ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2020

Problem Set 5

Due: 23:59, June 26, 2020

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1

Consider the following probabilistic process:

Set $X = 0$
  Repeat n times
    Toss a fair coin
      If heads then $X \leftarrow X + 1$
      Else then $X \leftarrow X - 1$

Assume the coin tosses are independent, and the coin is fair, so that heads and tail each have $1/2$ probability of occurring.

(a) Compute $E[X]$.

(b) For even n, show that $\Pr[X = 0] \sim \sqrt{2/(\pi n)}$. What about odd n?
Note: you will need to use Stirling's approximation, which says that

$$n! \sim \left(\frac{n}{e}\right)^2 \cdot \sqrt{2\pi n}$$

Problem 2

3-Coloring is a yes/no question, but we can phrase it as an optimization problem as follows.

Suppose we are given a graph $G = (V, E)$, and we want to color each node with one of three colors. We say that an edge (u, v) is satisfied if the colors assigned to u and v are different. Depending on the graph, we may not be able to satisfy all the edges. Instead, we simply try to satisfy as many edges as possible.

Let c^* be the maximum possible of number of satisfied edges for the given graph. Give a randomized polynomial-time algorithm that produces a 3-coloring satisfying at least $\frac{2}{3}c^*$ edges in expectation.

Problem 3

Imagine you have a large array A of size n containing 0's and 1's. We want to estimate the number of 1's in the array. That is, let $T = \sum_{i=1}^{n} A[i]$ be the number of 1's in array A.

We solve this problem via sampling. We repeat the following procedure s times: choose a random index in the array and check whether it is a 0 or a 1. Let X_i be the value of the ith sample, and $Y = (n/s) \sum_{i=1}^{s} X_i$. The value Y is an estimator for the sum T.

1. What is $E[X_i]$?
2. What is $\text{Var}[X_i]$?
3. What is $E[Y]$?
4. What is $\text{Var}[Y]$? (Hint: Show that $\text{Var}[Y] = (n^2/s)\text{Var}[X_1]$.)

From these results, do you expect that Y is a good estimator of T?

Problem 4

Consider the following maximization version of the 3-Dimensional Matching Problem. Given disjoint sets X, Y, Z and a set $T \subseteq X \times Y \times Z$ of ordered triples, a subset $M \subseteq T$ is a 3-dimensional matching if each element of $X \cup Y \cup Z$ is contained in at most one of these triples. The Maximum 3-Dimensional Matching Problem is to find a 3-dimensional matching M of maximum size. The size of the matching is the number of triples it contains, and you may assume $|X| = |Y| = |Z|$ if you want.
Give a polynomial-time algorithm that finds a 3-dimensional matching of size at least 1/3 times the maximum possible size.

Problem 5

Suppose you are given an $n \times n$ grid graph G as in Figure 1.

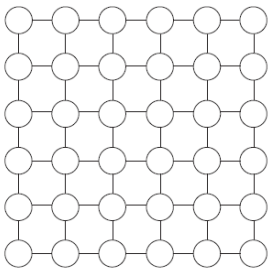

Figure 1: A grid graph.

Associated with each node v is a nonnegative integer weight $w(v)$. You may assume that the weight of all nodes are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of the weights of the nodes in S is as large as possible.

Consider the following greedy algorithm for this problem.

```
The "heaviest-first" greedy algorithm:
  Start with S equal to the empty set
  While some node remains in G
    Pick a node v of maximum weight
    Add v to S
    Delete v and its neighbors from G
  Endwhile
  Return S
```

(a) Let S be the independent set returned by the "heaviest-first" algorithm, and let T be any other independent set in G. Show that, for each node $v \in T$, either $v \in S$, or there is a node $v' \in S$ so that $w(v) < w(v')$ and (v, v') is an edge of G.

(b) Show that the "heaviest-first" algorithm returns an independent set of total weight at least 1/4 times the maximum total weight of any independent set in G.

Problem 6

Suppose you're acting as a consultant for the Port Authority of a small Pacific Rim nation. They're currently doing billions of dollars of business each year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here's a basic sort of problem they face. A ship arrives, with n containers with weights $w_1, \ldots, w_n$. Standing on the dock is a set of trucks, each of which can hold K units of weight. You can assume that K and each w_i are integers. You can stack multiple containers in each truck, subject to the weight restriction of K. The goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers $1, 2, 3, \ldots$ into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

1. Give an example of a set of weights and a value of K where this algorithm does not use the minimum possible number of trucks.

2. Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

Problem 7

Suppose you are given a set of positive integers $A = a_1, a_2, \ldots, a_n$ and a positive integer B. A subset $S \subseteq A$ is called feasible if the sum of the numbers in S does not exceed B, i.e. $\sum_{a_i \in S} a_i \leq B$. The sum of the numbers in S will be called the total sum of S. The goal is to find a feasible subset with the largest possible total sum.

Example. If $A = 8, 2, 4$ and $B = 11$, then the optimal solution is the subset $S = 8, 2$.

(a) Here is an algorithm for this problem.

```
Initially  S = φ
Define  T = 0
For  i = 1, 2, ..., n
     If  T + a_i ≤ B  then
          S ← S ∪ {a_i}
          T = T + a_i
     Endif
Endfor
```

Give an instance in which the total sum of the set S returned by this algorithm is less than half the total sum of some other feasible subset of A.

(b) Give a polynomial-time approximation algorithm for this problem with the following guarantee: It returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$. Your algorithm should have a running time of at most $O(n \log n)$.