

Data Structures and Algorithms
INFO 6205, Wed
Homework 10
Due: November 30, 2019

Put all your java, compiled class files and documentation files into a zip file named Homework10.zip and submit it via the dropbox on the blackboard before the END of due date. Put your name on all .java files. There will be a short Quiz on this homework.

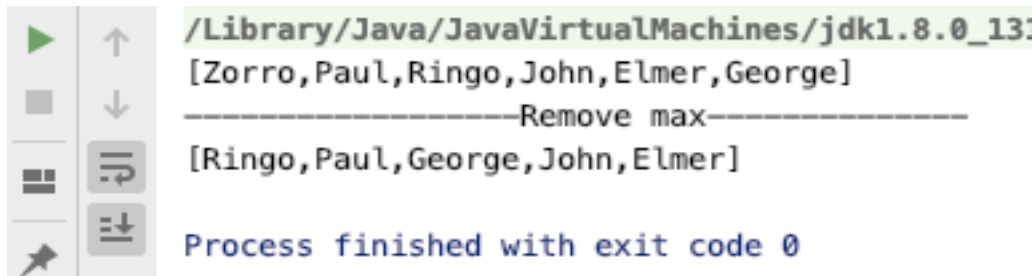
1. A MinHeap is a complete binary tree where the minimum-valued element is stored at the root node and every node is less than or equal to both of its child nodes.

Note: Java code is provided for MaxHeap algorithm in attached file.

- a) compile and test MaxHeap code.
- b) modify MaxHeap code to MinHeap, and then compile and test the code.

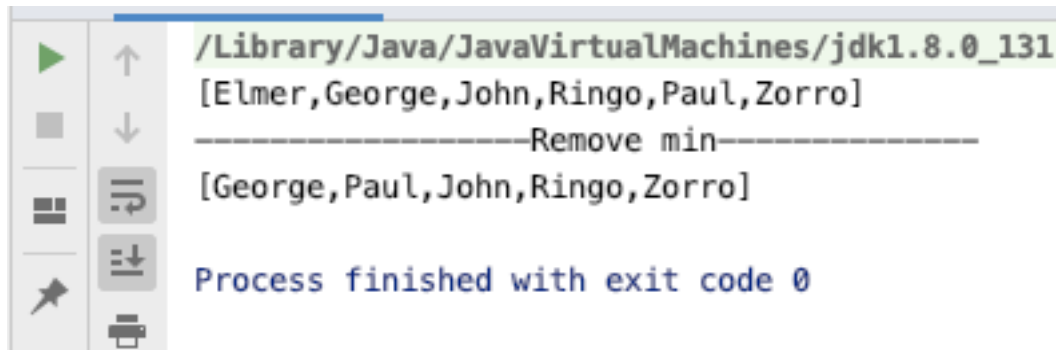
Answer:

a) The code is in H10_1a.java



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_131
[Zorro,Paul,Ringo,John,Elmer,George]
-----Remove max-----
[Ringo,Paul,George,John,Elmer]
Process finished with exit code 0
```

b) The code is in H10_1b.java



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_131
[Elmer,George,John,Ringo,Paul,Zorro]
-----Remove min-----
[George,Paul,John,Ringo,Zorro]
Process finished with exit code 0
```

2. Consider the following Text and Pattern

Text: ABC ADC BAB ABC DABCDABDE

Pattern: BCD

- a) Apply Brute-Force substring search algorithm to scan Pattern in Text string. Show step-by-step of the algorithm. Write Java code for the algorithm for input data. What is time complexity?
- b) Apply Robin-Karp substring search algorithm to scan pattern in the

text string. Show step-by-step of algorithm. Write Java code for the algorithm for input data. What is time complexity?
c) What is the difference between the two time complexity?

Answer:

a) Algorithm:

```
For i=0, i<= lengthOfText - lengthOfPattern, i++
  for j=0, j<lengthOfPattern, j++
    if text.charAt(i+j) != pattern.charAt(j)
      break
  if( j == lengthOfPattern ) print the result.
```

H2

a)

①

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ x

B	C	D
---	---	---

②

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ v ↑ v ↑ x

B	C	D
---	---	---

③

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ x

B	C	D
---	---	---

④

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ x

B	C	D
---	---	---

⑤

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ x

B	C	D
---	---	---

b)

