1. Recall the extension of Euclid's algorithm that we discussed in class.

```
1  def extended_Euclid(a,b):
2      """
3      a, b are non-negative integers
4      The function returns (u, v, d) such that d = gcd(a,b)
5      and d = ua + vb
6      """
7
8      if b == 0: return (1, 0, a)
9      (u, v, d) = extended_Euclid(b, a % b)
10     return (v, u - v * (a//b), d)
```

We argued in class that the algorithm correctly returns (u, v, d) as stated in the comment in the beginning of the code. Suppose for a certain input (a, b), where $a > b \geq 1$, the call to extended_Euclid(a, b) executes line 9 a total of t times (where $t \geq 1$). Let the value of (a, b) in the i-th call to extended_Euclid(a, b) be (a_i, b_i); let $(a_0, b_0) = (a, b)$. Let the value (u, v) returned by the i-th call be (u_i, v_i), so that $u_i a_i + v_i b_i = d$; thus $(u_t, v_t) = (1, 0)$. Then, for $i = 1, 2, \ldots, t$, we have

$$\begin{bmatrix} a_{i-1} \\ b_{i-1} \end{bmatrix} = \begin{bmatrix} q_i & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix};$$

$$\begin{bmatrix} u_{i-1} & v_{i-1} \end{bmatrix} = \begin{bmatrix} u_i & v_i \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix},$$

where q_i is the quotient obtained on dividing a_{i-1} by b_{i-1}.

(a) Show that $|u_i| \leq b_i/d$ and $|v_i| \leq a_i/d$, where $d = \mathsf{gcd}(a, b)$. You may use induction to show that the claim holds for $i - 1$ assuming it holds for i; what is the base case?

(b) Suppose a and b are n-bit integers. Show that the total number of bit operations needed for extended_Euclid(a, b) is $O(n^3)$, assuming that integer division of ℓ-bit integers can be done in using $O(\ell^2)$ bit operations.

2. Consider the following modification to Euclid's algorithm.

```
1  def modified_Euclid(a,b):
2      """
3      a, b are non-negative integers
4      The function returns (u, v, d) such that d = gcd(a,b)
5      and d = ua + vb
6      """
7
8      if b == 0: return a
9      r = a % b
10     if r < b/2:
11         return modified_Euclid(b, r)
12     else:
13         return modified_Euclid(b, b-r)
```

(a) Argue that for integers $a > b > 0$, modified_Euclid(a, b) returns the gcd of a and b.

(b) How many times is modified_Euclid called recursively after modified_Euclid(a, b) is called with Fibonacci numbers $a = F_{t+1}$ and $b = F_t$?

3. Describe an algorithm to determine if a given positive number $N \geq 2$ can be written in the form $N = Q^E$, where Q and E are both integers at least 2. For n-bit numbers N, your algorithm should run in time $O(n^k)$ for some small constant k (fixed independent of n).

4. Suppose x, y and ℓ are n-bit numbers, such that $x > y$. Suppose the binary expansion of the fraction x/y is

$$0.b_0b_1b_2b_3\ldots = \sum_{i\geq 1} b_i 2^{-i},$$

which in general may not terminate. Describe an algorithm to determine b_ℓ, given x, y and ℓ. Your algorithm should run in time $O(n^k)$ for some small constant k (fixed independent of n).

(Due 30 Aug 2023)

5. Here is a problem closely related to quicksort, which we briefly discussed in class. Let X be a totally ordered set with at least n elements. Suppose $x_1, x_2, \ldots, x_n$ are drawn from X uniformly without replacement, and these elements are inserted into a *binary search tree* one after another.

   (a) There is an ordering for which $n(n-1)/2$ comparisons need to be made. How many such orderings are there?

   (b) Suppose $i > j$. We wish to determine the probability that x_i will be compared with x_j, when x_j is eventually inserted. Consider the distribution of $x_1, x_2, \ldots, x_i$ and x_j. Now, x_j is equally likely to appear in any of the $i + 1$ gaps when $x_1, x_2, \ldots, x_i$ are arranged in sorted order. For x_i and x_j to be compared (when x_j is eventually inserted), in which gaps must x_j fall? (Notice how the answer to this part is related to the previous part.)

   (c) Conclude that the expected number of comparisons for building the binary search tree is precisely

$$\sum_{i=1}^{n} \left(\frac{2}{i+1}\right)(n-i),$$

   and show that this quantity is at most $2(n-1)\ln(n+1)$.

