CS 341, Winter 2015
Andrew Arnold, Timothy Chan, Doug Stinson

Assignment 5 (due **Thursday**, April 2, 4pm)

Please read `https://www.student.cs.uwaterloo.ca/~cs341/policies.html` for general instructions.

1. [*12 marks*] Lieutenant John McClane and shop owner Zeus Carver are by a water fountain and are given two water jugs: one jug that holds n_1 liters of water, the other n_2 liters (where n_1 and n_2 are positive integers). They are allowed to perform the following operations:

   (i) completely fill a jug with water;

   (ii) completely empty a jug of water;

   (iii) pour the contents of one jug into the other jug, stopping only when either the jug being filled is full, or when the jug being emptied is empty.

   Their task is to determine if there is a sequence of such operations that leaves exactly n_3 liters of water in the bigger jug (where n_3 is another positive integer).

   Thankfully, John McClane has taken a third-year algorithms course and realizes that he can pose the problem as a graph problem.

   (a) [*9 marks*] Model the problem as a graph problem. Give a precise definition of the graph involved and state the specific question about this graph that needs to be answered. [*Hint*: define a vertex to be a pair of integers (i, j) where $0 \le i \le n_1$ and $0 \le j \le n_2$.]

   (b) [*3 marks*] What algorithm can be applied to solve the problem and what is its time complexity as a function of n_1 and n_2?

   Answer:

   (a) Without loss of generality suppose $n_1 \ge n_2$
   Per the hint, we define a graph whose vertices are the tuples (i, j), for $0 \le i \le n_1$, $0 \le j \le n_2$. Each tuple (i, j) represents the state where the first jug contains i litres of water and the second jug contains j litres of water. We add a directed edge from vertex u to vertex v if there is an operation to change from the corresponding state of u to that of v. Namely, we add edges

   (i) $((i, j), (n_1, j))$,
   $((i, j), (i, n_2))$,

   (ii) $((i, j), (0, j))$,
   $((i, j)), (i, 0))$,

   (iii) $((i, j), (\max(i + j - n_2, 0), \min(n_2, i + j))$,
   $((i, j), (\min(n_1, i + j), \max(i + j - n_1, 0))$.

The set of vertices reachable from $(0,0)$ correspond to the set of possible states we can reach. There exists a way to get n_3 litres of water in the bigger jug if and only if there exists a path from $s = (0,0)$ to a vertex of the form (n_3, j) for some j.

(b) We can perform a breadth or depth first search on the graph, starting from $(0,0)$. As our graph has $\mathcal{O}(n_1 n_2)$ vertices and $\mathcal{O}(1)$ edges leaving every vertex, the total cost of the search is $\mathcal{O}(|V| + |E|) = \mathcal{O}(n_1 n_2)$.

2. [*12 marks*] Given a set P of n points in 2D, we consider the following *2-clustering problem*: partition P into two subsets P_1, P_2 to maximize

$$d(P_1, P_2) = \min_{p \in P_1} \min_{q \in P_2} d(p, q),$$

where $d(p,q)$ denotes the Euclidean distance between p and q. (That is, $d(P_1, P_2)$ denotes the smallest distance between P_1 and P_2.)

(a) [*4 marks*] Let G be the complete graph with vertex set P where the edge pq has weight $w(pq) = d(p,q)$. Let T be the minimum spanning tree of G. Let e be the largest-weight edge in T. Deleting e from T creates two components P_1, P_2. Prove that $d(P_1, P_2) = w(e)$. [*Hint*: you may use a lemma proved in class. You may assume that all edge weights are distinct.]

(b) [*4 marks*] Let e be the edge from part (a). Let P_1^*, P_2^* be the optimal solution. Prove that $d(P_1^*, P_2^*) \leq w(e)$. [*Hint*: again use the lemma from class.]

(c) [*4 marks*] Finally show that the 2-clustering problem can be solved in $O(n^2)$ time by applying an algorithm from class. Use parts (a) and (b) to argue correctness.

