CS 170 HW 13

Due **2021-05-04, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, write "none".

2 $\sqrt{n}$ coloring

(a) Let G be a graph of maximum degree Δ. Show that G is $(\Delta + 1)$-colorable.

(b) Suppose G is a 3-colorable graph. Let v be any vertex in G. Show that the graph induced on the neighborhood of v is 2-colorable. *Clarification: the graph induced on the neighborhood of v refers to the subgraph of G obtained from the vertex set V' comprising vertices adjacent to v (but not v itself) and edge set comprising all edges of G with both endpoints in V'.*

(c) Give a polynomial time algorithm that takes in a 3-colorable n-vertex graph G as input and outputs a valid coloring of its vertices using $O(\sqrt{n})$ colors. Prove that your algorithm is correct and also analyze its runtime.
Hint: think of an algorithm that first colors "high-degree" vertices and their neighborhoods, and then colors the rest of the graph. The previous two parts might be useful.

Solution:

(a) Let the color palette be $[\Delta + 1]$. While there is a vertex with an unassigned color, give it a color that is distinct from the color assigned to any of its neighbors (such a color always exists since we have a palette of size $\Delta + 1$ but only at most Δ neighbors).

(b) In an induced graph, we only care about the direct vertices adjacent to v and v itself. In any 3-coloring of G, v must have a different color from its neighborhood and therefore the neighborhood must be 2-colorable.

(c) While there is a vertex of degree-$\geq \sqrt{n}$, choose the vertex v, pick 3 colors, use one color to color v, and the remaining 2 colors to color the neighborhood of v (2-coloring is an easy problem). Never use these colors ever again and delete v and its neighborhood from the graph. Since each step in the while loop deletes at least $\sqrt{n}$ vertices, there can be at most $\sqrt{n}$ iterations. This uses only $3(\sqrt{n} + 1)$ colors. After this while loop is done we will be left with a graph with max degree at most $\sqrt{n}$. We can $\sqrt{n} + 1$ color with fresh colors this using the greedy strategy from the solution to part a. The total number of colors used is $O(\sqrt{n})$.

3 Cuts from Colors

Given a graph $G = (V, E)$ on n vertices and m edges, and a vector $x \in \{\pm 1\}^n$, we say

$$\mathsf{Cut}(G, x) := \frac{1}{m} \sum_{\{i,j\} \in E} \left(\frac{x_i - x_j}{2} \right)^2$$

and define

$$\mathsf{MaxCut}(G) := \max_{x \in \{\pm 1\}^n} \mathsf{Cut}(G, x).$$

For every algorithmic question below, please analyze your runtime and prove that your algorithm is correct.

(a) (Warmup; ungraded) Let G be any graph. Prove that

$$\mathsf{MaxCut}(G) \geq \frac{1}{2}$$

always, and give a polynomial time randomized algorithm that outputs $\boldsymbol{x} \in \{\pm 1\}^n$ satisfying $\mathbf{E}[\mathsf{Cut}(G, \boldsymbol{x})] \geq \frac{1}{2}$.

What if each x_i is chosen to be $+1$ or -1 uniformly independently? **Solution:** Let $\boldsymbol{x} \sim \{\pm 1\}^n$ and apply linearity of expectation.

(b) Let G be any graph. Prove that

$$\mathsf{MaxCut}(G) \geq \frac{1}{2} + \Omega\left(\frac{1}{n}\right)$$

always, and give a polynomial time randomized algorithm that outputs $\boldsymbol{x} \in \{\pm 1\}^n$ satisfying $\mathbf{E}[\mathsf{Cut}(G, \boldsymbol{x})] \geq \frac{1}{2} + \Omega\left(\frac{1}{n}\right)$.

Hint: try to construct a simple distribution over ± 1 vectors such that for $\boldsymbol{x}$ drawn from this distribution any $i, j \in [n]$, $\mathbf{E}[\boldsymbol{x}_i \boldsymbol{x}_j] = -\frac{c}{n}$ for some absolute constant $c > 0$.

Solution: The idea in this problem is to sample a random balanced cut. Let $\boldsymbol{x} \sim \{y : y \in \{\pm 1\}^n, \sum_{i=1}^n y_i = 0\}$ (restrict the sum to be 1 if n is an odd number). This distribution can be sampled from in polynomial time by just picking the $n/2$ nodes which will be assigned 1. Note that for any $i \neq j$, if $\boldsymbol{x}_i = 1$ then $\Pr[\boldsymbol{x}_j = 1] = \frac{n/2-1}{n}$ whereas $\Pr[\boldsymbol{x}_j = -1] = \frac{n/2}{n} = \frac{1}{2}$; a similar argument applies to the opposite case. So, $\mathbf{E}[\boldsymbol{x}_i \boldsymbol{x}_j] = \frac{-1}{n}$. The overall result can be found by applying linearity of expectation.

