CSC373 – Problem Set 8

Remember to write your **full name(s)** and **student number(s)** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due WEDNESDAY, Dec 2, 2020, 22:00; required files: ps8.pdf, ps8.tex

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

[15 points]

A wireless network company has a number of sensors $s_1, \ldots, s_n$, each located at coordinates $(x_i, y_i) \in \mathbb{R}^2$ on some map. The company would like to make the sensors more reliable by assigning each sensor s_i a backup set $B_i \subseteq \{s_1, \ldots, s_n\} - \{s_i\}$ so that if s_i fails, a sensor in B_i can be used instead. The company decides that:

- Each backup set B_i must contain sensors within distance at most d from s_i.
- Each backup set B_i must contain at least r backup sensors.
- Every sensor s_i must belong at most b backup sets.

1. **(7 points)** Given the sensors, their coordinates, and the parameters d, r, b described, design an algorithm to output the backup sets for each sensor, or reports that choosing backup sets is not possible with the given parameters.

   Our algorithm consists of the following steps:

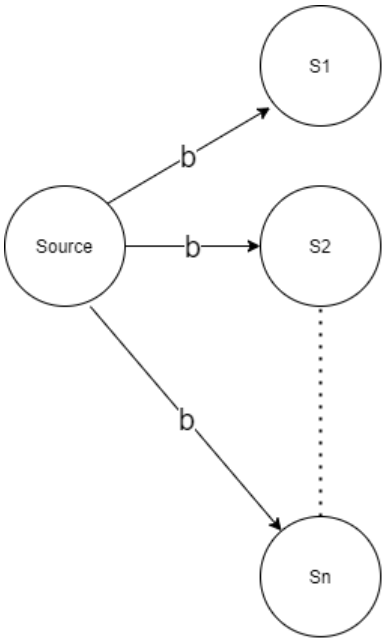
   Step 1:

   First we calculate the distance between each sensor. If the distance between two sensors is less than or equal to d, then one of the sensors can be the other sensor's backup sensor, and vice versa. So we create a dictionary to store these sensors, where the key are the name of the sensors, and the value would be a list containing the candidates for the backup sensors for that sensor.

Step 2:

We create a directed graph G with a single source point s, connect s to each sensors with a forward edge with capacity b.

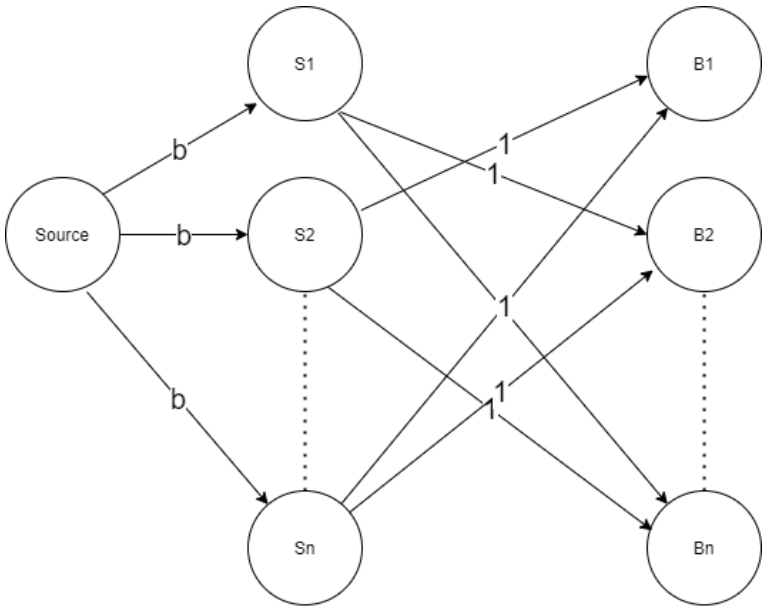
Figure 1:



Step 3:

We create a list of backup sets. B_1 is the backup set of S_1, B_2 is the backup set of S_2.... Then build forward edges between each sensor and the corresponding backup sets (based on whether it can become the backup sensor of that backup set based on information from our dictionary) with capacity 1.

