Illinois Institute of Technology
Department of Computer Science

Lecture 1: August 19, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Why Study Mathematics and Theoretical Computer Science?

This question is best answered by an example. Consider yourself an employee of a factory. Your boss comes up to you with the following dilemma: "Because of the tough economic times, we need to operate at peak efficiency. We need you to determine what our peak efficiency is at the lowest cost possible."

After further discussion with your employer, you determine the following facts:

- The function relating efficiency to the number of employees is **unimodal**. That is, efficiency will increase to a maximum, then decrease as the number of employees increases.
- The only way to determine efficiency is to build a pilot plant.
- Pilot plants are expensive to build, and your boss will be unhappy with any solution that requires the construction of more pilot plants than necessary.

There is an obvious solution to this problem: you could build a plant for one employee then find the efficiency of that plant, then build a plant with two employees and find its efficiency, then build a plant for three employees, and so on, until the efficiency goes down. You will then have the peak.

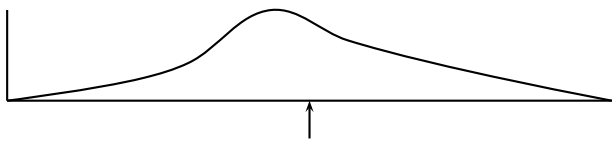
1.1 Finding a Better Solution

Unfortunately, this is not a very efficient solution. It runs in **linear** time—that is, time proportional to the number of employees. (Consider that the peak may be at a few thousand employees. Your boss won't be happy building thousands of pilot plants if he can do better.) But, as you are familiar with the **binary search algorithm** and that it is a faster way to search for items. You may even remember that this process is *logarithmic* is nature.

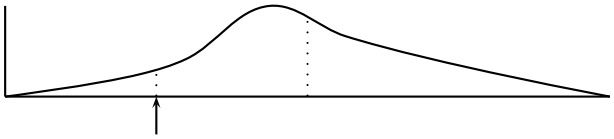
In binary search, we probe the center point and can immediately discard either the first half or the second half of the search space. This problem does not exactly match those required for a binary search, but we can modify it somewhat.

For example, we consider the range of employee numbers in our experiments to be between 1 and a zillion: the total number of employees will not exceed the world population. Then we begin probing certain points to collect information.

- Probe the center point ($\frac{1}{2}$)



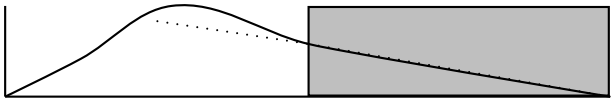
- Probe the center of one resulting segment ($\frac{1}{4}$)



- Determine if the efficiency is rising or falling within the segments and discard inappropiate segment

  If the line between $\frac{1}{4}$ and $\frac{1}{2}$ is ascending, as shown in the figure below, the lower $\frac{1}{4}$ is discarded because the curve is rising from 0 to $\frac{1}{4}$, and from $\frac{1}{4}$ to $\frac{1}{2}$. The peak can not exist in the lower $\frac{1}{4}$ segment.

  If the line between $\frac{1}{4}$ and $\frac{1}{2}$ is descending, as shown in the figure below, the upper $\frac{1}{2}$ is discarded (why?).



- Start again (recursively) on the new, shorter segment

  In case 1 (curve is ascending), the shorter segment is the upper $\frac{3}{4}$ and the probe we have at $\frac{1}{2}$ is not at the center of the new segment. Therefore we still need to make two probes, at the center and the $\frac{1}{4}$ point of the new segment respectively, to apply our approach.

  In case 2 (curve is descending), the shorter segment is the lower $\frac{1}{2}$ and the probe we have at $\frac{1}{4}$ is at the center of the new segment. We can reuse this probe and only make one more probe at the $\frac{1}{4}$ point of the new segment.

To analyze the cost of this approach, let us consider the worst case. That is, for every recursion, we need to make two probes and can only remove $\frac{1}{4}$ of the current segment. Notice that the size of the resulting segment is $\frac{3}{4}$ the size of the original. With this piece of information, we can create a **recurrence relation** to describe this process.

Consider the cost of determining the peak efficiency of n employees. If you define the function $\text{cost}(n)$ as the dollars required, after each step you can make 2 probe and get a segment $\frac{1}{4}$ shorter. This can be written as:

$$\text{cost}(n) = \text{cost}\left(\frac{3}{4}n\right) + 2$$

There is also an initial step, in which you probed in the center. This is the base case:

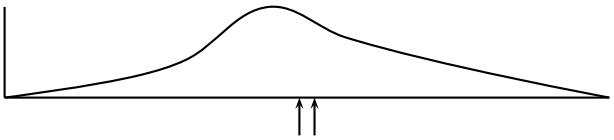
$$\text{cost}(1) = 1$$

From these facts, and through methods learned later in the course, you will show that:

$$\text{cost}(n) \approx 2\log_{\frac{4}{3}} n \ \approx 4.8188 \log_2 n$$

This approach actually works better since there is a possibility that $\frac{1}{2}$ will be removed with one probe.

We can use a better approach, as suggested by a former student, which makes two probes near the center of the segment, hereby removing almost $\frac{1}{2}$ of the segment everytime.



The recurrence relation for this approach is:

$$\text{cost}(n) \approx \text{cost}\left(\frac{1}{2}n\right) + 2$$

From this relation we get:

$$\text{cost}(n) \approx 2\log_2 n$$

We can see this is better than the previous method. But is it the best one? The answer is no, as we will see soon.

1.2 Finding the Best Solution

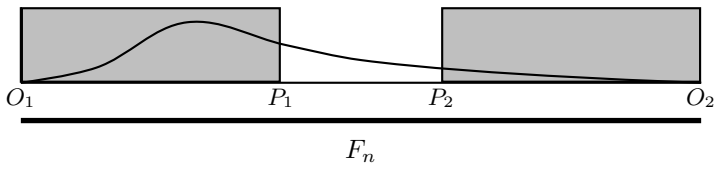
The question that you should now ask is, "Where is the best place to make my probes?" The best solution will try to make as few probes as possible. If during every step, we can use a probe made previously, we only need to make one more probe. This requires that we pick the probe points very carefully so that if a probe point falls into the shortened segment, it is still at a "good" position.

First we make two probes. The two probe points have the same distance to the center point. This makes sure that no matter which part is removed, the remaining segment will have the same size. In the figure below, P_1 and P_2 are the two probe points. We need the property

$$O_1P_2 = P_1O_2$$

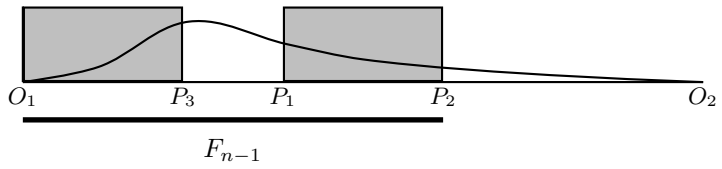
Which implies

$$O_1P_1 = P_2O_2$$



We use the notation: F_n to represent the level (or problem size) currently being worked on. F_{n-1} is the next level to be worked on and F_{n-2} is the level after that. Notice that $F_n \geq F_{n-1} \geq F_{n-2}$.

Suppose that we keep the lower part of the segment. We hope that we need only another one probe at P_3 and can reuse P_1, as shown by the figure below:



We have:

$$F_{n-1} = O_1P_2$$

so the search can continue recursively. By the same reason for the next iteration to work correctly we must have

$$F_{n-2} = O_1P_1 = P_2O_2$$

$$F_n = O_1O_2 = O_1P_2 + P_2O_2 = O_1P_2 + O_1P_1$$

Those equations give us:

$$F_n = F_{n-1} + F_{n-2}$$

This formula is used to generate the Fibonacci sequence of numbers $1, 1, 2, 3, 5, 8, 13, 21, 34 \cdots$, and $F_n \approx \phi F_{n-1}$, as we will see in a few weeks.

This approach has the property

$$\text{cost}(F_n) = \text{cost}(F_{n-1}) + 1$$

Letting $N = F_n$, we have the recursive relation

$$\text{cost}(N) = \text{cost}\left(\frac{N}{\phi}\right) + 1,$$

where

$$\phi = \frac{1+\sqrt{5}}{2} \approx 1.6180$$

ϕ is the **golden ratio** (which, incidentally, bears an uncanny resemblance to the zip code of Urbana).

From the recursion we get:

$$\text{cost}(n) = \log_\phi n \approx 1.4405 \log_2 n$$

This cost is **optimal**, that is, no probing strategy can do better.

Illinois Institute of Technology
Department of Computer Science

Lecture 2: August 21, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Proof by contradiction

We will now look at an important but easily misunderstood proof technique. The essence of this technique is that to prove $A \Rightarrow B$, we will instead show $\bar{B} \Rightarrow \bar{A}$.

1.1 Why does a proof by contradiction work?

It can be shown that these two forms are equivalent by examining the truth table of both functions. Since $A \Rightarrow B \iff \bar{B} \Rightarrow \bar{A}$ is a **tautology**, proving one implication proves the other implication, and disproving one implication disproves the other.

A	B	$A \Rightarrow B$	$\bar{B}$	$\bar{A}$	$\bar{B} \Rightarrow \bar{A}$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

1.2 A sample proof by contradiction

Theorem: $\sqrt{2}$ is irrational. That is, $\sqrt{2}$ cannot be written as a/b, where a and b are integers with no common factors.

We first need to convert this to the $A \Rightarrow B$ form as above. One simple conversion is T $\Rightarrow \sqrt{2}$ is irrational. (Convince yourself by examining the truth table above that this is indeed a valid conversion.)

Proof by contradiction: We will show that $\sqrt{2}$ is rational implies F — that is, that if we assume that $\sqrt{2}$ is rational, we can derive a contradiction.

By the definition of rationality, $\sqrt{2} = \frac{a}{b}$, for two relatively prime integers a and b. Thus $\sqrt{2}b = a$. It follows that $2b^2 = a^2$ and, by the definition of an even number, that a^2 is even.

We now take a small diversion to help us arrive at our goal.

Lemma: a^2 is even $\Rightarrow a$ is even

Proof by contradiction: We show a is odd $\Rightarrow a^2$ is odd. Since a is odd, it has the form $2n + 1$, for some integer n. Thus $a^2 = (2n + 1)^2 = 4n^2 + 4n + 1 = 2(2n^2 + 2n) + 1$. Thus a^2 is odd.

Now that we have concluded that a^2 is even $\Rightarrow a$ is even, we can resume our original proof. Since a is even, it can be written as $2c$, where c is an integer. Thus $2b^2 = (2c)^2$, so $2b^2 = 4c^2$, and $b^2 = 2c^2$. Thus b^2 is even, and by the lemma we know that b is even.

So both a and b are even. They share the common factor 2. But we originally assumed that a and b had no common factors! Thus we have arrived at a contradiction and proved the original theorem, that $\sqrt{2}$ is

irrational.

1.3 Bertrand Russell's Proof

Bertrand Russell, the famous philosopher/mathematician, was challenged that because a false proposition implies any proposition, could he prove that if $2 + 2 = 5$, then he is the pope. "Yes," he responded:

- Suppose $2 + 2 = 5$.
- Subtract 2 from each side, giving $2 = 3$.
- Transpose to $3 = 2$.
- Subtract 1 from each side, giving $2 = 1$.

"Now, the pope and I are two, but two equals one. Therefore, I am the pope."

2 Proof by induction

2.1 Growth of harmonic numbers

We define the sequence of **harmonic numbers** as:

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{i=1}^{n} \frac{1}{i}$$

The recursion for harmonic numbers is

$$H_0 = 0$$

$$H_n = H_{n-1} + \frac{1}{n}, n \geq 1$$

They are called "harmonic numbers" because each term beyond the first is the *harmonic mean* of the two neighbors, where the harmonic mean of $x_1, x_2, \ldots, x_k$ is defined as

$$\frac{k}{1/x_1 + 1/x_2 + \cdots + 1/x_k},$$

and we have

$$\frac{2}{\frac{1}{1/(n-1)} + \frac{1}{1/(n+1)}} = \frac{2}{n-1+n+1} = \frac{1}{n}.$$

The harmonic mean gives the correct "average" in many situations involving ratios: if you connect k resistors in *parallel*, $1/k$ times the harmonic mean gives the effective resistance. The name "harmonic" comes the ancient Greek use in music because it gave "harmonious" ratios.

Before we proceed any further, let's tie this into a recurring concept in computer science.

2.1.1 Rate of growth

Computer scientists find it useful to describe how fast functions grow as their input grows. Consider the function $f(n) = n$. This function grows at a **linear** rate: roughly speaking, if n is doubled, $f(n)$ will also be doubled. Other examples of functions exhibiting linear growth are $f(n) = 4n$ and $f(n) = \frac{n}{7}$.

Now consider $f(n) = n^2$. If n is doubled, $f(n)$ is quadrupled. This is a **quadratic** rate of growth. Likewise, $f(n) = n^3$ has a **cubic** rate of growth: if n is doubled, $f(n)$ increases by a factor of 8. We can similarly speak of rates of growth as quartic, quintic, and so on.

With the function $f(n) = 2^n$, the situation differs. If n is doubled, $f(n)$ is squared. We call this an **exponential** rate of growth. Comparing an exponential function with any of the polynomial functions discussed above will make it clear that the exponential grows much faster than any polymonial.

If you have seen the quicksort algorithm, you may recall that it makes $2nH_n$ comparisons in the average case. Where does that fall in the hierarchy presented above? We will attempt to obtain some information about the rate of growth of the harmonic numbers, and to do that we will use the technique of *mathematical induction*.

2.1.2 A result about the growth of harmonic numbers

Theorem: $H_{2^n} \geq 1 + \frac{n}{2}$, where $n = 0, 1, \ldots$

Proof by induction: First we consider the *base case*. (In a domino setting, this case is analogous to the tapping of the first domino.) We need to show that $H_{2^0} \geq 1 + \frac{0}{2}$, or, simplified, that $H_1 \geq 1 + 0$. By the definition of H_n, H_1 is 1. As $1 = 1$, the inequality holds.

Now we consider the *inductive step*. (This is analogous to the contact of each domino with the next domino.) For some arbitrary n, we first assume that $H_{2^n} \geq 1 + \frac{n}{2}$. We must then show that, based on this assumption, $H_{2^{n+1}} \geq 1 + \frac{n+1}{2}$.

Expanding H_{2^n} as per the definition, we arrive at:

$$1 + \frac{1}{2} + \cdots + \frac{1}{2^n}$$

Similarly expanding $H_{2^{n+1}}$ produces:

$$1 + \frac{1}{2} + \cdots + \frac{1}{2^n} + \frac{1}{2^n + 1} + \cdots + \frac{1}{2^{n+1}}$$

Notice that the first 2^n terms of these expansions are identical. That is, the sum of the first 2^n terms of $H_{2^{n+1}}$ is H_{2^n}. From our inductive hypothesis, we know that $H_{2^n} \geq 1 + \frac{n}{2}$. Let's look at the remaining terms. It is not difficult to show that $\frac{1}{2^n+1} \geq \frac{1}{2^{n+1}}$. (Consider that n is positive and manipulate the inequality algebraically.) Likewise, $\frac{1}{2^n+2} \geq \frac{1}{2^{n+1}}$, and so on. Of course, $\frac{1}{2^{n+1}} \geq \frac{1}{2^{n+1}}$. So each of the terms of $H_{2^{n+1}}$ past the nth is at least $\frac{1}{2^{n+1}}$.

How many of these terms are there? The entire harmonic number has 2^{n+1} terms, and we're not looking at the first 2^n of them right now. That leaves $2^{n+1} - 2^n = 2^n$ terms. So the sum of the last 2^n terms is at least $\frac{2^n}{2^{n+1}}$. This is simply $\frac{1}{2}$.

Adding the first 2^n terms to the remaining terms, we now know that $H_{2^{n+1}} \geq 1 + \frac{n}{2} + \frac{1}{2} = 1 + \frac{n+1}{2}$. This is precisely what we needed to show in the inductive step, so our proof is complete.

Similarly, we can prove

Theorem: $H_{2^n} \leq 1 + n$, where $n = 0, 1, \ldots$

Proof by induction: First we consider the *base case*. We need to show that $H_{2^0} \leq 1+0$. By the definition of H_n, H_1 is 1. As $1 = 1$, the inequality holds.

Now we consider the *inductive step*. For some arbitrary n, we first assume that $H_{2^n} \leq 1+n$. We must then show that, based on this assumption, $H_{2^{n+1}} \leq 1+(n+1)$.

Expanding $H_{2^{n+1}}$ produces:

$$1 + \frac{1}{2} + \cdots + \frac{1}{2^n} + \frac{1}{2^n+1} + \cdots + \frac{1}{2^{n+1}}$$

Notice that the sum of the first 2^n terms of $H_{2^{n+1}}$ is H_{2^n}. From our inductive hypothesis, we know that $H_{2^n} \leq 1+n$. Let's look at the remaining terms.

Similar to last proof, $\frac{1}{2^{n+1}} \leq \frac{1}{2^n}$. Likewise, $\frac{1}{2^{n+1}-1} \leq \frac{1}{2^n}$, and so on. Of course, $\frac{1}{2^n+1} \leq \frac{1}{2^n}$. So each of the terms of $H_{2^{n+1}}$ past the nth is at most $\frac{1}{2^n}$ and there are totally 2^n these terms. Therefore the sum of the last 2^n terms cannot exceed $\frac{2^n}{2^n} = 1$.

Adding the first 2^n terms to the remaining terms, we now know that $H_{2^{n+1}} \leq (1+n)+1 = 1+(n+1)$. This is precisely what we needed to show in the inductive step, so our proof is complete.

2.1.3 Extending this result to an arbitrary n

We have shown an inequality that gives us information about H_{2^n}, but we were looking for information about H_n, where n may or may not be a power of 2.

Let $k = 2^n$. Equivalently, $\ln k = \ln 2^n = n \ln 2$. So $n = \frac{\ln k}{\ln 2}$, and we can conclude that $H_k \geq 1 + \frac{\ln k}{2 \ln 2}$ and $H_k \leq 1 + \frac{\ln k}{\ln 2}$. These two bounds tell us that $H_k = \Theta(\ln k)$. We will sharpen this result considerably in a later lecture.

2.1.4 Similar sums

Instead of sums of reciprocals of integers, what about sums of reciprocals of the odd numbers? Call this O_n, with the corresponding sum of reciprocals of even numbers being E_n. Then, $H_n = O_n + E_n$ and $E_n = H_n/2$, implying that as n gets large both O_n and E_n diverge, tending to $(\ln n)/2$.

What about sums of reciprocals of *powers of integers*? Define

$$H_n^{(2)} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots + \frac{1}{n^2} = \sum_{i=1}^{n} \frac{1}{i^2}$$

Does $H_n^{(2)}$ also grow unboundedly as n gets large? No, for we can write

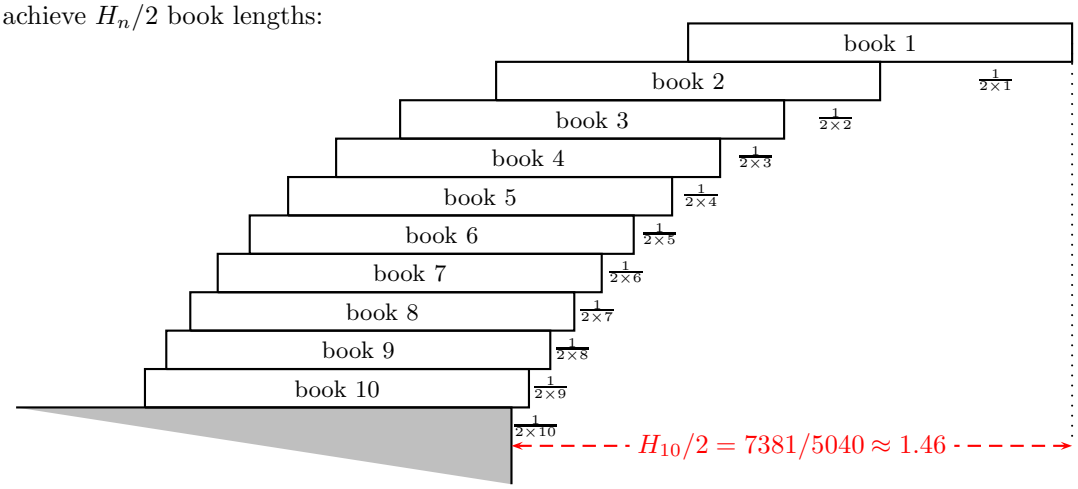
$$\begin{aligned} H_n^{(2)} &= 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots \\ &= 1 + \left(\frac{1}{2^2} + \frac{1}{3^2}\right) + \left(\frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \frac{1}{7^2}\right) + \cdots \end{aligned}$$

where each parenthesized term contains 2^k terms beginning with $1/2^{2k}$. Thus,

$$\begin{aligned} H_n^{(2)} &< 1 + \frac{2}{2^2} + \frac{4}{4^2} + \cdots \\ &= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots \\ &= 2. \end{aligned}$$

2.1.5 Stacking books

How far can a stack of n books extend over the edge of a table without the stack falling over? For n books we can achieve $H_n/2$ book lengths:



Label the top book of the stack 1, the second book on the stack 2, ..., and the bottom book on the stack n; the table's edge is considered the $(n+1)$st book on the stack.

Let d_i be the distance from the right edge of book i to the right edge of book 1 (the dotted line). This makes $d_1 = 0$, d_2 the amount by which the top book overhangs book 2, the second-from-the-top book, etc. Then d_{n+1} is the amount by which the top book overhangs the table edge. We must make d_{i+1} the center of gravity of the top i books—that is, each book is placed so its center is just above the center of gravity of the stack of books below it.

The center of gravity of k objects having weights $w_1, \ldots, w_k$ with respective centers of gravity at positions $p_1, \ldots, p_k$ is

$$\frac{w_1 p_1 + w_2 p_2 + \cdots + w_k p_k}{w_1 + w_2 + \cdots + w_k}$$

so that if we measure from the right edge of the top-most book, the center of the ith book is at position $p_i = d_i + 1/2$. We'll assume that each book weighs 1 unit, so the center of gravity of the n books is

$$\frac{1(d_1 + 1/2) + \cdots + 1(d_n + 1/2)}{n}$$

which must be at the table's edge which is d_{n+1}. Thus

$$d_{n+1} = \frac{1(d_1 + 1/2) + \cdots + 1(d_n + 1/2)}{n}$$

which we can rewrite as

$$n d_{n+1} = d_1 + \cdots + d_n + n/2.$$

But this holds for all $n \geq 0$, so it also holds for $n - 1$, as long as $n \geq 1$, giving

$$(n-1) d_n = d_1 + \cdots + d_{n-1} + (n-1)/2.$$

Subtracting these two equations gives

$$n d_{n+1} - (n-1) d_n = d_n + 1/2$$

or

$$d_{n+1} = d_n + 0.5/n.$$

Because $d_1 = 0$, $d_n = H_{n-1}/2$.

In other words, a 30-volume encyclopedia can be stacked so that it overhangs the table edge by $H_{30}/2 = \frac{9227046511387}{4658179125600} \approx 2$ volume lengths!

How many volumes would the encyclopedia need to get a volume overhanging the table edge by 4 volume lengths?

For further results on book stacking see Paterson and Zwick's paper "Overhang," *Amer. Math. Monthly*, January, 2009, pp. 19–44.

Illinois Institute of Technology
Department of Computer Science

Lecture 3: August 26, 2013

CS 330 Discrete Structures
Fall Semester, 2013

Three Applications of Induction

Crossing the desert

The same idea as in the book-stacking example leads to the following clever observation: If you have to cross the desert by Jeep with no sources of fuel other than what you can carry, you can do it if you have enough Jeeps and drivers, no matter how wide the desert is.

We'll measure distance by ToGs, "tankfuls of gas." If we only one Jeep, it can travel 1 ToG into the desert, at which point it is stranded. But if two Jeeps leave together, they can travel 1/3 ToG, at which point Jeep 2 can transfer half of its remaining fuel (1/3 tank) to Jeep 1; Jeep 2 can then use its remaining 1/3 tank to return to the starting point. Meanwhile, Jeep 1 will have traveled 1/3 ToG but now has a full tank, so it can continue 1 ToG, reaching a distance of $1 + 1/3$ ToG from the starting point.

With 3 Jeeps, they travel 1/5 ToG, at which point Jeep 3 transfers 1/5 to each of Jeeps 1 and 2, leaving its tank 2/5 full. Jeeps 1 and 2, whose tanks are now full, continue as in the previous paragraph: after Jeep 2 transfers 1/3 of a tank to Jeep 1, Jeep 2 returns to Jeep 3; Jeep 2's tank is empty, but Jeep 3 has 2/5 of a tank, allowing both of them to return to the starting point. Jeep 1 can now travel $1 + 1/3 + 1/5$ ToG from the starting point.

In general, with n Jeeps, the Jeeps travel $1/(2n-1)$ ToG at which point Jeep n transfers $1/(2n-1)$ of a tank to each of the other $n-1$ Jeeps which then have full tanks, leaving Jeep n with $1 - \frac{n}{2n-1} = \frac{n-1}{2n-1}$ tanks of gas. Jeeps $1, \ldots, n-1$ proceed recursively, with $n-2$ Jeeps returning empty to Jeep n, who then transfers $\frac{1}{2n-1}$ tanks of gas to each of them, leaving it also with $\frac{1}{2n-1}$ tanks of gas; Jeeps $2, \ldots, n$ can then return to the starting point. Jeep 1 can then travel $\frac{1}{2n-1}$ ToG further than it could with $n-1$ Jeeps. If Jeep 1 can travel D_k ToG with k Jeeps, then $D_1 = 1$ ToG and $D_k = D_{k-1} + 1/(2k-1)$.

