Intro to Algorithms - 1

Wednesday, January 10, 2024 10:53 AM

Date: January 10

Statement:

Is this algorithm,

Make change of C cents using fewest coins,

Quarters = C / 25

R = C − 25 * Quarters

Dimes = R / 10
R = R - 10 * Dimes
Nickels = R / 5
Pennies = R − 5 * Nickels

still correct, i.e., are fewest coins used if same algorithm without nickels?

Make change of C cents using fewest coins,

Quarters = C / 25

R = C − 25 * Quarters

Dimes = R / 10

Pennies = R − 10 * Dimes

My work:

No, the algorithm is not optimized to recieve the fewest coins.

Corrected Sol'n:

Solution is correct, but here's another example.

C = 30

Quarters = 30 / 25

R = 30 − 25 * 1

Dimes = 5 / 10

Pennies = 5 − 10 * 0

∴30¢ is 1 quarter & 5 pennies when it

should be 3 dimes with no nickels (or a quarter & a nickel with nickels).

Date: January 10

Statement:

Can we obtain the upper bounds on p_{10}, p_5, p_1 (the number of dimes, nickels, & pennies in the optimal soln, respectively)?

My work:

Dime(10) ≡ 2 * Nickel(5)
Nickel(5) ≡ 5 * Penny(1)

These values create bounds for each other.

Corrected soln:

Theorem 1: $p_{10} \leq 2$, $p_5 \leq 1$, $p_1 \leq 4$ b/c
3 dimes could be replaced w/a quarter & nickel,
2 nickels could be replaced w/a dime,
& 5 pennies could be replaced w/a nickel.

Theorem 2: $p_5 = 1$, $p_{10} \leq 1$ b/c
if we have a nickel.

if we have a nickel)

then 2 dimes could be replaced w/a quarter & our nickel.

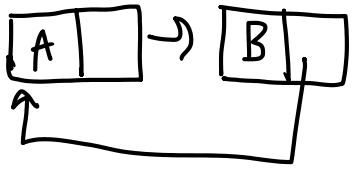
Date: January 10

Statement:

Given an efficient algorithm that uses a minimum of extra space for testing whether a linked list is "broken", i.e., has a next pointer that points to previous node in the linked list, thereby forming a cycle.

Head Not broken

Head Broken



My work:

I propose using a set of pointers representing the next ptrs. Cycling through the list & finding any duplicate occurrence would terminate the cycle until a NULL address was found at the end of the list. This should run in $\Omega(n)$ time complexity

list. This should run in O(n) time complexity w/ O(n) space complexity.

corrected soln:

Hare & Tortoise soln)

```
Hare = Head
Tortoise = Head
Broken = false
while (Hare != NULL && Broken == false)
{
    Hare = Hare -> next
    if ( Hare == Tortoise)
    {
        Broken = true
    }
    if (Hare != NULL)
    {
        Hare = Hare -> next
        if ( Hare == Tortoise)
```

```
        {
            Broken = true
        }
    }
    Tortoise = Tortoise->next
}
```

Hare goes twice as fast through the list compared to Tortoise. Hare will encounter Tortoise in cycle if broken, otherwise Hare will reach end of list.

Date: January 12

Statement:

Number of digits.
Show that the number n of decimal digits of p is approximately $\log_{10}(p)$.
What about number of binary digits? Octal digits? Hexadecimal digits?

My work:

1 to 9 digit of p if single digit. for example, 0.1, 0.2, 0.3, 0.4, ..., 0.9.

$n \approx \log_{10}(p)$

$p = 10^n$

Above example is $10^1 = 10$ number of decimal digits of p.

$p = 2^n$ for binary digits.

$p = 8^n$ for octal digit.

$p = 16^n$ for hexadecimal digits.

corrected soln:

3-digit number has $100 \le p \le 999$ digits in decimal.

$100 \le p \le 999$

$b^2 \le p \le b^3 - 1$

$10^{n-1} \le p \le 10^n - 1$ (Generalize)

$10^{n-1} \le p$

$n - 1 \le \log_{10}(p)$

$n \le 1 + \log_{10}(p)$

$p \le 10^n - 1$

$p + 1 \le 10^n$

$\log_{10}(p+1) \le n$

$\log_{10}(p+1) \le n \le 1 + \log_{10}(p)$

$n \approx \log_{10}(p)$

Binary: $n \approx \log_2(p)$

octal: $n \approx \log_8(p)$

Hex: $n \approx \log_{16}(p)$

Date: January 12

Statement:
Changing to binary
Given pseudocode for a recursive function BinRep(n) for converting a number n to its binary (base 2) representation stored in a string. Assume + performs the operation of adding a digit, i.e.,
"1011" + 0 → "10110"

My work:

Hint that the least significant digit of n mod 2 in binary & decimal is $\lfloor \frac{n}{2} \rfloor$.

I propose recursing the decimal value & concatenating the binary least significant digit.

Corrected soln:

```
BinRep(n)
{
    if (n == 0)
    {
        ...
```

```
    {
        return "1"
    }
    else
    {
        return BinRec(n / 2) + n mod 2
    }
}
```

Date: January 12

Statement:

Computing cryptographic key K.

How about computing key K as follows,

$K = (b^m \bmod p)(b^n \bmod p) = b^{m+n} \bmod p$?

Alice knows: $b, ?, n, b^n \bmod ?, b^m \bmod p$

Bob knows: $b, ?, m, b^n \bmod ?, b^m \bmod p$

Eve knows: $b, ?, b^n \bmod ?, b^m \bmod p$

will Eve know k?

My work:

I'm not sure how secure the key is.

Corrected Sol'n:

This is the famous "Discrete Logarithm Problem" that some of the greatest minds in mathematics & computer science have (unsuccessfully) tried to crack throughout the years.

GCD - 1

Sunday, January 14, 2024 7:53 PM

Date: January 17

Statement:

Using Euclid's algorithm, compute $\gcd(585, 1035)$.
Requires recursion relation $\gcd(a, b) = \gcd(b, r)$, where $r = a \bmod b$. with initial condition $\gcd(a, 0) = a$.

My work:

$\gcd(a, b) = \gcd(585, 1035)$, $r = a \bmod b = 585 \bmod 1035 = 585$

$\gcd(b, r) = \gcd(1035, 585)$, $r = a \bmod b = 1035 \bmod 585 = 450$

$\gcd(b, r) = \gcd(585, 450)$, $r = a \bmod b = 585 \bmod 450 = 135$

$\gcd(b, r) = \gcd(450, 135)$, $r = a \bmod b = 450 \bmod 135 = 45$

$\gcd(b, r) = \gcd(135, 45)$, $r = a \bmod b = 135 \bmod 45 = 0$

since $b = 0$, $a = 45$, then $\gcd(585, 1035) = 45$

Corrected sol'n:

No need, my work is correct.

Date: January 19

Statement:

Write the algorithm (Recursive Binary search) from below that's in pseudocode, in C++:

function BinarySearch(L[0:n-1], low, high, X)

Input: L[0:n-1] (an array of n list elements, sorted in increasing order)

low, high (nonnegative integers)

X (a search item)

Output: returns the index of an occurrence of X in the sublist L[0:n-1, low, high] or -1 if X is not in the list.

```
if high < low then return(-1) endif // empty list
mid ← ⌊(low + high) / 2⌋
if X = L[mid] then return(mid) endif
if X < L[mid] then
    BinarySearch(L[0:n-1], low, mid - 1, X)
else
    BinarySearch(L[0:n-1], mid + 1, high, X)
endif
```

end BinarySearch

My work:

```
/* Inputs:
     ItemType  list[MAX] (an array of n list elements,
                          sorted in increasing order)
     ItemType  searchItem (a search item)
     int low, int high    (nonnegative integers)
   Output: returns the index of an occurrence of SearchItem
   in the sublist L or -1 if SearchItem is not in the list
*/

int BinarySearch(ItemType list[MAX], ItemType searchItem, int low, int high)
{
    // initial condition, list empty
    if (low > high) return -1;

    // compute midpoint
    int mid = (low + high) / 2;
    // if the midpoint is the search element, return mid
    if (searchItem == list[mid]) return mid;

    // if the search element is in front half of list, partition
    else if (searchItem < list[mid]) return BinarySearch(list, searchItem, low, mid - 1)
    // if the search element is in back half of list, partition
    else if (searchItem > list[mid]) return BinarySearch(list, searchItem, mid + 1, high)
}
```

Corrected Soln:

Corrected Soln:
No need, my work is correct.

Date: January 19

Statement:

Give C++ code for a recursive version of binary search where no equality check with midpoint is made & analyze the algorithm.

My work:

```
/* Inputs:
     ItemType  list[MAX]  (an array of n list elements,
                           sorted in increasing order)
     ItemType  searchItem  (a search item)
     int low, int high     (nonnegative integers)
   Output: returns the index of an occurrence of SearchItem
   in the sublist L or -1 if SearchItem is not in the list
*/

int BinarySearch(ItemType list[MAX], ItemType searchItem, int low, int high)
{
    // initial condition, list empty
    if (low > high) return -1;

    // if the list has one element
    if (low == high)
    {
        if (list[low] == searchElement) return low;
        else return -1;
```

```
}

    // compute midpoint
    int mid = (low + high) / 2;
    // if the search element is in front half of list, partition
    if (searchItem < list[mid]) return BinarySearch(list, searchItem, low, mid - 1)
    // if the search element is in back half of list, partition
    else if (searchItem > list[mid]) return BinarySearch(list, searchItem, mid + 1, high)
}
```

I'm not sure how to analyze this algorithm.

Corrected soln:

complexity analysis, Note: A(n) is average
B(n) is best
W(n) is worst

Assume $n = 2^k$.

Then, $B(n) = W(n) = k = \lg(n)$.

for any probability distribution on the sample space of inputs of size n

$B(n) \leq A(n) \leq W(n)$.

It follows that, $A(n) = \lg(n)$.

The worst-case complexity is equal to twice the longest string of midpoints ever generated by the algorithm for input X.

generated by the algorithm for input X.

In particular, if we assume $n = 2^k - 1$, $k \in \mathbb{N}$, then such a string is generated by searching for $x = L[0]$. We then compare X successively to the midpoints $2^{k-1}-1, 2^{k-2}-1, \ldots, 0$, so that this longest string has length k. To express k in terms of n, we note that $n+1 = 2^k$, so that we have $k = \lg(n+1)$. Since comparisons are involved at each stage,

$W(n) = 2\lg(n+1) \approx 2\lg(n)$.

The best-case complexity occurs when X is found in the midpoint position $\lfloor (n-1)/2 \rfloor$ of $L[0:n-1]$ & involves a single operation.

So, $B(n) = 1$.

Date: January 22

Statement:

obtain a general formula for i in terms of X, low, high, L[low], L[high] on the line contained in the three points:

(low, L[low]); (i, X); (high, L[high])

My work:

I know we can compute the slope $m = \frac{\Delta y}{\Delta x}$ for two points, I'm not sure what we would do once we obtain multiple slopes.

corrected soln:

slope $\Delta y / \Delta x$ of two points (low, L[low]) & (high, L[high]):

$\Delta y / \Delta x = (L[high] - L[low]) / (high - low)$.

slope $\Delta y / \Delta x$ of two points (low, L[low]) & (i, X):

(i, X):

$\Delta y / \Delta x = (X - L[low]) / (i - low)$.

Equating these two slopes above, we get:

$(L[high] - L[low]) / (high - low) = (X - L[low]) / (i - low)$

Solving for i, we obtain:

$i = low + (X - L[low])(high - low) / (L[high] - L[low])$

Date: January 22

Statement:

Write pseudocode for the recursive version of Interpolation search.

My work:

I'm not sure how to implement the specifics of interpolation search.

Corrected Sol'n:

function InterpolationSearch(L[0:n-1], low, high, X)

Input: L[0:n-1] (an array of n list elements, sorted in increasing order)

low, high (nonnegative integers)

X (a search item)

Output: returns the index of an occurrence of X in the sublist L[0:n-1, low, high] or -1 if X is not in the list.

if high < low then return(-1) endif // empty list

if X < L[low] or X > L[high] return(-1) endif

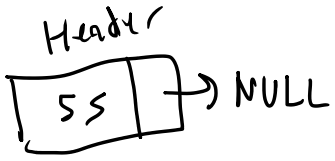
```
mid ← ⌊ low + (X - L[low])(high - low)/(L[high] - L[low]) ⌋
if X = L[mid] then return(mid) endif
if X < L[mid] then
        InterpolationSearch(L[0:n-1], low, mid - 1, X)
else
        InterpolationSearch(L[0:n-1], mid + 1, high, X)
endif
end InterpolationSearch
```

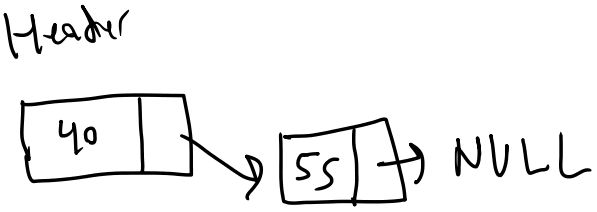
Date: January 24

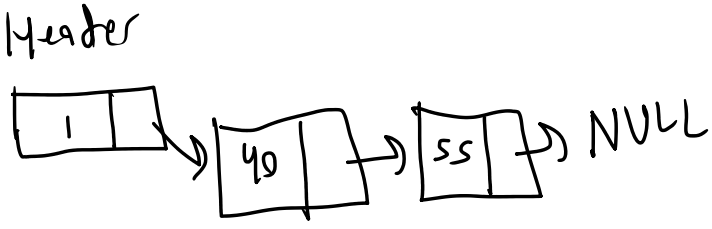
Statement:

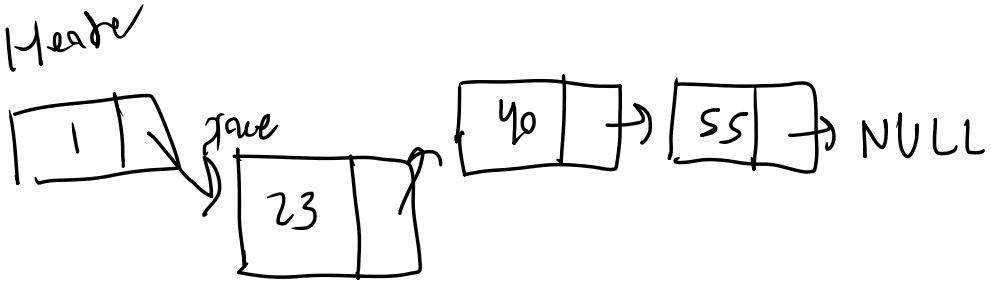
Show pictorially the action of insertion sort implemented using a linked list for the same sample list: 55, 40, 1, 23, 61, 8.

