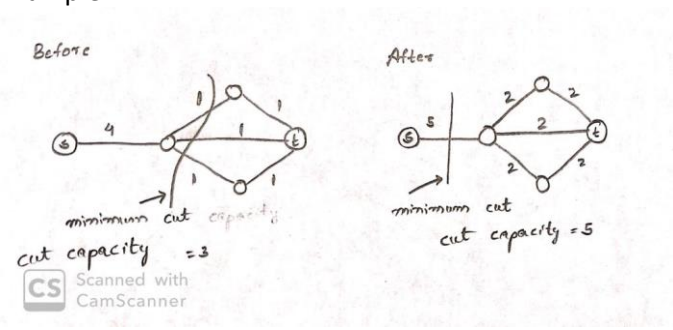
Chakradhar Reddy Donuri
E949F496

# CS-721 Advanced Algorithms and Analysis
# Homework-5

*Chakradhar Reddy Donuri*
*E949F496*

**Q1) (10 points) Let G = (V, E) be a flow network on which (A, B) is a minimum cut between source vertex 's' and sink vertex 't'. Suppose capacity of each edge of the network is increased by 1. Must (A, B) necessarily remain a minimum cut? If your answer is "yes" prove it. If your answer is "no", provide a counter example.**

**Answer:**
No, (A, B) may not remain to be a minimum cut after the capacity of each edge is increased by '1'.
Example:



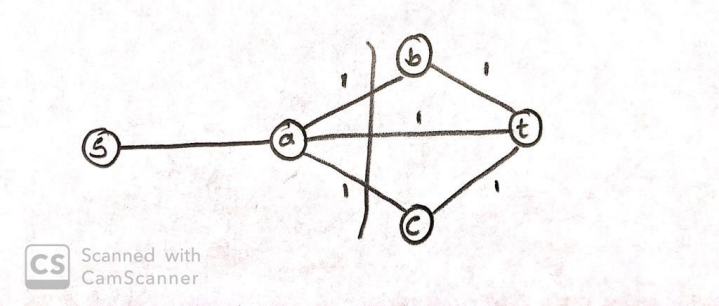

**Q2) (15 points) Let G = (V, E) be a flow network where each edge has identical capacity c. How will you compute max ow of this network? What is the running time?**

**Answer**:

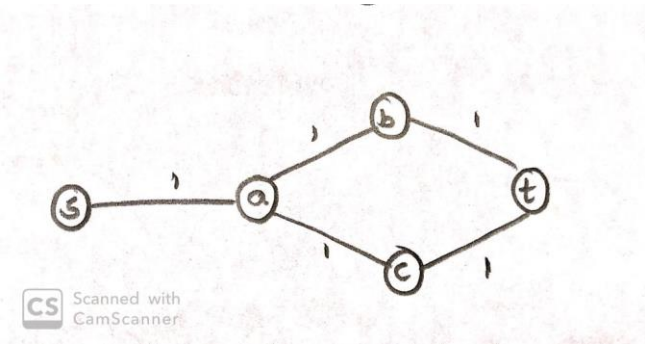
- We can use Ford- Fulkerson Algorithm to compute max flow of the network
- Let the Identical capacity of each edge be 'c', The source node can be connected to at most |V|-1 other nodes.
  Max Possible Flow is c * (|V|-1) = O(|V|)
- Hence Ford- Fulkerson Algorithm must run at most O(|V|) time
- The Bottle neck i.e. residual capacity of any augmenting path in the residual network is 'c' and thus Ford- Fulkerson Algorithm will run at most |V| iterations. Each Iteration takes O(|E|) time
- Total Running time is O(|V||E|).

**Q3) (10 points) Let G = (V, E) be a flow network carrying maximum ow. Let (A, B) be any cut of this network between source vertex s and sink vertex t. Let (a, b) belongs to E be an edge of this cut carrying the maximum flow. Suppose this edge is removed to get a new flow network G' = (V, E'), where E' = E \ {(a, b)}. Will the maximum flow of the resulting graph G' be strictly less than the maximum flow of G? If your answer is "yes", prove it. If your answer is no, provide a counter example.**

**Answer:**
No, the maximum flow of the resulting graph will not decrease. Consider the following example where capacity of each edge is 1.

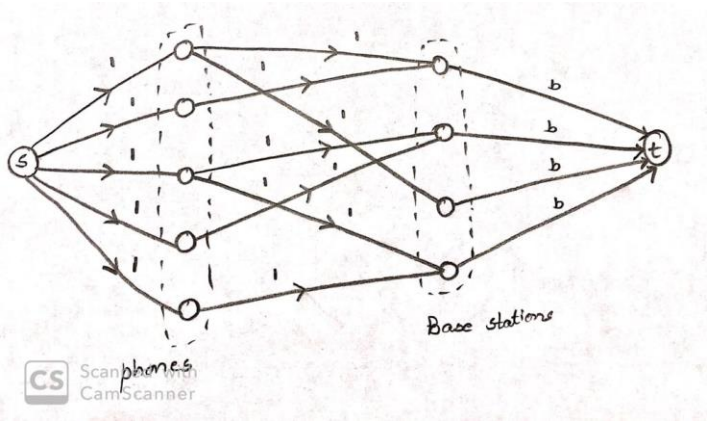



If the edge (a, t) is carrying the flow '1' and if we remove it, the resulting graph will still be having the maximum flow of '1'.




