Randomized Algorithms 2014/5A
Lecture 2 – Maximal Independent Set, Analysis of Randomized Quicksort, Universal Hashing and Dictionaries, Concentration bounds- Chernoff [*]

Moni Naor

1 Maximal Independent Set

We considered a distributed model of computing where nodes in a graph contain processors and they can talk to their immediate neighbors. The goal now is to collectively compute something. In particular for the maximal independent set problem the goal is to a pick a set of nodes S such that S is a maximal independent set of the underlying graph and each node determines whether or not it is in S. In this model we may need randomness simply to break symmetry (think of a cycle), but even if processors have unique id's (so the greedy algorithm can be executed) no local deterministic algorithm is known (one that take polylog in n time, regardless of the graph). The algorithm we say in a variant of a 1986 algorithm by Mike Luby [5], see lecture notes by Wattenhaffer [6]. The expected number of edges removed in each phase is at least 1/3 and the expected number of rounds until all nodes know wether there are in the MIS or not is $O(\log n)$.

A question I raised in class is whether there are *Pseudo-deterministic* algorithm for MIS. An algorithm is called Pseudo-deterministic is it gives the same output with high probability. For a decision problem it simply means that it decides consistently whp. But what if the output is a larger object such as a distributed set. On second thought the problem is rather hard.

Claim 1. *If there is a Pseudo-deterministic algorithm for MIS then there is also a deterministic one that operates in the same amount of time.*

2 Nuts and Bolts and QuickSort

We saw the nuts and bolts puzzle and that the natural algorithm for solving ti corresponds to quicksort where the pivot is chosen at random. We saw a slick argument showing that the *expected*

[*]These notes summarize the material covered in class, usually skipping proofs, details, examples and so forth, and possibly adding some remarks, or pointers. The exercises are for self-practice and need not be handed in. In the interest of brevity, most references and credits were omitted.

time is $2n \ln n + O(n)$. The argument was based on analyzing the probability that element i and j are compared (this is $2/(j-i+1)$ and then by the *linearity of expectation* it is a Harmonic sum.

For both algorithms in order to get a high probability type we need a Chernoff type inequality. The are several of them and we will bring here the one we use for both. These bounds talk about the probability that a random variable which is a sum of small independent events is far away from its expectation.

Theorem 2. *Let $X_1, X_2, \dots X_n$ be independent Poisson trials such that $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. For $0 < \delta < 1$ we have*

$$\Pr[|X - \mu| \geq \delta\mu] \leq 2e^{-\mu\delta^2/3}.$$

See Chapter 4.2 and 4.3 of Mitzenmacher and Upfal.

We also used a *union bound*: the probability of a bad event that is the union of several bad ones is bounded by the sum of those events. In case of quicksort the bad event was that the algorithms takes too long and smaller events are that element i took a long time. These smaller events are not independent (in fact they are *positively correlated* for close i and j) but the union bound does not need that. The nice thing about it is that it is always applicable.

3 Hash Tables

Hash tables is very well studied subject in computer science and one of the more useful practices. Knuth's "The Art of Computer Programming" Volume 3 devotes a lot of space to the various possibilities and the origin of the idea is attributed to a 1956 paper by Arnold Dumey [4]. A major issue is how to resolve collisions and popular suggestions are chaining and Open addressing (or closed hashing). For the latter we need to specify a probing scheme and one of the more popular ones is linear probing which takes advantage of the locality properties of computer memory (in the various levels of hierarchy).

The 'modern' era of investigating hashing can be seen in the work of Carter and Wegman [2] who suggested the idea of thinking of the input as being worst case and the performance is investigated when the hash function is chosen at random from a predefined family (rather than assuming a truly random function).

The simplest way to obtain dictionaries with expected $O(1)$ per operation is to use chained hashing with a table of size $O(n)$. Here, it is enough to choose the hash function from a δ-universal family, for δ which is $O(1/n)$. The *expected length* of a chain is now $O(1)$ and the length of the chain is what determines the cost of an operation. Note however that there will be long chains. Universality on its own only suffices to guarantee that the expected length of the *longest* chain is $O(\sqrt{(}n))$. The upper bound follows from considering all potential collisions: there are $\binom{n}{2}$ of them. Each collision occurs with probability $O(1/n)$, so the expected number of collisions is $O(n)$. On the other hand, in a chain of length ℓ there are $\binom{\ell}{2}$ collisions. Therefore the expected length of the longest chain cannot be larger than $O(\sqrt{(}n))$. For the lower bound see Alon et al. [1].

What happens if the hash function is truly random? Then this is the "ball and bins" scenario which we will talk in the next lecture and here the heaviest bin/chain is likely to contain $\Theta(\log n/\log\log n)$ elements.

Universality on its own also just guarantees *expected* $O(1)$ performance and not, say, high probability $(1-1/poly(n))$ amortized $O(1)$ performance. There are several ways to obtain this sort of result. In future lecture we will explore "Cuckoo Hashing" a method that uses two hash functions h_1 and h_2 and where each element x in the set resides either in location $h_1(x)$ or location $h_2(x)$. This means that lookup requires just two accesses to the memory. Insertion may be more involved and requires relocating elements.

A lecture be Eric Demaine on hashing is available and recommended [3].

References

[1] Noga Alon, Martin Dietzfelbinger, Peter B. Miltersen, Erez Petrank, and Gabor Tardos, *Linear Hashing* Journal of the ACM, Vol. 46(5), 1999. http://www.brics.dk/RS/97/16/BRICS-RS-97-16.pdf

[2] J. L. Carter and M. N. Wegman, Universal classes of hash functions, J. Comput. Syst. Sci. 18 (1979) 143–154. http://www.cs.princeton.edu/courses/archive/fall09/cos521/Handouts/universalclasses.pdf

[3] Eric Demaine, Lecture 10 in course 6.851: Advanced Data Structures (Spring'12) on Dictionaries, https://courses.csail.mit.edu/6.851/spring12/lectures/L10.html

[4] Arnold Isaac Dumey, *Indexing for rapid random-access memory*, Computers and Automation 5 (12), 6–9, 1956.

[5] Michael Luby, *A Simple Parallel Algorithm for the Maximal Independent Set Problem*, SIAM Journal on Computing 15 (4): 10361053, 1986.

[6] Roger Wattenhofer, http://dcg.ethz.ch/lectures/fs10/podc/lecture/mis.pdf