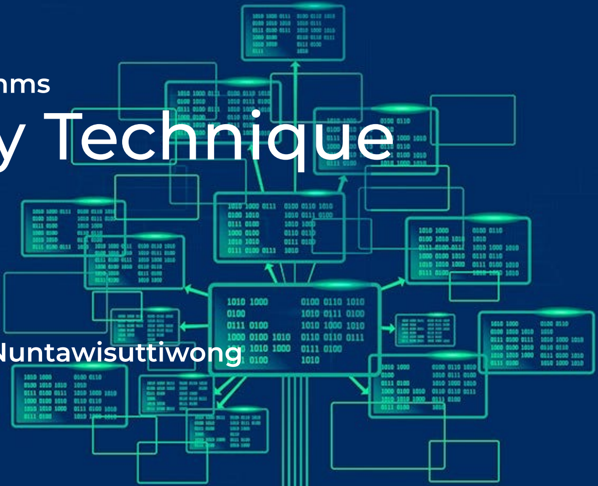
CPE 231 Algorithms

Greedy Technique



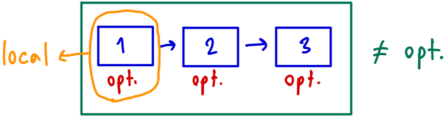
Dr. Taweechai Nuntawisuttiwong

Contents

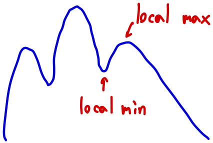
1. Greedy Technique
2. Prim’s Algorithm
3. Kruskal’s Algorithm
4. Huffman Trees and Codes

Greedy Technique

Greedy Algorithms



- Greedy algorithms solve problems by making a sequence of choices that seem the best at the moment.
- Construct a solution step-by-step, making the locally optimal choice at each step.
  - Feasible: Must satisfy the problem's constraints.
  - Locally Optimal: The best choice among all feasible ones at the step.
  - Irrevocable: Once made, choices can't be changed.



The Change-Making Problem

- **Problem**: Give change for an amount using the least number of coins.
- **Strategy**: Use the highest denomination available.

Example: Coin denomiations are $d_1 = 25$ (quarter), $d_2 = 10$ (dime), $d_3 = 5$ (nickel), and $d_4 = 1$, (penny). How would you give changes with coins of these denominations of 48 cents?

n	$d_1 = 25$	$d_2 = 10$	$d_3 = 5$	$d_4 = 1$	
48	0	0	0	0	
48-25 = 23	1	0	0	0	
23-10 = 13	1	1	0	0	
13-10 = 3	1	2	0	0	
3-1 = 2	1	2	0	1	
2-1 = 1	1	2	0	2	
1-1 = 0	1	2	0	3	← Solution # coins = 6

The Change-Making Problem

Example: Coin denomiations are $d_1 = 25$ (quarter), $d_2 = 10$ (dime), and $d_3 = 1$, (penny). How would you give changes with coins of these denominations of 30 cents?

n	$d_1 = 25$	$d_2 = 10$	$d_3 = 1$	
30	0	0	0	
30 − 25 = 5	1	0	0	
5 − 1 = 4	1	0	1	
4 − 1 = 3	1	0	2	
3 − 1 = 2	1	0	3	Optimal solution → # coins = 3
2 − 1 = 1	1	0	4	
1 − 1 = 0	1	0	5	← Solution # coins = 6

Remark: Greedy isn't always globally optimal.

Bruteforce → แก้ปัญหาทั่วไป

Greedy → แก้ปัญหาพวก optimization problem

Applications in Graph Theory

- Minimum Spanning Trees (MST)
  - Prim’s Algorithm
  - Kruskal’s Algorithm
  - Both solve MST problems optimally using the greedy approach.
- Shortest path
  - Dijkstra’s Algorithm
  - Greedily pick the nearest unvisited node.

Optimality Proof Techniques

Inductive Proof Method

- This method uses mathematical induction to prove that a partially constructed solution can always be extended to an optimal solution.
- Example:
  - Consider Prim's Algorithm for Minimum Spanning Trees.
  - **Base Case**: Start with a single vertex and add the shortest edge.
  - **Inductive Step**: Assume the current solution is part of the optimal spanning tree. The next added edge must connect to the nearest vertex, ensuring that each step keeps the solution optimal.
- Demonstrate that the partial greedy solution is extendable to an optimal one.