6. (a) $A[1..m]$ and $B[1..n]$ are two lists of integers sorted in ascending order. We wish to determine the k-th largest element in the union of A and B. Give an algorithm that runs in time $O(\log m + \log n)$. Assume that the $m + n$ elements are all distinct.

   (b) Suppose $A[1..m; 1..n]$ is an $m \times n$ array of integers. Suppose, first each row of A is sorted independently in ascending order from left to right; then, the columns of A are sorted independently in ascending order from top to bottom. Show that the rows of A remain sorted in the final array.

7. Problem 2.23 of [DPV].

8. Problem 2.32 of [DPV].

9. Suppose we are given a sequence $\mathbf{b} = b_0 b_1 \dots b_{m-1} \in \{+1, -1\}^m$ called text and another shorter sequence $\mathbf{a} = a_0 a_1 \dots a_{n-1} \in \{+1, -1\}^n$ called pattern (we use $\{+1, -1\}$ instead of $\{0, 1\}$), we say that $\mathbf{a}$ occurs in $\mathbf{b}$ at position j if $j \leq m - n$, and $a_k = b_{j+k}$ for $k = 0, 1, \dots, n-1$. Notice that $\mathbf{a}$ occurs in $\mathbf{b}$ at position j iff $\sum_{k=0}^{n-1} a_k b_{k+j} = n$.
   (a) Describe polynomials $A(X)$ and $B(X)$ whose coefficients are derived from $\mathbf{a}$ and $\mathbf{b}$ such that by examining the coefficients of $C(X) = A(X)B(X)$, we can determine if $\mathbf{a}$ occurs in $\mathbf{b}$ at position j.
   (b) Now, suppose some of the elements of the pattern $\mathbf{a}$ are allowed to be $\star$, and we say that $\mathbf{a}$ occurs in $\mathbf{b}$ at position j if $j \leq m - n$ and ($a_k = \star$ or $a_k = b_{j+k}$) for $k = 0, 1, \dots, n-1$. In this new setting, how would you modify the polynomials above to determine if $\mathbf{a}$ occurs in $\mathbf{b}$ at position j?
   (c) Based on the above, what method would you use to determine all positions j such that $\mathbf{a}$ occurs in $\mathbf{b}$ at position j. How long would it take? When is this method preferable to brute force search?

(Due 11 Sep 2023)

10. Suppose $G = (V, E)$ is an undirected unweighted graph with n vertices and m edges. Suppose $s, t \in V$ are vertices of G whose distance in G is strictly greater than $n/2$. Show that there is a vertex (other than s and t) whose deletion disconnects s from t. Describe an algorithm (assume that adjacency lists are available) running in time $O(m + n)$.

11. Suppose $G = (V, E)$ is a connected undirected graph. Suppose DFS starting at a vertex v and BFS starting at the same vertex v produce the same tree. Then, show that G is a tree.

12. Suppose G is a directed graph with n vertices and m edges. Describe an algorithm (assume adjacency lists are available) running in time $O(m + n)$ if G has a vertex v from where every other vertex is reachable.

13. Problem 3.28 (page 106) of [DPV].

14. Problem 4.19 (page 130) of [DPV].

(Due 27 Sep 2023)

15. Consider the Bellman-Ford algorithm (see the code below) for determining the shortest distance in a weighted graph from a source vertex s to all other vertices. For all vertices v, the algorithm maintains $\mathsf{v} \cdot \mathsf{dist}$ and $\mathsf{v} \cdot \mathsf{parent}$. Assume the graph has no negative-weight cycle. Prove or disprove (to disprove provide a counter example) the following statements:

(a) at every point in the execution of the algorithm, for every vertex v with $\mathsf{v} \cdot \mathsf{dist} < \infty$, the directed path (written backwards here):

$$\mathsf{v} \leftarrow \mathsf{v.parent} \leftarrow \mathsf{v.parent.parent} \leftarrow \cdots$$

leads from s to v;

(b) the cost of this path is $\mathsf{v} \cdot \mathsf{dist}$.

