

Analysis of Algorithms
CS 375, Fall 2023

The Eight Problems: Information for Project 2

The Eight Problems: Information for Project 2

Our CS375 Project 2 references eight problems to be solved by exhaustive search algorithms. This document contains the eight problems, along with some additional instructions to be considered along with the general instructions for doing Project 2.

1 The Eight Problems to be Solved

Below are the eight problems to be solved, divided into three Categories. Note that each of the problems is a *decision problem*—it asks for a True / False answer to be given.

For each of the problems below, in all three Categories, your exhaustive search algorithm will need to look through either all subsets of a set or all permutations of a list. For your work, please assume that you can use algorithms to create the relevant lists for your exhaustive search, meeting the specifications given here (the same as those in our lecture notes) and with the time and space complexities given here:

- **Generate-All-Subsets** You may use a **Generate-All-Subsets**(S) algorithm that has *time complexity* $\Theta(n \cdot 2^n)$ and *space complexity* $\Theta(n \cdot 2^n)$ on input S of size n , meeting these specifications:

Input: $S = \{s_0, s_1, s_2, \dots, s_{n-1}\}$, a set of n elements

Output: L , a list of all subsets of S

- **Generate-All-Permutations** You may use a **Generate-All-Permutations**(L) algorithm that has *time complexity* $\Theta(n \cdot n!)$ and *space complexity* $\Theta(n \cdot n!)$ on input L of size n , meeting these specifications:

Input: $L = [s_0, s_1, s_2, \dots, s_{n-1}]$, a list of n elements

Output: PSL , a list of all permutations of L

Note that you are not told how these algorithms work, and they may not be identical to the ones we derived in class—you should just assume that they exist for your use and meet these specifications.

In addition, here are some reminders about *sets* and *graphs* that might be useful:

- The specifications for some of the problems below involve sets. Please recall that by the definition of a set, no two values in a set can be equal to each other. All of the specifications were written to be consistent with this definition.
- The specifications for some of the problems below involve graphs. Every graph G is defined as a combination of a set V of vertices in the graph and a set E of edges that connect some (or all, or none) of the vertices in the graph; for short, we say $G = (V, E)$. See CLRS, Appendix B.4 (pg. 1168) for more about graphs—and, as always, please feel free to ask me any questions about definitions regarding graphs!

The following sections give the three Categories containing the eight problems to be solved.

1.1 Category: Taking Stuff

The problems to be solved in this Category are:

1. **Fair Share** You and a friend are in a room with n valuable items—with values $c_1, c_2 \dots c_n$ —and you want to take all of them! But only if you each take *exactly* the same value with you. Is that possible with the items in front of you?

The Fair-Share problem:

Input: A set $C = \{c_1 \dots c_n\}$ of n positive integer values (the values of the n items).

Output: *True* if there exists a subset S of C for which the sum of the values in S is *exactly* the sum of the values not in S ; *False* otherwise.

For example: If $C = \{3, 6, 9, 12\}$, a correct algorithm for **Fair-Share** would return *True*, because there's a set $S = \{6, 9\}$ where the sum of the values is 15, and the sum of values not in S is $3 + 12 = 15$. On the other hand, if we consider set $C = \{1, 5, 10, 25, 50, 100\}$, there is no set S of values from C that could equal the values not in S (try it—no subset of C works for these values!), so a correct algorithm would return *False*.

2. **Price is *Exactly* Right** You're in a store with n items, with costs $c_1, c_2 \dots c_n$, and you have an amount V to spend on these items. Can you spend *exactly* V on some (or all) of the items from this store?

The Price-Exactly-Right problem:

Input: A set $C = \{c_1 \dots c_n\}$ of n positive integer values (the values of the n items); and a positive integer V .

Output: *True* if there exists a subset S of C for which the sum of the values in S is *exactly* equal to V ; *False* otherwise.

For example: If $C = \{1, 2, 3, 9\}$ and $V = 12$, a correct algorithm should return *True*, because there exists subset $C = \{1, 2, 9\}$ for which the values add up to 12. (That's not the only subset with values that add to 12, but one is enough for the algorithm to return *True*.) On the other hand, for the same $C = \{1, 2, 3, 9\}$, if $V = 8$, there is no subset S of C for which the values of S could add up to *exactly* 8.

3. **Book Bag** You're at a used book sale where there's a deal available: If you pay a flat fee—let's call the fee amount K —they give you a bag with a capacity of C and you can take as many books as you want, as long as they all fit in that bag. (The numbers K and C can be in whatever units you like—as long as we're consistent throughout the problem, it doesn't matter which they are.) You can choose books to take from a set $B = \{b_1 \dots b_n\}$ of n books, and each book b_i has a size $s(b_i)$ and a value $v(b_i)$. Is it possible to find some subset S of the books such that all the books in S could fit in the bag they give you, and the total value of the books in S add up to more than the amount K that you'd pay for the deal?

