

ALGORITHMS AND DATA

STRUCTURES'

CSci 4041

FACULTY: NIKOS PAPANIKOLOPOULOS

— · —

ALGORITHMS

- * AN ALGORITHM IS ANY WELL-DEFINED COMPUTATIONAL PROCEDURE THAT TAKES SOME VALUE, OR SET OF VALUES, AS INPUT AND PRODUCES SOME VALUE, OR SET OF VALUES, AS OUTPUT.
- * AN ALGORITHM CAN BE VIEWED AS A TOOL FOR SOLVING A WELL-SPECIFIED COMPUTATIONAL PROBLEM.
- * AN ALGORITHM IS SAID TO BE CORRECT IF, FOR EVERY INPUT INSTANCE, IT PROVIDES THE CORRECT OUTPUT.
- * EXAMPLE: SORTING PROBLEM
 - INPUT: A SEQUENCE OF n NUMBERS $\langle a_1, \dots, a_n \rangle$
 - OUTPUT: A PERMUTATION (REORDERING) OF THE INPUT SEQUENCE $\langle a'_1, \dots, a'_n \rangle$ SUCH THAT $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

ALGORITHMS

* Θ-NOTATION

$\Theta(g(n)) = \{ f(n) : \text{THERE EXIST POSITIVE CONSTANTS } c_1, c_2 \text{ AND } n_0 \text{ SUCH THAT } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ FOR ALL } n \geq n_0 \}$

EXAMPLE: $f(n) = \frac{1}{2}n^2 - 3n$

$c_1 = 1/14$, $c_2 = 1/2$ AND $n_0 = 7$ PROVIDE EVIDENCE THAT $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

* O-NOTATION

$O(g(n)) = \{ f(n) : \text{THERE EXIST POSITIVE CONSTANTS } c \text{ AND } n_0 \text{ SUCH THAT } 0 \leq f(n) \leq c g(n) \text{ FOR ALL } n \geq n_0 \}$

- ASYMPTOTIC UPPER BOUND

ALGORITHMS

* Ω-NOTATION

$\Omega(g(n)) = \{ f(n) : \text{ THERE EXIST POSITIVE CONSTANTS } c \text{ AND } n_0 \text{ SUCH THAT } 0 \leq c g(n) \leq f(n) \text{ FOR ALL } n > n_0 \}$

- ASYMPTOTIC LOWER BOUND

* THEOREM

FOR ANY TWO FUNCTIONS $f(n)$ AND $g(n)$,

$f(n) = \Theta(g(n))$ IF AND ONLY IF $f(n) = O(g(n))$

AND $f(n) = \Omega(g(n))$

* EXAMPLE

THE FACT THAT $an^2 + bn + c = \Theta(n^2)$ FOR

ANY CONSTANTS a, b , AND c ($a > 0$) IMMEDIATELY

IMPLIES THAT $an^2 + bn + c = \Omega(n^2)$ AND $an^2 + bn + c =$

$O(n^2)$

BINARY SEARCH

* PROBLEM

LET $\langle a_1, a_2, \dots, a_n \rangle$ BE A SEQUENCE OF REAL NUMBERS SUCH THAT $a_1 \leq a_2 \leq \dots \leq a_n$. GIVEN A REAL NUMBER z , WE WANT TO FIND WHETHER z APPEARS IN THE SEQUENCE, AND, IF IT DOES, TO FIND AN INDEX i SUCH THAT $a_i = z$.

* INPUT

A (A SORTED ARRAY IN THE RANGE 1 TO n)
z (THE SEARCH KEY)

* OUTPUT

POSITION (AN INDEX i SUCH THAT $A[i] = z$, OR 0 IF NO SUCH INDEX EXIST)

* ORDER IS IMPORTANT

BINARY SEARCH

* ALGORITHM (PSEUDOCODE)

FIND(z, left, right) {IT RETURNS INTEGER}

if $left = right$ then

 if $A[left] = z$ then return $left$

 else return 0

else

 middle $\leftarrow \lceil \frac{1}{2}(left + right) \rceil$

 if $z < A[middle]$ then

 temp $\leftarrow \text{FIND}(z, left, middle - 1)$

 else

 temp $\leftarrow \text{FIND}(z, middle, right)$

 return temp

* COMPLEXITY

THE NUMBER OF COMPARISONS IS $O(\log n)$

BINARY SEARCH

- * BINARY SEARCH IN A CYCLIC SEQUENCE
(PROBLEM DESCRIPTION)

GIVEN A CYCLICALLY SORTED LIST, FIND THE POSITION OF THE MINIMAL ELEMENT IN THE LIST (IT IS ASSUMED THAT THIS POSITION IS UNIQUE)

- * A SEQUENCE $\langle a_1, a_2, \dots, a_n \rangle$ IS CYCLICALLY SORTED IF THE SMALLEST NUMBER IN THE SEQUENCE IS a_i FOR SOME UNKNOWN i , AND THE SEQUENCE $\langle a_i, a_{i+1}, \dots, a_n, a_1, \dots, a_{i-1} \rangle$ IS SORTED IN INCREASING ORDER.
- * THE ALGORITHM IS A VARIATION OF THE BINARY SEARCH.
- * THE COMPLEXITY IS SIMILAR TO THE COMPLEXITY OF BINARY SEARCH.

BINARY SEARCH

* ALGORITHM (PSEUDOCODE)

CYCLIC-FIND(left, right) { IT RETURNS INTEGER }

if left = right then

 return left

else

 middle $\leftarrow \lfloor \frac{1}{2} (\text{left} + \text{right}) \rfloor$

 if A[middle] < A[right] then

 temp \leftarrow CYCLIC-FIND(left, middle)

 else

 temp \leftarrow CYCLIC-FIND(middle+1, right)

 return temp

BINARY SEARCH

* BINARY SEARCH FOR A SPECIAL INDEX

(PROBLEM DESCRIPTION)

GIVEN A SORTED SEQUENCE OF DISTINCT
INTEGERS $\langle a_1, \dots, a_n \rangle$ DETERMINE WHETHER
THERE EXISTS AN INDEX i SUCH THAT $a_i = i$

* ALGORITHM

SPECIAL-FIND (left, right) {ST RETURNS INTEGER}

if $left = right$ then

 if $A[left] = left$ then return $left$

 else return 0

else

 middle $\leftarrow \lceil \frac{1}{2}(left+right) \rceil$

 if $A[middle] < middle$ then

 temp \leftarrow SPECIAL-FIND(middle+1, right)

 else

 temp \leftarrow SPECIAL-FIND(left, middle)

 return temp

THE STUTTERING-SUBSEQUENCE

PROBLEM

* THE PROBLEM:

GIVEN TWO SEQUENCES A AND B, FIND THE MAXIMAL VALUE OF i SUCH THAT B^i IS A SUBSEQUENCE OF A.

* GIVEN A SEQUENCE B, WE DEFINE B^i TO BE THE SEQUENCE B WITH EACH CHARACTER APPEARING i TIMES CONSECUTIVELY.

EXAMPLE: $B = xyz \quad B^2 = xxyyyzz$

* SEQUENCE B (LENGTH m) IS SUBSEQUENCE OF THE SEQUENCE A (LENGTH n) IF WE CAN EMBED B INSIDE A IN THE SAME ORDER BUT WITH POSSIBLE HOLES.

* THE ALGORITHM THAT DETERMINES IF B IS SUBSEQUENCE OF A IS SIMPLE AND THE RUNNING TIME IS $O(n+m)$.

INTERPOLATION SEARCH

* ALGORITHM (PSEUDOCODE)

I_FIND(z , left, right) { IT RETURNS INTEGER }

if $A[\text{left}] = z$ then return left

else if $\text{left} = \text{right}$ or $A[\text{left}] = A[\text{right}]$ then
return 0

else

$$\text{next-guess} \leftarrow [\text{left} + \frac{(z - A[\text{left}])(\text{right} - \text{left})}{A[\text{right}] - A[\text{left}]}]$$

if $z < A[\text{next-guess}]$ then

$\text{temp} \leftarrow I_FIND(z, \text{left}, \text{next-guess}-1)$

else

$\text{temp} \leftarrow I_FIND(z, \text{next-guess}, \text{right})$

return temp

* COMPLEXITY

THE AVERAGE NUMBER OF COMPARISONS IS
 $O(\log \log n)$

SORTING

- * THE PROBLEM:

GIVEN n NUMBERS a_1, a_2, \dots, a_n , ARRANGE
THEM IN INCREASING ORDER.

- * A SORTING ALGORITHM IS CALLED IN-PLACE
IF NO ADDITIONAL WORKSPACE IS USED BESIDES
THE INITIAL ARRAY THAT HOLDS THE ELEMENTS.

- * DIFFERENT SORTING ALGORITHMS

BUCKET SORT

MERGE SORT

COUNTING SORT

SELECTION SORT

HEAPSORT

QUICK SORT

INSERTION SORT

SHELL'S SORT

- * SORTING IS ONE OF THE MOST POPULAR PROBLEMS
IN COMPUTER SCIENCE.

