Final Exam

Saturday, June 04, 2016 9:43 PM

Matthew Koken

~~1~~ ~~2~~ ~~3~~ ~~4~~ 5 6~~ab~~c (2) 7~~ab~~c

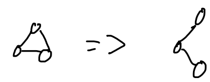
algf5

Name: Matthew Koken

CSCI 163/COEN 179 Spring 2016

Final

You may use your book and notes but no other sources of information. Explain all answers clearly. The due date is Wednesday, June 8. If you will submit later than 3pm on the 8th, then please scan and e-mail me your answers. Prior to 3pm, you may submit either a hard copy or a scan.



1. Explain why Depth First Trees have no cross edges, while Breadth First trees have no back edges.

DFS: Can only discover tree and back edges. If a vertex is already discovered/visited, then it is a back-edge - return to an ancestor - a ancestor of the source vertex.

p - parent vertex, c child vertex

Case: edge (p, c). c discovered, not closed - subtree of root p

Case: edge (p, c). c discovered and closed - impossible, edge (c, p) it just discovered.

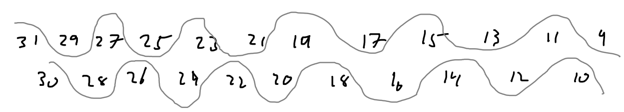
So an edge leading to a discovered vertex must be a back-edge, no cross edges.

BFS: Only tree, cross edges. Unvisited vertex becomes attached by a tree edge as a child. An already visited is already attached, so it gets a cross-edge. If it's already connected, it isn't necessarily a descendant, it's a sibling - so it can't be a back-edge. A back edge would have been used from visiting the ancestor, so you won't have them

2. The recursive version of the Fast Fourier Transform is used to evaluate

$$p(z) = a_{31}z^{31} + a_{30}z^{30} + \ldots + a_1 z + a_0$$

at the 32nd roots of unity. Along the way, it must evaluate some cubic polynomials at the 4th roots of unity. List these cubic polynomials.



1st root)

$P_0(z) = a_{31}z^{15} + a_{29}z^{14} + a_{27}z^{13} + \ldots \; a_1$

$P_E(z) = a_{30}z^{15} + a_{28}z^{14} + \ldots + a_0$

4th root

$P_0(z) = a_{31}z^7 + \ldots \; a_3$

$P_E(z) = a_{29}z^7 + \ldots \; a_1$ =>

$P_0(z) = a_{31}z^7 + a_{27}z^6 + a_{23}z^5 + a_{19}z^4 + a_{15}z^3 + a_{11}z^2 + a_7 z + a_3$

$P_E(z) = a_{29}z^7 + a_{25}z^6 + a_{21}z^5 + a_{17}z^4 + a_{13}z^3 + a_9 z^2 + a_5 z + a_1$

$\bar{P}_0(z) = a_{30}z^7 + a_{26}z^6 + a_{22}z^5 + a_{18}z^4 + a_{14}z^3 + a_{10}z^2 + a_6 z + a_2$

$P_E(z) = a_{28}z^7 + a_{24}z^6 + a_{20}z^5 + a_{16}z^4 + a_{12}z^3 + a_8 z^2 + a_4 z + a_0$

cubic polynomials (but 8th root of unity)

↳ wasn't sure what was wanted so went the extra step just in case

31 27 23 19 15 11 7 3

$P_0(z): a_{31}z^3 + a_{23}z^2 + a_{15}z + a_7$

$P_E(z): a_{27}z^3 + a_{19}z^2 + a_{11}z + a_3$

29 25 21 17 13 9 5 1

$P_0(z) = a_{29}z^3 + a_{21}z^2 + a_{13}z + a_5$

$P_E(z) = a_{25}z^3 + a_{17}z^2 + a_9 z + a_1$

30 26 22 18 14 10 6 2

$P_0(z) = a_{30}z^3 + a_{22}z^2 + a_{14}z + a_6$

$P_E(z) = a_{26}z^3 + a_{18}z^2 + a_{10}z + a_2$

[illegible]

