The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms

Final Exam Multiple Choice Questions

— *Additional Practice Solutions* —

INSTRUCTIONS: This document contains 120 multiple choice questions and 32 written questions to help you prepare for the final exam. The written questions begin on page 56 (starting with question 121). The multiple choice questions can be roughly broken down into the following categories:

- Questions 1-32: Hash Tables
- Questions 33-66: Trees
- Questions 67-82: Graphs
- Questions 83-110: Algorithm Families
- Questions 111-115: Shortest Path Algorithms
- Questions 116-120: Computational Geometry

The written problems are not organized by topic. See page 56 for more details.

---

1. Properties of Maps

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Which of the following statements is **FALSE**?

A) The `std::map` data structure is implemented using a binary search tree under the hood

B) The worst-case time complexity of searching a binary search tree occurs when the nodes are in a stick formation

C) Hash tables are extremely useful because searching for an element always takes $\Theta(1)$ time

D) The dictionary ADT can be implemented using a hash table

E) None of the above

2. Setting Things Up

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Which of the following statements is **FALSE**?

A) Unlike an `std::unordered_set`, an `std::unordered_map` has a value associated with each key

B) An `std::unordered_map` does not allow for duplicate keys

C) If you want to sort the keys of an `std::unordered_map`, you can pass its iterators into the `std::sort()` function

D) Using `operator[]` on a nonexistent key in an `std::unordered_map` causes the key to exist

E) None of the above

Record your answers in the bubbles next to each question.

3. Random Hashing

Ⓐ ●

True or false? The `rand()` function, which generates random numbers, is great for hashing since it greatly reduces the chances of collision.

A) True

B) False

4. Hashing Complexities

Ⓐ ●

True or false? Both unordered and ordered maps have $\Theta(1)$ average search and insert time complexities.

A) True

B) False

5. To Hash or Not to Hash

Ⓐ ●

True or false? A hash table is the best data structure to use if you want to keep track of the number of students who have birthdays on each of the 31 days in January.

A) True

B) False

6. Horrible Hashing

Ⓐ ● Ⓒ Ⓓ Ⓔ

Which of the following is not a characteristic of a good hash function?

A) The capability to distribute keys evenly in a hash table

B) The capability to keep similar keys close together in a hash table

C) The capability to compute a hash for every possible key

D) The capability to compute the same hash for the same key

E) All of the above are characteristics of a good hash function

Record your answers in the bubbles next to each question.

7. Compression by Modulo

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

An easy way to compress an integer key into a hash table of size M is to mod it by M (i.e., $key \mod M$). For which of the following collection of keys is this compression method the most ideal?

A) A list of all scores on a 20-question multiple choice exam, where each question is worth 5 points

B) A collection of "unluckiest numbers" collected in a survey of 500 Ann Arbor adults

C) The total number of minutes Dr. P spends with each student during office hours the day a project is due (rounded to the nearest minute)

D) The number of credits each full-time student at the university is taking this semester

E) The collection of student IDs of all students currently enrolled in EECS 281

8. Finding Sums

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Given two unsorted arrays of distinct numbers, you want to find all pairs of numbers, one from array 1 and one from array 2, that sum to a given value. For instance, if you are given

```
arr1[] = {1, 2, 3, 4, 5, 7, 11}
arr2[] = {2, 3, 4, 5, 6, 8, 12}
```

you would return `(1, 8)`, `(3, 6)`, `(4, 5)`, `(5, 4)`, and `(7, 2)`. What is the average time complexity of doing this if you use the most efficient algorithm?

A) $\Theta(1)$

B) $\Theta(\log(n))$

C) $\Theta(n)$

D) $\Theta(n \log(n))$

E) $\Theta(n^2)$

Record your answers in the bubbles next to each question.

9. Symmetric Pairs

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Two pairs (a, b) and (c, d) are symmetric if b = c and a = d. Suppose you are given an array of pairs, and you want to find all symmetric pairs in the array. The first element of all pairs is distinct. For instance, if you are given the following array

```
arr1[] = {(14, 23), (11, 2), (52, 83), (49, 38), (38, 49), (2, 11)}
```

you would return `{(11, 2), (2, 11)}` and `{(49, 38), (38, 49)}`. What is the average time complexity of doing this if you use the most efficient algorithm?

A) $\Theta(1)$

B) $\Theta(\log(n))$

C) $\Theta(n)$

D) $\Theta(n\log(n))$

E) $\Theta(n^2)$

10. Minimum Deletions

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

You are given an array of n elements where elements may be repeated, and you want to find the minimum number of elements that need to be deleted from the array so that all elements in the array are equal. For instance, if you are given the following array

```
arr1[] = {3, 6, 8, 6, 2, 7, 6, 3, 1, 3, 6}
```

you would return 7, as this is the minimum number of deletions required to obtain an array where all elements are the same (in this case, you would get an array with all 6's). What is the average time complexity of doing this if you use the most efficient algorithm?

A) $\Theta(1)$

B) $\Theta(\log(n))$

C) $\Theta(n)$

D) $\Theta(n\log(n))$

E) $\Theta(n^2)$

Record your answers in the bubbles next to each question.

Use the code below to answer questions 11-13.

```
1  #include <iostream>
2  #include <string>
3  #include <utility>
4  #include <unordered_map>
5  using namespace std;
6
7  int main() {
8     unordered_map<string, string> myMap;
9     myMap.insert(make_pair("Paoletti","Darden"));
10    myMap.insert(make_pair("Angstadt","Darden"));
11    myMap.insert(make_pair("Paoletti","Angstadt"));
12    myMap["Angstadt"] = "Paoletti";
13    myMap.insert(make_pair("Paoletti", "Garcia"));
14    cout << myMap["Paoletti"] << endl;
15    cout << myMap["Darden"] << endl;
16    myMap.erase("Paoletti");
17    cout << myMap["Angstadt"] << endl;
18    cout << myMap.size() << endl;
19 } // main()
```

11. Keeping Track of Professors, Part I Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

What does line 14 print?

A) `Paoletti`

B) `Darden`

C) `Angstadt`

D) `Garcia`

E) An empty string

12. Keeping Track of Professors, Part II **Ⓐ** Ⓑ Ⓒ Ⓓ Ⓔ

What does line 17 print?

A) `Paoletti`

B) `Darden`

C) `Angstadt`

D) `Garcia`

E) An empty string

Record your answers in the bubbles next to each question.

Use the code below to answer questions 11-13.

```
1  #include <iostream>
2  #include <string>
3  #include <utility>
4  #include <unordered_map>
5  using namespace std;
6
7  int main() {
8     unordered_map<string, string> myMap;
9     myMap.insert(make_pair("Paoletti","Darden"));
10    myMap.insert(make_pair("Angstadt","Darden"));
11    myMap.insert(make_pair("Paoletti","Angstadt"));
12    myMap["Angstadt"] = "Paoletti";
13    myMap.insert(make_pair("Paoletti", "Garcia"));
14    cout << myMap["Paoletti"] << endl;
15    cout << myMap["Darden"] << endl;
16    myMap.erase("Paoletti");
17    cout << myMap["Angstadt"] << endl;
18    cout << myMap.size() << endl;
19 }
```

13. Keeping Track of Professors, Part III

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

What does line 18 print?

A) 1

B) 2

C) 3

D) 4

E) 7

Record your answers in the bubbles next to each question.

14. High Score, Part I

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

The Project 4 autograder just got released! On the first day, eight students submitted. The scores that each of these students received on their first submission are shown in the table below.

Student ID	522	883	768	417	130	614	349	265
Score	34.9	56.7	21.4	75.4	61.1	35.7	23.1	45.7

Suppose these students are inserted into a `std::map<int, double>` named `P4Scores`, where *Student ID* represents the key and *Score* represents the value. If `it = P4Scores.end()`, what is the value of `(--it)->first`?

A) 130

B) 265

C) 417

D) 768

E) None of the above

15. High Score, Part II

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Suppose you wanted to print out the student ID with the highest score, 417. Which of the following would successfully accomplish this? Select all that apply.

A) Inserting student 417 **first** into a `std::unordered_map<double, int>` named `P4Scores` with *Score* as the key and *Student ID* as the value, setting an iterator `it` to `P4Scores.begin()`, and printing `it->second`

B) Inserting student 417 **last** into a `std::unordered_map<double, int>` named `P4Scores` with *Score* as the key and *Student ID* as the value, setting an iterator `it` to `P4Scores.end()`, and printing `(--it)->second`

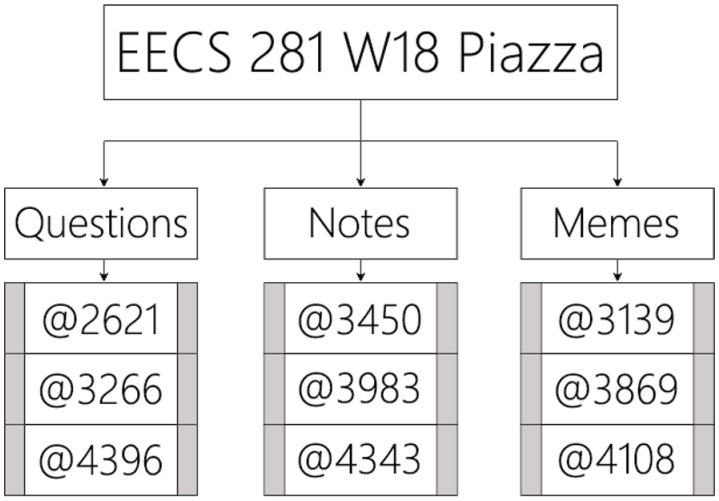
C) Inserting student 417 **last** into a `std::map<double, int>` named `P4Scores` with *Score* as the key and *Student ID* as the value, setting an iterator `it` to `P4Scores.begin()`, and printing `it->second`

D) Inserting student 417 **first** into a `std::map<double, int>` named `P4Scores` with *Score* as the key and *Student ID* as the value, setting an iterator `it` to `P4Scores.end()`, and printing `(--it)->second`

E) Inserting student 417 **last** into a `std::map<double, int>` named `P4Scores` with *Score* as the key and *Student ID* as the value, setting an iterator `it` to `P4Scores.end()`, and printing `it->second`

Record your answers in the bubbles next to each question.

On the Winter 2018 EECS 281 Piazza page, posts can be categorized into three unique groups: questions, notes, and memes. A representation of this is shown below:



16. Mapping Piazza, Part I

(A) (B)

True or false? An `enum` class can be used to track whether a Piazza post is a question, a note, or a meme.

A) True

B) False

17. Mapping Piazza, Part II

(A) (B) (C) (D) (E)

Suppose you implemented the following unordered map that maps the type of each post to the post IDs that correspond to each of these types (for instance, the key `notes` maps to `[3450,3983,4343]`):

```
std::unordered_map<std::string, std::vector<int>> posts;
```

While searching through Piazza, you discover that post @4753 is a meme. Which of the following successfully appends 4753 to the vector associated with the key `memes`?

A) `posts["memes"] = 4753;`

B) `posts["memes"] = memes.push_back(4753);`

C) `posts["memes"] = posts.push_back(4753);`

D) `posts["memes"].push_back(4753);`

E) `posts["memes"].second.push_back(4753);`

Record your answers in the bubbles next to each question.

18. Mapping Piazza, Part III

Ⓐ Ⓑ

True or false? Running the following lines of code would give you an error.

```
1  posts.insert("polls");
2  std::cout << posts["polls"] << std::endl;
```

A) True

B) False

For questions 19-22, consider the following code:

```
1   #include <iostream>
2   #include <string>
3   #include <vector>
4   #include <utility>
5   #include <map>
6   #include <unordered_map>
7   using namespace std;
8
9   int main() {
10     unordered_map<string, pair<string, int>> foods;
11     foods["cabbage"] = make_pair("vegetable", 1);
12     foods["banana"] = make_pair("fruit", 2);
13     foods["donut"] = make_pair("dessert", 3);
14     foods["apple"] = make_pair("fruit", 4);
15     foods["eggplant"] = make_pair("vegetable", 5);
16     vector<string> vec;
17     for (auto x : foods)
18        vec.push_back(x.first);
19     cout << vec.front() << endl;
20     return 0;
21  }
```

19. Categorizing Food, Part I

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

What is the type of `x` on line 17?

A) `string`

B) `pair<string, int>`

C) `pair<string, string>`

D) `pair<string, pair<string, int>>`

E) `pair<pair<string, int>, string>`

Record your answers in the bubbles next to each question.

20. Categorizing Food, Part II Ⓐ Ⓑ Ⓒ Ⓓ ● Ⓔ

What does line 19 print?

A) `apple`

B) `fruit`

C) `cabbage`

D) `vegetable`

E) Impossible to determine

21. Categorizing Food, Part III ● Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Suppose line 10 were replaced with the following line:

```
map<string, pair<string, int>> foods;
```

What does line 19 print now?

A) `apple`

B) `fruit`

C) `cabbage`

D) `vegetable`

E) Impossible to determine

22. Categorizing Food, Part IV Ⓐ Ⓑ ● Ⓒ Ⓓ Ⓔ

Which of the following lines of code prints out `1`? Use the original code *without* the replacement made in question 21.

A) `cout << foods[0] << endl;`

B) `cout << foods["cabbage"] << endl;`

C) `cout << foods["cabbage"].second << endl;`

D) `cout << foods["cabbage"].second.second << endl;`

E) None of the above

Record your answers in the bubbles next to each question.

23. Open Addressing (A) (B) (C) (D) (E)

Which of the following collision resolution methods is not a form of open addressing?

A) Separate Chaining

B) Linear Probing

C) Quadratic Probing

D) Double Hashing

E) All of the above use open addressing

24. Maintaining the Load (A) (B) (C) (D) (E)

Which of the following statements is **FALSE**?

A) The load factor α of a hash table can exceed 1

B) As α increases, the performance of separate chaining does not deteriorate as quickly as the performances of open addressing methods

C) The time complexities of searching and removing are both $\Theta(\alpha)$ for a hash table that uses separate chaining to resolve collisions

D) If α is less than 0.5, linear probing is better than double hashing at preventing keys in a hash table from clustering together

E) More than one of the above

25. Problematic Quadratics (A) (B) (C) (D) (E)

Which of the following is **NOT** a disadvantage of using quadratic probing to resolve collisions?

A) Two elements that hash to the same position will still have the same probe sequence, regardless of how far they land from their hashed location

B) Quadratic probing may insert keys into positions of a hash table that are far from the index that they normally hash to

C) Depending on the size of the hash table involved, it is possible for quadratic probing to never consider specific indices while searching for an open position

D) The performance of quadratic probing can deteriorate dramatically as the load factor increases

E) None of the above

Record your answers in the bubbles next to each question.

26. Load Factor

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

A hash table of size 100 has 40 empty elements and 25 deleted elements. What is its load factor?

A) 0.25

B) 0.35

C) 0.60

D) 0.65

E) 0.75

27. Linear Probing

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Suppose you have a hash table of size $M = 10$ that uses the hash function $H(n) = 3n + 7$ and the compression function $C(n) = n \bmod M$. **Linear probing** is used to resolve collisions. You enter the following nine elements into this hash table in the following order: `{10, 8, 18, 17, 4, 20, 6, 3, 16}`. No resizing is done. After all collisions are resolved, which index of the hash table remains empty?

A) 3

B) 4

C) 5

D) 6

E) None of the above

28. Quadratic Probing

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Suppose you have a hash table of size $M = 7$ that uses the hash function $H(n) = n$ and the compression function $C(n) = n \bmod M$. **Quadratic probing** is used to resolve collisions. You enter the following six elements into this hash table in the following order: `{24, 11, 17, 21, 10, 4}`. No resizing is done. After all collisions are resolved, which index of the hash table remains empty?

A) 1

B) 2

C) 5

D) 6

E) None of the above

Record your answers in the bubbles next to each question.

29. Tracking the Hash

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

A hash table of size 10 uses open addressing with a hash function $H(k) = k$, compression function $C(k) = k \bmod 10$, and linear probing. After entering six values into an empty hash table, the state of the table is as shown below. Which of the following insertion orders is possible?

0	1	2	3	4	5	6	7	8	9
		62	43	24	82	76	53		

A) 76, 62, 24, 82, 43, 53

B) 24, 62, 43, 82, 53, 76

C) 76, 24, 62, 43, 82, 53

D) 62, 76, 53, 43, 24, 82

E) None of the above

30. Who's Teaching Class?

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Suppose you are using a hash function where each string is hashed to an integer representing its first letter. The integer each letter hashes to is shown in the table below:

a	b	c	d	e	f	g	h	i	j	k	l	m	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13

o	p	q	r	s	t	u	v	w	x	y	z
14	15	16	17	18	19	20	21	22	23	24	25

The compression function $C(x) = x \bmod M$ is used, where M represents the size of the hash table.

You are using a hash table of size M. If the strings "paoletti" and "darden" collide in this hash table, which of the following is not a possible value of M?

A) 3

B) 6

C) 9

D) 12

E) None of the above

Record your answers in the bubbles next to each question.

31. When Planets Collide

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Suppose you are using a hash function where each string is hashed to an integer representing its first letter. The integer each letter hashes to is shown in the table below:

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**
0	1	2	3	4	5	6	7	8	9	10	11	12	13

o	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**
14	15	16	17	18	19	20	21	22	23	24	25

You have a hash table of size 10, and you insert the planets of the solar system (with the sun) into this hash table in the following order:

1. "sun"
2. "mercury"
3. "venus"
4. "earth"
5. "mars"
6. "jupiter"
7. "saturn"
8. "uranus"
9. "neptune"

Collisions are resolved using the **double hashing** formula

$$t(key) + i \times (q - (t(key) \bmod q))$$

where $t(key)$ is the integer that a key hashes to, i is the number of collisions that have occurred so far with that key, and q is 7. Compression is done using the formula $C(n) = n \bmod M$, where M is 10 (the size of the hash table). No resizing is done. After all collisions are resolved, which index of the hash table remains empty?

A) 2

B) 5

C) 6

D) 7

E) None of the above

Record your answers in the bubbles next to each question.

32. Grocery Shopping

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Suppose you have a hash table of size $M = 13$ that uses the same hash function as mentioned previously. You enter the following foods as keys into this hash table in the following order:

1. "apples"
2. "almonds"
3. "avocados"
4. "asparagus"

Collisions are resolved using the **double hashing** formula:

$$t(key) + i \times (q - (t(key) \bmod q))$$

After all collisions are resolved, you notice that the key "asparagus" ended up at index 7. Knowing this, what is a possible value of q?

A) 3

B) 5

C) 7

D) 9

E) 11

33. Binary Tree Arrays

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

What is the worst-case *memory* complexity of implementing a binary tree using an array?

A) $\Theta(\log(n))$

B) $\Theta(n)$

C) $\Theta(n \log(n))$

D) $\Theta(n^2)$

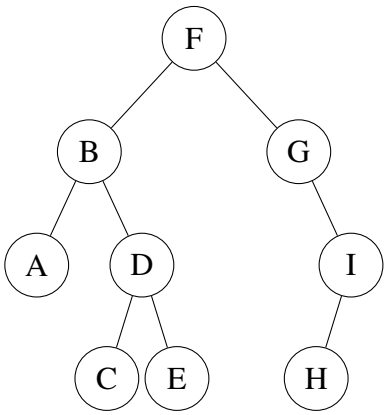
E) $\Theta(2^n)$

Record your answers in the bubbles next to each question.