INSERTION SORT

* ALGORITHM (PSEUDOCODE)

INSERTION-SORT(A) { A IS THE ARRAY OF NUMBERS}

for $j \leftarrow 2$ to $\text{length}(A)$ do

 key $\leftarrow A[j]$

{INSERT $A[j]$ INTO THE}

{SORTED SEQUENCE $A[1..j-1]$ }

$i \leftarrow j-1$

 while $i > 0$ and $A[i] > \text{key}$ do

$A[i+1] \leftarrow A[i]$

$i \leftarrow i-1$

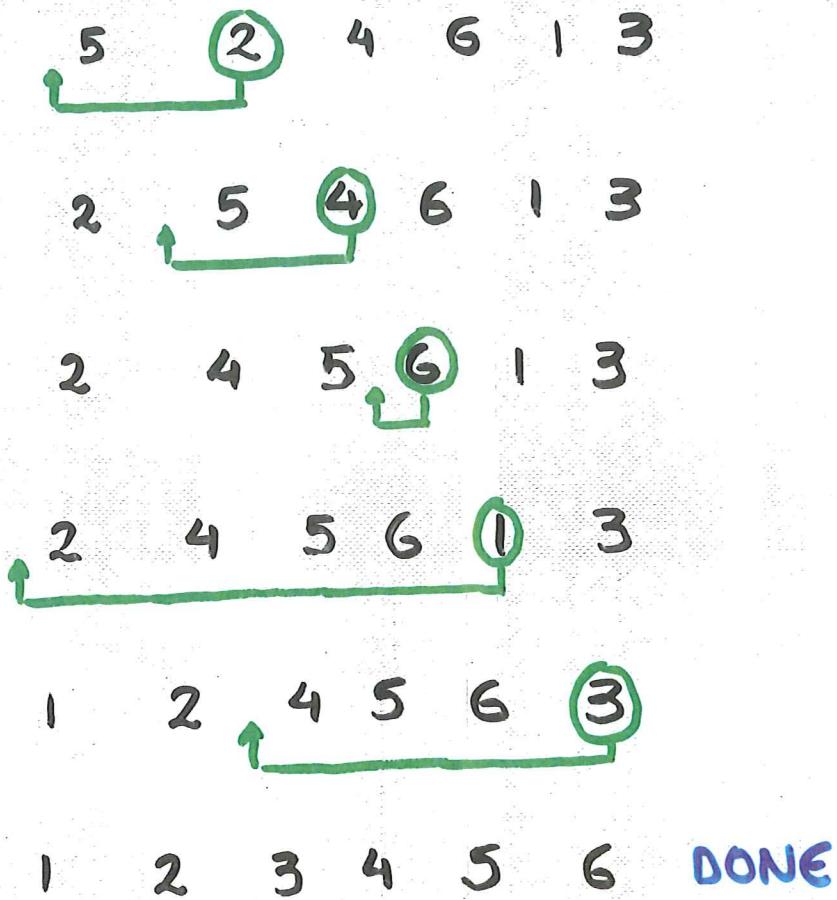
$A[i+1] \leftarrow \text{key}$

* BEST CASE

* WORST CASE

INSERTION SORT

* EXAMPLE



- * THE BEST CASE OCCURS WHEN THE ARRAY IS ALREADY SORTED.
- * THE WORST CASE OCCURS WHEN THE ARRAY IS IN REVERSE SORTED ORDER.

INSERTION SORT

* ANALYSIS OF THE ALGORITHM

ALGORITHM	COST	TIMES
INSERTION-SORT(A)		
for $j \leftarrow 2$ to $\text{length}(A)$ do	c_1	n
key $\leftarrow A[j]$	c_2	$n-1$
$i \leftarrow j-1$	c_3	$n-1$
while $i > 0$ and $A[i] > \text{key}$ do	c_4	$\sum_{j=2}^n t_j$
$A[i+1] \leftarrow A[i]$	c_5	$\sum_{j=2}^n (t_j - 1)$
$i \leftarrow i-1$	c_6	$\sum_{j=2}^n (t_j - 1)$
$A[i+1] \leftarrow \text{key}$	c_7	$n-1$

t_j : THE NUMBER OF TIMES THE WHILE LOOP TEST
IS EXECUTED FOR THAT VALUE OF j

RUNNING TIME:

$$\begin{aligned}
 T(n) = & c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^n t_j \\
 & + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7(n-1)
 \end{aligned}$$

INSERTION SORT

* ANALYSIS OF THE ALGORITHM

- ARITHMETIC SERIES

$$\sum_{j=1}^n j = \frac{1}{2} n(n+1) \quad \sum_{j=2}^n j = \frac{1}{2} (n+1)n - 1$$

$$\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

- BEST CASE ($t_j=1$ for $j=2,3,\dots,n$)

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1) \\ &= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) \\ &= \alpha n + b \text{ (LINEAR FUNCTION OF } n) \end{aligned}$$

- WORST CASE ($t_j=j$ for $j=2,3,\dots,n$)

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1) \\ &= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n \\ &\quad - (c_2 + c_3 + c_4 + c_7) \\ &= \alpha n^2 + bn + c \text{ (QUADRATIC FUNCTION OF } n) \end{aligned}$$

- AVERAGE CASE IS OFTEN ROUGHLY AS BAD AS THE WORST CASE

BUCKET SORT

* ALGORITHM (PSEUDOCODE)



BUCKET-SORT(A) { $A[i] \leq 1$ AND $A[i] > 0\}$

$n \leftarrow \text{Length}(A)$

for $i \leftarrow 1$ to n do

 insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$

for $i \leftarrow 0$ to $n-1$ do

 sort list $B[i]$ with insertion sort

concatenate the lists $B[0], B[1], \dots, B[n-1]$

together in order

* EXAMPLE

A	B
1 .78	0 /
2 .17	1 → .12 → .17
3 .39	2 → .21 → .23 → .26
4 .26	3 → .39
5 .72	4 /
6 .94	5 /
7 .21	6 → .68
8 .12	7 → .72 → .78
9 .23	8 /
10 .68	9 → .94

RADIX SORT

* ALGORITHM (PSEUDOCODE)

RADIX-SORT(A,d) {d:NUMBER OF DIGITS}

for $i \leftarrow 1$ to d do

use a stable sort to sort array A

on digit i

- 1 IS THE LOWEST-ORDER DIGIT

- d IS THE HIGHEST-ORDER DIGIT

* EXAMPLE

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
136	657	355	657
720	329	457	720
355	839	657	839

↑ ↑ ↑

* COMPLEXITY DEPENDS ON THE INTERMEDIATE
SORTING ALGORITHM

MERGESORT

* ALGORITHM (PSEUDOCODE)

M.SORT (left, right)

if right - left = 1 then

 if A[left] > A[right] then swap(A[left], A[right])

else if left ≠ right then

 middle ← ⌊½(left + right)⌋

 M.SORT(left, middle - 1)

 M.SORT(middle, right)

 i ← left

 j ← middle

 k ← 0

 while (i ≤ middle - 1) and (j ≤ right) do

 k ← k + 1

 if A[i] ≤ A[j] then

 TEMP[k] ← A[i]

 i ← i + 1

 else

 TEMP[k] ← A[j]

 j ← j + 1

 if j > right then

 for t ← 0 to middle - 1 - i do

 A[right - t] ← A[middle - 1 - t]

 for t ← 0 to k - 1 do

 A[left + t] ← TEMP[t]

MERGESORT

* EXAMPLE

INPUT 6 2 8 5 10 9 12 1 15 7 3 13 4 11 16 14

2 6 8 5 10 9 12 1 15 7 3 13 4 11 16 14

2 6 5 8 10 9 12 1 15 7 3 13 4 11 16 14

2 5 6 8 9 10 12 1 15 7 3 13 4 11 16 14

2 5 6 8 9 10 12 1 15 7 3 13 4 11 16 14

2 5 6 8 9 10 12 15 7 3 13 4 11 16 14

2 5 6 8 1 9 10 12 15 7 3 13 4 11 16 14

1 2 5 6 8 9 10 12 15 7 3 13 4 11 16 14

1 2 5 6 8 9 10 12 7 15 3 13 4 11 16 14

: : : :

1 2 5 6 8 9 10 12 3 4 7 11 13 14 15 16

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

MERGESORT

* COMPLEXITY

$T(n)$: NUMBER OF COMPARISONS THAT MERGESORT REQUIRES IN THE WORST CASE

n IS POWER OF 2

$$T(2n) = 2T(n) + O(n) \quad T(2) = 1$$



$$T(n) = O(n \log n)$$

* GENERAL DIVIDE AND CONQUER RELATION

$$T(n) = aT(n/b) + cn^k \quad \left(\begin{array}{l} n=b^m \\ T(1)=c \end{array} \right)$$



$$T(n) = c \sum_{i=0}^m a^{m-i} b^{ik} = ca^m \sum_{i=0}^m \left(\frac{b^k}{a}\right)^i$$

$$\hookrightarrow O(n^{\log_b a}) \quad \text{if } a > b^k$$

$$T(n) = \begin{cases} \hookrightarrow O(n^{\log_b n}) & \text{if } a = b^k \\ \hookrightarrow O(n^k) & \text{if } a < b^k \end{cases}$$

$a > 1, b \geq 2, c$ AND k POSITIVE CONSTANTS

QUICKSORT

* ALGORITHM (PSEUDOCODE)

Q-SORT(left, right)

if left < right then

PARTITION(A, left, right)

Q-SORT(left, middle - 1)

Q-SORT(middle + 1, right)

PARTITION(A, left, right) { IT SELECTS middle AND }
{ CHANGES A }

pivot $\leftarrow A[\text{left}]$

l \leftarrow left

r \leftarrow right

while l < r do

 while A[l] \leq pivot and l \leq right do

 l \leftarrow l + 1

 while A[r] $>$ pivot and r $>$ left do

 r \leftarrow r - 1

 if l < r then

 exchange A[l] and A[r]

middle \leftarrow r

exchange A[left] with A[middle]

QUICKSORT

* EXAMPLE OF PARTITIONING

6 2 8 5 10 9 12 1 15 7 3 13 4 11 16 14
6 2 4 5 10 9 12 1 15 7 3 13 8 11 16 14
6 2 4 5 3 9 12 1 15 7 10 13 8 11 16 14
6 2 4 5 3 1 12 9 15 7 10 13 8 11 16 14
1 2 4 5 3 6 12 9 15 7 10 13 8 11 16 14

* EXAMPLE OF THE ALGORITHM

6 2 8 5 10 9 12 1 15 7 3 13 4 11 16 14
1 2 4 5 3 6 12 9 15 7 10 13 8 11 16 14
1 2 4 5 3 6 12 9 15 7 10 13 8 11 16 14
1 2 3 4 5 6 12 9 15 7 10 13 8 11 16 14
1 2 3 4 5 6 8 9 11 7 10 12 13 15 16 14
1 2 3 4 5 6 7 8 11 9 10 12 13 15 16 14
1 2 3 4 5 6 7 8 10 9 11 12 13 15 16 14
1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

QUICKSORT

* WORST-CASE PARTITIONING

THIS HAPPENS WHEN THE INPUT ARRAY IS
COMPLETELY SORTED.

$$T(n) = T(n-1) + O(n) \text{ AND } T(1) = O(1)$$



$$T(n) = T(n-1) + O(n) = \sum_{k=1}^n O(k) = O\left(\sum_{k=1}^n k\right) \\ = O(n^2)$$

* BEST-CASE PARTITIONING

THIS HAPPENS WHEN THE PARTITIONING PROCEDURE
PRODUCES TWO REGIONS OF SIZE $n/2$.

$$T(n) = 2T(n/2) + O(n) \text{ AND } T(2) = 1$$



$$T(n) = O(n \log n)$$

* BALANCED PARTITIONING

THE AVERAGE CASE RUNNING TIME OF QUICKSORT
IS MUCH CLOSER TO THE BEST CASE THAN TO THE
WORST CASE.

