EECS 3101 A: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101A`
Also on Moodle

Recall: Divide-and-Conquer

- Divide: If the input size is too large to deal with in a straightforward manner, divide the problem into two or more disjoint subproblems
- Conquer: Use divide and conquer recursively to solve the subproblems
- Combine: Take the solutions to the subproblems and "merge" these solutions into a solution for the original problem

This works when the subproblems are independent

Computing Fibonacci Numbers

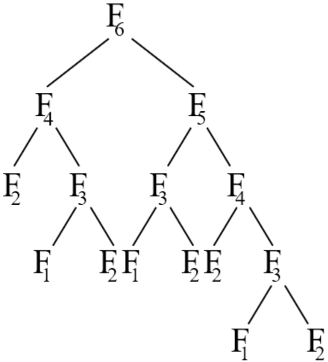
- $F_0 = 0, F_1 = 1$ and for $n > 1$, $F_n = F_{n-1} + F_{n-2}$
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 . . .
- Straightforward recursive procedure:

```
FIBONACCI(n)
if n ≤ 1
        return n
else return Fib[n − 1] + Fib[n − 2]
```

This is slow!

- Why? How slow? Let's draw the recursion tree ...

Computing Fibonacci Numbers - 2



The same subproblems are solved over and over!
We can show that the running time is exponential

Computing Fibonacci Numbers - 3

Options:

- Do not use recursion

  FIBONACCI(n)

  $Fib[0] = 0$
  $Fib[1] = 1$
  for $i = 2$ **to** n
  $\quad Fib[i] = Fib[i-1] + Fib[i-2]$

- Use recursion but store each computed value
  For each recursive call, lookup value if available, else compute it and store

Computing Fibonacci Numbers - Lessons

Options:

- We were able to reduce redundant computation by evaluating the recurrence in a certain order, and remembering previous values.
- This is called memoization (no typo). This is used very often in dynamic programming.

Dynamic Programming (DP)

What is it?

- An algorithmic paradigm
- Used most often for solving optimization problems (we will see some other uses)
- The word "programming" does not refer to computer programming
- Some problems are solved more efficiently using this technique, but others are not
- We will look at several examples where DP works well

Example 1: Optimizing an Itinerary

- We want to go from city 0 to city n using buses
- The only connecting road goes through cities $1, 2, \ldots, n-1$
- The cost of going from city i to city j is c_{ij}
- Assume monotonic paths only (all edges go forward)
- What is the minimum cost of going from 0 to n?

Example 2: A Parsing Problem

Suppose we encode text using the following:
$a : 1, b : 2, \ldots, y : 25, z : 26.$

- Note that the code for b is a prefix for the code for y. So, this is not a prefix-free code
- So parsing is ambiguous:
  Given 1125: possible decodings are $aabe, aay, ale, kbe, ky$
- Problem: Given a string of digits, find the number of valid decodings.

Example 3: Counting Paths on Lattices

You are given a $m \times n$ lattice of points. Starting from the top left corner, you are required to take right and down steps to reach the bottom right corner

- Q: How many different paths are there?
  A: There is an analytical solution
- Suppose that some of the lattice points are marked "no entry"
- Problem: How many different paths are there that avoid these points?

Return to example 1: Optimizing an Itinerary

What is the minimum cost of going from 0 to n?

- This is an optimization (minimization) problem
- Exponential number of paths possible (2 choices at each station – may or may not change buses there, $n - 1$ stations)
- Important property: The optimal cost of going from (say) 2 to 9 has no relation with the same from 11 to 16.
- This independence of subproblems is crucial
- The solution constitutes of a sequence of choices

Optimizing an Itinerary: Ideas

- Step 1: Define subproblems
  We want to make local choices and remember them systematically. Let $T(j)$ be the minimum cost of going from city 0 to city j. So $T(n)$ is the answer.
- What can we say about T(j)?
- Step 2: Express solution recursively
  Suppose someone tells you the best last choice (go from i to n). Does it help?
- Recursively, you can assume you know the best way to go from 0 to i.
- Then you can glue the solutions together and get the optimal solution!

Optimizing an Itinerary: Ideas - 2

- The best way to go from 0 to i is $T(i)$, and $T(i)$ is a smaller subproblem than $T(n)$.

