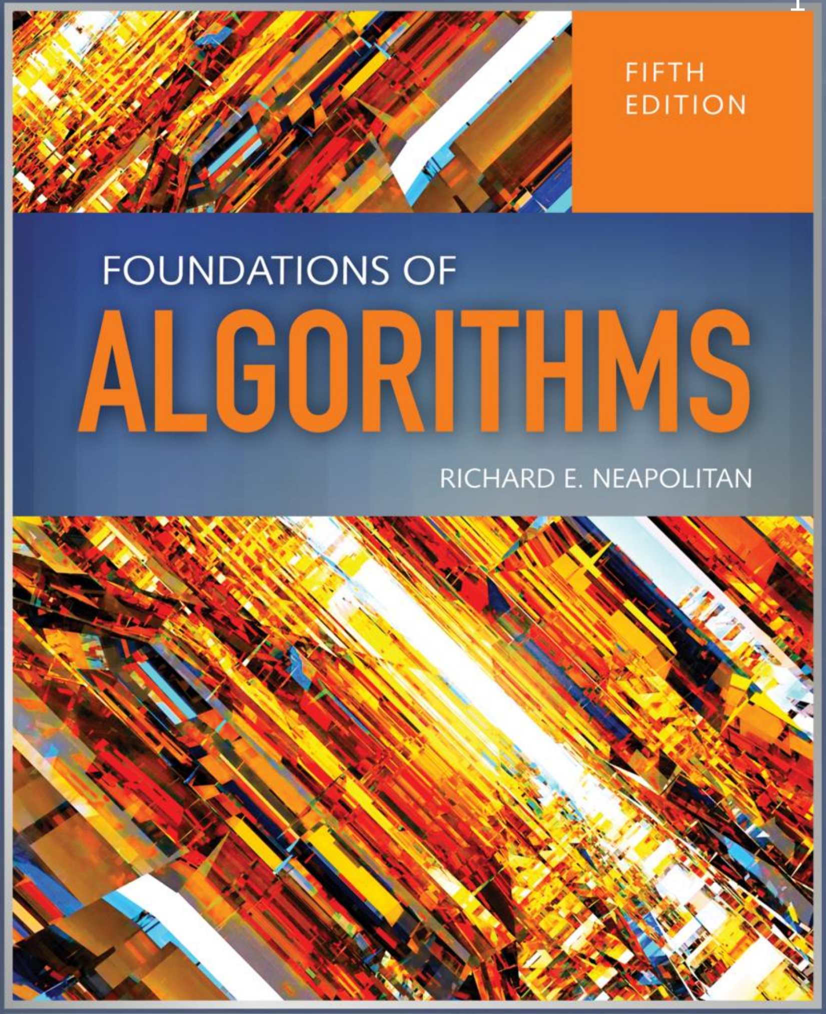


The Greedy Approach

Chapter 4

Objectives

- Describe the Greedy Programming Technique
- Contrast the Greedy and Dynamic Programming approaches to solving problems
- Identify when greedy programming should be used to solve a problem
- Prove/disprove greedy algorithm produces optimal solution
- Solve optimization problems using the greedy approach

Greedy Approach vs Dynamic Programming

- Solve optimization problems
- Greedy – problem is not divided into smaller sub-problems as in Dynamic Programming
- Obtain a solution by making a sequence of choices, each choice appears to be the best choice at that step
- Locally optimal
- Once a choice is made, it cannot be reconsidered
- Choice made without regard to past or future choices

Greedy Programming vs Dynamic Programming

- Goal is to produce a globally optimal solution
- Optimal – must be proven

Greedy Approach

- Initially the solution is an empty set
- At each iteration, items added to the solution set until the set represents a solution to that instance of the problem

Greedy Algorithm

- ***Selection procedure***: Choose the next item to add to the solution set according to the greedy criterion satisfying the locally optimal consideration
- ***Feasibility Check***: Determine if the new set is feasible by determining if it is possible to complete this set to provide a solution to the problem instance
- ***Solution Check:*** Determine whether the new set produced is a solution to the problem instance.

