Dynamic programming

Václav Hlaváč
Czech Technical University in Prague
Czech Institute of Informatics, Cybernetics and Robotics (CIIRC)
160 00 Prague 6, Jugoslávských partyzánů 1580/3, Czech Republic
vaclav.hlavac@cvut.cz, http://people.ciirc.cvut.cz/hlavac/

What is DP ?

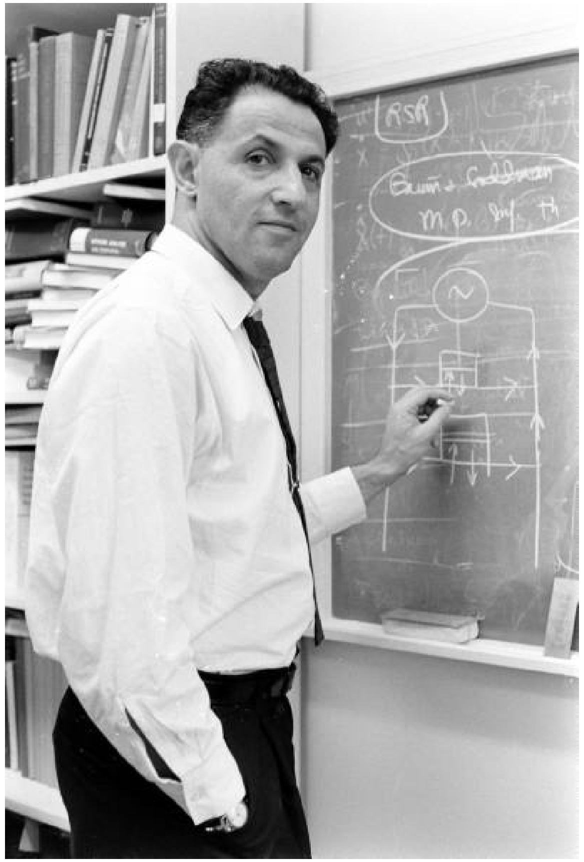
- Dynamic programming is a general algorithm design technique for solving optimization tasks defined by or formulated as recurrences with overlapping subinstances *(cf. to divide-and-conquer, where instances are non-overlapping).*
- "Programming" here means "planning". DP is also named linear programming, mathematical programming.
- Applies to problems where the cost function can be:
  - decomposed into a sequence (ordering) of stages, and
  - each stage depends on only a fixed number of previous stages.
- The cost function need not be convex (if variables are continuous).

DP, the main idea

- Set up a recurrence relating a solution to a larger instance to solutions of some smaller instances.
- Solve smaller instances once.
- Record solutions in a table. *(Some people say that DP means "table filling".)*
- Extract solution to the initial instance from that table.
- DP reduces computation by
  - Solving subproblems in a bottom-up fashion.
  - Storing solution to a subproblem the first time it is solved.
  - Looking up the solution when subproblem is encountered again.

DP inventors

- American mathematician Richard Bellman (1920-1984) introduced the principle of optimality and dynamic programming in 1952.
- Principle of optimality: Given an optimal sequence of decisions or choices, each subsequence must also be optimal.
- Stuart Dreyfus. *Richard Bellman on the Birth of Dynamic Programming.* Operations Research, Vol. 50, No. 1, 48-51, 2002
- Related: Lev Pontryagin's (1908-1988) maximum principle formulated in 1956.



Examples of DP algorithms

- Computing a binomial coefficient
- Longest common subsequence
- Warshall’s algorithm for transitive closure
- Floyd’s algorithm for all-pairs shortest paths
- Constructing an optimal binary search tree
- Some instances of difficult discrete optimization problems:
  - Traveling salesman
  - Knapsack task: *Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.*

