A Simple Near-Linear Pseudopolynomial Time Randomized Algorithm for Subset Sum

Ce Jin, Hongxun Wu

Tsinghua University

SOSA 2019

Subset Sum

Subset Sum

Given a (multi)set S of n positive integers and a target integer t, does there exist a subset $R \subseteq S$ that sums to exactly t?

Subset Sum

Subset Sum

Given a (multi)set S of n positive integers and a target integer t, does there exist a subset $R \subseteq S$ that sums to exactly t?

Example: $S = \{1, 1, 3, 10\}$, $t = 12$

Subset Sum

Subset Sum

Given a (multi)set S of n positive integers and a target integer t, does there exist a subset $R \subseteq S$ that sums to exactly t?

Example: $S = \{1, 1, 3, 10\}$, $t = 12$
YES! $R = \{1, 1, 10\}$, $\text{sum}(R) = 1 + 1 + 10 = 12$

Subset Sum

Subset Sum

Given a (multi)set S of n positive integers and a target integer t, does there exist a subset $R \subseteq S$ that sums to exactly t?

Example: $S = \{1, 1, 3, 10\}$, $t = 12$
YES! $R = \{1, 1, 10\}$, $\text{sum}(R) = 1 + 1 + 10 = 12$

A classic NP-hard problem. $O(2^{n/2})$ algorithm [Horowitz and Sahni, JACM 1974]

Pseudopolynomial time algorithm

Pseudopolynomial time algorithm: in poly(n, t) time.

Pseudopolynomial time algorithm

Pseudopolynomial time algorithm: in poly(n, t) time.
Dynamic programming algorithm [Bellman, 1957]:

$$(S = \{s_1, \ldots, s_n\})$$

$$f[i, x] := f[i-1, x] \text{ OR } f[i-1, x-s_i]$$
$$f[0, x] := [x == 0]$$

Output $f[n, t]$

Pseudopolynomial time algorithm

Pseudopolynomial time algorithm: in poly(n, t) time.
Dynamic programming algorithm [Bellman, 1957]:

$$(S = \{s_1, \ldots, s_n\})$$

$$f[i, x] := f[i-1, x] \text{ OR } f[i-1, x-s_i]$$
$$f[0, x] := [x == 0]$$

Output $f[n, t]$
$O(nt)$ time

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]
- $\tilde{O}(n + \sqrt{n}t)$ deterministic algorithm. [Koiliaris and Xu, SODA 2017]
  ($\tilde{O}(T)$ stands for $O(T \operatorname{poly}\log T)$)

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]
- $\tilde{O}(n+\sqrt{nt})$ deterministic algorithm. [Koiliaris and Xu, SODA 2017]
  ($\tilde{O}(T)$ stands for $O(T \operatorname{poly}\log T)$)
- $\tilde{O}(n+t)$ randomized algorithm. [Bringmann, SODA 2017]

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]
- $\tilde{O}(n+\sqrt{nt})$ deterministic algorithm. [Koiliaris and Xu, SODA 2017]
  ($\tilde{O}(T)$ stands for $O(T \operatorname{poly}\log T)$)
- $\tilde{O}(n+t)$ randomized algorithm. [Bringmann, SODA 2017]
  - more precisely, $O(n + t \log t \log^4 n)$ time (failure probability $\frac{1}{n}$)

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]
- $\tilde{O}(n+\sqrt{n}t)$ deterministic algorithm. [Koiliaris and Xu, SODA 2017]
  ($\tilde{O}(T)$ stands for $O(T \operatorname{poly}\log T)$)
- $\tilde{O}(n+t)$ randomized algorithm. [Bringmann, SODA 2017]
  - more precisely, $O(n + t \log t \log^4 n)$ time (failure probability $\frac{1}{n}$)
  - uses color coding, layer splitting, FFT

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]
- $\tilde{O}(n+\sqrt{n}t)$ deterministic algorithm. [Koiliaris and Xu, SODA 2017]
  ($\tilde{O}(T)$ stands for $O(T \operatorname{poly}\log T)$)
- $\tilde{O}(n+t)$ randomized algorithm. [Bringmann, SODA 2017]
  - more precisely, $O(n + t \log t \log^4 n)$ time (failure probability $\frac{1}{n}$)
  - uses color coding, layer splitting, FFT

