Homework #5 Olivia Woodhouse

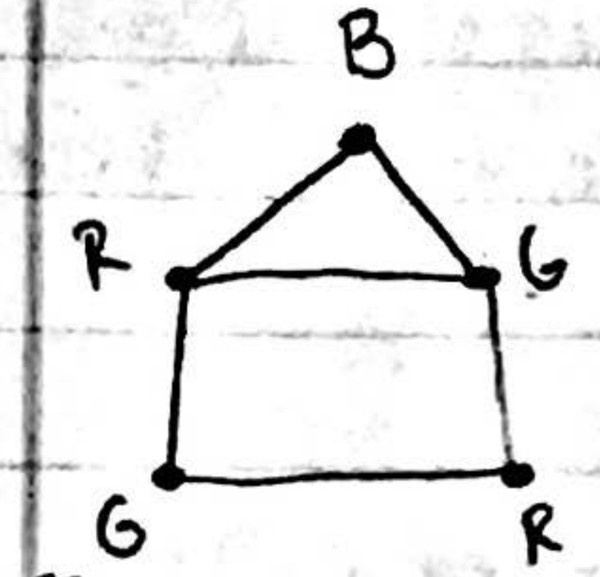
1.) Either prove NP-complete or show that it belongs to P by giving a polytime algorithm

Input: An undirected graph G

Question: Does G have a proper coloring with three colors R, G, B that assigns the color B to exactly one vertex?

→ I will show the input belongs to P by giving a polytime alg!

Algorithm:



- Create n subgraphs, each missing 1 vertex: So for all the vertices in G, create a graph that copies all the contents of G except for V (and all of V's edges)
- For each subgraph: Complete a 2-col using DFS (with R and G)
  - If 2-col was possible on the subgraph, add V (and all of V's edges) back into the graph and color V blue
  - If this is a proper 3-col of the subgraph: Output yes
  - Else, Output No
- If at least one of the "sub-tests" outputs yes: Output Yes
- Else, Output No

n times: $O(n(n+m))$

DFS 2-COL: → $O(m+n)$

Run Time: $O(n(n+m))$

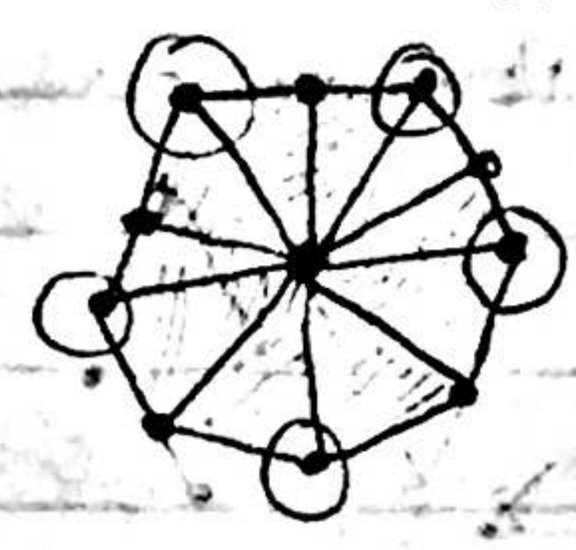
∴ This Algorithm is a polytime alg ⇒ belongs to P!

(2.) Triangle elimination Problem: undirected graph $G = (V, E)$, want to fin the smallest possible set of vertives $U \subseteq V$ s.t. deleting these vertives removes all the tiangles (i.e. cycles of length 3) from the graph