That is, Jeep 1 can travel the sum of reciprocals of the odd numbers, which we have seen grows like $(\ln n)/2$.

Good guys versus bad guys

There is a room with a large number of people, say n, all knowledgeable about logic and mathematical induction, each wearing either a white hat or a black hat; nobody can see the color of his/her own hat, but everybody can see all the other hats in the room. A announcement is made that every hour a chime will sound and when it does, anybody who could logically deduce the color of his/her hat must leave the room; furthermore, the announcement says that there is at least one person wearing a black hat in the room.

Theorem. *If there are exactly k people in the room wearing black hats, those k leave the room at the kth chime.*

Proof. To get the idea of the induction, consider what happens with $k = 1$ black hat in the room. The person wearing that hat sees no other black hats, yet he knows that there is at least one, so it must be

his/her own; therefore he leaves the room at the first chime. Suppose there are $k = 2$ black hats in the room worn by A and B. At the first chime, A sees another black hat and so can deduce nothing about his/her own hat; similarly, B can deduce nothing about his/her own hat. But at the second chime, A knows that B did not leave the room on the first chime (and vice versa), so A can deduce that his/her own hat is black (if it were white, B would see only white hats and would have left at the first chime). B similarly deduces that he/she is wearing a black hat (again, if it were white, A would see only white hats and would have left at the first chime). Thus A and B leave at the second chime.

To make the inductive argument, let S_i be the statement "Everybody knows that at least i people are wearing black hats." We claim that S_i is true at the ith chime, that is, everybody knows that at least i people are wearing black hats. We have seen that this is true for $i = 1$ and $i = 2$. Suppose it is true for some $i \geq 2$. So, at the $(i+1)$st chime, each person reasons, "We all knew there were at least i black hats in the room, but nobody left; therefore *everybody* must be seeing at least i black hats because anybody seeing only $i - 1$ black hats would have deduced that he was wearing a black hat and would have left the room. But "everyone" includes at least one person wearing a black hat because of the initial announcement: that person, too, must be seeing i black hats, so there must be at least $i+1$ black hats in the room; everybody in the room makes that same deduction. Therefore at the $(i+1)$st chime, everybody knows that at least $i+1$ people are wearing black hats, proving S_{i+1}.

At the kth chime, therefore, everybody in the room knows that there are at least k black hats, but each person wearing a black hat can see only $k-1$; black hats, so those people deduce that they are wearing black hats and simultaneously leave the room. □

This example appears useless, but in suitable generalizations tell us how quickly information can spread among a network of computers ("Hat guessing games," *SIAM J. Discrete Math.*, vol. 22, no. 2, 2008, pp. 592–605).

Euclid's algorithm

Euclid's algorithm gives a method for computing the greatest common divisor of two integers u and v. WLOG we assume that $u \geq v$. The algorithm can be described by a sequence of remainders r_i, defined as:

$$r_0 = u$$

$$r_1 = v$$

$$r_{i+1} = r_{i-1} \bmod r_i$$

For any u and v, eventually one arrives at a remainder of 0; call it r_{k+1}. Then the greatest common divisor of u and v is r_k, so the algorithm gives a sequence

$$r_0 = u \geq r_1 = v > r_2 > r_3 > \cdots > r_k > r_{k+1} = 0.$$

This is a *decreasing* sequence of non-negative numbers, so it must be finite and hence the algorithm must terminate.

The algorithm can (should!) be expressed recursively:

$$\gcd(u, v) = \begin{cases} u & \text{if } v = 0, \\ \gcd(v, u \bmod v) & \text{otherwise,} \end{cases}$$

(Exercise: Prove this form is equivalent to the iterative form.)

We can ask two questions about this algorithm. First, how do we know that when it ends, r_k is actually the greatest common divisor of u and v? Second, for an input of a given size, how large is k? That is, how many iterations can proceed before a result is found?

The answer to the first question is given in Rosen on pages 267–269.

The second question is more interesting (see Rosen, page 347). Let us assume that u (the larger number) has n decimal digits, or, equivalently, that

$$10^n \leq u < 10^{n+1}.$$

The remainders decrease at each iteration; we make this observation more precise by noting that in general

$$r_{i-1} \geq r_i + r_{i+1}$$

Why is this so? Assume that $u \geq v$ so that $r_0 \geq r_1$, since $r_0 = u$ and $r_1 = v$. We know that the definition of r_{i+1} is the remainder when r_{i-1} is divided by r_i:

$$r_{i+1} = r_{i-1} \bmod r_i$$

or, in other words,

$$r_{i-1} = \text{ (some multiple of } r_i) + r_{i+1}$$

Since $r_{i-1} \geq r_i$, the "multiple of r_i" cannot be zero, so

$$r_{i-1} \geq r_i + r_{i+1}$$

for $i = 1, 2, \ldots$.

This means we have a chain of inequalities:

$$\begin{aligned} r_0 &\geq r_1 + r_2 \\ r_1 &\geq r_2 + r_3 \\ r_2 &\geq r_3 + r_4 \\ &\vdots \end{aligned}$$

Suppose that the algorithm ends after k iterations with $r_{k+1} = 0$, so r_k is the greatest common divisor sought. Then successive substitution in the chain of inequalities leads to

$$\begin{aligned} r_0 &\geq r_1 + r_2 \\ &\geq (r_2 + r_3) + r_2 = 2r_2 + r_3 \\ &\geq 2(r_3 + r_4) + r_3 = 3r_3 + 2r_4 \\ &\geq 3(r_4 + r_5) + 2r_4 = 5r_4 + 3r_5 \\ &\geq 5(r_5 + r_6) + 3r_5 = 8r_5 + 5r_6 \\ &\vdots \\ &\geq F_k r_k + F_{k-1} r_{k+1} = F_k r_k \end{aligned}$$

where F_0, F_1, F_2, $\ldots$ is the Fibonacci sequence $F_0 = 0$, $F_1 = 1$, $F_{i+1} = F_i + F_{i-1}$. Since r_k is the greatest common divisor, $r_k \geq 1$ gives

$$u = r_0 \geq F_k.$$

If u has n digits, then because $u = r_0$, we know

$$10^{n+1} > F_k.$$

We will show later this semester that

$$F_k \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k.$$

Thus

$$10^{n+1} > \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k$$

and taking logarithms and solving for k gives

$$\frac{n+1}{\log_{10}(\frac{1+\sqrt{5}}{2})} + \frac{\log_{10}\sqrt{5}}{\log_{10}(\frac{1+\sqrt{5}}{2})} > k$$

$$k \approx 4.785n,$$

so that k, the number of iterations, is at most approximately $4.785n$ for n-digit numbers.

Thus, the number of iterations executed in Euclid's algorithm grows at a rate *linear* in the number of digits of u and v, or, equivalently, *logarithmic* in their actual values.

Illinois Institute of Technology
Department of Computer Science

Lecture 4: August 28, 2013

CS 330 Discrete Structures
Fall Semester, 2013

This lecture presents some elementary techniques for counting the number of configurations satisfying specified properties. Such techniques are fundamental to the analysis of algorithms, especially algorithms for sorting, searching, merging, and other problems of a combinatorial nature.

For example, consider the algorithm listed below that finds the maximum of n numbers $x[1]$, $x[2]$, $\ldots$, $x[n]$. To analyze this algorithm fully we want to know exactly how many times each of the instructions is executed, but for the moment, let us ask only how many times the statement "$m \longleftarrow i$" is executed. The answer, of course, depends not on the particular values of the $x[i]$, but rather on their relative order: If $x[1]$ is the maximum of the n elements the statement will *never* be executed. If $x[1] < x[2] < \cdots < x[n]$ then the statement will be executed $n - 1$ times. For how many of the relative arrangements of the inputs $x[1], x[2], \ldots, x[n]$ will the statement be executed exactly k times? The answer to this question is at the heart of the analysis of the algorithm.

```
m ⟵ 1
FOR i := 2 TO n DO
    IF x[i] > x[m] THEN /* x[i] is the largest seen so far */
        m ⟵ i
```

The Rules of Sum and Product

In order to have as wide an applicability as possible, we will talk in terms of how many ways an "event" can occur. What is an event? That depends on the context. It could be "picking an orange, speckled sock from your sock drawer"; it could be "seating a group of five cannibals, three vegetarians, and a chicken at a table in such a way that nothing gets eaten" ; it could be the "choosing an arrangement of $x[1]$, $x[2]$, $\ldots$, $x[n]$ that causes the statement "$m \longleftarrow i$" in the algorithm above to be executed k times.

In counting the number of ways an event can happen, we try to break the event down into simpler events whose combination results in the event under study. For instance, if we are trying to determine the number of ways that the roll of a pair of dice can result in a seven, we could consider the roll of each die to be a simpler event than the roll of the pair. There are two basic rules for counting the number of ways that simpler events can be combined to result in more complex events:

Rule of Sum If an event E_1 can occur in e_1 different ways and a *separate* event E_2 can occur in e_2 different ways, the compound event $E_1 \cup E_2$ can occur in $e_1 + e_2$ different ways.

Rule of Product If an event E_1 can occur in e_1 different ways and a *separate* event E_2 can occur in e_2 different ways, the compound event $E_1 \cap E_2$ can occur in $e_1 e_2$ different ways.

The word "separate" in these two rules is very important, and we will define it precisely below. These two rules are deceptively simple and, as is often true with general principles, they are easy to understand but

sometimes tricky to apply. To illustrate the application of these rules, we will consider several examples designed to demonstrate them and their limitations.

Three Simple Examples

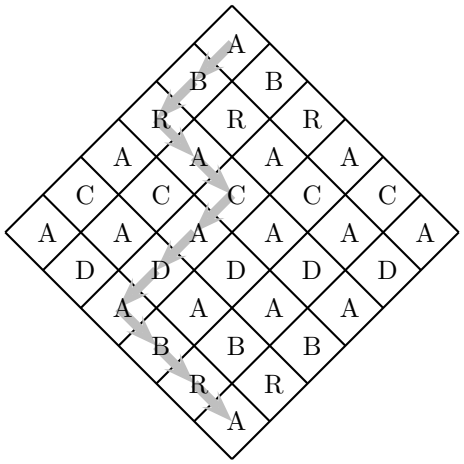
Suppose I have 5 short-sleeve shirts, 3 long-sleeve shirts, 4 pairs of pants, 7 ties, and 1 pair of shoes. Assuming that there is no issue of mis-matched colors or patterns, how many ways can I choose an ensemble each morning? For a shirt I can choose either a long- or a short-sleeve shirts ($5 + 3 = 8$ possible shirts), a pair of pants (4 possible pairs of pants), to wear a tie or not (7 possible ties or no tie at all, 8 choices in all), and only 1 pair of shoes. The total number of ensembles is thus $(5 + 3) \times 4 \times (7 + 1) \times 1 = 256$.

In a deck of cards we have 4 suits and 13 values (2 through ace), giving a total of $4 \times 13 = 52$ possible cards.

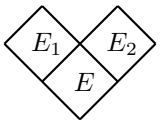
A zip code is made of 5 decimal digits, so the number of possible zip codes is $10 \times 10 \times 10 \times 10 \times 10 = 10^5 = 100000$. The number of zip codes with only odd digits is $5 \times 5 \times 5 \times 5 \times 5 = 10^5 = 3215$. The number of zipcodes with no repeated digit is $10 \times 9 \times 8 \times 7 \times 6 = 30240$.

A Bigger Example: Rule of Sum

The amulet picture below, consisting of 36 letters arranged on a 6×6 grid of diamond-shaped cells was said to have magical powers because any path of neighboring diamonds from the top "A" to the bottom "A" spells out the word "abracadabra"; furthermore, no other type of path will spell out this word. The obvious question arises: How many ways are there to spell out the sequence of letters A-B-R-A-C-A-D-A-B-R-A? The solution involves counting the number of ten-step paths in the diamond from top to bottom, with each step being "down to the left" or "down to the right", as illustrated by the arrows.



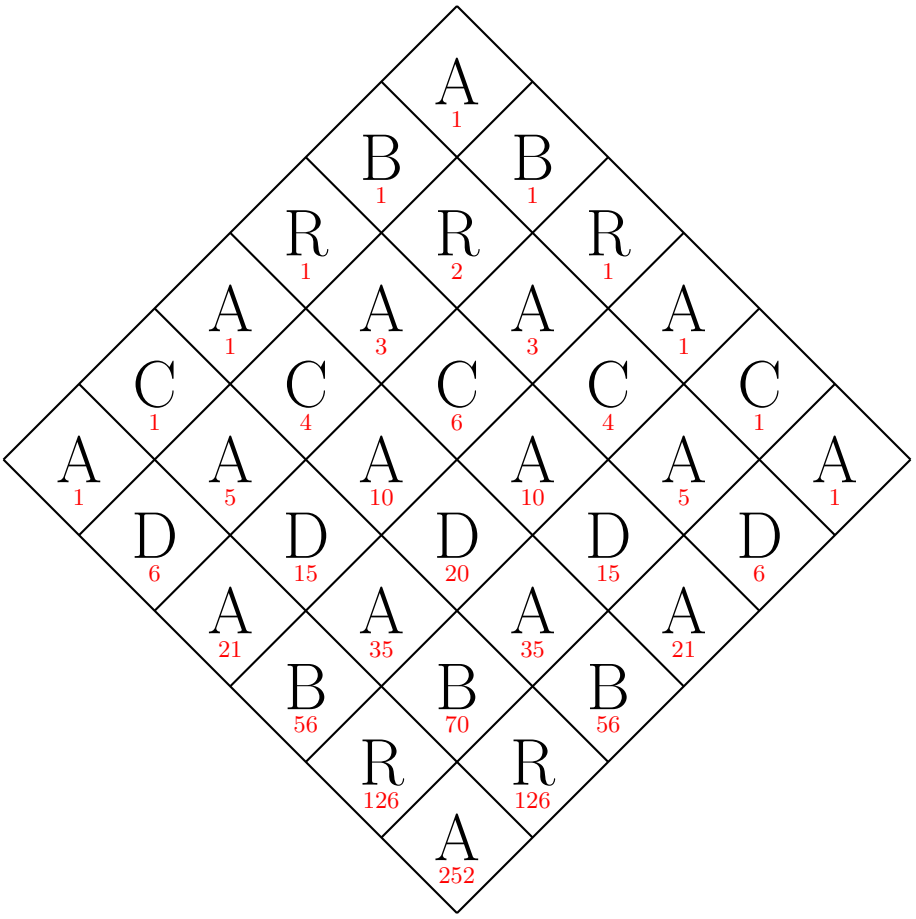
For the purpose of this example, an "event" is "arriving at a particular cell of the diamond from the top cell only by steps that are down-and-right or down-and-left." As the following diagram depicts, to reach an event E (labelled by the cell "E") we must either go through event E_1 or event E_2:



Suppose we can arrive at the cell labeled E_1 (this is the event E_1) in e_1 ways and at the cell E_2 (this is the event E_2) in e_2 ways. Then we can apply the rule of sums to see that event E can happen in exactly $e_1 + e_2$ ways. Note that there is only one way to arrive at a cell along the top left or top right boundaries. We can now compute the number of ways to spell "abracadabra" as follows:

- Fill in the 6×6 grid with ones along the top left and right boundaries
- apply the rule of sum by adding the two numbers in the cells above an empty cell
- write the resulting sum in the empty cell.

Our hard work pays off with the gleeful conclusion that "abracadabra" can be spelled out in 252 different ways. You might recognize the numbers in our diagram as the entries in Pascal's triangle.



For our second example, we choose an instance where the rule of sum is *not* applicable. Suppose that of the roughly 200 students in CS 330, 150 are taking Math 247, and 100 are taking Physics 113. How many of the CS 330 students have taken *either* Math 247 *or* Physics 113? Applying the rule of sum suggests 250 out of the 200 students in CS 330 have taken one course or the other! This nonsense results from applying the rule of sum to events that are *not* separate from one another—there are enterprising souls who are enrolled in both Math 247 and Physics 113, as well as some students who are currently taking neither. Thus, the

correct answer depends on how many of the students took both courses, not just either course. This tells us how to make the notion of "separate" events precise:

Events E_1 and E_2 are *separate* if $E_1 \cap E_2 = \emptyset$.

In other words, the rule of sum can be seen as a specific case (i.e. $E_1 \cap E_2 = \emptyset$) of the familiar principle of inclusion and exclusion:

$$|E_1 \cup E_2| = |E_1| + |E_2| - |E_1 \cap E_2|$$

Examples: Rule of Product

We now turn to an example that uses the rule of product. How many permutations (arrangements) of n distinct terms $x_1, x_2, \ldots, x_n$ are there in total? We reason as follows. The first element of the permutation can be any one of the n terms (n possibilities). The second element can be any one of the n elements *except* the term that was chosen as the first, giving $n - 1$ possibilities. The third element can be any one of the n elements *except* the terms that were chosen as the first two elements ($n - 2$ possibilities), and so on. Eachelement of the permutation, in order, corresponds to a separate event (think about why this is true), so that the total number of permutations is thus, by the rule of product, $n \times (n-1) \times (n-2) \times \cdots \times 1$, usually written $n!$. The symbol "$n!$" is read "n factorial."

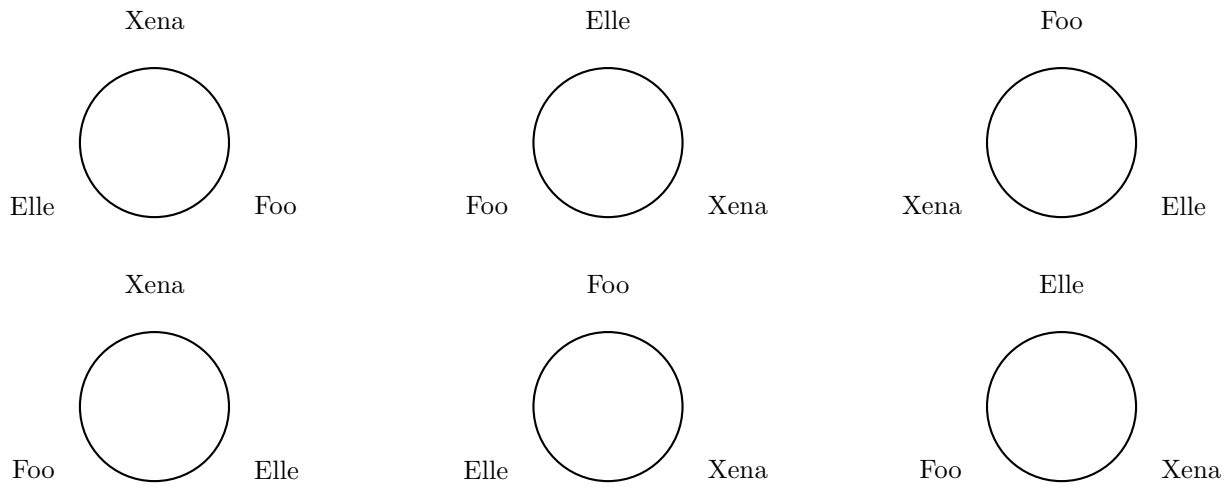
For example, there are $2! = 2 \times 1 = 2$ ways to arrange two distinct objects x_1 and x_2: either x_1, x_2 or x_2, x_1. There are $3! = 3 \times 2 \times 1 = 6$ ways to arrange the three letters A, E, T into a "word": the first letter can be either A,E,T (3 choices). Let us say that we picked A; then the second letter can be either E or T (2 choices). Let us say that we picked E; the the third letter can be only T (1 choice). Altogether, we can form the following six words: AET, ATE, EAT, ETA, TAE, TEA. There are $4! = 4 \times 3 \times 2 \times 1 = 24$ orders in which the courses of a four course meal can be served.

Another example where we may apply the rule of products is with subsets of a set. Specifically, how many ways are there to choose a subset of a set containing n elements? We can regard the choice of a subset as a sequence of n decisions (events) whether or not to include each of the n elements in the subset. Each of these events can happen in two ways—an element is included or is rejected. Thus the rule of product tells us that number of different compound events, each event being the choice of a subset, can happen in $2 \times 2 \times \cdots \times 2 = 2^n$ ways.

For example, if we look at the set { H, E, L, P }. The first event is whether or not we include H (two possibilities). The second event is whether or not we include E (also two possibilities), and so forth.

Variations of the Rule of Product

With a slight variation of the rule of product, we can solve many other interesting problems, such as the following vexing question: How many *different* ways are there to seat n people around a circular table for dinner? In this case, the table has no orientation, so rotating it does not generate a new seating. For three people, we can look at the $3! = 6$ arrangements of their names, say Xena, Elle, and Foo, around a circular table:



We see that the arrangements in the first row differ from each other only by a rotation of the table—Foo is always at Xena's left and Elle is always at Xena's right. Similarly, the arrangements in the second row differ from each other only by a rotation of the table. There are thus only two essentially different arrangements around the table:

In the general problem of n people (instead of 3), we answer the question by using an important variation on the rule of product.

Rule of Product (Variation) If an event E can occur in e different ways, and E is a compound event $E_1 \cap E_2$ in which E_1 and E_2 are separate events and E_1 can occur in e_1 different ways, then the event E_2 can occur in $e_2 = e/e_1$ different ways.

Let E be the event of seating n people in a row, which can happen in $e = n!$ ways, as we have seen. View E as a compound event $E_1 \cap E_2$ in which E_2 is the event of seating the n people in *different* ways (with respect to rotation) and E_1 is the event of starting at one of these people and reading off their neighbors clockwise around the circle. Since there are n places to start reading off neighbors, E_1 can happen in $e_1 = n$ ways. The variation on the rule of product then tells us that E_2, the event of seating the people in different circular arrangements (or "circular orders" as they are called), can happen in $e_2 = e/e_1 = n!/n = (n-1)!$ ways.

In essence, we have formed a correspondence between permutations and circular orders. Under this correspondence, each of the $n!$ permutations corresponds to a circular order, and n different permutations each give the same circular order.

How many circular arrangements are there of $n = 1$ elements? The answer is clearly one, and our formula

above gives $(1-1)! = 0!$. It is thus natural for us to define $0! = 1$, a slight extension to our formula

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1.$$

This extension is consistent with the notion of a unique "empty" permutation of zero elements.

With this extension, we can describe the value of $n!$ recursively:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ n \times (n-1)! & \text{otherwise.} \end{cases}$$

This recursive definition has a nice interpretation in terms of permutations. Suppose we want to list all of the permutations of n elements. We can do so recursively by starting with a list of all permutations of $n-1$ of the elements and then inserting the nth element into each of the n possible locations in each of the $(n-1)!$ permutations of $n-1$ elements.

We can ask a slightly different question about circular arrangements by considering n different colored beads on a loop of string (a "rosary"): In how many ways can the beads be made into a rosary? Now not only does the circular order need to be considered, but also the effect of flipping the loop, that is, taking its mirror image (which does not intrinsically change the loop). Again we use the variation on the rule of product. Let E be the event of forming a circular permutation, which we know can happen in $e = (n-1)!$ ways from the above discussion. View E as a compound event $E_1 \cap E_2$ in which E_2 is the event of forming a rosary permutation and E_1 is the event of choosing one of the two mirror images. Thus E_1 can happen in $e_1 = 2$ ways and so E_2 can happen in $e_2 = e/e_1 = (n-1)!/2$. The number of rosary permutations is thus $(n-1)!/2$.

Does this formula mean that for $n = 1$ and $n = 2$ there is half a rosary permutation? No, of course not! In our argument above we stated that E_1, the event of choosing one of the mirror images, can happen in $e_1 = 2$ ways. This is indeed true, but *only* when there are three or more beads—the two mirror images of the loop are identical if the loop contains only one or two beads. The complete answer to the question of the number of rosary permutations is thus

$$\begin{cases} (n-1)!/2 & \text{for } n \geq 3, \\ 1 & \text{otherwise.} \end{cases}$$

Food for thought:

The Problem of Derangements How many ways can you arrange n volumes of an encyclopedia in such a way that no volume is in its correct place (in the normal ordering of the volumes)?

The *Menage* problem You are have to seat n separated husband/wife couples (that's $2n$ people in total; bigamy is illegal) at your daughter's wedding. How many ways are there of seating these people around one table in such a way that no ex-couple is sitting in adjacent chairs?

Illinois Institute of Technology
Department of Computer Science

Lecture 5: September 4, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 How big is $n!$?

In the preceding material we have seen that $n!$ occurs in many contexts when counting arrangements of elements. Furthermore, in subsequent sections we will find that it figures centrally in most of the counting problems that we pursue. It is reasonable, then, to ask about the behavior of $n!$ as a function n. Specifically, how quickly does $n!$ grow as n becomes large? The answer to this question will give us asymptotic information about how the number of configurations grows in various cases; we will use such information in the next section, for example, to establish a benchmark for the performance of sorting algorithms.

Brief computation reveals that $n!$ grows fast as n increases:

n	0	1	2	3	4	5	6	7	8	9	10
$n!$	1	1	2	6	24	120	720	5,040	40,320	362,880	3,628,800

For example, if a code-cracking program requires the examination of each of the permutations of n items, the program will (most likely) be practical for $n \leq 7$, but will start to get expensive for $n = 8$, 9, 10 and become impossible to use for larger values of n, again, because $n!$ grows fast.

Exactly how does the growth rate of $n!$ compare with other functions of n? In the remainder of this section we will prove *Stirling's approximation* that will help us get a handle on this growth rate:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n, \tag{1}$$

where

$$e = \lim_{k\to\infty}\left(1+\frac{1}{k}\right)^k \approx 2.71828284590452354$$

is the base of the natural logorithms, and

$$\pi \approx 3.1415926535897932385$$

is the ratio of the circumference of a circle to its diameter, as usual.