Answer:

(a) The key lemma we would like to use from class states that, for any partition of V into two sets P_1 and P_2, the minimum spanning tree (MST) of $G = (V, E)$ contains the least-weight edge over all edges $(p,q) \in E$ with $p \in P_1, q \in P_2$. In other words, the minimum spanning tree contains the edge of weight $\min_{p \in P_1} \min_{q \in P_2} w(p,q)$. Note that the only edge in the minimum spanning tree between P_1 and P_2 is e. It follows that $d(P_1, P_2) = w(e)$.

(b) Let T^* be the minimum spanning tree of G. As P_1^* and P_2^* form a partition of V, then by the previously aforementioned lemma, T^* will also contain the least-weight edge (u,v) such that $u \in P_1^*$, $v \in P_2^*$. As e is the maximal-weight edge in the MST of G, $w(e) \geq w(u,v)$. Thus we have that

$$d(P_1^*, P_2^*) = \min_{p \in P_1^*} \min_{q \in P_2^*} w(p,q) = w(u,v),$$

and $w(u,v) \leq w(e)$ by the maximality of e.

(c) We solve the 2-clustering problem by computing the MST T^* of G. We then select the maximal-weight edge of T^*, and remove it to produce two sets of vertices P_1, P_2. We can use a depth-first search in order to produce the vertices in each set P_1 and P_2. Per part (a) this gives a solution with value $d(P_1, P_2) = w(e)$. Per part (b) this value is optimal, and thus the algorithm is correct.

We can construct the MST using Prim's algorithm, using either no data structure with cost $\mathcal{O}(n^2 + m)$, or Fibonacci heaps with cost $\mathcal{O}(n \log n + m)$, where $n = |V|, m = |E|$. As $m \in \mathcal{O}(n^2)$, either gives us cost $\mathcal{O}(n^2)$. Performing a depth-first search on each half of the tree will cost $\mathcal{O}(n)$, as the MST has $\mathcal{O}(n)$ edges.

3. [*15 marks*]

(a) [*6 marks*] Convert the following optimization problem into a decision problem and show that the corresponding decision problem is in NP.

Input: positive integers $v_1, \ldots, v_n, w_1, \ldots, w_n, W_1, W_2$.

Output: disjoint subsets $A, B \in \{1, \ldots, n\}$ such that $\sum_{k \in A} w_k \leq W_1$ and $\sum_{k \in B} w_k \leq W_2$, while maximizing the total value $\sum_{k \in A} v_k + \sum_{k \in B} v_k$.

[*Note*: You have seen this problem before! Remember that the bit complexity of the input is polynomial in n and $\log U$, where U is the largest integer in the input.]

(b) [*3 marks*] Show that if we could solve your decision problem in (a) in time polynomial in the number of input bits, then we could also compute the maximum total value in time polynomial in the number of input bits.

(c) [*6 marks*] Show that if we could solve your decision problem in (a) in time polynomial in the number of input bits, then we could also compute the optimal subsets A and B in time polynomial in the number of input bits. [You may use part (b) as a subroutine.]

Answer:

(a) The decision problem takes as input the same input as the optimization problem, and a nonnegative integer V. The output should be "yes" if and only if there exist $A, B \in \{1, 2, \ldots, n\}$ such that $\sum_{k \in A} w_k \leq W_1$, $\sum_{k \in B} w_k \leq W_2$, and $\sum_{k \in A \cup B} v_k \geq V$. For simplicity we will define U to be the largest integer input for a given problem instance, such that the input size is $\mathcal{O}(n \log U)$.

The certificate for this problem are the subsets A and B. We can represent A and B as lists containing the integers comprising each, with space $\mathcal{O}(n \log U)$. We could also represent A and B as vectors of n bits, where the i-th bit of the vector for A is set to 1 if and only if $i \in A$, and similarly for B. In either case we have that the certificate has polynomial size with respect to the input size.

Verification comprises testing that $\sum_{k \in A} w_k \leq W_1$, $\sum_{k \in B} w_k \leq W_2$, and $\sum_{k \in A \cup B} v_k \geq v$. None of these sums will exceed nU. As integer addition has linear cost in the bit-size of its inputs, computing these sums would cost $\mathcal{O}(n^2 \log U)$, which is polynomial time. It follows that this decision problem is in NP.