Optimality Proof Techniques

Distance-Based Method

- This method involves showing that each step taken by the greedy algorithm is at least as effective as any other algorithm in achieving the problem's objective.
- Example: Knight's Minimum Moves Problem
  - Find the minimum number of moves needed for a chess knight to reach the diagonally opposite corner on a 100x100 board.
  - On each move, jump as close as possible to the target using L-shaped moves.
  - By tracking the "Manhattan distance" to the target, which is the sum of horizontal and vertical distances, the greedy algorithm always reduces this distance by 3 on each move, which is the best possible.
  - Since no other sequence of moves can reduce the distance faster, this greedy solution is optimal.
- Show that each greedy step optimally advances toward the final goal.

Optimality Proof Techniques

Result-Based Proof Method

- This technique proves optimality by evaluating the final output of the algorithm rather than focusing on each individual step.
- Example: Chip Placement Problem
  - Place the maximum number of chips on an 8x8 board such that no two chips are adjacent.
  - Start at one corner of the board and place each new chip to maximize available space for future placements.
  - The board can be divided into sixteen 4x4 sections. Since it is impossible to place more than one chip in each of these sections without having adjacent chips, the total number of chips cannot exceed 16.
  - The greedy approach places exactly 16 chips, which matches the proven upper bound, confirming that this solution is optimal.
- Analyze the final solution and compare it with theoretical limits.

Prim’s Algorithm

Minimum Spanning Tree

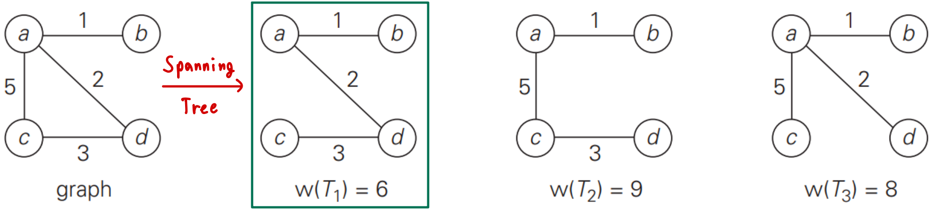
Graph → Tree (Acyclic Graph) DFS, BFS

Spanning Tree no weight

A connected acyclic subgraph that includes all vertices.

Minimum Spanning Tree (MST) weighted graph

A spanning tree with the smallest possible total weight.



graph

$w(T_1) = 6$

$w(T_2) = 9$

$w(T_3) = 8$

↓ minimum spanning tree

Minimum Spanning Tree

Given a connected weighted graph $G = (V, E)$, find a subset of edges $T \subseteq E$ such that:

- T connects all vertices (spanning tree). → Spanning Tree
- The sum of edge weights is minimal → Minimum

Applications

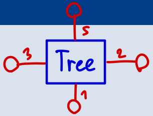
- Connecting cities with minimum total road cost
- Designing computer or communication networks efficiently
- Clustering data points in machine learning

Prim's Algorithm

Greedy Strategy

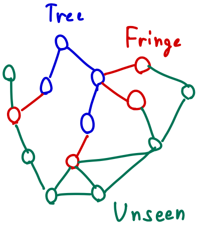
- Always choose the edge with the **minimum weight** connecting the current tree to a new vertex.
- No need to reconsider past choices.

Algorithm Overview



1. Start with an arbitrary vertex.
2. Repeatedly:
   - Find the smallest-weight edge connecting the tree to a vertex outside it.
   - Add this edge and vertex to the tree.
3. Stop when all vertices are in the tree.

Prim’s Algorithm



- Each vertex maintains:
    - Nearest tree vertex
    - Distance label (edge weight)
    - ∞ (infinity) for unreachable vertices
- Fringe & Unseen Vertices:
    - *Fringe*: Adjacent to tree, candidates for next vertex
    - *Unseen*: Not yet connected

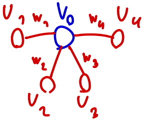
Pseudocode

```
ALGORITHM  $Prim(G)$
    //Prim's algorithm for constructing a minimum spanning tree
    //Input: A weighted connected graph $G = \langle V, E \rangle$
    //Output: $E_T$, the set of edges composing a minimum spanning tree of $G$
    $V_T \leftarrow \{v_0\}$    //the set of tree vertices can be initialized with any vertex
    $E_T \leftarrow \varnothing$
    for $i \leftarrow 1$ to $|V| - 1$ do
        find a minimum-weight edge $\boxed{e^* = (v^*, u^*)}$ among all the edges $\boxed{(v, u)}$
        such that $v$ is in $V_T$ and $u$ is in $\underbrace{V - V_T}_{\text{Vertices ที่ไม่อยู่ใน tree}}$
        $V_T \leftarrow V_T \cup \{u^*\}$
        $E_T \leftarrow E_T \cup \{e^*\}$
    return $E_T$
```