$P_0(z) = a_{28}z^3 + a_{20} + z^2 + a_{12}z + a_4$

28 24 20 16 12 8 4 0 \ $P_E(z) = a_{24}z^3 + a_{16}z^2 + a_8 z + a_0$

3. In Strazzzen's algorithm, 3x3 matrices are multiplied by another 3x3 matrices using 25 multiplications and 100 additions/subtractions. As with Strassen, Strazzzen's generalizes to cover multiplying 3^kx3^k matrices, by subdividing each matrix into 9 3^{k-1}x3^{k-1}'s. Write down a recurrence for the number of operations $T(n)$ that Strazzzen uses to multiply two nxn matrices, where $n = 3^k$ for some k. Find the order of $T(n)$.

Strassen $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$ $C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$,

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$
$C_{12} = A_{11}B_{12} + A_{12}B_{22}$
$C_{21} = A_{21}B_{11} + A_{22}B_{21}$
$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

$\Rightarrow$ $M_1 = (A_{11}+A_{22})(B_{11}+B_{22})$
$M_2 = (A_{21}+a_{22})B_{11}$
$M_3 = a_{11}(B_{12}-b_{22})$
$M_4 = a_{22}(b_{21}-b_{11})$
$M_5 = (a_{11}+a_{12})b_{22}$
$M_6 = (a_{21}-a_{11})(b_{11}+b_{12})$
$M_7 = (a_{12}-a_{22})(b_{21}+b_{22})$

$\Rightarrow C_{11} = M_1 + M_4 - M_5 + M_7$
$C_{12} = M_3 + M_5$
$C_{21} = M_2 + M_4$
$C_{22} = M_1 - M_2 + M_3 + M_6$

$\underline{7}(2^{k-1} \times 2^{k-1})$ mult, $7 \times 18(2^{k-1} \times 2^{k-1})$ add/sub

- mults: $T(n) = 7 \cdot T(n/2) = 7^2 \cdot T(n/4) \cdots = 7^k T(1) = 7^k$

- $n = 2^k$

$k = \log_2 n$

$7^k = 7^{\log_2 n} = n^{\log_2 7} \sim n^{2.81}$

- adds & subs, $T(n) = 7 \cdot T(n/2) + 18(n/2)^2$

$a=7,\ b=2 \quad d = \log_2 7 \approx 2.81$

$f(n) = \Theta(n^2) < n^d$

$T(n) = \Theta(n^{\log_2 7})$

Strazzzen:

$n = 3^k$

$\underline{9}(3^{k-1} \times 3^{k-1})$ matrices

$9(3^{k-1} \times 3^{k-1})$ mult, $9 \times 100(3^{k-1} \times 3^{k-1})$

mults: $T(n) = 9T(n/3)$

$= 9^2 T(n/9)$

$\cdots$

$= 9^k T(1) = 9^k$

$\rightarrow n = 3^k$

$k = \log_3 n$

$9^k = 9^{\log_3 n} = n^{\log_3 9} \approx n^2$

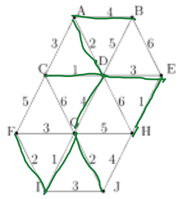
add/subs: $T(n) = 9 \cdot T(n/3) + 100(n/3)^2$

$a = 9,\ b = 3 \quad d = \log_3 9 = 2$

$c = 2 \quad \underline{c = d}$

$T(n) \in \Theta(n^c \log^{k+1} n)$

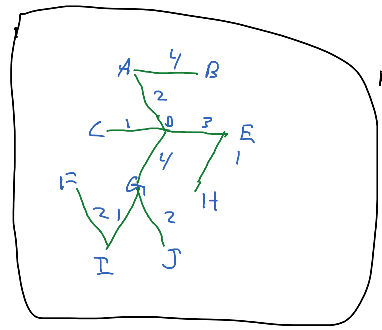
$T(n) = \Theta(n^2 \log n)$



4. Consider the graph $\mathcal{G}$ with the following weights:

Use either Prim's algorithm or Kruskal's algorithm to determine a minimum spanning tree for $\mathcal{G}$.