34. Tree Terminology

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Consider the following tree below:



Which of the following nodes is **NOT** an internal node?

A) Node B

B) Node D

C) Node F

D) Node I

E) All of the above are internal nodes

35. Wasted Space

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

If you used an array to represent a binary tree, where the root is at index 1 and the left and right children of the node at index n are $2n$ and $2n + 1$ respectively, which of the following indices would be filled in the case of a rightward-facing stick (i.e., a stick where nodes only have right children)?

A) 2

B) 17

C) 31

D) 49

E) 64

Record your answers in the bubbles next to each question.

36. Binary Search Sticks

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Suppose that you want to insert 12 distinct elements into a binary search tree. How many worst-case trees are possible for these 12 elements?

A) 2

B) 2,047 $(2^{11} - 1)$

C) 2,048 (2^{11})

D) 4,095 $(2^{12} - 1)$

E) 4,096 (2^{12})

37. Worst-Case Trees

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

For which of the following element insertion orders is the resulting binary search tree a worst-case tree?

A) 51, 13, 38, 41, 49, 46, 47, 22, 53, 8

B) 53, 8, 51, 13, 22, 49, 47, 38, 41, 46

C) 51, 53, 8, 46, 49, 38, 41, 13, 47, 22

D) 8, 13, 46, 41, 47, 53, 51, 38, 22, 49

E) None of the above

38. Barking Up the Wrong Tree

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following statements is **TRUE**?

A) The inorder successor of the root node can have two children

B) The array-based binary tree implementation is most efficient for trees that have fewer nodes

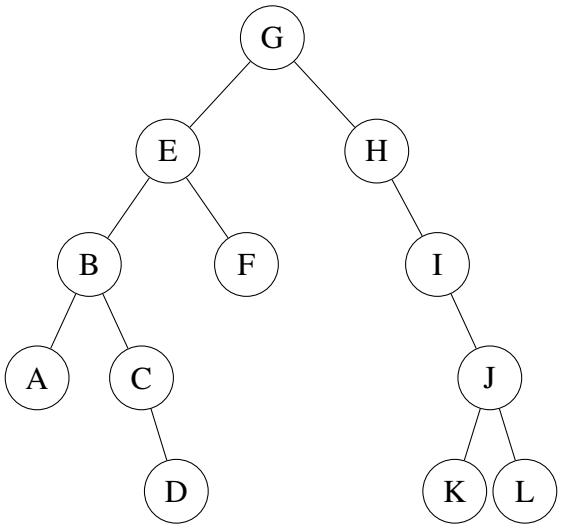
C) The worst-case time complexity of inserting an element into a pointer-based binary tree is better than the worst-case time complexity of inserting an element into an array-based one

D) For a pointer-based implementation of a binary tree with n nodes, the best-case auxiliary space complexity is equal to the worst-case auxiliary space complexity

E) More than one of the above

Record your answers in the bubbles next to each question.

For questions 39-42, consider the tree below:



39. The Tree Family, Part I (A) (B) (C) (D) **(E)**

Node _____ is the inorder predecessor of node G, and node _____ is the inorder successor of node G.

A) D; K

B) E; H

C) F; K

D) A; L

E) F; H

40. The Tree Family, Part II **(A)** (B) (C) (D) (E)

Which of the following statements about the tree is **FALSE**?

A) Node B is a sibling of node I

B) Node K is a descendant of node H

C) Node L is a child of node J

D) Node E is an ancestor of node C

E) None of the above

Record your answers in the bubbles next to each question.

41. The Tree Family, Part III

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following statements about the tree is **TRUE**?

A) The height of node F is one greater than the height of node C

B) The height of node B is one greater than the height of node A

C) The height of node F is equal to the height of node B

D) The height of node I is equal to the height of node B

E) More than one of the above

42. The Tree Family, Part IV

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following is the correct postorder traversal of this tree?

A) G, E, B, A, C, D, F, H, I, J, K, L

B) D, A, C, B, F, E, K, L, J, I, H, G

C) A, D, C, B, F, E, G, K, L, J, I, H

D) A, D, C, B, F, E, K, L, J, I, H, G

E) None of the above

43. Tree Recursion

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

You are given the following snippet of code:

```
1  int question_43(Node *root) {
2     if (!root)
3        return 0;
4     else if (!root->left && !root->right)
5        return 0;
6     else
7        return 1 + question_43(root->left) + question_43(root->right);
8  }
```

What does this function do if `root` is the root of a binary tree?

A) It returns the number of leaf nodes

B) It returns the number of internal nodes

C) It returns the number of nodes in the tree

D) It returns the height of the tree

E) It returns the depth of the tree

Record your answers in the bubbles next to each question.

44. Tree Treeversal, Part I

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Suppose you are given the preorder and inorder traversals of a binary tree:

Preorder: 12, 6, 9, 18, 15, 13, 11
Inorder: 6, 18, 9, 12, 13, 15, 11

What is the postorder traversal of this tree?

A) 9, 18, 6, 11, 13, 15, 12

B) 9, 18, 6, 13, 11, 15, 12

C) 18, 9, 6, 12, 13, 11, 15

D) 18, 9, 6, 13, 11, 15, 12

E) None of the above

45. Tree Treeversal, Part II

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Suppose you are given the postorder and inorder traversals of a binary tree:

Postorder: 14, 22, 43, 18, 23, 35, 12, 27, 16
Inorder: 22, 14, 23, 43, 18, 16, 12, 35, 27

What is the preorder traversal of this tree?

A) 14, 43, 22, 18, 23, 16, 27, 35, 12

B) 14, 43, 22, 18, 35, 23, 12, 27, 16

C) 16, 23, 22, 14, 18, 43, 27, 12, 35

D) 16, 23, 22, 18, 14, 43, 27, 12, 35

E) None of the above

46. Not Enough Information?

Ⓐ Ⓑ

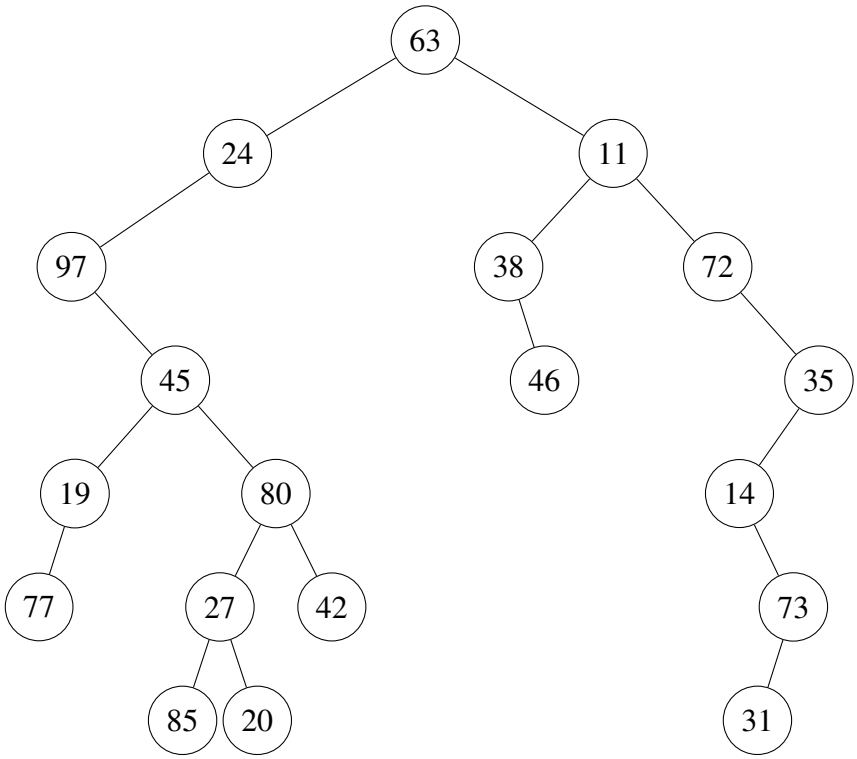
True or false? Only knowing the preorder and postorder traversals of a binary tree is **NOT** enough to uniquely identify that binary tree.

A) True

B) False

Record your answers in the bubbles next to each question.

For questions 49-51, consider the following tree:



47. Insanitree, Part I (A) (B)

True or false? This tree is a dense graph.

A) True

B) False

48. Insanitree, Part II (A) (B) (C) (D) (E)

The inorder predecessor of the root node has a value of a, and the inorder successor of the root node has a value of b. What is the value of $a + b$?

A) 35

B) 62

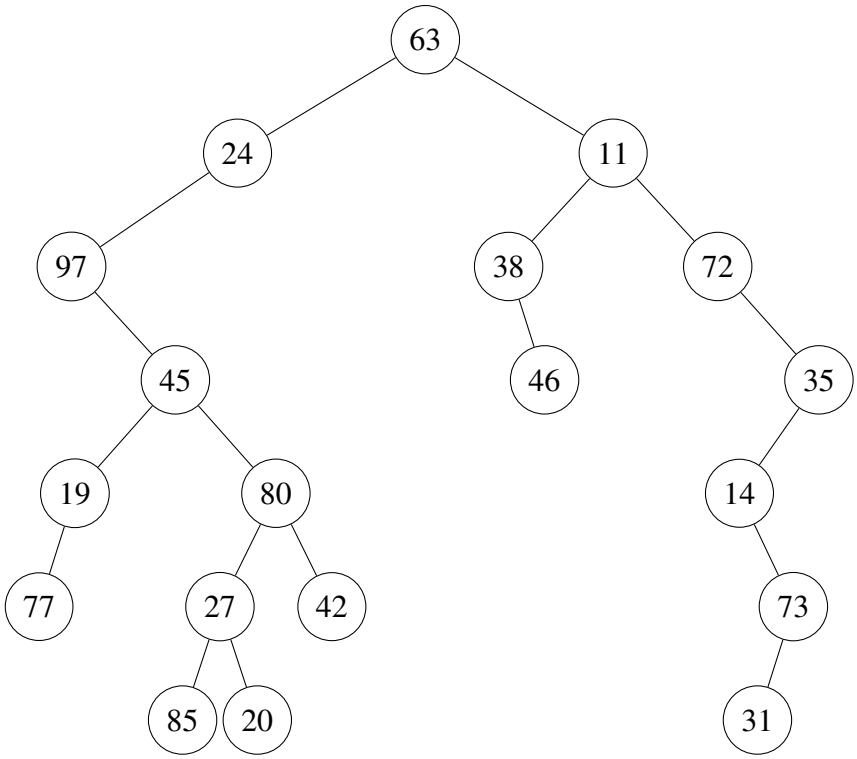
C) 70

D) 80

E) None of the above

Record your answers in the bubbles next to each question.

For questions 47-49, consider the following tree:



49. Insanitree, Part III (A) (B) (C) (D) (E)

Suppose the following statements are true about this tree:

1. The 17th element of a preorder traversal has a value of v.
2. The 17th element of an inorder traversal has a value of w.
3. The 3rd element of an postorder traversal has a value of x.
4. The 13th element of an postorder traversal has a value of y.
5. The 15th element of a level-order traversal has a value of z.

What is the value of $v + w + x + y + z$? *Note: this is just for practice — we will never ask you a question this involved on the actual exam.*

A) 203

B) 280

C) 287

D) 364

E) None of the above

Record your answers in the bubbles next to each question.

50. Take a Breadth

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following traversal methods conducts a breadth-first search?

A) Preorder traversal

B) Postorder traversal

C) Inorder traversal

D) Level-order traversal

E) None of the above

51. Mystery Sum

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Consider a tree that satisfies the following conditions:

1. The tree is a binary search tree of integers.
2. The number of elements in the root node's left and right subtrees are the same.
3. There are no duplicate values in the tree.
4. The first element of an inorder traversal of the tree is 11.
5. The last element of an inorder traversal of the tree is 24.
6. The last element of a postorder traversal of the tree is 16.

What is the largest possible integer you can attain by summing up all the values in a tree that satisfies the above constraints?

A) 171

B) 176

C) 191

D) 210

E) None of the above

52. Post Your Order

Ⓐ **Ⓑ**

True or false? If you are given a binary search tree of integers, the first number in a postorder traversal of the tree must also be the smallest element in the tree.

A) True

B) False

Record your answers in the bubbles next to each question.

53. A Reverse Traversal

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

For a certain binary tree, its inorder traversal is the exact reverse of its postorder traversal. What can you infer about this binary tree?

A) It is a leftward-facing stick (i.e., a stick where nodes only have left children)

B) It is a rightward-facing stick (i.e., a stick where nodes only have right children)

C) Its preorder traversal is the exact reverse of its inorder traversal

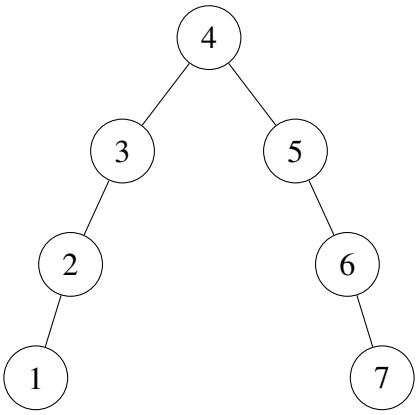
D) Its preorder traversal is the same as its postorder traversal

E) More than one of the above

54. I'm Balanced or Imbalanced?

Ⓐ **Ⓑ**

True or false? The following tree is balanced.



A) True

B) False

55. Tricky Rotations

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Which of the following statements about AVL trees is **TRUE**?

A) To balance any tree with a node that has a negative balance factor, one should conduct a single rightward rotation about this node

B) To balance any tree with a node that has a negative balance factor, one should conduct a single leftward rotation about this node

C) A tree with more elements on its right side than its left side will have a positive balance factor

D) Both A and B

E) None of the above

Record your answers in the bubbles next to each question.

56. AVL Sort Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

You are given an array of unsorted elements, and you want to sort these elements and print them out in sorted order. What is the worst-case time complexity of doing this if you use an AVL tree?

A) $\Theta(\log(n))$

B) $\Theta(n)$

C) $\Theta(n \log(n))$

D) $\Theta(n^2)$

E) $\Theta(n^2 \log(n))$

57. AVL Search Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

What is the worst-case time complexity of searching for an element in an AVL tree with $n^4 3^n$ elements?

A) $\Theta(\log(n))$

B) $\Theta(n)$

C) $\Theta(n \log(n))$

D) $\Theta(n^4)$

E) $\Theta(3^n)$

58. Triple Rotations Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

For which of the following operations is a triple rotation possible?

A) Inserting an element into an AVL tree containing 31 nodes

B) Inserting an element into an AVL tree containing 32 nodes

C) Inserting an element into an AVL tree containing 33 nodes

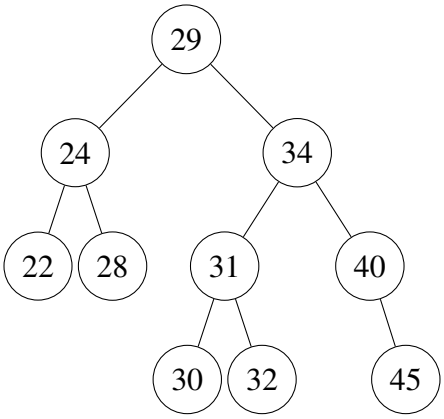
D) Both A and B

E) None of the above

Record your answers in the bubbles next to each question.

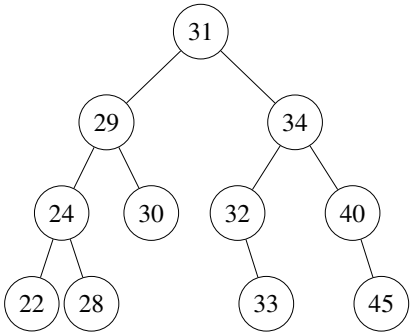
59. Next Level Rotation (A) B C D E

Consider the following AVL tree:

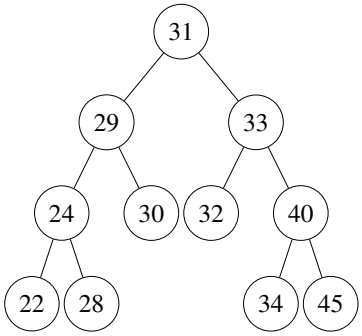


What does the AVL tree look like after 33 is inserted and all rotations are completed?

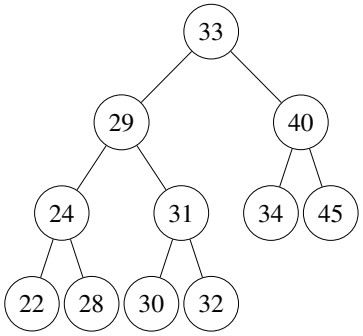
A)



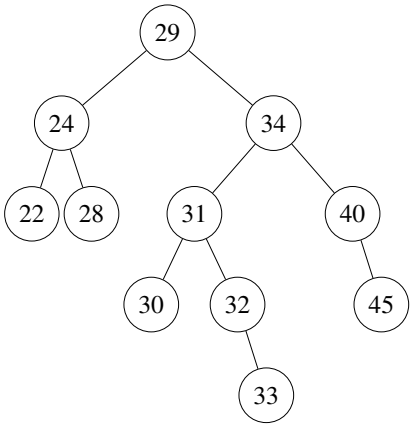
C)



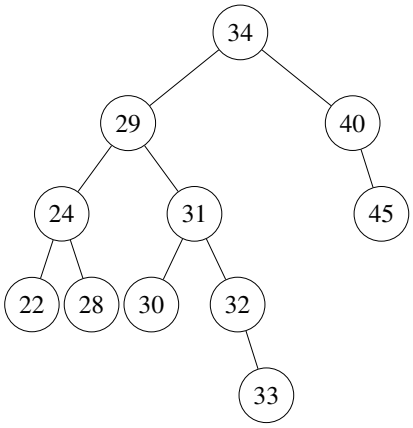
E)



B)



D)



Record your answers in the bubbles next to each question.

60. AVL Insertion (A) (B) **(C)** (D) (E)

Insert the following elements into an empty AVL tree, rebalancing when necessary.

23, 26, 24, 25, 29, 11, 13, 9, 8, 16, 17

After all the elements are inserted, what is the level-order traversal of this tree?

A) 23, 11, 24, 8, 16, 26, 9, 13, 17, 25, 29

B) 23, 11, 25, 8, 16, 24, 29, 9, 13, 17, 26

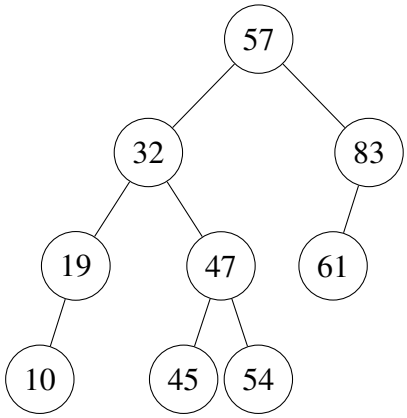
C) 24, 13, 26, 9, 17, 25, 29, 8, 11, 16, 23

D) 24, 13, 26, 17, 9, 25, 29, 16, 23, 8, 11

E) None of the above

61. AVL Deletion, Part I **(A)** (B) (C) (D) (E)

Consider the following AVL tree:



Suppose you deleted 57 from the AVL tree above and rebalanced the tree by replacing the root with the **inorder successor**. What is the *level-order* traversal of the resulting tree?

