Two Cardinal Directions

1. Breadth-First Search: Time over Space.
2. Depth-First Search: Space over Time.
Two Cardinal Directions

1. Breadth-First Search: Time over Space.
2. Depth-First Search: Space over Time.

- Models of computation are commonly introduced as “machines” or “grammars” but we will emphasize graphs.
- Graph nodes are snapshots $I, J, K, \ldots$ of the memory map.
Two Cardinal Directions

1. Breadth-First Search: Time over Space.
2. Depth-First Search: Space over Time.

- Models of computation are commonly introduced as “machines” or “grammars” but we will emphasize graphs.
- Graph nodes are snapshots $I, J, K, \ldots$ of the memory map.
- Called configurations or instantaneous descriptions (IDs).
Two Cardinal Directions

1. Breadth-First Search: Time over Space.
2. Depth-First Search: Space over Time.

- Models of computation are commonly introduced as “machines” or “grammars” but we will emphasize graphs.
- Graph nodes are snapshots $I, J, K, \ldots$ of the memory map.
- Called configurations or instantaneous descriptions (IDs).
- $I \rightarrow J$ means “$I$ can go to $J$ in one step.” Directed edge.
Two Cardinal Directions

1. Breadth-First Search: Time over Space.
2. Depth-First Search: Space over Time.

- Models of computation are commonly introduced as “machines” or “grammars” but we will emphasize *graphs*.
- Graph nodes are snapshots $I, J, K, \ldots$ of the *memory map*.
- Called *configurations* or *instantaneous descriptions* (IDs).
- $I \vdash J$ means “$I$ can go to $J$ in one step.” *Directed edge*.
- Desired that the string representations of $I$ and $J$ have *edit distance* at most 1 or at most 2.
A *Turing Machine* (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of *states*.
- $s$, a member of $Q$, is the *start state*.
- $F$, a subset of $Q$, is the set of *desired final states*, also called *accepting states*.
- $\Sigma$ is the *input alphabet*; often $\Sigma = \{0, 1\}$.
- $\Gamma$ is the *work alphabet* and contains the blank $B$.
- $\delta$ is a finite set of *instructions* (aka. “tuples” or “transitions”) of the form $\delta = (p, c, d, D, q)$ where $p, q \in Q$, $c, d \in \Gamma$, and the “direction” $D$ is either *Left*, *Right*, or *Stay*.

A multitape Turing machine makes $Q^k$ instead for some $k > 1$. 

[Show “O-O” notation and “3n+1” example.]
A *Turing Machine* (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of *states*.
- $\Sigma$ is the *input alphabet*; often $\Sigma = \{0, 1\}$.
- $\Gamma$ is the *work alphabet* and contains $\Sigma$ and the blank $B$.
- $\delta$ is a finite set of instructions (aka. "tuples" or "transitions") of the form $(p, c, d, D)$ where $p, q \in Q$, $c, d \in \Gamma$, and the "direction" $D$ is either Left, Right, or Stay.
- $B$ is the blank symbol.
- $s$ is the *start state*.
- $F$, a subset of $Q$, is the set of desired final states, also called *accepting states*.
Turing Machines

A Turing Machine (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of states.
- $s$, a member of $Q$, is the start state.
A Turing Machine (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of states.
- $s$, a member of $Q$, is the start state.
- $F$, a subset of $Q$, is the set of desired final states, also called accepting states.
A Turing Machine (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of states.
- $s$, a member of $Q$, is the start state.
- $F$, a subset of $Q$, is the set of desired final states, also called accepting states.
- $\Sigma$ is the input alphabet; often $\Sigma = \{0, 1\}$. 

...
Turing Machines

A *Turing Machine* (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of *states*.
- $s$, a member of $Q$, is the *start state*.
- $F$, a subset of $Q$, is the set of *desired final* states, also called *accepting* states.
- $\Sigma$ is the *input alphabet*; often $\Sigma = \{0, 1\}$.
- $\Gamma$ is the *work alphabet* and contains $\Sigma$ and the *blank* $B$. 


