Algorithms and Data Structures (ECE 345)

Issued on: March 23rd, 2017
Due on: April 10th, 2017

Assignment 4

Exercise 1 Dynamic Programming, 15 points

Elon Musk is building a prototype Hyperloop line that travels through cities $X = x_1, x_2, ..., x_n$ in that order. The Hyperloop may travel through the same city multiple times. He wishes to construct a public transit service using the Hyperloop, by constructing stops at a subset of the cities on the route. In order to be useful, the transit service must stop in a list of cities in order, and then go back through those same cities in the reverse order.

To maximize the usage of the transit service, it should have as many stops as possible. For instance, consider the route $X =$ London (L), Kitchener (K), Hamilton (H), Burlington (B), Toronto (T), Ottawa (O), Montreal (M), Pickering (P), Toronto (T), Burlington (B), Kitchener (K). Two valid transit paths would be KTTK and KBTMTBK, the latter of which is optimal. Note that Montreal is only stopped at once, but the route is valid as long as it reads the same backwards and forwards.

Given a Hyperloop route X, devise a dynamic programming algorithm to find the length of the optimal (longest) transit path of X. Write the pseudocode, recurrence relation, and analyze the run time complexity of your algorithm.

Exercise 2 Dynamic Programming, 15 points

Two notorious robbers have just made a big score and managed to escape to their safe-house. Their loot is a collection of n diamonds of various value and an associate has helped them evaluate the price of each diamond. Being in an immensely cerebral mood, instead of splitting the loot in half, they decide to play a game that will determine the cut. They allocate the n diamonds arbitrarily into a row. Suppose the diamond row is denoted as a sequence $d_1, d_2, \ldots, d_n$, with respective values $v_1, v_2, \ldots, v_n$, where diamond d_i has value v_i. Remember that these values are known to both of them. The game goes as follows: The two robbers take turns picking a diamond from the sequence, but can only pick the first or the last diamond of the (remaining) sequence. The goal from each robber's perspective is to collect the maximum possible value of diamonds by the end of the game. Finally, suppose that n is even.

1. Imagine that the strategy of the robber that starts first is to always pick the available diamond of greater value. Remember they are only allowed to pick the first or last diamond of the remaining sequence at each turn. This is a natural greedy strategy. Is it optimal? Show a small sequence of diamonds (decide the values yourselves) such that this strategy will not achieve the maximum possible total value for the robber that starts first. Show why this greedy strategy fails in your sequence and also show what would be the optimal strategy for the robber in that particular sequence.

2. You need to describe an $O(n^2)$ algorithm to compute an optimal strategy for the first robber. Given the initial sequence, your algorithm should precompute in $O(n^2)$ time some information, and then the first robber should be able to make each move optimally in $O(1)$ time by looking up the precomputed information. Assume that the second robber will always make an optimal move: After every move the first robber makes, the second robber will make the move that minimizes the total possible value that the first robber can collect.

    (a) Describe the problem and the sub-problems your algorithm has to compute. Denote your sub-problems as $V(i, j)$. What does $V(i, j)$ represent?

    (b) Give a recursive formula that expresses the sub-problem $V(i, j)$ in terms of other smaller sub-problems. Justify the formula.

(c) Explain the dynamic programming order. That is, explain how memoization works in the problem and how Robber 1 chooses what move to make from your precomputed information. Further, you need to provide pseudocode that computes the maximum value and outputs the moves that Robber w1 has to make in order to achieve it. Finally, justify why your algorithm has the desired run-time of $O(n^2)$.

EXERCISE 3 NPC Reductions, 15 points

The k-spanning tree problem is defined as follows:

Problem k-SPANNING-TREE

Input: $G = (V,E)$

Question: Does G contain a spanning tree T in which for all vertices v, $degree(v) \leq k$?

a. Show that for any $k \geq 2$, k-SPANNING-TREE is in NP.

b. Show that 2-SPANNING-TREE is NP-complete. *Hint:* Use a reduction from HAM-CYCLE.

c. Show that for all $k \geq 2$, k-SPANNING-TREE is NP-complete. *Hint:* Use a reduction from 2-SPANNING-TREE to k-SPANNING-TREE.

