Advanced Algorithms - Mandatory Assignment 1

Frank Andersen [fand@itu.dk],
Alexander Mønnike Hansen [almh@itu.dk]

September 2020

Convolution and FFT

1. The company Zamazon occasionally runs contests: If you can put exactly k items into your cart to get a total price of exactly T, then you don't have to pay at checkout. Zamazon sells n different items. You crawled the Zamazon website and now have a list of all items and their prices.

   (a) In this part of the exercise, assume that the item prices are real numbers. Give an algorithm that finds k items of total price T or outputs correctly that this is not possible. Multiple copies of the same item are allowed. You may spend $O(k \cdot n^{\lceil k/2 \rceil} e \log n)$ arithmetic operations.

Answer:
$k = O(1)$
$k = 3, \longleftarrow$ It's the 3-SUM problem
- With this as base we have a well studied algorithm. And by trying all triples we can sum to T in $O(n^3)$. This is not good enough, as we are asked for at solution in $O(k \cdot n^{\lceil k/2 \rceil e \log n})$.
for $k = 3$ we have $O(k \cdot n^{\lceil 3/2 \rceil e \log n}) = O(n^2 \log n)$ which is better for large n's.

```
$O(n^2)$            $P \leftarrow$ create all pairs of items.
$O(n \log n)$       $L \leftarrow$ sort list of items by total price
$O(n^2)$            — for each pair: $(\forall p \in P)$ eg. $p = (p_1, p_2)$
— $O(\log n)$       using binary search check:
                    if $T - p_i - p_{i+1}$ in L return *items*
                    else *none*
```

So we end up with $O(n^2 \log n)$

But this do not cover k-Sum, so we have do rearrange a bit, so we can, solve it within $O(k \cdot n^{\lceil k/2 \rceil e \log n})$.

```
$O(n^{k/2})$                        $P \leftarrow$ create $k/2$-tuples of items.
$O(n^{k/2} \log n^{k/2})$           $L \leftarrow$ sort list of $k/2$-tuples by total price
$O(n^{k/2})$                        — for every $k/2$-tuples:
— $O(k \cdot \log n)$               using binary search check:
                                    if $T - \sum_{i \in t} price(i)$ in L return $items$
                                    else $none$
```

So we end up with $O(n^{k/2} \log n)$

(b) In this part of the exercise, assume that the item prices are integers smaller than $1000n$. Give an algorithm that decides correctly whether there are k items of total price T, in $O(k \cdot n \log n)$ arithmetic operations. Multiple copies of the same item are allowed.

Answer:
$O(k \cdot n \log n)$
$O(n \log n) \longleftarrow$ for $k = O(1)$
By using FFT
items: $s_1 \ldots s_n \leq 1000n$, just a constant so time $o(n)$
Can be written using either polynomials or convolutions, because it the same in that sense that the coefficient's you get from multiplication of to polynomials are the same as you would get out of convolutions.

We represent each items as it's own nominal, with degree of price s_i, and give it as an shared polynomial.
Define polynomial (to encode all items data available):

$$p(x) = \sum_{i=1}^{n} x^{s_i} \qquad deg(p) \leq 1000n$$

$$p^2(x) = \sum_{k=0}^{D} a_k \cdot x^k \qquad deg(p^2) \leq 2000n$$

$$a_k \neq 0 \text{ iif. } \exists_{i,j} : \quad s_i + s_j = k$$

Compute $p^k(x)$ and check if coefficients of $x^T x$ is $\neq 0$ in $p^k(x)$ in $O(\sqrt{k} \cdot n \log n)$ with FFT.

