11 A: Algorithm Design Techniques

CS1102S: Data Structures and Algorithms

Martin Henz

March 31, 2010

Generated on Tuesday 6th April, 2010, 14:22

1. NP-Complete Problems
2. Greedy Algorithms
3. Divide and Conquer
4. Dynamic Programming

Example: Hamiltonian Cycles

Given Input

An undirected connected graph

Desired Output

A path that starts and ends in the same vertex and contains all other vertices exactly once.

No efficient algorithm known

We do not know if there is a k such that the problem can be solved in $O(N^k)$.

Our Last Question

Question

If we do not have a polynomial algorithm, can we always prove that there is none?

Answer

No: we cannot (at this moment) prove that there is no polynomial algorithm for the Hamiltonian cycle problem

Verifying Solutions

Example: Hamiltonian cycle problem

If we have a candidate of a solution to the problem, we can easily check that it is correct.
Simply check that the last vertex in the cycle is that same as the first, and that every vertex of the graph is contained in the cycle.

Definition

We call the class of problems for which solution candidates can be checked in polynomial time *NP*.

NP-Complete Problems

Other problems in NP

- Boolean satisfiability problem (SAT)
- Graph coloring problem
- Clique problem

Reducibility

All these problems can be transformed into each other in polynomial time! Thus if we can solve one, we can solve all.

NP-Complete Problems

The class of problems that can be transformed into the Hamiltonian path problem in polynomial time is called the class of *NP-complete problems*.

1 NP-Complete Problems

2 Greedy Algorithms
- Scheduling
- Huffman Codes

3 Divide and Conquer

4 Dynamic Programming

Nonpreemptive Scheduling

Input

A set of jobs with a running time for each

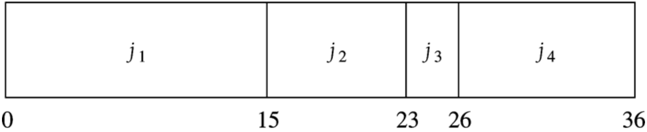
Desired output

A sequence for the jobs to execute on on single machine, minimizing the average completion time

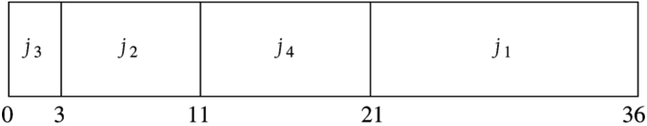
Example

Job	Time
j_1	15
j_2	8
j_3	3
j_4	10

Some schedule:



The optimal schedule:

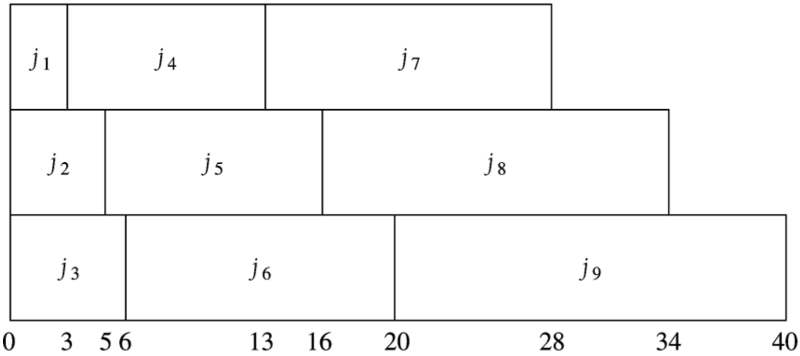


The Multiprocessor Case

N processors

Now we can run the jobs on N identical machines. What is a schedule that minimizes the average completion time?

Example



A “Slight” Variant

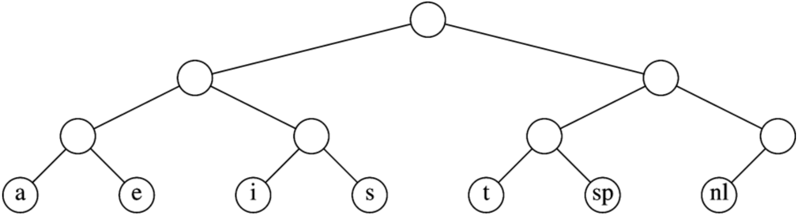
Miniming *final* completion time

If we want to minimize the *final* completion time (completion time of the last task), the problem becomes NP-complete!