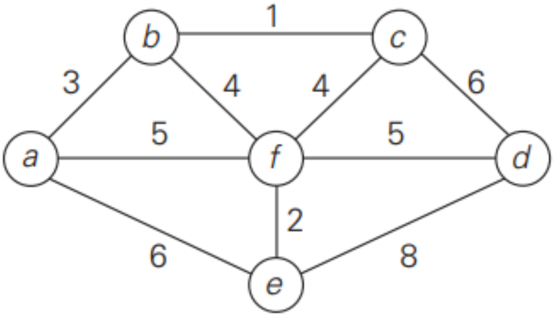
edge ที่ต่อกับ tree

fringe

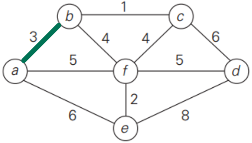
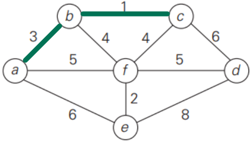
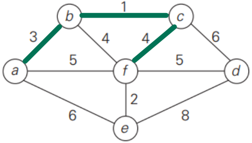
edge ที่มี minimum weight



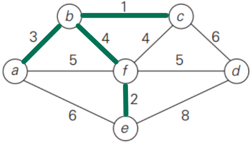
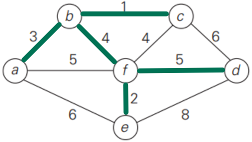
Example



Example

Tree vertices	Remianing vertices	Illustration
$\overset{u}{a}(\overset{v}{-},\overset{w}{-})$	$\boxed{b(a,3)}$, $c(-,\infty)$, $d(-,\infty)$ $e(a,6)$, $f(a,5)$	 a b c d e f 1 3 4 4 6 5 5 2 6 8
$b(a,3)$	$\boxed{c(b,1)}$, $d(-,\infty)$, $e(a,6)$, $f(b,4)$	 a b c d e f 1 3 4 4 6 5 5 2 6 8
$c(b,1)$	$d(c,6)$, $e(a,6)$, $\boxed{f(b,4)}$	 a b c d e f 1 3 4 4 6 5 5 2 6 8

Example

Tree vertices	Remianing vertices	Illustration
f(b,4)	d(f,5) , $\boxed{e(f,2)}$	 b c a f d e 1 3 4 4 6 5 5 2 6 8
e(f,2)	$\boxed{d(f,5)}$	 b c a f d e 1 3 4 4 6 5 5 2 6 8

Time Complexity

Priority Queue → Heap ↗ Max-heap → Parent > Children

↘ Min-heap → Parent < Children

Graph Representation	Priority Queue	Time Complexity
Weight matrix	Unordered array	$O(V^2)$
Adjacency list	Min-heap	$O(E \log V)$

Kruskal’s Algorithm

Kruskal’s Algorithm

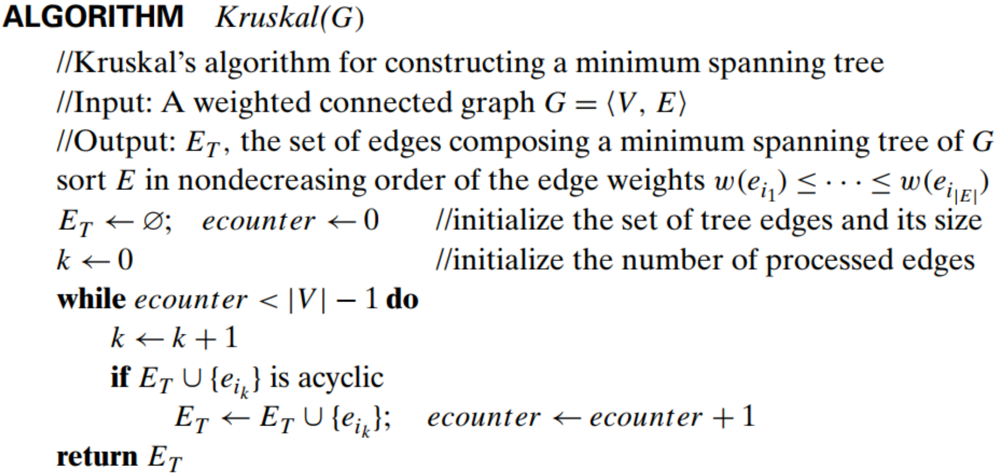
- Greedy algorithm that constructs an MST edge by edge, not vertex by vertex.
- Keep edges with smallest weight while avoiding cycles.
- Start with an empty graph and keep adding edges in order of increasing weight until all vertices are connected.

