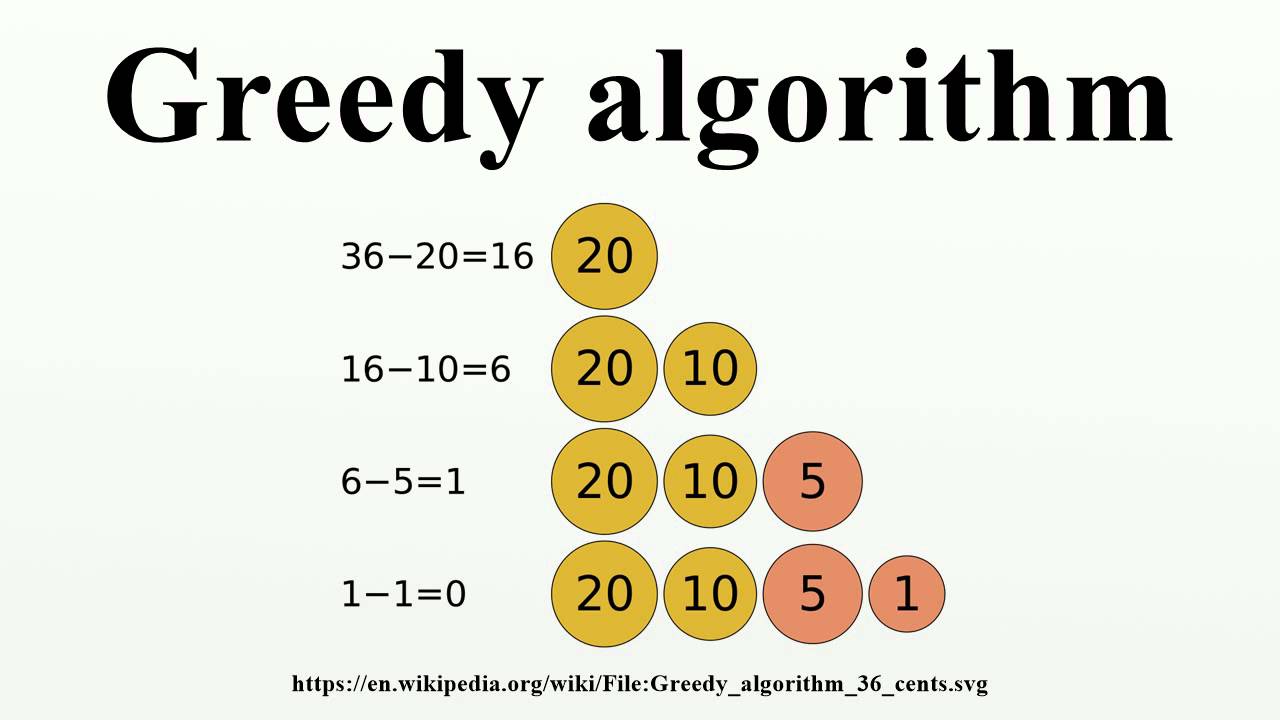
**GREEDY ALGORITHM**

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems.



**10 GREEDY ALGORITHMS**

1. **Kruskal’s Minimum Spanning Tree Algorithm**

**What is Minimum Spanning Tree?**   
Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?   
A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.   
What are the applications of Minimum Spanning Tree? 

Below are the steps for finding MST using Kruskal’s algorithm

**1.** Sort all the edges in non-decreasing order of their weight.   
**2.** Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.   
**3.** Repeat step#2 until there are (V-1) edges in the spanning tree.

**Time Complexity:**O(ElogE) or O(ElogV). Sorting of edges takes O(ELogE) time. After sorting, we iterate through all edges and apply find-union algorithm. The find and union operations can take atmost O(LogV) time. So overall complexity is O(ELogE + ELogV) time. The value of E can be atmost O(V2), so O(LogV) are O(LogE) same. Therefore, overall time complexity is O(ElogE) or O(ElogV).

1. **Huffman Coding**

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.  
The variable-length codes assigned to input characters are [Prefix Codes](http://en.wikipedia.org/wiki/Prefix_code" \t "https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/_blank), means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be “cccd” or “ccb” or “acd” or “ab”.  
   
There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

* ****Steps to build Huffman Tree****  
  Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

**Time complexity:** O(nlogn) where n is the number of unique characters. If there are n nodes, extractMin() is called 2\*(n – 1) times. extractMin() takes O(logn) time as it calles minHeapify(). So, overall complexity is O(nlogn).  
If the input array is sorted, there exists a linear time algorithm. We will soon be discussing in our next post.

#### Applications of Huffman Coding:

1. They are used for transmitting fax and text.
2. They are used by conventional compression formats like PKZIP, GZIP, etc.
3. It is useful in cases where there is a series of frequently occurring characters.
4. **Prim’s Minimum Spanning Tree (MST)**

Like Kruskal’s algorithm, Prim’s algorithm is also a [Greedy algorithm](https://www.geeksforgeeks.org/archives/18528). It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

A group of edges that connects two set of vertices in a graph is called [cut in graph theory](http://en.wikipedia.org/wiki/Cut_(graph_theory)" \t "https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/_blank). So, at every step of Prim’s algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).  
**How does Prim’s Algorithm Work?** The idea behind Prim’s algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a Spanning Tree. And they must be connected with the minimum weight edge to make it a Minimum Spanning Tree.

**Time Complexity**:

The time complexity of the Prim's Algorithm is O ( ( V + E ) l o g V ) because each vertex is inserted in the priority queue only once and insertion in priority queue take logarithmic time.

