Chapter 9: Greedy Algorithms

From the first day to this, sheer greed was the driving spirit of civilization. (Friedrich Engels)

Introduction

Greedy algorithms is a technique for solving problems with the following properties:

- The problem is an optimization problem, to find the solution that minimizes or maximizes some value (cost/profit).
- The solution can be constructed in a sequence of steps/choices.
- For each choice point:
  - The choice must be feasible.
  - The choice looks as good or better than alternatives.
  - The choice cannot be revoked.

Example of Making Change

- Suppose we want to give change of a certain amount (say 24 cents).
- We would like to use fewer coins rather than more.
- We can make a solution by repeatedly choosing a coin $\leq$ to the current amount, resulting in a new amount.
- The greedy solution is to always choose the largest coin value possible (for 24 cents: 10, 10, 1, 1, 1, 1).
- If there were an 8-cent coin, the greedy algorithm would not be optimal (but not bad).

Prim's Algorithm

Minimum Spanning Trees

- A *spanning tree* of a connected graph is a tree containing all the vertices.
- A *minimum spanning tree* of a weighted graph is a spanning tree with the smallest weight.
- The *weight* of a spanning tree is the sum of the edge weights.

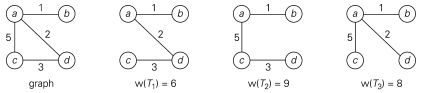
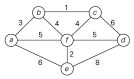


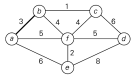
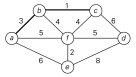
FIGURE 9.1 Graph and its spanning trees; T_1 is the minimum spanning tree

Prim's Algorithm

algorithm $Prim(G)$
  // Returns the MST by Prim's Algorithm
  // Input: A weighted connected graph $G = (V, E)$
  // Output: Set of edges comprising a MST
  $V_T \leftarrow$ {any vertex in G}
  $E_T \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    $e \leftarrow$ minimum-weight edge (v, u)
      with $v \in V_T$ and $u \in V - V_T$
    $V_T \leftarrow V_T \cup \{u\}$
    $E_T \leftarrow E_T \cup \{e\}$
  return E_T

Example of Prim's Algorithm



Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	

Example of Prim's Algorithm, Continued

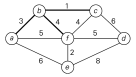
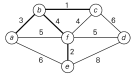
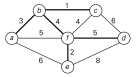
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

FIGURE 9.2 Application of Prim's algorithm. The parenthesized labels of a vertex in the middle column indicate the nearest tree vertex and edge weight; selected vertices and edges are shown in bold.

Comments on Prim's Algorithm

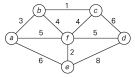
- This algorithm repeatedly chooses the smallest-weight edge from the tree so far to the other vertices.
- If a spanning tree has a weightier edge between V_T and $V - V_T$, it can be improved by replacing it with e.
- We can put V_T edges into a priority queue, then dequeue edges until one goes between V_T and $V - V_T$.
- Prim's Algorithm using a binary heap to implement a priority queue is $O(E \log E) = O(E \log V)$.

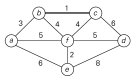
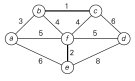
Kruskal's Algorithm

Kruskal's Algorithm

algorithm $Kruskal(G)$
  // Returns the MST by Kruskal's Algorithm
  // Input: A weighted connected graph $G = (V, E)$
  // Output: Set of edges comprising a MST
  sort the edges E by their weights
  $E_T \leftarrow \emptyset$
  while $|E_T| + 1 < |V|$ **do**
    $e \leftarrow$ next edge in E
    if $E_t \cup \{e\}$ does not have a cycle **then**
      $E_T \leftarrow E_T \cup \{e\}$
  return E_T

Example of Kruskal's Algorithm



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

Example of Kruskal's Algorithm, Continued

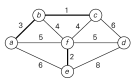
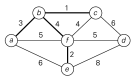
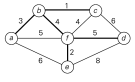
ef 2	bc ef **ab** bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ab 3	bc ef ab **bf** cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bf 4	bc ef ab bf cf af **df** ae cd de 1 2 3 4 4 5 5 6 6 8	
df 5		

FIGURE 9.4 Application of Kruskal's algorithm. Selected edges are shown in bold.

Comments on Kruskal's Algorithm

- This algorithm repeatedly chooses the smallest-weight edge that does not form a cycle.
- Kruskal's Algorithm is $O(E \log V)$ using efficient cycle detection.
- Disjoint Set Implementation for Detecting Cycles:
  Put each vertex into a singleton set.
  e is chosen if its vertices are in different sets.
  When e is chosen, the two sets are unioned.

