

Unit 5.2 The Greedy Method, II

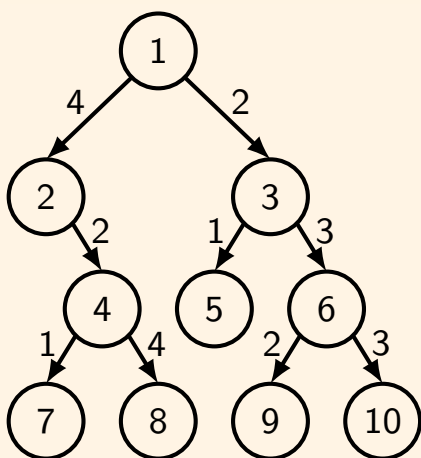
Algorithms

EE3980

Apr. 30, 2020

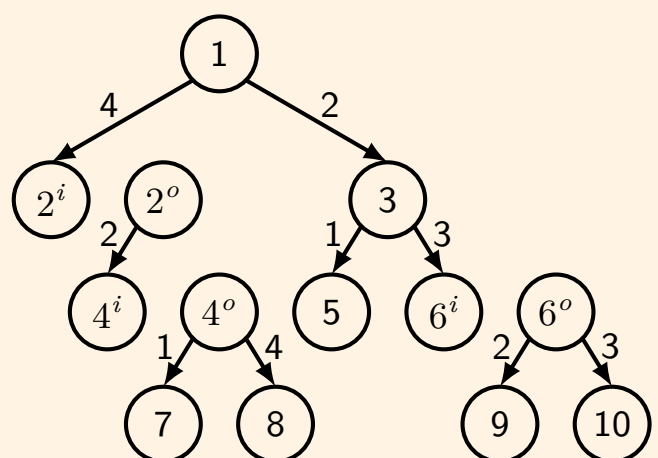
Tree Vertex Splitting Problem

Original tree T



$$d(T) = 10.$$

Tree with vertices splatted T/X



$$d(T/X) = 5.$$

Tree Vertex Splitting Problem – Definition

- $T = (V, E, w)$ is weighted directed tree.
 - V is the vertex set, E is the edge set, and w is weight function for the edges.
 - $w(i, j)$ is defined if the edge $\langle i, j \rangle \in E$; $w(i, j)$ is undefined if $\langle i, j \rangle \notin E$.
 - A **source vertex** is a vertex with in-degree 0.
 - A **sink vertex** is a vertex with out-degree 0.
 - For any path P in the tree, its **delay**, $d(P)$, is defined to be the sum of the weights on the path.
 - The **delay of the tree**, $d(T)$, is the maximum of all the path delays.
- T/X is the forest resulted from splitting every vertex u in $X \subseteq V$ into two nodes u^i and u^o such that all the edges $\langle i, u \rangle$ are replaced by $\langle i, u^i \rangle$ and all the edges $\langle u, j \rangle$ are replaced by $\langle u^o, j \rangle$.
- The **Tree Vertex Splitting Problem (TVSP)** is to find a set $X \subseteq V$ with minimum cardinality for which $d(T/X) \leq \delta$ for some specified tolerance δ .
 - Note that a TVSP has solution only if the maximum edge weight is less than or equal to δ .
 - Any $X \subseteq V$ with $d(T/X) \leq \delta$ is a feasible solution.
 - The optimal solution is the feasible X with the minimum number of vertices.

Tree Vertex Splitting Problem – Algorithm

Algorithm 5.2.1. TVS

```
// Find the minimum set  $X$  for vertex splitting.
// Input: tree  $T$ , maximum edge weight  $\delta$ 
// Output: solution  $X$ .
1 Algorithm TVS( $T, \delta, X$ )
2 {
3     if ( $T \neq \emptyset$ ) then {
4          $d[T] := 0$ ;
5         for each child  $v$  of  $T$  do {
6             TVS( $v, \delta, X$ );
7              $d[T] := \max(d[T], d[v] + w(T, v))$ ;
8         }
9         if (( $T$  is not the root ) and ( $d(T) + w(\text{parent}(T), T) > \delta$ )) then {
10              $X := X \cup \{T\}$ ;
11              $d[T] := 0$ ;
12         }
13     }
14 }
```

- Note that d is a global array that stores the *delay* for each vertex.

Tree Vertex Splitting Problem – Algorithm II

Algorithm 5.2.2. TVS1

```
// Tree vertex splitting with tree stored in an array tree[1 : n].
// Input: root i, maximum edge weight  $\delta$ 
// Output: solution X.
1 Algorithm TVS1(i,  $\delta$ , X)
2 {
3     if (tree[i]  $\neq$  0) then {
4         if ( $2 \times i > N$ ) then d[i] := 0; // i is a leaf.
5         else {
6             TVS1( $2 \times i$ ,  $\delta$ , X);
7             d[i] := max(d[i], d[ $2 \times i$ ] + w[ $2 \times i$ ]);
8             if ( $2 \times i + 1 \leq N$ ) then {
9                 TVS1( $2 \times i + 1$ ,  $\delta$ , X);
10                d[i] := max(d[i], d[ $2 \times i + 1$ ] + w[ $2 \times i + 1$ ]);
11            }
12        }
13        if ((i  $\neq$  1) and (d[i] + w[i]  $>$   $\delta$ )) then {
14            X := X  $\cup$  {i};
15            d[i] := 0;
16        }
17    }
18 }
```

Tree Vertex Splitting Problem – Complexity and Optimality

- In this version the directed **binary tree** is stored in an array *tree*
- The weight is stored in array *w* and *w*[*i*] is the weight of the parent of vertex *i* to vertex *i*.
- Array *d* is still the *delay* of each vertex.
- The time complexity of Algorithm TVS is $\Theta(n)$.
 - Every vertex of *T* is traversed once.

