CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
TIN093	Algorithms			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
TIN093-0042-NBE		25/08-22	13	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	11	
2	X	9	
3	X	8	
4	X	15	
5	X	13	
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		56	

3

Polynomial

- Each iteration, the shared edge in two paths P_1 & P_2 is neither in resulting paths P_1' nor P_2',
  => total edges in all paths decreases by 2.
- Obviously, once the path between each node in each pair, is the shortest, the algorithm will terminate.
- If we choose arbitrary path by BFS, this will guarantee that each path is atmost $|E|$. ($O(k(|V|+|E|))$

=> All paths is at most $k|E|$

=> # Edges in all paths after x iterations is at most $k|E| - 2x$

=> will finish in $O(k|V| + k|E|) + O(\frac{k}{2}|E|)$
$= O(k|V| + k|E|)$

We have $k \leq V$ so

=> $O(|V|^2 + |V||E|)$

Clearly polynomial.

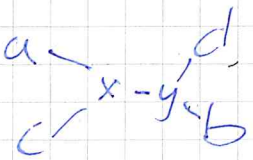
Nice.

3

Valid

Obvious each pair always has a path

- Assume result is invalid, i.e., there is a pair $\{a,b\}, \{c,d\}$ that have atleast one common edge in their Paths $P_{1,2}$. Call this $x-y$.



- We can always split according to algorithm s.t. connectivity is withheld except $\{a,b\} = \{c,d\}$

=> An invalid result can only appear if the algorithm has not yet finished.

Since the total number of edges in all paths is bound by $O(|V|^2 + |V||E|)$, and decreases strictly in each iteration the algorithm must terminate.

Since cannot terminate with an invalid solution and must terminate

=> will terminate with a valid solution

Very clear, too.

8

3

1.2)

Keep the same pairs as in 1.1, but do a BFS search for the shortest path from s to u and u to v respectively (e.g., ensure that the resulting pairs recieve the shortest path).

Motivations

- Still polynomial, as BFS is a polynomial time algorithm, and there is a polynomial number of itercations in which BFS is called twice.
- Must give shortest length, because BFS finds shortest path between two nodes
- The argument of validity from 1.1 still applies,
  - We cannot lose connectivity
  - If there is a shared edge between paths, we can always reassign the pairings to remove it.

    As the number of edges in all paths decreases each time and cannot become negative, we must eventually find an assignment without overlapping edges.

3

You missed that the new (shortest paths) may share edges again. One must iterate "1.1".

3

2.1

Let OPT(

$$\text{Let } OPT(k) = \max\left(\max_{i \in [1,k) \text{ AND } x_i \leq x_k - J} (OPT(i)) + 1,\ 1 \right)$$

Correctness comes from $OPT(k) \geq 1 \ \forall k \in [1, n]$, guaranteed by outer max and stemming from one-length subsequences being allocated. Furthermore, if we find a subsequence ~~with~~ ending in $\hat{i}$, and ($\hat{i} < k$ AND $x_{\hat{i}} \leq x_k - J$), we can clearly include x_k and extend this subsequence by one. Now just find longest such $\Rightarrow$ inner max operator with associated constraints.

Computing

- Set $OPT(k) = 1 \quad \forall k \in [1, n]$
- For $k = 1, \ldots, n$ iterate over $i = 1, \ldots k-1$ and save largest $OPT(i)$ found where $x_i \leq x_k + J$.
- Set $OPT(k)$ equal to this $OPT(i)$ for each k. Or equal to one if no such i exists.

6

3

2.2| Has for $k = 1, \ldots, n$
for $i = 1, \ldots, k$. This double for loop does $O(n^2)$ iterations. Constant time inside double for loop.
$\Rightarrow O(n^2)$

3

3

- Run the usual algorithm, but add that there exist an edge (u,v) to the criteria for computing the pairwise distance between (u,v). As in the original, the pairs can be identified in $O(n)$ time during the conquer step.

slightly unclear what happens

  - Assume can check for ~~connectivity in~~ if u and v has an edge (u,v) in $O(1)$ through ~~it~~ e.g. an available dictionary implementation of E.
  - If no ~~edge~~ valid in subproblem, set $d = \infty$

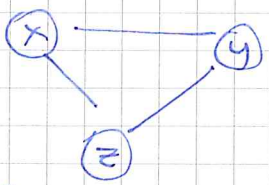
? - High connectivity means much find small d.

8

3

① H is triangle-free

Assume there exists a triangle of nodes x, y, z.



