Class 12

Algorithmic LLL

Announcements

- HW5 due Friday
- HW6 out now!
- HW7 isn't due until after fall break! (Friday 12/2)
- No class on Tuesday: Democracy Day!

If you are eligible to vote, then 

!

φ is a k-CNF formula on n variables with m clauses.

Recap: Algorithmic LLL (for k-SAT)

- Given φ:
  - Choose a random assignment σ for each of the variables that appear in φ
  - While there is some clause C of φ that is not satisfied:
    - Update σ by randomly re-selecting the variables that appear in C.
  - Return σ

- **Theorem:**
  - Suppose that each clause C in φ shares variables with at most $d + 1 = 2^{k-c}$ clauses (including C itself), for some constant c.
  - Then φ is satisfiable and the algorithm above finds a satisfying assignment quickly.

$\mathcal{A}$ is a collection of bad events determined by variables in V.
Vbl(A) is the set of variables involved with $A \in \mathcal{A}$

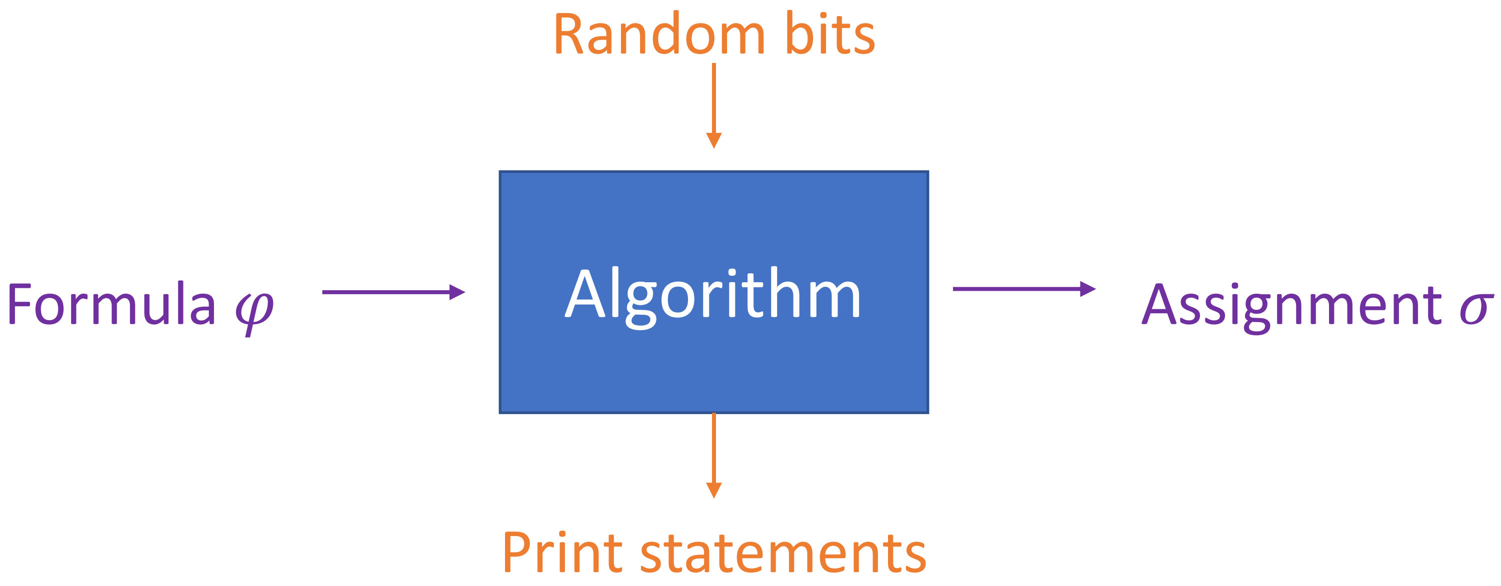
Algorithmic LLL more generally

- Given V and $\mathcal{A}$:
  - Choose a random assignment σ_v for each of the random variables $v \in V$
  - While there is some $A \in \mathcal{A}$ so that $A(\sigma) = 1$:
    - Choose (arbitrarily) an event A with $A(\sigma) = 1$.
    - Update σ by re-selecting $\{\sigma_v : v \in \mathrm{Vbl}(A)\}$ randomly.

- Suppose that for all $A \in \mathcal{A}$:
  - $|\Gamma(A)| \leq d + 1$
  - $\Pr[A] \leq \frac{1}{e(d+1)}$
- Then this algorithm will find an assignment to the variables in V so that no event of $\mathcal{A}$ occurs with $O\left(\frac{|\mathcal{A}|}{d+1}\right)$ re-randomizations.

Proof of Algorithmic LLL

- Add some print statements to our algorithm.
- If the algorithm runs for too long, it will be too good of a compression algorithm.



Questions?

Algorithmic LLL, Quiz?

Q1: Applying alg. LLL

- $S_1, S_2, \dots, S_M \subset X$ are sets of size $k < |X| = N$
- Each S_i intersects at most 10 other sets S_j
- Color points of X red or blue iid with prob ½.
- A_i is the event that S_i is monochromatic.

- |V|=
- d =
- For what k does alg. LLL apply?
- What is expected number of re-randomizations?

Q2. Changing the proof

- What if we print "trying to fix clause i_ℓ" instead of "trying to fix the ℓ'th child"?

- Q2.1 How many bits get outputted in print statements?

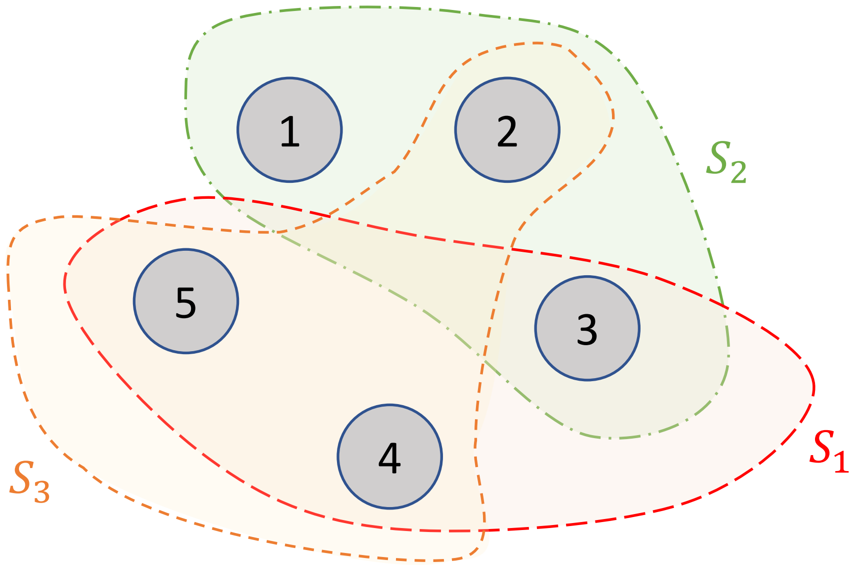
- Q2.2. What would that prove?

Today: More practice with the Algorithmic LLL

- We saw the proof for k-SAT
- Today you'll prove it for set coloring!