Kruskal's Algorithm

Algorithm Overview

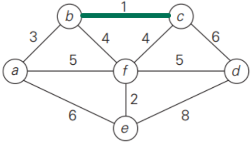
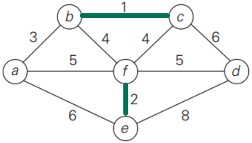
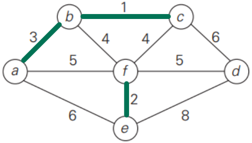
1. Sort all edges in nondecreasing order of weights.
2. Start with an empty edge set $E_T = \emptyset$.
3. For each edge (u, v) in the sorted list:
   - If adding (u, v) does not create a cycle, include it in E_T.
   - Otherwise, skip it.
4. Stop when $|E_T| = |V| - 1$.

Pseudocode

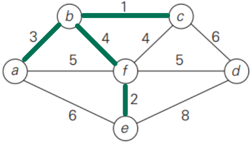
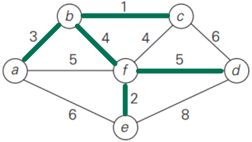


```
ALGORITHM Kruskal(G)
    //Kruskal's algorithm for constructing a minimum spanning tree
    //Input: A weighted connected graph $G = \langle V, E \rangle$
    //Output: $E_T$, the set of edges composing a minimum spanning tree of $G$
    sort $E$ in nondecreasing order of the edge weights $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$
    $E_T \leftarrow \varnothing$;   $ecounter \leftarrow 0$       //initialize the set of tree edges and its size
    $k \leftarrow 0$                             //initialize the number of processed edges
    while $ecounter < |V| - 1$ do
        $k \leftarrow k + 1$
        if $E_T \cup \{e_{i_k}\}$ is acyclic
            $E_T \leftarrow E_T \cup \{e_{i_k}\}$;   $ecounter \leftarrow ecounter + 1$
    return $E_T$
```

Example

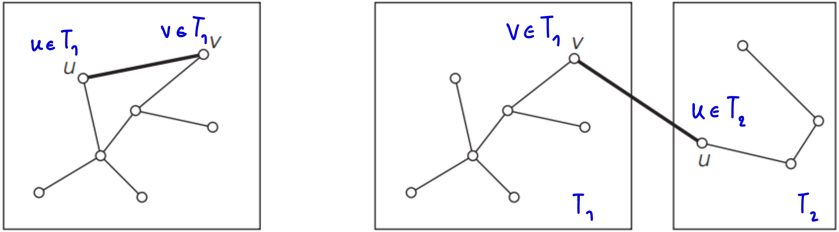
Tree edges	Sorted list of edges	Illustration
	bc 1 ef 2 ab 3 bf 4 cf 4 af 5 df 5 ae 6 cd 6 de 8	
bc 1	~~bc 1~~ ef 2 ab 3 bf 4 cf 4 af 5 df 5 ae 6 cd 6 de 8	
ef 2	~~bc 1~~ ~~ef 2~~ ab 3 bf 4 cf 4 af 5 df 5 ae 6 cd 6 de 8	

Example

Tree edges	Sorted list of edges	Illustration
ab 3	~~bc 1~~ ~~ef 2~~ ~~ab 3~~ bf 4 cf 4 af 5 df 5 ae 6 cd 6 de 8	
bf 4	~~bc 1~~ ~~ef 2~~ ~~ab 3~~ ~~bf 4~~ cf 4 af 5 df 5 ae 6 cd 6 de 8	
df 5		

Cycle Detection

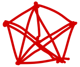
A new edge (u, v) creates a cycle if both u and v are in the same connected component.

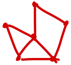


Union-Find Algorithm

Prim vs Kruskal

Feature	Prim's Algorithm	Kruskal's Algorithm
Approach	Expand one tree	Merge many trees
Edge Selection	Smallest edge connecting to tree	Smallest edge globally
Data Structure	Min-Heap	Union-Find
Best for	* edges มาก Dense graphs	* edge น้อย Sparse graphs
Complexity	$O(E \log V)$	$O(E \log E)$





Huffman Trees and Codes

Huffman Coding

CPE → 4x3 bits
↓
C x P x E → 3×3×2

ASCII → 8 bits

i มาก h น้อย

- Huffman coding is a variable-length encoding algorithm used to minimize the number of bits needed for storing or transmitting data.
- Named after David Huffman, who invented it while he was a graduate student.
- Similar to Morse Code, where frequently used letters (like "e" (·) or "a" (·–)) have shorter sequences compared to less common ones (like "q" (– – ·–) or "z" (– – ··)).
- Application:
  - Text compression
  - multimedia encoding (JPEG, MP3)
  - many other data compression techniques.

