COL351: Analysis and Design of Algorithms (COE, IITD, Semester I 2016-17) Homework 05

Name: Shivank Goel

Entry number: 2011CS10565

There are 5 questions for a total of 75 points.

1. Solve the following two problems:

(a) (15 points) An edge in a network flow graph is called *downwards critical* if decreasing the capacity of this edge decreases the maximum flow in the network. Design an algorithm to find a downwards critical edge in a network. Give pseudocode, discuss running time and give proof of correctness.

Pseudocode

→ Run the ford-fulkerson algorithm to find a flow of maximum value. Let it be f.

→ Now find the min cut of the graph (A,B) where A = {set of all vertices reachable in G_f from source s} and B = V\A. (This can be done via DFS)

→ Now all the edges across this minimum cut will serve as downwards critical since if we decrease either of them we are getting a cut of value lower than value of min cut of original graph. And since in any graph maxflow ≤ capacity of any cut & hence it implies maxflow in new graph will decrease if we decrease capacity of any edge in min cut.

→ Output any edge (u,v) s.t $u \in A$, $v \in B$. That would be a downwards critical edge.

Time Complexity

→ Ford Fulkerson $O(mC)$

→ DFS. $O(m+n)$ & since in connected graph, $m \geq n$ $O(m)$

→ Total time complexity $O(mC+m) = O(mC)$

→ Note: If we used some another algorithm for finding max flow, time complexity changes accordingly

• Edmond Karp $O(m^2n + m) = O(m^2n)$ • Scaling flow $O(m^2 \log C)$

Proof Of Correctness

Let capacity of min cut in original graph G_i is C_f. Now let we dec capacity of any edge across min cut to C_f. We now that value of max flow in this new graph must be less than equal to C_f.

$$(\text{max flow})_{new} \leq C_f < C_i$$

Hence, maxflow will decrease. So $e=(u,v)$ st $u \in A$, $v \in B$ of any mincut (A,B) will be a critical edge.

(b) (5 points) An edge in a network flow graph is called *upwards critical* if increasing the capacity of this edge increases the maximum flow in the network. Design an algorithm to find an upwards critical edge in a network in case there exists one. Give pseudocode, discuss running time and give proof of correctness.

The max flow will increase if capacity of every min cut of original graph increase. So if we take an edge belonging to intersection of all min cuts & increase its capacity then max flow will increases. If such an edge doesn't exist i.e $\cap(A,B)$ = empty then no upward critical edge exist
$\forall (A,B)$ = min cuts of G

Since value of min cut remain same.

Pseudocode

→ Find a max flow in graph say f.

→ Find a min cut of the graph using DFS i.e A = all vertices reachable from s in G_f. $B = V \setminus A$

→ For (all edges e across min cut):
- → Inc the capacity of e by 1
- → Calculate new max-flow by trying to augment once more
- → If max-flow increases (return e)

→ return (null)

Time Complexity

→ Finding max flow $O(mC)$, OR $O(m^3)$ or $O(m^2 \log C)$

→ Finding min cut $O(m)$ (DFS $m \geq n$ $O(m+n) = O(m)$)

→ Max no. of edges across min cut $\leq$ max flow $\leq C$
∴ Edges across min cut $\leq \min\{m, C\}$
Each augmentation takes $O(m)$ time. Thus $O(mC)$

→ Total time complexity $O(mC + m + mC) = O(mC)$ (for ford fulkerson)

→ $O(m^2) + m + mC$ $= O(m(mn+C))$ for Edmond Karp

→ $O(m^2 \log C + m + mC) = O(m(m \log C + C))$ for Scaling max flow.

Proof Of Correctness

→ Since we have to find an edge $\in$ ~~$mf(A,B)$~~ (A,B) is a min cut, thus it must belong to all min cuts.

→ So we check all the edges of one of the min cuts we obtained ~~that~~ if they are candidate for upwards critical or not.

→ If on inc capacity of e to (C_e+1) max flow remains same, this implies there exists another min cut in which e is not present & hence e can't belong to intersection, so e can't be upward critical.