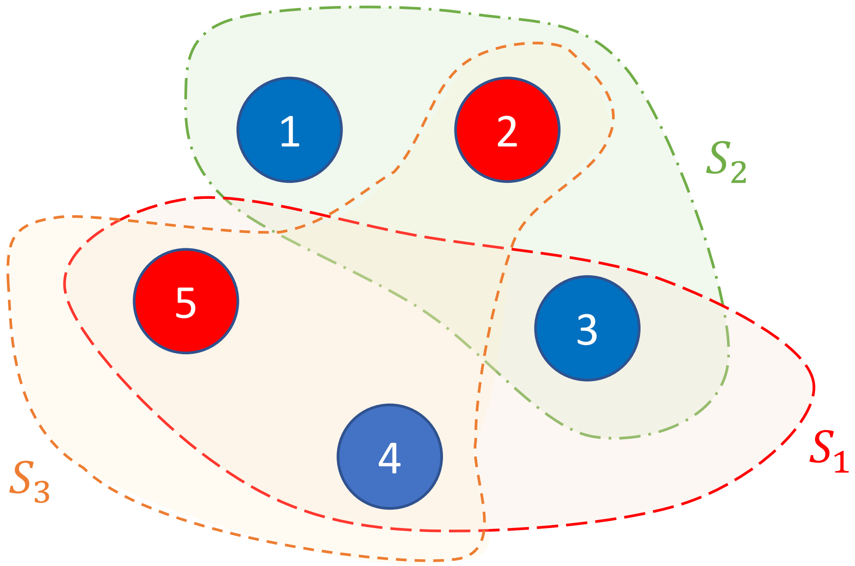
The problem

- n points, $\{1,2,\ldots,n\}$
- m sets, $S_1, S_2, \ldots, S_m \subseteq \{1, 2, \ldots, n\}$
- Each set has size k.
- Each set overlaps with no more than d other sets.
- Goal: color the n points red or blue so that none of the sets is monochromatic.



The problem

- n points, $\{1,2,\ldots,n\}$
- m sets, $S_1, S_2, \ldots, S_m \subseteq \{1, 2, \ldots, n\}$
- Each set has size k.
- Each set overlaps with no more than d other sets.
- Goal: color the n points red or blue so that none of the sets is monochromatic.



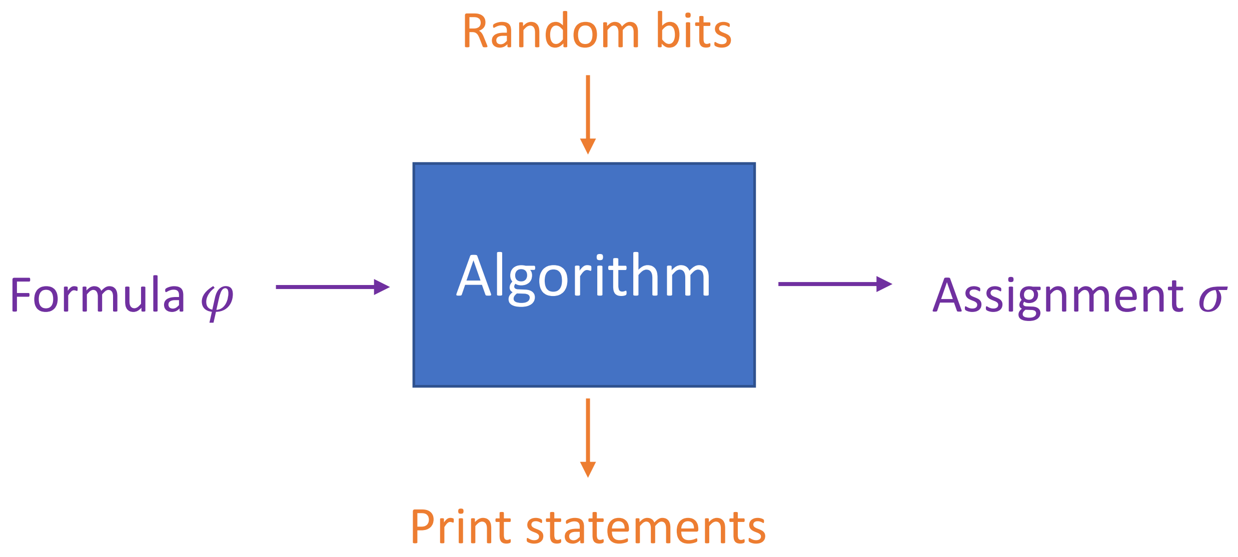
Algorithmic LLL gives an algorithm to do this

- While not done:
  - Pick a monochromatic set, S_i.
  - Re-color all of the numbers in S_i, uniformly at random.

- But we didn't prove that this works.
  - We only proved it for k-SAT

- Goal of today:
  - Mimic the k-SAT argument to give an algorithm that provably works for no-monochromatic-coloring.

Quick recap of the proof idea for k-SAT



- We wrote the algorithm in a recursive way and added some print statements.
- From the print statements, you could figure out the random bits that went into the algorithm.
- If the algorithm runs for too long (too many re-randomizations), then we can compress the random bits!
- But that’s impossible.

Group work!

- Give a proof!
  - What is the same between the k-SAT proof and this proof?
  - What needs to change?

Outline:

- We wrote the algorithm in a recursive way and added some print statements.
- From the print statements, you could figure out the random bits that went into the algorithm.
- If the algorithm runs for too long (too many re-randomizations), then we can compress the random bits!
- But that’s impossible.

For inspiration, here was the k-SAT algorithm

Your job: adapt to set-coloring!

- **FindSat**($\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$):
  - Choose a random assignment σ for each of the variables that appear in φ
  - For each clause C_i in φ that is not satisfied:
    - $\sigma \leftarrow$ **Fix**(φ, i, σ)
  - Return σ

Fixing set i!

- **Fix**(φ, i, σ):
  - Update σ by re-randomizing every variable that appears in the clause C_i
  - Let $C_{i_1}, C_{i_2}, \ldots C_{i_{d+1}}$ be the clauses that share variables with C_i
  - For $j = 1, \ldots, d+1$:
    - If C_{i_j} is violated:
      - $\sigma \leftarrow$**Fix**(φ, i_j, σ)
  - Return σ

Trying to fix the j'th child

All done with this level.

After T re-randomizations,

I give up. I've got σ

What needs to change?

- **FindSat**($\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$):
  - Choose a random assignment σ for each of the variables that appear in φ
  - For each clause C_i in φ that is not satisfied:
    - $\sigma \leftarrow$ **Fix**(φ, i, σ)
  - Return σ

Fixing set i!

- **Fix**(φ, i, σ):
  - Update σ by re-randomizing every variable that appears in the clause C_i
  - Let $C_{i_1}, C_{i_2}, \ldots C_{i_{d+1}}$ be the clauses that share variables with C_i
  - For $j = 1, \ldots, d+1$:
    - If C_{i_j} is violated:
      - $\sigma \leftarrow$**Fix**$(\ \varphi, i_j\ , \sigma)$
  - Return σ

Trying to fix the j'th child

All done with this level.

After T re-randomizations,

I give up. I've got σ

Our algorithm?

- **FindSat**($S_1, S_2, \ldots, S_m$):
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \ldots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \ldots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$**Fix**(i_j, σ)
  - Return σ

Trying to fix the j'th child

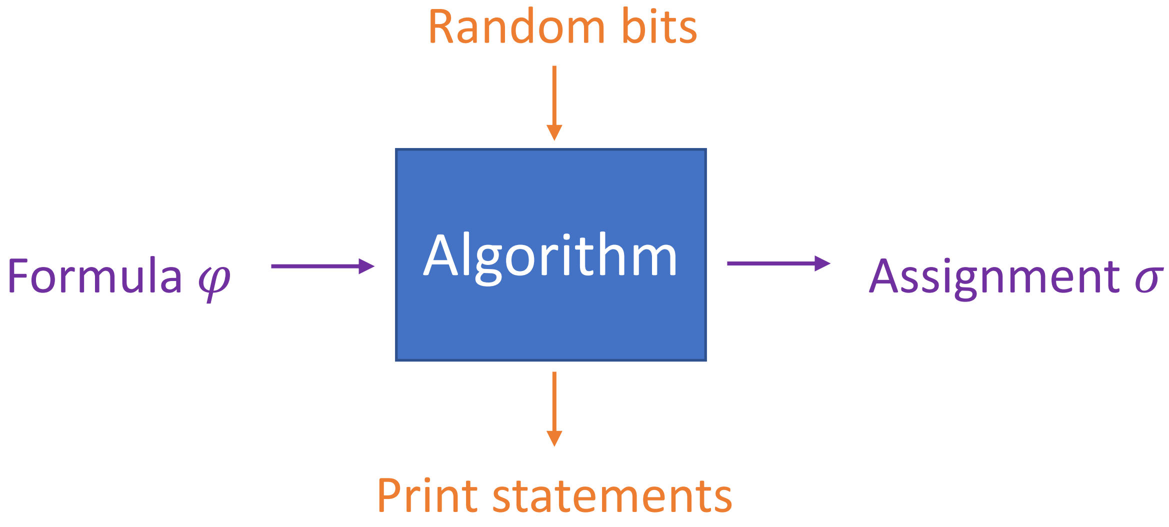
All done with this level.

