CSCI 3104 Spring 2014 Problem Set 1
Firstname Lastname (MM/DD)

---

1. (a) We add a source s and sink t into $P \cup I$. Then we add edges $\{(s,p)|p \in P\}$ and $\{(i,t)|i \in I\}$ into E. So $V' = \{s,t\} \cup P \cup I$, and $E' = \{(s,p)|\ p \in P\} \cup \{(i,t)|\ i \in I\} \cup E$.
For $e \in E'$, the lower bound $l(e)$, capacity $c(e)$, demand $r(v)$ are set by follows:

$$(l(e), c(e)) = \begin{cases} (0,1), & e \in E \\ (b_p, t_p), & e = (s,p) \in \{(s,p)|\ p \in P\} \\ (l_i, u_i), & e = (i,t) \in \{(i,t)|\ i \in I\} \end{cases}$$
$$r(v) = \quad 0, \quad v \in P \cup I$$

So we got a graph $G' = \{V',\ E'\}$. Obviously if we find a routing with lower bounds, a issue i will be answered by at least l_i persons and at most u_i persons because of the limitation by lower bound $l(e) = l_i$ and capacity $c(e) = u_i$ of the edge (i,t). A person p will be question by at least b_p issues and at most t_p issues because of the limitaion by lower bound $l(e) = b_p$ and capacity $c(e) = t_p$ of the edge (s,p).
Now the parameter feasibility problem have been formulated as the problem of finding a routing with lower bounds.

(b)
- We add an edge from the target t to source s. The new edges limitation is [0 , INF]. We should make the conservation of the flow in the network. So the problem is trying to convert into the max-flow problem where there is no lower bound.
- Remove the lower bound of each edge. Introduce new source s_{new} and target t_{new} into the network to make the conservation of the flow in the network. For each edge $e = (u,v) : l(e) > 0$, we add a edge (u, t_{new}) with capacity $l(e)$ and a edge (s_{new}, v) with capacity $l(e)$. Then we update the capacity of $e = (u,v)$ with $c(e) - l(e)$. Now we remove the lower bound from the origin network.
- Then, we calculate the max-flow from the new source to new target. If and only if all of the flow from new source s_{new} and flow into new sink t_{new} is full, it means that there is a feasible routing from the origin source to origin target which satisfy the lower bound the network.

So we get the $\hat{G} = \hat{V}, \hat{E}$, in which $\hat{V} = \{s_{new}, t_{new}\} \cup V'$ and $\hat{E} = E' \cup \{(s_{new}, v), (u, t_{new}) | e(u, v) \in E' \quad \& \quad l(e) > 0\}$. The capacity $c(e) = c(e) - l(e)$ for each $e \in E'$. And the capacity of $(s_{new}, v), (u, t_{new})$ both are $l(e)$, in which $e = (u, v)$.

(c) We implement the convertion in the fucntion solution.problem2_c() in solytion.py.

(d) We implement the FordFulkerson algorithm and use it to solve the problem in the function solution.problem2_d() in solution.py.

(e) We implement the test case generator in solution.problem2_e() in solution.py

2. (a) We can solve this problem with dynamic programming.
Take the example of the network graph provided in the problem.
We define $f(A)$ as the number of different ways that the spider can reach the node A. Obviously $f(A) = 1, f(J) = 1, f(H) = 1$. And $f(B), f(S), f(K), f(G)$ can be caculated by $f(A), f(J), f(H)$. We iterative caculate the other node. At last we got $f(fly)$ by $f(fly) = f(D) + f(O) + f(E)$.

We use a bipartite graph $G = \{V, E\}$ to represent whether there is a way from node u to node v:$(u, v) \in E$.
Step 1. Init a vertex set V_{reach} to store the nodes which have reached, and put the start node **Spider** into V_{reach}. And set $f(Spider) = 1$.
Step 2. For every $v \in V, \notin V_{reach}$, caculate $U = \{u | (u, v) \in E\}$. If $U \subseteq V_{reach}$, then put v into V_{reach}. And caculate $f(v) = \sum_{(u,v) \in E} f(u)$.
Step 3. Repeat Step 2 until the node fly put into the set V_{reach}.

(b) We implement the algorithm as the function problem4() in solution.py
We use problem4_test() to generater a Graph as input for probelm4().
The graph provided in this problem has **141** ways from 'Spider' to 'Fly'. We use networkx.all_simple_paths() to check our result and it is correct.

3. (a) If n is odd , it is true that at least one person not hit by a bollon.

We defined a distance tuple $(personA, personB, dis)$ to represent the distance between $personA$ and $personB$.

The algorithm takes input as a distance tuple list and the number of the persons and it outputs a list of person who was hit by bollon.

At first, the algorithm sort the distance tuple list in ascending order. Then it traversal the distance tuple list. For every tuple $(personA, personB, dis)$, if $personA$ is not in $person\_hurl$ set, it puts $personA$ into the $person\_hurl$ set and $personB$ into the $neighbor\_hit$ set. Do the same if $personB$ is not in $person\_hurl$ set.

If the $len(neighbor\_hit) < n$, it is true that at least one person not hit by a bollon, otherwise it is flase.

```
problem5(dis_list, num_person){
    sort(dis_list)
    for dis in dis_list
        if dis.personA not in person_hurl:
           person_hurl.add(personA)
           neighbor_hit.add(personB)

        if dis.personB not in person_hrul:
           person_hurl.add(personB)
           neighbor_hit.add(personA)

    if len(neighbor_hit) < n:
       return True
    else:
       return false
}
```

(b) We implement the algorithm in solution.py.
We also inplement a algorithm to generate the data of person and distance which satisfies everyone has a unique nearlest neighbor. Obviously if we put the n persons in random n position, some of the people may have two or more nearlest neighbors. For example, everyone has two nearlest neighbor if we put three people in the vertexs of equilateral triangle,

(c) It can be proved that when n is odd, the answer is always true.

Lemma 1 *(Pigeonhole Principle) At least one person not hit by a balloon if and only if at least one person hit by two or more balloons.*

- Step 1. we can pop the minimum distance $(personA, personB, dis)$ from the distance tuple list. Obviously the $personA$ and $personB$ are the nearlest neighbor of each other. Both of PersonA hit by the other one. We put $personA$ and $personB$ in the set $neibor\_hit$. There remain n-2 people not hit by balloon.
- Step 2. Then we pop the minimum distance $(personA, personB, dis)$from the distance tuple list again.
  - $personA(or\, personB) \in neighbor\_set$, $personA(orpersonB)$ would hurl the balloon at one of person in $neighbor\_set$, which means one person hit by two balloon. We know that at least one person not hit by balloon by the Lemma. It proved.
  - else if $personA \notin neighbor\_hit$ and $personB \notin neighbor\_hit$, we put $personA, personB$ into $neighbor\_set$. So there remains n-4 perple not hit by ballon.
- Step 3. Repeat Step 2 until there is no person put into the $neighbor\_set$. Because n is odd and the length k of $neigibor\_set$ is even, there always remain $n - k$ people not hit by ballon.

Proved.

(d) The length of distance tuple list is $(n * (n + 1))/2$. The time complexity of part of sort function is $O(n^2 log(n))$. The time complexity of finding the people hit set is $O(n^2)$. So the time complexity is $O(n^2 log(n))$.

Also we can solve the problem 5(a) in time complexity $O(n^2)$. We just need find the nearest neighbor of everyone $(O(n^2))$, caculate the number of people hitted.

We use our algorithm for clearly proving that it always is true when n is odd.