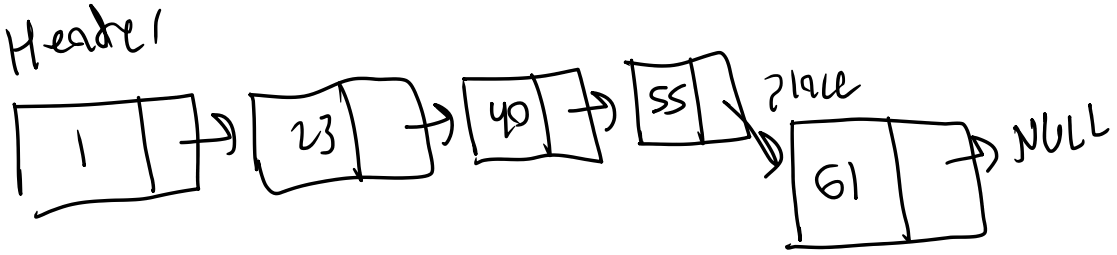
My work:

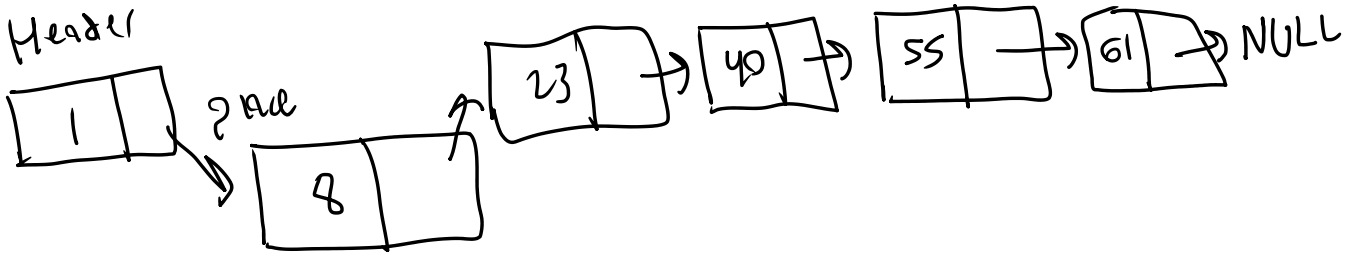












Corrected Soln:

work above is correct.

Asymptotic Behavior of Functions - 1

Saturday, January 20, 2024 2:14 PM

Date: January 25

Statement:

For the example, $f(n) = 100n + 50$ & $g(n) = n^2 - n$, show that $f(n) \in O(g(n))$.

My work:

We know $\exists$ positive constants c, n_0 such that $\forall n \geq n_0$, $f(n) = c \cdot g(n)$.

I'm not sure how we apply this formula.

Corrected sol'n:

We're proving $100n + 50 \in O(n^2 - n)$.

One way we can show this is choosing $c = 1$, $n_0 = 102$. Then,

$100n + 50 \leq 1 \cdot (n^2 - n)$ $\forall$ $n \geq 102$, $n \geq 101.49$ (derived from $-n^2 + 101n + 50 \leq 0$)

Another choice is $c = 1000$, $n_0 = 1$. Then,

$100n + 50 \leq 1000 \cdot (n^2 - n)$ $\forall n \geq 1$. (derived from

$100n + 50 \leq 1000 \cdot (n^2 - n) \; \forall \; n \geq 1.$ (derived from $-20n^2 + 22n + 1 \leq 0$)

$n \geq 1.14$

Asymptotic Behavior of Functions - 2

Saturday, January 20, 2024 3:24 PM

Date: January 25

statement:

$n^2 - n \in O(100n + 50)$.

why not just choose a really, really huge constant c? Show that,

$n^2 - n \leq 100000000000000000(100n + 50) \; \forall n \geq 1$

is not true.

My work:

$n^2 - n \leq 100000000000000000(100n + 50) \; \forall n \geq 1$

$n - 1 \leq 100000000000000000(100 + \frac{50}{n}) \; \forall n \geq 1$

As $n \to \infty$:

$n - 1 \to \infty$, but,

$100000000000000000(100 + \frac{50}{n}) \to c \cdot 100$,

where $c = 100000000000000000$.

so,

$n - 1 \nleq 100000000000000000(100 + \frac{50}{n}) \; \forall n \geq 1$

Corrected sol'n:

The same argument applies for any

The same argument applies for any positive constants c, n_0.

$\therefore$ it follows that,

$n^2 - n \notin O(100n + 50)$.

Asymptotic Behavior of Functions - 3

Saturday, January 20, 2024 3:42 PM

Date: January 25

Statement:

Show that $\log_a(n)$ & $\log_b(n)$ have the same order, independent of the choices of bases a & b.

My work:

$$\frac{\log_b(n)}{1} = \frac{\log_a(n)}{\log_a(b)} \quad \text{(log change of base formula)}$$

$\Rightarrow \log_a(n) = \log_a(b) \cdot \log_b(n)$ (cross multiply)

$\Rightarrow \log_a(n) \in \Theta(\log_b(n))$ ($\log_a(b)$ is constant, drop it)

$\Rightarrow \log_a(n)$ & $\log_b(n)$ have the same order.

constant multiple still has the same growth rate

corrected sol'n:
Adding details.

∴ we can drop the base when talking about the order of logarithmic functions, i.e. $\Theta(\log(n))$.

Date: January 26

Statement:

Using the Ratio Limit Theorem, show that worst-case complexity of Binary Search has smaller order than the worst-case complexity of Linear Search, i.e., $O(\lg(n)) \subset O(n)$.

Reminder: "$A \subset B$" means "A is a proper subset of B".

Comparing orders with the Ratio Limit Theorem

Let $f(n), g(n) \in F$. If the ratio $\frac{f(n)}{g(n)}$ exists as $n \to \infty$, then f & g are comparable. Moreover, assuming $L = \lim_{n\to\infty} \frac{f(n)}{g(n)}$ exists, then the following results hold true:

1. $0 < L < \infty \Rightarrow f(n) \in \Theta(g(n))$ (f & g have the same order)
2. $L = 0 \Rightarrow O(f(n)) \subset O(g(n))$ (f has a smaller order than g)
3. $L = \infty \Rightarrow O(g(n)) \subset O(f(n))$ (f has a larger order than g)

L'Hôpital's Rule

Suppose $\lim_{n\to\infty} f(n)$ & $\lim_{n\to\infty} g(n)$ are both 0 or both ∞. Then,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}.$$

My work:

Let $f(n) = \lg(n)$, $g(n) = n$.

$L = \lim_{n\to\infty} \frac{f(n)}{g(n)}$ (Ratio Limit Theorem)

$= \lim_{n\to\infty} \frac{\lg(n)}{n}$ (substitute $f(n)$ & $g(n)$)

I'm not sure where to go from here.

corrected soln:

$= \lim_{n\to\infty} \frac{\lg(e)\cdot\ln(n)}{n}$ (from previous problem, $\log_a(n) = \log_a(b)\cdot\log_b(n)$, so $\log_2(n) = \log_2(e)\cdot\log_e(n)$)

$\overset{L}{=} \lim_{n\to\infty} \frac{\lg(e)\cdot\frac{1}{n}}{1}$ (apply L'Hôpital's Rule, $(\lg(e)\cdot\ln(n))' = (\lg(e))'\cdot\ln(n) + \lg(e)\cdot(\ln(n))' = 0 + \lg(e)\cdot\frac{1}{n}$, $(n)' = 1$)

$= \lim_{n\to\infty} \frac{\lg(e)}{n}$ (simplify)

$= 0$ (apply limit)

∴ by the Ratio Limit Theorem, it follows that $O(\lg(n)) \subset O(n)$.

Important Asymptotic Formulas - 1

Saturday, January 20, 2024 5:50 PM

Date: January 29

Statement:

Strongly asymptotic result for polynomials.

Proposition: Let $p(n)$ be the k^{th} degree polynomial, $a_k n^k + \ldots + a_1 n + a_0$, where $a_k > 0$. Then,

$$p(n) \sim a_k n^k.$$

Corollary: $p(n) \in \Theta(n^k)$.

Prove the proposition.

Strong Asymptotic Behavior

We say $f(n)$ is strongly asymptotic to $g(n)$, denoted $f(n) \sim g(n)$, if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

By the Ratio Limit Theorem:

$$f(n) \sim g(n) \Rightarrow f(n) \in \Theta(g(n)).$$

My work:

Let $f(n) = P(n) = a_k n^k + \ldots + a_1 n + a_0$,
$g(n) = a_k n^k$.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \qquad \text{(Ratio Limit Theorem)}$$

$$= \lim_{n\to\infty} \frac{a_k n^k + \ldots + a_1 n + a_0}{a_k n^k}$$

$$= \lim_{n\to\infty} 1 + \frac{a_{k-1}}{a_k n} + \ldots + \frac{a_1}{a_k n^{k-1}} + \frac{a_0}{a_k n^k}$$

$= 1$

This proves the proposition.

Corrected soln:
work above is correct.

Important Asymptotic Formulas - 2

Saturday, January 20, 2024 6:39 PM

Date: January 29

Statement:

Prove $f(n) \in \Theta(g(n))$
$\Leftrightarrow f(n) \in O(g(n))$ & $f(n) \in \Omega(g(n))$

My work:

Suppose $f(n) \in \Theta(g(n))$. Then,
$\exists$ positive constants c_1, c_2, n_0 such that
$c_1 g(n) \leq f(n) \leq c_2 g(n)$, $\forall n \geq n_0$.

$f(n) \leq c_2 g(n)$, $\forall n \geq n_0 \Rightarrow f(n) \in O(g(n))$.

$f(n) \geq c_1 g(n)$, $\forall n \geq n_0 \Rightarrow f(n) \in \Omega(g(n))$.

Corrected sol'n:

Conversely, suppose $f(n) \in O(g(n))$ & $f(n) \in \Omega(g(n))$.
Then, $\exists$ positive constants c_1, c_2, n_0, n_1 such that
$f(n) \leq c_2 g(n)$, $\forall n \geq n_0$ & $f(n) \geq c_1 g(n)$, $\forall n \geq n_1$.

$\therefore\ c_1 g(n) \le f(n) \le c_2 g(n),\ \forall\ n \ge \max\{n_0, n_1\}$

$\Rightarrow f(n) \in \Theta(g(n))$

Date: January 31

Statement:

Obtain recurrence relation for W(n) & solve for the version of Binary search where no equality check is done until list has size 1. For convenience, assume $n=2^k$.

Recursive version of Binary Search where no equality check is done until list has size 1 in C++

```
/* Inputs:
     ItemType   list[MAX] (an array of n list elements,
                           sorted in increasing order)
     ItemType   searchItem (a search item)
     int low, int high     (nonnegative integers)
   Output: returns the index of an occurrence of SearchItem
   in the sublist L or -1 if SearchItem is not in the list
*/
int BinarySearch(ItemType list[MAX], ItemType searchItem, int low, int hi)
{
    // initial condition, list empty
    if (low > high) return -1;

    // if the list has one element
    if (low == high)
    {
        if (list[low] == searchElement) return low;
        else return -1;
    }
```

```
    }

    // compute midpoint
    int mid = (low + high) / 2;
    // if the search element is in front half of list, partition
    if (searchItem < list[mid]) return BinarySearch(list, searchItem, low, mid - 1)
    // if the search element is in back half of list, partition
    else if (searchItem > list[mid]) return BinarySearch(list, searchItem, mid + 1, high)
}
```

My work:

$n = 2^k \Rightarrow k = \lg(n)$.

$W(n) = W(\frac{n}{2}) + 1$ from recursive midpoint, &

$W(1) = 1$ for the initial condition where we only make one comparison & the list is empty.

Corrected Sol'n:

Applying the technique of repeated substitution, we obtain:

$$W(n) = W(\tfrac{n}{2}) + 1 = \left(W(\tfrac{n}{4}) + 1\right) + 1$$

$$= W(\tfrac{n}{4}) + 2 = \left(W(\tfrac{n}{8}) + 1\right) + 2$$

$$= W(\tfrac{n}{8}) + 3 = \left(W(\tfrac{n}{16}) + 1\right) + 3$$

$$\vdots \qquad\qquad \vdots$$

$$= W(\tfrac{n}{2^k}) + k$$

$= W\left(\frac{2^k}{2^k}\right) + k$ (apply $n = 2^k$)

$= W(1) + k$ (simplify)

$= 1 + \lg(n)$ (apply $W(1) = 1$ & $k = \lg(n)$)

$\sim \lg(n)$

Date: January 31

Statement:

Obtain recurrence relation for the Best-Case complexity B(n) of Merge Sort & solve using repeated substitution.

```
void MergeSort(DataType L[MAX], int low, int high)
{
    // bootstrap condition -- a list of size one at most
    // is a priori sorted

    if (low < high)
    {
        mid = (low + high) / 2;
        // recursively sort first half
        MergeSort(L, low, mid - 1);
        // recursively sort second half
        MergeSort(L, mid+1, high);
        // merge two halves
        Merge(L, low, mid, high);
    }
}
```

My work:

$n = 2^k \Rightarrow k = \lg(n)$.

$B(n) = B(\frac{n}{2})$ (the number of comparisons to recursively sort first half)
$+ B(\frac{n}{2})$ (the number of comparisons to recursively sort second half)
$+ \frac{n}{2}$. (fewest comparisons to merge left & right sublists that are merged evenly in half)

We have the recurrence relation:

$B(n) = 2 \cdot B(\frac{n}{2}) + \frac{n}{2}$ & $W(1) = 0$ for the initial condition where we need not compare a sorted list of size 1. Here, we approximate n with $n-1$. Note: $\frac{n}{2} \sim n$, so $B(n) = 2 \cdot B(\frac{n}{2}) + n$.

