Greedy Algorithm

prem sagar

March 2017

Contents

1 Greedy Algorithmn

1.1 What is Greedy Algorithm

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage wA greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage[1] with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

1.2 Components of Greedy Algorithms:

- A candidate set, from which a solution is created
- selection function, which chooses the best candidate to be added to the solution
- feasibility function, that is used to determine if a candidate can be used to contribute to a solution
- An objective function, which assigns a value to a solution, or a partial solution, and
- A solution function, which will indicate when we have discovered a complete solution

1.3 Greedy choice property

We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the subproblem. It iteratively makes one greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution

1.4 Applications

Greedy algorithms mostly (but not always) fail to find the globally optimal solution, because they usually do not operate exhaustively on all the data. They can make commitments to certain choices too early which prevent them from finding the best overall solution later. For example, all known greedy coloring algorithms for the graph coloring problem and all other NP-complete problems do not consistently find optimum solutions. Nevertheless, they are useful because they are quick to think up and often give good approximations to the optimum.

If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods like dynamic programming. Examples of such greedy algorithms are Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees, and the algorithm for finding optimum Huffman trees.

Greedy algorithms appear in network routing as well. Using greedy routing, a message is forwarded to the neighboring node which is "closest" to the destination. The notion of a node's location (and hence "closeness") may be determined by its physical location, as in geographic routing used by ad hoc networks. Location may also be an entirely artificial construct as in small world routing and distributed hash table.

1.4.1 Examples

- Kruskal's Minimum Spanning Tree Algorithm
- Prim's Minimum Spanning Tree Algorithm
- Dijkstra's Shortest Path Algorithm
- Job Sequencing Problem
- Maximum Subarray problem
- Fractional knapsack problem

2 Minimum Spanning Tree

2.1 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.
A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

2.2 Applications of Minimum Spanning Tree Problem

Minimum Spanning Tree (MST) problem: Given connected graph G with positive edge weights, find a min weight set of edges that connects all of the vertices.
MST is fundamental problem with diverse applications.

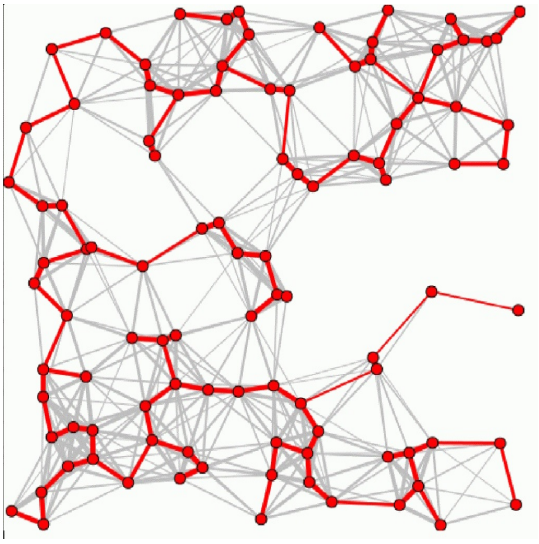

Figure 1: Application of Mst

Network design.
telephone, electrical, hydraulic, TV cable, computer, road

The standard application is to a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money. **Indirect applications**

- max bottleneck paths
- LDPC codes for error correction
- image registration with Renyi entropy
- 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

Cluster analysis
k clustering problem can be viewed as finding an MST and deleting the k-1 most expensive edges.

3 Kruskal's Minimum Spanning Tree

3.1 Procedure

- Sort all the edges in non-decreasing order of their weight.
- 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.
- Repeat step 2 until there are (V-1) edges in the spanning tree.

3.2 Algorithm

Kruskals
{
1. sortg(graph); // Sort all the edges in non-decreasing order of their weight.
2. for each edge e in sorted array
3. if endpoints of e are disconnected in s //where s is set of edge in mst
4. add e to s
}

Creating Graph in Matlab
G={V,E}
V: Vertices
E: Edge{Src,Dest,Weight}
Considera simple connected graph has e no of edges for each edge it has three value crossponding to source, destination and weight
E={src,dest,weight}
Let a graph g be a array of structure whose each value crosspond to an edge of graph
graph=struct
ith element of graph consist of edge crossponds to
graph(i).src
graph(i).dest
graph(i).weight
for i=1:e
. g(i).src=input('src:');
. g(i).dest=input('dest:');
. g(i).weight=input('weight:');

