

Algorithms 2012: Design by Induction

(Based on [Manber 1989])

Yih-Kuen Tsay

1 Introduction

Introduction

- It is not necessary to design the steps required to solve a problem from scratch.
- It is sufficient to guarantee the following:
 1. It is possible to solve one small instance or a few small instances of the problem. (base case)
 2. A solution to every problem/instance can be constructed from solutions to smaller problems/instances. (inductive step)

2 Evaluating Polynomials

Evaluating Polynomials

Problem 1. *Given a sequence of real numbers $a_n, a_{n-1}, \dots, a_1, a_0$, and a real number x , compute the value of the polynomial*

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0.$$

Motivation: different approaches to the inductive step may result in algorithms of very different time complexities.

Evaluating Polynomials (cont.)

- Let $P_{n-1}(x) = a_{n-1} x^{n-1} + \dots + a_1 x + a_0$.
- **Induction hypothesis** (first attempt)
We know how to evaluate a polynomial represented by the input a_{n-1}, \dots, a_1, a_0 , at the point x , i.e., we know how to compute $P_{n-1}(x)$.
- $P_n(x) = a_n x^n + P_{n-1}(x)$.

Evaluating Polynomials (cont.)

- **Induction hypothesis** (second attempt)
We know how to compute $P_{n-1}(x)$, and we know how to compute x^{n-1} .
- $P_n(x) = a_n x(x^{n-1}) + P_{n-1}(x)$.

Evaluating Polynomials (cont.)

- Let $P'_{n-1}(x) = a_n x^{n-1} + a_{n-1} x^{n-2} + \dots + a_1$.
- **Induction hypothesis** (final attempt)
We know how to evaluate a polynomial represented by the coefficients a_n, a_{n-1}, \dots, a_1 , at the point x , i.e., we know how to compute $P'_{n-1}(x)$.
- $P_n(x) = P'_n(x) = P'_{n-1}(x) \cdot x + a_0$.

Evaluating Polynomials (cont.)

- More generally,

$$\begin{cases} P'_0(x) = a_n \\ P'_i(x) = P'_{i-1}(x) \cdot x + a_{n-i}, \text{ for } 1 \leq i \leq n \end{cases}$$

Evaluating Polynomials (cont.)

Algorithm Polynomial_Evaluation (\bar{a}, x);

begin

$P := a_n$;

for $i := 1$ **to** n **do**

$P := x * P + a_{n-i}$

end

This algorithm is known as *Horner's rule*.

3 Maximal Induced Subgraph

Maximal Induced Subgraph

Problem 2. Given an undirected graph $G = (V, E)$ and an integer k , find an induced subgraph $H = (U, F)$ of G of maximum size such that all vertices of H have degree $\geq k$ (in H), or conclude that no such induced subgraph exists.

Design Idea: in the inductive step, we try to remove one vertex (that cannot possibly be part of the solution) to get a smaller instance.

4 One-to-One Mapping

One-to-One Mapping

Problem 3. Given a finite set A and a mapping f from A to itself, find a subset $S \subseteq A$ with the maximum number of elements, such that (1) the function f maps every element of S to another element of S (i.e., f maps S into itself), and (2) no two elements of S are mapped to the same element (i.e., f is one-to-one when restricted to S).

Design Idea: similar to the previous problem; in the inductive step, we try to remove one element (that cannot possibly be part of the solution) to get a smaller instance.

One-to-One Mapping (cont.)

```
Algorithm Mapping ( $f, n$ );
begin
   $S := A$ ;
  for  $j := 1$  to  $n$  do  $c[j] := 0$ ;
  for  $j := 1$  to  $n$  do increment  $c[f[j]]$ ;
  for  $j := 1$  to  $n$  do
    if  $c[j] = 0$  then put  $j$  in Queue;
  while Queue not empty do
    remove  $i$  from the top of Queue;
     $S := S - \{i\}$ ;
    decrement  $c[f[i]]$ ;
    if  $c[f[i]] = 0$  then put  $f[i]$  in Queue
end
```

5 Celebrity

Celebrity

Problem 4. Given an $n \times n$ adjacency matrix, determine whether there exists an i (the “celebrity”) such that all the entries in the i -th column (except for the ii -th entry) are 1, and all the entries in the i -th row (except for the ii -th entry) are 0.

Note: A celebrity corresponds to a sink of the directed graph.

Note: Every directed graph has at most one sink.

Motivation: the trivial solution has a time complexity of $O(n^2)$. Can we do better, in $O(n)$?