Pseudopolynomial time algorithm

- $O(nt)$ [Bellman, 1957]
- $\tilde{O}(n+\sqrt{n}t)$ deterministic algorithm. [Koiliaris and Xu, SODA 2017]
  ($\tilde{O}(T)$ stands for $O(T \operatorname{poly}\log T)$)
- $\tilde{O}(n+t)$ randomized algorithm. [Bringmann, SODA 2017]
  - more precisely, $O(n + t\log t\log^4 n)$ time (failure probability $\frac{1}{n}$)
  - uses color coding, layer splitting, FFT

Near-optimal! (No $t^{1-\varepsilon}2^{o(n)}$ time algorithm for any $\varepsilon > 0$, unless SETH fails) [Abboud, Bringmann, Hermelin, Shabtay, SODA 2019]

Our result

We present a randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time.

Our result

We present a randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time.

Our techniques are simple and quite different from Bringmann's algorithm.

- generating functions
- FFT

Our result

We present a randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time.

Our techniques are simple and quite different from Bringmann's algorithm.

- generating functions
- FFT

Precise running time (with failure probability $\frac{1}{n+t}$) is $O(n + t\log^2 t)$.

Our result

We present a randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time.

Our techniques are simple and quite different from Bringmann's algorithm.

- generating functions
- FFT

Precise running time (with failure probability $\frac{1}{n+t}$) is $O(n + t\log^2 t)$.
Can be improved to $O(n + t\log t)$ time (slightly less simple).

- faster than Bringmann's algorithm by a $\log^4 n$ factor

Our result

We present a randomized algorithm for Subset Sum in $\tilde{O}(n+t)$ time.

Our techniques are simple and quite different from Bringmann's algorithm.

- generating functions
- FFT

Precise running time (with failure probability $\frac{1}{n+t}$) is $O(n + t\log^2 t)$.
Can be improved to $O(n + t\log t)$ time (slightly less simple).

- faster than Bringmann's algorithm by a $\log^4 n$ factor

Our approach: generating functions

$(S = \{s_1, s_2, \ldots, s_n\}$, target integer $= t)$
Consider the generating function of this instance

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i})$$

Our approach: generating functions

$(S = \{s_1, s_2, \ldots, s_n\}$, target integer $= t)$
Consider the generating function of this instance

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i})$$

Expand it

$$A(x) = 1 + \sum_{i \ge 1} a_i x^i$$

Our approach: generating functions

$(S = \{s_1, s_2, \ldots, s_n\}$, target integer $= t)$
Consider the generating function of this instance

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i})$$

Expand it

$$A(x) = 1 + \sum_{i \ge 1} a_i x^i$$

a_i = number of subsets summing to i
Output YES if $a_t \neq 0$

Our approach: generating functions

$(S = \{s_1, s_2, \ldots, s_n\}$, target integer $= t)$
Consider the generating function of this instance

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i})$$

Expand it

$$A(x) = 1 + \sum_{i \ge 1} a_i x^i$$

$a_i =$ number of subsets summing to i
Output YES if $a_t \neq 0$

Example: $S = \{1, 2, 3\}, t = 3$
$A(x) = (1 + x^1)(1 + x^2)(1 + x^3) = 1 + x + x^2 + 2x^3 + x^4 + x^5 + x^6$
$2x^3$: $\{1, 2\}, \{3\}$

Our approach: generating functions

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

Goal: efficiently compute $A(x) \bmod x^{t+1}$

Our approach: generating functions

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

Goal: efficiently compute $A(x)$ mod x^{t+1}

Straightforwardly expanding needs $O(nt)$ arithmetic operations

Need a smarter way to do this

Our approach: generating functions

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

Goal: efficiently compute $A(x)$ mod x^{t+1}
Straightforwardly expanding needs $O(nt)$ arithmetic operations
Need a smarter way to do this

The coefficients are too large ($\sim 2^n$)

Our approach: generating functions

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

Goal: efficiently compute $A(x)$ mod x^{t+1}
Straightforwardly expanding needs $O(nt)$ arithmetic operations
Need a smarter way to do this

The coefficients are too large ($\sim 2^n$)
Work in finite field F_p for some $\Theta(\log t)$-bit prime p. Assume an arithmetic operation takes constant time.