Motivation: Fibonacci numbers 1

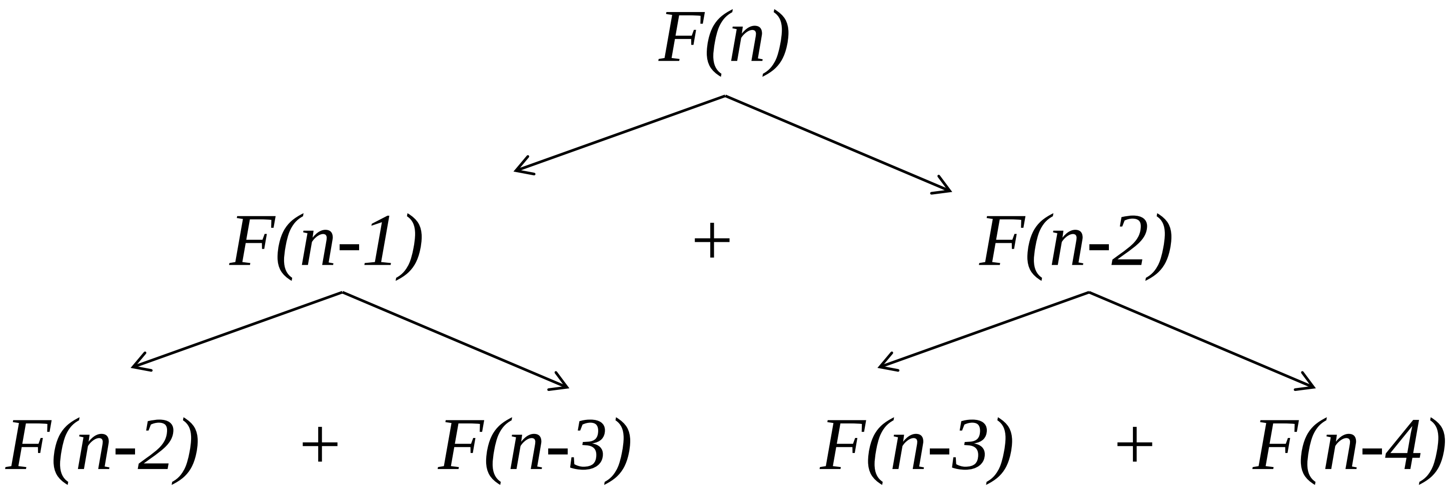
- Fibonacci number, the definition

n	0	1	2	3	4	5	6	7	8	9	10	11	12	…	*n-2*	*n-1*	*n*
F(n)	0	1	1	2	3	5	8	13	21	34	55	89	144	…	*F(n-2)*	*F(n-1)*	*F(n)*

- By Leonardo Fibonacci Pisano, literally "Leonardo, son of Bonacci, of Pisa", (published 1202)
- $F(0) = 0; F(1) = 1; F(2) = 1 + 0 = 1; F(3) = F(2) + F(1) = 2;$
  $F(4) = F(3) + F(2) = 3; \ldots$
  $F(n\text{-}2) = ; F(n\text{-}1) = ; F(n\text{-}1) + F(n\text{-}2)$
- function fib(*n*)
  if $n \leq 1$ return *n*
  return fib(*n-1*) + fib(*n-2*)

Fibonacci numbers 2, example

Computing the n-th Fibonacci number recursively (top-down, also divide-and-conquer strategy):

