Q1
a) (0, 0), $(\frac{9}{11}, \frac{23}{11})$, (0, $\frac{8}{5}$), $(\frac{12}{7}, 0)$

b)
LP: optimal solution is $x_1 = \frac{9}{11}, x_2 = \frac{23}{11}$. Optimal objective value is $\frac{23}{11}$.
IP: optimal solution is $x_1 = 0, x_2 = 1$ or $x_1 = 1, x_2 = 1$. Optimal objective value is 1.

c)
min $8y_1 + 12y_2$
$7y_2 - 3y_1 \geq 0$
$5y_1 + 3y_2 \geq 1$
$y_1 \geq 0$ $y_2 \geq 0$

y_1 corresponds to $-3x_1 + 5x_2 \leq 8$
y_2 corresponds to $7x_1 + 3x_2 \leq 12$

d)
LP: optimal solution is $y_1 = \frac{7}{44}, y_2 = \frac{3}{44}$. Optimal objective value is $\frac{23}{11}$.
IP: optimal solution is $y_1 = 0, y_2 = 1$. Optimal objective value is 12.

Since $12 > 1$, strong duality does not hold for this particular pair of primal and dual IP.

Q2
1. max $\Sigma_{i=1}^{n}\Sigma_{j=1}^{m} c_i x_{(i,j)}$
2. for j = 1, 2, ..., m have:
   $\Sigma_{i=1}^{n} x_{(i,j)} = 1$
3. for i = 1, 2, ..., n; j = 1, 2, ..., m have:
   $x_{(i,j)} \geq 0$
4. for i = 1, 2, ..., n; j = 1, 2, ..., m have:
   $x_{(i,j)}$ is integer
5. $\Sigma_{i=1}^{n}\Sigma_{j=1}^{m} s_{ij} x_{(i,j)} \geq 1$
6. for i = 1, 2, ..., n have:
   $\Sigma_{j=1}^{m} x_{(i,j)} \leq 1$
7. for each pair of $(i, k) \in I$ have:
   $\Sigma_{j=1}^{m} x_{(i,j)} + \Sigma_{j=1}^{m} x_{(k,j)} \leq 1$

Each $x_{i,j}$ (i = 1, 2, ..., n, j = 1, 2, ..., m) represents player i being assigned on position j.
1. Since the goal is to pick a team that maximizes the total celebrity rating of all selected players, we need to maximize $\Sigma_{i=1}^{n}\Sigma_{j=1}^{m}c_i x_{(i,j)}$.
2 & 3: We cannot have no one or have more than one person being assigned on position j, therefore we have constraints $x_{(i,j)} \geq 0$ and $\Sigma_{i=1}^{n} x_{(i,j)} = 1$
4. Since we either assign person i to position j or we do not assign person i to position j, $x_{(i,j)}$ need to be an integer.
5. Since we need to make sure that the total suitability of players for their assigned positions is at least 1, we have constraint $\Sigma_{i=1}^{n}\Sigma_{j=1}^{m} s_{ij} x_{(i,j)} \geq 1$.
6. Since we cannot assign one person to the team twice, we have constraint $\Sigma_{j=1}^{m} x_{(i,j)} \leq 1$.
7. Since we should not put incompatible people on the team, then for each pair (i, k) $\in$ I, $\Sigma_{j=1}^{m} x_{(i,j)} + \Sigma_{j=1}^{m} x_{(k,j)} \leq 1$.

Q3
a)
AllSmallCycles is in *co*NP.

a[0, ..., k]: a polynomial size advice, which is a list of nodes claimed to be a cycle in G including vertex s and having total weight larger than B
checkLargeCycle(G = (V, E), a[0, ..., k], B):

```
includeS = false
if edge(a[0], a[k]) ∉ E: // not even a cycle
    return false
totalWeight = w(a[0], a[k])
if a[0] = s:
    includeS = true
iterate through a[1, ..., k]:
    if a[i] = s:
        includeS = true
    if edge(a[i - 1], a[i]) ∉ E:
        return false
    if a[i] ∈ a[0, ..., i - 1]:
        return false // not a simple cycle
    totalWeight += w(a[i - 1], a[i])
if includeS is false:
    return false
```

```
if totalWeight ≤ B
    return false
return true
```

checkLargeCycle checks if polynomial size advice a is a simple cycle in G and if the total weight is larger than B. And if checkLargeCycle returns true, AllSmallCycles should return false. checkLargeCycle has time complexity $O(|V^2|)$. Therefore AllSmallCycles is in *co*NP.

