CS624 - Analysis of Algorithms

Cook-Levin Theorem

December 8, 2025

SAT is NP-complete

Definition

A (1-tape) Turing machine consists of

- A finite set Q of *states*. Three of these states are special, and have special names:
  - There is an *initial state*, which we will denote by q_0.
  - There is an *accepting state*.
  - There is a *rejecting state*.
- A finite set $T = \{\tau_0, \tau_1, \tau_2, \dots\}$ of *tape symbols*. There is a subset $I \subseteq T$ of *input symbols*. And there is a special *blank symbol* in $T \setminus I$.
- A *move* or *transition* function

$$\delta : Q \times T \to Q \times T \times \{L, R\}$$

Turing Machine

- The tape can be extended infinitely in both directions. At any stage of the computation:
- The machine is at one state of the machine, q.
- The tape has a finite sequence of tape symbols in its cells.
- The machine points to one of the cells.
- These three items constitute a configuration or an instantaneous description (ID) of the machine.

Turing Machine

- A configuration is represented as uq_iv where u and v are (possibly empty) strings of tape symbols, q_i is a state, and the convention is that
  - The cells of the tape hold the string uv (i.e., u concatenated with v).
  - The machine is pointing to the cell containing the first symbol in v if v is non-empty. If v is empty, the machine is pointing to the "next cell" after u, which contains the blank symbol.
- The *initial configuration*, or *initial ID* of the machine is q_0w where w is some string formed from input symbols (i.e., elements of I).

Turing Machine

- The machine proceeds as follows: If an ID is given by uaq_ibv, where a and b are tape symbols, then $\delta(q_i, b)$ is computed (or, really, looked up, since $Q \times T$ is a finite set).
- It is of the form $\langle q_j, \tau, L\rangle$ or $\langle q_j, \tau, R\rangle$. L means "move to the left" and R means "move to the right". We consider these two cases separately:
  - $\delta(q_i, b) = \langle q_j, \tau, R\rangle$. The machine changes b to τ, enters state q_j, and moves one cell to the right. So the new state is $ua\tau q_j v$.
  - $\delta(q_i, b) = \langle q_j, \tau, L\rangle$. The machine changes b to τ, enters state q_j, and moves one cell to the left. So the new state is $uq_ja\tau v$.

Turing Machine

- This process is then repeated. One of three things can happen:
  - Eventually the machine winds up in the accepting state. The machine then stops, and we say that it has accepted the initial input string w.
  - Eventually the machine winds up in the rejecting state. The machine then stops, and we say that it has rejected the initial input string w.
  - The machine goes on forever without ever entering either the accept or reject states.

A Language

Definition

A *language* (more properly, a *formal language*) over the set T is simply a set of finite strings over T.

- Example: UNDIRECTED HAMILTONIAN CYCLE. Let T be any set of symbols that we can use to write a description of an undirected graph in.
- We can define UNDIRECTED HAMILTONIAN CYCLE to be the language consisting of those (finite) strings over T that correspond to undirected graphs having a Hamiltonian cycle.

A Language

- Given a set T and a language L over T, find an algorithm that takes as input a string over T and tells whether or not it is in L.
- We can replace "algorithm" with "Turing machine".
- If such a machine exists, we say that L is *Turing decidable* or simply *recursive*.
- Since all the problems we have been considering can be solved by exhaustive search, it is pretty evident that they are all decidable. Our problem is efficiency.

Definition

A language L over a set T is in the class P iff there is a Turing machine that decides L in polynomial time w.r.t the length of the input string.

Non-deterministic Turing machines and NP

- Ordinary Turing machines, such as we have described above are *deterministic*.
- Every configuration determines the next one, so starting from an initial configuration, the entire sequence of configurations of the machine is completely determined, and there is no choice involved.
- Let us consider a *non-deterministic* machine.

Definition

In a *non-deterministic Turing machine* the transition function δ is no longer a function, as it returns a set of possible choices.
$\delta : Q \times T \rightarrow P\big(Q \times T \times \{L, R\}\big)$ where for any set X, $P(X)$ is the *power set* of X – the set of all subsets of X. It is also often written as 2^X.

Non-deterministic Turing machines and NP

- At each step every choice spawns a set of sub-processes.
- We have a tree of computations rather than just one linear sequence.
- If any path leads to an accepting state then the initial string is accepted by the machine.
- This is definitely not a very practical model...

Non-deterministic Turing machines and NP

Definition

The class NP is the class of languages that are decided in polynomial time by non-deterministic Turing machines.

- A deterministic Turing machine is a special case of a non-deterministic Turing machine.
- So with this definition, we know a priori that $P \subseteq NP$.
- It is not obvious that this definition is equivalent to the one we have been using up to now, in terms of polynomial-time verifiability.
- It is not hard to show that these two definitions are equivalent.
- It should be clear that the problems that we have considered so far are in NP in this new definition.

The Cook-Levin Theorem

Theorem (Cook-Levin)

SAT is NP-complete.

Proof.

