

You are encouraged to think about all of the following exercises to help your understanding.

However, only solutions to the final two questions (A, and B) are to be submitted for assessment.

Practice Exercises

1. Consider a directed graph $G = (V, E)$ where each edge $(u, v) \in E$ has a nonnegative distance $d(u, v)$. (We define $d(u, v) = \infty$ whenever $(u, v) \notin E$.) The **length** of a path $p = \langle v_0, v_1, \dots, v_n \rangle$ is defined as $\sum_{i=1}^n d(v_{i-1}, v_i)$. A **shortest path** from u to v is a path from u to v of minimal length. Some vertex v_0 is designated as the **source**, and we want to find the shortest path from the source to each of the other vertices.

The following greedy algorithm claims to find the lengths of these shortest paths.

GREEDY-PATHS(G, d)

```

1   $C \leftarrow V \setminus \{v_0\}$ 
2  for each  $v \in C$  do  $P[v] \leftarrow d(v_0, v)$ 
3  while  $C \neq \emptyset$  do
4      $v \leftarrow$  some element of  $C$  minimizing  $P[v]$ 
5      $C \leftarrow C \setminus \{v\}$ 
6     for each  $w \in C$  do
7          $P[w] \leftarrow \min(P[w], P[v] + d(v, w))$ 
8  return  $P$ 

```

- (a) In the algorithm GREEDY-PATHS, let $S = V \setminus C$ denote the set of vertices that have already been chosen. Call a path from the source v_0 to some other vertex an ***S*-path** if all intermediate vertices along the path belong to S .

Prove the following.

Lemma: Every time the algorithm performs the **while**-loop test on line 3, the following is true of $P[v]$ for every vertex $v \neq v_0$:

- (i) if $v \in S$, then $P[v]$ is the length of the shortest path from v_0 to v ;
- (ii) if $v \notin S$, then $P[v]$ is the length of the shortest S -path from v_0 to v .

- (b) Deduce that the algorithm GREEDY-PATHS is correct.

2. Suppose that we have a set of activities to schedule among a large number of lecture halls. We wish to schedule all the activities using as few lecture halls as possible. Give an efficient greedy algorithm to determine which activity should use which lecture hall. Analyse the running time of your algorithm.
3. **Knapsack Problem:** You plan to go hiking, and you want to pack your stuff in a knapsack. Your knapsack has a total volume of V cubic inches, and you have n different items of volumes v_1, \dots, v_n that you want to take with you. Alas, you can't take them all, since their total volume exceeds the volume of your knapsack. So you need to decide which items to take with you, and which items to leave at home.

Suppose that you want to make your decision so as to utilize the volume of the knapsack as much as possible. That is, you want to find a subset of the items so as to maximize the sum of their volumes subject to the constraint that this sum is no more than V . We call the subset which maximizes this sum the *optimum packing* of v_1, \dots, v_n in volume V .

- (a) Fix a list of items v_1, \dots, v_n . For any $i \leq n$ and any V denote by $\text{OPT}_i(V)$ the volume of the optimum packing of v_1, \dots, v_i in volume V . Prove that for all i and V , we have $\text{OPT}_i(V) = \max \left(\text{OPT}_{i-1}(V), v_i + \text{OPT}_{i-1}(V - v_i) \right)$.
 - (b) Assume that all the volumes v_1, \dots, v_n are integer units. Design a dynamic programming algorithm to solve the problem. Analyze your algorithm.
4. **Printing neatly.** Consider the problem of neatly printing a paragraph on a printer. The input text is a sequence of n words of length $\ell_1, \ell_2, \dots, \ell_n \leq M$ characters, respectively. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j ($i \leq j$), and we leave exactly one space between words, the number of extra space characters at the end of the line is $M - j + i - \sum_{k=i}^j \ell_k$ (which must be nonnegative so that the words fit on the line). We wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of lines.
 - (a) Give an example in which the obvious greedy algorithm, where you repeatedly fill lines with as many words as possible, fails to provide an optimal solution.
 - (b) Design a dynamic programming algorithm to solve this problem.
 - (c) Analyze the running time and space requirements of your algorithm.

