National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2015

Handout #24
January 12, 2016
TAs: Chau-Chin Huang & I-Peng Wu

Sample Solutions to Homework #5

1. (10) Problem 26-3 (pages 761-762)

   a. Prove by contradiction, assume that $E_j \in T$ for a finite-capacity cut (S, T) of G, and $\exists I_k \in S, I_k \in R_j$. However, this is impossible, because the capacity of edge(I_k, E_j) is ∞, $\Rightarrow$ this capacity cut (S, T) is infinite ($\leftrightarrow$ in the assumption, the cut is finite). We can conclude that if $E_j \in T, \forall I_k \in R_j \in T$.

   b. See Figure 1. If S and T are implemented using disjoint sets(with union by rank and path compression), then the running time of Net-Revenue is $O(n + m)$.

```
Net-Revenue(I, E, c, p, cut(S,T))
1   revenue ← 0
2   for i ← 1 to m
3       if E_i ∈ T
4           revenue←revenue+p_i
5   for i ← 1 to n
6       if I_i ∈ T
7           revenue←revenue−c_i
8   return revenue
```

   Figure 1: Determine the maximum net revenue from the capacity of the minimum cut.

   c. See Figure 2. We know that the running time of Edmonds-Karp algorithm is $O(VE^2)$. In this network G, $|V| = n + m + 2, |E| = n + m + r$. If S and T are implemented using disjoint sets (with union by rank and path compression), then the running time = the running time of Edmonds-Karp + Net-Revenue + $O(n+m) = O((n+m+2) \times (n+m+r)^2) + O(n+m) = O((n+m) \times (n+m+r)^2)$.

```
NASA( G , I, E, c, p, s, t)
1   Carry-E ← ∅
2   Carry-I ← ∅
3   apply Ford-Fulkerson(G, s, t) to get the minimum-cut C(S, T)
4   call Net-Revenue(I, E, c, p, cut(S,T) to get the net revenue
5   for i ← 1 to m
6       if E_i ∈ T
7           Carry-E ← Carry-E ∪ {E_i}
8   for i ← 1 to n
9       if I_i ∈ T
10          Carry-I ← Carry-I ∪ {I_i}
11  return Carry-E, Carry-I, revenue
```

Figure 2: Determine which experiments to perform and which instruments to carry, where Carry-E represents which experiments to perform, Carry-I represents which instruments to carry, and revenue represents the net revenue.

2. (10)

(a) The construction of this switch matrix (which has no separating switch) is shown in Figure 3(a). For this graph, its corresponding bipartite matching problem is as follows: Given a graph, is there a bipartite matching of size $n_t + n_b$? (Assume $n_t + n_l + n_b \leq 3$.)

(b) The construction of this switch matrix (which contains separating switches) is shown in Figure 3(b). For this network, its corresponding network-flow problem is as follows: Given a network, is there a feasible flow where s_t supplies a flow of n_t, s_l supplies a flow of n_l, s_b supplies a flow of n_b, and t receives a flow of $n_t + n_l + n_b$.

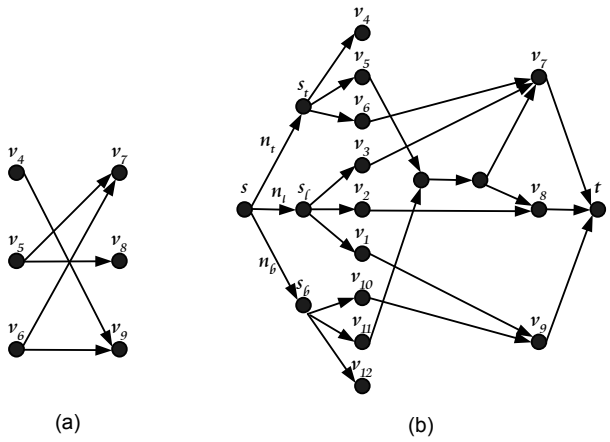

Figure 3: (a) The bipartite graph. (b) The flow network. (All edges are unit-capacity edges expect for (s, s_t), (s, s_l), and (s, s_b).

3. (10)

   (a) Exercise 34.1-4 (page 1060)

   The time complexity of the method for Exercise 16.2-2 is $O(nW)$, where n is the number of items and W is the maximum weight. Whether it is a polynomial-time algorithm depends on the input size (how the items and the weight are encoded). With only $l = \lg W$ bits, we can encode the weight value; as a result, the complexity becomes $O(n2^l)$, which is not a polynomial function to the input size l. So it is not a polynomial-time algorithm.