After T re-randomizations,

I give up. I've got σ

To do the proof

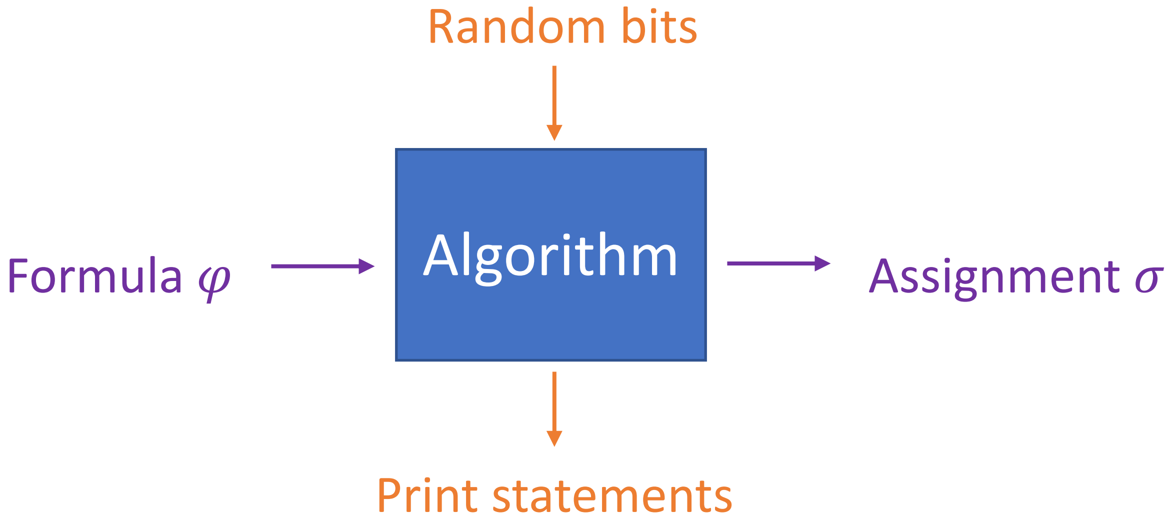
- We need to count the number of random bits that go in in the first T re-randomizations.
- We need to count the number of bits of print statements that come out in the first T re-randomizations.
- We need to argue that we can recover the random bits that go in from the print statements that come out.



To do the proof

- We need to count the number of random bits that go in in the first T re-randomizations.
- We need to count the number of bits of print statements that come out in the first T re-randomizations.
- We need to argue that we can recover the random bits that go in from the print statements that come out.

Whoops! This doesn't hold!!



Recovering the random bits

example

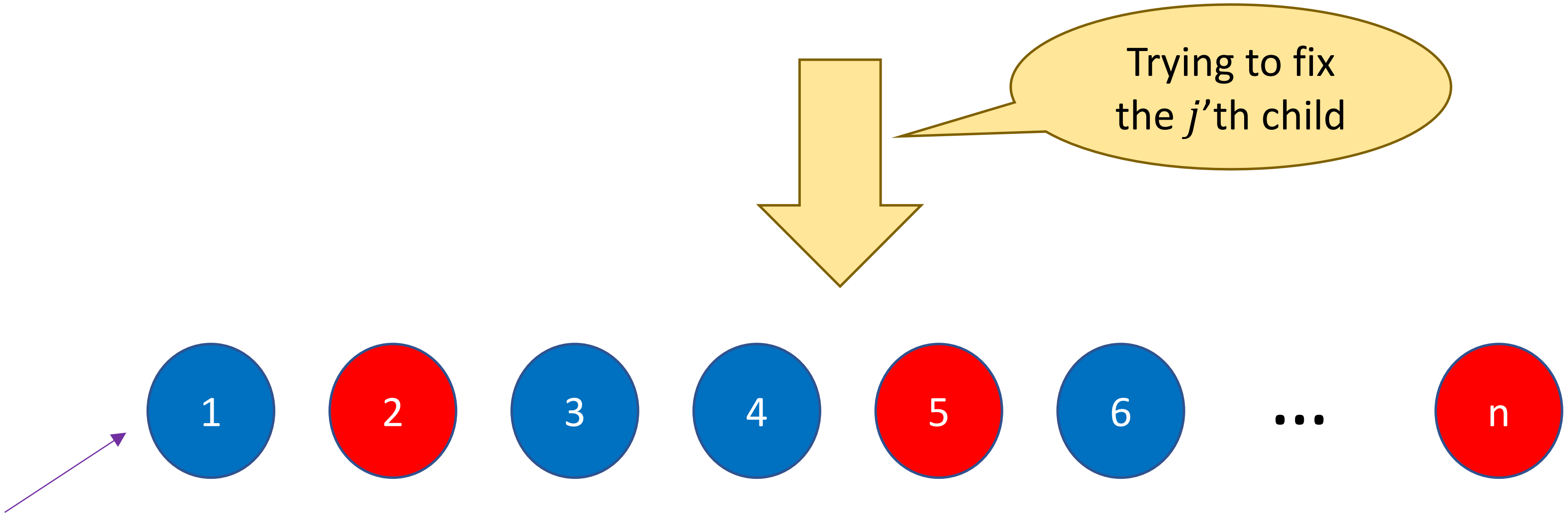
- The print statements allow us to reconstruct the recursion tree.
- Then…

Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Recovering the random bits

example

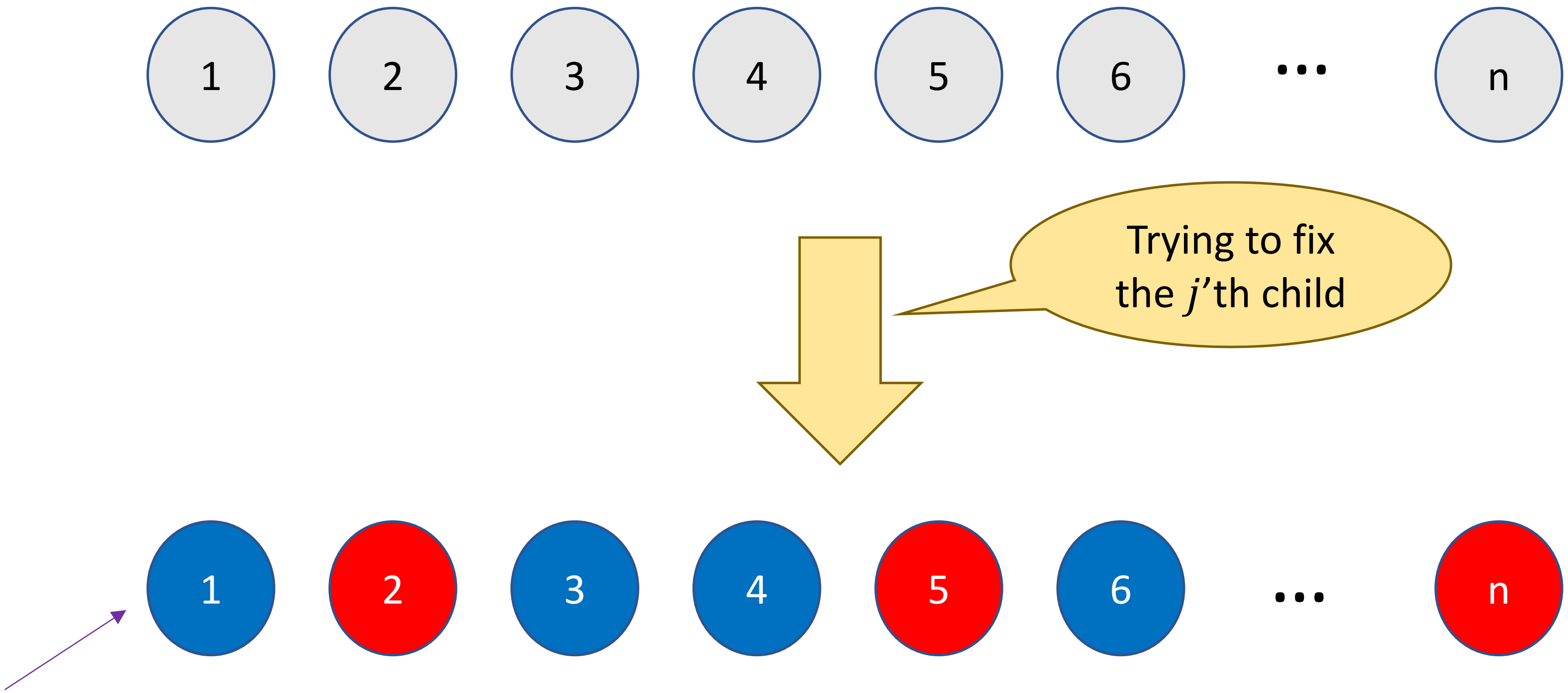
- The print statements allow us to reconstruct the recursion tree.
- Then…



Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Recovering the random bits
example

- The print statements allow us to reconstruct the recursion tree.
- Then…

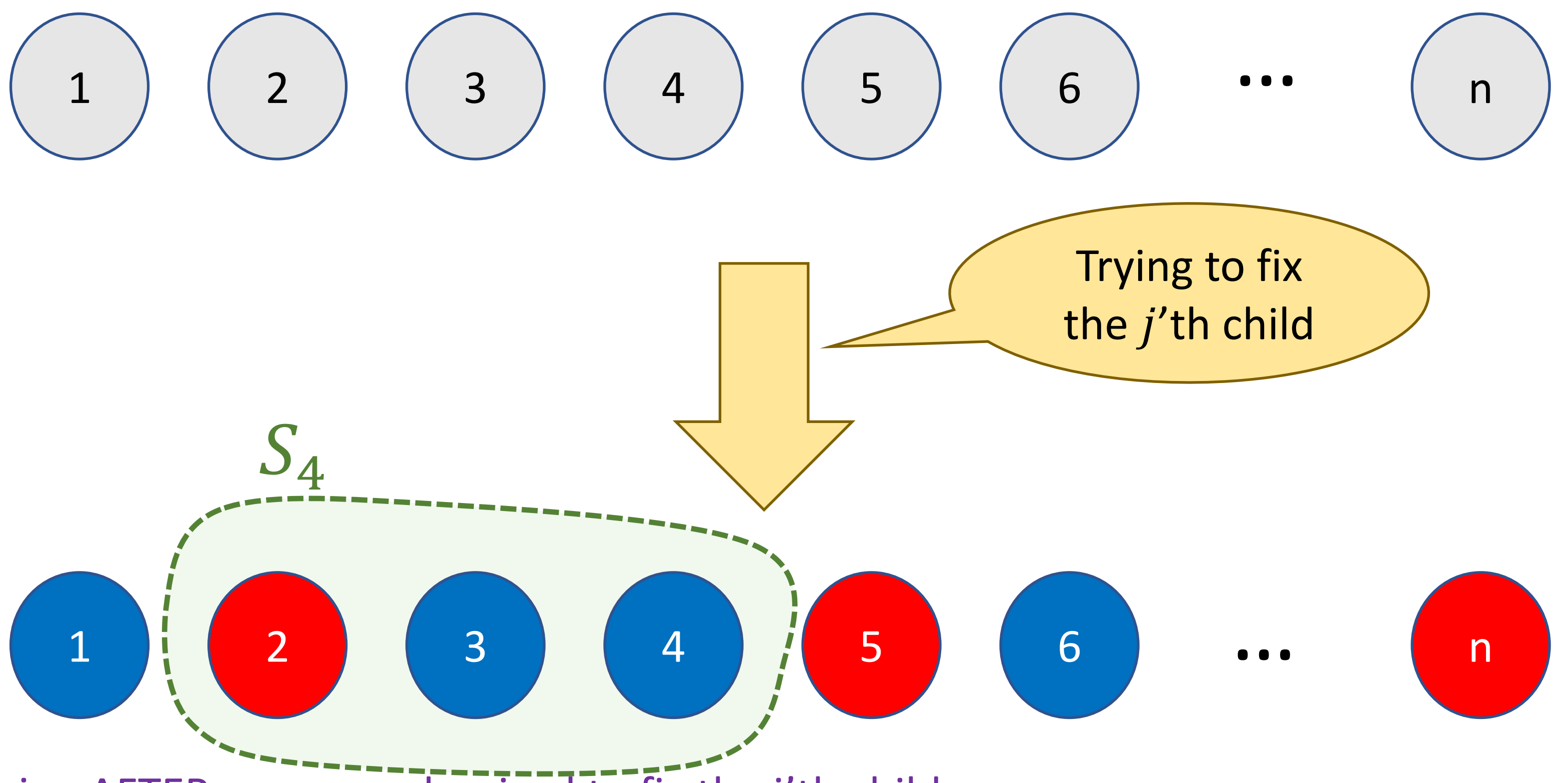


Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Recovering the random bits

example

- The print statements allow us to reconstruct the recursion tree.
- Then…



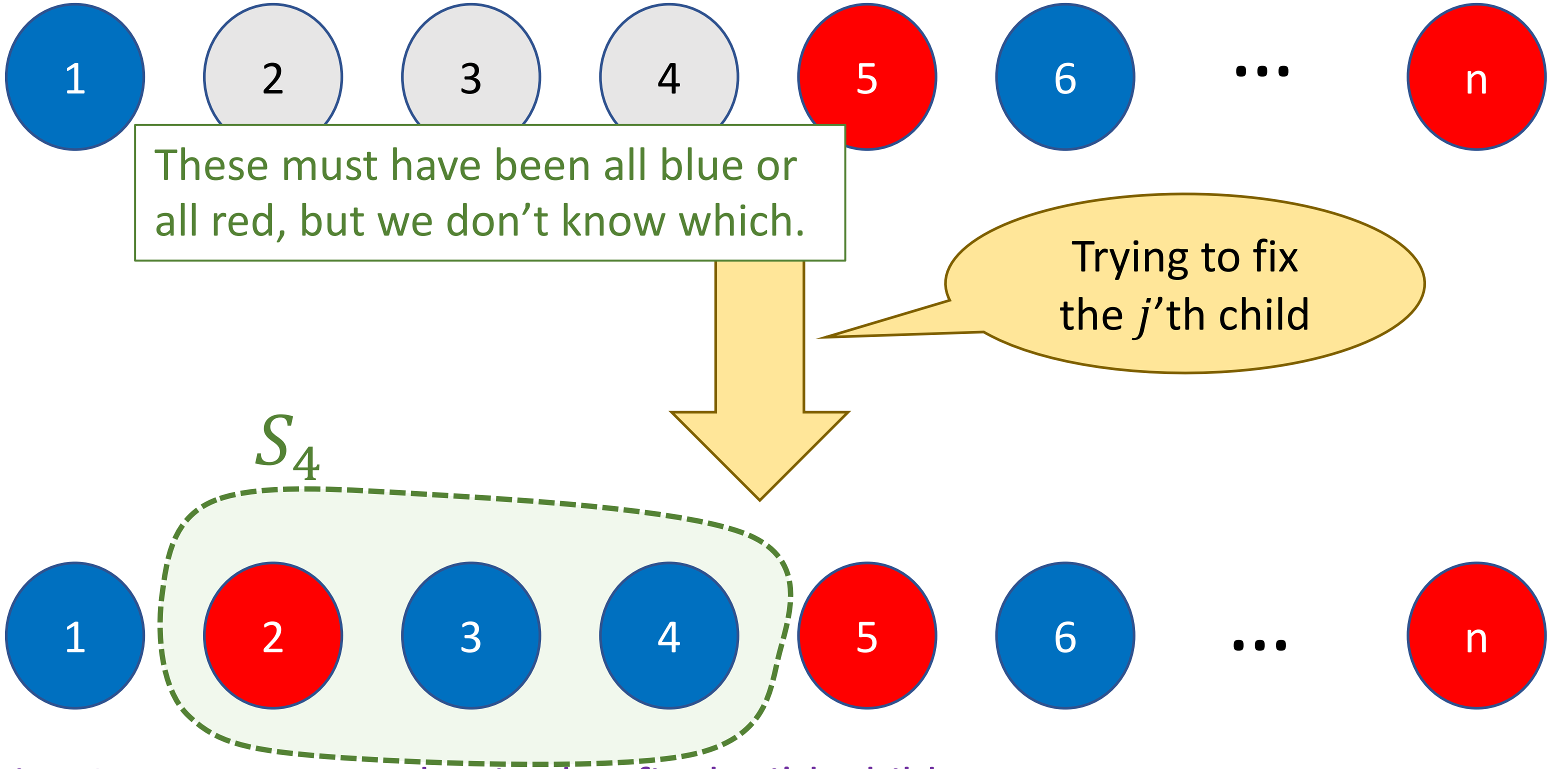
Since I know the recursion tree, I know that at this point "the j'th child" means S_4

Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Recovering the random bits

example

- The print statements allow us to reconstruct the recursion tree.
- Then…



Since I know the recursion tree, I know that at this point "the j'th child" means S_4

Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Our algorithm

- **FindSat**($S_1, S_2, \ldots, S_m$):
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \ldots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \ldots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$ **Fix**(i_j, σ)
  - Return σ

All done with this level.

Trying to fix the j'th child

…because it was all red! (or blue, as appropriate)

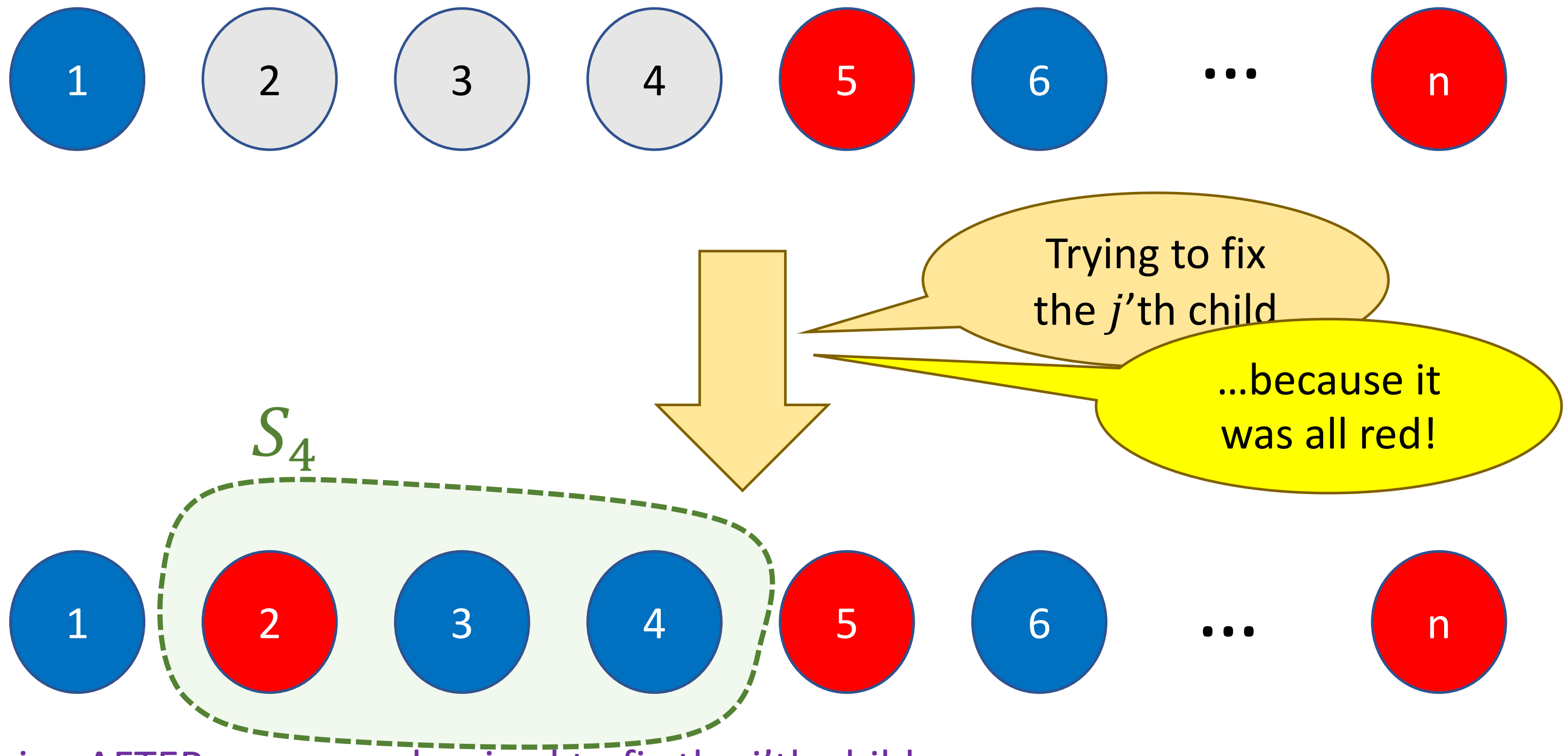
After T re-randomizations,

I give up. I've got σ

Recovering the random bits

example

- The print statements allow us to reconstruct the recursion tree.
- Then…



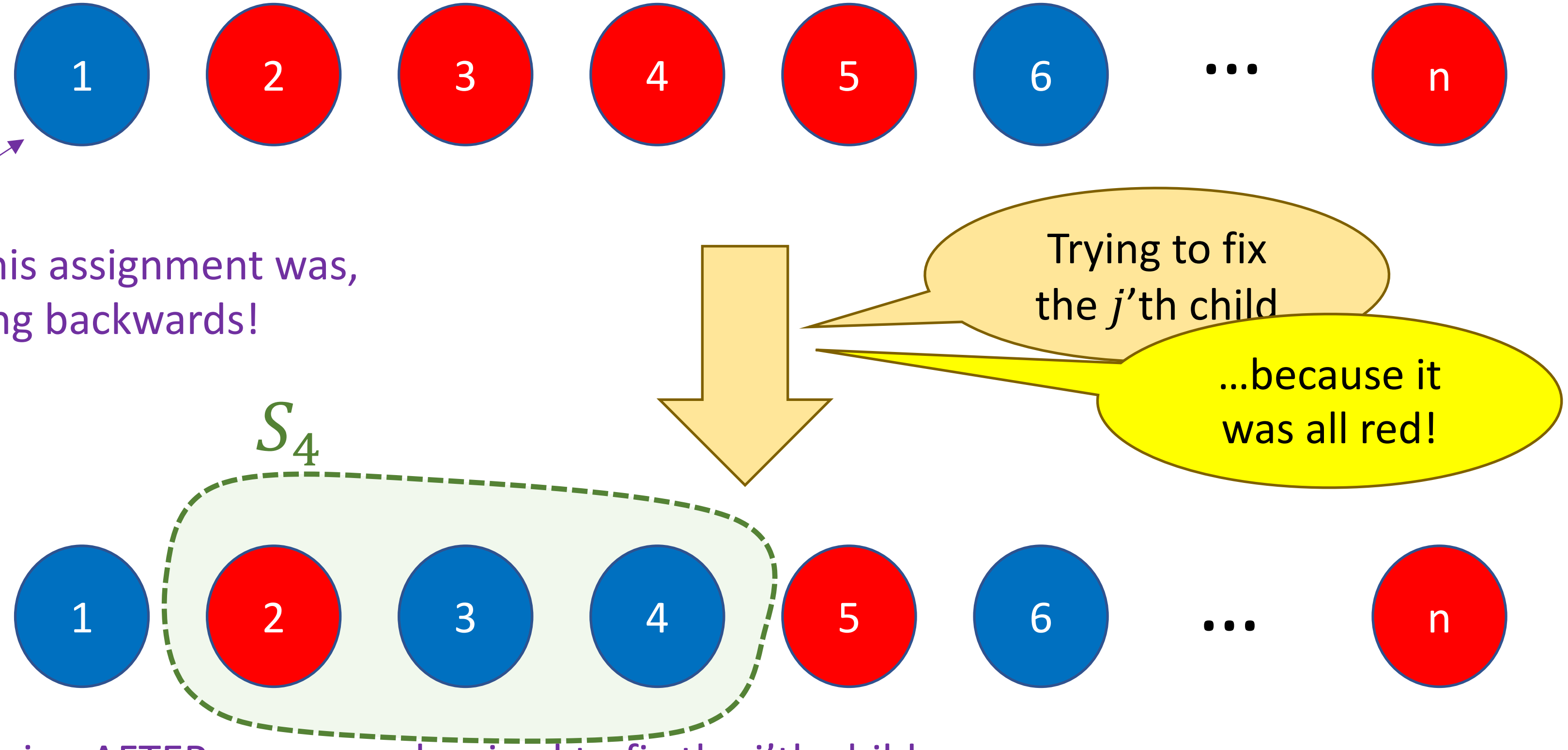
Since I know the recursion tree, I know that at this point "the j'th child" means S_4

Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Recovering the random bits

example

- The print statements allow us to reconstruct the recursion tree.
- Then…



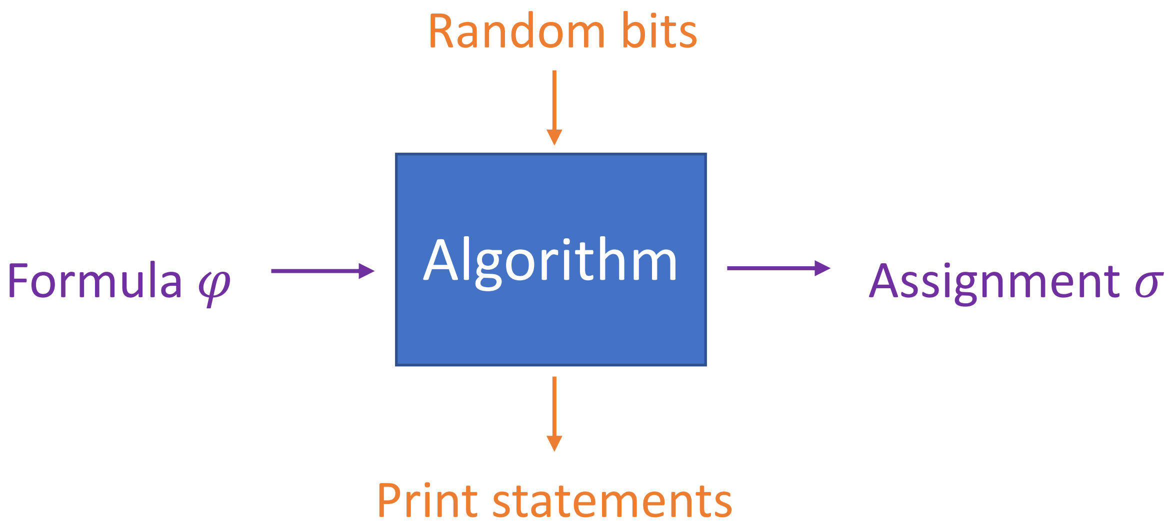
Now we know what this assignment was, so we can keep working backwards!

Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

Since I know the recursion tree, I know that at this point "the j'th child" means S_4

To do the proof

- We need to count the number of random bits that go in in the first T re-randomizations.
- We need to count the number of bits of print statements that come out in the first T re-randomizations.
- We need to argue that we can recover the random bits that go in from the print statements that come out.



Our algorithm

- **FindSat**($S_1, S_2, \ldots, S_m$):
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

Random bits in:

$n + k \cdot T$

original σ

k bits per re-randomization

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \ldots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \ldots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$**Fix**(i_j, σ)
  - Return σ

Trying to fix the j'th child

...because it was all red! (or blue, as appropriate)

All done with this level.

After T re-randomizations,

I give up. I've got σ

Our algorithm

- **FindSat**$(S_1, S_2, \dots, S_m)$:
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \dots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \dots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$ **Fix**(i_j, σ)
