Lecture 6:

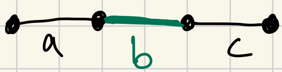
Sublinear time algorithms
via
simulating greedy algorithms

Matching:

$M \subseteq E$ is "matching" if $\forall v \in V$,
v is in at most one edge in M

"maximum matching": largest $|M|$

"maximal matching": can't add any other edge & make it bigger ← greedy algorithm



$\{a, c\}$ maximum & maximal matching

$\{b\}$ is maximal matching

Today's goal:

Estimate size of maximal matching in degree bounded graph.

Why?

- relation to Vertex cover:

these are disjoint

$VC \geq MM$ ← for each edge in matching, ≥ 1 endpt must be in VC

$VC \leq 2 \cdot MM$ ← put all MM nodes in VC
if any edge not covered by VC, violates maximality of MM

$\Longrightarrow$ 2-approx to VC

- a step towards approx maximum matching

Note: if $\deg \leq \Delta$, maximal matching is $\Omega(n/\Delta)$

see this by running greedy algorithm.
each step removes $\leq 2\Delta$ edges
so size of matching $\cdot 2\Delta \geq m$

$\Longrightarrow$ size of MM $\geq \frac{n}{2} \cdot \frac{\text{ave deg}}{\text{max degree}}$

Greedy sequential matching:

$M \leftarrow \emptyset$

$\forall e \in (u,v) \in E$

  if neither u,v matched

    add e to M

Output M

output only depends on ordering of input edges

observe: M maximal since if $\underset{(u,v)}{e} \notin M$, either u or v already matched earlier

Oracle reduction framework:

assumption: given deterministic oracle $O(e)$ which tells you if $e \in M$ or not in one step.

Reduction algorithm:

at least $\Omega(\frac{1}{\Delta})$ fraction of nodes in M

- $S' \leftarrow s = \Theta(\frac{\Delta}{\varepsilon^2})$ nodes chosen iid
- $\forall v \in S'$

$$X_v = \begin{cases} 1 & \text{if any call to } \mathcal{O}((v,w)) \text{ for } w \in N(v) \text{ returns "yes"} \\ 0 & \text{o.w.} \end{cases}$$

v is matched

- output $\underbrace{\frac{n}{2s}}_{\text{since 2 nodes matched for each edge in } M} \sum_{v \in S'} X_v + \underbrace{\frac{\varepsilon}{2} \cdot n}_{\text{makes underestimate unlikely}}$

Behavior of output: why does it work?

$$|M| = \frac{1}{2} \sum_{v \in V} X_v$$

$$E[|\text{output}|] = E\left[\frac{n}{2s} \sum_{v \in S'} X_v\right] + \frac{\varepsilon}{2} n$$

$$= \frac{n}{2s} \sum_{v \in S'} \underbrace{E[X_v]}_{E[X_v] = \frac{2|M|}{n}} + \frac{\varepsilon}{2} n$$

$$= \frac{n}{2s} \cdot s \cdot \frac{2|M|}{n} + \frac{\varepsilon}{2} \cdot n$$

$$= |M| + \frac{\varepsilon}{2} \cdot n$$

$$\Pr\left[\,\left|\text{output} - E[\text{output}]\right| \geq \frac{\varepsilon}{2} n\right]$$

$$\shortparallel$$

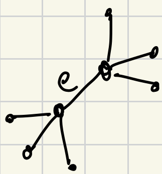
$$\Pr\left[\,\left|\frac{n}{2s}\sum_{v\in S} E[X_v] - |M|\right| \geq \frac{\varepsilon}{2} n\right] < \frac{1}{3}$$ by additive Chernoff-Hoeffding $\blacksquare$

How do you implement the oracle?

Main idea   figure out "what $\underbrace{\text{would greedy}}_{\text{how?}}$ do on (v,w)?"

according to which input order?
do we need to figure out greedy decisions on all earlier nodes?

Implement oracle based on greedy?



To decide if edge e in matching:

- need to know decisions for adjacent edges that came before e in ordering
- do not need to know anything about edges after e in ordering since not considered by greedy until after e is processed.

processing e (high level):

(recursively) call procedure on all edges adjacent to e & before e in ordering

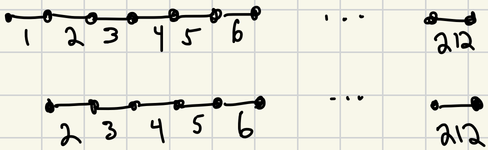
if any adjacent edge before e in ordering is matched,

then e is not matched

else e is matched

problem greedy is "sequential"
can have long dependency chains

example:



even if you know the graph is a line, how do you know if query edge is odd or even in the order?