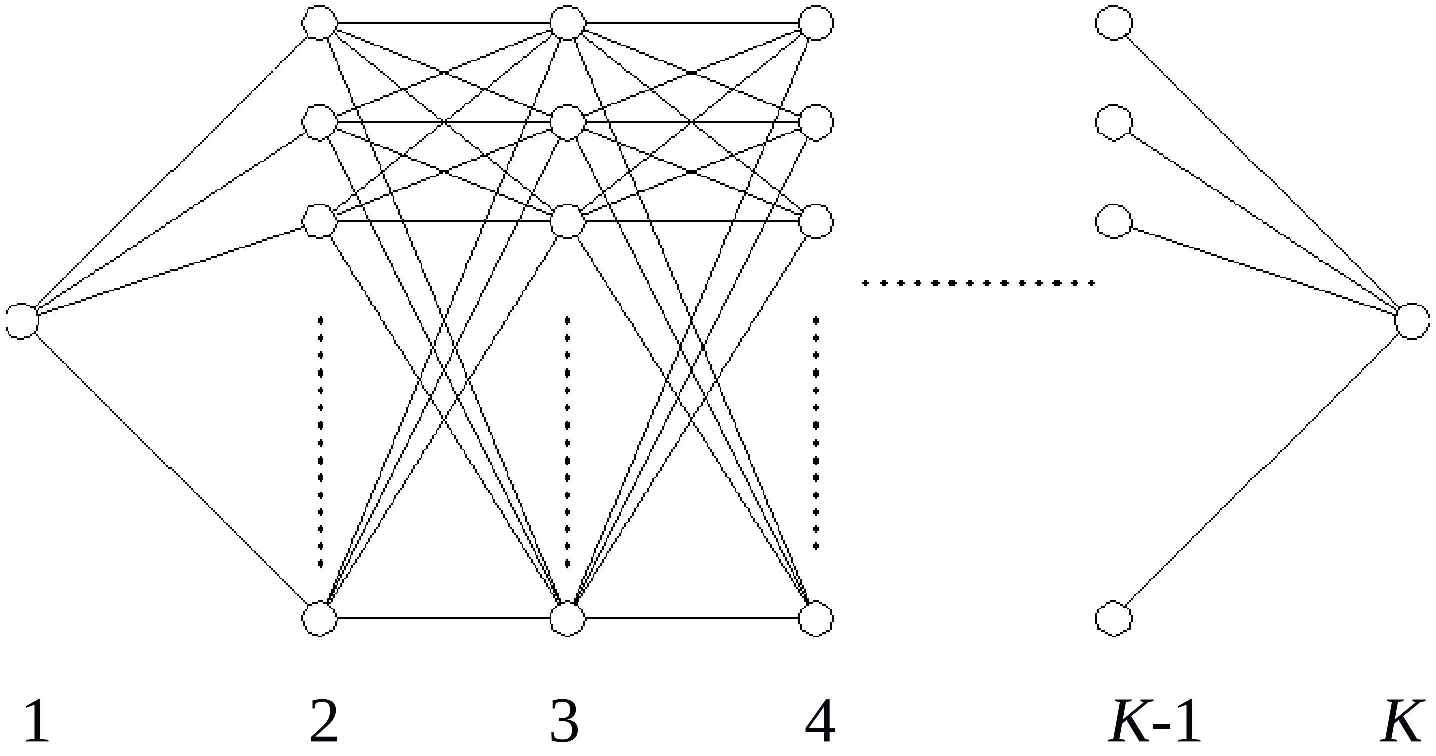


Intuition behind DP

- **The idea:** Compute the solutions to the subsub-problems *once* and store the solutions in a table, so that they can be reused (repeatedly) later.
- **Remark:** We trade space for time.

- **Dynamic programming** (DP) solves every subsubproblem exactly once, and is therefore more efficient in those cases where the subsubproblems are not independent.

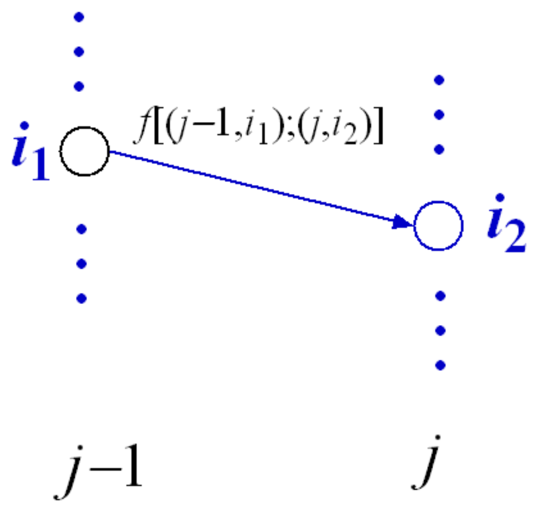
Trellis graph



$$f\big((k-1,i),(k,j)\big)$$

Trellis graph 2

- Distance (weight) from point i_1 at stage $(j-1)$ to point i_2 at stage j:



- The total value of cost function:

$$L = \sum_{j=1}^{K} f\Big((j-1, i_{j-1}), (j, i_j)\Big) \rightarrow \min$$

Preparation for DP

- Cost function:

$$G_k(n) = \min\left\{ \sum_{j=1}^{k} f\left(p_{j-1}(i_{j-1}), p_j(i_j)\right) \right\}$$

- Recursive equation:

$$G_k(n) = \min_i \left\{ G_k(i) + f\left((k-1, i), (k, n)\right) \right\}$$

- Initialization: $G_0(n) = 0$

Complexity issues

- Exhaustive search: $O(n^K)$
- Dynamic programming algorithm: $O(Kn^2)$

where K is the number of stages,

n is the number of points in a stage