   (b) Since C is a value and C is stored with $k = \lg C$ bits, the actual input size is k. Thus, the complexity is $O(VE\lg C) = O(VEk)$, which is polynomial.

4. (20) Problem 34-1 (pages 1101-1102)

   (a) The decision problem can be formulated as follows: Given a graph $G(V, E)$, determine if G has an independent set of size b.

   To show that the problem is in NP, we can use $V' \subseteq V$ as a certificate. Whether V' is a independent set of G can be checked in polynomial time by checking whether each pair is not connected by an edge in E.

   To show that the problem is NP-hard, we can reduce the clique problem, which is already known to be NP-hard, into that problem. For any problem instance of the clique problem with $G(V, E)$ and b, we can construct a problem instance of the independent set problem with $G'(V, E')$ and b where E' is the complement of E. Since V' is a clique of size b in G iff V' is an independent set of G' of size b and the construction of G' and b from G and b can be done in polynomial time, we can conclude that the independent set problem is NP-hard, too.

   To sum up, the independent set problem is NP-complete.

   (b) Given a graph $G(V, E)$ and the described black-box $B(G, b)$, we can find the dependent set as follows:

   (1) Find the maximum possible size b^* of an independent set by binary search with B and this takes $O(\log V)$. (2) Initialize set I as an empty set. $O(1)$ (3) For each $v \in V$, construct $G'(V', E')$ by

removing v and the associated edges from $G(V, E)$. Query if $B(G', b^*)$ is true. If so, $G \leftarrow G'$ Else, $G \leftarrow G''$,where G'' is derived from G by removing all vertices connected to v and the associated edges. ($O(E) \times O(V)$ iterations.)

The vertices in the final G must be a independent set of size b^* by construction and the time complexity is $O(VE)$.

(c) Such a graph must be formed by independent circles. For each circle, we can randomly start from a vertex on the circle and sequentially assign $1, 0, 1, 0, \ldots$ to the vertices along the circle. Finally we choose the vertices assigned 0 and these vertices is the maximum independent set.

The process takes $O(V)$ and the correctness can be proved by arguing that we cannot find better result for all the independent circles.

(d) To find an independent set of a bipartite graph, we can first find a maximum matching for the graph and then choose the vertices not in the maximum matching and one vertex from each edge in the maximum matching that is not connected to the vertices not in the maximum matching. For each edge (u, v) in the maximum matching, we can find at least one such vertex or else the matching is not maximum. Such a set is independent by construction and must be maximum because the sum of the size of any independent set and the size of the maximum matching cannot exceed $|V|$.

The time complexity depends on the method used to solve the maximum matching.

5. (30) Problem 34-3 (pages 1103-1104)

(a) First, set every vertex in G as *unlabeled*. Randomly pick a vertex v_s in G as the starting vertex and label v_s with either 1 or 2. Then, apply the DFS algorithm to visit each vertex in G from v_s. Each time a new vertex v_i is visited, check the labels of all the neighboring vertices. If v_i is neighboring to both vertices with label type 1 and 2, that means there is no feasible solution for 2-coloring of G. Otherwise, if v_i is neighboring to vertices with the same label type, v_i is assigned to the other label type. If all the vertices are visited without violation, a feasible solution for 2-coloring of G is determined. The time complexity is $O(V + E)$, which is the same with the DFS algorithm.

(b) The decision problem L of the graph-coloring problem is: Given an undirected graph G and a positive integer k, does a k-coloring for G exist?

If we can determine if a k-coloring for a given graph G exists or not in polynomial time, the graph-coloring problem can be solved by assigning k from 1 to $|V|$ and applying at most $|V|$ times of L. That means the graph-coloring problem can be also solved in polynomial time. In contrast, if the graph-coloring problem can be solved and the minimum number n_c of colors needed can be found in polynomial time, L can be also solved by checking if $k \geq n_c$ in constant time. Now, we have proved that L is solvable in polynomial time if and only if the graph-coloring problem is solvable in polynomial time.