Actually, we will establish only that $n!$ grows proportionately to $\sqrt{n}(n/e)^n$, without proving that the constant of proportionality is $\sqrt{2\pi}$, because proving things with $\sqrt{2\pi}$ in them usually involves some complex analysis which is "beyond the scope of this course." As an interesting side note, one may also ask how big are the Harmonic numbers H_n that we studied in previous lectures.

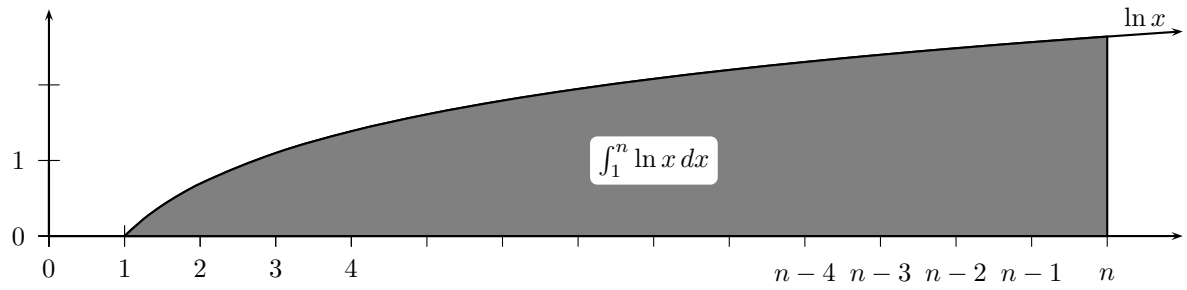
The technique we use to compute the approximate value of $n!$ is to transform the product to a sum by taking logarithms and then to estimate the size of the sum by comparing it to an integral. Taking logarithms gives

$$\begin{aligned}\ln n! &= \ln(1\times 2\times 3\times\cdots\times n)\\ &= \ln 1+\ln 2+\cdots+\ln n\\ &= \sum_{k=1}^{n}\ln k.\end{aligned}$$

Approximating the sum by an integral gives

$$\ln n! \approx \int_1^n \ln x \, dx$$

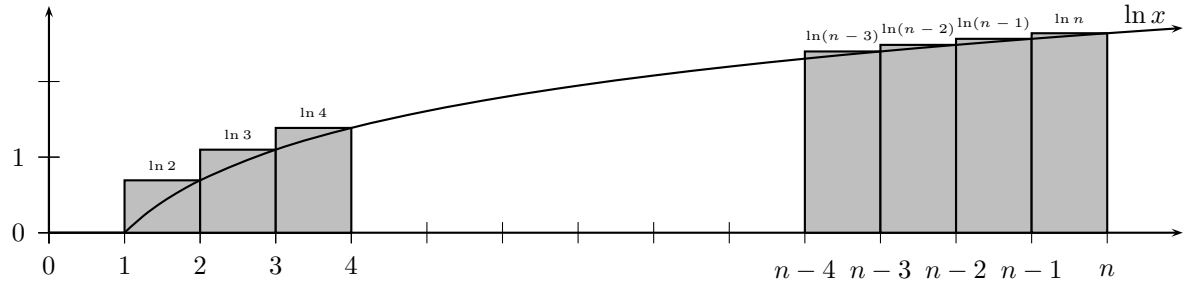
Consider $\int_1^n \ln x \, dx$; it can be taken to mean *the area under the curve* $f(x) = \ln x$ *from* $x = 1$ *to* n, or graphically:



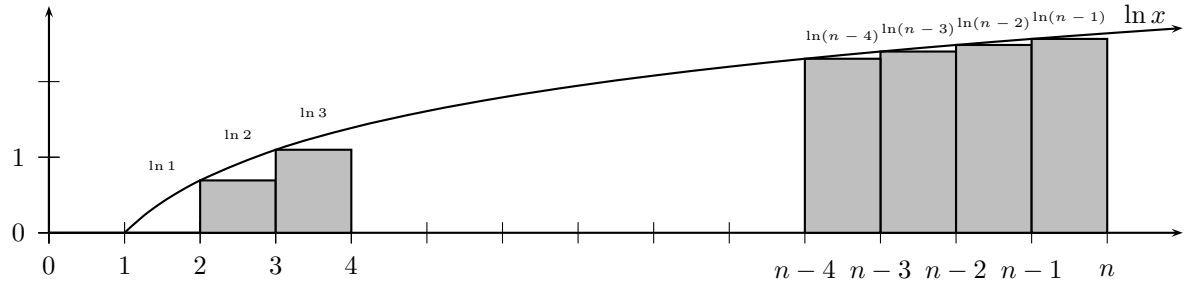
But if we split the area up into many rectangles, we notice that:

$$\int_1^n \ln x \, dx \leq \sum_{i=2}^{n} \ln i \tag{2}$$

Graphically, we see that the area under $f(x) = \ln x$ is *always* less than or equal to the area of the rectangles:



If we were to use rectangles below the curve, however, we see:



Which leads to the following inequality:

$$\int_1^n \ln x\, dx \quad \geq \quad \sum_{i=1}^{n-1} \ln i$$

Combining the two results we have:

$$\ln n + \int_1^n \ln x\, dx \quad \geq \quad \sum_{i=1}^{n} \ln i \geq \ln 1 + \int_1^n \ln x\, dx$$

Which can be reduced to:

$$\ln n + [x \ln x - x]_{x=1}^{n} \geq \ln n! \geq [x \ln x - x]_{x=1}^{n},$$

or

$$n \ln n - n + \ln n - 1 \geq \ln n! \geq n \ln n - n + 1,$$

so that

$$\ln n! = n \ln n - n + O(\log n).$$

an error margin of $O(\log n)$ in our approximation. However, this is much worse than Stirling's approximation which narrowed the margin to $O(1)$:

$$\begin{aligned} n! &\longrightarrow \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \\ \ln n! &\longrightarrow \frac{\ln 2\pi}{2} + \frac{\ln n}{2} + n \ln\left(\frac{n}{e}\right) \\ &= \frac{\ln 2\pi}{2} + \frac{\ln n}{2} + n(\ln n - 1) \\ &= \frac{\ln n}{2} + n \ln n - n + O(1) \end{aligned}$$

Thus, the approximation we showed with our rectangles was rather crude. We can, however, salvage our idea with a minor twist; if we use trapeziods to approximate the area rather than rectangles, we can get a much better approximation:

$$\ln n! = (n + \frac{1}{2}) \ln n - n + O(1) \tag{3}$$

Exercise Use (3) to determine the growth rate of $\log_2 \binom{2n}{n}$, where $\binom{2n}{n} = \frac{(2n)!}{n!n!}$.

2 How big is H_n

Let's try to approximate harmonic numbers using rectangles. Similar to the case of n!, we have

$$\int_1^n \frac{1}{x}\, dx \leq H_n - \frac{1}{n}$$

and

$$\int_1^n \frac{1}{x}\, dx \geq H_n - 1$$

Therefore we get

$$\int_1^n \frac{1}{x}\,dx + \frac{1}{n} \leq H_n \leq \int_1^n \frac{1}{x}\,dx + 1$$

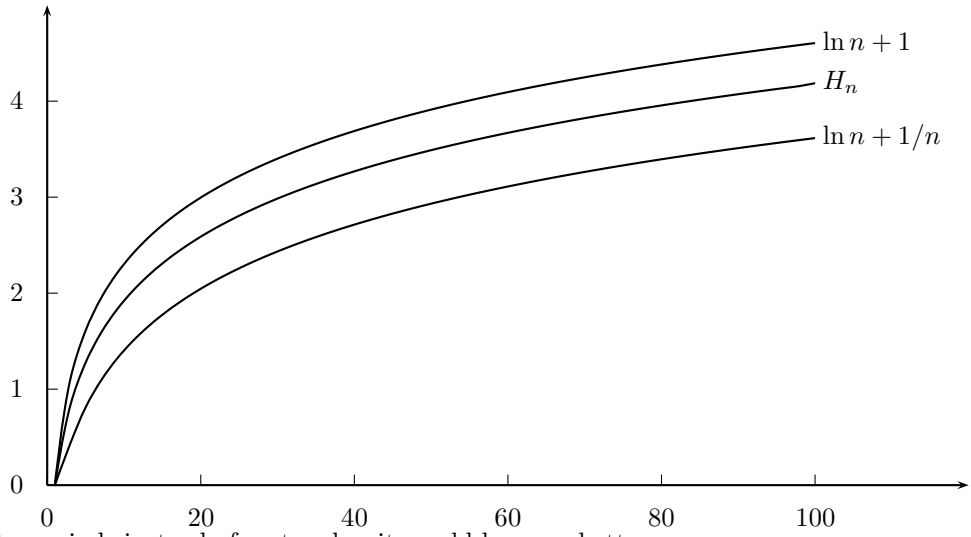
We know that

$$\int_1^n \frac{1}{x}\,dx = \ln n$$

So

$$\ln n + \frac{1}{n} \leq H_n \leq \ln n + 1$$

This is a good approximation as we can see:



If we use trapeziods instead of rectangles, it would be even better.

Exercise Use the ideas above to determine the growth rate of $\sum_{i=1}^{n} i^k$.

Illinois Institute of Technology
Department of Computer Science

Lecture 6: September 9, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Combinations

We have examined the question of how many ways there are to arrange (permute) n different items. Suppose instead we want to arrange only k of the n items, $k < n$. For example, if we have ten empty, single-bed hotel rooms numbered $1, 2, \ldots, 10$ and four guests arrive on a stormy night, in how many ways can we assign each guest to a room?

The first guest can be given any of the ten rooms; the second guest can be given any of the remaining nine rooms; the third guest can be given any of the remaining eight rooms; finally, the fourth guest can be given any of the remaining seven rooms. The rule of product tells us that there are

$$10 \times 9 \times 8 \times 7 = 5040$$

different ways to make the room assignments. In general, if there are n rooms and $k < n$ guests, the room assignments can be made in

$$n \times (n-1) \times (n-2) \times \cdots \times (n-k+1)$$

ways. This number can be rewritten conveniently using the factorial notation:

$$n \times (n-1) \times (n-2) \times \cdots \times (n-k+1) = \frac{n!}{(n-k)!}; \tag{1}$$

this is called the *number of permutations of n things taken k at a time*, and is sometimes denoted $P(n, k)$. The rewritten form $n!/(n-k)!$ is also valid for $k = n$ since it yields

$$\frac{n!}{(n-n)!} = \frac{n!}{0!} = n!$$

(by our definition that $0! = 1$). This agrees with our discovery that there are $n!$ permutations of n different items. Furthermore, when $k = 0$, $n!/(n-k)! = 1$, in agreement with our convention that there is a unique (empty) permutation of zero items. Rewriting the product in the form $n!/(n-k)!$ also suggests an alternative proof of the result based on the variation on the rule of product: Let E be the event of forming a permutation of all n items, viewed as a compound event $E = E_1 \cap E_2$ in which E_2 is the event of entering the first k of the n elements and E_1 is the event of arranging the remaining $n-k$ elements. The variation on the rule of product then tells us that E_2, that is forming a permutation of k of n items, can happen in $e_2 = e/e_1$. But, we know that $e = n!$ and $e_1 = (n-k)!$ since these are simply the numbers of permutations of n and $n-k$ items, respectively. It follows that $e_2 = n!/(n-k)!$.

Returning to the problem of how to assign guests to empty hotel rooms, let is consider the point of view of the hotel cleaning staff. As far as the cleaning staff is concerned, guests are indistinguishable from one another—the staff only cares about which rooms have been occupied and need cleaning. Thus, if there are n hotel rooms and k guests, we might want to ask: how many different arrangements of the k rooms might the cleaning staff be asked to clean? Again we use the variation on the rule of product. Let E be the event

of assigning k guests to k of n hotel rooms; we have seen from (1) that this can be done in $e = n!/(n-k)!$ ways. View E as a compound event $E = E_1 \cap E_2$ in which E_2 is the event of choosing which k of the n rooms will be occupied and E_1 is the event of arranging the k guests in the rooms to be occupied. The event E_1 can happen in $e_1 = k!$ ways since there are $k!$ different orders in which the guests can be assigned to occupied rooms. Thus event E_2, choosing k of n rooms, can occur in

$$e_2 = \frac{e}{e_1} = \frac{n!}{k!(n-k)!}$$

ways. In our example of 10 hotel rooms and 4 guests, the rooms to be occupied can be chosen in

$$\frac{10!}{4!(10-4)!} = \frac{10!}{6!}/4! = \frac{5040}{24} = 210$$

ways.

The value $\frac{n!}{k!(n-k)!}$ is so important and occurs so often in the solution of combinatorial problems (and hence in the analysis of algorithms for sorting, searching, and merging) that the shorthand notation

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

is almost always used. The symbol "$\binom{n}{k}$" is read "n choose k," since its value counts the number of ways to choose k items from a set of n items; it is also sometimes written $C(n,k)$ meaning the number of combinations of n elements, k taken at a time. In the remainder of this section we examine a few of the remarkable properties of the values $\binom{n}{k}$ and some applications. For reasons that will become clear in a few pages, the values $\binom{n}{k}$ are called *binomial coefficients.*

2 Pascal's Triangle

Binomial coefficients hide some very beautiful and powerful properties behind their simple form. We will now study some of these properties in the form of various identities.

The algebraic formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

is symmetric in k and $n-k$, suggesting the identity

$$\binom{n}{k} = \binom{n}{n-k}. \tag{2}$$

This identity has a simple *combinatorial interpretation*: To choose a k-element subset of a set with n elements we can either choose the k elements of the subset, or we can choose the $n-k$ elements not in the subset. In this case, an algebraic proof of (2) can be easily realized by substituting $n-k$ for k in the definition of $\binom{n}{k}$ and working through the arithmetic to see that we get the same expression; however, combinatorial interpretation of formulas is a basic technique that has wide application and is generally easier to realize than its algebraic counterpart; we will examine a number of instances of it in this section.

Viewing $\binom{n}{k}$ as the number of ways to choose k of n different objects, we can use the rule of sum to calculate a different powerful identity

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}. \tag{3}$$

To see that this identity is correct, let us call the n different objects be $O_1, O_2, \ldots, O_n$. We split the choice of k of these objects into two separate cases:

1. The object O_1 is one of the chosen k objects.
2. The object O_1 is *not* one of the chosen k objects.

In the first case, there remain $k-1$ objects to be chosen from among the $n-1$ objects $O_2, O_3, \ldots, O_n$; we know that this event can occur in $\binom{n-1}{k-1}$ different ways. In the second case, all k of the objects must be chosen from the remaining $n-1$ objects $O_2, O_3, \ldots, O_n$; this event can occur in $\binom{n-1}{k}$ different ways. Applying the rule of sum then proves the identity.

Equation (3) is, perhaps, the single most important identity satisfied by the binomial coefficients. We could also have proven it algebraically from definitions:

$$\begin{aligned}
\binom{n-1}{k-1} + \binom{n-1}{k} &= \frac{(n-1)!}{(k-1)![(n-1)-(k-1)]!} + \frac{(n-1)!}{k![(n-1)-k]!} \\
&= \frac{(n-1)!}{(k-1)!(n-1-k)!}\left[\frac{1}{n-k} + \frac{1}{k}\right] \\
&= \frac{(n-1)!}{(k-1)!(n-1-k)!}\left[\frac{n}{(n-k)k}\right] \\
&= \frac{n!}{k!(n-k)!} \\
&= \binom{n}{k}.
\end{aligned}$$

Equation (3) also allows us to calculate the value of $\binom{n}{k}$ without having to compute large factorials. Specifically, we calculate all the values of $\binom{n}{k}$ (for $k = 0, 1, 2, \ldots, n$) iteratively from the values of $\binom{n-1}{k}$ (for $k = 0, 1, 2, \ldots, n-1$). Such calculation can be organized into a visually pleasing structure known as Pascal's triangle:

					(k=0)						
($n=0$)					1		(k=1)				
($n=1$)				1		1		(k=2)			
($n=2$)			1		2		1		(k=3)		
($n=3$)		1		3		3		1		(k=4)	
($n=4$)	1		4		6		4		1		(k=5)
($n=5$)	1		5		10		10		5		1

Pascal's triangle is organized into rows that correspond to different values of n and right-to-left diagonals corresponding to different values of k. Thus, we can read off $\binom{4}{2} = 6$ which is in the $(n = 4)$ row and the $(k = 2)$ diagonal. Notice that the entry $\binom{0}{0} = 1$ has been added to the top of the triangle for completeness. We see that in solving the Abracadabra problem we were actually computing binomial coefficients.

Many fascinating identities on the binomial coefficients can be found by examining Pascal's triangle. For example, we see that if we sum across the nth row of the triangle, we get 2^n. Specifically, we can see this for the first few rows: $1 = 2^0$, $1 + 1 = 2^1$, $1 + 2 + 1 = 2^2$, $1 + 3 + 3 + 1 = 2^3$, and so on. This observation can be proved in several ways. An *algebraic proof* notices that each element in the $n - 1$st row is used twice in computing the elements of the nth row (we can think of imaginary rows of zeroes along the sides of the triangle, making this statement true for the ones also). In other words, the total across the nth row will be twice the total across the $n - 1$st row. Since the zeroth row has sum 2^0, the result follows.

A combinatorial argument is also possible. We want to prove that

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = 2^n. \tag{4}$$

We may consider the kth term on the left hand side to be the number of ways a subset of size k can be chosen from a set of size n. We may apply the rule of sum to note that the terms on the left hand side add to give all possible ways of picking a subset from a set of size n. We have shown earlier, by the rule of product, that there are 2^n different ways of picking a subset from a set of size n, and so the identity is proved.

Exercise Prove (4) by induction.

Another identity can be observed by summing along any diagonal from *upper left to lower to right*:

$$\sum_{i=0}^{k} \binom{n+i}{i} = \binom{n}{0} + \binom{n+1}{1} + \cdots + \binom{n+k}{k} = \binom{n+k+1}{k}; \tag{5}$$

for example, 1+3+6 = 10, 1+6+21+56+126 = 210, and so on. Again, two proofs are possible. An algebraic proof of equation (5) follows by mathematical induction from equation (3):

Base Case $k = 0$. Equation (5) is true from the definition that $\binom{n}{0} = \binom{n-1}{0} = 1$. **Inductive Hypothesis** Assume that equation (5) is true for k. **To show:** Equation (5) is true for $k + 1$:

$$\begin{aligned}
\sum_{i=0}^{k+1} \binom{n+i}{i} &= \sum_{i=0}^{k} \binom{n+i}{i} + \binom{n+k+1}{k+1}; \\
&= \binom{n+k+1}{k} + \binom{n+k+1}{k+1} \quad \text{(by induction)} \\
&= \binom{n+k+2}{k+1} \quad \text{[by equation (3)].}
\end{aligned}$$

Thus, the induction is complete.

The combinatorial proof of (5) observes that the right hand side of equation (5) is the number of ways to choose k of $n + k + 1$ distinct objects. Let those objects be O_1, O_2, ..., O_{n+k+1}. If we insist that object O_1 *not* be chosen, then all k elements must be chosen from the $n + k$ elements O_2, O_3, ..., O_{n+k+1}; this can be done in $\binom{n+k}{k}$ ways. On the other hand, if we insist that O_1 be chosen and O_2 *not* be chosen then the remaining $k - 1$ elements (aside from O_1) must be chosen from $O_3, O_4, \ldots, O_{n+k+1}$; this can be done in

$\binom{n+k-1}{k-1}$ ways. If we insist that both O_1 and O_2 be chosen, but not O_3, this can be done in $\binom{n+k-2}{k-2}$ ways. We finally arrive at the situation of insisting that O_1, O_2, ..., O_k all must be chosen; this can be done in $\binom{n}{0}$ ways. (Notice that we stop here, since the first element not chosen can not appear after O_{k+1}, in which case at least $k+1$ elements are chosen). Equation (5) now follows from the rule of sum. Note that to make this combinatorial proof rigorous, we would either have to formulate our left-hand side terms more carefully, or use induction.

To get Equation (5) we summed along diagonals from left to right. Using the left-right symmetry of Pascal's triangle that comes (algebraically) from equation (2) (i.e. we can flip the triangle around its middle) we see that if instead we sum along diagonals from right to left we get a similar identity, namely

$$\sum_{i=k}^{n} \binom{i}{k} = \binom{k}{k} + \binom{k+1}{k} + \cdots + \binom{n}{k} = \binom{n+1}{k+1}. \tag{6}$$

This identity can be proved algebraically by iterating equation (3). It can also be proven combinatorially; but most easily it can be proven by using equation (2) to transform each term of equation (5). It is convenient to define $\binom{i}{k} = 0$ for $k > i$ (i.e. there is *no* way to choose k distinct elements from a smaller set of i elements) and to extend the limits of summation in equation (6) to run from $i = 0$ to $i = n$. We obtain

$$\sum_{i=0}^{n} \binom{i}{k} = \binom{0}{k} + \binom{1}{k} + \cdots + \binom{n}{k} = \binom{n+1}{k+1} \tag{7}$$

Exercise Prove (7) by induction. Prove it by a combinatorial argument.

The special case $k = 1$ of equation (7) is

$$\begin{aligned} \binom{0}{1} + \binom{1}{1} + \binom{2}{1} + \cdots + \binom{n}{1} &= 0 + 1 + 2 + \cdots + n \\ \sum_{k=1}^{n} k &= \binom{n+1}{2} = \frac{n(n+1)}{2}. \end{aligned}$$

In a similar fashion, we can write

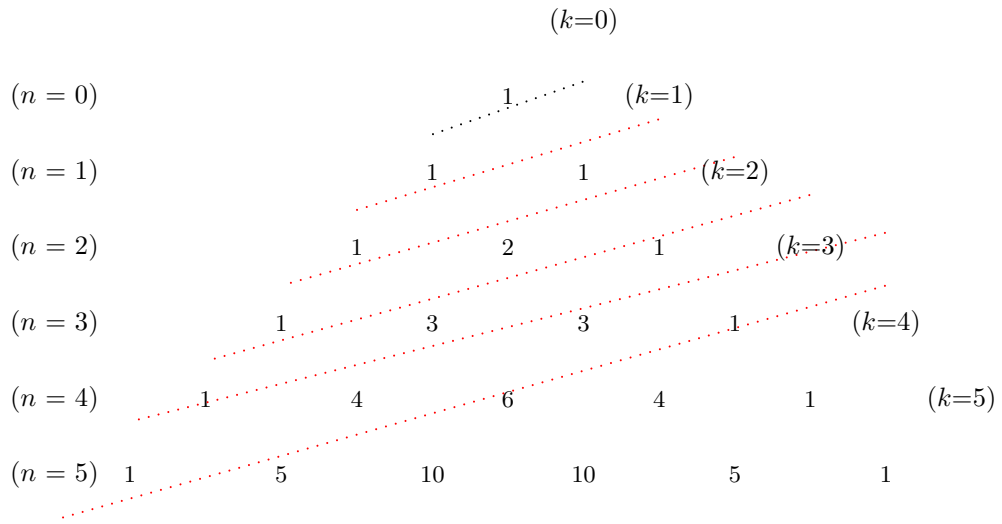
$$\begin{aligned} i^2 &= 2\binom{i}{2} + \binom{i}{1} \\ \sum_{i=0}^{n} i^2 &= \sum_{i=0}^{n} \left[2\binom{i}{2} + \binom{i}{1} \right] \\ &= 2\sum_{i=0}^{n} \binom{i}{2} + \sum_{i=0}^{n} \binom{i}{1} \end{aligned}$$

and use equation (7) to get:

$$\begin{aligned} \sum_{i=0}^{n} i^2 &= 2\binom{n+1}{3} + \binom{n+1}{2} \\ &= \frac{n(2n+1)(n+1)}{6} \end{aligned}$$

Exercise Find the sum $\sum_{i=0}^{n} i^3$ using this technique.

3 An observation on Pascal's triangle



By adding the numbers along diagonals as indicated by the above figure, we find the relation between binomial coefficients and Fibonacci numbers, $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$, $n > 1$:

$$\binom{0}{0} = 1 = F_1$$

$$\binom{1}{1} = 1 = F_2$$

$$\binom{2}{0} + \binom{1}{1} = 2 = F_3$$

$$\binom{3}{0} + \binom{2}{1} = 3 = F_4$$

$$\binom{4}{0} + \binom{3}{1} + \binom{2}{2} = 5 = F_5$$

$$\binom{5}{0} + \binom{4}{1} + \binom{3}{2} = 8 = F_6$$

...

Generally, we have

$$\sum_{k \geq 0} \binom{n-k}{k} = F_{n+1}$$

Exercise Prove this by induction.

Exercise Prove this by a combinatorial argument.

Illinois Institute of Technology
Department of Computer Science

Lectures 7–8: September 11–16, 2013

CS 330 Discrete Structures
Fall Semester, 2013

Another combinatorial identity

Let us examine the identity

$$\binom{n}{k}\binom{k}{r} = \binom{n}{r}\binom{n-r}{k-r} \tag{1}$$

An algebraic proof is simple, but the combinatorial proof is more interesting. As before, we demonstrate that the combinatorial problem solved by the expression on the left-hand side of the equal sign and the combinatorial problem solved by the expression on the right-hand side of the equal sign are actually the same problem approached in two different fashions.

