CS218 Design and analysis of algorithms **Jan-Apr 2025**

Exercises: Greedy, Dynamic Programming

Greedy algorithms

1. Let $G(V, E)$ be a graph with edge weights and $V = V_1 \cup V_2$ be a partition of vertices. Let C be the set of cut edges connecting V_1 with V_2, i.e.,

$$C = \{(u, v) : u \in V_1, v \in V_2\}.$$

   Let e^* be the minimum weight edge in C. Prove that there must exists a minimum weight spanning tree containing e^*. assume V1 has and V2 have mst of own and some f connects them now you can always add e* and remove f to get the ST T' whose weight is minimum.

2. Let $G(V, E)$ be a graph with edge weights and let v_1 be an arbitrary vertex. Let e^* be the minimum weight edge incident on v_1. Prove that there must exists a minimum weight spanning tree containing e^*. make a cut and use cut property of it on minimum weight edge inclusion on every mst.

3. Let $G(V, E)$ be a graph with edge weights. Let e^* be the maximum weight edge. If deletion of e^* disconnects the graph, then e^* must be present in any spanning tree. Else prove that there is a minimum weight spanning tree that does not contain e^*. Design an algorithm based on this idea.
   Kruskal algorithm

4. Interval scheduling: Suppose we have two rooms. And we have a set of requests each with a time interval. We want to allocate the rooms to maximum number of requests. That is, we want to find two subsets S_1, S_2 of intervals with $S_1 \cap S_2 = \emptyset$ such that each of two is a set of disjoint intervals and want to maximize $|S_1| + |S_2|$. Prove that the following greedy algorithm will give an optimal solution.

   Initialize S_1 and S_2 as empty sets. t_1, t_2 will denote the largest finish times among the intervals in S_1 and S_2, respectively. Initially $t_1 = t_2 = 0$. Go over the given intervals in increasing order of finish times. When we are at the i-th interval (a_i, b_i),

   - if $a_i < t_1$ and $a_i < t_2$ then discard this interval
   - if $a_i \geq t_1$ and $a_i < t_2$ then put this interval in S_1.
   - if $a_i < t_1$ and $a_i \geq t_2$ then put this interval in S_2.
   - if $a_i \geq t_1 \geq t_2$ then put this interval in S_1.
   - if $a_i \geq t_2 > t_1$ then put this interval in S_2.

   By exchange arguments make any optimal solution to agree with Greedy's optimal solution

   Prove the optimality of this greedy algorithm. One way to prove this is as follows. Assume that there is an optimal solution that agrees with the choices made by greedy algorithm on first $i - 1$ intervals. Modify this solution to get another optimal solution that agrees with the choices made by greedy algorithm on first i intervals. Then inductively, you can finish the proof.

5. Suppose you are given a set of n course assignments today, each of which has its own deadline. Let the i-th assignment have deadline d_i and suppose to finish the i-th assignment it takes ℓ_i time. Given that there are so many assignments, it might not be possible to finish all of them on time. If you finish an assignment at time t_i which is more than its deadline d_i, then the difference $t_i - d_i$ is called the lateness of this assignment (if $t_i < d_i$ then the lateness is zero). Since you want to maintain a balance among courses, you want that the maximum lateness over all assignments is as small as possible. You want to find a schedule for doing the assignments which minimizes the maximum lateness over all assignments.

   Example: $d_1 = 20, \ell_1 = 10$, $d_2 = 40, \ell_2 = 20$, $d_3 = 60, \ell_3 = 30$. If the assignments are done in order $(1, 3, 2)$ then the maximum lateness will be 20 (for assignment 2). If the assignments are done in order $(1, 2, 3)$ then the maximum lateness will be 0.

   Can you show that a greedy algorithm will give you an optimal solution?

increase in lateness is less than the decrease in lateness of other if swapped according to deadline. Overall reducing the lateness effectively.

- Greedy Strategy 1: Do the assignments in increasing order of their lengths (ℓ_i).
- Greedy Strategy 2: Do that assignment first whose deadline is the closest.
- Greedy Strategy 3: Do that assignment first for which $d_i - \ell_i$ is the smallest.

Strategies 1 and 3 don't work. Give examples to show that they don't work.