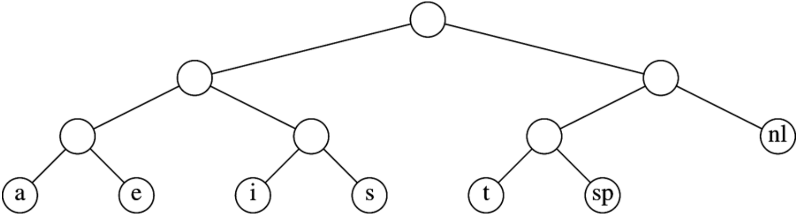
Standard Coding Scheme

Character	Code	Frequency	Total Bits
a	000	10	30
e	001	15	45
i	010	12	36
s	011	3	9
t	100	4	12
space	101	13	39
newline	110	1	3
Total			174

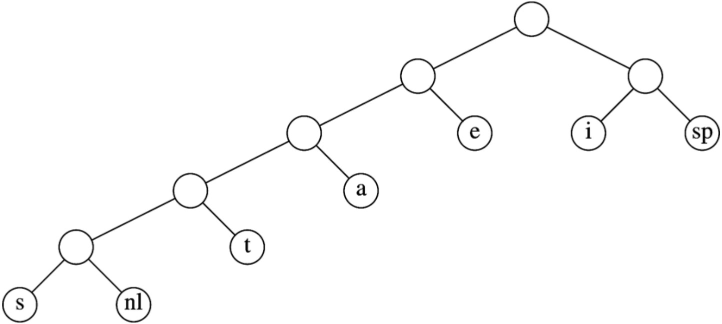
Representation in a Tree



A Slightly Better Representation



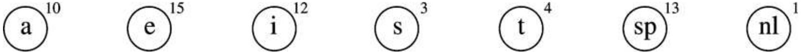
Optimal Prefix Code in Tree Form



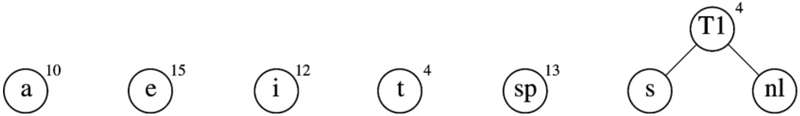
Optimal Prefix Code in Table Form

Character	Code	Frequency	Total Bits
a	001	10	30
e	01	15	30
i	10	12	24
s	00000	3	15
t	0001	4	16
space	11	13	26
newline	00001	1	5
Total			146