The left-hand side of (1) counts the number of possible outcomes of a two stage selection process of a set R of r elements from n. First, a subset $K \subseteq R$ of k elements is chosen and then r of these k are selected to form R. The rule of product says that this choice of R can happen in $\binom{n}{k}\binom{k}{r}$ ways. We can count the number of ways the same event can happen by first directly choosing r of the n elements as R (this can be done in $\binom{n}{r}$ ways) and then choosing from the other $n-r$ elements the remaining $k-r$ elements which when added to R form K (this can be done in $\binom{n-r}{k-r}$ ways). By the rule of product the compound event can occur in $\binom{n}{r}\binom{n-r}{k-r}$ ways. Since the compound event is the same in both of these applications of the rule of product, our proof is complete.

Vandermonde's identity

In the identities established so far, the algebraic proof has been very easy, almost eliminating the need for a combinatorial proof. We now present two identities for which the combinatorial proof is relatively simple and direct, but algebraic verification is not. Vandermonde's identity states that

$$\begin{aligned}\binom{n+m}{k} &= \sum_{i=0}^{k}\binom{n}{i}\binom{m}{k-i} \\ &= \binom{n}{0}\binom{m}{k} + \binom{n}{1}\binom{m}{k-1} + \cdots + \binom{n}{k}\binom{m}{0}\end{aligned} \tag{2}$$

The left-hand side of (2) is the number of ways to select a subcommittee of k people from a committee of n men and m women. On the other hand, the right-hand side counts the number of outcomes for the same problem: If the subcommittee is to have i men and $k-i$ women, the rule of product says that it can be chosen in $\binom{n}{i}\binom{m}{k-i}$ ways. By the rule of sum we must add this value for $i = 0, 1, \ldots, k$ to count the number of ways the subcommittee can be chosen. This proves Vandermonde's identity.

A similar identity states

$$\binom{n+m}{n} = \sum_{i=0}^{n}\binom{n}{i}\binom{m}{i}$$

$$= \binom{n}{0}\binom{m}{0} + \binom{n}{1}\binom{m}{1} + \cdots + \binom{n}{n}\binom{m}{n} \tag{3}$$

Note that, by our convention that $\binom{i}{k} = 0$ for $k > i$, if $n > m$, the last $n - m$ terms of the sum will be zero. Again, an algebraic proof is not as simple as a combinatorial one here. The left-hand side of (3) is the number of ways to select a subcommittee of n people from a committee of n men and m women. The selection of such a committee can also be done, however, by first choosing i, $0 \le i \le n$, to be the number of women on the subcommittee, choosing the i women in one of the $\binom{m}{i}$ possible ways, and finally choosing the i men *not* on the subcommittee in $\binom{n}{i}$ ways—that is, the $n - i$ men *on* the committee in one of the $\binom{n}{n-i} = \binom{n}{i}$ possible ways. The rules of sum and product give the right-hand side of (3).
When $n = m$, (3) becomes the interesting identity

$$\sum_{i=0}^{n} \binom{n}{i}^2 = \binom{2n}{n}.$$

The Binomial Theorem

> *To this day I comprehend the binomial theorem, a very rare accomplishment in an author. For many years, indeed, I was probably the only American newspaper editor who knew what it was.*
>
> —H. L. Menken: *Predjudices: First, Second, and Third Series,* Library of America, 2010, page 452.

We turn now to one of the most important applications of the binomial coefficients, indeed, the justification of that name for the values $\binom{n}{k}$. We begin by applying the combinatorial reasoning developed so far to a purely algebraic problem, the evaluation of $(1 + x)^n$. Writing down the first few values we find

$$\begin{aligned}
(1+x)^0 &= 1 \\
(1+x)^1 &= 1 + x \\
(1+x)^2 &= 1 + 2x + x^2 \\
(1+x)^3 &= 1 + 3x + 3x^2 + x^3 \\
(1+x)^4 &= 1 + 4x + 6x^2 + 4x^3 + x^4 \\
&\vdots
\end{aligned}$$

The coefficients of the powers of x on the right-hand sides of this equation are a reproduction of Pascal's triangle. Why? Using the algebraic rules of polynomial multiplication, we can reason as follows. Let

$$(1+x)^{n-1} = \binom{n-1}{0} + \binom{n-1}{1}x + \binom{n-1}{2}x^2 + \cdots + \binom{n-1}{n-1}x^{n-1}.$$

Multiplying this by $(1 + x)$ involves adding $(1 + x)^{n-1}$ and $x(1 + x)^{n-1}$:

$$\begin{aligned}
(1+x)^{n-1} &= \binom{n-1}{0} + \binom{n-1}{1}x + \binom{n-1}{2}x^2 + \cdots + \binom{n-1}{n-1}x^{n-1} \\
x(1+x)^{n-1} &= \binom{n-1}{0}x + \binom{n-1}{1}x^2 + \cdots + \binom{n-1}{n-1}x^{n-1} + \binom{n-1}{n-1}x^n
\end{aligned}$$

which gives

$$\begin{aligned}(1+x)^n &= \binom{n-1}{0} + \left[\binom{n-1}{1} + \binom{n-1}{0}\right] x + \left[\binom{n-1}{2} + \binom{n-1}{1}\right] x^2 + \cdots \\ &\quad + \left[\binom{n-1}{n-1} + \binom{n-1}{n-2}\right] x^{n-1} + \binom{n-1}{n-1} x^n \\ &= \binom{n}{0} + \binom{n}{1} x + \binom{n}{2} x^2 + \cdots + \binom{n}{n} x^n\end{aligned}$$

Thus the computation of $(1+x)^n$ is called the *binomial theorem*:

$$\begin{aligned}(1+x)^n &= \sum_{k=0}^{n} \binom{n}{k} x^k \\ &= \binom{n}{0} + \binom{n}{1} x + \binom{n}{2} x^2 + \cdots + \binom{n}{n} x^n, \end{aligned} \tag{4}$$

which we just proved by induction. Now, as before, we concentrate on a combinatorial argument. We ask what the coefficient of x^k is in the simplified product

$$(1+x)^n = \underbrace{(1+x)(1+x)\cdots(1+x)}_{n \text{ times}}$$

If we expand the product into the unsimplified sum of all the monomials (products of xs and 1s), we can then ask how many of those monomials will be x^k; that number will be the coefficient of x^k in $(1+x)^n$. We obtain the monomial x^k for each term in the unsimplified product formed by having k of the factors $(1+x)$ contribute an x and the other $n-k$ contribute a 1. Since there are n factors, this can happen in $\binom{n}{k}$ ways, verifying (4).

Equation (4) has many interesting consequences. For example, setting $x = 1$ gives

$$\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = 2^n. \tag{5}$$

yet another way. Setting $x = -1$ gives

$$0 = (1-1)^n = \binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \binom{n}{3} + \cdots, \tag{6}$$

The equation (from the previous lecture)

$$\sum_{i=0}^{n} \binom{i}{k} = \binom{0}{k} + \binom{1}{k} + \cdots + \binom{n}{k} = \binom{n+1}{k+1} \tag{7}$$

can also be proved from (4) by a slightly more elaborate (and interesting) argument. We want to compute the value of

$$\sum_{i=0}^{n} \binom{i}{k}.$$

Now, $\binom{i}{k}$ is the coefficient x^k in $(1+x)^i$, so that the sum to be evaluated must be the coefficient of x^k in

$$\sum_{i=0}^{n}(1+x)^i = \frac{(1+x)^{n+1}-1}{(1+x)-1} = \frac{(1+x)^{n+1}-1}{x}$$

by the formula for the sum of a geometric progression.[1] This coefficient is the coefficient of x^{k+1} in $(1+x)^{n+1}-1$, which is $\binom{n+1}{k+1}$ by the binomial theorem, establishing (7).

Applications of the binomial theorem

Equation (4) is useful when we are faced with the evaluation of a sum of binomial coefficients, because it allows us to transform such a sum into a sum of terms in a geometric progression.

Consider

$$\sum_{i=0}^{k}\binom{n+i}{i} \tag{8}$$

that we talked about in the previous lecture. We know that

$$\begin{aligned}\binom{n+i}{i} &= \text{coefficient of } x^i \text{ in } (1+x)^{n+i} \\ &= \text{coefficient of } x^n \text{ in } (1+x)^{n+i}x^{n-i}.\end{aligned}$$

Therefore,

$$\sum_{i=0}^{k}\binom{n+i}{i} = \text{coefficient of } x^n \text{ in } \textstyle\sum_{i=0}^{k}(1+x)^{n+i}x^{n-i}$$

and

$$\begin{aligned}\sum_{i=0}^{k}(1+x)^{n+i}x^{n-i} &= (1+x)^n x^n \sum_{i=0}^{k}(1+x)^i x^{-i} \\ &= (1+x)^n x^n \sum_{i=0}^{k}\left(\frac{1+x}{x}\right)^i \\ &= (1+x)^n x^n \frac{\left(\frac{1+x}{x}\right)^{k+1}-1}{\frac{1+x}{x}-1}\end{aligned}$$

Simplifying this polynomial, we have

$$\begin{aligned}&= (1+x)^n x^n \frac{\frac{(1+x)^{k+1}}{x^{k+1}}-1}{1/x} \\ &= (1+x)^n x^{n+1}\left[\frac{(1+x)^{k+1}-x^{k+1}}{x^{k+1}}\right] \\ &= (1+x)^n x^{n-k}\left[(1+x)^{k+1}-x^{k+1}\right] \\ &= (1+x)^{n+k+1}x^{n-k} - (1+x)^n x^{n+1}.\end{aligned}$$

[1] By long division of polynomials, $\frac{r^{n+1}-1}{r-1} = 1 + r + r^2 + \cdots + r^n$.

Our sum is the coefficient of x^n in this sum. But $(1+x)^n x^{n+1}$ has no term x^n, so our sum is

$$\begin{aligned}\sum_{i=0}^{k}\binom{n+i}{i} &= \text{coefficient of } x^n \text{ in } (1+x)^{n+k+1}x^{n-k}\\ &= \text{coefficient of } x^k \text{ in } (1+x)^{n+k+1}\\ &= \binom{n+k+1}{k},\end{aligned}$$

as we already knew. Other such summations require the more complicated manipulations, including the use of differentiation or integration.

Another example: What is $\sum_{k=0}^{n} k\binom{n}{k}$? We reason as follows. $\binom{n}{k}$ is the coefficient of x^k in $(1+x)^n$, so that $k\binom{n}{k}$ is the coefficient of x^k in $d(1+x)^n/dx = n(1+x)^{n-1}$. Thus $\sum_{k=0}^{n} k\binom{n}{k} = n2^{n-1}$ because setting $x=1$ sums the coefficients. We can see the result as the number of ways to select a committee of size k from a group of n, *and specify a chairperson.* How?

Exercise What is $\sum_{k=0}^{n}\binom{n}{k}/k$?

More applications of the binomial theorem

As presented so far, the binomial theorem applies only to nonnegative integer powers of $(1+x)$. A much more general version can be derived by elementary calculus:

$$\begin{aligned}(1+x)^t &= 1 + tx + \frac{t(t-1)}{2!}x^2 + \frac{t(t-1)(t-2)}{3!}x^3\\ &\quad + \cdots + \frac{t(t-1)(t-2)\cdots(t-k+1)}{k!}x^k + \cdots.\end{aligned} \tag{9}$$

This formula, which is the Taylor series expansion of $(1+x)^t$ around $x=0$, is exactly equation (5) of last lecture, where t is a nonnegative integer. When t is negative or noninteger, the right-hand side of (9) is an infinite series that can be shown to converge for $|x|<1$. Equation (9) suggests the generalization of the symbol $\binom{t}{k}$ to nonpositive or noninteger values of t:

$$\binom{t}{k} = \begin{cases} 1 & \text{if } k=0,\\ t(t-1)(t-2)\cdots\frac{(t-k+1)}{k!} & \text{if } k>0.\end{cases}$$

This allows us to write (9) more tersely as

$$(1+x)^t = \sum_{k=0}^{\infty}\binom{t}{k}x^k. \tag{10}$$

Notice that (10) includes (4) as a special case, since $\binom{n}{k}=0$ for $k>n$ and integer n.

Of special interest is the case

$$(1-x)^{-n} = \sum_{k=0}^{\infty}\binom{-n}{k}(-x)^k$$

where n is an integer.

By the definition of $\binom{t}{k}$, we have

$$\begin{aligned}\binom{-n}{k} &= \frac{(-n)(-n-1)(-n-2)\cdots(-n-k+1)}{k!} \\ &= \frac{n(n+1)(n+2)\cdots(n+k-1)}{k!}(-1)^k \\ &= \binom{n+k-1}{k}(-1)^k\end{aligned}$$

This gives

$$\begin{aligned}(1-x)^{-n} &= \sum_{k=0}^{\infty}\binom{n+k-1}{k}(-1)^k(-x)^k \\ &= \sum_{k=0}^{\infty}\binom{n+k-1}{k}x^k \qquad (11)\end{aligned}$$

Equation (11) has a useful combinatorial interpretation. Rewriting $(1-x)^{-n} = (1+x+x^2+x^3+\cdots)^n$ by using the formula for the sum of a geometric progression, we obtain

$$\underbrace{(1+x+x^2+\cdots)(1+x+x^2+\cdots)\cdots(1+x+x^2+\cdots)}_{n \text{ times}} = \sum_{k=0}^{\infty}\binom{n+k-1}{k}x^k$$

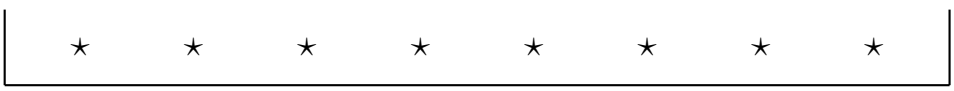
The coefficient of x^k on the left-hand side (which must, of course be the same as that on the right-hand side) is the number of ways to choose k objects from a set of n objects *with unlimited repetition*; that is, a single object can be chosen 0, 1, 2, ..., or k times. Why? Examine the way a term is formed in the unsimplified product on the left-hand side: it is a power of x from the first sum times a power of x from the second sum times a power of x from the third sum and so on. If x^k is to be formed in this way, the sum of the exponents in the powers of x must equal k. Such a product of powers $x^{i_1}x^{i_2}\cdots x^{i_n}$ equaling x^k thus corresponds to a selection of k objects as follows: i_1 of the first object, i_2 of the second object, ..., i_n of the nth object, for a total of $i_1+i_2+\cdots+i_n = k$ objects. It follows that the number of ways that x^k can appear in the unsimplified product is the number of choices of k objects from n objects with unlimited repetition. Equation (11) tells us that this is $\binom{n+k-1}{k}$.

Why is there an "$n-1$" in the above result? Let us think about it in another way. We want to know the number of ways to choose k objects from a set of n objects with unlimited repetition. Let $k=8$ and $n=5$. Suppose we have stars in 5 different colors, namely red, green, blue, yellow, and cyan, each with unlimitted number. How many ways are there to choose 8 stars from them?

We can do this with "stars and bars counting" (Rosen, page 425); see also

`http://en.wikipedia.org/wiki/Stars_and_bars_%28combinatorics%29`

We can pick the stars like this: first, we put $k=8$ stars in a long box.



Then, we insert $n-1=4$ boundary bars into the long box, which will seperate the long box into 5 smaller boxes. If there are m stars in the ith box, we will pick m stars with the ith color (Notice that m can be

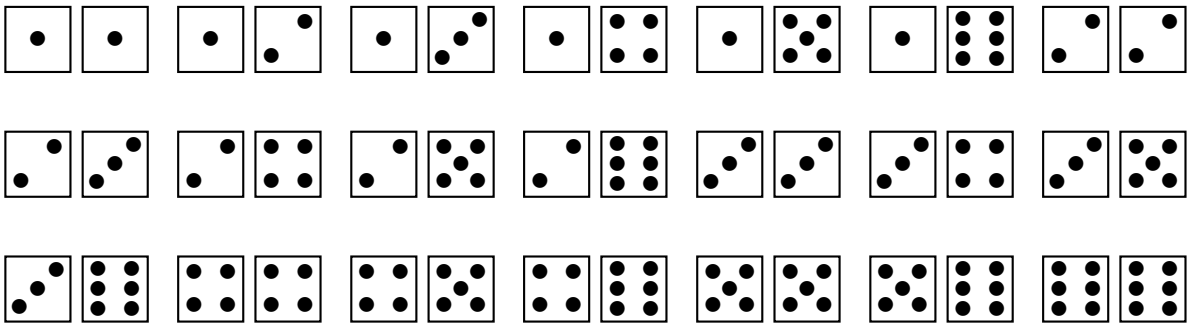
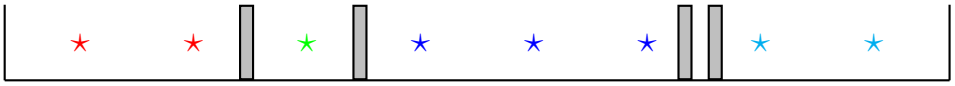


Figure 1: All possible pairs of dice.

0). In the case shown below, we will pick 2 red stars, 1 green star, 3 blue stars, no yellow stars, and 2 cyan stars.



Imagine that the long box has $n-1+k$ slots. In each of the slots we can put either a star or a bar. We just need to choose $n-1$ of them to put the bars, or k of them to put the stars. Therefore, the number of ways to pick the stars should be

$$\binom{n+k-1}{n-1} = \binom{n+k-1}{k}$$

Let's look at another example: how many outcomes are possible if m standard dice are rolled? Each die has 6 faces that could be up, so we are choosing m faces (one for each die) from 6 possibilities, with repetition. Our discussion above showed that there are

$$\binom{6+m-1}{m} = \binom{m+5}{m} = \binom{m+5}{5}$$

ways to make the choice. For one die, there are just the

$$\binom{1+5}{5} = \binom{6}{5} = 6$$

obvious outcomes. For two dice there are the

$$\binom{2+5}{5} = \binom{7}{5} = \frac{7!}{5!2!} = 21$$

outcomes (shown in Figure 1).

As a final example of the binomial theorem, we compute the coefficients in the binomial expansion of square roots:

$$(1+x)^{\frac{1}{2}} = \sum_{k=0}^{\infty} \binom{\frac{1}{2}}{k} x^k,$$

where

$$\binom{\frac{1}{2}}{k} = \begin{cases} 1 & k = 0, \\ \frac{\frac{1}{2}(\frac{1}{2}-1)(\frac{1}{2}-2)\cdots(\frac{1}{2}-k+1)}{k!} & k = 1, 2, 3, \ldots \end{cases}$$

For $k = 1, 2, 3, \ldots$

$$\begin{aligned}
\binom{\frac{1}{2}}{k} &= \frac{\frac{1}{2}\left(-\frac{1}{2}\right)\left(-\frac{3}{2}\right)\left(-\frac{5}{2}\right)\cdots\left(-\frac{2k-3}{2}\right)}{k!} \\
&= \left(\frac{1}{2}\right)^k \frac{(-1)(-3)(-5)\cdots(-2k+3)}{k!} \\
&= (-1)^{k-1}\frac{1\times 3\times 5\cdots(2k-3)}{2^k k!} \\
&= (-1)^{k-1}\frac{1\times 2\times 3\times 4\times 5\cdots(2k-3)(2k-2)}{[2^k k!][2\times 4\times 6\cdots(2k-2)]} \\
&= (-1)^{k-1}\frac{(2k-2)!}{[2^k k!][2^{k-1}(k-1)!]} \\
&= \frac{(-1)^{k-1}}{2^{2k-1}}\frac{(2k-2)!}{k!(k-1)!} \\
&= \frac{(-1)^{k-1}}{2^{2k-1}}\frac{1}{k}\binom{2k-2}{k-1}.
\end{aligned}$$

Thus,

$$(1+x)^{\frac{1}{2}} = 1 + \frac{1}{2}x - \frac{1}{2^3}\frac{1}{2}\binom{2}{1}x^2 + \frac{1}{2^5}\frac{1}{3}\binom{4}{2}x^3 - \cdots. \tag{12}$$

Even this equation has an interesting combinatorial significance.

Exercise What is the coefficient of x^n in $C(x) = 1/\sqrt{1-4x}$? How about in $2C'(x/4)$?

Illinois Institute of Technology
Department of Computer Science

Lecture 9: September 23, 2013

CS 330 Discrete Structures
Spring Semester, 2013

Polynomial evaluation: how complexity depends on the method

As part of the development of a polynomial approximation to the arctangent function (such as would be required in most compilers), a first-year graduate student in Computer Science at Cornell University[1] was asked to write a program to compute the coefficients of a polynomial given its roots. In other words, values $r_1, r_2, \ldots, r_n$ were given, and it was necessary to compute the coefficients $a_0, a_1, \ldots, a_{n-1}$ in the nth degree polynomial

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} + x^n = (x - r_1)(x - r_2)\cdots(x - r_n).$$

The student remembered, from a high-school course in algebra, that

$$a_k \quad = \quad (-1)^{n-k} \cdot \left[\begin{array}{l}\text{the sum of all } \binom{n}{n-k} \text{ possible prod-}\\ \text{ucts of } n-k \text{ of the roots}\end{array}\right]. \tag{1}$$

We can prove (1) with the kind of argument we used to prove the binomial theorem: We ask how a term x^k can be formed in the unsimplified product $(x - r_1)(x - r_2)\cdots(x - r_n)$. Such a term can occur if k of the n terms contribute their x-term, and the remaining $n - k$ contribute their r-term; there are, of course $\binom{n}{n-k}$ ways to choose the terms that contribute their r-term.

Having come up with a correct algorithm for the problem he faced, the student rushed ahead to write the program implementing the algorithm suggested by equation (1). His behavior was typical—the algorithm is correct and relatively easy to implement, so no thought was given to the *quality* of the algorithm. The program worked beautifully on the various polynomials used as test cases, but when applied to the twelfth and higher degree polynomials for which it was expressly written, it used unreasonable amounts of time. After frittering away much of his time allocation[2] in this manner, he turned, in desperation, to a professor who specialized in combinatorial analysis and numerical analysis[3] for help. The help he received was the following analysis of what he was doing.

How much work was involved in computing a_0 by Equation (1)? In this case (1) simplifies to

$$a_0 = (-1)^n r_1 r_2 \cdots r_n$$

(why?), so that $n - 1$ multiplications of the roots are required. For a_1,

$$a_1 = (-1)^{n-1}[r_2 r_3 \cdots r_n + r_1 r_3 r_4 \cdots r_n + \cdots + r_1 r_2 \cdots r_{n-1}];$$

this has $\binom{n}{1}$ terms, each of which requires $n - 2$ multiplications of roots (we ignore multiplying by $(-1)^{n-1}$ because this simply switches the sign). Adding these $\binom{n}{1}$ terms requires $\binom{n}{1} - 1$ additions. Similarly, computing a_{n-1} involves adding $\binom{n}{n-1}$ terms, each of which is simply one of the r_j.

[1] Professor Reingold, as it happens.

[2] Five minutes per semester. Punch cards were issued to the machine operator once in the morning and once in the afternoon, and he fed them to the computer, located off-campus near the airport. He would return printouts of the output at his next trip to campus.

[3] Professor Robert J. Walker.

In general, computing a_i requires adding $\binom{n}{i}$ terms, each of which is a product of $n-i$ roots; in other words, $\binom{n}{i}-1$ additions and $\binom{n}{i}(n-i-1)$ multiplications are needed to get the value of a_i. In total then, the computation of all of the n coefficients requires

$$\text{Additions:} \quad \sum_{i=0}^{n-1}\left[\binom{n}{i}-1\right]$$

$$\text{Multiplications:} \quad \sum_{i=0}^{n-1}\binom{n}{i}(n-i-1)$$

In order to analyze these better, we must first figure out how to evaluate:

$$\sum_{k=0}^{n} k\binom{n}{k}$$

One way to do this is with the binomial theorem:

$$(1+x)^n = \sum_{k=0}^{n}\binom{n}{k}x^k$$

Taking derivatives of both sides with respect to x gives us:

$$n(1+x)^{n-1} = \sum_{k=0}^{n} k\binom{n}{k}x^{k-1}$$

Setting $x=1$ we see that:

$$n2^{n-1} = \sum_{k=0}^{n} k\binom{n}{k}$$

We may also compute this sum combinatorially. Consider the question of choosing a committee of k people from among n of them, and then electing a chairperson for this committee. There are $\binom{n}{k}$ different ways of choosing the committee, and k ways of electing its chair giving $\sum_{k=0}^{n} k\binom{n}{k}$ total ways to choose any committee with a chair. This is the left- hand side of the desired identity. Another way to count such committees is to first choose the chairperson, which may be done in n ways, and, for each remaining person, decide individually whether or not that person will be in the committee, which may be done in 2^{n-1} ways by the rule of products, giving a total of $n2^{n-1}$ possibilities. This proves the identity.

We now have the tools to properly analyze the number of additions and multiplications needed for student's implementation of the program. The number of additions is:

$$\begin{aligned}\sum_{i=0}^{n-1}\left[\binom{n}{i}-1\right] &= \sum_{i=0}^{n-1}\left[\binom{n}{i}\right]-n \\ &= \sum_{i=0}^{n}\left[\binom{n}{i}\right]-(n+1) \\ &= 2^n-(n+1)\end{aligned}$$

The number of multiplications is:

$$\sum_{i=0}^{n-1}\binom{n}{i}(n-i-1) = \sum_{i=0}^{n-1}(n-1)\binom{n}{i}-\sum_{i=0}^{n-1} i\binom{n}{i}$$

$$\begin{aligned} &= (n-1)\sum_{i=0}^{n-1}\binom{n}{i} - \left(\sum_{i=0}^{n} i\binom{n}{i} - n\binom{n}{n}\right) \\ &= (n-1)(2^n-1) - n(2^{n-1}-1) \\ &= (n-2)2^{n-1} \ + \ 1 \end{aligned}$$

Using these formulas we can compute a table of how many arithmetic operations are used:

n	1	2	3	4	5
additions	0	1	4	11	26
multiplications	0	1	5	17	49

n	6	7	8	9	10
additions	57	120	247	502	1,013
multiplications	129	321	769	1,793	4,097

n	11	12	13	14	15
additions	2,036	4,083	8,178	16,369	32,752
multiplications	9,217	20,481	45,057	98,305	212,993

Almost a quarter of million arithmetic operations to get the coefficients of a fifteenth degree polynomial and this does not count auxiliary operations done in loop control, assignment statements, or initialization! At one microsecond per instruction, that is almost a quarter of second; when this process is used inside a loop, enormous amounts of computer time will be required. The truth is that these days, computers can easily handle this amount of computation, but if we then try to solve the problem with $n = 30$, we will square the number of operations to $45,366,018,049$...which no computer can handle efficiently.

