Intelligent Scissors

Algorithm Project

- Faculty of Computer and Information Sciences - Ain Shams University

Team Number 8:

Members:

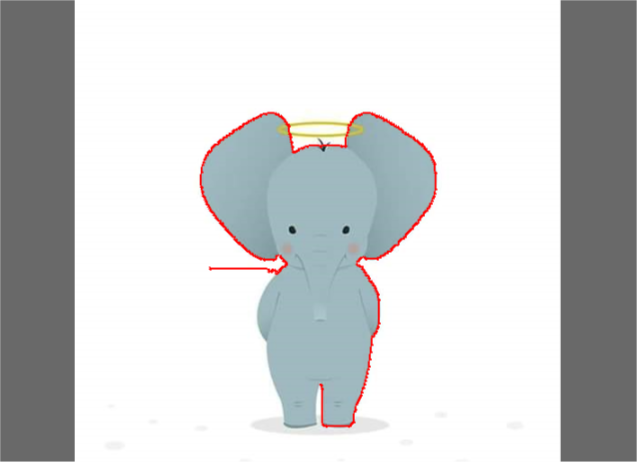
1-Ahmed Araby Kamel Mohamed. Secrion2.

Intelligent Scissors

Algorithm Project

Description of the Project:

Intelligent Scissors is an application that could do free cut for images and intelligently correct your flaws in selecting the part that you want to cut by sticking to the edges of the shape , selection here is meant to put an anchor point on the position from where the user is going to start cutting and moving with the mouse to specify area to be cut.



(Image from the running time of the project)

Technical part in the Project:

* Edge detection to detect the edges in the image that will help us in the process of correcting the user flaws.
* graph representation to represent the image in the memory To be able to work on it where each node is a pixel.
* single shortest path algorithm (DIJKSTRA) which is the core of how we correct the flaws in selecting the shape that the user want to crop.

Graph construction:

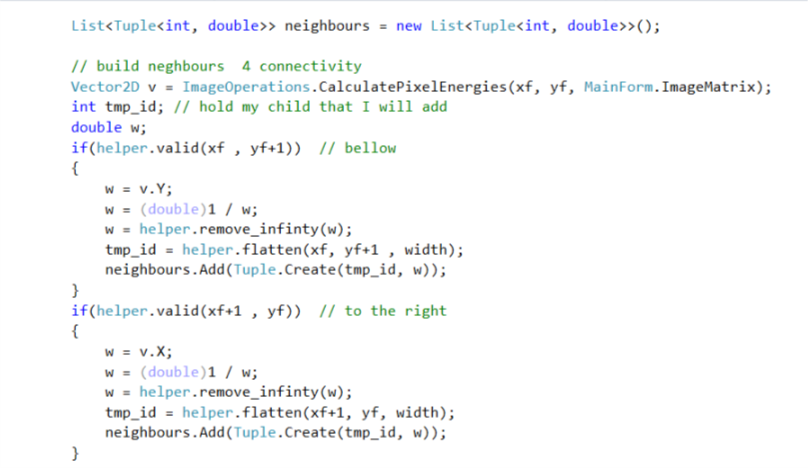
We actually do not do explicit construction for the graph because that will be not efficient in case of high quality images so we come with the idea of dealing with the graph as implicit graph which helps a lot in memory saving,

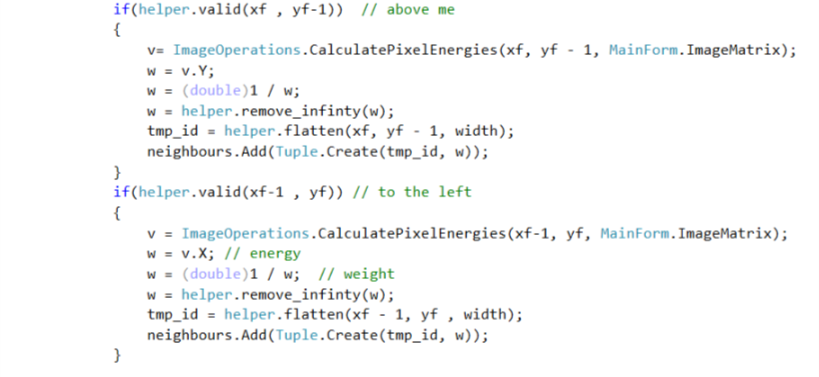
How we deal with the implicit graph?