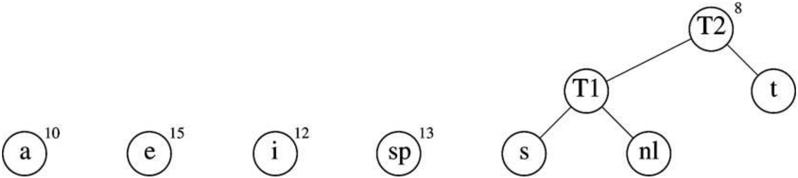
Huffman's Algorithm: An Example



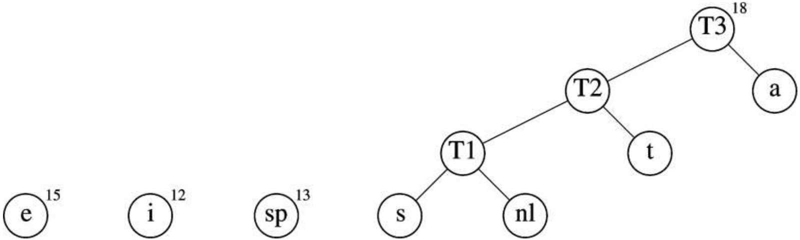
Huffman's Algorithm: An Example



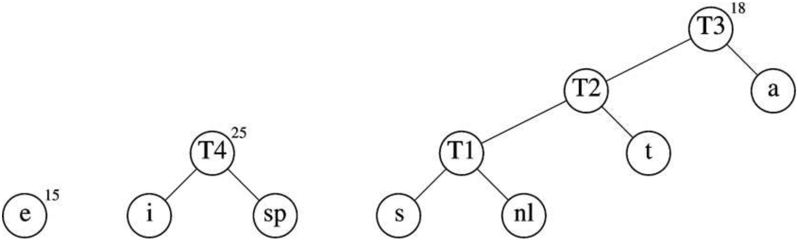
Huffman's Algorithm: An Example



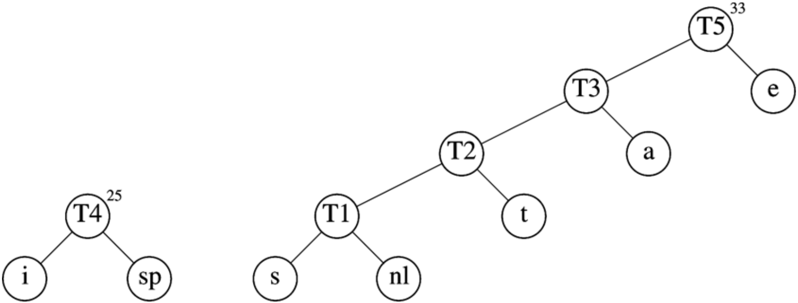
Huffman's Algorithm: An Example



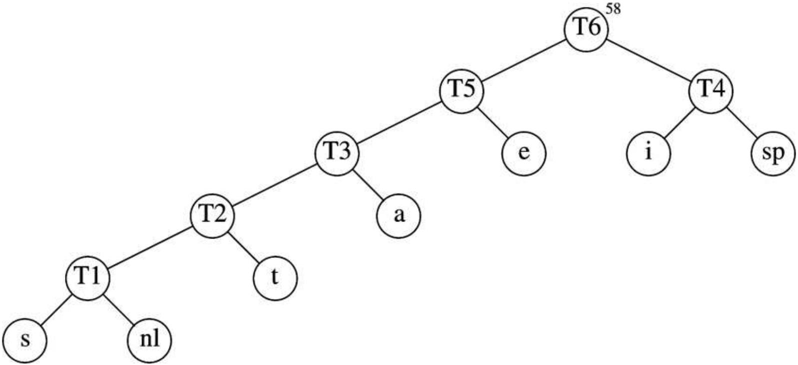
Huffman's Algorithm: An Example



Huffman's Algorithm: An Example



Huffman's Algorithm: An Example



1 NP-Complete Problems

2 Greedy Algorithms

3 Divide and Conquer
- Sorting
- Running Time
- Closest-Points Problem

4 Dynamic Programming

Sorting using Divide and Conquer

Idea of Mergesort

Split array in two (trivial); sort the two (recursively); merge (linear)

Idea of Quicksort

Split array in two, using pivot (linear); sort the two (recursively); merge (trivial)

Running Time of Divide and Conquer Algorithms

Merge Sort: $T(N) = 2T(N/2) + O(N)$

$$O(N \log N)$$

Generalization: $T(N) = aT(N/b) + \Theta(N^k)$

$$\begin{aligned} T(N) &= O(N^{\log_b a}) && \text{if } a > b^k \\ T(N) &= O(N^k \log N) && \text{if } a = b^k \\ T(N) &= O(N^k) && \text{if } a < b^k \end{aligned}$$