Corrected sol'n:

Applying the technique of repeated substitution, we obtain:

$$B(n) = 2 \cdot B(\tfrac{n}{2}) + n = 2 \cdot (2 \cdot B(\tfrac{n}{2^2}) + \tfrac{n}{2}) + n$$

$$= 2^2 \cdot B(\tfrac{n}{2^2}) + 2n = 2^2 \cdot (2 \cdot B(\tfrac{n}{2^3}) + \tfrac{n}{2^2}) + 2 \cdot (\tfrac{n}{2})$$

$$= 2^3 \cdot B(\tfrac{n}{2^3}) + 3 \cdot (\tfrac{n}{2})$$

$$\vdots \qquad\qquad \vdots$$

$$= 2^k \cdot B(\tfrac{n}{2^k}) + k \cdot (\tfrac{n}{2})$$

$$= 2^k \cdot B(\tfrac{2^k}{2^k}) + k \cdot (\tfrac{n}{2}) \qquad (\text{apply } n = 2^k)$$

$$= 2^k \cdot B(1) + k \cdot (\tfrac{n}{2}) \qquad (\text{simplify})$$

$$= 2^k \cdot 0 + k \cdot (\tfrac{n}{2}) \qquad (\text{apply } B(1) = 0)$$

$$= k \cdot (\tfrac{n}{2}) \qquad (\text{simplify})$$

$$= \tfrac{1}{2} \cdot n \cdot \lg(n)$$

Date: February 2

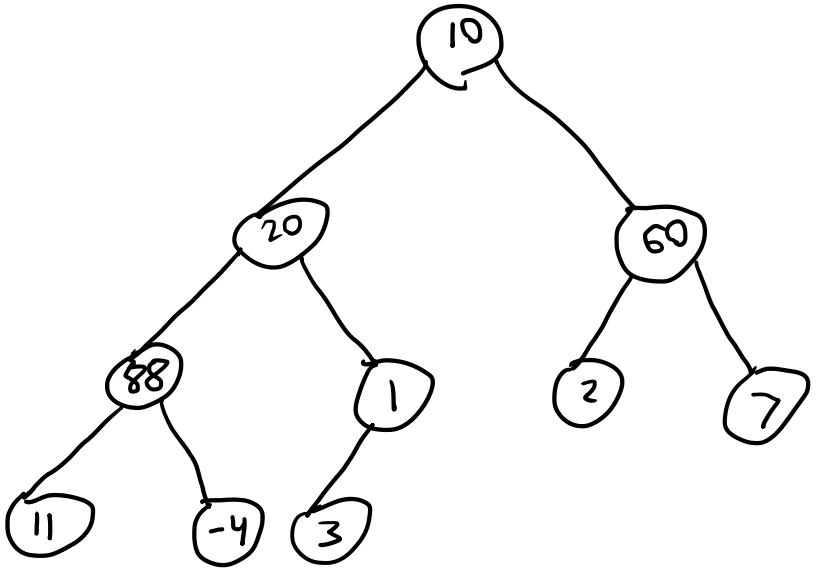
Statement:

Implementing a full & complete binary tree.

a.) Draw a complete binary tree with values on nodes implemented by the array:

0	1	2	3	4	5	6	7	8	9
10	20	60	88	1	2	7	11	-4	3

My work:



Statement:

b.) Given the index i of a node in a complete binary tree, give the formula for the index of the parent of the node at i.

My work:

parent_index = (i - 1) / 2

statement:

c.) Given the index i of a node in a complete binary tree, give the formulas for the indices of the left & right children of the node at i.

My work:

left_child = 2 * i + 1

right_child = 2 * i + 2

Corrected Soln:

work above is correct.

Date: February 2

Statement:

Give C++ code for Preorder, Inorder, & Postorder traversals, invoking function Visit() to perform operations on data in the node.

Assume the nodes of the tree are implemented using the following structure:

```
typedef int datatype;
struct Node {
    datatype item;
    Node *left;
    Node *right;
}; // END TREENODE

typedef *Node ptrNode;
```

My work:

```
void Preorder(ptrNode root)
{
    if (root != nullptr)
    {
        Visit(root->item);
        Preorder(root->left);
        Preorder(root->right);
    }
}

void Inorder(ptrNode root)
{
    if (root != nullptr)
    {
        Inorder(root->left);
        Visit(root->item);
        Inorder(root->right);
```

```
    }
}

void postorder(ptrNode root)
{
    if (root != nullptr)
    {
        postorder(root->left);
        postorder(root->right);
        visit(root->item);
    }
}
```

Corrected Soln:

Work above is correct.

Mathematical Properties of Trees with Applications to Lower Bound Theory - 1

Saturday, February 10, 2024 1:12 PM

Statement: Date: February 5

Number of edges vs. number of nodes in a tree.

Theorem. The number m of edges of any tree T is one less than the number n of nodes.

Proof by induction. We perform induction on the number n of nodes.

Basis step. A tree with one node has no edges, i.e., we have $m = 0 = n - 1$.

Induction step. Assume for $n = k$, i.e., any tree having k nodes has $k-1$ edges.

Now consider a tree T having $n = k+1$ nodes. For convenience, let $n(T)$ & $m(T)$ denote the number of nodes & edges of T.

To apply the induction hypothesis, we need to perform an operation that reduces T to a tree T' having k

an operation that [illegible]
nodes. How do we do this?

My work:

Any node at the deepest level (highest level number) must necessarily be a leaf. Thus, T contains at least one leaf node.

Remove a leaf node (& its incident edge) to obtain a tree T' having k nodes.

corrected soln:
work above is correct.

Mathematical Properties of Trees with Applications to Lower Bound Theory - 2

Sunday, January 28, 2024 1:53 PM

Date: February 5

Statement:

A 2-tree is a tree where every node that is not a leaf node has exactly two children.

An internal node is a node that is not a leaf node.

Let $I(T)$ & $L(T)$ denote the number of internal & leaf nodes of a 2-tree T, respectively. For convenience, let $I = I(T)$ & $L = L(T)$.

Proposition: Let T be a 2-tree. Then $L = I + 1$.

Clearly, the total number n of nodes satisfies $n = I + L$, so we have:

Corollary 1: $n = 2 \cdot I + 1$.

Corollary 2: $n = 2 \cdot L + 1$.

Parametrizing the induction. We must decide, which parameter we will perform induction on, i.e., the number I of internal nodes, the number L of leaf nodes, or the total number n

nodes, the number L of leaf nodes, or the total number n of nodes. We will choose L.

Basis step: $L = 1$

A single node 2-tree has one leaf node, the root, & 0 internal nodes, so we have

$L = 1 = I + 1$.

Induction step: $L = k$

Assume,
all 2-trees T with k leaf nodes have $k-1$ internal nodes.

Now consider <u>any</u> 2-tree T having $k+1$ leaf nodes.

To apply the induction hypothesis, we need to perform an operation that reduces T to a tree T' with k leaf nodes.

How do we do this?

My work:

I'm not sure how to prove the induction hypothesis.

Corrected soln:

Find a node of T, both of whose children are leaf nodes, & delete both of the node's child leaf nodes (& their incident edges). Then the resulting 2-tree, T', has one fewer leaf nodes & one fewer internal nodes than T.

Since T' has one fewer leaf nodes than T, i.e., T' has k leaf nodes, we can apply the induction hypothesis, i.e.,

$$L(T') = I(T') + 1.$$

Thus,

$$L(T) = L(T') + 1 = (I(T') + 1) + 1 = I(T) + 1$$

proving the proposition by the inductive step.

Mathematical Properties of Trees with Applications to Lower Bound Theory - 3

Sunday, January 28, 2024 3:06 PM

Date: February 5

Statement:

To verify that this construction is valid, we must prove that every 2-tree T contains a node, both of whose children are leaf nodes. Prove this.

Hint: Consider a node X at the deepest (i.e., highest) level. Let P be its parent & let Y be its sibling (i.e. the other child of P). Use proof by contradiction to show that Y is also a leaf node.

My work:

Assume that Y is not a leaf node, i.e., Y has a child C. Now C is at a deeper level than Y. Since X is at the same level as Y, it follows that X is not at the deepest level, contradicting our assumption about X. Thus, both X & Y are leaf nodes.

Corrected soln:
work above is correct.

Date: February 7

Statement:

Searching a BST.

Write a recursive function SearchBST in C++ for searching a binary tree for a search key. If found, it returns a pointer to a node where found; otherwise, it returns nullptr.

Assume the nodes of the tree are implemented using the structure:

```
typedef int KeyType;
struct Node {
   KeyType Key;
   Node *Left;
   Node *Right;
}; // END TREENODE

typedef *Node ptrNode;
```

My work:

```
ptrNode SearchBST(ptrNode Root, KeyType SearchKey)
{
```

```
// returns nullptr if SearchKey not found
if (Root == nullptr)
{
    return nullptr;
}
// returns pointer to node that contains search key
else if (SearchKey == Root->Key)
{
    return Root;
}
else if (SearchKey < Root->Key)
{
    return SearchBST(Root->Left, SearchKey);
}
else if (SearchKey > Root->Key)
{
    return SearchBST(Root->Right, SearchKey);
}
} // end SearchBST
```

Corrected sol'n:

No need, my work is correct.

Binary Search Trees (BSTs) - 2

Saturday, February 10, 2024 4:07 PM

Date: February 7

Statement:

Write a recursive function InsertBST in C++ for inserting into a BST.

My work:

```
void InsertBST(ptrNode Root, KeyType SearchKey)
{
    // create a new node for NewKey
    ptrNode NewPtr = new Node;
    NewPtr->Key = NewKey;
    NewPtr->Left = nullptr;
    NewPtr->Right = nullptr;

    if (Root == nullptr)
    {
        Root = NewPtr;
    }
    else if (NewKey < Root->Key)
```

```
{
    if ( Root -> Left == nullptr)
    {
        Root -> Left = NewPtr;
    }
    else
    {
        InsertBST(Root->Left, NewKey);
    }
}
else if (NewKey > Root->Key)
{
    if ( Root -> Right == nullptr)
    {
        Root -> Right= NewPtr;
    }
    else
    {
        InsertBST(Root->Right, NewKey);
```

}

}

} // end InsertBST

Corrected sol'n:

No need, my work is correct.

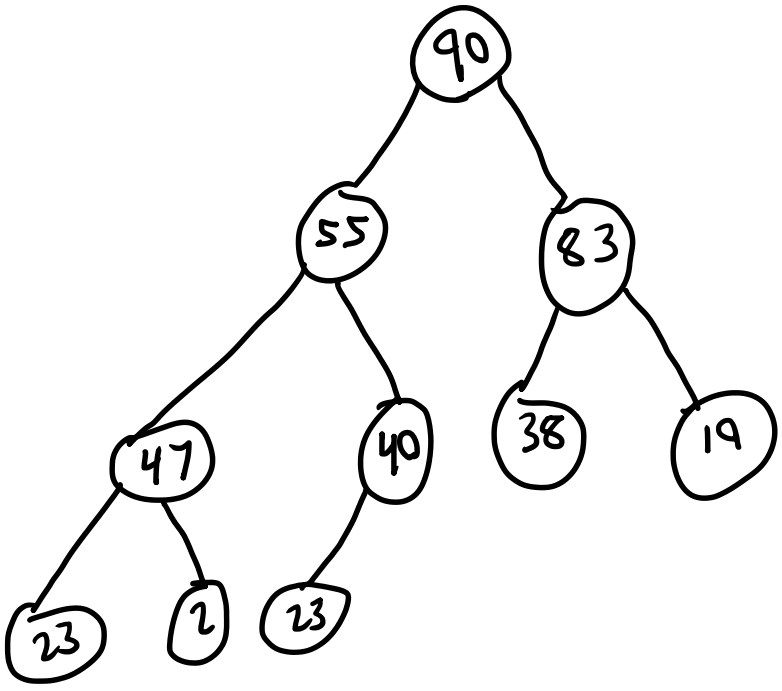
Priority Queues, Heaps, Heapsort - 1

Saturday, January 27, 2024 9:22 PM

Date: February 9

Statement:

Show the action of an array for inserting 70 into the max-heap:

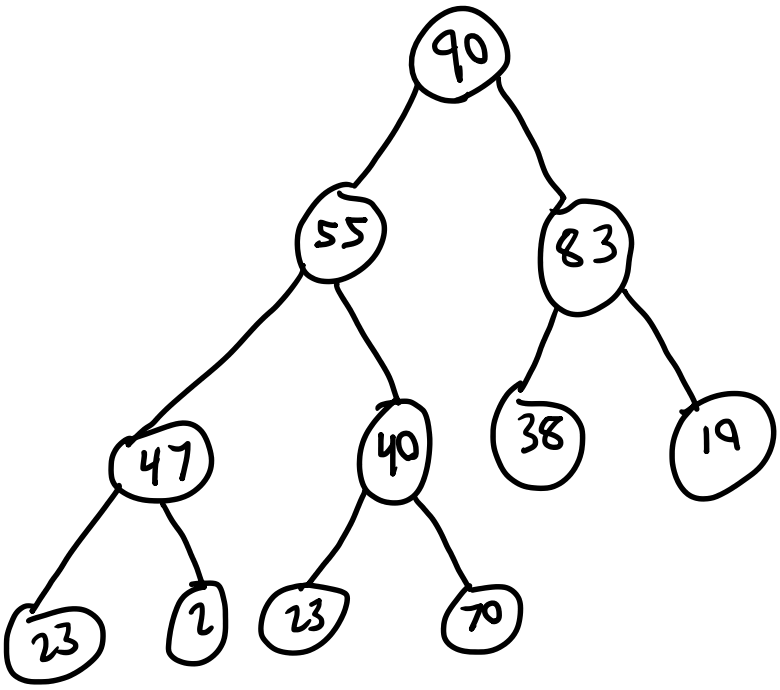


[90, 55, 83, 47, 40, 38, 19, 23, 2, 23]

My work:

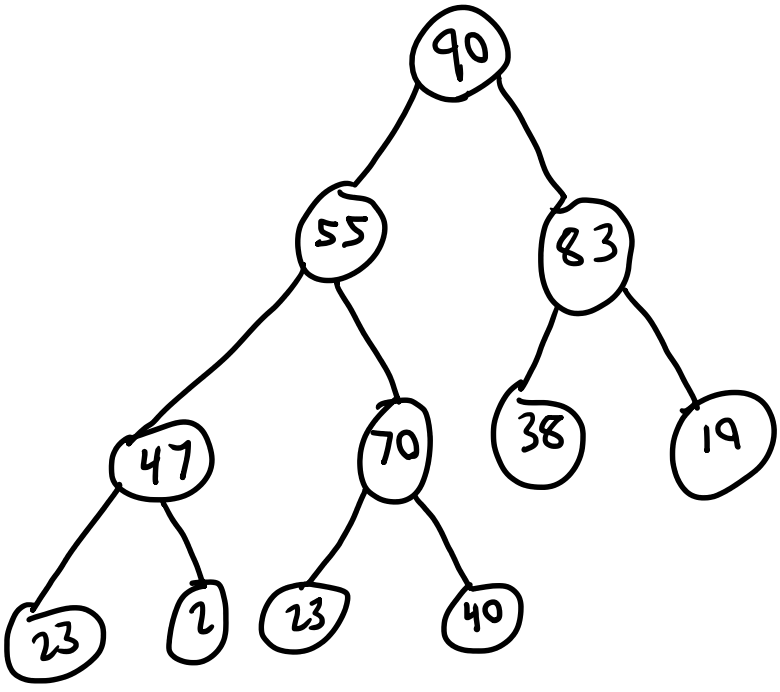
1.)

1.)



