Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

8.1 Greedy algorithms [Chapter 9]

change-making problem: Using coins of 25 cents, 10 cents, 5 cents, and 1 cent, how can you make change of n cents such that the total number of coins is minimized? What would be your answer if $n = 48$? If your answer is (25,10,10,1,1,1), you are using a greedy technique. The plan here is to give as many 25-cent coins as possible, then as many 10-cent coins as possible, then as many 5-cent coins as possible, and finally a number of 1-cent coins that makes the sum equal to n. Is this technique correct for this problem? Why? Is it always correct for different combinations of coin values (e.g., 7-cent, 5-cent, and 1-cent)? Why?

A greedy algorithm is a sequence of steps, where each step leads to a *partial solution* to the problem. Each step must be:

- feasible: i.e., it does not violate the problem's rules.
- locally optimal: i.e., it leads to the best choice at the current stage.
- irrevocable: i.e., once a decision is made, it is never changed (no backtracking).

In order for a greedy algorithm to be correct, the problem must satisfy the following property: local optimal choices lead to a globally optimal solution.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

8.2 Minimum spanning tree

A **spanning tree** of a connected graph $G(V, E)$ is a connected acyclic[2] subgraph[3] of G.
A **minimum spanning tree** of a weighted connected graph is a spanning tree with a minimum weight, where the weight of a graph is the sum of the weights of its edges.
How can we find a minimum spanning tree of a connected weighted graph?

Prim's algorithm

Prim's algorithm is a greedy algorithm that constructs a minimum spanning tree by starting with a subgraph of a single vertex and then at each step, one vertex is added to the subgraph until all vertices are added. At each step, the algorithm selects the vertex that can be added with the minimum additional weight (and, hence, it is called greedy!).

Correctness proof of Prim's algorithm:
Let G_i be the subgraph resulting from step i in the algorithm. To prove the correctness of the algorithm, we just need to show that every G_i, $i = 1, 2, \ldots, n$, is part of some minimum spanning tree. This would imply that $G_{|V|}$ is a minimum spanning tree. We can prove this for all i's by mathematical induction as follows.
The basis step: it is trivial that G_1 is part of some spanning tree because G_1 consists of a single vertex and no edges.

[1] This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.
[2] Acyclic graph is one with no cycles.
[3] A graph $G'(V', E')$ is a subgraph of a graph $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

```
Prim's algorithm(G(V, E))
Initialize a priority queue PQ;
foreach v ∈ V do
    key[v]:=∞;
    Neighbor[v]:=null;
end
key[v_0] := 0;
foreach v ∈ V do
    Enqueue(PQ,v);
end
while PQ ≠ φ do
    v:=Dequeue(PQ);
    foreach u such that (u, v) ∈ E do
        if u ∈ PQ and weight(u, v) < key[u] then
            Neighbor[u]:=v;
            key[u]:=weight(u, v);
            Update(PQ,u);
        end
    end
end
return Neighbor[0..|V| − 1];
```

The inductive step: we show that if G_{n-1} is part of a minimum spanning tree T, this implies that G_n is part of some minimum spanning tree. We prove this by contradiction. Let u be the vertex added in step n and (u, v) be the edge added in step n by Prim's algorithm. Assume that G_n is not part of any minimum spanning tree (i.e., (u, v) is not part of any minimum spanning tree). Now, if we add (u, v) to T, this results in a cycle. This cycle involves another edge $(u^*, v^*) \neq (u, v)$ connecting a vertex in G_{n-1} to a vertex not in G_{n-1}. Therefore, if wee add (u, v) and remove (u^*, v^*), this will result in another spanning tree T' whose weight is at most as that of T. This contradicts with the assumption that G_n is not part of any minimum spanning tree, and this proves that if G_{n-1} is part of a minimum spanning tree, then G_n is part of some minimum spanning tree.

Time complexity of Prim's algorithm

Let's analyze the time complexity of Prim's algorithm when the graph is represented using adjacency lists. When the priority queue is implemented using **min-heap**, Enqueue, Dequeue, and Update operations of the priority queue take $O(\log |V|)$ time. Prim's algorithm has $|V|$ Enqueue operations, $|V|$ Dequeue operations, and at most $|E|$ Update operations. Therefore, the running time for Prim's algorithm is $(2|V| + |E|)O(\log |V|) \in O(|E| \log |V|)$. Exercise: How can we implement the Update operation in $O(\log |V|)$ time?

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

8.3 Dijkstra's algorithm [Section 9.3]

Given a weighted connected graph $G(V, E)$, for a given vertex s, we are interested in finding shortest paths from s to all other vertices.
Dijkstra'a algorithm is a greedy algorithm that finds such paths when the weights are non-negative.

Time complexity of Dijkstra's algorithm

When the priority queue is implemented using **min-heap**, Enqueue, Dequeue, and Update operations of the priority queue take $O(\log |V|)$ time. Prim's algorithm has $|V|$ Enqueue operations, $|V|$ Dequeue operations, and at most $|E|$ Update operations. Therefore, the running time for Prim's algorithm is $(2|V| + |E|)O(\log |V|) \in O(|E| \log |V|)$.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

Dijkstra's algorithm$(G(V, E), s)$
Initialize a priority queue PQ;
foreach $v \in V$ **do**
    $key[v]$:=∞;
    $Neighbor[v]$:=null;
end
$key[s] := 0$;
foreach $v \in V$ **do**
    Enqueue(PQ,v);
end
while $PQ \neq \phi$ **do**
    v:=Dequeue(PQ);
    foreach u *such that* $(u, v) \in E$ **do**
        if $weight(u, v) + key[v] < key[u]$ **then**
            $Neighbor[u]$:=v;
            $key[u]$:=$weight(u, v) + key[v]$;
            Update(PQ,u);
        end
    end
end
return $Neighbor[\,]$;

Computer Science Department
College of Computer and Information Sciences
King Saud University

CSC 311: Design and Analysis of Algorithms[1]
Dr. Waleed Alsalih

8.4 The Bellman-Ford algorithm

Given a weighted directed graph $G(V, E)$, for a given vertex s, we are interested in finding shortest paths from s to all other vertices.
The Bellman-Ford algorithm finds such paths even when edge weights may be negative. However, if the graph involves a negative-weight cycle reachable from s, no meaningful shortest paths can be found; in this case the Bellman-Ford algorithm will return a boolean value indicating the existence of such a cycle.

Time complexity of the Bellman-Ford algorithm

With adjacency lists representation, the Bellman-Ford algorithm runs in $O(|V||E|)$ time.

[1]This is a summary of the material we cover from the textbook: *Introduction to the Design & Analysis of Algorithms*, A. Levitin, Second Edition, Pearson Addison-Wesley, 2006.

```
Bellman-Ford algorithm(G(V, E), s)
foreach v ∈ V do
    key[v]:=∞;
    Neighbor[v]:=null;
end
key[s] := 0;
for i = 1 to |V| − 1 do
    foreach edge (u, v) ∈ E do
        if weight(u, v) + key[u] < key[v] then
            Neighbor[v]:=u;
            key[v]:=weight(u, v) + key[u];
        end
    end
end
foreach edge (u, v) ∈ E do
    if weight(u, v) + key[u] < key[v] then
        return (FALSE,Neighbor[ ])
    end
end
return (TRUE,Neighbor[ ]);
```