- **SAT is in NP:** A non-deterministic Turing machine can decide SAT in polynomial time, because each path down the tree can correspond to checking satisfiability for a different choice of truth values for the variables, and this can be verified in a linear time.
- **SAT is NP-hard:** For each language L in NP over the set of symbols T, and each finite string w over T, we need to create a SAT expression such that
  - The algorithm taking the string w to the SAT expression is polynomial.
  - w is in L iff the SAT expression is satisfiable.

The Cook-Levin Theorem

- We have to also encode the language L in the SAT expression.
- We only know that L is in NP.
- We start with a ND Turing machine M that decides L and the input string w.
- We will create a SAT instance from M and w in polynomial time s.t. the instance is satisfiable iff M accepts w.
- In other words – we will show that every problem in NP is polynomially reducible to SAT.

The Cook-Levin Theorem

- We know M decides every string in w in a polynomial time.
- Assume every part of the computation in of length $(p|w|)$ where p is some polynomial.
- We can represent the possible states of the machine at every stage of the computation in an array indexed from $-(p|w|)$ to $(p|w|)$.
- Initially, M points to index 0.

The Cook-Levin Theorem

- Let us define the following:
- α is the index among the states of the accepting state. So q_α is the accepting state.
- β is the index among the tape symbols of the blank symbol. So τ_β is the blank symbol.
- The tape symbols in the initial string w are $w_0, w_1, \ldots, w_{|w|-1}$.

The Cook-Levin Theorem

- To construct our SAT expression, we need to specify the variables in that expression.
- Each of these variables will say something about the state of our machine M at some point in the computation.
- To make things as simple as possible, we name our variables using three capital letters:
  - C stands for *cell*.
  - S stands for *state*.
  - H stands for *head* (As in a tape recorder head).

The Cook-Levin Theorem

- Here are the variables:
- $C_{i,j,t}$ is a variable whose value is True iff the i^{th} cell on the tape contains the tape symbol τ_j at time t.
- This is a whole family of variables: there are $2p(|w|)+1$ possible values for i, $|T|$ possible values for j, and $p(|w|)$ possible values for t, so we just defined $\big(2p(|w|)+1\big)|T|p(|w|)$ variables.
- The particular value of this is unimportant, only that the number of these variables is bounded by a fixed polynomial in $|w|$.
- $S_{k,t}$ is a variable whose value is True iff M is in state q_k at time t. Here we have $|Q|p(|w|)$ variables.
- $H_{i,t}$ is a variable whose value is True iff at time t the machine is pointing at cell i. Here we have $\big(2p(|w|)+1\big)p(|w|)$ variables.

The SAT Expression

- We use an auxiliary function – $U(x_1, x_2, \ldots, x_r)$ will be True iff exactly one of its arguments is True (and the rest are False).
- To construct an expression for U, note that

$$x_1 \vee x_2 \vee \cdots \vee x_n \quad \text{is True} \iff \text{at least one variable is True}$$

$$\bigwedge_{1 \le i < j \le n} (\bar{x}_i \vee \bar{x}_j) \quad \text{is True} \iff \text{no two of the variables are True}$$

- Therefore, we can write

$$U(x_1, x_2, \ldots, x_n) = (x_1 \vee x_2 \vee \cdots \vee x_n) \wedge \bigwedge_{1 \le i < j \le n} (\bar{x}_i \vee \bar{x}_j)$$

- Note that this expression is in CNF.

The SAT Expression

- We are going to construct a SAT expression as follows: We will create expressions A, B, C, and so on to encode the following 7 statements:
  - A: The machine is pointing to exactly one cell at each time. (We will call this the "current cell".)
  - B: Each cell contains exactly one tape symbol.
  - C: The machine is in exactly one state at each time.
  - D: Exactly one tape cell (the current cell) is modified from one time to the next.
  - E: The first configuration is the initial configuration.
  - F: The state in the last configuration is the accepting state.
  - G: The change in state, current cell, and cell contents between successive times is allowed by the transition function of M.
- Each of these expressions will be in CNF.
- That expression will be True iff the machine M accepts w.

The Expressions

A. The machine is pointing to exactly one cell at each time. Let A_t assert that at time t the machine is pointing to exactly one cell. Then $A = A_0 \wedge A_1 \wedge \cdots \wedge A_{p(|w|)}$ and we have

$$A_t = U(H_{-p(|w|),t}, \ldots, H_{-1,t}, H_{0,t}, H_{1,t}, \ldots, H_{p(|w|),t})$$

Each A_t is in CNF and so is A.

B. This asserts that each cell contains exactly one tape symbol Let $B_{i,t}$ assert that the i^{th} cell contains exactly one symbol at time t. Then $B = \bigwedge_{i,t} B_{i,t}$, and $B_{i,t} = U(C_{i,1,t}, C_{i,2,t}, \ldots, C_{i,|T|,t})$. Again, B is in CNF.

C. This asserts that the machine is in exactly one state at each time. We have $C = \bigwedge_{0 \leq t \leq p(|w|)} U(S_{1,t}, S_{2,t}, \ldots, S_{|Q|,t})$. C is in CNF.

