Cook-Levin (using CIRCUIT-SAT)

given combinational circuit, can we set inputs to make output = 1

1) CIRCUIT-SAT ∈ NP

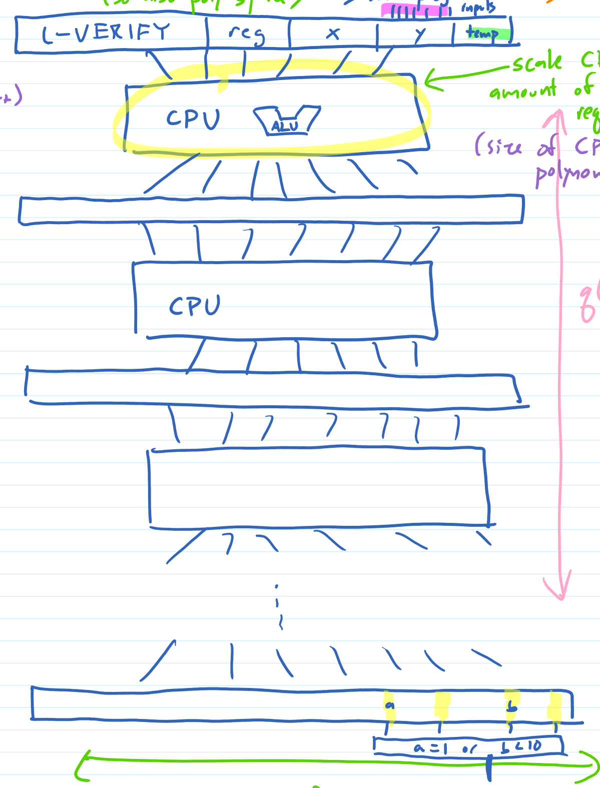
2) ∀ L ∈ NP, $L \le_p$ CIRCUIT-SAT [given input x to L, construct circuit C (in poly-time) s.t. L(x) = YES ⟺ C is satisfiable]

there is some poly g(|x|) s.t. L-VERIFY stops in ≤ g(|x|) steps

Suppose L∈NP. Then there is a poly-time verification alg L-VERIFY for L

(so also poly-space) → some program

there is a y s.t. L-VERIFY(x,y) = YES ⟺ L(x) = YES



for (int i=0; i<4; i++)
  a[i]=0;
  b[i]=0;

a[0]=0;
a[1]=0;
a[2]=0;
a[3]=0;
b[0]=0;
b[1]=0;
b[2]=0;
b[3]=0;

scale CPU for amount of memory required for x

(size of CPU is polynomial in |x|)

g(|x|) copies

a=1 or b<10
if (....)
  return YES
else
  return NO
return result;

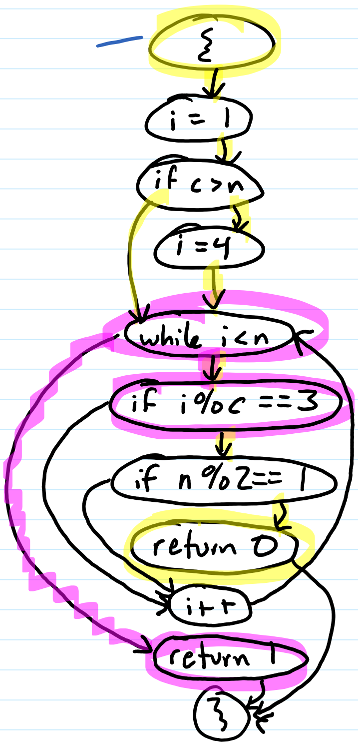
p(|x|) memory (polynomial)

to answer "is L(x) = YES", build this circuit and ask is it satisfiable – is there a y s.t. circuit outputs 1 / L-VERIFY(x,y) = YES

