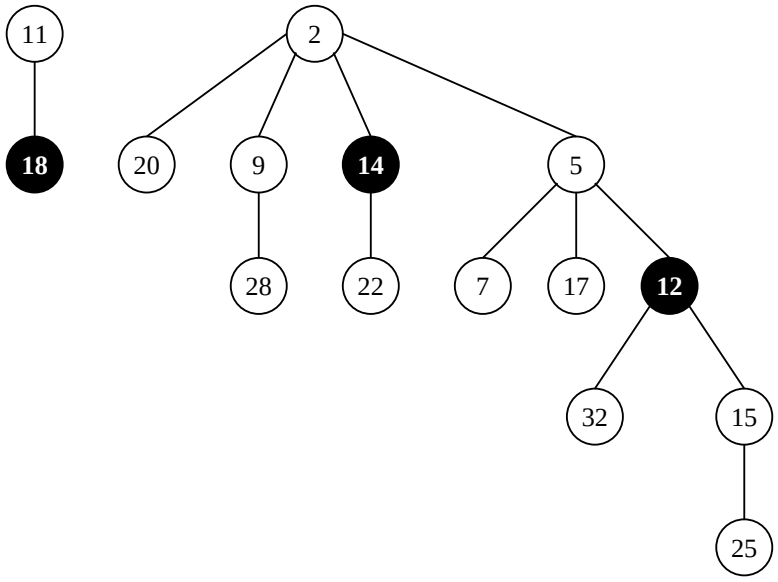
CS31005 Algorithms–II, Autumn 2022–2023

September 2022 **Mid-Semester Test** Maximum marks: 60

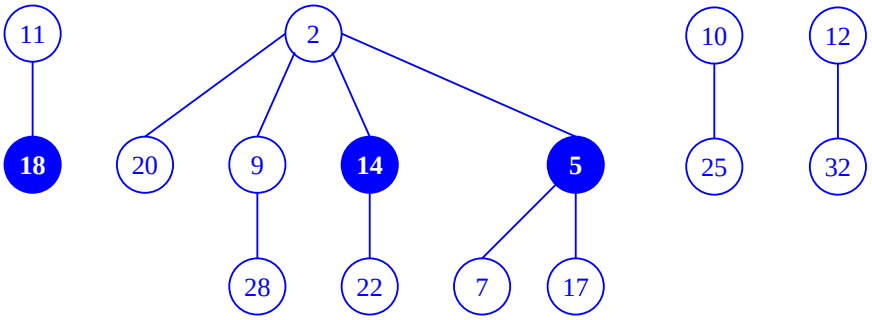
[*Answer all questions.*]

1. **(a)** The adjacent figure shows a Fibonacci heap consisting of two trees. The marked nodes are shown as solid black nodes. Consider the node storing the priority value 15. You need to decrease this priority to 10. Draw the Fibonacci heap just after this decrease-priority operation. Show also all the marked nodes in this modified heap. No explanation is needed.

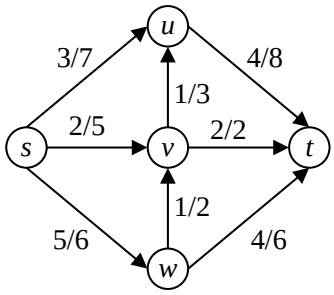


(5)

Solution The updated heap is given below. The marked nodes are again shown as solid ones.

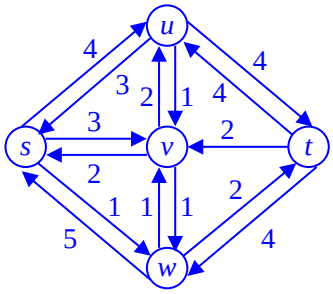


(b) A flow network with source s and sink t is shown in the adjacent figure. The flow and the capacity of an edge e is shown as $f(e)/c(e)$ beside that edge. Draw the residual network for this flow, and find an augmenting path that the Edmonds–Karp algorithm would select. No explanation is needed.



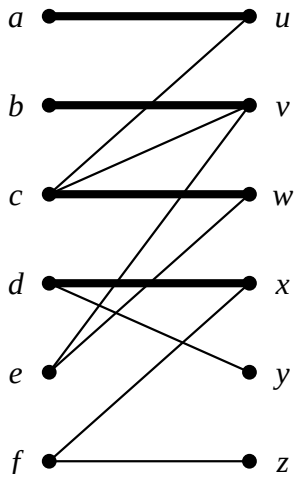
(5)

Solution The residual network is given below.