  - Return σ

All done with this level.

Trying to fix the j'th child

…because it was all red! (or blue, as appropriate)

Bits out:

$$\le \underbrace{m[\log(m)+C]}_{\text{"Fixing clause } i\text{"}}$$

$$+ \; T\underbrace{[\log(d+1)+1+C]}_{\text{"trying to fix } j^{th} \text{ child because it was red"}}$$

call Fix T times

Also "all done"

$$+ \underbrace{n+C}_{\text{"I give up. } \sigma\text{."}}$$

After T re-randomizations,

I give up. I've got σ

Win if random bits in $\gg$ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Random bits in: $n + k \cdot T$

original σ

k bits per re-randomization

Bits out: $\le \underbrace{m[\log(m)+C]}_{\text{"Fixing clause } i\text{"}} + T\underbrace{[\log(d+1)+1+C]}_{\text{"trying to fix } j^{th} \text{ child because it was red"}} + \underbrace{n+C}_{\text{"I give up. } \sigma\text{."}}$

call Fix T times

Win if random bits in ≫ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Random bits in: $$\underbrace{n}_{\text{original } \sigma} + \underbrace{k}_{k \text{ bits per re-randomization}} \cdot T$$

Bits out: $$\leq \underbrace{m[\log(m)+C]}_{\text{"Fixing clause } i\text{"}} + \underbrace{T}_{\text{call Fix } T \text{ times}}\underbrace{[\log(d+1)+1+C]}_{\text{"trying to fix } j^{th} \text{ child because it was red"}} + \underbrace{n+C}_{\text{"I give up. } \sigma.\text{"}}$$

Want: $$n + kT \gg m(\log m + C) + T(\log(d+1) + 1 + C) + n + C$$

Win if random bits in $\gg$ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Want: $n + kT \gg m(\log m + C) + T(\log(d+1) + 1 + C) + n + C$

Win if random bits in ≫ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Want: $$n + kT \gg m(\log m + C) + T(\log(d+1) + 1 + C) + n + C$$

Aka: $$m(\log m + C) \ll T(k - \log(d+1) + 1 + C)$$

Win if random bits in $\gg$ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Want: $n + kT \gg m(\log m + C) + T(\log(d+1) + 1 + C) + n + C$

Aka: $m(\log m + C) \ll T(k - \log(d+1) + 1 + C)$

Provided that $k \geq \log(d+1) + 100000$, this happens for $T = \text{poly}(m)$.

What happens if there are $t > 2$ colors?

Our algorithm

- **FindSat**($S_1, S_2, \ldots, S_m$):
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \ldots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \ldots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$**Fix**$(i_j\ , \sigma)$
  - Return σ

All done with this level.

