Sample Solution for Problem Set 13

Data Structures and Algorithms, Fall 2020

January 4, 2021

Contents

1 Problem 1

(a) Suppose that $\emptyset$ is complete for P. Let $L = \{0,1\}*$. Then L is clearly in P, and there exists a polynomial time reduction function f such that $x \in \emptyset$ if and only if $f(x) \in L$. However, it's never true that $x \in \emptyset$, so this means it's never true that $f(x) \in L$, a contradiction since every input is in L.

Now suppose $\{0,1\}*$ is complete for P, Let $L' = \emptyset$. Then L' is in P and there exists a polynomial time reduction function f'. Then $x \in \{0,1\}*$ if and only if $f'(x) \in L'$. However x is always in $\{0,1\}*$, so this implies $f'(x) \in L'$ is always true, a contradiction because no binary input is in L'.

Finally, let L be some language in P which is not $\emptyset$ or $\{0,1\}*$, and let L' be any other language in P. Let $y_1 \notin L$ and $y_2 \in L$. Since $L' \in P$, there exists a polynomial time algorithm A which returns 0 if $x \notin L'$ and 1 if $x \in L'$. Define $f(x) = y_1$ if $A(x)$ returns 0 and $f(x) = y_2$ if $A(x)$ returns 1. Then f is computable in polynomial time and $x \in L'$ if and only if $f(x) \in L$. Thus, $L' \leq_P L$.

(b) Since L is in NP, we have that $\overline{L} \in coNP$. Since every $coNP$ language has its complement in NP, suppose that we let S be any language in $coNP$ and let $\overline{S}$ be its complement, Suppose that we have some polynomial time reduction f from $\overline{S} \in NP$ to L. Then consider using the same reduction function. We will have that $x \in S \iff x \notin \overline{S} \iff f(x) \notin L \iff f(x) \in \overline{L}$. This shows that this choice of reduction function does work. So we have shown that the complement of any NP complete problem is also $coNP$ complete. To see the other direction, we just negate everything and the proof goes through identically.

2 Problem 2

(a) Let A denote the polynomial time algorithm which returns 1 if input x is a satisfiable formula, and 0 otherwise. We'll define an algorihm A' to give a satisfying assignment. Let $x_1, x_2, \ldots, x_m$ be the input variables. In polynomial time, A' computes the boolean formula x' ehich results from replacing x_1 with true. Then A' runs A on this formula. If it is satisfiable, then we have reduced the problem to finding a satisfying assignment for x' with input variables $x_2, \ldots, x_m$, so A' recursively calls it self. If x' is not satisfiable, then we set x_1 to false, and need to find a satisfying assignment for the formula x' obtained by replacing x_1 in x by false. Again, A' recursively calls it self to do this. If $m = 1$, A' takes a polynomial-time brute force approach by evaluating the formula when x_m is true, and when x_m is false. Let $T(n)$ denote the runtime of A' on a formula whose encoding is of length n. Note that we must have $m \leq n$. Then we have $T(n) = O(n^k) + T(n')$ for some k and $n' = |x'|$ and $T(1) = O(1)$. Since we make a recursive call once for each input variable there are m recursive calls, and the input size strictly decreases each time, so the overall runtime is still polynomial.

(b) Suppose that the original formula was $\wedge_i(x_i \vee y_i)$, and the set of variables were $\{a_i\}$. Then consider the directed graph which has a vertex corresponding both to wach variable, and each negation of a variable. Then for each of the clauses $x \vee y$, we will place an edge going from $\neg x$ to y and an edge from $\neg y$ to x. Then anytime that there an edge in the directed graph, that means if the vertex the edge is coming from is true, the vertex the edge is going to has to be true. Then, what we would need to see in order to say that the formula is satisfiable is a path from a vertex to the negation of that vertex, or vice versa. The naive way of doing this would be to run all pairs shortest path, and see if there is a path from a vertex to its negation. This takes time $O(n^3)$.

3 Problem 3

For any 3-SAT with clauses $C_1, C_2, \ldots, C_m$ where $C_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$, consider the following zero-one integer programming problem

- The i-th constraint is $y_{i_1} + y_{i_2} + y_{i_3} \geq 1$, where $y_{i_j} = z_{i_j}$ if $l_{i_j} = x_{i_j}$, and $y_{i_j} = 1 - z_{i_j}$ otherwise.

