Algorithms and datastructures I
Lecture 13: dynamic programming

Jan Hubička

Department of Applied Mathematics
Charles University
Prague

May 12 2020

Dynamic programming


Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Dynamic programming

Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Fib(n)

1. If $n \leq 1$: return n.
2. Return Fib$(n-1)+$ Fib$(n-2)$.

Dynamic programming

Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Fib(n)

1. If $n \leq 1$: return n.
2. Return Fib($n-1$) $+$ Fib($n-2$).

$F_n \approx 1.618^n \implies$ exponential runtime

Dynamic programming

Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Fib(n)

1. If $n \leq 1$: return n.
2. Return Fib($n-1$) + Fib($n-2$).

$F_n \approx 1.618^n \implies$ exponential runtime

Fib(n) (with memoization)

1. If $T[n]$ is defined: return $T[n]$.
2. If $n \leq 1$: $T[n] \leftarrow n$.
3. else $T[n] \leftarrow$ Fib($n-1$) + Fib($n-2$).
4. Return $T[n]$.

Dynamic programming

Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Fib(n)

1. If $n \leq 1$: return n.
2. Return Fib$(n-1)$ + Fib$(n-2)$.

$F_n \approx 1.618^n \implies$ exponential runtime

Fib(n) (with memoization)

1. If $T[n]$ is defined: return $T[n]$.
2. If $n \leq 1$: $T[n] \leftarrow n$.
3. else $T[n] \leftarrow$ Fib$(n-1)$ + Fib$(n-2)$.
4. Return $T[n]$.

$O(n)$ runtime

Dynamic programming

Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Fib(n)

1. If $n \leq 1$: return n.
2. Return Fib$(n-1)$ + Fib$(n-2)$.

$F_n \approx 1.618^n \implies$ exponential runtime

Fib(n) (with memoization)

1. If $T[n]$ is defined: return $T[n]$.
2. If $n \leq 1$: $T[n] \leftarrow n$.
3. else $T[n] \leftarrow$ Fib$(n-1)$ + Fib$(n-2)$.
4. Return $T[n]$.

$O(n)$ runtime

Fib(n) (without recursion)

1. $T[0] \leftarrow 0$, $T[1] \leftarrow 1$
2. For $k = 2, \ldots n$: $T[k] \leftarrow T[k-1] + T[k-2]$
3. Return $T[n]$.
4. Return Fib$(n-1)$ + Fib$(n-2)$.

Dynamic programming

Richard Ernest Bellman

Dynamic programming is a method of solving optimization problems by breaking them recursively into smaller problems and using a table to avoid repeated recursive computations.

Fib(n)

1. If $n \leq 1$: return n.
2. Return Fib$(n-1)$ + Fib$(n-2)$.

$F_n \approx 1.618^n \implies$ exponential runtime

Fib(n) (with memoization)

1. If $T[n]$ is defined: return $T[n]$.
2. If $n \leq 1$: $T[n] \leftarrow n$.
3. else $T[n] \leftarrow$ Fib$(n-1)$ + Fib$(n-2)$.
4. Return $T[n]$.

$O(n)$ runtime

Grand plan:

1. Start with a recursive algorithm
2. Determine repeated invocations
3. Add a table (cache) memoizing the results
4. Determine an order of filling the cache avoiding the recursion

Fib(n) (without recursion)

1. $T[0] \leftarrow 0$, $T[1] \leftarrow 1$
2. For $k = 2, \ldots n$: $T[k] \leftarrow T[k-1] + T[k-2]$
3. Return $T[n]$.
4. Return Fib$(n-1)$ + Fib$(n-2)$.

Recall: Floyd-Washall algorithm

Let G be a graph with vertices $V = \{1, 2, \ldots n\}$.
Instead of distances from a given vertex v_0 we want to compute distance matrix D such that $D_{i,j} = d(i, j)$.

Definition

Let D^k be a matrix such that $D^k_{i,j}$ is the length of shortest path from i to j such that all internal vertices are in $\{1, 2, \ldots, k\}$.