We only need information from the graph in the step of calculating the shortest path between two pixels, and so we get the Neighbors of the pixel that we will relax from it in the moment of doing relaxation which make our graph implicit and saves memory, thus we get Neighbors in a list that contain pixel id of the Neighbors and weight between current pixel and each of theses neighbouring pixel.

Complexity of graph construction:

There is no complexity of the graph construction as a stand alone function so we will mention it’s complexity in the implementation of DIJKSTRA





Priority Queue:

* Based on min-heap data structure.
* We implement our heap as an array
* We use the priority queue in the shortest path algorithms which help us making it more efficient in complexity because it’s O(log(v)) complexity of it’s operation as we will demonstrate bellow .

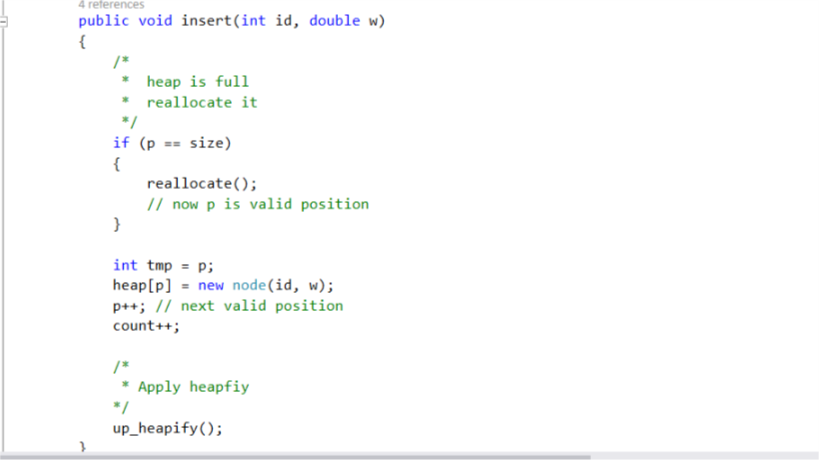
1- Priority queue Insertion:

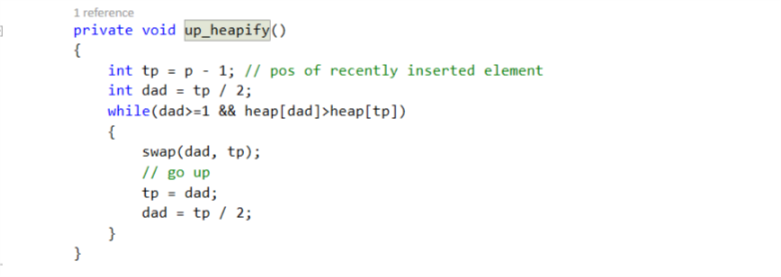
we put the newly inserted node into the first empty cell of our array

then we pull it up until we find that it’s bigger in priority than it’s parent or it’s

the root, and the complexity of this is

complexity = O (log(V)).





2- Priority queue deletion:

Actually the only node that we need to delete as we use priority queue based

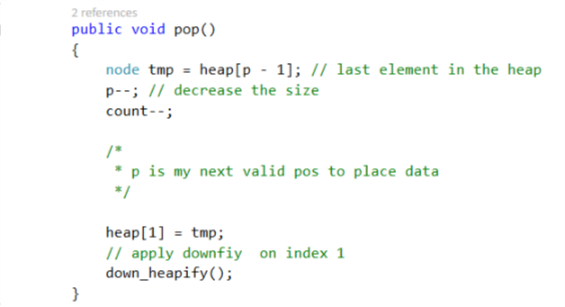
Min-heap is the node with the most priority which is the root so we delete it

