

15-351 / 15-650 / 02-613: Exercises 1b

1. Let $f(n)$ and $g(n)$ be functions that take on positive values for all $n > 1$ and such that $f(n) = O(g(n))$. Prove or give a counterexample to the following statements:

- (a) $2^{n+1} = O(2^n)$
- (b) $2^{2n} = O(2^n)$
- (c) $f(n) = \Theta(f(n/2))$
- (d) $4^{\log_2 n} = O(2^{\log_2 n})$
- (e) $f(n)^2 = O(g(n)^2)$

2. Consider the following problem:

- You are given an undirected graph $G = (V, E)$ and two nodes $u, v \in V$. All edges have weight 1. Design a $O(V + E)$ algorithm to find the number of shortest paths between u, v . If u and v are not connected, the number should be zero.

For each of the following potential solutions, determine if it is correct, and if it is not correct, explain why.

- (a) To find all shortest paths, we can run a modified BFS. We only care about paths that don't backtrack, so we just check if the distance is strictly increasing.
- Initialize `dist[]` to ∞ for all nodes, `dist[u]` to 0.
 - Initialize a counter `total_paths` to 0.
 - Push u to a Queue Q .
 - While Q is not empty, pop the front node x .
 - If $x == v$, increment `total_paths++`. Don't check neighbors.
 - Otherwise, for every neighbor y of x , if `dist[x]+1 <= dist[y]`, set `dist[y] = dist[x]+1` and push y into Q .
 - Return `total_paths`.
- (b) We can use BFS to traverse the graph level by level. We will maintain an array `dist[]` initialized to ∞ and an array `count[]` initialized to 0.
- Set `dist[u]` to 0, `count[u]` to 1 and push u to a queue Q .
 - While Q is not empty, pop the front node x .
 - For every neighbor y of x :
 - If `dist[y] ==  $\infty$` (first time seeing y , set `dist[y] = dist[x]+1` and `count[y] = 1`, and push y into Q .
 - Else if `dist[y] == dist[x]+1` (found another shortest path to y , increment `count[y]++`.

(c) We will use a BFS, but store on each node w the number `paths[w]` of shortest paths from u to w . Initialize `paths[w]=0` for all nodes. Start at node u and terminate at v .

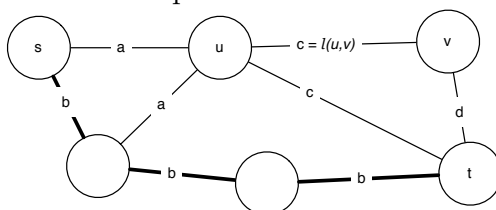
- Let layer 0 be just the start u . Let layer i of the BFS contain all nodes that are distance i from u .
- In layer 1, store the value 1 on every node, because that is the number of shortest paths from u to that node.
- For every layer $j + 1$, for each node w , sum up the values of each neighbor of w in layer j , because that is the total number of shortest paths from u to w .

After the BFS finishes, return `paths[v]`.

3. In tree $T = (V, E)$, if we remove edge $e = (u, v) \in E$ from T , then the tree is no longer connected. In other words, in $T \setminus \{e\}$, many node pairs would become disconnected (i.e., there would no path between some node pairs). We therefore define the importance of edge $e = (u, v) \in E$ as the number of node pairs that will become disconnected if we remove e from T . Design a $O(V)$ algorithm to find the most important edge in any given tree. Note that if there are multiple edges with the same importance, you can pick any of them.
4. Let $G = (V, E)$ be a connected, undirected graph, and suppose $T_s = (V, F)$ is a tree on G created by exploring G using depth-first search starting from vertex s . Vertex s is the root of tree T_s . Argue that s has more than one child in T_s if and only if removing s from G breaks G into several (more than one) disconnected parts.
5. The *diameter* of a connected, undirected graph $G = (V, E)$ is the length (in number of edges) of the *longest* shortest path between two nodes. (a) Show that if the diameter of a graph is $d > 1$ then there is some set $S \subseteq V$ with $|S| \leq |V|/(d - 1)$ such that removing the vertices in S from the graph would break it into several disconnected pieces. (b) Give an efficient algorithm to find S .
6. Let $G = (V, E)$ be a connected, undirected graph with positive edge weights $d(u, v)$, and suppose $|E| = \Omega(|V|^2)$. That is, $|E|$ is perhaps $\binom{|V|}{2}/100$. Give an implementation of Dijkstra's shortest path algorithm that is asymptotically faster on such dense graphs than the heap-based version we saw in class.
7. Let $G = (V, E)$ be an undirected, **bipartite** graph (you know it's bipartite, but don't know any 2-coloring). An *independent set* of G is a subset of the vertices $U \subseteq V$ such that no two vertices in U are connected by an edge of G . Give a $O(|V| + |E|)$ -time algorithm to find an independent set of size $\geq |V|/2$ in G .
8. Suppose $G = (V, E)$ is a directed, acyclic graph (a DAG) with positive weights $d(u, v)$ on each edge. Let s be a vertex of G with no incoming edges and such that every other node is reachable from s through some path.

Give an $O(|V| + |E|)$ -time algorithm to compute the shortest paths from s to all other vertices in G . Note that this is faster than Dijkstra's algorithm in general.

9. You are given an undirected graph $G = (V, E)$ with a letter $\ell(u, v)$ drawn from an alphabet Σ on edges E . For example:



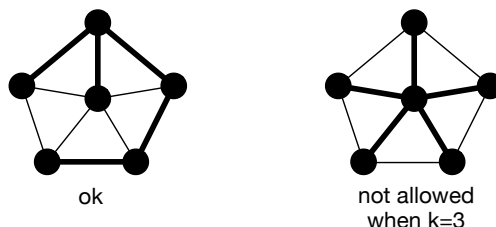
You want to compute a path P_{st} in G from a given node s to a given node t that has the *fewest number of different symbols on it* among all possible $s - t$ paths. For example, this the bold path in the figure above. Consider the following:

Algorithm sketch. Run a Dijkstra-like greedy tree growing algorithm, but choose the frontier edge $\{u, v\}$ that has the smallest value of $|g(u) \cup \{\ell(u, v)\}|$ to next add to the growing tree, where $g(u)$ is the set of letters used on the path found to u .

(Here $X \cup Y = \{a : a \in X \text{ or } a \in Y\}$ and represents the union of two sets, and $|X|$ is the number of elements in the set X .)

Either prove that the algorithm above finds the desired $s - t$ path or argue why it fails to.

10. The *degree-constrained minimum spanning tree problem* is this: you are given an undirected graph $G = (V, E)$ with positive weights on the edges, and you are also given an integer k . You must find a minimum weight spanning tree T on G (as we saw in class) with the extra restriction that no vertex has more than k edges adjacent to it in T . That is, if $k = 3$ then the left MST is allowed, but the right MST is not.



This problem is hard, and no polynomial-time algorithm is known for it.

Explain how to apply A* to solve this problem. You don't need to explain the A* algorithm; just explain how you would use an A* "library" to solve the problem. Briefly explain why your algorithm will find an optimal solution. You do not need to show anything about the running time.

11. Give an $O(|E|)$ -time algorithm to check whether a given tree T is a shortest-path tree of an undirected, connected graph $G = (V, E)$ with positive weights $d(u, v)$ on each edge.

Hint: check whether the edges that are not in T are correctly excluded.