Control Flow Graphs

8 9

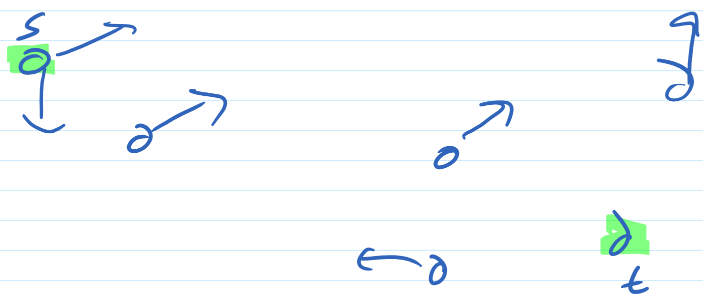
```
int foo(int n, int c)
{
  int i = 1;
  if (c > n)
  {
    i = 4;
  }
  while (i < n)
  {
    if (i % c == 3)
    {
      if (n % 2 == 1)
      {
        return 0;
      }
    }
    i++;
  }
  return 1;
}
```



vertex = line of code

edge (u,v) means
v can follow u

paths { ⇝ return 0
- shortest: 6 — polynomial (BFS)
- longest: 7 — NP-complete (reduce from HP)
- number: 2
- average: 6.5



\# simple paths s ⇝ t is ≥ 10

how can I convince you / how would you verify my evidence?

↳ show you each path

verify no repeated paths
all claimed paths are valid

NUM-PATHS-VERIFY(G, s, t, k, ℓ) — ℓ: len(ℓ) could be ≥ n! **not** poly-sized certificate

polynomial in length of input? YES

```
for i = 1 to len(ℓ)
    for j = i+1 to len(ℓ)
        if ℓ[i] = ℓ[j]          ← poly-time
            output NO
for i = 1 to len(ℓ)
    check if ℓ[i] is a valid path   ← poly-time
    if not, output NO
if len(ℓ) < k
    output NO
else
    output YES
```

NUM-PATHS ∈ NP?

not by this alg....

#P: counting problems where answer is # accepting paths for some nondeterministic poly-time algorithm

COUNT-PATHS: given directed graph and two vertices s, t
count distinct simple paths $s \rightsquigarrow t$

COUNT-PATHS $\in$ #P

```
PICK-AND-VERIFY-PATH(G, s, t)
p = s
while len(p) < n and last in p ≠ t
    randomly choose vertex v not in p
    if edge (last in p, v) then add v to p
    else output NO
if last in p = t
    return YES
else
    return NO
```

each path $s \rightsquigarrow t$ is one sequence of choices that gets me here

#SAT: given φ, how many satisfying assignments are there?

#SAT $\in$ #P

```
PICK-AND-VERIFY-ASSIGN(φ)
for i = 1 to n
    randomly set x_i to T or F
if φ(x) is T      return YES
else              return NO
```

2^n paths to get here — one for each potential assignment

only the satisfying assigns get us here

use to get answer to counting problem!

```
COUNT-ASSIGNS(φ)
count ← 0
for each execution path
    if PICK-AND-VERIFY-ASSIGN(φ) = YES
    for that execution path then
        count ← count + 1
return count
```

2^n execution paths

#P-complete: X is #P complete if
1) $X \in$ #P
2) for all $Y \in$ #P, $Y \le_p X$

def of reduction modified for counting problems

#SAT is #P-complete

There are problems in P for which the corresponding counting problem is #P-complete!
for example, 2-CNF-SAT

(3-SAT except 2 terms per clause)

PSPACE

SAT-BRUTE-FORCE(φ)
$n \leftarrow$ # variables in φ
for $i = 0$ to $2^n - 1$ — 2^n iterations
  generate ith possible assignment A
  if A makes φ true
    return YES
return NO

polynomial space

TSP-BRUTE-FORCE(G, k)
for $i = 0$ to $n! - 1$ — $n!$ iterations
  generate ith permutation of vertices P
  $t \leftarrow$ total weight of corresponding tour
  if $t \leq k$
    return YES
return NO

P = set of decision problems solvable in polynomial time

PSPACE = set of decision problems solvable in polynomial space

NUMBER-PATHS, SAT, TSP $\in$ PSPACE

P, NP, and PSPACE

$P \subseteq NP$

$P \subseteq PSPACE$

$NP \subseteq PSPACE$

$P \subseteq NP \subseteq PSPACE$

proper ?

$P \subsetneq NP \subsetneq PSPACE$ $\quad$ $P = NP \subsetneq PSPACE$ $\quad$ $P \subsetneq NP = PSPACE$ $\quad$ $P = NP = PSPACE$