**ADVANCED ALGORITHMS**

**Project#1**

**NETWORK FLOW AND IMAGE SEGMENTATION**

**By**

**Sukesh Reddy (sg121)**

**Shashank Reddy (sa123)**

**Vandana Rao(vk34)**

**Rishi Reddy (rc81)**

**Professor**

**Dr. Zhong-Hui Duan**

**Academic Integrity Pledge**

Program: - Network Flow and Image Segmentation

Programmer: - Sukesh Reddy, Shashank Reddy, Vandana and Rishi Reddy

Academic Integrity Pledge

* All source code and documentation used in my program is either my original work, or was derived, by me, from the source code published in the text book for this course or presented in class. Any source code in this project that is derived from code published elsewhere is documented as such so the original author receives due credit.
* I have not used source code obtained from another student, or any other unauthorized source, either modified or unmodified. I have not helped another student write their program by providing a printed or electronic copy of my solution.
* I have not discussed coding details about this project with anyone other than my instructor. I understand that I may discuss the concepts of this program with other students, and that another student may help me debug my program. However, the responsibility to write each program belongs solely to the program's author.
* I have violated neither the spirit nor letter of these restrictions.

Sukesh Reddy

Shashank Reddy

Vandana Rao

Rishi Reddy 2/9/2014

Signature of Programmer Date

**Introduction:**

Image segmentation is the process of dividing the image into multiple segments by joining the pixels. Image segmentation simplifies the representation of an image by locating the object boundaries and makes it easy to analyze. In this project we are separating the image into foreground and background. For segmenting the image, we are using Ford Fulkerson algorithm which determines the maximum flow for a given path from source (S) to sink (T) in the graph. An S-T cut is a graph cut which separates the source and sink into two different subsets. We can use this cut to segment the image into foreground and background. According to the max flow min cut theorem, a maximum flow path is the minimum graph cut. So eventually we can use Ford Fulkerson algorithm to find the minimum cut and Image Segmentation consequently.

**Description and Implementation:**

In this project, we are using Ford Fulkerson algorithm for image segmentation. Ford Fulkerson Algorithm makes use of the Breadth first search (BFS) which gives the shortest path from source(S) to sink (T) in a graph (G). This project is divided into 3 different modules.

1. Breadth First Search module
2. Ford Fulkerson module
3. Image Segmentation module

**1**. **Breadth First Search**: BFS module is used for finding the shortest path from source to sink in a graph. For the BFS implementation, we used queue data structure for finding the paths from the source to sink. We are making use of colors like white, gray and black for the nodes along with the distance to make track of the nodes whether they are visited or not and how many levels it has moved. White color indicates node is undiscovered, gray color indicates node has been discovered and black color indicates that particular node neighbors are discovered and it is explored. We used arrays to store the nodes and hash maps to store each node connection (edges) with other nodes along with the weight on the edges. So we are using array as a graph in our implementation.

Input for the BFS: Graph as an array.

Output: A shortest path from source to sink

BFS(Graph, source)

For each vertex ‘u’ belongs to G.V-{s}

u.color = white

u.distance = infinity.

u.parent = null.

For the source vertex,

s.color = gray

s.distance = 0

s.parent = null

Queue = null

Enqueue(Graph, Source)

While Queue!= null

u = Dequeue

for each joining vertex from ‘u🡪 v’ in graph

if vertex ‘v’ == White

v.color = gray

v.distance = u.distance +1

v.parent = u

Enqueue(Q, v)

u.color = Black

**2. Ford Fulkerson algorithm:** Ford Fulkerson module is used to find the maximum flow from source to sink in a given graph. In this implementation, various concepts come into picture like residual graph, flow capacity, augmenting flow etc. Residual graph is a graph which consists of flow from (u🡪v) and from (v🡪u) if we consider u and v as vertices in graph. Flow capacity is the capacity on the edges from (u🡪v) or (v🡪u). Augmenting flow is the flow which increases every time the shortest path is found from source to sink in a graph.

Ford Fulkerson algorithm takes BFS shortest path as an input and finds the minimum flow capacity (cf (p)) along the edges in a shortest path found. This min flow ((u, v).f) obtained is stored in a variable and is augmented every time the shortest path is found. Min flow capacity obtained is subtracted along the edges (u🡪v) in a path and an edge with the min flow value ((v, u).f) is added in the reverse direction (v🡪u) if edge doesn’t exist else increase the flow in the reverse direction for that particular edge in a path. Later, the edges with the capacity ‘zero’ are removed in both the directions of flow. The graph obtained is nothing but a residual graph. This residual graph is given as input to the BFS function which finds the shortest path again which in-turn should be given as input to Ford Fulkerson module. So this process repeats until all the paths exhaust from source to sink in a residual graph. After all the paths exhaust, the flow ((u, v).f) which is stored in a variable which augments for the each shortest path found is the maximum flow for a graph.

(u,v).f 🡪 flow from vertex u to v.

Cf(u,v) 🡪 Capacity from vertex u to v

Input: Array of nodes and hash-map as edges between the nodes.

Output: A path of min s-t cut i.e max flow.

Ford- Fulkerson (G,s,t)

For each edge (u,v) in G,E

(u,v).f = 0

While there exists a path p from s to t in the residual network Gf

Cf(p) = min {cf(u,v) : (u,v) is in p}

For each edge (u,v) in p

If ( u,v) in E

(u,v).f = (u,v).f + cf(p)

Else (v,u).f = (v,u).f – cf(p)

**3. Image Segmentation:**

* Read the data from the .pgm file.
* Establish connections between pixels and represent as an adjacency matrix which represents the capacities.
* Array of nodes (number of row pixels) are created and adjacency matrix edge weights are stored in the hash map w.r.t each row pixel node. So here we are considering graph as an array.
* Array graph is given as input to BFS which will give us the shortest path and this shortest path is given as input to the Ford Fulkerson module which gives the maximum flow and segment the image.

