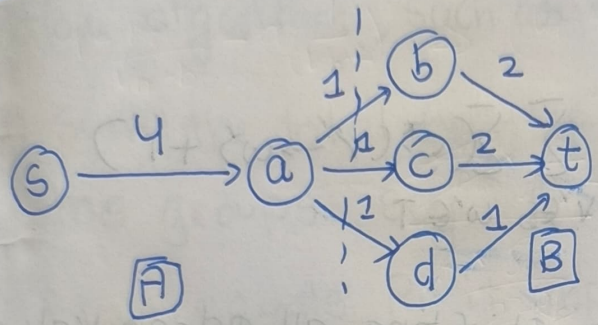
Problem Set - 5

Name - Varun Gupta

~~Phone No~~ Roll No - 2023101108

Ans 1) ~~Proof~~ Disproved using Counter Example

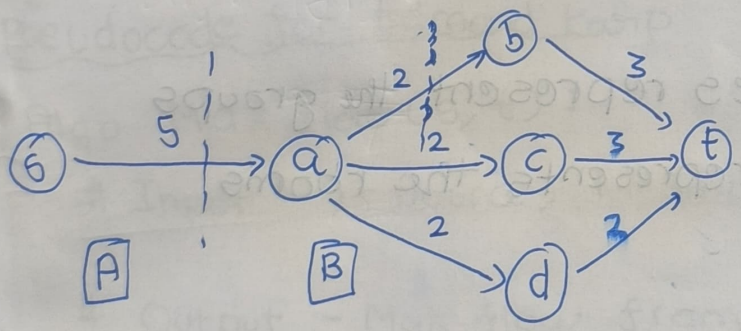


~~Given~~

Currently min_cut is $\{(a,b), \{a,c\}, \{a,d\}\}$

$$\text{min\_cut} = 1 + 1 + 1 = 3$$

Now adding 1 to each edge.



Now min_cut changed the new min_cut is $\{(s,a)\}$

$$\text{min\_cut} = 5$$

∴ min_cut Changed for $\forall e : e \in E$.

However one thing to note is that if $\forall e : e \in E$ is equal ~~for~~ $\forall e \in E$ Then min cut will remain same.

Let min cut (A, B) have capacity: $\sum_{v \in S} \sum_{w \in T} c(v \to w)$

Since (A, B) is min cut, $\sum_{v \in S} \sum_{w \in T} (c(v \to w)) \leq \sum_{v' \in S'} \sum_{w' \in T} c^*(v'$

Other cuts of s-t graph

Upon changing,

$$\sum_{v \in S} \sum_{w \in T} [c(v \to w) + 1] \leq \sum_{v' \in S'} \sum_{w' \in T} (c^*(v' \to w') + 1)$$

↳ This still holds ~~shifts~~ shifting all edges values by 1 does not affect min. cut if ce is same ∀ edge

---

Ans 2) This can be modelled as a Bipartite matching problem where:

- one set of nodes represent the groups
- the other set represents the rooms

Approach :-

Build a flow network

1) Create a source S and a sink T
2) Creat n nodes rep groups & m rep the rooms.
3) For each group i, add an edge to node from s to group i with capacity g_i, rep the no of researchers in grp i,

4) For each room j, add on edge from j to T ~~rep~~ with s_j capacity repr seating capacity

5) Add a edge between each group 'i' & room 'j' rep that one researcher from group i can potentially be seated in ~~rom~~ room j.

6) Compute the maximum flow from S to T using a max flow algorithm, such as the Edmond-Karp algo.

7) Sum the total num of researchers across all the groups say $R = \sum_{i=1}^{N} R_i$. IF the maximumum flow equals R, it implies that we have seated each researcher in different rooms which respected rom capacity otherwise it's not possible.

