CS 180 Homework 6

Jiaping Zeng

11/29/2020

6.20 **Algorithm**:

1. Let $G(c, h)$ be the maximum total grade for c courses using h hours. Set $G(0, h) = 0$ and $G(c, 0) = \sum_{i=0}^{c} f_i(0)$.
2. For each course i, do the following:
   - For $0 \leq h \leq H$, set $G(i, h) = \max_{0 \leq t \leq h}\{f_i(t) + G(i-1, h-t)\}$.
   - Store the corresponding t that maximizes $G(i, h)$.
3. $G(n, H)$ is our maximum total grade, backtrack to retrieve the hours spent for each course.

Proof of correctness: By induction on n, the number of courses.
Base case: $n = 0$, since $G(0, h)$ is initialized to 0 in step 1, our algorithm returns the correct result.
Inductive step: Suppose our algorithm is correct for n courses, we want to show that it is also correct for $n + 1$ courses. Note that when calculating $G(n + 1, h)$, we are finding the maximum of $G(n, h - t)$ values for various t. By inductive hypothesis, these $G(n, h - t)$ values are correct. So our algorithm is correct for $n + 1$ courses.
Therefore our algorithm is correct by induction.
Time complexity: $O(nH^2)$; looping through each course is $O(n)$, then looping through H is $O(H)$ and finding the maximum grade each time is $O(H)$. Therefore our main loop is $O(nH^2)$ and backtracking is $O(n)$, so overall our algorithm is $O(nH^2)$.

6.21 **Algorithm**:

1. Let $R(i, j)$ denote the maximum return of a buy b and sell s between days i and j, i.e. $R(i, j) = \max p(s) - p(b)$ for $i \leq b \leq s \leq j$.
2. For each interval between days i and j, set $R(i, j) = \max\{p(i) - p(j), R(i, j - 1), R(i + 1, j)\}$.
3. Let $P(m, d)$ be the maximum profit of an m-shot strategy over d days. Set $P(0, \cdot) = 0$. In addition set $P(m, d) = -\infty$ if $2m > d$ since it is not possible to execute such strategies.
4. For each $1 \leq m \leq k$ and $1 \leq d \leq n$, set $P(m, d) = \max_{1 \leq i < j \leq d} R(i, j) + P(m - 1, i - 1)$. Store the days i and j that corresponds to the maximum profit strategy.
5. $P(k, n)$ is the maximum profit of the k-shot strategy, backtrack using stored intervals to retrieve the buy/sell dates.

Proof of correctness: By induction on n, the number of days.

Base case: $n = 2$, then we return a strategy with $-\infty$ profit for $k \geq 2$ as such strategies are not possible; for $k = 0$ we return a strategy with no profit since there is no transaction; for $k = 1$ we return a strategy that buys on day 1 and sells on day 2 as expected.

Inductive step: Suppose our algorithm works a k-shot strategy with n days, we want to show that it will also work for one with $n+1$ days. Note that in the calculation of $P(k, n+1)$, we have $1 \leq i \leq d \leq n+1 \implies i-1 \leq n$, so $P(m-1, i-1)$ would cover only strategies over n days or less. By inductive hypothesis we will always have the correct value for those, so we will also have the correct value for $P(k, n+1)$.

Therefore our algorithm works by induction.

Time complexity: $O(kn^2)$; constructing $R(i,j)$ takes $O(n^2)$. Calculating $P(m,d)$ takes $O(kn)$ for the outer loop and $O(n)$ per iteration, so it is $O(kn^2)$. Therefore overall our algorithm is $O(kn^2)$.

6.24 **Algorithm**: Let A be the total number of party A votes and a_i denote the number of party A votes in precinct P_i. Note that party A would need $\frac{nm}{4} + 1$ votes in each district to win.

1. Let $G(p,q,v)$ return true there is a set of p precincts out of the first q precincts that contains exactly v party A votes. Set $G(0,\cdot,0) = True$.
2. For each precinct $1 \leq p \leq \frac{n}{2}$, then for each precinct $1 \leq q \leq n$, do the following:
   - For $1 \leq v \leq A - \frac{nm}{4} - 1$, set $G(p,q,v) = True$ if $p = q = 1$ and $v = a_1$, or if $G(p-1, q-1, v-a_p) = True$, or if $G(p, q-1, v) = True$.
   - Else set $G(p,q,v) = False$.
3. Return true if any $G(\frac{n}{2}, n, v)$ for $\frac{nm}{4} + 1 \leq v \leq A - \frac{nm}{4} - 1$ is true.

Proof of correctness: By induction on n, the number of precincts.

Base case: we have 1 precinct, then our algorithm checks if we have $v = a_1$ and returns correspondingly, as expected. (This does not make much sense in terms of gerrymandering since 1 precinct cannot be split up into districts, however it is a functional base case.)

