CSCI 561 – Foundations of Artificial Intelligence

Instructor: Dr. K. Narayanaswamy

Assignment 1 – Search Algorithms

Due: 02/15/2013 11:59:59pm

Image stitching is a technique to generate a huge panoramic image by stitching many pieces of regular images together. One major step for image stitching is to search for a **stitching curve** over overlapping regions of every pair of adjacent images. Figure 1 gives an example of how image stitching works.

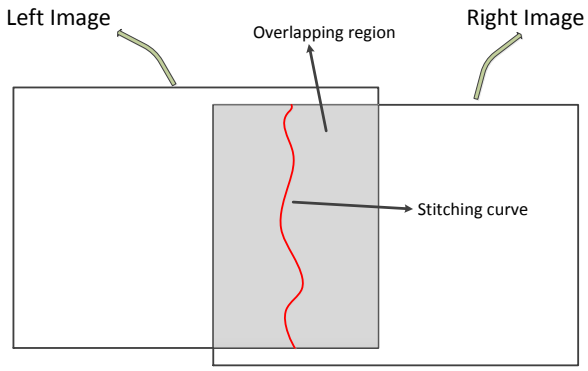


Figure 1: Image stitching

In this assignment, a simulated overlapping region is given, and you are asked to **search for stitching curves using different search algorithms (breadth-first search, depth-first search, and uniform-cost search)**. The simulated region is a small image patch with size 8×5 (i.e. 8 rows and 5 columns). Figure 2(a) shows this region, in which **circles** stand for pixels, and a **link** between two orthogonally adjacent pixels p_i, p_j indicates that a path can go from p_i to p_j directly. (Beware! Images might look differently when printed.) Coordinates of this region range from $(1,1)$ (top left pixel) to $(5,8)$ (bottom right pixel). Furthermore, there is a cost associated to each node, whose value is shown in Figure 2(b).

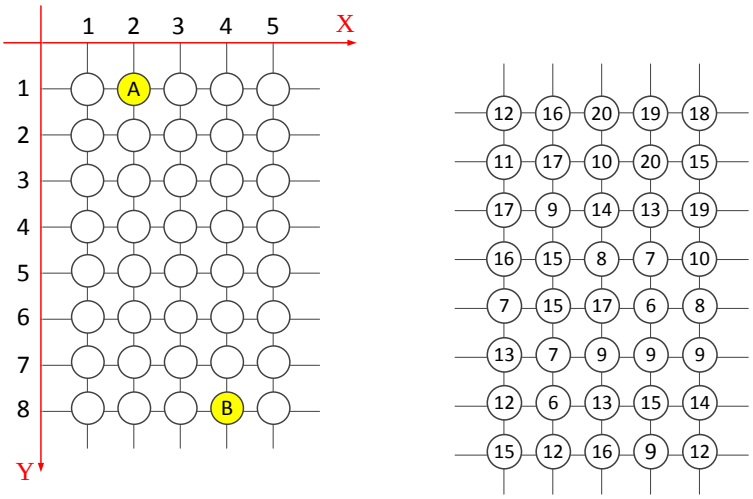


Figure 2: Simulated overlapping region (left (a) and right (b))

In this assignment, you will write a program to implement the following search algorithms, using Pixel A (2,1) as the starting pixel and Pixel B (4,8) as the ending pixel:

1. **Breadth-first search**
2. **Depth-first search**
3. **Uniform-cost search**

A sample stitching curve may take the shape as the green segments in Figure 3(a). The cost of a stitching curve is defined to be summation of costs of all pixels on that stitching curve. Figure 3(b) shows the definition of cost of a stitching curve.

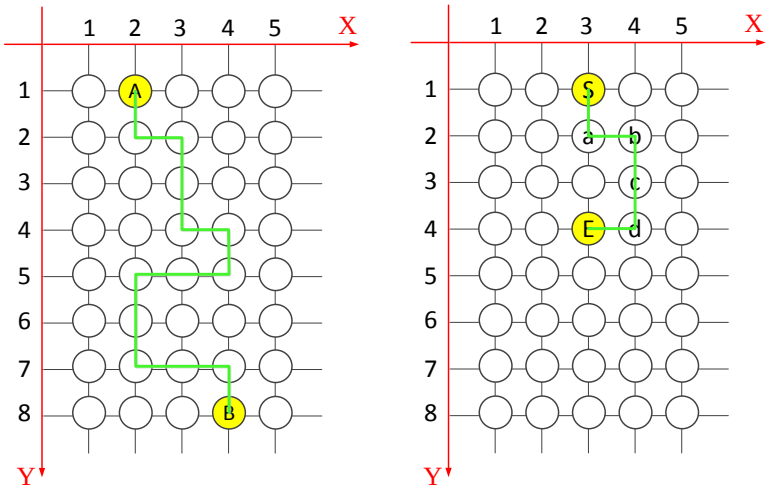
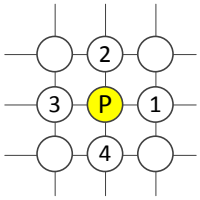


Figure 3: (a) Sample stitching curve, (b) Cost of a stitching curve: **cost of a stitching curve** $S-a-b-c-d-E$ starting at pixel S and ending at pixel E is $C_S + C_a + C_b + C_c + C_d + C_E$, where $C_i (i = S, a, b, c, d, E)$ is the cost of pixel i

Input: You should hard-code the information above in your program as a tree or a graph. This will serve as the search space for the three different algorithms: breadth-first search, depth-first search, and uniform-cost search. There is no restriction on what data structure you may use and the grading will not be based on it.

