CS31005 Algorithms–II, Autumn 2022–2023

23-Nov-2022, 09:00am–12:00pm **End-Semester Test** Maximum marks: 50

[*Answer all questions.*]

1. (a) You are given a maximum-flow algorithm A that works correctly only if (i) there are no incoming edges to the source and no outgoing edges from the sink, and (ii) both the directed edges (u,v) and (v,u) do not exist together for any pair of vertices u,v. However, you are given a graph G that violates both the above conditions. Can you still find the maximum flow in G using A? Justify your answer briefly and clearly. **(4)**

Solution Yes, we can use A with the following modifications on G.

(i) Add a new dummy source with a directed edge to s with capacity infinity, and from t to a new dummy node with capacity infinity. Find the flow between the dummy source and dummy sink. Another possibility is that we can remove all the incoming edges to s and all the outgoing edges from t. Any maximum flow can be maintained without these edges.

(ii) Replace one of the edges, say (u,v), with two edges (u,w) and (w,v) with the same capacity as that of (u,v), where w is a new node introduced.

(b) Define Monte Carlo and Las Vegas algorithms. **(2)**

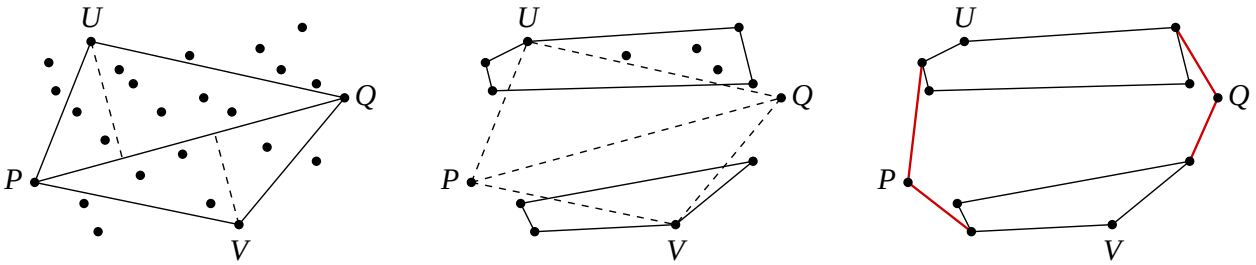
(c) Consider an application using a Bloom filter. The maximum number of entries to be inserted in the filter is 500,000, and the probability of false positives should not exceed 0.02%. What is the size of the filter (in MB, rounded up to 2 decimal places) that you will use? What will be the number of hash functions to be used? Show your calculations. You do not need to design the hash functions. **(4)**

Solution Size in bits = $-500000\ln(0.0002)/(\ln 2)^2 \approx 88.637 \times 10^5$, so the size in MB is approximately 8.87 MB (will accept anything between 8.8 and 8.9). Number of hash functions = $(88.637/5)\ln 2 = 13$ (taking ceiling, can accept 12 also).

(d) Consider the Euclidean TSP problem on a complete graph with n nodes (numbered from 1 to n), where each node is a point in the 2-d plane, and the weight of the edge (u,v) is the Euclidean distance between u and v. Suppose that you start a branch-and-bound algorithm to find the optimal tour cost starting from node 1. Consider any node u in the state-space tree (or computation tree) that represents a partial tour of $k+1$ nodes $\langle 1, x_1, x_2, \ldots, x_k \rangle$ with cost C_k, where $x_i \in \{2,3,\ldots,n\}$ for $1 \leqslant i \leqslant k < n$, and $x_i \neq x_j$ if $i \neq j$. Let the cost of the best complete tour seen so far be C_{best}. Suggest an $O(1)$-time pruning heuristic (bounding function) that you can apply to the node u to decide whether to explore the subtree rooted at u or not. For you to get any marks, your heuristic must be different from any min-weight-based heuristic, and its correctness must depend on the triangle inequality. Give a brief justification as to why it is correct. **(4)**

Solution If $C_k + w(x_k,1) \geqslant C_{best}$, do not explore the subtree, else explore. In order see why this heuristic works, let $x_{k+1}, x_{k+2}, \ldots, x_n$ be any permutation of the nodes yet to visit. By triangle inequality, the cost of any complete tour in this case will be $C_k + w(x_k,x_{k+1}) + w(x_{k+1},x_{k+2}) + \cdots + w(x_{n-1},x_n) + w(x_n,1) \geqslant C_k + w(x_k,x_{k+1}) + w(x_{k+1},x_{k+2}) + \cdots + w(x_{n-1},1) \geqslant C_k + w(x_k,x_{k+1}) + w(x_{k+1},x_{k+2}) + \cdots + w(x_{n-2},1) \geqslant \cdots \geqslant C_k + w(x_k,1)$.