Closest-Points Problem

Input

Set of points in a plane

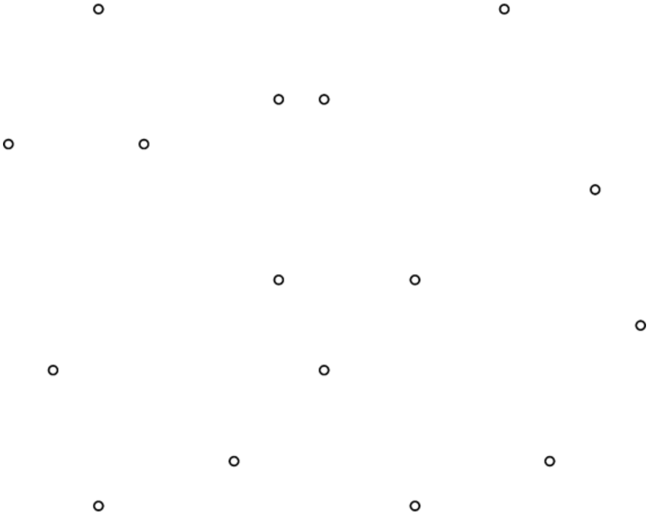
Euclidean distance between p_1 and p_2

$$[(x_1 - x_2)^2 + (y_1 - y_2)^2]^{1/2}$$

Required output

Find the closest pair of points

Example



Naive Algorithm

Exhaustive search

Compute the distance between each two points and keep the smallest

Run time

There are N^2 pairs to check, thus $O(N^2)$

Idea

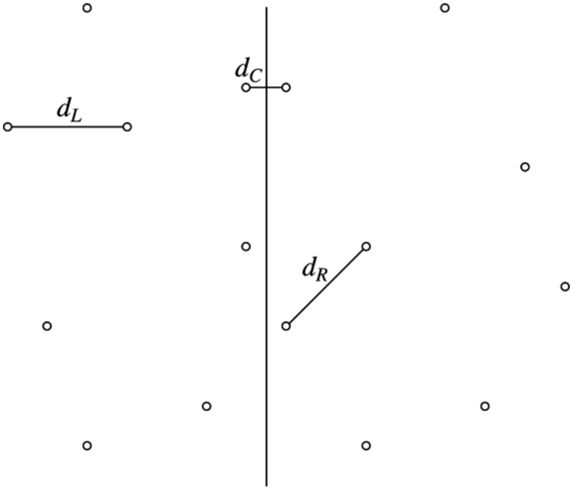
Preparation

Sort points by x coordinate; $O(N \log N)$

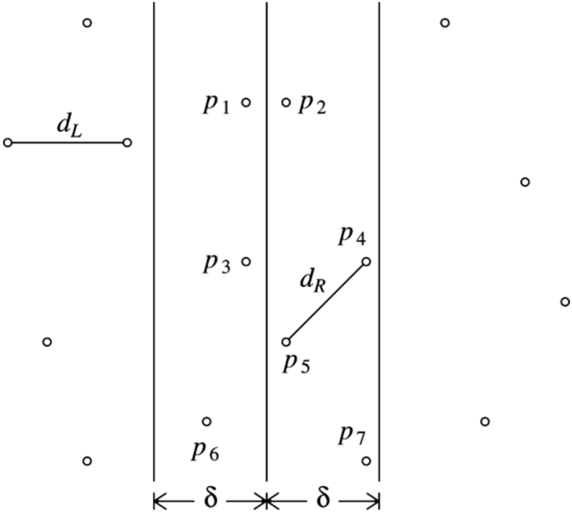
Divide and Conquer

Split point set into two halves, P_L and P_R.
Recursively find the smallest distance in each half.
Find the smallest distance of pairs that *cross* the separation line.

Partitioning with Shortest Distances Shown



Two-lane Strip



Brute Force Calculation of $min(\delta, d_C)$

```
// Points are all in the strip

for( i = 0; i < numPointsInStrip; i++ )
    for( j = i + 1; j < numPointsInStrip; j++ )
        if( dist(p_i, p_j) < δ )
            δ = dist(p_i, p_j);
```

Better Idea

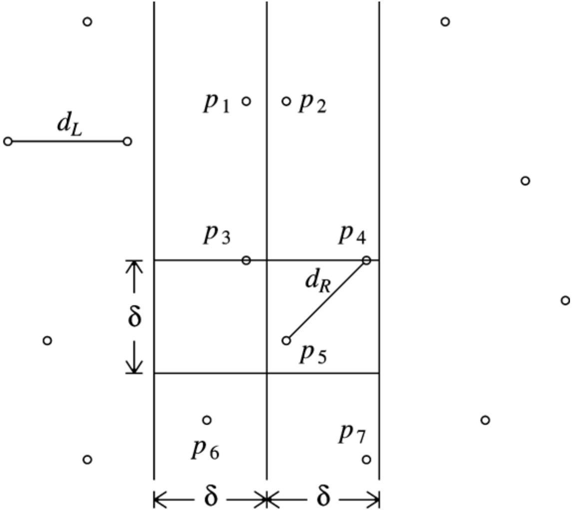
Sort points by y coordinate

This allows a *scan* of the strip.