Pseudocode for Edmond Karp

```
Algo Max flow (s, t):
    # Input : s - Source, T - Sink, capacity - Matrix
                                    of edge capac
    # Output - Max flow from S to T
Initialize
    flow [u][v] = 0  ∀ edges (u,v)
    maxflow = 0
```

// Step1: find a Augmenting path P with BFS
↳ Parent Relation Formed

If Path not found then : break

Pathflow = ∞

v = T

while v != S :

    u = parent[v]

    Pathflow = min(pathflow, cap[u][v] - flow[u][v])

    v = u

v = T

while v != S

    u = parent[v]

    flow[u][v] += pathflow

    flow[v][u] -= pathflow

    v = u

maxflow += Pathflow

return maxflow

Proof of Correctness :→

1. Each edge in network respects the capacity constraints by the construction itself.
2. Each edge from group to a room has capacity 1, ensuring no ~~oth~~ more than one researcher from any group is seated in a single room satisfying diff. room requirements.
3. Maximum flow ensures all researcher are seated.

*Not Proving Edmond Karp as used in Class & Tut - 9

Analysis :-

Time Complexity → Network construction : $O(m \times n)$ ignoring 2

Max flow (Edmond) : $O(V \times E^2)$ where $V = n + m + 2$

$E = n + m + n \times m$

$\Rightarrow O((n+m) \times (n+m+n \times m)^2)$

Space Complexity → Nodes : $O(n+m+2)$

Edges : $O(n \times m)$

Edmond Karp : $O(n \times m + n + m)$

Total $\Rightarrow O(n \times m)$

better Time Complexity ↑

Use ford-fulkerson for ↑

Ans 3)

Part-1 : (By Contradiction)

Let's assume that the max flow value v^* is not even int.
$\Rightarrow v^*$ is an odd integer. By Max-flow min cut theorem, there exists a minimum cut S that has capacity equal to v^*. The capacity of cut S is the sum of capacities of edges crossing from S to its complement T

Let $E(S,T)$ so $v^* = \sum_{e \in E(S,T)} c_e$. Since each c_e is an even integer (given), their sum must also be an even integer. But we assumed v^* is odd. This is contradiction, ∴ Max flow must be even integer.

Part-2 :-

Let create a new network G' from our orignal network G : G' has the same structure as G For each edge e in G', its capacity $c_e' = c_e/2$ since all c_e in G were given even positive integers, all c_e' in G' are +ve integers. By max flow exist. theorem, G' has a maximum flow f' where all f'_e are integers ~~(This is a fundamental thm)~~

Now let's construct a flow f in our original network G ~~(a) f is fe~~

(a) f is feasible in G : If f' satisfies flow conserv at each vertex, so does 2f'. For each edge e, $f_e = 2f'_e \leq 2(c_e/2) = c_e$ (capacity const.)

(b) All flow values in f are even integers, f_e' is int $\therefore 2f_e'$ is even int.

(c) f is maximum in G : Let v' be value of max flow in G' $\therefore 2v'$ is value in G. If f isn't max in G $\exists$ flow of value $v > 2v'$ Then $v/2$ would be flow achievable in G'. Contradicts v' being maximum in G' $\therefore$ f is a max flow in G where every edge carries an even flow value.

Hence Proved

Ans 4)

Algo → ① If the flow on the edge e' is already atleast 1 unit below the orignal capacity then the old maximum flow is still valid & we're done

② Otherwise we need to find a path from s to t that contain edge e' and the flow on each edge of the path >0. We can use BFS for finding Path.

③ We decrement every edge on Path by one unit which decreases the total flow by 1 unit.

④ We then run one iteration of the ford-fulkerson algo to find the new max flow.
[Not describing ford-fulkerson algo as discussed in class]

Proof of Correctness →

If the flow on the edge e' is already atleast one unit below the original capacity, then reducing the capacity by 1 unit will not affect the max flow & this old max flow is still valid.

