CS 388R: Randomized Algorithms, Fall 2023 August 21st, 2023

Lecture 1: Introduction; min-cut

Prof. Eric Price *Scribe: Trung Nguyen, Daniel Kuddes*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

This lecture introduced some basic concepts related to randomized algorithms, and gave two examples about randomized algorithms which are quicksort and Karger's algorithm for global min-cut.

2 Randomized Algorithms

Roughly speaking, a randomized algorithm is an algorithm that makes random choices that are independent of inputs by making use of a random number at one or more points during its process to make decisions.

2.1 Benefits of Randomized Algorithms

We can think of a randomized algorithm (RA) as a deterministic algorithm with random seeds. For any fixed input, there are usually many bad seeds, but on average or with high probability over seeds, the outcome is good. We might say a randomized algorithm works on all inputs and most seeds.

Benefits of RA:

- Simplicity: many randomized algorithms are simpler to describe and implement than deterministic algorithms with same performance.
- Speed: many randomized algorithms run faster than best known deterministic algorithms.
- Robust to adversarial input: Randomized algorithms can get guarantees that are impossible in worst-case for deterministic algorithms.

3 Quicksort Example

The first example of a randomized algorithm we saw in this class is quicksort. Consider the sorting problem. Given a set of n real numbers, sort them in ascending order. Here is the description of the well-known quicksort algorithm. Let the inputs set be $I = \{x_1, \ldots, x_n\}$. The algorithm chooses a random pivot t between 1 and n. Then, it partitions the inputs into three disjoint sets: $I_1 = \{x_i | x_i < x_t\}, I_2 = \{x_t\}, I_3 = \{x_i | x_i \geq x_t, i \neq t\}$. Thus, $quicksort(I) = (quicksort(I_1), x_t, quicksort(I_3))$.

3.1 Quicksort Pseudocode

Algorithm 1 `QuickSort`(X)

Input: List X
Choose random pivot $t \in$ `range(len(`X`))`
return `QuickSort`$([X_i | X_i < X_t]) + [X_t] +$ `QuickSort`$([X_i | X_i > X_t])$

3.2 Quicksort Running Time

We now analyze the expected running time of quicksort. Let T be the running time of quicksort. Observe that T is proportional to the number of comparisons we make during the sorting process. Suppose the sorted sequence is $y_1 \leq \ldots \leq y_n$. Define random variables $Z_{i,j}, 1 \leq i < j \leq n$ as follows.

$$Z_{i,j} = \begin{cases} 1 & \text{if } y_i \text{ and } y_j \text{ are compared.} \\ 0 & \text{otherwise.} \end{cases}$$

Then, $T = \Theta(\sum Z_{i,j})$.
Observe that $Z_{i,j} = 1$ if and only if the first pivot element chosen among $y_i, y_{i+1}, \ldots, y_j$ is y_i or y_j, and each element $y_i, y_{i+1}, \ldots, y_j$ is equally likely to be the first among these elements to be chosen as a pivot element. So, $\mathbb{P}[Z_{i,j} = 1] = 2/(j - i + 1)$. We have

$$\begin{aligned} \mathbb{E}[T] \propto \mathbb{E}[\sum_{i<j} Z_{i,j}] = \sum_{i<j} \mathbb{E}[Z_{i,j}] &= \sum_{i<j} \frac{2}{j-i+1}. \\ &= 2\sum_{i=1}^{n-1} \left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-i+1}\right) \\ &< 2\sum_{i=1}^{n-1} \left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) < 2n \log n. \end{aligned}$$

Hence, $\mathbb{E}[T] = O(n \log n)$.

4 Types of Randomized Algorithms

4.1 Las Vegas

These algorithms always produce a correct result, but their runtime is not deterministic.

- Always correct
- Random amount of time to complete
- Examples: Quick Sort, Hash Table

4.2 Monte Carlo

These algorithms usually complete in a predictable amount of time, but there's a small probability they might produce an incorrect result.