[Max flow stayed same, since $\exists$ another min cut of same value as of (A,B) that we found]

→ If on inc capacity of e to (C_e+1) max flow increase then this implies either this is the only min-cut or e belongs to all min cuts, since capacity of all min cuts must had increase to increase the flow.

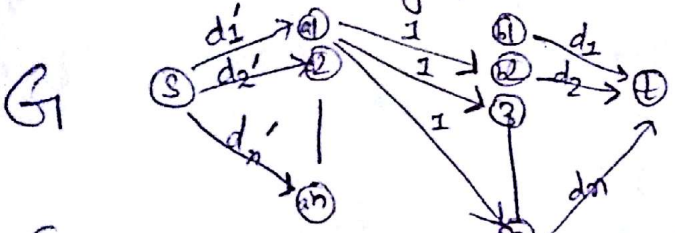
→ Note on inc capacity of e, max flow can't dec as previous flow f is still feasible.

→ Checking/Calculating max flow on inc capacity of e takes only 1 augmentation & hence $O(m)$ time. Since each augmentation inc flow by atleast 1 & flow can be inc by atmost 1 so atmost 1 more augmentation is req.

2. (10 points) You are given n pairs of integers $(d_1, d_1'), \ldots, (d_n, d_n')$ such that $\forall i. d_i, d_i' \geq 0$. You have to design an algorithm that determines if there exists a directed graph $G = (\{1, \ldots, n\}, E)$ such that the in-degree of vertex i is d_i and the out-degree of vertex i is d_i'. Your algorithm should also output a graph with the given degree sequence, in case there exists one. Give pseudocode, discuss running time and proof of correctness.

⊛ Algorithm & its Proof Of Correctness

We Consider the graph as follows



Note

$$\sum_{i=1}^{n} d_i = \sum_{i=1}^{n} d_i' = N \text{ (say)}$$

Consider graph G as follows. $(\{s, a_1, a_2 \ldots a_n, b_1, b_2 \ldots b_n, t\}, E)$

- $\exists$ edge (s, a_i) $\forall i$ with capacity d_i'
- $\exists$ edge (b_i, t) $\forall i$ with capacity d_i
- $\exists$ edge (a_i, b_j) $\forall i \neq j$ with capacity 1

⊛ No. of outgoing edges out of a_i of flow value = 1 will denote outdegree of ith vertex of G.

⊛ No. of incoming edges into b_j of flow value = 1 will denote in degree of jth vertex of G.

So we want $\sum_{e \text{ out of } a_i} f(e) = d_i'$ (outdegree) $\leq d_i$ (in degree) $\forall i$. This is only possible

iff $f((s, a_i)) = \sum_{e \text{ into } a_i} f(e) = \sum_{e \text{ out of } a_i} f(e) = d_i'$

$f((b_i, t)) = \sum_{e \text{ out of } b_i} f(e) = \sum_{e \text{ into } b_i} f(e) = d_i$

} Hence all edges (s, a_i) (b_i, t) $\forall i$ must be fully saturated for a possible solution.

⊛ **Claim** Thus there exist an flow of value N iff its possible to have n vertices with respective degrees. Also this would be max flow since $\{N = \sum_{e \text{ out of } s} c_e\}$

⇒ Suppose $\exists$ a flow of value N. Then we can use this flow to create a satisfiable graph. Create an edge from i to j iff $f(a_i, b_j) = 1$. Then in such a graph outdegree of all nodes will be d_i' & indegree will be d_i.

⇐ Suppose we have such a graph. Then we can generate a max flow of value N, by saturating all edges (s, a_i) & (b_i, t) & setting $f(a_i, b_j) = 1$ iff $\exists$ an edge from i to j in given graph.