Theorem 5.2.3.

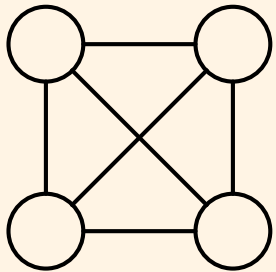
Algorithm TVS finds a minimum cardinality set *X* such that $d(T/X) \leq \delta$ on any tree *T*, provided that no edge of *T* has weight greater than δ .

- Proof please see textbook [Horowitz], pp. 225 - 226.

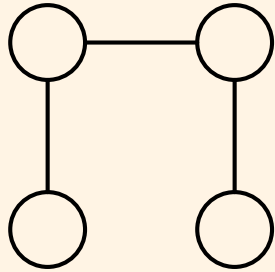
Minimum-Cost Spanning Trees

Definition 5.2.4.

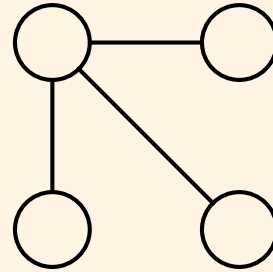
Let $G = (V, E)$ be an undirected connected graph. A sub-graph $T = (V, E')$ with $E' \subseteq E$ is a **spanning tree** of G if and only if T is a tree.



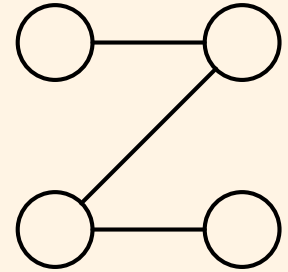
Undirected graph
 G .



Spanning tree
 T_1 .



Spanning tree
 T_2 .



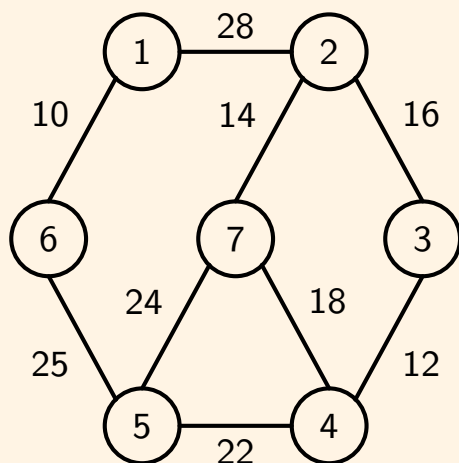
Spanning tree
 T_3 .

• Notes

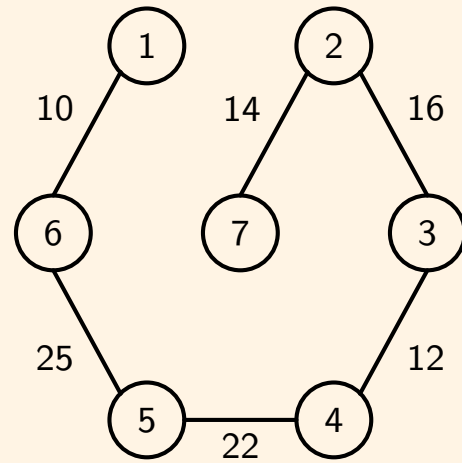
- Spanning tree is not unique.
- Spanning trees have $n - 1$ edges ($n = |V|$.)

Minimum-Cost Spanning Tree, Example

- In addition, there is a cost function associated with each edge, $w : E \rightarrow \mathbb{R}$.
- The cost of a tree is the sum of the costs of the tree edges.
- A **feasible solution** of the minimum-cost spanning tree of a undirected graph G is any spanning tree T of G .
- The **optimal solution** is a spanning tree with the minimum cost.



An undirected graph, G .



Minimum-cost spanning tree, T .

Minimum-Cost Spanning Tree, Generic Algorithm

- Using the greedy methodology, let T be a subset of a spanning tree, at each step an edge (u, v) is added to T to maintain the feasibility of the solution.
- An edge, (u, v) , is **safe** to a set of edges T if $T \cup \{(u, v)\}$ is still a subset of a spanning tree.
- The generic algorithm for the minimum-cost spanning tree then is:

Algorithm 5.2.5. Generic minimum-cost spanning tree

```
// Given a graph  $G(V, E)$  with cost function  $w$  find minimum cost spanning tree.
// Input:  $V, E, n, w$ 
// Output: minimum cost tree  $T$ .
1 Algorithm MCST( $V, E, n, w, T$ )
2 {
3      $T := \emptyset$ ;
4     while ( $|T| < n - 1$ ) do {
5         select an edge  $(u, v) \in E$  {
6             if  $(u, v)$  is safe to  $T$  then  $T := T \cup (u, v)$ ;
7              $E := E - \{(u, v)\}$ ;
8         }
9     }
10 }
```

- The key is in line 5, how to select an edge.