QUICKSORT

* RANDOMIZED VERSIONS OF QUICKSORT

- THE PIVOT IS SELECTED RANDOMLY.
- EACH OF THE $A[i]$ HAS THE SAME PROBABILITY OF BEING SELECTED (ASSUMPTION)

RUNNING TIME:

$$T(n) = n-1 + T(i-1) + T(n-i) \quad (\text{iTH SMALLEST ELEMENT IS THE PIVOT})$$

AVERAGE RUNNING TIME:

$$\begin{aligned} T(n) &= n-1 + \frac{1}{n} \sum_{i=1}^n (T(i-1) + T(n-i)) \\ &= n-1 + \frac{2}{n} \sum_{i=0}^{n-1} T(i) \end{aligned}$$

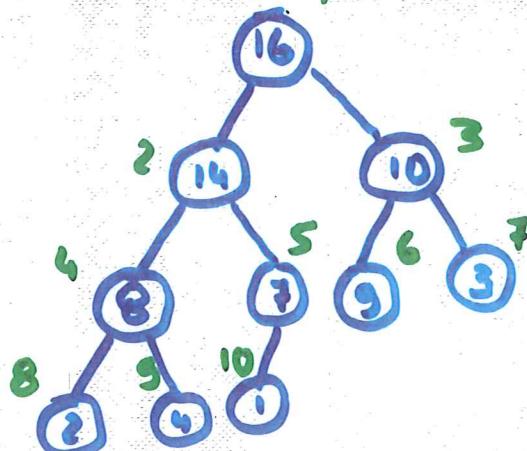


$$T(n) = O(n \log n)$$

- * PIVOT IS STORED IN A REGISTER (SPEED ISSUE).
- * SELECTION OF THE BASE OF INDUCTION.
- * SEVERAL VERSIONS OF QUICKSORT.

HEAPSORT

* HEAP



1 2 3 4 5 6 7 8 9 10
16 14 10 8 7 9 3 2 4 1

* HEAP PROPERTY

- FOR EVERY NODE i OTHER THAN THE ROOT

$$A[\text{PARENT}(i)] \geq A[i]$$

* CODE

PARENT(i)

return $\lfloor i/2 \rfloor$

LEFT(i)

return $2i$

RIGHT(i)

return $2i+1$

* THE HEIGHT OF A HEAP WITH n ELEMENTS IS $\Theta(\log n)$.

HEAPSORT

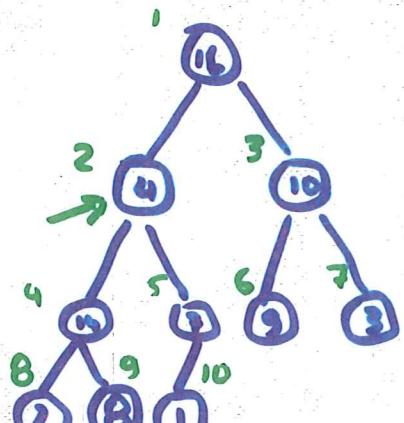
* ALGORITHM FOR HEAPIFY (PSEUDOCODE)

```
HEAPIFY( A, i ) { THE INPUTS ARE AN ARRAY }  
{ A AND AN INDEX i INTO THE }  
{ ARRAY }  
p ← LEFT(i) { THE BINARY TREES ROOTED }  
r ← RIGHT(i) { AT LEFT(i) AND RIGHT(i) ARE }  
{ ASSUMED TO BE HEAPS. }  
if p ≤ heap-size(A) and A[p] > A[i] then  
    largest ← p  
else  
    largest ← i  
if r ≤ heap-size(A) and A[r] > A[largest] then  
    largest ← r  
if largest ≠ i then  
    exchange ( A[i] , A[largest] )  
    HEAPIFY( A, largest )
```

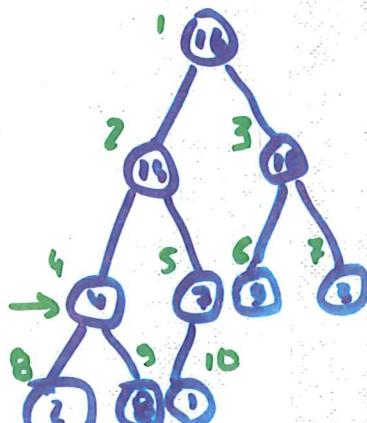
- `heap-size(A)`: THE NUMBER OF ELEMENTS IN THE HEAP STORED WITHIN ARRAY A

HEAPSORT

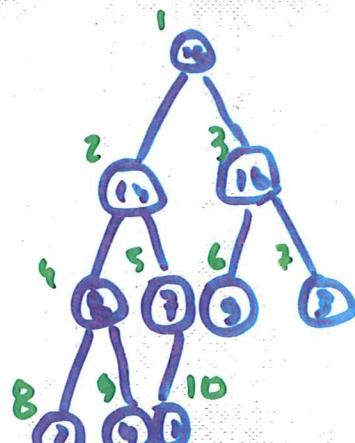
* EXAMPLE OF HEAPIFY



(a)



(b)



(c)

* RUNNING TIME OF HEAPIFY

$$T(n) = O(\log n)$$

HEAPSORT

* BUILDING A HEAP

BUILD-HEAP(A)

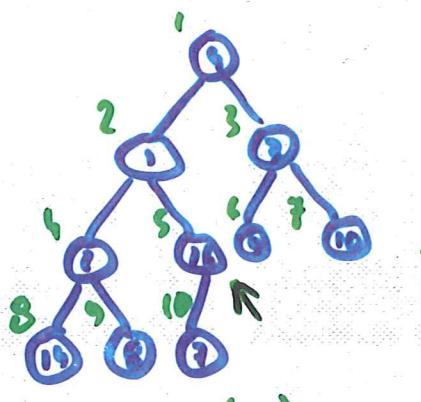
`heap-size(A) \leftarrow length(A)`

for $i \leftarrow \lfloor \text{Length}(A)/2 \rfloor$ downto 1 do

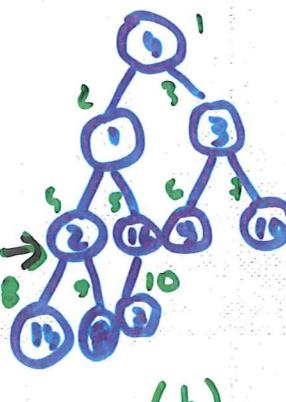
HEAPIFY(A,i)

* EXAMPLE OF BUILDING A HEAP

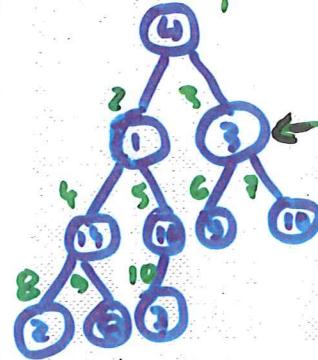
A → 4 1 3 2 16 9 10 14 8 7



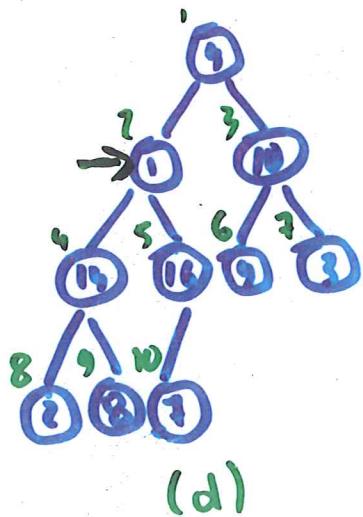
(a)



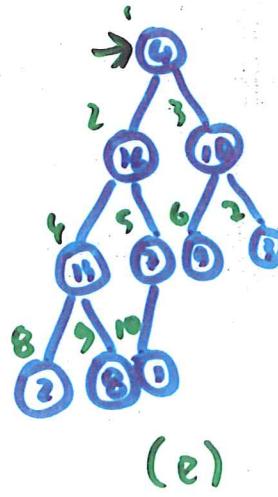
(b)



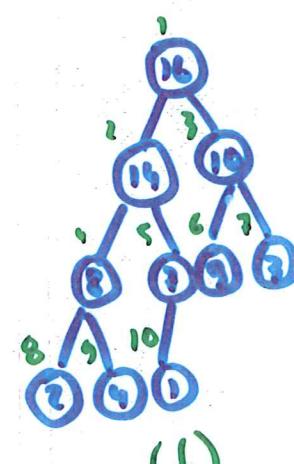
6



(d)



(e)



(4)

HEAPSORT

* ALGORITHM (PSEUDO CODE)

HEAPSORT(A)

BUILD-HEAP(A)

for $i \leftarrow \text{length}(A)$ down to 2 do

exchange($A[1], A[i]$)

heap-size(A) $\leftarrow \text{heap-size}(A) - 1$

HEAPIFY(A, 1)

- * THE HEAPSORT ALGORITHM TAKES TIME $O(n \log n)$ SINCE THE CALL TO BUILD-HEAP TAKES TIME $O(n)$ AND EACH OF THE $n-1$ CALLS TO HEAPIFY TAKES TIME $O(\log n)$.
- * HEAPS ARE USEFUL FOR THE IMPLEMENTATION OF PRIORITY QUEUES.