Trying to fix the j'th child

…because it was all red! (or blue, **or purple, or….** as appropriate)

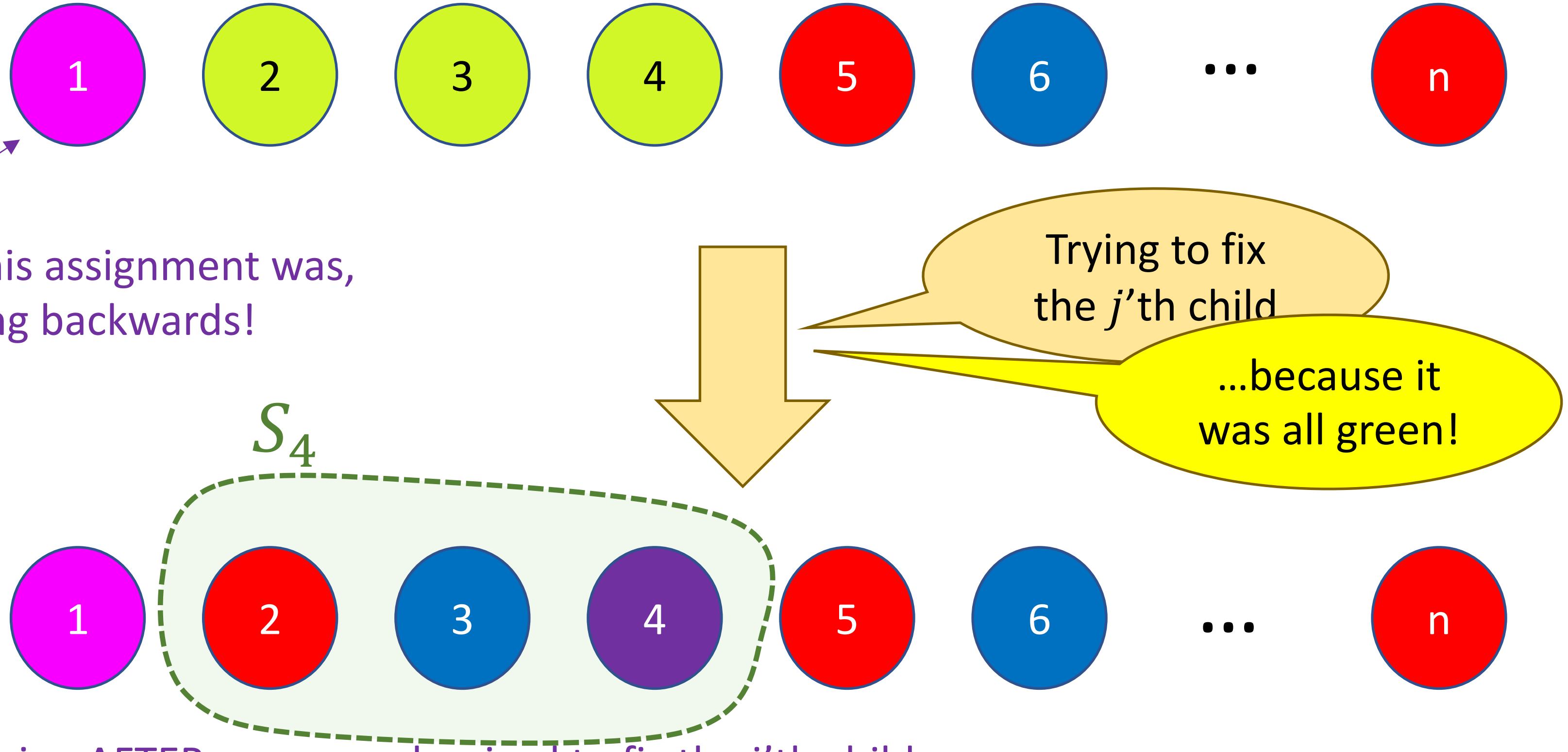
After T re ndomizations,

I give up. I've got σ

Recovering the random bits

example

- The print statements allow us to reconstruct the recursion tree.
- Then…



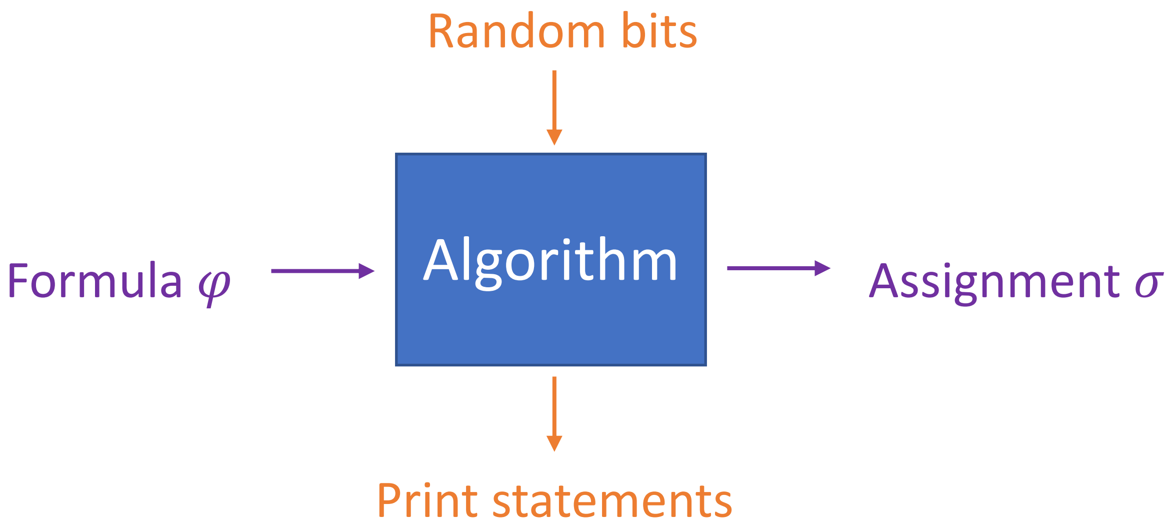
Now we know what this assignment was, so we can keep working backwards!

Since I know the recursion tree, I know that at this point "the j'th child" means S_4

Say we know the coloring AFTER we re-randomized to fix the j'th child.
(We know the final assignment since it was printed out, and we're working backwards.)

To do the proof

- We need to count the number of random bits that go in in the first T re-randomizations.
- We need to count the number of bits of print statements that come out in the first T re-randomizations.
- We need to argue that we can recover the random bits that go in from the print statements that come out.



Our algorithm

- **FindSat**$(S_1, S_2, \ldots, S_m)$:
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

Random bits in:

$n + k \cdot T \cdot \log(t)$

original σ

$k \cdot \log(t)$ bits per re-randomization.

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \ldots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \ldots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$ **Fix**(i_j, σ)
  - Return σ

All done with this level.

Trying to fix the j'th child

...because it was all red! (or blue, **or purple, or….** as appropriate)

After T re-randomizations,

I give up. I've got σ

Our algorithm

Bits out: $\leq \underbrace{m[\log(m)+C]}_{\text{"Fixing clause } i\text{"}}$

$+ \; T\underbrace{[\log(d+1) + \cancel{1}\,\log(t) + C]}_{\text{"trying to fix } j^{\text{th}} \text{ child because it was red"}}$

call Fix T times

$+ \; \underbrace{n + C}_{\text{"I give up. }\sigma\text{."}}$

Also "all done"

- **FindSat**$(S_1, S_2, \ldots, S_m)$:
  - Choose a random coloring σ for each of numbers
  - For each S_i that is monochromatic:
    - $\sigma \leftarrow$ **Fix**(i, σ)
  - Return σ

Fixing set i!

- **Fix**(i, σ):
  - Update σ by re-randomizing every number in S_i
  - Let $S_{i_1}, S_{i_2}, \ldots S_{i_{d+1}}$ be the sets that intersect S_i
  - For $j = 1, \ldots, d+1$:
    - If S_{i_j} is monochromatic:
      - $\sigma \leftarrow$ **Fix**(i_j, σ)
  - Return σ

All done with this level.