Minimum-Cost Spanning Tree, Prim's Algorithm

Algorithm 5.2.6. Prim

```
// Given a graph  $G(V, E)$  with cost function  $w$  find minimum cost spanning tree.
// Input:  $V, E, n, w$ 
// Output: minimum cost tree  $T$  and  $mincost$ .
1 Algorithm Prim( $V, E, n, w, T$ )
2 {
3     Find edge  $(k, \ell) \in E$  with the minimum cost ;
4      $mincost := w[k, \ell]$ ; //  $mincost$  set to minimum edge cost.
5      $T[1, 1] := k$ ; // Add  $(k, \ell)$  to spanning tree.
6      $T[1, 2] := \ell$ ;
7     for  $i := 1$  to  $n$  do // Init near array for every vertices.
8         if  $(w[i, \ell] < w[i, k])$  then  $near[i] := \ell$ ;
9         else  $near[i] := k$ ;
10     $near[k] := near[\ell] := 0$ ; // Vertices already in the spanning tree.
11    for  $i := 2$  to  $(n - 1)$  do {
12        Find  $j$  such that  $near[j] \neq 0$  and  $w[j, near[j]]$  is minimum ;
13         $T[i, 1] := j$ ; // Add minimum cost near edge to tree.
14         $T[i, 2] := near[j]$ ;
15         $mincost := mincost + w[j, near[j]]$ ; // Update  $mincost$ .
16         $near[j] := 0$ ; // Reset near array for selected vertex.
17        for  $k := 1$  to  $n$  do // update near array for the other unselected vertices.
18            if  $((near[k] \neq 0) \text{ and } (w[k, near[k]] > w[k, j]))$  then  $near[k] := j$ ;
19    }
20    return  $mincost$ ;
21 }
```

Minimum-Cost Spanning Tree, Prim's Algorithm II

- In Algorithm **Prim**

1. The edge with the minimum cost is first selected as the initial tree
2. The array **near** keeps the node already selected in the tree with the smallest single-edge cost for each node
3. Among the all the **near** edges, the minimum is selected and the node added to the tree
4. Array **near** is then updated and go back to step 3 until all nodes have been selected

- The time complexity is dominated by

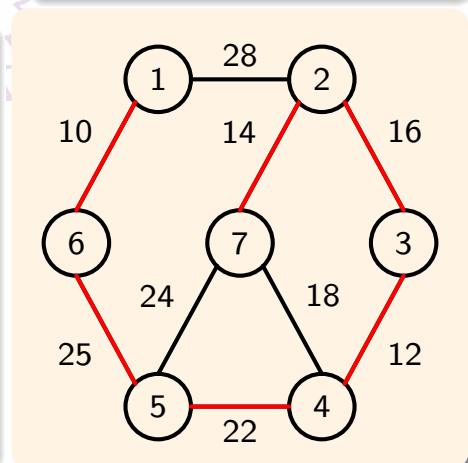
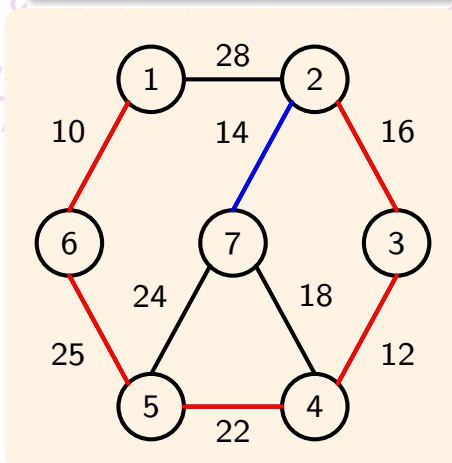
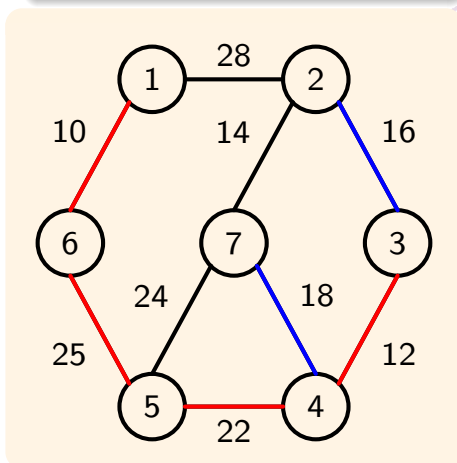
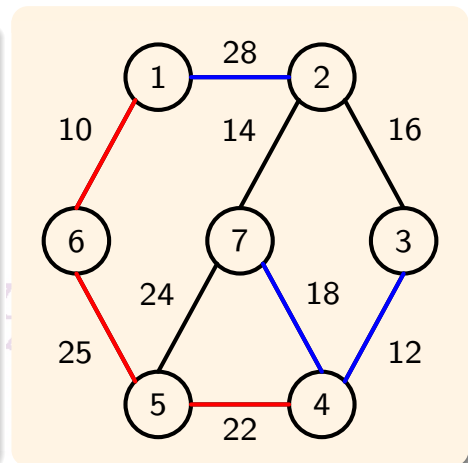
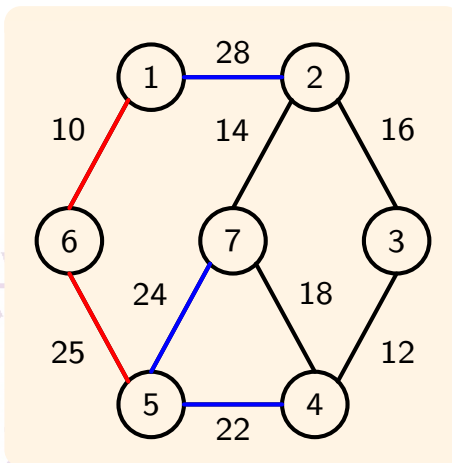
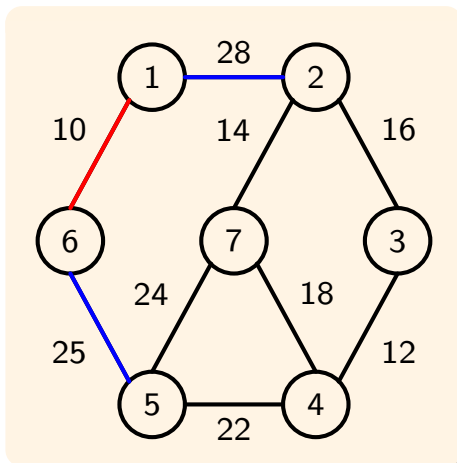
- Finding the minimum-cost edge on **line 3**, $\mathcal{O}(|E|) \approx \mathcal{O}(n^2)$
- Loop on **lines 7-9**, $\mathcal{O}(n)$
- Loop on **lines 11-19**
 - Inner loops **line 12** and **lines 17-18**
 - Complexity $\mathcal{O}(n^2)$