**Q4) (15 points) Suppose at any instant in the Wichita area there are k non-moving cell phones and p bases stations, where each cell phone can be directly connected to a base station (without going through other cell phones or base stations) which is not more than d distance away. Locations of all base stations and cell phones are given to you. Each base station can handle at most b cell-phones. Provide an efficient algorithm to assign cell-phones to base stations and analyze its running time. Hint: Can you use bipartite matching? Explain how will you convert the given problem to a flow problem.**

**Answer:**



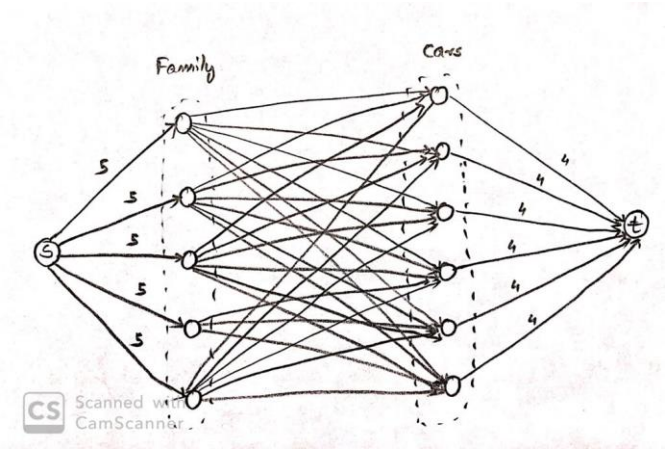

Construct a Flow network as follows

- Assign a node for each cell phone and each base station.
- Assign an additional source node 's' and additional sink node 't'
- Source node 's' has an edge to each cell phone with capacity '1'

- Each base station has an edge to sink node with capacity 'b'
- If the cell phone is within a distance 'd' of any base station, then assign an edge from this cell phone to the base station with capacity '1'.
- Due to Flow conservation, this construction will ensure that each cell phone will be assigned to at most one base station and each base station can handle at most 'b' cell phones.
- Now, we can use Ford-Fulkerson algorithm to solve the bipartite matching problem.
- The max flow is at most k *1 = k, thus Ford-Fulkerson algorithm will run at most 'k' iterations. The total number of edges in the network is at most
  p + kp + p = O(kp)
  where 1st term is due to edge between 's' and cell phones
  2nd term is due to all possible edges between cell phones and base stations.
  3rd term is due to edges between base stations and 't'
  Therefore |E| = O(kp)
- Running time of Ford-Fulkerson is O(f * |E|) = O(k. kp) = $O(k^2p)$
  Where 'f' is value of the maximum flow.

**Q5) (15 points) Several families are planning a shared car trip to Disneyland. To minimize the possibility of any quarrels, they want to assign individuals to cars so that no two members of a family are in the same car. Suppose there are m families and n cars, where each family has at most 5 members and each car can accommodate at most four persons. Explain how to formulate this problem as a network flow problem. Describe an efficient $O(m^2n)$ algorithm to assign individuals to cars so that no two members of a family are in the same car.**

**Answer:**




Construct a flow network as follows

- Assign a node for each family, each car, an additional source node 's' and sink node 't'.
- Source node 's' has an edge to each family node with capacity 5
- Each car node has an edge to sink node 't' with capacity 4
- Assign edges from each family node to each car node with capacity 1
- Due to flow conservation, this construction will ensure that no two family members are assigned in the same car
- Maximum possible flow of this network is at most 5m. Therefore $f^* <= 5m$
- Edges present in the network is at most
  5m + mn + 4n = O(mn) (since p +kp + p =O(kp))
  Therefore |E| <= O(mn)

- Running time of the Ford-Fulkerson algorithm is $O(f^*|E|) = O(m^2n)$

**Q6) (15 points) Suppose Dr. Smart Joe wins Nobel prize for proving that 3-COLOR problem (which is NP-complete) is solvable in the wort case in $O(n_5)$ time, where n denotes the number of vertices in the graph. Now answer whether the following statements are true or false by providing proper justification.**

**(a) All NP-complete problems are solvable in polynomial time.**

True, since 3-COLOR is NP-complete, all problems in NP (complete or not) can be polynomially reduced to 3-COLOR. Therefore, a polynomial time solution to 3-COLOR problem implies every problem in NP can be solved in polynomial time.

