Saving Space by Algebraization

Daniel Lokshtanov and Jesper Nederlof

6 June 2010

Dynamic Programming (DP)

- From the 1950's by Richard Bellman in his book "Bottleneck Problems and Dynamic Programming"; Nowadays a prominent algorithmic technique in designing algorithms.

Dynamic Programming (DP)

- From the 1950's by Richard Bellman in his book "Bottleneck Problems and Dynamic Programming"; Nowadays a prominent algorithmic technique in designing algorithms.
- Uses a big table of data that has to be stored in the memory.

Dynamic Programming (DP)

- From the 1950's by Richard Bellman in his book "Bottleneck Problems and Dynamic Programming"; Nowadays a prominent algorithmic technique in designing algorithms.
- Uses a big table of data that has to be stored in the memory.
- A relatively easy procedure computes new table entries using already computed table entries.

Dynamic Programming (DP)

- From the 1950's by Richard Bellman in his book "Bottleneck Problems and Dynamic Programming"; Nowadays a prominent algorithmic technique in designing algorithms.
- Uses a big table of data that has to be stored in the memory.
- A relatively easy procedure computes new table entries using already computed table entries.
- This easy procedure is often so easy that we just write it down as a single formula, obtaining a recurrence.

Dynamic Programming (DP)

- From the 1950's by Richard Bellman in his book "Bottleneck Problems and Dynamic Programming"; Nowadays a prominent algorithmic technique in designing algorithms.
- Uses a big table of data that has to be stored in the memory.
- A relatively easy procedure computes new table entries using already computed table entries.
- This easy procedure is often so easy that we just write it down as a single formula, obtaining a recurrence.
- We are interested in conditions that are sufficient for being able to reduce the space requirement of DP algorithms significantly (without significant loss of speed).

The approach in a nutshell

- Usually the dependency between table entries is highly unpredictable and interleaved.

The approach in a nutshell

- Usually the dependency between table entries is highly unpredictable and interleaved.
- The best one can do is keeping track of (almost) the whole table.

The approach in a nutshell

- Usually the dependency between table entries is highly unpredictable and interleaved.
- The best one can do is keeping track of (almost) the whole table.
- We use a transformation to transform the table into one where the dependency between table entries is very restricted and systematic.

UNIVERSITETET I BERGEN
Institutt for informatikk

The approach in a nutshell

- Usually the dependency between table entries is highly unpredictable and interleaved.
- The best one can do is keeping track of (almost) the whole table.
- We use a transformation to transform the table into one where the dependency between table entries is very restricted and systematic.
- This allows us to turn DP algorithms of which the recurrence only uses certain operators in space efficient ones.

Subset Sum

- Given integers $\{e_1, \ldots, e_n\}$ and t in binary representation, count the number of subsets $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} e_i = t$.

Subset Sum

- Given integers $\{e_1, \dots, e_n\}$ and t in binary representation, count the number of subsets $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} e_i = t$.
- Brute force gives a $\mathcal{O}(2^n)$ time, polynomial space algorithm. DP gives an $\mathcal{O}(nt)$ time and $\mathcal{O}(t)$ space algorithm:

Subset Sum

- Given integers $\{e_1, \ldots, e_n\}$ and t in binary representation, count the number of subsets $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} e_i = t$.
- Brute force gives a $\mathcal{O}(2^n)$ time, polynomial space algorithm. DP gives an $\mathcal{O}(nt)$ time and $\mathcal{O}(t)$ space algorithm:
- Define $A[i, j]$ as the number of subsets $S \subseteq \{1, \ldots, i\}$ such that $\sum_{k \in S} e_k = j$. Then

Subset Sum

- Given integers $\{e_1, \dots, e_n\}$ and t in binary representation, count the number of subsets $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} e_i = t$.
- Brute force gives a $\mathcal{O}(2^n)$ time, polynomial space algorithm. DP gives an $\mathcal{O}(nt)$ time and $\mathcal{O}(t)$ space algorithm:
- Define $A[i,j]$ as the number of subsets $S \subseteq \{1, \dots, i\}$ such that $\sum_{k \in S} e_k = j$. Then

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

Subset Sum

- Given integers $\{e_1, \dots, e_n\}$ and t in binary representation, count the number of subsets $S \subseteq \{1, \dots, n\}$ such that $\sum_{i \in S} e_i = t$.
- Brute force gives a $\mathcal{O}(2^n)$ time, polynomial space algorithm. DP gives an $\mathcal{O}(nt)$ time and $\mathcal{O}(t)$ space algorithm:
- Define $A[i,j]$ as the number of subsets $S \subseteq \{1, \dots, i\}$ such that $\sum_{k \in S} e_k = j$. Then

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

- The answer of the Subset Sum instance can be read from $A[n,t]$.