... Either show 3-factor approx. algorithm or give counter example

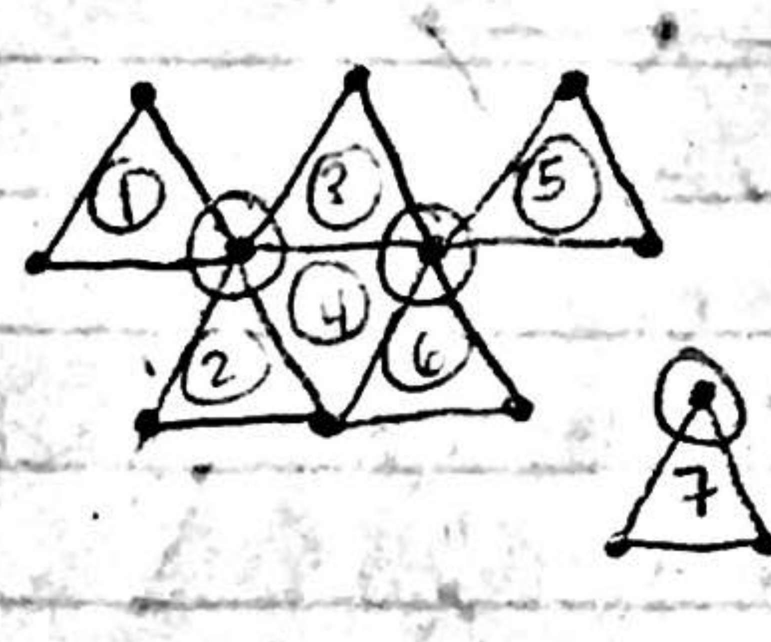
(I) Let T be the set of m triangles outputted by the algorithm.
If S is the set of vertices deleted by the algorithm
$|S| = 3m$ because the algo deletes all three vertices of each triangle
Optimal solution should be deleting at least one vertex from from each triangle $\Rightarrow$ $|Opt|$ is at least m
$\therefore$ Algorithm I is a 3-factor approximation

(II) Not a 3-factor approximation algorithm $\rightarrow$ counter example
let G:



Alg picks 1 arbitrarily chosen vertex
$\Rightarrow$ in this case output $|S| = 5$
BUT $|OPT| = 1$ by deleting the middle vertex
$\Rightarrow$ Output is greater than $3 \times OPt$ so Not 3-Factor approx. algo!

(III) If the alg is 3-factor approximation it deletes at most n-2 if n = # of vertices and at Least 1 triangle every loop



-deletes 1 vertex per loop and at least 1 triangle per loop

Let I be a set of subsets each containing the number of triangles that overlap (share a single vertex)

$I = \{\{1,2,3,4\}, \{3,4,6,5\}, \{7\}\}$

1^{st} deletion $\rightsquigarrow \{\{5,6\}, \{7\}\}$

So Solution S must be of Size at most I because if the vertex is deleted that all the triangles in a subset of I have in common, all of the triangles in that subset will also be deleted

The optimal solution is also at most I because it is deleting the smallest possible set of vertices such that deleting these vertices removes all the triangles meaning you want to delete the vertex that is in the most number of triangles

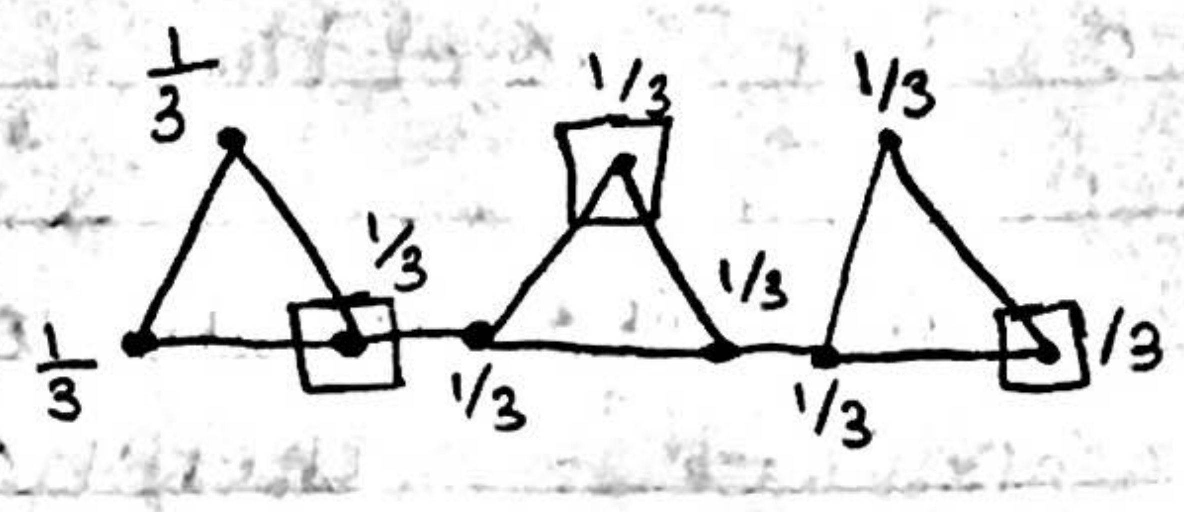
$\Rightarrow$ Output $\leq 3 \times opt(LP) \leq 3 \times$ min set of U $\Rightarrow$ Algorithm III is 3-factor approx