- Overall complexity is $\mathcal{O}(n^2)$

- The time complexity can be improved to $\mathcal{O}((n + |E|) \lg n)$

- If the non-selected vertices are stored in a red-black tree

Minimum-Cost Spanning Tree, Prim's Algorithm Example



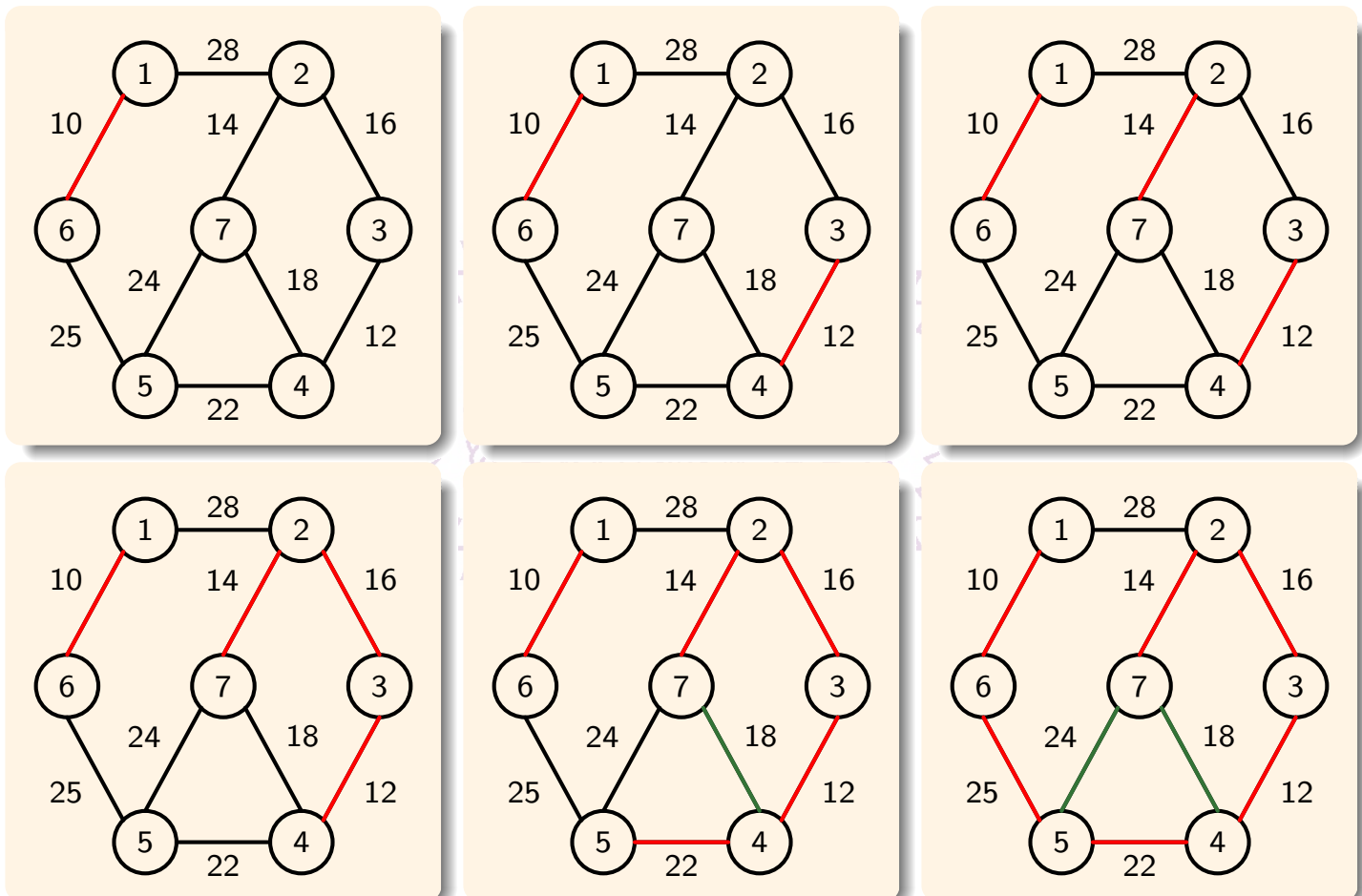
Kruskal's Algorithm – High Level

- A different approach to finding the minimum-cost spanning tree
- High level description of the algorithm

Algorithm 5.2.7. Kruskal's Algorithm

```
// Given a graph  $G(V, E)$  with cost function  $w$  find minimum cost spanning tree.
// Input:  $V, E, n, w$ 
// Output: minimum cost tree  $T$ .
1 Algorithm KruskalH( $V, E, n, w, T$ )
2 {
3      $T := \emptyset$ ;
4     while (( $T$  has less than  $(n - 1)$  edges) and ( $E \neq \emptyset$ )) do {
5         Find the edge  $(u, v) \in E$  with the minimum cost ;
6         Delete( $u, v$ ) from  $E$ ;
7         if  $(u, v)$  does not create a cycle in  $T$  then  $T := T \cup (u, v)$ ;
8         else discard  $(u, v)$ ;
9     }
10 }
```

Kruskal's Algorithm – Example



Kruskal's Algorithm

Algorithm 5.2.8. Kruskal's Algorithm

```
// Given a graph  $G(V, E)$  with cost function  $w$  find minimum cost spanning tree.
// Input:  $V, E, n, w$ 
// Output: minimum cost tree  $T$  and  $mincost$ .
1 Algorithm Kruskal( $V, E, n, w, T$ )
2 {
3     Construct a min heap from the edge costs using Heapify;
4     for  $i := 1$  to  $n$  do  $parent[i] := -1$ ; // Enable cycle checking
5      $i := 0$ ;
6      $mincost := 0$ ;
7     while ( $(i < n - 1)$  and (heap not empty)) do {
8         delete a minimum cost edge  $(u, v)$  from the heap ;
9         Adjust the heap ;
10         $j := Find(u)$ ; // using parent array
11         $k := Find(v)$ ;
12        if ( $j \neq k$ ) then {
13             $i := i + 1$ ;
14             $T[i, 1] := u$ ;
15             $T[i, 2] := v$ ;
16             $mincost := mincost + w[u, v]$ ;
17            Union( $j, k$ ); // modify parent array
18        }
19    }
20    if ( $i \neq n - 1$ ) then write("No spanning tree");
