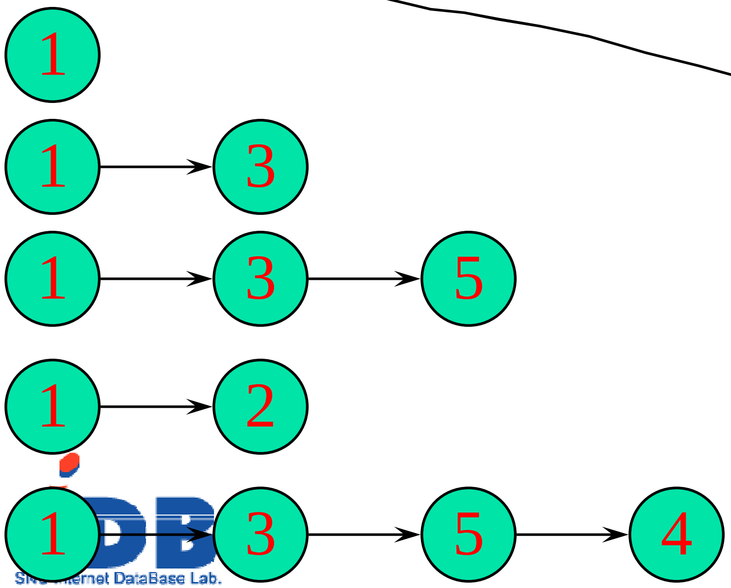




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	14
p	-	1	1	5	3	3	1

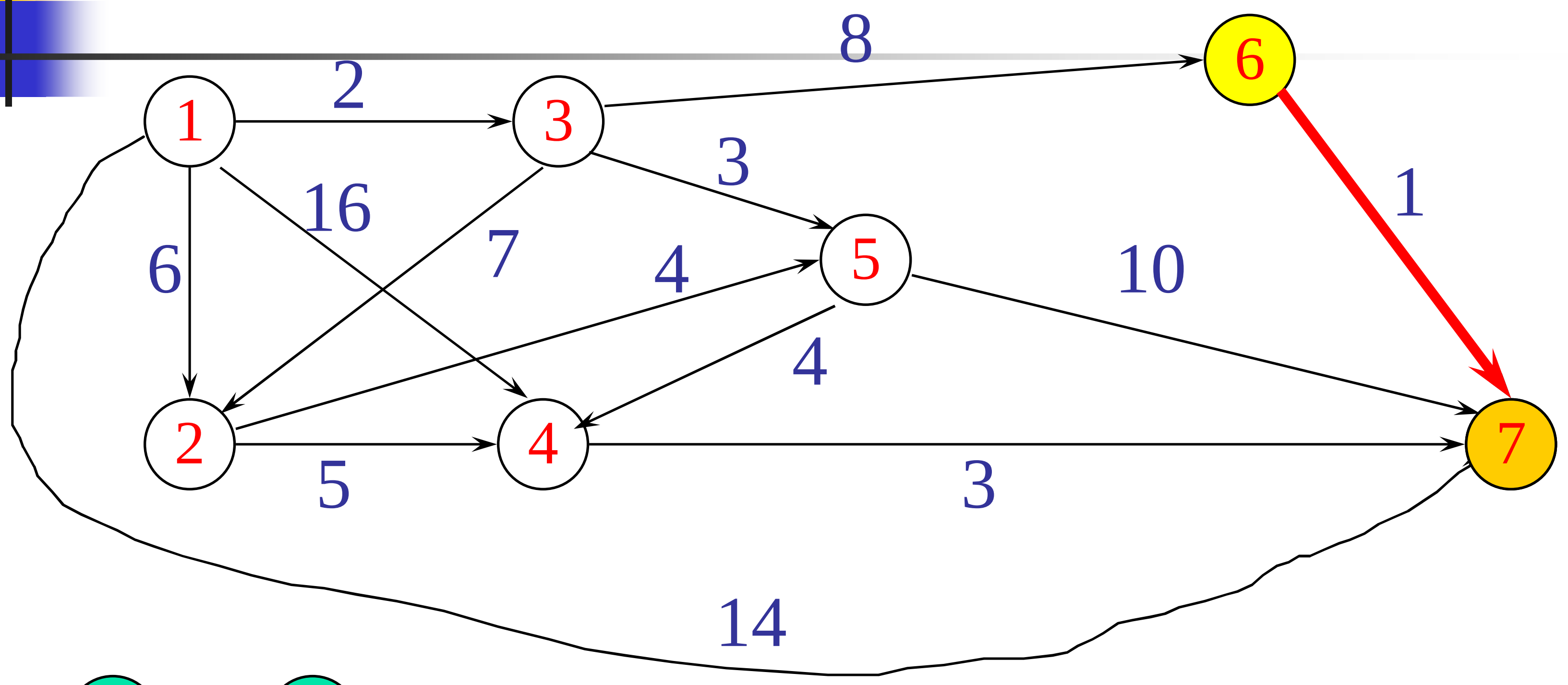
Greedy Single Source All Destinations : Example (5)