Make Change Problem

- Problem: minimize total number of coins returned as change by a sales clerk to a customer
- Assumption: unlimited supply of coins
- Solution Set: The amount of change in the customer's hand

Make Change Algorithm

```
while (there are more coins and the instance is not solved)
    {
        grab the largest remaining coin;
        if (adding the coin makes the change exceed amount owed)
        {
            reject coin;
        }
        else
        {
            add the coin to the change;
        }
        if (total value of the change equals the amount owed)
        {
            the instance is solved;
        }
    }
```

Optimal Solution? Prove

- Set of coins finite – {H,Q,D,N,P}
- Brute force, show greedy algorithm produces an optimal solution to be made for $.01 - $.50
- Any amount of change > $.50 would be a multiple of what was shown (use induction)
- Include a 12-cent coin: coins .50, .25, .12, .10, .05, .01
  - Produce $.16 in change: not optimal

Spanning Tree

- Connected, weighted, undirected graph G
- Spanning tree is a sub-graph of G containing all of the vertices in G and is a tree

Minimum Spanning Tree

- Connected, weighted, undirected graph G
- Remove edges from G such that G remains connected such that the sum of the weights on the remaining edges is minimal

Minimum Spanning Tree for G

- Let G = (V , E)
- Let T be a spanning tree for G: T = (V, F) where F $\subseteq$ E
- Find T such that the sum of the weights of the edges in F is minimal

Greedy Algorithms for finding a Minimum Spanning Tree

- Prim’s Algorithm
- Kruskal’s Algorithm
- Each uses a different locally optimal property
- Must prove each algorithm

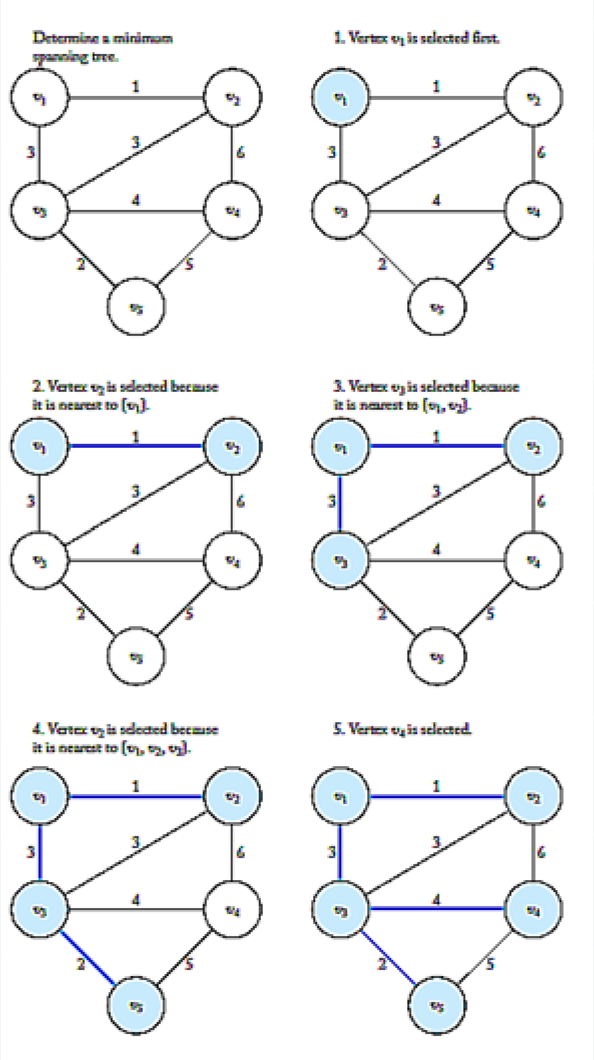
Prim's Algorithm

- Empty subset of edges F
- Subset of vertices Y initialized to an arbitrary vertex
- $Y = \{v_1\}$
- Select a vertex nearest to Y from V-Y connected to a vertex in Y by a minimum weight edge
  - Add the selected vertex to Y
  - Add the edge connecting the selected vertex to F
- Ties broken arbitrarily
- Repeat the process until Y = V

```
F = ∅
Y = {v 1}
while (instance not solved)
{
        select vertex in V – Y
nearest to Y; //selection
procedure and feasibility

                    //check
        add the vertex to Y;
        add the edge to F:
        if (Y == V)
                the instance is
solved;         //solution check
}
```

Figure 4.4


Determine a minimum spanning tree.
1. Vertex v_1 is selected first.
2. Vertex v_2 is selected because it is nearest to {v_1}.
3. Vertex v_3 is selected because it is nearest to {v_1, v_2}.
4. Vertex v_5 is selected because it is nearest to {v_1, v_2, v_3}.
5. Vertex v_4 is selected.

Every-Case Time Complexity of Prim's Algorithm 4.1

- Input Size: n (the number of vertices)
- Basic Operation : Two loops with n – 1 iterations inside repeat loop
- Repeat loop has n-1 iterations
- Time complexity:
  - $T(n) = 2(n - 1)(n - 1) \in \theta(n^2)$

Spanning Tree Produced by Prim's Algorithm Minimal?