(b) Since all the inputs are positive integers, the optimal value v^* is also a positive integer. Moreover, clearly v^* cannot exceed $\sum_{i=1}^n v_i$. Thus, if there exists an algorithm to solve the decision problem in polytime, we could use that algorithm to do a binary search on v^* in the range $[0, \sum_{i=1}^n v_i]$

Let $f(v)$ be the result of this decision problem, obtained by a polynomial-time algorithm, for a given value of v. We can solve the optimization problem as follows:

```
1. $(\ell, h) \leftarrow (0, \sum_{i=1}^n v_i)$
2. while $\ell < h$ do
3.     $v \leftarrow \lceil (h+\ell)/2 \rceil$
4.     if $f(v) =$ "yes" then $\ell \leftarrow v$
5.     else $h \leftarrow v - 1$
6. Output $v^* \leftarrow \ell$
```

As $h - \ell$ decreases by at least half every iteration of the while loop, the total number of iterations is $\mathcal{O}(\log(\sum_{i=1}^n v_i)) \in \mathcal{O}(\log(nU))$. Moreover, $f(v)$ may be computed in time $\mathcal{O}(p(n + \log U))$, for some polynomial U. Thus this approach takes $\mathcal{O}(p(n + \log U)\log(nU))$ time, which is polynomial in $n + \log U$. This is polynomial in the input size.

(c) Let v^* be the optimal value for the original optimization problem. Consider the modified optimization problem with values $(v_2, \ldots, v_n)$, weights $(w_1, \ldots, w_n)$, and bounds $W_1' = W_1 - w_1$ in place of W_1, and W_2. Let v' be the optimal value for this problem. Observe that the original problem has an optimal solution (A^*, B^*) with $1 \in A^*$if and only if $v^* - v_1 = v'$. The remainder of A^* and B^* may be obtained by removing item 1 from the problem and iterating over the remaining items. This entails updating W_1 to W_1' and v^* to v'. The analogous result in the case that $i \in B^*$ is straightforward. The pseudocode below describes this procedure.

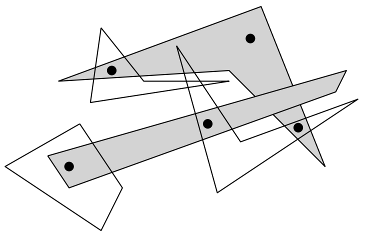
```
1. $A, B \leftarrow \{\}$
2. $v^* \leftarrow$ sol'n to opt. problem with inputs $(v_1, \ldots, v_n), (w_1, \ldots, w_n), W_1, W_2$
3. for $i \leftarrow 1$ to $n$ do
4.     $v' \leftarrow$ sol'n to opt. problem with inputs $(v_{i+1}, \ldots, v_n), (w_{i+1}, \ldots, w_n), W_1 - w_i, W_2$
5.     $v'' \leftarrow$ sol'n to opt. problem with inputs $(v_{i+1}, \ldots, v_n), (w_{i+1}, \ldots, w_n), W_1, W_2 - w_i$
6.     if $v' = v^* - v_i$ then
7.         $A \leftarrow A \cup \{v_i\}$
8.         $W_1 \leftarrow W_1 - w_i$
9.         $v^* \leftarrow v'$
10.    else if $v'' = v^* - v_i$ then
11.        $B \leftarrow B \cup \{v_i\}$
12.        $W_2 \leftarrow W_2 - w_i$
13.        $v^* \leftarrow v''$
14. Output $A, B$
```

4. [*21 marks*] A quadrilateral is a polygon with 4 vertices (which may or may not be convex). Consider the following problem called Quad-Cover:

Input: a set P of m points, a set Q of n quadrilaterals in 2D, and an integer k.

Output: "yes" iff there exists a subset $T \subseteq Q$ of at most k quadrilaterals such that every point in P lies inside some quadrilateral in T.

In the following example, the answer is "yes" for $k = 2$ (with the optimal subset shaded).



(a) [*4 marks*] Prove that QUAD-COVER is in NP.

Note: you may assume that testing whether a point is inside a quadrilateral can be done in time polynomial in the number of bits.

