1 Exercise 1

1.

Character	Frequency	Codeword
A	6	110
B	4	1110
C	9	10
D	4	1111
E	19	0

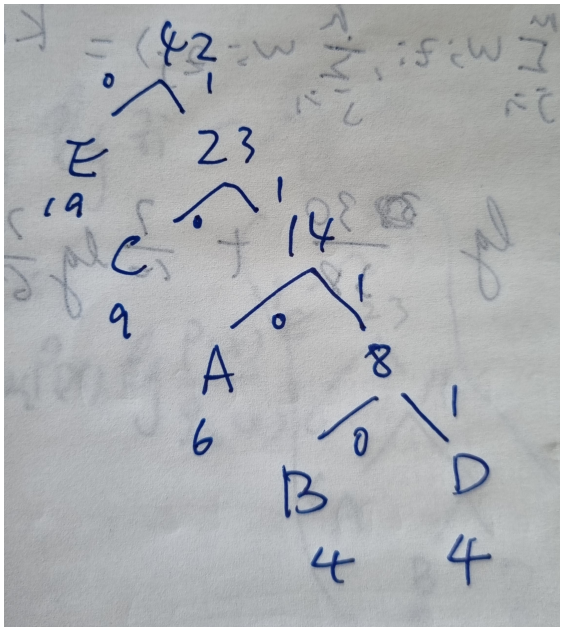


Figure 1: Hoffman coding

Thus, 011010101100 represents EACCAE.

2
2.1 20, 70, 30, 0
2.2 4, 1, 1, N
2.2 20, 70, 30, 0
2.4 4, 1, 1, N

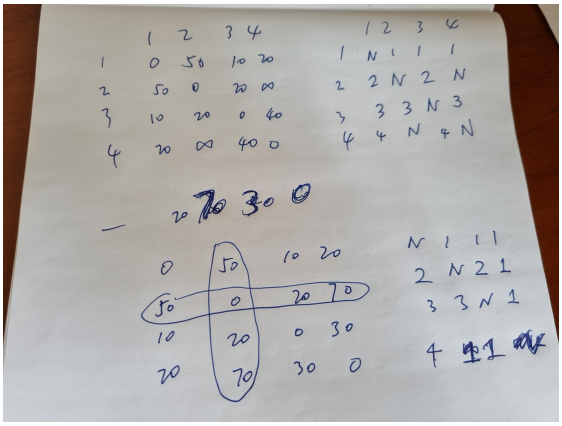


Figure 2: Floyd-Warshall

3

3.1

☐ **a)** Linear time: $O(n)$. ☐ **b)** Quadratic time: $O(n^2)$.
☐ **c)** Logarithmic time: $O(\log n)$. ☒ d) Exponential time: $O(a^n)$.

3.2

☐ **a)** Linear time: $O(n)$. ☐ **b)** Quadratic time: $O(n^2)$.
☐ **c)** Logarithmic time: $O(\log n)$. ☒ d) Exponential time: $O(a^n)$.

3.3

☒ a) Linear time: $O(n)$. ☐ **b)** Quadratic time: $O(n^2)$.
☐ **c)** Logarithmic time: $O(\log n)$. ☐ **c)** Exponential time: $O(a^n)$.

4.

4.1 *PUSH*: 1, *POP*: 1, *EXPAND*: n

4.2

Operation	Actual cost	Amorized cost
$PUSH(S, x)$	1	3
$POP(S)$	1	0
$EXPAND(S)$	n	0

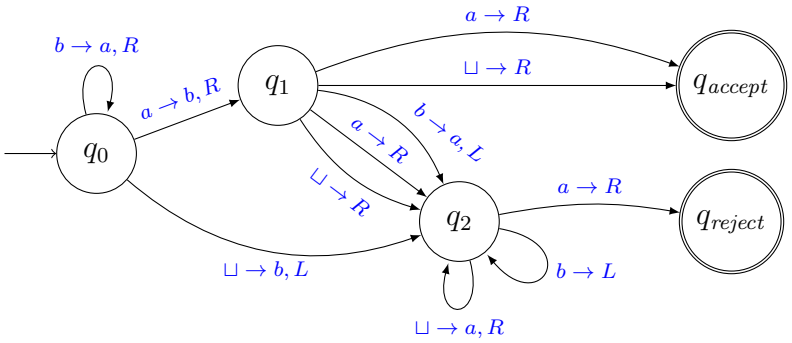
4.3

- Each stack operations takes at most cost 3.

- A sequence of n stack operations is at most $3n$, thus the time complexity of a sequence of n stack operations is $O(n)$.

Exercise 2, The C-Part, 50 points in total

1.(*15 points*) Consider the following non-deterministic Turing Machine, M. The state q_0 is the initial state.



1.1 Mark the correct statements about the execution of M on w, for each of the following input words (w).

w	can reach q_{accept}	can reach q_{reject}	can loop	$w \in L(M)$	$w \notin L(M)$
a	X		X	X	
ab			X		X
bb		X			X
aaa	X	X		X	

1.2 Is M a decider? ☐ **Yes** ☒ **No**

It is not a decider because it may loop forever for some inputs.

1.3 Is L(M) a recognizable language? ☒ **Yes** ☐ **No**

The non-deterministic turing machine M recognizes L(M), so we know that there exists a deterministic TM that also recognizes the same language.

