CSCC63 TUT0002
Tutorial 5

The P Complexity Class
Proving that a Decision Problem is in P

Decision Problems: Problems with Yes/No answers

P: Set of all Decision Problems that can be solved in Polynomial Time by a Deterministic TM

O(N^c) for a fixed C

Prove that something is in P
1. Justify why its a Decision Problem
2. Write an algorithm that solves it in P time

MODEXP = { <a, b, c, p> | a, b, c, and p are positive binary integers and $a^b = c \mod p$ }

input size:

- Strings: number of characters
- Graphs: number of nodes + number of edges
- Arrays/Lists: number of elements
- Positive Binary Integer a: $\log a$

Time Complexity of naive exponentiation

for binary integers N = log b <-> b = 2^N

computing a^b = O(b) = O(2^N)

3^6

1. 3 * 3
2. 3 * 3 * 3
3. 3 * 3 * 3 * 3
4. 3 * 3 * 3 * 3 * 3
5. 3 * 3 * 3 * 3 * 3 * 3

Fast Exponentiation

a^b = (a^2)^(b/2) for even number b
(a) (a^2)^((b - 1)/2) for odd number a

accomplished in O(log b) = O(log 2^N) = O(N)

3^6
1. (3^2)^(3) = 9^3
2. computing 9^3 = 9 * (9^2)^(1) = 9*(9^2)

MODEXP = { <a, b, c, p> | a, b, c, and p are positive binary integers and a^b = c mod p }

1. Yes this is a decision problem, <a, b, c, p> is either in the set MODEXP or its not.
2. Give a Polynomial time Algorithm

```
# F computes a^b via fast exponentiation
F on input <a, b>:
  if b = 0:
    return 1
  if b is odd:
    return a * F<(a^2), ((b - 1)/2)>
  if b is even:
    return  F<(a^2), (b/2)>
Via fast exponetiation, F computes a^b in O(log b) = O(N)
P on input <a, b, c, p>:
  return F<a, b> == c % p # O(logb)
```

Since a DTM can solve MODEXP in Polynomial time, MODEXP is in P

TRIANGLE = { <G=(V, E)> | G contains a triangle }

1. TRIANGLE is a Decision Problem, A graph either contains a triangle or it doesn't
2.

```
P on input <G=(V, E)>:
  for v in V: # O(|V|)
    for u in V: # O(|V|)
      for w in V: # O(|V|)
        if (v, u) in E and (u, w) in E and (w, v) in E: # O(1) given adjacency matrix representation of E
             accept
reject
```

This runs on O(|V|^3|E|), |V| and |E| less or equal to |G|, O(|G|^4)

Show that P is closed under Union
Suppose that L1 and L2 are in P, L1 union L2 is in P

If L1 and L2 are in P then there exists M1 and M2 such that they solve L1 and L2 in polynomial time.

```
P on input <x>:
  run M1 on x #O(N^c)
  run M2 on x #O(N^k)
  if either accept, accept
  else reject
```

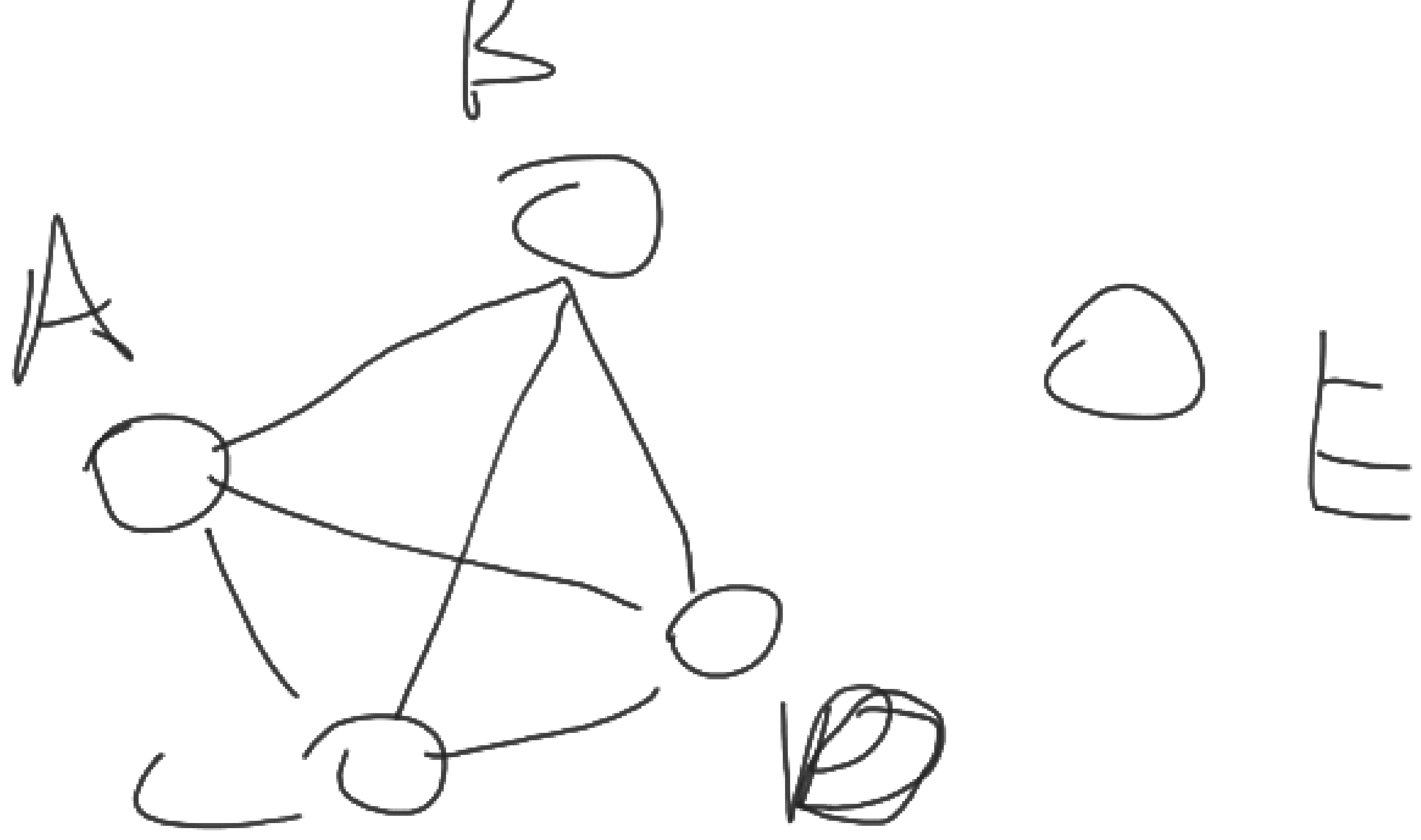
P runs on O(N^max(c, k)) this is clearly polynomial time.