(c) Let G be any k-colorable graph. Prove that

$$\mathsf{MaxCut}(G) \geq \frac{1}{2} + \Omega\left(\frac{1}{k}\right).$$

Note that we are not asking you to give an algorithm to find such a cut, but instead just asking you to prove existence. Try to reduce this problem to the previous part. **Solution:** Consider the graph G_k obtained by collapsing each color class to a super-vertex and placing an edge of weight equal to the total weight of edges between corresponding color classes. Apply the same algorithm from the previous part to G_k, and assign every vertex in a color class the same sign as its super-vertex; any edge in G_k is cut with probability $\frac{1}{2} + \Omega\left(\frac{1}{k}\right)$.

(d) Let G be any 3-colorable graph. Give a polynomial time randomized algorithm to find a $\boldsymbol{x} \in \{\pm 1\}^n$ satisfying:

$$\mathbf{E}[\mathsf{Cut}(G, \boldsymbol{x})] \geq \frac{1}{2} + \Omega\left(\frac{1}{\sqrt{n}}\right).$$

Hint: Part (b) of Question 2 may be useful.

Solution: Use the algorithm from question 2 c to $O(\sqrt{n})$ color the graph and apply the solution from the previous part.

(e) Let G be any graph with maximum degree Δ. Prove that

$$\mathsf{MaxCut}(G) \geq \frac{1}{2} + \Omega\left(\frac{1}{\Delta}\right).$$

Give a polynomial time randomized algorithm to find a $\boldsymbol{x} \in \{\pm 1\}^n$ satisfying:

$$\mathbf{E}[\mathsf{Cut}(G, \boldsymbol{x})] \geq \frac{1}{2} + \Omega\left(\frac{1}{\Delta}\right).$$

Hint: Part (a) of Question 2 might be useful here. **Solution:** Use the greedy algorithm from 2 a to $\Delta + 1$ color G and then apply the solution given in part c.

4 Reservoir Sampling

(a) Design an algorithm that takes in a stream $z_1, \ldots, z_M$ of M integers in $[n]$ and at any time t can output a uniformly random element in $z_1, \ldots, z_t$. Your algorithm may use at most polynomial in $\log n$ and $\log M$ space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream. *Hint:* $\frac{1}{t} = 1 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} \cdots \frac{t-1}{t}$.

(b) For a stream $S = z_1, \ldots, z_{2n}$ of $2n$ integers in $[n]$, we call $j \in [n]$ a *duplicative element* if it occurs more than once. Prove that S must contain a duplicative element, and design an algorithm that takes in S as input and with probability at least $1 - \frac{1}{n}$ outputs a duplicative element. Your algorithm may use at most polynomial in $\log n$ space. Prove the correctness and analyze the space complexity of your algorithm. Your algorithm may only take a single pass of the stream.

Solution:

(a) Maintain a counter n_1 initially 0 to keep track of how many elements have arrived so far and maintain a "current element" x initially 0. When an element z arrives, increment n_1 by 1 and replace x with z with probability $1/n_1$. When queried at any time, output x. To analyze the probability that z_t is outputted at time $t' > t$, observe that z_t must be chosen and then never replaced. This happens with probability

$$\left(\frac{1}{t}\right) \cdot \left(\frac{t}{t+1} \cdot \frac{t+1}{t+2} \cdots \frac{t'-1}{t'}\right).$$

(b) S must contain a duplicative element by the pigeonhole principle. The algorithm is to maintain $\log n$ independent instantiations of the sampling algorithm from part (a) and when a new element z arrives at time t, first query all the instantiations and if any of them outputs $(z, *)$, ignore the rest of the stream and output z as the 'duplicative element' at the end of the stream. Otherwise, stream (z, t) as input to each of the independent instantiations of the sampling algorithm and continue. If the stream ends without the algorithm ever 'committing' to a z, output 'failed'. Note that if the algorithm returns an element, it is certainly a duplicative element. We now turn our attention to upper bounding the probability that the algorithm returns 'failed'. Suppose the algorithm returns failed, then let $(x_1, t_1), \ldots, (x_{\log n}, t_{\log n})$ be the samples held by each of the instantiations at the end of the stream. Each t_i is a uniformly random number between 1 and $2n$. Note that for the algorithm to have failed t_i must be the time of the final occurrence of element x_i (otherwise x_i would have been identified as a duplicative element in the next occurrence of x_i). Since there are at most n distinct values for x_i, there are at most n distinct times t such that the element z that arrived at time t is the final occurrence of z in the stream. Thus, the probability that t_i is the timestamp of a final occurrence is bounded by $1/2$. Thus the probability of the algorithm failing is bounded by $1/2^{\log n} = 1/n$.