b)
AllLargeCycles is in P. Algorithm:

dijkstra(G, l, s):
Input: Graph G = (V, E), positive edge length $l : E \to \mathbb{Z}^+$, vertex s $\in$ V
Output: For all vertices u reachable from s, dist(u) is set to the distance from s to u.

```
AllLargeCycles(G = (V, E), w, s):
1   for each v in V:
2       if edge(s, v) in E:
3           let E' = E - edge(s, v)
4           let l = w
5           run dijkstra(G' = (V, E'), l, s)
6           if dist(v) + w(s, v) < B:
7               return false
8    return true
```

Correctness:
A simple cycle that includes edge (u, v) is a simple path between vertex u and vertex v, which does not include edge (u, v), plus edge (u, v). For each vertex v in V such that (s, v) is an edge in E, we first remove edge (s, v) from the graph, and then run dijkstra on the remaining graph. dijkstra finds the distance D, which is the same as the total weight, of the shortest simple path (since all edge has positive weights, the shortest path is the same as the shortest simple path) between s and v which does not include edge (s, v). We then add the edge (s, v) back so that it forms a shortest simple cycle that includes edge (s, v), and the sum of the weight D and w(s, v) is the weight of the shortest simple cycle that includes edge (s, v). If the weight of the shortest simple cycle that includes edge (s, v) for each v is no smaller than B, then every cycle in G that includes vertex s have total weight at least B.

Time complexity:
The for loop on line 1 iterates at most V times. Inside each for loop: checking if edge(x,

v) in E and remove an edge from E takes O(1) if we use adjacency matrix representation of G, and dijkstra takes $O((|V| + |E|) \log |V|)$. Therefore time complexity is $O(|V|(|V| + |E|) \log |V|)$

c)
SomeLargeCycles is in np.

a[0, ..., k]: a polynomial size advice, which is a list of nodes claimed to be a cycle in G including vertex s and having total weight at least B
checkLargeCycle(G = (V, E), a[0, ..., k], B):

```
includeS = false
if edge(a[0], a[k]) ∉ E: // not even a cycle
    return false
totalWeight = w(a[0], a[k])
if a[0] = s:
    includeS = true
iterate through a[1, ..., k]:
    if a[i] = s:
        includeS = true
    if edge(a[i - 1], a[i]) ∉ E:
        return false
    if a[i] ∈ a[0, ..., i - 1]:
        return false // not a simple cycle
    totalWeight += w(a[i - 1], a[i])
if includeS is false:
    return false
if totalWeight < B
    return false
return true
```

checkLargeCycle checks if polynomial size advice a is a simple cycle in G and if the total weight is no smaller than B. And if checkLargeCycle returns true, SomeLargeCycles should return true. checkLargeCycle has time complexity $O(|V^2|)$. Therefore SomeLargeCycles is in np.

d)
SomeSmallCycles is in P. Algorithm:

dijkstra(G, l, s):
Input: Graph G = (V, E), positive edge length $l : E \to \mathbb{Z}^+$, vertex s $\in$ V
Output: For all vertices u reachable from s, dist(u) is set to the distance from s to u.

SomeSmallCycles(G = (V, E), w, s):
```
1    for each v in V:
2        if edge(s, v) in E:
3            let E' = E - edge(s, v)
4            let l = w
5            run dijkstra(G' = (V, E'), l, s)
6            if dist(v) + w(s, v) ≤ B:
7                return true
8     return false
```

Correctness:
A simple cycle that includes edge (u, v) is a simple path between vertex u and vertex v, which does not include edge (u, v), plus edge (u, v). For each vertex v in V such that (s, v) is an edge in E, we first remove edge (s, v) from the graph, and then run dijkstra on the remaining graph. dijkstra finds the distance D, which is the same as the total weight, of the shortest simple path (since all edge has positive weights, the shortest path is the same as the shortest simple path) between s and v which does not include edge (s, v). We then add the edge (s, v) back so that it forms a shortest simple cycle that includes edge (s, v), and the sum of the weight D and w(s, v) is the weight of the shortest simple cycle that includes edge (s, v). Compare the weight of the shortest simple cycle that includes edge (s, v) for each v to B, and determine a cycle in G include vertex s and have total weight at most B exists.

