CS302 Final Exam – Fall, 2009 – James S. Plank

Question 1

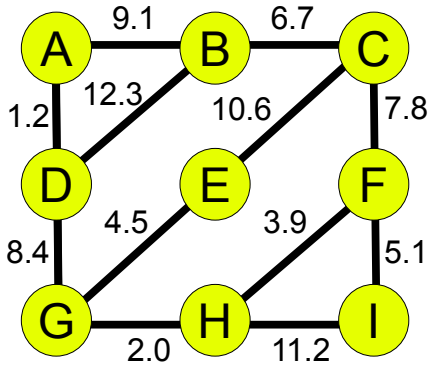
In each of the following parts, you will be given two algorithms, one labeled A and one labeled B. For each part, you need to answer one of the following: "A is faster than B." "B is faster than A", "They are the same." Use the answer sheet, and unless it is specified, you are comparing big-O running time of the worst case. For sorting algorithm, the number of items is *n*. For graph algorithms, the graph has *n* nodes and *O(n)* edges. For Union-Find, the number of items is *2n*, and there have been a variety of Union and Find operations that have been executed already. Assume the best implementation.

	Algorithm A	**Algorithm B**
Part 1:	Merge Sort	Quicksort
Part 2:	Depth First Search	Network Flow
Part 3:	Shortest path from *a* to *b* on an unweighted graph	Depth First Search
Part 4:	Heap Sort	Insertion Sort
Part 5:	Heap Sort	Merge Sort
Part 6:	Bucket Sort	Merge Sort
Part 7:	Performing *n* Union operations	Performing *n* Find operations
Part 8:	Minimum spanning tree using Prim's algorithm	Dijkstra's Algorithm.
Part 9:	Minimum spanning tree using Kruskal's algorithm	Performing *n* Union operations
Part A:	Merge Sort	Cycle detection
Part B:	Shortest path from *a* to *b* on an unweighted graph	Determining connected components
Part C:	Performing *n* Find operations	Doing matching on a bipartite graph
Part D:	Bucket Sort	Traversing a STL map with *n* elements.
Part E:	Edmonds-Karp	Dijkstra's Algorithm
Part F:	The dice-or-no-dice lab, *n*-sided die, 10 throws	Minimum spanning tree using Prim's algorithm

Question 2:

Given the graph on the right:

Suppose you run Dijkstra's algorithm on this graph starting at node **A**. List the order in which the nodes are visited.



Question 3:

Give the order of the edges that are added to the minimum spanning tree of the graph to the right using Prim's Algorithm and using Kruskal's Algorithm. If the algorithm requires a starting node, use node A.

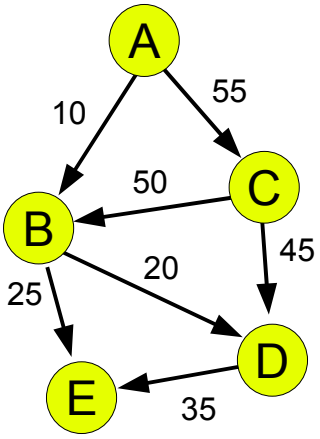
Question 4:

```
class Node {
  public:
    vector <class Node *> edges;
    int tmp;
};
class Graph {
  public:
     vector <Node *> nodes;
     int Shortest_Path(Node *a, Node *b);
};
```

Given the class definitions above for an unweighted graph, implement **Graph::Shortest_Path(a, b),** which returns the length of the shortest path from node a to node b, and returns -1 if there is no path.

Question 5:

You are given the directed graph to the right with source **A** and sink **E**.



What is the maximum flow through the graph?
What is the minimum cut of the graph?

We discussed three augmenting path algorithms in class: Greedy DFS, Modified Dijkstra, and Edmonds-Karp. For each of these algorithms, I want you to tell me three things about the algorithm on this graph:

A. What is the first augmenting path and what is its flow?
B. What is the second augmenting path and what is its flow?
C. What is the residual graph after the first augmenting path is processed?

To be specific, your answer needs to have 11 components:

1. The maximum flow through the graph. This is a number.
2. The minimum cut through the graph. You should know how to specify a cut.
3. For Greedy DFS: The first augmenting path and its flow. You may specify a path as, for example, "ACBDE".
4. For Greedy DFS: The second augmenting path and its flow.
5. For Greedy DFS: The residual graph after the first augmenting path is processed.
6. For Modified Dijkstra: The first augmenting path and its flow.
7. For Modified Dijkstra: The second augmenting path and its flow.
8. For Modified Dijkstra: The residual graph after the first augmenting path is processed.
9. For Edmonds-Karp: The first augmenting path and its flow.
10. For Edmonds-Karp: The second augmenting path and its flow.
11. For Edmonds-Karp: The residual graph after the first augmenting path is processed.

