Dynamic Programming

Chapter 15

Dynamic Programming

- We know that we can use the divide-and-conquer technique to obtain efficient algorithms
  - Examples:
- Sometimes, the direct use of divide-and-conquer produces really bad and inefficient algorithms
- Dynamic programming improves inefficient recursive algorithms

Dynamic Programming

- Not really dynamic
- Not really programming
- Name is used for historical reasons
- It comes from the term “mathematical programming”, which is a synonym for optimization.
- “Program” is optimal plan for action that is produced (see Wikipedia!)

Fibonacci Numbers

- Fibonacci numbers are defined by the following recurrence:

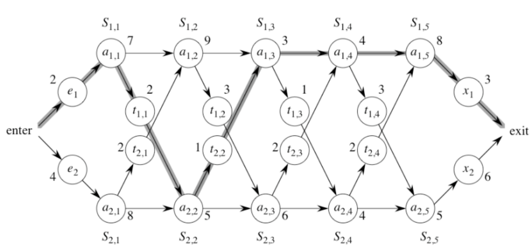
$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n \geq 2 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 \end{cases}$$

- What is the running time of this algorithm?

Four Steps for Dynamic Programming

- Characterize the structure of an optimal solution
- Recursively define the value of an optimal solution
- Compute the value of an optimal solution in a bottom-up fashion
- Construct an optimal solution from computed information

Assembly Line Scheduling



Scheduling

- Factory with two assembly lines
  - Each line has n stations: $S_{1,1} \ldots S_{1,n}$ and $S_{2,1} \ldots S_{2,n}$
  - Corresponding stations perform the same function but take different amounts of time $a_{1,j}$ and $a_{2,j}$
  - Entry times e1 and e2
  - Exit times x1 and x2
  - After going through a station, can either
    - Stay on same line; no cost
    - Transfer to other line; cost after $S_{i,j}$ is $t_{i,j}$

Problem

- Given all these costs, what stations should be chosen from line 1 and from line 2 for fastest way through the factory?

Assembly Line Scheduling

- Can you come up with a solution?
- What is its running time?

Step 1: Structure of Fastest Way

- Think about fastest way from entry through S1,j
  - If j = 1:
  - If j >= 2:

Optimal Substructure

- For assembly line scheduling, an optimal solution to a problem contains within it an optimal solution to subproblems

Step 2: Recursive Solution

- Let $f_i[j]$ = fastest time to get through $S_{i,j}$ where i = 1, 2 and j = 1, 2, …, n
- Goal: fastest time to get all the way through = f*
- f* =
- $f_1[1]$ =
- $f_2[1]$ =

Step 2 Continued

- For j = 2, 3, …, n:
  - f1[j] =
  - f2[j] =

Step 2 Continued

- fi[j] gives the value of an optimal solution. What if we want to construct an optimal solution?
  - li[j] =
  - l* = line # whose station is used

Step 3: Compute an Optimal Solution

- FASTEST-WAY(a, t, e, x, n)

Step 4: Construct Fastest Way

- PRINT-STATIONS(l, n)