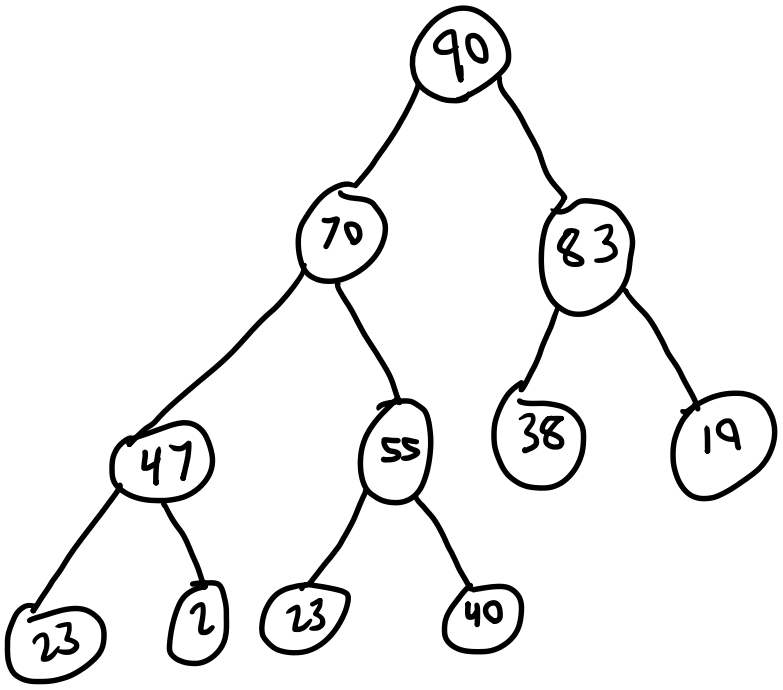
[90, 55, 83, 47, 40, 38, 19, 23, 2, 23, 70]

2.)



[90, 55, 83, 47, 70, 38, 19, 23, 2, 23, 40]

3.)



[90, 70, 83, 47, 55, 38, 19, 23, 2, 23, 40]

Corrected sol'n:

No need, my work is correct.

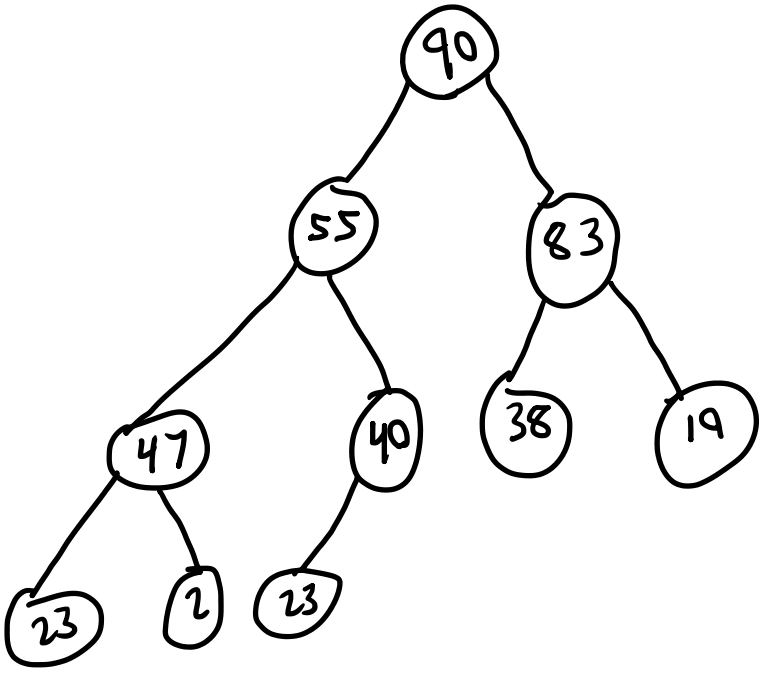
Priority Queues, Heaps, Heapsort - 2

Saturday, January 27, 2024 9:40 PM

Date: February 9

statement:

Show the action of an array for deleting of an element (e.g. the root) from the max-heap:

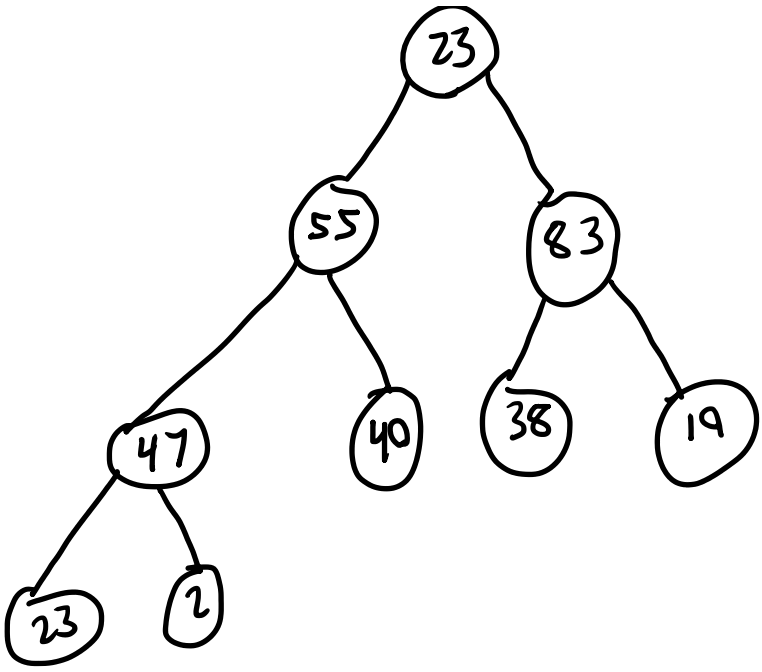


[90, 55, 83, 47, 40, 38, 19, 23, 2, 23]

My work:

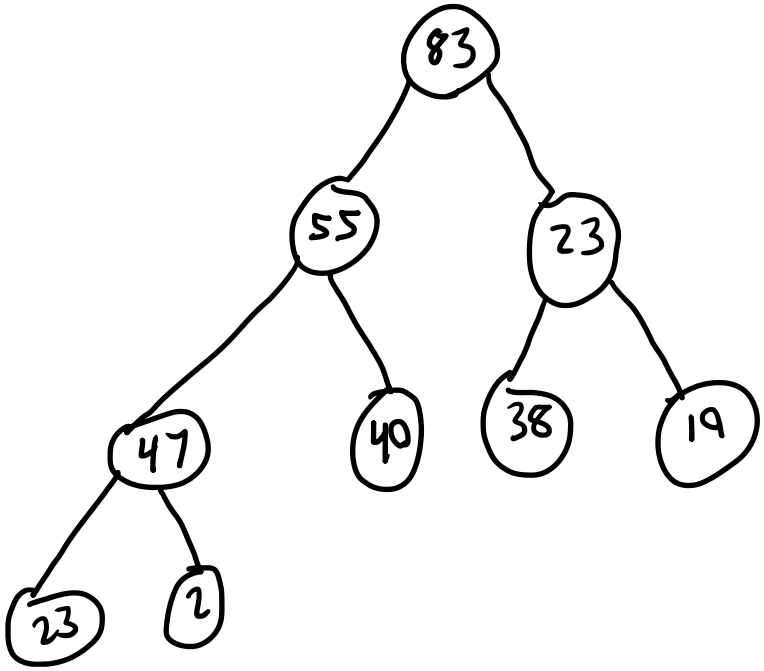
1.)

1.)



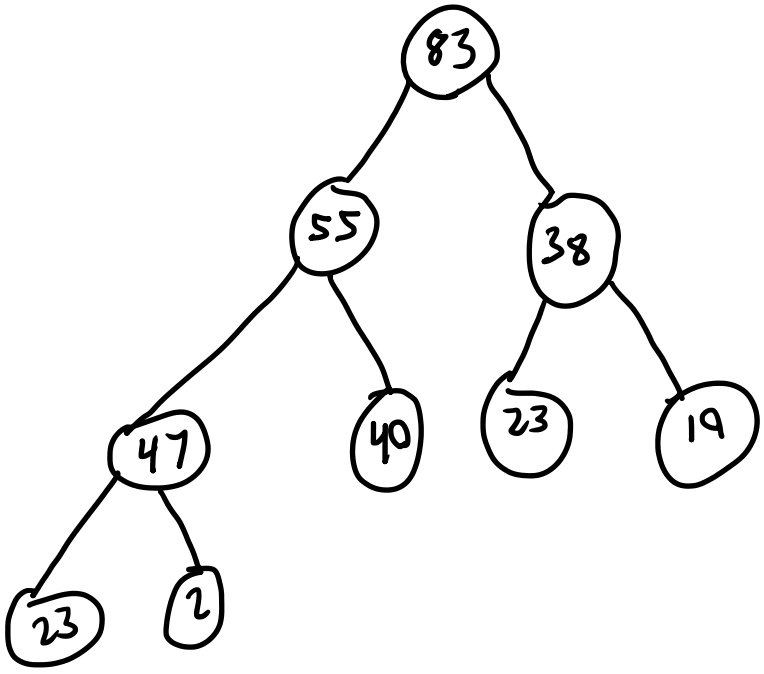
[23, 55, 83, 47, 40, 38, 19, 23, 2]

2.)



[83, 55, 23, 47, 40, 38, 19, 23, 2]

3.)



[83, 55, 38, 47, 40, 23, 19, 23, 2]

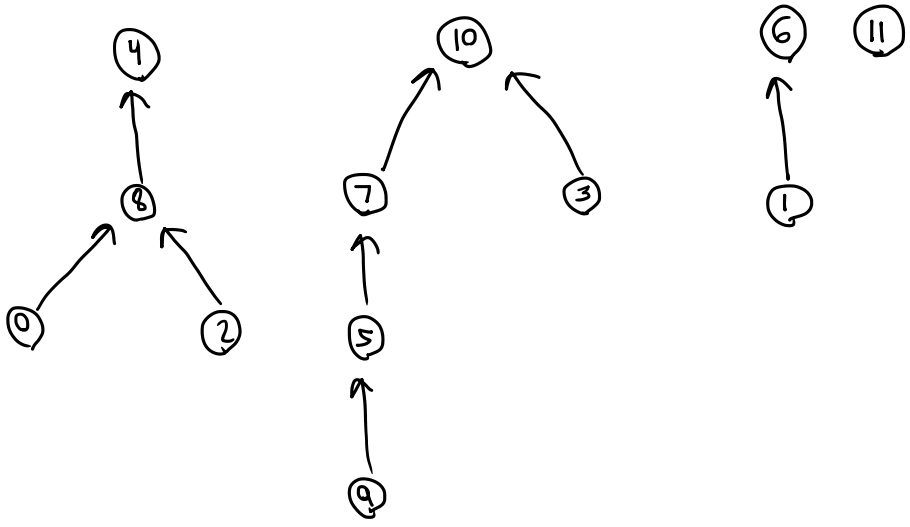
Corrected sol'n:

No need, my work is correct.

Date: February 13

For the disjoint sets represented as a forest (a collection of tree data structures),

a.) Show the action of Find(9).
b.) Show the action of Find(2).



My work:

a.) Parent(9) = 5, Parent(5) = 7, Parent(7) = 10, Parent(10) = -1.
Find(9) = 10.

b.) Parent(2) = 8, Parent(8) = 4, Parent(4) = -1.
Find(2) = 4.

Corrected soln:

No need, my work is correct.

Date: February 14

Statement:

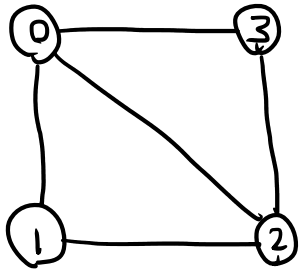
Which is more general, a graph or digraph?

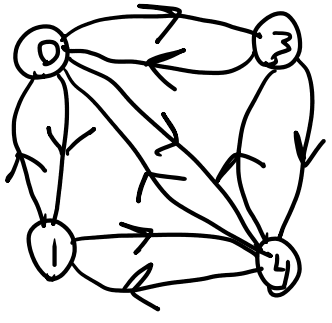
My work:

I believe a digraph is because it holds more information.

Corrected Sol'n:

A graph $G=(V, E)$ can be identified with its symmetric digraph $\hat{G}=(V, \hat{E})$ where $\hat{E}$ is obtained by replacing each unordered $\{a, b\}$ with two ordered pairs (a, b) & (b, a) for every edge $\{a, b\} \in E$.

$G=$ 

$\hat{G}=$ 

A two-way communication is equivalent to two one-way communications.

Date: February 16

Statement:

Modify the recursive version of DFS to output the parent array of the DFS tree.

```
procedure DFS(G, v) recursive
Input:  G  (a graph with n vertices &
           m edges)
        v  (a vertex where search begins)

output: the depth-first search of graph G
        starting with vertex v.

        Mark[v] ← true  // mark v as visited
        call Visit(v)
        for each vertex u adjacent to v do
            if Mark[u] = 0 then call DFS(G, u) endif
        endfor
end DFS
```

My work:

procedure DFS(G, v, Parent[0:n-1]) recursive

Input: G (a graph with n vertices & m edges)

v (a vertex where search begins)

Parent[0:n-1] (parent array for DFS tree rooted at v)

Output: the depth-first search of graph G starting with vertex v.

```
Mark[v] ← true  // mark v as visited
call Visit(v)
for each vertex u adjacent to v do
    if Mark[u] = 0 then
        call DFS(G, u)
        Parent[u] ← v
    endif
endfor
```

end DFS

Corrected Soln:

work above is correct.

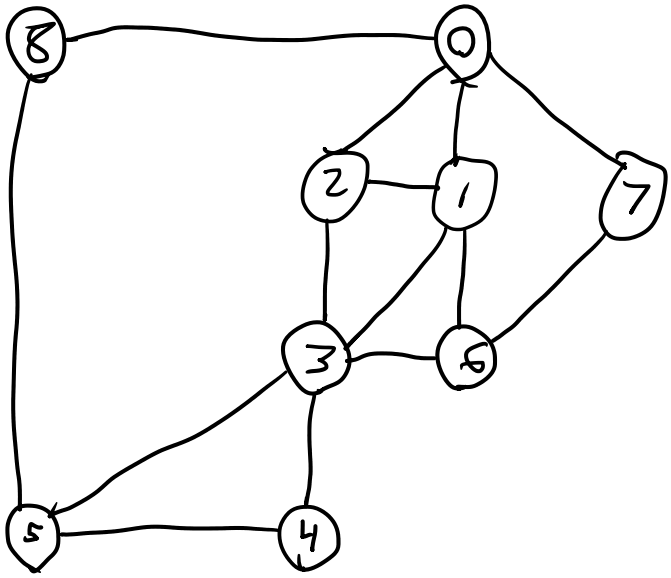
work above is correct.

Date: February 16

Statement:

Repeat the action of DFS in counting the DFS Tree & give the DFS order, where the initial vertex is v=0.