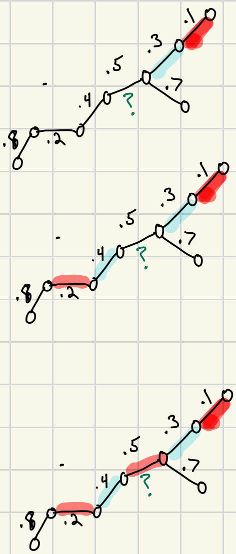
"break" length of dependency chains?

idea: assign random ordering to edges

via random # in [0,1]

example

is edge 0.5 in M?



- recurse on 0.3
  - recurse on 0.1
    - no other adjacent edges so match 0.1
    - so don't match 0.3
- don't recurse on 0.7 since bigger
- recurse on 0.4
  - recurse on 0.2
    - don't recurse on 0.8 since bigger
  - match 0.2
  - don't match 0.4
- match 0.5

Implementation of oracle

assume random ranks r_e assigned to each edge e

to check if $e \in M$:

$\forall e'$ neighboring e,

- if $r_{e'} < r_e$ recursively check e'
  - & if $e' \in M$ return "$e \notin M$" & halt
  - (else continue)

return "$e \in M$"

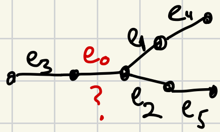
← since no e' of lower rank is in M

Correctness? follows from correctness of greedy

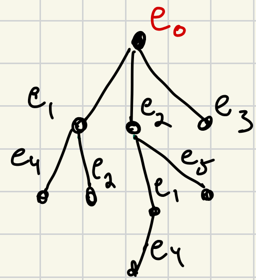
query complexity?

Claim expected # queries to graph per oracle query is $2^{O(\Delta)}$ ← poly d is achievable if recurse "smallest first"

Claim $\Rightarrow$ total query complexity is $\frac{2^{O(\Delta)}}{\varepsilon^2}$



pf of claim:



- Consider query tree where root node labelled by original query edge.
- Children of each node are labelled by adjacent edges.

- algorithm only queries tree paths that are monotone decreasing in rank

- $\Pr[\text{path } p \text{ of length } k \text{ explored}] = \frac{1}{(k+1)!}$

  represents k+1 edges

- # edges in original graph at dist k in tree $\leq (2\Delta)^k$
- $E[\#\text{ edges explored at dist } k] \leq \frac{(2\Delta)^k}{(k+1)!}$.
- $E[\text{total edges explored per query}] \leq \sum_{k=0}^{\infty} \frac{(2\Delta)^k}{(k+1)!}$

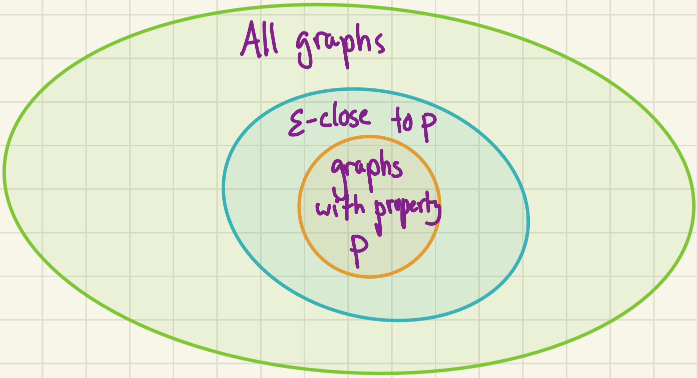
$$\leq \frac{e^{O(\Delta)}}{\Delta}$$

- $E[\text{total query complexity (for all queries)}] \leq \Delta \cdot \frac{e^{O(\Delta)}}{\Delta} = e^{O(\Delta)} = 2^{O(\Delta)}$ ∎

New topic:

What sort of approximations make sense for decision problems?

Property Testing



P is a subset of graphs

Can we distinguish graphs in P from graphs that are not in P? not even ε-close?

Goal if G has property P, pass
if G ε-far from P, fail

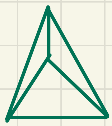
(if G is ε-close, can either Pass or fail)

For today: def deg $\le \Delta$ graph G is ε-close to P if can remove some $\le \varepsilon \Delta n$ edges to turn G into $G' \in P$

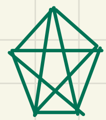
Planarity:

<u>def</u> a Planar graph can be drawn in plane st. edges intersect only at endpts.

e.g.



planar



not planar

<u>Cool Thm</u> [Kuratowski]

G is planar iff does not contain K_5 or $K_{3,3}$ as minor

K_5: Complete graph on 5 nodes

$K_{3,3}$: Complete bipartite graph with 3 nodes on side

minor: subgraph repeatedly contract edges into nodes