(c) Obviously, L can be verified in polynomial time by visiting each vertex in the graph G, by counting the number of colors used, and checking if there are two neighboring vertices with the same color. Thus, we have proved that $L \in$ NP. Also, 3-COLOR is a subset of L and naturally 3-COLOR $\leq_p L$ and thus $L \in$ NP-hard. Therefore, if 3-COLOR is NP-complete, L is also NP-complete.

(d) Since a variable and its negation are both connected to the vertex RED, they cannot be colored c(RED) and hence have to be colored either c(TRUE) or c(FALSE). Moreover, since a variable is connected to its negation, exactly one of them is colored c(TRUE) and the other is colored c(FALSE).

If a variable x_i is set to be TRUE in a truth assignment then we color x_i c(TRUE) and color its negation $\overline{x_i}$ c(FALSE). The triangle $x_i\overline{x_i}$(RED) are properly 3-colored and the triangle (TRUE)(FALSE)(RED) are properly 3-colored by definition. The graph consisting of the literal edges consists of exactly these triangles and thus, can be 3-colored.

(e) For ease of explaining, let us label the vertices of Figure 34.20 in the textbook as shown in Figure 4.

By part (d), none of the vertices x, y and z can be colored c(RED). Thus, it suffices to show that there is no 3-coloring of the widget with all x, y and z receiving the color c(FALSE).

Suppose x, y and z are colored c(FALSE). Then vertices 3 and 4 have to be colored either c(TRUE) or c(RED) but not c(FALSE). Now, if vertex 2 is colored c(RED) then both vertices 3 and 4 have

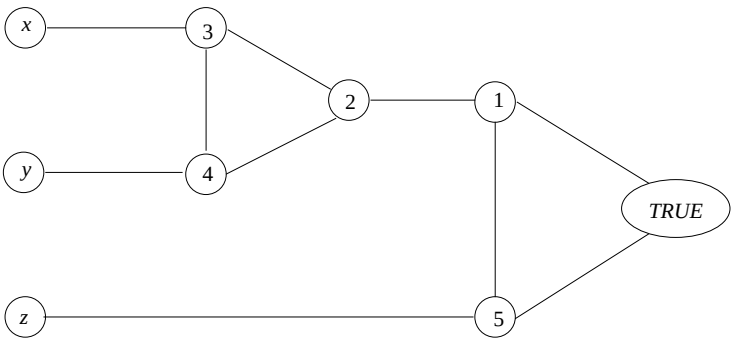

Figure 4: Label each vertex in Figure 34.20 in the textbook for ease of explaining.

to be colored c(TRUE) which is illegal since they are connected. Hence, vertex 2 has to be colored either c(TRUE) or c(FALSE). By the same argument, vertex 2 cannot be colored c(TRUE).

Thus, we suppose vertex 2 is colored c(FALSE). Then vertex 1 being adjacent to both the vertices 2 and TRUE, has to be colored c(RED). This forces the vertex 5 to receive the color c(FALSE) since it is adjacent to both vertices 1 and TRUE. However, we have an illegal coloring here since both vertex 5 and z are adjacent ana are colored c(FALSE).

(f) As in part (c), the 3-COLOR problem is clearly in NP. To prove that 3-COLOR is NP-hard, we reduce 3-CNF-SAT to 3-COLOR. The reduction is as described in the problem (the paragraph before part (d)).

Suppose ϕ is satisfiable by the truth assignment θ. We color the vertices x c(TRUE) and $\overline{x}$ c(FALSE) if $\theta(x) = true$. Otherwise, we color x c(FALSE) and $\overline{x}$ c(TRUE). By part (d), this partial coloring forms a 3-coloring of the graph containing just the literal edges. Since ϕ is satisfied, each clause has a literal that is set to true and hence the corresponding vertex receives the color c(TRUE). By part (e), the widget corresponding to this clause can be 3-colored.

Conversely, suppose the graph G is 3-colorable. We can assign the value $true$ to a variable if its corresponding vertex in G gets color c(TRUE), otherwise we assign $false$ to it. We assign value to its negation in the same manner and by part (d), such assignment is a valid one (*i.e.*, we do not have a situation where we assign where we assign both x and $\overline{x}$ the same value). Moreover, since the widget corresponding to each clause is 3-colorable, part (e) implies that each clause has a literal that is assigned $true$. Thus, ϕ is satisfiable.

6. (12) Independent-Set Problem

(a) ISP$_D$: Given a graph G and a positive integer k, determine whether there is an independent set with size k in G.