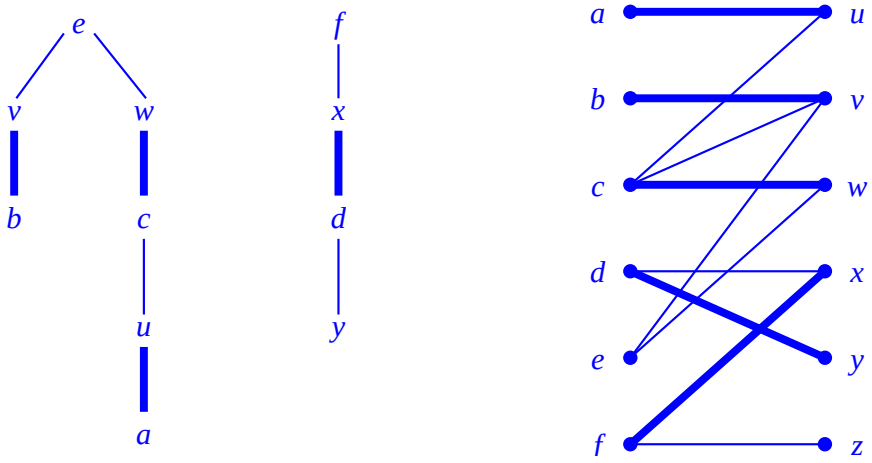
The Edmonds–Karp algorithm chooses shortest s,t paths. Two such paths are s,u,t and s,w,t.

(c) The adjacent figure shows the matching $M = \{(a,u),(b,v),(c,w),(d,x)\}$ in a bipartite graph with the bipartitioning $L = \{a,b,c,d,e,f\}$ and $R = \{u,v,w,x,y,z\}$. In order to find an augmenting path, we make a (restricted) DFS traversal from free nodes in the left part L. The free nodes are to be chosen from top to bottom. If one (or more) free nodes fail to identify any augmenting path, we continue the DFS from another free node in L (if any is remaining). The DFS neighbors of nodes in L should be explored in the top-to-bottom order. Draw the alternating forest, identify an alternating path that augments M, and show the augmented matching. No explanation is needed.



(5)

Solution We start DFS from e, but fail to discover any augmenting path. So we continue the DFS from f, and discover the augmenting path f,x,d,y. The augmented matching is $\{(a,u),(b,v),(c,w),(d,y),(f,x)\}$.

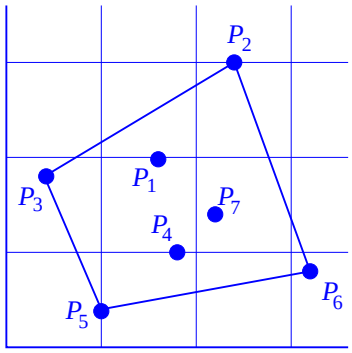


(d) Show the working of Jarvis march for computing the convex hull of the seven points

$$P_1 = (8,10),\ P_2 = (12,15),\ P_3 = (2,9),\ P_4 = (9,5),\ P_5 = (5,2),\ P_6 = (16,4),\ \text{and}\ P_7 = (11,7).$$

Briefly explain how each step works. **(5)**

Solution



We may start from the bottommost point. Using a min calculation on the y-coordinates, we identify P_5 as the starting point. We then compute the angles of the remaining points with respect to the horizontal ray emanating from P_5 and moving right. The angle calculations may be done exactly (up to floating-point approximations using atan2), or relatively using side calculations. The smallest angle is given by P_6, so we march to P_6. We then compute the angles of P_1,P_2,P_3,P_4,P_7 with respect to the ray P_5P_6. The smallest angle corresponds to P_2, so we discover the second edge P_6P_2 on the hull, and march to P_2. Two other edges P_2P_3 and P_3P_5 are discovered using similar angle calculations. Since we eventually come back to the point P_5 from which we started the march, the algorithm terminates.

We may also start from the leftmost point P_3, and discover the edges P_3P_2, P_2P_6, P_6P_5 and P_5P_3 in the clockwise order. The starting reference is the vertically upward ray through P_3. During the march, you have to identify the next point by maximizing the angle (if computed in the counterclockwise sense).

2. IIT Kharagpur has hired you to write an algorithm to schedule their final exams. Each semester, IIT Kgp offers n different courses. There are r different rooms on campus and m different time slots in which exams can be offered. You are given two arrays $E[1 \ldots n]$ and $S[1 \ldots r]$, where $E[i]$ is the number of students enrolled in the i-th course, and $S[j]$ is the number of seats in the j-th room. At most one final exam can be held in each room during each time slot. Course i can hold its final exam in room j only if $E[i] \leqslant S[j]$. Assume that no two courses share common registrants, so any number of courses can be scheduled in the same time slot. The only restriction is imposed by the availability and suitability of the rooms as mentioned above. Describe and analyze an efficient algorithm to assign a room and a time slot to each course (or report correctly that no such assignment is possible). **(10)**

Solution Create three sets of vertices, one set C for each course, one set R for each room, and one set T for each slot. Also create a source s and a sink t.

