Greedy Algorithms

SMITA SANKHE
Assistant Professor
Department of Computer Engineering

Problem Characteristics

- Decomposable?
- Solution steps be undone?
- Unique solution or set of solutions?
- Any solution or best solution?
- Solution –a path or a state

Greedy Algorithms

- typically apply to optimization problems in which we make a set of choices in order to arrive at an optimal solution.
- make each choice in a locally optimal manner.
- Does not give optimal solutions always; but do most of the times
- Quite powerful and works well for a wide range of problems.

The greedy method

- Suppose that a problem can be solved by a sequence of decisions. The greedy method has that each decision is locally optimal. These locally optimal solutions will finally add up to a globally optimal solution.
- Only a few optimization problems can be solved by the greedy method.

The general method

```
1   Algorithm Greedy(a, n)
2   // a[1 : n] contains the n inputs.
3   {
4       solution := ∅; // Initialize the solution.
5       for i := 1 to n do
6       {
7           x := Select(a);
8           if Feasible(solution, x) then
9               solution := Union(solution, x);
10      }
11      return solution;
12  }
```

Greedy algorithms: The working

- Make a choice at each step.
- Make the choice before solving the subproblems.
- Solve top-down.

Problems typically solved with Greedy Strategy

- Knapsack problem
- Minimum cost spanning trees-Kruskal and prim's algorithm
- Single source shortest path
- Job sequencing with deadlines

Knapsack problem

- Given positive integers P_1, P_2, …, P_n, W_1, W_2, …, W_n and M.
- Find X_1, X_2, … ,X_n, $0 \leqq X_i \leqq 1$ such that

$$\sum_{i=1}^{n} P_i X_i$$ is maximized.

- Subject to $$\sum_{i=1}^{n} W_i X_i \leq M$$

Knapsack Problem Example

- M = 20, $(P_1, P_2, P_3)=(25,24,15)$
  $(W_1, W_2, W_3) = (18, 15, 10)$
- Three feasible solutions, 1 is optimal

	(X_1, X_2, X_3)	$\Sigma W_i X_i$	$\Sigma P_i X$
1.	(1,2/15,0)	20	28.2
2.	(0, 2/3, 1)	20	31
3.	(0, 1, 1/2)	20	31.5

Applications of - Knapsack Problem

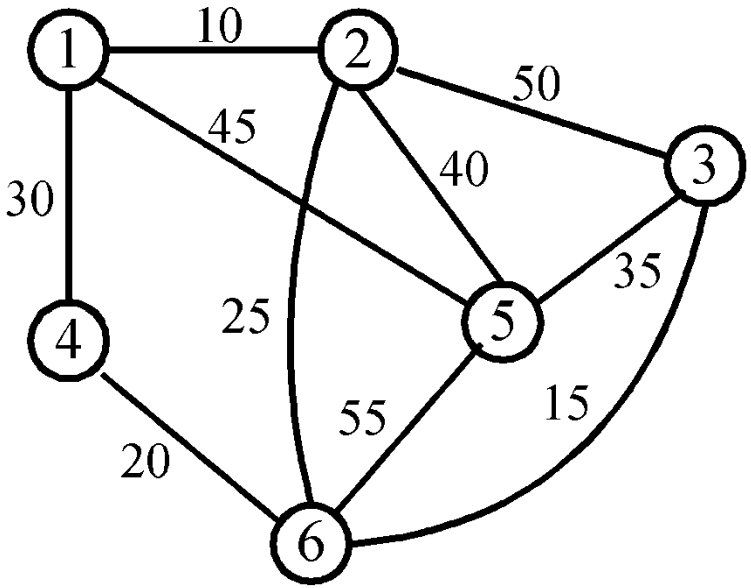
- finding the least wasteful way to cut raw materials,
- seating contest of investments and portfolios,
- seating contest of assets for asset-backed securitization,
- generating keys for the Merkle–Hellman and other knapsack cryptosystems
- construction and scoring of tests in which the test-takers have a choice as to which questions they answer
- Resource optimization

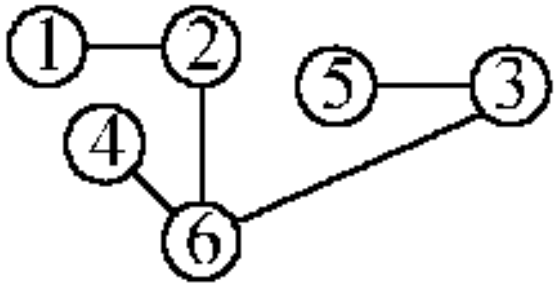
Minimum spanning trees (MST)

- G = (V, E): weighted connected undirected graph
- Spanning tree : S = (V, T), T $\subseteq$ E, undirected tree
- Minimum spanning tree(MST) : a spanning tree with the smallest total weight.

