Project instructions for CS358: Data Structure (A)

Shanghai Jiao Tong University

Introduction

We will have four projects for this course. In each project, you are requested to complete three or four questions. For questions labeled "Report needed", you are requested to complete the tasks using C++, write a brief report to explain your design, and analyze the time complexity for each algorithm required. For questions labeled "Report not needed", you are requested to complete the tasks using C++, but you do not need to write the report. For questions labeled "Bonus", you may choose whether to complete them by yourselves. If you should choose to complete them and if your answer is (partially) correct, you will be given a maximal of 1 extra point for each bonus question. This can make up for your deduction of marks for mistakes in required questions, but your total points gained for all project questions may not exceed 20.

Each project counts for a total of five points in your final score. We will decide the points you obtain based on the correctness of your codes and report. Questions are originally designed by the teaching group and are of a relatively high difficulty. You will NOT find any existing solutions on the web, but you have to think very hard and try your best efforts to complete each task. We strongly recommend you to start early; otherwise you may have a hard time when a deadline approaches.

Policy: Each student has another total of three free late days for submissions. The calculation of the free late submission days will be independent from the statistics for homework submissions. Submissions exceeding the free late days will NOT be considered.

IMPORTANT NOTICE: Discussions among students are encouraged, but plagiarism is STRICTLY PROHIBITED. We will check and treat it VERY SERIOUSLY. Once a plagiarism is found, all students involved into the event will obtain a FAILURE GRADE for this course.

Deadlines:

Deadline for Project 1: 23:59, Nov 3 2014
Deadline for Project 2: 23:59, Nov 17 2014
Deadline for Project 3: 23:59, Dec 1 2014
Deadline for Project 4: 23:59, Dec 15 2014

Project 1: Lists

Question 1.1 (Report needed)

Indexing. Given a page of English text, create an indexed list of words sorted alphabetically. (If you do not know what is Alphabetical order, see here: http://en.wikipedia.org/wiki/Lexicographical_order)

One test case for example:

Input:

The input is a file containing the following paragraph:

"*Why, my dear, you must know, Mrs. Long says that Netherfield is taken by a young man of large fortune from the north of England;*"

Output:

The output should be a file, with each line keeps the following format ('_' means a space):

Alphabet:_word1_word2_word3_..... ,

where Alphabet should be one of the letters from A-Z and the alphabets should increase alphabetically from the first line to the last line. In addition, the following words contain the same initials as that alphabet and they should also be sorted alphabetically in ascending order. Consequently, part of the output based on the input given above should be:

……

K: know

L: large, Long

M: man, Mrs., must, my

N: Netherfield, north

O: of

……

Question 1.2 (Report not needed)

Input two decimals, or two integers, or one decimal and one integer, each number being of an arbitrary length. Denote them as *x* and *y*. You are required to use linked lists to store these two decimals, each element storing one digit only (or the decimal point). You are required to implement a class that supports operations plus, minus and multiplication, i.e., *x* + *y*, *x* – *y* and *x* * *y*. If *y* is an integer, you are also requested to calculate x^y.

Test cases:

Input:

x= 3.14159265358979323846, y= 0.5772156649015328

Output:

Print the results of x+y, x-y and x*y.

Input:

x= 3.14159265358979323846, y= 2

Output:

Print the results of x+y, x-y, x*y and x^y.

Question 1.3 (Bonus)

A polynomial can be stored in a linked list. Each summand of the polynomial is represented by its coefficient and exponent, with the two parameters stored in a node of a linked list. For example, $x^5 + 2x^4 + 5x^2 + 1$ can be stored in a list as follows:

Please design a function to accomplish the addition of two polynomials.