2. The Fourier transform of a polynomial and its inverse change between the coefficient representation

$$p(x) = a_0 + \ldots + a_{d-1}x^{d-1}$$

of a polynomial p and its point-value representation

$$(x_0, p(x_0)), \ldots, (x_{d-1}, p(x_{d-1}))$$

at evaluation points $x_i = {\omega_i}^d$ for $i = 0, \ldots, d-1$. The FFT algorithm shows that either of these representations can be computed from the other using $O(n \log n)$ arithmetic operations. In this exercise, we show that even for arbitrarily chosen evaluation points x_i in the point-value representation, the coefficient representation can still be computed from the point-value representation—albeit at higher computational cost. That is, given a list of point-value pairs

$$(x_0, y_0), \ldots, (x_d, y_d)$$

with $x_i \neq x_j$ for $i \neq j$, show how to determine the coefficients $a_0, \ldots, a_d$ of a polynomial p of degree d such that $p(x_i) = y_i$ holds for all $i = 0 \ldots d$. You may use that the matrix

$$\begin{pmatrix} x_0^0 & \cdots & x_0^d \\ \vdots & \ddots & \vdots \\ x_d^0 & \cdots & x_d^d \end{pmatrix}$$

has full rank whenever $x_0, \ldots, x_d$ are pairwise distinct numbers. Which running time do you achieve?

Answer:
In order to change from the point-value representation of a polynomial, to the coefficient representation, without using the roots of unity, we can use the following approach:

For each point-value pair, we create a standard polynomial of degree d, example of polynomial with $d = 4$:

$$y = a \cdot x^3 + b \cdot x^2 + c \cdot x + d \tag{1}$$

We then insert the point as x, and the value as y, for a point-value pair (3, 27), it would look like this:

$$\begin{aligned} 27 &= a \cdot 3^3 + b \cdot 3^2 + c \cdot 3 + d \\ &= a \cdot 27 + b \cdot 9 + c \cdot 3 + d \end{aligned} \tag{2}$$

We do this for each point-value pair, which gives us d such equations. We then solve these equations as a linear equation.

We do this by first ordering the the equations into an augmented matrix and then using Gauss Jordan elimination, to change it to reduced row echelon form, at which point we have recreated the coefficient representation of the original polynomial.

The running time of this approach is dominated by Gauss Jordan elimination which uses $O(d^3)$ time, where d is the degree of the polynomial.

---

3. Let n be a power of two, and let $\omega_0, \ldots, \omega_{n-1} \in \mathbb{C}$ be the n-th roots of unity, where $\omega_k = \cos(2\pi \cdot k/n) + i \sin(2\pi \cdot k/n)$.

$$D = \begin{pmatrix} \omega_0^0 & \ldots & \omega_0^{n-1} \\ \vdots & \ddots & \vdots \\ \omega_{n-1}^0 & \ldots & \omega_{n-1}^{n-1} \end{pmatrix}$$

It can be checked that the columns $D_*, 1, ..., D_*, n$ are a basis for the vector space $\mathbb{C}^n$, the so-called Fourier basis. Because the columns $D_*, 1, ..., D_*, n$ are a basis, every vector $v \in \mathbb{C}^n$ has a unique representation in this basis. This means that there is a vector of coefficients $\overrightarrow{v} \in \mathbb{C}^n$ such that

$$v = \sum_{j=1}^{n} \overrightarrow{v}(j) \cdot D_{*j}$$

(a) Given as input v, compute the matrix-vector product Dv with $O(n \log n)$ arithmetic operations. It will help to look again at the previous exercise. (Note that computing Av for general matrices A takes at least $O(n^2)$ operations. The particular structure of the matrix D shows that this is not necessary; it is not even needed to provide D as part of the input.)

Answer:

Creating matrix D with $n = 4$

$$D = \begin{pmatrix} \omega_0^0 & \omega_0^1 & \omega_0^2 & \omega_0^3 \\ \omega_1^0 & \omega_1^1 & \omega_1^2 & \omega_1^3 \\ \omega_2^0 & \omega_2^1 & \omega_2^2 & \omega_2^3 \\ \omega_3^0 & \omega_3^1 & \omega_3^2 & \omega_3^3 \end{pmatrix}$$

$$Dv = \begin{bmatrix} \omega_0^0 & \omega_0^1 & \omega_0^2 & \omega_0^3 \\ \omega_1^0 & \omega_1^1 & \omega_1^2 & \omega_1^3 \\ \omega_2^0 & \omega_2^1 & \omega_2^2 & \omega_2^3 \\ \omega_3^0 & \omega_3^1 & \omega_3^2 & \omega_3^3 \end{bmatrix} \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_3 \\ v_4 \end{bmatrix}$$

If we do this matrix vector multiplication as is, we see that for each root of unity or row in the Vandermonde matrix, we get a polynomial equation corresponding to evaluating a polynomial on that specific root of unity, where the values in the vector v, act as the coeeficients of that polynomial.