3.2.1 Function to take input graph from user

```
function g= ginput(e)
1.  %ginput Summary of this function goes here
2.  % Detailed explanation goes here
3.  disp('enter the source destination and weight of edge');
4.  fprintf('src dest weight');
5.  g=struct;
6.  for i=1:e
7.      fprintf('edge%2d',i);
8.      g(i).src=input('src:');
9.      g(i).dest=input('dest:');
10.      g(i).weight=input('weight:');
11.  end
12.  end
```

3.2.2 Function to sort the graph according to weight

```
function outStructArray = gsort( g, fieldName )
1.   if (~ isempty(g) && length (g) >0)
2.      [~,I] =sort(arrayfun (@(x) x.(fieldName), g)) ;
3.     outStructArray = g(I) ;
4.   else
5.     disp ('graph is empty');
6.  end
7.  end
```

3.2.3 A utility function to find set of an element i

```
   function y = finds(subset,i)
1.    if (subset(i).parent(~ =i)
2.    subset(i).parent=finds(subset,subset(i).parent);
3.    end
4.    y=subset(i).parent;
5.  end
```

3.2.4 function that does union of two sets of x and y

```
   function subset=union( subset,x,y)
1.    xroot = finds(subset, x);
2.    yroot = finds(subset, y);
3.    % Attach smaller rank tree under root of high rank tree
5.    % (Union by Rank)
6.    if (subset(xroot).rank <subset(yroot).rank)
7.       subset(xroot).parent = yroot;
8.    elseif (subset(xroot).rank >subset(yroot).rank)
9.       subset(yroot).parent = xroot;
10.    % If ranks are same, then make one as root and increment
11.    % its rank by one
12.    else
13.       subset(yroot).parent = xroot;
14.       subset(xroot).rank=subset(xroot).rank+1;
15.    end
16.  end
```

3.2.5 Function to Display graph

```
   function disp_mst(mst)
1.    X(v,v)=0;%X IS ADJECENCY MATRIX
2.  for i=1:v
3.      for j=1:v
4.        X(i,j)=0; %Initializing matrix element with zero
5.    end
6.      end
7.  for i=1:e
8.       X(mst(i).src,mst(i).dest)=1;
```

```
9.      X(mst(i).dest,mst(i).src)=1;
10.   end
11.    e=size(mst,2);
12.    for i=1:e
13.       fprintf('%d—-%d=%2d',mst(i).src,mst(i).dest,mst(i).weight);
14.    end
```

3.2.6 Driving program for kruskals algorithm

```
   1.   clc;
2.   fprintf('Enter no of vertices and edge in graph');
3.   v=input('no of vertices:');
4.   e=input('no of edges:');
5.   %ginput(int v) is a function which takes no of edge of graph
as input
6.   %returns a graph having source destination and weight of
crossponding edges
7.   graph=gin(e);
8.   fprintf('enter coordinates of %d vertices in order',v);
9.   V=ginput(v); %V is vx2 matrix containing coordinate of each
v vertices
10.   %sorting the edge of graph crossponding to its weight and
storing its value
11.   %in new graph graph g
12.   g=gsort(graph,'weight');
13.   subset=struct;
14.   for i=1:v
15.       subset(i).parent=i;
16.       subset(i).rank=0;
17.   end
18.   i=1;
19.   E=1;
20.   result=struct;
21.   % Number of edges to be taken is equal to v-1
22.   while(E<v)
23.   %Step 2: Pick the smallest edge. And increment the index
24.   %for next iteration
25.       next_edge=g(i);
26.       x = finds(subset, next_edge.src);
```

```
27.       y = finds(subset, next_edge.dest);
28.       i=i+1;
29.    % including this edge does't cause cycle,
30.    % include it in result and increment the index of result for
31.    %next edge
32.    if (x ~= y)
33.       result(E).src= next_edge.src;
34.       result(E).dest= next_edge.dest;
35.       result(E).weight= next_edge.weight;
36.       subset=union(subset, x, y);
37.       E=E+1;
38.    end
39.    %Else discard the next_edge
40.    end
41.    %print the contents of result[] to display the built MST
42.    disp('Following are the edges in the constructed MST');
43.    X=disp_mst(result,v); %X is adjecency matrix of mst
44.    %disp(X);
45.    %PLLOTING THE REQUIRED MST
46.    C = rand(v,1);%for choosing random color for node
47.    k = 1:v;
48.    hold on
49.    scatter(V(:,1), V(:,2), [],C,'filled');
50.    gplot(X(k,k),V(k,:),'-or');
51.    text(V(:,1), V(:,2),[repmat(' ',v,1), 52.    num2str((1:v)')]);
53.    hold off
```