An example of MST

- A graph and one of its minimum costs spanning tree





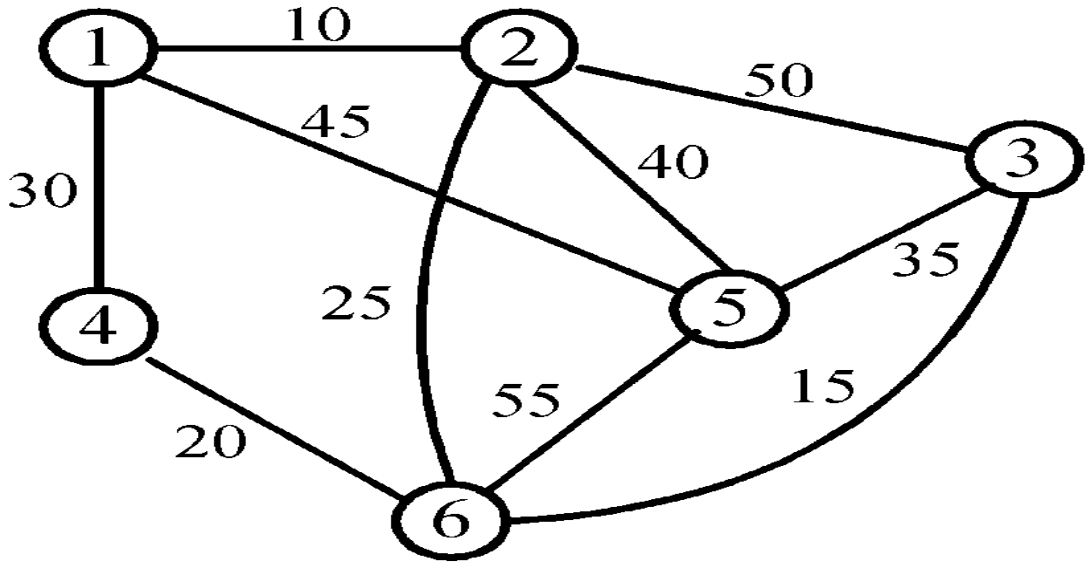
Kruskal's algorithm for finding MST

Step 1: Sort all edges into nondecreasing order.

Step 2: Add the next smallest weight edge to the forest if it will not cause a cycle.

Step 3: Stop if n-1 edges. Otherwise, go to Step2.

An example of Kruskal's algorithm



The details for constructing MST

- How do we check if a cycle is formed when a new edge is added?
  - By the SET and UNION method.
- A tree in the forest is used to represent a SET.
- If (u, v) $\in$ E and u, v are in the same set, then the addition of (u, v) will form a cycle.
- If (u, v) $\in$ E and u$\in S_1$, v$\in S_2$, then perform UNION of S_1 and S_2 .

Prim's algorithm for finding MST

Step 1: $x \in V$, Let $A = \{x\}$, $B = V - \{x\}$.

Step 2: Select $(u, v) \in E$, $u \in A$, $v \in B$ such that (u, v) has the smallest weight between A and B.