Turing Machines

A *Turing Machine* (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, B, s, F)$ where:

- $Q$ is a finite set of *states*.
- $s$, a member of $Q$, is the *start state*.
- $F$, a subset of $Q$, is the set of *desired final* states, also called *accepting* states.
- $\Sigma$ is the *input alphabet*; often $\Sigma = \{0, 1\}$.
- $\Gamma$ is the *work alphabet* and contains $\Sigma$ and the *blank* $B$.
- $\delta$ is a finite set of *instructions* (aka. “tuples” or “transitions”) of the form
  \[ \tau = (p, c, d, D, q) \]
  
  where $p, q \in Q$, $c, d \in \Gamma$, and the “direction’ $D$ is either *Left*, *Right*, or *Stay*.

A *multitape Turing machine* makes $\delta \subset Q \times \Gamma^k \times \Gamma^k \times \{L, R, S\}^k \times Q$ instead for some $k > 1$. [Show “O-O” notation and “3n+1” example.]
DTM and NTM and Halting

- The definition allows two different instructions 
  \((p, c, d, D, q), (p, c, d', D', q')\) to begin with the same \(p\) ad \(c\) (or \(k\)-tuple of chars).
DTM and NTM and Halting

- The definition allows two different instructions \((p, c, d, D, q), (p, c, d', D', q')\) to begin with the same \(p\) ad \(c\) (or \(k\)-tuple of chars).
- When that happens, \(M\) has *nondeterminism* at state \(p\) reading \(c\). Any such case makes it an NTM for *nondeterministic Turing machine*.
DTM and NTM and Halting

- The definition allows two different instructions \((p, c, d, D, q), (p, c, d', D', q')\) to begin with the same \(p\) ad \(c\) (or \(k\)-tuple of chars).
- When that happens, \(M\) has non-determinism at state \(p\) reading \(c\). Any such case makes it an NTM for non-deterministic Turing machine.
- If it never happens, then \(M\) is deterministic and is called a DTM.
DTM and NTM and Halting

- The definition allows two different instructions 
  \((p, c, d, D, q), (p, c, d', D', q')\) to begin with the same \(p\) ad \(c\) (or \(k\)-tuple of chars).
- When that happens, \(M\) has *nondeterminism* at state \(p\) reading \(c\). Any such case makes it an NTM for *nondeterministic Turing machine*.
- If it never happens, then \(M\) is *deterministic* and is called a DTM.
- If there is *no* instruction for a state \(p\) and char(s) \(c\), then if and when \(M\) reaches state \(p\) where it is reading \(c\), \(M\) *halts*. Then \(M\) *accepts* if and only if \(p \in F\).
DTM and NTM and Halting

- The definition allows two different instructions \((p, c, d, D, q), (p, c, d', D', q')\) to begin with the same \(p\) ad \(c\) (or \(k\)-tuple of chars).
- When that happens, \(M\) has \textit{nondeterminism} at state \(p\) reading \(c\). Any such case makes it an NTM for \textit{nondeterministic Turing machine}.
- If it never happens, then \(M\) is \textit{deterministic} and is called a DTM.
- If there is \textit{no} instruction for a state \(p\) and char(s) \(c\), then if and when \(M\) reaches state \(p\) where it is reading \(c\), \(M\) \textit{halts}. Then \(M\) \textit{accepts} if and only if \(p \in F\).
- On any input string \(x\) over the alphabet \(\Sigma\) (notation: \(x \in \Sigma^*\)—the * means “zero or more” chars so the \textit{empty string} \(\lambda\) is included), \(M\) starts with \(x\) on its first tape and any other tapes completely blank, and its head scans the first char \(x_1\) of \(x\).
DTM and NTM and Halting