Dijkstra's Algorithm

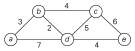
Single-Source Shortest Paths

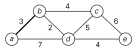
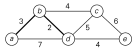
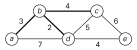
- The *single-source shortest paths problem* finds the shortest paths from a given vertex (the *source* vertex) to all the other vertices.
- Dijkstra's algorithm can be used if there are no negative edges.
- The Bellman-Ford algorithm is used if there are negative edges.
- Floyd's algorithm is used for all-pairs shortest paths if there are no negative weights.

Dijkstra's Algorithm

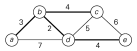
algorithm $Dijkstra(G, s)$
  // Solves SSSP by Dijkstra's Algorithm
  // Input: A weighted connected graph $G = (V, E)$
  // with no negative weights, and source vertex v
  // Output: The length and path from s to every v
  for $v \in V$ **do**
    $d_v \leftarrow \infty$; $p_v \leftarrow$ **null**; unmark v
  $d_s \leftarrow 0$
  while
    $u \leftarrow$ unmarked vertex with smallest d_u; mark u
    for each v adjacent from u **do**
      if v is unmarked **and** $d_u + w(u, v) < d_v$ **then**
        $d_v \leftarrow d_u + w(u, v)$; $p_v \leftarrow u$

Example of Dijkstra's Algorithm



Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	

Example of Dijkstra's Algorithm, Continued

c(b, 7)	**e(d, 9)**	
e(d, 9)		

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are

from a to b:	$a - b$	of length 3
from a to d:	$a - b - d$	of length 5
from a to c:	$a - b - c$	of length 7
from a to e:	$a - b - d - e$	of length 9

FIGURE 9.10 Application of Dijkstra's algorithm. The next closest vertex is shown in bold.

Comments on Dijkstra's Algorithm

- This algorithm repeatedly processes the closest unmarked vertex u.
- When u is marked, the SP to u is known.
  Basis: First value of u is s, whose SP is known.
  Induction: Assume SPs to previously marked vertices are known.
  SPs to unmarked vertices must start with some marked vertices.
  u is closest unmarked vertex.
- Dijkstra's Algorithm using a binary heap to implement a priority queue is $O(E \log V)$.

Huffman Trees

Encodings

- We can represent an n-character set by assigning a *codeword*, a bit string, to each character.
- *Fixed-length encoding* (e.g., ASCII) assigns m-bit strings to each character, $m \geq \log_2 n$.
- *Variable-length encoding* (e.g., Morse code) assigns different lengths.
- To avoid ambiguity, use *prefix codes*, i.e., no codeword is a prefix of another codeword.

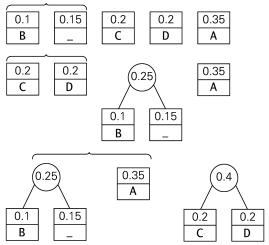
Optimal Prefix Codes

- Problem: If the probabilities of the characters are known, what is the best binary prefix code?
- "Best" means the expected length of a codeword, sum of probabilities times code length.
- Any binary tree with edges labeled with 0's and 1's and characters assigned to its leaves yields a prefix code.
- The two smallest probabilities must have the same length.
- If made siblings, the size of the problem decreases by one.

Huffman's Algorithm

algorithm $Huffman(W[0..n-1])$
// Computes optimal prefix code.
// Input: Array W of character probabilities
// Output: The Huffman tree.
for $i \leftarrow 0$ **to** $n-1$ **do**
$T \leftarrow$ node labeled with character i
$T.weight \leftarrow W[i]$; add T to set S
for $i \leftarrow 0$ **to** $n-2$ **do**
$L, R \leftarrow$ remove two min. weight trees from S
$T \leftarrow$ node with children L and R
$T.weight \leftarrow L.weight + R.weight$
add T to set S
return T

Example of Huffman's Algorithm



Example of Huffman's Algorithm, Continued

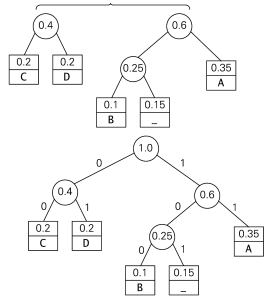


FIGURE 9.11 Example of constructing a Huffman coding tree