Floyd-Washall Algorithm

Input: Matrix of length of edges D^0

1. For $k = 0, \ldots, n - 1$
2. For $i = 1, \ldots, n$
3. For $j = 1, \ldots, n$
4. $D^{k+1}_{i,j} = \min(D^k_{i,j}, D^k_{i,k+1} + D^k_{k+1,j})$

Output: Matrix of distances D^n

Time complexity $\Theta(n^3)$.
Memory complexity can be reduced by $\Theta(n^2)$ by modifying matrix “in place”
(it holds that $D^{k+1}_{k+1,j} = D^k_{k+1,j}$ and $D^{k+1}_{i,k+1} = D^k_{i,k+1}$).

Walks in Manhattan

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \dots, x_m), (y_1, \dots, y_n), i, j)$

1. If $i > n$: return $m - j + 1$.
2. If $j > m$: return $n - i + 1$.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j)$

1. If $i > n$: return $m - j + 1$.
2. If $j > m$: return $n - i + 1$.
3. $\ell_r \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j + 1)$.
4. If $x_i \neq y_j$: $\ell_r \leftarrow \ell_r + 1$.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j)$

1. If $i > n$: return $m - j + 1$.
2. If $j > m$: return $n - i + 1$.
3. $\ell_r \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j + 1)$.
4. If $x_i \neq y_j$: $\ell_r \leftarrow \ell_r + 1$.
5. $\ell_d \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j) + 1$.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \dots, x_m), (y_1, \dots, y_n), i, j)$

1. If $i > n$: return $m - j + 1$.
2. If $j > m$: return $n - i + 1$.
3. $\ell_r \leftarrow$ Edit $((x_1, \dots, x_m), (y_1, \dots, y_n), i + 1, j + 1)$.
4. If $x_i \neq y_j$: $\ell_r \leftarrow \ell_r + 1$.
5. $\ell_d \leftarrow$ Edit $((x_1, \dots, x_m), (y_1, \dots, y_n), i + 1, j) + 1$.
6. $\ell_i \leftarrow$ Edit $((x_1, \dots, x_m), (y_1, \dots, y_n), i, j + 1) + 1$.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j)$

1. If $i > n$: return $m - j + 1$.
2. If $j > m$: return $n - i + 1$.
3. $\ell_r \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j + 1)$.
4. If $x_i \neq y_j$: $\ell_r \leftarrow \ell_r + 1$.
5. $\ell_d \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j) + 1$.
6. $\ell_i \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j + 1) + 1$.
7. Return $\min(\ell_r, \ell_d, \ell_i)$.

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j)$

1. If $i > n$: return $m - j + 1$.
2. If $j > m$: return $n - i + 1$.
3. $\ell_r \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j + 1)$.
4. If $x_i \neq y_j$: $\ell_r \leftarrow \ell_r + 1$.
5. $\ell_d \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i + 1, j) + 1$.
6. $\ell_i \leftarrow$ Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j + 1) + 1$.
7. Return $\min(\ell_r, \ell_d, \ell_i)$.

Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j)$

1. For $i = 1, \ldots, n + 1$: $T[i, m + 1] \leftarrow n - i + 1$.
2. For $j = 1, \ldots, m + 1$: $T[n + 1, j] \leftarrow m - j + 1$.
3. For $i = n, \ldots, 1$:
4.   For $j = m, \ldots, 1$:
5.   If $x_i = x_j$: $\delta \leftarrow 0$ else $\delta \leftarrow 1$
6. $T[i, j] \leftarrow \min(\delta + T[i + 1, j + 1], 1 + T[i + 1, j],$
7. $1 + T[i, j + 1])$
8. Return $T[1, 1]$.

Runtime: $O(nm)$

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit distance (Vladimir Iosifovich Levenshtein, 1965)

Edit operation is insertion, deletion or replacement of a single character.
Edit distance of string X and Y is the minimal number of operations needed to turn X to Y.

Edit $((x_1, \ldots, x_m), (y_1, \ldots, y_n), i, j)$

1. For $i = 1, \ldots, n+1$: $T[i, m+1] \leftarrow n - i + 1$.
2. For $j = 1, \ldots, m+1$: $T[n+1, j] \leftarrow m - j + 1$.
3. For $i = n, \ldots, 1$:
4. For $j = m, \ldots, 1$:
5. If $x_i = x_j$: $\delta \leftarrow 0$ else $\delta \leftarrow 1$
6. $T[i, j] \leftarrow \min(\delta + T[i+1, j+1], 1 + T[i+1, j],$
7. $1 + T[i, j+1])$
8. Return $T[1, 1]$.