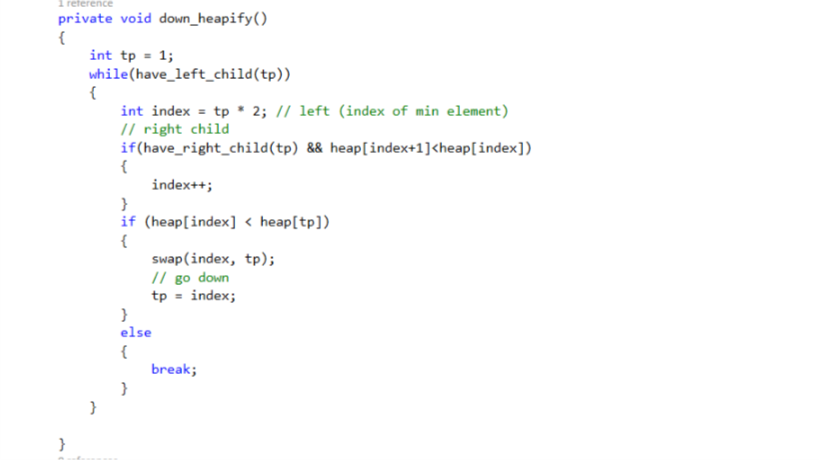
By placing a leaf in it’s place then we keep pushing this leaf down until we find

That it’s root of sub tree which all of it’s childes are less priority than him or

It become a leaf one more time, and the complexity of this process is.

Complexity =O(log(V)).





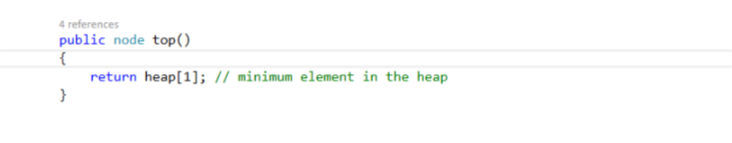
3- Priority queue top element:

It’s a main functionality for DIJKSTRA Algorithm in selecting the

Closest pixel to the current pixel that we will relax from it, it’s a very

It’s a very simple function that just take a copy of the root of the

Priority queue, and thus complexity is O(1).



Shortest Path Algorithm(DIJKSTRA):

It take destination and source which both are pixels, Source is the pixel of the current anchor point and destination is the free point of the mouse position which we use it to select out shape

But normally there will be much flaws and this algorithms is our assistant to correct this flaws,

How does this happen?

based on the edge detection algorithm of image processing we have pixels in the same edge have very small weight between them, but the pixels remaining out side the edge whether in the inside the image we want to select or out side the image which is the part that we want to discard both have big weight with the pixels in the edge itself so as the user select an anchor point in the edge then (DIJKSTRA) algorithm will be able to build a path to the free point using neighboring node in the edge also ignoring the others which give us the precise selection.

DIJKSTRA Algorithm Explanation:

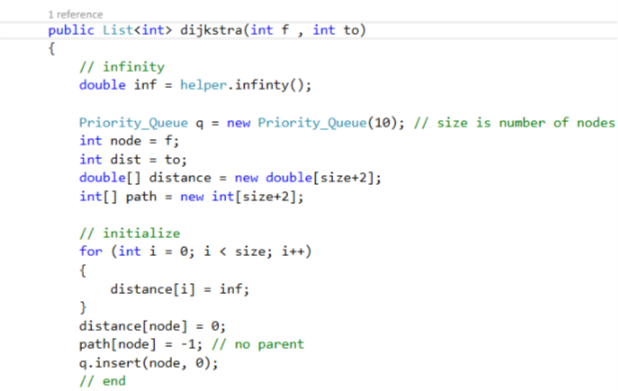
* Pick the closest pixel to the source:

It take it’s source pixel and destination pixel and then using the priority queue it does it’s gready selection to pick the closest pixel to it as current pixel then It delete this pixel from the priority queue and start to do the relaxation process using this pixel.