  Aside: When did $T(i)$ go from a cost to a subproblem?

- Then the recursion is $T(n) = c_{in} + T(i)$
- In reality, we do not know the best last choice
- So we take the minimum over all last choice possibilities!

$$T(j) = \min_{k} \left[c_{kj} + T(k)\right], k < j$$

Optimizing an Itinerary: Algorithm

$T(0) = 0$ and for $j > 0$, $T(j) = \min_k [c_{kj} + T(k)], k < j$

- Hopefully we can systematically compute $T(j)$ and get an efficient (polynomial-time) algorithm
- If we do naive recursion, we have the same problems as before
- We can memoize, or
- We can start from $T(1)$. $T(1) = c_{01}$ because there is only one way to get to 1. Then we compute $T(2), T(3), \ldots$ using the recursion above until we reach $T(n)$

Optimizing an Itinerary: Getting the full solution

- $T(n) =$ minimum cost of going from 0 to n. What is the sequence of cities?
- Need to remember more information; Specifically the sequence of choices made.
- $T(j) = \min_k[c_{kj} + T(k)],\ k < j$
  $C(j) = \arg\min k$
- What's the last choice? $C(n)$
- What's the next one? $C(C(n))$!
- The next one is $C(C(C(n)))$. The next one is $C(C(C(C(n))))$. Keep going until you hit 0.

Optimizing an Itinerary: Analysis

- Correctness: Defer for later
- Running time:
  Computing $T(j)$ takes $\Theta(j)$ time. Computing $C(j)$ takes $O(1)$ time. So the algorithm takes $\Theta(n^2)$ time.

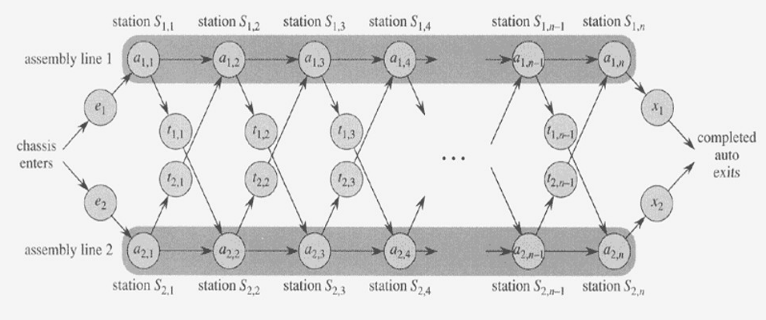
Optimizing an Itinerary: Other DP formulations

- Let $S(i, j)$ be the minimum cost of going from city i to city j. So $S(0, n)$ is the answer.

- How does the efficiency compare with the previous formulation?

An Activity Selection Problem

- Two assembly lines, A_i, B_i, each with n stations
- Each job must complete go through A_i or B_i for each i
- Different costs for going from A_i to B_{i+1}, A_i to A_{i+1}, B_i to B_{i+1} , B_i to A_{i+1}, start to A_1, start to B_1, A_n to exit, B_n to exit.



An Activity Selection Problem - 2

- Exponential number of paths possible (2 choices, n stations)
- Again, suppose you know the first choice. Does that help?
- Can we express the cost recursively?
- Add the costs of the first choice and the best path for the remainder of the job
- Because we do not know the best first choice, we take the minimum over all the possible ones

An Activity Selection Problem - Algorithm

Define $f_1[j]$ to be the cost of going to the j^{th} station on assembly line 1 from the start. Define $f_2[j]$ similarly for assembly line 2. Then:

- $$f_1[j] = \begin{cases} e_1 + a_{1,1}, & \text{if } j = 1 \\ \min[f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}] & \text{if } j > 1 \end{cases}$$
- Similarly for $f_2[j]$
- Finally, $f^* = \min[f_1[n] + x_1, f_2[n] + x_2]$

Activity Selection - Constructing Solutions

- Remember the choices made in an array $l[]$

```
PRINT-STATIONS(l, n)
i ← l*
print "line " i ", station " n
for j ← n downto 2
      do i ← l_i[j]
         print "line " i ", station " j − 1
```

- Running Time: Constant amount of work to compute $f_1[j], f_2[j]$, for each j, and for f^*. Total running time $\Theta(n)$.