CS 224: Advanced Algorithms Fall 2014

Lecture Lecture 25 — November 25, 2014

Prof. Jelani Nelson *Scribe: Keno Fischer*

1 Today

- Finish faster exponential time algorithms (Inclusion-Exclusion/Zeta Transform, Möbius inversion)
- Streaming/Sketching

2 Last time

Fact 1 (Inclusion-Exclusion)**.**

$$\forall R \subseteq T \qquad \sum_{R \subseteq S \subseteq T} (-1)^{|T \setminus S|} = [R = T]$$

Definition 2 (Zeta transform)**.** *Given a function*

$$f : \textbf{\textit{sets}} \rightarrow \textit{some ring}$$

let $\hat{f}$ *be the zeta transform given by*

$$\hat{f}(S) = \sum_{R \subseteq S} f(R)$$

Claim 3 (k-coloarbility)**.** *Define*

$$f(S) = [S \textit{ is a non-empty independent set }]$$

Then, a graph G *is* k*-colorable iff* $\sum_{S \in V} (-1)^{V \setminus S} (\hat{f}(S))^k > 0$

Will see this proof as special case of something more general:

- $\mathcal{O}^*(3^n)$ time, poly(n) space [BH06]
- $\mathcal{O}^*(2^n)$ time/space [Koi06]
- The first of the 2 papers also gets $\mathcal{O}^*(2.2461^n)$ time and poly(n) space
  OPEN: $\mathcal{O}^*(2^n)$ time, poly(n) space.

3 Zeta transforms, Möbius inversion

3.1 More generally

Definition 4. *Zeta transform, Möbius inversion*

Given

$$f : \textbf{\textit{sets}} \rightarrow \textit{some ring}$$

let the Zeta transform of $\hat{f}$ of f be given by

$$\hat{f}(S) = \sum_{R \subseteq S} f(R)$$

and define the Möbius inversion $\tilde{f}$ of f to be

$$\tilde{f}(S) = \sum_{R \subseteq S} (-1)^{|S \setminus R|} f(R)$$

Claim 5 (Zeta Inverse)**.**

$$\hat{\tilde{f}} = \tilde{\hat{f}} = f$$

Proof. We will show $\hat{\tilde{f}} = f$ (Showing $\tilde{\hat{f}}$ is similar)

$$\hat{\tilde{f}}(T) = \sum_{S \subseteq T} \tilde{f}(S)$$

so

$$\begin{aligned}
\hat{\tilde{f}}(T) &= \sum_{S \subseteq T} \sum_{R \subseteq S} (-1)^{|S \setminus R|} f(R) \\
&= \sum_{S \subseteq T} \sum_{R \subseteq S} (-1)^{|T \setminus R|} (-1)^{|T \setminus S|} f(R) \\
&= \sum_{R} (-1)^{|T \setminus R|} f(R) \left(\sum_{S} [R \subseteq S \subseteq T] (-1)^{|T \setminus S|} \right) \\
&= \sum_{R} f(R) (-1)^{|T \setminus R|} [R = T] \text{ (by Inclusion-Exclusion)} \\
&= f(R)
\end{aligned}$$

□

3.2 Proving Theorem 3 (k-colorability)

Define

$$g(S) = \#\text{ of ways to write S as a union of } k \text{ non-empty independent sets}$$

Fact 6. $g(V) > 0 \iff G$ *is k-colorable*

Claim 7. $g(V) = \tilde{\hat{g}}(V) = \widetilde{\hat{f}^k}(V)$ *(where f as in Theorem 3)*

Proof. We will show that $\hat{g}(S) = (\hat{f}(S))^k$. Under the definition of the zeta transform $\hat{g}(S)$ counts the number ways to construct k-independent subsets of S such that there union is contained in S (g counts the same for the union being equal to S). Note that any union of k non-empty independent subsets of S satisfies, this definition, so $\hat{g}(S)$ is given by the number of independent sets raised to the k-th power, which is precisely the right hand side.