Test case:

Input:

1x^5+2x^4+5x^2+1x^0 (standing for $x^5 + x^4 + 5x^3 + 1$)

6x^4+4x^2-3 (standing for $6x^4 + 4x^2 - 3$)

Output:

1x^5+7x^4+5x^3+4x^2-2

Project 2: Trees

Question 2.1 (Report needed)

Consider the emergency department of a hospital. When a patient arrives, he will first receive initial check and be determined the emergency level of his/her physical condition (e.g., a patient may be judged as “critical”, “urgent”, “semi-urgent” or “non-urgent”). All patients wait for doctors in a queue, but once there is a vacancy to see the doctor, the patient with the most emergent level will be called to see the doctor first. If multiple patients are equally emergent, then the first arriver will be the first to see the doctor. During the waiting process, his/her emergency level of physical condition may change (once it changes, the patient is regarded as a new arriver in the new queue (with arrival time being the time of change level). Patients may also choose to leave the queue by themselves.

In this problem, you are required to complete the following two tasks:

(1) We assign each patient with his/her name, ID and emergency level. A higher value of emergency level means a more serious physical condition and should be considered with a higher priority. Use two methods to implement this priority queue.

(a) Heap. The priority of a node represents the patient’s emergency level. You are required to support the following functions: enqueue (a new patient arrives), dequeue (a patient sees the doctor), updatePriority (the emergency level changes), remove(key) (a patient leaves the queue before he/she sees the doctor). Hint: Read page 185 of the textbook.

(b) Multiple queues using linked list. Each queue serves for patients with the same emergency level. Only when the queue with a higher level is empty, the patients in the lower level emergency queue begin to see the doctor. You are required to maintain k queues using linked list (where k is the number of possible emergency levels), each supporting enqueue (a new patient arrives), dequeue (a patient sees the doctor), remove(key) (a patient leaves the queue before he/she sees the doctor). You should also implement the function switchQueue (the emergency level changes).

Input: In each line, you receive a message concerning an operation of a patient or doctor. Possible cases include:

1) Enqueue, Tom, 123, 4 (representing that a patient named Tom with ID 123 arrives at the hospital. His emergency is 4).
2) Quit, 123 (representing that a patient with ID 123 quits the queue).
3) Update, 123, 5 (representing that a patient with ID 123 changes his/her emergency level to 5).
4) Dequeue (representing that a doctor is available and one selected patient should see the doctor)
5) Print (representing a request of printing out all waiting patients in order of time to be

expected to see the doctor)

Output: You don't need to have any output for input cases 1, 2 and 3. For input case 4, you should print out the name and ID of the patient to see the doctor. For input case 5, you should print out the sequence of patients with their priority from highest to lowest, using the heap and multiple queue implementations, respectively.

(2) Compare the above two approaches.

Let k be the number of emergency levels, and n_i(i=1,2,…,k) be the average number of patients of level k waiting in the queue. Analyze the average time complexity of each approach. Draw a conclusion whether method (a) or (b) is preferred (in terms of time complexity needed) depending on the relationship of k and n_i.

Question 2.2 (Report not needed)

Given the preorder enumeration and the inorder enumeration of a binary tree,

(1) determine its postorder enumeration;
(2) calculate the height of the tree;
(3) given a key value, search if it appears in a node in the tree; if yes, calculate the numbers of descends and ancestors of the node.

One test case:
Input:
preorder: *Y A L B E C D W X M*
inorder: *B L E A C W X D Y M*
Output:

a) Print the postorder enumeration and the height of the tree.
b) If the given key value is '*N*', your search function should return false. If the given key value is '*C*', print out the numbers of descends and ancestors of the node.

Question 2.3 (Bonus)

Given a paragraph of English text, find the k words with highest frequency of occurrences and output the number of occurrences of the k words. You can choose appropriate data structures and explain why you choose them.

One test case:
Input:
The input file may contain the following paragraph:
"*Go back to Mississippi, go back to Alabama, go back to South Carolina, go back to Georgia, go back to Louisiana, go back to the slums and ghettos of our northern cities, knowing that somehow this situation can and will be changed. Let us not wallow in the valley of despair.*

I say to you today, my friends, so even though we face the difficulties of today and tomorrow, I still have a dream. It is a dream deeply rooted in the American dream."