- The definition allows two different instructions 
  \((p, c, d, D, q), (p, c, d', D', q')\) to begin with the same \(p\) ad \(c\) (or \(k\)-tuple of chars).
- When that happens, \(M\) has nondeterminism at state \(p\) reading \(c\). Any such case makes it an NTM for nondeterministic Turing machine.
- If it never happens, then \(M\) is deterministic and is called a DTM.
- If there is no instruction for a state \(p\) and char(s) \(c\), then if and when \(M\) reaches state \(p\) where it is reading \(c\), \(M\) halts. Then \(M\) accepts if and only if \(p \in F\).
- On any input string \(x\) over the alphabet \(\Sigma\) (notation: \(x \in \Sigma^*\)—the * means “zero or more” chars so the empty string \(\lambda\) is included), \(M\) starts with \(x\) on its first tape and any other tapes completely blank, and its head scans the first char \(x_1\) of \(x\).
- If \(x = \lambda\) then all tapes are blank and the head scans \(B\).
Configurations

- Configurations of a 1-tape TM can have the form

\[ I = u(q) v \]

where \( q \) is the current state, \( c \) the character scanned, \( u \in \Gamma^* \) stretches out to the leftmost nonblank cell, and \( v \in \Gamma^* \) stretches out to the rightmost nonblank cell.
Configurations

- Configurations of a 1-tape TM can have the form
  \[ I = u(q) v \]
  where \( q \) is the current state, \( c \) the character scanned, \( u \in \Gamma^* \) stretches out to the leftmost nonblank cell, and \( v \in \Gamma^* \) stretches out to the rightmost nonblank cell.

- Possibly \( u, v = \lambda \) and possibly \( c = B \). All cells not included in \( ucv \) are blank.
Configurations

- Configurations of a 1-tape TM can have the form
  \[ I = u(q) v \]
  where \( q \) is the current state, \( c \) the character scanned, \( u \in \Gamma^* \) stretches out to the leftmost nonblank cell, and \( v \in \Gamma^* \) stretches out to the rightmost nonblank cell.
- Possibly \( u, v = \lambda \) and possibly \( c = B \). All cells not included in \( ucv \) are blank.
- Initial ID on an input \( x \in \Sigma^n \) is
  \[ I_0(x) = (x_1, x_2, \ldots, x_n); \quad I_0(\lambda) = (B, \ldots, B). \]
Configurations

- Configurations of a 1-tape TM can have the form
  \[ I = u(q,c)v \]
  where \( q \) is the current state, \( c \) the character scanned, \( u \in \Gamma^* \) stretches out to the leftmost nonblank cell, and \( v \in \Gamma^* \) stretches out to the rightmost nonblank cell.
- Possibly \( u, v = \lambda \) and possibly \( c = B \). All cells not included in \( ucv \) are blank.
- Initial ID on an input \( x \in \Sigma^n \) is
  \[ I_0(x) = (s_{x_1})x_2 \cdots x_n; \quad I_0(\lambda) = (s_B). \]
- Note this is a string over the “ID alphabet” \( \Gamma' = \Gamma \cup (Q \times \Gamma) \).
Configurations

- Configurations of a 1-tape TM can have the form
  \[ I = u(q_c)v \]
  where \( q \) is the current state, \( c \) the character scanned, \( u \in \Gamma^* \) stretches out to the leftmost nonblank cell, and \( v \in \Gamma^* \) stretches out to the rightmost nonblank cell.
- Possibly \( u, v = \lambda \) and possibly \( c = B \). All cells not included in \( ucv \) are blank.
- Initial ID on an input \( x \in \Sigma^n \) is
  \[ I_0(x) = (x_1^s)x_2 \cdots x_n; \quad I_0(\lambda) = (B^s). \]

- Note this is a string over the “ID alphabet” \( \Gamma' = \Gamma \cup (Q \times \Gamma) \).
- For multitape TMs we get \( k \)-tuples of strings, each indicating the current location of the head on its tape, but we treat the whole thing as one memory map.
The Computation Graph

Write $I \vdash_M J$ if there is an instruction $\tau = (p, c, d, D, q)$ such that $I = u(p)c \nu$ and carrying out the action of $\tau$ on $I$ leaves $J$. 