A LOWER BOUND FOR SORTING

- * A LOWER BOUND FOR A PARTICULAR PROBLEM IS A PROOF THAT NO ALGORITHM CAN SOLVE THE PROBLEM BETTER.
- * DECISION TREES CAN HELP.
- * DECISION TREES ARE DEFINED AS BINARY TREES WITH TWO TYPES OF NODES (INTERNAL NODES AND LEAVES).
- * EACH INTERNAL NODE IS ASSOCIATED WITH A QUERY.
- * EACH LEAF IS ASSOCIATED WITH A POSSIBLE OUTPUT.
- * THE WORST-CASE RUNNING TIME IS ASSOCIATED WITH THE HEIGHT OF THE TREE.
- * EVERY DECISION TREE ALGORITHM FOR SORTING HAS HEIGHT $\Omega(n \log n)$.

FINDING THE k th-SMALLEST ELEMENT

* PROBLEM

GIVEN A SEQUENCE S OF n ELEMENTS, AND AN INTEGER k SUCH THAT $1 \leq k \leq n$, FIND THE k th-SMALLEST ELEMENT IN S .

* ALGORITHM (PSEUDOCODE)

SELECT- k (left, right, k) { IT RETURNS THE k th-{SMALLEST ELEMENT}

if $\text{left} = \text{right}$ then
 return left

else

PARTITION(A, left, right) {IT IS SIMILAR TO}

{THE PARTITION OF}
{QUICKSORT}

if $\text{middle} - \text{left} + 1 \geq k$ then

 temp \leftarrow SELECT- k (left, middle, k)

else

 temp \leftarrow SELECT- k (middle + 1, right,
 $k - (\text{middle} - \text{left} + 1)$)

return temp

DATA COMPRESSION

- * DATA COMPRESSION IS AN IMPORTANT TECHNIQUE FOR SAVING STORAGE.
- * EACH OF THE 2⁶ ENGLISH CHARACTERS IS REPRESENTED BY A UNIQUE STRING OF BITS, CALLED THE ENCODING OF THE CHARACTER.
- * EXAMPLE
ASCII: A \leftrightarrow 1000001, B \leftrightarrow 1000010, ETC.
- * PREFIX CONSTRAINT.
- * PROBLEM
GIVEN A TEXT, FIND AN ENCODING FOR THE CHARACTERS THAT SATISFIES THE PREFIX CONSTRAINT AND THAT MINIMIZES THE TOTAL NUMBER OF BITS NEEDED TO ENCODE THE TEXT.

DATA COMPRESSION

- * LENGTH OF THE FILE F COMPRESSED BY USING ENCODING E IS:

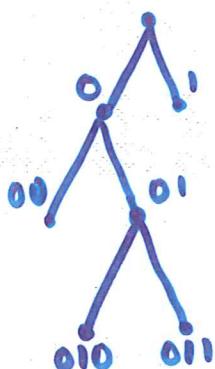
$$L(E, F) = \sum_{i=1}^n s_i f_i$$

n: NUMBER OF CHARACTERS

s_i: LENGTH OF THE BIT STRING THAT REPRESENTS THE CHARACTER C_i

f_i: FREQUENCY OF THE CHARACTER C_i

- * TREE REPRESENTATION OF ENCODING.



- * HUFFMAN INVENTED A GREEDY ALGORITHM THAT CONSTRUCTS AN OPTIMAL PREFIX CODE CALLED A HUFFMAN CODE.

DATA COMPRESSION

* HUFFMAN'S ALGORITHM (PSEUDOCODE)

HUFFMAN_ENCODING(S, f) { S IS A STRING OF }
{ CHARACTERS }
{ f IS AN ARRAY OF }
{ FREQUENCIES }

{ INSERT ALL CHARACTERS INTO A HEAP H ACCORDING }
{ TO THEIR FREQUENCIES }

while H is not empty do

if H contains only one character A then

make A the root of the tree

else

pick two characters A and B with lowest frequencies and delete them from H .

replace A and B with a new character C whose frequency is the sum of frequencies of A and B .

insert C to H .

make A and B children of C in the tree.

DATA COMPRESSION

* EXAMPLE OF THE HUFFMAN'S ALGORITHM

FREQUENCY

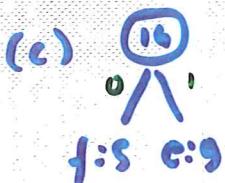
(a) f:5 e:9 c:12 b:13 d:16 a:45

(b) c:12 b:13



d:16

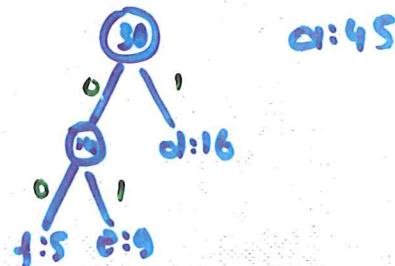
a:45



c:12 b:13

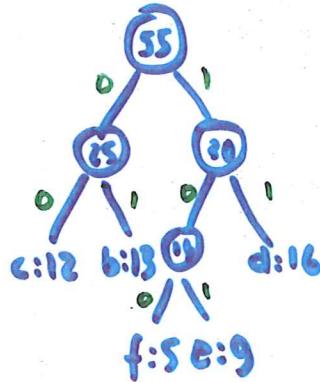
a:45

(d) c:12 b:13

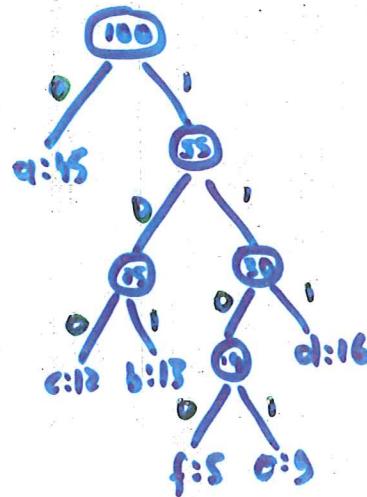


a:45

(e) a:45



(f)



STRING MATCHING

* PROBLEM

GIVEN TWO STRINGS A AND B, FIND THE FIRST OCCURENCE(IF ANY) OF B IN A.

* STRAIGHTFORWARD STRING MATCHING

A: abaabababbababababbabaa

B: ababbabaaa

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

abaa bab abb a b a b a b b a b a b a a

1: abab.....

2: a.....

3: ab.....

4: ababb.....

5: d.....

6: ababba ba ba a

7: a.....

8: ab a.....

9: a.....

10: a.....

11: ababb.....

12: a.....

13: a b a b b a b a b a a

STRING MATCHING

* A BETTER ALGORITHM.

* THE PREPROCESSING OF B IS THE ESSENCE OF
THE IMPROVED ALGORITHM.

* ALGORITHM FOR PREPROCESSING OF B (PSEUDOCODE)

COMPUTE-NEXT(B, m) { IF ASSIGNS VALUES TO THE }

next[1] \leftarrow -1 {ELEMENTS OF THE ARRAY B}

next[2] \leftarrow 0

for i \leftarrow 3 to m do

j \leftarrow next[i-1] + 1

while b[i-1] \neq b[j] and j > 0 do

j \leftarrow next[j] + 1

next[i] \leftarrow j

* EXAMPLE

i :	1	2	3	4	5	6	7	8	9	10	11
B :	a	b	a	b	b	a	b	a	b	a	a
next :	-1	0	0	1	2	0	1	2	3	4	3

STRING MATCHING

* MAIN ALGORITHM (PSEUDOCODE)

STRING-MATCH(A, n, B, m)

{IT RETURNS THE INDEX}

{START?}

$j \leftarrow 1$

$i \leftarrow 1$

START $\leftarrow 0$

while $START=0$ AND $i \leq n$ do

 if $B[j] = A[i]$ then

$j \leftarrow j + 1$

$i \leftarrow i + 1$

 else

$j \leftarrow next[j] + 1$

 if $j=0$ then

$j \leftarrow 1$

$i \leftarrow i + 1$

 if $j=m+1$ then $START=i-m$

* THIS ALGORITHM IS KNOWN AS THE KNUTH, MORRIS,
AND PRATT (KMP) ALGORITHM.

* THE KMP ALGORITHM RUNS IN TIME $O(m+n)$.

SEQUENCE COMPARISONS

* PROBLEM

FIND THE MINIMUM NUMBER OF EDIT STEPS (CHANGES) REQUIRED TO CHANGE ONE STRING INTO ANOTHER.

* MANY APPLICATIONS TO PROBLEMS IN MOLECULAR BIOLOGY.

* MAIN TECHNIQUE USED IS DYNAMIC PROGRAMMING.

* DYNAMIC PROGRAMMING IS USED IN CASES WHERE THE SOLUTION OF THE PROBLEM DEPENDS ON MANY SOLUTIONS OF SLIGHTLY SMALLER PROBLEMS.

* THE RUNNING TIME OF THE ALGORITHM THAT PROVIDES A SOLUTION IS $O(nm)$ (n IS THE SIZE OF STRING A AND m IS THE SIZE OF STRING B).

* THE DYNAMIC PROGRAMMING IS GENERALLY LESS EFFICIENT THAN THE DIVIDE-AND-CONQUER APPROACH.

FINDING A MAJORITY

* PROBLEM

GIVEN A SEQUENCE OF NUMBERS, FIND THE MAJORITY IN THE SEQUENCE OR DETERMINE THAT NONE EXISTS.

* THE MULTIPLICITY OF α IN THE SEQUENCE A IS THE NUMBER OF TIMES α APPEARS IN A .