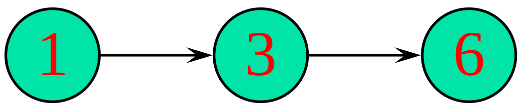




	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	12
p	-	1	1	5	3	3	4

Greedy Single Source All Destinations : Example (6)





	[1]	[2]	[3]	[4]	[5]	[6]	[7]
d	0	6	2	9	5	10	11
p	-	1	1	5	3	3	6

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

SNU Internet DataBase Lab.

Minimum-Cost Spanning Tree

- Spanning tree for weighted connected undirected graph
  - Cost of spanning tree is sum of edge costs
- Goal: Find a spanning tree that has minimum cost!
- Sometimes called, minimum spanning tree

- Complexity Analysis

  The minimum spanning tree can be obtained in polynomial time
  - Kruskal's algorithm
  - Prim's algorithm
  - Sollin's algorithm

Kruskal's Algorithm (1)

- Kruskal's Algorithm selects the n-1 edges one at a time using the greedy criterion:
  - From the remaining edges, select a least-cost edge that does not result in a cycle when added to the set of already selected edges
  - A collection of edges that contains a cycle cannot be completed into a spanning tree

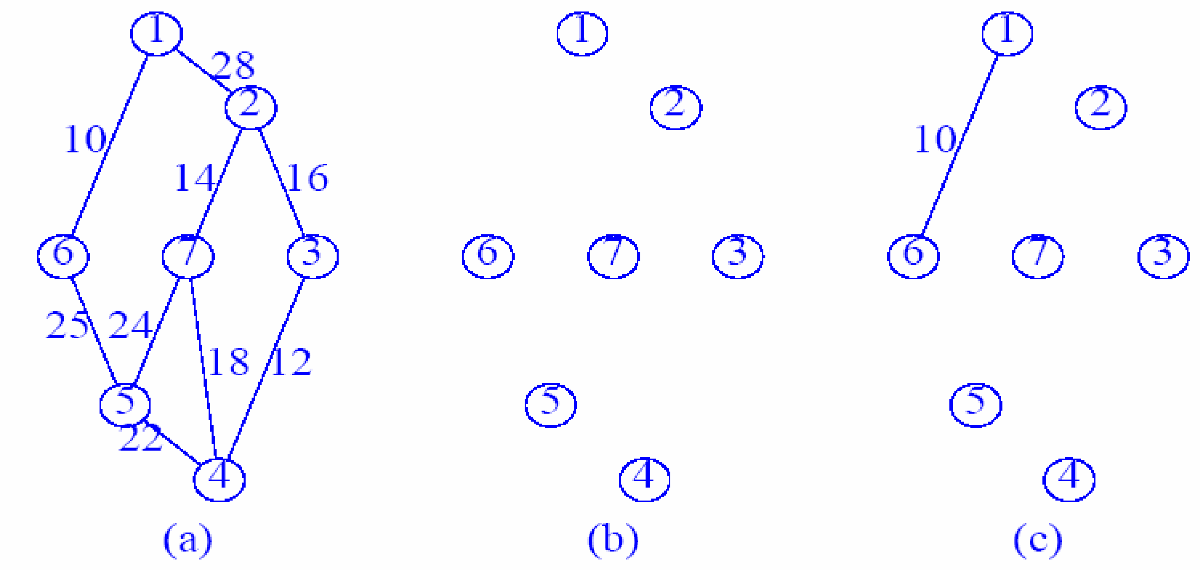


Figure 18.11 Constructing a minimun-cost spanning tree

Kruskal's Algorithm (2)