NP are all DP that can be verified in P time

We write a verifier that determines whether or not some input is accepted by the DP, we also get a certificate

Clique = { <G, k> | such that G contains a k clique }

G:



{A, B, C, D}

forms a 4-clique

Verify Clique in P time

let certificate C be the set that makes up this clique.

```
V will take input <G = (V, E), k, C>:
  assert that |C| >= k
  for each node v in C:
    # assert that v is connected to every other node in C
    for each node u in C:
        assert (v,u) and (u, v) exists in E
  is all assertions pass, accept otherwise reject
```

Show that P is closed under *
Suppose that L1 is in P, L1* is in P

If L1 is in P the there is some TM M that solves L1 in P time

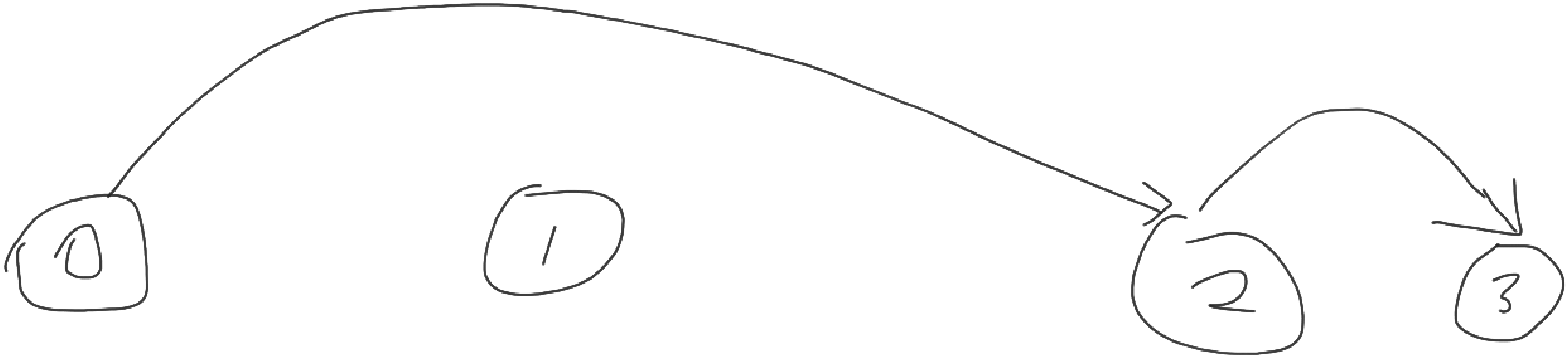
```
P on input <x>:
  define a graph A = (V, E) on |x| nodes, initializing all edges to 0
  for i = 0 to |x|: # O(|x|)
    for j = i to |x|, j != i: # O(|x|)
      if M accepts x[i:j]: # O(|x|^c)
        E[i][j] = 1 # O(1)
```

run DFS on graph A from node 0, to node |x| if there is a path, accept otherwise reject. O(|V| + |E|)

This algorithm is O(|x|^(2 + c))

xyz, xy in L1, z in L1

x[0:2] x[2:3]



string: xyxyxyxy, xy L1, xyxy in L1, xyx in L1