Images in the foreground: pixel value

Images in the background: colorscale –pixel value

Separation among the pixels: colorscale – (difference between the pixels)

For the Image segmentation implementation, we are using 4 classes.

**Image Segmentation: -** This is the Class name where main() is present. The main implementation of this class is to connect the pixels in the image and represent it as a graph.

**Ford: -** This class consists of different methods and which are mentioned below.

Graph(graphnode[] nodesList): Copying the nodesList data to nodesList\_1.

setRoot(Node n): First node of the array list is set as the root node.

setSink(Node n): Last node of the array list is set as the sink node.

addNode(Node n): To add the node to the array list

setFileData(numberOfPixelsRowCopy, numberOfPixelsColCopy, colorScaleCopy): It sets the number of rows, columns and color scale.

nodeConnect(start node, end node, weight): To connect the nodes in the graph.

Adjacency\_list (): Creating the array of nodes and storing the edge weights of adjacency matrix in the hash map and creating the graph which is represented in array.

Ford\_BFS(Object[] graph): To calculate the shortest path.

maxflow(int maxflow): prints the max flow.

getpath(Object[] graph, Integer start\_node, Integer lastvertex): Prints the shortest path found and finds the min flow capacity on the edges in that shortest path.

Updategraph(object[] graph, Integer lastvertex): updates the path flow on the edges on the shortest path found and adds the reverse edges for a path if reverse edges doesn’t exist else updates the flow on the reverse edges if already exists.

removeedge(Object[] graph, Integer lastvertex): removes the edges with the flow capacity zero for a shortest pathfound and sends the updated residual graph to Ford\_BFS function.

createFile(): To create a new file for the resultant output image.

**Graph node:** This class consists of a constructor which is used to initialize each node with a label and its pixel color.

**Node:** This class consists of a constructor which is used to initialize each node with a name and its value i.e nothing but a capacity on the edge.It also consists of hashmap declaration where we store the edge weights.

Roles of Team Members:

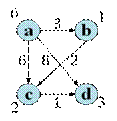
BFS and Ford Fulkerson: Rishi, Sukesh, Shashank, Vandana

Image – Graph Mapping: Rishi, Sukesh, Shashank, Vandana

Integration: Rishi, Sukesh, Shashank, Vandana

**Test Cases:**

**1. Breadth First Search:**



Adjacency list: Number of vertices 4

Start node = 0 and last node= 3

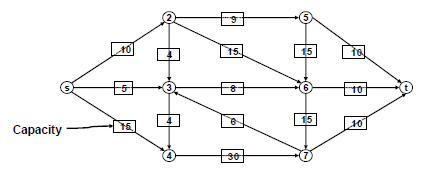
1 3 2 6 3 8

2 2

3 1

Path: 0->3

**2. Ford Fulkerson**

****

Adjacency list:

Number of vertices 8

1 10 2 5 3 15

4 9 2 4 5 15

5 8 3 4

6 30

5 15 7 10

6 15 7 10

7 10 2 6

Max flow: 28

Execution Time:

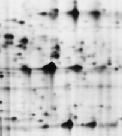
**3. Image Segmentation**

**Input Image: Output Image:**

**C:\Users\RishiReddy\Desktop\3600.jpg C:\Users\RishiReddy\Desktop\3600_out.jpg**

Execution time: 5 seconds

**Input Image: Output Image:**

Execution time: 500 seconds

**Problems Faced:**

The main implementation difficulties we faced in this project is mapping ford Fulkerson algorithm to the image pixels. First we have to read the image pixels from a .pgm file and then create nodes which are stored in an array and then making use of adjacency matrix to store the capacity of edges by finding the difference between the pixel values. Then we are converting the adjacency matrix representation to the adjacency list format which saves lot of space and creating an array graph by taking the adjacency list format as input.

**Graph:**

Figure 1: Nodes vs. time for Image Segmentation Figure 2: Nodes vs. time for Ford Fulkerson

X-axis: Number of nodes X-axis: Number of nodes

Y-axis: Time in seconds Y-axis: Time in seconds

**Complexity Analysis:**

Breadth first search algorithm the operations of enqueuing and dequeuing take O (1) time. The total time devoted to queue operations is O (V) time. The total time spent in scanning adjacency lists is O (E). So the total running time spent is O (V+E).

Ford Fulkerson Algorithm for each edge in the adjacency list it takes O (E). It takes O (V + E) to find the shortest path using Breadth first search. Iterating to each path while updating the flow capacities, adding the edges and removing the edges takes O (VE) to find the augmenting path. Thus the total time complexity of ford Fulkerson is BFS and augmenting path which is O (VE) \* O (V + E) that is O (VE²).

Image segmentation implementation we read each input from the file and store them in a matrix which takes O (E) . We convert the matrix into an adjacency list which takes O(V). We pass the adjacency list to the BFS which takes O(V+E). Then we iterate thorugh each path by adding or removing the edges which takes O(VE) for each augmenting path. Thus the total time complexity for the Image segementation is converting the matrix to adjacency list , BFS and Ford fulkerson which is O(V) + (O(VE)\*O(V+E) which is O(VE²) .

**References:**

1. Introduction to algorithms, 2nd Edition by THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, CLIFFORD STEIN
2. <http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/07MaximumFlowApplications-2x2.pdf>