Output:

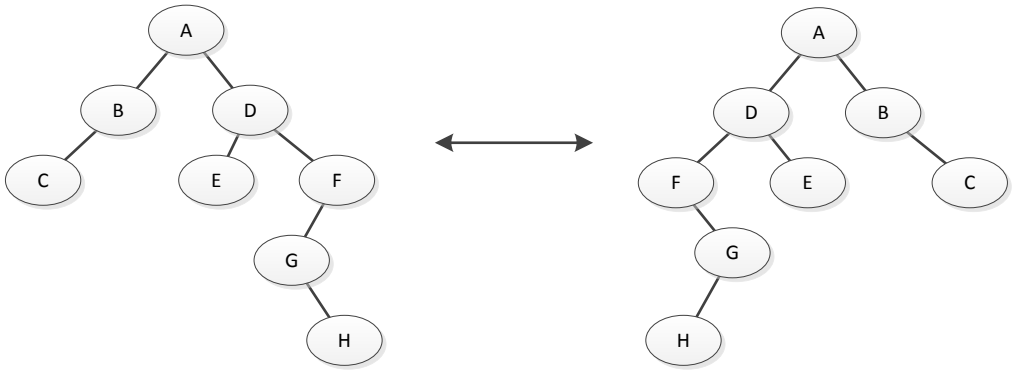
The output should be stored in a file, with each line containing a word and its frequency of occurrences in the following format ('_' means a space):

Word:_freq ,

Where *freq* equals the number of occurrences of the word divided by the total number of words in the text. You are required to find 5 words with the highest frequency and the frequencies are sorted in the descending order from the first line to the fifth line.

Question 2.4 (Bonus)

Build the mirror of a given binary tree and calculate the number of nodes that have 2 children. The following figure shows the concept of 'mirror'.



Test cases:

Input:

Given the preorder and inorder traversal of a binary tree, you should first rebuild that tree.

Preorder: *Y A L B E C D W X M*

Inorder: *B L E A C W X D Y M*

Output:

Print out the postorder traversal of the mirror of the original tree and the number of nodes with 2 children in the mirror.

Project 3: Search

Question 3.1 (Report required)

There are 3 bus lines in SJTU. Line A goes from Jingjingtang, Minhang Campus to Xuhui Campus; line B goes clockwise within Minhang Campus; line C goes counter clockwise within Minhang Campus. The bus schedule is known for each line (see **Appendixes A and B**).

In order to catch a bus to Xuhui we would like to take line B or C from a bus stop listed in Appendix A to Jingjingtang. For example, if we are now at the SEIEE building, it will take us 6 minutes with line B, or 15 minutes with line C, to get to Jingjingtang. For any given time and a specific bus station, determine the best plan of catching the next bus to Xuhui while minimizing the amount of time waiting at Jingjingtang.

Test case:

Input:

The input should follow the format ('_' means a space):

time_station ,

where *time* is in the format XX:XX representing the current time and *station* indicates the current bus station (see also **Appendix A**, e.g. *S4* means we are at *Canteen 4*).

Output:

The output should follow the format:

line:_t1_t2_t3 ,

where

line: the bus line you choose (B or C);

t1: the expected time that you can catch that bus (in format XX:XX);

t2: the expected time that you can catch the bus to Xuhui (in format XX:XX);

t3: the waiting time (minutes) at Jingjingtang (an integer).

For example, if the current time is 13:20 and we are at SEIEE, the best plan is to take a bus of line C at 13:45 and so as to get to Jingjingtang at 14:01 where we can catch the bus of Line A at 14:10. Then the input should be '*13:20 S12*' and the output should be '*C: 13:45 14:10 9*'

Question 3.2 (Report not needed)

You are given a paragraph written in English as input, consisting of a number of words.

(1) Sort these words in terms of Alphabetical order.

(2) Use a dynamic binary search tree to store the words and support search, insert and delete operations.

(3) Calculate the appearance frequency of each alphabet in this paragraph. Based on your calculation, use Huffman coding tree to design a Huffman coding scheme for this paragraph.

Test cases:

Input:

The input is a file containing the following paragraph:

"*Why, my dear, you must know, Mrs. Long says that Netherfield is taken by a young man of large fortune from the north of England;*"

Output:

The output should be a file, with each line keeps the following format ('_' means a space):

Alphabet:_freq_code..... ,

where Alphabet should be one of the letters from A-Z and the alphabets should increase alphabetically from first line to the last line. *freq* is the appearance frequency of the alphabet and *code* is the Huffman code of that alphabet.

Question 3.3 (Bonus)

a) Implement red-black tree and AVL tree, design experiments to compare the efficiency of different operations on them. You are required to compare insert, search and delete operations.

b) Add some functions to the implementation of binary search trees:

(1) delete nodes with key value less (larger) than a given number;
(2) delete all the nodes with key value between two given numbers;
(3) return depth of a given node in the BST.

One test case:

Input:

Using the following array to build a BST: {200, 250, 122, 77, 75, 80, 99, 70, 110, 120, 300}

Output:

a) Print out preorder and inorder traversal of the tree after deleting nodes from the original tree less than 100; print out preorder and inorder traversal of the tree after deleting nodes from the original tree larger than 200;
b) Print out preorder and inorder traversal of the tree after deleting nodes from the original tree between 100 and 200;
c) Print out the depth of 110 in the original tree.

Project 4: Graph

Question 4.1 (Report needed) Implement an Approximate Steiner Tree.

In this project, we consider n nodes located in a given space. Each node's location is determined by its horizontal position x and vertical position y. We assume that there is a link between two nodes if and only if their geographical distance (distance between node i and j is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$) is less than or equal to $\frac{\mathbf{r}}{\sqrt{\mathbf{n}}}$ (where r is a given positive real value). We guarantee that given any two nodes, we can find a path to connect them; in other words, this is a connected graph. Denote this graph as G. If there is a link between two nodes, then the weight of this link is equal to the geographical distance. Among all these nodes, we select a subset of them and denote the subset as V' ($V' \subseteq V$). We define a graph G' consisting of all nodes in V' as well as their links in between.

Given the above graph, we want to construct a tree spanning all nodes in V' with the minimal total weight. Different from the MST we learnt in class, we allow nodes in set V but not V' to be added in this spanning tree too, such that the total weight may be further reduced compared to MST. Such a tree is called Steiner tree in graph theory.

Steiner tree is important in wireless communication network designs. However, constructing a Steiner tree is NP-hard (interested students may refer to Chapter 17.2 of our textbook to see the concept of NP-Hard), and we consider a quick algorithm to construct an approximate Steiner tree. Let us take the following steps to complete this task.

1) Let us add a "virtual link" between two nodes in V' if there is no real link between them. The weight of the virtual link is set as the geographical distance between them. Denote the set of real and virtual links as E'. By doing this, you are requested to construct a fully connected graph H(V',E'). We construct a minimum spanning tree (MST) in H. Define this MST as T1.

2) Given two nodes, we define "hop" as the number of links in a path between two nodes, and define "minimum-hop shortest path" as a path with smallest total weight among all paths that have the minimum number of hops. In other words, for any given two nodes, if we first select all possible paths with the minimum number of hops, and then select one with the smallest total weight from them, then we have the minimum-hop shortest path.

   You are requested to remove the virtual links in T1 and then add the minimum-hop shortest paths in place of the virtual links between two nodes, so as to obtain a new graph T2. Note that during this process, we allow nodes in V-V' to be added to the tree. Later, we call the new nodes we add into T2 as relay nodes.

3) Let us assume there is a root node in graph T1 (you can randomly pick up any node you like as the root). Now you can view the graph T1 as a directed one. You may notice in T2,

if one relay node (which can be in V' or in V\V') is on multiple minimum-hop shortest paths, some node might be reached from the root via multiple paths. In other words, there might be redundant edges. In this step, you are required to eliminate all these redundant edges in T2 and output a new tree T3, such that the root has one and only one path to any other particular node in T2. See Figure 1 as an example. If there are multiple paths from the root to N2, we only need one path (root-N1-c-N2, or root-N3-c-N2). Your algorithm should successfully eliminate the edge N3-c or N1-c.

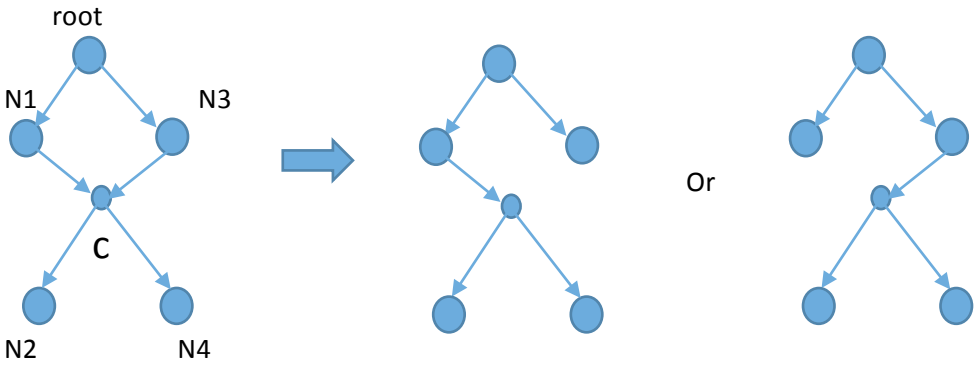


Figure 1. Redundancy elimination

Input:

We give 6 groups of inputs (*test1.txt – test5.txt* and *challenge.txt*). In each file, the first line is: **m, n, r,** where **m** is the number of nodes in V' and **n** is the number of nodes in V. The following lines are the coordinates (x, y) of nodes in [0,1] (each line corresponds to one particular node), where the first m lines are the coordinates of nodes in V'.

Output:

For each input file, print out the total number of hops, and the total length of minimum-hop shortest paths. For the first five test files, you are also required to print out the nodes you choose to construct the spanning tree in forms of (x, y, index), where x and y are the position coordinates of a node in the tree, and index is the index of its parent node (assume that the node in the n-th line is with index n-1).

Hints:

(1) You may add modifications to Dijkstra algorithm to find the minimum-hop shortest paths, where you need to compare a pair of attributes, hops and lengths, instead of a single attribute.

(2) You can find that m and n are large in some of the input files. Therefore some techniques are needed here to reduce the complexity of the algorithm. One bottleneck is to speed up your Dijkstra algorithm by some data structures like "binary heap" (you may want to refer to the link http://en.wikipedia.org/wiki/Dijkstra's_algorithm#Using_a_priority_queue, or directly search *Dijkstra binary heap*).

(3) Some properties of the graph: all points are *uniformly distributed* and the edges could be stored easily. During the search of shortest path, only a small part of vertices are involved since the source and destination are comparably close to each other. These properties could help you reduce the complexity of your design.

Question 4.2 (Report not needed)

Use a sequence of pairs of the form (A,B) to create a directed graph, where an edge from vertex A to vertex B is determined by the pair (A,B). Store the graph by an adjacency list and decide if the graph has an Euler circuit. (If you do not know what is Euler circuit, see here: http://en.wikipedia.org/wiki/Eulerian_path)

Test cases:

Input:

a) (A,B) (B,F) (E,A) (E,B) (D,E) (F,E) (C,D) (B,C)

b) (A,B) (F,B) (E,A) (E,B) (D,E) (F,E) (C,D) (B,C)

Output:

Determine if the graph has an Euler circuit in situation a) and b), respectively.

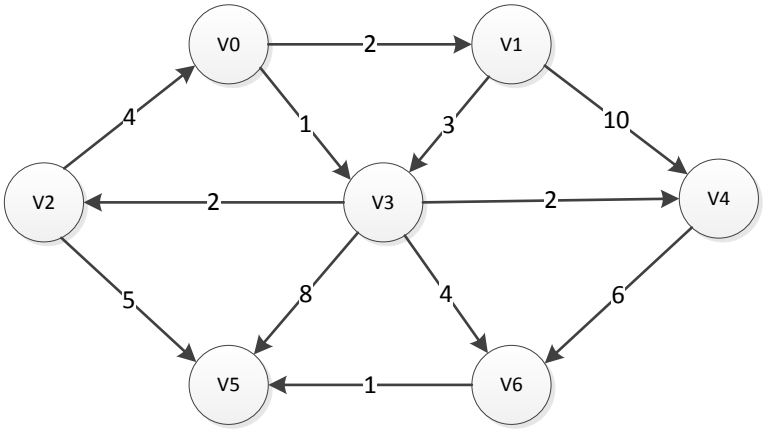
Question 4.3 (Bonus)

Implement an algorithm to calculate the shortest path in a directed graph.

