Lecture 2: Sept 13

Lecturer: Mohammad R. Salavatipour *Scribe: Zac Friggstad*

2.1 Quicksort Analysis

The Quicksort algorithm on a set A is given as:

Quicksort(A)
  if A has only one element **then return**
  pick a pivot $a_i \in A$
  partition $A \setminus \{a_i\}$ into sets B_L and B_R such that:
    1) $b_l < a_i$ for all $b_l \in B_L$
    2) $a_i < b_r$ for all $b_r \in B_R$
  call *Quicksort*(B_L)
  output a_i
  call *Quicksort*(B_R)

Forming sets B_l and B_r takes $\Theta(n)$ time. If we are able able to pick the median as the pivot in at most $O(n)$ time then the time complexity $T(n)$ of *quicksort* on input A of size n satisfies the recurrence

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

with a base case of $\Theta(1)$. By the master theorem as presented in [1], this recurrence has a solution of $\Theta(n \log n)$. However, picking the median is not so trivial. We are going to rely on the fact that "even if the pivot is not the median but close to the median then the running time should be still $\Theta(n \log n)$". Using this intuition, instead of picking the median as the pivot we pick a random element $a_i \in A$ uniformly. To analyze the behaviour of *Quicksort* with this pivot selection strategy it is useful to define indicator variables $X_{i,j}, 1 \le i < j \le n$ where

$$X_{i,j} = \begin{cases} 1 & \text{if elements } x_i \text{ and } x_j \text{ are ever compared,} \\ 0 & \text{otherwise} \end{cases}$$

for $\{x_1, x_2, \ldots, x_n\} = A$ where $x_1 < x_2 < \ldots < x_n$. Notice that two elements can be compared at most once since one must be the pivot if they are compared and the pivot is not found in either B_L or B_R. Thus the expected number of comparisons is

$$E[\sum_{i<j} X_{i,j}] = \sum_{i<j} E[X_{i,j}] = \sum_{i<j} P_{i,j}$$

by the linearity of expectation where $P_{i,j}$ is the probability that elements x_i and x_j are compared.

To determine this probability, consider the smallest recursive call which contains both elements x_i and x_j. In this call, x_i and x_j have not been compared yet since none has been the pivot yet. Moreover, none of the elements $x_i, x_{i+1}, \ldots, x_j$ have been picked as the pivot yet. If this were not true then x_i and x_j would have been placed into different partitions at an earlier recursive call. Since this is the smallest recursive call that contains both x_i and x_j then one of $x_i, x_{i+1}, \ldots, x_j$ must have been picked as the pivot. If neither x_i nor x_j is picked then x_i and x_j will never be compared. Thus the probability of comparing x_i and x_j is the probability of picking one of them as the pivot from $x_i, x_{i+1}, \ldots, x_j$ in this recursive call. This probability is exactly $\frac{2}{j-i+1}$.

The expected number of comparisons is then conditioned by

$$\begin{aligned}
\sum_{i<j} P_{i,j} &= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=2}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \\
&\leq 2\sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2\sum_{i=1}^{n} H_n = 2nH_n
\end{aligned}$$

where H_n is the n'th harmonic number defined by

$$H_n = \sum_{i=1}^{n} \frac{1}{i}.$$

As a consequence of the following proposition, the expected number of comparisons in total is $\Theta(n \ln n)$.

Proposition 2.1 $H_n = \ln n + \Theta(1) = \Theta(\ln n)$.

Proof: Since $\frac{1}{x}$ is a positive, decreasing function on the interval $(0, \infty)$ we have

$$\int_1^{n+1} \frac{1}{x}\,dx \;\leq\; \sum_{k=1}^{n} \frac{1}{k} \;\leq 1 \;+ \int_2^{n+1} \frac{1}{x-1}\,dx$$

or

$$\ln(n+1) \leq H_n \leq 1 + \ln n.$$

Therefore $H_n = \ln n + O(1)$ and $H_n \geq \ln(n+1) \geq \ln n$. If we look closely at Riemann sum H_n as compared with the area under $\frac{1}{x}$ from 1 to $n+1$ the error of the approximation $H_n \approx \ln n$ is lower bound by $H_1 - \ln 2 > 0$ (the error for the approximation of H_1). Since the error is lower bounded by a positive constant we have $H_n = \ln n + \Omega(1)$. Therefore, $H_n = \ln n + \Theta(1)$. The weaker statement $H_n = \Theta(\ln n)$ follows immediately. ■

Note that this analysis of *Quicksort* holds for *any* input distribution. Also, this algorithm always produces a correct solution. It is just the running time that depends on the random selections made during the execution. Also note that if the pivot is chosen to be the smallest element of the set at each recursive call, then there will be

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \Theta(n^2)$$

comparisons in total. The worst case of this algorithm is then $\Theta(n^2)$ whereas the average case is $\Theta(n \ln n)$.