Weight →

items ↓

item	e_i	0	1	2	3	4	5	6	7	8
1	1	1	1	0	0	0	0	0	0	0
2	2	1	1	1	1	0	0	0	0	0
3	5	1	1	1	1	0	1	1	1	1

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

Weight →

items ↓		0	1	2	3	4	5	6	7	8
1	1	1	1	0	0	0	0	0	0	0
2	2	1	1	1	1	0	0	0	0	0
3	5	1	1	1	1	0	1	1	1	1

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

"Saving space by algebraization", STOC 2010

items		0	1	2	3	4	5	6	7	8
1	1	1	1	0	0	0	0	0	0	0
2	2	1	1	1	1	0	0	0	0	0
3	5	1	1	1	1	0	1	1	1	1

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

Weight →

items ↓ (1 … n)

	0	1	2	3	4	5	6	7	8	…	$t-6339$	…	$t-3175$	…	t	…	nt
1	1	1	0	0	0	0	0	0	0	…	0	…	0	…	0	…	0
2	1	1	1	1	0	0	0	0	0	…	0	…	0	…	0	…	0
5	1	1	1	1	0	1	1	1	1	…	0	…	0	…	0	…	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮		⋮		⋮
3156	1	1	1	1	0	1	1	1	1	…	1	…	1	…	0	…	0
3164	1	1	1	1	0	1	1	1	1	…	65	…	2	…	0	…	0
3175	1	1	1	1	0	1	1	1	1	…	82	…	67	…	2	…	0

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

"Saving space by algebraization", STOC 2010

Weight →

items ↓ (1 … n)

	0	1	2	3	4	5	6	7	8	$t-6339$	$t-3175$	t	nt
1	1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	0	0	0	0	0	0	0	0	0
5	1	1	1	1	0	1	1	1	1	0	0	0	0
3156	1	1	1	1	0	1	1	1	1	1	1	0	0
3164	1	1	1	1	0	1	1	1	1	65	2	0	0
3175	1	1	1	1	0	1	1	1	1	82	67	2	0

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

Weight →

items ↓ (1 … n)

		0	1	2	3	4	5	6	7	8	$t-6339$	$t-3175$	t	nt
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
	2	1	1	1	1	0	0	0	0	0	0	0	0	0
	5	1	1	1	1	0	1	1	1	1	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	3156	1	1	1	1	0	1	1	1	1	1	1	0	0
	3164	1	1	1	1	0	1	1	1	1	65	2	0	0
n	3175	1	1	1	1	0	1	1	1	1	82	67	2	0

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

Weight →

items (1 to n) ↓

	0	1	2	3	4	5	6	7	8	$t-6339$	$t-3175$	t	nt
1	1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	0	0	0	0	0	0	0	0	0
5	1	1	1	1	0	1	1	1	1	0	0	0	0
3156	1	1	1	1	0	1	1	1	1	1	1	0	0
3164	1	1	1	1	0	1	1	1	1	65	2	0	0
3175	1	1	1	1	0	1	1	1	1	82	67	2	0

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

"Saving space by algebraization", STOC 2010

Definition (Convolution operator)

- Define the operator $\otimes$ on column vectors of size N as

$$\mathbf{a} \otimes \mathbf{b} = \Big(\sum_{i+j=k} a_i b_j \Big)^T_{0 \le k < N}$$

Definition (Convolution operator)

- Define the operator $\otimes$ on column vectors of size N as

$$\mathbf{a} \otimes \mathbf{b} = \Big(\sum_{i+j=k} a_i b_j \Big)^T_{0 \le k < N}$$

Example

$$\begin{pmatrix} 1 \\ 3 \\ 4 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 2 \\ 3 \\ 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 9 \\ 20 \\ 21 \\ 12 \end{pmatrix}$$

Definition (Convolution operator)