21    else return  $mincost$ ;
22 }
```

Kruskal's Algorithm – Complexity and Optimality

- The time complexity of Kruskal algorithm is dominated by the while loop, lines 7-19, – $\mathcal{O}(|E|)$
 - Line 8 finding minimum cost edge, $\mathcal{O}(1)$
 - Line 9 Adjust the heap, $\mathcal{O}(\lg |E|)$
 - Overall complexity $\mathcal{O}(|E| \lg |E|)$.

Theorem 5.2.9.

Kruskal's algorithm (Algorithm 5.2.8) generates a minimum-cost spanning tree for every undirected connected graph G .

- Proof please see textbook [Horowitz], p. 244.

Minimum-Cost Spanning Tree, Properties

- A different approach to prove Kruskal's algorithm.
- We define the following terms.
 - A **cut** $(S, V - S)$ of an undirected graph $G = (V, E)$ is a partition of V , i.e., $S \in V$.
 - An edge $(u, v) \in E$ is said to **cross** the cut $(S, V - S)$ if one of its end points is in S and the other in $V - S$.
 - A cut is said to **respect** a set T of edges if no edges in T crosses the cut.
 - An edge is said to be a **light edge** crossing a cut if its cost is the minimum of any edge crossing the cut.

Theorem 5.2.10.

Let $G = (V, E)$ be a connected, undirected graph with a cost function w defined on E . Let T be a subset of E that is subset of a spanning tree of G , let $(S, V - S)$ be any cut of G that respects T , and let (u, v) be a light edge crossing $(S, V - S)$. Then, edge (u, v) is safe for T .

- Proof please see textbook [Cormen], pp. 627-628.

Minimum-Cost Spanning Tree, Properties, II

Corollary 5.2.11.

Let $G = (V, E)$ be a connect, undirected graph with cost function w defined on E . Let T be a subset of E that is included in a minimum spanning tree of G , and let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_T = (V, T)$. If (u, v) is a light edge connecting C to some other component in G_T , then (u, v) is safe for T .

- Proof please see textbook [Cormen], pp. 629.
- Algorithm **Prim** can be shown to be a special case of Theorem (5.2.10), and it also returns an optimal solution.