Our approach: generating functions

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

Goal: efficiently compute $A(x)$ mod x^{t+1}
Straightforwardly expanding needs $O(nt)$ arithmetic operations
Need a smarter way to do this

The coefficients are too large ($\sim 2^n$)
Work in finite field F_p for some $\Theta(\log t)$-bit prime p. Assume an arithmetic operation takes constant time.

Computing $(\ln A(x)) \bmod x^{t+1}$

Computing $(\ln A(x)) \bmod x^{t+1}$

Natural logarithm as a series

$$\ln(1+x) := x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{k-1}x^k}{k} + \cdots$$

Computing $(\ln A(x)) \bmod x^{t+1}$

Natural logarithm as a series

$$\ln(1+x) := x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{k-1}x^k}{k} + \cdots$$

$$\ln A(x) = \ln \prod_{1 \le i \le n} (1 + x^{s_i})$$

Computing $(\ln A(x)) \bmod x^{t+1}$

Natural logarithm as a series

$$\ln(1+x) := x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{k-1}x^k}{k} + \cdots$$

$$\begin{aligned}\ln A(x) &= \ln \prod_{1\le i\le n} (1 + x^{s_i}) \\ &= \sum_{1\le i\le n} \ln(1 + x^{s_i})\end{aligned}$$

Computing $(\ln A(x)) \bmod x^{t+1}$

Natural logarithm as a series

$$\ln(1+x) := x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{k-1}x^k}{k} + \cdots$$

$$\begin{aligned}
\ln A(x) &= \ln \prod_{1 \le i \le n} (1 + x^{s_i}) \\
&= \sum_{1 \le i \le n} \ln(1 + x^{s_i}) \\
&= \sum_{1 \le j \le t} m_j \ln(1 + x^j) \qquad (m_j \text{ denotes the number of } i\text{'s with } s_i = j)
\end{aligned}$$

Computing $(\ln A(x)) \bmod x^{t+1}$

Natural logarithm as a series

$$\ln(1+x) := x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{k-1}x^k}{k} + \cdots$$

$$\begin{aligned}
\ln A(x) &= \ln \prod_{1\le i\le n} (1 + x^{s_i}) \\
&= \sum_{1\le i\le n} \ln(1 + x^{s_i}) \\
&= \sum_{1\le j\le t} m_j \ln(1 + x^j) \qquad (m_j \text{ denotes the number of } i\text{'s with } s_i = j) \\
&= \sum_{1\le j\le t} m_j \sum_{k\ge 1} \frac{(-1)^{k-1}x^{jk}}{k}
\end{aligned}$$

Computing $(\ln A(x)) \bmod x^{t+1}$

Natural logarithm as a series

$$\ln(1+x) := x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots + \frac{(-1)^{k-1}x^k}{k} + \cdots$$

$$\begin{aligned}
\ln A(x) &= \ln \prod_{1\le i\le n} (1 + x^{s_i}) \\
&= \sum_{1\le i\le n} \ln(1 + x^{s_i}) \\
&= \sum_{1\le j\le t} m_j \ln(1 + x^j) \qquad (m_j \text{ denotes the number of } i\text{'s with } s_i = j) \\
&= \sum_{1\le j\le t} m_j \sum_{k\ge 1} \frac{(-1)^{k-1}x^{jk}}{k}
\end{aligned}$$

$(\ln A(x)) \bmod x^{t+1}$ only has $\sum_{1\le j\le t}\lfloor t/j\rfloor = O(t \log t)$ nonzero terms!

Computing $A(x) \pmod{x^{t+1}}$

We can compute $B(x) := (\ln A(x)) \bmod x^{t+1}$ in $O(t \log t)$ time.

Next: Recover $A(x) \bmod x^{t+1}$ by computing $\exp(B(x)) \bmod x^{t+1}$.