It can be seen that given 3-SAT is satisfiable if and only if corresponding zero-one integer programming problem can be satisfied.

4 Problem 4

(a) For each vertex $v \in V$, create two vertices $v_1, v_2 \in V'$, and an edge $(v_1, v_2) \in E'$ such that the capacity of (v_1, v_2) is $l(v)$. For each edge $(u, v) \in E$, create an edge $(u_2, v_1) \in E'$. Then we have $|V'| = 2|V|$ and $|E'| = |E| + |V|$.

(b) According to flow decomposition theorem, each flow can be decomposed to several $s - t$ path and circles. Thus, after decomposition, there must be a circle with flow 1 that contain (v, s). By deleting the circle we can get a flow f' such that $f'(v, s) = 0$. Algorithm: Find a path from s to v in the flow graph to find a circle containning (v, s). Decrease the flow in each edge in the circle by 1 to create f'.

(c) $2\min(|L|, |R|) - 1$. Example: consider $L = \{l_1, l_2, ..., l_n\}$ and $R = \{r_1, r_2, ..., r_n\}$ and edges $(l_1, r_1), (r_1, l_2), (l_2, r_2), (r_2, l_3), ..., (l_n, r_n)$ with capacity 1. An augmenting path $(l_1, r_1, l_2, r_2, ..., l_n, r_n)$ with flow 1.

5 Problem 5

Overview:

Consider the undirected edges to be two directed edges of capacity 1, run the following algorithm.

Algorithm 1: EDGECONNECTIVITY(G)

```
s ← any vertex in G.V;
ans ← +∞;
for t ∈ (G.V − {s}) do
    maxflow = MAXIMUMFLOW(G, s, t);
    ans = min(ans, maxflow) ;
end
return ans;
```

Correctness:

According to the MAXIMUM FLOW MINIMUM CUT THEOREM, MAXIMUMFLOW(G, s, t) is equal to the minimum number of edges that s and t are disconnected.

6 Problem 6

Algorithm 2: BUILDGRAPH(M, D, S, c)

Note: M is the set of doctors.
// vertices
$V \leftarrow \{s, t\}$;
for $M_i \in M$ **do**
  $V \leftarrow V \cup \{v_i\}$;
end
for $D_j \in D$ **do**
  $V \leftarrow V \cup \{v_j\}$;
end
for $day_k \in \cup_j D_j$ **do**
  $V \leftarrow V \cup \{v_k\}$;
end
// Edges
$E \leftarrow \emptyset$;
for $M_i \in M$ **do**
  $E \leftarrow E \cup \{(s, v_i)\}$;
  $capacity((s, v_i)) \leftarrow c$;
end
for $S_i \in S$ **do**
  for $D_j \in S_i$ **do**
    $E \leftarrow E \cup \{(v_i, v_j)\}$;
    $capacity((v_i, v_j)) \leftarrow 1$;
  end
end
for $D_j \in D$ **do**
  for $day_k \in D_j$ **do**
    $E \leftarrow E \cup \{(v_j, v_k)\}$;
    $capacity((v_j, v_k)) \leftarrow 1$;
  end
end
for $day_k \in \cup_j D_j$ **do**
  $E \leftarrow E \cup \{(v_k, t)\}$;
  $capacity((v_k, t)) \leftarrow 1$;
end
return $\{\langle V, E\rangle, s, t\}$;

Algorithm 3: SOLVE(M, D, S, c)

$\{G, s, t\} \leftarrow$ BUILDGRAPH(M, D, S, c);
$maxflow \leftarrow$ MAXFLOW(G, s, t);
if $maxflow = |\cup_j D_j|$ **then**
  return true;
end
else
  return false;
end

7 Problem 7

(a)

run dfs/bfs on modified residual network to determine if there exists a path with positive capacity from s to t. It is worth mentioning that modified residual network can be calculated in $O(V + E)$.

(b)

If $f(u, v) < c(u, v)$, decrease $c(u, v)$ by 1; Otherwise, run bfs/dfs to find a positive flow path P containing (u, v) from s to t, decrease $f(x, y)$ by 1 for all $(x, y) \in P$, decrease $c(u, v)$ by 1, and lastly run bfs/dfs on residual network like (a).