- Define the operator $\otimes$ on column vectors of size N as

$$\mathbf{a} \otimes \mathbf{b} = \Big(\sum_{i+j=k} a_i b_j \Big)^T_{0 \leq k < N}$$

Example

$$\begin{pmatrix} 1 \\ 3 \\ 4 \\ 0 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 2 \\ 3 \\ 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 9 \\ 20 \\ 21 \\ 12 \end{pmatrix}$$

$$(1 + 3x + 4x^2)(2 + 3x + 3x^2) = 2 + 9x + 20x^2 + 21x^3 + 12x^4$$

Convolution

Definition (Pointwise multiplication)

- Let $\cdot$ be the point-wise multiplication of two vectors, that is:

$$\mathbf{a} \cdot \mathbf{b} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{N-1} \end{pmatrix} = \begin{pmatrix} a_0 b_0 \\ a_1 b_1 \\ \vdots \\ a_{N-1} b_{N-1} \end{pmatrix}$$

Convolution

- Assume we have an invertible matrix $\mathbf{T}$ such that for every $\mathbf{a}$, $\mathbf{b}$

$$\mathbf{T}(\mathbf{a} \otimes \mathbf{b}) = \mathbf{Ta} \cdot \mathbf{Tb},$$

- and we want to compute $\mathbf{d}_t$, where (for example)

$$\mathbf{d} = (\mathbf{a} \otimes \mathbf{b}) \otimes (\mathbf{c} + \mathbf{a}).$$

- Then we know that

$$\begin{aligned}\mathbf{Td} &= \mathbf{T}\big((\mathbf{a} \otimes \mathbf{b}) \otimes (\mathbf{c} + \mathbf{a})\big)\\ &= \mathbf{T}(\mathbf{a} \otimes \mathbf{b}) \cdot \mathbf{T}(\mathbf{c} + \mathbf{a})\\ &= (\mathbf{Ta} \cdot \mathbf{Tb}) \cdot (\mathbf{Tc} + \mathbf{Ta})\end{aligned}$$

- And $\mathbf{d}_t$ can be obtained using

$$\mathbf{d}_t = (\mathbf{T}^{-1}\mathbf{Td})_t = \sum_{i=0}^{N-1} \mathbf{T}^{-1}_{ti}\big((\mathbf{Ta})_d(\mathbf{Tb})_d\big)\big((\mathbf{Tc})_d + (\mathbf{Ta})_d\big)$$

Discrete Fourier Transform (DFT)

Definition

Let ω be a number s.t. $\omega^N = 1$ and $\omega^i \neq 1$ for $1 < i < N$, then the discrete Fourier transform F is defined as:

$$\mathbf{F} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{N-1} \\ \vdots & \vdots & \omega^{ij} & \vdots \\ 1 & \omega^{N-1} & \dots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

Lemma

$$\mathbf{F}^{-1} = \frac{1}{N}\left(\omega^{-ij}\right)_{0 \le i,j < N} \qquad \textit{and also} \qquad \mathbf{F}(\mathbf{a} \otimes \mathbf{b}) = \mathbf{Fa} \cdot \mathbf{Fb}$$

Subset Sum revisited

- Now we will use the DFT on the dynamic programming algorithm.
- In order to achieve this we first have to change perspective slightly, and consider the DP table as an array of row vectors.
- For an integer k, denote I_k as the k'th column of the identity matrix.

Example

$$\begin{pmatrix}1\\3\\4\\0\\0\end{pmatrix} \otimes \begin{pmatrix}0\\0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\3\\4\end{pmatrix}$$

Metamorphosis

So we had

$$A[i,j] = \begin{cases} 0 & \text{if } i = 1, e_1 \neq j, \text{ and } e_1 \neq 0 \\ 1 & \text{if } i = 1 \text{ and } (e_1 = j \text{ or } j = 0) \\ A[i-1,j] + A[i-1,j-e_i] & \text{if } i > 1 \end{cases}$$

And we rewrite it as

$$A[i] = \begin{cases} I_0^T + I_{e_1}^T & \text{if } i = 1 \\ A[i-1] + A[i-1] \otimes I_{e_i}^T & \text{if } i > 1 \end{cases}$$

- Recall we are interested in $A[n,t] = A[n]_t$.