Trying to fix the j'th child

...because it was all red! (or blue, **or purple, or….** as appropriate)

After T re-randomizations,

I give up. I've got σ

Win if random bits in ≫ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Random bits in: $\underbrace{n}_{\text{original } \sigma} + \underbrace{k}_{k \text{ bits per re-randomization}} \cdot T_{\log(t)}$

Bits out: $\le \underbrace{m[\log(m)+C]}_{\text{"Fixing clause } i\text{"}} + \underbrace{T}_{\text{call Fix } T \text{ times}}\underbrace{[\log(d+1) + \cancel{1}^{\log(t)} + C]}_{\text{"trying to fix } j^{th} \text{ child because it was red"}} + \underbrace{n + C}_{\text{"I give up. } \sigma\text{."}}$

Win if random bits in $\gg$ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Random bits in: $\underbrace{n}_{\text{original } \sigma} + \underbrace{k}_{k \text{ bits per re-randomization}} \cdot T \log(t)$

Bits out: $\leq \underbrace{m[\log(m) + C]}_{\text{"Fixing clause } i\text{"}} + \underbrace{T}_{\text{call Fix } T \text{ times}} \underbrace{[\log(d+1) + \cancel{1}^{\log(t)} + C]}_{\text{"trying to fix } j^{th} \text{ child because it was red"}} + \underbrace{n + C}_{\text{"I give up. } \sigma\text{."}}$

Want: $n + kT\log(t) \gg m(\log m + C) + T(\log(d+1) + \cancel{1}^{\log(t)} + C) + n + C$

Win if random bits in ≫ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Want: $$n + kT\log(t) \gg m(\log m + C) + T(\log(d+1) + \overset{\log(t)}{\cancel{1}} + C) + n + C$$

Win if random bits in ≫ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Want: $$n + k\overset{\log(t)}{T} \gg m(\log m + C) + T(\log(d+1) + \overset{\log(t)}{\cancel{1}} + C) + n + C$$

Aka: $$m(\log m + C) \ll T(k^{\log(t)} - \log(d+1) + \overset{\log(t)}{\cancel{1}} + C)$$

Win if random bits in $\gg$ bits out

aka, then we'd get a contradiction and conclude that there must be $< T$ re-randomizations.

Want: $$n + k\,T\,\log(t) \gg m(\log m + C) + T\left(\log(d+1) + \overset{\log(t)}{\cancel{1}} + C\right) + n + C$$

Aka: $$m(\log m + C) \ll T\left(k\,\log(t) - \log(d+1) + \overset{\log(t)}{\cancel{1}} + C\right)$$

Provided that $$k \geq \frac{\log(d+1) + \log(t)}{\log(t)} + 9999 = \frac{\log(d+1)}{\log(t)} + 10000$$

this happens for $T = \mathrm{poly}(m)$

Conclusion

As long as $k \geq \frac{\log(d+1)}{\log(t)} + 10000$, we can find a good coloring with $\text{poly}(m)$ re-randomizations!

How does this compare to the general constructive LLL in the lecture notes?

Corollary 3. *Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \leq d+1$, and $\Pr[A] \leq \frac{1}{e(d+1)}$, then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.*

How does this compare to the general constructive LLL in the lecture notes?

Corollary 3. *Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \leq d+1$, and $\Pr[A] \leq \frac{1}{e(d+1)}$, then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.*

$A_i = \{ S_i \text{ is monochromatic} \}$

$\mathbb{P}\{A_i\} = \frac{t}{t^k}$ for t colors.

Same thing!

How does this compare to the general constructive LLL in the lecture notes?

Corollary 3. *Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \leq d+1$, and $\Pr[A] \leq \frac{1}{e(d+1)}$, then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.*

$A_i = \{ S_i \text{ is monochromatic} \}$

$\mathbb{P}\{A_i\} = \frac{t}{t^k}$ for t colors.

Need: $\mathbb{P}\{A_i\} \leq \frac{1}{e(d+1)}$

How does this compare to the general constructive LLL in the lecture notes?

Corollary 3. *Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \le d+1$, and $\Pr[A] \le \frac{1}{e(d+1)}$, then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.*

$$A_i = \{ S_i \text{ is monochromatic} \}$$

$$\mathbb{P}\{A_i\} = \frac{t}{t^k} \quad \text{for } t \text{ colors.}$$

Need: $\mathbb{P}\{A_i\} \le \dfrac{1}{e(d+1)}$

$$t^{-(k-1)} \le \frac{1}{e(d+1)}$$

$$(k-1)\log(t) \ge 1 + \log(d+1)$$

How does this compare to the general constructive LLL in the lecture notes?

Corollary 3. *Let V be a finite set of independent random variables. Let $\mathcal{A}$ be a finite set of events determined by the random variables in V. If for all $A \in \mathcal{A}$, $|\Gamma(A)| \leq d+1$, and $\Pr[A] \leq \frac{1}{e(d+1)}$, then Algorithm 2 will find an assignment to the variables V such that no event of $\mathcal{A}$ occurs. Additionally, the expected number of "re-randomizations" performed by the algorithm is bounded by $O(|\mathcal{A}|/(d+1))$.*

$$A_i = \{ S_i \text{ is monochromatic} \}$$

$$\mathbb{P}\{A_i\} = \frac{t}{t^k} \quad \text{for } t \text{ colors.}$$

Need: $\mathbb{P}\{A_i\} \leq \frac{1}{e(d+1)}$

$$t^{-(k-1)} \leq \frac{1}{e(d+1)}$$

$$(k-1)\log(t) \geq 1 + \log(d+1)$$

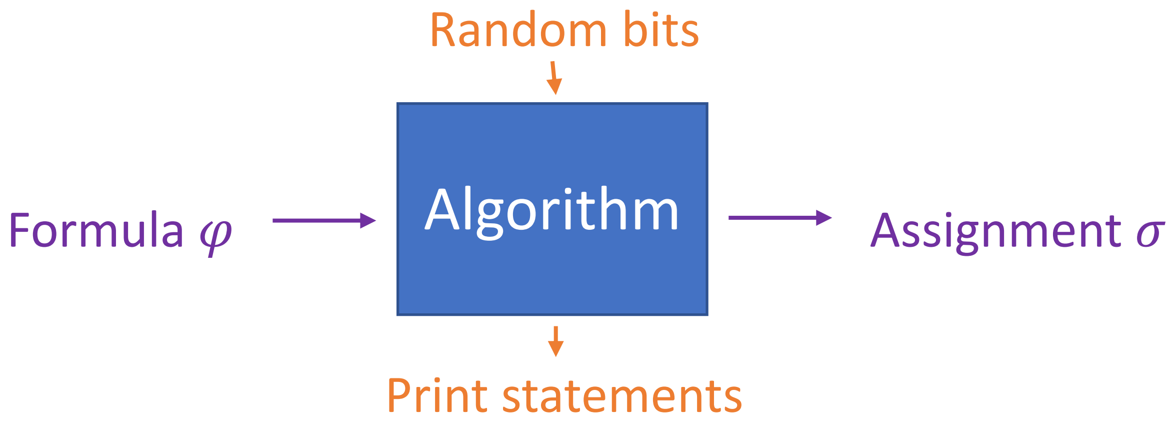
$$k \geq \frac{\log(d+1)}{\log(t)} + [\text{constant}]$$

Same thing!

Conclusions

- As long as $k \geq \frac{\log(d+1)}{\log(t)} + 10000$, we can find a good coloring with $\text{poly}(m)$ re-randomizations!

- You now have some idea of how you might adapt this proof to deal with other examples (aka, ones with $\Pr[A_i] \leq p$ for a general p)....

- 

is a very cute idea.

(This method of proof is called "entropy compression")