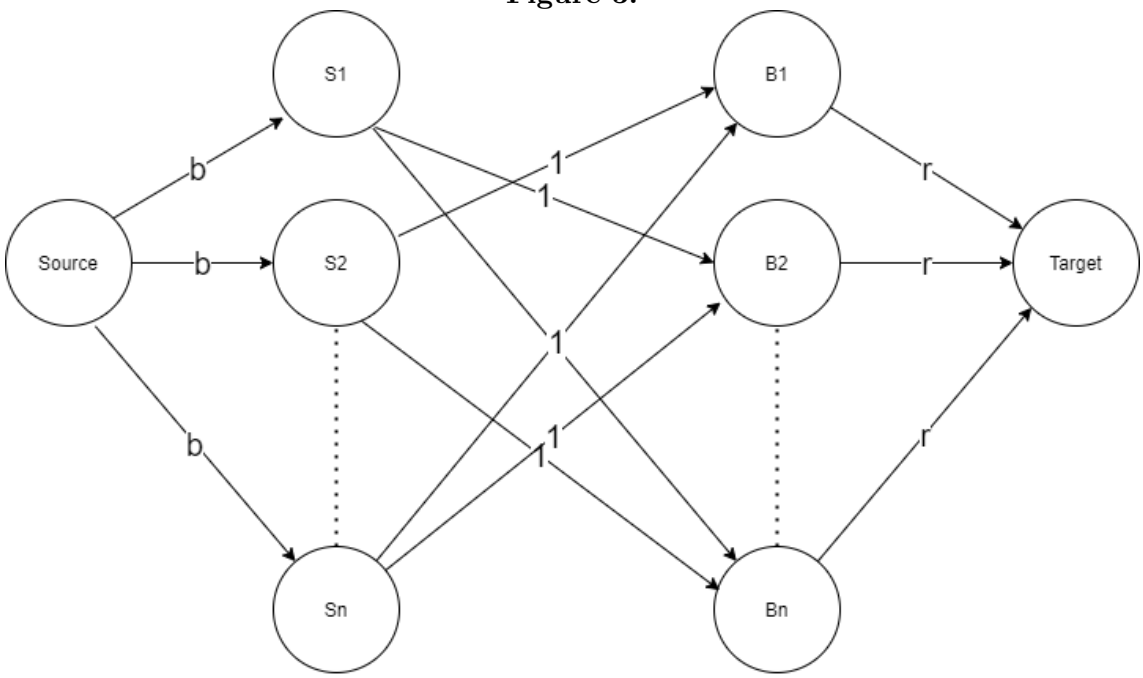
Figure 2:



Step 4:

To finish the graph, we connect each element B_i to the sink t with a capacity r in graph G. The final graph will look like that following:

Figure 3:



Step 5:

Finally we run a ford-fulkerson algorithm to find out the maximum flow of the graph. If the maximum flow is less then $len(sensors) * r$, this means that some backup set B_i does not have enough backup sensors (less than r backup sensors). So we will not able to find proper backup sets. Otherwise we return B, which means we find the backup sets successfully.