Celebrity (cont.)

```
Algorithm Celebrity ( $Know$ );
begin
   $i := 1$ ;
   $j := 2$ ;
   $next := 3$ ;
  while  $next \leq n + 1$  do
    if  $Know[i, j]$  then  $i := next$ 
    else  $j := next$ ;
     $next := next + 1$ ;
  if  $i = n + 1$  then  $candidate := j$ 
  else  $candidate := i$ ;
```

Celebrity (cont.)

```
 $wrong := false$ ;
 $k := 1$ ;
 $Know[candidate, candidate] := false$ ;
while not  $wrong$  and  $k \leq n$  do
```

```

    if Know[candidate, k] then wrong := true;
    if not Know[k, candidate] then
      if candidate ≠ k then wrong := true;
      k := k + 1;
    if not wrong then celebrity := candidate
      else celebrity := 0;
end

```

6 The Skyline Problem

The Skyline Problem

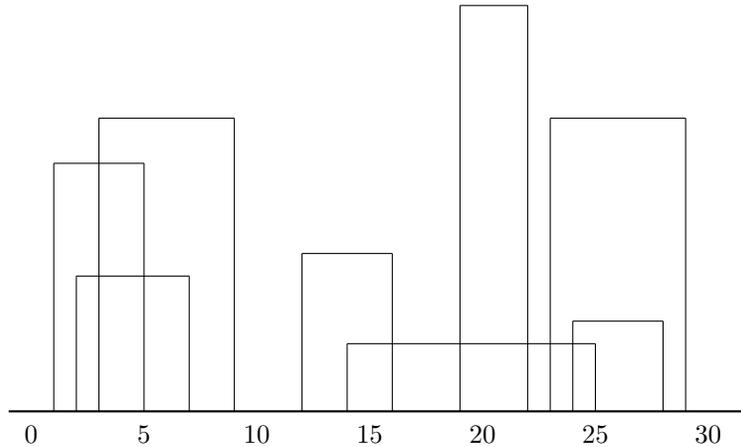
Problem 5. *Given the exact locations and shapes of several rectangular buildings in a city, draw the skyline (in two dimension) of these buildings, eliminating hidden lines.*

Motivation: different approaches to the inductive step may result in algorithms of very different time complexities.

Compare: adding buildings one by one to an existing skyline **vs.** merging two skylines of about the same size

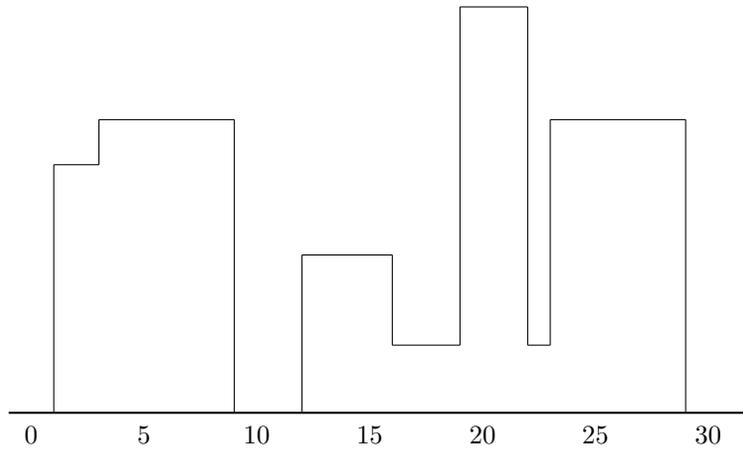
Representation of a Skyline

(1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), and (24,4,28).



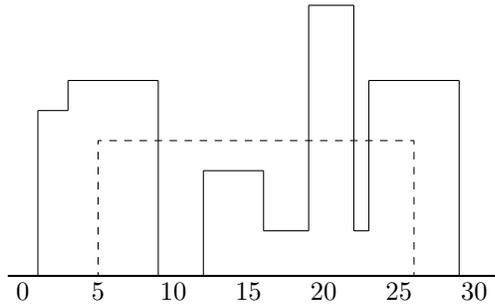
Representation of a Skyline (cont.)

(1,11,3,13,9,0,12,7,16,3,19,18,22,3,23,13,29).



Adding a Building

- Add (5,9,26) to (1,11,3,13,9,0,12,7,16,3,19,18,22,3,23,13,29).



- The skyline becomes (1,11,3,13,9,9,19,18,22,9,23,13,29).

Merging Two Skylines

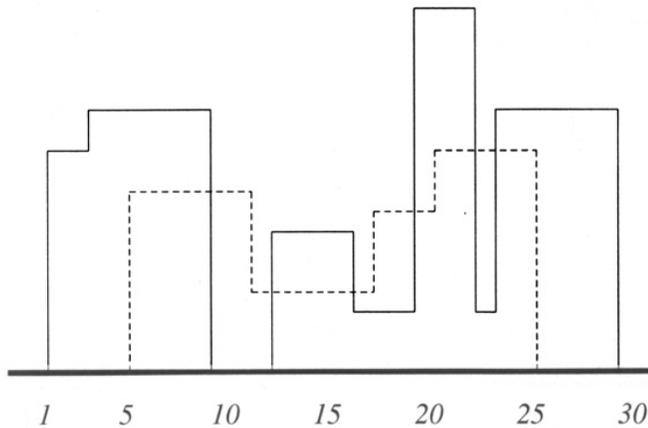


Figure 5.7 Merging two skylines.