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ x

B	C	D
---	---	---

7

A B C A D C B A B A B C D A B C D A B D E

↑v ↑x
B C D

8

A B C A D C B A B A B C D A B C D A B D E

↑x
B C D

9

A B C A D C B A B A B C D A B C D A B D E

↑v ↑x
B C D

10

A B C A D C B A B A B C D A B C D A B D E

↑x
B C D

11

A B C A D C B A B A B C D A B C D A B D E

↑v ↑v ↑v
B C D

print (10)

12

A B C A D C B A B A B C D A B C D A B D E

↑x
B C D

13

A B C A D C B A B A B C D A B C D A B D E

↑x
B C D

14

A B C A D C B A B A B C D A B C D A B D E

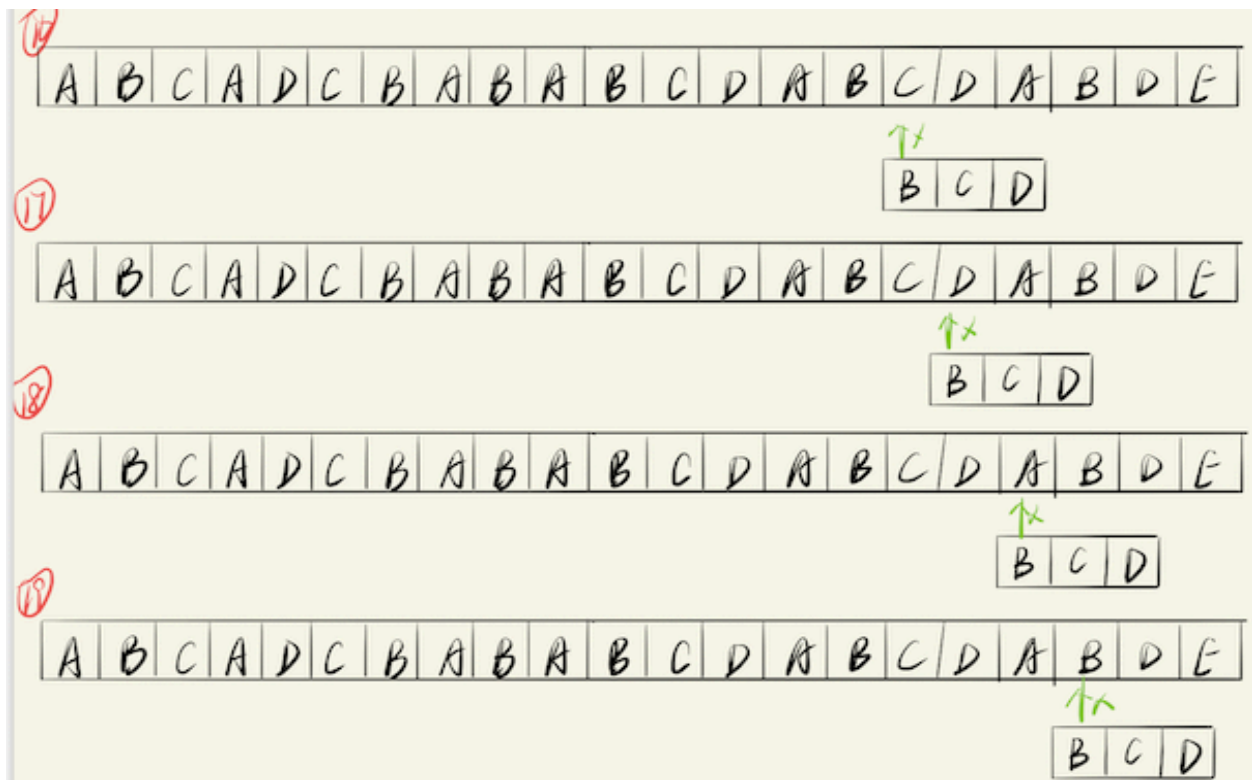
↑x
B C D

15

A B C A D C B A B A B C D A B C D A B D E

print (14)

↑v ↑v ↑v
B C D



The code is in H10_2a.java

```

/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home
---The result of Brute-Force substring search algorithm---
Pattern found at index 10
Pattern found at index 14

Process finished with exit code 0

```

Time complexity of Brute-force substring search: $O(nm)$

b) Algorithm:

calculate the hash value of pattern. M is length of pattern

calculate the hash value of the first M letter of text in order.

if hash value of pattern = hash value of the first M letter of text

for $j=0, j < M, j++$

if ($\text{txt.charAt}(i+j) \neq \text{pat.charAt}(j)$)

break;

if ($j == M$) print the result.