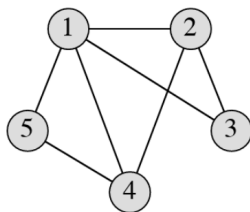


Figure 1: An example graph, referred to in examples accompanying problem statements. (Figure taken from CLRS.)

The Book-Bag problem:

Input: Set $B = b_1 \dots b_n$ so that each b_i has a positive integer size s_i and a positive integer value v_i ; positive integer capacity C ; positive integer fee K .

Output: *True* if there exists a subset $S = \{a_1 \dots a_m\}$ of B for which the sum of the sizes $\sum_{i=1}^m s(a_i)$ is less than or equal to C and the sum of the values $\sum_{i=1}^m v(a_i)$ is greater than or equal to K ; *False* otherwise.

For example: Let $B = \{b_1, b_2, b_3\}$ where b_1 has size 1 and value 5, b_2 has size 2 and value 12, and b_3 has size 3 and value 8. Then, if $C = 5$ and $K = 18$, a correct algorithm should return *True* on inputs B, C, K , because the set of books $\{b_2, b_3\}$ has their total size equal to 5, less than or equal to C , and their total value equal to 20, greater than or equal to K . For the same B and K , however, but $C = 4$, a correct algorithm should return *False*; one way to see this is that the greatest value possible for a set of books from B with total size less than or equal to 4 would be 17—from the subset $\{b_1, b_2\}$ —and that’s not greater than or equal to K .

1.2 Category: Social Networks

In all of the problems in this Category, we’ll be using graphs to represent social networks! Every vertex in a graph will represent a person, and every edge between two people will represent that the people know each other. Please assume graphs for problems in this Category are undirected.

The problems to be solved in this Category are:

1. **Clique** A *clique* is defined to be a collection $C = \{c_1 \dots c_j\}$ of people such that every pair of people in C know each other. Because we’re using a graph $G = (V, E)$ to represent the social network, a clique is a subset C of the vertices of the graph for which every pair of vertices in C has an edge between them. (C could, in principle, be equal to V , which would be a *complete* graph.) **The question:** Given a number K , is there a clique of size K in the social network we’re studying?

The Clique problem:

Input: Graph $G = (V, E)$, positive integer $K \leq |V|$.

Output: *True* if there is a clique C of size K in G ; *False* otherwise.

For example: In Figure 1, the graph contains multiple cliques of size 3, such as the set $\{1, 2, 4\}$. The set $\{1, 2, 3, 4\}$ is not a clique, because 3 is not connected to 4.

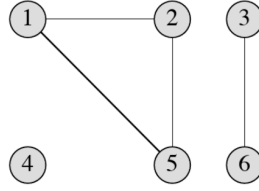


Figure 2: An example graph, referred to in examples regarding the *complement* of a graph. (Figure taken from CLRS.)

2. **Strangers** In the social network $G = (V, E)$, we will define a group of people $S = \{s_1 \dots s_j\}$ to be *strangers to each other* when for every pair of people in S , they do not know each other. Because we're using graph $G = (V, E)$ to represent the social network, a *set of strangers* is a subset S of the vertices of the graph for which no edge in E exists between any two people in S . (S could, in principle, be equal to V , which would be a maximally *sparse* graph.) **The question:** Given a number K , is there a set of strangers S of size K in the social network we're studying?

The **Strangers** problem:

Input: Graph $G = (V, E)$, positive integer $K \leq |V|$.

Output: *True* if there is a set of strangers S where S has size K in G ; *False* otherwise.

For example: In Figure 1, the set $\{2, 5\}$ is a set of strangers of size 2 (there are also others), but there is no set of strangers of size 3—for every three people in the graph, there's a connection between some two of them. (Try it!)

3. **Network Cover** In the social network $G = (V, E)$, we will define a group of people $P = \{p_1 \dots p_j\}$ to be a *network cover* if, across all the people in P , every social connection in the network involves at least one of the people in P . Because we're using graph G to represent the social network, a network cover is a subset P of the vertices in G such that every edge in E involves at least one person in P . **The question:** Given a number K , is there a network cover P of size K in the social network we're studying?

The **Network-Cover** problem:

Input: Graph $G = (V, E)$, positive integer $K \leq |V|$.

Output: *True* if there is a network cover P where P has size K in G ; *False* otherwise.

For example: In Figure 1, the set $\{1, 2, 4\}$ is a network cover of size 3 (there are also others), but the set $\{1, 4\}$ is not a network cover, because the connection between 2 and 3 is not covered by $\{1, 4\}$.

Hint: This hint only applies to *reductions* (see Section 3 of the Project 2 document), not other parts of the Project.