- Matroid theory explains why greedy method solves the maximum/minimum subset problems.
 - The theory was developed by Hassler Whitney, American Mathematician, in 1935 by generalizing the structure of linear independence in vector space
 - Jack Edmonds, American Computer Scientist, applied to greedy algorithms
 - Reference: Bernhard Korte and Jens Vygen, *Combinatorial Optimization – theory and algorithms*, 4th edition, Springer, 2008.
- The concept of independence system is generalized from vector space.

Independent Systems

Definition 5.2.12. Independence System

Let S be a finite set and $\mathcal{I} = \{X : X \subseteq S\}$, then the set system (S, \mathcal{I}) is an **independence system** if

- (M1) $\emptyset \in \mathcal{I}$;
- (M2) If $Y \in \mathcal{I}$ and $X \subseteq Y$ then $X \in \mathcal{I}$.

The elements of \mathcal{I} are called **independent**, the elements of $2^S \setminus \mathcal{I}$ **dependent**. Minimal dependent sets are called **circuits**, maximal independent sets are called **bases**. For $X \subseteq S$, the maximal independent subsets of X are called bases of X .

- The set \mathcal{I} can be defined by its property, instead of listing all elements.

Definition 5.2.13.

Let (S, \mathcal{I}) be an independence system. For $X \subseteq S$ we define the **rank** of X by

$$r(X) = \max\{|Y| : Y \subseteq X, Y \in \mathcal{I}\}.$$

Examples of Independence Systems

- Example M1:

Let S_V be the set of columns of a matrix \mathbf{A} and $\mathcal{I}_V = \{X \subseteq S_V : \text{the column vectors in } X \text{ are linearly independent}\}$, then the set system (S_V, \mathcal{I}_V) is an independence system.

- (1) It is apparent $\emptyset \in \mathcal{I}$.

- (2) If $Y \in \mathcal{I}$ then any subset $X \subseteq Y$ also contains independent column vectors.

- Example M2:

Given a undirected graph $G(V, E)$, let $S_G = E$, the set of all edges, and $\mathcal{I}_G = \{Y : Y \subseteq E \text{ and } Y \text{ is a forest}\}$, then the set system (S_G, \mathcal{I}_G) is an independence system.

- (1) It is apparent $\emptyset \in \mathcal{I}$.

- (2) If $Y \in \mathcal{I}$, then Y is a forest, and any subset $X \subseteq Y$ is also a forest.

- Example M3:

Given any finite set S_U , let k be an integer, $k \leq |S_U|$, and $\mathcal{I} = \{Y : Y \subseteq S_U \text{ and } |Y| \leq k\}$, then the set system (S_U, \mathcal{I}_U) is an independence system.

- (1) It is apparent $\emptyset \in \mathcal{I}$.

- (2) If $Y \in \mathcal{I}$, then $|Y| \leq k$. Any $X \subseteq Y$ has $|X| \leq |Y| \leq k$.

Matroid

Definition 5.2.14. Matroid

An independence system (S, \mathcal{I}) is a **matroid** if

(M3) If $X, Y \in \mathcal{I}$ and $|X| > |Y|$, then there is an $x \in X \setminus Y$ such that $Y \cup \{x\} \in \mathcal{I}$.

- **Vector Matroid:** The independence system (S_V, \mathcal{I}_V) in Example M1 is a matroid.

- (M3) property is observed (S_V, \mathcal{I}_V) .

- **Graphic Matroid:** The independence system (S_G, \mathcal{I}_G) in Example M2 is a matroid.

- (M3) property is observed. If $|X| > |Y|$ and for every $x \in X$ either $x \in Y$ or $Y \cup \{x\}$ forms a cycle. In either case, both vertices of the edge x belong to the same connected component in Y . The number of such edge cannot exceed $|Y|$ while maintaining a forest property, thus $|X| \leq |Y|$. This contradicts to the assumption $|X| > |Y|$.

- **Uniform Matroid:** The independence system (S_U, \mathcal{I}_U) in Example M3 is a matroid.

- (M3) property is observed. If $|X| > |Y|$ then there is an $x \in X \setminus Y$ and then $|Y \cup \{x\}| = |Y| + 1 \leq |X| \leq k$.

Theorem 5.2.15.

Let (S, \mathcal{I}) be an independence system. Then the following statements are equivalent:

- (M3) If $X, Y \in \mathcal{I}$ and $|X| > |Y|$, then there is an $x \in X \setminus Y$ with $Y \cup \{x\} \in \mathcal{I}$.
- (M3') If $X, Y \in \mathcal{I}$ and $|X| = |Y| + 1$, then there is an $x \in X \setminus Y$ with $Y \cup \{x\} \in \mathcal{I}$.
- (M3'') For each $X \subseteq S$, all bases of X have the same cardinality.

Proof. It is easy to see $(M3) \Leftrightarrow (M3')$ and $(M3) \Rightarrow (M3'')$. To prove $(M3'') \Rightarrow (M3)$, let $X, Y \in \mathcal{I}$ and $|X| > |Y|$. By $(M3'')$, Y cannot be a basis of $X \cup Y$. So there must be an $x \in (X \cup Y) \setminus Y = X \setminus Y$ such that $Y \cup \{x\} \in \mathcal{I}$. \square

- Thus, an independence system can also be shown to be a matroid using either property $(M3')$ or $(M3'')$.
- For the graphic matroid, it is known that a spanning tree of a connect graph $G(V, E)$ has $|V| - 1$ edges. Thus, the rank $r(S_G) = |S_G| - 1$ if G is connected.