* Relaxation process from the current pixel that we hold:

we iterate throw all the neighbors of the current pixels and try to reach one of them with smaller weight than that it have if this happen we update the it’s weight , previous and push it into the priority queue.

Initialization process of data structures:

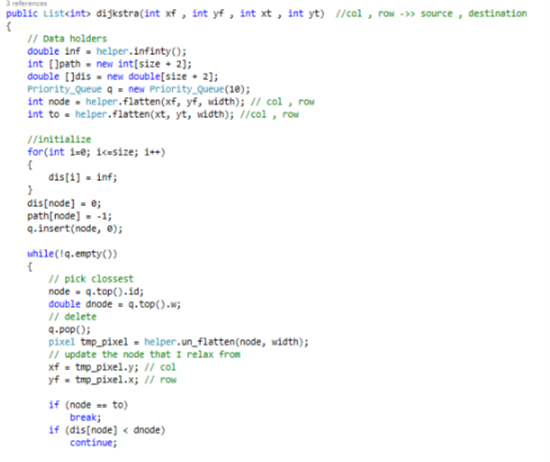


Complexity is:

we allocate in the memory by the number of pixels in the image and we also initialize the distance container which also contain (V) cells the number of pixels in the image

O(V). ->>>1

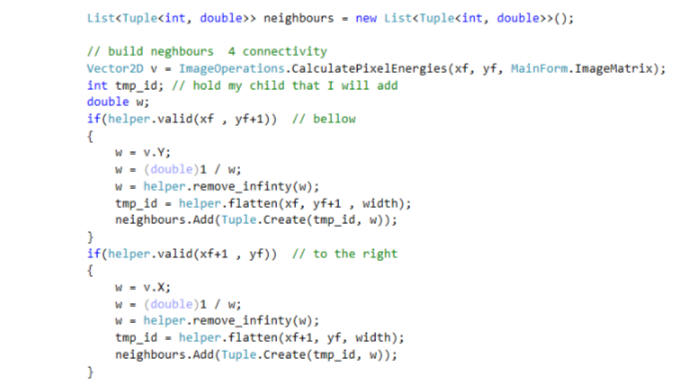
Core process of the algorithm:

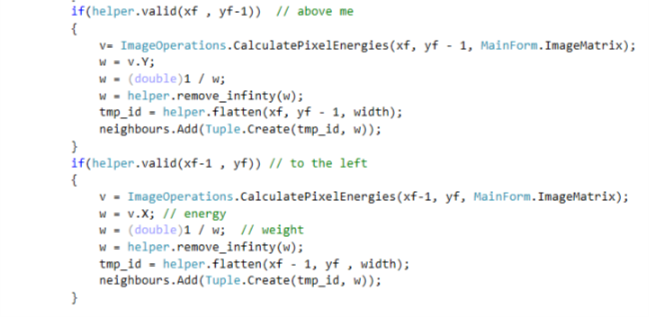


This while loop will loop for (V) iteration so this part will be repeated for (V) times, complexity of this part alone is log(v) for deleting the top element of the priority queue , thus

Complexity is:

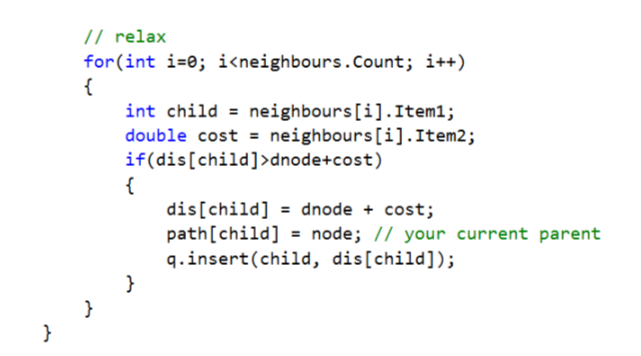
V\*O(log(V)) ->>>>> 2.1





this part get the neighbors of the current pixel (node) to apply the relax operation on them it bring just 4 connectivity each connection do it’s work In constant time , and it get repeated with the while loop for (V) iterations so we can consider it as exploring the graph

Thus Complexity is: O(4\*V) equivalently in our situation it could be O(V+E). -which is more precise ->>>2.2



This part explore the whole connection of one pixel (node) at a time which are at most 4 connections for a pixel and it update the distance and path of it’s neighbors if the current pixel provide a path that better than current exist one for each neighbor, better here mean less in distance, and it insert the neighbor that get updated in the priority queue in O(log(v)) , and it get repeated for (v) times with the while loop which also can be described as exploring the while graph.