The Computation Graph

- Write $I \vdash_M J$ if there is an instruction $\tau = (p, c, d, D, q)$ such that $I = u(p)_c v$ and carrying out the action of $\tau$ on $I$ leaves $J$.

- (A precise formal definition is a self-study exercise; the “edge cases” are tricky when $I$ involves expanding out to a new cell or contracting by blanking out a cell on the end.)
The Computation Graph

- Write $I \vdash_M J$ if there is an instruction $\tau = (p, c, d, D, q)$ such that $I = u(p,c)v$ and carrying out the action of $\tau$ on $I$ leaves $J$.

- (A precise formal definition is a self-study exercise; the “edge cases” are tricky when $I$ involves expanding out to a new cell or contracting by blanking out a cell on the end.)

- Write $I \vdash^0_M I$ for all $I$, and for $k \geq 2$, define $I \vdash^k_M J$ if there are IDs $I_1, \ldots, I_{k-1}$ such that

$$I \vdash_M I_1 \vdash_M I_2 \vdash_M \cdots \vdash_M I_{k-1} \vdash_M J.$$  

This just expresses that there is a path from node $I$ to node $J$ in the directed graph we’ve defined.
The Computation Graph

- Write $I \vdash_M J$ if there is an instruction $\tau = (p, c, d, D, q)$ such that $I = u(p)c_v$ and carrying out the action of $\tau$ on $I$ leaves $J$.
- (A precise formal definition is a self-study exercise; the “edge cases” are tricky when $I$ involves expanding out to a new cell or contracting by blanking out a cell on the end.)
- Write $I \vdash_0 M I$ for all $I$, and for $k \geq 2$, define $I \vdash_k M J$ if there are IDs $I_1, \ldots, I_{k-1}$ such that

$$I \vdash_M I_1 \vdash_M I_2 \vdash_M \cdots \vdash_M I_{k-1} \vdash_M J.$$ 

This just expresses that there is a path from node $I$ to node $J$ in the directed graph we’ve defined.
- Then $M$ accepts $x$ if there is a path from $I_0(x)$ to some halting ID $J = u(q)c_v$ in which $q \in F$. And $L(M) = \{x \in \Sigma^* : M$ accepts $x\}$. 
“Good Housekeeping” Normal Form

If $M$ halts in state $q$ reading $c$, we can always add a transition $(q, c, c, R, q')$ with a new state $q'$ that begins a routine doing the following:

- Move to the rightmost non-blank character (on each tape).
“Good Housekeeping” Normal Form

If $M$ halts in state $q$ reading $c$, we can always add a transition $(q, c, c, R, q')$ with a new state $q'$ that begins a routine doing the following:

- Move to the rightmost non-blank character (on each tape).
- Sweep right-to-left blanking out the entire tape(s).
“Good Housekeeping” Normal Form

If $M$ halts in state $q$ reading $c$, we can always add a transition $(q, c, c, R, q')$ with a new state $q'$ that begins a routine doing the following:

- Move to the rightmost non-blank character (on each tape).
- Sweep right-to-left blanking out the entire tape(s).
- If $q$ was accepting, end in a unique accepting state $q_a$ scanning a solitary 1.
“Good Housekeeping” Normal Form

If $M$ halts in state $q$ reading $c$, we can always add a transition $(q, c, c, R, q')$ with a new state $q'$ that begins a routine doing the following:

- Move to the rightmost non-blank character (on each tape).
- Sweep right-to-left blanking out the entire tape(s).
- If $q$ was accepting, end in a unique accepting state $q_a$ scanning a solitary 1. If not, end in the rejecting ID $I_r = (q_r^0)$ instead.
“Good Housekeeping” Normal Form

If $M$ halts in state $q$ reading $c$, we can always add a transition $(q, c, c, R, q')$ with a new state $q'$ that begins a routine doing the following:

- Move to the rightmost non-blank character (on each tape).
- Sweep right-to-left blanking out the entire tape(s).
- If $q$ was accepting, end in a unique accepting state $q_a$ scanning a solitary 1. If not, end in the rejecting ID $I_r = (q_r 0)$ instead.