(b) ISP$_D$ is NP because given any vertex subset with k vertices, we can check that no edge exists among them in polynomial time.

We prove ISP$_D$ is NP-hard as follows. Given an instance ϕ of 3SAT with m clauses $C_1, \ldots, C_m$, with each clause being $C_i = (\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$, with the α_{ij}'s being either Boolean variables or negations thereof. The reduction constructs a graph $G = (V, E)$, where $V = \{v_{ij} : i = 1, \ldots, m, j = 1, 2, 3\}$ and $E = \{(v_{ij}, v_{ik}) : i = 1, \ldots, m, j \neq k\} \cup \{(v_{ij}, v_{lk}) : i \neq l, \alpha_{ij} = \neg\alpha_{lk}\}$. In other words, there is a node for every appearance of a literal in a clause; the first set of edges defines the m *triangles*, and the second group joins opposing literals. See an example construction in the given figure.

We claim that there is an independent set of m nodes in G if and only if ϕ is satisfiable. Suppose that such an independent set I exits. Since I contains m nodes, it must contain a node from each triangle. Since the nodes are labeled with literals, and I contains no two nodes corresponding to opposite literals, I is a truth assignment that satisfies ϕ: The true literals are just those which are labels of nodes of I (variables left unassigned by this rule can take any value). We know that this gives a truth assignment because any two contradictory literals are connected by an edge in G, and

so they cannot both be in I. And since I has a node from every triangle, the truth assignment satisfies all clauses.

Conversely, if a satisfying assignment exists, then we identify a true literal in each clause, and pick the node in the triangle of this clause labeled by this literal: This way we collect m independent nodes.

Therefore, we conclude that ISP_D is NP-complete.

7. (10) Exercise 35.2-3 (page 1117)

   Obviously, the closest-point heuristic follows the concept of the Prim's algorithm and the cost of resulting tour is smaller than a full walk of the corresponding MST generated by Prim's algorithm. According to the analysis of Theorem 35.2, the closest-point heuristic also has a ratio bound of 2.

8. (25) Problem 35-1 (page 1134)

   (a) Given a positive integer k, the decision problem (BP_D) of the bin packing problem is to determine whether the set of n objects can be packed into k bins. In the following, we prove that BP_D is NP-hard by reducing a known NP-complete problem (i.e. the number-partitioning problem) to the problem. The number-partitioning problem: given a set of n objects, where the ith object has size p_i, and ask whether they can be partitioned into two subsets of equal total size. Given an arbitrary instance of the number-partitioning problem, it can be transformed to an instance of BP_D as follows: Let $s_i = (2 \times p_i)/\sum_{j=1}^{n} p_j$ for $i = 1, ..., n$ and $k = 2$. Therefore, the packing is possible if and only if the partition is possible. Further, the transformation is linear to the object number and thus is polynomial.

   (b) Since the total size packed in x bins is at most x, the number of bins required for total size S is at least $\lceil S \rceil$.

   (c) The first-fit heuristic stops packing objects into some bin only when there is no more objects can be packed into it. Therefore, if there are more than one bin is less than half full, every two bins, somehow, must be combine into one bin to follow the idea of first-fit heuristic. Finally, only at most one bin less than half full is possible.

   (d) According to the argument in (c), the total packed object size is strictly larger than $(k-1)/2$ when using k bins. If the usage of bin number is at least $2\lceil S \rceil + 1$, the total packed size would be strictly larger than $\lceil S \rceil$. The contradiction shows that the number of bins used by the first-fit heuristic is never more than $\lceil 2S \rceil$.

   (e) Based on the argument in (b) and (d), we know that any solution including the optimal case has at least $\lceil S \rceil$ bins and the first-fit heuristic has the bin number never more than $\lceil 2S \rceil$. The approximation ratio of $\lceil 2S \rceil / \lceil S \rceil = 2$ is proven.

   (f) First, the objects are sorted in non-increasing order by size. Along this order, objects are packed into bins one by one. Once a object cannot be packed due to the insufficient remaining space of the last bin used, a new bin is added for remaining objects. Obviously, through this manner, only at most one bin less than half full is possible, and thus the bin number never more than $\lceil 2S \rceil$ is fulfilled.

   For the running time, the sorting is $O(n \lg n)$, and the packing is $O(n)$. In sum, the running time of the first-fit heuristic is $O(n \lg n)$.