Add an edge from s to each vertex in C with capacity 1.

Add an edge from vertex i in C to vertex j in R with capacity 1 if and only if $E[i] \leqslant S[j]$.

Add an edge from each vertex in R to each vertex in T with capacity 1.

Add an edge from each vertex in T to sink t with capacity r (anything $\geqslant r$, including ∞, will do).

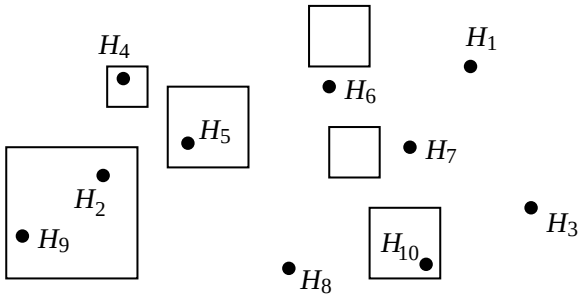
Now find maximum flow. It is possible only if all edges from s are saturated (or equivalently, the maximum flow in n).

3. We run the Gale–Shapley algorithm to compute a stable matching between n men and n women. Assume that men propose, and women take decisions (accept, reject, or replace). Prove that no matter how large n is, at most one man can be paired with his last choice. **(8)**

Solution If no man gets his last choice, we are done. So suppose that m is the first of the n men, who gets his last choice. Let the preference list for m be $w_1, w_2, \ldots, w_n$. That m is paired with w_n means that he must have proposed to w_n. At the time he makes this proposal, m must be unengaged, that is, he has earlier been rejected or replaced by all of $w_1, w_2, \ldots, w_{n-1}$. But a woman rejecting or replacing m must be paired with a partner better than m. Moreover, an engaged woman can never become unengaged (she can improve her partner though). Therefore at the time when m proposes to w_n, all of $w_1, w_2, \ldots, w_{n-1}$ are engaged. If w_n is engaged too at that time, then there are n engaged women and $\leqslant n-1$ engaged men (m is unengaged), a contradiction. So w_n is unengaged at the time of the proposal. But $n-1$ women can be engaged to exactly $n-1$ men, that is, at the time of the proposal, m is the only unengaged man and w_n is the only unengaged woman. But then w_n accepts the proposal of m, and this new engagement completes a perfect matching, and the algorithm terminates, that is, no man other than m ever feels the need to propose to his last choice.

Although not asked in this question, one man can get his last choice. For instance, the example covered in the class has the pairing (A, G).

4. A town in Geomerica suffers from a power outage caused by an intense solar storm. The town has m emergency light sources $L_1, L_2, \ldots, L_m$. The i-th light source L_i can illuminate an axis-aligned square region of side length r_i with $L_i = (x_i, y_i)$ at the center. For simplicity, assume that the m illumination zones do not overlap with each other. The town has n houses $H_1, H_2, \ldots, H_n$ considered as points (h_j, k_j) for $j = 1, 2, \ldots, n$. The mayor of the town wants to figure out which houses can be illuminated by the emergency light sources, so he can notify the other house-owners for making their own arrangements. The following figure illustrates the situation for six light sources and ten houses. The squares in the figure show the illumination zones (their centers L_i are not shown). Only the five houses $H_2, H_4, H_5, H_9, H_{10}$ can be illuminated.



The inputs to the problem are x_i, y_i, r_i for $i = 1, 2, \ldots, m$ and h_j, k_j for $j = 1, 2, \ldots, n$. For each j, the mayor can find out whether H_j can be illuminated by any of the light sources L_i. This gives an $O(mn)$-time algorithm. You are required to design an $O((m+n)\log(m+n))$-time algorithm based on the line-sweep paradigm, where one vertical line sweeps from left to right. You may make suitable general-position assumptions if needed. Clearly mention the following in your solution.

(a) What the types of the events are.
(b) How each type of event is handled.
(c) What data structure you use for maintaining the event queue. Justify.
(d) What data structure you use for maintaining the information pertaining to the sweep line. Justify.
(e) A justification that your algorithm runs in $O((m+n)\log(m+n))$ time. **(12)**

Solution (a) The sweep line keeps track of the active light sources, that is, those whose illumination zones intersect with the sweep line. There are three types of events.

Enter illumination zone (E): These events happen at the x-coordinates $x_i - \frac{r_i}{2}$ for $i = 1, 2, \ldots, m$.

Leave illumination zone (L): These events happen at the x-coordinates $x_i + \frac{r_i}{2}$ for $i = 1, 2, \ldots, m$.