How *should* the coefficients be computed? The answer is that simple multiplication of polynomials works efficiently. If we have already computed that

$$a_0 + a_1x + a_2x^2 + \cdots + a_{k-2}x^{k-2} + x^{k-1} = (x-r_1)(x-r_2)\cdots(x-r_{k-1}),$$

then

$$\begin{aligned} (x-r_1)(x-r_2)\cdots(x-r_k) &= (a_0 + a_1x + \cdots + a_{k-2}x^{k-2} + x^{k-1})(x-r_k) \\ &= -r_ka_0 + (a_0 - r_ka_1)x + (a_1 - r_ka_2)x^2 + \cdots + (a_{k-2} - r_k)x^{k-1} + x^k. \end{aligned}$$

Thus we can use the following simple code to compute the coefficients:

```
1: a[0] ← 1
2: for i ← 1 to n do
3:    a[i] ← 0
4: end for
5: for k ← 1 to n do
6:    {at this point we have computed (x − r_1)(x − r_2)···(x − r_{k−1}) =}
7:    {a_0 + a_1 x + ··· + a_{k−2} x^{k−2} + x^{k−1}; we now multiply this by (x − r_k).}
8:    a[k] ← 1
9:    for i ← k − 1 to 1 by −1 do
10:      a[i] ← a[i − 1] − r_k × a[i]
11:   end for
12:   a[0] ← −r_k × a[0]
13: end for
```

Exercise Use the comment in lines 6–7 to prove by induction that the loop computes the coefficients correctly.

Have we made any improvement over the algorithm based on direct calculation from equation (1)? The inner loop executes statement 10 a total of

$$\sum_{k=1}^{n}\sum_{i=1}^{k-1} 1 = \sum_{k=1}^{n}(k-1) = \frac{n(n-1)}{2}$$

times. Statement 12 is executed n times. Thus the algorithm requires $n(n-1)/2$ subtractions and $n(n+1)/2$ multiplications. To compare this with the previous algorithm we note that a subtraction requires the same amount of time as an addition and that in neither case have we counted auxiliary operations such as loop control, assignment statements, initialization, and so on (these will be similar in both cases). Computing a table similar to the one before we find

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
subtractions	0	1	3	6	10	15	21	28	36	45	55	66	78	91	105
additions	1	3	6	10	15	21	28	36	45	55	66	78	91	105	120

Our better algorithm is thus more efficient for $n \geq 4$. Specifically, for a polynomial of degree fifteen, it requires only 0.09% of the time required by the other algorithm. More important, however, is the fact that the new algorithm requires time proportional to n^2 as n gets large, while the first algorithm requires time proportional to $n2^n$, an enormous difference!

Illinois Institute of Technology
Department of Computer Science

Lecture 10: September 25, 2013

CS 330 Discrete Structures
Spring Semester, 2013

"I think you're begging the question," said Haydock, "and I can see looming ahead one of those terrible exercises in probability where six men have white hats and six men have black hats and you have to work it out by mathematics how likely it is that the hats will get mixed up and in what proportion. If you start thinking about things like that, you would go round the bend. Let me assure you of that!"

—Agatha Christie: *The Mirror Crack'd* (1962)

Misunderstanding of probability may be the greatest of all impediments to scientific literacy.

—Stephen Jay Gould : *Dinosaur in a Haystack: Reflections on Natural History* (1996)

I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be.

—Lord Kelvin (William Thomson) (1883)

1 Probability

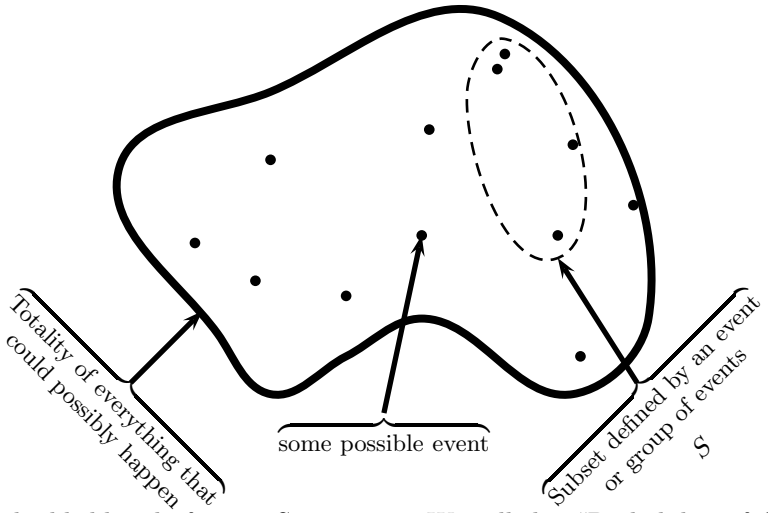
When the weather-woman says "there is a 30% chance of rain", what does she mean? Does she mean that:

- rain will fall on 30% of the viewing area?
- in the last 100 years, it rained 30 times on this date?
- under present conditions, recorded history for comparable conditions shows it rained 30% of the time?

Similarly, what does it mean for an algorithm to be correct 99% of the time? Can we trust it to guard our nuclear warheads?
In order to answer these questions, we first need to know a little about **probability**.

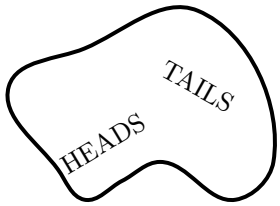
2 Basics of Probability

Consider the following amoeba-like graph:



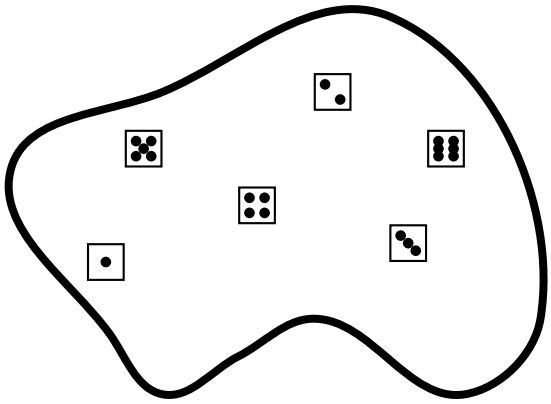
We wish to measure the likelihood of event S occurring. We call this "Probability of S" and we write it as $\mathbf{Pr}(S)$.

Consider the following experiment: We will flip a "fair" coin. We have the following "universe" of outcomes:



Our intuition tells us that $\mathbf{Pr}(\text{TAILS}) = 1/2$ because TAILS happens about half the time.

Consider this experiment: We will roll a "fair" die. We have the following "universe" of outcomes:



Our intuition tells us that $\mathbf{Pr}(\text{Rolling a one}) = 1/6$, because rolling a "one" occurs about one sixth of the time.

From these we can see a simple definition of Probability as:

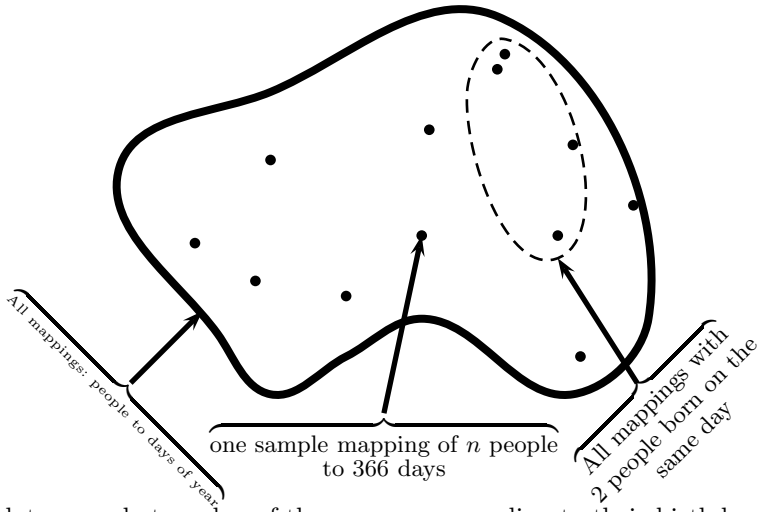
$$\mathbf{Pr}(S) = \frac{\text{number of events in } S}{\text{number of events altogether}}$$

This definition will be ample for our purposes.

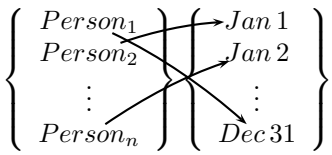
3 The Birthday Problem

An illustration of the power of probability is the Birthday problem: If we have n people in a room, what is the probability that two people celebrate their birthday on exactly the same day?

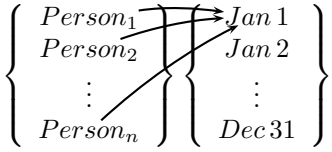
After a careful examination, you will notice that the set we have defined is:



A mapping simply relates people to a day of the year corresponding to their birthday. For instance



and



are mappings.

There are 366^n ways to map n people to birthdays. There are $\frac{366!}{(366-n)!}$ ways to pick the birthdays so that no two people share the same birthday. Using our definition of probability:

$$\mathbf{Pr}(\text{No two birthdays on the same day}) = \frac{\frac{366!}{(366-n)!}}{366^n} = 1(1-\frac{1}{366})(1-\frac{2}{366})\cdots(1-\frac{n-1}{366})$$

Here we can see how this probability decreases when n grows:

n	1	2	3	4	5	6	7	8	9	10
$\frac{\frac{366!}{(366-n)!}}{366^n}$	1	0.9973	0.9918	0.9837	0.9729	0.9596	0.9439	0.9259	0.9056	0.8834
n	11	12	13	14	15	16	17	18	19	20
$\frac{\frac{366!}{(366-n)!}}{366^n}$	0.8592	0.8334	0.8061	0.7774	0.7477	0.7171	0.6857	0.6539	0.6217	0.5894
n	21	22	23	24	25	26	27	28	29	30
$\frac{\frac{366!}{(366-n)!}}{366^n}$	0.5572	0.5252	0.4937	0.4627	0.4323	0.4028	0.3742	0.3466	0.3201	0.2947

It turns out that when there are 23 people in the room the **Pr**(No two birthdays on the same day) ≈ 0.5. This result has direct relation to computer science and hash tables, because it says that a hash table can be nearly empty (23 people compared to 366 days of the year, in our analogy), but it is still very likely that there will be a hashing conflict (that you will have two people with the same birthday, in our analogy).

For much more information on the birthday problem, including some fascinating graphs, see

`http://en.wikipedia.org/wiki/Birthday_problem`

Exercise How many people are needed to have the probability of the same birth *month* be at least 0.5? The same week of the year?

Exercise How many people are needed to have probability 50% that one of then has a birthday on December 25?

4 An Application—GUIDs

According to Wikipedia,

> A globally unique identifier or GUID is a 128-bit identifier used in software applications to provide a reference number which is unique in any context (hence, "globally"), for example, in defining the internal reference for a type of access point in a software application, or for creating unique keys in a database. While each generated GUID is not guaranteed to be unique, the total number of unique keys ($2^{128} \approx 3.4 \times 10^{38}$) is so large that the probability of the same number being generated twice is extremely small.

But how small? Specifically, suppose that a million GUIDs are generated every hour of every day for 100 years; what is the probability of a duplicate GUID?

The total number of GUIDs generated will be less than $10^6 \times 24 \times 366 \times 100 < 10^{12}$. Reasoning as in the birthday problem, the probability of a duplication is

$$\frac{2^{128}!}{(2^{128})^{10^{12}}(2^{128} - 10^{12})!} < 10^{-38}$$

by Stirling's approximation.

5 Another Application—Elevators

Ever wonder why the wait for an elevator seems interminable? Why the next elevator to come along is usually going in the wrong direction? Let's figure out why. Assume we have elevators that go up and down

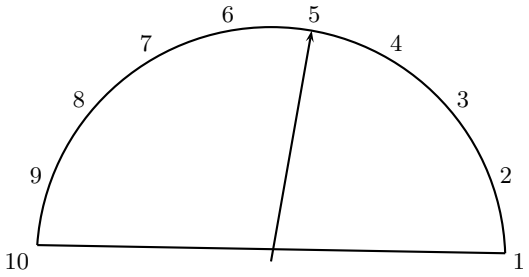
continuously from the bottom floor to the top floor and then back again in cyclic fashion; assume all elevators are independent of one another (this is *not* usually true: elevator software uses sophisticated techniques to balance the traffic). Finally, we assume that at the moment we begin to wait for an elevator, the elevators are all at random floors. The key value is

$$p = \frac{\text{distance from our floor to the bottom floor}}{\text{distance from top floor to bottom floor}}.$$

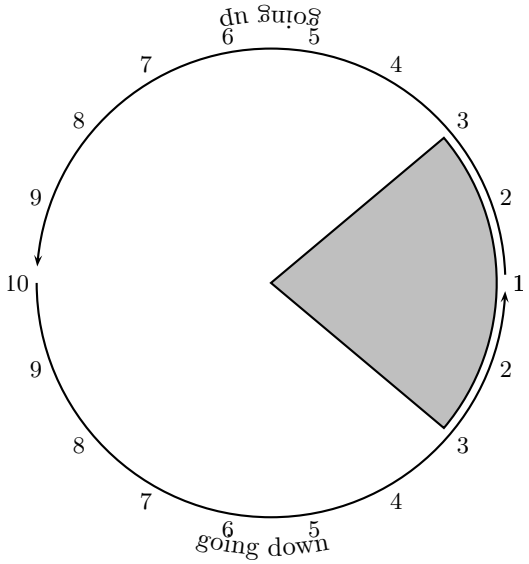
5.1 One elevator

Approaching the one elevator at a random time, there is probability p that it is below us (and hence will be going up when it reaches our floor) and probability $1 - p$ that it is above us (and hence will be going down when it reaches our floor). So, if we are waiting on the third floor to go up in an ten story building, $p = 2/9$, so seven times out of nine the next elevator to stop where we are waiting will be going the wrong way!

Recall the very old style elevator indicator:



The arrow sweeps counterclockwise when the elevator is going up and it sweeps clockwise when the elevator is going down. To make the movement always counterclockwise, the elevator's position can be thought of as being at points on a circle:

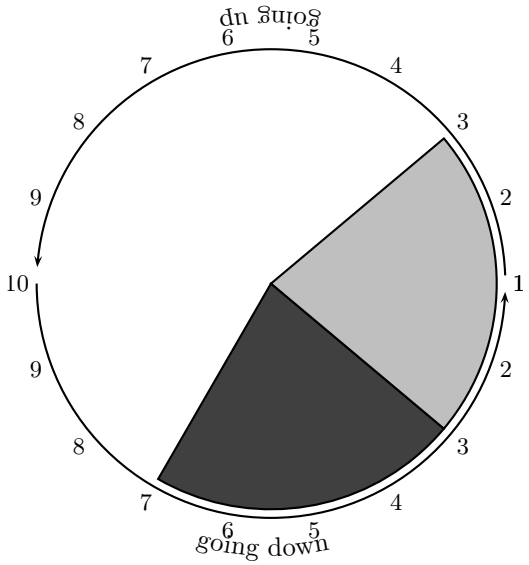


If the elevator is in the light gray region, it will being going up when it comes to the third floor; if it is anywhere else, it will being going up when it comes to the third floor. The light gray region is 2/9 of the

circle and we can thus understand the 2/9 probability as being the likelihood that the elevator is in that region.

5.2 Two elevators

With multiple elevators the problem is more complex because we are concerned with the direction of the *first* elevator that stops on our floor. Suppose there are two elevators. Now the diagram looks like this:



If both elevators are in the unshaded region, the first elevator to come to our floor will be going down.

If there is an elevator in the dark gray region and no elevator in the light gray region, the first elevator to come to our floor will be going down. If there is an elevator in the light gray region and no elevator in the dark gray region, the first elevator to come to our floor will be going up. These cases are mirror images.

If there is an elevator in each of the gray regions, the closer elevator reaches us first; but for every such case in which an elevator going down, the mirror image case has it going up. Thus in half of the cases of an elevator in each region, it will be going up and in half of the cases it will be going down.

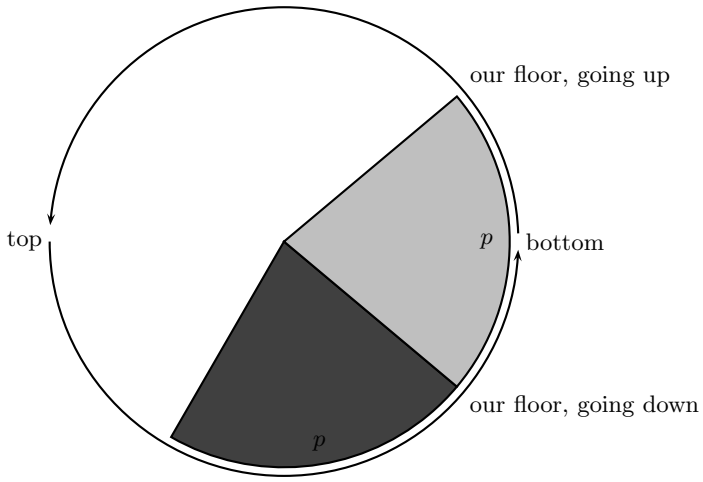
So, what is the probability that the first elevator is going down? It is the probability that both elevators are in the unshaded region, plus half of the probability that they are not:

$$(1 - 4/9)^2 + \frac{1}{2}\left(1 - (1 - 4/9)^2\right) = \frac{53}{81},$$

so the probability that the first elevator is going up is 28/81.

5.3 Many elevators

Now suppose there n elevators; scale the picture so the circumference of the circle is 1. We have



with each gray region having arc length p. We can assume that $p \leq 1/2$ because if $p > 1/2$ we could consider the analogous problem with $1 - p$ and the directions reversed. The calculation is now

$$\Pr(\text{next elevator going down}) = (1-2p)^n + \frac{1}{2}\left(1 - (1-2p)^n\right) = \frac{1}{2} + \frac{1}{2}(1-2p)^n.$$

For example, in a 23-story building with 6 elevators, if we are waiting on the third floor, we have $p = 2/22 = 1/11$, $n = 6$, so the probability that the next elevator is going down is

$$\frac{1}{2} + \frac{1}{2}(1 - 2/11)^6 \approx 0.65,$$

so almost 2/3 of the time the first elevator to come by will be going down.

Illinois Institute of Technology
Department of Computer Science

Lecture 11: September 30, 2013

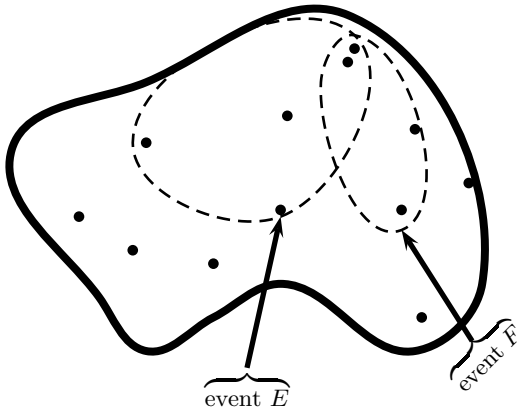
CS 330 Discrete Structures
Spring Semester, 2013

1 Conditional Probability

Consider now the following experiment: You are given three identical urns. Urn 1 contains 2 black balls. Urn 2 contains 2 gray balls. Urn 3 contains 1 black and 1 gray ball. You pick urn at random and take out a gray ball. How likely was it that you picked Urn 1? Urn 2? Urn 3?

For Urn 1, the result is easy since there are no gray balls. There are 2 gray balls in urn 2, so it has a $\frac{2}{3}$ probability while urn 3 has a $\frac{1}{3}$ probability.

Now lets replace the gray ball back into the urn and pick a ball again from the *same* urn. Again we get a gray ball; what is the probability that we had picked Urn 2? To answer this question we need to study **Conditional Probability**; this is inverts the usual question of how likely an event *will be* to, given that we *know* that an event occurred which could have been caused in various ways, what can we say about the probability that it was caused in a certain way. Specifically, conditional probability asks "What is the likelihood of event E happening if we already know that event F happened" and is written as: $\mathbf{Pr}\{E|F\}$. The situation is as follows:



If we know that event F happened, we know that the only part of event E that can happen is in the intersection of E and F. From this we have the result:

$$\mathbf{Pr}\{E|F\} = \frac{\mathbf{Pr}\{E \wedge F\}}{\mathbf{Pr}\{F\}}$$

$$\mathbf{Pr}\{F|E\} = \frac{\mathbf{Pr}\{F \wedge E\}}{\mathbf{Pr}\{E\}}$$

$$\Rightarrow \mathbf{Pr}\{E \wedge F\} = \mathbf{Pr}\{E\}\mathbf{Pr}\{F|E\}$$

$$\Rightarrow \mathbf{Pr}\{E|F\} = \frac{\mathbf{Pr}\{E\}\mathbf{Pr}\{F|E\}}{\mathbf{Pr}\{F\}} \quad (1)$$

The identity in Equation 1 is quite important, and is known as **Bayes' Theorem**.[1] Note also that there is nothing special about the letters E and F; we can interchange what events E and F represent and still have the same identities.

Here is an important example[2] from real life. An asymptomatic woman in her forties has a routine mammogram and the results come back positive; what is the probability that she actually has cancer? Mammograms are imperfect and can result in false positive results, so we know the *outcome* (a positive mammogram) and we want to know the probability of a *cause* (that the woman has cancer). We have the following data taken from the medical literature as of 2011:

$$\mathbf{Pr}\{\text{breast cancer in one's forties}\} = \frac{40}{10000}$$

$$\mathbf{Pr}\{\text{positive mammogram among breast cancer patients}\} = \mathbf{Pr}\{\text{positive mammogram} \mid \text{breast cancer}\} = \frac{32}{40}$$

$$\mathbf{Pr}\{\text{positive mammogram among all women}\} = \frac{1028}{10000}.$$

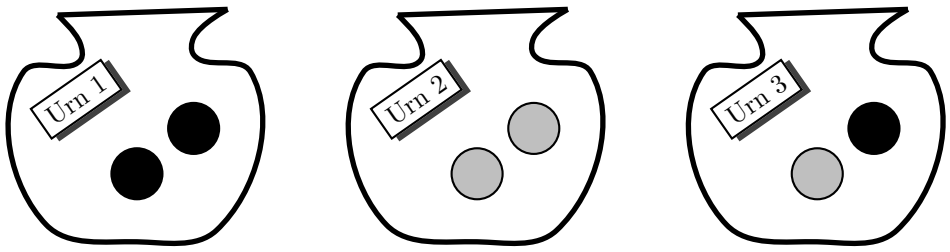
By Bayes' Theorem then, $\mathbf{Pr}\{\text{breast cancer in one's forties} \mid \text{positive mammogram}\}$

$$\begin{aligned} &= \frac{\mathbf{Pr}\{\text{breast cancer in one's forties}\} \times \mathbf{Pr}\{\text{positive mammogram} \mid \text{breast cancer}\}}{\mathbf{Pr}\{\text{positive mammogram}\}} \\ &= \frac{\frac{40}{10000} \times \frac{32}{40}}{\frac{1028}{10000}} \\ &\approx 0.03. \end{aligned}$$

This means that the likelihood of cancer in such a case is 3% which is somewhat reassuring for woman with the positive mammogram (though she must still undergo further tests to be sure).

This type of Bayesian analysis is critical in automatic spelling correction (given what the person typed "misteak", what are the probabilities that he/she meant to type "mistake", "mistook", or "steak"?) and spam filtering (given that the message contains the words "viagra" and "penis", what is the probability that it is spam? See pages 472–475 in Rosen.)

Let's go back to the experiment with urns and balls. We have three urns, the Urn 1 contains two black balls, Urn 2 contains two gray balls and Urn 3 contains a black and a gray ball.



Having selected an urn at random, we pull out a gray ball. What is the probability, for each urn, that we had selected that urn. In other words, if someone came to you and told you that they had picked a gray

[1] Reverend Thomas Bayes was a British cleric who was looking for a way to prove the existence of God. His method was to argue that, given what we see around us, what is the probability that God exists? For a fascinating history of the controvesy around "Bayesian inference," see *The Theory that Would not Die* by Sharon Bertsch McGrayne, Yale University Press, 2011.

[2] From Appendix B of McGrayne's book, but corrected here because she does not get it quite right.

ball, and asked you to guess to which urn the ball belonged; what would be the probability of you being right if you guessed a particular urn? In order to answer this question, let us first examine the probability of pulling out a gray ball from each of the three urns.

The first urn has $\mathbf{Pr}\{\text{picked gray}\} = \frac{0}{2} = 0$ since there are no gray balls to pick.

The second urn has $\mathbf{Pr}\{\text{picked gray}\} = \frac{2}{2} = 1$ since there are only gray balls to pick.

The third urn has $\mathbf{Pr}\{\text{picked gray}\} = \frac{1}{2}$ since there is one gray and one black ball to pick.