□

Claim 8. *Can compute all $\hat{f}(S)$ (or $\tilde{f}(S)$) values sequentially in time $\mathcal{O}^*(3^n)$, poly(n) space*

Proof.

$$\hat{f}(S) = \sum_{R \subseteq S} f(R)$$

We want to compute $\hat{f}(S)$ for all S. Time up to star is $\sum_{S \subseteq T} 2^{|S|} = \sum_{i=1}^{n} \binom{n}{i} 2^i = 3^n - 1$

□

What about a faster algorithm?

3.3 Yates' algorithm

We can solve this problem in $\mathcal{O}^*(2^n)$ time/space using Yates' algorithm, which is a dynamic programming approach.

We want to compute $\hat{f}(S)\ \forall S \subset \{1, \ldots, n\}$.

Define

$$g(i, S) = \sum_{R \subseteq S} [S(i) = R(i)] \cdot f(R)$$

$$S(i) = S \cap \{i + 1, \ldots, n\}$$

Note: $\hat{f}(S) = g(n, S)$

Claim 9. *We can compute $g(i, S)$ using the recurrence*

$$g(i, S) = \begin{cases} f(S) & \text{if } i = 0 \\ g(i-1, S) + [i \in S] \cdot g(i-1, S \backslash \{i\}) & \text{if } i > 0 \end{cases}$$

hence we can compute $g(n, S)$ in space $\mathcal{O}^*(2^n)$ and time proportional to space as every computation if a constant number of recursive calls plus possibly a constant number of arithmetic operations.

Proof. We can write

$$g(i, S) = \sum_{\substack{R \subseteq S \\ i \in R}} [S(i) = R(i)] \cdot f(R) + \sum_{\substack{R \subseteq S \\ i \notin R}} [S(i) = R(i)] \cdot f(R)$$

We have two cases:

$i \notin S$: Since $i \notin S$ and $R \subseteq S$, $i \notin R$, so the first sum is zero. Now, since $i \notin S$, $S(i) = S(i-1)$ and similarly $R(i) = R(i-1)$, so the second sum is given by

$$\sum_{R \subseteq S} [S(i-1) = R(i-1)] \cdot f(R) = g(i-1, S)$$

$i \in S$: Since i is in both R and S, $S(i) = R(i)$ iff $S(i-1) = R(i-1)$.

So the first sum equals:

$$\sum_{R \subseteq S} [S(i-1) = R(i-1)] \cdot f(R) = g(i-1, S)$$

In the second sum we want R which agrees with S from i onward, so it agrees with $S \backslash \{i\}$ from i onward:

$$\sum_{R \subseteq S} [(S \backslash \{i\})(i-1) = R(i-1)] \cdot f(R) = g(i-1, S \backslash \{i\})$$

□

We can solve other problems using this technique

- min steiner tree
- find all k-colorable induced subgraphs
- For more applications, see Husfeldt, "Invitation to algorithmic uses of inclusionexclusion" [Hus11]

To get the actual k-coloring (rather than just deciding whether one exists), there is a reduction given in [BH06] that finds the k-coloring in $\mathcal{O}^*(2^n)$ time and space.

Computing the Möbius inversion is left as an exercise.

Streaming and Sketching

Final topic of class

For more in depth exposure take "Algorithms for big data".

First Streaming (small space data structures).

Model We have some high dimensional vector $x \in \mathbb{R}^n$ (or high dimensional matrix $X \in R^{n \times n}$). We want to support updates to x and queries about x.

Will look at one kind of update: turnstile model [Mut05]:

- Each update (i,v) causes $x_i \leftarrow x_i + v$
- x starts off as $\vec{0} \in \mathbb{R}^n$
- example queries:
  - What is x_i ?
  - What is $|\text{support}(x)|$

There's always the solution of either remembering x or the whole stream (so either $\approx n$ or m space).