When thinking about designing reductions involving the problems in the Social Networks Category (Section 1.2), you might want to consider the *complement* of a graph as part of your reduction. By definition, for a graph $G = (V, E)$, the *complement* of G is a graph $G' = (V, E')$, where the vertices are the same as in G but the edges are *all edges not in E* . More precisely, considering every edge in a graph as a pair of vertices, $E' = \{(u, v) \mid u, v \in V, \text{ but } (u, v) \notin E\}$. As a concrete example, let G be the graph in Figure 2; then, there would be edges from vertex 4 to every other vertex in the complement G' , because none of those edges are in G , but edge $(1, 2)$ would not be in G' because there is an edge between vertices 1 and 2 in G .

You don't need to use the complement of a graph in your reduction, but you might want to in some cases. As always, please feel free to talk with me about these concepts!

1.3 Category: Maps and Touring

In the problems in this Category, we'll be using graphs to represent maps! Every vertex in a graph will represent a location, and every edge between two locations will represent that we can travel between those locations in either direction. Please assume graphs for problems in this Category are undirected.

For these problems, we'll define a *tour* on a map $G = (V, E)$: A tour is a path that starts in some initial city c_1 in V and then, following edges in E , passes through every other city in V exactly once before returning to c_1 . For example, in the map represented by the graph in Figure 3, one possible tour is represented by the shaded edges: the one from u to w to v to x and then back to starting city u . Note that there are many possible tours through all cities on this map, this is just one possibility.

Also, for the problems in this section, note that in a tour, it doesn't really matter which city we indicate as the starting city—since the tour goes through all of them exactly once before looping back to where it started, its starting point could equivalently be anywhere for these problems. The direction also doesn't matter for the problems in this section, because the graphs are undirected. For example, the tour indicated by the shaded edges in Figure 3 could be viewed as starting at city x and going x to v to w to u to x just as well as u to w to v to x to u .

The problems to be solved in this Category are:

1. **Traveling Salesman** In this classic CS problem, we start with a *complete* graph $G = (V, E)$, in which every pair of cities v_i, v_j in V is connected by an edge in E . In addition to the graph, there is a distance function d that gives a distance $d(v_i, v_j)$ between every pair of cities; assume that the distance is the same in either direction, so $d(v_i, v_j) = d(v_j, v_i)$ for every pair of cities.

It is our traveling salesperson's job to make a tour—to start from their home city, then follow edges in the graph to visit every other city *exactly once* before returning to their

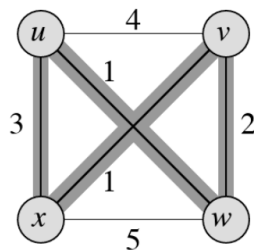


Figure 3: An example *tour* on a graph, referred to in examples accompanying problem statements. Here, there are many possible tours through all cities on this map, including one from u to w to v to x and then back to starting city u (shown by shaded edges in the graph). (Figure taken from CLRS.)

home city. **The question:** Given a number K , is it possible to make a tour while covering total distance K or less?

The Traveling-Salesman problem:

Input: Graph $G = (V, E)$, distance function d as described above, and positive integer K .

Output: *True* if there is a tour of V having distance K or less; *False* otherwise.

For example: In Figure 3, the distance function d giving distances between cities is indicated by the number over each edge—e.g., $d(v, x) = 1$ and $d(x, w) = 5$. Consider the map in that Figure, that distance function d , and number $K = 5$. A correct algorithm would return *False*, because there is no tour of all those cities (remember, it has to end up back where it started!) with distance 5 or less. On the other hand, with that map, that distance function, and $K = 9$, a correct algorithm would return *True*—indeed, the shaded edges are a tour of distance 7, which is less than 9.

2. **Hamiltonian Tour** Unlike the Traveling Salesman example, the touring company of *Hamilton* does *not* start with a complete graph—instead, they start with some map $G = (V, E)$ that may or may not have edges between any two cities. There is also no distance function to be considered here; all that matters is whether or not they can get from one city to the next.

Their goal, however, is to make a tour, in the same technical sense of *tour* used in the Traveling Salesman problem—to visit every city exactly once before returning home.

The question: Given a map $G = (V, E)$, is there a tour in the map, visiting each city exactly once before returning to the city from which it started?

The Hamiltonian-Tour problem:

Input: Graph $G = (V, E)$.

Output: *True* if there is a tour in G ; *False* otherwise.

For example: In Figure 1, there are many paths that are not tours in this sense. For example, if a possible tour started at city 5, and then went to 1, and then 2 and then

3, it could not get to city 4 without going back to 1 along the way, and that's not a tour in our sense of the word—it would visit city 1 more than once before returning to where it started. Nonetheless, there is a tour through that graph—for example, it could go from 1 to 3 to 2 to 4 to 5 before going back to 1—so a correct algorithm would return *True* on that graph.