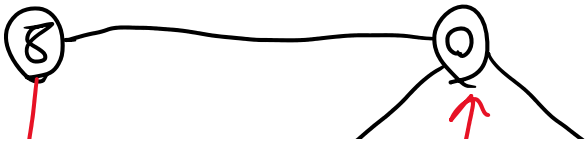
G =



My work:

DFS order: 0, 1, 2, 3, 4, 5, 8, 6, 7

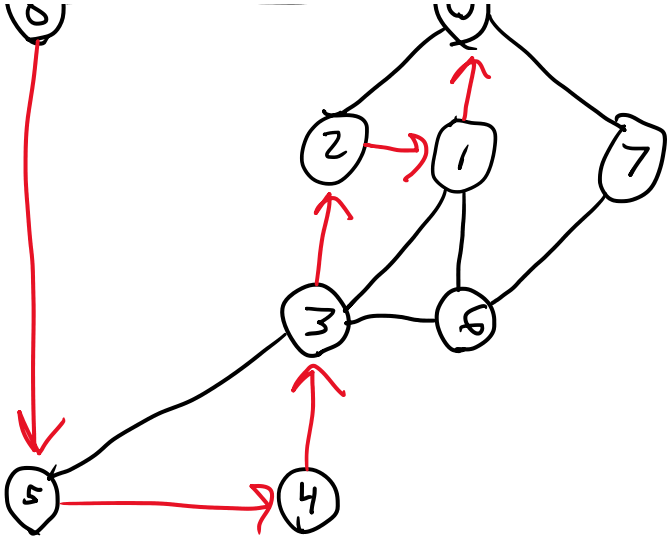
1.)



1.)

Get:

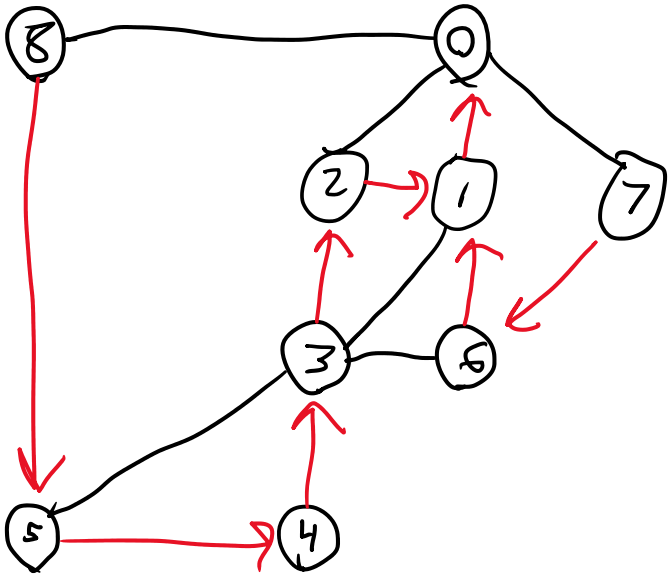
0
1
2
3
4
5
~~6~~ 8



2.)

backtrack to 7, get:

6
7



Corrected Soln:

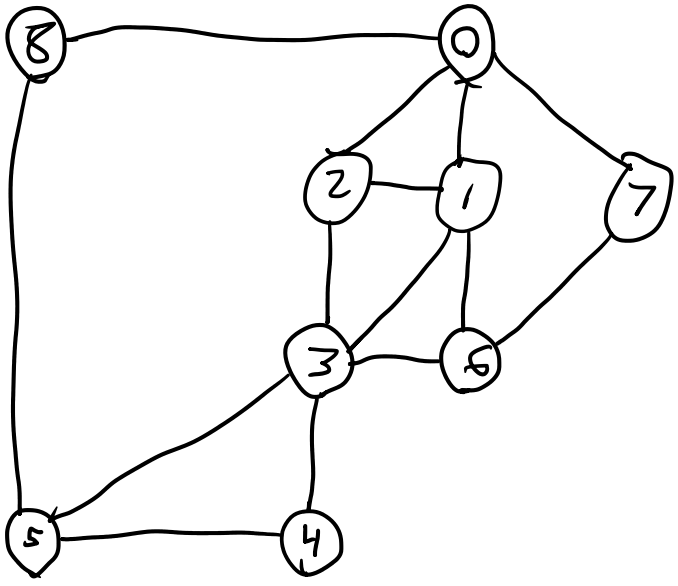
work above is correct.

Date: February 16

Statement:

Repeat the action of BFS in counting the BFS Tree & give the BFS order, where the initial vertex is v = 0.

G =



My work:

BFS order: 0, 1, 2, 7, 8, 3, 6, 5, 4

1.)

1.)

Get:

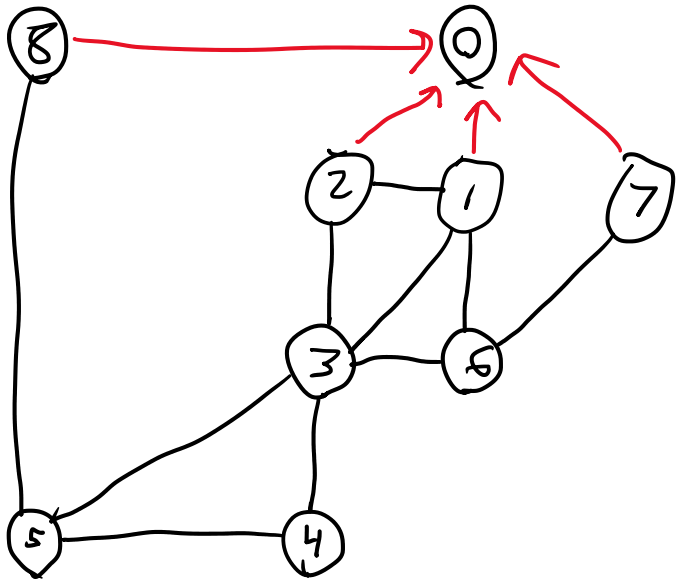
0

1

2

7

8



2.)

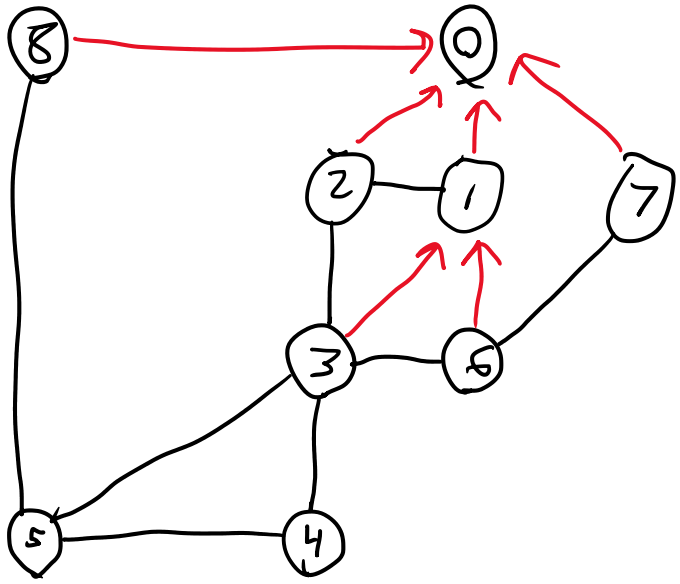
Go to 1

in queue,

get:

3

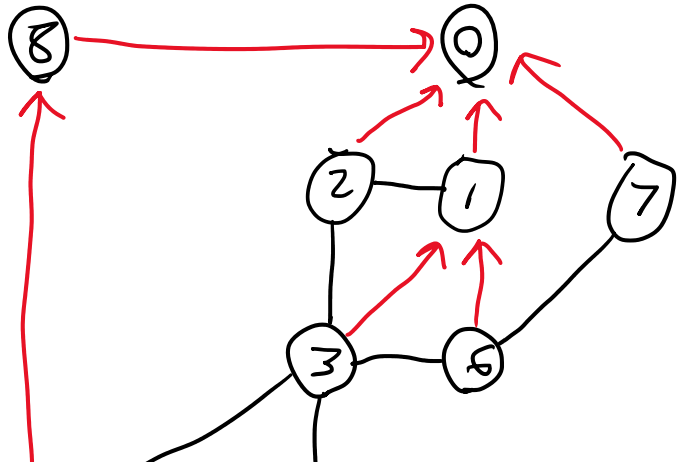
6



3.)

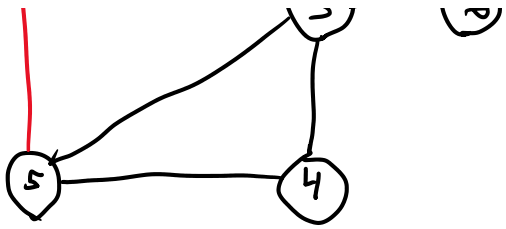
2 & 7 brings in nothing in queue go to 8 in queue, get:

5



queue, get:

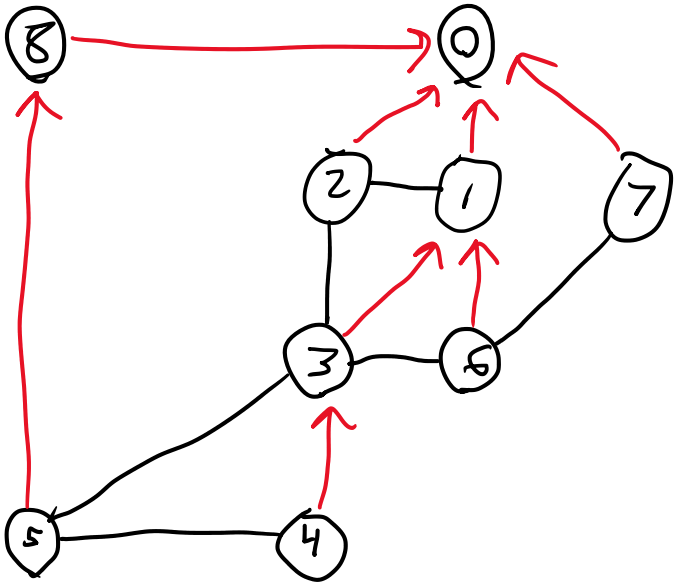
5



4.)

Go to 3 in queue, get:

4



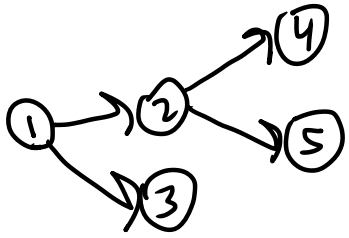
Directed Acyclic Graph (DAGs) and Topological Sorting - 1

Saturday, February 10, 2024 11:28 AM

Statement: Date: February 23

Show that a (finite) DAG must always contain a source & a sink.

A directed acyclic graph (DAG) is a digraph without any directed cycles.



Source - a vertex v where all edges incident with v are directed out of v, i.e., in-degree(v) = 0.

Sink - a vertex v where all edges incident with v are directed into v, i.e., out-degree(v) = 0.

My work:

I am unsure how we would effectively prove this.

Corrected Soln:

Start at any vertex & keep growing a path until a sink is reached. Since a digraph is acyclic, we cannot return to the same vertex, so that we must eventually reach a sink.

Reverse the orientation on all the edges & find a sink. This sink is a source in the original digraph.

Statement: Date: February 23

construct a digraph $D=(V, E)$ where V is the set of tasks & $(u,v) \in E$ whenever task u must precede, i.e., be performed before, task v.

Prove D is a DAG.

My work:

I am unsure how we would effectively prove this.

Corrected soln:

Proof by contradiction.

Suppose D is not a DAG. Then it contains

suppose ...

a directed cycle $v_1, v_2, \ldots, v_k, v_1$. It follows the definition of D that,

Task v_1 must be performed before task v_2.

Task v_2 must be performed before task v_3.

$\vdots$

Task v_{n-2} must be performed before task v_{n-1}.

Task v_{n-1} must be performed before task v_1.

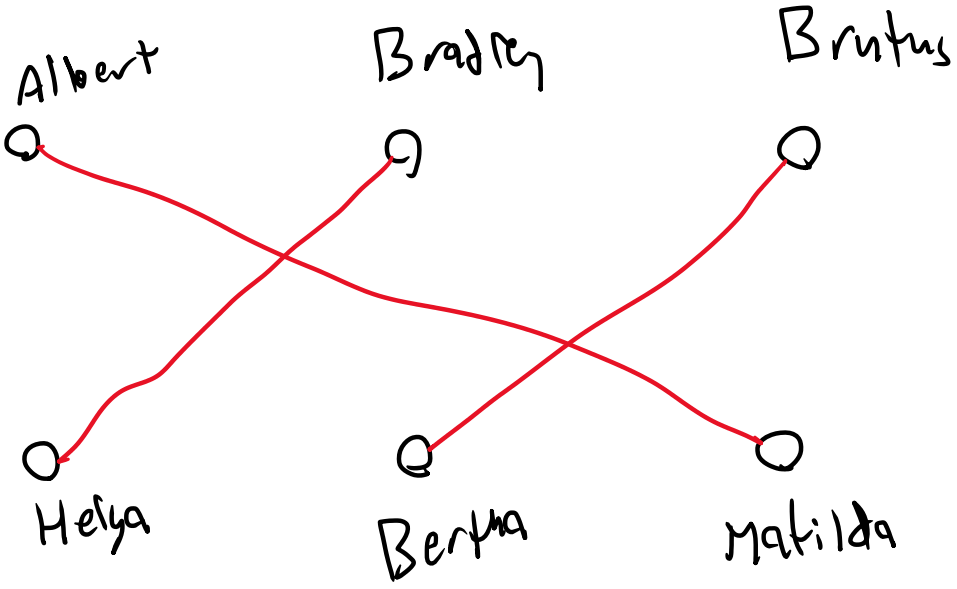
But this implies that task v_1 must be performed before itself, a contradiction.

Stable Bipartite Graph Matchings - 1

Sunday, February 4, 2024 6:46 PM

Date: February 26

Statement:



	1st	2nd	3rd
Albert	Helga	Bertha	Matilda
Bradley	Bertha	Helga	Matilda
Brutus	Helga	Bertha	Matilda

	1st	2nd	3rd
Helga	Bradley	Albert	Brutus
Bertha	Albert	Bradley	Brutus
Matilda	Albert	Bradley	Brutus

Is this perfect matching stable?

My work:

No, this perfect matching is not stable.

The pair (Albert, Helga) is unstable, Albert would be ~~better~~ matched with Helga & Brutus (who was with Helga) with Matilda (who was with Albert).

Date: February 28

Statement:

Solve the following instance of the knapsack problem with capacity $C = 15$:

object	b_0	b_1	b_2	b_3	b_4
value	100	20	30	15	66
weight	50	2	10	5	12

My work:

object	b_0	b_1	b_2	b_3	b_4
value	100	20	30	15	66
weight	50	2	10	5	12
density	2	10	3	3.75	5.5

density	2	10	3	3.75	5.5

Choose all b_1 - remaining weight is 13

Choose all b_4 - remaining weight is 1

Choose $\frac{1}{4}$ of b_3 - knapsack is full

value of knapsack is $20 + 66 + \frac{1}{4} \cdot 15 = 89.75$

corrected soln:
work above is correct.

Date: February 28

Statement:

Approximation to 0/1 knapsack using greedy.

Since the greedy solution to the knapsack problem chooses the whole objects, except for possibly the last object chosen, we can use this to get a solution to the 0/1 knapsack by simply removing the last object if a fraction of it is chosen.

Show that this greedy solution fails to give a good approximation to the 0/1 knapsack. In particular, show that for any given ϵ, there are examples where it gives worse than an M-approximation, i.e., the ratio of the value of an optimal solution to the value of the greedy solution is greater than M. For example, if $M = 1000000$, then the value of the optimal solution is more than one million times greater.