16. In the Bellman-Ford algorithm, one picks an edge (u, v) such that

$$\mathsf{u.dist} + \ell(\mathsf{u}, \mathsf{v}) < \mathsf{v.dist},$$

and sets $\mathsf{v.dist} = \mathsf{u.dist} + \ell(\mathsf{u}, \mathsf{v})$. (If no such edges exist, we stop.) To locate the next edge to perform this update operation, the algorithm scans the edges in a fixed order in each iteration. Could we have picked the next edge to update arbitrarily? Show that for all large n, there is an acyclic weighted graph with n vertices and an order of updates (each update should reduce $\mathsf{v.dist}$ for some vertex v), so that the algorithm performs $2^{\Omega(n)}$ updates before it terminates.)

17. Consider the following part of the code for the Bellman-Ford algorithm.

```
s.dist = 0
s.parent = None
for i = 1, 2, ..., T:       # the outer for loop
    for v in V:
        for (w,ell) in adj[v]:
            if v.dist + ell < w.dist:
                w.dist = v.dist + ell    # update distance
                w.parent = v             # update parent
```

Show the following (when appropriate use induction; state the induction hypothesis precisely):

(a) At all times, for all vertices v, if $\mathsf{u} = \mathsf{v.parent}$ is not None, then $\mathsf{u.dist} + \ell(\mathsf{u}, \mathsf{v}) \leq \mathsf{v.dist}$.

(b) If $\mathsf{v.dist}$ was updated in iteration k of the *outer for loop,* then the first $k+1$ elements of the sequence

$$\mathsf{v}, \mathsf{v.parent}, \mathsf{v.parent.parent}, \ldots \tag{1}$$

are not None .

(c) Suppose $T = |V|$ and $\mathsf{v.dist}$ was updated in the last iteration. Argue that the path described in eq. (1) ends in a cycle, and the sum of the lengths of the edges of that cycle is negative (beware of ∞).

(d) Suppose there is a negative-weight cycle C reachable from vertex s. Argue that for some vertex v in C, $\mathsf{v.dist}$ will be updated in iteration $|V|$ of the *outer for loop.*

18. Consider the following algorithm for finding a minimum weight spanning tree in a connected undirected graph. Initially, let T consist of an arbitrary vertex v and no edges. Then, repeatedly add to T the minimum weight edge with exactly one vertex in T. (This algorithm, similar to Dijkstra's algorithm, is called Prim's algorithm.)

(a) Based on the blue and red rules discussed in class, show that the algorithm is correct.

(b) Describe an implementation of the algorithm that runs in time $O((m+n)\log n)$. (Assume that you have an implementation of a heap that supports findmin and deletemin in $O(\log s)$ steps, for a heap of s elements; and can build a heap on s elements in $O(s)$ steps.)

19. Let $(\mathcal{S}, \mathcal{I})$ be a matroid (recall the definition we discussed in class).

(a) Suppose $A, B \subseteq \mathcal{I}$, such that $|A| < |B|$. Show that there is an element $x \in B \setminus A$, such that $A \cup \{x\}$ is independent.

(b) Let I be a maximal independent and let $x \notin I$. Show that there is a unique cycle c contained in $I \cup \{x\}$. (A cycle is a minimal *dependent* set.)

(Due 18 Oct 2023, Wednesday, before the class)

20. (Greedy scheduling) There are n tasks, $T_1, T_2, \ldots, T_n$. We are given n pairs

$$(d_1, p_1), (d_2, p_2), \ldots, (d_n, p_n),$$

where $d_i \in \{1, 2, \ldots, n\}$ refers to the deadline of the i-th task T_i, and p_i is the penalty if T_i is not performed by the deadline. Each task needs one unit-length time slot. We wish to assign to each task a different time slot s_i in the range $\{1, 2, \ldots, n\}$. We say that a task i is delayed under this assignment if $s_i < d_i$. The cost of the assignment is

$$\sum_{j:T_j \text{ is delayed}} p_j.$$

Show that the following greedy strategy produces an optimal solution.

> Consider the task in the monotonically decreasing order of their penalties (consider tasks with higher penalty earlier). When considering task T_i determine if some time slot that helps it meet the deadline d_i is still available. If there is such a slot, set s_i to be the *last* slot that still allows it to meet the deadline. Otherwise, schedule T_i in the last available slot.

State your argument by carefully by establishing that *at each stage, there is an optimal solution that extends the current partial assignment of tasks to slots.* State how you will implement this strategy as an algorithm, and how much time your algorithm will take in the worst case.

21. (Converse of Kraft's inequality) We wish to establish the following claim.