3.3 Example

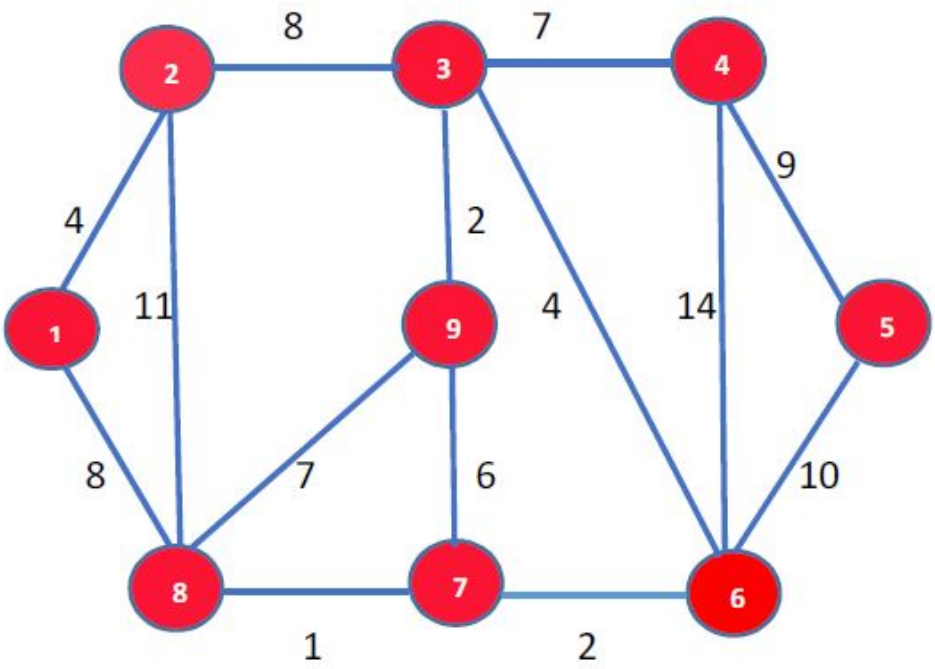

Figure 2: Given graph

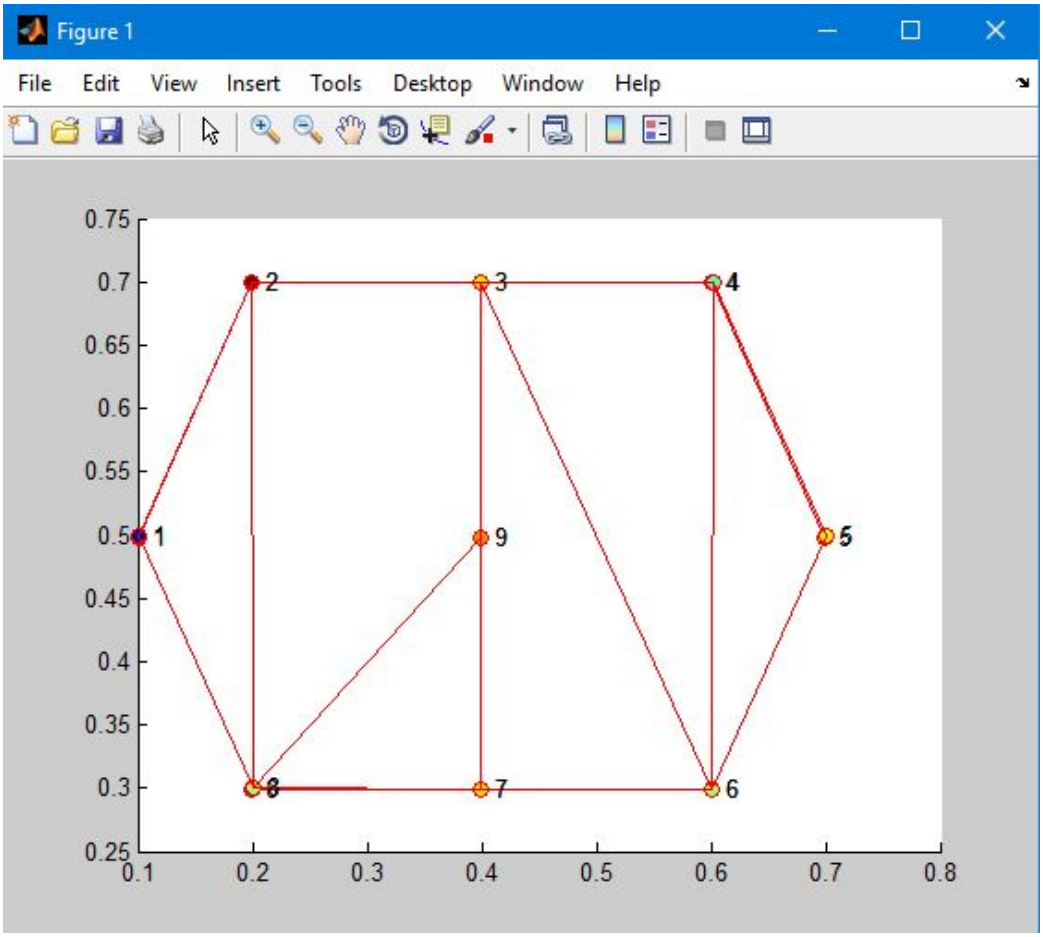

Figure 3: input graph

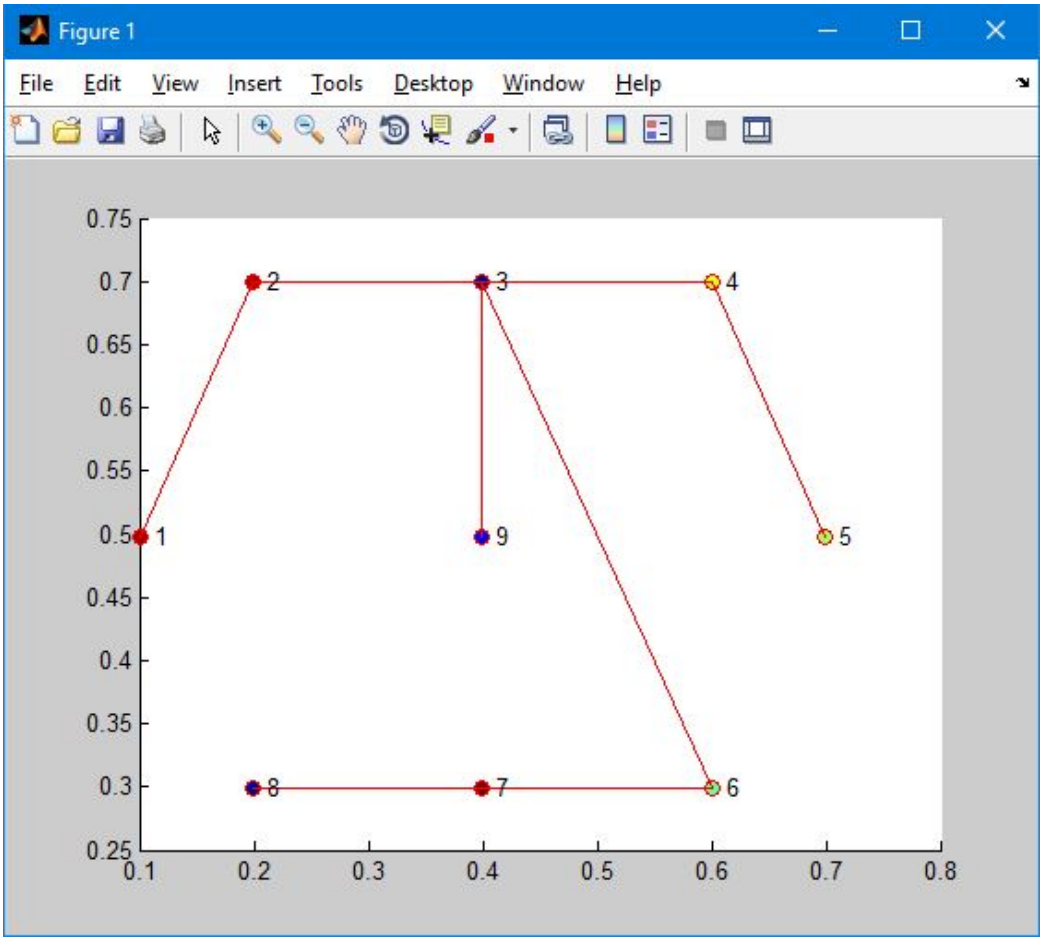

Figure 4: output mst

3.4 Time Complexity

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 $O(V^2)$, so O(LogV) are O(LogE) same. Therefore, overall time complexity is O(ElogE) or O(ElogV)