Show your work clearly.



minimum spanning tree

Kruskal's

— add an edge

X already visited/added

Tree Edge	Graph Edge		
CD 1	CD	1	—
EH 1	EH	1	—
GI 1	GI	1	—
AD 2	AD	2	—
GJ 2	GJ	2	—
FI 2	FI	2	—
DE 3	DE	3	—
IJ 3	IJ	3	X
FG 3	FG	3	X
AC 3	AC	3	X
AB 4	AB	4	—
HJ 4	HJ	4	X
DG 4	DG	4	—
	FC	5	X
	BE	6	X

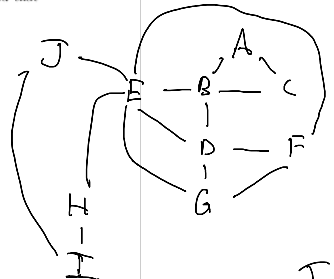
Done

5. Consider the graph, given by adjacency lists, below. (Λ represents a null pointer)

Find the Depth First Search and Breadth First Search trees for the graph, starting at vertex A. When Depth First Search is implemented as a stack, what are the contents of the stack just before and after the moment that it could be first determined that the graph is not biconnected.

0 A : B → C Λ
1 B : A → C → D → E Λ
2 C : A → B Λ
3 D : B → E → F → G Λ
4 E : B → D → F → G → H → J Λ
5 F : D → E → G Λ
6 G : D → E → F Λ
7 H : E → I Λ
8 I : H → J Λ
9 J : E → I Λ

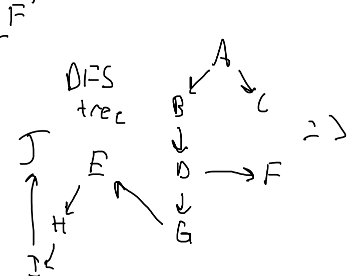
Biconnected - no articulation nodes

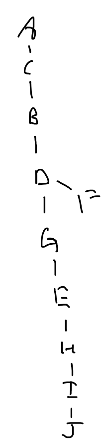


DFS stack

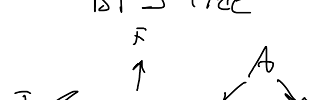
A
A B
A B C
A B
A B D
A B D F
A B D
A B D G
A B D G E
A B D G E H
A B D G E H I
A B D G E H I J

DFS tree

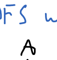




BFS Tree



DFS w/ Back values



To find not biconnected find

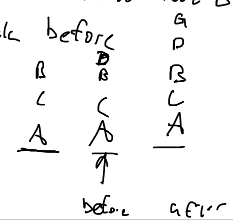
DFS w/ Back values

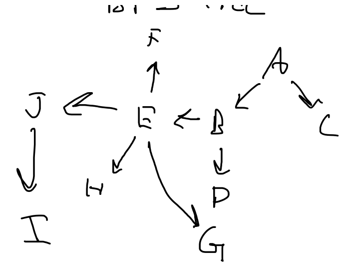
To find not biconnected, find an articulation point

If backval(c) ≥ val(p), then remove P disconnects C from the rest, so P is an articulation node.