Metamorphosis

$$A[i] = \begin{cases} I_0^T + I_{e_1}^T & \text{if } i = 1 \\ A[i-1] + A[i-1] \otimes I_{e_i}^T & \text{if } i > 1 \end{cases}$$

Metamorphosis

$$\mathbf{F}(A[i]) = \begin{cases} \mathbf{F}(I_0^T + \quad I_{e_1}^T) & \text{if } i = 1 \\ \mathbf{F}(A[i-1] \ + \quad A[i-1] \otimes \qquad I_{e_i}^T) & \text{if } i > 1 \end{cases}$$

Metamorphosis

$$\mathbf{F}(A[i]) = \begin{cases} \mathbf{F}\ \ I_0^T + \mathbf{F} I_{e_1}^T & \text{if } i = 1 \\ \mathbf{F}(A[i-1]) + \mathbf{F}(A[i-1] \otimes \qquad I_{e_i}^T) & \text{if } i > 1 \end{cases}$$

Metamorphosis

$$\mathbf{F}(A[i]) = \begin{cases} \mathbf{F}\ \ I_0^T + \mathbf{F} I_{e_1}^T & \text{if } i = 1 \\ \mathbf{F}(A[i-1]) + \mathbf{F}(A[i-1] \qquad) \cdot \mathbf{F}(I_{e_i}^T) & \text{if } i > 1 \end{cases}$$

Metamorphosis

$$\mathbf{F}(A[i]) = \begin{cases} \mathbf{F} \;\; I_0^T + \mathbf{F} I_{e_1}^T & \text{if } i = 1 \\ \mathbf{F}(A[i-1]) + \mathbf{F}(A[i-1] \qquad) \cdot \mathbf{F}(I_{e_i}^T) & \text{if } i > 1 \end{cases}$$

- And all dependency between different components of vectors is gone, since we only use addition and point wise multiplication.

The transformed table

index of vector →

which vector ↓

which vector		0	1	2	3	4	5	6	7	8	$t-6339$	$t-3175$	t	nt
1	1	5	23	68	79	14	143	87	401	413	154	294	513	94
	2	41	25	325	83	25	325	6	72	9	97	32	273	26
	5	43	12	13	91	150	13	267	65	89	256	426	18	103
	3156	6	65	44	12	109	44	325	9	25	169	113	93	284
	3164	72	43	98	72	83	98	83	43	78	365	52	265	185
n	3175	516	12	78	283	12	247	43	102	13	62	77	112	394

The transformed table

index of vector →

which vector ↓ (from 1 to n)

	0	1	2	3	4	5	6	7	8	$t-6339$	$t-3175$	t	nt
1	5	23	68	79	14	143	87	401	413	154	294	513	94
2	41	25	325	83	25	325	6	72	9	97	32	273	26
5	43	12	13	91	150	13	267	65	89	256	426	18	103
3156	6	65	44	12	109	44	325	9	25	169	113	93	284
3164	72	43	98	72	83	98	83	43	78	365	52	265	185
3175	516	12	78	283	12	247	43	102	13	62	77	112	394

- Any component of the bottom row can be computed using $\mathcal{O}(n)$ additions and multiplications!

The finishing touch

- So we can compute any component of the vector $\mathbf{F}(A[n])$ fast.
- Now we can compute $A[n]_t$ according to

$$A[n]_t = (\mathbf{F}^{-1}(\mathbf{F}(A[n])))_t =$$

$$\left(\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{-(N-1)} \\ \vdots & \vdots & \omega^{-ij} & \vdots \\ 1 & \omega^{-(N-1)} & \dots & \omega^{-(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} (\mathbf{F}(A[n]))_1 \\ \vdots \\ (\mathbf{F}(A[n]))_{N-1} \end{pmatrix} \right)_t$$

The finishing touch

- So we can compute any component of the vector $\mathbf{F}(A[n])$ fast.
- Now we can compute $A[n]_t$ according to

$$A[n]_t = (\mathbf{F}^{-1}(\mathbf{F}(A[n])))_t =$$

$$\left(\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega & \dots & \omega^{-(N-1)} \\ \vdots & \vdots & \omega^{-ij} & \vdots \\ 1 & \omega^{-(N-1)} & \dots & \omega^{-(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} (\mathbf{F}(A[n]))_1 \\ \vdots \\ (\mathbf{F}(A[n]))_{N-1} \end{pmatrix} \right)_t$$

- and we are done!