Put your answer on the answer sheet provided. Also, I have included some work sheets. DO ALL OF YOUR WORK ON THE WORK SHEETS AND THEN GIVE ME NICE NEAT ANSWERS ON THE ANSWER SHEET. If your answer is messy, get another answer sheet and redo it. Do not hand in the work sheet.

Question 6:

This is a dynamic programming problem. Suppose the functions **F1(int x, int y)** and **F2(int x, int y)** are all defined for you, and they both return positive doubles. We define the function **X(int x, int y)** as follows:

- *X(0,0)* equals 1.
- *X(x,y)* is equal to negative infinity if either *x* or *y* is less than zero.
- Otherwise, *X(x,y)* equals the maximum of *(F1(x,y)+X(x-1,y)* and *(F2(x,y)+X(x,y-1))*.

Behold the class definition to the right.

Your job is to implement CalcX::X using a recursive dynamic program with memoization.

Do a good job with this – try not to give me sloppy code. You may not add anything to the class definition.

```
typedef vector <double> Dvec;

class CalcX {
  protected:
    vector <Dvec> cache;
  public:
    double X(int x, int y);
};
```

Answer Sheet: Name & Preferred Email:

Question 1: Circle your answers:

Part 1:	A is faster than B.	B is faster than A.	They are the same.
Part 2:	A is faster than B.	B is faster than A.	They are the same.
Part 3:	A is faster than B.	B is faster than A.	They are the same.
Part 4:	A is faster than B.	B is faster than A.	They are the same.
Part 5:	A is faster than B.	B is faster than A.	They are the same.
Part 6:	A is faster than B.	B is faster than A.	They are the same.
Part 7:	A is faster than B.	B is faster than A.	They are the same.
Part 8:	A is faster than B.	B is faster than A.	They are the same.
Part 9:	A is faster than B.	B is faster than A.	They are the same.
Part A:	A is faster than B.	B is faster than A.	They are the same.
Part B:	A is faster than B.	B is faster than A.	They are the same.
Part C:	A is faster than B.	B is faster than A.	They are the same.
Part D:	A is faster than B.	B is faster than A.	They are the same.
Part E:	A is faster than B.	B is faster than A.	They are the same.
Part F:	A is faster than B.	B is faster than A.	They are the same.

Question 2: _____

Question 3: Order of edges in Prim's Algorithm: _____

Question 3: Order of edges in Kruskal's Algorithm: _____

Question 5:

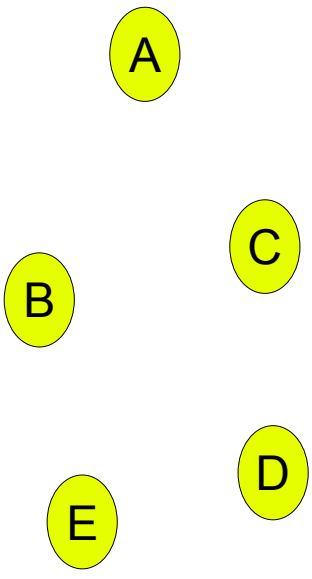
Max Flow:_____ Minimum Cut: _____

Greedy DFS:

1st Path & Flow: _____

2nd Path & Flow: _____

Residual after 1st Path Below:

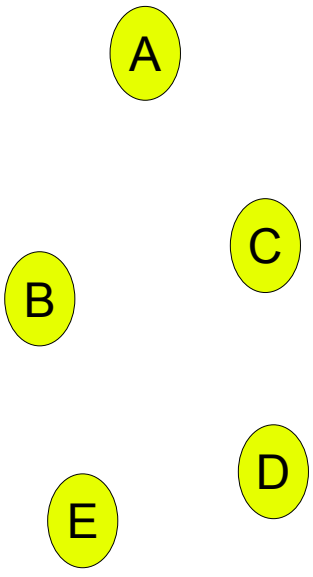


Modified Dijkstra

1st Path & Flow: _____

2nd Path & Flow: _____

Residual after 1st Path Below:

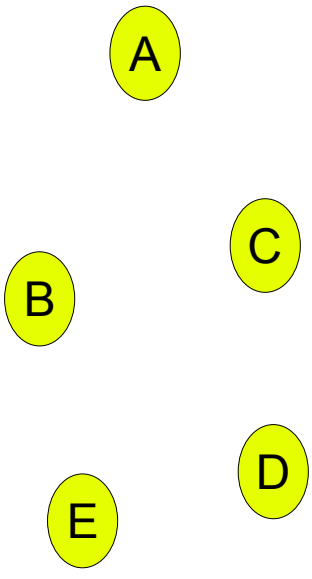


Edmonds-Karp

1st Path & Flow: _____

2nd Path & Flow: _____

Residual after 1st Path Below:



Answer Sheet, Page 2: Name & Preferred Email:

Question 4:

```
int Graph::Shortest_Path(Node *a, Node *b)
{









}
```

Question 6:

```
double CalcX::X(int x, int y)
{









}
```