Greedy Algorithms

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. So, the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

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.

package Dijsktra;

// A Java program for Dijkstra's single source shortest path algorithm.

// The program is for adjacency matrix representation of the graph

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class ShortestPath {

// A utility function to find the vertex with minimum distance value,

// from the set of vertices not yet included in shortest path tree

static final int V = 9;

int minDistance(int dist[], Boolean sptSet[])

{

// Initialize min value

int min = Integer.MAX\_VALUE, min\_index = -1;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min) {

min = dist[v];

min\_index = v;

}

return min\_index;

}

// A utility function to print the constructed distance array

void printSolution(int dist[])

{

System.out.println("Vertex \t\t Distance from Source");

for (int i = 0; i < V; i++)

System.out.println(i + " \t\t " + dist[i]);

}

// Function that implements Dijkstra's single source shortest path

// algorithm for a graph represented using adjacency matrix

// representation

void dijkstra(int graph[][], int src)

{

int dist[] = new int[V]; // The output array. dist[i] will hold

// the shortest distance from src to i

// sptSet[i] will true if vertex i is included in shortest

// path tree or shortest distance from src to i is finalized

Boolean sptSet[] = new Boolean[V];

// Initialize all distances as INFINITE and stpSet[] as false

for (int i = 0; i < V; i++) {

dist[i] = Integer.MAX\_VALUE;

sptSet[i] = false;

}

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of vertices

// not yet processed. u is always equal to src in first

// iteration.

int u = minDistance(dist, sptSet);

// Mark the picked vertex as processed

sptSet[u] = true;

// Update dist value of the adjacent vertices of the

// picked vertex.

for (int v = 0; v < V; v++)

// Update dist[v] only if is not in sptSet, there is an

// edge from u to v, and total weight of path from src to

// v through u is smaller than current value of dist[v]

if (!sptSet[v] && graph[u][v] != 0 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

// print the constructed distance array

printSolution(dist);

}

// Driver method

public static void main(String[] args)

{

/\* Let us create the example graph discussed above \*/

int graph[][] = new int[][] {

{ 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 }

};

ShortestPath t = new ShortestPath();

t.dijkstra(graph, 0);

}

}

Disadvantages:

* It doesn’t works well with negative edges.