Runtime: $O(nm)$

Optimal search trees

Donald E. Knuth

Definition

Given set of elements $x_1, x_2, \ldots, x_n$ and weights $w_1, w_2, \ldots, w_n$ the optimal search tree is a binary search tree minimizing

$$\sum_{i=1}^{n} w_i F(x_i)$$

where $F(x_i)$ is the number of vertices visited by Find(x_i).

Optimal search trees

Donald E. Knuth

Definition

Given set of elements $x_1, x_2, \ldots, x_n$ and weights $w_1, w_2, \ldots, w_n$ the optimal search tree is a binary search tree minimizing

$$\sum_{i=1}^{n} w_i F(x_i)$$

where $F(x_i)$ is the number of vertices visited by Find(x_i).

OptTree $((x_1, \ldots, x_n), (w_1, \ldots, w_n), i, j)$

1. If $i > j$: return 0.
2. $W \leftarrow w_i + \cdots + w_j$
3. $C \leftarrow +\infty$
4. For $k = 1, \ldots, j$:
5. $\quad c_\ell \leftarrow$ OptTree $((x_1, \ldots, x_n), (w_1, \ldots, w_n), i, k-1)$
6. $\quad c_r \leftarrow$ OptTree $((x_1, \ldots, x_n), (w_1, \ldots, w_n), k+1, j)$
7. $\quad C = \min(C, c_\ell + c_r + W)$
8. Return C.

Optimal search trees

Donald E. Knuth

Definition

Given set of elements $x_1, x_2, \dots, x_n$ and weights $w_1, w_2, \dots, w_n$ the optimal search tree is a binary search tree minimizing

$$\sum_{i=1}^{n} w_i F(x_i)$$

where $F(x_i)$ is the number of vertices visited by Find(x_i).

OptTree $((x_1, \dots, x_n), (w_1, \dots, w_n), i, j)$

1. If $i > j$: return 0.
2. $W \leftarrow w_i + \cdots + w_j$
3. $C \leftarrow +\infty$
4. For $k = 1, \dots, j$:
5. $\quad c_\ell \leftarrow$ OptTree $((x_1, \dots, x_n), (w_1, \dots, w_n), i, k-1)$
6. $\quad c_r \leftarrow$ OptTree $((x_1, \dots, x_n), (w_1, \dots, w_n), k+1, j)$
7. $\quad C = \min(C, c_\ell + c_r + W)$
8. Return C.

OptTree $((x_1, \dots, x_n), (w_1, \dots, w_n), i, j)$ (Knuth 1971)

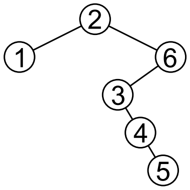
1. For $i = 1, \dots, n+1$: $T[i, i-1] \leftarrow 0$
2. For $\ell = 1, \dots, n$, $i = 1, \dots, n - \ell + 1$
3. $\quad j \leftarrow i + \ell - 1$
4. $\quad W \leftarrow w_i + \cdots + w_j$
5. $\quad T[i, j] \leftarrow +\infty$
6. $\quad$ For $k = 1, \dots, j$:
7. $\qquad C \leftarrow T[i, k-1] + T[k+1, j] + W$
8. $\qquad$ If $C < T[i.j]$: $T[i, j] \leftarrow C$, $K[i, j] \leftarrow k$

Optimal search trees

$w_1 = 1$
$w_2 = 10$
$w_3 = 3$
$w_4 = 2$
$w_5 = 1$
$w_6 = 9$

T	0	1	2	3	4	5	6
1	0	1	12	18	24	28	52
2	–	0	10	16	22	26	50
3	–	–	0	3	7	10	25
4	–	–	–	0	2	4	16
5	–	–	–	–	0	1	11
6	–	–	–	–	–	0	9
7	–	–	–	–	–	–	0

K	1	2	3	4	5	6
1	1	2	2	2	2	2
2	–	2	2	2	2	2
3	–	–	3	3	3	6
4	–	–	–	4	4	6
5	–	–	–	–	5	6
6	–	–	–	–	–	6




"That's all Folks!"