CS 180 Final

Jiaping Zeng

12/16/2020

P1 (a) Each points on the left within our selected distance is compared to at most 8 points since our distance d is chosen to be less than the distance of any two points. So we only need to check 8 squares of side length $\frac{d}{2}$.

(b) We keep it sorted by sorting the points initially and breaking it down each time. Since the list is originally sorted, any sublist broken down from it will also be sorted.

P2 (a) By contradiction; suppose we have a minimum spanning tree of G' with weight w that includes an edge not in T. But this implies that there is a tree in G with weight less than the minimum spanning tree, which is a contradiction by definition.

(b) Since the edges remain sorted after increasing the weight, our minimum spanning tree algorithm will be the same. Then after running the algorithm, which will give us a minimum spanning tree, we can simply decrease the weight back to the original.

P3 We assume that there are no duplicate numbers.

Algorithm:

1. We first undo the shift as follows:
   - Go to the element in the middle of B and compare it to its neighbors. Take the floor of $n/2$ if n is odd.
   - If the number is smaller than $B[0]$, we repeat the process on the left half. Otherwise, repeat on the right half.
   - Once we have the minimum, its index is the shift. Undo the shift to retrieve A.
2. Run binary search on A to see if X is in it.

Proof of correctness: We will show that our algorithm will find the correct shift by induction on n, the number of elements.

Base case: $n = 1$; there is no shift, then bisection will return the only number, as expected.

Inductive step: Suppose our algorithm works for a list of n elements or fewer, we want to show that it will also work for a list of $n + 1$ elements. We will examine the following scenarios:

- n is odd: our algorithm will divide the $n + 1$ elements into two sublists, then we need to make sure it check the correct sublist. Since A was sorted and if the center element is smaller than $B[0]$, then the minimum has to be on the left half. Proceed analogously otherwise.
- n is even: then $n + 1$ is odd and our algorithm will take the floor of $\frac{n+1}{2}$ and proceed identically as it would on a list with n elements. Therefore by inductive hypothesis the shift will be correct.

Therefore our algorithm will always find the correct shift by induction, and since binary search is correct our algorithm will return the correct result.

Time complexity: $O(\log n)$; undoing the shift takes $O(\log n)$ (we cut the list in half each time) then binary search takes $O(\log n)$, so overall it is $O(\log n)$ as well.

P4 We assume that there will be no duplicate numbers in the given list; if there are duplicates we can simply remove them as the same number cannot appear more than once in the solution anyway.

Algorithm:

1. Denote $f(s, r, q, p) = A[s] - A[r] + A[q] - A[p]$ and n as length of the array. Let $M(s, q)$ denote $\max_{s>r>q>p} f(s, r, q, p)$, i.e. fix s and q, $M(s, q)$ is the max for arbitrary r, p such that $s > r > q > p$.
2. For s such that $3 \leq s < n$ and q such that $0 < q < s - 1$, do the following:
   - Find r and p such that r corresponds to $\min_{s>r>q} A[r]$ and p corresponds to $\min_{q>p>0} A[p]$.
   - Store $M(s, q) = f(s, r, q, p)$.
3. Return $\max M(s, q)$.

Proof of correctness: By induction on n, the size of the array.

Base case: $n = 4$, then by our setup we must have $s = 3$ and $q = 1$. Then the sum of the four elements is returned, as expected.

Inductive step: Suppose our algorithm works for n elements, we want to show that it will also work for $n + 1$ elements. Note that with $n + 1$ elements, we would need to evaluate an extra $M(n + 1, q)$, which will be correct since f and min are correct. Since our algorithm works for $M(n, q)$, the rest of the table is also correct. Therefore the correct max value of the table will be returned.

Time complexity: $O(n^3)$; finding s and r takes $O(n^2)$ and choosing r and p each time takes $O(n)$ since we traverse the list once to retrieve both values, so our algorithm is $O(n^3)$ overall.

P5 **Proof**: ST-Hamiltonian path is polynomial time as we need to check every vertex for a path. We can also find a Hamiltonian path with ST-Hamiltonian in polynomial time by checking any possible combinations of s and t. Since Hamiltonian path problem is NP-Complete, so is ST-Hamiltonian path problem.

P6 **Algorithm**:

1. We set up a four-levels graph as follows:
   - Level 1: source node
   - Level 2: a node for each team
   - Level 3: a node for each table
   - Level 4: sink node
2. Connect the source node to each team node with capacity set to the number of members; similarly, connect the sink node to each table with capacity set to the number of chairs at the table.
3. Between the team nodes and table nodes, construct an edge of capacity 1 from each team to each table.
4. Run Ford-Fulkerson to find the max-flow and return it.

Proof of correctness: By contradiction; suppose our algorithm is incorrect, then it must have returned a result where either the teams are not fully seated or if a table has more than 1 members from the same team. The first case cannot happen as we have $M \geq N$, so everyone should be able to find a seat. The second case is not possible since Ford-Fulkerson will not overflow an edge, i.e. no table will have more than 1 member from the same team. Therefore our algorithm is correct by contradiction.

Time complexity: $O(k|E|)$ where k is the max flow since that is the complexity of Ford-Fulkerson.