2.2 A Randomized Min-Cut Algorithm

Definition 2.2 *A* cut *of a graph $G(V, E)$ is a partition of V into two non-empty subsets S and $\bar{S}$. The size of a cut is measured as the sum of the weights of the edges having one endpoint in S and the other endpoint in $\bar{S}$. In the case of an unweighted graph, each edge is said to have weight 1.*

Sometimes we refere to a cut as the set of edges $C \subseteq E$ that are the edges with one endpoint in a partition S of V and another endpoint in the other partition $\bar{S} = V - S$. Not all subsets of E describe a valid cut.

Definition 2.3 *A* min-cut *of a graph $G(V, E)$ is a cut $(S, \bar{S})$ such that for all other cuts $(S', \bar{S}')$, the size of $(S', \bar{S}')$ is no less than the size of $(S, \bar{S})$.*

An optimization problem in graph theory is to find a min-cut for a given graph $G(V, E)$. One of the common variants of this problem is: given such a graph with two distinct vertices s and t, the source and sink respectively, to find a min-cut $(S, \bar{S})$ with $s \in S$ and $t \in \bar{S}$. The max-flow/min-cut theorem states that the maximum flow from s to t is precisely the cost of the minimum cut that separates s and t. Polynomial time network flow algorithms exist that find the maximum flow so, as a consequence, finding the min-cut separating s and t is a polynomial time problem as well. To determine the size of the min-cut of the entire graph, we can try all $O(|V|^2)$ pairs of vertices and find the minimum cut separating these two vertices. Taking the minimum size of all of these answers we have a min-cut algorithm for G in $O(|V|^2 F(|V|, |E|))$ where $F(n, m)$ is the cost of the network flow algorithm with n vertices and m edges. To date, the best deterministic algorithm for finding the maximum flow between two vertices s and t takes $O(|V||E| \log \frac{|V|^2}{|E|})$ time. A quick improvement over the above mentioned algorithm comes from noting that: picking an arbitrary vertex as the source s, it must belong to one part of a minimum cut. Therefore, it is enough to try all other vertices as the sink and solve the minimum s, t-cut problem, and then take the minimum. This reduces the running time to $O(|V| F(|V|, |E|))$. With some skill, finding the min-cut for a graph can be solved in simply $O(F(|V|, |E|))$ time, but this algorithm can be slow in practice on large inputs.

Here we present a min-cut algorithm for unweighted graphs. Karger and Stein present the following algorithm in [2] for finding a cut of minimum size in an unweighted graph. However, we need one more definition before we proceed.

Definition 2.4 *A* contraction *of an edge (v_1, v_2) in a multigraph $G(V, E)$ is the formation of the multigraph $G'(V', E')$ by merging v_1 and v_2 into a common vertex, deleting any edges between v_1 and v_2 and attaching all other edges that that shared an endpoint with one of v_1 or v_2 to this new vertex. The result is denoted by $G/(v_1, v_2)$.*

Note that if G is a graph, the process of contraction might form a multigraph. For the sake of implementation efficiency we can simply keep a count of the number of edges between two vertices so that each vertex has, at most, $n - 1$ adjacent edges to keep track of. Now the algorithm can be presented.

simple_min-cut(G)
    $i \leftarrow 0$, $G_0 \leftarrow G$
    while G_i has more than 2 vertices **do**
        pick a random edge e_i uniformly from G_i
        $G_{i+1} \leftarrow G_i / e_i$
        $i \leftarrow i + 1$
    output edges of G_i

Assuming $|V| = n$, the loop runs exactly $n-2$ times since the number of vertices decreases by exactly 1 in each step. Since we are only keeping a count of the number of edges between two vertices in the multigraph G_i for all $0 \le i \le n-2$ then the number of adjacent edges to any vertex of any G_i is $O(n)$. Thus, the contraction step can be done in $O(n)$ time. So the running time of the algorithm is $O(n^2)$.

Before determining the probability that a min-cut was returned, a few lemmas are presented first.

Lemma 2.5 *Every cut in $G_i, 0 \le i \le n-1$ corresponds to a cut of the same size in G.*

Proof: The proof proceeds by induction on i. Obviously any cut in G_0 corresponds to a cut in G since they are the same graphs. Assume that any cut of G_i $(i \ge 0)$ corresponds to a cut of G. We prove any cut of G_{i+1} corresponds to a cut of G_i, too and this completes the induction step. To see this result, consider the edge $e = v_1v_2$ that was contracted (into a vertex v') in the formation of G_{i+1} from G_i. Any cut $(S', \bar{S}')$ in G_{i+1} has the new merged vertex v' in one of the partitions, wlog say in S'.