pseudo-code:

```
def distance(coord1, coord2):
  """
  calculate the distance between two coordinates
  """
  return sqrt((abs(coord1[0] - coord2[0]) ** 2) +
  (abs(coord1[1] - coord2[1]) ** 2))


def Bottleneck (p,f):
    return min (cf(u,v))


def AUGMENT (f,p,sensors,B):
    let b <- Bottleneck (p,f)
    for all (u,v) in p:
```

```
16          if (u,v) is a forward edge :
17              f(e) <- f(e) + b
18              if u in sensors and v in B:
19                  add sensor "u" to backup set "v"
20          if (u,v) is a backward edge :
21              f(e) <- f(e) - b
22              if u in sensors and v in B:
23                  remove sensor "u" from backup set "v"
24      return f
25
26
27  def Max_Flow (G,s,t,sensors,B):
28      set f(e)<- 0 for all e in E(G)
29      while there exist s-t path in Gf:
30          p = a simple (s,t) path in Gf
31          f' <- AUGMENT (f,p, sensors,B)
32          set f <- f'
33          compute Gf '
34          set Gf <- Gf '
35      return f
36
37
38  def backup_set(sensors, coordinates, d,r,b):
39      # 1
40      in_range = dict(key:sensors,  value:[])
41      for i in range(sensors):
42          for j in range(sensors):
43              if j == i:
44                  continue
45              if distance(coordinates[sensors[i]],
46               coordinates[sensors[j]]) <= d:
47                  in_range[sensors[i]].append(sensors[j])
48
49      # 2
50      G  = (s,none) # a directed graph with only a source point s
51      for i in sensors:
52          build forward edge (s->i) with capacity(s->i)=b, and store in G
53
54      # 3
55      B = [B1,B2,B3....Bn] # A list of backup sets (use python list here)
56      for i in sensors:
57          for j in in_range[i]:
58              build forward edge (j->B[i]) with capacity(j->B[i])=1, and stor
59
60      # 4
61      for i in B:
62          build forward edge (B[i]->t) with capacity(b[i]->t)=r, and store in
63
64      # 5 run Ford - Fulkerson
65      MaxFlow = Max_Flow (G,s,t, sensors,B)
```

```
66      if MaxFlow == len(sensors) * r:
67          return B
68      else:
69          print("choosing backup sets is not
70          possible with the given parameters")
71      return
```

2. **(8 points)** Briefly justify the correctness and runtime of the algorithm designed in the previous part.

To prove the correctness of our algorithm, let G be the flow graph we built using our algorithm from part (1), we will:

(1). Prove "if there exists backup set for each sensor, then max-flow(G) $= n * r$ where n is the total number of backup sets".

(2). Prove "if max-flow(G) $= n * r$ where n is the total number of backup sets, then there exists backup set for each sensor".

(3). Prove our python list of lists contains the correct backup sets with each list representing a backup set.

(1). To prove this claim, let's assume that there exists a backup set for each sensor, then this implies that we have n backup sets with each backup set having at least r sensors.

First, since r is the number of sensors that can be added to the backup set, and we cannot add part of the sensor to the set, nor produce part of the sensor and expect it to work. So, all flows are **integers**.

Consider a max-flow in G,

For each vertex s_i

(a). Capacity$(s, s_i) = b$, because each sensor can belong in **at most** b backup sets.

Then, flow$(s, s_i) \leqslant b$ or we go above our limit, which is not allowed.

Therefore, the conservation constraint holds for all s_i.

(b). Capacity$(s_i, B_j) = 1$, because all sensors are unique and can only be added to **exactly** 1 time to a certain backup set, and flow$(s_i, B_j) = 1$ **if and only if** the backup set for $s_j = B_j$ is within distance "d" of s_i and s_i hasn't being added to B_j yet.

So, flow$(s, s_i) = \sum_{j=1}^{N}$flow$(s_i, B_j)$, as we will only add s_i if another sensor is close enough and s_i hasn't been added.

Then, there are as many s_i been added as the number of sensors that are close to it, which implies flow_in(s_i)=flow_out(s_i).

Therefore, the conservation constraint holds.

For each vertex B_i

(a). Capacity$(s_j, B_i) = 1$, and flow$(s_j, B_i) = 1$ **if and only if** s_j is added to B_i and flow$(s, s_j) \leqslant$capacity(s, s_j) (in other words, the mandatory upper limit "b" for s_j hasn't been reached yet).

So, flow$(s_j, B_i) \epsilon \{0, 1\}$(0 when s_i is not assigned B_j, or 1 otherwise), and since

capacity$(s_j, B_i) = 1$, flow$(s_j, B_i) \leqslant$capacity(s_j, B_i).