(IV) Integer Linear Program:

$\min \sum_{u \in V} x_u$

s.t. $x_u + x_v + x_w \geq 1 \quad \forall uvw \in$ Triangles of G

$x_u \in \{0,1\} \quad \forall u \in V$



Linear Programming Relaxation of previous ILP:

$\min \sum_{u \in V} x_u$

s.t. $x_u + x_v + x_w \geq 1 \quad \forall uvw \in$ Triangles of G

$0 \leq x_u \leq 1 \quad \forall u \in V$

Let the Optimal solution for the linear program be x_u^*

Let $\hat{x}_u = \begin{cases} 0 & \text{if } x_u^* < \frac{1}{3} \\ 1 & \text{if } x_u^* \geq \frac{1}{3} \end{cases}$

Since $x_u^* + x_v^* + x_w^* \geq 1 \Rightarrow x_u^* \geq \frac{1}{3}$ or $x_v^* \geq \frac{1}{3}$ or $x_w^* \geq \frac{1}{3}$

$\Rightarrow$ at least one of $\hat{x}_u, \hat{x}_v$, or $\hat{x}_w$ is 1

$\Rightarrow \hat{x}_u + \hat{x}_v + \hat{x}_w \geq 1 \Rightarrow$ this is a feasible solution to the ILP

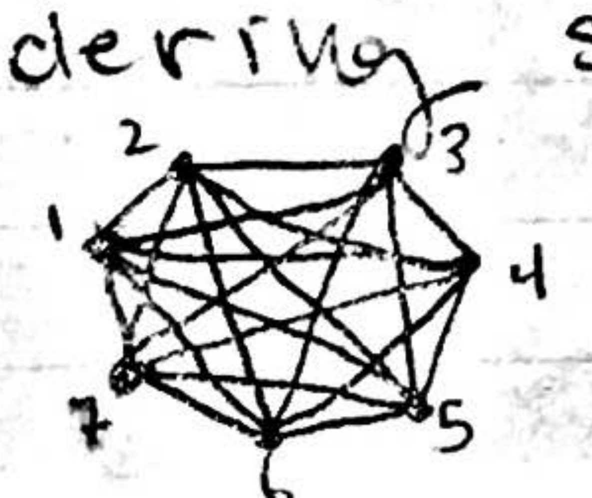
$\Rightarrow \sum_{u \in V} \hat{x}_u \leq 3 \sum_{u \in V} x_u^*$ because $\hat{x}_u \leq 3 x_u^*$

$\Rightarrow$ Output $\leq 3 \times$ Opt LP $\leq 3 \times$ min set of $U \Rightarrow$ Algorithm IV is a 3-factor approximation

3.) Input: Graph G, ordering $\forall v \in V$ s.t. every vertex has at most 5 neighbors that appear before v in that order (but v can also have other neighbors that appear later in that order)

Show: a) G can be colored w a 6-coloring

→ Since v must have at most 5 neighbors appearing before it for every $v \in V$, the max-clique in the graph must be no more than 6 or else an ordering such as this would not be possible



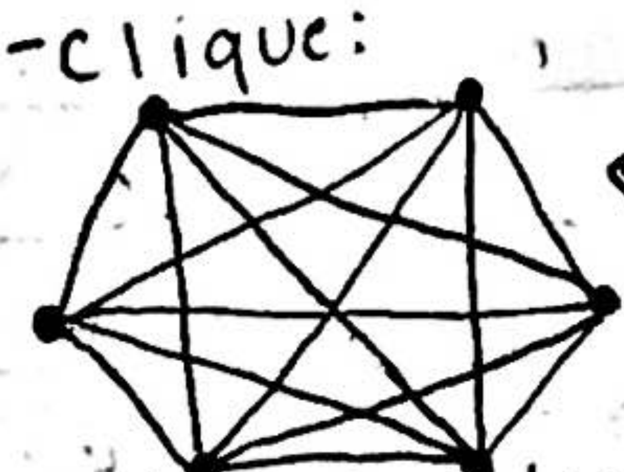
$\{1, 2, 3, 4, 5, 6, 7\}$

↑ any vertex in this spot must have at least 6 of its neighbors before it $\Rightarrow$ ordering not possible

Since the max clique must be less than or equal to 6 this means that G can be properly colored using 6 colors because if G contains a clique of size k all the vertices in the clique are pairwise adjacent therefore a proper coloring must have as many colors as the size of the largest clique.