Complexity is:

O(V+E)\*log(V). ->>>>2.3

Thus complexity of Dijkstra as all is

=1+2.1+2.2+2.3

=V+V\*log(v)+(V+E)+(V+E)\*log(V).

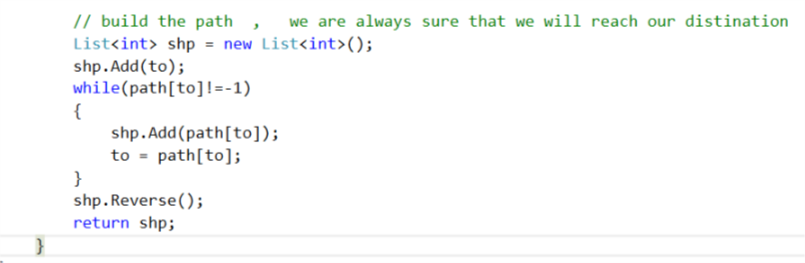
Taking into account the worst case of dense graph when E=V^2

=V+V\*log(V)+(V+V^2)+(V+V^2)\*log(v)

=V+V\*log(V)+V+V^2+Vlog(V)+V^2\*log(V).

The dominate term us V^2\*log(V) complexity is:O(V^2\*log(V)).

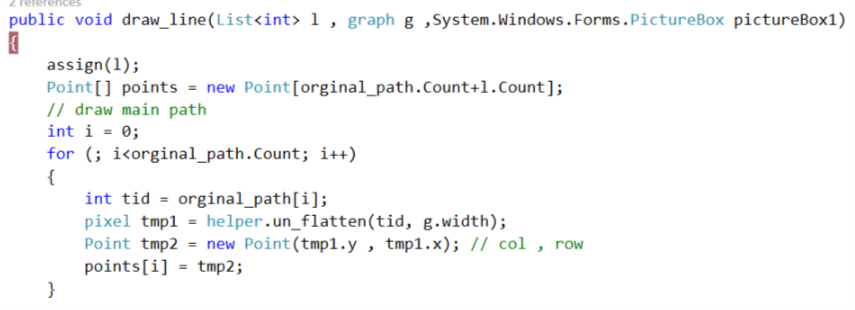
Building the path:

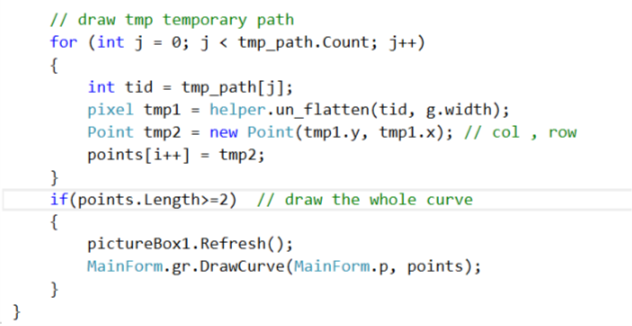


We back track the parent nodes from the destination to the source to build our path , we get the path In the reverse direction so we will reverse it to get the correct order but actually the order in this situation do not matter a lot because it we just give the DrawLine function an array of points and it draw throw them the curve so both ordering will be okay , it works in linear time.

Complexity is: O(V).

Draw the Line:





After construction of the shortest path we take the list that contain the path and give it a function DrawLine that maintain the old path that is constant with me and the temporary path that I just passed it and this function build an array of points using the two list if old path and new path

Then it call the built in function if C# drawcurve and pass to it the array of points to draw it that also take linear time

Complexity is: O(V)