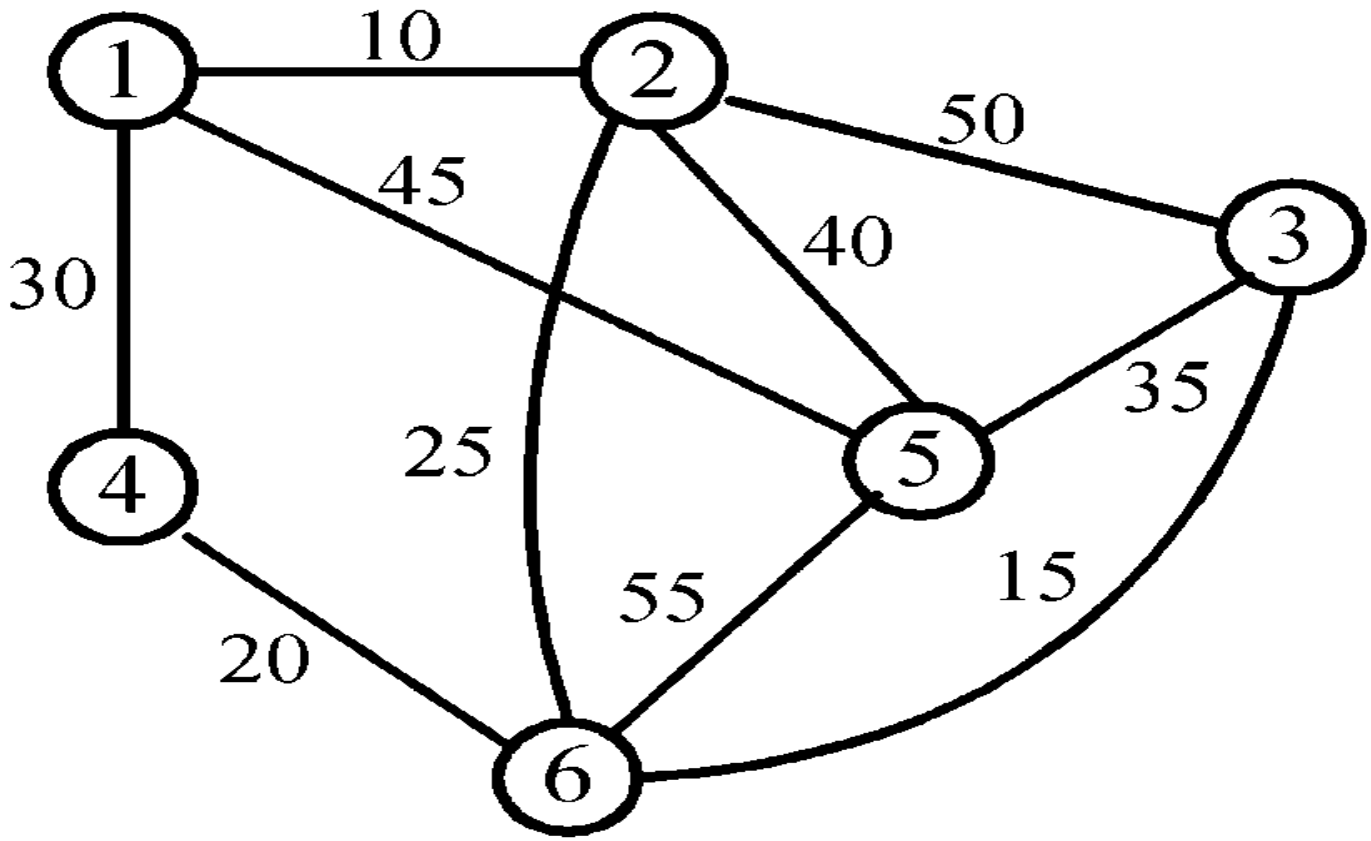
Step 3: Put (u, v) in the tree. $A = A \cup \{v\}$, $B = B - \{v\}$

Step 4: If $B = \varnothing$, stop; otherwise, go to Step 2.

- Time complexity : $O(n^2)$, $n = |V|$.

(see the example on the next page)

An example for Prim's algorithm



Applications of - MST

- **Network design.**
  – *telephone, electrical, hydraulic, TV cable, computer, road*
- **Approximation algorithms for NP-hard problems.**
  – *traveling salesperson problem*, *Steiner tree*
- **Indirect applications.**
  – max bottleneck paths
  – codes for error correction
  – learning salient features for real-time face verification
  – reducing data storage in sequencing amino acids in a protein
  – model locality of particle interactions in turbulent fluid flows
  – autoconfig protocol for Ethernet bridging to avoid cycles in a network
- Circuit design: implementing efficient multiple constant multiplications, as used in finite impulse response filters.
- Regionalisation of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.

Single Source Shortest Path

- Problem Definition

Given a directed graph G=<V,E>, a weighting function cost for edges of G and a source vertex v_0 , the problem is to compute the shortest paths from v_0 to all the remaining vertices of G.

Assumption:- all weights are positive.

Shortest path between v_0 to some other node v is an ordering among a subset of edges.

Algorithm

```
1   Algorithm ShortestPaths(v, cost, dist, n)
2   // dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
3   // path from vertex v to vertex j in a digraph G with n
4   // vertices. dist[v] is set to zero. G is represented by its
5   // cost adjacency matrix cost[1 : n, 1 : n].
6   {
7       for i := 1 to n do
8       { // Initialize S.
9           S[i] := false; dist[i] := cost[v, i];
10      }
11      S[v] := true; dist[v] := 0.0; // Put v in S.
12      for num := 2 to n − 1 do
13      {
14          // Determine n − 1 paths from v.
15          Choose u from among those vertices not
16          in S such that dist[u] is minimum;
17          S[u] := true; // Put u in S.
18          for (each w adjacent to u with S[w] = false) do
19              // Update distances.
20              if (dist[w] > dist[u] + cost[u, w])) then
21                  dist[w] := dist[u] + cost[u, w];
22      }
23  }
```

Single Source Shortest Path Conditions

- Feasibility condition:- All possible paths with source vertex i.e a. n-1 possible paths and

  b. possible iterations n-2 excluding source and destination vertex

- Optimal condition :- path that includes all vertices with minimum cost

Complexity

- Overall operations-

= N vertices * (n-2) iterations

= n(n-2)

= $n^2 – 2n$

= $O(n^2)$

Applications of - SSSP

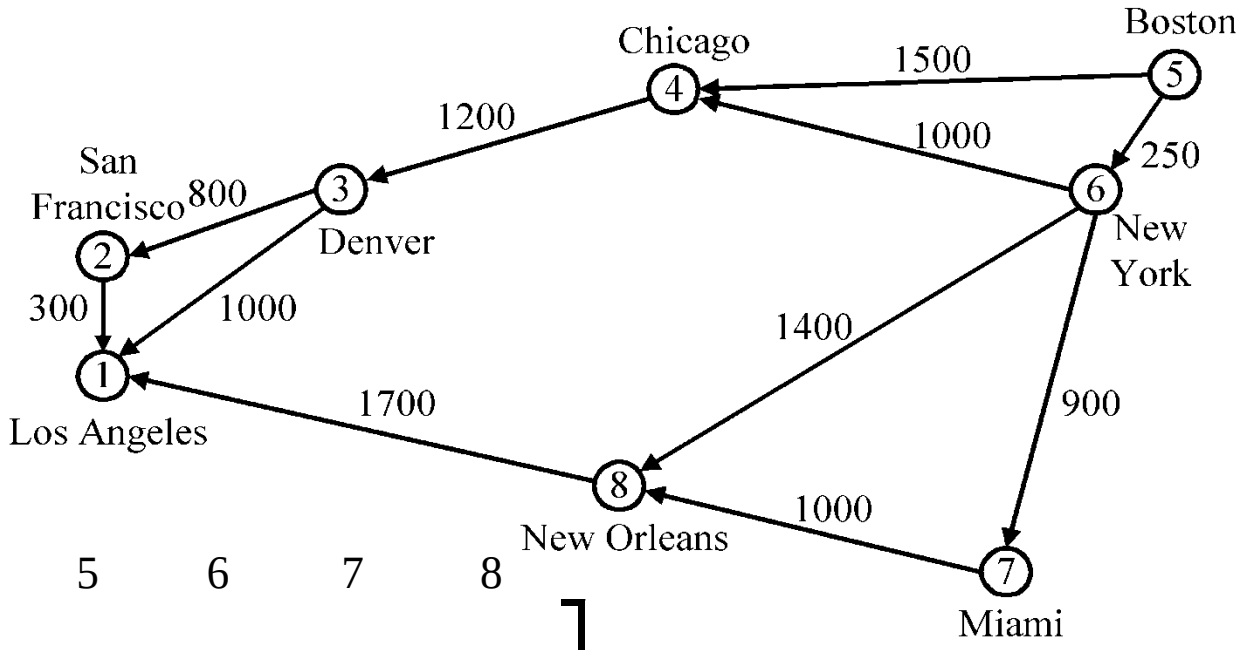
- automatically find directions between physical locations, such as driving directions on web mapping websites
- When the problem is represented as a FSA, can be used to find an optimal sequence of choices to reach a certain goal state, or to establish lower bounds on the time needed to reach a given state.
- In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and usually tied with a widest path problem
- plant and facility layout, robotics, transportation, and VLSI design