* A NUMBER z IS A MAJORITY IN A IF ITS MULTIPLICITY IS GREATER THAN $n/2$ (n IS THE NUMBER OF THE ELEMENTS IN THE SEQUENCE).

* SORTING IS A STRAIGHT FORWARD SOLUTION.

* THE FOLLOWING OBSERVATION MAY HELP:

IF $a_i \neq a_j$ AND WE ELIMINATE BOTH OF THESE ELEMENTS FROM THE LIST, THEN THE MAJORITY IN THE ORIGINAL LIST REMAINS A MAJORITY IN THE NEW LIST.

FINDING A MAJORITY

* ALGORITHM (PSEUDOCODE)

MAJORITY (A, n) { IT RETURNS THE MAJORITY,
{ IN A IF IT EXISTS, OR -1 }
{ OTHERWISE. }

$c \leftarrow A[1]$

$m \leftarrow 1$

for $i \leftarrow 2$ to n do

if $m = 0$ then

$c \leftarrow A[i]$

$m \leftarrow 1$

else

if $c = A[i]$ then $m \leftarrow m + 1$

else $m \leftarrow m - 1$

if $m = 0$ then $\text{maj} \leftarrow -1$

else

$\text{COUNT} \leftarrow 0$

for $i \leftarrow 1$ to n do

if $A[i] = c$ then $\text{COUNT} \leftarrow \text{COUNT} + 1$

if $\text{COUNT} > n/2$ then $\text{maj} \leftarrow c$

else $\text{maj} \leftarrow -1$

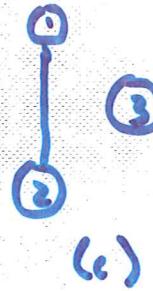
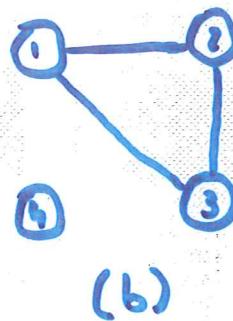
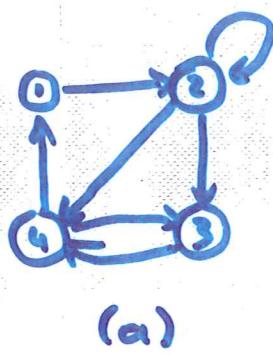
return maj

GRAPHS

GRAPHS

- * A GRAPH $G = (V, E)$ CONSISTS OF A SET V OF VERTICES (ALSO CALLED NODES), AND A SET E OF EDGES.

* EXAMPLES



- * A GRAPH CAN BE DIRECTED, OR UNDIRECTED.
- * THE EDGES IN A DIRECTED GRAPH ARE ORDERED PAIRS.
- * THE EDGES IN A UNDIRECTED GRAPH ARE UNORDERED PAIRS.
- * TREES ARE SIMPLE EXAMPLES OF GRAPHS.
- * A MULTIGRAPH IS A GRAPH WITH POSSIBLY SEVERAL EDGES BETWEEN THE SAME PAIR OF VERTICES.

GRAPHS

- * THE DEGREE $d(v)$ OF A VERTEX v IS THE NUMBER OF EDGES INCIDENT TO v .
- * IN A DIRECTED GRAPH, WE HAVE THE INDEGREE AND OUTDEGREE OF A VERTEX v .
- * THE INDEGREE OF A VERTEX v IS THE NUMBER OF EDGES FOR WHICH v IS THE HEAD.
- * THE OUTDEGREE OF A VERTEX v IS THE NUMBER OF EDGES FOR WHICH v IS THE TAIL.
- * IF (u, v) IS AN EDGE IN A GRAPH $G = (V, E)$, WE SAY THAT VERTEX v IS ADJACENT TO VERTEX u .
- * A PATH OF LENGTH k FROM A VERTEX u TO A VERTEX u' IN A GRAPH $G = (V, E)$ IS A SEQUENCE $\langle v_0, v_1, \dots, v_k \rangle$ OF VERTICES SUCH THAT $u = v_0$, $u' = v_k$, AND $(v_{i-1}, v_i) \in E$ FOR $i = 1, 2, \dots, k$.
- * A PATH IS CALLED SIMPLE IF EACH VERTEX APPEARS IN IT AT MOST ONCE.

GRAPHS

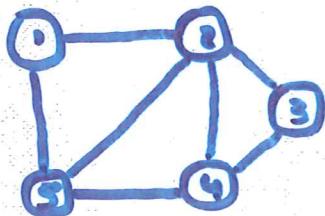
- * VERTEX u IS SAID TO BE REACHABLE FROM VERTEX v IF THERE IS A PATH FROM v TO u .
- * A CIRCUIT IS A PATH WHOSE FIRST AND LAST VERTICES ARE THE SAME.
- * A CIRCUIT IS CALLED SIMPLE (CYCLE) IF, EXCEPT FOR THE FIRST AND LAST VERTICES, NO VERTEX APPEARS MORE THAN ONCE.
- * A GRAPH IS CALLED CONNECTED IF (IN ITS UNDIRECTED FORM) THERE IS A PATH FROM ANY VERTEX TO ANY OTHER VERTEX.
- * A FOREST IS A GRAPH THAT DOES NOT CONTAIN A CYCLE (UNDIRECTED FORM OF THE GRAPH).
- * A SUBGRAPH OF THE GRAPH $G = (V, E)$ IS A GRAPH $G' = (V', E')$ SUCH THAT $V' \subseteq V$ AND $E' \subseteq E$.

GRAPHS

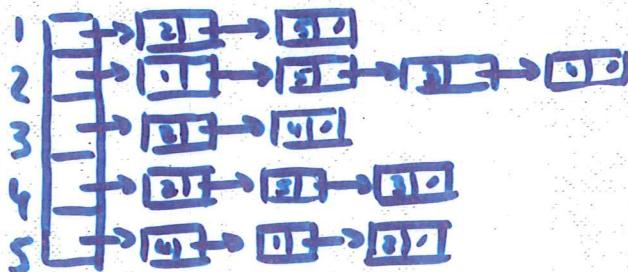
- * A CONNECTED, ACYCLIC, UNDIRECTED GRAPH IS CALLED (FREE) TREE.
- * A BIPARTITE GRAPH IS AN UNDIRECTED GRAPH $G = (V, E)$ IN WHICH V CAN BE PARTITIONED INTO TWO SETS V_1 AND V_2 SUCH THAT $(u, v) \in E$ IMPLIES EITHER $u \in V_1$ AND $v \in V_2$ OR $u \in V_2$ AND $v \in V_1$. IN OTHER WORDS, ALL EDGES GO BETWEEN THE TWO SETS V_1 AND V_2 .
- * A DIRECTED ACYCLIC GRAPH IS CALLED DAG.
- * A WEIGHTED GRAPH IS A GRAPH WITH WEIGHTS ASSOCIATED WITH THE EDGES.
- * A CONNECTED GRAPH WHOSE ALL VERTICES HAVE EVEN DEGREES IS CALLED EULERIAN GRAPH.
- * DIFFERENT REPRESENTATIONS OF THE DIRECTED AND UNDIRECTED GRAPHS.

GRAPHS

* TWO REPRESENTATIONS OF AN UNDIRECTED GRAPH.



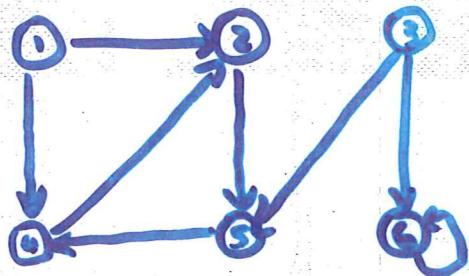
(A)



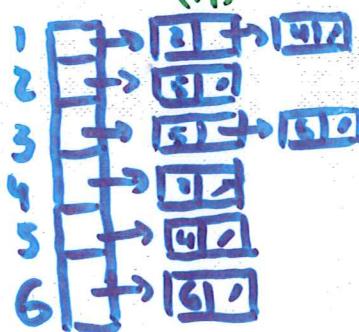
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(B)

* TWO REPRESENTATIONS OF A DIRECTED GRAPH.



(A)



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(B)

GRAPHS

* GRAPH TRAVERSALS

* DEPTH-FIRST SEARCH (DFS)

* BREADTH-FIRST SEARCH (BFS)

* BREADTH-FIRST SEARCH (BFS)

BFS(G, s)

{ s IS THE SOURCE.}

for each vertex $u \in V[G] - \{s\}$ do

color[u] \leftarrow WHITE

d[u] $\leftarrow \infty$

$\pi[u] \leftarrow \text{NIL}$

color[s] \leftarrow GRAY

d[s] $\leftarrow 0$

$\pi[s] \leftarrow \text{NIL}$

$Q \leftarrow \{s\}$

while $Q \neq \emptyset$ do

$u \leftarrow \text{head}[Q]$

for each $v \in \text{Adj}[u]$ do

if color[v] = WHITE then

color[v] \leftarrow GRAY

d[v] $\leftarrow d[u] + 1$

$\pi[v] \leftarrow u$

ENQUEUE(Q, v)

DEQUEUE(Q)

color[u] \leftarrow BLACK

{THE DISTANCE FROM}

{THE SOURCE s TO THE}

{VERTEX u IS STORED}

{IN $d[u]$.}

- * THE COLOR OF EACH VERTEX $u \in V$ IS STORED IN $\text{color}[u]$ AND THE PREDECESSOR IS STORED IN $\pi[u]$.