A) 32, 19, 61, 10, 47, 83, 45, 54

B) 47, 32, 61, 19, 45, 54, 83, 10

C) 54, 32, 83, 19, 47, 61, 10, 45

D) 61, 32, 83, 19, 47, 10, 45, 54

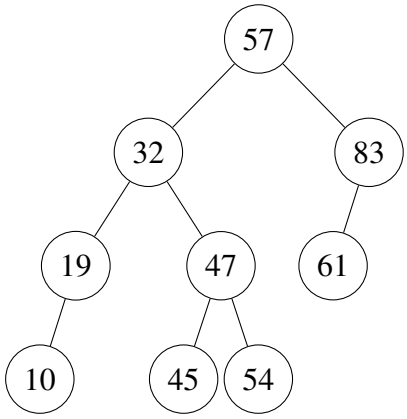
E) None of the above

Record your answers in the bubbles next to each question.

62. AVL Deletion, Part II

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Consider the following AVL tree:



Suppose you deleted 57 from the AVL tree above and rebalanced the tree by replacing the root with the **inorder predecessor**. What is the *postorder* traversal of the resulting tree?

A) 10, 19, 32, 45, 47, 54, 61, 83

B) 10, 19, 45, 47, 32, 61, 83, 54

C) 10, 45, 19, 47, 61, 32, 83, 54

D) 45, 54, 10, 47, 83, 19, 61, 32

E) None of the above

63. Min and Max

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Suppose you have a tree with a height of 5, where the leaf nodes have height 1. Let a represent the maximum number of nodes in a balanced AVL tree of height 5, and let b represent the minimum number of nodes in a balanced AVL tree of height 5. What is the value of $a - b$?

A) 12

B) 19

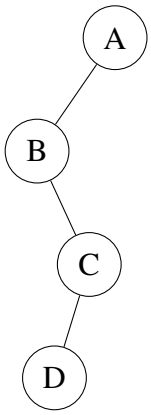
C) 20

D) 31

E) 43

Record your answers in the bubbles next to each question.

For questions 64-65, consider the following stick:



You are told to balance this stick. To do this, start from the bottom node and move upwards toward the root, calculate the balance factor, and rotate whenever necessary.

64. Balancing a Stick, Part I (A) (B) (C) (D) (E)

After the stick is balanced, which node becomes the root node?

A) A

B) B

C) C

D) D

E) Impossible to tell

65. Balancing a Stick, Part II (A) (B) (C) (D) (E)

After the stick is balanced, what is the balance factor of the root node?

A) -2

B) -1

C) 0

D) 1

E) 2

Record your answers in the bubbles next to each question.

66. Build and Balance

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

You are told that a certain AVL tree has the following postorder traversal:

8, 11, 21, 14, 30, 28, 35, 37, 34, 26

Suppose you add the values of 9 and 29 to this AVL tree, in this order, and balance the tree accordingly. What is the *preorder* traversal of the resulting tree?

A) 26, 14, 8, 9, 11, 21, 34, 28, 30, 29, 37, 35

B) 26, 14, 9, 8, 11, 21, 34, 29, 28, 30, 37, 35

C) 26, 14, 34, 8, 21, 28, 37, 11, 30, 35, 9, 29

D) 26, 14, 34, 9, 21, 29, 34, 8, 11, 28, 30, 35

E) Not enough information is provided to answer this question

67. Directed Graphs

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following graphs is directed?

A) A graph where the vertices represent people at a business meeting, and the edges represent handshakes that have occurred between two people

B) A graph where the vertices represent all intersections in New York City, and the edges represent all two-way streets that pass through these intersections

C) A graph where the vertices represent all students at the University of Michigan, and the edges represent whether a student shares a class with another student

D) A graph where the vertices represent all students and staff in EECS 281, and the edges represent whether someone has gotten help from someone else during office hours

E) A graph where the vertices represent all EECS students, and the edges represent whether a student has ever collaborated with another student on an EECS assignment

68. Equal Weights

Ⓐ **Ⓑ**

True or false? For a weighted, undirected graph, the cost of going from point A to point B will always be the same regardless of how you get there.

A) True

B) False

Record your answers in the bubbles next to each question.

69. Graph Implementations

(A) (B) (C) (D) (E)

Suppose you have a graph with 100 edges and 100 vertices. Is the graph sparse or dense, and should you represent this graph using an adjacency list or an adjacency matrix?

A) This graph is sparse, and an adjacency list should be used

B) This graph is sparse, and an adjacency matrix should be used

C) This graph is dense, and an adjacency list should be used

D) This graph is dense, and an adjacency matrix should be used

E) None of the above

70. Memory Trees

(A) (B) (C) (D) (E)

Suppose you have a binary tree of height 281, where the leaf nodes have height 1, and you want to search for an element k. You know that k exists as a distinct element in this tree, and that it is a leaf node. Which of the following statements is **FALSE**?

A) If you conduct a depth-first search, you'll never have to store more than 281 nodes in memory

B) Conducting a breadth-first search would require much more memory than a depth-first search

C) Using a stack to implement this search is preferable to using a queue

D) The path from the root to element k returned by BFS is shorter than the path returned by DFS

E) None of the above

71. Seeing the Forest for the Trees

(A) (B) (C) (D) (E)

You are doing a *breadth*-first search on a binary tree with depth 281. You are using a deque to conduct this search, and you want to find an element k. You don't know where this element k is, but you do know that k exists as a unique value in this tree. Once k is first encountered, the search terminates immediately. You start the search by pushing the root into the deque. After running the BFS for a while, you realize that an element at a depth of 121 was popped out of your deque. Knowing this information, which of the following statements is *definitely* **FALSE**?

A) The element k can be found at a depth of 120

B) The element k can be found at a depth of 121

C) The element k can be found at a depth of 122

D) Both A and B

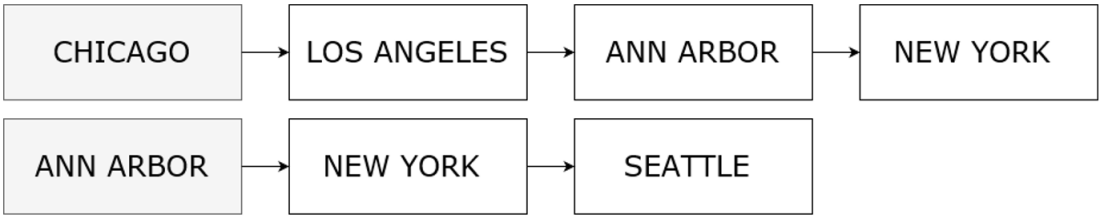
E) Both B and C

Record your answers in the bubbles next to each question.

72. Connecting Cities

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Consider the following adjacency list:



Which of the following statements is **FALSE**?

A) This adjacency list represents a directed graph

B) Ann Arbor has a direct connection with New York

C) New York has a direct connection with Seattle

D) Chicago has a direct connection with New York

E) More than one of the above

73. Finding Your Flight, Part I

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

You currently have a graph that represents several airports (vertices) and the flights that connect each of them (edges), weighted by the distance of each flight. You decide to implement this graph using an adjacency list. The number of vertices is represented by V, and the number of edges is represented by E.

Given an airport X, what is the average-case time complexity of finding the closest airport to airport X?

A) $\Theta(1)$

B) $\Theta(E)$

C) $\Theta(V)$

D) $\Theta(1 + \frac{E}{V})$

E) $\Theta(V^2)$

Record your answers in the bubbles next to each question.

74. Finding Your Flight, Part II

(A) (B) (C) (D) (E)

You currently have a graph that represents several airports (vertices) and the flights that connect each of them (edges), weighted by the distance of each flight. You decide to implement this graph using an adjacency list. The number of vertices is represented by V, and the number of edges is represented by E.

Given an airport X, what is the average-case time complexity of finding if any flights depart from airport X?

A) $\Theta(1)$

B) $\Theta(E)$

C) $\Theta(V)$

D) $\Theta(1 + \frac{E}{V})$

E) $\Theta(V^2)$

75. Mystery Weights

(A) (B) (C) (D) (E)

The partially-filled table below represents the distances between six locations. The graph associated with this table is simple and undirected.

	A	B	C	D	E	F
A		258			447	$-\mathbf{x}-$
B				611		
C	114					745
D		$-\mathbf{y}-$	331			
E						583
F	252					

What is the value of $x + y$?

A) 583

B) 778

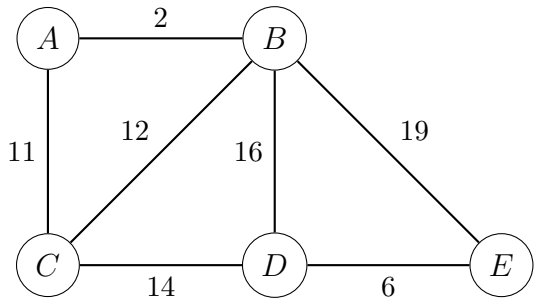
C) 863

D) 997

E) Not enough information is given to answer this question

Record your answers in the bubbles next to each question.

For questions 76-78, consider the following graph:



76. MST Builder, Part I (A) (B) (C) (D) (E)

Before doing any calculations, how many edges does the graph's MST have?

A) 3

B) 4

C) 5

D) 6

E) 7

77. MST Builder, Part II (A) (B) (C) (D) (E)

Using Prim's algorithm on the graph above (starting at vertex A), which vertex is added last?

A) A

B) B

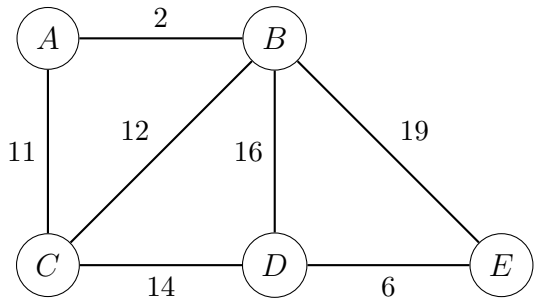
C) C

D) D

E) E

Record your answers in the bubbles next to each question.

For questions 76-78, consider the following graph:



78. MST Builder, Part III (A) (B) (C) (D) (E)

Using Prim's algorithm on the graph above (starting at vertex A), what is the total weight of the MST?

A) 31

B) 33

C) 45

D) 80

E) None of the above

79. MST Edge Cases (A) (B) (C) (D) (E)

Which of the following statements is **TRUE**?

A) Kruskal's algorithm builds a "forest" of trees one vertex at a time

B) If an edge has the smallest weight value in a graph, it must be included in the graph's MST

C) If an edge has the largest weight value in a graph, it can never be included in the graph's MST

D) Prim's and Kruskal's algorithms work on both directed and undirected graphs

E) None of the above

Record your answers in the bubbles next to each question.

80. Kruskal's Algorithm Ⓐ Ⓑ Ⓒ Ⓓ ⓔ

Which of the following statements is **FALSE**?

A) The complexity of Kruskal's algorithm on a valid graph with E edges is $\Theta(E \log(E))$

B) The sorting algorithm involved in Kruskal's is the bottleneck of the entire algorithm

C) The efficiency of Kruskal's algorithm relies on the efficiency of the union-find data structure

D) For certain graphs, Kruskal's and Prim's algorithms may produce different MSTs

E) Kruskal's algorithm does not work for graphs with negative edge weights

81. Changing the Weights Ⓐ ⓑ Ⓒ Ⓓ Ⓔ

Consider the following four scenarios:

I. You increase the weight of an edge that is in the MST

II. You decrease the weight of an edge that is in the MST

III. You increase the weight of an edge that is not in the MST

IV. You decrease the weight of an edge that is not in the MST

For which of these scenarios could the MST change?

A) I and III only

B) I and IV only

C) II and III only

D) II and IV only

E) I, II, III, and IV

Record your answers in the bubbles next to each question.

82. Enter the Matrix

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Suppose you are given the following distance matrix for five different vertices.

	V_1	V_2	V_3	V_4	V_5
V_1	0	9	12	10	7
V_2	9	0	14	6	8
V_3	12	14	0	13	11
V_4	10	6	13	0	5
V_5	7	8	11	5	0

What is the total weight of the MST that connects these five vertices?

A) 26

B) 27

C) 28

D) 29

E) None of the above

83. Greedy Sorts

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Which of the following sorts does **NOT** use the greedy algorithm?

A) Bubble Sort

B) Selection Sort

C) Quicksort

D) Insertion Sort

E) None of the above

Record your answers in the bubbles next to each question.

84. Avaricious Algorithms

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Consider the following three algorithms:

I. Prim's Algorithm

II. Kruskal's Algorithm

III. Dijkstra's Algorithm

Which of these algorithms are greedy?

A) I only

B) III only

C) I and II only

D) I and III only

E) I, II, and III

85. Brute Forcing a Vacation

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

You decide to use the brute force algorithm to calculate the optimal distance needed to travel to Chicago. Using this method, you obtain a result of 243 miles. Which of the following can you safely assume?

A) The optimal distance is exactly 243 miles

B) The optimal distance may be less than 243 miles

C) The optimal distance may be greater than 243 miles

D) All paths to Chicago will take exactly 243 miles

E) We cannot safely assume any of the above from this result

86. Breaking into a Computer

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Suppose you are trying to break into Dr. Paoletti's computer so that you can ~~peek at the 281 final exam~~ surprise him with a thank you note for all the work he has put into the course. You noticed during lecture that he had a 9-character password, so you are randomly generating passwords of length 9. What algorithm is this an example of?

A) Backtracking

B) Divide and Conquer

C) Branch and Bound

D) Brute Force

E) Dynamic Programming

Record your answers in the bubbles next to each question.

87. Scheduling Classes (A) B C D E

Your course registration is tomorrow, and you know that you must take five courses next semester to stay on track for graduation. Each of these classes has multiple sections, all at different times and locations. You take out your paper and pencil and try to make a schedule that fits the following constraints: (1) no two classes can overlap, (2) a half-hour block must be reserved between classes on Central and North, and (3) consecutive classes cannot last more than four hours in duration. Which one of the following algorithm families would be best at accomplishing this task?

A) Backtracking

B) Divide and Conquer

C) Branch and Bound

D) Brute Force

E) Greedy

88. Cedar Point (A) B C D E

Your favorite amusement park, Cedar Point, is closing in two hours, and you still haven't gotten to most of the rides! You know that each of the rides left have wait times of between 15 and 90 minutes. In addition, different rides give you different levels of satisfaction (e.g., riding the *Top Thrill Dragster* may give you 100 units of satisfaction while riding *Snoopy's Express Railroad* may only give you 5 units of satisfaction despite a lower wait time). Both wait time and satisfaction take on integer values. You want to maximize the satisfaction you get from these last two hours, and you don't want to ride the same ride twice. Which one of the following algorithm families would be best at accomplishing this task?

A) Dynamic Programming

B) Brute Force

C) Backtracking

D) Divide and Conquer

E) Greedy

Record your answers in the bubbles next to each question.

89. Dining Hall Woes

Ⓐ Ⓑ Ⓒ Ⓓ ⓔ

The dining hall just closed for the night, and the workers have started kicking everyone out. As the other students leave, you realize that you have a problem on your hands – you got way too much food! You still have several uneaten dishes on your table, and each of these dishes gives you different levels of satisfaction. However, your stomach can only handle so much, and you prefer not to throw up after you leave. Which algorithm family would be most efficient at determining what you should eat to maximize your satisfaction, while also considering the capacity of your stomach? If you begin a dish, you do NOT need to finish it in its entirety.

A) Dynamic Programming

B) Brute Force

C) Backtracking

D) Divide and Conquer

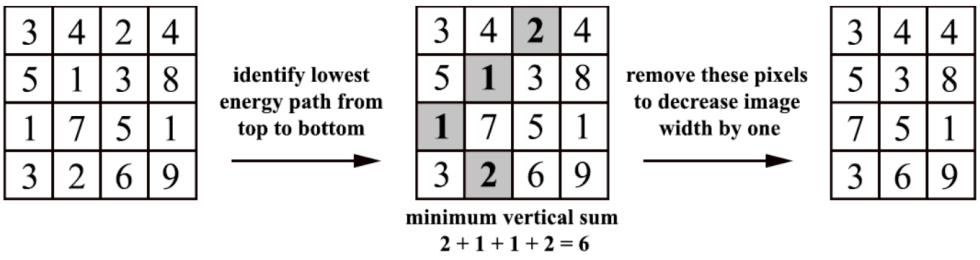
E) Greedy

90. Seam Carving

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Seam carving is an algorithm that can be used to perform content-aware resizing of images, allowing them to be scaled without losing meaningful content. Each pixel of an image is assigned a number (called an energy value) that captures how different its color is from surrounding pixels.

When the width of an image is decremented, the algorithm looks for the sequence of pixels that goes from top to bottom with the minimum total sum. A valid sequence can only contain *one pixel per row*, and the column position of pixels in adjacent rows *must differ by at most one*. This "lowest-energy" sequence is then removed to reduce the width of the image by 1.



The seam carving process is best done using which one of the following algorithms?

A) Brute Force

B) Divide and Conquer

C) Dynamic Programming

D) Backtracking

E) Branch and Bound

Record your answers in the bubbles next to each question.

91. Coloring the Country

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

You want to color a graph of the United States such that no adjacent states share the same color. Which one of the following algorithm families would be best at accomplishing this task?

A) Branch and Bound

B) Backtracking

C) Brute Force

D) Combine and Conquer

E) Greedy

92. Constraint Satisfaction

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following is a constraint satisfaction problem and not an optimization problem?

A) Finding the route with the minimum distance to New York City

B) Finding an MST of a connected graph

C) Finding the shortest path out of a maze

D) Finding a way to walk from the BBB to the IOE building without going outside

E) None of the above

93. Bound to Backtrack

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

What is the difference between backtracking and branch and bound?

A) Unlike backtracking, branch and bound prunes solutions that do not work

B) Unlike backtracking, branch and bound stores partial solutions that may be needed later

C) Unlike backtracking, branch and bound can be used to find an optimal solution to a problem

D) Unlike backtracking, branch and bound stops immediately once a solution is found

E) More than one of the above

Record your answers in the bubbles next to each question.

94. Algorithm Choices Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Which of the following statements is **FALSE**?

A) The branch and bound solution to the 0-1 Knapsack problem runs in $\Theta(n2^n)$ time if you are given n items to choose from

B) The branch and bound algorithm can be used to find a Hamiltonian Cycle with the lowest weight

C) Given n items and a knapsack capacity of m, the dynamic programming solution to the 0-1 Knapsack problem runs in $\Theta(mn)$ time

D) Because backtracking avoids looking at large portions of the search space by pruning, the asymptotic complexity of backtracking is always better than that of brute force

E) The backtracking algorithm can be used to check if n queens can be placed on an $n \times n$ chessboard without any threatening each other

95. Dropping a Solution Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

You are currently running the traveling salesperson problem to try to find the minimum distance required to visit 50 locations in the state of Michigan. The upper bound you have using the branch and bound algorithm is 331 miles. Currently, you are considering a path where the distance required to visit 25 locations is 165 miles, the MST connecting the remaining 25 locations has a total distance of 141 miles, and the connections that link the current path to the MST have a total distance of 10 miles. Which of the following statements is **TRUE**?

A) Because the distance of the current path, 316 miles, is less than the current upper bound of 331 miles, the upper bound should be changed to 316 miles

B) Because the distance of the current path, 316 miles, is less than the current upper bound of 331 miles, the current path being considered should be dropped as a potential solution

C) The current path being considered is guaranteed to be the lowest-weighted Hamiltonian cycle

D) The current path being considered cannot be the lowest-weighted Hamiltonian cycle

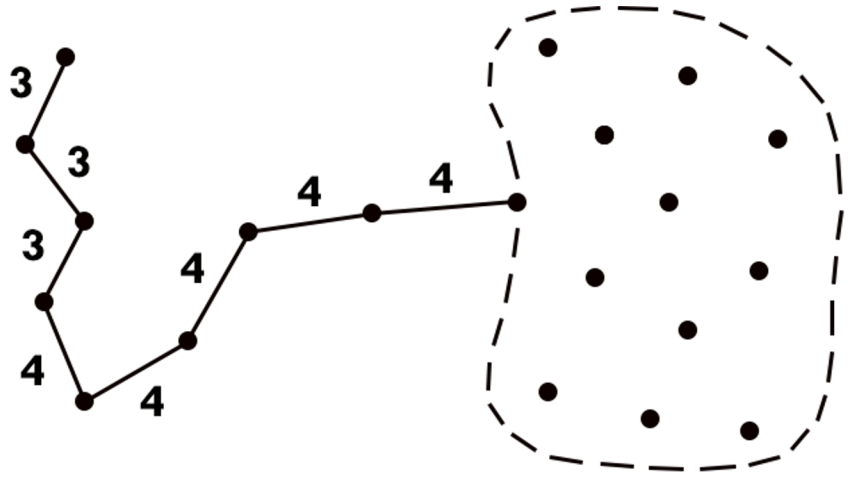
E) Because the weight of the path being considered has a minimum possible distance of 316 miles, it would be okay to consider this as a lower bound for extending this potential solution

Record your answers in the bubbles next to each question.

96. Tune Your Bounds

Ⓐ Ⓑ Ⓒ Ⓓ ● Ⓔ

You are currently using branch and bound to calculate the TSP path for 20 different points. A snapshot of the branch and bound process is shown below:



At the moment of the above snapshot, the value of the upper bound is 129. Let d represent the total weight of the MST that connects all 12 points within the dotted region, including the point on the border. Which of the following statements **MUST** be true?

A) If d is less than 100, the upper bound should be changed

B) If d is equal to 100, the upper bound should be changed

C) If d is greater than 100, the upper bound should be changed

D) Both A and C

E) None of the above

97. TSP Heuristics

● Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Which of the following statements is **FALSE**?

A) Heuristics are algorithms that can be used to calculate an optimal solution very quickly

B) Heuristics often have time complexities much faster than that of brute force

C) Heuristics are extremely useful if the standard method for solving a problem takes too long

D) Heuristics can be used to estimate an upper-bound to a TSP problem

E) None of the above

Record your answers in the bubbles next to each question.

98. Estimating an Upper Bound

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

Four EECS 281 students are trying to solve the 0-1 Knapsack problem using the branch and bound algorithm. Each student uses a different implementation to estimate the upper bound used for pruning:

- Student 1 is using dynamic programming to estimate the upper bound
- Student 2 is using the greedy algorithm to estimate the upper bound
- Student 3 is using brute force to estimate the upper bound
- Student 4 randomly generates permutations of items to place in the knapsack and sets the value generated by the first feasible permutation as the initial upper bound

Which student has the most efficient implementation for determining the upper bound?

A) Student 1

B) Student 2

C) Student 3

D) Student 4

E) None of the above implementations are efficient

99. Greedy Knapsacks

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Suppose you had the following five items, and you want to place them in a 0-1 knapsack of capacity 13:

Size	1	3	6	7	10
Value	9	11	23	28	30

Which of the following statements is **TRUE**?

A) The greedy approach of selecting items with the highest value first will produce the optimal solution

B) The greedy approach of selecting items with the lowest size first will produce the optimal solution

C) The greedy approach of selecting items with the highest value-to-size ratio first will produce the optimal solution

D) The greedy approach of selecting items with the lowest size-to-value ratio first will produce the optimal solution

E) None of the above

Record your answers in the bubbles next to each question.

100. Know Your Knapsacks

Ⓐ Ⓑ **Ⓒ** Ⓓ Ⓔ

Which of the following algorithms does **NOT** guarantee an optimal solution to the 0-1 Knapsack problem?

A) Branch and Bound

B) Dynamic Programming

C) Greedy

D) Brute Force

E) More than one of the above

101. Largest Subset

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

You are given a set of N integers and a target integer K. Suppose you want to find the subset of these N integers with the largest size, such that the total sum of all elements in the subset is less than or equal to K. Which algorithm family should you use to solve this problem, if you want the best time complexity?

A) If $\log(N) < K$, choose greedy, otherwise choose dynamic programming

B) If $\log(N) > K$, choose greedy, otherwise choose dynamic programming

C) If $N < K$, choose greedy, otherwise choose brute force

D) If $N > K$, choose greedy, otherwise choose brute force

E) The greedy algorithm should be chosen for all values of N and K

102. Professor Darden's Vacation

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

Professor Darden plans to visit the computer science buildings of every major university in the United States, and he wants to find the shortest route that will allow him to visit each campus exactly once before returning to Ann Arbor. What is the worst-case time complexity of using *brute force* to calculate this shortest path?

A) $\Theta(2^n)$

B) $\Theta(n2^n)$

C) $\Theta(n^2)$

D) $\Theta(n^n)$

E) $\Theta(n!)$

Record your answers in the bubbles next to each question.

103. Partial Knapsack

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Assume you are implementing a dynamic programming approach to the 0-1 knapsack problem, and you are trying to find the maximum possible value you can take in your knapsack of weight capacity 5. You have the following items:

Item id	Weight	Value
1	2	\$35
2	1	\$16
3	4	\$61
4	3	\$53

The memo for this problem is shown below. Some of the values in the memo have already been filled in.

id/weight	0	1	2	3	4	5
0	\$0	\$0	\$0	\$0	\$0	\$0
1	\$0	\$0	\$35	\$35		
2	\$0					
3	\$0					
4	\$0				T	

After the memo is completely filled in, what would be the value of T?

A) \$61

B) \$64

C) \$69

D) \$70

E) \$88

Record your answers in the bubbles next to each question.

104. B&B Truths, Part I Ⓐ ●

True or false? Running the branch and bound algorithm on the traveling salesperson problem is *guaranteed* to make the program run faster, compared to a brute force solution.

A) True

B) False

105. B&B Truths, Part II Ⓐ ●

True or false? The worst-case time complexity of running the traveling salesperson problem using the branch and bound algorithm is better than the worst-case time complexity of running the traveling salesperson problem using brute force.

A) True

B) False

106. DP Truths, Part I Ⓐ ●

True or false? Dynamic programming often reduces both the time and memory used to run a program with multiple overlapping subproblems.

A) True

B) False

107. DP Truths, Part II Ⓐ Ⓑ Ⓒ ● Ⓔ

Which of the following statements about dynamic programming is **FALSE**?

A) Bottom-up dynamic programming can be used any time top-down dynamic programming is used

B) Top-down dynamic programming only computes the answers to subproblems that are needed

C) Dynamic programming can be used to reduce the running time of a recursive function to be less than the time required to evaluate the function

D) Dynamic programming can be used to solve a problem that can also be solved using divide and conquer

E) None of the above

Record your answers in the bubbles next to each question.

108. Hailstone Numbers Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

The hailstone sequence for a positive integer X is a sequence of numbers that begins at X and uses the following rules to determine the next number in the sequence:

1. If X is even, the next number in the sequence is $\frac{X}{2}$
2. If X is odd, the next number in the sequence is $3X + 1$

The sequence ends when a value of 1 is reached. For example, the hailstone sequence for the number 3 would be $3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$. You want to write a function that takes in an integer N and returns the integer less than or equal to N with the longest hailstone sequence. Which of the following algorithmic approaches should you use to solve this problem most efficiently?

A) Divide and Conquer

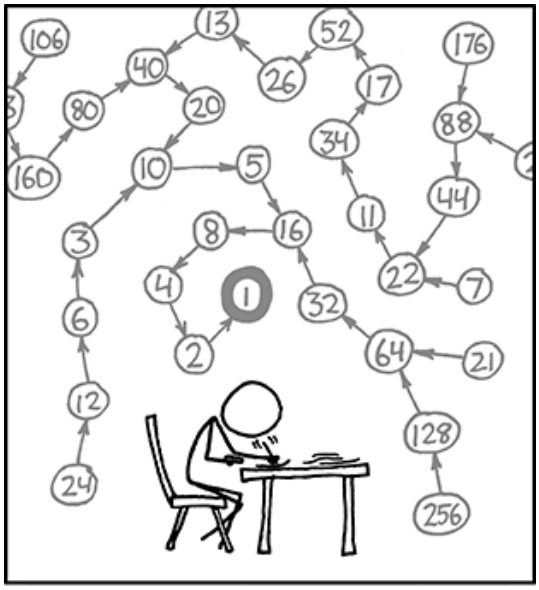
B) Dynamic Programming

C) Backtracking

D) Brute Force

E) Greedy

Bonus Practice: Try to implement a program that solves this problem!



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

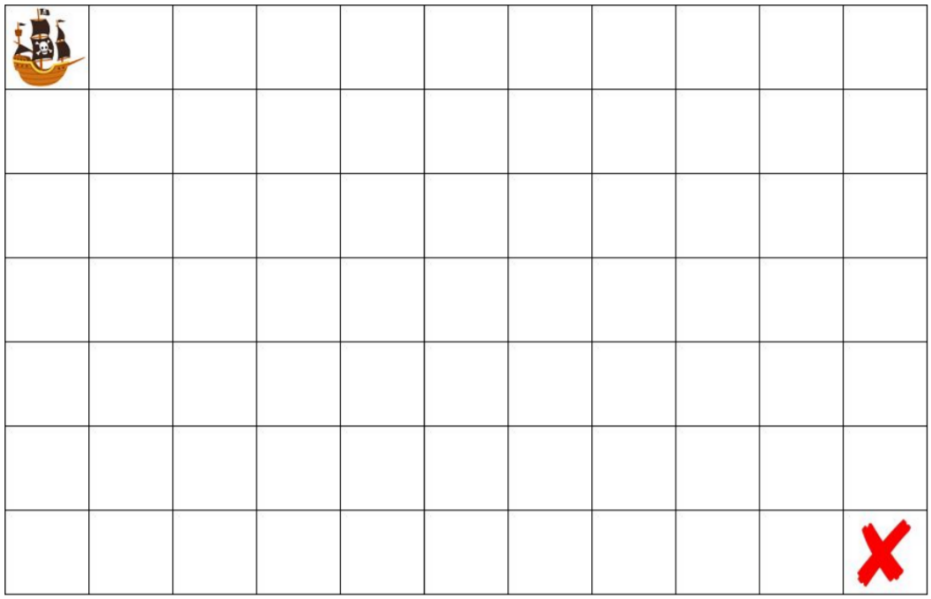
Image source: https://xkcd.com/710/

Record your answers in the bubbles next to each question.

109. The Lost Boys Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

Captain Dave "Foo-Bar" Paoletti and his band of pirates are traversing the high seas in search of the elusive Kraken. In the midst of their search, they happen upon an abandoned ship full of thousands of gold coins. The pirates decide to gather up and figure out a way to, in the eloquent words of shipmate Kevin, "divvy the booty, yarr." Whilst discussing the divvy method, Kevin attempts to steal all the booty and abandon the ship. But alas, shipmate Marcus "Parrot-Cop" Darden catches him and decides to make him walk the plank. Unfortunately, after throwing Kevin overboard, the other pirates realize that he was the only pirate who knew the super secret pirate-ship-sailing algorithm that they needed to get home!

Now, Captain "Foo-Bar" and his band of pirates are very lost. But just how lost are they? Consider the following 7×11 map:



Without Kevin, Captain "Foo-Bar" must choose a random combination of RIGHT and DOWN movements and hope that the resulting path gets his pirates home (this is a sailing ship, and thus can only sail with the wind). A valid route is any possible combination of RIGHT or DOWN movements that does NOT send the pirates off the 7×11 map. The pirate ship is located at the upper-left corner of the map, and the pirate lair is located at the lower-right corner of the map. How many valid routes exist that can safely take Captain "Foo-Bar" and his crew back home? *Hint: it may be faster to write a program that can help you solve this problem.*

A) 3,003

B) 5,005

C) 8,008

D) 12,376

E) 19,448

Record your answers in the bubbles next to each question.

110. Speed Dating

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

A group of n college students are attending a speed dating event. At this event, each student can remain single or partner up with another student, as long as they partner up only once. You want to find the total number of ways in which these students can remain single or be paired up. For example, for a group of 3 students, there are 4 possible outcomes:

```
{1}, {2}, {3}
 {1, 2}, {3}
 {1}, {2, 3}
 {1, 3}, {2}
```

What is the recurrence relation for this problem, and what is the time complexity of solving this problem if you use dynamic programming?

A) Recurrence Relation: $f(n) = f(n-1) + f(n-2)$
Time Complexity: $\Theta(n)$

B) Recurrence Relation: $f(n) = f(n-1) + (n-1) \times f(n-2)$
Time Complexity: $\Theta(n)$

C) Recurrence Relation: $f(n) = f(n-1) + f(n-2)$
Time Complexity: $\Theta(n^2)$

D) Recurrence Relation: $f(n) = f(n-1) + (n-1) \times f(n-2)$
Time Complexity: $\Theta(n^2)$

E) None of the above

111. Running Dijkstra's Algorithm

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

If you run Dijkstra's algorithm once on a directed graph, you will learn

A) The shortest distance from every vertex to every other vertex

B) The shortest distance from one vertex to every other vertex

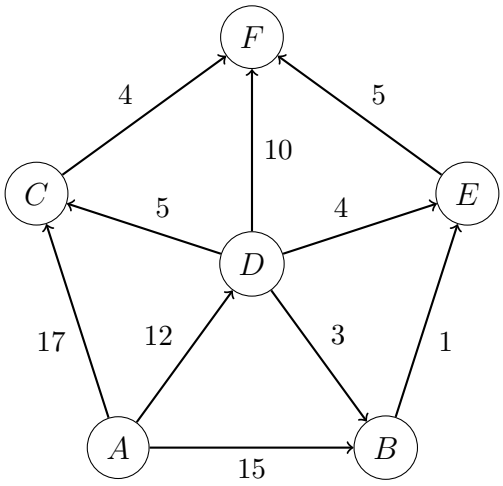
C) The shortest distance from every vertex to one vertex

D) More than one of the above

E) None of the above

Record your answers in the bubbles next to each question.

For questions 112-113, consider the following directed graph:



112. Directed Dijkstra's, Part I (A) (B) (C) **(D)** (E)

Consider the above directed graph. What path does Dijkstra's algorithm discover from A to F? *Only update predecessors for weights that are improvements.*

A) $ABEF$

B) ACF

C) $ADCF$

D) $ADEF$

E) Any of the above

113. Directed Dijkstra's, Part II (A) **(B)** (C) (D) (E)

Consider the above directed graph, but **add 100 to each edge weight**. What path does Dijkstra's algorithm discover from A to F? *Only update predecessors for weights that are improvements.*

A) $ABEF$

B) ACF

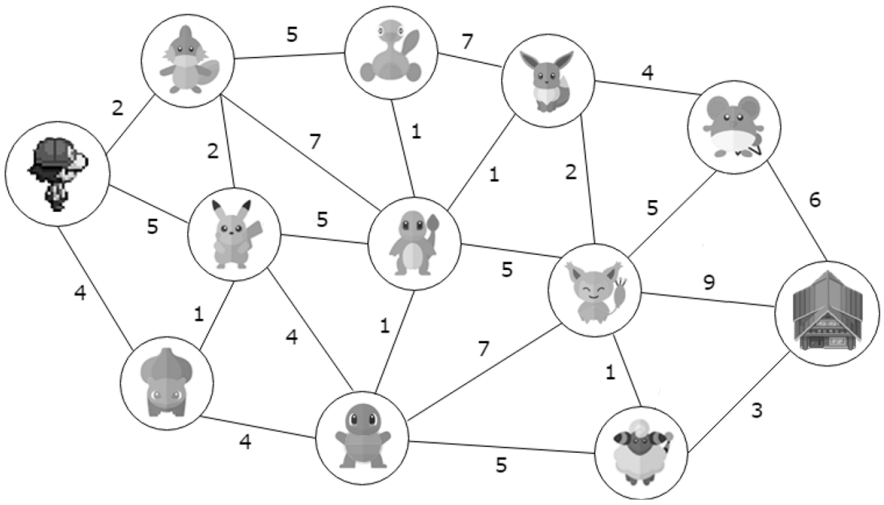
C) $ADCF$

D) $ADEF$

E) Any of the above

Record your answers in the bubbles next to each question.

For questions 114-115, consider the following graph:



Suppose that a Pokémon trainer is currently located at the vertex on the very left, and he wants to go back to his house (located at the vertex on the very right). The routes that he can take to get back home are represented by the edges above, with distances as marked.

114. Dijkstra-mon, Part I (A) **(B)** (C) (D) (E)

Using Dijkstra's algorithm, what is the distance of the shortest possible path the trainer can take to get back to his house?

A) 14

B) 15

C) 16

D) 17

E) 18

115. Dijkstra-mon, Part II (A) (B) (C) **(D)** (E)

On his route home, the trainer will encounter several different Pokémon. A Pokémon can be found at any vertex that the trainer crosses, excluding his initial position and his house. While taking the shortest path home, how many Pokémon does the trainer encounter?

A) 3

B) 4

C) 5

D) 6

E) 7

Record your answers in the bubbles next to each question.

Computational geometry will NOT be on the final exam this semester. However, the following five practice questions (116-120) are provided in case you wanted to try this topic out.

116. Image Processing

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Consider the following three statements regarding raster and vector graphics.

I. Images that are created using raster graphics can be easily enlarged without losing quality or detail.

II. Vector graphics are resolution independent, as they are constructed using anchor points that are connected using mathematical equations.

III. If a raster image utilizes the RGB color model, each pixel of that image stores three integers in the range $[0, 255]$ that can be used to identify its color.

Which of these statements is/are **TRUE**?

A) I only

B) III only

C) I and II only

D) II and III only

E) I, II, and III

117. Closest Pair of Points

Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ

You are given a vector of n points on an xy-plane, and you want to find the closest pair of points in the vector. Which of the following statements is **TRUE**?

A) An optimal brute force solution for this problem runs in $\Theta(n^2)$ time, and an optimal divide and conquer solution runs in $\Theta(n \log(n))$ time

B) An optimal brute force solution for this problem runs in $\Theta(2^n)$ time, and an optimal divide and conquer solution runs in $\Theta(n \log(n))$ time

C) An optimal brute force solution for this problem runs in $\Theta(n^2)$ time, and an optimal divide and conquer solution runs in $\Theta(\log(n))$ time

D) An optimal brute force solution for this problem runs in $\Theta(2^n)$ time, and an optimal divide and conquer solution runs in $\Theta(\log(n))$ time

E) An optimal brute force solution for this problem runs in $\Theta(2^n)$ time, and an optimal divide and conquer solution runs in $\Theta(n^2)$ time

Record your answers in the bubbles next to each question.

118. Ray Sweep

Ⓐ **Ⓑ** Ⓒ Ⓓ Ⓔ

You are given n line segments, whose endpoints have integer coordinates (x, y) where $x \geq 1$ and $y \geq 1$. The two endpoints of any line segment are distinct, and n is guaranteed to be non-zero. Consider the following function:

```
1  struct Point { int x; int y; };
2  struct Segment { Point a; Point b; };
3
4  double question_118(vector<Segment> &segments) {
5     vector<pair<double, int>> ends;
6     for (Segment &s : segments) {
7        double slope_a = double(s.a.y) / double(s.a.x);
8        double slope_b = double(s.b.y) / double(s.b.x);
9        ends.push_back({min(slope_a, slope_b), -1});
10       ends.push_back({max(slope_a, slope_b), +1});
11    }
12    sort(ends.begin(), ends.end());
13    int count = 0, best = 0;
14    double slope = 0;
15    for (auto &p : ends) {
16       count -= p.second;
17       if (count > best) {
18          best = count;
19          slope = p.first;
20       }
21    }
22    return slope;
23 }
```

What does this function attempt to do, given a vector of line segments?

A) It returns the slope of a line through the origin that crosses as few line segments as possible

B) It returns the slope of a line through the origin that crosses as many line segments as possible

C) It returns the slope of a line that crosses as many segment endpoints as possible

D) It returns the slope of the longest line segment in the vector

E) It returns the slope that is shared by the greatest number of line segments in the vector

Record your answers in the bubbles next to each question.

119. Intersection Within Disk

Ⓐ Ⓑ Ⓒ **Ⓓ** Ⓔ

Given n line segments and a disk (represented as a center point and a radius) in an xy-plane, you want to determine whether any two line segments intersect inside of the disk. What is the worst-case time complexity of solving this problem, if you use the most efficient algorithm?

A) $\Theta(\log(n))$

B) $\Theta(\sqrt{n})$

C) $\Theta(n)$

D) $\Theta(n \log(n))$

E) $\Theta(n^2)$

120. Point Inside Polygon

Ⓐ Ⓑ Ⓒ Ⓓ **Ⓔ**

A horizontal line is projected out from a point Q that lies on the same plane as an unknown polygon P. If the horizontal line touches P exactly seven times as it travels away from Q, what can you conclude?

A) Point Q lies inside the polygon P

B) Point Q lies outside the polygon P

C) Point Q lies on an edge of polygon P

D) You cannot make any of the above conclusions because a vertical line test was not performed on Q

E) You cannot make any of the above conclusions because epsilon tests were not performed on Q and its horizontal line

The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms

Final Exam Written Questions

— *Additional Practice* —

INSTRUCTIONS: This section contains 34 written questions to help you prepare for the final exam. For the previous exam questions, you can download the starter files for each question on Canvas and submit your solutions to Gradescope. Good luck!

List of Practice Questions

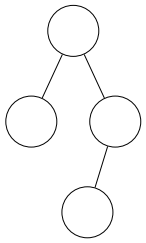
Previous Exam Questions (with Autograders)

121. Minimum Nodes in an AVL Tree

You are given an AVL tree of height h. Write a function that returns the minimum number of nodes the AVL tree of height h can have. For this problem, assume that a tree with height 1 only has one node.

Hint: Since AVL trees must be balanced, the height of each node's left and right children cannot have a difference that exceeds 1.

Example: If h is 3, your function should return 4, since the smallest possible AVL tree with height 3 has 4 nodes (see the tree below).



Complexity: $O(h)$ time and $O(h)$ auxiliary memory.

Implementation: Implement your solution in the space below. You may use anything from the STL. Line limit: 15 lines of code.

```
int min_nodes_in_AVL(int h) {
  vector<int> memo(h + 1);
  memo[0] = 0;
  memo[1] = 1;
  for (int i = 2; i < h + 1; ++i) {
    memo[i] = 1 + memo[i - 1] + memo[i - 2];
  }
  return memo[h];
}
```

122. Largest Distance Between Repeated Elements

You are given an array with repeated elements. Implement a function that identifies the maximum distance between any two occurrences of a repeated element.

Example: Given the following input

1	2	3	2	2	1	3	3	2

your function should return 7, since the maximum distance between any two repeated elements is the distance between the 2 at index 1 and the 2 at index 8, or 8 - 1 = 7.

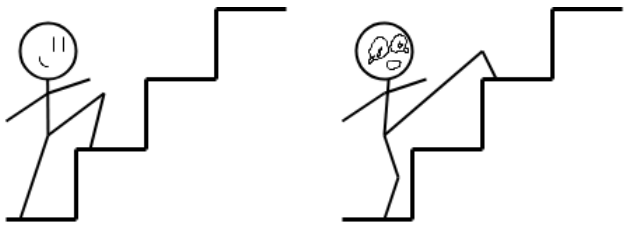
Complexity: Average $O(n)$ time, $O(n)$ auxiliary memory, where n is the length of the array.

Implementation: Implement your solution in the space below. You may use anything from the STL. Line limit: 15 lines of code.

```
int max_repeated_distance(const vector<int> &vec) {
  unordered_map<int, int> firstIndices;
  int result = 0;
  for (int i = 0; i < static_cast<int>(vec.size()); ++i) {
    if (firstIndices.find(vec[i]) == firstIndices.end()) {
      firstIndices[vec[i]] = i;
    }
    else {
      result = max(result, i - firstIndices[vec[i]]);
    }
  }
  return result;
}
```

123. Climbing Stairs

You are currently climbing a staircase where n steps are needed to reach the top. When you climb the stairs, you can either choose to move up one step or move up two steps, as shown below:



Implement the following function, which takes in the value of n and determines the number of distinct ways one can climb to the top of the staircase, given that they can either move up one step or two steps with each move.

Complexity: $O(n)$ time and $O(n)$ auxiliary memory.

Implementation: Implement your solution in the space below. You may use anything from the STL. Line limit: 15 lines of code.

```
int climb_stairs(int n) {
  vector<int> memo(n);
  memo[0] = 1;
  memo[1] = 2;
  for (int i = 2; i < n; ++i) {
    memo[i] = memo[i - 1] + memo[i - 2];
  }
  return memo[n - 1];
}
```

124. Jump Game

You are given a vector of non-negative integers, and you are initially positioned at the first index of the vector. Each element in the array represents your maximum jump length at that position.

Write a program that returns whether you are able to reach the last index.

Example 1: Given the following vector:

```
[2, 3, 1, 1, 4]
```

return **`true`**. You can reach the last index by jumping 1 step from index 0 to index 1, then jumping 3 steps to the last index.

Example 2: Given the following vector:

```
[3, 2, 1, 0, 4]
```

return **`false`**. You will always arrive at index 3 no matter what. Since your maximum jump length at index 3 is 0, it is impossible to reach the last index.

Complexity: $O(n)$ time and $O(n)$ auxiliary space, where n is the number of elements in the vector.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 15 lines of code.

```
bool can_jump_out(vector<int> &nums) {
  if (!nums.empty() && !nums[0]) {
    return nums.size() == 1;
  }
  if (nums.size() > 1) {
    vector<int> dp(nums.size(), nums[0]);
    for (int i = 1; i < static_cast<int>(nums.size()) - 1; ++i) {
      dp[i] = max(nums[i], dp[i - 1] - 1);
      if (!dp[i]) {
        return false;
      }
    }
  }
  return true;
}
```

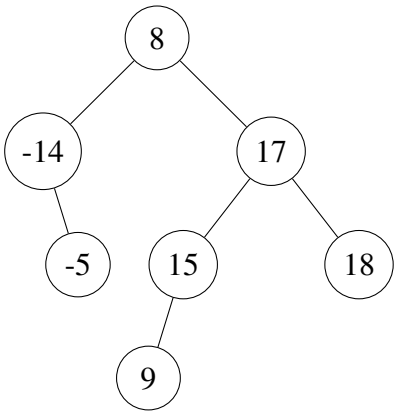
125. Trim BST to Fit Element Range

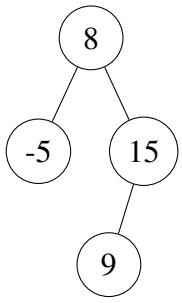
You are given a binary search tree, as well as two integers, `min_val` and `max_val`. Implement a function that removes all nodes in the BST whose values are less than `min_val` or greater than `max_val` while maintaining the BST property. You may assume that `min_val` $\leq$ `max_val`.

Each node of the tree is defined as follows:

```
struct Node {
   int val;
   Node *left;
   Node *right;
   Node(int x) : val{ x }, left{ nullptr }, right{ nullptr } {}
};
```

Example: Given the tree on the left and the values `min_val` = -12 and `max_val` = 15, your function should trim the tree so that all elements are in the range $[-12, 15]$ (as shown by the tree on the right):





Complexity: $O(n)$ time and $O(n)$ auxiliary space, where n is the number of nodes in the tree.

Implementation: Implement your solution on the back of this page. You may use anything from the STL. After you trim the tree, you should return the root in your function. Line limit: 25 lines of code.

Implement your solution to "Trim BST to Fit Element Range" below:

```
Node * trim_BST(Node *root, int min_val, int max_val) {
  if (!root) {
    return root;
  }
  root->left = trim_BST(root->left, min_val, max_val);
  root->right = trim_BST(root->right, min_val, max_val);
  if (root->val < min_val) {
    Node *right_child = root->right;
    delete root;
    return right_child;
  }
  if (root->val > max_val) {
    Node *left_child = root->left;
    delete root;
    return left_child;
  }
  return root;
}
```

126. Two City Scheduling

A company is planning to interview $2N$ people, and they want to fly these interviewees to an on-site location. The company has two locations, one in city A and one in city B. You are given a vector `costs`, where the cost of flying the `i`$^{\text{th}}$ person to city A is `costs[i][0]`, and the cost of flying the `i`$^{\text{th}}$ person to city B is `costs[i][1]`.

Return the minimum cost to fly every person to a city such that exactly N people arrive in each city.

Example: Given the following vector:

```
costs = [ [10, 20], [30, 200], [400, 50], [30, 20] ]
```

you would return 110, since that is the minimum cost required to have half of the people interviewing in each city (interviewees 0 and 1 go to city A, and interviewees 2 and 3 go to city B: the total cost of this process is 10 + 30 + 50 + 20 = 110).

Complexity: $O(N \log(N))$ time and $O(\log(N))$ auxiliary space.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 20 lines of code.

```
int min_travel_cost(vector<vector<int>> &costs) {
  int total_cost = 0;
  // this just sorts the vector in ascending order based on the cost
  // difference between cities (cost_A - cost_B) for each person
  // you don't need to use a lambda, a custom comparator also works
  sort(costs.begin(), costs.end(), [](vector<int> &v1, vector<int> &v2) {
    return (v1[0] - v1[1] < v2[0] - v2[1]);
  });
  for (size_t i = 0; i < costs.size() / 2; ++i) {
    total_cost += costs[i][0] + costs[i + costs.size() / 2][1];
  }
  return total_cost;
}
```

127. Dice Throw

You are given N dice, each with M faces that are numbered from 1 to M. Implement a function that returns the number of ways to get a sum of X, where X is the summation of values on each face after all of the dice are thrown.

Example: Given $N = 2$ (two dice), $M = 6$ (each die has 6 sides), and $X = 12$ (the target sum), your function should return 1, since there is only one way to get a total of 12 (by rolling a 6 on each die).

Complexity: $O(MNX)$ time and $O(NX)$ auxiliary space.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 15 lines of code.

```
int find_ways(int M, int N, int X) {
  vector<vector<int>> memo(N + 1, vector<int>(X + 1));
  for (int j = 1; j <= M && j <= X; ++j) {
    memo[1][j] = 1;
  }
  for (int i = 2; i <= N; ++i) {
    for (int j = 1; j <= X; ++j) {
      for (int k = 1; k <= M && k < j; ++k) {
        memo[i][j] += memo[i - 1][j - k];
      }
    }
  }
  return memo[N][X];
}
```

128. Generate Parentheses

Given n pairs of parentheses, write a function that generates all combinations of balanced parentheses.

Example: Given $n = 3$, you would generate all combinations of valid parentheses orderings using 3 pairs of parentheses. The output is shown below:

```
output = ["((()))", "(()())", "(())()", "()(())", "()()()"]
```

Complexity: $O(\frac{4^n}{\sqrt{n}})$ time and $O(\frac{4^n}{\sqrt{n}})$ auxiliary space.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 20 lines of code.

```
vector<string> generate_parentheses(int n) {
  vector<string> solution;
  backtrack(solution, "", n, n);
  return solution;
}

void backtrack(vector<string> &solution, string current, int left, int right) {
  if (left == 0 && right == 0) {
    solution.push_back(current);
    return;
  }
  if (left > 0) {
    backtrack(solution, current + "(", left - 1, right);
  }
  if (right > left) {
    backtrack(solution, current + ")", left, right - 1);
  }
}
```

129. Clone List with Random Pointers

You are given a singly-linked list where each node contains an additional random pointer that could point to any node in the list or `nullptr`. Each node is represented as follows:

```
struct Node {
  int val;
  Node *next;
  Node *random;
  Node(int val_in) : val{ val_in }, next{ nullptr }, random{ nullptr } {}
};
```

Write a function that returns a deep copy of the list.

Complexity: $O(n)$ time and $O(n)$ auxiliary space. However, it is possible to solve this problem with $\Theta(1)$ auxiliary space!

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 25 lines of code.

```
Node * copy_random_list(Node *head) {
  if (!head) {
    return nullptr;
  }
  unordered_map<Node *, Node *> ptr_map;
  Node *head_copy = new Node(head->val);
  ptr_map.emplace(head, head_copy);
  Node *orig = head->next;
  Node *copy = head_copy;
  while (orig) {
    Node *temp = new Node(orig->val);
    ptr_map.emplace(orig, temp);
    copy->next = temp;
    copy = temp;
    orig = orig->next;
  }
  orig = head;
  copy = head_copy;
  while (orig) {
    if (orig->random) {
      copy->random = ptr_map[orig->random];
    }
    else {
      copy->random = nullptr;
    }
    orig = orig->next;
    copy = copy->next;
  }
  return head_copy;
}
```

130. Nearest Zero

You are given a matrix of 0's and 1's. Write a function that finds the distance of the nearest 0 for each cell in the matrix. The distance between two adjacent cells is 1.

Example: Given the following matrix:

```
matrix = [ [0, 0, 0],
           [0, 1, 0],
           [1, 1, 1] ]
```

you should return the following:

```
result = [ [0, 0, 0],
           [0, 1, 0],
           [1, 2, 1] ]
```

You may assume that there is at least one 0 in the given matrix. Cells are only adjacent in four directions: up, down, left, and right.

Complexity: $O(MN)$ time and $O(MN)$ auxiliary space, where M and N are the dimensions of the matrix.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 30 lines of code.

```
vector<vector<int>> distance_to_zero(vector<vector<int>> &matrix) {
  vector<vector<int>> dir = { {1, 0}, {0, 1}, {0, -1}, {-1, 0} };
  queue<pair<int, int>> bfs;
  int m = matrix.size(), n = matrix[0].size();
  vector<vector<int>> dist(m, vector<int>(n, -1));
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j) {
      if (matrix[i][j] == 0) {
        bfs.push({i, j});
        dist[i][j] = 0;
      }
    }
  }
  while (!bfs.empty()) {
    pair<int, int> current = bfs.front();
    bfs.pop();
    for (vector<int> &offset : dir) {
      int r = current.first + offset[0];
      int c = current.second + offset[1];
      if (!(r == m || c == n || r < 0 || c < 0) && dist[r][c] == -1) {
        bfs.push({r, c});
        dist[r][c] = dist[current.first][current.second] + 1;
      }
    }
  }
  return dist;
}
```

131. Course Scheduler

There are a total of n courses you have to take to graduate, each labeled with an integer from 0 to $n-1$. Some courses may have prerequisites — for example, if `prerequisites[i] = [a, b]`, you must take course `b` *before* course `a` (i.e., `b` is a prerequisite of `a`).

Given the total number of courses `num_courses` and a vector of all `prerequisite` pairs, return the ordering of courses you should take to finish all courses. If there are many valid answers, return any of them. If it is impossible to finish all courses, return an empty array.

Example: Given `num_courses = 4` and `prerequisites = [[1, 0], [2, 0], [3, 1], [3, 2]]`, you would output `[0, 1, 2, 3]` OR `[0, 2, 1, 3]`. This is because course 3 can only be taken after finishing both 1 and 2, and courses 1 and 2 can only be taken after finishing course 0.

Complexity: $O(n+p)$ time and $O(n)$ auxiliary space, where n is the number of courses and p is the number of prerequisite connections between courses.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 30 lines of code.

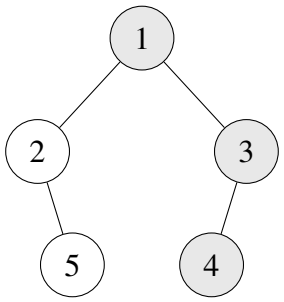
```
vector<int> find_order(int num_courses, vector<vector<int>> &prerequisites) {
  vector<vector<int>> graph(num_courses);
  vector<int> in_degrees(graph.size(), 0);
  for (auto &prereq : prerequisites) {
    graph[prereq[1]].push_back(prereq[0]);
    ++in_degrees[prereq[0]];
  }
  vector<int> result;
  queue<int> bfs;
  for (int course_num = 0; course_num < num_courses; ++course_num) {
    if (in_degrees[course_num] == 0) {
      bfs.push(course_num);
    }
  }
  while (!bfs.empty()) {
    int next_course = bfs.front();
    bfs.pop();
    result.push_back(next_course);
    for (int neighbor : graph[next_course]) {
      if (--in_degrees[neighbor] == 0) {
        bfs.push(neighbor);
      }
    }
  }
  return result.size() == num_courses ? result : vector<int>();
}
```

132. Binary Tree Right Side View

Given a binary tree, imagine yourself standing on the *right* side of it. Return the values of the nodes you can see, ordered from top to bottom. Each node is defined as follows:

```
struct Node {
   int val;
   Node *left;
   Node *right;
   Node(int x) : val{ x }, left{ nullptr }, right{ nullptr } {}
};
```

Example: Given the following tree, you would return `[1, 3, 4]`.



Complexity: $O(n)$ time and $O(n)$ auxiliary space, where n is the number of nodes in the tree.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 25 lines of code.

```
vector<int> right_side_view(Node *root) {
  vector<int> rhs_view;
  traverse(root, rhs_view, 0);
  return rhs_view;
}

void traverse(Node *root, vector<int> &rhs_view, int level) {
  if (!root) return;
  if (rhs_view.size() == level) {
    rhs_view.push_back(root->val);
  }
  traverse(root->right, rhs_view, level + 1);
  traverse(root->left, rhs_view, level + 1);
}
```

Alternative solution to "Binary Tree Right Side View":

```
vector<int> right_side_view(Node *root) {
  if (!root) return {};
  queue<Node *> bfs;
  vector<int> result;
  bfs.push(root);
  while (!bfs.empty()) {
    size_t level_size = bfs.size();
    Node *current = nullptr;
    for (size_t i = 0; i < level_size; ++i) {
      current = bfs.front();
      bfs.pop();
      if (current->left) {
        bfs.push(current->left);
      }
      if (current->right) {
        bfs.push(current->right);
      }
    }
    result.push_back(current->val);
  }
  return result;
}
```

133. Stock Trader

You are given a vector, `prices`, that stores the price of a given stock on each day. The value at `prices[i]` represents the price of the stock on day `i`.

Implement a function that returns the maximum profit you can make from buying and selling this stock under the following restrictions:

- You cannot engage in multiple transactions at the same time. Once you buy a stock, you must sell it before you can buy another one.
- After you sell a stock, you cannot buy stock on the next day.

Example: Given the following vector:

```
prices = [1, 2, 3, 0, 2]
```

you should return 3. This maximum profit can be attained by buying on day 0, selling on day 1, waiting on day 2, buying on day 3, and selling on day 4. You make $1 on the first transaction and $2 on the second transaction.

Complexity: $O(n)$ time and $O(n)$ auxiliary space, where n is the size of the `prices` vector.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 20 lines of code.

```
int max_profit(vector<int> &prices) {
  if (prices.size() < 2) {
    return 0;
  }
  vector<int> buy(prices.size()), sell(prices.size()), rest(prices.size());
  buy[0] = -prices[0];
  sell[0] = 0;
  rest[0] = std::numeric_limits<int>::min();
  for (size_t i = 1; i < prices.size(); ++i){
    sell[i] = max(sell[i - 1], rest[i - 1]);
    buy[i] = max(buy[i - 1], sell[i - 1] - prices[i]);
    rest[i] = buy[i - 1] + prices[i];
  }
  return max(sell[prices.size() - 1], rest[prices.size() - 1]);
}
```

Alternative solution:

```
int max_profit(vector<int> &prices) {
  vector<int> s{0, std::numeric_limits<int>::min(), 0};
  for (int i : prices) {
    s = {max(s[0], s[2]), max(s[0] - i, s[1]), s[1] + i};
  }
  return max(s[0], s[2]);
}
```

134. k-Combinations

Given two integers n and k, return all possible combinations of k numbers out of $1, \ldots, n$. You may return the answer in any order.

Example: Given $n = 4$, $k = 2$, one possible solution is:

```
[ [2, 4], [3, 4], [2, 3], [1, 2], [1, 3], [1, 4] ]
```

Complexity: $O\left(n^{\min(k,n-k)}\right)$ time and $O\left(n^{\min(k,n-k)}\right)$ auxiliary space.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 20 lines of code.

```
vector<vector<int>> k_combinations(int n, int k) {
  vector<vector<int>> solution;
  backtrack(n, k, 1, solution, vector<int>());
  return solution;
}

void backtrack(int n, int k, int start, vector<vector<int>> &solution,
  if (k == 0) {                                vector<int> current) {
    solution.push_back(current);
    return;
  }
  for (int i = start; i <= n; ++i) {
    current.push_back(i);
    backtrack(n, k - 1, i + 1, solution, current);
    current.pop_back();
  }
}
```

135. Minimum Deletions to Make Character Counts Unique

You are given a string `s` consisting of n lowercase letters. Write a function that returns the minimum number of characters that need to be deleted so that every character in `s` appears a unique number of times. You only need to care about the occurrences of letters that appear at least once in the final string.

Example: Given the string `s = "example"`, you would need to return 4, since a minimum of 4 letters need to be deleted for every letter in the string to appear a unique number of times. In this case, you must delete the letters 'a', 'm', 'p', and 'l' for every letter to have a unique count ('e' = 2, 'x' = 1) — note that any of these four letters can be switched with 'x' for the same result.

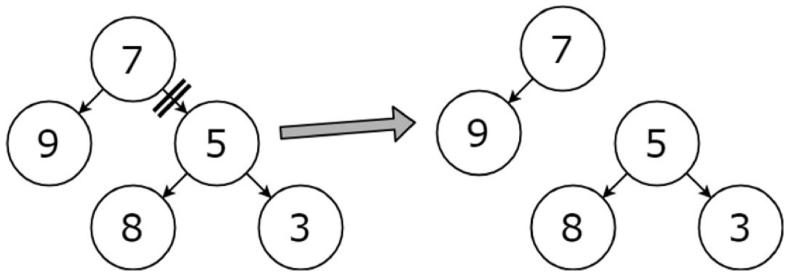
Complexity: $O(n)$ time and $O(n)$ auxiliary space.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 25 lines of code.

```
int min_deletions_for_unique_count(string s) {
  vector<int> freq(26);
  unordered_set<int> used;
  int result = 0;
  for (char c : s) {
    ++freq[c - 'a'];
  }
  for (int i = 0; i < 26; ++i) {
    int count = freq[i];
    while (count > 0 && !used.insert(count--).second) {
      ++result;
    }
  }
  return result;
}
```

136. Equal Tree Sum Partition

You are given a binary tree with n nodes. Write a function that checks if it is possible to partition the tree into two subtrees that have equal sums after removing exactly *one* edge of the original tree. For example, the following tree would return **`true`**, as removing the edge marked with a double slash would split the tree into two smaller subtrees that both have a sum of 16:



Each node of the tree shares the same structure as a node in problems 125 and 132 (with `val`, `*left`, and `*right` member variables).

Complexity: $O(n)$ time and $O(n)$ auxiliary space.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 25 lines of code.

Hint: trees are recursive structures, so think recursion! It may be helpful to write a helper function that can find the sum of a subtree when given its root.

```
bool equal_sum_subtree(Node *root) {
  unordered_map<int, int> sum_freqs;
  int sum = calc_sum(root, sum_freqs);
  if (sum == 0) {
    return sum_freqs[0] > 1;
  }
  return sum % 2 == 0 && sum_freqs[sum / 2] > 0;
}

int calc_sum(Node *root, unordered_map<int, int> &sum_freqs) {
  if (!root) return 0;
  int sum = root->val + calc_sum(root->left, sum_freqs)
                      + calc_sum(root->right, sum_freqs);
  ++sum_freqs[sum];
  return sum;
}
```

137. Edit Distance

You are given two strings `s1` and `s2`, and you are allowed to perform the following three operations to convert `s1` into `s2`:

- insert a letter
- delete a letter
- replace a letter

Implement a function that returns the minimum number of operations required to convert `s1` into `s2`.

Example: Given `s1 = "bunny"` and `s2 = "banana"`, you should return 3, since a minimum of 3 operations are needed to convert `"bunny"` into `"banana"`:

- insert `a` between the two `n`'s → `"bunany"`
- replace `u` with `a` → `"banany"`
- replace `y` with `a` → `"banana"`

Complexity: $O(mn)$ time and $O(mn)$ auxiliary space, where m is the length of `s1` and n is the length of `s2`.

Implementation: Implement your solution in the space below. You may use anything from the STL. Line limit: 25 lines of code.

```
int edit_distance(string s1, string s2) {
  vector<vector<int>> memo(s1.size() + 1, vector<int>(s2.size() + 1));
  for (int i = 0; i <= s1.size(); ++i) {
    for (int j = 0; j <= s2.size(); ++j) {
      if (i == 0)
        memo[i][j] = j;
      else if (j == 0)
        memo[i][j] = i;
      else if (s1[i - 1] == s2[j - 1])
        memo[i][j] = memo[i - 1][j - 1];
      else
        memo[i][j] = 1 + min({memo[i][j - 1],
                              memo[i - 1][j],
                              memo[i - 1][j - 1]});
    }
  }
  return memo[s1.size()][s2.size()];
}
```

138. Combination Lock

You have a lock in front of you with four circular wheels. Each wheel has ten slots: `'0'`, `'1'`, `'2'`, `'3'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`. The wheels can rotate freely and wrap around (e.g., `'9'` can become `'0'` in a single turn, and vice versa). Each move consists of a turning one wheel one slot.

The lock initially starts at `"0000"`, a string representing the state of the four wheels.

You are given a vector of `deadends`, and if the lock displays any of these dead ends, the wheels of the lock will stop turning and you will be unable to open it.

Given a `target` representing the value that will unlock the lock, return the minimum number of turns needed to open the lock, or `-1` if it is impossible.

Example: Given `deadends = ["0201", "0101", "0102", "1212", "2002"]` and `target = "0202"`, the function would return `6`, since six moves are needed to get to `"0202"` without hitting any of the dead ends: `"0000"` → `"1000"` → `"1100"` → `"1200"` → `"1201"` → `"1202"` → `"0202"`.

Implementation: You are given the following function, which returns all sequences that can be attained in a single turn from the original sequence `orig_seq`:

```
1   vector<string> single_turn_seqs(string orig_seq) {
2     vector<string> result;
3     for (int i = 0; i < 4; ++i) {
4       string temp = orig_seq;
5       temp[i] = (orig_seq[i] - '0' + 1) % 10 + '0';
6       result.push_back(temp);
7       temp[i] = (orig_seq[i] - '0' - 1 + 10) % 10 + '0';
8       result.push_back(temp);
9     }
10    return result;
11  }
```

Implement your solution on the next page. You may use anything from the STL. Line limit: 30 lines of code. You may use the `single_turn_seqs()` function in your solution — the provided implementation of this function does not contribute to the line count.

Implement your solution to "Combination Lock" below:

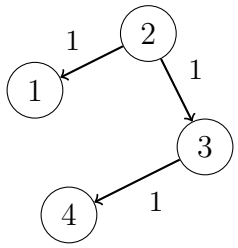
```
int min_turns_to_open_lock(vector<string> &deadends, string target) {
  unordered_set<string> de_set(deadends.begin(), deadends.end());
  unordered_set<string> visited;
  queue<string> bfs;
  string init = "0000";
  if (de_set.find(init) != de_set.end()) {
    return -1;
  }
  visited.insert(init);
  bfs.push(init);
  int result = 0;
  while (!bfs.empty()) {
    size_t curr_layer_size = bfs.size();
    for (size_t i = 0; i < curr_layer_size; i++) {
      string next = bfs.front();
      bfs.pop();
      if (next == target) {
        return result;
      }
      vector<string> possible_seqs = single_turn_seqs(next);
      for (auto seq : possible_seqs) {
        if (visited.find(seq) == visited.end() &&
            de_set.find(seq) == de_set.end()) {
          bfs.push(seq);
          visited.insert(seq);
        }
      }
    }
    ++result;
  }
  return -1;
}
```

139. Network Delay Time

You are given a network of n nodes, labeled 1 to n, and a vector of travel times `times` as **directed** edges, where `times[i] = (u, v, w)`, where `u` is the source node, `v` is the target node, and `w` is the time it takes for a signal to travel from source to target.

Write a function that takes in a vector of delay times, the number of nodes n, and a starting node k, and returns the time it would take for all nodes to receive a signal that is sent from node k. If it is not possible for all nodes to receive the signal, return `-1`.

Example: Given `times = [[2, 1, 1], [2, 3, 1], [3, 4, 1]]`, `n = 4`, and `k = 2`, you would return `2`. This is because it takes two units of time for all nodes in the network to receive the signal (where node 2 to 4 takes the longest).



Complexity: $O(t \log(n))$ time and $O(n + t)$ auxiliary space, where t is the length of the `times` vector.

Implementation: Implement your solution below. You may use anything from the STL. Line limit: 25 lines of code.

```
int network_delay_time(vector<vector<int>> &times, int n, int k) {
  unordered_map<int, vector<pair<int, int>>> graph;
  for (auto &time_vec : times) {
    graph[time_vec[0]].emplace_back(time_vec[1], time_vec[2]);
  }
  priority_queue<pair<int, int>, vector<pair<int, int>>,
                 greater<pair<int, int>>> pq;
  vector<int> dist(n + 1, std::numeric_limits<int>::max());
  pq.emplace(0, k);
  dist[k] = 0;
  while (!pq.empty()) {
    int u = pq.top().second;
    pq.pop();
    for (auto it = graph[u].begin(); it != graph[u].end(); ++it) {
      int v = it->first, w = it->second;
      if (dist[v] > dist[u] + w) {
        dist[v] = dist[u] + w;
        pq.emplace(dist[v], v);
      }
    }
  }
  int max_dist = *max_element(dist.begin() + 1, dist.end());
  return max_dist == std::numeric_limits<int>::max() ? -1 : max_dist;
}
```

140. Counting Rectangles

You are given a vector of points on a Cartesian (x-coordinate, y-coordinate) plane. Each point's `.x` member variable is its x-coordinate, and its `.y` member variable is its y-coordinate. Write a function that counts the number of **unique** rectangles that can be formed by these points. Do not count a rectangle more than once (e.g., a rectangle with points (A, B, C, D) is equivalent to a rectangle with points (D, A, C, B)). **Only consider rectangles whose sides are parallel to the x-axis and y-axis in your count.** All points in the `points` vector are unique, but they may be given in any order.

Example: Given the following points: $A = (1, 1)$, $B = (2, 1)$, $C = (3, 1)$, $D = (3, 2)$, $E = (2, 3)$, $F = (1, 3)$, and $G = (1, 2)$, you would return 2, since there are two unique rectangles that can be formed using these points ($ABEF$ and $ACDG$).

Hints: A rectangle is defined as a quadruple of points $(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)$ where $x_1 \neq x_2$ and $y_1 \neq y_2$. Rectangles can be described using either two vertical lines or two horizontal lines. When counting rectangles, any pair of vertical lines with the same pair of y-coordinates, or any pair of horizontal lines with the same pair of x-coordinates, forms a rectangle.

Complexity: $O(n^2 \log(n))$ time and $O(n^2)$ auxiliary space, where n is the number of coordinate points in the `points` vector.

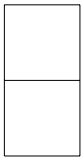
Implementation: Implement your solution below. You may use anything from the STL. Line limit: 20 lines of code.

```
struct Point { int x; int y; };
int count_rectangles(vector<Point> &points) {
  map<pair<int, int>, int> lines;
  int count = 0;
  for (auto &p1 : points) {
    for (auto &p2 : points) {
      if (p1.x == p2.x && p1.y < p2.y) {
        pair<int, int> vertical_line{ p1.y, p2.y };
        count += lines[vertical_line]++;
      }
    }
  }
  return count;
}
```

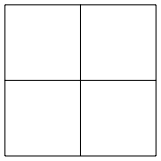
Fall 2016 Final Exam Question 25: Board Tiling

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

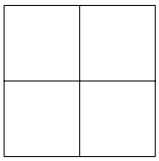
You are given a 2 x n board ($n > 0$) You are interested in the number of **distinct** ways to tile the given board using tiles of dimensions (2 x 1), (1 x 2) and (2 x 2) as shown below:

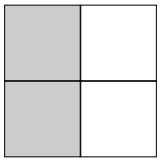

2 x 1

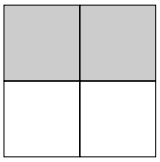

1 x 2


2 x 2

For example, for a 2 x 2 (i.e., $n = 2$) board, there are 3 ways:


Using one 2x2 tile


Using two 2x1 tiles


Using two 1x2 tiles

Requirements: Your solution must be $O(n)$ time. You may use up to $O(n)$ auxiliary space.

Implementation: Limit: 12 lines of code (points deducted if longer). You may use any C, C++, or STL function, algorithm, or data structure that you wish.

```
int number_of_tilings(int n) {
  vector<int> dp_vec(n + 1);
  dp_vec[1] = 1;
  dp_vec[2] = 3;
  for (int i = 3; i <= n; ++i){
    dp_vec[i] = dp_vec[i - 1] + 2 * dp_vec[i - 2];
  }
  return dp_vec[n];
}
```

Fall 2016 Final Exam Question 26: Searching in a Tree

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

A *set* is a data structure that can efficiently (less than $O(n)$ time) see if it contains a given element. Using the following definition of a binary tree node:

```
struct Node {
    int val;
    Node *left;
    Node *right;
};
```

Implement the `exists()` function on a BST-based set.

Requirements: In the average case, your solution must be $O(\log n)$, and you may use up to $O(\log n)$ auxiliary space.

Implementation: Limit: 15 lines of code (points deducted if longer). You may use any C, C++, or STL function/algorithm that you wish.

```
bool exists(Node *node, int val) {
  while (node != nullptr) {
    if (val == node->val) return true;
    if (val < node->val) node = node->left;
    else node = node->right;
  }
  return false;
}
```

Fall 2016 Final Exam Question 27: Searching in a Hash Table

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

A *set* is a data structure that can efficiently (less than $O(n)$ time) see if it contains a given element. Given the following definition of a hash table structure:

```
enum class Status { Empty, Occupied, Deleted };
struct HashTable {
   std::vector<int> buckets;
   std::vector<Status> status;
};
```

Suppose this hash table implements the quadratic probing mechanism for collision resolution. When hashing an integer value `val`, the hash function is simply `val % M`, where `M` represents the size of the hash table. Implement the `exists()` function for a set based on a hash table.

Requirements: Your solution must be $O(n)$ time. You may use $O(1)$ auxiliary space.

Implementation: Limit: 16 lines of code (points deducted if longer). You may use any C, C++, or STL function/algorithm that you wish.

```
bool exists(const HashTable &tbl, int val) {
  if (tbl.buckets.empty()) return false;
  size_t start_index = val % tbl.buckets.size();
  size_t index = start_index, j = 0;
  do {
    if (tbl.status[index] == Status::Empty)
      return false;
    if (tbl.status[index] == Status::Occupied && val == tbl.buckets[index])
      return true;
    ++j;
  } while ((index = (start_index + j * j) % tbl.buckets.size()) != start_index);
  return false;
}
```

Winter 2019 Final Exam Question 25: Zero-Sum Game

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

Write a function that, given a vector of integers, returns **`true`** if the vector contains a *contiguous* subsequence that sums to 0 and **`false`** otherwise. Note that your function should return **`false`** for an empty vector.

For example, if `vector<int> a = {7, 3, 5, -9, 1}`, calling `zero_contiguous_sum(a)` would return **`true`** because the contiguous subsequence `{3, 5, -9, 1}` sums to zero.

Similarly, if `vector<int> b = {3, -2, 4, 2, -3}`, calling `zero_contiguous_sum(b)` would return **`false`** because no contiguous subsequence in `b` sums to zero.

Complexity: $O(n)$ average-case time and $O(n)$ space, where n is the number of elements in the vector.

Implementation: Limit: 15 lines of code (points deducted if longer). You **MAY** use anything in the STL.

```
bool zero_contiguous_sum(vector<int> &nums) {
  int curr_sum = 0;
  unordered_set<int> sums;
  sums.insert(curr_sum);
  for (const auto &num : nums) {
    curr_sum += num;
    if (sums.find(curr_sum) != sums.end()) {
      return true;
    }
    sums.insert(curr_sum);
  }
  return false;
}
```

Winter 2019 Final Exam Question 26: Range Queries

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

You are given two vectors of input, `data` and `queries`. The `data` vector contains **N** unsigned integers, and the `queries` vector contains **Q** query structs. Query structs are defined as follows:

```
struct Query { unsigned int id, start, end; };
```

Write a function that prints the answers to all queries on a single line. An answer to a query is the number of times in the `data` vector that `id` appears in the range $[\texttt{start}, \texttt{end}]$, **inclusive**.

For example, given these **`data`** and **`queries`** vectors, your code should produce the output shown:

`data`: `{5, 4, 3, 5, 3, 3, 2, 1, 3}`
`queries`: `{ {4, 7, 8}, {3, 0, 3}, {3, 0, 8}, {7, 0, 8}, {5, 0, 4} }`

Output: `0 1 4 0 2`

Explanation: The first element of the result vector is `0` because `4` never occurs between indices `7` and `8` (inclusive) in `data`. Similary, the second element of the result vector is `1` because `3` occurs once between indices `0` and `3` (inclusive). Notice that queried items may not appear in the data at all.

Complexity: $O(N^2 + Q)$ average-case time and $O(N^2)$ space.

Implementation: Limit: 25 lines of code (points deducted if longer). You **MAY** use anything in the STL.

Hint: You may want to preprocess the data before answering queries.