1.4 Is L(M) a decidable language? ☒ **Yes** ☐ **No** Justify your answer.

Yes, M is not a decider we can construct a decider that recognizes the same language. The loop on q_0 is finite because it will have one iteration for each b at the beginning of the input. The only infinite loop that can happen is on q_2, and anytime q_2 is reached we cannot go to q_{accept}anymore. Therefore, we can directly reject instead of going to q_2.

2 (*15 points*) Let $X \in$ **NP-complete**. Mark the cells that are correct statements and can be proved according to our current knowledge without any assumptions (it is possible that none or all statements are correct). Justify your answer below (no points will be awarded for unjustified answers).

2.1 [X] **a)** X is a language. [ ] **b)** X is a Turing Machine.
Complexity classes contain languages and not Turing Machines.

2.2 [X] **a)** 3SAT $\preceq_P X$ [X] **b)** $X \preceq_P$ 3SAT
All problems in NP can be polynomially reduced to any NP complete language. 3SAT is in NP, therefore it can be reduced to X. X is in NP therefore it can be reduced to 3SAT.

2.3 [X] **a)** 3SAT $\preceq_M X$ [X] **b)** $X \preceq_M$ 3SAT
Same as above, if a problem is polynomially reducible to another, then it is also mapping reducible.

2.4 [X] **a)** X is decidable [X] **b)** X is recognizable
All NP problems are decidable and recognizable, because we can enumerate all certificates and then validate the solution in polynomial time. The amount of certificates that we need to consider is bounded for each input (as the length of the certificate is only polynomially long).

2.5 [ ] **a)** X is in **P** [ ] **b)** X is not in **P**
We cannot know whether X is in P or not. Otherwise the P vs NP question would not be open. If $P = NP$ then $X \in P$ (as $X \in NP$). If $P \neq NP$ then $X \notin P$, as $X \in NP - hard$.

2.6 [X] **a)** X is in **EXPTIME** [ ] **b)** X is not in **EXPTIME**
All NP problems are solvable in exponential time, because we can enumerate all certificates and then validate the solution in polynomial time. The amount of certificates that we need to consider is (exponentially) bounded for each input (as the length of the certificate is only polynomially long).

3 (*20 points*) Alice goes to an excursion trip, and wants to know if there is a way of visiting a number of locations, starting and ending on any of them, but without passing by the same location twice. She therefore wants to solve the following problem. Given a set of locations L and connections C, and a constant k, is there a way of traversing k locations without repeating any of them? The trip can start and end at any two (different) locations.

We define the following language:

$$\text{EXCURSION} = \{(\langle C \rangle, k) \mid \text{ there exist distinct } l_1, \ldots, l_k \text{ s.t. } (l_i, l_{i+1}) \in C\}$$

where $\langle C \rangle$ is a string defining a set of pairs of connected locations, and k is an integer represented by a decimal encoding.

For example "(A,B),(A,C),(C,D),(D,C),3" belongs to EXCURSION because there exists a path with 3 locations: A – C – D. However, "(A,B),(A,C),(C,D),(D,C),4" does not belong to EXCURSION because no path can visit the four locations.

Prove that EXCURSION is **NP-complete**. In your proof, remember to **specify clearly the different steps that need to be proven.** You may refer to other problems that we have seen in the lecture: SAT, 3SAT, CLIQUE, HAMPATH, VERTEX_COVER and SUBSET_SUM.

Step 1: Show that EXCURSION is NP. (1.5 points)

1.B (5 points) Certificate: a sequence of locations of length k

We can validate the certificate in polynomial time by the following pseudocode:

count the number of locations in C, if $k >$ #locations reject. Check whether the certificate has k different locations, otherwise reject. Then, for each pair of consecutive locations (l_i, l_{i+1}) in the certificate check whether $(l_i, l_{i+1}) \in C$ and if yes accept otherwise reject.

1.C (2 points) The algorithm runs in polynomial time, as all steps can be done by iterating over elements in C in $O(|C|)$, and there are at most $O(|C|)$ steps.

Step 2: Show that EXCURSION is NP-hard. (1.5 points)

2.B (3 points) We prove this by polynomial reduction from HAMPATH.

HAMPATH = $\{\langle G, s, t \rangle \mid G$ is an directed graph with a Hamilton path from s to $t\}$.

2.C (5 points) Reduction: For any input (G, s, t), we construct (C, k) such that C has a path of length k iff G has a Hamiltonian path from s to t.

If we ignore s and t, we can do that by choosing G=C, and k = the number of nodes in G.

In order to force the path to start at s at end at t, we add two extra nodes I and G with only two edges: $((I, s)$, and $(t, G))$. Thus, $C = G \cup \{(I, s), (t, G)\}$ and k = the number of nodes in C.

2.D (2 points)

Reduction is polynomial time.

Note that the reduction can be done in polynomial time as we only are required to copy G, insert two additional edges, and compute the number of nodes in G, all of which can be done in linear time.

Reduction is “correct”:

If G has a Hamiltionian path from s to t, then there is a path of length $k+2$ in C by starting at $I \to s \to path \to t \to G$.

If there is a path of length $k+2$ in C, then it must necessarily start at I and end at G (otherwise those nodes could not be visited), so the path is $I \to s \to path \to t \to G$. Note that path is a hamiltonian path for G as it does not visit any node twice and it must visit all nodes.