Knowing this, we realize that, rather than doing the regular matrix multiplication, which takes $O(n^2)$ time, we can instead use FFT, which generates the same outcome but in $O(n \log n)$ time. Because FFT calculates the roots of unity itself, the Matrix D is not needed at all, so we can simply run FFT on our vector v.

(b) Given as input v, compute the vector $\hat{v}$ with $O(n \log n)$ arithmetic operations.

Answer:

Since we concluded in part a, that the result of multiplying the matrix D with a vector, is the same as running FFT on that vector, we should be able to use FFT to go from the product of $\hat{v}$ and matrix D (v), back to $\hat{v}$. We do this by running FFT on v, in order to get $\hat{v}$ back, this is done with $O(n \log n)$ operations.

(c) Bonus: Give a description of (the real part of) column $D_{*,k}$ for $0 \leq k \leq n-1$ as an audio signal. What does D having full rank mean in terms of audio signals? What does the k-th entry of $\hat{v}$ mean in terms of audio?

Answer:

Algorithms using matrix multiplication

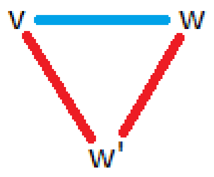
1. In this exercise, we extend the triangle detection algorithm from the lecture. Let $G = (V, E)$ be an undirected graph on n vertices. Recall that ω denotes the optimal exponent of matrix multiplication. You may use a subroutine that multiplies $t \times t$ matrices in time $O(t^\omega)$.

   (a) A *triangle* is a vertex subset $T \subseteq$ V of size 3 such that $uv \in E$ for all $u, v \in T$ with $u \neq v$. Give an algorithm that counts triangles in $O(n^\omega)$ time

   Answer:

   Adj. matrix A^2: 2-paths

   $A_{v,w}$

   $(A^2)_{v,w}$

   $\forall v \in V$:

   ① computer A^2: $O(n^\omega)$
   (computes the square of the adjacent matrix)
   ② for $v \in V$ :: $O(n^2)$
   for $w \in V$:
   checks neighbors, if exists increment count by 1

   So we end up with $O(n^\omega)$ as it dominates (n^2)

   (b) A k-clique is a vertex subset $T \subseteq V$ of size k such that $uv \in E$ for all $u, v \in T$ with $u \neq v$. Give an algorithm that decides whether G contains a k-clique in $O(n^{\omega\lceil k/3\rceil})$ time. (Hint: Split the clique into three equal-sized parts.)

   Answer:
   To find cliques in G of size $k = 3t$

① Enumerate all subsets of size $t \in V$ costs is $\binom{n}{t} = O(n^t)$

We can use this to create an auxiliary graph whose triangles corresponds to k-cliques in G.

```
② For each t-subset S: $O(n^t)$
      create a vertex $V_s$ in Aux if
         S is a t-clique in G
③ For every pair $(V_s, V_s')$ in Aux: $O(n^{2t})$
      connect by edge if all edges between S,S' is present in G.

④ Find $\triangle$ in Aux : $O(n^t)$
      true if $\triangle$ exists else false
```

So we get a total running time of: $O(n^{w \cdot t}) = O(n^{w/3 \cdot k})$

(c) Like above but *count* the cliques.

Answer:
Instead of returning in step ④, we just increment a counter for each $\triangle$ found.

Brainteaser (optional)

Spent way too much time on part 2 and 3, so we ran out of time :(.