Computing $\exp(B(x)) \bmod x^{t+1}$

Given coefficients of $B(x)$, we can compute $\exp(B(x)) \bmod x^{t+1}$ in $O(M(t))$ time, where $M(t)$ denotes the running time of multiplying two polynomials of degree t. [Brent, 1976]

- FFT: $M(t) = O(t \log t)$

Computing $\exp(B(x)) \bmod x^{t+1}$

Given coefficients of $B(x)$, we can compute $\exp(B(x)) \bmod x^{t+1}$ in $O(M(t))$ time, where $M(t)$ denotes the running time of multiplying two polynomials of degree t. [Brent, 1976]

- FFT: $M(t) = O(t \log t)$
- uses Newton's iteration method

Computing $\exp(B(x)) \bmod x^{t+1}$

Given coefficients of $B(x)$, we can compute $\exp(B(x)) \bmod x^{t+1}$ in $O(M(t))$ time, where $M(t)$ denotes the running time of multiplying two polynomials of degree t. [Brent, 1976]

- FFT: $M(t) = O(t \log t)$
- uses Newton's iteration method

Computing $\exp(B(x)) \bmod x^{t+1}$

Given coefficients of $B(x)$, we can compute $\exp(B(x)) \bmod x^{t+1}$ in $O(M(t))$ time, where $M(t)$ denotes the running time of multiplying two polynomials of degree t. [Brent, 1976]

- FFT: $M(t) = O(t \log t)$
- uses Newton's iteration method

Here we describe a slightly simpler (and slower) folklore algorithm in $O(M(t) \log t) = O(t \log^2 t)$ time.

Computing $\exp(B(x)) \bmod x^{t+1}$

Let $C(x) := \exp(B(x)) = \sum_{i \geq 0} c_i x^i$. $\qquad (c_0 = 1)$

Computing $\exp(B(x)) \bmod x^{t+1}$

Let $C(x) := \exp(B(x)) = \sum_{i \geq 0} c_i x^i$. $\qquad (c_0 = 1)$

$$C'(x) = \big(\exp(B(x))\big)' = \exp(B(x))B'(x) = C(x)B'(x)$$

Computing $\exp(B(x)) \bmod x^{t+1}$

Let $C(x) := \exp(B(x)) = \sum_{i \geq 0} c_i x^i$. $\qquad (c_0 = 1)$

$$C'(x) = \big(\exp(B(x))\big)' = \exp(B(x))B'(x) = C(x)B'(x)$$

Let $B'(x) = \sum_{i \geq 0} b_i x^i$. Comparing x^{i-1} term gives a recurrence relation,

$$c_i = \frac{1}{i} \sum_{j=0}^{i-1} c_j b_{i-1-j}$$

Computing $\exp(B(x)) \bmod x^{t+1}$

Let $C(x) := \exp(B(x)) = \sum_{i \geq 0} c_i x^i$. $\qquad (c_0 = 1)$

$$C'(x) = \big(\exp(B(x))\big)' = \exp(B(x))B'(x) = C(x)B'(x)$$

Let $B'(x) = \sum_{i \geq 0} b_i x^i$. Comparing x^{i-1} term gives a recurrence relation,

$$c_i = \frac{1}{i} \sum_{j=0}^{i-1} c_j b_{i-1-j}$$

$b_0, b_1, \ldots, b_{t-1}$ are known. Want to compute $c_1, c_2, \ldots, c_t$.

Divide and Conquer

$$c_i = \frac{1}{i} \sum_{j=0}^{i-1} c_j b_{i-1-j}$$

Initialize $c_0 = 1$, $c_i \leftarrow 0 (1 \le i \le t)$.
Call $\textsc{Compute}(0, t)$.

procedure $\textsc{Compute}(L, R)$
  if $L < R$ **then**
    $M \leftarrow \lfloor \frac{L+R}{2} \rfloor$
    $\textsc{Compute}(L, M)$ ▷ Now $c_j (0 \le j \le M)$ are finalized
    for $i \leftarrow M+1, M+2, \ldots, R$ **do**

$$c_i \leftarrow c_i + \frac{1}{i} \sum_{j=L}^{M} c_j b_{i-1-j}$$

    $\textsc{Compute}(M+1, R)$

Divide and Conquer

$$c_i = \frac{1}{i} \sum_{j=0}^{i-1} c_j b_{i-1-j}$$

Initialize $c_0 = 1$, $c_i \leftarrow 0 (1 \le i \le t)$.
Call $\textsc{Compute}(0, t)$.