Strategy 2 works. Prove that it works correctly as follows. Consider any particular schedule for the assignments. Pick any two assignments A_i and A_j which are adjacent, that is, A_j is done immediately after A_i. If $d_i \leq d_j$ do then nothing, but if $d_i > d_j$ then swap the positions of A_i and A_j. Argue that after this step, the maximum lateness cannot increase.

Repeatedly use the same argument to show that increasing order of deadlines is an optimal schedule.

Pick up the assignments in the decreasing order of rewards and have an array/ of time units to maximum deadline among all. Loop for chosen assignment from it's deadline to lowest time slot available if free slot found then schedule it there if no free slot from it's deadline to first time slot found then discard this deadline

6. Consider another variant. Now, all assignments are equally long, so let's say each takes a unit time to finish. The ith assignment has deadline d_i and a reward r_i. You get the reward only if you finish the assignment within the deadline, otherwise the reward is zero. Design an algorithm to find the maximum possible reward you can get.

7. Hard problem. Consider another variant. For each assignment, you know its deadline d_i and the time ℓ_i it takes to finish it. Suppose you get zero marks for finishing an assignment after its deadline. So, either you should do the assignment within the deadline or not do it at all. How will you find the maximum number of assignments possible within their deadlines.

maximum intersecting interval is the minimum chromatic colors needed to each color each interval differently.

8. Given a set of intervals, you need to assign a color to each interval such that no two intersecting intervals have the same color. Design an efficient algorithm find a coloring with minimum number of colors. To put the problem in another way, given arrival and departure times of trains at a station during the day, what is the minimum number of platforms that is sufficient for all trains.

exchange argument to make it agree with the greedy choice.

9. For the above problem on minimum number of platforms, consider the following greedy strategy. Mark the platforms as $1, 2, 3, \ldots,$. For each platform j, maintain the time t_j at which the last train from that platform departed. Initially, all $t_j = 0$. Sort the trains in increasing order of departure times (finish times). For the i-th train with $(arrival, departure) = (a_i, d_i)$, send it to the platform j such that $t_j \leq a_i$ and which has the largest value of t_j. Prove or disprove that the algorithm utilizes the minimum number of platforms possible.

first condition check if the total sum of the given degree is even or not then use Havel-Hakimi algorithm.

10. Given a list of n natural numbers $d_1, d_2, \ldots, d_n$, we want to check whether there exists an undirected graph G on n vertices whose vertex degrees are precisely $d_1, d_2, \ldots, d_n$ (that is the ith vertex has degere d_i) and construct such a graph if one exists. G should not have multiple edges between the same pair of vertices and should not have self-loop (edges having same vertex as the two endpoints).

Example 1: (2, 1, 3, 2) . The graph with the set of edges $(v_1, v_3), (v_1, v_4), (v_2, v_3), (v_3, v_4)$ has this degree sequence.

sort on decreasing order of degrees, pick one and remove one degree from that number of remaining vertices. If degree of any vertices becomes less than zero then no such graph exists.

Example 2: (3, 3, 1, 1) . There is no graph on four vertices having these degrees.

11. Suppose you are an advertisement company who wants to advertise something to all n people in the city. You know that each of these n people will come to the city center on Sunday for some interval of time. You have acquired these time intervals for all people through some unethical means. You cannot put ads at the city center, but you can pay people to carry your ad on them (maybe by wearing a t-shirt). Assume that if X is carrying the ad, then anyone whose time interval intersects with the time interval of X will see the ad (of course, X will also see the ad). You want to choose minimum number of people to whom you should pay so that everyone sees the ad. Design an algorithm for this and prove its correctness.

12. Suppose you have n objects and you are given pairwise distances between those objects $\{d_{i,j} : 1 \leq i < j \leq n\}$. For a clustering of these objects into k clusters, say $S_1 \cup S_2 \cup \cdots \cup S_k = \{1, 2, \ldots, n\}$, we define the spacing of this clustering is the minimum distance between any two objects that are in different clusters. Formally, the spacing is

$$\min\{d_{i,j} : 1 \leq i < j \leq n, i \text{ and } j \text{ belong to different clusters }\}.$$

We want to find a clustering with k clusters that maximizes the spacing. Can you do this in $O(n \log n)$ time? Think first about the $k = 2$ case.