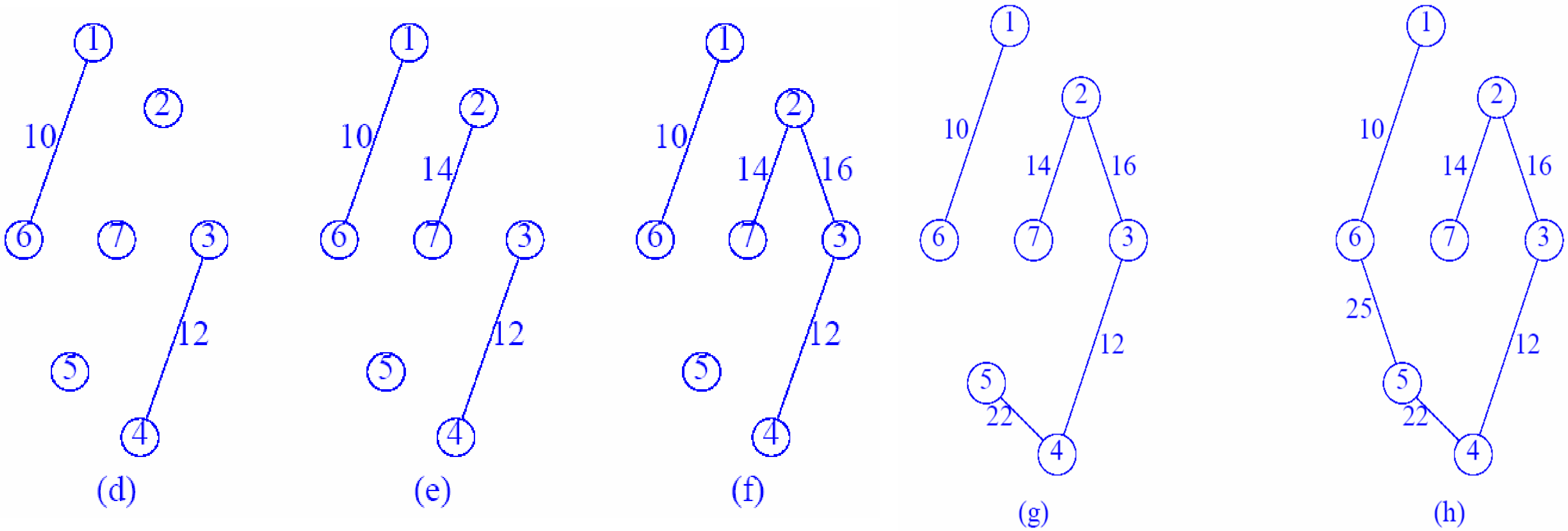


Figure 18.11 Constructing a minimum-cost spanning tree

O(n+e**log*(e)) where n nodes & e edges

SNU Internet DataBase Lab.

Prim’s Algorithm

- Prim’s Algorithm constructs the minimum-cost spanning tree by selecting edges one at a time like Kruskal’s
- The greedy criterion:
  - From the remaining edges, select a least-cost edge whose addition to the set of selected edges forms a tree
  - Consequently, at each stage the set of selected edges forms a tree