House (H): These events happen at the x-coordinates h_j for $j = 1, 2, \ldots, n$.

The x-coordinates may be assumed to be in the general position. However, if one E and one H happen at the same x-coordinate, E is given priority over H. Also, if one L and one H happen at the same x-coordinate, H is given priority over L.

(b) The events are handled as follows.

Event E: Insert the corresponding light source in the sweep-line information.

Event L: Delete the corresponding light source from the sweep-line information.

Event H: Search the sweep-line information to check whether the house causing the event (say, H_j) is covered by any of the active light sources. The search succeeds if and only if $k_j \in [y_i - \frac{r_i}{2}, y_i + \frac{r_i}{2}]$ for some active light source L_i.

(c) The event queue is processed in the increasing order of x-coordinates. All of the $2m+n$ events are known a priori. The event queue should support findMin and deleteMin. A priority queue (a binary min-heap) can be used to implement this. The queue is initialized by makeHeap. A sorted array can also be used.

(d) The sweep-line information should support arbitrary insert, arbitrary delete, and search. For efficiently handling these operations, we can implement this as a height-balanced binary search tree. The ordering is with respect to the y-coordinates. Since the illumination zones do not overlap with each other, storing only y_i, r_i for the active light sources suffice. But these information can be made available from the input, so only the indices i of the active light sources may be stored in the nodes of the tree.

(e) If the event queue is implemented as a heap, it can be initialized in $O((2m+n))$ (which is also $O(m+n)$) time. Subsequently, each deleteMin can be done in $O(\log(2m+n))$ time, which is also $O(\log(m+n))$. There are $2m+n = O(m+n)$ operations on the priority queue after initialization.

If the event queue is implemented as a sorted array, the initial sorting takes $O((2m+n)\log(2m+n))$, that is, $O((m+n)\log(m+n))$ time. Subsequently, each event can be identified in $O(1)$ time.

Each insert, delete, or search in the height-balanced BST implementing the sweep-line information can be done in $O(\log m)$ time, since there can be at most m active light sources at any point of time. The total number of events is $2m+n$. So the total cost associated with this data structure is $O((2m+n)\log m)$ which is $O((m+n)\log(m+n))$.

5. Let $\Phi(x_1, x_2, \ldots, x_n)$ be a Boolean formula in n variables $x_1, x_2, \ldots, x_n$. By COMPLEMENTSAT, denote the problem of deciding whether there exists a truth assignment $(t_1, t_2, \ldots, t_n)$ of the variables such that both $\Phi(t_1, t_2, \ldots, t_n)$ and $\Phi(\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n})$ are true, where $\bar{t}$ denotes the complement of the truth value t. Prove that COMPLEMENTSAT is NP-Complete. **(10)**

Solution COMPLEMENTSAT is in NP, since a truth assignment $(t_1, t_2, \ldots, t_n)$ for which both $\Phi(t_1, t_2, \ldots, t_n)$ and $\Phi(\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n})$ are true constitutes a succinct certificate for Φ to be in Accept(COMPLEMENTSAT), and can be verified in polynomial time (in n and the size of Φ).

For proving the NP-Hardness of COMPLEMENTSAT, we propose a reduction SAT $\leqslant$ COMPLEMENTSAT. Let $\Psi(x_1, x_2, \ldots, x_n)$ be an instance for SAT. Introduce n new variables $y_1, y_2, \ldots, y_n$, and generate the instance $\Phi(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) = \Psi(x_1, x_2, \ldots, x_n) \vee \Psi(\overline{y_1}, \overline{y_2}, \ldots, \overline{y_n})$ for COMPLEMENTSAT. This conversion can be done in polynomial time (indeed in time linear in n and the size of Ψ).

If Ψ is satisfiable, then there exists a truth assignment $(t_1, t_2, \ldots, t_n)$ of $(x_1, x_2, \ldots, x_n)$ for which $\Psi(t_1, t_2, \ldots, t_n)$ is true. Take the truth assignment $(t_1, t_2, \ldots, t_n, t_1, t_2, \ldots, t_n)$ of $(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)$ for Φ. For this truth assignment, Φ evaluates to $\Psi(t_1, t_2, \ldots, t_n) \vee \Psi(\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n}) = 1 \vee \Psi(\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n}) = 1$, whereas for the complement truth assignment, Φ evaluates to $\Psi(\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n}) \vee \Psi(t_1, t_2, \ldots, t_n) = \Psi(\overline{t_1}, \overline{t_2}, \ldots, \overline{t_n}) \vee 1 = 1$.

On the other hand, if Ψ is not satisfiable, then for any truth assignment, it evaluates to 0. Φ too evaluates to $0 \vee 0 = 0$ for any truth assignment. So Ψ is not even satisfiable (let alone being complement-satisfiable).