The new max flow value can't exceed the prev value as reducing the capacity will never lead to new augmenting paths to be found which were already not present in the previous graph. This is obvious

Now it can't exceed the capacity as flow <= capacity. If flow == capacity, we need to satisfy this contraint flow ≤ capacity which is violated so we first satisfy this constraint by decreasing the flow on the path from start by 1 unit. The path will exist as the flow on edge e' was positive [f(e) == C(e) ↙ +ve] So there will exist a path from s to t containing e' with positive flow on all edges. For finding this path, we ran a DFS from s to u where we took only those edges whose flow is +ve and this algo will definitely terminated on s from ~~s to~~ start u as the flow is acyclic, we run the same from v to t. For the final path we just concatenate s to u, edge e', and v to t. And decrease flow on this path by 1. This will decrease the max flow by 1 unit. Now either this is a maxflow or maxflow - 1. Running one iteration of the ford fulkerson will either find no augmenting Path, in which case the new maximum flow is the current flow or it will increase the flow by 1 unit. Now either this.

is a maxflow or maxflow - 1.

Running one iteration of the ford fulkerson algorithm will either find no augmenting Path, in which case the new maximum flow is the current flow, or it will increase the flow by 1 unit, which is the maximum possible increase given the capacity reduction. Since the flow values are integer valued and the flow strictly increases the algorithm will either find no augmenting path & terminate or it will increase the flow by 1 unit & then terminate

Analysis →

Time complexity: Finding Path - O(m+n)

Decrease - O(n)

One it of Ford - O(m+n)

Overall → O(m+n)

Space : Input - O(m+n) Visited array - O(n)

Visite array - O(n) Parent array - O(n)

Path - O(n) Path - O(n)

Total → O(m+n)

Ans 5) This problem can be solved using bipartite matching with Edmond-Karp if $y \geq 2x$, using individual clearing is optimal so we will clear all the mines while incurring a cost of x on each mine. For $y < 2x$, we use bipartite matching to maximum pairs of adjacent mines by viewing the grid as a chessboard, connecting cells of alternate colours containing a mine & finding the maxflow on this bipartite matching using Edmonds-Karp.

Pseudocode →

```
Minimum_Mine_Clearing_Cost(grid[1...n][1...m], x, y):
// Input: grid[i][j] = 1 if mine present, 0 otherwise
// x: Cost of clearing single mine
// y: cost of clearing adjacent pairs
  total_mines = COUNT_MINES(grid)
  if y >= 2*x:
      return total_mines * x
```

// Construct Bipartite graph
G = GREATE_FLOW_NETWORK (grid)

// Find max matching using edmand karp

max_pairs = EDMONDS_KARP (G, S, t)

remaining_mines = total_mines - 2 * max_pairs

return max_pairs * y + remaining_mines * x

CREATE_FLOW_NETWORK (grid):

Let G = (V, E) be a flow network with source 2 sink t.

~~// Add mine ver~~

```
for i = 1 to n:
    for j = 1 to m:
        if grid[i][j] == 1:
            Add vertex v[i,j]
            if (i+j) % 2 == 0:
                Add edge (S, v[i,j]) with capacity 1
            else:
                Add edge (v[i,j], t) with capacity 1

for i = 1 to n:
    for i = 1 to m:
        if grid[i][j] == 1 & (i+j) % 2 == 0:
            for (ni, nj) in [(i+1, j), (i-1, j), (i, j+1), (i, j-1)]:
                If Valid_cell (ni, nj) & grid[ni][nj] == 1:
                    Add edge (v[i,j], v[ni, nj]) with cap 1
```

return G

EDMONDS-KARP(G, s, t):

    Initialize flow f to 0
    while BFS finds augmenting path p from s to t:
        augment flow f along path p
    return value of maximum flow f

Proof of Correctness:

1. Case $y \geq 2x$:

   For any pair of mines, clearing individually costs $2x \leq y$. Therefore, clearing all mines individually is optimal.