Dynamic Programming

Flavour 1: where the subproblems are on suffixes/prefixes of the input.

p table calculation is crucial.

13. Work out the remaining details of the problem of finding a set of disjoint intervals that maximizes the total length. Describe an iterative (non-recursive) implementation. Argue that the running time is $O(n \log n)$.

Same as weighted interval scheduling and weight as the length of the interval and "p" calculation
dp[0] = 0, dp[i] = max{ dp[i-1], L[i] + dp[p[i]] } // p is the for any interval last interval in order that do not intersects.

14. The algorithm we described in the class only computed the optimal total length. Update the pseudocode to also output an optimal set of intervals.

After final max length is found we can traceback using "p" table to find the optimal solutions.

15. Design an $O(n \log n)$ time algorithm for the weighted version of the above problem: each interval has a weight and we want to select a subset of disjoint intervals that maximizes the total weight.

dp[0] = 0, dp[i] = max{ dp[i-1], L[i] + dp[p[i]] } , once max weight is calculated we can trace back to find the intervals as well

16. Design an $O(n \log n)$ time algorithm for the counting version of the above problem, that is, we want to find the number of possible subsets of disjoint intervals.

count[0] = 1
little modification to the weighted scheduling will help. when selecting the maxi of dp[i-1] and dp[p(i)], if they are same then we'll keep track of count as well and update as count[i] = count[i-1] + count[p(i)]

17. You are going on a car trip from city A to city B that will take multiple days. On the way, you will encounter many cities. You plan to drive only during the day time and on each night you will stay in one of the intermediate cities. Suppose you can drive at most d kilometers in a day. You are given the distances of the intermediate cities from city A, say, $d_1, d_2, \ldots, d_n$. And distance of B from A is d_{n+1}. You are also given the costs of staying in various cities for one night, say, $c_1, c_2, \ldots, c_n$. Find a travel schedule, that is, in which all cities you should do a night stay, such that your total cost of staying is minimized.

It tells us that it is DAG

18. We are given a directed graph, where each edge goes from a lower index vertex to a higher index vertex. Want to find the longest path from vertex 1 to vertex n.

if dp[i] + 1 > dp[j]: update dp[j] = dp[i]+1, and parent[j] = i;   // parent pointer to keep track of vertices along the longest path.

19. Given a sequence of numbers, find the longest increasing subsequence in $O(n \log n)$ time.

keep an auxiliary array which is referred to as tails[] where each element represents the smallest possible ending of increasing subsequence.

20. Given an array of integers, you want to find a subset with maximum total sum such that no two elements in the subset are adjacent. For example, for the array $\{6, 4, 3, 2, 1, 5\}$, the desired subset is $\{6, 3, 5\}$ with total sum 14. Design an $O(n)$-time algorithm for this problem, where n is the length of the array.

dp[i] = max(dp[i-1], A[i]+dp[i-2]; // dp[0] = A[0], dp[1] = max(A[0], A[1])

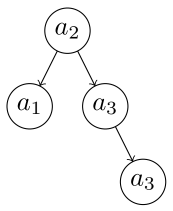
Flavour 2: where the subproblem is on a substring of the input.

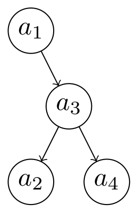
21. The naive algorithm to multiply two matrices of dimensions $p \times q$ and $q \times r$ takes time $O(pqr)$. Suppose we have four matrices A, B, C, D which are $2 \times 4, 4 \times 3, 3 \times 2, 2 \times 5$ respectively. If you multiply $ABCD$ in the order $(AB)(CD)$, it will take time $2 \times 4 \times 3 + 3 \times 2 \times 5 + 2 \times 3 \times 5 = 84$, on the other hand if you multiply in the order $A((BC)D)$, it will take time $4 \times 3 \times 2 + 4 \times 2 \times 5 + 2 \times 4 \times 5 = 104$.

    Given matrices $A_1, A_2, \ldots, A_n$ with dimensions $p_1 \times p_2, p_2 \times p_3, \ldots, p_n \times p_{n+1}$, design an algorithm to find the order in which you should multiply $A_1 A_2 \cdots A_n$ which minimizes the multiplication time.