Needed for this is that $M$ never writes $B$ except in the final phase, so $ucv$ never has an internal blank which could deceive this routine, and/or maintains endmarkers $\wedge, $ to bound the tape(s). We always assume this form—many texts including Sipser’s define it.
“Good Housekeeping” Normal Form

If $M$ halts in state $q$ reading $c$, we can always add a transition $(q, c, c, R, q')$ with a new state $q'$ that begins a routine doing the following:

- Move to the rightmost non-blank character (on each tape).
- Sweep right-to-left blanking out the entire tape(s).
- If $q$ was accepting, end in a unique accepting state $q_a$ scanning a solitary 1. If not, end in the rejecting ID $I_r = (q_r^0)$ instead.

Needed for this is that $M$ never writes $B$ except in the final phase, so $ucv$ never has an internal blank which could deceive this routine, and/or maintains endmarkers $\wedge, \$\$ to bound the tape(s). We always assume this form—many texts including Sipser’s define it.

Thus the “ID Graph” $G_M$ has a unique goal node $I_f = (q_a^1)$ and one other sink $I_r$. 
Time and Space Consumed

- The *time* for an accepting computation $I_0(x) \vdash_I^{t_M} I_f$ is just the number $t$ of steps.
Time and Space Consumed

- The *time* for an accepting computation $I_0(x) \vdash_M^t I_f$ is just the number $t$ of steps.
- The *space* is the number of cells whose contents were *changed* to another non-blank char.
Time and Space Consumed

- The *time* for an accepting computation \( I_0(x) \xrightarrow{t \ M} I_f \) is just the number \( t \) of steps.
- The *space* is the number of cells whose contents were *changed* to another non-blank char.
- So if the cells holding the input bits \( x_1, \ldots, x_n \) are left alone (until the final erasure) they are not charged against the space bound.
Time and Space Consumed

- The *time* for an accepting computation $I_0(x) \vdash^t_M I_f$ is just the number $t$ of steps.
- The *space* is the number of cells whose contents were *changed* to another non-blank char.
- So if the cells holding the input bits $x_1, \ldots, x_n$ are left alone (until the final erasure) they are not charged against the space bound.
- Convenient to hold $x$ on a separate *read-only input tape*. 

Kolkata Algorithms Short Course: I. The Algorithm-Complexity Landscape
Time and Space Consumed

- The *time* for an accepting computation $I_0(x) \vdash^t_M I_f$ is just the number $t$ of steps.
- The *space* is the number of cells whose contents were *changed* to another non-blank char.
- So if the cells holding the input bits $x_1, \ldots, x_n$ are left alone (until the final erasure) they are not charged against the space bound.
- Convenient to hold $x$ on a separate *read-only input tape*.
- A DTM *runs within time* $t(n)$ *and space* $s(n)$ if for all $n$ and inputs $x \in \Sigma^n$, the unique computation halts within $t(n)$ steps having used space at most $s(n)$.
Time and Space Consumed

- The time for an accepting computation $I_0(x) \vdash_M^t I_f$ is just the number $t$ of steps.
- The space is the number of cells whose contents were changed to another non-blank char.
- So if the cells holding the input bits $x_1, \ldots, x_n$ are left alone (until the final erasure) they are not charged against the space bound.
- Convenient to hold $x$ on a separate read-only input tape.
- A DTM runs within time $t(n)$ and space $s(n)$ if for all $n$ and inputs $x \in \Sigma^n$, the unique computation halts within $t(n)$ steps having used space at most $s(n)$.
- For NTMs we require this of all computation paths.
Time and Space Consumed

- The *time* for an accepting computation $I_0(x) \vdash_M^t I_f$ is just the number $t$ of steps.
- The *space* is the number of cells whose contents were *changed* to another non-blank char.
- So if the cells holding the input bits $x_1, \ldots, x_n$ are left alone (until the final erasure) they are not charged against the space bound.
- Convenient to hold $x$ on a separate *read-only input tape*.
- A DTM runs *within time* $t(n)$ and *space* $s(n)$ if for all $n$ and inputs $x \in \Sigma^n$, the unique computation halts within $t(n)$ steps having used space at most $s(n)$.
- For NTMs we require this of *all* computation paths.
- $\text{DTIME}[t(n)] = \text{the class of languages } L(M)$ for DTM$s$ that run within time $t(n)$. 