b) Color the vertices of G with 5 colors so as to maximize the number of edges that are properly colored (that is they have different colors on their endpoints)... Design a $\frac{14}{15}$-factor approximation algorithm for this problem

6-clique:



← 15 edges ⇒ For a clique of size 6 there must be 15 edges

Assign a color randomly to the vertices in the graph
For $\forall e \in E$, let $e=1$ if endpoints have different colors
let $e=0$ if endpoints have the same color

The probability that $e=0$ is $\frac{1}{15}$ and the probability that $e=1$ is $\frac{14}{15}$

Taking these probabilities into account, let X_{uv} for all $uv \in E$
$= [\text{propability that } e_{uv}=0] \times 0 + [\text{propability that } e_{uv}=1] \times 1 = \frac{14}{15}$

so the output $= \sum_{uv \in E} X_{uv} = \frac{14}{15} m$

Since the optimal solution of coloring the vertices with 5 colors so as to maximize the number of edges that are properly colored $= m$ (m being the number of edges)

⇒ This algorithm is <u>at least</u> $\frac{14}{15} \times$ optimal solution
⇒ This algorithm is a $\frac{14}{15}$-factor approximation

(4) x = a string of length n of 0's and 1's

- del(x, i) (for $1 \le i \le n$) deletes the i-th bit of the string x, and thus decreases its length to n-1
- set(x, i, b) (for $1 \le i \le n$ and $b \in \{0, 1\}$) sets the i-th bit of x to the bit b
- insert(x, i, b) (for $1 \le i \le n+1$ and $b \in \{0, 1\}$) inserts b after the (i-1)-th bit of x, and thus increases the length of x

* d(a, b) = the smallest number of operations required to convert a to b

(a) Show that d(a, b) = d(b, a)

- if a del(x, i) function must be called to remove an extra element in a that is not in b, insert(x, i, b) can be called on b to insert that element that is in a and not in b ⇒ del(x, i) and insert(x, i, b) are opposite functions both of cost 1
- if set(x, i, b) function needs to be called to change an element in a that does not match the one in the same place in b, set(x, i, b) can also be called on b to change an element in b that does not match the one in a

⇒ d(a, b) = d(b, a) because the optimal solution for d(a, b) can be matched by performing a different set of functions (as displayed above) for d(b, a)

(b) Explain briefly how d(a, b) can be computed in polynomial time using dynammic programming.

Dynamic Programming → table to store values produced by subproblems

Algorithm: - Loop through each bit similtaneously in both Strings
- If bits are the same ignore and continue to iterate remaining bits recursively
- If bits are different we have 3 options:

1. del(x,i)
2. set(x,i,b)
3. insert(x,i,b)

...So call each of these three options on the character in a (because we are trying to determine d(a,b) which is specifically the number of operations it takes to get a → b) and continue to recursively compute the cost of d(a,b) for each

* The idea is that we will store these values along the way to solve the problem in polynomial time

Ex: a = 1001 b = 11

	X	1	0	0	1
X	0	1	2	3	4
1	1	0	1	2	3
1	2	1	1	2	2

This chart can then be used to compute exactly what changes to make to minimize d(a,b)

This solution take $O(n^2)$ time ⇒ This approach is in polynomial time

(C) Input: 3 strings a, b, c and we want to find a fourth string d that minimizes d(a,d)+d(b,d)+d(c,d)

→ Give a $\frac{4}{3}$-approx algorithm

a = 1001
b = 101
c = 111!
d = ? 0 11
d(a,b) = 1
d(b,c) = 2
d(a,c) = 2

- Let d be an empty string to start and construct an undirected graph s.t. the edge

- This alg takes the min cycle that includes all vertices in the constructed graph, using the "difference" of each vertex (or string) and adding the difference onto the string in d (so the "difference" from d→b is 101 so d becomes 101)

- The alg then finds the total min cycle in the graph (a→b→c→a)

- The alg then divides the 1^{st} cycle by the 2^{nd} cycle to find d

The min cycle length of the cycle that includes all nodes = 4

The min cycle length = 3

1^{st} cycle length = 4, 2^{nd} cycle length = 3

d = (d→b) 101 → (b→a) 1001 → (a→c) 11|1001

just add onto d

∴ Output ≤ $\frac{4}{3}$ × OPT

Cycle = 3+1+2+4

MinCycle: a→b→c→a

⇒ Alg is a $\frac{4}{3}$-approximation alg