The Expressions

D. This asserts that exactly one tape cell (the current cell) is modified from one time to the next.

$$D = \bigwedge_{\substack{0 \le t < p(|w|) \\ -p(|w|) \le i \le p(|w|) \\ 1 \le j \le |T|}} (\overline{C_{i,j,t}} \vee H_{i,t} \vee C_{i,j,t+1})$$

This can be read as follows:

- Either the i^{th} cell does not contain the symbol τ_j at time t,
- or it does, in which case either
  - it also contains it at time $t+1$, or
  - the machine is pointing at the i^{th} cell at time t.

D is in CNF

The Expressions

E. This asserts that the first configuration is the initial configuration.

$$E = S_{0,0} \wedge H_{0,0} \wedge \bigwedge_{i=-p(|w|)}^{-1} C_{i,\beta,0} \wedge \bigwedge_{i=0}^{|w|-1} C_{i,w_i,0} \wedge \bigwedge_{i=|w|}^{p(n)} C_{i,\beta,0}$$

E is in CNF.

F. This asserts that the state in the last configuration is the accepting state q_α.

$$F = S_{\alpha,p(|w|)}$$

F is in CNF.

The Expressions

G. This asserts that the change in state, current cell, and cell contents between successive times is allowed by the transition function δ of M.

- This expression is more complex than the previous ones.
- Let us pretend to start out with that our Turing machine is actually deterministic, so for each state q and tape symbol τ, there is a unique state q', tape symbol τ' and direction d' (i.e., either L or R) such that $\delta(q, \tau)$ is $\langle q', \tau', d \rangle$.
- In this case G will be the conjunction (i.e., the "and") of a number of clause groups. Each group is composed of three clauses, like this:

$$(\overline{H_{i,t}} \vee \overline{S_{k,t}} \vee \overline{C_{i,j,t}} \vee H_{i',t+1}) \wedge$$
$$(\overline{H_{i,t}} \vee \overline{S_{k,t}} \vee \overline{C_{i,j,t}} \vee S_{k',t+1}) \wedge$$
$$(\overline{H_{i,t}} \vee \overline{S_{k,t}} \vee \overline{C_{i,j,t}} \vee C_{i,j',t+1})$$

The Expressions

- There is a clause group for each t from 0 to $p(|w|) - 1$ and for each set of values (i, j, k, i', j', k') such that $-p(|w|) \leq i \leq p(|w|)$ and such that when we compute $\delta(q_k, \tau_j)$, either
  - $\delta(q_k, \tau_j) = \langle q_{k'}, \tau_{j'}, R \rangle$, in which case we set $i' = i + 1$, or
  - $\delta(q_k, \tau_j) = \langle q_{k'}, \tau_{j'}, L \rangle$, in which case we set $i' = i - 1$.

  The way to understand these clauses is as follows: the only way $\overline{H_{i,t}} \vee \overline{S_{k,t}} \vee \overline{C_{i,j,t}}$ can be false is if at time t,
  - the machine is pointing at cell i,
  - the machine is in state k,
  - and the cell i contains τ_j.

The Expressions

- In that case
  - The first clause says that at time $t+1$, the machine is pointing to cell i' (which is either cell $i+1$ or cell $i-1$).
  - The second clause says that at time $t+1$ the machine is in state k'.
  - And the third clause says that at time $t+1$ cell i contains $\tau_{j'}$.
- Clearly G is in CNF. However, in general, the machine is non-deterministic. This means that for each state q and tape symbol τ, $\delta(q,\tau)$ is a (finite!) set of triples of the form

$$\langle q', \tau', d' \rangle$$
$$\langle q'', \tau'', d'' \rangle$$
$$\langle q''', \tau''', d''' \rangle$$
$$\ldots$$

The Expressions

- Suppose that each of these possibilites gives rise to an expression G', G'', G''', and so on.
- Our real expression G is just $G = G' \vee G'' \vee G''' \vee \ldots$
- The only problem is that G is no longer in CNF.
- Without going into details, it is clear that just by using the distributive property over and over again, this expression can be rewritten in CNF.
- The only problem is that the size of the rewritten expression may be exponentially larger than the size of the original expression.

The Expressions

- For our purposes here it doesn't matter.
- The reason is that the size of each original expression $G' \vee G'' \vee \ldots$ does not depend on n but on the properties of the Turing machine itself.
- Therefore it doesn't really matter how big G is, since the size of G does not vary with n either.
- The *number* of such expressions G that we need depends on n – there are $p(|w|)\big(2p(|w|)+1\big)$ such expressions – and this is polynomial in $|w|$, which will be multiplied by the constant size of G, so it is still a polynomial function of $|w|$.

Concluding Remarks

- Without going into many details, it is easy to see that everything here in this construction is polynomially bounded.
- Therefore the expression

$$A \wedge B \wedge C \wedge D \wedge E \wedge F \wedge G$$

  is in CNF which is derivable in a polynomially bounded way from M and w, and which is satisfiable iff M accepts w.
- And that completes the proof.