H2 b)

A	B	C	A	D	C	B	A	B	A	B	C	D	A	B	C	D	A	B	D	E
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

B	C	D
---	---	---

"BCD" = 0

①

"ABC" = 59 ≠

② "BCA" = 98 ≠

③ "CAD" = 81 ≠

④ "ADC" = 66 ≠

⑤ "DCB" = 73 ≠

⑥ "CBA" = 81 ≠

⑦ "BAB" = 92 ≠

⑧ "ABA" = 87 ≠

⑨ "BAB" = 92 ≠

⑩ "ABC" = 59 ≠

⑪ "BCD" = 0 → 0=0 "BCD" = "BCD"

print (pattern found at index 10)

⑫ "CDA" = 38 ≠

⑬ "DAB" = 66 ≠

⑭ "ABC" = 59 ≠

⑮ "BCD" = 0

0=0 "BCD" = "BCD" print (---- at index 14)

⑯ "CDA" = 38 ≠

⑰ "DAB" = 66 ≠

⑱ "ABD" = 60 ≠

⑲ "BDE" = 55 ≠

The code is in H10_2a.

```

/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/
---The result of Robin-Karp substring search algorithm---
Pattern found at index 10
Pattern found at index 14

Process finished with exit code 0

```

Time complexity of Robin Karp substring search: $O(mn)$

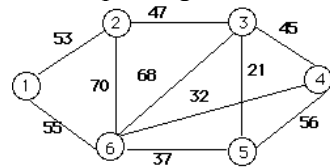
c)

Time complexity of Brute-force substring search: $O(nm)$

Time complexity of Robin Karp substring search: $O(mn)$

$O(mn)$ is much faster than $O(nm)$ in brute force. The reason is that comparing Pattern letters with Text letters is expensive operation. With Robin-Karp method you only do comparisons if $\text{hash}(\text{pattern}) == \text{hash}(\text{substring})$, otherwise you Don't do any comparison.

3. Solve the Minimum Spanning Tree for the following Graph,



- Kruskal's algorithm step-by step
- Prim's Algorithm step-by-step
- Write Java code for (a) and (b) and compile
- Compare space and time complexity between two algorithms
- Is following code Prim's or Kruskal's algorithm, explain?

1. Make a queue (Q) with all the vertices of G (V);
2. For each member of Q set the priority to INFINITY;
3. Only for the starting vertex (s) set the priority to 0;
4. The parent of (s) should be NULL;
5. While Q isn't empty
 6. Get the minimum from Q – let's say (u); (priority queue);
 7. For each adjacent vertex to (v) to (u)
 8. If (v) is in Q and weight of (u, v) < priority of (v) then
 9. The parent of (v) is set to be (u)
 10. The priority of (v) is the weight of (u, v)

Answer:

a)

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is no formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

A3 a)