22. Suppose you have n keys $a_1 < a_2 < \cdots < a_n$ and you want to build a binary search tree that allows efficient search for these keys. If the search queries are distributed uniformly across the keys then it would make sense to build a balanced binary tree. However, if the some keys are more frequent than others then a balanced binary tree might not be the most efficient structure. Consider an example with four keys $a_1 < a_2 < a_3 < a_4$ with the binary search tree shown in Figure 1a.

    Suppose their search frequencies are distributed like $0.7, 0.1, 0.1, 0.1$, respectively. Naturally, we can assume that the search time for a key is proportional to its depth (distance from the root). Then, the average search time for this search tree (Figure 1a) will be $0.7 \times 2 + 0.1 \times 1 + 0.1 \times 2 + 0.1 \times 3 = 2$.


(a) A binary search tree


(b) A binary search tree

Now, consider another search tree in Figure 1b. The average search time for this tree will be $0.7 \times 1 + 0.1 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 1.5$, which is better than the first search tree.

Given search frequencies of n keys, say $f_1, f_2, \ldots, f_n$, we want to find the binary search tree that minimizes the average search time defined as $\sum_i f_i d_i$, where d_i is the depth of the node containing a_i.

23. Segmented least squares. Section 6.3 from Kleinberg Tardos

    Flavour 3: where the subproblem (recursive call) has two different parameters.

24. A subsequence of a string is obtained by possibly deleting some of the characters without changing the order of the remaining ones. For example, *ephn* is a subsequence of *elephant*.

    You are given a string A of length n and another string B of length m $(\leq n)$. If we want to check whether A contains B as a subsequence, there is greedy algorithm for it: For $1 \leq i \leq m$, match the character $B[i]$ with its first occurrence in A after the matching of $B[i-1]$.

    For example, if $A = bacbcbabcacba$ and $B = bcbca$, then the greedy approach will match B as follows (shown in red).

    $$bacbcbabcacba$$

    Now, suppose to match with the j-th character in string A, you have to pay a cost p_j. And to match B with a subsequence in A, you have to pay the sum of costs of the matched characters.

    In the above example, if the prices for the characters in A were $2, 5, 3, 1, 2, 5, 3, 1, 3, 1, 2, 4, 1$ then the matching *bacbcbabcacbaa* has cost $2 + 3 + 1 + 2 + 3 = 11$. While the matching *bacbcbabcacba* has cost only $1 + 2 + 1 + 2 + 1 = 7$.

    Design an algorithm that given A, B and $p_1, p_2, \ldots, p_n$, can find the minimum cost subsequence in A that can be matched with B. Ideally your algorithm should run in time $O(mn)$.

25. **Knapsack problem.** Suppose there are n objects with their weights being $w_1, w_2, \ldots, w_n$ and their values being $v_1, v_2, \ldots, v_n$. You need to select a subset of the objects such that the total weight is bounded by W, while the total value is maximized. Your algorithm should run in time poly(n, W).

    Consider the case when the weights are too large, that is, they are exponential in n. Then the above algorithm is not really efficient. Suppose on the other hand, then values $\{v_1, v_2, \ldots, v_n\}$ are small. Can you design an algorithm running in time poly$(n, \sum_i v_i, \log W)$?

    Flavour 4: where subproblems are parameterized by a prefix and something more

26. Suppose you have a movable shop that you can take from one place to another. You usually take your shop to one of the two cities, say A and B, depending on whichever place has more demand. Suppose you have quite accurate projections for the earnings per day in both the cities for the next n days. However, you cannot simply go to the higher earning city each day because it takes one whole day and costs c to move from one city to the other. For example, if you are in city A on day 5 and want to move to city B, then on day 6 you will have no earnings, you will pay a cost of c and on day 7 you will have earnings of city B. Design a polynomial time algorithm that takes as input the moving cost ,the earnings per day in the two cities say, $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, and outputs a schedule for the n days that maximizes the total earnings. Assume that you can start with any of the two cities on day 1, without costing anything.

27. Kleinberg Tardos Section 6.5 RNA secondary structure
28. Kleinberg Tardos Exercises 10, 16, 28
29. Recall the Bellman Ford algorithm seen in the class. How will you find the optimal path?
30. What does Bellman Ford algorithm output if there are negative weight cycles? Can you detect negative cycles if there exists one.