We can now apply Conditional Probability to the problem: If we choose an urn at random and get a gray ball, what was the probability of having picked a particular urn? We begin by finding the probability of a gray ball overall:

$$\begin{aligned}\mathbf{Pr}\{\text{Gray}\} &= \mathbf{Pr}\{\text{Urn 1}\} \times \mathbf{Pr}\{\text{Gray in Urn 1}\} + \mathbf{Pr}\{\text{Urn 2}\} \times \mathbf{Pr}\{\text{Gray in Urn 2}\} \\ &\quad + \mathbf{Pr}\{\text{Urn 3}\} \times \mathbf{Pr}\{\text{Gray in Urn 3}\} \\ &= \frac{1}{3} \times 0 + \frac{1}{3} \times 1 + \frac{1}{3} \times \frac{1}{2} \\ &= \frac{1}{2}.\end{aligned}$$

Now we can write three separate equations (and apply the definition of conditional probability for the first and Bayes' Theorem for the next two):

$$\mathbf{Pr}\{\text{Urn 1}|\text{Gray}\} = \frac{\mathbf{Pr}\{\text{Urn 1}\} \times \mathbf{Pr}\{\text{Gray}|\text{Urn 1}\}}{\mathbf{Pr}\{\text{Gray}\}} = \frac{\frac{1}{3} \times 0}{\frac{1}{2}} = 0$$

$$\mathbf{Pr}\{\text{Urn 2}|\text{Gray}\} = \frac{\mathbf{Pr}\{\text{Urn 2}\} \times \mathbf{Pr}\{\text{Gray}|\text{Urn 2}\}}{\mathbf{Pr}\{\text{Gray}\}} = \frac{\frac{1}{3} \times 1}{\frac{1}{2}} = \frac{2}{3}$$

$$\mathbf{Pr}\{\text{Urn 3}|\text{Gray}\} = \frac{\mathbf{Pr}\{\text{Urn 3}\} \times \mathbf{Pr}\{\text{Gray}|\text{Urn 3}\}}{\mathbf{Pr}\{\text{Gray}\}} = \frac{\frac{1}{3} \times \frac{1}{2}}{\frac{1}{2}} = \frac{1}{3}$$

Let us consider a related problem: If we choose an urn and pick one ball, return it to the urn and pick another ball from the same urn, and get two gray balls, what was the probability of having picked a particular urn? The probability of getting two gray balls overall is

$$\begin{aligned}\mathbf{Pr}\{\text{2 Gray balls}\} &= \mathbf{Pr}\{\text{Urn 1}\} \times \mathbf{Pr}\{\text{2 Gray balls in Urn 1}\} + \mathbf{Pr}\{\text{Urn 2}\} \times \mathbf{Pr}\{\text{2 Gray balls in Urn 2}\} \\ &\quad + \mathbf{Pr}\{\text{Urn 3}\} \times \mathbf{Pr}\{\text{2 Gray balls in Urn 3}\}\end{aligned}$$

$$\begin{aligned} &= \frac{1}{3} \times 0 + \frac{1}{3} \times 1 + \frac{1}{3} \times \frac{1}{4} \\ &= \frac{5}{12}. \end{aligned}$$

Applying Bayes' Theorem we have

$$\mathbf{Pr}\{\text{Urn 1}|\text{2 Gray Balls}\} = \frac{\mathbf{Pr}\{\text{Urn 1}\} \times \mathbf{Pr}\{\text{2 Gray Balls}|\text{Urn 1}\}}{\mathbf{Pr}\{\text{2 Gray Balls}\}} = \frac{\frac{1}{3} \times 0}{\frac{5}{12}} = 0$$

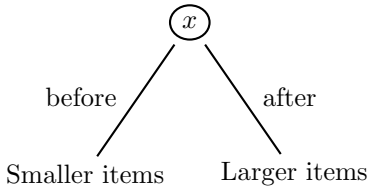
$$\mathbf{Pr}\{\text{Urn 2}|\text{2 Gray Balls}\} = \frac{\mathbf{Pr}\{\text{Urn 2}\} \times \mathbf{Pr}\{\text{2 Gray Balls}|\text{Urn 2}\}}{\mathbf{Pr}\{\text{2 Gray Balls}\}} = \frac{\frac{1}{3} \times 1}{\frac{5}{12}} = \frac{4}{5}$$

$$\mathbf{Pr}\{\text{Urn 3}|\text{2 Gray Balls}\} = \frac{\mathbf{Pr}\{\text{Urn 3}\} \times \mathbf{Pr}\{\text{2 Gray Balls}|\text{Urn 3}\}}{\mathbf{Pr}\{\text{2 Gray Balls}\}} = \frac{\frac{1}{3} \times \frac{1}{4}}{\frac{5}{12}} = \frac{1}{5}$$

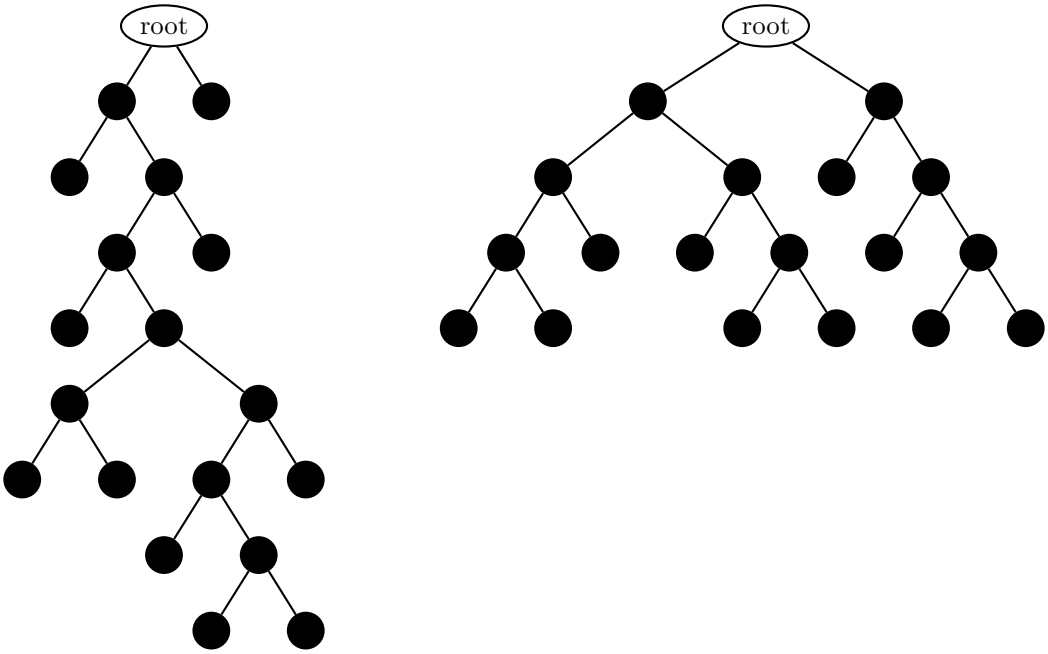
What if we get a gray ball three times? Four times? n times?

2 Probabilistically Balanced Lexicographical Trees

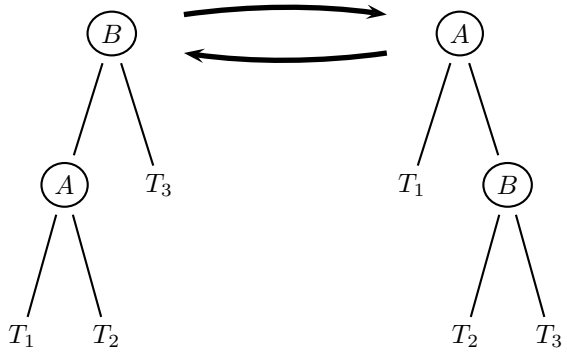
We can see computer application of Conditional Probability in probabilistically balanced lexicographical trees. A lexicographical tree is a data structure that embodies the idea of binary search. When you do a binary search, depending on the relative value of your current position and the value you are looking for, will descend either left or right. A lexicographical tree does the same:



This rule is then applied recursively to each and every node in the tree. To search through the tree, when you visit a node you compare what you are looking for with the value store in the node. If your value is less, examine the left child, otherwise examine the right child. The maximum number of nodes examined will be the height of the tree. What we want is a tree that has a low height as opposed to a tree with a big height:



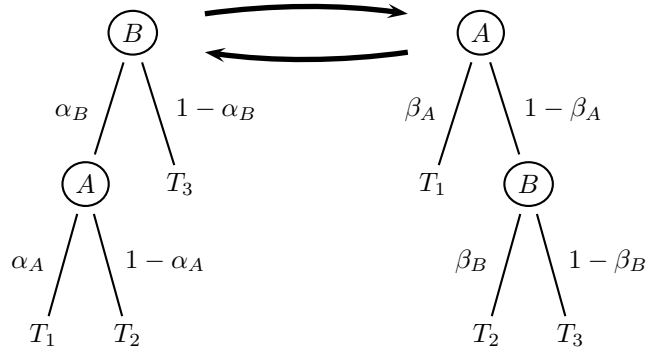
You can see that the tree on the left is a much deeper tree, so it will take longer to search that the tree on the right. In fact, if a tree is really bad, all the nodes will be in a single line. This is why it's very important to balance these trees. The basic step in tree balancing is a **rotation**. In order to rearrange the tree so it's short and fat, we apply the following transformation to certain branches:



You will notice that the lexicographical property is not affected by this transformation; in other words, we may rotate any branch in a lexicographic tree and still have a lexicographic tree.

If we are given the *probability* that, when we visit a node, we will go right, and the *probability* that, when we visit a node, we will go left, we can attempt to equate these probabilities in order to balance our tree. The problem is that after the rotation, the various probabilities will change and mess up our results. We must find a way to keep track of the probabilities of going left and the probabilities of going right, even after we rotate a branch.

We will use α and β, respectively to refer to probabilities in the two trees. We label the various probabilities concerning a rotation as follows:



We want to find the αs in terms of the βs, and vice versa. $\alpha_B = \mathbf{Pr}\{x < B\}$ where x is what we're searching for. We can see that for $x < B$ we need to go to either T_1 or T_2. To go to T_2 we have to go through A and B. Using the rules of sum and product, we have

$$\alpha_B = \beta_A + (1-\beta_A)\beta_B.$$

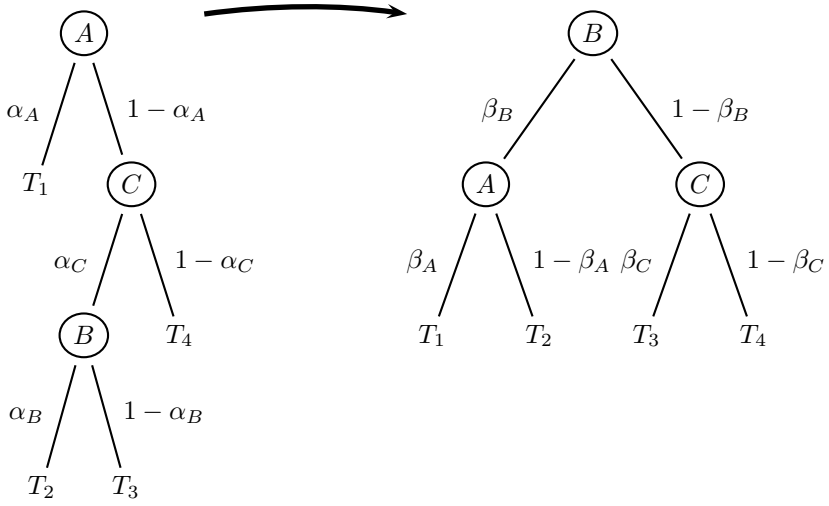
Similarly, we get:

$$\alpha_A = \mathbf{Pr}\{x < A | x < B\} = \frac{\mathbf{Pr}\{x < A\}\mathbf{Pr}\{x < B | x < A\}}{\mathbf{Pr}\{x < B\}} = \frac{\beta_A \cdot 1}{\beta_A + (1-\beta_A)(\beta_B)}$$

$$\beta_B = \mathbf{Pr}\{x < B | x > A\} = \frac{\mathbf{Pr}\{x < B\}\mathbf{Pr}\{x > A | x < B\}}{\mathbf{Pr}\{x > A\}} = \frac{\alpha_B(1-\alpha_A)}{1-\alpha_B\alpha_A}$$

$$\beta_A = \mathbf{Pr}\{x < A\} = \alpha_B\alpha_A$$

We also need to consider the *double rotation* and the resulting probabilities:



Exercise Use Bayes' Theorem and the rules of sum and product to compute the αs in terms of the βs and vice versa.

3 More Examples

See the Wikipedia entries
`http://en.wikipedia.org/wiki/Conditional_probability`
`http://en.wikipedia.org/wiki/Bayes'_theorem`
for other examples and historical information.

Illinois Institute of Technology
Department of Computer Science

Lecture 12: October 2, 2013

CS 330 Discrete Structures
Spring Semester, 2013

1 Analysis of algorithms

We now return to the problem that originally led us into the discussion of probability:

Consider the algorithm to find the maximum of n numbers $x_1,\ x_2,\ \ldots,\ x_n$:

```
1: m ← 1
2: for k ← 2 to n do
3:    if x_k > x_m then
4:       m ← k
5:    end if
6: end for
7: return m
```

To analyze this algorithm we want to know exactly how many times each of the instructions is executed; but for the moment, let us ask only how many times the statement "$m \leftarrow k$" in line 4 is executed. The answer, of course, depends not on the particular values of the x_i, but rather on their relative order: If x_1 is the maximum of the n elements, the statement will *never* be executed. If $x_1 < x_2 < \cdots < x_n$ then the statement will be executed $n-1$ times. For how many of the relative arrangements of the inputs $x_1, x_2, \ldots, x_n$ will the statement be executed exactly i times? The answer to this question is at the heart of the analysis of the algorithm.

We have shown that the best case of algorithm 1 is zero assignments and in the worst case there are $n-1$ assignments. But what about the average case? Let's examine the case where $n = 3$. The following is list of all the possible outcomes of relative x_i, for $i = 1, 2, 3$

$$\text{average number of assignments} = \left\{ \begin{array}{ll} x_1 < x_2 < x_3 & \text{has 2 assignments} \\ x_1 < x_3 < x_2 & \text{has 1 assignments} \\ x_2 < x_1 < x_3 & \text{has 1 assignments} \\ x_2 < x_3 < x_1 & \text{has 0 assignments} \\ x_3 < x_1 < x_2 & \text{has 1 assignments} \\ x_3 < x_2 < x_1 & \text{has 0 assignments} \end{array} \right\} = \frac{2+1+1+0+1+0}{6} = \frac{5}{6}$$

It is possible to do this for three, maybe even four, but we need to generalize this argument for a list of length n.

We can make some easy observations first: How many many inputs of length n have 0 swaps? There are $(n-1)!$, since there are $(n-1)!$ permuations with x_1 as the maximum. How many inputs of length n have $n-1$ swaps? There is 1, since only when the elements are in increasing order does this occur.

We now need to know the general case: How many many inputs of length n cause i assignments? Let's call this number $P_n(i)$.

We have the following two possibilities (either with an execution at the last step or without):

$$\underbrace{x_1, x_2, x_3, x_4, \cdots, x_{n-2}, x_{n-1},}_{P_{n-1}(i-1)} \overbrace{x_n}^{\text{largest element}}$$

$$\underbrace{x_1, x_2, x_3, x_4, \cdots, x_{n-2}, x_{n-1},}_{P_{n-1}(i)} \overbrace{x_n}^{\overbrace{\text{not the largest element}}^{n-1 \text{ choices}}}$$

But applying the rules of sum and product, we have:

$$P_n(i) = P_{n-1}(i-1) + (n-1)P_{n-1}(i)$$

From the discussion above we also know that

$$P_n(0) = (n-1)!$$

and

$$P_n(n-1) = 1.$$

2 Expected value

Now we know the number of permutations with i swaps, but what about the average case? We've see that it's just the sum of the i divided by the number of permutations, or:

$$\text{average case} = \frac{\sum_{i=0}^{n-1} iP_n(i)}{n!} = \sum_{i=0}^{n-1} i\frac{P_n(i)}{n!} = \sum_{i=0}^{n-1} i\mathbf{Pr}(i)$$

This is called the *expected value* of i.

By the definition of probability, $\mathbf{Pr}(i) = \frac{P_n(i)}{n!} = p_n(i)$, where $p_n(i)$ is the probability of executing i assignments in a list of length n.

The initial conditions for P_n tell us that

$$p_n(0) = 1/n$$

and

$$p_1(i) = \begin{cases} 1 & i = 0, \\ 0 & i > 0. \end{cases}$$

Since $P_n(i) = P_{n-1}(i-1) + (n-1)P_{n-1}(i)$:

$$p_n(i) = \frac{P_n(i)}{n!} = \frac{P_{n-1}(i-1)}{n!} + \frac{(n-1)P_{n-1}(i)}{n!} = \frac{1}{n}p_{n-1}(i-1) + \frac{n-1}{n}p_{n-1}(i), i \geq 1$$

This gives us the recurrence relation between the probabilities. You can see this from the below table showing some values of $p_n(i)$:

	$i=0$	$i=1$	$i=2$	$i=3$	$\ldots$
$n=1$	1	0	0	0	$\cdots$
$n=2$	1/2	1/2	0	0	$\cdots$
$n=3$	1/3	1/2	1/6	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Let us consider the **probability generating function** defined as:

$$\mathcal{P}_n(x) = p_n(0)x^0 + p_n(1)x^1 + p_n(2)x^2 + p_n(3)x^3 + \cdots = \sum_{i=0}^{\infty} p_n(i)x^i$$

Notice that

$$\mathcal{P}'_n(1) = e_n = 0 \cdot p_n(0) + 1 \cdot p_n(1) + 2 \cdot p_n(2) + \cdots = \sum_{i=0}^{\infty} i \cdot p_n(i)$$

We would like to determine $\mathcal{P}_n(x)$. First, recall the special case of $p_n(0) = \frac{1}{n}$. We have:

$$\begin{aligned}
\mathcal{P}_n(x) &= \sum_{i=0}^{\infty} p_n(i)x^i \\
&= p_n(0)x^0 + \sum_{i=1}^{\infty} p_n(i)x^i \\
&= \frac{1}{n}x^0 + \sum_{i=1}^{\infty} p_n(i)x^i \\
&= \frac{1}{n} + \sum_{i=1}^{\infty} \frac{1}{n} p_{n-1}(i-1)x^i + \sum_{i=1}^{\infty} \frac{n-1}{n} p_{n-1}(i)x^i \\
&= \frac{1}{n} + \frac{1}{n}\sum_{i=1}^{\infty} p_{n-1}(i-1)x^i + \frac{n-1}{n}\sum_{i=1}^{\infty} p_{n-1}(i)x^i \\
&= \frac{1}{n} + \frac{x}{n}\sum_{i=1}^{\infty} p_{n-1}(i-1)x^{i-1} + \frac{n-1}{n}\sum_{i=1}^{\infty} p_{n-1}(i)x^i \\
&= \frac{1}{n} + \frac{x}{n}\sum_{i=0}^{\infty} p_{n-1}(i)x^i + \frac{n-1}{n}\sum_{i=1}^{\infty} p_{n-1}(i)x^i \\
&= \frac{1}{n} + \frac{x}{n}\mathcal{P}_{n-1}(x) + \frac{n-1}{n}\sum_{i=1}^{\infty} p_{n-1}(i)x^i \\
&= \frac{1}{n} + \frac{x}{n}\mathcal{P}_{n-1}(x) + \frac{n-1}{n}\left(\sum_{i=0}^{\infty} p_{n-1}(i)x^i - p_{n-1}(0)\right) \\
&= \frac{1}{n} + \frac{x}{n}\mathcal{P}_{n-1}(x) + \frac{n-1}{n}\left(\sum_{i=0}^{\infty} p_{n-1}(i)x^i - \frac{1}{n-1}\right) \\
&= \frac{x}{n}\mathcal{P}_{n-1}(x) + \frac{n-1}{n}\mathcal{P}_{n-1}(x) + \frac{1}{n} - \frac{1}{n} \\
&= \frac{x+n-1}{n}\mathcal{P}_{n-1}(x)
\end{aligned}$$

We can now find the derivative of $\mathcal{P}_n(x)$ with respect to x:

$$\begin{aligned}\mathcal{P}'_n(x) &= \frac{d}{dx}(\frac{x+n-1}{n})\mathcal{P}_{n-1}(x) + (\frac{x+n-1}{n})\mathcal{P}'_{n-1}(x) \\ &= \frac{1}{n}\mathcal{P}_{n-1}(x) + (\frac{x+n-1}{n})\mathcal{P}'_{n-1}(x),\end{aligned}$$

so that

$$\mathcal{P}'_n(1) = \frac{1}{n}\mathcal{P}_{n-1}(1) + \mathcal{P}'_{n-1}(1)$$

By the definition of $\mathcal{P}(n)$, we know that $\mathcal{P}_{n-1}(1) = 1$. This gives us:

$$\mathcal{P}'_n(1) = \frac{1}{n} + \mathcal{P}'_{n-1}(1)$$

Let $\mathcal{P}'_n(1) = e_n$. We can substitute as follows:

$$\begin{aligned}e_n &= \frac{1}{n} + e_{n-1} \\ &= \frac{1}{n} + \frac{1}{n-1} + e_{n-2} \\ &\vdots \\ &= \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \cdots + e_0 \\ &= H_n - 1 \\ &\approx \ln n\end{aligned}$$

When we get such a simple answer after a complicated derivation, we should ask ourselves whether a simpler derivation is possible. The answer is yes (though our more complicated method above will be needed later, when we ask for more sophisticated information). Notice that an important basic principle holds:

The expectation of the sum = the sum of the expectations.

That means that if q_k is the probability that (in the algorithm) a new maximum element is found at iteration $k > 1$, then the its contribution to the expected value is $1 \times q_k = q_k$. Of the k possible relative sizes of x_k among the first $k-1$ values, all equally likely, the assignment statement is made only for one relative size: x_k must be larger than the first $k-1$ values. Thus $q_k = 1/k$ and the expected number of executions of the assignments statement is $\sum_{1<k\leq n} 1/n = H_n - 1$.

3 Two Similar Problems

With that analysis under our belt, we can look at two similar problems.

3.1 Weather Records

First, how often are records set? That is, suppose we look at expected rainfall and ask how often does the rainfall in a given year exceed the rainfall in all previous years (thus setting a new record)? To address this problem we make the (not entirely convincing) assumption that amount rainfall in a year is independent of rainfall in prior years. The problem is then identical to our algorithm analysis above because we can view each time we see a new largest element as setting a record; the only difference is that the first year's rainfall is (by definition) a new record. Thus the expected number of record-breaking years in a sequence of n years is $H_n \approx \ln n$ (Question: What happened to the -1 that occurs in the analyis of the maximum value algorithm?). For example, in New York City's Central Park there were 6 record rainfall years in the 160-year period 1835–1994; $H_{160} \approx 5.7$.

Another to look at the problem is to ask, on the average, how long do we have to wait to see a weather record broken? Suppose, for example, we look at snowfall totals on Christmas Day in Chicago. There was no snow on Christmas 2011. What is the expected number of years we have to wait to see that amount of snow surpassed on Christmas Day (in other words, what is our expected wait for snow on Christmas)? We again make the assumption that amount snowfall on Christmas Day in a year is independent of snowfall on Christmas Days in prior years. Let s_i be the Christmas snowfall for the ith year of the data, so $s_1 = 0.0$ for 2011, s_2 is the snowfall for 2012, etc. Let s_N be the first year after 2011 that beats the 2011 Christmas snowfall record. What is the probability that $N > n-1$, that is, that the record is broken only after all previous values, $s_1, s_2, \ldots, s_{n-1}$? We've seen from our analysis of the maximum algorithm that (because of the independence assumption) the probability is $1/(n-1)$ that the maximum of $s_1, s_2, \ldots, s_{n-1}$ occurs at s_1. In short,

$$\mathbf{Pr}(N > n-1) = \frac{1}{n-1}.$$

Similarly,

$$\mathbf{Pr}(N > n) = \frac{1}{n}.$$

But,

$$\mathbf{Pr}(N = n) = \mathbf{Pr}(N > n-1) - \mathbf{Pr}(N > n)$$

(that is, the probablity that N is bigger than $n-1$ but not bigger tha n). In other words,

$$\mathbf{Pr}(N = n) = \frac{1}{n-1} - \frac{1}{n} = \frac{1}{(n-1)n}.$$

Now we can calculate the expected future year in which the snowfall exceeds s_1:

$$\mathbf{E}(N) = \sum_{n=2}^{\infty} n\mathbf{Pr}(N = n) = \sum_{n=2}^{\infty} \frac{1}{n-1} = \infty$$

because the harmonic series diverges. In other words, our *expected wait* for snow on Christmas Day is infinite! The difficulty here is that the "average value" does not give us very useful information: what we really want here is the *full distribution* $\mathbf{Pr}(N = n) = \frac{1}{(n-1)n}$, $n > 1$, which tells the probabilty that we get snow for Christmas in the n year: the odds are 50% next year, $16.666\cdots$% the year after that, and so on. But if it doesn't snow next Christmas, the same logic tells us that *then* the likelihood is (again) 50% the following year, not $16.666\cdots$%. Can you explain what is going on?

3.2 Cracker Jack Prizes

Our second problem is a bit more complex. Cracker Jack, the carmel-covered popcorn treat, has a small toy as a prize in each box; if there n different prizes, how many boxes would one expect to buy to get a complete set of prizes? (Mathematicians call this the "coupon collector's problem".) Because the expectation of the sum is the sum of the expectations, the answer is the expected number of boxes to get the first prize, plus the expected number of boxes to get the second prize (that is, a prize that differs from the first prize we got), plus the expected number of boxes to get the third prize (that is, a prize that differs from the first two prizes we got), etc.