(e) We compute the convex hull H of a set S of n points in the plane using an idea described now. Assume that the given points are in general position. We first compute the leftmost point P and the rightmost point Q in S. We also identify the most distant (perpendicular distance) points U and V on the two sides of the line PQ. See Part (a) of the figure below. The points inside the triangles PQU and PQV cannot lie on the convex hull, so we remove these points from S. We then recursively compute the convex hull H_1 of the subset of points of S, that lie above PQ. We also recursively compute the convex hull H_2 of the subset of points of S, that lie below PQ. See Part (b) of the figure below. Finally, we compute appropriate tangents from P and Q to the hulls H_1 and H_2 (Part (c) of the figure below) to get H.



Write, with clear justifications, the recurrence for the worst-case time complexity $T(n)$ of this algorithm, and deduce that $T(n)$ is $O(n^2)$. **(4)**

Solution Let the recursive calls be on n_1 and n_2 points. P and Q can be identified in $O(n)$ time. U and V can also be identified in $O(n)$ time. Finally, the four tangents can be computed and H can be constructed in $O(n)$ time too. This gives

$$T(n) = T(n_1) + T(n_2) + O(n)$$

with $n_1 + n_2 \leqslant n-2$. In the worst case, no points are removed from the two triangles PQU and PQV, so $n_1 + n_2 = n-2$. Moreover, the partitioning is so skew that we have either $n_1 = n-2$ or $n_2 = n-2$. But then, we have $T(n) = T(n-2) + O(n) = O(n^2)$.

2. Foomazon Logistic Company serves n customers. Its items are dispatched from $t \geqslant 2$ warehouses. Each warehouse has a vehicle. Each vehicle starts from its warehouse, serves a subset of customers (each customer only once), and comes back to its warehouse. A cost (a positive integer, assumed symmetric) is given to travel from one customer to another, and also from each warehouse to each customer. Foomazon wants to schedule its vehicles to serve disjoint subsets of customers in such a way that the total cost of serving all the customers is minimized. Note that each customer is to be served exactly once by exactly one vehicle.

(a) Foomazon faces an optimization problem. Propose an equivalent decision version of the problem, and show that the optimization problem can be solved in polynomial time if and only if the decision problem can be solved in polynomial time. **(4)**

Solution You are additionally given a positive bound B. Decide whether Foomazon can schedule its vehicles so that the total cost of serving all the customers is $\leqslant B$.

If the optimization problem can be solved in polynomial-time, one just checks whether OPT $\leqslant B$.

Let c_{min} and c_{max} respectively stand for the minimum and the maximum costs of each leg of travel. Then OPT is bounded as $(n+t)c_{min} \leqslant \text{OPT} \leqslant (n+t)c_{max}$. If a polynomial-time decider for the decision problem is available, then we can invoke the decider multiple times in order to stage a binary search for locating OPT. The number of invocations of the decider is bounded by $\log_2\big((n+t)c_{max}\big)$, which is polynomial in the input size.

(b) Prove that the decision problem of Part (a) is NP-complete. Use (the decision version of) TSP in your reduction, but note that $t \geqslant 2$ in the Foomazon problem. **(6)**

Solution First, show that the decision problem is in NP. Then, make a reduction from TSP with cost bound k. Let I be an instance of TSP. Make two identical copies I_1, I_2 of I. Add a cost of ∞ from each location (warehouse/customer) of one copy to each location in the other copy. Also, take $t = 2$ and $B = 2k$.

If the TSP instance has a solution, then replicate the tour in the two copies to get a solution of the decision problem of Part (a).

Conversely, suppose that the decision problem of Part (a) has a solution. This solution cannot use an edge connecting the two copies, because such an edge is of infinite cost. Therefore this solution corresponds to two TSP tours of costs k_1 and k_2 in the two copies. Since $k_1 + k_2 \leqslant B = 2k$, we have either $k_1 \leqslant k$ or $k_2 \leqslant k$.