My work:

sort in descending order, i.e., the price & keep choosing as much of the most pricey coffee as possible until the coffee bag is full.

corrected soln:
work above is correct.

Sunday, February 25, 2024 2:28 PM

Statement: Date: March 1

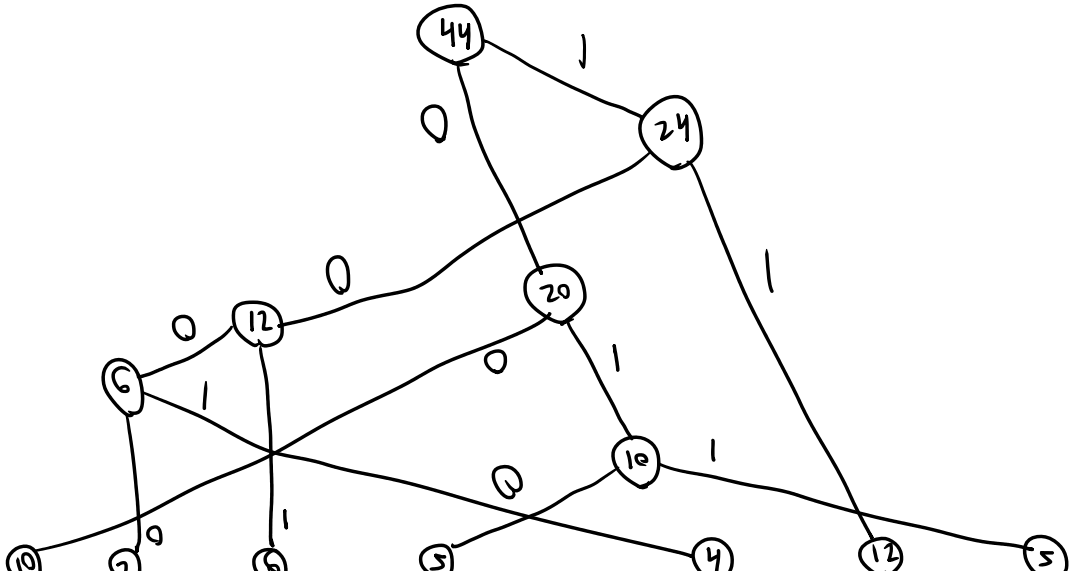
Illustrate the greedy method for computing Huffman tree & Huffman code following instance:

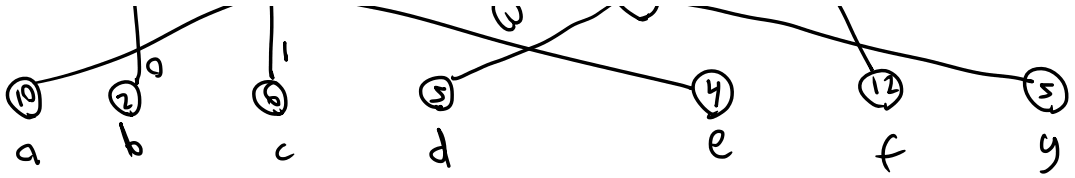
Symbol	a	b	c	d	e	f	g
frequency	10	2	6	5	4	12	5

My work:

I am unsure of how to draw this.

Corrected Sol'n





Symbol	Binary code
a	00
b	1000
c	101
d	010
e	1001
f	11
y	011

Greedy Method: Minimum Spanning Trees (MSTs), Kruskal's Algorithm - 1

Sunday, March 3, 2024 2:43 PM

Statement: Date: March 4

Give transformations showing:

a.) The MST problem for weights can be reduced to the MST problem for positive weights.

b.) The MST problem with some weights repeated can be reduced to the MST problem where all the weights are disjoint.

My work:

I am unsure of how to prove this.

- could solve

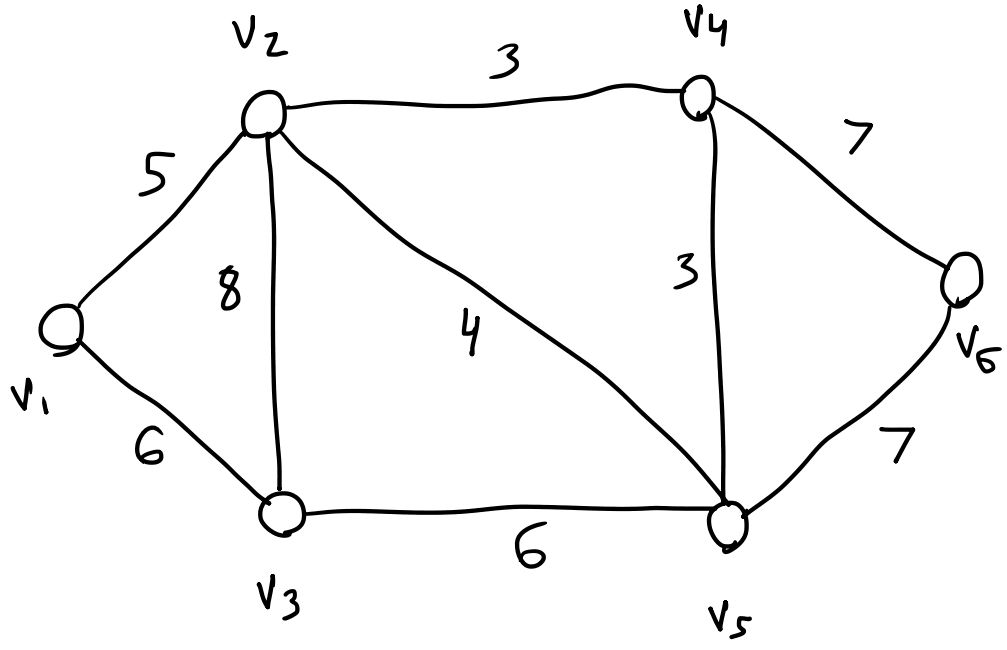
Corrected Soln:

a.) Let w' be the weighting obtained from w by adding a weight M to each edge, where M is chosen to be large enough so that all the w'-weights become positive. Then a spanning tree is a MST with respect to w iff it is a MST with respect to w'.

b.) Let w'' be the weighting obtained from w by adding a small weight ϵ_i to each edge e_i, where the ϵ_i's are distinct & their sum is smaller than the smallest weight. Then a spanning tree is a MST with respect to w iff it is a MST with respect to w''.

Statement: Date: March 6

Find a MST in the following weighted graph using Prim's algorithm starting at vertex v_3.



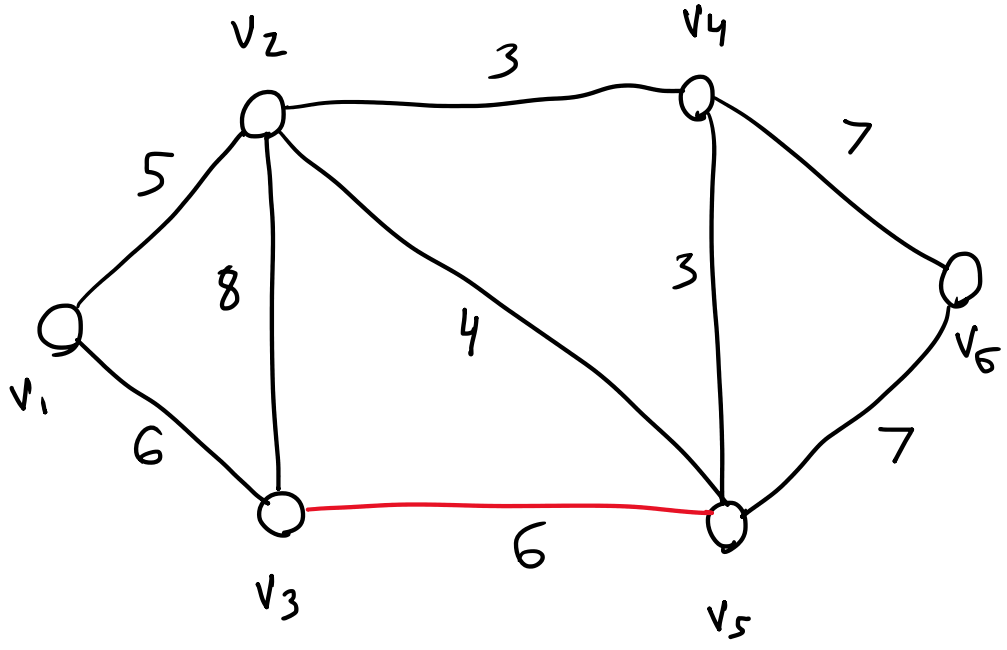
My work:

At each stage choose the smallest edge in cut (T).

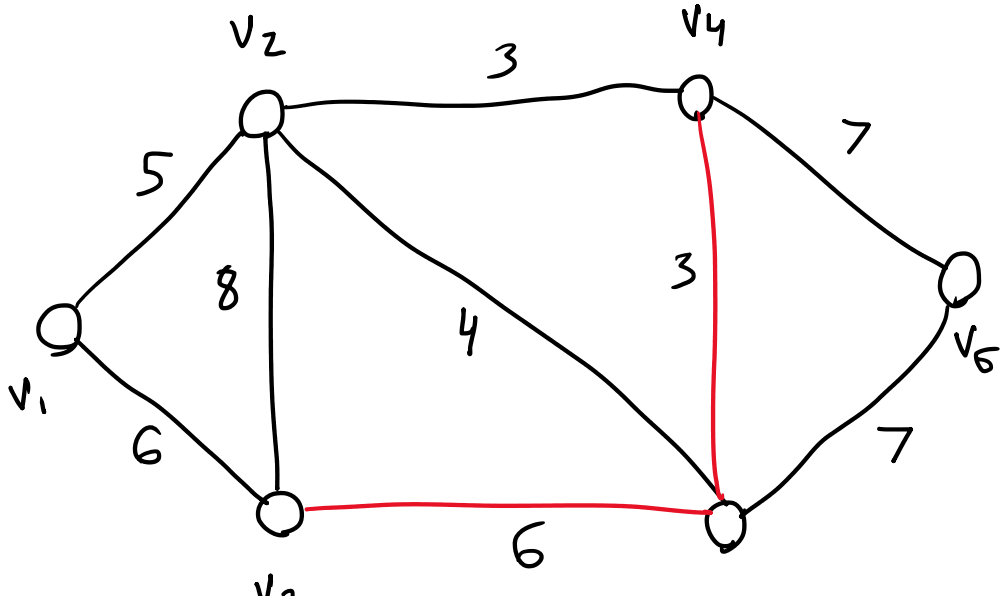
[illegible] choice can

when there is a tie, an arbitrary choice can be made.

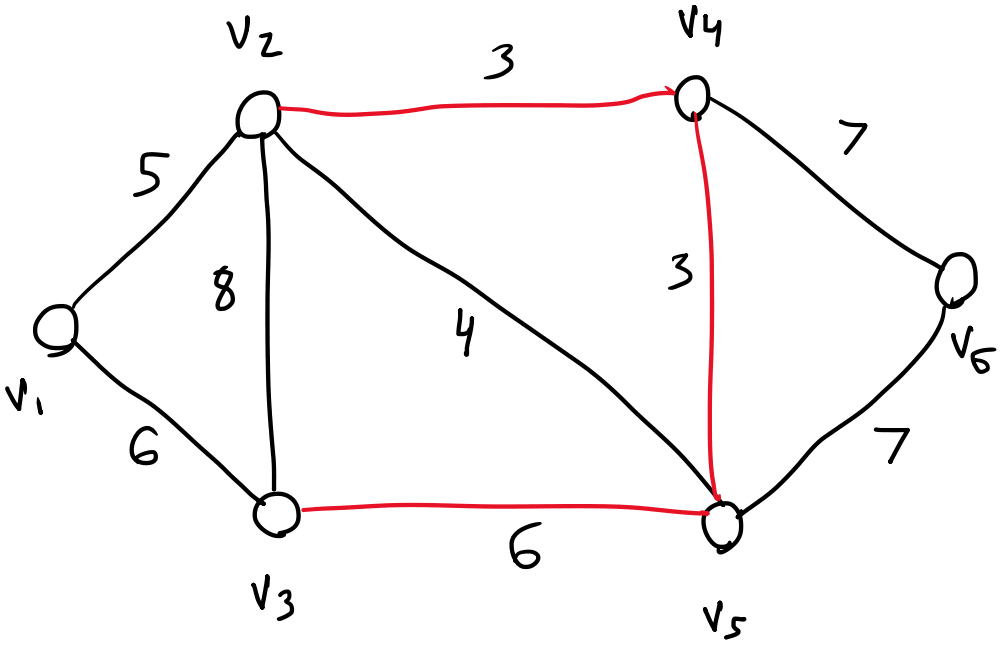
1.)



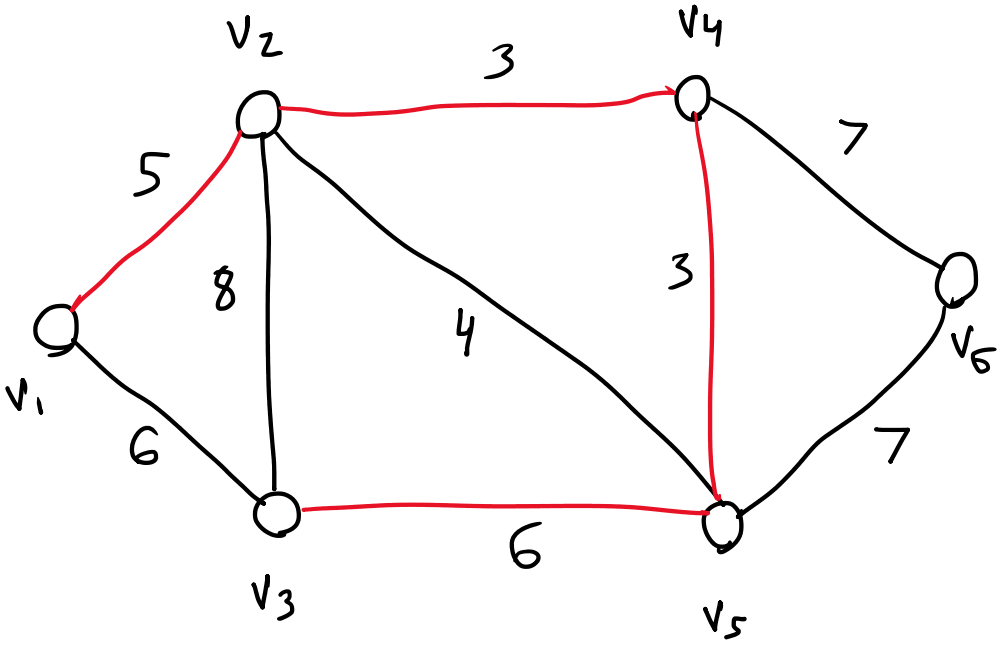
2.)



