

	BCDEA\n ABCDE\n
H\n	H\n
GEXKAL\n	EGXKAL\n EGXKAL\n EGKXAL\n AEGKXL\n AEGKLX\n

Problem 5: [Merge Sort](#) (20p)

Your program needs to read a single line of ASCII characters. The characters are constrained to 'A' - 'z' only. The length of the line is unknown, but well less than 100 characters. The line is terminated by '\n'. Sort all individual characters using the merge sort algorithm.

Remember that this algorithm uses a "divide&conquer" approach, by which it repeatedly breaks the sequence of characters into smaller portions before sorting (left half and right half). For your convenience, in this problem the number of input characters is a power of 2 (so you can repeatedly divide your input in halves if required).

Use **recursion** to implement merge sort; i.e. make your own "sort" function call itself recursively with smaller parts of the data. Sort the "left half" first. Whenever one instance of a recursion finishes print the **result of this recursion only**, followed by '\n'. Print the final sorted list exactly once at the end of your program.

Hint: Have your main program call your sort function with the full character sequence to start the recursion; this call will typically also print the final solution.

Example Input	Required Output
ABCD\n	AB\n CD\n ABCD\n
HGFEDCBA\n	GH\n EF\n EFGH\n CD\n AB\n ABCD\n ABCDEFGH\n

Problem 6: [Breadth First Search \(BFS\)](#) (15p)

Your program shall read in a graph in adjacency list format and perform a breadth first search (BFS) on the graph, starting at the node first read. Each node's name is a single letter 'A'-'Z', so you can safely assume less than 27 nodes for your graph. The format of each line to be read in is specified as

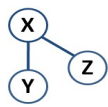
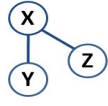
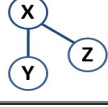
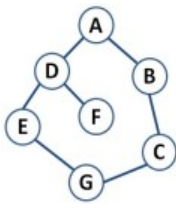
<nodeName>-<string of neighboring node names>\n (see example below).

The requested output of your program is one line of node names followed by '\n', in which the order of names represents one possible exploration following breadth first search.

Note (1): for most inputs several different outputs are possible. Here we only ask you to find one of them!

Note (2): all graphs here are undirected, i.e. if a connection **A- . . . B . . .** exists you will also find a connection **B- . . . A . . .**

Note (3): the provided example inputs are easy cases. In order to get full marks, your code needs to be able to solve more complex tasks (i.e. larger graph sizes, varying ratio of nodes and edges in the graphs, special graph topologies, ...)

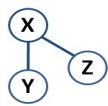
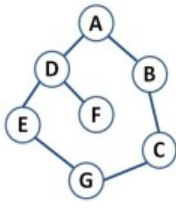
Example Input	Graph View	Required Output (one example solution)
X-YZ\n Y-X\n Z-X\n		XYZ\n
X-YZ\n Y-X\n Z-X\n		XZY\n
Y-X\n X-YZ\n Z-X\n		YXZ\n
A-BD\n B-AC\n C-BG\n D-AEF\n E-DG\n F-D\n G-CE\n		ADBEFCG\n

Problem 6B: Breadth First Search -BONUS- (10p)

Same situation as in problem 6 (BFS), but here your program needs to find **all** possible BFS explorations for the given graph and the given starting node, not just one of all possibilities. No particular order of your BFS solutions required.

Note (1): this might seem simple, but I believe it's quite a tricky problem; hence a bonus question.

Note (2): as soon as your program starts exploring a node's children, it should add all this node's children to the queue. Do NOT hop to a different node before all children of the current node have been enqueued (e.g. in the example 2 below: ABDCEFG is NOT a valid solution: the program started exploring D's children and enqueued F, so it has to finish D by enqueueing E before moving to B and enqueueing C).

Example Input	Graph View	Required Output (order of lines irrelevant)
X-YZ\n Y-X\n Z-X\n		XYZ\n XZY\n
A-BD\n B-AC\n C-BG\n D-AEF\n E-DG\n F-D\n G-CE\n		ABDCEFG\n ABDCFEG\n ADBEFCG\n ADBFECG\n

Problem 7: Depth First Search (DFS) (15p)

Your program shall read in a graph in adjacency list format and perform a Depth-First-Search (DFS) on the graph, starting at the node first read. Each node's name is a single letter 'A'-'Z', so you

can safely assume less than 27 nodes for your graph. The format of each line to be read in is specified as

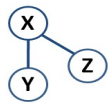
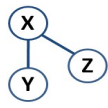
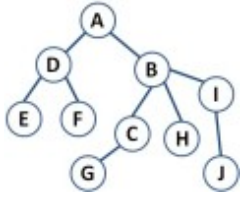
<nodeName>-<string of neighboring-node-names>\n (see example below).

The requested output of your program is one line of node names followed by '\n', in which the order of names represents one possible exploration following depth first search.

Note (1): for most inputs several different outputs are possible. Here we only ask you to find one of them!

Note (2): all graphs here are undirected, i.e. if a connection **A-...B...** exists you will also find a connection **B-...A...**

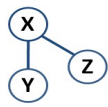
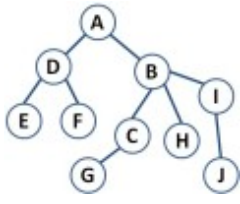
Note (3): the provided example inputs are easy cases. In order to get full marks, your code needs to be able to solve more complex tasks (i.e. larger graph sizes, varying ratio of nodes and edges in the graphs, special graph topologies, ...)