Pseudocode

→ Create a graph G as mentioned in prev algorithm.
→ Run the ford fulkerson algorithm & let max flow be f with value v(f) & residual graph G_f
→ if v(f)<N return null
→ Now to output graph of given degree seq, initialize empty graph $\bar{G}$ of n vertices & no edges.
→
```
for (i=1 ; i≤n ; i++ ) {
    for ( j=1 ; j≤n ; j++) {
        if (i != j) {
            if (f(ai,bj) == 1) {add edge (i,j) to Ḡ }
        }
    }
}
```
→ Return $\bar{G}$

Time Complexity

→ Creating G = $O(n + {}^nC_2 + n) = O(n^2)$
→ ~~Final fulkerson~~
Finding Max flow $O(mC)$ or $O(m^2 \log C)$ or $O(m^2 n)$
C=N ∴ $O(mN)$ or $O(m^2 \log N)$ or $O(m^2 n)$
$m = O(n^2)$ ∴ $O(n^2 N)$ or $O(n^4 \log N)$ or $O(n^4)$
→ To create $\bar{G}$ $O(n^2)$
→ Hence best time complexity would be to use edmond Karp (since we expect $N > n^2$) = $O(n^4)$

else if $(N < n^2)$ the use ford fulkerson $O(n^2 N)$

3. (15 points) There is an $n \times n$ grid in which some cells are empty and some are filled. The empty/filled cells are given by an $n \times n$, 0/1 matrix F. Cell (i, j) is empty iff $F[i, j] = 0$. You have unbounded supply of 2×1 tiles (called dominoes). Each domino could be placed on the empty cells of the grid in horizontal and vertical manner (note that you need two consecutive empty cells on the grid for doing this). You have to design an algorithm to determine if the grid can be covered by placing these 2×1 dominos such that each empty cell is covered by exactly one domino. Give pseudocode, discuss running time and give proof of correctness.

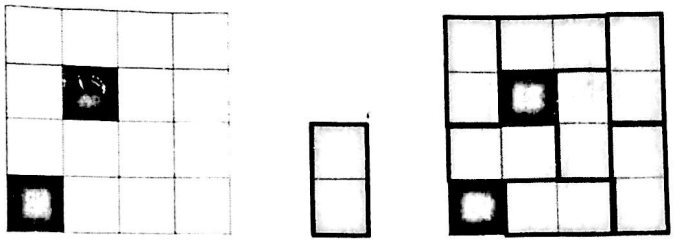
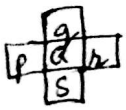


Figure 1: Example of a 4×4 grid that can be covered using 2×1 domino tiles.

Algorithm & Proof of Correctness

Consider a grid of $n \times n$. Lets assume boxes in grid are numbered from $1, 2, \ldots, n^2$ row-wise order. eg $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}$ 2×2

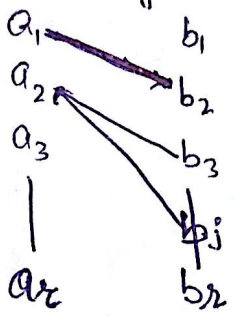
- Let box at index (i,j) (if its empty) be coloured green if $(i+j)$ is even. & be coloured white if $(i+j)$ is odd (like a chess board).
- Let filled box is coloured black.
- Now a domino will cover a white box & green box always & hence no. of white & green boxes must be same.
- Now a dominoes can be placed in 4 ways over a box 'a' as (q,a), (a,r), (a,s), (p,a).

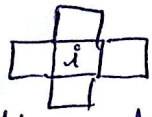


- Let $v_1, v_2, \ldots, v_x$ be empty vertices in the increasing order of their numbers. ($x \leq n^2$).
- $a_1, a_2 \ldots a_p$ be vertices of green colour.
  $b_1, b_2 \ldots b_q$ be vertices of white colour.

Note: Since $p+q = x$ & $p = q \Rightarrow p = q = \frac{x}{2} = r$ (say) because if p is not equal to q then we can't place dominoes since each domino exactly covers one green & one white box.

- So we create a graph like this. ~~like this.~~ bipartite

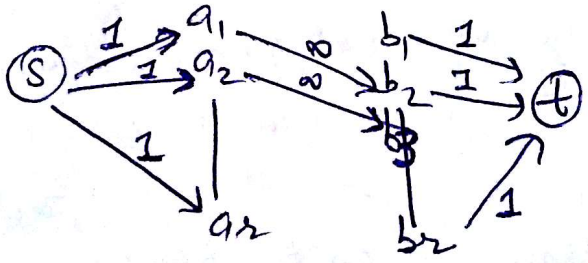
$G_1 =$ 

where an edge exist between a_i & b_j iff j is adjacent to i i.e 

j is either right, left, top or bottom to i.