Time complexity:
The for loop on line 1 iterates at most V times. Inside each for loop: checking if edge(x, v) in E and remove an edge from E takes O(1) if we use adjacency matrix representation of G, and dijkstra takes $O((|V| + |E|) \log |V|)$. Therefore time complexity is $O(|V|(|V| + |E|) \log |V|)$

Q4
a)
FriendlyRepresentatives is in np:

S[1, ..., m]: a polynomial size advice, which is a list of people claimed such that every person who is not in S is friends with someone who is in S

```
checkFriendlyRepresentatives(S[1, ..., m], N[1, ..., n], F):
    for i = 1, ..., n:
        if N[i] ∉ S: // N[i] should know someone in S
            knowSomeoneInS = false
            for j = 1, ..., n:
                if (N[i], N[j]) ∈ F:
                    if N[j] ∈ S:
                        knowSomeoneInS = true
            if knowSomeoneInS is false:
                return false
    return true
```

checkFriendlyRepresentatives checks if polynomial size advice S is a list of people such that every person who is not in S is friends with someone who is in S. checkFriendlyRepresentatives has time complexity $O(n^5)$. Therefore FriendlyRepresentatives is in np.

b)

Claim: Connected Vertex Cover $\leq_p$ FriendlyRepresentatives.

Given any ConnectedVertexCover problem, which takes a connected graph G = (V, E) as input and decides whether it admits a connected vertex cover of size exactly k, we construct a FriendlyRepresentatives problem as follows:

Let N = V such that each person n_i in N represents a vertex v_i in V. Also let N' = N. For each edge e(v_i, v_j) in E, add fake person n_{ij} to N', add (n_i, n_j), (n_i, n_{ij}), (n_j, n_{ij}) to F. Run FriendlyRepresentitives(N', F, k). If FriendlyRepresentitives(N', F, k) returns true, then there exists S $\subseteq$ N' with $|S|$ = k such that every person who is not in S is friends with someone who is in S.

Construct S' as follows:

```
1   S' = { }
2   for h = 1, ..., m
3       if S[h] in N:
4           add S[h] to S'
5       else:
6           let n_ij represent S[h]
7           if n_i not in S:
```

8 add n_i to S'
9 else if n_j in S:
10 add n_j to S'
11 else
12 for g = 1, ..., n
13 if n_g not in S
14 add n_g to S'

Claim: S' $\subseteq$ N with $|S'|$ = k such that every person who is not in S' is friends with someone who is in S'.
Proof:
Claim: Every person who is not in S' is friends with someone who is in S'.
S and S' only differ in three ways.
1: $n_{ij} \in S$ for some i, j, and $n_i \notin S$, but $n_{ij} \notin S'$ and $n_i \in S'$. This swap is valid because n_{ij} and n_j still knows someone in S', which is n_i. The argument for $n_{ij} \in S$ for some i, j, and $n_j \notin S$ is similiar.
2: $n_{ij} \in S$ for some i, j, and $n_i \in S$, $n_j \in S$. We simply do not add n_{ij} to S'. This is valid because v_i and v_j are both in S'. They do not need n_{ij} to also be in S'.
3: n_m not in S but is added to S' through line 14. This is valid because adding one more person to S does not hurt.

S' $\subseteq$ N because we did not add any fake person n_{ij} for some i, j to S'.

$|S'|$ = k. This is because when we decide not adding $n_{ij} \in S$ to S', we always add another person n_m not in S to S'.

Claim: S' is a connected vertex cover of size exactly k for G = (V, E)
Proof:
For each edge e(v_i, v_j), since S' $\subseteq$ N with $|S'|$ = k such that every person who is not in S' is friends with someone who is in S', then at least one of v_i, v_j is in S' to make sure n_{ij} knows someone in S'. Since N = V, therefore S' $\subseteq$ V and $|S'|$ = k such that each edge e(v_i, v_j) is incident to at least one vertex in S'.

Claim: For any graph G = (E, V), if G has connected vertex cover of size k, then the FriendlyRepresentatives problem, with constructed N' and F as input, of size k would return true.
Proof:
If S is a connected vertex cover such that S $\subseteq$ V and $|S|$ = k such that every edge is incident to at least one vertex in S, then it satisfies FriendlyRepresentatives(N', F, k).

Because in N', if a person is not in S there are two cases. Case 1: this person is a fake person n_{ij}. Since S is a connected vertex cover, for edge(v_i, v_j), at least one of v_i, v_j is in S. Suppose, without loss of generality, v_j is in S, then n_{ij} knows someone in S, which is n_j. Case 2: this person is a real person n_j. Since S is a connected vertex cover, there is some edge e(v_i, v_j) such that v_i is in S. Then n_j also knows someone in S, which is n_i.

Since any valid connected vertex cover of size k satisfies the FriendlyRepresentatives problem of size k, then if FriendlyRepresentatives problem cannot find an answer for size k, then there is not a connected vertex cover of size k. Since we can get a connected vertex cover by modifying the solution S found by FriendlyRepresentatives problem in polynomial time, and the size of the FriendlyRepresentatives problem constructed is polynomial in size of the original connected vertex cover problem, then if FriendlyRepresentatives problem can be solved in polynomial time, the connected vertex cover problem can also be solved in polynomial time.