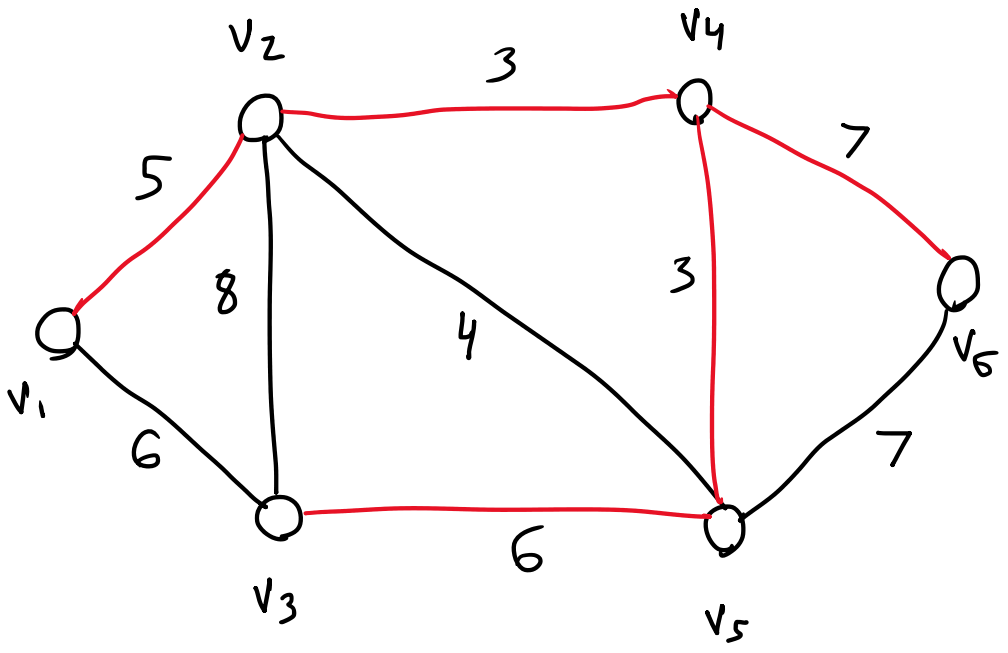
3.)



4.)



5.)

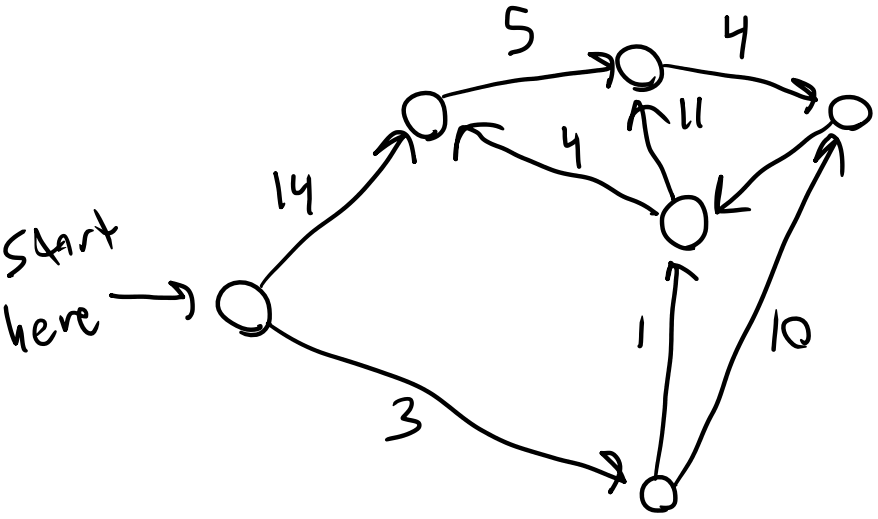


corrected soln:

work above is correct.

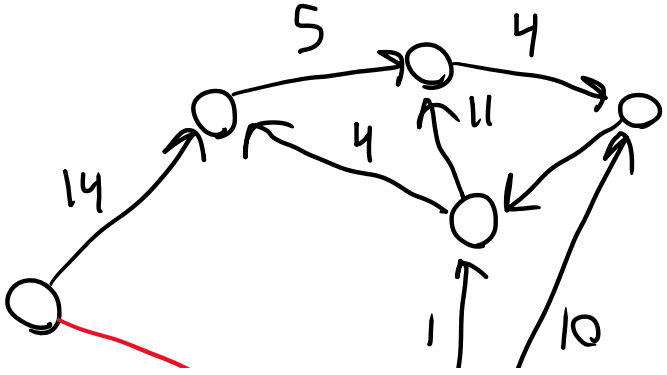
Statement: Date: March 8

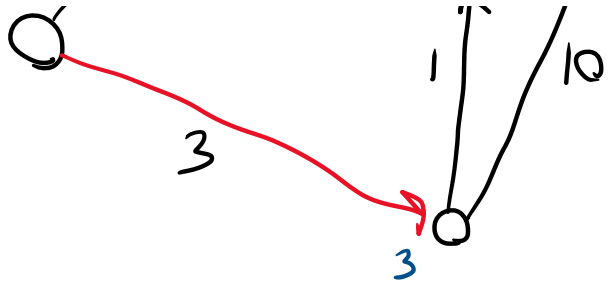
Show the action of Dijkstra's algorithm for the following digraph



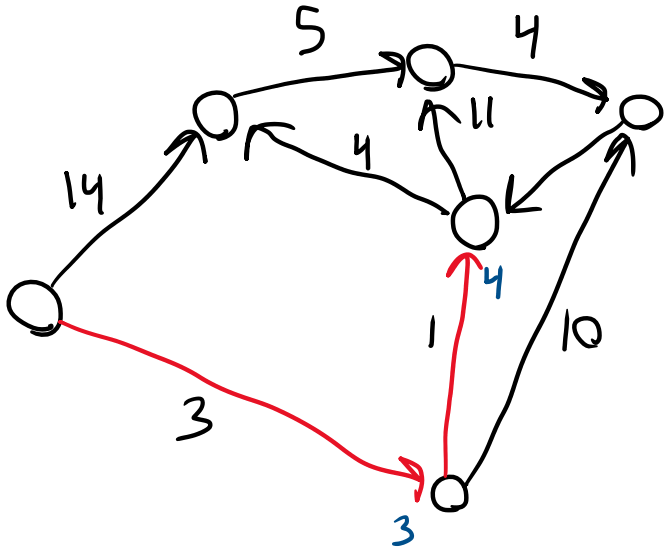
My work:

1.)

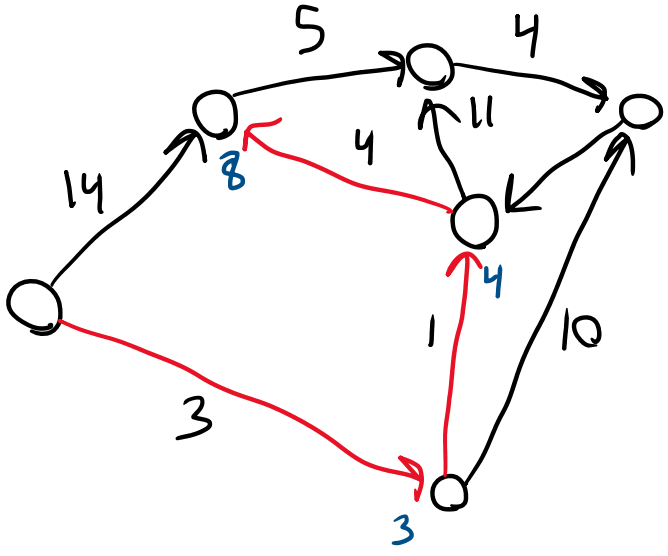




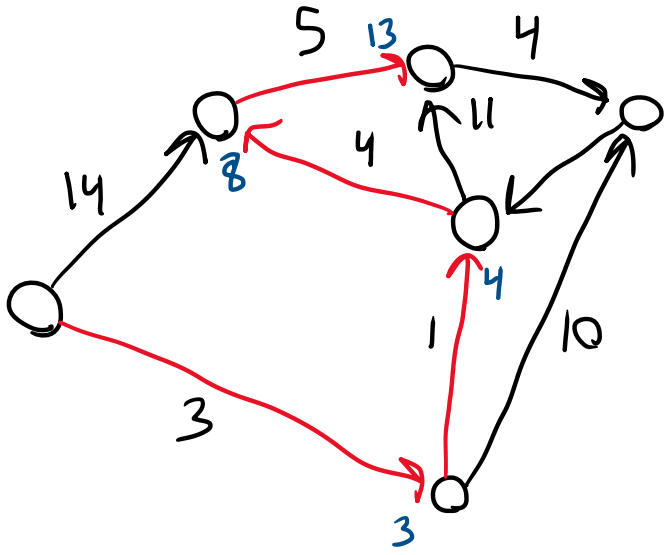
2.)



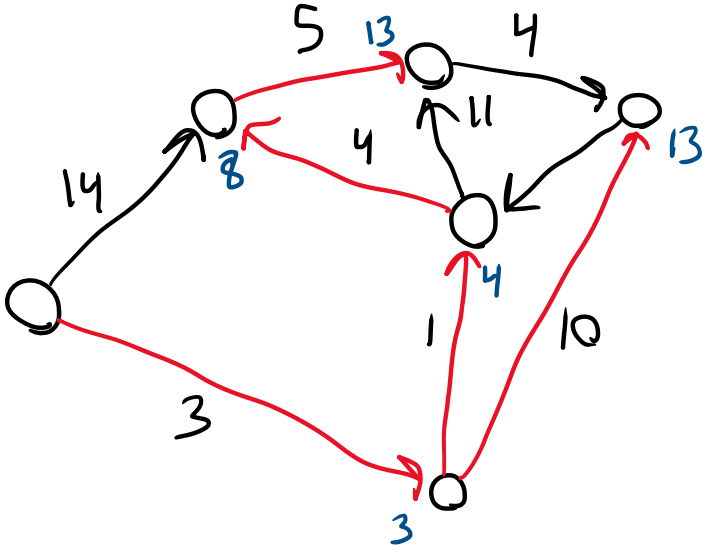
3.)



4.)



5.)



corrected soln:

work above is correct.

Matrix Multiplication Using Divide-&-Conquer - 1

Thursday, March 21, 2024 8:34 PM

Date: March 20

Statement:

Verify correctness for c_{01}.

My work:

Consider entry in first row & second column

$$m_3 + m_5 = a_{00} * (b_{01} - b_{11}) + (a_{00} + a_{01}) * b_{11}$$

$$= (a_{00} b_{01} - a_{00} b_{11}) + (a_{00} b_{11} + a_{01} b_{11})$$

$$= a_{00} b_{01} + a_{01} b_{11} = c_{01}$$

Corrected soln:

work above is correct.

Matrix Multiplication Using Divide-&-Conquer - 2

Thursday, March 21, 2024 8:43 PM

Date: March 20

Statement:

Verify $O(n^{2.795})$ result.

> Nearly ten years after Strassen discovered his identities, Pan found a way to multiply two 70 x 70 matrices that involves only 143,640 multiplications (compared to over 150,000 multiplications used by Strassen's method), yielding an algorithm that performs O(n^2.795) multiplications.

My work:

Recurrence Relation for worst-case Complexity of Pan's Algorithm.

Pan's algorithm divides each of the two $n \times n$ matrices A & B into 4900 block submatrices each of dimensions $\frac{n}{70} \times \frac{n}{70}$, then performs 143,640 recursive calls with these matrices of $\frac{1}{70}$ th the size

... recursive calls with these matrices of $\frac{1}{70}$th the size. Assuming $n = 70^k$,

$$W(n) = 143,640\,W\left(\frac{n}{70}\right), \quad W(1) = 1.$$

Corrected soln:

$$W(n) = 143,640\,W\left(\frac{n}{70}\right)$$

$$= 143,640\left(143,640\,W\left(\frac{n}{70^2}\right)\right)$$

$$= 143,640^2\,W\left(\frac{n}{70^2}\right)$$

$$= 143,640^2\left(143,640\,W\left(\frac{n}{70^3}\right)\right)$$

$$= 143,640^3\left(W\left(\frac{n}{70^3}\right)\right)$$

$$\vdots$$

$$= 143,640^k\,W\left(\frac{n}{70^k}\right)$$

$$= 143,640^k\,W(1)$$

$$= 143,640^k$$

$$= \left(70^{\log_{70}(143,640)}\right)^k$$

$$= \left(70^k\right)^{\log_{70}(143,640)}$$

$$= n^{\log_{70}(143,640)}$$

$$\approx n^{2.79512}$$

Discrete Fourier Transform (DFT) and Fast Fourier Transform Algorithm (FFT) - 1

Thursday, March 21, 2024 11:12 AM

Statement: Date: March 22

Compute DFT$_\omega$ P(x) for $P(x) = x^3 + 3x^2 + 2x - 1$ & $\omega = -i$ & give associated coefficient arrays.

My work:

$P(x) = x^3 + 3x^2 + 2x - 1 \leftrightarrow [-1, 2, 3, 1]$

$P(1) = 5$

$P(-i) = -4 + i$

$P(-1) = -1$

$P(i) = -4 - i$

corrected soln:

$Q(x) = \text{DFT}_\omega P(x)$

$= (-4 - i)x^3 - x^2 + (-4 + i)x + 5$

$\Leftrightarrow [5, -4+i, -1, -4-i]$

Discrete Fourier Transform (DFT) and Fast Fourier Transform Algorithm (FFT) - 2

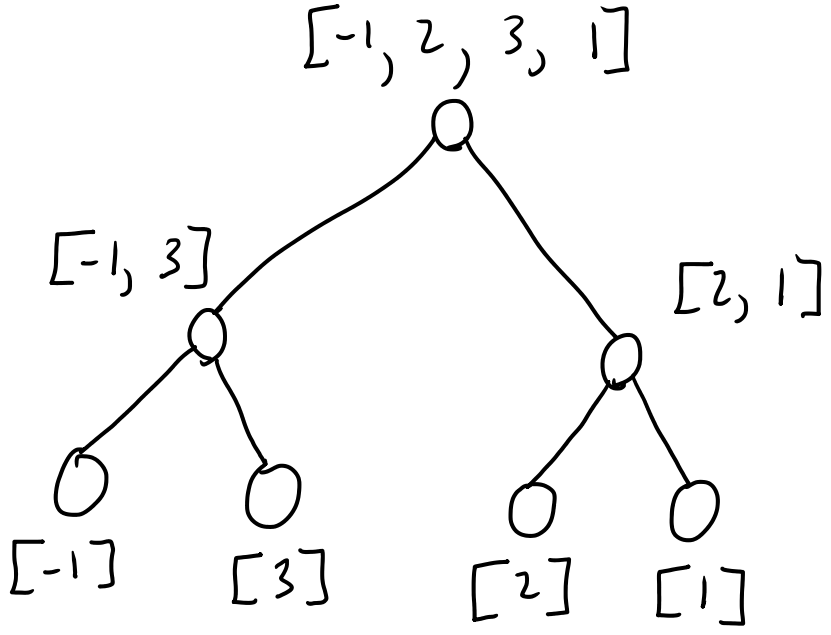
Thursday, March 21, 2024 4:46 PM

Statement: Date: March 22

Show the action of FFT in computing DFT(ω) $P(x)$ for $P(x) = x^3+3x^2+2x-1$ & $\omega = -i$ & give associated coefficient arrays.