Sort points by y coordinate

This allows a *scan* of the strip.

Only p_4 and p_5 Need To Be Considered

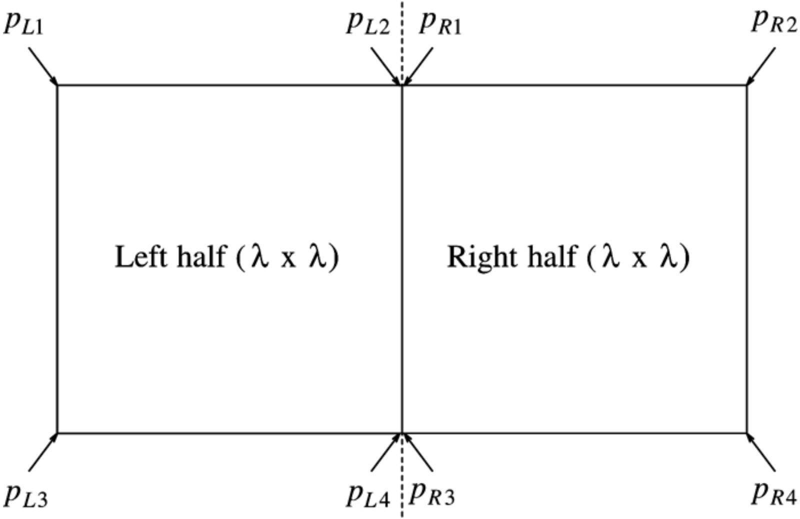


Refined Calculation of $min(\delta, d_C)$

```
// Points are all in the strip and sorted by y-coordinate

for( i = 0; i < numPointsInStrip; i++ )
    for( j = i + 1; j < numPointsInStrip; j++ )
        if( p_i and p_j's y-coordinates differ by more than δ )
            break;        // Go to next p_i.
        else
        if( dist(p_i, p_j) < δ )
            δ = dist(p_i, p_j);
```

At Most Eight Points Fit in Rectangle

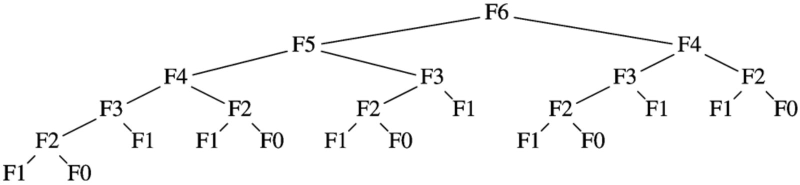


1. NP-Complete Problems
2. Greedy Algorithms
3. Divide and Conquer
4. Dynamic Programming
   - Fibonacci Numbers
   - Optimal Binary Search Tree
   - All-pairs Shortest Path

Inefficient Algorithm

```
public static int fib(int n) {
  if (n <= 1)
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}
```

Trace of Recursion



Memoization

```
int[] fibs = new int[100];
public static int fib(int n) {
  if (fibs[n]!=0) return fibs[n];
  if (n <= 1) return 1;
  int new_fib = fib(n - 1) + fib(n - 2);
  fibs[n] = new_fib;
  return new_fib;
}
```

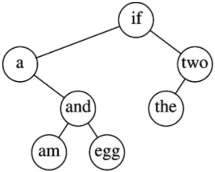
A Simple Loop for Fibonacci Numbers

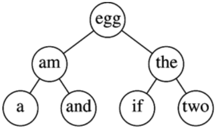
```
public static int fib(int n) {
  if (n <= 1) return 1;
  int last = 1, nextToLast = 1; answer = 1;
  for (int i = 2; i <= n; i++) {
    answer = last + nextToLast;
    nextToLast = last;
    last = answer;
  }
  return answer;
}
```