Therefore, the capacity constraint holds.

(b). Capacity$(B_i, t) = r$ and flow$(B_i, t) = r$ **if and only if** r "qualified" sensors are added to backup set B_i.

And flow$(B_i, t) = \sum_{j=1}^{N}$flow$(s_j, B_i)$ for some s_j depending on the backup set S_i, so flow_in(c_i) =flow_out(c_i).

Therefore, we conclude that the conservation constraint holds.

Now, by proving the 2 constraints hold on all internal vertices, we have shown that the Ford-Fulkerson algorithm will terminate and returns the maximum flow to us.

Then, by construction of G, we know that max-flow(G) = "the total number of sensors that belong in a backup set" = m for some integer m. Out of the n backup sets in total, we need at least r sensors in every set to make the statement true, so we need at least $n * r$ sensors that are added to our backup sets to make the claim hold.

Moreover, due to the capacity$(B_i, t) = r$, we upper-bound our backup set's cardinality by r, so we only need to check if the maxflow(G) = m to know if we have a working group of backup sets.

Hence, if $m = n * r$, then we satisfies the conditions and proves that there exists a backup set for each sensor.

(2). Similar to the part (2), we prove that the maxflow(G) = "the total number of sensors that belong in a backup set" = m. Then if $m = n * r$ where n="number of sensor" and r="mandatory limit for a single sensor to exist in multiple backup sets", then by construction of G, we know that there are at least $n * r$ sensors that belong in a backup set, and precisely each backup set contain **exactly** r "qualified" sensors, which finishes our proof for this claim. (We can add more "qualified" sensor to a backup set, but we will threshold it to r)

(3). As we progress through the algorithm, our Augment function will add each candidate sensor s_i to the corresponding backup set.

If we used a backward edge to "undo" a previous choice, then our Augment function will also take out the sensor based on the "undo" path. (Which is also the reason why a greedy approach will likely to fail at this problem).

So, by the end of the algorithm when we have explored all possible path, we will have the correct sensors in the correct backup sets.

Therefore, by proving all 3 claims, we have shown that the algorithm is correct.

Time complexity:

We first consider the runtime of building the flow-graph G:

(1). initializing all vertices in G and connect each s_i to s and each B_j to t. (O(1+n+n+1)=>O(n) where n="number of sensors")

(2). build edges based on each sensor's distance. ($O(n(n-1)) => O(n^2)$, worst case being r=n-1, and all n sensors are within distance of one another).

So, the total time complexity of building a graph is "O(n)+$O(n^2)$ => $O(n^2)$".

Our modification to the BFS algorithm(checking if (B_i, t) still has capacity for flow)

takes constant time (look up in list and add to set), which is dominated by BFS's generic time complexity, which is then dominated by Ford-Fulkerson's complexity, so we will focus solely on complexity of Ford-Fulkerson.
Time complexity of the FF algorithm depends on the max-flow on G and the number of edges in G.

For the number of edges, let's consider the 4 layers of vertices in G.

(s, s_i): At this layer, we have n edges with n = "the total number of sensors". (check figure 1 in part (1) for details)

(s_i, B_j): At this layer, we can have maximum number of $O(n^2)$ edges, which is the scenario where r=n-1 and all sensors are within distance of one another (then wee connect each s_i with every backup set except B_i, check figure 2 in part (1) for details).

(B_j, t): At this layer, we have n edges, where each edge represent the number of sensors added to that backup set.

In total, $wehaveO(n + n^2 + n) => O(n^2)$ edges. And the maximum max-flow is $r * n$, which is the case where all n backup sets meet the minimum requirement of sensors (the worst case scenario). In conclusion, we have $O(r*n*n^2) = O(rn^3)$ for our Ford-Fulkerson algorithm.

Combining with the time complexity of building the flow-graph, we have $O(n^2) + O(rn^3) => O(rn^3)$ time complexity, which satisfies the requirement.