2. Case $y < 2x$:

   Bipartite property:

   Grid cells alternate between black & white like a chessboard adjacent mines must be on diff. colours This ensures valid pairing poss.

   Maximum Matching Optimality:

   Each augmenting path in Edmonds-Karp repa valid way to increase matching

   BFS ensures shortest augmenting paths are found when no augmenting path exists, matching is maximum matching directly corresponds to maximum no of mine pairs that can be cleared together.

Cost Optimality:

Each matched pair saves $2x-y$ cost compared to clearing indiv. max matching maximizes these savings remaining mines must be cleared indiv
∴, Cost calculation gives global minimum.

Hence, we just found the max no of pairs that can be cleared using bipartite matching, cleared them using y rather than using x indiv (total $2x$) & cleared the rest using x. This gives the min cost. Graph constr. & maxflow finding using edmond karps gives max no of pairs that can be removed.

Analysis :-

Time Complexity :→ Grid Scan - $O(m \times n)$
Graph Const - $O(m \times n)$
Edmond - $O(V \cdot E^2)$ where $V = O(m \times n)$, $E = O(m \times n)$

⇓

Total → $O(m^3 n^3)$ $= O(m^3 n^3)$

Space Complexity :→ Grid - $O(m \times n)$
Grap - $O(m \times n)$
BFS queue & vis array - $O(V) = O(m \times n)$

⇓

Total : $O(m \times n)$

Ans 6)

The problem can be solved by binary searching on the weight w that each truck carries. For each weight w, we transform the original network into a flow network where each edge capacity becomes $\lfloor c/w \rfloor$, reps how many trucks carrying weight w can pass through that edge. We then check if x trucks can be routed from source to sink using max_flow.

Binary Search Range ⇒ [1, min (capacities)]

For each weight w:

| Transform edge capacities $c\_new = \lfloor c/w \rfloor$
| And max_flow in tranformed network
| Check if max flow ≥ x

Proof of Correctness →

Lemma 1: If x trucks each carrying weight w can be routed, them x trucks each carrying weight $w' < w$ can also be routed.

Proof: For each weight w, let the flow be f(e) on e

For any edge e with capacity c(e):

$f(e) \leq \lfloor c(e)/w \rfloor$

Since $w' < w$, $\lfloor c(e)/w \rfloor \leq \lfloor c(e)/w' \rfloor$

∴, Some flow pattern remain valids for w'.

Lemma 2: The max wt w* found by binary search is optimal.

Proof: By construction, w^* is feasible (n trucks can be routed).

For $w^* + 1$:

There exists some edge e where:

$[c(e) / w^* + 1]$ trucks can pass

This is strictly less than flow required in optimal solution.

∴ $w^* + 1$ is infeasible.

By lemma 1 & 2; w^* is optimal

Theorem: The algo finds the max total weight that can be transported.

Proof:- Total weight = $w^* \times x$ where w^* is max feasible weight

w^* is optimal by Lemma 2

Flow decomposition theorem ensures we can decompose flow into actual paths.

Integer flow property ensures valid truck routes exist

∴, $w^* * x$ is maximum total weight possible

Note:→ Here we are implicitly assume correctness of ford fulkerson

Analysis:→

Time Complexity:→ $O(m * n * \log(c_{max}))$

Binary Search:→ $O(\log(c_{max}))$ iterations

For each weight:

Ford-Fulkerson takes $O(m * f)$ where f is max flow

In our case, $f \leq x$ as we stop once we achieve flow of x.

∴ Each iteration takes $O(m * x)$

↳ $O(m * x * \log(C_{max}))$

Space Complexity: $O(m+n)$

Residual Graph: $O(m)$

Path finding in ford-fulkerson: $O(n)$

Additional variables: $O(1)$

Note: →

Edge cases

1.) No Path exists b/w source & sink
Returns 0

2) $x = 0 \Rightarrow$ Returns 0

3) All capacities are 0 ⇒ Return 0