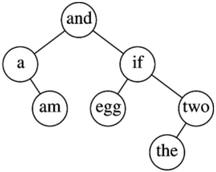
Sample Input

Word	Probability
a	0.22
am	0.18
and	0.20
egg	0.05
if	0.25
the	0.02
two	0.08

Three Possible Binary Search Trees



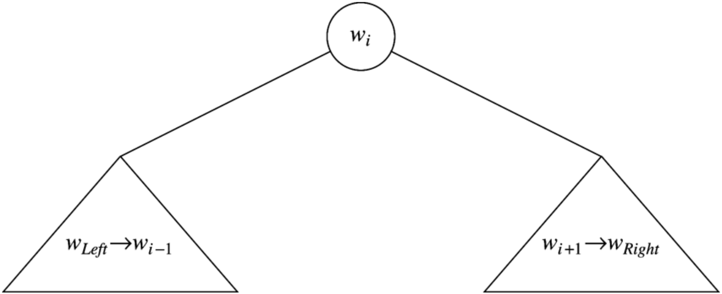




Comparison of the Three Trees

Input		Tree #1		Tree #2		Tree #3	
Word w_i	Probability p_i	Access Cost Once	Access Cost Sequence	Access Cost Once	Access Cost Sequence	Access Cost Once	Access Cost Sequence
a	0.22	2	0.44	3	0.66	2	0.44
am	0.18	4	0.72	2	0.36	3	0.54
and	0.20	3	0.60	3	0.60	1	0.20
egg	0.05	4	0.20	1	0.05	3	0.15
if	0.25	1	0.25	3	0.75	2	0.50
the	0.02	3	0.06	2	0.04	4	0.08
two	0.08	2	0.16	3	0.24	3	0.24
Totals	1.00		2.43		2.70		2.15

Structure of Optimal Binary Search Tree



Idea

Proceed in order of growing tree size

For each range of words, compute optimal tree

Memoization

For each range, store optimal tree for later retrieval

Computation of Optimal Binary Search Tree

	Left=1		Left=2		Left=3		Left=4		Left=5		Left=6		Left=7	
Iteration=1	a..a		am..am		and..and		egg..egg		if..if		the..the		two..two	
	.22	a	.18	am	.20	and	.05	egg	.25	if	.02	the	.08	two
Iteration=2	a..am		am..and		and..egg		egg..if		if..the		the..two			
	.58	a	.56	and	.30	and	.35	if	.29	if	.12	two		
Iteration=3	a..and		am..egg		and..if		egg..the		if..two					
	1.02	am	.66	and	.80	if	.39	if	.47	if				
Iteration=4	a..egg		am..if		and..the		egg..two							
	1.17	am	1.21	and	.84	if	.57	if						
Iteration=5	a..if		am..the		and..two									
	1.83	and	1.27	and	1.02	if								
Iteration=6	a..the		am..two											
	1.89	and	1.53	and										
Iteration=7	a..two													
	2.15	and												

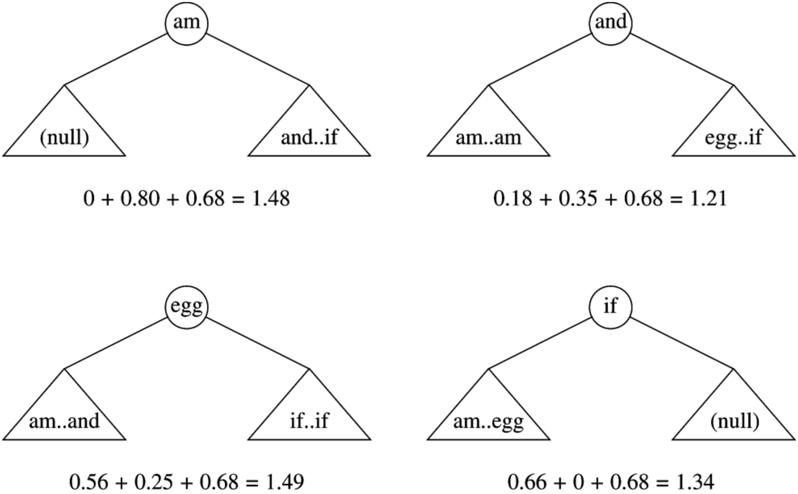
Run Time

For each cell of table
Consider all possible roots

Overall runtime

$$O(N^3)$$

Example


am
(null)
and..if
0 + 0.80 + 0.68 = 1.48
and
am..am
egg..if
0.18 + 0.35 + 0.68 = 1.21
egg
am..and
if..if
0.56 + 0.25 + 0.68 = 1.49
if
am..egg
(null)
0.66 + 0 + 0.68 = 1.34