3. Let $G = (V, E)$ be an undirected graph. For two subsets X, Y of V (not necessarily disjoint), denote by $N(X, Y)$ the number of edges of G with one endpoint in X and the other in Y. If X contains only a single vertex x, we also write $N(x, Y)$ to denote the number of neighbors of x in Y. A 3-cut of G is a partition of V into three mutually disjoint non-empty subsets U_1, U_2, U_3. The size of the 3-cut U_1, U_2, U_3 is

$C(U_1,U_2,U_3) = N(U_1,U_2)+N(U_2,U_3)+N(U_3,U_1)$. The MAX-3-CUT problem deals with finding a 3-cut U_1,U_2,U_3, for which $C(U_1,U_2,U_3)$ is as large as possible.

Let $v_1,v_2,v_3,\ldots,v_n$ be an arbitrary ordering of the vertices of G. Consider the approximation algorithm Q3 for solving the MAX-3-CUT problem. Assume that $n \geqslant 3$.

Algorithm Q3

Initialize $U_1 = \{v_1\}$, $U_2 = \{v_2\}$, and $U_3 = \{v_3\}$.
For $(i = 4;\ i \leqslant n;\ i{+}{+})$ repeat:
    Find the minimum of $N(v_i,U_1)$, $N(v_i,U_2)$, and $N(v_i,U_3)$.
    If the minimum is achieved by U_j (ties are broken arbitrarily), add v_i to U_j.
Return U_1,U_2,U_3.

(a) Find and prove an invariance (an inequality involving the counts $N(U_i,U_j)$) of the loop in Algorithm Q3. (An invariance of the loop is a statement that is true at every instant when the loop condition $i \leqslant n$ is checked.) **(4)**

Solution The invariance of the loop is

$$N(U_1,U_2)+N(U_2,U_3)+N(U_3,U_1) \geqslant 2\Big[N(U_1,U_1)+N(U_2,U_2)+N(U_3,U_3)\Big].$$

Initially, the right side of the inequality is 0, and the left side is one of $0,1,2,3$. Let the minimum neighborhood size of v_i in the loop body be δ. Then, the right side of the inequality increases by δ, whereas the left side increases by at least 2δ.

(b) Using the invariance of Part (a), establish that Algorithm Q3 is a $\frac{2}{3}$-approximation algorithm. **(2)**

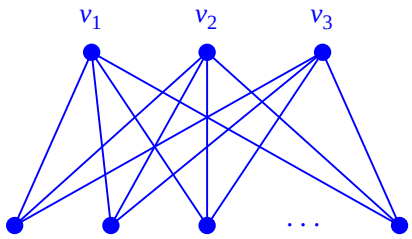
Solution The number of edges in G is

$$\begin{aligned} m &= \Big[N(U_1,U_2)+N(U_2,U_3)+N(U_3,U_1)\Big] + \Big[N(U_1,U_1)+N(U_2,U_2)+N(U_3,U_3)\Big] \\ &\leqslant \Big[N(U_1,U_2)+N(U_2,U_3)+N(U_3,U_1)\Big] + \frac{1}{2}\Big[N(U_1,U_2)+N(U_2,U_3)+N(U_3,U_1)\Big] \\ &= \frac{3}{2}\Big[N(U_1,U_2)+N(U_2,U_3)+N(U_3,U_1)\Big]. \end{aligned}$$

Therefore SUBOPT $\geqslant \frac{2}{3}m$, whereas OPT $\leqslant m$, so $\rho = \dfrac{\text{SUBOPT}}{\text{OPT}} \geqslant \dfrac{2}{3}$.

(c) Prove that this approximation ratio is tight. Your proof must use an *infinite* family of *connected* graphs. **(4)**

Solution Consider the complete bipartite graph $K_{3,n}$ with the initial three vertices v_1,v_2,v_3 as shown in the figure below.



In each iteration, the new vertex v_i has exactly one neighbor in each of U_1,U_2,U_3. No matter how the ties are broken, the algorithm produces a 3-cut of size $2n$. On the other hand, if we take $U_1 = \{v_1,v_2\}$ and $U_2 = \{v_3\}$, and put the remaining n vertices in U_3, then the size of the 3-cut is $3n$.

4. (a) Consider an undirected graph $G = (V,E)$ with $|V| = n$, and with a positive weight $w(v)$ assigned to each vertex v of the graph. Let Δ be the maximum degree of a vertex in G. An independent set of the graph is a subset S of V such that no two vertices in S are adjacent to each other in G. The weight of an independent set is the sum of the weights of the vertices in that set. The *maximum weighted independent set problem* is to find an independent set in G with the maximum weight. The problem is known to be NP-hard. The greedy randomized algorithm Q4 is proposed for solving this problem.