– 1st occurrence node B

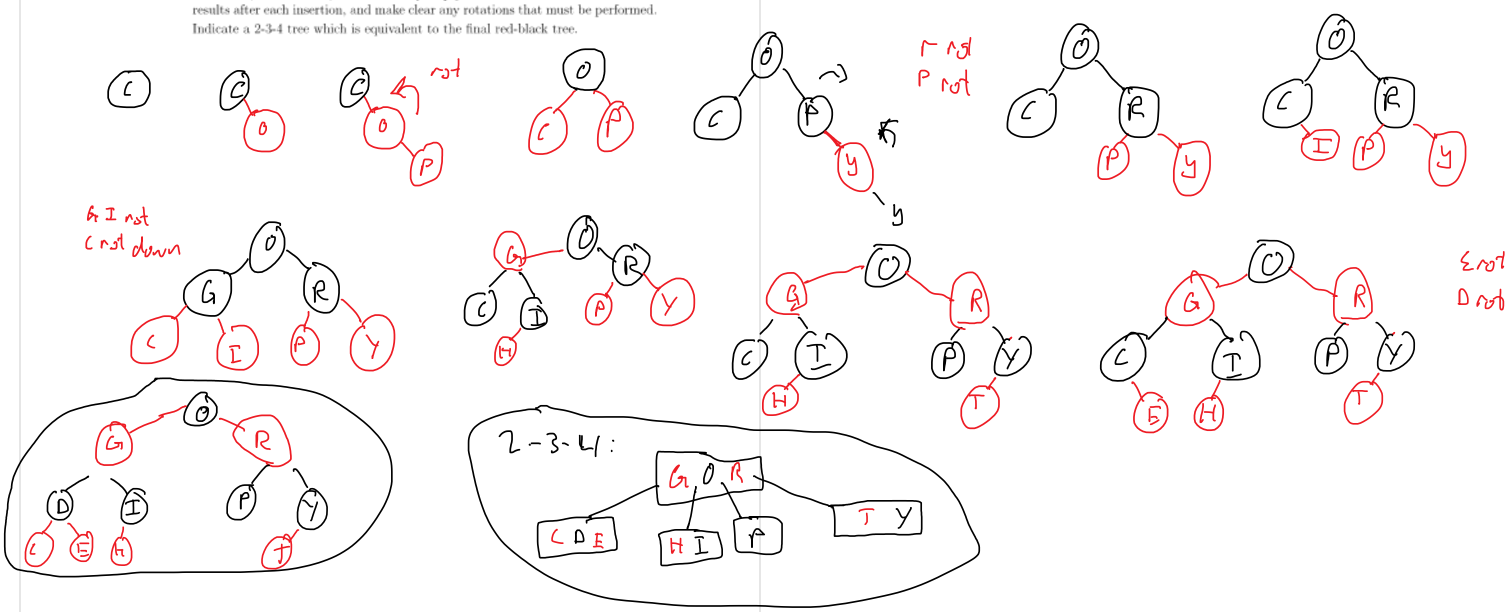
Stack before





6. Choose two of the following three problems:

(a) Show the red-black tree that results after each of the letters of C O P Y R I G H T E D is inserted, in that order, into an initially empty red-black tree. Show the tree that results after each insertion, and make clear any rotations that must be performed. Indicate a 2-3-4 tree which is equivalent to the final red-black tree.



$(AB)C$ $\quad A = a_1 \times a_2$, $B = b_1 \times b_2$, $C = c_1 \times c_2$ $\Rightarrow (a_1 \cdot b_1 \cdot b_2) + (a_2 \cdot c_1 \cdot c_2)$

$A(BC) \Rightarrow (b_1 \cdot c_1 \cdot c_2) + (a_1 \cdot b_1 \cdot c_2)$

(b) In class, we used dynamic programming to find the most efficient parenthesization to multiply $ABCDEF$ where the sequence of dimensions was 6, 4, 5, 8, 2, 7, 3. The initial calculations are at `http://math.scu.edu/~bwalden/chain1.html`. Modify the method we used to find the least efficient parenthesization. You may program this, or just do it by hand.

↳ $A = a \times b$, $B = b \times c$, $C = c \times d$

$(AB)C = (a \cdot b \cdot c) + (b \cdot c \cdot d)$

$A(BC) = (b \cdot c \cdot d) + (a \cdot b \cdot d)$

A B C D E F

6 4 5 8 2 7 3

A	B	C	D	E	F
6x4	4x5	5x8	8x2	2x7	7x3

(AB)C 360

A(BC) 352

. . .