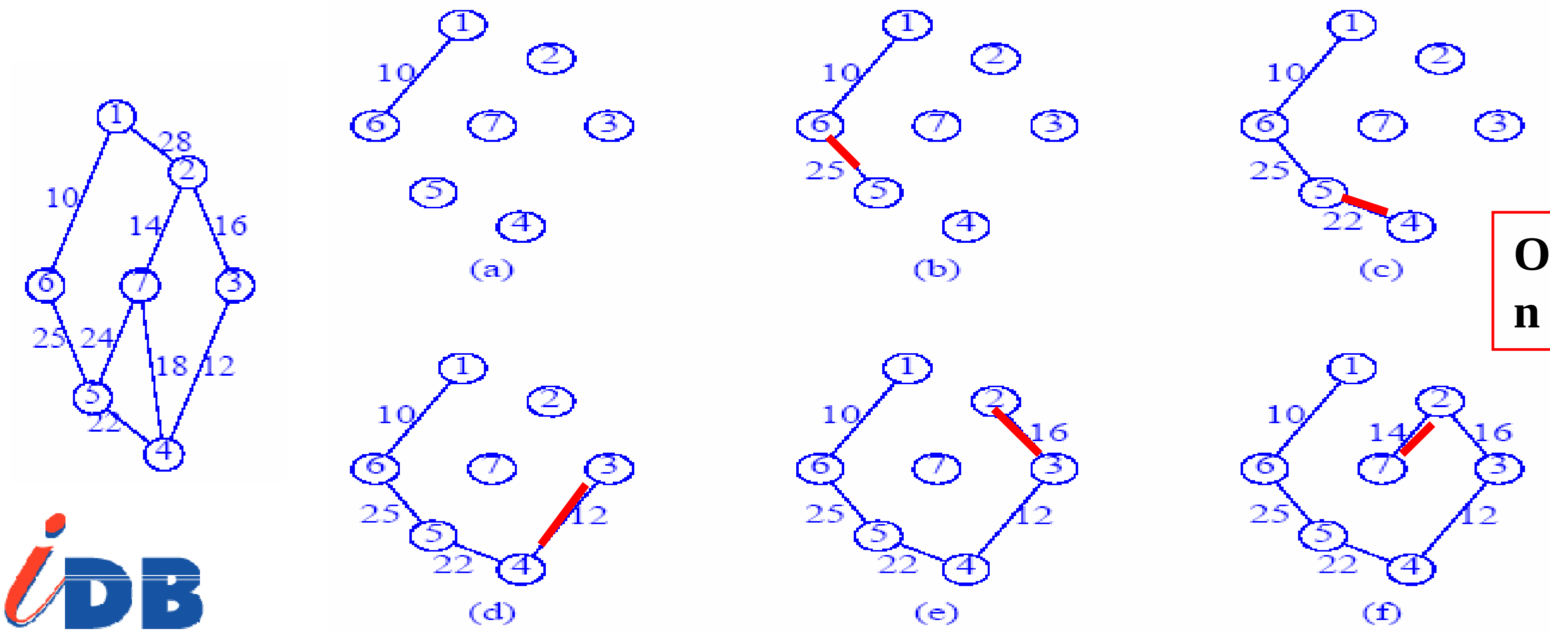
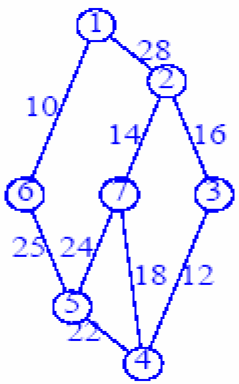


Figure 18.14 Stages in Prim’s algorithm

O(n^2) when n nodes

Sollin's Algorithm

- Sollin's Algorithm selects several edges at each stage
  - select one edge for each tree in the forest so that trees are connected and form a minimum spanning tree
    - Greedy criterion: This selected edge has a minimum-cost edge between two trees
  - At the initial stage (a), vertices 1 to 7 are scanned, and each choose the closest vertex from itself ➔ (1,6), (2,7), (3,4), (4,3), (5,4), (6,1), (7,2)
    - Eliminate the duplicates to get (1,6), (2,7), (3,4), (5,4)
    - At the stage (b), consider the 3 trees in stage (a) as 3 single vertices (t1, t2, t3)
    - (t1, t3), (t2, t3), (t3, t2) ➔ (t1, t3), (t2, t3) by eliminating duplicates



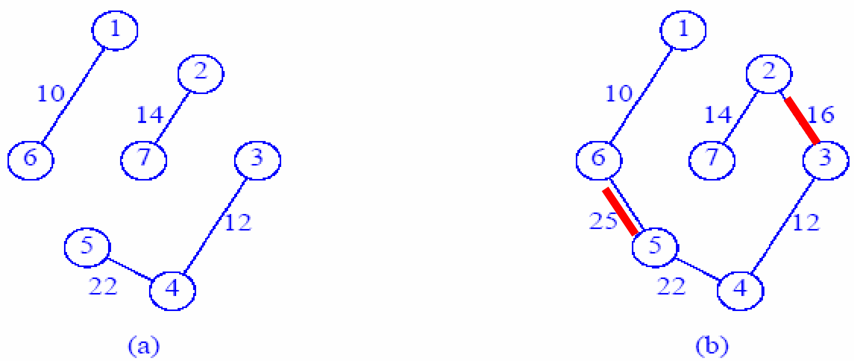


Figure 18.15 Stages in Sollin's algorithm

O(e**log*n)
with n nodes and e edges

Table of Contents

- Optimization problems
- The Greedy method
- Applications
  - Container Loading
  - 0/1 knapsack problem
  - Topological sorting
  - Bipartite cover
  - Single-source shortest paths
  - Minimum-cost spanning trees

SNU Internet DataBase Lab.

BIRD'S-EYE VIEW

- Enter the world of algorithm-design methods
- In the remainder of this book, we study the methods for the design of good algorithms
- Basic algorithm methods (Ch18~22)
  - Greedy method
  - Divide and conquer
  - Dynamic Programming
  - Backtracking
  - Branch and bound

- Other classes of algorithms
  - Amortized algorithm method
  - Genetic algorithm method
  - Parallel algorithm method