1 Study Group

List the names and SIDs of the members in your study group.

2 2-SAT

Please provide solutions to parts (d), (e) and (f) of Question 3.28 from http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf.

3 Minimum Spanning $k$-Forest

Given a graph $G(V, E)$ with nonnegative weights, a spanning $k$-forest is a cycle-free collection of edges $F \subseteq E$ such that the graph with the same vertices as $G$ but only the edges in $F$ has $k$ connected components. For example, consider the graph $G(V, E)$ with vertices $V = \{A, B, C, D, E\}$ and all possible edges. One spanning 2-forest of this graph is $F = \{(A, C), (B, D), (D, E)\}$, because the graph with vertices $V$ and edges $F$ has components $\{A, C\}, \{B, D, E\}$.

The minimum spanning $k$-forest is defined as the spanning $k$-forest with the minimum total edge weight. (Note that when $k = 1$, this is equivalent to the minimum spanning tree). In this problem, you will design an algorithm to find the minimum spanning $k$-forest. For simplicity, you may assume that all edges in $G$ have distinct weights.

(a) Define a $j$-partition of a graph $G$ to be a partition of the vertices $V$ into $j$ (non-empty) sets. That is, a $j$-partition is a list of $j$ sets of vertices $\Pi = \{S_1, S_2, \ldots, S_j\}$ such that every $S_i$ includes at least one vertex, and every vertex in $G$ appears in exactly one $S_i$. For example, if the vertices of the graph are $\{A, B, C, D, E\}$, one 3-partition is to split the vertices into the sets $\Pi = \{\{A, B\}, \{C\}, \{D, E\}\}$.

Define an edge $(u, v)$ to be crossing a $j$-partition $\Pi = \{S_1, S_2, \ldots, S_j\}$ if the set in $\Pi$ containing $u$ and the set in $\Pi$ containing $v$ are different sets. For example, for the 3-partition $\Pi = \{\{A, B\}, \{C\}, \{D, E\}\}$, an edge from $A$ to $C$ would cross $\Pi$.

Show that for any $j$-partition $\Pi$ of a graph $G$, if $j > k$ then the lightest edge crossing $\Pi$ must be in the minimum spanning $k$-forest of $G$.

(b) Give an efficient algorithm for finding the minimum spanning $k$-forest.

Please give a 3-part solution.

4 Steel Beams

You’re a construction engineer tasked with building a new transit center for a large city. The design for the center calls for a $T$-foot-long steel beam for integer $T > 0$. Your supplier can
provide you with an *unlimited* number of steel beams of integer lengths \(0 < c_1 < \ldots < c_k\) feet. You can weld as many beams as you like together; if you weld together an \(a\)-foot beam and a \(b\)-foot beam you’ll have an \((a + b)\)-foot beam. Unfortunately, every weld increases the chance that the beam might break, so you want as few as possible.

Your task is to design an algorithm which outputs how many beams of each length you need to obtain a \(T\)-foot beam with the minimum number of welds, or ‘not possible’ if there’s no way to make a \(T\)-foot beam from the lengths you’re given. (If there are multiple optimal solutions, your algorithm may return any of them.)

(a) Consider the following greedy strategy. Start with zero beams of each type. While the total length of all the beams you have is less than \(T\), add the longest beam you can without the total length going over \(T\).

(i) Suppose that we have 1-foot, 2-foot and 5-foot beams. Show that the greedy strategy always finds the optimum.

(ii) Find a (short) list of beam sizes \(c_1, \ldots, c_k\) and target \(T\) such that the greedy strategy fails to find the optimum. Briefly justify your choice.

(b) Give a dynamic programming algorithm which always finds the optimum.

(i) State your recurrence relation.

(ii) Prove correctness of your algorithm by induction.
   i. Show that the base case is correct.
   ii. Assuming that your recurrence relation is correct for previous subproblems, show that it gives the correct value for the current subproblem.
   iii. Give the order where you can solve the subproblems, and show that for this order, evaluating the recurrence relation will use only subproblems that have already been computed.

(iii) Find the running time and space requirement of your algorithm.

5 A Dice Game

Consider the following 2-player game played with a 6-sided die. On your turn, you can decide either to roll the die or to pass. If you roll the die and get a 1, your turn immediately ends and you get 1 point. If you instead get some other number, it gets added to a running total and your turn continues (i.e. you can again decide whether to roll or pass). If you pass, then you get either 1 point or the running total number of points, whichever is larger, and it becomes your opponent’s turn. For example, if you roll 3, 4, 1 you get only 1 point, but if you roll 3, 4, 2 and then decide to pass you get 9 points. The first player to get to \(N\) points wins, for some positive \(N\).

Alice and Bob are playing the above game. Let \(W(x, y, z)\) be the probability that Alice wins given that it is currently Alice’s turn, Alice’s score (in the bank) is \(x\), Bob’s score is \(y\) and Alice’s running total is \(z\).

(a) Give a recursive formula for the winning probability \(W(x, y, z)\).
(b) Prove correctness of your algorithm by induction.

(i) Show that the base case is correct.

(ii) Assuming that $W(x, y, z)$ is computed correctly for previous subproblems, show that it is correct for the current subproblem.

(iii) Give the order where you can solve the subproblems. Show that for this order, evaluating the recurrence relation will use only subproblems that have already been computed.

(c) Find the runtime of your algorithm.

6 Non-Prefix Code

As we have learned in lecture, the Huffman code satisfies the Prefix Property, which states that the bit string representing each symbol is not a prefix of the bit string representing any other symbol. One nice property of such codes is that, given a bit string, there is at most one way to decode it back to a sequence of symbols. However, this is not true anymore once we are working with codes that do not satisfy the Prefix Property. For example, consider the code that maps $A$ to 1, $B$ to 01 and $C$ to 101. A bit string 101 can be interpreted in two ways: as $C$ or as $AB$.

Your task is to, given a bit string $s$, determine how many ways one can interpret $s$. The mapping from symbols to bit strings of the code will be given to you as a dictionary $d$ (e.g., in the example, $d = \{A : 1, B : 01, C : 101\}$); you may assume that you can access each symbol in the dictionary in constant time. Your algorithm should run in time at most $O(nm\ell)$ where $n$ is the length of the input bit string $s$, $m$ is the number of symbols, and $\ell$ is an upper bound on the length of the bit strings representing symbols.

Please give a 3-part solution.

7 Breaking Chocolate

There is a chocolate bar consisting of an $m \times n$ rectangular grid of squares. Some of the squares have raisins in them, and you hate raisins. You would like to break the chocolate bar into pieces so as to separate all the squares with raisins, from all the squares with no raisins. For example, shown below is a $6 \times 4$ chocolate bar with raisins in squares marked $R$. As shown in the picture, one can separate the raisins out in exactly four breaks.
At any point in time, a break is a cut either horizontally or vertically of one of the pieces at the time.

Design a DP based algorithm to find the smallest number of breaks needed to separate all the raisins out. Formally, the problem is as follows:

**Input:** Dimensions of the chocolate bar $m \times n$ and an array $A[i,j]$ such that $A[i,j] = 1$ if and only if the $ij^{th}$ square has a raisin.

**Goal:** Find the minimum number of breaks needed to separate the raisins out.

(a) Define your subproblem.

(b) Write down the recurrence relation for your subproblems.

(c) What is the time complexity of solving the above mentioned recurrence? Provide a justification.