Max([illegible]) = 9 3 4

```
#!/usr/bin/python
#python 2.7
"""
    @author Matthew Koken <mkoken@scu.edu>
    @file final_6b.py
    Calculates the least efficient paring matrix multiplications given a set of dimensions
"""
def chain_mult (arr):
    length = len(arr)
```

```
        @author Matthew Koken <mkoken@scu.edu>
        @file final_6b.py
        Calculates the least efficient paring matrix multiplications given a set of dimensions
"""
def chain_mult (arr):
    length = len(arr)
    max_mults = [[0 for x in range(length)] for x in range(length)]
            #set to 0 for multiplying one matrix
    for i in range(1, length):
        max_mults[i][i] = 0

    for idx in range(2, length):
        for i in range(1, length-idx+1):
            j = i+idx-1
            max_mults[i][j] = 0
            for k in range(i, j):

                #calculate the cost - num scalar multiplications
                cost = max_mults[i][k] + max_mults[k+1][j] + arr[i-1]*arr[k]*arr[j]
                if cost > max_mults[i][j]: #update if worse, because we want worse
                    max_mults[i][j] = cost

    return max_mults[1][length-1]


dimensions = [6, 4, 5, 8, 2, 7, 3]
mults = chain_mult(dimensions)
print "Mults: " + str(mults)
```

A(BC) 352

Max(ABCDEF) = 934

c) Consider the straight-line program for Horner's algorithm we developed in class:

$$
\begin{aligned}
s_1 &= a[n] * z \\
s_2 &= s_1 + a[n-1] \\
s_3 &= s_2 * z \\
s_4 &= s_3 + a[n-2] \\
&\vdots \\
s_{2n-1} &= s_{2n-2} * z \\
s_{2n} &= s_{2n-1} + a[0]
\end{aligned}
$$

If we add the following lines to the program,

$$
\begin{aligned}
s_{2n+1} &= s_1 + s_2 \\
s_{2n+2} &= s_{2n+1} * z \\
s_{2n+3} &= s_{2n+2} + s_4 \\
&\vdots \\
s_{2n+2j} &= s_{2n+2j-1} * z \\
s_{2n+2j+1} &= s_{2n+2j} + s_{2j+2} \\
&\vdots \\
s_{4n-4} &= s_{4n-5} * z \\
s_{4n-3} &= s_{4n-4} + s_{2n-2}
\end{aligned}
$$

what is being calculated in the last line? Explain.

7. We know that a polynomial of degree at most $n-1$ can be evaluated at a single point in $O(n)$ time using Horner's algorithm. We also saw that such a polynomial may be evaluated at the nth complex roots of unity in $O(n \log n)$ time using the Fast Fourier Transform algorithm. In this problem, we will consider the problem of evaluating a polynomial of degree at most $n-1$ at n arbitrary values.

We assume that we can use an algorithm called REMAINDER, that when passed the polynomials $a(x)$ and $p(x)$, returns the remainder $r(x)$ when $a(x)$ is divided by $p(x)$. For example, if $a(x) = 3x^3 + x^2 - 3x + 1$ and $p(x) = x^2 + x + 2$, then REMAINDER$(a(x), p(x))$ returns $r(x) = -7x + 5$, since

$$\frac{3x^3 + x^2 - 3x + 1}{x^2 + x + 2} = 3x - 2 + \frac{-7x+5}{x^2+x+2},$$

or equivalently,

$$a(x) = Q(x)\,(p(x)) + r(x)$$

$$3x^3 + x^2 - 3x + 1 = (3x-2)(x^2+x+2) + (-7x+5)$$

a) Given values $x_1, x_2, \dots x_n$ for evaluation, form polynomials $P_1(x) = \Pi_{i=1}^{n/2}(x - x_i)$ and $P_2(x) = \Pi_{i=n/2+1}^{n}(x - x_i)$. If $R_2(x)$ is the polynomial remaider when $A(x)$ is divided by $P_2(x)$, explain why $A(x_i) = R_1(x_i)$ for all $i \le n/2$ while $A(x_i) = R_2(x_i)$ for all $i > n/2$.

b) Assuming that `REMAINDER` can be implemented in $O(n \log n)$ time, for polynomials of degree less than n, provide a divide-and-conquer algorithm to evaluate $A(x)$ at $x_1, \dots, x_n$ in $O(n \log^2 n)$ time.

