University of Padova
Master of Science in Computer Science

Advanced Algorithms

Spring 2023

June 22, 2023 – 14:30–16:30

First Part: Theory Questions

Question 1 (4 points) Consider the following directed, weighted graph, represented by an adjacency matrix where each numerical value represents the weight of the corresponding edge, and where the symbol '−' indicates the absence of the edge between the corresponding vertices.

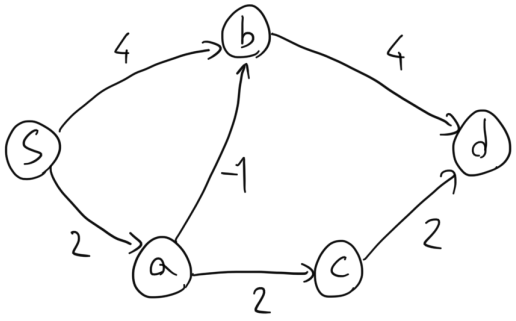
	s	a	b	c	d
s	-	2	4	-	-
a	-	-	-1	2	-
b	-	-	-	-	4
c	-	-	-	-	2
d	-	-	-	-	-

(a) Draw the graph.

(b) Run the Bellman-Ford algorithm on this graph, using vertex s as the source. You are to return the trace of the execution, i.e. a table with rows indexed by vertices and columns indexed by iteration indexes (starting from 0) where each entry contains the estimated distance between s and that vertex at that iteration.

Solution:

(a)



(b)

	0	1	2	3	4
s	0	0	0	0	0
a	∞	2	2	2	2
b	∞	4	1	1	1
c	∞	∞	4	4	4
d	∞	∞	8	5	5

Question 2 (4 points) For each of the following problems, say whether it is NP-hard or not and, if not, specify the complexity of the best algorithm seen in class.

(a) Maximum independent set

(b) All-pairs shortest paths

(c) Traveling salesperson problem

(d) Graph connectivity

Solution:

(a) NP-hard

(b) $O(n^3)$

(c) NP-hard

(d) $O(n + m)$

Question 3 (4 points) Define the set cover problem and briefly describe the $O(\log n)$-approximation algorithm seen in class.

Solution: See the lecture notes.

Second Part: Problem Solving

Problem 1 (9 points) Consider Dijkstra's algorithm seen in class, which returns the lengths of the shortest paths from a source vertex to all other vertices in directed graphs with nonnegative weights:

(a) Explain how to modify Dijkstra's algorithm to return the shortest paths themselves (and not just their lengths).

(b) Consider the following algorithm for finding shortest paths in a directed graph where edges may have negative weights: add the same large constant to each edge weight so that all the weights become nonnegative, then run Dijkstra's algorithm and return the shortest paths. Is this a valid method? Either prove that it works (i.e., the returned shortest paths are shortest paths in the original graph), or give a counterexample.

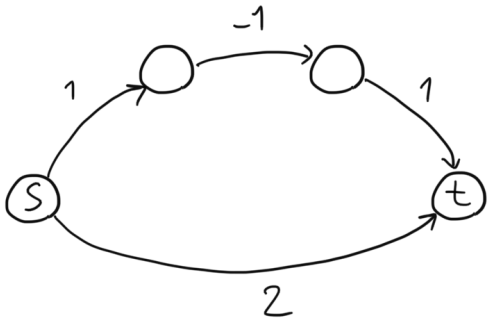
(c) Now let's switch to minimum spanning trees, and do the same: add the same large constant to each edge weight and then run Prim's algorithm. Either prove that the returned solution is a minimum spanning tree of the original graph, or give a counterexample.

Solution:

(a) Associate a pointer $predecessor(v)$ to each vertex $v \in V$. When an edge $(v^\star, w^\star)$ is selected in an iteration of the main while loop, assign $predecessor(w^\star)$ to $v^\star$. Then, at the end of the algorithm, to reconstruct a shortest path from s to a vertex v, follow the *predecessor* pointers backward from v to s.

(b) Alas, it does not work. In other words, you cannot reduce the SSSP problem with general edge weights to the special case of nonnegative weights in this way. The problem is that different paths from one vertex to another might not have the same number of edges, hence if we add some number to each edge weight, the lengths of different paths can increase by different amounts, and therefore a shortest path in the new graph might be different than in the original

graph. For example, consider the following graph:



The shortest path from s to t is the 3-hop path on top; however, if we add 1 to every edge's weight, the shortest path from s to t changes, and becomes the direct s-t edge—which has length 3, while the 3-hop path on top has length 4. (We would arrive at the same conclusion no matter which value we choose to force the graph to have nonnegative edge weights.) Moreover, if the original graph has negative cycles, these would not be detected.

(c) For minimum spanning trees this works. In fact, all spanning trees have the same number of edges, hence if we add some number to each edge weight the total weight of different trees increases by the same amount.

Problem 2 (10 points) Suppose you throw n balls into $\frac{n}{6\ln n}$ bins[1] independently and uniformly at random. Applying the following Chernoff bound show that, with high probability, the bin with maximum load (load = number of balls in the bin) contains at most $12\ln n$ balls. (Hint: focus first on one arbitrary bin and bound the probability of that bin's load exceeding $12\ln n\ldots$)

Theorem 1. *Let $X_1, X_2, \ldots, X_n$ be independent indicator random variables such that $E[X_i] = p_i, 0 < p_i < 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$. Then, for $0 < \delta \leq 1$,*

$$\Pr(X > (1+\delta)\mu) \leq e^{-\mu\delta^2/3}.$$

Solution: Focus on a particular bin. Applying the Chernoff bound, we now give an upper bound on the probability of that bin's load exceeding $12\ln n$. Let X_i, where $i = 1, 2, \ldots, n$, be the indicator random variable for whether the i-th ball is assigned to this particular bin. The load of the bin is then $X = \sum_{i=1}^{n} X_i$. Note that $\Pr(X_i = 1) = (6\ln n)/n$ because each ball is assigned to a bin chosen uniformly at random; also, X_i's are independent. To apply the Chernoff bound we set $12\ln n$ equal to $(1+\delta)\mu$; since $\mu = E[X] = \sum_{i=1}^{n} E[X_i] = n(6\ln n)/n = 6\ln n$, we get $\delta = 1$. Therefore,

$$\begin{aligned}
\Pr(X > 12\ln n) &= \Pr(X > (1+1)6\ln n) \\
&\leq e^{-\frac{6\ln n}{3}} \\
&= e^{-2\ln n} \\
&= e^{\ln n^{-2}} \\
&= 1/n^2.
\end{aligned}$$

We now show that, with high probability, the bin with maximum load contains at most $12\ln n$ balls. Applying the union bound over all the $\frac{n}{6\ln n}$ bins, we have that the probability that at least one bin gets more than $12\ln n$ balls is at most

$$\frac{n}{6\ln n} \cdot \frac{1}{n^2} = \frac{1}{6n\ln n}.$$

In other words, the load of no bin exceeds $12\ln n$ with probability at least $1 - 1/6n\ln n = 1 - o(1/n)$.

[1] Recall that $\ln n = \log_e n$.