Let E_i be the expected number of boxes we must buy to get ith new prize. Clearly, $E_1 = 1$ because the first prize we get is not a duplicate of a previously obtained prize. Using the rules of sum and product, let's examine what happens as we buy more boxes hoping to get a second toy. Getting a new toy on the second box we buy has probability $(n-1)/n$ because any of $n-1$ prizes would be okay; if that fails (with probability $1/n$ we get the toy we already had), the probability of getting a new toy on the next box we buy is $(n-1)/n$ for a total probability of their product $(n-1)/n^2$. In general, to get a new toy on the kth box means we failed $k-1$ times (that is, we got a duplicate of the first toy $k-1$ times) and succeeded on the kth try; that probability is $(n-1)/n^k$. The expected value is thus

$$E_2 = \sum_{k=1}^{\infty} k \times \frac{n-1}{n^k} = \frac{n-1}{n} \sum_{k=1}^{\infty} \frac{k}{n^{k-1}} = \frac{n}{n-1}.$$

The last summation (left as an exercise) comes from substituting $x = 1/n$ in the derivative of $(1-x)^{-1}$,

$$1 + 2x + 3x^2 + 4x^3 + \cdots = \frac{1}{(1-x)^2}.$$

What about getting a third toy differing from our first two toys? The analysis is similar: To get such a toy in the kth box we buy we must get a duplicate of one of our two toys $k-1$ times [probability $(2/n)^{k-1}$], then succeed in getting a new toy in the kth box [probability $(n-2)/n$]. Thus

$$E_3 = \sum_{k=1}^{\infty} k \times \left(\frac{2}{n}\right)^{k-1} \frac{n-2}{n} = \frac{n-2}{n} \sum_{k=1}^{\infty} k \times \left(\frac{2}{n}\right)^{k-1} = \frac{n}{n-2}$$

[substituting $x = 2/n$ in the derivative of $(1-x)^{-1}$].

Doing this calculation for the tth new toy, we find

$$E_t = \frac{n}{n-t+1}$$

and hence the expected number of boxes we must purchase to get n different toys is

$$\sum_{t=1}^{n} E_t = \sum_{t=1}^{n} \frac{n}{n-t+1} = n \sum_{t=1}^{n} \frac{1}{n-t+1} = n\left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{1}\right) = nH_n.$$

The above analysis tells, for example, that we have to roll a normal 6-sided die an average of $6H_6 = 6 \times 49/20 = 14.7$ times to see all 6 sides come up.

Exercise In the "birthday problem" what is the expected number of people we need in a room to get at least one person with each possible birthday?

Illinois Institute of Technology
Department of Computer Science

Lecture 13: October 9, 2013

CS 330 Discrete Structures
Fall Semester, 2013

Did you hear about the statistician who drowned in a river with average depth of three inches?

The average human being has one breast and one testicle.

1 How indicative of "real life" is the average?

The mean or expected value is often not enough when we are trying to draw some meaningful conclusions about the nature of a distribution. Suppose you were told that the average score on the CS 330 midterm was 50, you still do not know if everyone in the class scored a 50, or if half the class scored 0 and the other half 100, and so on.

In our analysis of the algorithm for determining the largest value in an array, we determined the average number of executions of the assignment statement $m \Leftarrow k$ to be $H_n - 1$. This, on its own, doesn't give us a lot of information about the behavior of the algorithm. We need some more information. So we now ask the question, how close to the mean are the actual values that I have?

1.1 The difference

One way to measure the "averageness" is to calculate the expected value of the deviation, that is, the average value of $x - a$, where x is the variable with average value a. Because the average of a difference is the difference of the averages (why?), the average value of $x - a$ is zero. That tells us that the variable x is just as often above a as it is below a, which we knew because a is the average of x.

1.2 Absolute difference

We might want to phrase our last statement as: "What is the expected value of $|x-a|$, where a is the mean?" However, the function $|x - a|$ does not suit our needs because it is not differentiable and hence cannot be subjected to several useful mathematical operations.

1.3 Variance and Standard Deviation

We modify our goal and say: "What is the expected value of $(x - a)^2$?" This value is referred to as the *variance* of the given distribution.

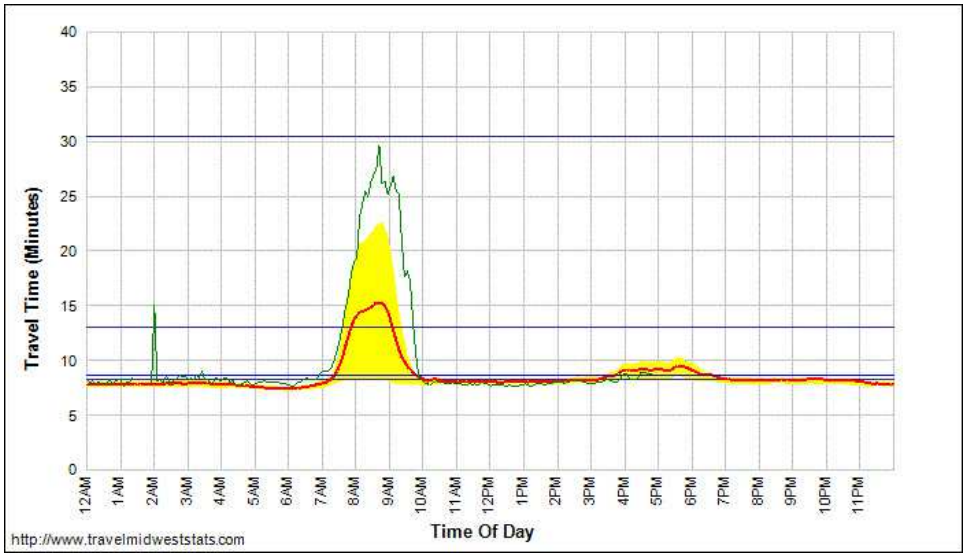
For any distribution, we know that the expected value of the random variable x is $E(x) = \sum_{-\infty}^{\infty} x \cdot Pr(x)$, where $Pr(x)$ is the probability that the random variable takes the value x.

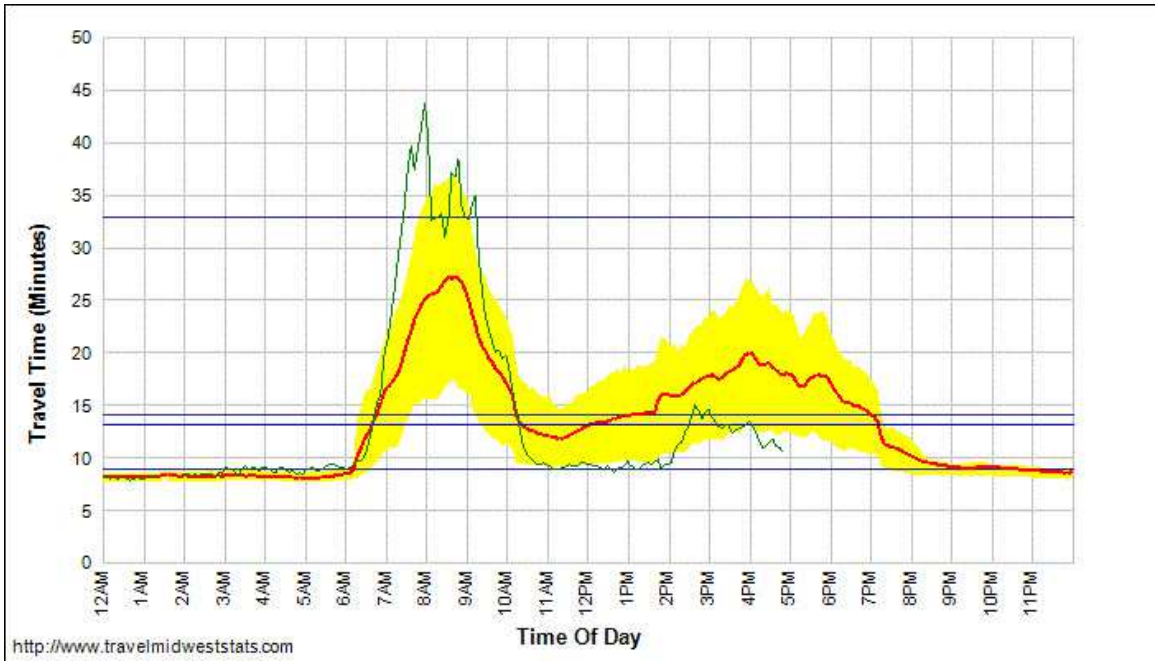
The variance is the expected value of $(x - a)^2$, and is given by $E[(x - a)^2] = E(x^2) - a^2$. The variance gives us a measure of the spread of values around the mean, except that it squares the deviation from the mean.

The *standard deviation*, often denoted by σ, is another common measure, and $\sigma = \sqrt{Variance(x)}$.

Note In general, $E[f(x)] = \sum_{-\infty}^{\infty} f(x)Pr(x)$.

Here is a nice example: The following two graphs show the commuting times from the north side of Chicago to the Loop. The first is along Lake Shore Drive from Bryn Mawr to Randolph, a distance of 7.62 miles and the second is along the Kennedy Expressway from Montrose to the Circle, a distance of 8.23 miles. In both cases the green line is the current travel time (on Monday, September 26, 2011 at 4:45pm), the red line is the average travel time for all samples ever collected, the yellow fill indicates one standard deviation from the mean (so about 68% of all travel times are in the yellow areas), the blue line is average overall travel time for all days/times, and the dark blue line indicate speed thresholds (55 mph, 35 mph, and 15 mph).





The distance from the red line to the (upper, say) edge of the yellow area is the *standard deviation* of travel time. At 8:30am for Lake Shore Drive it is about 7 minutes, while for the Kennedy it is about 10 minutes. The smaller value means that the average for the Drive is more indicative of what a driver actually experiences than for it is for the Kennedy. On the other hand, the standard deviations at, say, 3am are so small that the average travel times is a good indicator of what a driver would actually experience.

1.4 Moments of a distribution

The *moments* of a distribution are certain other measures that provide some more insight into the shape of the distribution. The kth moment of a distribution is defined to be $\sum_{-\infty}^{\infty}(x-a)^k Pr(x)$, where a is the mean. The *variance* is simply the 2nd moment of a distribution.

2 Analyzing algorithms, finding the variance

Now that we know what variance is, we can try to determine the variance of i, the number of times the assignment statement executes in the algorithm to determine the largest element of an array.

The **probability generating function** is:

$$\mathcal{P}_n(x) = p_n(0)x^0 + p_n(1)x^1 + p_n(2)x^2 + \cdots = \sum_{i=0}^{\infty} p_n(i)x^i = \frac{x+n-1}{n}\mathcal{P}_{n-1}(x).$$

We also determined that

$$\mathcal{P}'_n(1) = e_n = H_n - 1,$$

where e_n is the expected number of executions of the assignment statement when the array size is n.

We know that the variance is given by $E(i^2) - [E(i)]^2$, i being the number of executions of the assignment statement. And now we need to determine $E(i^2)$. Let us consider

$$\mathcal{P}''_n(x) = \sum_{i=2}^{\infty}(i)(i-1)x^{i-2}p(i) = \sum_{i=2}^{\infty} i^2 \cdot x^{i-2}p(i) - \sum_{i=2}^{\infty} i \cdot x^{i-2}p(i).$$

Comparing this with $E(i^2)$ gives us:

$$E(i^2) = (\mathcal{P}''_n(1) + 0^2 \cdot p(0) + 1^2 \cdot p(1)) + \sum_{i=2}^{\infty} i \cdot p(i) = \mathcal{P}''_n(1) + \mathcal{P}'_n(1).$$

So, the variance is

$$\mathcal{P}''_n(1) + \mathcal{P}'_n(1) - (\mathcal{P}'_n(1))^2.$$

We know from last time that the value of $\mathcal{P}'_n(1) = H_n - 1$, so all we have to evaluate is $\mathcal{P}''_n(1)$. Recall also from last time that

$$\mathcal{P}'_n(x) = \frac{1}{n}\mathcal{P}_{n-1}(x) + (\frac{x+n-1}{n})\mathcal{P}'_{n-1}(x)$$

so

$$\mathcal{P}''_n(x) = \frac{2}{n}\mathcal{P}'_{n-1}(x) + \frac{x+n-1}{n}\mathcal{P}''_{n-1}(x).$$

Now,

$$\begin{aligned}
\mathcal{P}''_n(1) &= \frac{2}{n}\mathcal{P}'_{n-1}(1) + \mathcal{P}''_{n-1}(1) \\
&= \frac{2}{n}(H_{n-1} - 1) + \mathcal{P}''_{n-1}(1) \\
&= \frac{2}{n}(H_{n-1} - 1) + \frac{2}{n-1}(H_{n-2} - 1) + \mathcal{P}''_{n-2}(1) \\
&= 2(\sum_1^n \frac{1}{i} H_{i-1} - \sum_1^n \frac{1}{i}) \\
&= 2(\sum_1^n \frac{1}{i}(H_i - \frac{1}{i}) - \sum_1^n \frac{1}{i}) \\
&= 2(\sum_1^n \frac{1}{i} H_i - \sum_1^n \frac{1}{i^2} - H_n)
\end{aligned}$$

We need to evaluate $\sum_1^n \frac{1}{i} H_i$. Call this sum S_n and imagine an $n \times n$ array in which position (i, j) contains the value $\frac{1}{ij}$. Now multiply out all the terms in the product

$$H_n^2 = (1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n})(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}).$$

That product is the sum of all n^2 array positions. But the array is symmetric around the main diagonal, so the sum of the elements on or above the diagonal equals the sum of the elements on or below the diagonal; S_n is that sum. In other words, H_n^2 would be $2S_n$, but since S_n includes the diagonal elements $1/i^2$, those elements are included twice in $2S_n$. Hence $H_n^2 = 2S_n - \sum 1/i^2$.)

Thus,

$$S_n = \frac{1}{2} \times (H_n^2 + \sum_1^n \frac{1}{i^2}).$$

Using this result, we obtain

$$\mathcal{P}''_n(1) = H_n^2 - \sum_1^n \frac{1}{i^2} - 2H_n$$

Further, we use $\sum_1^\infty \frac{1}{i^2} = \frac{\pi^2}{6}$ to find

$$\sigma^2 = H_n - O(1)$$

This is the variance for the number of times the assignment statement executes when we want to find the greatest value in an array. It is somewhat expected that the deviation from the mean will be greater as n increases, and that is what this result also tells us.

3 Problems to Think About

What is the variance for the "coupon collector's problem" discussed at the end of the previous lecture?

Illinois Institute of Technology
Department of Computer Science

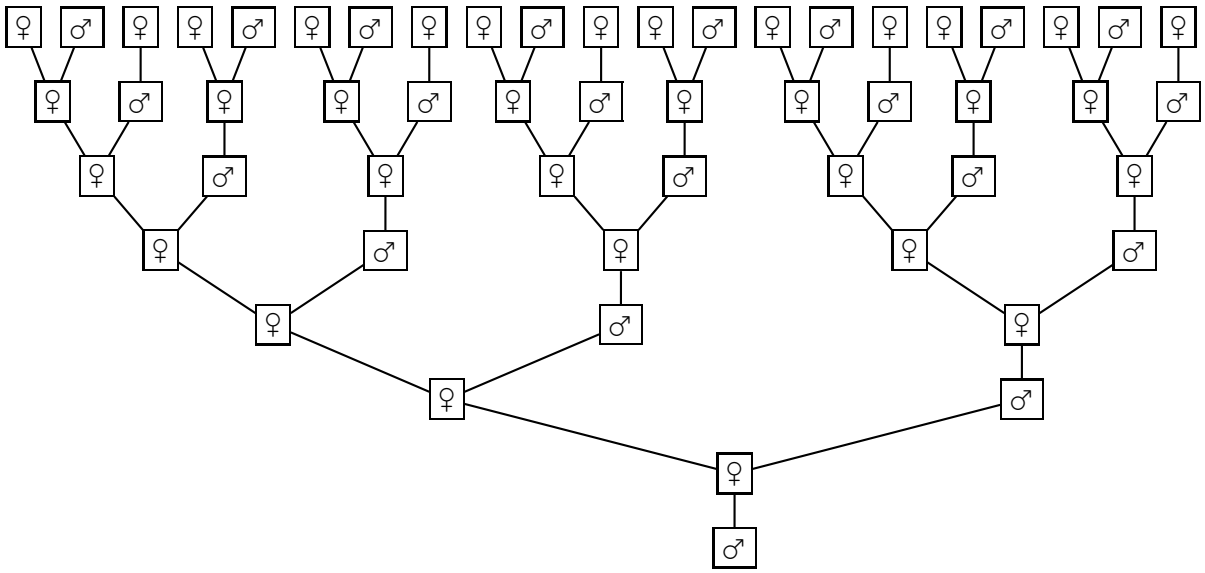
Lectures 14–16: October 14–21, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Solving Recurrences

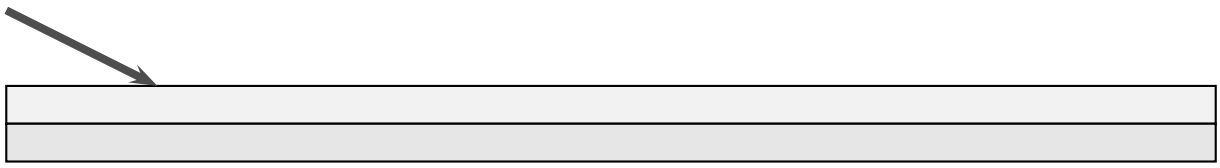
1.1 Fun with Fibonacci numbers

Consider the reproductive cycle of bees. Each male bee has a mother but no father; each female bee has both a mother and a father. If we examine the generations we see the following family tree:

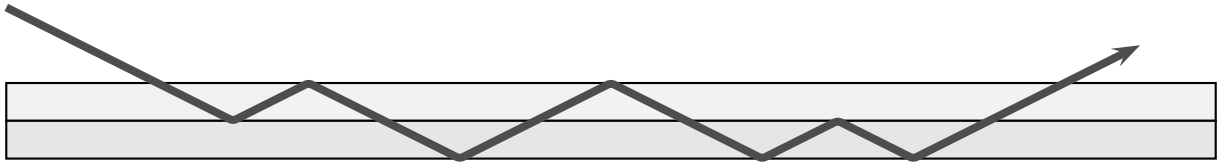


We easily see that the number of ancestors in each generation is the sum of the two numbers before it. For example, our male bee has three great-grandparents, two grandparents, and one parent, and $3 = 2 + 1$. The number of ancestors a bee has in generation n is defined by the Fibonacci sequence; we can also see this by applying the rule of sum.

As a second example, consider light entering two adjacent planes of glass:

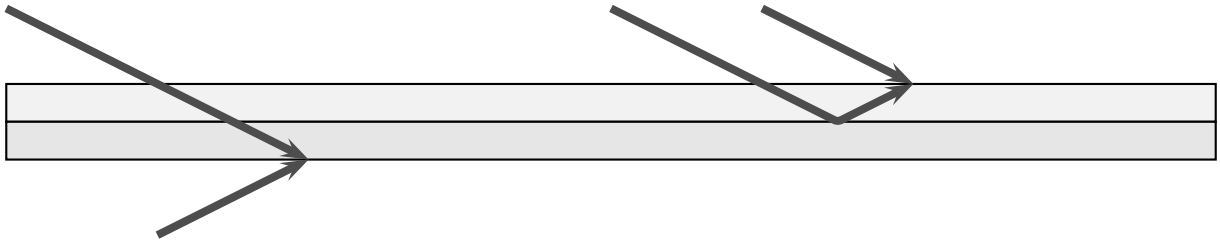


At any meeting surface (between the two panes of glass, or between the glass and air), the light may either reflect or continue straight through (refract). For example, here is the light bouncing seven times before it leaves the glass.



In general, how many different paths can the light take if we are told that it bounces n times before leaving the glass?

The answer to the question (in case you haven't guessed) rests with the Fibonacci sequence. We can divide the set of paths with n reflections into two subsets, depending on where the first reflection happens.



- Suppose the first bounce is on the boundary between the two panes. After the bounce, the light either leaves the glass immediately (so $n = 1$), or bounces again off the top of the upper pane. After the *second* bounce, if any, the path is equivalent to a path that enters from the top and bounces $n - 2$ times.
- Suppose the first bounce is *not* at the boundary between the two panes. Then either there are no bounces at all (so $n = 0$) or the first bounce is off the bottom pane. After the first bounce, the path is equivalent to a path that enters from the bottom and bounces $n - 1$ times. Entering through the bottom pane is the same as entering through the top pane (but flipped over).

Thus, we obtain the following recurrence relation for F_n, the number of paths with exactly n bounces. There is exactly one way for the light to travel with no bounces—straight through—and exactly two ways for the light to travel with only one bounce—off the bottom and off the middle. For any $n > 1$, there are F_{n-1} paths where the light bounces off the bottom of the glass, and F_{n-2} paths where the light bounces off the middle and then off the top.

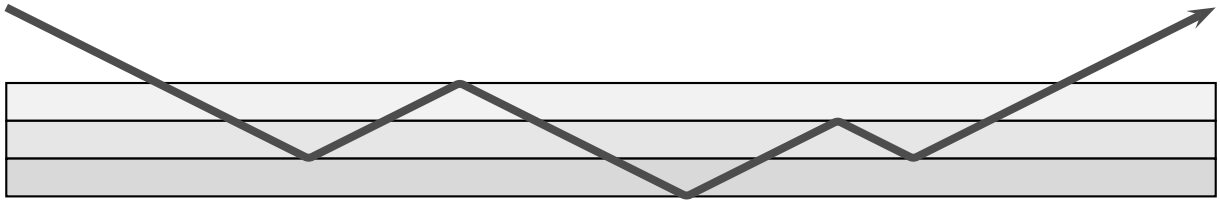
$$\begin{aligned} F_0 &= 1 \\ F_1 &= 2 \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

Stump a professor

What is the recurrence relation for *three* panes of glass? This question once stumped an anonymous professor[1] in a science discipline, but now you should be able to solve it with a bit of effort.

See `http://oeis.org/search?q=A006356`

[1] Not me!



1.2 Sequences, sequence operators, and annihilators

We have shown that several different problems can be expressed in terms of Fibonacci sequences, but we don't yet know how to explicitly compute the nth Fibonacci number, or even (and more importantly) roughly how big it is. We can easily write a program to compute the nth Fibonacci number, but that doesn't help us much here. What we really want is a *closed form solution* for the Fibonacci recurrence—an explicit algebraic formula without conditionals, loops, or recursion.

In order to solve recurrences like the Fibonacci recurrence, we first need to understand *operations* on infinite sequences of numbers. Although these sequences are formally defined as *functions* of mapping the natural numbers to the real numbers,, we will write them either as $A = \langle a_0, a_1, a_2, a_3, a_4, \ldots \rangle$ when we want to emphasize the entire sequence[2], or as $A = \langle a_i \rangle$ when we want to emphasize a generic element. For example, the Fibonacci sequence is $\langle 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots \rangle$.

We can naturally define several sequence operators:

- We can add or subtract any two sequences:

$$\langle a_i \rangle + \langle b_i \rangle = \langle a_0, a_1, a_2, \ldots \rangle + \langle b_0, b_1, b_2, \ldots \rangle = \langle a_0 + b_0, a_1 + b_1, a_2 + b_2, \ldots \rangle = \langle a_i + b_i \rangle$$

$$\langle a_i \rangle - \langle b_i \rangle = \langle a_0, a_1, a_2, \ldots \rangle - \langle b_0, b_1, b_2, \ldots \rangle = \langle a_0 - b_0, a_1 - b_1, a_2 - b_2, \ldots \rangle = \langle a_i - b_i \rangle$$

- We can multiply any sequence by a constant:

$$c \cdot \langle a_i \rangle = c \cdot \langle a_0, a_1, a_2, \ldots \rangle = \langle c \cdot a_0, c \cdot a_1, c \cdot a_2, \ldots \rangle = \langle c \cdot a_i \rangle$$

- We can shift any sequence to the left by removing its initial element:

$$\mathbf{E}\langle a_i \rangle = \mathbf{E}\langle a_0, a_1, a_2, a_3, \ldots \rangle = \langle a_1, a_2, a_3, a_4, \ldots \rangle = \langle a_{i+1} \rangle$$

Example 1. *We can understand these operators better by looking at some specific examples, using the sequence T of powers of two.*

$$\begin{aligned} T &= \langle 2^0, 2^1, 2^2, 2^3, \ldots \rangle = \langle 2^i \rangle \\ \mathbf{E}T &= \langle 2^1, 2^2, 2^3, 2^4, \ldots \rangle = \langle 2^{i+1} \rangle \\ 2T &= \langle 2 \cdot 2^0, 2 \cdot 2^1, 2 \cdot 2^2, 2 \cdot 2^3, \ldots \rangle = \langle 2^1, 2^2, 2^3, 2^4, \ldots \rangle = \langle 2^{i+1} \rangle \\ 2T - \mathbf{E}T &= \langle 2^1 - 2^1, 2^2 - 2^2, 2^3 - 2^3, 2^4 - 2^4, \ldots \rangle = \langle 0, 0, 0, 0, \ldots \rangle = \langle 0 \rangle \end{aligned}$$

[2]It really doesn't matter whether we start a sequence with a_0 or a_1 or a_5 or even a_{-17}. Zero is often a convenient starting point for many recursively defined sequences, so we'll usually start there.