- ⊛ Claim: We can cover grid with dominoes iff ∃ perfect matching of G_1.
Now since we want to cover all the a_i's & b_i's with dominoes there must exist a perfect matching i.e a matching of size r. If the perfect matching is feasible then it will tell us the way we should place dominoes i.e if (a_i, b_j) edge exist in perfect matching then place a domino at those 2 blocks in the grid. Conversely if we can place the dominoes in the grid then we can generate a perfect matching since if (a_i, b_j) has domino over it (in bipartite graph) than pick that edge & all such edges will give us a perfect matching.

- To find perfect matching we can create $\bar{G}$ as follows & check if the flow(maximum flow) $= r$. If its r then ∃ perfect matching else not as done in the class.

$\bar{G} =$ 

Pseudocode

→ For all the empty elements of grid partition them into two sets $A = \{a_1, \ldots, a_r\}$ if $(i+j)$ is even & $B = \{b_1, \ldots, b_s\}$ if $(i+j)$ is odd, where i & j are rows & columns where the element are present.

→ If $(r \neq s)$ return (false)

→ Create a bipartite graph with vertices A & B where (a_i, b_j) edge exist if they are neighbours of each other in grid. (say G)

→ To check the maximum matching $= r = s$, construct the graph $\overline{G}$ where s is connected to all a_i's with capacity 1 & t is connected all b_i's with capacity one & (a_i, b_j) is converted into a directed edge if it originally existed in G with capacity infinity

→ Find max-flow say f

→ If $(val(f) = r = s)$ return (true) else return (false)

Running Time

→ Partitioning elements $O(n^2)$

→ Creating $\overline{G}$: $O(r) = O(\frac{n^2}{2})$ For each cell we just have to check 4 of its neighbours & create 4 edges at max]

→ Finding max flow m = no. of edges in $\overline{G} = r + 4r + r = O(r) = O(\frac{n^2}{2})$

$C = \sum_{c \text{ out of } s} c(e) = O(r) = O(\frac{n^2}{2})$

Ford fulkerson $O(mC) = O(n^4)$

Edmond $O(mn^2) = O(n^4)$ (Since no. of nodes in $\overline{G} \propto n^2$)

→ Hence if we use edmond karp $O(n^4 + \frac{n^2}{2} + n^2) = O(n^4)$

4. (20 points) Given an s-t flow network G, design an algorithm to determine if there is a *unique* s-t min-cut in G. Give pseudocode, discuss running time, and give proof of correctness.

Algorithm & Proof of Correctness

→ If there is a unique s-t min-cut say (A^*, B^*) then increasing capacity of any edge across this min-cut (A^*, B^*) will increase the capacity of min-cut by 1. (Since all other cuts had capacity $> C(A^*, B^*)$ & $C(A^*, B^*)_{new} = C(A^*, B^*) + 1$, thus $C(\text{min cut new})$ increases by atleast 1 & hence max-flow would increase by atleast 1).

→ So if on increasing capacity of any edge e across (A^*, B^*) max-flow remains the same, then we can say there must be some other cut of capacity $C(A^*, B^*)$ that was also a min cut.

Algo

→ So we pick a min a cut (A^*, B^*)

→ We iterate over all e across (A^*, B^*)
  → Inc the capacity of e by 1
  → Check max flow by augmenting once more.
  → If max-flow doesn't increase (return false) // not a unique min-cut.

→ return (true).

→ Note to find if max-flow increased or not we need to just augment once more, since with each augmentation flow value increases by atleast 1 & if flow would have increased it would have increased by only 1. And hence it would take $O(m)$ time for all e across (A^*, B^*).

Pseudocode

→ Find a max-flow f in given G_1.

→ Find a min cut using DFS such that A = all vertices reachable from S in G_f & $B = V \setminus A$

→ For (all $e = (u, v)$ s.t. $u \in A$ & $v \in B$).