(One) Goal: come up with algorithm that uses much less space than the trivial solution.

Example 10. *query*$(x) = |$*support*$(x)| = F_0$

All updates (i, v) *have* $v = 1$. *Can use exactly* n *bits of space (as a bitstring) or* $m \log n$ *(remembering an index takes* $\log n$ *space).*

Claim 11. *Any deterministic algorithm for* F_0 *requires* $\Omega(\min(n, m))$ *bits of space [AMS99].*

Proof. We will prove this via an *encoding argument*. Suppose we had a space S streaming algorithm, we will show how to then design a compression mapping n-bit string to S-bit strings.

Alice receives $\{0, 1\}^n$ and wants to compress it using F_0 algorithm A. Alice creates and artifical stream and uses the memory content of A as compressions.

Encode(x): Create stream containing $\{i | x_i = 1\}$ runs A and outputs memory contents of A.

Decode(x): Given memory contents of A, we will continue running A and assign the x_i as follows:

```
z = query(A)
for i = 1:n
    insert i into stream
    if query(A) == z
        x_i = 1
    else
        x_i = 0
    end
    z = query(Z)
end
```

□

So we can't do Deterministic/exact F_0. So what about:

- Deterministic/exact: IMPOSSIBLE
- Det./approx: IMPOSSIBLE

- Rand/exact: IMPOSSIBLE
- Rand/approx: POSSIBLE [FM85]

Intuition for why we can beat linear space:

Goal: Develop a randomized algorithm $\mathcal{A}$ s.t for all streams S, $\mathbb{P}(|A(s) - F_0|) > \epsilon \cdot F_0) < \frac{1}{3}$.

We will develop an **idealized** randomized algorithm [FM85]:

Suppose we are given a totally random uniform hash function $h : [n] \to [0, 1] \subset \mathbb{R}$. In practice we can get away with a reasonable hash function instead that does not take this much space.

Algorithm:

Let $z = \min_{i \in S} h(i)$. To answer query, output $\frac{1}{z} - 1$.

To improve the algorithm, we can pick $h_1, \ldots, h_R$ independent has functions where $R = \Theta(\frac{1}{\epsilon^2})$. Now, let $z_j = \min_{i \in S} h_j(i)$ and output $\frac{1}{\bar{z}} - 1$ where $\bar{z} = \frac{1}{R} \sum_{j=1}^{R} z_j$.

Let's look at h. We're hashing F_0 items into $[0, 1]$, so we expect them to be roughly evenly spaced with gap $\frac{1}{F_0+1}$, with

$$\mathbb{E}\, z = \frac{1}{F_0 + 1}$$

or more formally

$$\mathbb{E}\, z = \int_0^1 \mathbb{P}(z > x)\, \mathrm{d}x = \int_0^1 \mathbb{P}(\forall i \in S, h(i) > X) = \int_0^1 (1 - x) F_0\, \mathrm{d}x$$

$$\mathbb{E}\, z^2 = \frac{2}{(F_0 + 1)(F_0 + 2)}$$

$$\mathbb{P}(|z - \mathbb{E}\, z| > \epsilon \cdot \mathbb{E}\, z) - \frac{\mathrm{Var}[z]}{\epsilon^2 \cdot (\mathbb{E}\, z)^2} \leq \frac{\mathbb{E}\, z^2}{\epsilon^2 \cdot (\mathbb{E}\, z)^2}$$

Bibliography.

References

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, February 1999.

[BH06] Andreas Björklund and Thore Husfeldt. Inclusion–exclusion algorithms for counting set partitions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 575–582. IEEE Computer Society, 2006.

[FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.* 31(2), pages 182–209, 1985.

[Hus11] T. Husfeldt. Invitation to Algorithmic Uses of Inclusion-Exclusion. *ArXiv e-prints*, May 2011.

[Koi06] Mikko Koivisto. An $\mathcal{O}^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 583–590. IEEE, 2006.

[Mut05] S Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.