**P** = $\bigcup_k \text{DTIME}[n^k]$, $\text{NP} = \bigcup_k \text{NTIME}[n^k]$, and $\text{NSPACE}[s(n)]$ are defined analogously.
Time and Space Consumed

- The time for an accepting computation $I_0(x) \vdash^t_M I_f$ is just the number $t$ of steps.
- The space is the number of cells whose contents were changed to another non-blank char.
- So if the cells holding the input bits $x_1, \ldots, x_n$ are left alone (until the final erasure) they are not charged against the space bound.
- Convenient to hold $x$ on a separate read-only input tape.
- A DTM runs within time $t(n)$ and space $s(n)$ if for all $n$ and inputs $x \in \Sigma^n$, the unique computation halts within $t(n)$ steps having used space at most $s(n)$.
- For NTMs we require this of all computation paths.
- $\text{DTIME}[t(n)] = \text{the class of languages } L(M) \text{ for DTMs that run within time } t(n)$.
- $\text{DSPACE}[s(n)], \text{NTIME}[t(n)],$ and $\text{NSPACE}[s(n)]$ are defined analogously. $P = \bigcup_k \text{DTIME}[n^k], \text{NP} = \bigcup_k \text{NTIME}[n^k]$. 
Polynomial time can be stated in terms of “scalability”:

There is a constant $K$ such that whenever your data size doubles, the time to process it goes up by a factor of no more than $K$.

Well, if the time is $O(n^2)$, then $K = 4$, if $O(n^3)$, then $K = 8$, and so on. But still “linear scaling.”

With $O(n)$ time we have $K = 2$ strictly. With $O(n \log n)$ time, or even $O(n(\log n)^k)$ time for $k > 1$, we have “$K = 2^+$ scaling.” This is called quasilinear time and will be contrasted with quadratic time later.

For space we can define sub-linear bounds, even “space zero.” Space zero is achieved by DTM$s$ and NTMs that do one left-to-right scan and halt upon reading the $B$ after the input in step $n + 1$. They are called (deterministic and nondeterministic) finite automata and accept regular languages.
What Low Space Means

A theorem:

\[ \text{REG} = \text{DSPACE}[0] = \text{NSPACE}[0]. \]

This states that NFAs and DFAs are equivalent for defining regular languages.

*Logarithmic* space represents problems that we can decide with finitely many fingers into a read-only data structure. We define:

\[ L = \text{DSPACE}[O(\log n)], \quad NL = \text{NSPACE}[O(\log n)]. \]

A typical problem in NL is, given a directed graph \( G \) and nodes \( s, f \), is there a path from \( s \) to \( f \) in \( G \)?

[Lecture transits to board showing logspace graph examples: TRIANGLE and GAP.]
Breadth-First Search for GAP

set<Node> FOUND = {s}
bool novel = true;
while (novel) {
    novel = false;
    foreach (u in FOUND) {
        foreach (v: u->v) {
            if (v not in FOUND) {
                novel = true;
                FOUND += {v};
            }
        }
    }
}
accept iff t in FOUND.
Better Version: Queue Found Nodes

```plaintext
set <Node> FOUND = {s}, POPPED = {};
bool novel = true;
while (novel) {
    novel = false;
    foreach (u in FOUND \ POPPED) {
        foreach (v: u—>v) {
            if (v not in FOUND) {
                novel = true;
                FOUND += {v};
            }
        }
    }
    POPPED += {u};  // Each edge polled at most once,
}  // so time = O(|V|+|E|) = O(m) = O(n^2).
accept iff t in FOUND.
```