# CS483 Analysis of Algorithms
# Lecture 06 – Greedy Algorithms *

Jyh-Ming Lien

February 26, 2009

*this lecture note is based on *Algorithms* by S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani and *Introduction to the Design and Analysis of Algorithms* by Anany Levitin.

# Greedy Algorithm



- Greedy algorithm is algorithm that makes the **locally optimal** choice at each stage with the hope of finding the global optimum
- Greedy algorithm never changes the choices that have been made
- Example: How do you compute the minimum number of US coins ($25, $10, $5, $1) to give while making change of 43 cents?

- Advantages
  - Simple and Intuitive
  - Work for problems such as **minimum spanning tree**, **shortest path problem**, and **data compression**.
- Disadvantages
  - Be very careful when use it. May not work for many problems
  - But still provide good approximate solution

# Minimum Spanning Tree (MST)



- Given a set of points (cities), how do you use the minimum amount of wire to connect these points?
- Example:

- Given a graph $G$, a **spanning tree** $T$ of G is a subgraph of $G$ that contains all vertices of $G$
- The **minimum spanning tree** $MST$ of $G$ is a spanning tree of $G$ of the smallest weight
- MST has many applications including: clustering, good approximation to traveling salesman problem, point connecting problem

# Prim's Algorithm

- Algorithm

**Algorithm 0.1:** PRIM($G = \{V, E\}$)

$x \leftarrow$ a random vertex from $G$
$V_{MST} \leftarrow \{x\}$
$E_{MST} \leftarrow \emptyset$
**for** $i \leftarrow \{1 \cdots |V| - 1\}$
**do** $\begin{cases} \text{find the minimum weight edge } e = \{u, v\} \\ \text{such that } u \in V_{MST} \text{ and } v \in V - V_{MST} \\ V_{MST} \leftarrow V_{MST} \cup v \\ E_{MST} \leftarrow E_{MST} \cup e \end{cases}$

- Example:

# Prim’s Algorithm



- Why is Prim’s algorithm correct?
- How to implement the first statement in Prim’s algorithm?
- What is the time complexity of this implementation?

# Kruskal's Algorithm



- Algorithm

**Algorithm 0.2:** KRUSKAL($G = \{V, E\}$)

Sort $E$ from small to large
$E_{MST} \leftarrow \emptyset$
**while** $|E_{MST}| < |V| - 1$
**do** $\begin{cases} \textbf{if } E_{MST} \cup E_i \text{ is acyclic} \\ \quad \textbf{then } E_{MST} \leftarrow E_{MST} \cup E_i \\ i \leftarrow i + 1 \end{cases}$

- Example:

# Kruskal’s Algorithm



- ☐ Why is Kruskal’s algorithm correct?

- ☐ How to check the acyclic property in Kruskal’s algorithm efficiently?

- ☐ What is the time complexity of this implementation?

# Huffman codes



- Any binary tree with edges labeled with 0s and 1s yields a prefix-free code of characters assigned to its leaves

- Optimal binary tree minimizing the expected (weighted average) length of a codeword can be constructed as follows

- Huffman's algorithm

**Algorithm 0.3:** HUFFMAN($W[1 \cdots n]$)

Build $n$ one-node tress $from$T
**for** $i \leftarrow \{1 \cdots n-1\}$
  **do** { Find two smallest tree $T_1, T_2$
     Merge $T_1, T_2$ using a root with weight $W(T_1) + W(T_2)$
**for each** node
  **do** mark left edge 0 and mark right edge 1

- Time complexity:

# Example



- Encode the following text (ignore cases and all the numbers):
  “Edsger Wybe Dijkstra was a Dutch computer scientist. He received the 1972 A. M. Turing Award for fundamental contributions in the area of programming languages, and was the Schlumberger Centennial Chair of Computer Sciences at The University of Texas at Austin from 1984 until his death in 2002”. — wikipedia

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s |
| | | | | | | | | | | | | | | | | | | |
| t | u | v | w | x | y | z | ( | ) | . | _ | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |

# Horn's formula



- Horn's formula is used for representing knowledge and has the following format:
  - Implications: $(z \wedge w) \Rightarrow u$
  - Pure negative clauses: $(\bar{u} \vee \bar{v} \vee \bar{y})$
- Horn-satisfiability (Horn-SAT)
  - Find the values for all literals that satisfy the assignment
  - Example: $(w \wedge y \wedge z) \Rightarrow x; (x \wedge z) \Rightarrow w; x \Rightarrow y; \Rightarrow x; (x \wedge y) \Rightarrow w; (\bar{w} \vee \bar{x} \vee \bar{y}); (\bar{z})$
  - Answer: x= , y= , z= , w=

# Horn's formula



- Horn-SAT can be solved in polynomial time.

**Algorithm 0.4:** HORNSAT($HornFormula$)

set all variable to false
**while** there is an implication that is not satisfied
  **do** set the right-hand variable of the implication to ture
**if** all pure negative clauses are satisfied
  **then return** (the assignment)
  **else return** (formula is not satisfiable)

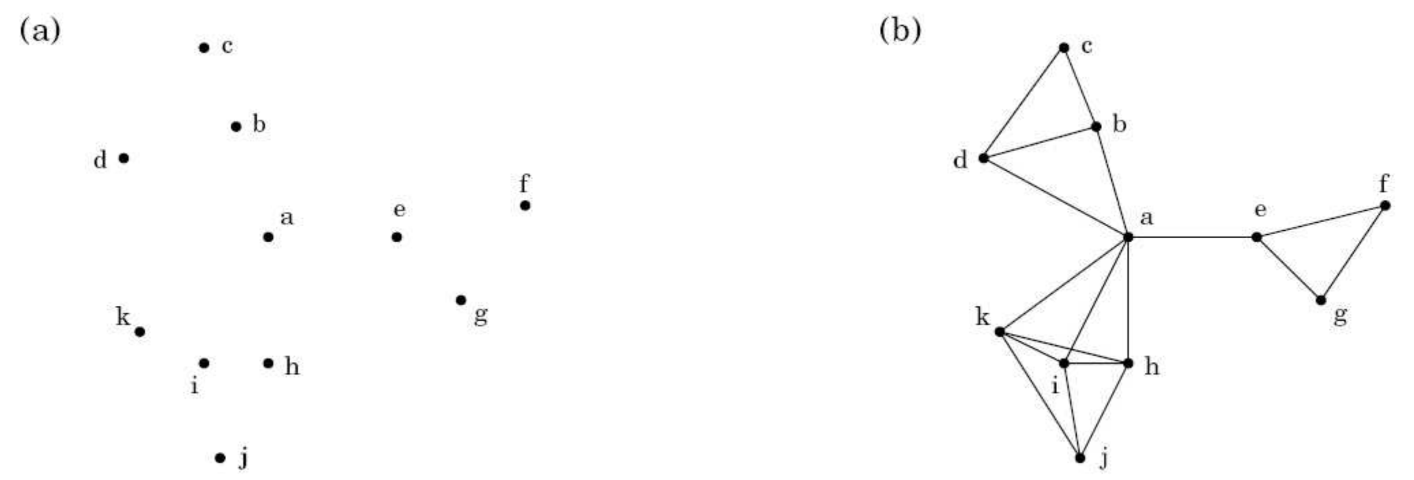
- SAT problem is NP hard
- Can we use Horn-SAT to solve P=NP problem?
  – Transforming a SAT problem to a Horn-SAT problem takes exponential time

# Set Cover



- Input: A set of elements $B$ and subsets $S_1, S_2, \ldots, S_m \subset B$
- Output: A selection of $S_i$ such that $\cup_i S_i = B$
- Cost: Number of sets, i.e., $|S_i|$
- Example:

**Figure 5.11** (a) Eleven towns. (b) Towns that are within 30 miles of each other.




- Greedy algorithm: