Design and Analysis of Algorithms
Greedy Technique

Fractional Knapsack Problem;
Minimum Spanning Tree -
Prim's and Kruskal's Algorithms;
Single Source Shortest Path - Dijkstra's Algorithm

Dr. Soharab Hossain Shaikh
BML Munjal University

1

Greedy Technique

Computer scientists consider ***greedy*** approach a general design technique despite the fact that it is applicable to optimization problems only. The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached. On each step—and this is the central point of this technique—the choice made must be:

- *feasible*, i.e., it has to satisfy the problem's constraints
- *locally optimal*, i.e., it has to be the best local choice among all feasible choices available on that step
- *irrevocable*, i.e., once made, it cannot be changed on subsequent steps of the algorithm

2

Fractional Knapsack Problem

- This is a variation of the 0-1 knapsack problem known as the Fractional Knapsack problem.
- **Problem Definition:** Given n items of sizes $s_1, s_2, \ldots, s_n$, values $v_1, v_2, \ldots, v_n$ and size C, representing the knapsack capacity, find nonnegative real numbers $x_1, x_2, \ldots, x_n$ such that

$$\sum_{i=1}^{n} x_i v_i ,$$

is maximized subject to the constraint

$$\sum_{i=1}^{n} x_i s_i \leq C .$$

3

Fractional Knapsack Problem: Strategies

- Given a knapsack of size C = 20 and three items with the following sizes and values

i	1	2	3
s_i	18	15	10
v_i	25	24	15

- **Greedy Strategy 1** : Sort the items in non-increasing order of value Then, starting from the first item in the sorted list onwards, fill the knapsack with as much as it can hold.
  - $s_1 = 18 < 20$ implies $x_1 = 1$.
  - $s_2 = 15 > 2$ implies $x_2 = 2/15$ (As the remaining capacity of the knapsack is 2).
  - $x_3 = 0$ (As the knapsack is now full).

$$\text{Total Value} = \sum_{i=1}^{3} x_i v_i = (1)(25) + (2/15)(24) + (0)(15) = 28.2 .$$

4

Fractional Knapsack Problem: Strategies (cont…)

- **Question**: Given the knapsack of size C = 20 and the three items

i	1	2	3
s_i	18	15	10
v_i	25	24	15

is the solution $x_1 = 1,\ x_2 = 2/15,\ x_3 = 0$ using Strategy 1 an optimal one?

- **Answer**: If $x_2 = 1,\ x_1 = 5/18,\ x_3 = 0$

$$\text{Total Value} = \sum_{i=1}^{3} x_i v_i = (5/18)(25) + (1)(24) + (0)(15) = 30.944444$$

$\therefore$ The solution $x_1 = 1,\ x_2 = 2/15,\ x_3 = 0$ is not optimal

- **Greedy Strategy 2** : Sort the items in non-decreasing order of size and then starting from the first item in the sorted list onwards, fill the knapsack with as much as it can hold.

Fractional Knapsack Problem: Optimal Strategy

- **Greedy Strategy 3** : Sort the items in non-increasing order of value per unit size, v_i/s_i and then starting from the first item in the sorted list onwards, fill the knapsack with as much as it can hold.
- Given the knapsack of size C = 20 and the three items as before

i	2	3	1
s_i	15	10	18
v_i	24	15	25
$\frac{v_i}{s_i}$	1.60	1.50	1.39

- The solution $x_2 = 1,\ x_3 = 0.5,\ x_1 = 0$ gives

$$\text{Total Value} = \sum_{i=1}^{3} x_i v_i = (0)(25) + (1)(24) + (0.5)(15) = 31.5.$$

- The total value is greater than that generated by the previous two greedy strategies.

Spanning Tree

DEFINITION A ***spanning tree*** of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a ***minimum spanning tree*** is its spanning tree of the smallest weight, where the ***weight*** of a tree is defined as the sum of the weights on all its edges. The ***minimum spanning tree problem*** is the problem of finding a minimum spanning tree for a given weighted connected graph.

7

Minimum Spanning Tree

- **Definition**: Let $G = (V, E)$ be a connected undirected graph with weights on its edges. A subgraph $T = (V, E_T)$ of G, where $E_T \subseteq E$, is called a minimum spanning tree if T is a tree that spans G and the sum of the weights of the edges in T is minimum.
  - For all spanning trees T, $|E_T| = |V| - 1$.
- Two algorithms to find the minimum spanning tree:
  - Kruskal's Algorithm
  - Prim's Algorithm
- Both algorithms follow the "greedy" approach.

8

Application MST

- **Minimum Spanning Tree**: Since T is acyclc and connects all of the vertices, it must form a tree. We call this tree a spanning tree since it "spans" all the vertices of G. It is also called a minimum spanning tree as it minimizes the total cost.

- **Application Model**: The electronic circuit design problem can be modeled as a minimum spanning tree problem as follows:
  - Construct a connected undirected graph $G = (V, E)$, where V represents the set of pins and E represents each possible interconnection between pairs of pins.
  - $\forall(u, v) \in E$, $c(u, v)$ specifies the cost of connecting u and v.
  - Find a subgraph $T = (V, E_T)$ of G such that
    - T connects all the vertices of V
    - Total cost $C(T) = \sum_{(u,v)\in E_T} c(u, v)$ is minimized.

9

Prim's Algorithm

- **Main Idea**: Prim's algorithm is similar to Dijkstra's algorithm. It starts finding the minimum spanning tree from a given initial starting vertex and then grows the tree by adding edges such that the minimum cost spanning tree is constructed.

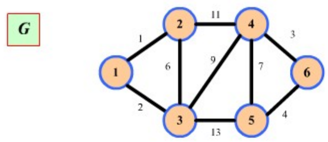
- Let $G = (V, E)$ be an undirected graph, where for simplicity $V = \{1, 2, \ldots, n\}$. The set of edges T of a minimum cost spanning tree is found using Prim's algorithm as follows:

Robert Prim rediscovered this algorithm in 1957.
The algorithm was first published by Czech mathematician **V. Jarnik** 27 years earlier.
Sometimes this algorithm is referred to as Prim-Jarnik algorithm.

10

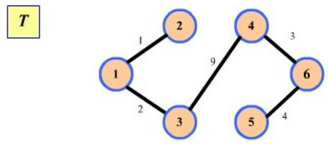
Prim’s Algorithm: Example

- **Example** Consider the graph G below:



X = {1, 2, 3, 4, 6, 5}

Y = {}



E_T

#	Edge	Weight
1	(1, 2)	1
2	(1, 3)	2
3	(3, 4)	9
4	(4, 6)	3
5	(5, 6)	4

11

Prim’s Algorithm (Pseudo code)
Ref Book: Anany Levitin

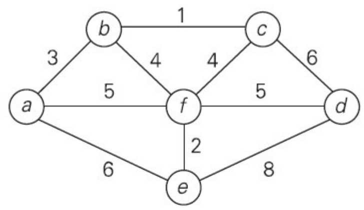
ALGORITHM *Prim*(G)

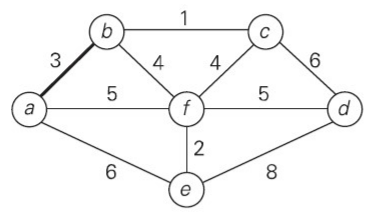
//Prim’s algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E\rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$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

Graph represented as weighted matrix. Priority queue implemented as unordered list.

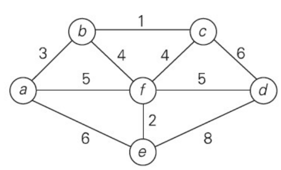
Time complexity: $O(n^2)$ for a graph containing n nodes.

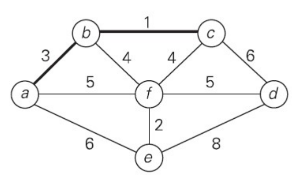
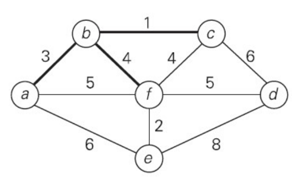
12



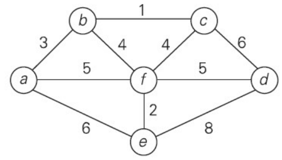
Tree vertices	Remaining vertices	Illustration
a(−, −)	**b**(**a**, **3**) c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	

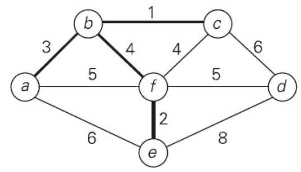
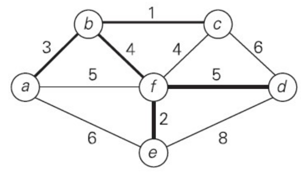
13



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

14



Tree vertices	Remaining vertices	Illustration
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

Prim-Jarnik's MST Algorithm

Algorithm PrimJarníkMST(G):
  Input: A weighted connected graph G with n vertices and m edges
  Output: A minimum spanning tree T for G
  Pick any vertex v of G
  $D[v] \leftarrow 0$
  for each vertex $u \neq v$ **do**
    $D[u] \leftarrow +\infty$
  Initialize $T \leftarrow \emptyset$.
  Initialize a priority queue Q with an item $((u, \text{null}), D[u])$ for each vertex u, where (u, null) is the element and $D[u]$ is the key.
  while Q is not empty **do**
    $(u, e) \leftarrow Q$.removeMin()
    Add vertex u and edge e to T.
    for each vertex z adjacent to u such that z is in Q **do**
      // perform the relaxation procedure on edge (u, z)
      if $w((u, z)) < D[z]$ **then**
        $D[z] \leftarrow w((u, z))$
        Change to $(z, (u, z))$ the element of vertex z in Q.
        Change to $D[z]$ the key of vertex z in Q.
  return the tree T

Graph represented as adjacency list.
Priority queue is implemented as min-heap.

Time complexity: O((n+m)logn) = O(mlogn) for a graph containing **n** vertices and **m** edges.

Check the Book: Algorithm Design by **Goodrich and Tamassia**, Pub: Wiley

Time Complexity of Prim-Jarnik's MST Algorithm

> **Graph Operations**

- We cycle through the incident edges once for each vertex

> **Label Operations**

- We set/get the distance, parent and locator labels of vertex *z* O(deg(*z*)) times
- Setting/getting a label takes O(1) time

> **Priority Queue Operations**

- Each vertex is inserted once into and removed once from the priority queue, where each insertion or removal takes O(log n) time
- The key of a vertex *w* in the priority queue is modified at most *deg(w)* times, where each key change takes O(log n) time

> **Prim-Jarnik's** algorithm runs in **O((n + m) log n)** time provided the graph is represented by the adjacency list structure

- Recall that Σ_v deg(v) = 2m
- The running time is **O(m log n)** since the graph is connected

17

Kruskal's Algorithm

➢ **Informal description of Kruskal's algorithm**

1. Initialize T to have the set of vertices V and an empty set of edges E_T.
2. Sort the set of edges E of G by weight in a nondecreasing order.
3. After that, the following is repeated until T is transformed into a tree:
   - Let $e \in E$ be the current edge under consideration. If adding e to T does not create a cycle, then do so; otherwise discard e.

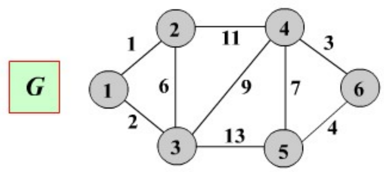
   - **Remark**: This process terminates after adding exactly $n - 1$ edges.

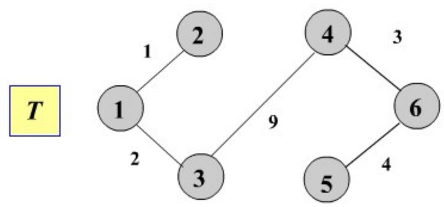
Proposed by **Joseph Kruskal**, 1956

18

Kruskal's Algorithm: Example

➢ **Example**: Consider the graph G below:





#	Edge	Weight
1	(1,2)	1 ✓
2	(1,3)	2 ✓
3	(4,6)	3 ✓
4	(5,6)	4 ✓
5	(2,3)	6 X
6	(4,5)	7 X
7	(3,4)	9 ✓
8	(2,4)	11 X
9	(3,5)	13 X

Data Structures for Kruskal's Algorithm

- The algorithm maintains a forest of trees
- A priority queue extracts the edges by increasing weight
- An edge is accepted it if connects distinct trees
- We need a data structure that maintains a partition, i.e., a collection of disjoint sets, with operations:
- **Makeset(u)**: create a set consisting of u. Every set has a unique set id.
- **Find(u):** return the set storing u (return the set id of the set containing u)
- **Union(A, B):** replace sets A and B with their union (after union A and B belongs to the same set identified by its unique set id)

Kruskal's Algorithm - MST

Algorithm KRUSKAL
Input A weighted connected undirected graph $G = (V, E)$ with n vertices and m edges.
Output The set of edges E_T of a minimum cost spanning tree for G.
Comment:Disjoint-sets data structure is used in the algorithm.

1. Sort the edges in E by non-decreasing weight.
2. **for** each vertex $v \in V$
3. MAKESET($\{v\}$)
4. **end for**
5. $E_T = \{ \}$
6. **while** $|E_T| < n - 1$
7. Let (x, y) be the next edge in E.
8. **if** FIND(x) $\neq$ FIND(y) **then**
9. Add (x, y) to E_T
10. UNION(x, y)
11. **end if**
12. **end while**

Graph containing n nodes and m edges.
Time complexity is dominated by the sorting step which may be done in O(m logm) time.

Total time needed for the union-find operations: O(n+m*log n).

Time complexity of Kruskal is :
O(mlogm)

21

Disjoint-Set Data Structures / Union-Find Algorithm

Check the book by Anany Levitin for a discussion on the Union-Find algorithms.

- **Disjoint set data structures is required to detect cycles in a graph.**
- **Find(x)** and **Find(y)** detects whether the two vertices X and Y belong to the same tree or not. If it is so, then adding the edge forms a cycle. So, that is not done.
- Basically **Find(X)** returns the root of the tree containing the element X. *(e.g. unique set id)*
- So, if **Find(X)!=Find(Y)** then they belong to two different trees. So, the edge (X,Y) can be added to the present Spanning tree without forming a cycle.
- Add(x,y) adds the edge to E_T
- **Union(X,Y)** adds the end vertices to the spanning tree. Combines the tree containing node X and the tree containing node Y into a single tree.
- Total cost of Find and Union operations is O(n+m*log n), for n-1 unions and m finds for a graph containing n vertices and m edges.

22

Single Source Shortest Path Problem

- Let $G=(V, E)$ be a directed weighted graph with V as the set of vertices and E as the set of edges with non-negative weights.
- The single source shortest path problem is to determine the distance from a given source vertex s to every other vertex in V, where the distance from vertex s to vertex $x \in V$ is defined as the length of the shortest path from s to x

23

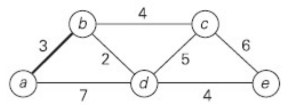
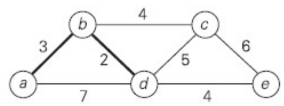
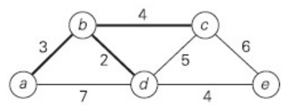
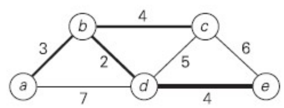
Dijkstra's Algorithm (Book: Anany Levitin)

ALGORITHM *Dijkstra(G, s)*

//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph $G = \langle V, E \rangle$ with nonnegative weights
// and its vertex s
//Output: The length d_v of a shortest path from s to v
// and its penultimate vertex p_v for every vertex v in V
Initialize(Q) //initialize priority queue to empty
for every vertex v in V
  $d_v \leftarrow \infty$; $p_v \leftarrow$ **null**
  Insert(Q, v, d_v) //initialize vertex priority in the priority queue
$d_s \leftarrow 0$; *Decrease*(Q, s, d_s) //update priority of s with d_s
$V_T \leftarrow \varnothing$
for $i \leftarrow 0$ **to** $|V| - 1$ **do**
  $u^* \leftarrow$ *DeleteMin*(Q) //delete the minimum priority element
  $V_T \leftarrow V_T \cup \{u^*\}$
  for every vertex u in $V - V_T$ that is adjacent to u^* **do**
    if $d_{u^*} + w(u^*, u) < d_u$
      $d_u \leftarrow d_{u^*} + w(u^*, u)$; $p_u \leftarrow u^*$
      Decrease(Q, u, d_u)

24

Dijkstra's Example (Book: Anany Levitin)

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

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

25

Dijkstra's Algorithm Time Complexity

Analysis like Prim's Algorithm

\# Graph represented as **weighted matrix.**
\# Priority queue implemented as **unordered list**.

\## Time complexity: O(n^2) for a graph containing n nodes.

\# Graph represented as **adjacency list**.
\# Priority queue is implemented as **min-heap**.

\## Time complexity: O(m log n) for a graph containing n nodes and m edges.

26

End of Lecture

27