Note: You will also need to account for computing the coefficients of $P_1(x)$ and $P_2(x)$ in your analysis.

c) Suppose you didn't know how to implement REMAINDER in $O(n \log n)$ time. (It's not an easy problem.) How efficient would the implementation have to be in order for your evaluation algorithm to be more efficient than just using Horner's method in turn on each evaluation?

$$\frac{P_1(x)}{P_2(x)} = Q(x) + \frac{R(x)}{P_2(x)}$$

$A(x) = [R_1(x)]$ concat w/ $[R_2(x)]$

$[(x-x_1)(x-x_2)\dots(x-x_{n/2})][(x-x_{\frac{n}{2}+1})\dots(x-x_n)]$

$$R_2(x) = \frac{A(x)}{P_2(x)}$$

$$\frac{P(x)}{(x-x_1)\cdots(x-x_{n/2})} = Q(x) + \frac{R_1(x)}{(x-x_1)\cdots(x-x_n)}$$

a) $\dfrac{a(x)}{p(x)}$ $\qquad p(x)\overline{)a(x)}$

$$P_1(x) = \prod_{i=1}^{n/2}(x - x_i)$$

$$P_2(x) = \prod_{i=n/2+1}^{n}(x - x_i)$$

$$\frac{P(x)}{(x-x_1)\cdots(x-x_{n/2})} = Q(x) + \frac{R_1(x)}{(x-x_1)\cdots(x-x_n)}$$

$A(x_i) = R_1(x_i)$, $i \le \frac{n}{2}$

$A(x_i) = R_2(x_i)$, $i > \frac{n}{2}$

$$\frac{A(x)}{(x-x_{n/2+1})\cdots(x-x_i)} = Q(x) + \frac{R_2(x)}{(x-x_1)\cdots(x-x_n}$$

We get these because of the roots of unity when evaluating. At the first root, there are 2 polynomials - we get the R_1 and R_2. Therefore, at the $i \le n/2 \to i = \lfloor n/2 \rfloor$
$i > n/2 \to i = \lceil n/2 \rceil$
we will get the $A(x_i) = R_1(x_i)$
$A(x_i) = R_2(x_i)$

Looking at the coefficient parity, the coefficients bit numbering follows this and gives the expected remainder groupings.
- R_1 gets 1/2 of the x, R_2 gets the other half

b) evaluate $A(x)$
$\left[P_1(x) = \prod_{i=1}^{n/2}(x-x_i)\right]\left[P_2(x) = \prod_{i=n/2+1}^{n}(x-x_i)\right] = P(x)$

$\left[\dfrac{A(x)}{P_1(x)} = R_1(x)\right]\left[\dfrac{A(x)}{P_2(x)} = R_2(x)\right]$

like synthetic division?

evaluate $A(x)$ at $x_1 \dots x_n$
↳ multiple evaluations - 1 at each x

$\dfrac{A(x)}{P(x)}$ $\qquad A_n(x) = a_0x^n + a_1x^{n-1} \dots + a_n$
$\Rightarrow a_0x^n + a_1x^{n-1} + \dots + a_n = (x-a)(b_0x^{n-1} + b_1x^{n-2} + \dots + b_{n-1})$ + remainder

evaluation should be $O(n \log n) \Rightarrow O(n \log^2 n)$ w/ REMAINDER evaluation should be $O(n \log n) \Rightarrow O(n \log^2 n)$ w/REMAINDER

$\Rightarrow$ algorithm:

$n \log^2 n$

$a_0 = b_0$
$a_1 = -ab_0 + b_1$
$a_2 = -ab_0 + b_1$
$\dots$
$a_{n-1} = -ab_{n-2} + a_{n-1}$
$a_n = -ab_{n-1} + r$

for each $P_n(x)$ - each can be evaluated independently, so in parallel

$Q(x) (x-x_1) \cdots (x-\,) + R$

$r = a_n + ab_{n-1}$

C) Remainder in $n \log n$?

Horner's is already optimal - n add, $\frac{(n^2+n)}{2}$ mult

↳ n add, $2n-1$ mults

The implementation needs to beat
$\lfloor n/2 \rfloor + 2$ multiplications and $n-1$ additions

↳ need to add remainder

algorithm w/REMAINDER needs to be $< O(n \log^2 n)$
if REMAINDER is inefficient, say $O(n \log^2 n)$
Then the algorithm has to be better than that.

division ~ multiplication for complexity

The implementation will need to be more or as efficient as Horner's - so