Applications of - SSSP

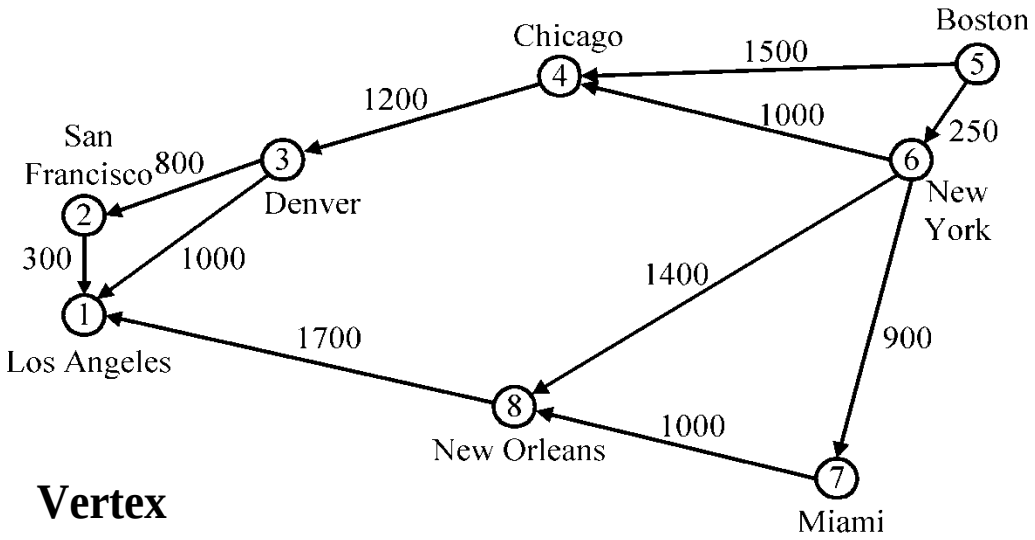
- Maps
- Robot navigation.
- Texture mapping.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Subroutine in advanced algorithms.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern

Dijkstra's algorithm

Cost adjacency matrix. All entries not shown are $+\infty$.



	1	2	3	4	5	6	7	8
1	0							
2	300	0						
3	1000	800	0					
4			1200	0				
5				1500	0	250		
6				1000		0	900	1400
7							0	1000
8	1700							0

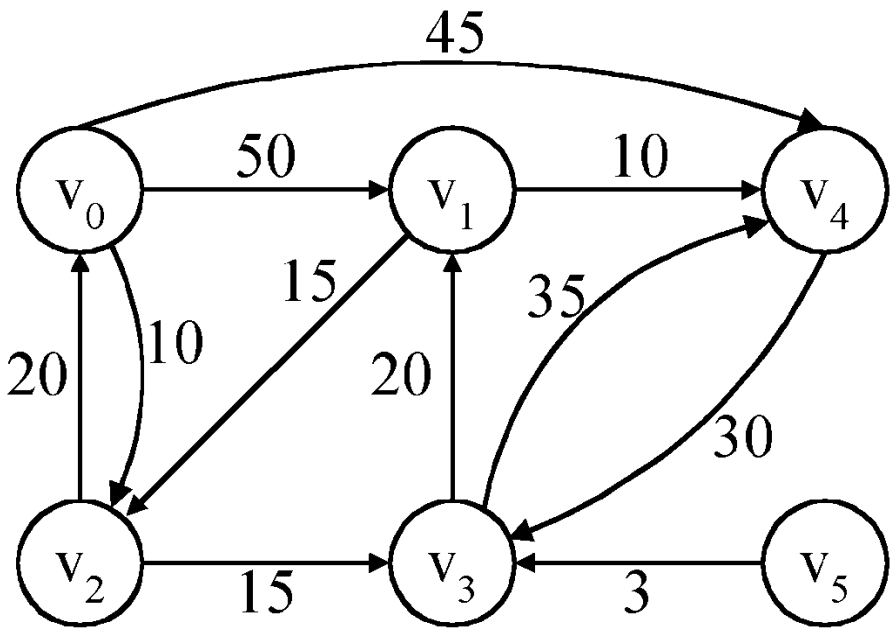


Iteration	S	Vertex Selected	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Initial		----								
1	5	6	+∞	+∞	+∞	1500	0	250	+∞	+∞
2	5,6	7	+∞	+∞	+∞	1250	0	250	1150	1650
3	5,6,7	4	+∞	+∞	+∞	1250	0	250	1150	1650
4	5,6,7,4	8	+∞	+∞	2450	1250	0	250	1150	1650
5	5,6,7,4,8	3	3350	+∞	2450	1250	0	250	1150	1650
6	5,6,7,4,8,3	2	3350	3250	2450	1250	0	250	1150	1650
	5,6,7,4,8,3,2		3350	3250	2450	1250	0	250	1150	1650

- Time complexity : $O(n^2)$

The single-source shortest path problem

- shortest paths from v_0 to all destinations



(a)

	Path	Length
1)	v_0v_2	10
2)	$v_0v_2v_3$	25
3)	$v_0v_2v_3v_1$	45
4)	v_0v_4	45

(b)

SSSP Solution

- Problem definition
- Feasibility & Optimality Condition
- OR
- Explain how the answer will be optimal
- Compute
- Final answer
  - Cost of every destination
  - Path to every destinatioin
- complexity