> If $\ell_1, \ell_2, \ldots, \ell_n$ are positive integers such that $\sum_{i=1}^{n} 2^{-\ell_i} \leq 1$, then there are binary strings $w_1, w_2, \ldots, w_n$, where $w_i \in \{0,1\}^{\ell_i}$, and w_i is not a prefix of a w_j (whenever $i \neq j$).

Consider the following greedy method. Maintain a set S consisting of *available* strings. Start with the initial set $S = \{\Lambda\}$, where Λ is the empty string. At the i-th iteration, determine w_i by performing the following steps.

- Find the longest string w in S of length $\ell \leq \ell_i$;
- Set $w_i \leftarrow w0^{\ell_i - \ell} = w\overbrace{000\cdots0}^{\ell_i-\ell}$;
- If $\ell = \ell_i$, then $S \leftarrow S \setminus \{w\}$; otherwise,

$$S \leftarrow (S \setminus \{w\}) \cup \{w1, w01, w001, \ldots, w\overbrace{00\cdots0}^{\ell_i-\ell-1}1\}.$$

(To understand what is happening, draw a tree whose leaves correspond to $w_1, w_2, \ldots, w_i$, and the strings in S.)

(I) What assignment of codewords does this algorithm produce for the following sequence $(\ell_1, \ell_2, \ldots, \ell_8)$: $3, 4, 3, 4, 2, 3, 4, 3$. You may represent the assignment as a tree with leaves labelled $1, 2, \ldots, 8$.

(II) Without assuming anything about the order in which the numbers ℓ_i are presented, show that the above method is correct; in particular, you must state why the algorithm never gets stuck. You might to want use induction to establish that the following hold before each iteration.

(a) The lengths of the strings in S are all _____;

(b) After w_i has been assigned; $\sum_{j>i} 2^{-\ell_j} \leq \sum_{w \in S}$ _____.

22. Suppose the m edges of a graph on vertex set $\{1, 2, \ldots, n\}$ are stored on a tape in the form $e_1, e_2, \ldots, e_m$. Design an algorithm that uses $O(n)$ space (assume that vertices and pointers can be stored in one cell of memory) and after performing one scan of the tape, determines if the graph is bipartite. (Hint: you might want to use a union-find data structure to keep track of the color classes of the graph as it is being built edge by edge.)

23. Problem 6.20 on page 184 of [DPV].

24. You have to prepare a five-volume collection of articles on algorithms. The lengths of the available articles (in number of pages) are as follows: $\ell_1, \ell_2, \ell_3, \ldots, \ell_n$, where ℓ_1 corresponds to the first article published on the subject, ℓ_2 corresponds to the next article, and ℓ_n corresponds to the most recent article. The articles must be published subject to the following constraints.

- No volume is allowed to have more that 300 pages.
- Every selected article must start on a fresh page, and must appear completely in one volume.
- All articles in volume $i + 1$ must have been published *after* all the articles in volume i.

Describe a method based on dynamic programming to determine a plan for publishing the maximum number of articles in the five-volume collection under the above constraints. The output must indicate which articles go into each volume. How long will your algorithm take? [Hint: Let $N[i, j]$ denote the minimum total number of pages (including blank pages at the end of volumes) needed to pick i articles from

the first j. Of course, you don't have to use this hint. For full credit, your solution must remain efficient even if each volume is allowed A pages, and we wish to have at most B volumes in all (then, the running time of the algorithm should be a polynomial of small degree in A, B and n); e.g., a solution that says that there can be at most 1800 articles, so the number of possibilities is bounded by $O(n^{1800})$, would not be considered efficient!]

(Due 1 Nov, Wednesday, before the class)

25. You may use the König-Egerváry theorem proved in class (or Hall's theorem: a bipartite graph $G = (V, W, E)$ has a matching saturating V iff for all subsets $U \subseteq V$, $|N(U)| \geq |U|$).

    (a) Suppose $G = (U, V, E)$ is a bipartite graph where every vertex has the same degree $r \geq 0$. (Such a graph is called an r-regular graph.) Show that G has a perfect matching. Show that $E = E_1 \cup E_2 \cup \cdots \cup E_r$ (disjoint union), where each E_i is a matching. (Make sure your proof works even if E has parallel edges, but no self-loops.)

    (b) A school with $20n$ children is to go on an excursion in n buses, each carrying 20 children. The principal and vice-principal of the school draw up two different plans to assign the children to the n buses. One does not know in advance which plan will be chosen. Now, one needs to assign a bus monitor for each bus from among the children travelling on the bus (and provide them cell phones). Show that there is a common set S of n children who can serve as bus monitors, no matter which plan is eventually chosen.