- Dynamic Programming Algorithm – show principle of optimality applies
- Greedy Algorithm – easier to develop – must formally prove optimal solution always produced
- Two parts to proof:
  - Lemma 4.1
  - Theorem 4.1

Promising

- G = (V, E)
- F $\subseteq$ E
- F is *promising* if edges can be added to it to form a minimum spanning tree

Lemma 4.1
Let

- $G = (V, E)$ connected weighted, undirected graph
- $F \subseteq E$ be promising
- $Y \subseteq V$ be the set of vertices connected by edges in F
- e be a minimum edge connecting some $v_y \in Y$ to $v_x \in V-Y$, $F \cup \{e\}$ is promising

Proof Lemma 4.1

- F is promising – must be a minimum spanning tree (v, F') such that $F \subseteq F'$
- if $e \in F'$, $F \cup \{e\} \subseteq F'$ => $F \cup \{e\}$ is promising (proof complete)
- If $e \notin F'$, $F' \cup \{e\}$ must contain a cycle containing e => there must be another $e' \in F'$ in the cycle connecting some $v_x \in Y$ to $v_y \in V - Y$
- Cycle disappears if $F' \cup \{e\} - \{e'\}$ => spanning
- Since e is minimum, e <=e' => $F' \cup \{e\} - \{e'\}$ must be a spanning tree

Proof Lemma 4.1

- $F \cup \{e\} \subseteq F' \cup \{e'\} - \{e'\}$
- Since F connects only vertices in Y, $e' \notin F$
- i.e. e was selected
  - adding e does not create a cycle => $F \cup \{e\}$ is promising

Theorem 4.1 – Proof by Induction

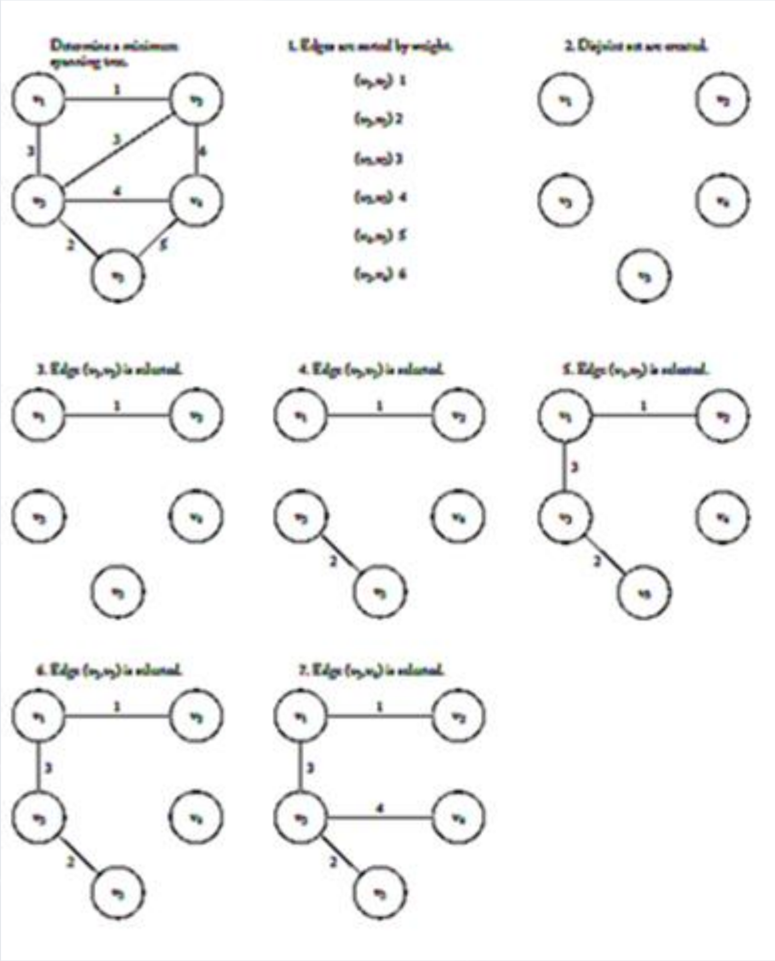
- Prim's Algorithm always produces a minimum spanning tree
- Induction Base: F = empty set is promising
- Induction Hypothesis: Assume after i iterations of the repeat loop, F is promising
- Induction Step: Show $F \cup \{e\}$ is promising where e is the edge selected in the next iteration
- Since e was selected, e is a minimum weight edge connecting a vertex in Y to a vertex in V – Y
- By Lemma 4.1, $F \cup \{e\}$ is promising