- Usually correct
- Fixed run time
- Examples: Karger's min-cut algorithm, Estimating π

Note: Las Vegas can become Monte Carlo by adding a stopping time and checker

5 Karger's Min-cut Algorithm

5.1 Min Cut Problem

Given an undirected graph with n vertices and m edges, we wish to partition the vertices of graph G into two nonempty sets $(S, \overline{S})$ so as to minimize the number of edges crossing between them.

More formally, a cut $(S, \overline{S})$ of a graph G is a partition of the vertices of G into two nonempty sets S and $\overline{S}$. An edge (v, w) crosses cut $(S, \overline{S})$ if one of v and w is in S and the other in $\overline{S}$. The value of a cut is the number of edges that cross the cut.

5.2 Naive Solution use Ford-Fulkerson

Given that value of the maximum flow will be equal to the min cut of a graph, we can use Ford-Fulkerson to find the min cut for every source and drain pair.

lemma: The value of the maximum flow is equal to the minimum cut.

5.3 Ford-Fulkerson Review

Inputs: Given a Network $G = (V, E)$ with flow capacity C, a source node s, and a sink node t.

Output: Compute a flow f from s to t of maximum value.

Algorithm 2 Ford-Fulkerson Algorithm for Maximum Flow (G, s, t)

Require: Flow network G, Source node s, Sink node t
Ensure: Maximum flow from s to t
  Initialize flow f in G to be 0
  while there exists an augmenting path p in the residual graph G_f **do**
    Find bottleneck capacity $c_f(p)$ on path p
    for each edge (u, v) in path p **do**
      if (u, v) is a forward edge in G **then**
        $f(u, v) = f(u, v) + c_f(p)$
      else
        $f(v, u) = f(v, u) - c_f(p)$
      end if
    end for
  end whilereturn flow f

5.4 Run Time

Runs in $O(n^3)$. To run for all possible s and t pairs requires n^2 time, so the running time of using Ford-Fulkerson to select the minimum max flow for all possible s and t pairs takes $O(n^5)$ Let's see if we can do better.

6 Karger's Algorithm

Algorithm 3 Karger's Min Cut (G, s, t)

Require: Flow network G, Source node s, Sink node t
Ensure: Maximum flow from s to t
  Initialize flow f in G to be 0
  while $n > 2$ **do**
    Choose a random edge
    Contract the edge ▷ merge the vertices and remove loops
  end whilereturn preimage of the two remaining vertices

6.1 Run Time

Claim: probability of returning the min cut in one run is $\theta(1/n^2)$

Let $d(u)$ denote the degree of vertex u. failure in this context means contracting edges in the min cut before all other edges.

First Step,

$$\begin{aligned}\mathbb{P}(\text{failure on first step}) =& \frac{\mathbb{E}(S,\overline{S})}{m}\\ \leq& \frac{\min d(u)}{m}\\ \leq& \frac{\frac{1}{n}\sum d(u)}{m} = \frac{2}{n}\end{aligned}$$

Second Step,

$$\mathbb{P}[\text{failure on second step— first step success}] \leq \frac{2}{n-1}$$

ith Step,

$$\mathbb{P}[\text{failure on } i\text{th step—success on 1 through } i\text{th }] \leq \frac{2}{n-i}$$

Final Step,

$$\mathbb{P}[\text{succeed in the all of steps}] \geq \prod_{i=1}^{n-2}\left(1-\frac{2}{n+1-i}\right) = \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdots\frac{2}{4}\cdot\frac{1}{3} = \frac{2}{n(n-1)} \geq \frac{2}{n^2}$$

6.2 Repeating n^2 times

When n is large, this guarantee is poor. However, if we repeat n^2 times and return the best result, then the failure probability becomes

$$\left(1-\frac{2}{n^2}\right)^{n^2} \approx \frac{1}{e^2} > \frac{2}{3}$$

The time complexity is $n^2 m\alpha(n) = n^2(m\log_{m/n} n)$ by Union-Find/Disjoint-set data structure whose time complexity is $O(\alpha(n))$.