# Chapter 15. Dynamic Programming

**Question:** Given a 7-element sequence $5, 3, 13, 6, 8, 7, 10$, the problem is to find a longest increasing subsequence in the sequence.

- 5, 6, 7, 10
- 5, 6, 8, 10
- 3, 6, 7, 10
- 5, 13
- 3, 8, 10
- ...

Notice that although the solution is not unique, the length of all solutions are identical, i.e., 4.

# Chapter 15. Dynamic Programming (cont.)

What's the running time of a naive algorithm for finding an increasing subsequence from a 7-element sequence $5, 3, 13, 6, 8, 7, 10$ by enumerating all possible subsequences?

The answer may not be unique, e.g.,

$$\begin{array}{rcl} A & \to & B[,,,,,,] \\ \{3,6,7,10\}, & \to & \{0,1,0,1,0,1,1\} \\ \{5,6,7,10\}, & \to & \{1,0,0,1,0,1,1\} \\ \{3,6,8,10\}, & \to & \{0,1,0,1,1,0,1\} \\ \{5,6,8,10\}, & \to & \{1,0,0,1,1,0,1\} \end{array}$$

where $B[i] = 0$ if $a_i$ is not in the increasing sequence; otherwise, $B[i] = 1$. The above naive solution takes $O(n \cdot 2^n)$ time.

# Chapter 15. Dynamic Programming

**Dynamic Programming** (DP for short) is a method of solving **an optimization problem (a minimization or maximization problem)**, by first solving some subproblems and then combining the results of subproblems.

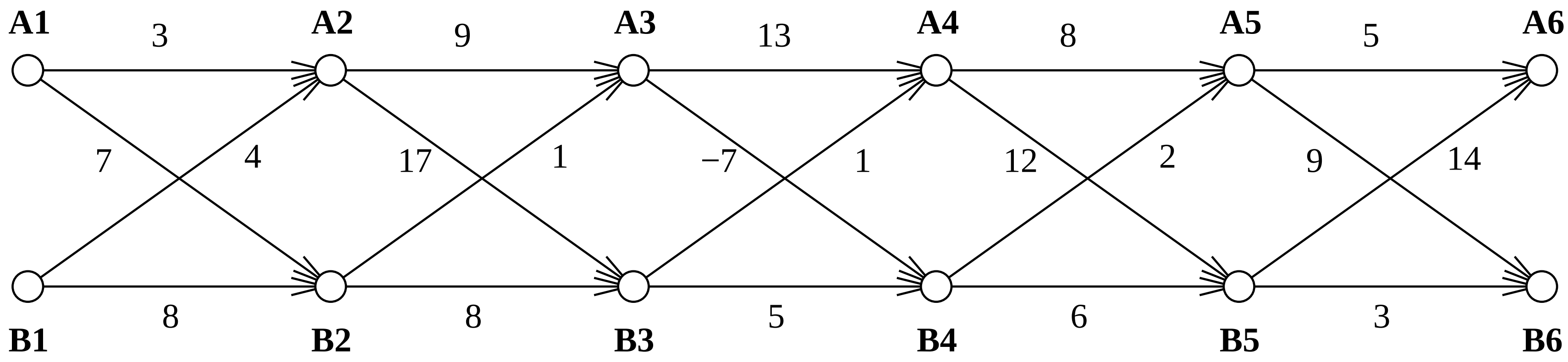
This is also true for the **Divide-and-Conquer** method. However, dynamic programming is useful when the subproblems have a large overlapping with each other, and even have sub-problems in common. The main requirements for dynamic programming are:

- The total number of (sub-)subproblems which may occur is fairly small.
- The solution to each subproblem can be deduced fairly easily from the solutions to the smaller subproblems.

The basic idea of dynamic programming is to solve the subproblems from smallest to largest, storing the solutions in a table.

# Dynamic Programming – multilayer network

Consider a network with $n$ pairs of nodes connected with arrows as in the picture. Each arrow has a length.




The problem is to find a shortest path from the left pair $\{A_1, B_1\}$ to the right pair $\{A_n, B_n\}$.

There are $2^{n-1}$ directed paths (Why?), so we don't want to try them all.

# Dynamic Programming – multilayer network (continued)

The key observation is: any shortest path from $\{A_1, B_1\}$ to $\{A_n, B_n\}$ consists of a shortest path from $\{A_1, B_1\}$ to $\{A_{n-1}, B_{n-1}\}$ plus one more arrow.

So, let us define some variables:

- $L[X,Y]$ = the length of (a directed edge) the arrow from $X$ to $Y$. (e.g., $L[B_2, A_3] = 1$)
- $P(X)$ = the length of the shortest path from $\{A_1, B_1\}$ to $X$.

Then, we have this recurrence:

- $P(A_1) = 0$ and $P(B_1) = 0$
- For $1 \leq i \leq n$:
  $P(A_i) = \min\{P(A_{i-1}) + L[A_{i-1}, A_i], P(B_{i-1}) + L[B_{i-1}, A_i]\}$
  $P(B_i) = \min\{P(A_{i-1}) + L[A_{i-1}, B_i], P(B_{i-1}) + L[B_{i-1}, B_i]\}$

# Dynamic Programming – multilayer network (continued)

Very important: What happens if we program this as a recursive function?

The correct method of solution is to apply the recurrence from smallest to largest:

$P(A_1) = 0;\quad P(B_1) = 0;\quad I(A_1) = 0;\quad I(B_1) = 0$

$P(A_2) = 3;\quad P(B_2) = 7;\quad I(A_2) = A_1;\quad I(B_2) = A_1$

$P(A_3) = 8;\quad P(B_3) = 15;\quad I(A_3) = B_2;\quad I(B_3) = B_2$

$P(A_4) = 16;\quad P(B_4) = 1;\quad I(A_4) = B_3;\quad I(B_4) = A_3$

$P(A_5) = 3;\quad P(B_5) = 7;\quad I(A_5) = B_4;\quad I(B_5) = B_4$

$P(A_6) = 8;\quad P(B_6) = 10;\quad I(A_6) = A_5;\quad I(B_6) = B_5.$

Therefore, the shortest path from $\{A_1, B_1\}$ to $\{A_6, B_6\}$ has length 8.

- ∗ What is the time complexity?
- ∗ How do we find the shortest path, rather than just its length?
- ∗ What is memoisation? (an array $I$ is used for this purpose)