00 → 0
01 → 1
11 → 3
10 → 2

a, e, i, o, u → 2 bits

ที่เหลือ ยกเว้น q, z → 3 bits

q, z → 4 bits

Fixed vs Variable-Length Encoding

Fixed-Length Encoding:

- ASCII encoding assigns a fixed 7 or 8-bit length to every character.
- Not efficient for characters that appear more frequently.

Variable-Length Encoding:

- Shorter codewords for frequent symbols to save space.
- How to decode efficiently? Without structure, codewords can be ambiguous.
    - If "A" is encoded as "0" and "B" as "01", how do we know if "010" is "A-B" or "B-A"?

Prefix-Free Codes

0010 | 1011 | 0101   fixed-length

0010 11010101   variable-length?

- A prefix-free code ensures no codeword is a prefix of another. This means that each symbol can be uniquely identified from a bitstream.
    - Encoding symbols “A”, “B”, “C” as "0", "10", "11" respectively is prefix-free.
- Enables straightforward decoding of variable-length encoded data.
- Binary Tree Representation:
    - Symbols are represented as leaves in a binary tree.
    - The path to each leaf from the root provides the codeword.

Constructing Huffman Trees

Frequency → เลือก freq. น้อยสุด 2 ตัว มาสร้าง Tree → update freq

Huffman's Algorithm

1. Initialize n one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's **weight**. (More generally, the weight of a tree will be equal to the sum of the frequencies in the tree's leaves.)
2. Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight. Make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight.

Remark: This is a greedy algorithm because, at every step, it selects the two smallest elements.

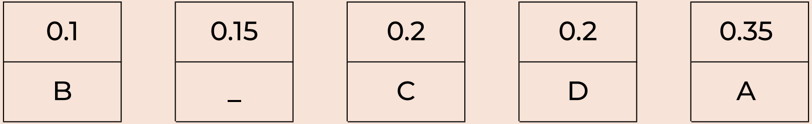
ref. text → count alphabet → frequency

Huffman Tree Construction

Example: Consider the five-symbol alphabet {A, B, C, D, _} with the following occurrence frequencies in a text made up of these symbols

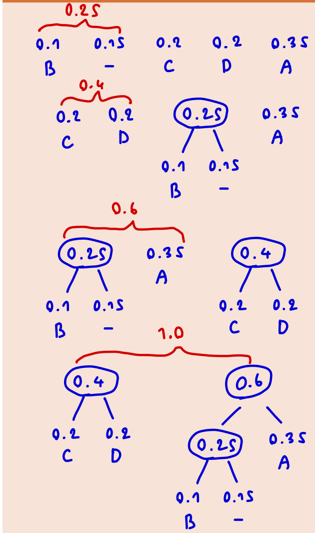
symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

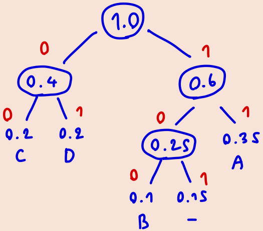
น้อย → มาก



0.1	0.15	0.2	0.2	0.35
B	_	C	D	A

Huffman Tree Construction





Huffman Tree Construction

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

DAD is encoded as 011101

100|11|01|101|11|01 is decoded as BAD_AD

Compression Ratio

$5 \text{ alphabets} \xrightarrow{\text{encode}}$

10 data → 4 bits

- A measure of how much storage space is saved using compression.
- Fixed-length encoding requires __3__ bits/symbol.
- Huffman code for the given example requires

max → 99 → 7 bits

$$\overset{A}{2(0.35)} + \overset{B}{3(0.1)} + \overset{C}{2(0.2)} + \overset{D}{2(0.2)} + \overset{-}{3(0.15)} = 2.25 \text{ bits/symbol}$$

Expected Value

- Compression Ratio:

fixed

$$\frac{3-2.25}{3} \times 100\% = 25\%$$

ใช้ memory น้อยลง 25 % เมื่อเทียบกับ fixed-length encoding

Compression Ratio

- A measure of how much storage space is saved using compression.
- Fixed-length encoding requires ______ bits/symbol.
- Huffman code for the given example requires

- Compression Ratio: