Chapter 22 Practice Exercises

Disclaimer: These practice questions are not official and may not have been vetted for course material. You may use these for practice, but you should prioritize official resources from current staff for a more accurate depiction of what you need to know for assignments and exams.

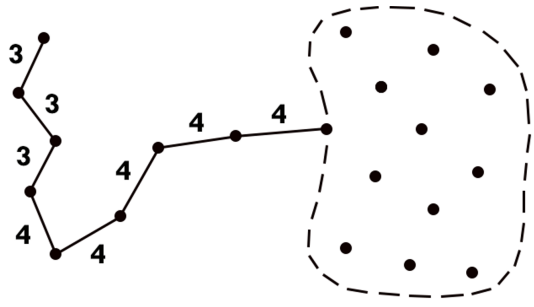
1. Which of the following is a constraint satisfaction problem and not an optimization problem?
   - **A)** Finding the route with the minimum distance to New York City
   - **B)** Finding the minimum spanning tree (MST) of a graph
   - **C)** Finding the shortest path out of a maze
   - **D)** Finding a route between two classrooms without going outside
   - **E)** None of the above

2. Which of the following statements is/are **TRUE**?
   - **I.** Branch and bound is an algorithmic strategy that can be used to find the optimal solution to a constraint satisfaction problem.
   - **II.** Branch and bound algorithms immediately stop exploring the search space once the first solution satisfying all constraints is found.
   - **III.** The efficiency and correctness of a branch and bound algorithm may depend on the effectiveness of the pruning function.

   - **A)** I only
   - **B)** III only
   - **C)** I and II only
   - **D)** I and III only
   - **E)** I, II, and III

3. You are currently running the traveling salesperson problem (TSP) 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 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

4. 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

5. Which of the following statements is **TRUE**?
   - **A)** Heuristics are algorithms that can be used to calculate an optimal solution very quickly
   - **B)** Heuristics often have time complexities that are much more optimal 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

6. Suppose you want to visit the computer science buildings of every major university in the United States, and you want to find the shortest route that will allow you to visit each campus exactly once before returning to your starting location. What is the worst-case time complexity of using brute force to calculate this shortest path, assuming that you have n campuses that you want to visit?
   A) $\Theta(2^n)$
   B) $\Theta(n2^n)$
   C) $\Theta(n^2)$
   D) $\Theta(n^n)$
   E) $\Theta(n!)$

7. Which of the following statements is/are **TRUE**?
   I. 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.
   II. The worst-case time complexity of running the traveling salesperson problem using a branch and bound algorithm is better than the worst-case time complexity of solving the traveling salesperson problem using brute force.
   III. When solving the traveling salesperson problem using branch and bound, overestimating the cost of extending a partial solution could cause you to fail to find the optimal solution.

   A) I only
   B) II only
   C) III only
   D) II and III only
   E) I, II, and III

8. Which of the following statements is/are **TRUE**?
   I. When solving a minimization problem using branch and bound, a branch should be pruned if the upper bound is less than the lower bound.
   II. When solving a maximization problem using branch and bound, a branch should be pruned if the upper bound is greater than the lower bound.
   III. When solving a maximization problem using branch and bound, the upper bound is calculated by taking the current partial solution and adding it to an underestimate of the value obtainable from completing the partial solution.

   A) I only
   B) II only
   C) I and III only
   D) II and III only
   E) I, II, and III

9. You are given two fully connected graphs, one with four vertices and the other with five vertices. How many more distinct Hamiltonian cycles exist in the graph with five vertices, compared to the one with four vertices? That is, if the graph with four vertices has h_4 distinct Hamiltonian cycles and the graph with five vertices has h_5 distinct Hamiltonian cycles, what is the value of $h_5 - h_4$?
   A) 24
   B) 48
   C) 72
   D) 96
   E) 120

10. Which of the following problems can be solved using a backtracking algorithm?
   I. Finding the minimum spanning tree of a graph
   II. Determining if a graph can be colored with n different colors without any adjacent vertices sharing the same color
   III. Implementing a Sudoku solver

   A) I only
   B) II only
   C) III only
   D) II and III only
   E) I, II, and III

11. Which of the following statements is true about branch and bound, but not necessarily true for backtracking?
   A) The algorithm can be used to solve problems involving constraints
   B) The algorithm will return an optimal solution
   C) The algorithm typically improves on the runtime performance of brute force
   D) The algorithm prunes partial solutions that cannot yield a valid solution
   E) None of the above

12. You are using a branch and bound algorithm to determine the shortest route you can take to visit five buildings on campus. The following table stores the distances between each pair of buildings that you want to visit:

	BBB	**DOW**	**EECS**	**FXB**	**GGBL**
BBB	0	8	19	30	17
DOW	8	0	20	28	13
EECS	19	20	0	16	12
FXB	30	28	16	0	15
GGBL	17	13	12	15	0

The shortest complete solution you have found so far has a total distance of 71. You begin looking down a new branch: BBB → FXB → DOW → GGBL. Should you prune this branch?

A) Yes, the partial branch has the same length as the shortest complete path seen so far
B) Yes, the partial branch is longer than the shortest complete path seen so far
C) Yes, the partial branch is shorter than the shortest complete path seen so far
D) No, the branch has the same length as the shortest complete path seen so far
E) No, the branch is shorter than the shortest complete path seen so far

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

```
std::vector<std::vector<int32_t>> k_combinations(int32_t n, int32_t k);
```

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

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

14. A valid IP address consists of exactly four integers separated by single dots. Each integer must have a value between 0 and 255 (inclusive) and cannot have any leading zeros (other than 0 itself). For example, "183.70.2.81" is a valid IP address, but "18.3.70.281" and "18.37.028.1" are not. Given a string `digits` containing only digits, return all possible IP addresses that can be formed by inserting dots into `digits`. You are not allowed to reorder or remove any digits, and you may return the valid IP addresses in any order.

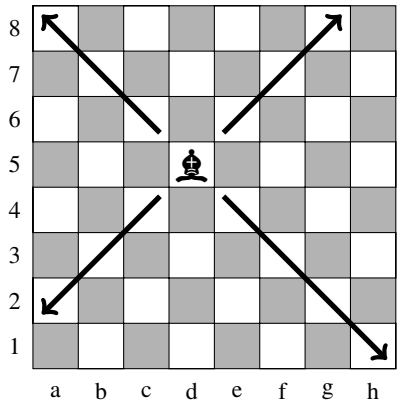
```
std::vector<std::string> get_valid_ips(const std::string& digits);
```

Example: Given `digits` = `"183203281"`, you would return the following IP addresses (in any order):

```
["183.20.32.81", "183.203.2.81", "183.203.28.1"]
```

Hint: You can use `std::stoi()` to parse a string into an integer.

15. In the game of chess, the *bishop* is a piece that can only move diagonally. The movement capability of a bishop is shown on the board below:



You want to implement a backtracking solution to the N-Bishops problem, where you want to return all possible placements of N bishops on an $N \times N$ chessboard such that no bishop threatens any other bishop on the board. Implement the following `promising()` function, which determines whether it is possible to place a bishop at row `row` and column `col`, given a `std::vector<std::vector<bool>>` that stores the current contents of the board (where `board[i][j]` being `true` indicates there is a bishop at row `i`, column `j`). You may assume that the board is square (i.e., same number of rows and columns), and that only bishops up to row `row` have already been placed.

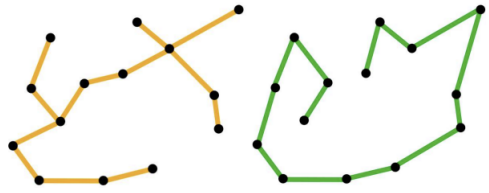
```
bool promising(const std::vector<std::vector<bool>>& board, size_t row, size_t col);
```

Chapter 22 Exercise Solutions

1. **The correct answer is (D).** For constraint satisfaction, you just want to know if a solution exists, not what the best solution is. This only applies to answer choice (D). In choice (A), you want to find the *minimum* distance to New York City; in choice (B), you want to find the *minimum* spanning tree; and in choice (C), you want to find the *shortest* path of the maze.

2. **The correct answer is (D).** Only statement II is false, since branch and bound continues to explore the remaining search space in case a better solution can be found. Statement I is true, since branch and bound can be used to find the optimal solution of a problem, and statement III is true, since a pruning function that prunes too much can result in an incorrect solution, and a pruning function that prunes too little may degrade performance since time is spent exploring paths that could have been discarded.

3. **The correct answer is (E).** The total weight of the MST of the non-visited points is a lower bound, but it is not guaranteed to be the optimal solution. This is because the MST of the remaining points may not actually be a valid path, as shown below:



   Here, the image on the left represents the MST, while the image on the right represents the optimal path that crosses these points. Note that the figure on the left is not a valid path that visits each point once! In this problem, you are told that the MST has a distance of 141 miles, but you cannot conclude that the optimal path also has a distance of 141 miles. Thus, choice (A) is not correct: the upper bound should not be changed to 316 miles, as the path length can only be 316 miles if the minimum spanning tree is also the optimal path (which, as shown above, is not always true). However, we do know that the optimal weight through the points *cannot be smaller* than the weight of the MST, as the MST must have the smallest possible weight. As a result, we know that the distance of the path we are considering cannot be less than 316 miles, making (E) the correct answer. Note that (B) is not true because the path is still promising (the best we can get from continuing the path is 316 miles, which is less than the best so far of 331 miles, so we should continue exploring the branch). Options (C) and (D) are false because you cannot conclude from this information that the current partial path leads to the optimal solution, or that it doesn't.

4. **The correct answer is (E).** The upper bound is the best complete solution encountered so far. Similar to the explanation for the previous problem, knowing the weight of the MST only allows you to identify whether to continue exploring the branch or not, but you should not update the upper bound until you compute the weight of a complete solution that is better than the best solution you have encountered so far.

5. **The correct answer is (A).** Heuristics do not always calculate the optimal solution. The precision of the calculation is sacrificed for speed, but these algorithms to provide answers that are close enough. This is useful for solving problems like TSP where the actual process of finding the best solution takes up a lot of time and may be too expensive to complete in practice.

6. **The correct answer is (E).** Brute force involves iterating over all possibilities and returning the minimum distance, which would require $\Theta(n!)$ time since there are $n!$ possible tours. In other words, brute force tries every possible permutation of these n cities, and there are $n!$ permutations in total (n locations to choose for location 1, $n-1$ ways to move from location 1 to location 2, $n-2$ ways to move from location 2 to location 3, …, 1 way to move from location $n-1$ to location n).

7. **The correct answer is (C).** Statement I is false because there is no guarantee that branch and bound will make the program run faster; it is possible that we generate permutations in an order such that the first permutation is better than our initial upper bound, the next permutation is better than the first permutation, and so on — in this case, nothing gets pruned, and the program does not run faster. Statement II is false for similar reasons, we could end up exploring branches in a way such that we are not able to prune anything, which would yield a time complexity that is the same as brute force. Statement III is true: if you overestimate the cost of extending a partial solution, you may end up treating the path is not promising even though it actually may lead to the optimal solution.

8. **The correct answer is (A).** Statement I is true: when solving a minimization problem using branch and bound, the upper bound is the best solution encountered so far, so if the lower bound of extending the current branch is already worse than this upper bound, the branch should be pruned. Statement II is false: when solving a maximization problem using branch and bound, the lower bound is the best solution encountered so far, so if the upper bound is greater than the lower bound, we could still find a better solution and the branch should not be pruned (note that we are maximizing, so a higher upper bound is desired since that means the current branch could yield a larger solution that the best we have encountered). Statement III is false: when solving a maximization problem, we want to overestimate the upper bound since underestimating would cause us to believe that a branch may not yield an optimal solution even when it could.

9. **The correct answer is (D).** There are $4! = 24$ distinct Hamiltonian cycles in the graph with four vertices, and $5! = 120$ distinct Hamiltonian cycles in the graph with five vertices. Thus, there are 120 - 24 = 96 more Hamiltonian cycles in the graph with five vertices than the one with four vertices.

10. **The correct answer is (D).** Backtracking can be used to solve constraint satisfaction problems, but it is not guaranteed to find an optimal solution. Thus, it cannot be used to solve I, which is an optimization problem, but it can be used to solve II and III, which care only about the existence of a solution rather than a best solution.

11. **The correct answer is (B).** Branch and bound can be used to find an optimal solution, while backtracking is designed to find the existence of a solution (and not necessarily the best). All the other choices apply to both backtracking and branch and bound.

12. **The correct answer is (A).** The current partial branch has a weight of $30 + 28 + 13 = 71$, which is the same as the best complete solution found so far. Since this partial branch already has a weight equal to a complete solution, there is no reason to continue extending this branch (since at best the outcome you get cannot go below 71), and the branch can be pruned.

13. This is a constraint satisfaction problem that can be solved using a backtracking approach. Similar to examples 22.1 and 22.2, we will add values to a running collection, storing each output in the solution once it reaches a size of k. An implementation of this is shown below:

```
std::vector<std::vector<int32_t>> k_combinations(int32_t n, int32_t k) {
  std::vector<std::vector<int32_t>> solution;
  backtrack(n, k, 1, solution, std::vector<int32_t>());
  return solution;
} // k_combinations()

void backtrack(int32_t n, int32_t k, int32_t start,
               std::vector<std::vector<int32_t>>& solution, std::vector<int32_t> current) {
  if (k == 0) {
    solution.push_back(current);
    return;
  } // if
  for (int32_t i = start; i <= n; ++i) {
    current.push_back(i);
    backtrack(n, k - 1, i + 1, solution, current);
    current.pop_back();
  } // for i
} // backtrack()
```

14. We can use a similar backtracking concept to solve this problem. We will iterate over all the characters of the input string and decide if we can append each potential number to generate a valid IP. If we successfully added four numbers to our IP from the provided characters without pruning the solution, then we know that we have a valid IP. An implementation of this is shown below:

```
std::vector<std::string> get_valid_ips(const std::string& digits) {
  std::vector<std::string> solution;
  backtrack(digits, "", 0, 0, solution);
  return solution;
} // k_combinations()

void backtrack(const std::string& digits, std::string curr_path, size_t idx, int32_t num_parts,
               std::vector<std::string>& solution) {
  // Can't have more than 4 numbers in an IP
  if (num_parts > 4) {
    return;
  } // if
  // This gets hit if we added exactly four numbers after processing all digits without pruning
  if (num_parts == 4 && idx == digits.length()) {
    // Remove trailing period
    curr_path.pop_back();
    solution.push_back(curr_path);
    return;
  } // if
  for (int32_t i = 1; i <= 3 && idx + i <= digits.length(); ++i) {
    std::string num = digits.substr(idx, i);
    // Prune invalid solutions
    if (num[0] == '0' && i != 1) {
      break;
    } // if
    else if (std::stoi(num) <= 255) {
      backtrack(digits, curr_path + digits.substr(idx, i) + ".", idx + i, num_parts + 1, solution);
    } // else if
  } // for i
} // backtrack()
```

15. To determine whether the placement of a bishop is promising, we just need to check if there are any other bishops in the diagonal directions from the provided position. If there are, then the solution is not promising. Since we know that only bishops up to row `row` have been placed, we only need to check the upper left and upper right diagonals (as there are no bishops below `row` in the given board). One implementation of this is shown below (note that this is similar to the N-Queens problem, but only checking the diagonals).

```
bool promising(const std::vector<std::vector<bool>>& board, size_t row, size_t col) {
  for (size_t r = row, c = col; r-- > 0 && c-- > 0; ) {
    if (board[r][c] == true) {
      return false;
    } // if
  } // for r, c
  for (size_t r = row, c = col; r-- > 0 && c++ < board.size(); ) {
    if (board[r][c] == true) {
      return false;
    } // if
  } // for r, c
  return true;
} // promising()
```