Consider the cut $(S, \bar{S})$ of G_i where $\bar{S} = \bar{S}'$ and $S = (S' - v')\bigcup\{v_1, v_2\}$. The claim is that the size of this cut is the same as that of $S', \bar{S}'$. The reason is that any edges that were deleted had both end-points in S. Thus the size of $S, \bar{S}$ cut has not changed in going to $S', \bar{S}'$. ■

Corollary 2.6 *If the min-cut of G has size k then the min-cut of G_i is of size at least k.*

Proof: If there was a cut of G_i that was of size less than k then, by the previous lemma, a cut of this size must also exist in G. This contradicts the fact that the min-cut of G is of size k. ■

Lemma 2.7 *In any graph $G(V, E)$ with min-cut of size k every vertex has degree at least k.*

Proof: If some vertex v had degree less than k then the cut $(\{v\}, V - \{v\})$ is of size $< k$ which contradicts the assumption that the min-cut size of G is k. ■

The following results are immediate from these lemmas.

Corollary 2.8 *If the min-cut of G is of size k then G has at least $\frac{nk}{2}$ edges.* ■

Corollary 2.9 *G_i has at least $\frac{(n-i)k}{2}$ edges.* ■

One last immediate observation is that if $C \subseteq E$ is a cut in G then the algorithm returns C if and only if none of the edges of C were contracted. Let C be any fixed minimum cut of G and let $k = |C|$. The probability that C is returned is the probability that no edge of C is ever contracted. Let $\mathcal{E}_i$ be the event that we didn't pick an edge of C in iteration i. Then

$$Pr[\mathcal{E}_1] = 1 - k/\frac{nk}{2} = 1 - \frac{2}{n}.$$

Extending this we see

$$Pr[\mathcal{E}_i | \bigcap_{j=1}^{i-1} \mathcal{E}_j] = 1 - \frac{2}{n-i+1}.$$

Since each all of these events are independent, the probability of $\mathcal{E}$, the event that no edge of C is ever removed is simply

$$\begin{aligned} Pr[\mathcal{E}] &= \prod_{i=1}^{n-2} Pr[\mathcal{E}_i | \bigcap_{j=1}^{i-1} \mathcal{E}_j] \\ &= \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) \\ &= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) \\ &= \frac{2}{n(n-1)} \end{aligned}$$

since the product is a telescoping product in that the numerator of one term i cancels with the denominator of the term $i+2$. So the probability of the min-cut surviving this algorithm is $\Omega(n^{-2})$. In other words, the probability of failure is no more than $1 - \frac{2}{n(n-1)}$.

If the algorithm is repeated t times, then the probability of failure is at most

$$\left(1 - \frac{2}{n(n-1)}\right)^t \leq \left(1 - \frac{2}{n^2}\right)^t.$$

Considering the fact that

$$\left(1 + \frac{c}{x}\right)^x \leq e^c$$

the probability of failure after running the algorithm $n^2 \ln n$ times is less than $e^{-2\ln n} = O\left(n^{-2}\right)$.

2.3 Types of Randomized Algorithm

There are two main classes of randomized algorithms.

Definition 2.10 *A randomized algorithm is a* Las Vegas *algorithm if the output answer is always correct but the running time is dependant on random decisions.*

The *quicksort* algorithm presented in this lecture is exactly this type of answer. It will always output the elements in sorted order, but it could take anywhere from $\Theta(n \ln n)$ time to $\Theta(n^2)$ time where $\Theta(n \ln n)$ is the expected number of comparisons and $\Theta(n^2)$ is the worst case number of comparisons. $\Theta(n \log n) = \Theta(n \ln n)$ is also a lower bound for the number of comparisons since, in the best case of the median always being picked, the master theorem tells us that this is the running time.

Definition 2.11 *A randomized algorithm is a* Monte Carlo *algorithm if the running time is fixed but the output is correct with some probability less than one.*

If an algorithm is both Las Vegas and Monte Carlo we still say the algorithm is Monte Carlo. An example of a Monte Carlo algorithm is the min-cut algorithm presented in this lecture. The running time is bound by $O(n^2)$ but, in general, an incorrect answer may be returned with some bounded probability.

References

1 T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms, 2nd Edition*, MIT Press, Cambridge, Massachusetts, 2001.

2 D.R. Karger and C. Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. *Proceedings of the 25th Annual ACM SYmposium on Theory of Computing*, 1993, pp. 757-765.