Algorithm Q4

Choose a random permutation $v_1, v_2, \ldots, v_n$ of the vertices.
Initialize $S = \emptyset$.
For $i = 1, 2, 3, \ldots, n$ (in that order), repeat:
    If no vertex in S is adjacent to v_i, then set $S = S \cup \{v_i\}$.
Return S.

For any $u \in V$, derive a lower bound on the probability that u is chosen in S. Your bound should be expressed in terms of Δ, and must be independent of the number of nodes, the number of edges, and the vertex weights. Give a clear justification (no marks without it). **(4)**

Solution Let d be the degree of u. The algorithm includes u in S if one of the following two cases happens.

(1) u is considered for inclusion in S before all of its neighbors are.

(2) One or more neighbors of u are considered before u for inclusion in S, but these neighbors were not included in S because some other neighbors of these neighbors were included in S in earlier iterations.

The two cases are mutually exclusive, so the desired probability is the sum of the probabilities of the two cases. The second case occurs with probability $\geqslant 0$ (this probability may be 0; consider a star graph with u at the center and with all its neighbors coming earlier than u in the ordering). Therefore $\Pr[u \in S]$ is bounded from below by the probability of the first case.

In a random permutation, u appears before its d neighbors with probability $\frac{1}{d+1}$. Moreover, $d \leqslant \Delta$. Therefore

$$\Pr[u \in S] \geqslant \frac{1}{d+1} \geqslant \frac{1}{\Delta+1}.$$

(b) Show that the expected weight of the independent set S returned by Algorithm Q4 is at least $\frac{1}{\Delta+1}$ times the optimal (the weight of the actual maximum-weight independent set). To show this, first define, for each $u \in V$, an indicator variable X_u which is 1 if u is included in S, 0 otherwise. Then use these variables to do the rest. Your proof must start with these indicator variables. **(4)**

Solution By Part (a), we have

$$\mathrm{E}[X_u] = 1 \times \Pr[u \in S] + 0 \times \Pr[u \notin S] \geqslant \frac{1}{\Delta+1}.$$

The weight of the independent set returned by the algorithm is the random variable W expressed in terms of these indicator variables as

$$W = \sum_{u \in V} w(u) X_u.$$

By linearity of expectation, we then have

$$\mathrm{E}[W] = \mathrm{E}\Big[\sum_{u \in V} w(u) X_u\Big] = \sum_{u \in V} w(u) \mathrm{E}[X_u] \geqslant \frac{1}{\Delta+1} \sum_{u \in V} w(u).$$

On the other hand, any optimal solution has weight $\leqslant \sum_{u \in V} w(u)$, and the result follows.

(c) Consider a Tweeter-like system used by 1 million (10^6) users. Each user's tweets are stored as a record of the form (username, collection of tweets). Given the large number of users, the data is divided into five servers, each containing the records of 200,000 users (one user's data is stored in exactly one server). A user can log in to any of the five servers, henceforth referred to as the log-in server. If the user's data are not stored in the log-in server, his/her tweets need to be brought from the server they are stored in. You should ensure that most of the time, the log-in server communicates with at most one other server (the correct one) to get the user's data. Propose a scheme based on Bloom Filters, to achieve this (no marks if Bloom filters are not used). In particular, clearly specify

(i) exactly what Bloom filter(s) is/are kept in each server, and **(2)**

(ii) the actions taken by the log-in server to get a user's data when that user logs in. **(2)**

Solution The constraint of accessing at most one other server most of the time implies that if the data is not in the log-in server, the log-in server must find out which server it is without contacting the relevant server most of the time (one access will be needed anyway to get the actual data if not present locally). But a Bloom filter just says Yes/No, and cannot say which server it is. This implies that each server must keep a Bloom filter for all servers.

So the design is: Each server keeps one Bloom filter (total five, including its own) for storing the users (user IDs) in each server (the local Bloom filters may be generated at the servers and distributed to other servers). On a login, the log-in server searches each Bloom filter to see which server stores the user's data. The possibilities are (i) exactly one Bloom filter answers *yes* (in which case this must be correct as all other *no* answers are correct), or (ii) more than one Bloom filter answer *yes,* in which case the log-in server has no option but to go to all of the corresponding servers and ask (note that still there is no need to go to the servers whose Bloom filters say *No*). We can however choose the parameters of the Bloom filters to acceptably low as required by the phrase "most of the time."

Another possibility is to use a single Bloom filter storing the pairs (user_ID, storage_server_ID) for all users. False positives would still appear albeit with an acceptably low probability.