EXERCISE 4 NPC Reductions, 15 points

Show that if $k \geq 3$, then the k-SAT problem is NP-complete. Use induction on k. For the basis step, use the fact that 3-SAT is NP-complete as you've seen in class.

EXERCISE 5 Programming Exercise, 40 points

The Traveling Salesman Problem (TSP) in its general form is an NP-hard optimization problem, notorious for being one of the most computationally expensive real-world problems. Broadly speaking, it considers the case where a salesman, starting at his hometown, wishes to visit a predetermined set of target cities to perform sales. During his journey, he has to visit each target city exactly once, and at the end return to his hometown. The question he wishes to answer before starting the trip is the following: "Given the pairwise distances between cities, what is the best order in which to visit them, so as to minimize the overall distance traveled?"

Formally speaking, let there be n target cities denoted as $1,2,\ldots,n$ with city 1 being the salesman's hometown. Let also $D = (d_{ij})$ be the matrix of pairwise intercity distances which are all known beforehand. This also implies that there is always a way to go from one city to another. In a graphical representation that would imply a complete, undirected graph with nodes being cities and edge lengths being the corresponding distances. The goal is to design a tour (simple cycle in the graph) that starts and ends at 1, includes all other cities exactly once, and has minimum total length.

How expensive is it to compute an optimal tour? If you think about it, the brute-force approach is to evaluate every possible tour and return the best one, which is prohibitively expensive. We will now see that dynamic programming yields a much faster solution.

Dynamic Programming for TSP

What is an appropriate subproblem for TSP? The most obvious partial solution to TSP is the initial portion of a tour. Suppose we start at city 1 as required, visit a few cities, and are now in city j. How can we extend this partial tour? To do so, we need to know j, since this will determine which cities are most convenient to visit next. And we also need to know all the cities visited so far, so that we don't repeat any of them. Here, then, is an appropriate subproblem:

For a subset of cities $S \subseteq \{1,2,\ldots,n\}$ that includes 1, and $j \in S$, let $C(S,j)$ be the length of the shortest path visiting each node in S exactly once, starting at 1 and ending at j.

When $|S| > 1$, we define $C(S,1) = \infty$ since the path cannot both start and end at 1. Of course, we can then leverage optimal substructure to express $C(S, j)$ in terms of smaller subproblems.

Below, you are to implement the DP approach with the subproblems defined as above and run some runtime evaluations.

Input:

Your executable will be named `tsp`. It will be given an input file named `distances` which represents distances between cities. It will be a 3-column file and each row will be of the format:
`city` i `city` j `distance`
The row above indicates that cities i and j have distance $d_{ij} =$ `distance`. Entries are separated by "space". Assume that distances are positive real numbers. City 1 is the starting and ending point. Your executable should be callable from command line as:
`tsp distances`

Output:

You program will be computing the optimal tour of cities that minimizes the total tour length. The tour itself should be printed to standard output, as well as the length of the tour.

Assuming that distances are positive real numbers, a hypothetical –yet valid– output for 5 cities should look like this:

```
OPTIMAL TOUR LENGTH: 1545.82
TSP TOUR:
1
4
3
5
2
1
```

Deliverables:

- Your source code, including a full build environment and instructions how to build in a `README` file.
- A written report as part of your Assignment 4 submission, where you address the following points:
  - What is the runtime complexity of the brute force approach described above? Justify your answer.
  - Devise a DP method to solve the problem. Use the definition of what subproblem $C(S, j)$ is, exactly as it is given above. Write down the recurrence that gives the optimal solution and explain your reasoning. You need to provide **pseudocode** that uses memoization to find the **length of the optimal tour**, but also **returns the optimal tour (sequence of cities)**. Clearly show the process in your pseudocode.
  - What is the asymptotic complexity of the DP approach you developed? Think of how many subproblems exist asymptotically. What is the asymptotic cost of evaluating the optimal length for one subproblem?
  - You will need to use multiple `distances` files of your making to run the following experiment. Create 10 different files such that all have a different number of cities: $5, 10, 15, \ldots, 50$. Run your program and show the results in a **time vs. # cities** plot.
  - Discuss the plot above. What do you observe? Does the empirical result match your asymptotic analysis?