Inductive step: Suppose our algorithm works for n precincts, we want to show that it will also work for $n+1$ precincts. Note for the $n+1$ iteration, our q in $G(p,q,v)$ is at most n which works by inductive hypothesis. Therefore our algorithm must also return the correct result for $n+1$ precincts.

Therefore our algorithm works by induction.

Time complexity: $O(n^3m)$; our main loop is $O(n^3m)$ and checking entries of G takes $O(nm)$, so overall the algorithm is $O(n^3m)$.

7.8 (a) **Algorithm**:

1. We set up a four-levels graph as follows:
   - Level 1: source node
   - Level 2: four supply nodes, one for each blood type
   - Level 3: four demand nodes, one for each blood type
   - Level 4: sink node

2. Connect the source node to each supply node with capacity set to the supply; similarly, connect the sink node to each demand node with capacity set to the demand.
3. Between the supply nodes and demand nodes, construct an edge only if the demand type can resource from the supply type. Set capacity to the demand.
4. Run Ford-Fulkerson to find the max-flow. Return true if our max-flow saturates demand edges to the sink and false otherwise.

Proof of correctness: By contradiction; suppose our algorithm is incorrect, then it would either return true for insufficient supplies or false for sufficient supplies, both of which contradicts with that Ford-Fulkerson is correct. Therefore our algorithm also return the correct result by contradiction.

Time complexity: $O(em)$ since Ford-Fulkerson is $O(em)$ where $e = |E|$ and m is the max-flow.

(b) There is not enough supply since we have 86 units of supplies of types O and A with 87 units of demand and they cannot receive from other types.

We can first use all supplies of AB and O to fulfill their demands, afterwards the remaining cannot be used for other types. So then we have 50 units of supply for type O with 45 units of demand; similarly we have 36 units of supply for type A with 42 units of demand. We can first fulfill the type O demand with 5 units leftover, which we can use to help with type A demand, leaving us with $42 - 36 - 5 = 1$ person who does not receive blood.

7.12 **Algorithm**:

1. Run Ford-Fulkerson algorithm; from the residual graph, find the reachable vertices. Store all edges from a reachable vertex to a nonreachable vertex as the min-cut edges.
2. If we have more than k edges, select any k edges from them. Return such edges.
3. Else if we have less than k edges, add any remaining edges until we have k edges. Return the edges.

Proof of correctness: By contradiction. Suppose our algorithm does not give us the desired edges, i.e. the edges are not in the min-cut. However, since Ford-Fulkerson is correct, we will always have edges in the min-cut. Therefore our algorithm is correct by contradiction.

Time complexity: $O(em)$ since Ford-Fulkerson is $O(em)$ where $e = |E|$ and m is the max-flow.

P6 **Algorithm**: We will assume that all elements in the input sequence are unique.

1. Let $S_>(i)$ be the length of the longest alternating subsequence ending at index i with the last element greater than the previous element; let $S_<(i)$ be the same but with the last element smaller than the previous element. Set $S_>(\cdot) = S_<(\cdot) = 1$.
2. For each element x_i of the list, do the following:
   - For $0 \leq j < i$, compare x_i and x_j.
   - If $x_i > x_j$, set $S_>(i) = \max\{S_>(i), S_<(j) + 1\}$.
   - Else if $x_j > x_i$, set $S_<(i) = \max\{S_<(i), S_>(j) + 1\}$.

3. $\max\{S_>(i), S_<(i)\}$ is the length of our longest alternating subsequence, backtrack to retrieve the subsequence and return it.

Proof of correctness: By induction on n, the length of the given sequence.

Base case: $n = 1$; the loop in step 2 will simply be skipped since there is no $j < i$. Then we return the original sequence as the longest alternating subsequence.

Inductive step: Suppose our algorithm works for an input sequence of length n, we want to show that it will also work for an input sequence of length $n + 1$. For the last element x_{n+1}, our algorithm will calculate either $\max\{S_>(n + 1), S_<(j) + 1\}$ or $\max\{S_<(n + 1), S_>(j) + 1\}$. $S_>(n + 1)$ and $S_<(n + 1)$ will be correct as we initialized them to 1 at the start, and we are only updating them respectively if values of $S_<(j) + 1$ and $S_>(j) + 1$ are greater. Note that since we set $j < i$, j is at most n. Then by inductive hypothesis our $S_>(j)$ and $S_<(j)$ calculations will be correct.

Therefore our algorithm is correct by induction.

Time complexity: $O(n^2)$; looping through x_j for every x_i is $O(n^2)$, then backtracking to retrieve the subsequence is $O(n)$, so overall the algorithm is $O(n^2)$.