My work:

$P(x) = x^3+3x^2+2x-1 \leftrightarrow [-1, 2, 3, 1]$



Corrected sol'n:

$n=1,\ \omega=1:\ [-1]\ [3]\ [2]\ [1]$

$n=2,\ \omega=-1:\ [-1+3,\ -1+3]\qquad [2+1][2-1]$

$=[2,\ -4]\qquad =[3,\ -1]$

$n=4,\ \omega=-i:\ [2+3,\ -4+(-1)(-i),\ 2-3,\ -4-(-1)(-i)]$

$=[5,\ -4+i,\ -1,\ -4,\ -i]$

$$Q(x)=(-4-i)x^3-x^2+(-4+i)x+5$$

Statement: Date: March 25

For $k=7$, i.e., $n=2^k=128$, what is the index i of a_i for leaf node 11?

My work:

11_{10} is 1011_2 in binary.

We add leading zeros to bring the number of digits to $k=7$, i.e., 0001011_2.

Then, we reverse the bits,

1101000_2,

which is 104_{10}, i.e., a_{11} ends up in leaf node 104.

Statement:

Date: April 1

Consider a diagraph $D = (V, E)$ whose edges are weighted with real numbers such that there are no negative cycles, i.e., no directed cycles so that the sum of the weights over the edges of the cycle is strictly negative. Then the principle of optimality holds for shortest paths.

My work:

I am unsure of what this question is asking. Why are we making a statement on the principle of optimality?

Corrected sol'n:

Proof by Contradiction.

Assume the principle of optimality does not hold for shortest paths in weighted digraphs. i.e.

hold for shortest paths in weighted digraphs, i.e., there is a shortest path P from x to y in digraph D that contains vertex z, but its subpaths Q & R from x to z & z to y, respectively, are not both shortest paths.

we can assume without loss of generality that Q is not shortest, i.e., there exists a shorter path Q' from x to z.

Replace Q with Q' in P. This will result in a walk W from x to z that is strictly shorter than P.

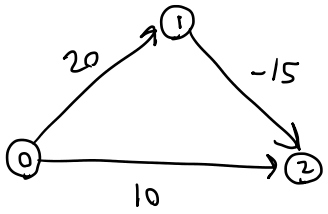
Since there are no negative cycles, removing cycles from W will result in a path P* that is no greater in length than W.

Since W is strictly shorter than P & P* is no greater in length than W, it follows that P* is strictly shorter than P, a contradiction to our assumption that P is a shortest path.

Statement:

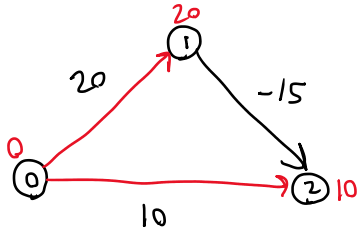
Date: April 1

Show that Dijkstra's algorithm fails for the following weighted DAG but the Floyd-Warshall algorithm works.



My work:

Applying Dijkstra's algorithm at vertex 0, we obtain the distances & tree shown in red below. But the path 01 of length 10 is not shortest. The shortest path is 012 of length $20 + (-15) = 5$.



Corrected soln:

The Floyd-Warshall algorithm yields correct distances & paths:

$$S_0 = \begin{pmatrix} 0 & 20 & 10 \\ \infty & 0 & -15 \\ \infty & \infty & 0 \end{pmatrix}, \quad S_1 = \begin{pmatrix} 0 & 20 & 10 \\ \infty & 0 & -15 \\ \infty & \infty & 0 \end{pmatrix}, \quad S_2 = \begin{pmatrix} 0 & 20 & 10 \\ \infty & 0 & -15 \\ \infty & \infty & 0 \end{pmatrix}, \quad S_2 = \begin{pmatrix} 0 & 20 & 10 \\ \infty & 0 & -15 \\ \infty & \infty & 0 \end{pmatrix}$$

$$P_1 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \quad P_2 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}, \quad P_2 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Statement: Date: April 3

$m(P) = m(P_1) + m(P_2) + pqr$

Give a proof by contradiction that shows that both the left & right parenthesizations P_1 & P_2 of P must be optimal for P to be optimal.

My work:

I am unsure of how to prove this.

corrected soln:

Assume that P is optimal, but at least one of P_1 & P_2 is suboptimal. Without loss of generality, we may assume that P_1 is suboptimal. Replace P_1 with an optimal parenthesization P_1' &

let P' denote the resulting parenthesization.

since $m(P_1') < m(P_1)$, we have,

$$m(P') = m(P_1') + m(P_2) + pqr$$
$$< m(P_1) + m(P_2) + pqr = m(P)$$

But this contradicts the assumption that P is optimal. QED.

Statement: Date: April 3

Find the optimal parenthesization for,

$d = (3, 10, 4, 10, 1)$

M_0, M_1, M_2, M_3 having dimensions

$3 \times 10, 10 \times 4, 4 \times 10, 10 \times 1$

My work:

$d = (3, 10, 4, 10, 1)$

$m_{00} = m_{11} = m_{22} = m_{33} = 0$

$m_{01} = 3 \times 10 \times 4 = 120$

$m_{12} = 10 \times 4 \times 10 = 400$

$m_{23} = 4 \times 10 \times 1 = 40$

$m_{02} = \min\{m_{00} + m_{12} + 3 \times 10 \times 10,\ m_{01} + m_{22} + 3 \times 4 \times 10\}$

$= \min\{0 + 400 + 300,\ 120 + 0 + 120\} = 240,$ First Cut = 1

$m_{13} = \min\{m_{11} + m_{23} + 10 \times 4 \times 1,\ m_{12} + m_{33} + 10 \times 10 \times 1\}$

$= \min\{0 + 40 + 40,\ 400 + 0 + 100\} = 80,$ First Cut = 1

$m_{03} = \min\{m_{00} + m_{13} + 3 \times 10 \times 1,\ m_{01} + m_{23} + 3 \times 4 \times 1,\ m_{02} + m_{33} + 3 \times 10 \times 1\}$

$= \min\{0 + 80 + 30,\ 120 + 40 + 12,\ 240 + 0 + 30\} = 110,$ First Cut = 0

Associated matrices:

$$m = \begin{pmatrix} 0 & 120 & 240 & 110 \\ & 0 & 400 & 80 \\ & & 0 & 40 \\ & & & 0 \end{pmatrix}$$

$$FirstCut = \begin{pmatrix} & & 1 & 0 \\ & & & 1 \end{pmatrix}$$

Computing optimal parenthesization:

$(M_0\ M_1\ M_2\ M_3)$

For 0_3, first cut at $k=0$:

$(M_0(M_1\ M_2\ M_3))$

For 1_3, first cut at $k=1$:

$(M_0(M_1(M_2\ M_3)))$

Corrected soln:
work above is correct.

Statement: Date: April 5

Prove the Principle of Optimality holds for Optimal Binary Search Tree.

My work:

I am unsure of how to prove this.

Corrected Sol'n:

Assume that T is an optimal binary search tree, but at least one of L & R is suboptimal. Without loss of generality, we may assume that L is suboptimal. Replace L with an optimal binary search tree L' & let T' denote the resulting binary searching tree. Since $A(L') < A(L)$, we have,

$A(L') < A(L)$, we have,

$$A(T') = A(L') + A(R) + \sigma(i, j)$$
$$< A(L) + A(R) + \sigma(i, j) = A(T)$$

But, this contradicts the assumption that T is optimal. QED.

Statement: Date: April 5

Solve the action of OptimalSearchTree for the following instance. Instead of drawing trees, compute Root[i, j] each stage.

i	0	1	2	3
p_i	0.25	0.25	0.05	
q_i	0.20	0	0	0.25

My work:

$A(T_{00}) = 0.25 + 0.20 + 0 = 0.45$,

$A(T_{11}) = 0.25 + 0 + 0 = 0.25$,

$A(T_{22}) = 0.50 + 0 + 0.25 = 0.30$

$A(T_{01}) = \min\{0 + A(T_{11}),\ A(T_{00}) + 0\} + 0.7$

$= \min\{\underline{0.25},\ 0.45\} + 0.7 = 0.95$

$Root[0, 1] = 0$

$Root[0,1] = 0$

$A(T_{12}) = \min\{0 + A(T_{22}),\ A(T_{11}) + 0\} + 0.55$

$= \min\{0.30,\ \underline{0.25}\} + 0.55 = 0.80$

$Root[1,2] = 2$

$A(T_{02}) = \min\{0 + A(T_{12}),\ A(T_{00}) + A(T_{22}),\ A(T_{01}) + 0\} + 1$

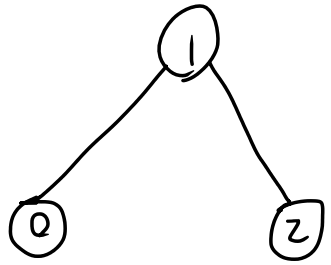
$= \min\{0.80,\ 0.45 + 0.30,\ 0.95\} + 1$

$= \min\{0.80,\ \underline{0.75},\ 0.95\} + 1 = 1.75$

$Root[0,2] = 1$

$$Root = \begin{pmatrix} & 0 & 1 \\ & & 2 \end{pmatrix}$$

$T_{02} =$



$A(T_{02}) = 1.75$

Statement: Date: April 8

Give pseudocode for recursive version of the Sum of Subsets problem backtracking solution using the variable tuple state space tree.

My work:

I am unsure of how to write this algorithm.

Corrected sol'n:

procedure SumOfSubsetsRec(k) recursive

Input: k (a nonnegative integer, 0 on initial call)

A[0:n-1] (global array of positive integers)

Sum (global positive integer less than A[0]+...+A[n-1])

X[0:n] (an array of integers, where X[1:n] stores variable-tuple problem states, X[1],..., X[k] have already been assigned, & where X[0]=-1 for convenience of pseudocode)

PathSum (global variable - A[X[1]]+A[X[k]], initialized to 0)

Output: print all descendant goal states of (X[1], ..., X[k]); that is, print all (X[1], ..., X[k], X[k+1], ..., X[q]) such that A[X[1]] + ... + A[X[k]] + A[X[k+1]] + ... + A[X[q]] = Sum

```
k ← k+1    // move on to the next level

Temp ← PathSum

for Child ← X[k-1]+1 to n-1 do
    X[k] ← Child
    PathSum ← Temp + A[X[k]]
    if PathSum ≥ Sum then    // X[1], ..., X[k] is bounded
        if PathSum = Sum then
            Print(X[1], ..., X[k])  //print goal state
        endif
    else
        SumOfSubsetsRec(k)  // (X[1], ..., X[k]) is not bounded
    endif
endfor
```

end SumOfSubsetsRec

Statement: Date: April 10

Give a linear time algorithm for solving the problem $k=2$?

My work:

Graph $G=(V, E)$, $n=|V|$, $m=|E|$.

Perform a BFS keeping track of the distance from the root (initial) vertex for each BFS.

If the distance is even, color the vertex 0, otherwise, if the distance is odd, color it 1.

This gives a proper vertex 2-coloring, if

This gives a proper vertex 2-coloring, if one exists, i.e., if the graph is bipartite. If any edge is monochromatic, i.e., both end vertices are colored the same, then the graph is not bipartite & the graph can't be properly vertex 2-colored.

This algorithm has computing time $O(m+n)$, where m & n denote the number of edges & vertices.

Corrected solution:

The work above is correct.

Statement: Date: April 10

Show that the k-clique can be solved in polynomial time for k constant.

a.) The 3-clique problem can be solved in time $O(n^3)$.

b.) For k constant, the k-clique problem can be solved in $O(n^k)$.

My work:

I am unsure how to solve this problem.

Corrected solution:

We obtain a polynomial function using brute force. In the case of $k=3$, check

brute force. In the case of $k=3$, check every three vertices to see whether it forms a triangle. The number of subsets of 3 vertices is

$$C(n,3) = n(n-1)(n-2)/6 \in \Theta(n^3).$$

For constant k, we look at every subset of k vertices to see whether one is a k-clique. The number of such subsets is

$$C(n,k) = n(n-1)\ldots(n-k-1)/k! \in \Theta(n^k).$$

Saturday, April 6, 2024 4:38 PM

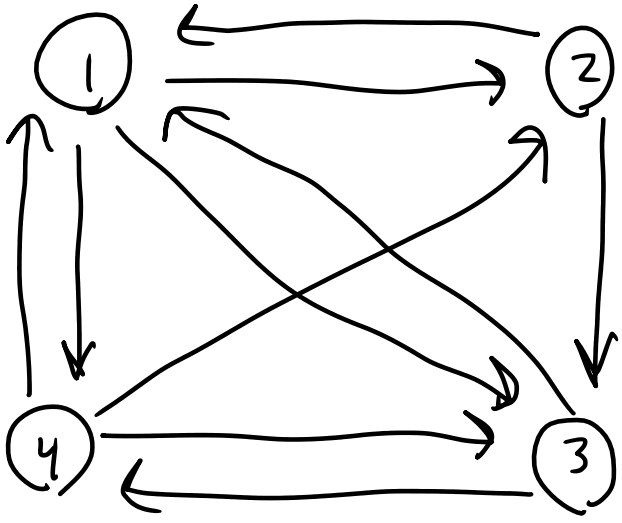
Statement: Date: April 12

For the mini-web below:

a.) Give the PageRank equations

b.) Give the associated matrix equation

c.) Give transpose matrix & verify it is the matrix for a random walk.



My work:

the PageRank equations

the PageRank equations

$$R_1 = \frac{1}{2} R_2 + \frac{1}{2} R_3 + \frac{1}{3} R_4$$

$$R_2 = \frac{1}{3} R_1 + \frac{1}{3} R_4$$

$$R_3 = \frac{1}{3} R_1 + \frac{1}{2} R_2 + \frac{1}{3} R_4$$

$$R_4 = \frac{1}{3} R_1 + \frac{1}{2} R_3$$

the associated matrix equation

Let B be the matrix for a random walk on W, i.e.,

$$B[i, j] = \begin{cases} \frac{1}{\text{outdeg}(i)}, & ij \in E(W) \\ 0, & \text{otherwise.} \end{cases}$$

For web, B is the matrix for a random

walk

$$\begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}$$

$$R = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{3} & 0 & 1 & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{1}{2} & 0 \end{pmatrix} R, \quad \text{where} \quad R = \begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \end{bmatrix}$$

or equivalently,

$$R = \begin{pmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & 1 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \vdots & \vdots & \vdots & \end{pmatrix}^T R$$

$$\begin{pmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}$$