procedure $\textsc{Compute}(L, R)$
  if $L < R$ **then**
    $M \leftarrow \lfloor \frac{L+R}{2} \rfloor$
    $\textsc{Compute}(L, M)$ ▷ Now $c_j (0 \le j \le M)$ are finalized
    for $i \leftarrow M+1, M+2, \ldots, R$ **do**
      $c_i \leftarrow c_i + \frac{1}{i} \sum_{j=L}^{M} c_j b_{i-1-j}$ ▷ $0 \le i-1-j \le R-1-L$
    $\textsc{Compute}(M+1, R)$

Divide and Conquer

$$c_i = \frac{1}{i} \sum_{j=0}^{i-1} c_j b_{i-1-j}$$

Initialize $c_0 = 1$, $c_i \leftarrow 0 (1 \le i \le t)$.
Call $\textsc{Compute}(0, t)$.

procedure $\textsc{Compute}(L, R)$
  if $L < R$ **then**
    $M \leftarrow \lfloor \frac{L+R}{2} \rfloor$
    $\textsc{Compute}(L, M)$ ▷ Now $c_j (0 \le j \le M)$ are finalized
    for $i \leftarrow M+1, M+2, \ldots, R$ **do**
      $c_i \leftarrow c_i + \frac{1}{i} \sum_{j=L}^{M} c_j b_{i-1-j}$ ▷ $0 \le i-1-j \le R-1-L$
    $\textsc{Compute}(M+1, R)$

Use FFT: $O((R-L)\log(R-L))$ time!
Total time: $T(t) = 2T(t/2) + O(t \log t) = O(t \log^2 t)$

Working in finite field F_p

To avoid large coefficients, we choose a prime p and do all arithmetic operations mod p.

Working in finite field F_p

To avoid large coefficients, we choose a prime p and do all arithmetic operations mod p.
A fraction x/y becomes $(x \cdot y^{-1})$ mod p, where y^{-1} denotes the multiplicative inverse of y mod p (can be found using (Extended) Euclid's algorithm)

Working in finite field F_p

To avoid large coefficients, we choose a prime p and do all arithmetic operations mod p.
A fraction x/y becomes $(x \cdot y^{-1}) \bmod p$, where y^{-1} denotes the multiplicative inverse of y mod p (can be found using (Extended) Euclid's algorithm)
Requirement: $y \not\equiv 0 \pmod{p}$

Working in finite field F_p

To avoid large coefficients, we choose a prime p and do all arithmetic operations mod p.
A fraction x/y becomes $(x \cdot y^{-1}) \bmod p$, where y^{-1} denotes the multiplicative inverse of y mod p (can be found using (Extended) Euclid's algorithm)
Requirement: $y \not\equiv 0 \pmod{p}$
Our fractions only have denominators from $\{1, 2, \ldots, t\}$. It's sufficient to choose $p > t$.

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

For prime $p > t$, we can compute $(a_i \bmod p)$ for all $1 \le i \le t$ in $\tilde{O}(n + t)$ time.

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

For prime $p > t$, we can compute $(a_i \bmod p)$ for all $1 \le i \le t$ in $\tilde{O}(n + t)$ time.

Output YES when $(a_t \bmod p) \neq 0$.

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

For prime $p > t$, we can compute $(a_i \bmod p)$ for all $1 \le i \le t$ in $\tilde{O}(n + t)$ time.

Output YES when $(a_t \bmod p) \neq 0$.

Will make a mistake if p divides a_t.

$$A(x) = \prod_{1 \le i \le n} (1 + x^{s_i}) = 1 + a_1 x + a_2 x^2 + \cdots + a_t x^t + \ldots$$

For prime $p > t$, we can compute $(a_i \bmod p)$ for all $1 \le i \le t$ in $\tilde{O}(n + t)$ time.

Output YES when $(a_t \bmod p) \neq 0$.

Will make a mistake if p divides a_t.

$a_t \le 2^n$, so a_t has at most n prime factors.

Randomly picking a prime from interval $[t + 1, (n + t)^3]$ makes failure probability less than $\frac{1}{n+t}$.

Closing remarks

Open problem: Can we achieve deterministic near-linear time for Subset Sum?
(current best: $\tilde{O}(n + \sqrt{nt})$ by Koiliaris and Xu [SODA 2017])

Closing remarks

Open problem: Can we achieve deterministic near-linear time for Subset Sum?
(current best: $\tilde{O}(n + \sqrt{nt})$ by Koiliaris and Xu [SODA 2017])

Does our technique apply to other problems?

Thank you!