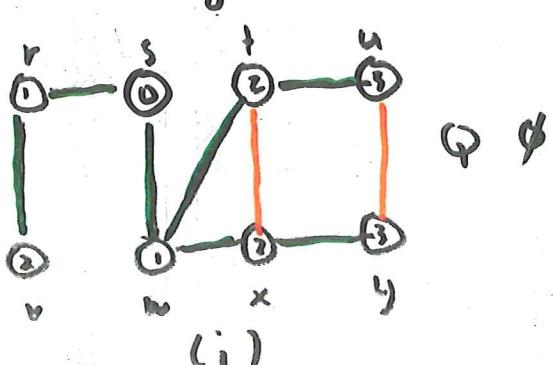
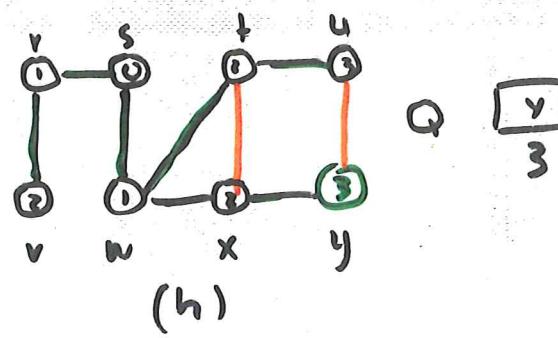
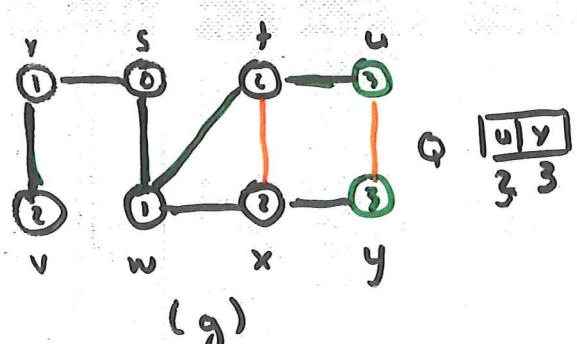
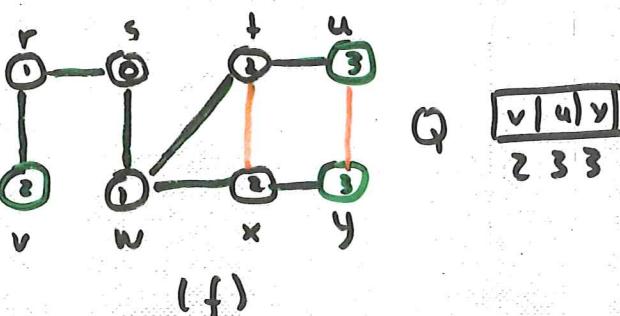
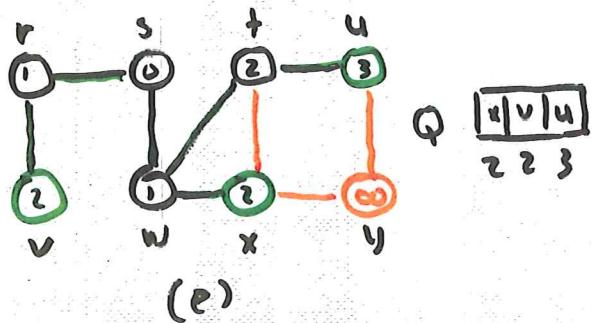
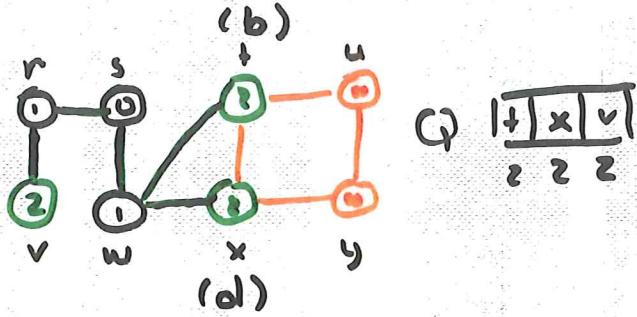
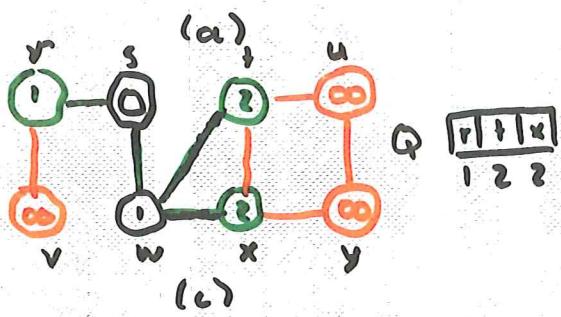
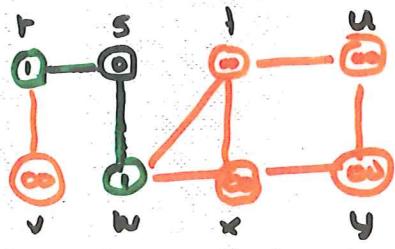
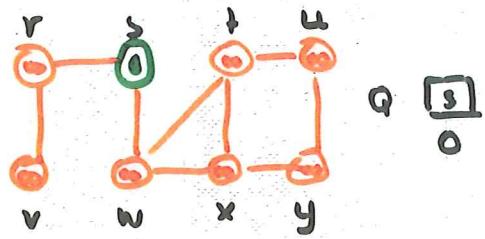
GRAPHS

* EXAMPLE OF BFS

- WHITE \rightarrow

- GRAY \rightarrow

- BLACK \rightarrow



GRAPHS

* DEPTH-FIRST SEARCH (DFS)

- ALGORITHM (PSEUDOCODE)

DFS(G)

for each vertex $u \in V[G]$ do

color[u] \leftarrow WHITE

$\pi[u] \leftarrow \text{NIL}$

time $\leftarrow 0$

for each vertex $u \in V[G]$ do

if color[u] = WHITE then

DFS-VISIT(u)

DFS-VISIT(u)

color[u] \leftarrow GRAY {WHITE VERTEX u HAS JUST

$d[u] \leftarrow \text{time} \leftarrow \text{time} + 1$ {BEEN DISCOVERED.}

for each $v \in \text{Adj}[u]$ do {EXPLORE EDGE (u, v) }

if color[v] = WHITE then

$\pi[v] \leftarrow u$

DFS-VISIT(v)

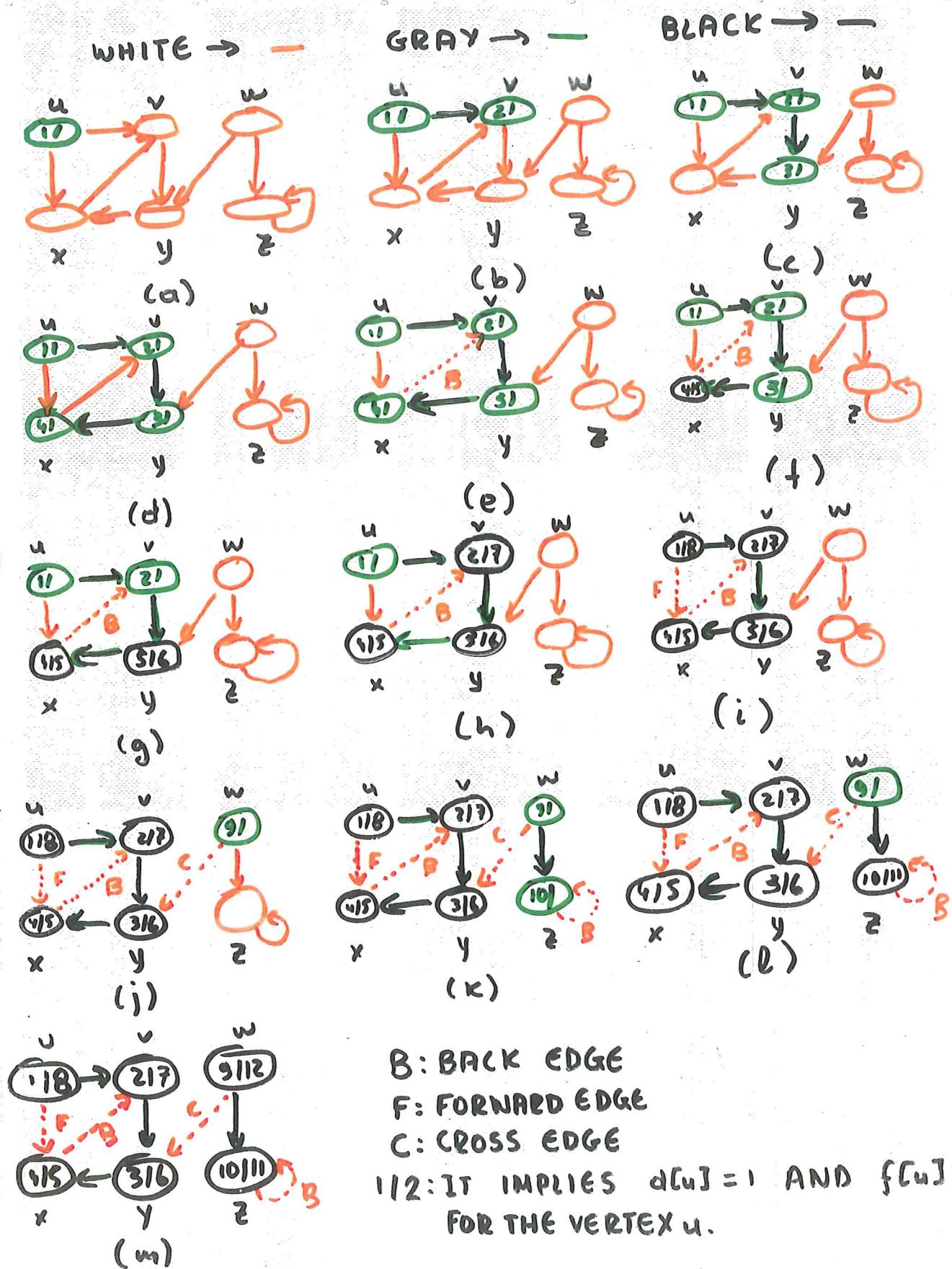
color[u] \leftarrow BLACK {BLACKEN u ; IT IS FINISHED}

$f[u] \leftarrow \text{time} \leftarrow \text{time} + 1$

- * THE TIME OF DISCOVERY OF THE VERTEX u IS STORED IN THE VARIABLE $d[u]$. THE END OF THE PROCESSING OF THE VERTEX u IS STORED IN THE VARIABLE $f[u]$ ($d[u] < f[u]$).