Source: [Manber 1989].

7 Balance Factors in Binary Trees

Balance Factors in Binary Trees

Problem 6. Given a binary tree T with n nodes, compute the balance factors of all nodes.

The balance factor of a node is defined as the **difference** between the height of the node's left subtree and the height of the node's right subtree.

Motivation: an example of why we must strengthen the hypothesis (and hence the problem to be solved).

Balance Factors in Binary Trees (cont.)

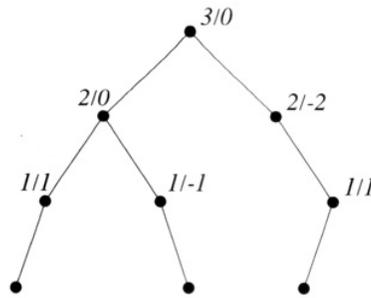


Figure 5.8 A binary tree. The numbers represent h/b , where h is the height and b is the balance factor.

Source: [Manber 1989].

Balance Factors in Binary Trees (cont.)

- **Induction hypothesis**
We know how to compute balance factors of all nodes in trees that have $< n$ nodes.
- **Stronger induction hypothesis**
We know how to compute balance factors and heights of all nodes in trees that have $< n$ nodes.

8 Maximum Consecutive Subsequence

Maximum Consecutive Subsequence

Problem 7. Given a sequence x_1, x_2, \dots, x_n of real numbers (not necessarily positive) find a subsequence x_i, x_{i+1}, \dots, x_j (of consecutive elements) such that the sum of the numbers in it is maximum over all subsequences of consecutive elements.

Example: In the sequence $(2, -3, 1.5, -1, 3, -2, -3, 3)$, the maximum subsequence is $(1.5, -1, 3)$.

Motivation: another example of strengthening the hypothesis.

Maximum Consecutive Subsequence (cont.)

- **Induction hypothesis**

We know how to find the maximum subsequence in sequences of size $< n$.

- **Stronger induction hypothesis**

We know how to find, in sequences of size $< n$, the maximum subsequence overall and the maximum subsequence that is a suffix.

Maximum Consecutive Subsequence (cont.)

Algorithm Max_Consec_Subseq (X, n);

begin

$Global_Max := 0$;

$Suffix_Max := 0$;

for $i := 1$ **to** n **do**

if $x[i] + Suffix_Max > Global_Max$ **then**

$Suffix_Max := Suffix_Max + x[i]$;

$Global_Max := Suffix_Max$

else if $x[i] + Suffix_Max > 0$ **then**

$Suffix_Max := Suffix_Max + x[i]$

else $Suffix_Max := 0$

end

9 The Knapsack Problem

The Knapsack Problem

Problem 8. Given an integer K and n items of different sizes such that the i -th item has an integer size k_i , find a subset of the items whose sizes sum to exactly K , or determine that no such subset exists.

Design Idea: use strong induction so that solutions to all smaller instances may be used.

The Knapsack Problem (cont.)

- Let $P(n, K)$ denote the problem where n is the number of items and K is the size of the knapsack.

- **Induction hypothesis**

We know how to solve $P(n - 1, K)$.

- **Stronger induction hypothesis**

We know how to solve $P(n - 1, k)$, for all $0 \leq k \leq K$.

The Knapsack Problem (cont.)

An example of the table constructed for the knapsack problem:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	O	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$k_1 = 2$	O	-	I	-	-	-	-	-	-	-	-	-	-	-	-	-	-
$k_2 = 3$	O	-	O	I	-	I	-	-	-	-	-	-	-	-	-	-	-
$k_3 = 5$	O	-	O	O	-	O	-	I	I	-	I	-	-	-	-	-	-
$k_4 = 6$	O	-	O	O	-	O	I	O	O	I	O	I	-	I	I	-	I

“I”: a solution containing this item has been found.

“O”: a solution without this item has been found.

“-”: no solution has yet been found.

The Knapsack Problem (cont.)

Algorithm Knapsack (S, K);

$P[0, 0].exist := true$;

for $k := 1$ **to** K **do**

$P[0, k].exist := false$;

for $i := 1$ **to** n **do**

for $k := 0$ **to** K **do**

$P[i, k].exist := false$;

if $P[i - 1, k].exist$ **then**

$P[i, k].exist := true$;

$P[i, k].belong := false$

else if $k - S[i] \geq 0$ **then**

if $P[i - 1, k - S[i]].exist$ **then**

$P[i, k].exist := true$;

$P[i, k].belong := true$