1. **Graph Coloring**

[Graph coloring](http://en.wikipedia.org/wiki/Graph_coloring" \t "https://www.geeksforgeeks.org/graph-coloring-applications/_blank) problem is to assign colors to certain elements of a graph subject to certain constraints.

****Vertex coloring**** is the most common graph coloring problem. The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using same color. The other graph coloring problems like **Edge Coloring** (No vertex is incident to two edges of same color) and **Face Coloring**(Geographical Map Coloring) can be transformed into vertex coloring.

****Chromatic Number:**** The smallest number of colors needed to color a graph G is called its chromatic number. For example, the following can be colored minimum 2 colors.

****Basic Greedy Coloring Algorithm:****

1. Color first vertex with first color.  
   **2.** Do following for remaining V-1 vertices.  
   …..**a)**Consider the currently picked vertex and color it with the  
   lowest numbered color that has not been used on any previously  
   colored vertices adjacent to it. If all previously used colors  
   appear on vertices adjacent to v, assign a new color to it.

**Complexity Analysis:**

**Time Complexity:** O(m^V).   
There are total O(m^V) combination of colors. So the time complexity is O(m^V).

**Space Complexity:** O(V).   
Recursive Stack of graphColoring(…) function will require O(V) space.

1. **Dijkstra’s shortest path algorithm**

Dijkstra’s algorithm is very similar to [Prim’s algorithm for minimum spanning tree](https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/" \t "https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/_blank). Like Prim’s MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

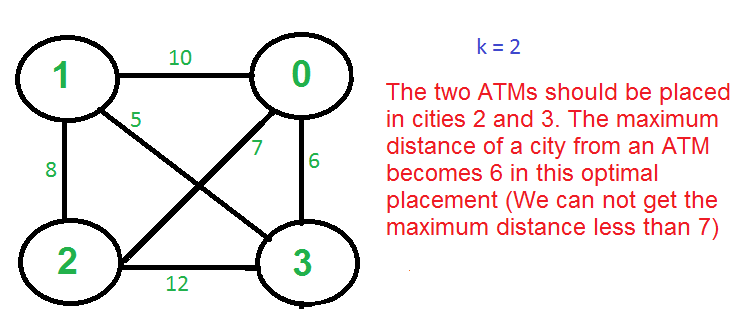
Below are the detailed steps used in Dijkstra’s algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

**Algorithm**  
**1)** Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.  
**2)** Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.  
**3)** While sptSet doesn’t include all vertices  
….**a)** Pick a vertex u which is not there in sptSet and has minimum distance value.  
….**b)** Include u to sptSet.  
….**c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

**Time Complexity** of Dijkstra's Algorithm is O ( V 2 ) but with **min**-priority queue it drops down to O ( V + E l o g V ) .

1. **K Centers Problem**

Given n cities and distances between every pair of cities, select k cities to place warehouses (or ATMs or Cloud Server) such that the maximum distance of a city to a warehouse (or ATM or Cloud Server) is minimized.   
For example consider the following four cities, 0, 1, 2 and 3 and distances between them, how do place 2 ATMs among these 4 cities so that the maximum distance of a city to an ATM is minimized.



There is no polynomial time solution available for this problem as the problem is a known NP-Hard problem. There is a polynomial time Greedy approximate algorithm, the greedy algorithm provides a solution which is never worse that twice the optimal solution. The greedy solution works only if the distances between cities follow [Triangular Inequality](http://en.wikipedia.org/wiki/Triangle_inequality" \t "https://www.geeksforgeeks.org/k-centers-problem-set-1-greedy-approximate-algorithm/_blank) (Distance between two points is always smaller than sum of distances through a third point).   
**The 2-Approximate Greedy Algorithm:**   
1) Choose the first center arbitrarily.   
2) Choose remaining k-1 centers using the following criteria.   
       Let c1, c2, c3, … ci be the already chosen centers. Choose   
       (i+1)’th center by picking the city which is farthest from already   
       selected centers, i.e, the point p which has following value as maximum   
                 Min[dist(p, c1), dist(p, c2), dist(p, c3), …. dist(p, ci)]

IMG_256

**Example (k = 3 in the above shown Graph)**   
a) Let the first arbitrarily picked vertex be 0.   
b) The next vertex is 1 because 1 is the farthest vertex from 0.   
c) Remaining cities are 2 and 3. Calculate their distances from already selected centers (0 and 1). The greedy algorithm basically calculates following values.   
        Minimum of all distanced from 2 to already considered centers   
        Min[dist(2, 0), dist(2, 1)] = Min[7, 8] = 7   
        Minimum of all distanced from 3 to already considered centers   
        Min[dist(3, 0), dist(3, 1)] = Min[6, 5] = 5   
        After computing the above values, the city 2 is picked as the value corresponding to 2 is maximum.   
Note that the greedy algorithm doesn’t give best solution for k = 2 as this is just an approximate algorithm with bound as twice of optimal.   
**Proof that the above greedy algorithm is 2 approximate.**   
Let OPT be the maximum distance of a city from a center in the Optimal solution. We need to show that the maximum distance obtained from Greedy algorithm is 2\*OPT.   
The proof can be done using contradiction.   
a) Assume that the distance from the furthest point to all centers is > 2·OPT.   
b) This means that distances between all centers are also > 2·OPT.   
c) We have k + 1 points with distances > 2·OPT between every pair.   
d) Each point has a center of the optimal solution with distance <= OPT to it.   
e) There exists a pair of points with the same center X in the optimal solution (pigeonhole principle: k optimal centers, k+1 points)   
f) The distance between them is at most 2·OPT (triangle inequality) which is a contradiction.