(b) [*14 marks*] Prove that QUAD-COVER is NP-complete via a polynomial-time reduction (i.e., a polynomial-time transformation) from VERTEX-COVER.

Note: You may use the fact from class that the following problem VERTEX-COVER is NP-complete:

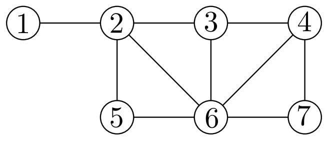
Input: an undirected graph $G = (V, E)$ and an integer k.

Output: "yes" iff there exists a subset $S \subseteq V$ of at most k vertices such that for every edge $uv \in E$, we have $u \in S$ or $v \in S$.

More Note: Remember to carefully define your construction of the input to QUAD-COVER given the input to VERTEX-COVER, and remember to prove that your reduction is correct.

Hint: Given a graph $G = (V, E)$ where $V = \{1, \ldots, n\}$, to construct the set P of points, map each $ij \in E$ $(i < j)$ to a point (i, j). For the set Q, you will need to define certain non-convex quadrilaterals to cover particular rows/columns somehow...

(c) [*3 marks*] Illustrate your reduction for the following graph G with $k = 3$:



Draw the set P of points and the set Q of quadrilaterals in your construction. Also draw the solution T that corresponds to the vertex cover $S = \{2, 4, 6\}$ of G.

Answer:

(a) First note that the input size must be at least $\Omega(m + n)$, in order to store the m points and n quadrilaterals.

The certificate comprises the k quadrilaterials that comprise the subset T. This is clearly polynomial in the input size, as it is a subset of the input.

Verification entails testing if each point $p \in P$ is contained within a quadrilateral $q \in T$, we can do this by testing if $p \in P$ for each $q \in T$. This requires $mk \in \mathcal{O}(mn)$ tests. Note the number of tests in polynomial in $m + n$, and hence must be polynomial in the input size. As we assume each such test is polynomial time, it follows that this verification is polynomial time. Thus QUAD-COVER is in NP.

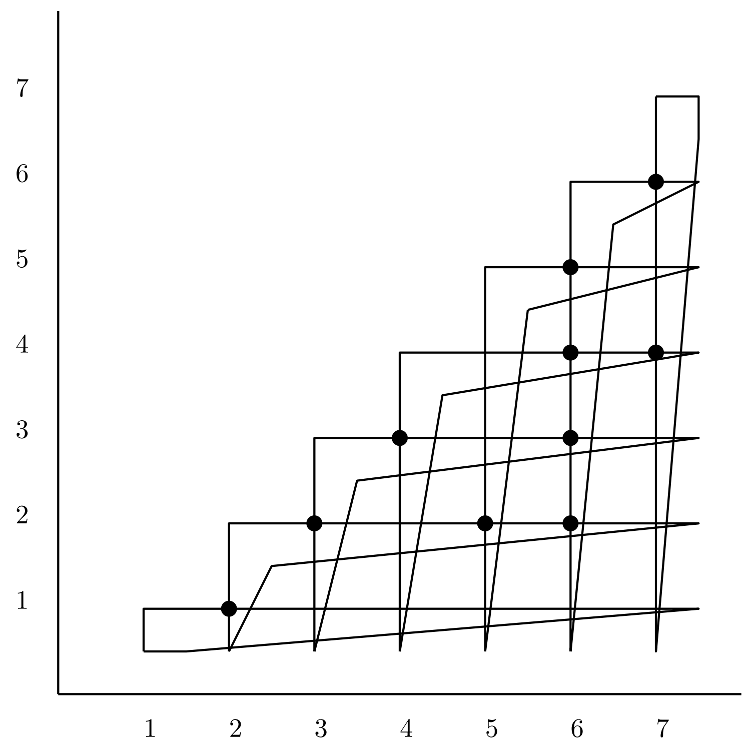
(b) Suppose we are given an instance $(G = (V, E), k)$ of VERTEX-COVER, where $V = \{1, 2, \ldots, n\}$. From this we will construct an instance of QUAD-COVER. We will let our set of $2D$ points be $P = \{(i, j) | ij \in E, i > j\}$. We will construct a set of n quadrilaterals $Q = \{q_1, \ldots, q_n\}$, where q_ℓ is given by the four $2D$ points (ℓ, ℓ), $(n+0.5, \ell)$, $(\ell+0.5, \ell-0.5)$, and $(\ell, 0.5)$, given in clockwise order. We assume that each quadrilateral contains its boundary. We make the following observation: the set of points with integer coordinates contained in q_ℓ are exactly the points (x, ℓ) and (ℓ, y) for $1 \leq y \leq \ell \leq x \leq n$.