Solution to "Range Queries":

```
void range_queries(const vector<unsigned int> &data, const vector<Query> &queries) {
   unsigned int index = 0;
   unordered_map<unsigned int, vector<unsigned int>> ranges;
   for (auto d : data) {
     auto &vec = ranges[d];
     if (vec.empty()) {
       vec.resize(data.size(), 0);
     }
     for (auto &r : ranges) {
       r.second[index] = index ? r.second[index - 1] : 0;
     }
     ++vec[index++];
   }
   for (auto &q : queries) {
     auto &vec = ranges[q.id];
     if (vec.empty()) {
       cout << "0 ";
     }
     else {
       unsigned int A = vec[q.start];
       unsigned int B = vec[q.end];
       cout << B - A << " ";
     }
   }
   cout << endl;
 }
```

Fall 2019 Final Exam Question 25: Hungry Squirrel

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

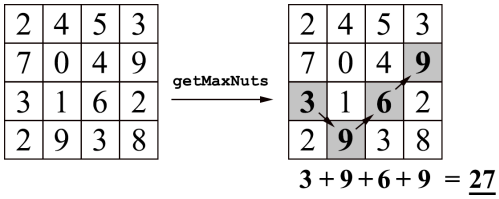
Winter came early, and the squirrels in the diag have been caught off guard! Now they must collect as many nuts as they can for hibernation! The `get_max_nuts()` function should find the maximum amount of nuts a squirrel can collect. You are given a $m \times n$ matrix, `diag`, representing trees in the diag, where m is the number of rows and n is the number of columns. In `diag`, the rows are the outer vectors and the columns are the inner vectors (so index `diag` using `diag[row][col]`). Each cell in the matrix corresponds to a different tree, and its value is a non-negative integer representing the number of nuts that can be found at that tree. The dimensions of the matrix are given to you in `nrow` and `ncol`.

Constraints: The squirrel must start its search for nuts in the first column of the matrix, but it may start at any row. From there on out, the squirrel can only move to the right ($\rightarrow$), diagonally up and to the right ($\nearrow$), or diagonally down and to the right ($\searrow$) from its current position. The search ends when the squirrel reaches the very last column of the matrix.

Complexity: $O(mn)$ time and $O(mn)$ space.

Implementation: Limit: 30 lines of code (points deducted if longer). You may use anything from the STL.

Example: Given the following matrix, `get_max_nuts()` should return `27`, since this is the most nuts the squirrel can collect by following the given movement constraints (starting in the first column and only moving $\rightarrow$, $\nearrow$, or $\searrow$).



```
int get_max_nuts(vector<vector<int>> &diag, int nrow, int ncol) {
  if (nrow == 0 || ncol == 0) return 0;
  vector<vector<int>> treeTable(nrow, vector<int>(ncol, 0));
  for (int col = 0; col < ncol; col++) {
    for (int row = 0; row < nrow; row++) {
      int right = (col == 0) ? 0 : treeTable[row][col - 1];
      int right_up = (row == nrow - 1 || col == 0) ? 0 :
                     treeTable[row + 1][col - 1];
      int right_down = (row == 0 || col == 0) ? 0 :
                       treeTable[row - 1][col - 1];
      treeTable[row][col] = diag[row][col] +
        max(right, max(right_up, right_down));
    }
  }
  int best = treeTable[0][ncol - 1];
  for (int i = 1; i < nrow; i++)
    best = max(best, treeTable[i][ncol - 1]);
  return best;
}
```

Alternative solution to "Hungry Squirrel":

```
int get_max_nuts(vector<vector<int>> &diag, int nrow, int ncol) {
  if (nrow == 0 || ncol == 0) return 0;
  vector<vector<int>> treeTable(nrow, vector<int>(ncol, -1));
  int col = ncol - 1;
  int best = -1;
  for (int row = 0; row < nrow; ++row) {
    treeTable[row][col] = helper(diag, treeTable, row, col);
    best = max(best, treeTable[row][col]);
  }
  return best;
}

int helper(vector<vector<int>> &diag,
           vector<vector<int>> &treeTable, int row, int col) {
  if (col == 0)
    return diag[row][col];
  int right = 0, right_up = 0, right_down = 0;
  if (treeTable[row][col - 1] == -1)
    treeTable[row][col - 1] = helper(diag, treeTable, row, col - 1);
  right = treeTable[row][col - 1];
  if (row != diag.size() - 1) {
    if (treeTable[row + 1][col - 1] == -1)
      treeTable[row + 1][col - 1] = helper(diag, treeTable, row + 1, col - 1);
    right_up = treeTable[row + 1][col - 1];
  }
  if (row != 0) {
    if (treeTable[row - 1][col - 1] == -1)
      treeTable[row - 1][col - 1] = helper(diag, treeTable, row - 1, col - 1);
    right_down = treeTable[row - 1][col - 1];
  }
  treeTable[row][col] = diag[row][col] + max(right, max(right_up, right_down));
  return treeTable[row][col];
}
```

Fall 2019 Final Exam Question 26: Minimizing Layovers

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

Implement the function `fewest_layovers()`, which returns the fewest number of intermediate flight connections between two airports. You are given the starting and ending airports as three-character strings and a vector of available flights, where `flights[i][0]` is the origin of flight `i` and `flights[i][1]` is the destination of flight `i`. You may assume that any flight can be taken at any time, and that all flights in `flights` are unique. If no path exists from your origin to your destination, you should return `-1`.

Complexity: $O(n)$ time and $O(n)$ memory, where n is the number of flights in the `flights` vector.

Implementation: Limit: 30 lines of code (points deducted if longer). You may use anything from the STL.

Example 1: Given origin = DTW, dest = SEA,

flights = [[ATL, JFK], [ORD, LAX], [MIA, PHL], [DTW, ATL], [SEA, ORD], [DTW, ORD], [SFO, SEA], [ORD, ATL], [LAX, SEA], [ATL, ORD], [LAX, ATL], [ORD, JFK]]

you should return `2`, since there are two intermediate airports along the path from DTW to SEA with the fewest layovers (DTW → **ORD** → **LAX** → SEA).

Example 2: Given origin = DTW and dest = ORD and the same flights vector as example 1, you should return `0` because no layovers are needed to fly from DTW to ORD (there exists a direct flight).

Example 3: Given origin = DTW, dest = SFO, flights = [[DTW, ORD], [ORD, ATL], [ATL, DTW]] you should return `-1`, since there is no path from DTW to SFO using the flights available.

```
int fewest_layovers(vector<vector<string>> &flights, string origin, string dest) {
  unordered_map<string, vector<string>> graph;
  unordered_set<string> visited;
  for (vector<string> &flight : flights)
    graph[flight[0]].push_back(flight[1]);
  queue<pair<string, int>> bfs;
  bfs.push({ origin, 0 });
  while (!bfs.empty()) {
    string depart = bfs.front().first;
    int distance = bfs.front().second;
    if (graph.count(depart) && !visited.count(depart)) {
      visited.insert(depart);
      for (string &arrive : graph[depart]) {
        if (arrive == dest)
          return distance;
        bfs.push({ arrive, distance + 1 });
      }
    }
    bfs.pop();
  }
  return -1;
}
```

Winter 2020 Final Exam Question 25: Super DP Bros.

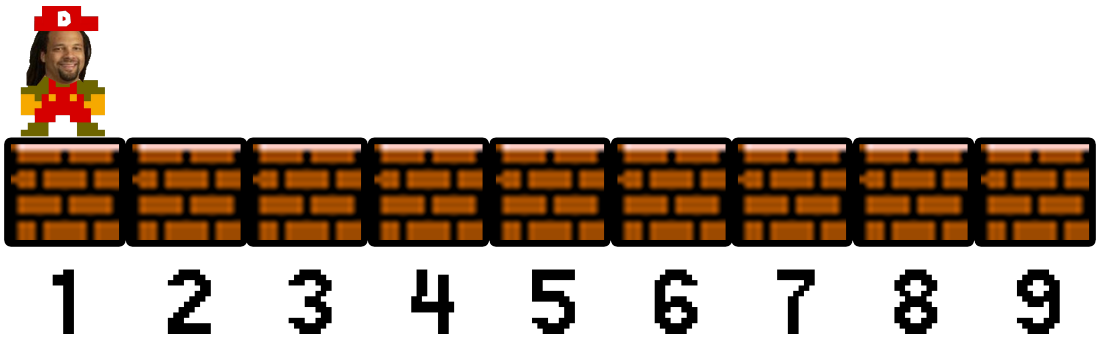
For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

The video game company that you work for, *Cin-tendo*, is developing an exciting new game for release next year: *Super DP Bros.*! This game details the lives of two Italian plumbers, Dario and Paoluigi (the DP Brothers) and their quest to save Princess Pear from a group of evil lobsters who are holding her hostage in a faraway castle codenamed "EECS280."

In the final level of this game, the evil lobsters require Dario to complete a challenge: given an integer M and a one-dimensional map of N tiles numbered 1 to N, Dario must discover the total number of ways he can make **exactly** M moves and end up back at his starting position S. The following three actions count as a single move:

1. Moving left by 1 tile (**L**)
2. Moving right by 1 tile (**R**)
3. Jumping, which does **not** change Dario's horizontal position (**J**)

For example, consider the following map where $N = 9$:



If the evil lobsters were to give Dario the values $M = 4$ and $S = 1$, Dario would need to return the value **9**, since there are 9 possible sequences of length 4 that Dario can make if he wants to start and end on tile 1: **JJJJ**, **JJRL**, **JRJL**, **JRLJ**, **RLRL**, **RRLL**, **RLJJ**, **RJJL**, and **RJLJ**. The tile positions on the map are 1-indexed, and Dario cannot jump off the edge or loop around.

Dario doesn't have much time, and he needs your help! Implement the `count_possible_moves()` function, which takes in three values, `num_moves`, `num_tiles`, and `start_pos`, and returns the total number of ways Dario can exhaust all `num_moves` moves and finish on tile `start_pos`. The fate of Princess Pear lies in your hands!

Complexity: $O(MN)$ time and auxiliary space, where $M =$ `num_moves` and $N =$ `num_tiles`. You may assume that `num_moves` > 0 and `num_tiles` > 1, and that `start_pos` will always be a valid position on the map.

Implementation: Limit: 30 lines of code (points deducted if longer). You may use anything from the STL.

Implement your solution to "Super DP Bros." below:

```
int count_possible_moves(int num_tiles, int num_moves, int start_pos) {
  vector<vector<int>> memo =
    vector<vector<int>>(num_moves + 1, vector<int>(num_tiles + 2, 0));
  memo[0][start_pos] = 1;
  for (int m = 1; m <= num_moves; ++m)
    for (int n = 1; n <= num_tiles; ++n)
      memo[m][n] = memo[m - 1][n - 1] + memo[m - 1][n] + memo[m - 1][n + 1];
  return memo[num_moves][start_pos];
}
```

Alternative solution to "Super DP Bros.":

```
int count_possible_moves(int num_tiles, int num_moves, int start_pos) {
  vector<vector<int>> memo(num_moves + 1, vector<int>(num_tiles + 2, -1));
  for (int i = 0; i <= num_tiles; ++i)
    memo[0][i] = (i == start_pos) ? 1 : 0;
  return helper(num_tiles, num_moves, start_pos, memo);
}

int helper(int num_tiles, int num_moves, int start_pos,
                                         vector<vector<int>> &memo) {
  if (start_pos == 0 || start_pos == num_tiles + 1)
    return 0;
  if (memo[num_moves - 1][start_pos - 1] == -1)
    memo[num_moves - 1][start_pos - 1] =
      helper(num_tiles, num_moves - 1, start_pos - 1, memo);
  if (memo[num_moves - 1][start_pos] == -1)
    memo[num_moves - 1][start_pos] =
      helper(num_tiles, num_moves - 1, start_pos, memo);
  if (memo[num_moves - 1][start_pos + 1] == -1)
    memo[num_moves - 1][start_pos + 1] =
      helper(num_tiles, num_moves - 1, start_pos + 1, memo);
  return memo[num_moves - 1][start_pos - 1] +
         memo[num_moves - 1][start_pos] +
         memo[num_moves - 1][start_pos + 1];
}
```

Winter 2020 Final Exam Question 26: Descendant Averages

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

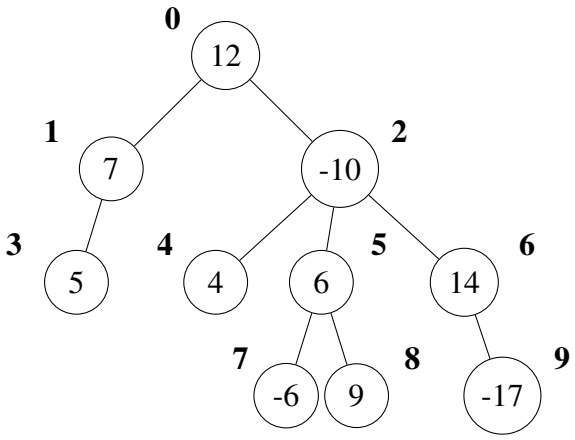
You are given the root node of a tree whose nodes can have **more than two children**. Each node is defined as follows:

```
struct Node {
   double val;
   vector<Node *> children;
   Node(double x) : val{x} {}
};
```

The children of a node are stored in its `children` vector in order from left to right. If a node has no children, its vector will be empty.

For this problem, the **index** of a node is defined as the node's position in a level-order traversal. The first node in a level-order traversal would have an index of 0, the second node in a level-order traversal would have an index of 1, and so on and so forth. The **descendant average** of a node is defined as the average value of the node and all of its descendants.

For example, consider the following tree. The index of each node is shown to the side in bold:



Here, the descendant average of the node at index 5 is:

$$\frac{\text{sum of all elements in subtree rooted at index 5}}{\text{size of subtree rooted at index 5}} = \frac{6+(-6)+9}{3} = \frac{9}{3} = 3$$

Implement the `query_descendant_averages()` function, which takes in the root of a tree (`root`) and a vector of q indices (`indices`) and returns a vector of doubles that stores the descendant average of the node located at each index of the `indices` vector. You may assume that all indices in the vector are valid, and that root is never `nullptr`. You should not modify the contents of the tree.

Example: Given the above tree and the following vector:

```
indices = [8, 2, 6, 0, 5]
```

You would return a vector with the following values:

```
result = [9, 0, -1.5, 2.4, 3]
```

This is because

- the node at index 8 has a descendant average of $\frac{9}{1} = 9$
- the node at index 2 has a descendant average of $\frac{-10+4+6+14+(-6)+9+(-17)}{7} = \frac{0}{7} = 0$
- the node at index 6 has a descendant average of $\frac{14+(-17)}{2} = \frac{-3}{2} = -1.5$
- the node at index 0 has a descendant average of $\frac{12+7+(-10)+5+4+6+14+(-6)+9+(-17)}{10} = \frac{24}{10} = 2.4$
- the node at index 5 has a descendant average of $\frac{6+(-6)+9}{3} = \frac{9}{3} = 3$

Complexity: $O(n + q)$ time and $O(n)$ auxiliary space, where n is the number of nodes in the tree.

Implementation: Limit: 30 lines of code (points deducted if longer). You may use anything from the STL.

```
vector<double> query_descendant_averages(Node *root, const vector<int> &indices) {
  vector<Node *> level_position;
  unordered_map<Node *, double> avgs;
  queue<Node *> bfs;
  bfs.push(root);
  while (!bfs.empty()) {
    Node *next = bfs.front();
    bfs.pop();
    level_position.push_back(next);
    for (Node *child : next->children)
      bfs.push(child);
  }
  calc_averages(root, avgs);
  vector<double> result;
  for (int index : indices)
    result.push_back(avgs[level_position[index]]);
  return result;
}

pair<double, int> calc_averages(Node *current,
                                unordered_map<Node *, double> &avgs) {
  double current_sum = current->val;
  int size = 1;
  for (Node *child : current->children) {
    pair<double, int> res = calc_averages(child, avgs);
    current_sum += res.first;
    size += res.second;
  }
  avgs[current] = current_sum / size;
  return { current_sum, size };
}
```

Alternative solution to "Descendant Averages":

```
vector<double> query_descendant_averages(Node *root, const vector<int> &indices) {
   vector<Node *> level_position;
   queue<Node *> bfs;
   bfs.push(root);
   while (!bfs.empty()) {
     Node *next = bfs.front();
     bfs.pop();
     level_position.push_back(next);
     for (Node *child : next->children)
       bfs.push(child);
   }
   unordered_map<Node *, pair<double, int>> subtrees;
   for (auto it = level_position.rbegin(); it != level_position.rend(); ++it) {
     Node *current_node = *it;
     subtrees[current_node] = { current_node->val, 1 };
     for (Node *child : current_node->children) {
       subtrees[current_node].first += subtrees[child].first;
       subtrees[current_node].second += subtrees[child].second;
     }
   }
   vector<double> result;
   for (int index : indices) {
     auto [sum, size] = subtrees[level_position[index]];
     result.push_back(sum / size);
   }
   return result;
 }
```

Winter 2020 Final Exam Question 27: K-Away

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

You are given a non-negative integer k and the root of a binary tree, where each node is defined as follows:

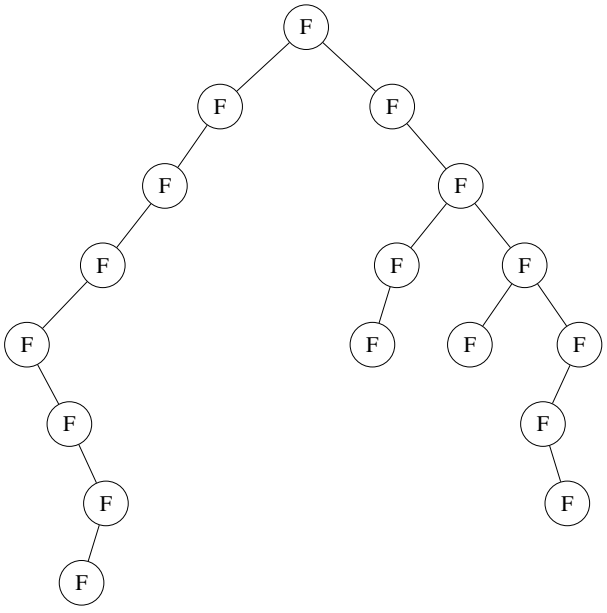
```
struct Node {
   Node *left;
   Node *right;
   bool selected;
};
```

Each node can either be **selected** (`selected == true`), or **unselected** (`selected == false`). Every node in the binary tree starts off **unselected**.

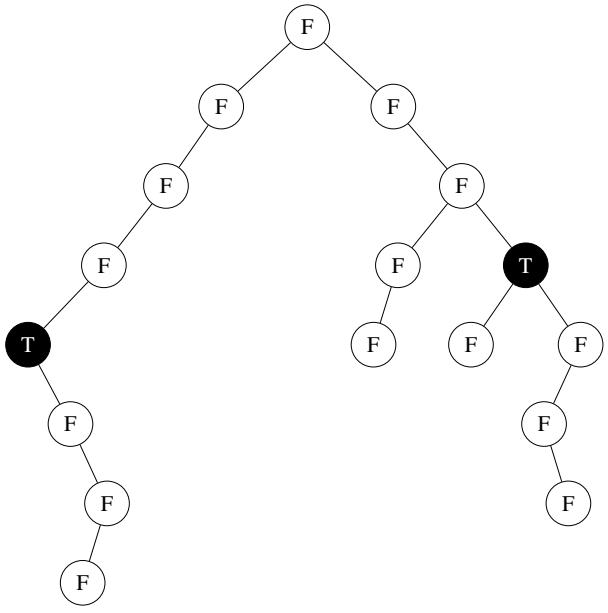
Implement the `k_away()` function, which returns the minimum number of nodes that need to be selected so that every node in the tree is at most k nodes away from a selected node. The function should also modify the nodes that are selected so that their `selected` values are **`true`**. The tree you are given may have any size, so make sure to consider all cases!

Example: If you are given the binary tree on the left (where **F** is **`false`** and **T** is **`true`**) and $k = 3$, the `k_away()` function should set the `selected` variable of the two darkened nodes to **`true`** and return the value **2**. This is because a minimum of 2 nodes need to be selected for every node in the tree to be at most $k = 3$ nodes away from a selected node. If there are multiple ways to select the nodes, you may select any combination that works.

Before `k_away(root, 3)` is called:



After `k_away(root, 3)` is called:



Complexity: $O(n)$ time and $O(n)$ auxiliary space, where n is the number of nodes in the tree.

Implementation: Limit: 25 lines of code (points deducted if longer). You may use anything from the STL.

Implement your solution to "K-Away" below.

Note: *This question is challenging, so if you are stuck, don't spend too much time on it! Questions 25 and 26 ("Super DP Bros." and "Descendant Averages") were the two main problems of the Winter 2020 final exam, and they are more representative of what you might see on your own final exam. That being said, you've learned all the material needed to solve this problem.*

```
unsigned int k_away(Node *root, int k) {
  unsigned int total = 0;
  if (traverse(root, k, total) < 0 && k != 0 && root) {
    root->selected = true;
    return total + 1;
  }
  return total;
}

int traverse(Node *root, int k, unsigned int &total) {
  if (!root) return 0;
  int left = traverse(root->left, k, total);
  int right = traverse(root->right, k, total);
  if (left == -k || right == -k) {
    ++total;
    root->selected = true;
    return k;
  }
  if (left + right > 0) {
    return max(left, right) - 1;
  }
  return min(left, right) - 1;
}
```

Alternative solution to "K-Away":

```
unsigned int k_away(Node *root, int k) {
  unsigned int count = 0;
  int root_val = helper(root, k, count);
  if (root_val > 0) {
    root->selected = true;
    ++count;
  }
  return count;
}

int helper(Node *root, const int k, unsigned int &count) {
  if (!root) return 0;
  int left = helper(root->left, k, count);
  int right = helper(root->right, k, count);
  int result = 0;
  if (right + left <= -1) {
    result = min(left, right) + 1;
  }
  else {
    result = max(left, right) + 1;
  }
  if (result == k + 1) {
    root->selected = true;
    ++count;
    return -k;
  }
  return result;
}
```

Spring 2020 Final Exam Question 25: Social Distance Traveling

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

Traveling during the era of COVID-19 is full of risk. In the event that a trip could not be postponed, the safest travel would avoid the areas with the highest number of reported cases. A 2D-grid has been overlaid on a (cartographic) map showing the total number of reported cases in each square area. In an attempt to protect yourself, you must find the path that goes through adjacent areas and completes the required travel while entering a collection of grid spaces that collectively sum to the lowest total number of reported cases.

Given a grid with M rows and N columns, find the path across the map that minimizes possible exposure – that is, it encounters the smallest sum of individual reported cases. **Travel is from North to South (top row to bottom row), and moves can only be made horizontally left or right, or vertically down. The path you must find may start anywhere in the top row and end anywhere in the bottom row, and be of any length, so long as the total number of encountered cases is minimized.** The $M \times N$ grid may be any size $M, N \geq 1$, with all non-negative values inside. Return the total number of cases encountered as an integer (the complete path does not need to be returned or printed).

Example: A 5×5 grid given with the path that encounters the lowest number of cases already highlighted, totaling 98.

13	**20**	63	53	81
67	**9**	80	70	73
23	**34**	75	50	52
10	97	42	12	30
2	15	11	95	33

Complexity: Your solution should be $O(MN)$ in runtime and use no more than $O(N)$ auxiliary space.

Implementation: Use the space below as work space, then write your solution NEATLY on the next page. You **may** use anything from the STL. Limit: 20 lines of code (points deducted for each line over).

Implement your solution to "Social Distance Traveling" below:

```
int covid_travel(vector<vector<int>> &grid) {
  size_t M = grid.size();
  size_t N = grid.empty() ? 0 : grid[0].size();
  for (int row = 1; row < M; ++row) {
    vector<int> current_row = grid[row];
    grid[row][0] += grid[row - 1][0];
    if (N > 1) {
      grid[row][N - 1] += grid[row - 1][N - 1];
    }
    for (int col = 1; col < N; ++col) {
      int cell = current_row[col];
      grid[row][col] = min(grid[row - 1][col] + cell, grid[row][col - 1] + cell);
    }
    for (int col = N - 2; col >= 0; --col) {
      int cell = current_row[col];
      grid[row][col] = min(grid[row][col], grid[row][col + 1] + cell);
    }
  }
  int min_risk = grid[M - 1][0];
  for(int col = 1; col < N; ++col) {
    min_risk = min(min_risk, grid[M - 1][col]);
  }
  return min_risk;
}
```

Spring 2020 Final Exam Question 26: Rotated Strings Subset

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

A character is said to be rotated by an integer $0 \leq k < 26$, if it is advanced by k characters (wrapping from `'z'` back to `'a'`, if necessary). For example, `'a'` rotated by four is `'e'`, and `'y'` rotated by four is `'c'`. If S is a string of lowercase ASCII characters (`'a'-'z'`) of length M, then T is a "k-rotation" of S if and only if T is of length M and for all indices $i < M$, $T[i]$ = ($S[i]$ rotated by k). For example:

- hal $\rightarrow$ ibm: 1-rotation
- hal $\rightarrow$ pit: 8-rotation
- buf $\rightarrow$ hal: 6-rotation

Strings S and T are said to be *rotationally equivalent* if there is some $0 \leq k < 26$ where T is a k-rotation of S, and S is the $(26 - k)$-rotation of T. Given a vector S, of N strings including only lowercase characters (ASCII `'a'-'z'`) with an average string length M, return any largest subset of S of rotationally equivalent strings (in any order).

Example: Given S = `{"hal", "them", "uifn", "fore", "pop", "shed", "ibm", "canes", "maple", "pit", "buf"}`, `roteq_subset()` returns a subset with the members `{"hal", "ibm", "pit", "buf"}`. Other subsets exist, but this is the largest.

Complexity: Your solution should be $O(MN)$ in runtime and use no more than $O(MN)$ auxiliary space.

Implementation: Write your solution NEATLY in the space below. You **may** us anything from the STL. Limit: 20 lines of code (points deducted for each line over).

```
vector<string> roteq_subset(const vector<string> &strs) {
  unordered_map<string, vector<string>> subsets;
  for (const string &str: strs) {
    string key = str;
    int k = str.empty() ? 0 : 'a' - str[0];
    transform(begin(key), end(key), begin(key), [k](char c) {
      return 'a' + (c - 'a' + k + 26) % 26;
    });
    subsets[key].push_back(str);
  }
  auto it = max_element(begin(subsets), end(subsets), [](auto &a, auto &b) {
    return a.second.size() < b.second.size();
  });
  return it->second;
}
```

Fall 2020 Final Exam Question 25: Peaky Binders

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

Suppose we represent a *binder* as a vector of integers corresponding to the page numbers it contains. You are given a shelf of binders (2D vector of integers) containing class notes. Your job is to implement a function that, given a shelf of binders, finds the minimal cost of combining all unique pages into a single binder of notes. **You can only combine binders that are adjacent to each other.**

The cost of each combination is the maximum sum of the non-duplicate page numbers of the two binders you are trying to combine (see the example to get a better understanding). Luckily, you already have a function to combine binders and compute the cost.

```
/*************************************************************
* Modifies: output contains result of combining first and second  *
* Returns: cost of carrying out the combination                   *
* Runtime: O(m), where m is the maximum page number               *
* Auxiliary Memory: O(m)                                          *
*************************************************************/
// You do NOT need to implement this function!
// It has already been implemented for you.
int combine(vector<int> &first, vector<int> &second, vector<int> &output);
```

For example, suppose you had three binders: (1, 2, 4, 5) (1, 3, 5) (2, 4, 5). We may consider two options:

- Combine (1, **2**, **4**, 5) and (1, **3**, 5) into (1, 2, 3, 4, 5). Note that pages 2 and 4 are non-duplicate in the first binder, and page 3 is non-duplicate in the second binder. Then, we take max(2 + 4, 3) to get 6 as the cost. This leaves us with two binders on the shelf: (1, 2, 3, 4, 5) and (2, 4, 5). We then combine the binders (**1**, 2, **3**, 4, 5) and (2, 4, 5) with a cost of max(1 + 3, 0) = 4 (since pages 1 and 3 are non-duplicate in the first binder, and there are no non-duplicate pages in the second binder). Therefore, the total cost would be 6 + 4 = 10.
- Combine (**1**, **3**, 5) and (**2**, **4**, 5) into (1, 2, 3, 4, 5) with a cost of max(1 + 3, 2 + 4) = 6 (since pages 1 and 3 are non-duplicate in the first binder, and pages 2 and 4 are non-duplicate in the second binder). This leaves us with two binders on the shelf: (1, 2, 4, 5) and (1, 2, 3, 4, 5). We then combine the binders (1, 2, 4, 5) and (1, 2, **3**, 4, 5) with a cost of 3. Therefore, the total cost would be 6 + 3 = 9.

The minimal cost across the two options is 9, so your implementation of the `min_combine_cost()` function should return 9.

```
vector<vector<int>> binders = { {1, 2, 4, 5},
                                {1, 3, 5},
                                {2, 4, 5}   };
cout << min_combine_cost(binders) << endl;  // prints 9
```

You may assume that page numbers are ≥ 1 and sorted. You may also assume that there is at least one binder in the input 2D vector, and that there are no duplicate pages in any individual binder. The binders we give you are **not** guaranteed to include all pages in the range $[1, M]$, where M is the largest page

number — there may be page numbers that are missing from *all* binders in the `binders` vector we give you. In this scenario, your final binder does **not** need to include these missing pages.

Complexity: Your solution should run in $O(N^3M)$ time and use no more than $O(N^2M)$ auxiliary space, where N is the number of binders and M is the largest page number.

Implementation: Write your solution NEATLY in the space below. You **may** use anything from the STL. Limit: 30 lines of code (points deducted for each line over).

```
int min_combine_cost(const vector<vector<int>> &binders) {
  // 2D vector of output vectors, 3D vector of ints
  vector<vector<vector<int>>> merges(binders.size(),
      vector<vector<int>>(binders.size(), vector<int>()));
  vector<vector<int>> memo(binders.size(), vector<int>(binders.size(), -1));
  // base cases
  for (int i = 0; i < binders.size(); ++i) {
    // combining binder with itself has cost of 0
    memo[i][i] = 0; // unnecessary if we initialize every memo element to be 0
    // but helps to explicitly see base case and when memo is changed
    // combining binder with itself gives same binder
    merges[i][i] = binders[i];
  }
  // goal is to fill in memo strictly above the diagonal,
  // starting at the second to last row and going up
  // and going from left to right across columns.
  // O(n)
  for (int row = (int)binders.size() - 2; row >= 0; --row) {
    // O(n)
    for (int col = row + 1; col < (int)binders.size(); ++col) {
      int min_cost = std::numeric_limits<int>::max();
      // O(n)
      vector<int> output;
      for (int i = row; i < col; ++i) {
        int cost = memo[row][i] + memo[i + 1][col];
        // O(m)
        cost += combine(merges[row][i], merges[i + 1][col], output);
        min_cost = std::min(min_cost, cost);
      }
      // O(m)
      merges[row][col] = output;
      memo[row][col] = min_cost;
    }
  }
  // O(n^3*m) time in total
  // outputs now takes O(n^2*m) space
  return memo[0][binders.size() - 1];
} // 16 lines not counting function signature
```

Alternative solution to "Peaky Binders":

```
int combine_cost(vector<vector<int>> &memo, vector<vector<vector<int>>> &merges,
                 const vector<vector<int>> &binders, int begin, int end){
  if (memo[begin][end] != -1){
    return memo[begin][end];
  }
  // base cases
  if (begin == end){
    // O(m)
    // combining a binder with itself gives same binder
    merges[begin][end] = binders[begin];
    // combining a binder with itself has cost of 0
    memo[begin][end] = 0;
    return 0;
  }
  int min_cost = std::numeric_limits<int>::max();
  // O(n) loop
  for (int i = begin; i < end; ++i){
    // treated as O(1) time
    int cost = combine_cost(memo, merges, binders, begin, i);
    // treated as O(1) time
    cost += combine_cost(memo, merges, binders, i + 1, end);
    // O(m) time
    cost += combine(merges[begin][i], merges[i+1][end], merges[begin][end]);
    min_cost = std::min(min_cost, cost);
  }
  // O(nm) to find cost for each cell, ~n^2/2 = O(n^2) cells to calculate in
  // memo -> O(n^3*m) time

  memo[begin][end] = min_cost;
  return min_cost;
} // 14 lines

int min_combine_cost(const vector<vector<int>> &binders){
  // O(n^2) space for memo
  vector<vector<int>> memo(binders.size(), vector<int>(binders.size(), -1));
  // O(n^2*m) space for merges
  vector<vector<vector<int>>> merges(binders.size(),
      vector<vector<int>>(binders.size(), vector<int>()));
  return combine_cost(memo, merges, binders, 0, (int)binders.size() - 1);
} // 3 lines not counting function signature
```

Fall 2020 Final Exam Question 26: Geeks and Dragons

For the following problem, you will be able to submit your solution to Gradescope for automatic feedback. Please see Canvas to download the starter files for this problem.

You are the ruler of a kingdom looking for a party of heroes to slay the mighty two-headed bitdragon. You can afford to hire **a single party** of at most k heroes, and you want to hire the most experienced heroes possible. The heroes in the kingdom are grouped into *parties*, which cannot be broken up (i.e., if you hire a member of a party, every member of that party must also be hired). **If any two heroes have worked together in the past, then they are automatically part of the same party.** Each hero is represented as a node in a graph and has a non-negative number representing their experience level.

Your job is to implement a function that is passed in

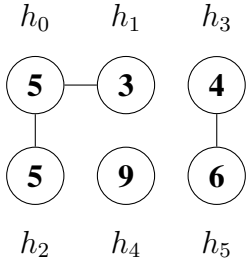
- a vector of experience levels for each hero (experience)
  - e.g., if `experience = {5, 3, 5, 4, 9, 6}`, hero #0 has experience level 5, hero #1 has experience level 3, ..., and hero #5 has experience level 6
- a 2D vector that stores the work history of each of the heroes (`partners`)
  - e.g., if `partners = { {1, 2}, {0}, {0}, {5}, {}, {3} }`, hero #0 has worked with heroes #1 and #2, ..., hero #4 has not worked with anyone (a solo adventurer), and hero #5 has worked with hero #3
- a value for k, the maximum number of heroes you can hire

and outputs the total experience of the party of heroes with size $\leq k$ that has the most experience. If there exists no parties with size $\leq k$, you would return 0 (since you cannot hire anyone).

Example: Given

- `experience = {5, 3, 5, 4, 9, 6}`
- `partners = { {1, 2}, {0}, {0}, {5}, {}, {3} }`
- $k = 2$

the function should return 10, since the most experienced group that is of size $\leq k$ is the group containing heroes #3 and #5, which has a combined experience of 4 + 6 = 10. This example is represented using the graph below (where h_n represents hero n):




```
vector<int> experience = {5, 3, 5, 4, 9, 6};
vector<vector<int>> partners = { {1, 2}, {0}, {0}, {5}, {}, {3} };
cout << hire_heroes(experience, partners, 2) << endl;  // prints 10
```

Note: It is possible for two heroes to be part of the same party without having worked together in the past (e.g., heroes #1 and #2 in the above example).

Complexity: Your solution should run in $O(V + E)$ time and use no more than $O(V)$ auxiliary space, where V is the number of heroes (vertices) and E is the number of partnerships between heroes (edges).

Implementation: Write your solution NEATLY in the space below. You **may** use anything from the STL. Limit: 30 lines of code (points deducted for each line over).

```
int hire_heroes(const vector<int> &experience,
                const vector<vector<int>> &partners, int k) {
  // Keep track of visited nodes
  vector<bool> visited(experience.size(), false);
  // For each node, bfs for connected components
  int max_sum = 0;
  for (size_t i = 0; i < partners.size(); ++i) {
    if (!visited[i]) {
      int sum = 0, count = 0;
      visited[i] = true;
      queue<int> bfs;
      bfs.push(int(i));
      while (!bfs.empty()) {    // Inner loop builds connected component
        ++count;
        int connected = bfs.front();
        sum += experience[connected];
        bfs.pop();
        for (auto next : partners[connected]) {
          if (!visited[next]) {
            bfs.push(next);
            visited[next] = true;
          }
        }
      }
      if (sum > max_sum && count <= k) {
        max_sum = sum;
      }
    }
  }
  // Return the most experienced team
  return max_sum;
}
```

Alternative solution to "Geeks and Dragons" (DFS instead of BFS):

```
int hire_heroes(const vector<int> &experience,
                const vector<vector<int>> &partners, int k) {
  // Keep track of visited nodes
  vector<bool> visited(experience.size(), false);
  // For each node, bfs for connected components
  int max_sum = 0;
  for (size_t i = 0; i < partners.size(); ++i) {
    if (!visited[i]) {
      int sum = 0, count = 0;
      visited[i] = true;
      stack<int> dfs;
      dfs.push(int(i));
      while (!dfs.empty()) {    // Inner loop builds connected component
        ++count;
        int connected = dfs.top();
        sum += experience[connected];
        dfs.pop();
        for (auto next : partners[connected]) {
          if (!visited[next]) {
            dfs.push(next);
            visited[next] = true;
          }
        }
      }
      if (sum > max_sum&& count <= k) {
        max_sum = sum;
      }
    }
  }
  // Return the most experienced team
  return max_sum;
}
```

Alternative solution to "Geeks and Dragons" (Union-Find):

```
int hire_heroes(const vector<int>& experience,
                const vector<vector<int>>& partners, int k) {
  vector<int> reps(partners.size());
  iota(reps.begin(), reps.end(), 0);
  for (int hero = 0; hero < partners.size(); ++hero) {
    for (int other : partners[hero]) {
      union_set(reps, hero, other);
    }
  }
  unordered_map<int, vector<int>> parties;
  for (size_t i = 0; i < reps.size(); ++i) {
    reps[i] = find_set(reps, i);
    parties[reps[i]].push_back(i);
  }
  int max_sum = 0;
  for (const auto& party : parties) {
    if (party.second.size() <= k) {
      int sum = accumulate(party.second.begin(), party.second.end(), 0,
                  [&experience](const int accum, size_t val) {
                    return accum + experience[val];
                  });
      if (sum > max_sum) {
        max_sum = sum;
      }
    }
  }
  return max_sum;
}

int find_set(vector<int>& reps, int x) {
  return (x == reps[x]) ? x : reps[x] = find_set(reps, reps[x]);
}

void union_set(vector<int> &reps, int a, int b) {
  reps[find_set(reps, a)] = find_set(reps, b);
}
```