**(b) All problems in NP, even those that are not NP-complete, are solvable in polynomial time.**

True, since 3-COLOR is NP-complete, all problems in NP (complete or not) can be polynomially reduced to 3-COLOR. Therefore, a polynomial time solution to 3-COLOR problem implies every problem in NP can be solved in polynomial time.

**(c) All NP-complete problems are solvable in $O(n_5)$ time.**

False, because while any NP-complete may be polynomially reduced to 3-COLOR problem, size of the problem may increase polynomially during this reduction. In particular, suppose an NP-complete problem X, while reducing to 3-COLOR, expands its size from n to $n^2$ (i.e. polynomial increase). Now solution for 3-COLOR will require time $O((n^2)^5) = O(n^{10})$ to solve this problem in worst case.

**Q7) (15 points) Suppose you have been hired as a new manager by Koch industries and your responsibility is to develop and lead a new algorithmic research lab. As a first order of business you need to hire employees. You have a fat file filled with CVs of potential hires. Each candidate has expertise on a list of topics. As a policy of Koch industries, you cannot hire two employees with overlapping expertise. Your job is to hire the largest number of employees from your file such that no two employees have overlapping expertise. Show that this is an NP-complete problem.**
**Hint: First show that the decision version of the problem is an instance of INDEPENDENCE-SET problem. Show that this problem is in NP. Next polynomially reduce it from CLIQUE to argue that this is an NP-complete problem.**

- Suppose each potential hire is represented as node in a graph G= (V, E), where there is an edge between any two nodes u, v ∈ V if and only if u and v have overlapping expertise.
- The decision version of the problem is given a graph G = (V, E) and a positive integer g, is there a subset V' ⊂ V, |V'| = g such that no two vertices of V' is connected by an edge i.e. which is an instance of INDEPENDENCE-SET problem
- We need to prove INDEPENDENCE-SET is NP complete i.e. 1) INDEPENDENCE-SET is NP
  2) A known NP-complete problem can be reduced to INDEPENDENCE-SET.
- INDEPENDENCE-SET is NP
  Given an instance of INDEPENDENCE-SET problem and a candidate solution V' where V' ⊂ V and |V'| = g, it is easy to check in polynomial time (in |E| and |V|) whether no edges exist between any pair of vertices from V' or not.
- Next we will show how to reduce CLIQUE to INDEPENDENCE-SET

Let (G, g) be an instance of CLIQUE. Construct a new graph G' = (V, E'), where E' contains precisely those edges that are not present in 'G'. Then a set of nodes 'S' in a CLIQUE of G if and only if 'S' is an INDEPENDENCE-SET of G'. Therefore, (G', g) is an instance of CLIQUE. Clearly this construction can be done in polynomial time in |V| and |E|.

- Now we need to show
  If G' has an independent set of size g, how to efficiently recover a CLIQUE of size g from G.
  - Simple choose the set of vertices that are member of independent set of G'.

  If G' has no independent set of size 'g' then G has no clique of size 'g'
  - It is easier to prove contra-positive, that is, if G has a clique of size 'g' then G' has an independent set of size 'g'. This is again easy. Let 'S' be a clique of size 'g' in G. By our construction all pairwise edges between vertices in S are removed in G'. Therefore, S is an independent set of size g in G'.

**Q8) (15 points) 4-SAT is similar to 3-SAT, except that each clause contains exactly 4 literals. Show that 4-SAT is NP-Complete by reducing it from 3-SAT.**

- We need to show that 4-SAT is in NP.
  Given an instance of a 4-SAT problem and possible truth assignment to the variables, we can check in polynomial time if the formula is true or false. Next, we will reduce 3-SAT to 4-SAT. Let Ø = C1 ∧ C2 ∧ …. ∧ Cn be an instance of a 3-SAT problem involving n clauses. We construct an instance Ø' = C'1 ∧ C'2 ∧ …. ∧ C'n ∧ C'n+1 of 4-SAT as follows. Introduce a new variable x and for each i = 1,2, … n, set C'i = (Ci ∨ x) and set C'n+1 = ($\bar{x}$ ∨ $\bar{x}$ ∨ $\bar{x}$ ∨ $\bar{x}$). Clearly this can be constructed in polynomial time in n.
- We need to show that
  If Ø' has a satisfying truth assignment, then Ø also has a satisfying truth assignment.
  - If Ø' has a satisfying truth statement, then C'n+1 must be true. This can be achieved by setting x to be false. Setting x to be false still ensures that each Ci is true. Thus Ø is true.

  If Ø' has no satisfying truth assignment, then Ø also has no satisfying truth assignment
  - It is easier to prove contrapositive, that is, if Ø has a satisfying truth assignment the Ø' also has a satisfying truth assignment. This is again easy. Simply set x to be false. This will ensure Ø' is a satisfying truth assignment.