Kruskal’s Minimum Spanning Tree Algorithm

- Create n disjoint subsets of V – one for every v ε V
- Each subset contains only one vertex
- Inspect edges according to non-decreasing weight. If an edge connects two vertices in disjoint subsets, add edge to final edge set and merge the two subsets
- Repeat until all subsets are merged into a single set

```
F = ∅;
Create disjoint subsets of V;
Sort the edges in E in non-decreasing
order;
while (the instance is not solved)
{
        select next edge; //selection
procedure
                //feasibility check
        if (edge connects 2 vertices in
disjoint subsets)
        {
                merge the subsets;
                add the edge to F;
        }
        if (all the subsets are merged)
//solution check
                instance is solved;
}
```

Figure 4.7



Algorithm 4.2 Kruskal

- Appendix C – disjoint set ADT
- initial(n) ε θ(n)
- p = find(i) sets p to point at the set containing index i
  - find ε θ(lg m) where m is the depth of the tree representing the disjoint data sets
- merge(p,q) merges 2 sets into 1 set
  - merge ε θ(c) where c is a constant
- equal(p,q) where p and q point to sets returns true p and q point to the same set
  - equal ε θ(c) where c is a constant

Worst-case Time Complexity Kruskal

- Basic operation: a comparison instruction
- Input size: n, number of vertices, and m, number of edges

3 considerations of Kruskal

1. Time to sort edges: $W(m) \in \theta(m \lg m)$
2. Time to initialize n disjoint data sets: $T(n) \in \theta(n)$
3. while loop: manipulation of disjoint data sets
   - Worst case, every edge is considered
   - $W(m) \in \theta(m \lg m)$

- To connect n nodes requires at least n-1 edges: $m >= n-1$
- G fully connected $m = n(n-1)/2 \in \theta(n^2)$
- $W(m,n) \in \theta(n^2 \lg n^2) = \theta(n^2 2\lg n) = \theta(n^2 \lg n)$

Spanning Tree Produced by Kruskal's Algorithm Minimal?

- Lemma 4.2
- Theorem 4.2

Lemma 4.2
Let

- G = (V, E) be a connected, weighted, undirected graph
- F is a promising subset of E
- Let e be an edge of minimum weight in E – F
- $F \cup \{e\}$ has no cycles
- $F \cup \{e\}$ is promising
- Proof of Lemma 4.2 is similar to proof of Lemma 4.1

Theorem 4.2

- Kruskal's Algorithm always produces a minimum spanning tree
- Proof: use induction to show the set F is promising after each iteration of the repeat loop
- Induction base: F = $\emptyset$ empty set is promising
- Induction hypothesis: assume after the ith iteration of the repeat loop, the set of edges F selected so far is promising
- Induction step: Show F $\cup$ {e} is promising where is the selected edge in the i+1 th iteration

Theorem Proof Continued

- e selected in next iteration, it has a minimum weight
- e connects vertices in disjoint sets
- Because e is selected, it is minimum and connects two vertices in disjoint sets
- By Lemma 4.2 $F \cup \{e\}$ is promising

Prim vs Kruskal

- Sparse graph
  - m close to n – 1
  - Kruskal $\theta(n \lg n)$ faster than Prim
- Highly connected graph
  - Kruskal $\theta(n^2 \lg n)$
  - Prim's faster

Dijkstra’s Single-Source Shortest Path

- $\theta(n^2)$
- Similar to Prim’s Minimum Spanning Tree Algorithm
- Only works for non-negative weight edges
- Complexity Analysis and Proof similar to Prim’s Algorithm
- Floyd’s all-pairs shortest Paths $\theta(n^3)$

Greedy vs Dynamic

- Both solve optimization problems
- Shortest Path
  - Floyd – all pairs dynamic
  - Dijkstra – single source greedy
- Greedy algorithms usually simpler
- Greedy algorithms do not always produce optimal solution – must formally prove
- Dynamic Programming – show principle of optimality applies

0-1 Knapsack Problem

- Thief breaks into jewelry store carrying a knapsack in which to place stolen items
- Knapsack has a weight capacity W
- Knapsack will break if weight of stolen items exceeds W
- Each item has a value
- Thief's dilemma is to maximize the total value of items stolen while not exceeding the total weight capacity W

Brute Force Solution

- Consider all subsets of the n items
- Discard subsets whose total weight exceeds W
- Of the remaining, take the one with maximum profit
- 2^n subsets of a set containing n items

Greedy Strategy

- Steal items with the largest profit first – stealing in non-increasing order according to profit
- Can easily be shown by example greedy strategy does not always produce an optimal solution