GRAPHS

* EXAMPLE OF THE DFS ALGORITHM



B: BACK EDGE

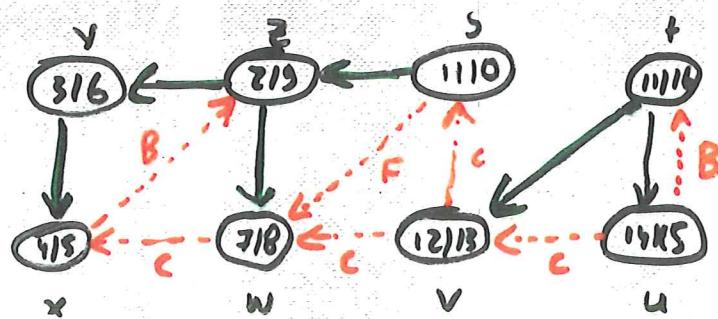
F: FORWARD EDGE

C: CROSS EDGE

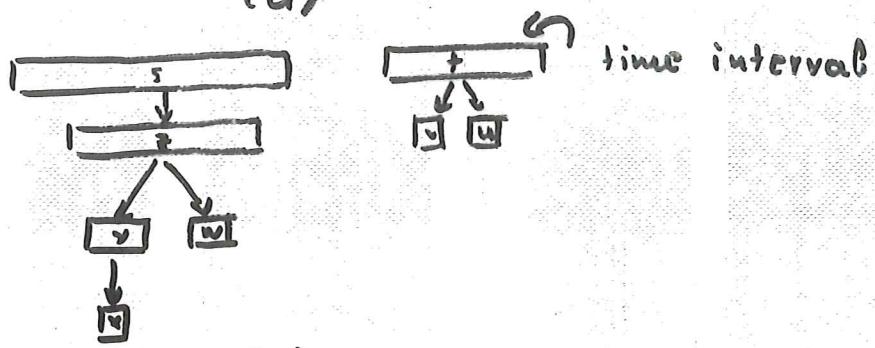
112: IT IMPLIES $d[u] = 1$ AND $f[u] = 2$
FOR THE VERTEX u .

GRAPHS

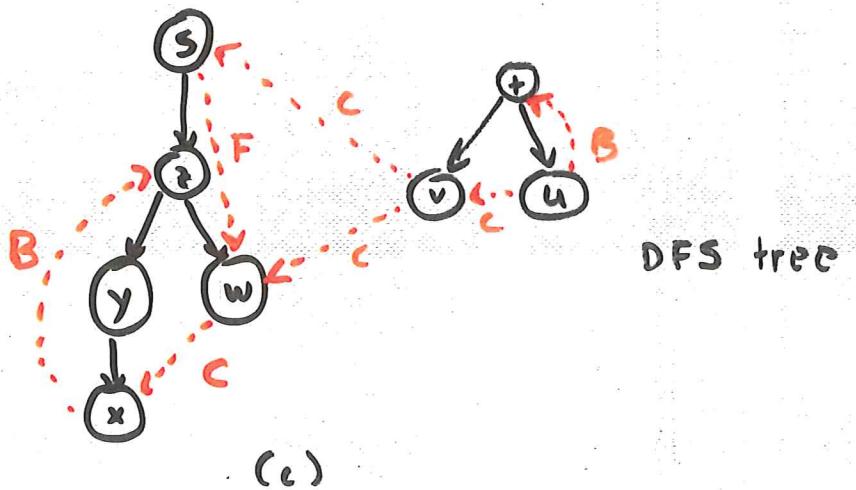
- * ANOTHER EXAMPLE OF DFS



(a)



(b)



DFS tree

- * THE RUNNING TIME OF DFS IS $\Theta(v+e)$.

- * THE DFS ALGORITHM CAN BE MODIFIED TO CLASSIFY EDGES AS IT ENCOUNTERS THEM.

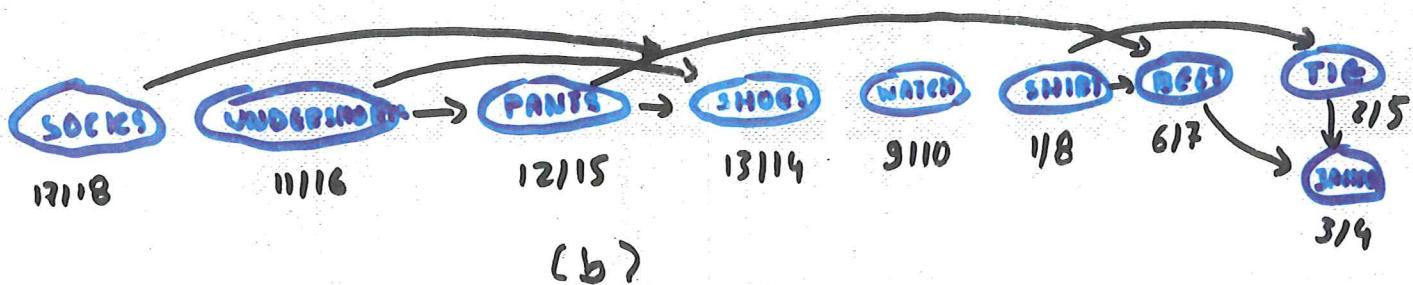
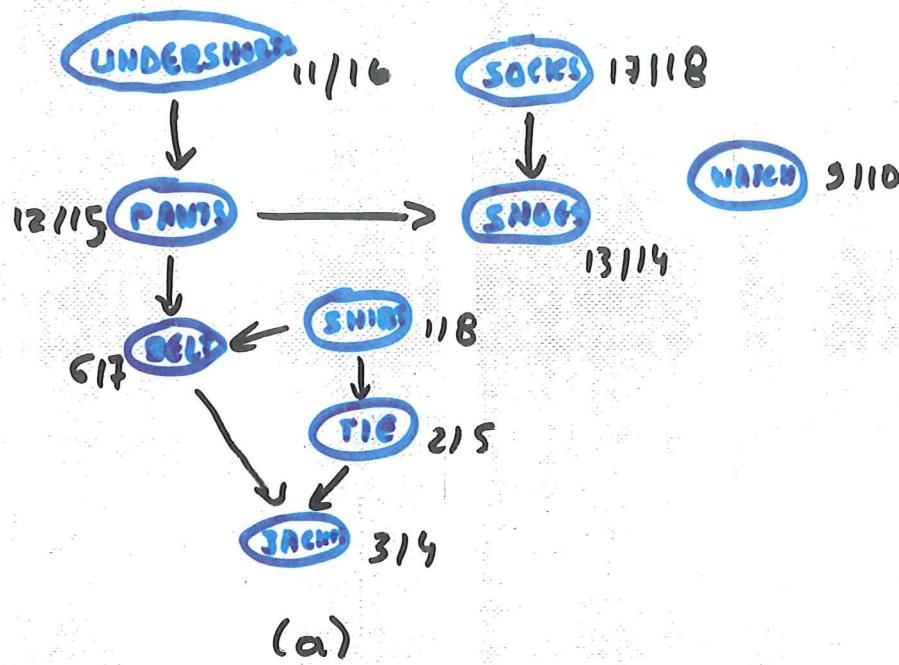
GRAPHS

- * DFS CAN BE USED TO FIND THE CONNECTED COMPONENTS OF A GRAPH.
- * DFS CAN BE USED TO FIND CYCLES IN A GRAPH.
- * DFS CAN BE USED FOR TOPOLOGICAL SORT.
- * TOPOLOGICAL SORT OF A DAG $G = (V, E)$ IS A LINEAR ORDERING OF ALL ITS VERTICES SUCH THAT IF G CONTAINS AN EDGE (u, v) , THEN u APPEARS BEFORE v IN THE ORDERING.
- * TOPOLOGICAL SORT (PSEUDOCODE)
 - 1 CALL DFS(G) TO COMPUTE FINISHING TIMES $f[v]$ FOR EACH v .
 - 2 AS EACH VERTEX IS FINISHED, INSERT IT ONTO THE FRONT OF A LINKED LIST.
 - 3 RETURN THE LINKED LIST OF VERTICES.

GRAPHS

* EXAMPLE OF TOPOLOGICAL SORT

PROFESSOR X TOPOLOGICALLY SORTS HIS CLOTHING WHEN GETTING DRESSED



* WE CAN PERFORM TOPOLOGICAL SORT IN TIME $\Theta(V+E)$,

SINCE DEPTH-FIRST SEARCH TAKES $\Theta(V+E)$ TIME AND

IT TAKES $O(1)$ TIME TO INSERT EACH OF THE $|V|$

VERTICES ONTO THE FRONT OF THE LINKED LIST.

GRAPHS

* SINGLE-SOURCE SHORTEST PATHS

- PROBLEM

GIVEN A WEIGHTED, DIRECTED GRAPH $G = (V, E)$
AND A VERTEX s , FIND SHORTEST PATHS FROM
 s TO ALL OTHER VERTICES OF G .

* SIMILAR PROBLEMS

- SINGLE-DESTINATION SHORTEST-PATHS PROBLEM
- SINGLE-PAIR SHORTEST-PATH PROBLEM
- ALL-PAIRS SHORTEST-PATHS PROBLEM

* SEVERAL ALGORITHMS

- DIJKSTRA'S ALGORITHM
- BELLMAN-FORD ALGORITHM

* NEGATIVE-WEIGHT EDGES ARE POSSIBLE.

- * THE USE OF RELAXATION IS POPULAR IN THESE
ALGORITHMS.