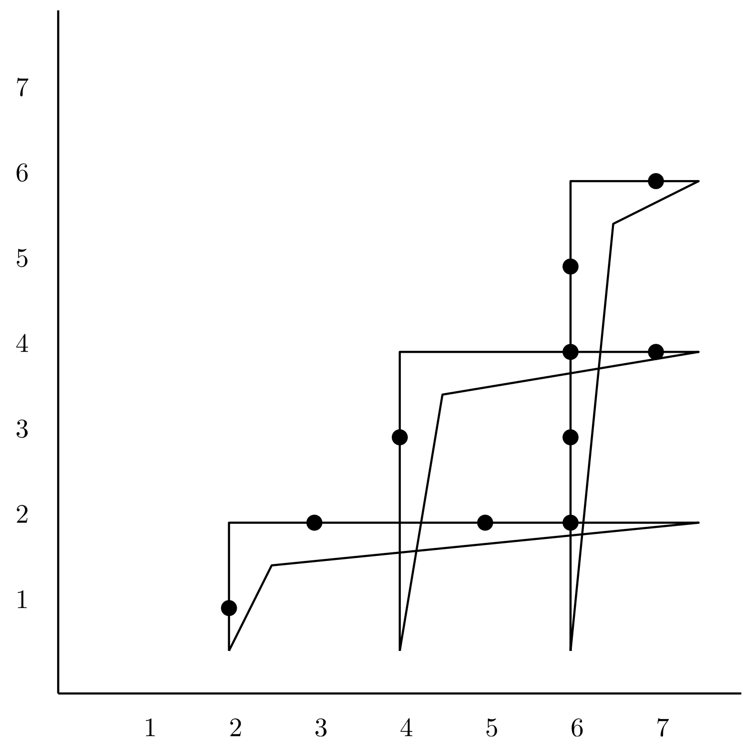
We take as our instance of QUAD-COVER the set of points P and quadrilaterals Q as described, with k the same as in the instance of VERTEX-COVER. This construction takes linear-time in the input size of VERTEX-COVER, i.e., this is a polynomial-time reduction.

It remains to prove the reduction is correct. Suppose that the answer for instance (G, k) is "yes". Then there exists S, a subset of at most k vertices, such that for every edge $ij \in E$, either $i \in S$ or $j \in S$, or both. Let $T = \{q_i | i \in S\}$. Clearly $|T| = |S| \leq k$. Let $(i, j) \in P$. By our construction $i > j$. If $i \in S$, then $q_i \in T$, and by our observation $(i, j) \in q_i$. Similarly if $j \in S$ we must have $(i, j) \in q_j \in T$. In either case we have p is in a quadrilateral in T, and the solution to the corresponding problem in QUAD-COVER is "yes".

Suppose now that the answer for instance (P, Q, k) of QUAD-COVER is "yes". Then there exists T, a subset of at most k quadrilaterals, such that every point $(i, j) \in P$ is contained in at least one quadrilateral in T. Now suppose $ij \in E$, where $1 \leq j < i \leq n$. We have $(i, j) \in q_\ell$ for some quadrilateral q_ℓ. By our observation we must have either $\ell = i$ or $\ell = j$. Note that $q_\ell \in T$ if and only if $\ell \in S$, such that at least one of i or j is in S. As edge ij was arbitrary, it holds for every edge. Thus the answer to the original VERTEX-COVER problem must be "yes" as well.

Thus we have a polynomial time reduction from VERTEX-COVER to QUAD-COVER. As VERTEX-COVER is NP-complete, QUAD-COVER is as well.





(c)