→ Set ($C(e) = C(e) + 1$)

→ Check if flow has increased by augmenting once more

→ If (flow has not increased) return (false)

→ return (true)

Running Time

→ Finding max flow $O(mC)$ or $O(m^2 \log C)$ or $O(m^2 n)$

→ Finding min cut $O(m+n) = O(m)$ (if $m \geq n$ for connected graph)

→ Augmenting a path $O(m)$

We have to augment at most $\min\{m, C\}$ times, since

no. of edges across min cut $\leq$ max flow value $\leq C$ (Since each edge carry atleast 1 flow)

& no. of edges across min cut $\leq m$ $\therefore \min\{O(m^2), O(mC)\}$

→ So total time complexity

Case-I : If ($m^2 > C$) → $O(mC) + O(m) + O(mC) = O(mC)$

Case-II : If ($m^2 < C$) → $O(m^2 n) + O(m) + O(m^2)$

$\equiv O(m^2 n) + O(m) + O(m^2) = O(m^3)$ ($m \geq n$)

5. (20 points) Given a bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$, an edge e is said to be *imperfect* iff there does not exist any perfect matching in G that includes e. Design an algorithm to output all imperfect edges of the given graph G. Give pseudocode, discuss running time, and give proof of correctness.

Pseudocode

→ Construct a graph $\bar{G}$ from G to find max-flow say f using bipartite matching graph $\bar{G}$ as discussed in class.

→ If $(v(f) < n)$ then no perfect matching exists for G & hence all edges of G are imperfect. Return(all edges $\in G$)

→ If $(v(f) = n)$ then we get a perfect matching using edges of $\bar{G}$ such that $f(e) = 1$ & $e = (u, v)$ s.t $u \in L$ & $v \in R$. Lets call this perfect matching M.

→ Now construct a graph G'' with vertices same as of G (L, R) s.t $\forall e \in G$ there exists a corresponding directed edge in G' from L to R. And $\forall e' \in M$ (found above) there exists a corresponding directed edge in G'' from R to L.

→ Imperfect = $\{\}$

→ For(all edges $(u, v) \in G$)

→ If u & v belong to same strongly connected component of G'' then (u, v) is in some perfect matching & is not imperfect

→ If u & v belong to different connected components of G'' then (u, v) is imperfect in original graph G. Imperfect = Imperfect $\cup \{(u, v)\}$

→ Return(Imperfect)

Time Complexity

→ Constructing $\bar{G_1}$ from G_1 = $O(n)$

→ Finding max-flow in bipartite = $O(mn)$ •

→ Constructing G_1'' = $O(n)$

→ Finding Strongly Connected Components in G_1'' = $O(m+n)$

Thus Overall Time Complexity $O(n+mn+n+m+n)$ = $O(mn)$

Proof Of Correctness.

→ If $v(f) < n$ ⇒ there is no perfect matching in G_1 & hence none of edges in $G \in$ perfect matching thus Imperfect = $\{e : e \in G_1\}$

→ Claim: (u, v) is imperfect iff u & v belong to different connected components of G_1''

- Let (u, v) lies in some perfect matching
- If $(u, v) \in M$ then in G'' ∃ edge from (u to v) & (v to u) thus belong to same SCC.
- If $(u, v) \notin M$ then there exist $v' \in R$ and $u' \in L$ s.t (u, v') $(u', v) \in M$

→ Clearly v is reachable from u in G_1''

~~$(u, v) \in M_2$.~~

~~Let (u', v) be matched in M_2~~

~~(u, v') to be matched in M_2~~

→ u is also reachable from v since we backtrack from v in orginal matching M as follows.

(u, u') ← (a_2, a_1) ← (a_1, u') ← (u', v) } edges of orginal matching M

And Hence v is reachable from u.

Reversing this argument proves that if (u, w) belong to same connected component in G'' then there exist a matching containing (u, w). Since we can obtain a path consisting of edges of M and G. & using this path we can generate a bipartite matching by using these edges.

(u, u') ——— (a_1, a_1) ——— (a_1, u') (u', w)