Weighted Matroid and Optimization Problems

Definition 5.2.16. Weighted Matroid

A matroid (S, \mathcal{I}) is **weighted** if it is associated with a weight function $w : S \rightarrow \mathbb{R}^+$. The weight function w extends to subsets of S by summation:

$$w(X) = \sum_{x \in X} w(x) \quad \text{for any } X \subseteq S. \quad (5.1)$$

- **Maximization problem of independence systems**
Given an independence system (S, \mathcal{I}) and the weight function $w : S \rightarrow \mathbb{R}^+$, find an $X \in \mathcal{I}$ such that $w(X) = \sum_{x \in X} w(x)$ is maximum.
- A corresponding minimization can be formulated
 - Solution algorithms can also be derived.

Greedy Algorithms

- Two types of algorithms possible
The first one is

Algorithm 5.2.17. Best-In Greedy Algorithm

```
// Given  $(S, \mathcal{I})$  and  $w : S \rightarrow \mathbb{R}$  find  $X \in \mathcal{I}$  such that  $w(X)$  is maximum.
// Input:  $(S, \mathcal{I})$  and  $w$ .
// Output:  $X$ 
1 Algorithm Best-In-Greedy( $S, \mathcal{I}, w$ )
2 {
3     Sort  $S$  into nonincreasing order by  $w$  ;
4      $X := \emptyset$ ; // Initialize to empty set.
5     for each  $x \in S$  in order do { // Try all elements.
6         if  $(X \cup \{x\} \in \mathcal{I})$  then { // Maintain independence then add.
7              $X := X \cup \{x\}$ ;
8         }
9     }
10    return  $X$ ;
11 }
```

Greedy Algorithms, II

Theorem 5.2.18.

The best-in greedy algorithm (5.2.17) solves the independence system (S, \mathcal{I}) maximization problem correctly if (S, \mathcal{I}) is a matroid.

Proof. By induction. The first x in the ordered S with $\{x\} \in \mathcal{I}$ is apparently the solution for any $X \subset S$ with $r(X) = 1$. Suppose the best solution has been found for any $X \subset S$ and $r(X) = k$ and $k < r(S)$, then there is a $Y \in \mathcal{I}$ such that $r(Y) = r(X) + 1$, further more there is $x \in S$ with $Y = X \cup \{x\}$ and x is in the remaining S . The x with largest $w(x)$ is apparently the choice, which would be picked first by the algorithm. Hence, this Y is the solution for $r(X) + 1$. By induction, the theorem is proven. \square

- Note that the maximization problem is defined for independence systems, and the algorithm works if the system is a matroid.
- The graph minimum spanning tree problem can be formulated as a minimization problem for a matroid system.
- The Kruskal's Algorithm is a best-in greedy algorithm.

- Alternative solution

Algorithm 5.2.19. Worst Out Greedy Algorithm

```

// Given  $(S, \mathcal{I})$  and  $w : S \rightarrow \mathbb{R}$  find a basis  $X$  of  $S$  such that  $w(X)$  is maximum.
// Input:  $(S, \mathcal{I})$  and  $w$ .
// Output:  $X$ 
1 Algorithm Worst-Out-Greedy( $S, \mathcal{I}, w$ )
2 {
3     Sort  $S$  into nondecreasing order by  $w$  ;
4      $X := S$ ; // Initialize to entire set.
5     for each  $x \in S$  in order do { // Try all elements.
6         if  $(r(X \setminus \{x\}) = r(X))$  then { // rank unchanged.
7              $X := X \setminus \{x\}$ ;
8         }
9     }
10    return  $X$ ;
11 }
```

- This algorithm can generate optimal solution if the given system is a matroid, and the proof is similar to the best-in greedy algorithm.

Job Sequencing with Deadlines

- Given a set of n jobs to be processed on one machine.
 - Each job takes 1 time unit to process.
 - Associated with job i , $1 \leq i \leq n$, there is a deadline d_i and profit p_i .
 - If job i is completed by d_i then p_i is earned.
- A feasible solution is a subset J of jobs that each job in J can be completed by its deadline.
 - The value of the subset J is $\sum_{i \in J} p_i$.
- An optimal solution is a feasible solution with the maximum value.

- Example, $n = 4$,
 $\{p_1, p_2, p_3, p_4\} = \{100, 10, 15, 27\}$,
 $\{d_1, d_2, d_3, d_4\} = \{2, 1, 2, 1\}$.

- Feasible solutions are

	Feasible solution	Processing sequence	Value
1	$\{1, 2\}$	2,1	110
2	$\{1, 3\}$	1,3 or 3,1	115
3	$\{1, 4\}$	4,1	127
4	$\{2, 3\}$	2,3	25
5	$\{3, 4\}$	4,3	42
6	$\{1\}$	1	100
7	$\{2\}$	2	10
8	$\{3\}$	3	15
9	$\{4\}$	4	27

- Solution 3 is optimal.