5. The **flow sum** f_1+f_2 of two flows f_1 and f_2 for a flow network is defined by

$$(f_1+f_2)(u, v) = f_1(u, v) + f_2(u, v).$$

If f_1 and f_2 are valid flows, which of the three flow properties must the flow sum f_1+f_2 satisfy, and which might it violate. Justify your answers.

6. (From CLRS, Exercise 26.1-4, page 714.) Given a flow f for a flow network G and a real number α , we can define the **scalar flow product** αf by

$$(\alpha f)(u, v) = \alpha \cdot f(u, v).$$

Prove that if f_1 and f_2 are valid flows, then so is $\alpha f_1 + (1-\alpha)f_2$ for any α in the range $0 \leq \alpha \leq 1$.

7. (From CLRS, Exercise 26.3-4, page 735.) Let $G = (V, E)$ be an undirected bipartite graph with vertex partition $V = L \cup R$, where $|L| = |R|$. A **perfect matching** in G is a matching in which every vertex is matched. For any $X \subseteq V$, define the **neighbourhood** of X by

$$N(X) = \{y \in V : (x, y) \in E \text{ for some } x \in X\},$$

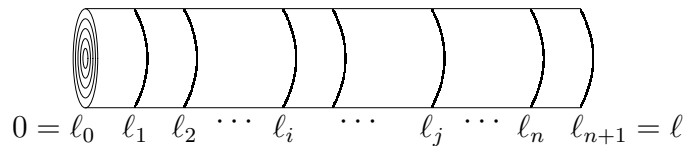
that is, the set of vertices adjacent to some member of X .

Prove the following.

Hall's Theorem: There exists a perfect matching in G if, and only if, $|A| \leq |N(A)|$ for every subset $A \subseteq L$.

(Hint: If there is a perfect matching, then the inequality can easily be demonstrated. If, on the other hand, there is no perfect matching, then consider the maximum flow f of the corresponding flow network G' ; let (S, T) be a cut with capacity equal to $|f|$, and let $A = L \cap S$.)

- A. We wish to store programs P_1, P_2, \dots, P_n on a disk. Program P_i needs s_i kilobytes, and the disk has capacity $D < \sum_{i=1}^n s_i$ kilobytes.
- Prove or disprove: to maximize the number of programs held on the disk, we can use a greedy algorithm that takes programs in order of nondecreasing size.
 - Prove or disprove: to maximize the disk usage, we can use a greedy algorithm that takes programs in order of nonincreasing size.
- B. You bring an ℓ -foot log of wood to your local sawmill. You want it cut in n specific places: $\ell_1, \ell_2, \dots, \ell_n$ feet from the left end. The sawmill charges $\mathcal{L}x$ to cut an x -foot log any place you like.



For example, suppose we wish to cut a log $\ell=12$ feet long at lengths $\ell_1=2$, $\ell_2=5$, and $\ell_3=8$ feet from the left end. The first cut (regardless of where it is made) will cost $\mathcal{L}12$, but the cost for the two subsequent cuts will depend on the order in which the cuts are made (as the lengths of the subsequent sublogs to be cut will be different). For example, cutting in the order ℓ_1, ℓ_2, ℓ_3 would cost $12+10+7 = \mathcal{L}29$, while cutting in the order ℓ_2, ℓ_1, ℓ_3 would cost $12+5+7 = \mathcal{L}24$. In this example, it makes sense to first cut in the most central spot, to minimize the length of the two remaining pieces.

- Consider a greedy algorithm that cuts the log so that the maximum length of the resulting two pieces is always as small as possible; that is, it cuts it in the most central spot. Show that this algorithm does not necessarily achieve the minimal cost, by giving an example in which it fails to do so. (Hint: Consider three cuts all close to the midpoint.)

Let $c[i, j]$ (for $0 \leq i < j \leq n+1$) be the optimal (i.e., least) cost of completely cutting the sublog whose left endpoint is at ℓ_i and whose right endpoint is at ℓ_j . We thus wish to compute $c[0, n+1]$.

- Give a recursive definition for $c[i, j]$. (Hint: Start by defining $c[i, i+1]$, and $c[i, i+2]$.)
- Give pseudocode for a dynamic programming algorithm which computes $c[0, n+1]$. (Note: You needn't compute the optimal order itself, just the optimal cost.)
- Analyze the run time and space requirement of this algorithm.