3, 5

21

4, 6

32

3, 6

37

3, 4

45

2, 3

47

1, 2

53

1, 6

55

4, 5

56

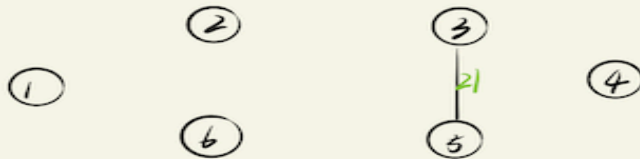
3, 6

68

2, 6

70

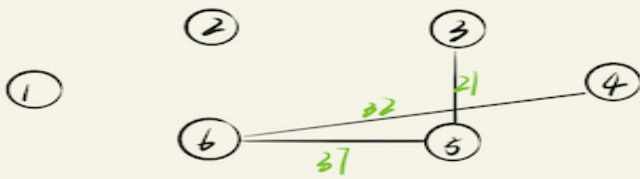
①



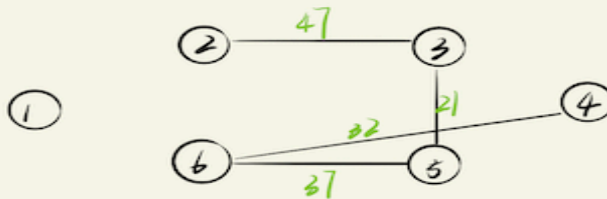
②



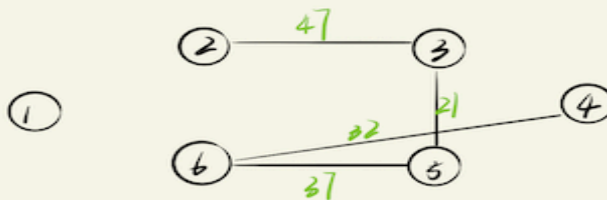
③



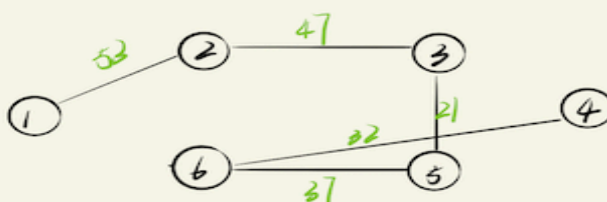
④



⑤



⑥

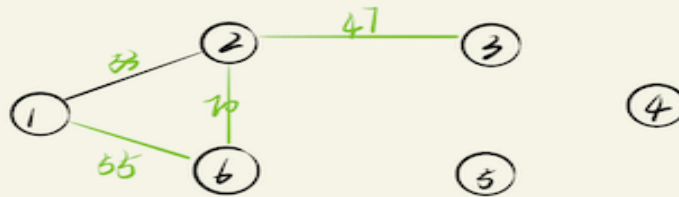


b)

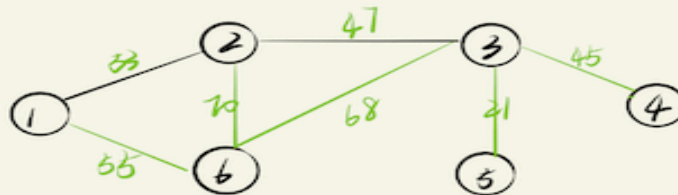
- 1) Create a set `mstSet` that keeps track of vertices already included in MST.
- 2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
- 3) While `mstSet` doesn't include all vertices
 -a) Pick a vertex `u` which is not there in `mstSet` and has minimum key value.
 -b) Include `u` to `mstSet`.
 -c) Update key value of all adjacent vertices of `u`. To update the key values, iterate through all adjacent vertices. For every adjacent vertex `v`, if weight of edge `u-v` is less than the previous key value of `v`, update the key value as weight of `u-v`

H3 b)

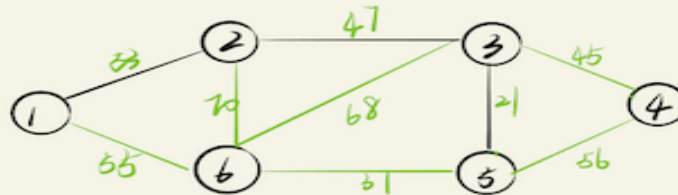
①



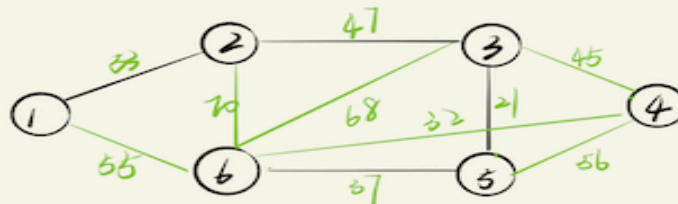
②



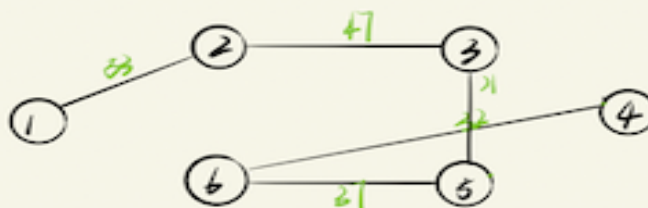
③



④



⑤



c)

The code for a) is H10_3a.java

```

/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home
Following are the edges in the constructed MST
3 — 5 == 21
4 — 6 == 32
5 — 6 == 37
2 — 3 == 47
1 — 2 == 53

```

The code for b) is H10_3b.java

```

/Library/Java/JavaVirtualMachines/jdk1.8.0_131
Edge    Weight
1 - 2   53
2 - 3   47
6 - 4   32
3 - 5   21
5 - 6   37

Process finished with exit code 0

```

d)

Kruskal's algorithm: Space $O(E+V)$, Time $O(E \log V)$

It traverse the edge only once and based on cycle it will either reject it or accept it.