Job Sequencing with Deadlines – Algorithm

- Applying greedy method to job sequencing problem

Algorithm 5.2.20. Job Sequencing – Greedy Method

```
// Solve job scheduling problem with jobs sorted in non-increasing profit.
// Input: int  $n$ , deadline  $d[1 : n]$ , profit  $p[1 : n]$ 
// Output: Optimal sequence  $J[1 : k]$ .
1 Algorithm JSgreedy( $n, d, p, J$ )
2 {
3      $J := \{1\}$ ; // init to highest profit job
4     for  $i := 2$  to  $n$  do { // check every job
5         if ( $J \cup \{i\}$  is feasible ) then
6              $J := J \cup \{i\}$ ;
7     }
8 }
```

- Example

Ordered sequence: $\{1, 4, 3, 2\}$, $p = \{100, 27, 15, 10\}$,

step 1: $J = \{1\}$, $p = 100$,

step 2: $J = \{1, 4\}$, $p = 127$,

step 3: reject job 3, not feasible, step 4: reject job 2, not feasible.

Job Sequencing with Deadlines – Algorithm Optimality

- Optimality of the algorithm follows from the following theorem.

Theorem 5.2.21.

Given a job sequencing problem Algorithm JSgreedy (Algorithm 5.2.20) generates an optimal solution.

- Note that there can be groups of jobs that only some of them can be selected as $\{1, 2\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, and $\{3, 4\}$ in the example. Follows the algorithm, $\{1, 4\}$ are selected. The only possibility that this solution is not optimal is that $P\{1, 4\}$ is smaller than $P\{2, 3\}$. But, job 4 has higher profit than either job 2 or 3. Thus, the selection by JSgreedy is the optimal solution.
- Unit execution time of each job is an important factor.

Algorithm 5.2.22. Job Sequencing

```

// Solve job scheduling problem with jobs sorted in non-increasing profit.
// Input: int n, deadline d[1 : n], profit p[1 : n]
// Output: Optimal sequence J[1 : k].
1 Algorithm JS(n, d, p, J)
2 {
3     d[0] := J[0] := 0; // to facilitate while loop stopping
4     J[1] := 1; k := 1; // init to highest profit job
5     for i := 2 to n do { // check every job
6         r := k;
7         while ((d[J[r]] > d[i]) and (d[J[r]] ≠ r)) do // find time slot job i fits
8             r := r - 1;
9         if ((d[J[r]] ≤ d[i]) and (d[i] > r)) then { // insert i into J
10             for q := k to (r + 1) step -1 do J[q + 1] := J[q]; // move jobs to make room
11             J[r + 1] := i; // assign job
12             k := k + 1;
13         }
14     }
15 }
    
```

- Line 3 creates a stopping condition for the while loop on line 7.
- The worst-case time complexity of JS algorithm is $\mathcal{O}(n^2)$.
 - Outer loop, lines 5–14
 - Inner loops, lines 7–8 and line 10.
- The space complexity of JS algorithm is $\mathcal{O}(n)$ for arrays J , p , and d .

Job Sequencing with Deadlines – Algorithm correctness

Theorem 5.2.23.

Given a job sequencing problem Algorithm JS (Algorithm 5.2.22) correctly generates the optimal solution.

- Proof can be found in textbook [Horowitz] pp. 230 - 232.
- Example: $n = 5$, Jobs = {A, B, C, D, E}, $p[] = \{20, 15, 10, 5, 1\}$, and $d[] = \{2, 2, 1, 3, 3\}$. Then, the execution sequence of the algorithm is as following.

i	$d[i]$	action	$J[]$	$d[J[]]$	$p[J[]]$	k
1	2	init to A	{A}	{2}	{20}	1
2	2	accepting B	{A, B}	{2, 2}	{20, 15}	2
3	1	rejecting C	{A, B}	{2, 2}	{20, 15}	2
4	3	accepting D	{A, B, D}	{2, 2, 3}	{20, 15, 5}	3
5	3	rejecting E	{A, B, D}	{2, 2, 3}	{20, 15, 5}	3

Job Sequencing with Deadlines – Matroid Formulation

- The job sequencing with deadline can be shown to be a matroid.
The set \mathcal{S} contains all the jobs, and a set A of jobs are independent if there is a schedule such that all jobs in A are done before their deadlines.

Lemma 5.2.25.

For any set of jobs A , the following statements are equivalent.

1. The set A is independent.
2. Let $N_t(A)$ denote the number of jobs completed before time t , then for $t = 0, 1, 2, \dots, n$, we have $N_t(A) \leq t$.
3. If the tasks in A are scheduled in order of monotonically increasing deadlines, the all jobs in A are completed before their deadlines.

Theorem 5.2.26.

If \mathcal{S} is a set of unit-time jobs with deadlines, and \mathcal{I} is the set of all independent sets of tasks, then the corresponding system $(\mathcal{S}, \mathcal{I})$ is a matroid.

- Since the job sequencing problem is a matroid, the greedy algorithm can be applied and it results in an optimal solution.

Summary

- Tree vertex splitting problem.
- Minimum-cost spanning tree problem.
- The theory of Matroid.
- Job sequencing with deadlines.