    (c) Suppose H is a finite group and K is a subgroup of H. Consider the left cosets $L = \{hK : h \in H\}$ and the right cosets $R = \{Kh : h \in H\}$ of K. Show that there is a set $S \subseteq H$ of $|H|/|K|$ elements which can simultaneously serve as coset representatives for the cosets in L and the cosets in R.

26. Let $f^* : E \to \mathbb{R}^{\geq 0}$ (non-negative reals) be the maximum flow in a network $G = (V, E, (c_e : e \in E))$. Let f be a flow in G, and let $\Delta = \mathsf{val}(f^*) - \mathsf{val}(f)$.

    (a) Show that there is a flow g^* in G^f (the residual network corresponding to the flow f) of value Δ. Note that g^* must assign flows only to the edges of G^f and should respect their capacities as defined in G^f.

    (b) Show that there is a path in G^f with bottleneck capacity at least $\Delta/|E|$.

27. Given a residual network G^f as above, show how a path with maximum bottleneck capacity can be found in time $O((|V| + |E|) \log |V|)$. Conclude that for a network with integer capacities, if the flow is repeatedly augmented using an augmenting path of maximum bottleneck capacity, then a maximum flow f^* will be obtained after at most $\lceil m \ln \mathsf{val}(f^*) \rceil$ augmentations. (You may use the inequality $1 + x \leq e^x$, valid for all $x \in \mathbb{R}$.)

28. Problem 7.17 of [DPV].

29. Problem 8.2 of [DPV].

30. Problem 8.4 of [DPV].

31. (Extra credit, simultaneous rounding via network flows) Let $x_1, \ldots, x_n$ be real numbers and let σ be a permutation of $\{1, \ldots, n\}$. We will write

$$S_k = x_1 + \cdots + x_k\,, \qquad \Sigma_k = x_{\sigma(1)} + \cdots + x_{\sigma(k)}\,, \qquad 0 \le k \le n\,,$$

for the partial sums in the two orderings. Assume $0 < x_i < 1$, and $S_n = m$ is an integer. Our goal is to round each x_k to $\bar{x}_k \in \{0, 1\}$ such that the rounded partial sums

$$\bar{S}_k = \bar{x}_1 + \cdots + \bar{x}_k\,, \qquad \bar{\Sigma}_k = \bar{x}_{\sigma(1)} + \cdots + \bar{x}_{\sigma(k)},$$

also satisfy

$$\lfloor S_k \rfloor \le \bar{S}_k \le \lceil S_k \rceil\,, \qquad \lfloor \Sigma_k \rfloor \le \bar{\Sigma}_k \le \lceil \Sigma_k \rceil$$

for $0 \le k \le n$. We will call this *simultaneous rounding*.

(a) Show a rounding algorithm that determines $\bar{x}_1, \ldots, \bar{x}_n$ satisfying the first condition

$$\lfloor S_k \rfloor \le \bar{S}_k \le \lceil S_k \rceil.$$

(b) The required simultaneous rounding will be obtained using network flows. Construct the following network with nodes

$$\{s,\, a_1, \ldots, a_m,\, u_1, \ldots, u_n,\, v_1, \ldots, v_n,\, b_1, \ldots, b_m,\, t\}$$

and the following arcs:

$$\begin{aligned}
s \to a_j &\;:\; \text{for } j = 1, 2, \ldots, m;\\
b_j \to t &\;:\; \text{for } j = 1, 2, \ldots, m;\\
u_k \to v_k &\;:\; \text{for } k = 1, 2, \ldots, n;\\
a_j \to u_k &\;:\; \text{if } [j-1, j) \cap [S_{k-1}, S_k) \neq \emptyset;\\
v_{\sigma(k)} \to b_j &\;:\; \text{if } [j-1, j) \cap [\Sigma_{k-1} \ldots \Sigma_k) \neq \emptyset.
\end{aligned}$$

All edge capacities are 1.

(i) Show that there is an s-t cut of value m in this network.

(ii) Show that there is an s-t flow of value m in this network. (This is the key.)

(iii) Argue that there is an integral flow of value m in this network.

(iv) Use the integral flow of the previous part to construct a simultaneous rounding.

(Due 15 November, if possible)