GRAPHS

* RELAXATION (PSEUDOCODE)

RELAX(u, v, w)

if $d[v] > d[u] + w(u, v)$ then

$d[v] \leftarrow d[u] + w(u, v)$

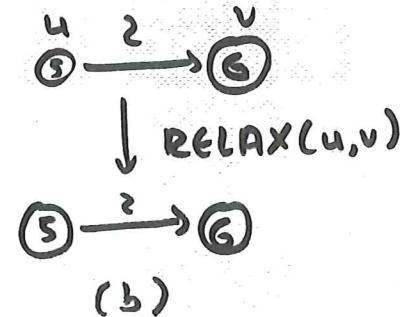
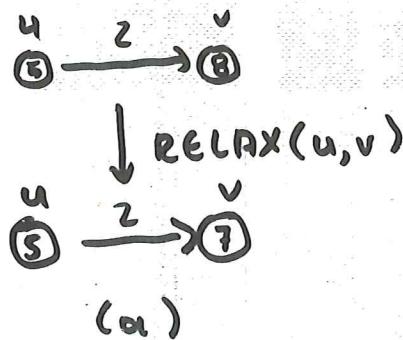
$\pi[v] \leftarrow u$

$\pi[v]$: v 's PREDECESSOR FIELD

$d[v]$: SHORTEST-PATH ESTIMATE

$w(u, v)$: WEIGHT

* RELAXATION OF AN EDGE (u, v)



* INITIALIZATION

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in V[G]$ do

$d[v] \leftarrow \infty$

$\pi[v] \leftarrow \text{NIL}$

$d[s] \leftarrow 0$

GRAPHS

* DIJKSTRA'S ALGORITHM (PSEUDOCODE)

DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

$S \leftarrow \emptyset$

{ S' IS A SET OF VERTICES WHOSE}

$Q \leftarrow V[G]$

{FINAL SHORTEST-PATH WEIGHTS}

{FROM THE SOURCE s HAVE ALREADY}

{BEEN DETERMINED.}

while $Q \neq \emptyset$ do

$u \leftarrow \text{EXTRACT-MIN}(Q)$

$S' \leftarrow S' \cup \{u\}$

for each vertex $v \in \text{Adj}[u]$ do

$\text{RELAX}(u, v, w)$

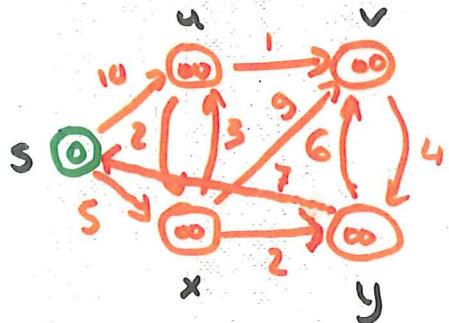
Q: PRIORITY QUEUE THAT CONTAINS ALL THE
VERTICES IN $V - S'$, KEYED BY THEIR d VALUES.

* DIJKSTRA'S ALGORITHM USES A GREEDY STRATEGY.

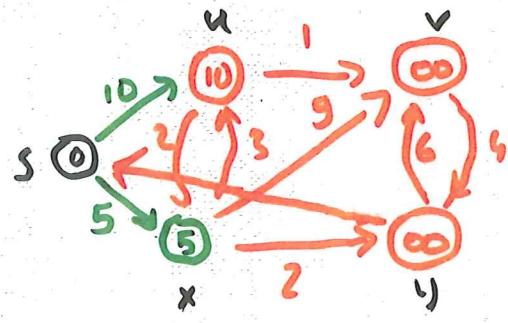
* THE RUNNING TIME IS $O(V^2 + E) = O(V^2)$.

GRAPHS

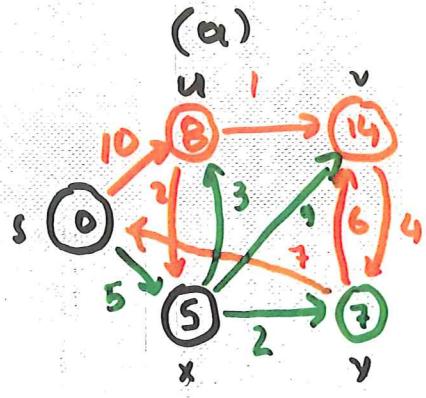
* EXAMPLE OF THE DIJKSTRA'S ALGORITHM



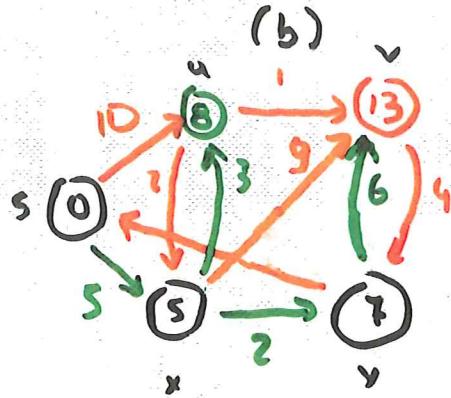
(a)



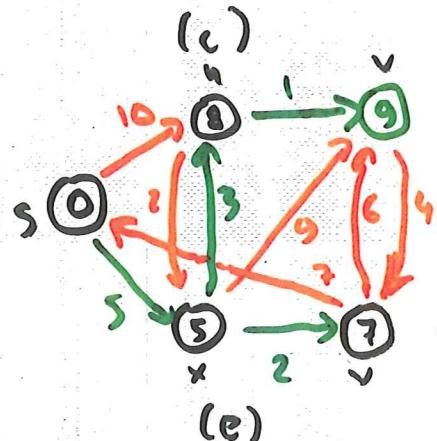
(b)



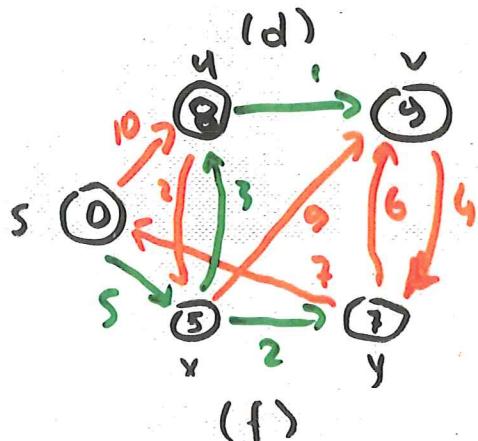
(c)



(d)



(e)



(f)

- IF EDGE (u, v) IS GREEN, THEN $\pi[v] = u$.
- BLACK VERTICES ARE IN THE SET S !

GRAPHS

* BELLMAN-FORD ALGORITHM (PSEUDO CODE)

BELLMAN-FORD(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

for $i \leftarrow 1$ to $|V[G]| - 1$ do

 for each edge $(u, v) \in E[G]$ do

 RELAX(u, v, w)

 for each edge $(u, v) \in E[G]$ do

 if $d[v] > d[u] + w(u, v)$ then

 return FALSE

 return TRUE

* THE ALGORITHM RETURNS TRUE IF AND ONLY IF

THE GRAPH CONTAINS NO NEGATIVE-WEIGHT CYCLES

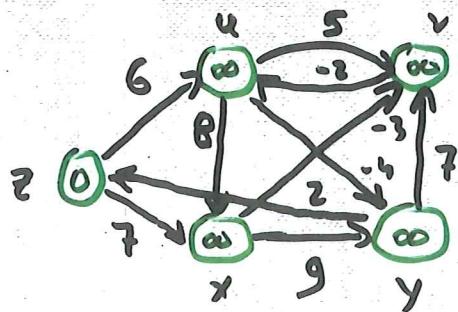
THAT ARE REACHABLE FROM THE SOURCE.

* THE BELLMAN-FORD ALGORITHM RUNS IN TIME

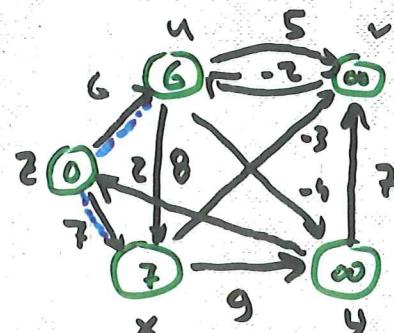
$O(VE)$.

GRAPHS

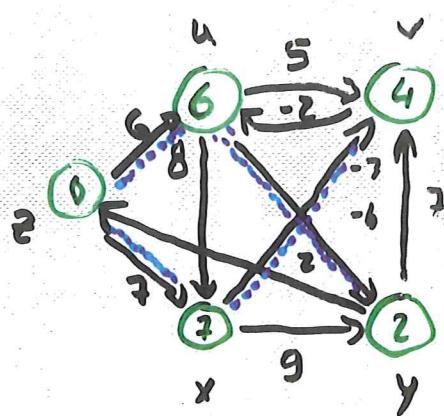
* EXAMPLE OF THE BELLMAN-FORD ALGORITHM.



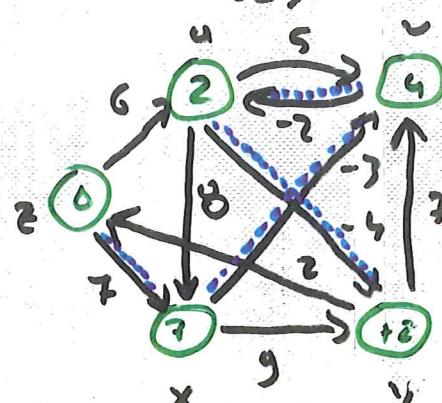
(a)



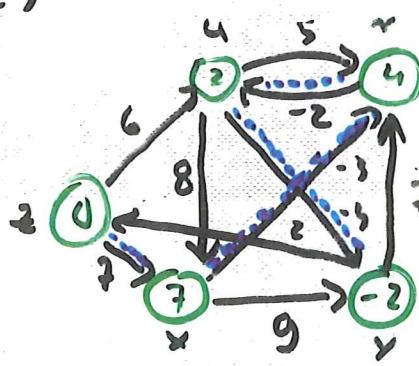
(b)



(c)



(d)



(e)

* EACH PASS RELAXES THE EDGES IN LEXICOGRAPHIC ORDER.