When expanding a node and there are multiple neighbors of equal cost (this will always happen in BFS and DFS, and sometimes happen in UCS), you should break tie by expanding these nodes in a **counter-clockwise** order, depicted in Figure 4.



counter-clockwise

Figure 4: Expansion order of the current pixel P, assuming all neighbors are of equal cost. The neighbor on the right is given the highest priority, followed by the neighbor above, on the left, and below. Note: numbers $1-2-3-4$ only indicate the order.

Algorithm Selection: Before executing the search, user should have the ability to indicate which search method to be used. Therefore, your program should take from standard input the name of the search algorithm to run. **Your program should have a structure which looks like the following.**

```
System.in.read(algo) // java
gets(algo) // c++
// You can use other input functions, the above is just a sample
search()
{
      switch(algo)
       case BFS:
         breadthFirstSearch()
       case DFS:
         depthFirstSearch()
       case UCS:
         uniformCostSearch()
      end
}
```

The grader will test your program by compiling ONLY ONCE, and then input different searching methods to standard input to check your results. No other changes will be performed or accommodated.

Output: In this assignment, in addition to the resulting stitching curve, we are also interested in how your search algorithms traverse the map. List the coordinates (separated by commas) in the order traversed by your program. Precisely, this is the order that your search algorithm extracts each node from the queue before expanding it. The list will vary according to the search algorithm that is selected. Also, print the resulting stitching curve as a list of coordinates separated by commas. Both lists should start with (2,1) and end with (4,8). A sample output, illustrating the output of a "random" search algorithm, is supplied below.

Sample Output:
Traversal Order:
(2,1), (2,2), (2,3), (2,4), (1,4), (3,2), (3,3), (3,4), (4,4), (4,5), (3,5), (2,5), (2,6), (2,7), (3,7), (4,7), (4,6), (3,6), (4,8)
Stitching Curve:
(2,1), (2,2), (3,2), (3,3), (3,4), (4,4), (4,5), (3,5), (2,5), (2,6), (2,7), (3,7), (4,7), (4,8)

Program Specifications:

1. Your program must be written in either Java or C++.
2. Your program MUST compile and run on aludra.usc.edu. A program that does not compile as submitted will be given 0 points. The version of Java on aludra might be older than that of the one you are using, so you have to make sure your code works fine on aludra prior to the deadline. More instructions of how to use aludra are posted on Blackboard under the "Assignments" section.
3. The answers output by your program should be calculated from your implementation of the search algorithms. Outputting hard-coded answers is considered cheating and you will be given 0 points.

Grading Policy: (Total Points: 100)

Programming (90 points):

1. Correct traversal for breadth-first search: 15 points
2. Correct solution path for breadth-first search: 15 points
3. Correct traversal for depth-first search: 15 points
4. Correct solution path for depth-first search: 15 points
5. Correct traversal for uniform-cost search: 15 points
6. Correct solution path for uniform-cost search: 15 points

Readme.txt (10 points):

1. Instructions on how to compile and execute your code. (5 points)

2. Instructions on how to select the search algorithm in your user interface. Also provide a brief description of the program structure and any other issues you think will be helpful for the grader to understand your program. (5 points)
3. **Please include your name, student ID and email address on the top.**
4. You must submit a program in order to get any credit for the Readme.txt. In short, if you submit ONLY a Readme.txt file you will get 0 points.

Submission Guidelines: Your program files will all be submitted via blackboard. Failure to follow the following guidelines will incur a -25 point penalty.

1. Compress and zip ALL your homework files (this includes the Readme.txt and all source files) into **one** .zip file. Note that only .zip file extensions are allowed. Other compression extensions such as .tar, rar, or 7z will **NOT** be accepted.
2. Name your zip file as follows: firstname_lastname.zip. For example, John_Smith.zip would be a correct file name.
3. To submit your assignment, simply select the appropriate assignment link from the Assignments subsection of the course blackboard website. Upload your zip file and click **submit** (clicking send is not enough).

Please make sure ALL source files are included in your zip file when submitted. Only your FINAL submission will be graded.

For policies on late submissions, please see the Syllabus from the course home page. These policies will be enforced with **no exceptions**.

Plagiarism on Programming Assignments: Discussion of concepts, design issues, and programming ideas with colleagues and fellow students is proper and an integral part of learning. However, everyone should do their own coding. Code will be automatically compared and any suspected cases of plagiarism will be investigated and pursued thoroughly. If you happen to reuse snippets of code from the Internet or other sources, please document the sources and nature of what was used in detail to avoid potential problems. If you have any questions or doubts regarding the proper modes of collaboration or using external reference sources, please check with the Professor or TA before you submit your assignment.