Prim:

Data Structure of Graph **Ordering** **Time Complexity** **Space Complexity**

Adjacency Matrix Searching $O(V^2)$ $O(E+V)$

Adjacency List Binary Heap $O(E \log V)$ $O(E+V)$

Adjacency List Priority Queue with decrease key $O(E^2 \log V)$ $O(E+V)$

Adjacency List Priority Queue without decrease key $O(E \log V)$ $O(E+V)$

each vertex is inserted in the priority queue only once and insertion in priority queue take logarithmic time.

e) It is Prim's Algorithm.

Make a queue to store all the vertices and initialize them as INFINITY.

Only set the starting vertices as 0 and its parent as Null.

While queue is not empty, get the minimum edge.

we start from a vertex, we push all the edges starting from this node to a priority queue and we chose the lightest edge. Going to the next node connected by this edge we append to the queue all the edges that aren't in the queue.

4. Consider Knapsack problem in this article:

<https://www.radford.edu/~nokie/classes/360/dp-knapsack.html>

a) Find solutions for $m=21$

b) How does it uses Dynamic Programming?

Answer:

a)

Name	A	B	C	D	E
Size	3	4	7	8	9
Value	4	5	10	11	12

i	c	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
init	B(c)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	L(c)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	B(c)	0	0	4	4	4	8	8	8	12	12	12	16	16	16	20	20	20	24	24	24	28
	L(c)	-	-	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
2	B(c)	0	0	4	5	5	8	9	10	12	13	14	16	17	18	20	21	22	24	25	26	28
	L(c)	-	-	A	B	B	A	B	B	A	B	B	A	B	B	A	B	B	A	B	B	A
3	B(c)	0	0	4	5	5	8	10	10	12	14	15	16	18	20	20	22	24	25	26	28	30
	L(c)	-	-	A	B	B	A	C	B	A	C	C	A	C	C	A	C	C	A	C	C	C
4	B(c)	0	0	4	5	5	8	10	11	12	14	15	16	18	20	21	22	24	25	26	28	30
	L(c)	-	-	A	B	B	A	C	D	A	C	C	A	C	C	D	C	C	A	C	C	C
5	B(c)	0	0	4	5	5	8	10	11	12	14	15	16	18	20	21	22	24	25	26	28	30
	L(c)	-	-	A	B	B	A	C	D	A	C	C	A	C	C	D	C	C	A	C	C	C

Find some solutions:

AAAAAAA:28

CCC:30

b)

Knapsack problem has both properties (Overlapping Subproblems Property and Optimal Substructure Property) of a dynamic programming problem.

Overlapping Subproblems Property: Dynamic Programming is mainly used when solutions of same subproblems are needed again and again. In dynamic programming, computed solutions to subproblems are stored in a table so that these don't have to be recomputed.

Optimal Substructure Property: A given problems has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems.

5. Consider Knuth-Morris-Paratt substring search Algorithm:

<https://www.ics.uci.edu/~eppstein/161/960227.html>

a) How the algorithm works for Text="banananobano", and Pattern="nano"

b) How Knuth-Morris-Paratt is different from Brute-force algorithm?

Answer:

a) algorithm kmp_search:

input:

an array of characters, S (the text to be searched)

an array of characters, W (the word sought)

output:

an array of integers, P (positions in S at which W is found)

an integer, nP (number of positions)

define variables:

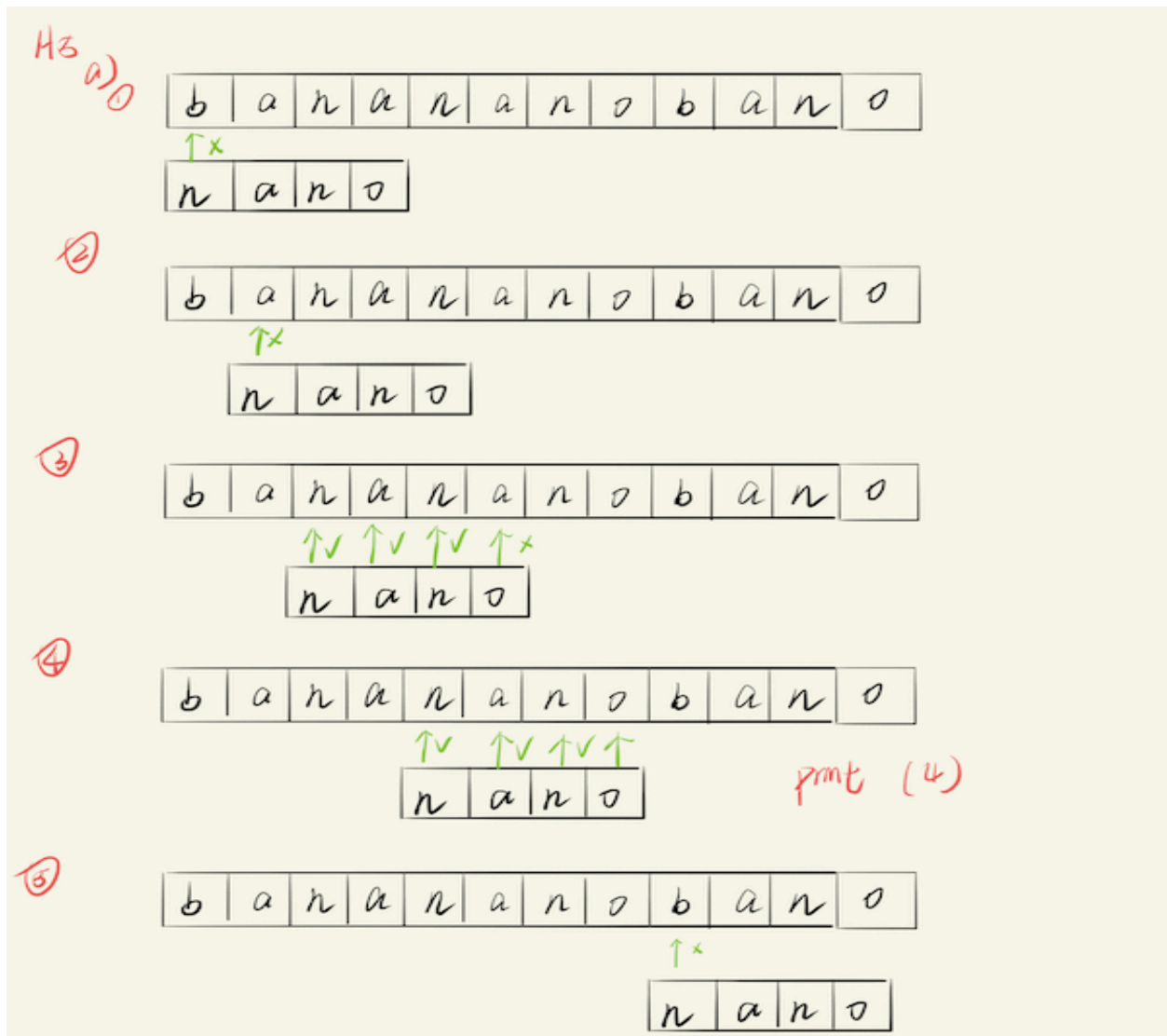
an integer, j \leftarrow 0 (the position of the current character in S)

an integer, $k \leftarrow 0$ (the position of the current character in W)
 an array of integers, T (the table, computed elsewhere)

let $nP \leftarrow 0$

```

while j < length(S) do
  if W[k] = S[j] then
    let j  $\leftarrow$  j + 1
    let k  $\leftarrow$  k + 1
    if k = length(W) then
      (occurrence found, if only first occurrence is needed,  $m \leftarrow j - k$  may be returned here)
      let P[nP]  $\leftarrow$  j - k, nP  $\leftarrow$  nP + 1
      let k  $\leftarrow$  T[k] (T[length(W)] can't be -1)
    else
      let k  $\leftarrow$  T[k]
  if k < 0 then
    let j  $\leftarrow$  j + 1
    let k  $\leftarrow$  k + 1
  
```



b)

The difference is that KMP makes use of previous match information that the Brute-force algorithm does not.

Time complexity is different.

Brute-force algorithm: $O(nm)$

KMP: $O(n+m)$

6. Read the following reference:

<https://www.hackerearth.com/practice/algorithms/greedy/basics-of-greedy-algorithms/tutorial/>

a) Solve the greedy problem for $A = \{8, 7, 6, 5, 4, 3, 2, 1\}$, $T = 15$

b) Write Java code for greedy algorithm with input data in (a)

c) Consider the Scheduling program, how does that work?

d) How (c) is different in (a)?