The transformed table

index of vector →

which vector ↓ (1 … n)

which vector	0	1	2	3	4	5	6	7	8	⋯	$t-6339$	⋯	$t-3175$	⋯	t	⋯	nt
1	5	23	68	79	14	143	87	401	413		154		294		513		94
2	41	25	325	83	25	325	6	72	9		97		32		273		26
5	43	12	13	91	150	13	267	65	89		256		426		18		103
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮		⋮		⋮
3156	6	65	44	12	109	44	325	9	25		169		113		93		284
3164	72	43	98	72	83	98	83	43	78		365		52		265		185
3175	516	12	78	283	12	247	43	102	13		62		77		112		394

The transformed table

index of vector →

which vector ↓ (1 … n)

which vector		0	1	2	3	4	5	6	7	8	…	$t-6339$	…	$t-3175$	…	t	…	nt
1	1	5	23	68	79	14	143	87	401	413		154		294		513		94
	2	41	25	325	83	25	325	6	72	9		97		32		273		26
	5	43	12	13	91	150	13	267	65	89		256		426		18		103
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮		⋮		⋮
	3156	6	65	44	12	109	44	325	9	25		169		113		93		284
	3164	72	43	98	72	83	98	83	43	78		365		52		265		185
n	3175	516	**12**	78	283	12	247	43	102	13		62		77		112		394

The transformed table

index of vector →

which vector ↓ (from 1 to n)

which vector	0	1	2	3	4	5	6	7	8	⋯	$t-6339$	⋯	$t-3175$	⋯	t	⋯	nt
1	5	23	68	79	14	143	87	401	413		154		294		513		94
2	41	25	325	83	25	325	6	72	9		97		32		273		26
5	43	12	13	91	150	13	267	65	89		256		426		18		103
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮		⋮		⋮
3156	6	65	44	12	109	44	325	9	25		169		113		93		284
3164	72	43	98	72	83	98	83	43	78		365		52		265		185
n 3175	516	12	**78**	283	12	247	43	102	13		62		77		112		394

The transformed table

index of vector →

which vector ↓ (from 1 to n)

	0	1	2	3	4	5	6	7	8	...	$t-6339$	...	$t-3175$	...	t	...	nt
1	5	23	68	79	14	143	87	401	413		154		294		513		94
2	41	25	325	83	25	325	6	72	9		97		32		273		26
5	43	12	13	91	150	13	267	65	89		256		426		18		103
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮		⋮		⋮
3156	6	65	44	12	109	44	325	9	25		169		113		93		284
3164	72	43	98	72	83	98	83	43	78		365		52		265		185
n 3175	516	12	78	283	12	247	43	102	13		62		77		112		**394**

The transformed table

index of vector →

which vector ↓ (1 … n)

which vector	0	1	2	3	4	5	6	7	8	$t-6339$	$t-3175$	t	nt
1	5	23	68	79	14	143	87	401	413	154	294	513	94
2	41	25	325	83	25	325	6	72	9	97	32	273	26
5	43	12	13	91	150	13	267	65	89	256	426	18	103
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
3156	6	65	44	12	109	44	325	9	25	169	113	93	284
3164	72	43	98	72	83	98	83	43	78	365	52	265	185
3175	516	12	78	283	12	247	43	102	13	62	77	112	394

The new algorithm uses $\tilde{O}(n^3 t \log t)$ time and $\tilde{O}(n^2)$ space.

Further remarks

- The algorithm has to work in a field where there exists an ω.
- This can be achieved by using for example complex numbers with finite precision.
- We also used the Möbius transformation to save space for a different type of DP algorithms
  - in combination with the DFT, this found applications to among others the Traveling Salesman and Weighted Steiner Tree problems.
- It would be interesting to find more transformations that also are useful to save space for existing DP algorithms.

Further research

- Can we efficiently save space for "Min-Sum DP algorithms"?
  - For the Knapsack problem the approach results in an algorithm much slower then the corresponding DP algorithm.
- Are there space and time efficient algorithms for deciding properties of partial k-trees (for example, maximum independent set)?
- Does there exists a positive ϵ such that
  - Subset Sum can be solved in $\mathcal{O}((2-\epsilon)^n)$ time and polynomial space?
  - Subset Sum can be solved in $\mathcal{O}(n^c t^{(1-\epsilon)})$ time?

Thanks for listening!