Example Input	Graph View	Required Output (one example solution)
X-YZ\n Y-X\n Z-X\n		XYZ\n
X-YZ\n Y-X\n Z-X\n		XZY\n
A-BD\n B-ACHI\n C-BG\n D-AEF\n E-D\n F-D\n G-C\n H-B\n I-BJ\n J-I\n		ADEFBCGHIJ\n

Problem 7B: Depth First Search -BONUS- (10p)

Same situation as in problem 7 (DFS), but here your program needs to find **all** possible DFS explorations for the given graph and the given starting node, not just one of all possibilities. No particular order of your DFS solutions required.

Note: again (as in 6B) this might seem simple, but I believe it's quite a tricky problem; hence a bonus question.

Example Input	Graph View	Required Output (order of lines irrelevant)
X-YZ\n Y-X\n Z-X\n		XYZ\n XZY\n
A-BD\n B-ACHI\n C-BG\n D-AEF\n E-D\n F-D\n G-C\n H-B\n I-BJ\n J-I\n		ADFEBIJHCG\n ADFEBHICJG\n ADFEBIJCGH\n ADFEBCGIJH\n ADFEBHCGIJ\n ADFEBCGHIJ\n ADEFBIJHCG\n ADEFBHICJG\n ADEFBIJCGH\n ADFEBCGIJH\n ADFEBHCGIJ\n ADEFBCGHIJ\n

```

ABIJHCGDFE\n
ABIJHCGDEF\n
ABHIJCGDFE\n
ABHIJCGDEF\n
ABIJCGHDFE\n
ABIJCGHDEF\n
ABCGIJHDFE\n
ABCGIJHDEF\n
ABHCGIJDFE\n
ABHCGIJDEF\n
ABCGHIJDFE\n
ABCGHIJDEF\n

```

Problem 8: Dijkstra's Algorithm (30p)

Your program shall read in a graph with associated costs for traversing edges, and find optimal paths from a starting node to all other nodes in this graph. This is the same problem as finding shortest distances on highways that connect cities! Each node's name is a single letter 'A'-'Z', so you can safely assume less than 27 nodes for your graph. Each connected pair of nodes will be provided as an independent input line in the following format:

<nodeNameA>-<nodeNameB>-<cost>\n (see example below).

You can assume the cost of traveling between neighboring nodes to be a positive integer number below 100 for each single edge: **0 < edge-cost < 100**.

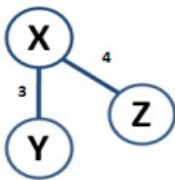
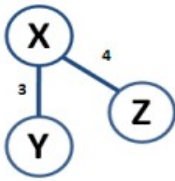
Your program needs to traverse the graph following Dijkstra's algorithm to find the shortest routes from a given starting node S to all other nodes in the graph. S is the one node that was mentioned first in the first line of input. When done, for each node N that exists in the graph, your program shall provide the backtracking path from N to S with the final (minimal) associated cost for traveling between N and S. Print exactly one such line (path and cost) for each node that exists in the system (see example below); the ordering of your lines does not matter.

Note (1): the starting node S is also a node in the system, with traversal cost of 0.

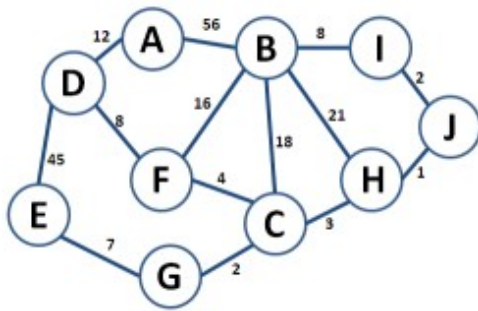
Note (2): all graphs here are undirected, i.e. if an edge **A-B-cost** exists you will also find an edge **B-A-cost**.

Note (3): you cannot assume the input to be ordered (e.g. alphabetical order, nodes presented in complete batches)

Note (4): the provided example inputs are easy cases. In order to get full marks, your code needs to be able to solve more complex tasks (i.e. larger graph sizes, varying ratio of nodes and edges in the graphs, special graph topologies, ...)

Example Input	Graph View	Required Output (order of lines irrelevant)
X-Y-3\n X-Z-4\n Y-X-3\n Z-X-4\n		X-0\n Y-X-3\n Z-X-4\n
Y-X-3\n X-Y-3\n X-Z-4\n Z-X-4\n		X-Y-3\n Y-0\n Z-X-Y-7\n
A-B-56\n A-D-12\n B-A-56\n B-C-18\n		A-0\n B-F-D-A-36\n C-F-D-A-24\n D-A-12\n

B-F-16\n
B-H-21\n
B-I-8\n
C-B-18\n
C-F-4\n
C-G-2\n
C-H-3\n
D-A-12\n
D-E-45\n
D-F-8\n
E-D-45\n
E-G-7\n
F-B-16\n
F-C-4\n
F-D-8\n
G-C-2\n
G-E-7\n
H-B-21\n
H-C-3\n
H-J-1\n
I-B-8\n
I-J-2\n
J-H-1\n
J-I-2



E-G-C-F-D-A-33\n
F-D-A-20\n
G-C-F-D-A-26\n
H-C-F-D-A-27\n
I-J-H-C-F-D-A-30\n
J-H-C-F-D-A-28

You are done! No A*-Algorithm in this programming assignment!