Test cases:

Input:

You should firstly build a direct graph as the figure below shows:



The input should be *Vx_Vy*, where '_' means a space and *x, y* can be any of the two different numbers within 0 to 7.

Output:

You should print out the distance between the two nodes. Then, the shortest path from Vx to Vy should be printed in the following format:

Vx->Vy:_Vx-Va-Vb-…-Vc-Vy ,

Where '_' means a space and the whole path should be printed out.

For example, if the input is '*V1 V0*', the shortest distance equals 9 and the output should be '*V1->V0: V1-V3-V2-V0*'.

Appendix A Time to Square of Jingjingtang (min)

Station	*line B (clockwise)*	*line C (counter clockwise)*
S1: *Square of Jingjing auditorium*	_	_
S2: *School of Materials Science and Engineering*	20	2
S3: *Yue-Kong Pao library*	19	3
SJTU-Hualian shopping center	19	4
S4: *Canteen 4*	17	5
S5: *Dormitory (W35-W70)*	16	7
S6: *Student service center*	15	8
S7: *Wenxuan building of medicine*	12	10
S8: *School of Naval Architecture Ocean and Civil Engineering*	11	11
S9: *East main gate*	10	12
S10: *School of mechanical engineering*	9	13
S11: *South main gate*	8	14
S12: *School of Electronic information and Electrical Engineering*	6	15
S13: *Administration building B*	5	16
S14: *Library and information tower*	5	17
S15: *East middle hall*	3	18
S16: *East upper hall*	3	19
S17: *SJTU Minhang hospital*	1	21

Appendix B Bus schedule

Line A:

06:40 08:00 10:10 12:15 14:10 16:00 17:00 18:30 20:40

Line B:

1. Square of Jingjing auditorium
   08:30 08:50 09:10 09:30 10:00 10:30 11:00 11:30 12:30
   13:30 14:00 14:30 15:00 15:30 16:00 16:35
2. School of Materials Science and Engineering
   Base+1min
3. Yue-Kong Pao library
   Base+2min
4. SJTU-Hualian shopping center
   Base+2min
5. Canteen 4
   Base+4min
6. Dormitory (W35-W70)
   Base+5min
7. Student service center
   Base+6min
8. Wenxuan building of medicine
   Base+9min
9. School of Naval Architecture Ocean and Civil Engineering
   Base+10min
10. East main gate
    Base+11min
11. School of mechanical engineering
    Base+12min

12. South main gate
    Base+13min
13. School of Electronic information and Electrical Engineering
    Base+15min
14. Administration building B
    Base+16min
15. Library and information tower
    Base+16min
16. East middle hall
    Base+18min
17. East upper hall
    Base+18min

18. SJTU Minhang hospital
    Base+20min

Line C:

1. Square of Jingjing auditorium
   07:30 07:45 08:00 08:15 08:25 08:40 09:00 09:20 09:40 10:00
   10:20 10:40 11:00 11:20 11:40 12:00 13:00 13:20 13:40 14:00
   14:20 14:40 15:00 15:20 15:40 16:00 16:10 16:30 16:35 17:00
   17:30 18:15 19:00 20:00
2. School of Materials Science and Engineering
   Base+19min
3. Yue-Kong Pao library
   Base+18min
4. SJTU-Hualian shopping center
   Base+17min
5. Canteen 4
   Base+16min
6. Dormitory (W35-W70)
   Base+14min
7. Student service center
   Base+13min
8. Wenxuan building of medicine
   Base+11min
9. School of Naval Architecture Ocean and Civil Engineering
   Base+10min
10. East main gate
    Base+9min
11. School of mechanical engineering
    Base+8min
12. South main gate
    Base+7min
13. School of Electronic information and Electrical Engineering
    Base+6min
14. Administration building B
    Base+5min
15. Library and information tower
    Base+4min
16. East middle hall
    Base+3min
17. East upper hall
    Base+2min
18. SJTU Minhang hospital
    Base+0min