Since all fresh nodes are incident to exactly one old node and each old node either has no edges, or is connected to only fresh nodes, it follows that atleast one of x, y, z is old.

Assume x is old.

Since xy and xz are edges, y and z must be fresh. But since y, z must connect to exactly one old node ⇒ this is x for both

⇒ y and z was created to substitute the edge (x, x). But self-edges are disallowed, so this is a contradiction.

Thus, H is triangle free ☒

OK

② Reduction Runs in Pol time

- For each edge, we create a constcent number of nodes and edges. Thus, clearly polynomial in number of edges in G. (linear even)

OK

4. ③ Indep set in G ⟷ Indep set w ≥ k+m nodes in H

4.3.1

Given an independent set of size k $n_1, \ldots, n_k$ in G, consider the corresponding k nodes in H. These are still an independent set size k. Additionally, one of x and y in each u-x-y-v can be added to the independent set, as u and v cannot both be in the independent set in G.

This gives $\frac{2m}{2} = m$ additional nodes,

$\Rightarrow$ H has an independent set of atleast k+m nodes.

Ok

4.3.2] Conversely, given an independent set of size k+m in H, we know atmost m of these can be fresh nodes. Otherwise, we've chosen two neighbouring x,y from a u-x-y-v
Thus, atleast k of the independent nodes are old and in G
⇒ G has an independent set of atleast k nodes

Ok

4.3.3 Why NP-Complete

Since we can verify a solution in polynomial time (same procedure as for independent set) and since we can reduce the NP-complete, general independent set to Independent set for triangle-free graphs in polynomial time,
the problem is atleast as hard as independent set and is in NP.

⇒ Indep Set for triangle-free graphs is also NP-Complete.

Ok

15

3

5.1

a) Why is the solution valid?

Mot: Otherwise does not constitute a solution.

The constraint is (i) it must start and return in r
(ii) must visit every node at least once.

The algorithm essentially performs a DFS in each subtree (since a tree is acyclic, the marking is not needed). We know that DFS will visit every node, and since called for each subtree that r is incident too, and r is finally visited, we must visit every node atleast once.

- Since starts in r, and ~~then~~ finishes by traversing $r_a \to r$, clearly both starts and finishes in r.

Addition:

- For each node, visits all edges except the one it came from.
- After visiting all edges, returns to node.

⇒ Visits all nodes in connected graph (~~=~~ tree!) and always returns to origin.

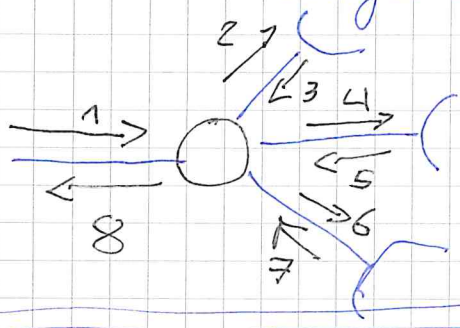
☒

OK, somewhat on the long side.

3

b) How often visits each edge?

When arriving in a node, the algorithm traverses down all outgoing edges. Since trees are acyclic, it returns exactly once for each edge as well



Going to/from node: w/ 4 edges

↖ deletes connect when going down so does not consid cycles before return

Thus, the algorithm traverses each edge twice.

OK

c) How often much traverse?

- We much traverse every edge atleast twice. Since trees are acyclic, if we go down one edge, we must pass the same edge on the way back or we will be stuck in a subtree.
- Additionally, we must traverse down every edge because no way to visit the node at the end of the edge otherwise (~~the~~ no cycles).

OK

3

d) Optimality

We know that:

(i) Each valid solution must visit each edge twice.

(ii) Our solution visits each node exactly twice.

(iii) Our solution is valid.

As such, it is clear that any solution will either visit all edges twice. Then it will have same length as Tour(T, r). OR, a solution can visit ~~att~~ atleast ~~ett~~ one edge more times. But since all edges has positive lengths, such a tour would be longer.

⟹ No valid solution can be shorter than the one given by Tour(T, r).

OK

12

3

5.2: Running Time

The algorithm is called once per edge. In each iteration of the algorithm, it deletes a number of edges that on average must be $\mathcal{O}(\frac{m}{n})$, since cycles cannot exist in trees.

Thus, the average time inside each function call is constant and the algorithm finishes in $\mathcal{O}(E)$.

Of course, we have $\mathcal{O}(V) = \mathcal{O}(E)$ in trees. 1

Why using the average?
There is an extremely simple worst-case argument.