TDT4120 Algorithms and Data Structures

Examination, August 5, 2024, 09:00–13:00

Academic contact Magnus Lie Hetland

Support material code E

Problems

1 The following is taken from ENQUEUE:

```
Q[Q.tail] = x
if Q.tail == Q.size
    [redacted]
else Q.tail = Q.tail + 1
```

What is the redacted part supposed to be?

2 Suppose you run MST-PRIM and MST-KRUSKAL on a disconnected graph. Which of the algorithms will find a minimum spanning tree for each of the connected components of the graph? Explain briefly.

That is, which one will construct a disconnected solution that covers the entire graph?

3 One of the loops in COUNTING-SORT goes from n down to 1 (**for** $j = n$ **downto** 1). What is the consequence of changing the direction of the loop (**for** $j = 1$ to n)?

Note: No explanation is required here.

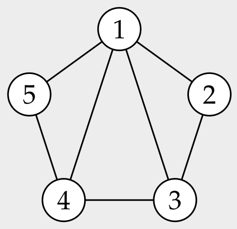
4 If you are to describe the best-case running time of an algorithm, which asymptotic notation (O, Ω, or Θ) should you use, if possible?

5 In a hash table with hash function h, what does it mean for the keys k_1 and k_2 to collide?

6 What is the amortized running time of TABLE-INSERT?

This refers to insertion into a dynamic table, where we either can insert the element directly if there is room, or must allocate a new and larger table otherwise. Provide the answer using Θ-notation.

Figure 1 Graph for problem 10



7 You have a directed, unweighted graph $G = (V, E)$, and you are to find the shortest paths *from all vertices* in V *to a given vertex t*. How would you proceed?

8 You want to find the longest simple path from vertex s to vertex t in a weighted graph. How could you do this? Are there cases where your method will not work? Explain briefly.

9 The solution to the 0-1 knapsack problem has a running time of $\Theta(nW)$, where n is the number of items and W is the capacity of the knapsack. Is this a polynomial-time algorithm? Explain briefly.

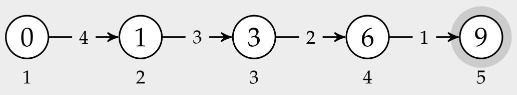
10 You are to represent the graph in Figure 1 as an adjacency matrix. Fill in 0 and 1 in the table below.

	1	2	3	4	5
1					
2					
3					
4					
5					

11 What is a vertex cover?

12 In the textbook definition of flow networks, antiparallel edges are not allowed, meaning we cannot have both an edge from v_1 to v_2 and one from v_2 to v_1. If we do have such edges, how can we handle the situation?

13 In a similar manner to BELLMAN-FORD, you are to perform RELAX on all edges in the following graph once. The order is not specified.



When you are finished, what are the smallest and largest values 5.d can have (i.e., $v.d$ for vertex 5, which is highlighted)? Provide the answer as two numbers, separated by a comma.

Each vertex's d-value before you start is given in the vertex in the figure, so for example, $4.d = 6$.

You are not to perform the entire Bellman-Ford algorithm, but update the estimate once along each edge, in some order.

14 Solve the following recurrence:

$T(n) = T(n-1) \cdot 2^{2^n} \qquad (n \geqslant 1)$

$T(0) = 2$

Give your answer exactly, that is, without asymptotic notation.

15 In Transitive-Closure, t_{ij} indicates whether there is a path from i to j. Now assume that the directed graph you receive as input is acyclic. How can you modify the algorithm so that t_{ij} becomes the number of paths from i to j?

You may describe your solution as a modification of Floyd-Warshall if you prefer.

16 A problem with Quicksort is that the running time is poor if the pivot element is poorly chosen. Can you choose the pivot so that the running time is guaranteed to be $\Theta(n \lg n)$? Explain.

This is about modifying only how the pivot is chosen; the rest of Quicksort should be performed as usual. Each recursive call should also be executed in the same manner, so you cannot, for example, use Merge-Sort initially to "cheat" your way to the correct running time.

17 You have n items and are to give one to each of n people. The people may prefer different items. Ideally, you don't want anyone to envy anyone else, but you realize this probably is not possible.

Instead, you create a lottery for each item. Rather than giving out the items directly, you give each person a random priority for each item. Your goal is that no one should envy someone who has a lower priority.

Will it always be possible to distribute the items in this way in polynomial time (assuming $P \neq NP$)? If so, how? If not, why not?

If you have received item x, I should not envy you unless I have a lower priority for item x.

18 Your friend Lurvik is studying two decision problems, A and B, where he has an exponential algorithm for A and a polynomial algorithm for B. He has shown that A cannot be solved faster than exponentially.

Algorithm 1 The inverse of ZIP

```
UNZIP($x$)
1  if $x$ == NULL
2      let L, R be new lists
3  else allocate new nodes $y$ and $z$
4      $y.key = x.key[1]$
5      $z.key = x.key[2]$
6      $L = \text{UNZIP}(x.next)[1]$
7      $R = \text{UNZIP}(x.next)[2]$
8      LIST-PREPEND(L, $y$)
9      LIST-PREPEND(R, $z$)
10 return $\langle L, R \rangle$
```

Lurvik has also found reductions from A to B and from B to A. What can you say about the running time of each of these reductions? Explain briefly.

If we consider A and B as formal languages, Lurvik has found two reduction functions f and g, where

$x \in A$ if and only if $f(x) \in B$ and

$x \in B$ if and only if $g(x) \in A$.

The question is what you can say about the running time required to compute the reduction functions f and g.

In the syllabus, a reduction is generally assumed to have polynomial running time, but you can disregard that here.

19 In many programming languages, there is a function called ZIP, which takes in two lists and returns a list of pairs, where pair i consists of element i from each of the two lists.

The procedure UNZIP (Algorithm 1) does the opposite. It takes the head of a linked list of pairs (arrays of length 2) and distributes them into two lists L and R.

How would you change the algorithm to improve the running time? What is the running time before and after your improvement? Explain.

20 COUNTING-SORT(A, n, k) takes in an array $A[1:n]$ with integers in the range $0, \ldots, k$ and fills an array $C[0:k]$ with the number of occurrences in A of each possible value, using $\Theta(n+k)$ operations.

You are to perform a similar counting where A is already sorted. You can assume that you also receive C as a parameter, and that C is initialized such that $C[i] = 0$ for $i = 0, \ldots, k$.

Use the *divide and conquer* method to construct an algorithm that solves the problem with a running time of $O(n)$ in general, but which is faster than this when

A contains many duplicates.