Answer:

a)

Handwritten solution for the greedy scheduling problem. It shows the selection of tasks based on their duration (T) and the number of things (NT) to be completed. The tasks are selected in descending order of their duration, and the total duration (CT) and number of things (NT) are updated at each step.

H6
a) $A = \{1, 2, 3, 4, 5, 6, 7, 8\}$ $T = 15$

current time \rightarrow CT
number of thing \rightarrow NT

① $CT = 1$
 $NT = 1$

② $CT = 1 + 2 = 3$
 $NT = 2$

③ $CT = 3 + 3 = 6$
 $NT = 3$

④ $CT = 6 + 4 = 10$
 $NT = 4$

⑤ $CT = 10 + 5 = 15$
 $NT = 5$

\rightarrow Answer = 5

b) The code is in H10_6c.java



c)

Integer N for the number of jobs you want to complete

Lists P: Priority (or weight)

List T: Time that is required to complete a task

To understand what criteria to optimize, you must determine the total time that is required to complete each task.

$C(j) = T[1] + T[2] + \dots + T[j]$ where $1 \leq j \leq N$

While there are many objective functions in the "Scheduling" problem, your objective function F is the weighted sum of the completion times.

$F = P[1] * C(1) + P[2] * C(2) + \dots + P[N] * C(N)$

d)

Difference:

The scheduling problem has one more parameter called Priority (or weight) than a. So you need to calculate weighted sum of the completion times.

7. Read this article on Genetic Algorithm:

<https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>

- a) What are the steps of GA described in the article?
- b) Read the example Java code as how it relates to steps in (a)
- c) Compile and run the code, explain the result.

a)

Five phases are considered in a genetic algorithm.

Initial population

Fitness function

Selection

Crossover

Mutation

b)

START

Generate the initial population

Compute fitness

REPEAT

Selection

Crossover

Mutation

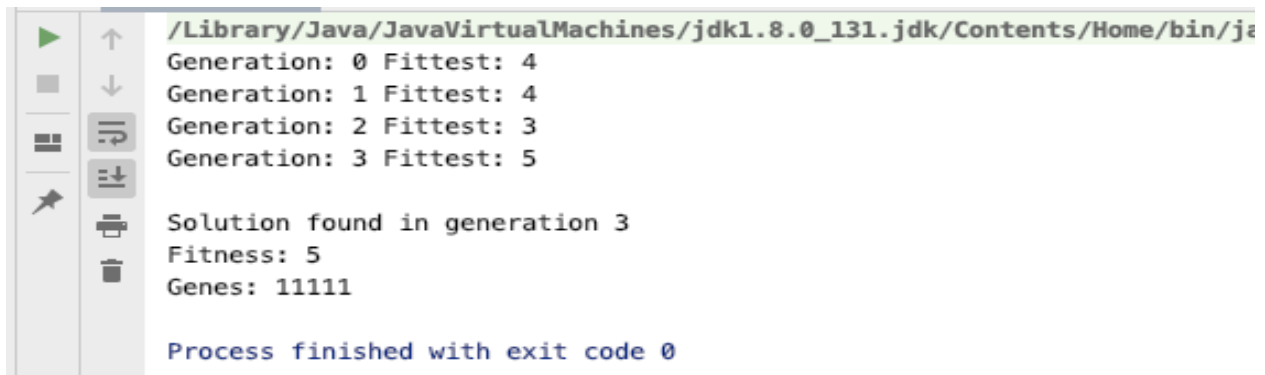
Compute fitness

UNTIL population has converged

STOP

c)

The code is in H10_7c.java



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home/bin/java
Generation: 0 Fittest: 4
Generation: 1 Fittest: 4
Generation: 2 Fittest: 3
Generation: 3 Fittest: 5

Solution found in generation 3
Fitness: 5
Genes: 11111

Process finished with exit code 0
```

Given a set of 5 genes, each gene can hold one of the binary values 0 and 1.

The fitness value is calculated as the number of 1s present in the genome. If there are five 1s, then it is having maximum fitness. If there are no 1s, then it has the minimum fitness.

This genetic algorithm tries to maximize the fitness function to provide a population consisting of the fittest individual, i.e. individuals with five 1s.

The fitness of generation 0 is 4.

The fitness of generation 1 is 4.

The fitness of generation 2 is 3.

The fitness of generation 3 is 5.

Already find the maximum fitness, output “solution found in generation 3”

“Fitness: 5”

“Genes: 11111”