1.2.1 Properties of operators

It turns out that the distributive property holds for these operators, so we can rewrite $\mathbf{E}T - 2T$ as $(\mathbf{E}-2)T$. Since $(\mathbf{E}-2)T = \langle 0,0,0,0,\ldots\rangle$, we say that the operator $(\mathbf{E}-2)$ *annihilates* T, and we call $(\mathbf{E}-2)$ an *annihilator* of T. Obviously, we can trivially annihilate any sequence by multiplying it by zero, so as a technical matter, we do not consider multiplication by 0 to be an annihilator.

What happens when we apply the operator $(E-3)$ to our sequence T?

$$(\mathbf{E}-3)T = \mathbf{E}T - 3T = \langle 2^{i+1}\rangle - 3\langle 2^i\rangle = \langle 2^{i+1} - 3\cdot 2^i\rangle = \langle -2^i\rangle = -T$$

The operator $(\mathbf{E}-3)$ did very little to our sequence T; it just flipped the sign of each number in the sequence. In fact, we will soon see that *only* $(\mathbf{E}-2)$ will annihilate T, and all other simple operators will affect T in very minor ways. Thus, if we know how to annihilate the sequence, we know what the sequence must look like.

In general, $(\mathbf{E}-c)$ annihilates any geometric sequence $A = \langle a_0, a_0c, a_0c^2, a_0c^3, \ldots\rangle = \langle a_0c^i\rangle$:

$$(\mathbf{E}-c)\langle a_0c^i\rangle = \mathbf{E}\langle a_0c^i\rangle - c\langle a_0c^i\rangle = \langle a_0c^{i+1}\rangle - \langle c\cdot a_0c^i\rangle = \langle a_0c^{i+1} - a_0c^{i+1}\rangle = \langle 0\rangle$$

To see that this is the only operator of this form that annihilates A, let's see the effect of operator $(\mathbf{E}-d)$ for some $d \neq c$:

$$(\mathbf{E}-d)\langle a_0c^i\rangle = \mathbf{E}\langle a_0c^i\rangle - d\langle a_0c^i\rangle = \langle a_0c^{i+1} - da_0c^i\rangle = \langle (c-d)a_0c^i\rangle = (c-d)\langle a_0c^i\rangle$$

So we have a more rigorous confirmation that an annihilator annihilates exactly one type of sequence, but multiplies other similar sequences by a constant.

We can use this fact about annihilators of geometric sequences to solve certain recurrences. For example, consider the sequence $R = \langle r_0, r_1, r_2, \ldots\rangle$ defined recursively as follows:

$$\begin{aligned} r_0 &= 3 \\ r_{i+1} &= 5r_i \end{aligned}$$

We can easily prove that the operator $(\mathbf{E}-5)$ annihilates R:

$$(\mathbf{E}-5)\langle r_i\rangle = \mathbf{E}\langle r_i\rangle - 5\langle r_i\rangle = \langle r_{i+1}\rangle - \langle 5r_i\rangle = \langle r_{i+1} - 5r_i\rangle = \langle 0\rangle$$

Since $(\mathbf{E}-5)$ is an annihilator for R, we must have the closed form solution $r_i = r_0 5^i = 3\cdot 5^i$. We can easily verify this by induction, as follows:

$$\begin{aligned} r_0 &= 3\cdot 5^0 = 3 && \text{[definition]} \\ r_i &= 5r_{i-1} && \text{[definition]} \\ &= 5\cdot(3\cdot 5^{i-1}) && \text{[induction hypothesis]} \\ &= 5^i\cdot 3 && \text{[algebra]} \end{aligned}$$

1.2.2 Multiple operators

An operator is a function that transforms one sequence into another. Like any other function, we can apply operators one after another to the same sequence. For example, we can multiply a sequence $\langle a_i\rangle$ by a constant d and then by a constant c, resulting in the sequence $c(d\langle a_i\rangle) = \langle c\cdot d\cdot a_i\rangle = (cd)\langle a_i\rangle$. Alternatively, we may multiply the sequence by a constant c and then shift it to the left to get $\mathbf{E}(c\langle a_i\rangle) = \mathbf{E}\langle c\cdot a_i\rangle = \langle c\cdot a_{i+1}\rangle$.

This is exactly the same as applying the operators in the reverse order: $c(\mathbf{E}\langle a_i\rangle) = c\langle a_{i+1}\rangle = \langle c \cdot a_{i+1}\rangle$. We can also shift the sequence twice to the left: $\mathbf{E}(\mathbf{E}\langle a_i\rangle) = \mathbf{E}\langle a_{i+1}\rangle = \langle a_{i+2}\rangle$. We will write this in shorthand as $\mathbf{E}^2\langle a_i\rangle$. More generally, the operator $\mathbf{E}^k$ shifts a sequence k steps to the left: $\mathbf{E}^k\langle a_i\rangle = \langle a_{i+k}\rangle$.

We now have the tools to solve a whole host of recurrence problems. For example, what annihilates $C = \langle 2^i + 3^i\rangle$? Well, we know that $(\mathbf{E} - 2)$ annihilates $\langle 2^i\rangle$ while leaving $\langle 3^i\rangle$ essentially unscathed. Similarly, $(\mathbf{E} - 3)$ annihilates $\langle 3^i\rangle$ while leaving $\langle 2^i\rangle$ essentially unscathed. Thus, if we apply both operators one after the other, we see that $(\mathbf{E} - 2)(\mathbf{E} - 3)$ annihilates our sequence C.

In general, for any integers $a \neq b$, the operator $(\mathbf{E}-a)(\mathbf{E}-b)$ annihilates any sequence of the form $\langle c_1a^i+c_2b^i\rangle$ but nothing else. We will often 'multiply out' the operators into the shorthand notation $\mathbf{E}^2 - (a+b)\mathbf{E} + ab$. It is left as an exhilarating exercise to the student to verify that this shorthand actually makes sense—the operators $(\mathbf{E} - a)(\mathbf{E} - b)$ and $\mathbf{E}^2 - (a+b)\mathbf{E} + ab$ have the same effect on every sequence.

We now know finally enough to solve the recurrence for Fibonacci numbers. Specifically, notice that the recurrence $F_i = F_{i-1} + F_{i-2}$ is annihilated by $\mathbf{E}^2 - \mathbf{E} - 1$:

$$\begin{aligned}(\mathbf{E}^2 - \mathbf{E} - 1)\langle F_i\rangle &= \mathbf{E}^2\langle F_i\rangle - \mathbf{E}\langle F_i\rangle - \langle F_i\rangle \\ &= \langle F_{i+2}\rangle - \langle F_{i+1}\rangle - \langle F_i\rangle \\ &= \langle F_{i+2} - F_{i+1} - F_i\rangle \\ &= \langle 0\rangle\end{aligned}$$

Factoring $\mathbf{E}^2 - \mathbf{E} - 1$ using the quadratic formula, we obtain

$$\mathbf{E}^2 - \mathbf{E} - 1 = (\mathbf{E} - \phi)(\mathbf{E} - \hat{\phi})$$

where $\phi = (1 + \sqrt{5})/2 \approx 1.618034$ is the golden ratio and $\hat{\phi} = (1 - \sqrt{5})/2 = 1 - \phi = -1/\phi$. Thus, the operator $(E - \phi)(E - \hat{\phi})$ annihilates the Fibonacci sequence, so F_i must have the form

$$F_i = c\phi^i + \hat{c}\hat{\phi}^i$$

for some constants c and $\hat{c}$. We call this the *generic solution* to the recurrence, since it doesn't depend at all on the base cases. To compute the constants c and $\hat{c}$, we use the base cases $F_0 = 0$ and $F_1 = 1$ to obtain a pair of linear equations:

$$\begin{aligned}F_0 = 0 &= c + \hat{c} \\ F_1 = 1 &= c\phi + \hat{c}\hat{\phi}\end{aligned}$$

Solving this system of equations gives us $c = 1/(2\phi - 1) = 1/\sqrt{5}$ and $\hat{c} = -1/\sqrt{5}$.

We now have a closed-form expression for the ith Fibonacci number:

$$\boxed{F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}} = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^i - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^i}$$

With all the square roots in this formula, it's quite amazing that Fibonacci numbers are integers. However, if we do all the math correctly, all the square roots cancel out when i is an integer. (In fact, this is pretty easy to prove using the binomial theorem.)

1.2.3 Degenerate cases

We can't quite solve *every* recurrence yet. In our above formulation of $(\mathbf{E} - a)(\mathbf{E} - b)$, we assumed that $a \neq b$. What about the operator $(\mathbf{E} - a)(\mathbf{E} - a) = (\mathbf{E} - a)^2$? It turns out that this operator annihilates

sequences such as $\langle ia^i \rangle$:

$$\begin{aligned}(\mathbf{E}-a)\langle ia^i\rangle &= \langle (i+1)a^{i+1} - (a)ia^i\rangle \\ &= \langle (i+1)a^{i+1} - ia^{i+1}\rangle \\ &= \langle a^{i+1}\rangle \\ (\mathbf{E}-a)^2\langle ia^i\rangle &= (\mathbf{E}-a)\langle a^{i+1}\rangle = \langle 0\rangle\end{aligned}$$

More generally, the operator $(\mathbf{E}-a)^k$ annihilates any sequence $\langle p(i)\cdot a^i\rangle$, where $p(i)$ is any polynomial in i of degree $k-1$. As an example, $(\mathbf{E}-1)^3$ annihilates the sequence $\langle i^2\cdot 1^i\rangle = \langle i^2\rangle = \langle 1,4,9,16,25,\ldots\rangle$, since $p(i)=i^2$ is a polynomial of degree $n-1=2$.

As a review, try to explain the following statements:

- $(\mathbf{E}-1)$ annihilates any constant sequence $\langle\alpha\rangle$.
- $(\mathbf{E}-1)^2$ annihilates any arithmetic sequence $\langle\alpha+\beta i\rangle$.
- $(\mathbf{E}-1)^3$ annihilates any quadratic sequence $\langle\alpha+\beta i+\gamma i^2\rangle$.
- $(\mathbf{E}-3)(\mathbf{E}-2)(\mathbf{E}-1)$ annihilates any sequence $\langle\alpha+\beta 2^i+\gamma 3^i\rangle$.
- $(\mathbf{E}-3)^2(\mathbf{E}-2)(\mathbf{E}-1)$ annihilates any sequence $\langle\alpha+\beta 2^i+\gamma 3^i+\delta i3^i\rangle$.

1.2.4 Summary

In summary, we have learned several operators that act on sequences, as well as a few ways of combining operators.

Operator	Definition
Addition	$\langle a_i\rangle + \langle b_i\rangle = \langle a_i + b_i\rangle$
Subtraction	$\langle a_i\rangle - \langle b_i\rangle = \langle a_i - b_i\rangle$
Scalar multiplication	$c\langle a_i\rangle = \langle ca_i\rangle$
Shift	$\mathbf{E}\langle a_i\rangle = \langle a_{i+1}\rangle$
Composition of operators	$(\mathbf{X}+\mathbf{Y})\langle a_i\rangle = \mathbf{X}\langle a_i\rangle + \mathbf{Y}\langle a_i\rangle$
	$(\mathbf{X}-\mathbf{Y})\langle a_i\rangle = \mathbf{X}\langle a_i\rangle - \mathbf{Y}\langle a_i\rangle$
	$\mathbf{XY}\langle a_i\rangle = \mathbf{X}(\mathbf{Y}\langle a_i\rangle) = \mathbf{Y}(\mathbf{X}\langle a_i\rangle)$
k-fold shift	$\mathbf{E}^k\langle a_i\rangle = \langle a_{i+k}\rangle$

Notice that we have not defined a multiplication operator for two sequences. This is usually accomplished by *convolution*:

$$\langle a_i\rangle * \langle b_i\rangle = \Big\langle \sum_{j=0}^{i} a_j b_{i-j}\Big\rangle.$$

Fortunately, convolution is unnecessary for solving the recurrences we will see in this course.

We have also learned some things about annihilators, which can be summarized as follows:

Sequence	Annihilator
$\langle \alpha \rangle$	$\mathbf{E} - 1$
$\langle \alpha a^i \rangle$	$\mathbf{E} - a$
$\langle \alpha a^i + \beta b^i \rangle$	$(\mathbf{E} - a)(\mathbf{E} - b)$
$\langle \alpha_0 a_0^i + \alpha_1 a_1^i + \cdots + \alpha_n a_n^i \rangle$	$(\mathbf{E} - a_0)(\mathbf{E} - a_1) \cdots (\mathbf{E} - a_n)$
$\langle \alpha i + \beta \rangle$	$(\mathbf{E} - 1)^2$
$\langle (\alpha i + \beta) a^i \rangle$	$(\mathbf{E} - a)^2$
$\langle (\alpha i + \beta) a^i + \gamma b^i \rangle$	$(\mathbf{E} - a)^2 (\mathbf{E} - b)$
$\langle (\alpha_0 + \alpha_1 i + \cdots \alpha_{n-1} i^{n-1}) a^i \rangle$	$(\mathbf{E} - a)^n$
If $\mathbf{X}$ annihilates $\langle a_i \rangle$, then $\mathbf{X}$ also annihilates $c\langle a_i \rangle$ for any constant c.	
If $\mathbf{X}$ annihilates $\langle a_i \rangle$ and $\mathbf{Y}$ annihilates $\langle b_i \rangle$, then $\mathbf{XY}$ annihilates $\langle a_i \rangle \pm \langle b_i \rangle$.	

1.3 Solving Linear Recurrences

1.3.1 Homogeneous Recurrences

The general expressions in the annihilator box above are really the most important things to remember about annihilators because they help you to solve any recurrence for which you can write down an annihilator. The general method is:

1. Write down the annihilator for the recurrence
2. Factor the annihilator
3. Determine the sequence annihilated by each factor
4. Add these sequences together to form the generic solution
5. Solve for constants of the solution by using initial conditions

Example 2. *Let's show the steps required to solve the following recurrence:*

$$\begin{aligned} r_0 &= 1 \\ r_1 &= 5 \\ r_2 &= 17 \\ r_i &= 7r_{i-1} - 16r_{i-2} + 12r_{i-3} \end{aligned}$$

1. *Write down the annihilator. Since* $r_{i+3} - 7r_{i+2} + 16r_{i+1} - 12r_i = 0$*, the annihilator is* $\mathbf{E}^3 - 7\mathbf{E}^2 + 16\mathbf{E} - 12$.
2. *Factor the annihilator.* $\mathbf{E}^3 - 7\mathbf{E}^2 + 16\mathbf{E} - 12 = (\mathbf{E} - 2)^2(\mathbf{E} - 3)$.
3. *Determine sequences annihilated by each factor.* $(\mathbf{E} - 2)^2$ *annihilates* $\langle (\alpha i + \beta) 2^i \rangle$ *for any constants* α *and* β*, and* $(\mathbf{E} - 3)$ *annihilates* $\langle \gamma 3^i \rangle$ *for any constant* γ.
4. *Combine the sequences.* $(\mathbf{E} - 2)^2(\mathbf{E} - 3)$ *annihilates* $\langle (\alpha i + \beta) 2^i + \gamma 3^i \rangle$ *for any constants* α, β, γ.
5. *Solve for the constants. The base cases give us three equations in the three unknowns* α, β, γ*:*

$$\begin{aligned} r_0 = 1 &= (\alpha \cdot 0 + \beta) 2^0 + \gamma \cdot 3^0 = \beta + \gamma \\ r_1 = 5 &= (\alpha \cdot 1 + \beta) 2^1 + \gamma \cdot 3^1 = 2\alpha + 2\beta + 3\gamma \\ r_2 = 17 &= (\alpha \cdot 2 + \beta) 2^2 + \gamma \cdot 3^2 = 8\alpha + 4\beta + 9\gamma \end{aligned}$$

We can solve these equations to get $\alpha = 1$, $\beta = 0$, $\gamma = 1$. Thus, our final solution is $\boxed{r_i = i2^i + 3^i}$, *which we can verify by induction.*

1.3.2 Non-homogeneous Recurrences

A *height balanced tree* is a binary tree, where the heights of the two subtrees of the root differ by at most one, and both subtrees are also height balanced. To ground the recursive definition, the empty set is considered a height balanced tree of height -1, and a single node is a height balanced tree of height 0.

Let T_n be the smallest height-balanced tree of height n—how many nodes does T_n have? Well, one of the subtrees of T_n has height $n-1$ (since T_n has height n) and the other has height either $n-1$ or $n-2$ (since T_n is height-balanced and as small as possible). Since both subtrees are themselves height-balanced, the two subtrees must be T_{n-1} and T_{n-2}.

We have just derived the following recurrence for t_n, the number of nodes in the tree T_n:

$$\begin{aligned} t_{-1} &= 0 && \text{[the empty set]} \\ t_0 &= 1 && \text{[a single node]} \\ t_n &= t_{n-1} + t_{n-2} + 1 \end{aligned}$$

The final '+1' is for the root of T_n.

We refer to the terms in the equation involving t_i's as the *homogeneous* terms and the rest as the *non-homogeneous* terms. (If there were no non-homogeneous terms, we would say that the recurrence itself is homogeneous.) We know that $\mathbf{E}^2 - \mathbf{E} - 1$ annihilates the homogeneous part $t_n = t_{n-1} + t_{n-2}$. Let us try applying this annihilator to the entire equation:

$$\begin{aligned} (\mathbf{E}^2 - \mathbf{E} - 1)\langle t_i \rangle &= \mathbf{E}^2\langle t_i \rangle - \mathbf{E}\langle t_i \rangle - 1\langle t_i \rangle \\ &= \langle t_{i+2} \rangle - \langle t_{i+1} \rangle - \langle t_i \rangle \\ &= \langle t_{i+2} - t_{i+1} - t_i \rangle \\ &= \langle 1 \rangle \end{aligned}$$

The leftover sequence $\langle 1, 1, 1, \ldots \rangle$ is called the *residue*. To obtain the annihilator for the entire recurrence, we compose the annihilator for its homogeneous part with the annihilator of its residue. Since $\mathbf{E} - 1$ annihilates $\langle 1 \rangle$, it follows that $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 1)$ annihilates $\langle t_n \rangle$. We can factor the annihilator into

$$(\mathbf{E} - \phi)(\mathbf{E} - \hat{\phi})(\mathbf{E} - 1),$$

so our annihilator rules tell us that

$$t_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma$$

for some constants α, β, γ. We call this the *generic solution* to the recurrence. Different recurrences can have the same generic solution.

To solve for the unknown constants, we need three equations in three unknowns. Our base cases give us two equations, and we can get a third by examining the next nontrivial case $t_1 = 2$:

$$\begin{aligned} t_{-1} = 0 &= \alpha\phi^{-1} + \beta\hat{\phi}^{-1} + \gamma = \alpha/\phi + \beta/\hat{\phi} + \gamma \\ t_0 = 1 &= \alpha\phi^0 + \beta\hat{\phi}^0 + \gamma = \alpha + \beta + \gamma \\ t_1 = 2 &= \alpha\phi^1 + \beta\hat{\phi}^1 + \gamma = \alpha\phi + \beta\hat{\phi} + \gamma \end{aligned}$$

Solving these equations, we find that $\alpha = \frac{\sqrt{5}+2}{\sqrt{5}}$, $\beta = \frac{\sqrt{5}-2}{\sqrt{5}}$, and $\gamma = -1$. Thus,

$$\boxed{t_n = \frac{\sqrt{5}+2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{\sqrt{5}-2}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n - 1}$$

Here is the general method for non-homogeneous recurrences:

1. Write down the homogeneous annihilator, directly from the recurrence
1½. 'Multiply' by the annihilator for the residue
2. Factor the annihilator
3. Determine what sequence each factor annihilates
4. Add these sequences together to form the generic solution
5. Solve for constants of the solution by using initial conditions

1.3.3 Some more examples

In each example below, we use the base cases $a_0 = 0$ and $a_1 = 1$.

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + 2}$
  - The homogeneous annihilator is $\mathbf{E}^2 - \mathbf{E} - 1$.
  - The residue is the constant sequence $\langle 2, 2, 2, \ldots \rangle$, which is annihilated by $\mathbf{E} - 1$.
  - Thus, the annihilator is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 1)$.
  - The annihilator factors into $(\mathbf{E} - \phi)(\mathbf{E} - \hat{\phi})(\mathbf{E} - 1)$.
  - Thus, the generic solution is $a_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma$.
  - The constants α, β, γ satisfy the equations

$$\begin{aligned} a_0 = 0 &= \alpha + \beta + \gamma \\ a_1 = 1 &= \alpha\phi + \beta\hat{\phi} + \gamma \\ a_2 = 3 &= \alpha\phi^2 + \beta\hat{\phi}^2 + \gamma \end{aligned}$$

  - Solving the equations gives us $\alpha = \frac{\sqrt{5}+2}{\sqrt{5}}$, $\beta = \frac{\sqrt{5}-2}{\sqrt{5}}$, and $\gamma = -2$
  - So the final solution is $\boxed{a_n = \frac{\sqrt{5}+2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{\sqrt{5}-2}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n - 2}$

  (In the remaining examples, I won't explicitly enumerate the steps like this.)

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + 3}$

  The homogeneous annihilator $(\mathbf{E}^2 - \mathbf{E} - 1)$ leaves a constant residue $\langle 3, 3, 3, \ldots \rangle$, so the annihilator is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 1)$, and the generic solution is $a_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma$. Solving the equations

$$\begin{aligned} a_0 = 0 &= \alpha + \beta + \gamma \\ a_1 = 1 &= \alpha\phi + \beta\hat{\phi} + \gamma \\ a_2 = 4 &= \alpha\phi^2 + \beta\hat{\phi}^2 + \gamma \end{aligned}$$

  gives us the final solution $\boxed{a_n = \frac{\sqrt{5}+3}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{\sqrt{5}-3}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n - 3}$

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + 2^n}$

  The homogeneous annihilator $(\mathbf{E}^2 - \mathbf{E} - 1)$ leaves an exponential residue $\langle 4, 8, 16, 32, \ldots \rangle = \langle 2^{i+2} \rangle$, which is annihilated by $\mathbf{E} - 2$. Thus, the annihilator is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 2)$, and the generic solution is $a_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma 2^n$. The constants α, β, γ satisfy the following equations:

$$\begin{aligned} a_0 = 0 &= \alpha + \beta + \gamma \\ a_1 = 1 &= \alpha\phi + \beta\hat{\phi} + 2\gamma \\ a_2 = 5 &= \alpha\phi^2 + \beta\hat{\phi}^2 + 4\gamma \end{aligned}$$

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + n}$

  The homogeneous annihilator $(\mathbf{E}^2 - \mathbf{E} - 1)$ leaves a linear residue $\langle 2, 3, 4, 5 \ldots \rangle = \langle i + 2 \rangle$, which is annihilated by $(\mathbf{E} - 1)^2$. Thus, the annihilator is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 1)^2$, and the generic solution is $a_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma + \delta n$. The constants $\alpha, \beta, \gamma, \delta$ satisfy the following equations:

$$\begin{aligned} a_0 = 0 &= \alpha + \beta + \gamma \\ a_1 = 1 &= \alpha\phi + \beta\hat{\phi} + \gamma + \delta \\ a_2 = 3 &= \alpha\phi^2 + \beta\hat{\phi}^2 + \gamma + 2\delta \\ a_3 = 7 &= \alpha\phi^3 + \beta\hat{\phi}^3 + \gamma + 3\delta \end{aligned}$$

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + n^2}$

  The homogeneous annihilator $(\mathbf{E}^2 - \mathbf{E} - 1)$ leaves a quadratic residue $\langle 4, 9, 16, 25 \ldots \rangle = \langle (i + 2)^2 \rangle$, which is annihilated by $(\mathbf{E} - 1)^3$. Thus, the annihilator is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 1)^3$, and the generic solution is $a_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma + \delta n + \epsilon n^2$. The constants $\alpha, \beta, \gamma, \delta, \epsilon$ satisfy the following equations:

$$\begin{aligned} a_0 = 0 &= \alpha + \beta + \gamma \\ a_1 = 1 &= \alpha\phi + \beta\hat{\phi} + \gamma + \delta + \epsilon \\ a_2 = 5 &= \alpha\phi^2 + \beta\hat{\phi}^2 + \gamma + 2\delta + 4\epsilon \\ a_3 = 15 &= \alpha\phi^3 + \beta\hat{\phi}^3 + \gamma + 3\delta + 9\epsilon \\ a_4 = 36 &= \alpha\phi^4 + \beta\hat{\phi}^4 + \gamma + 4\delta + 16\epsilon \end{aligned}$$

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + n^2 - 2^n}$

  The homogeneous annihilator $(\mathbf{E}^2 - \mathbf{E} - 1)$ leaves the residue $\langle (i + 2)^2 - 2^{i-2} \rangle$. The quadratic part of the residue is annihilated by $(\mathbf{E} - 1)^3$, and the exponential part is annihilated by $(\mathbf{E} - 2)$. Thus, the annihilator for the whole recurrence is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - 1)^3(\mathbf{E} - 2)$, and so the generic solution is $a_n = \alpha\phi^n + \beta\hat{\phi}^n + \gamma + \delta n + \epsilon n^2 + \eta 2^i$. The constants $\alpha, \beta, \gamma, \delta, \epsilon, \eta$ satisfy a system of six equations in six unknowns determined by $a_0, a_1, \ldots, a_5$.

- $\boldsymbol{a_n = a_{n-1} + a_{n-2} + \phi^n}$

  The annihilator is $(\mathbf{E}^2 - \mathbf{E} - 1)(\mathbf{E} - \phi) = (\mathbf{E} - \phi)^2(\mathbf{E} - \hat{\phi})$, so the generic solution is $a_n = \alpha\phi^n + \beta n\phi^n + \gamma\hat{\phi}^n$. (Other recurrence solving methods will have a "interference" problem with this equation, while the operator method does not.)

Our method does not work on recurrences like $a_n = a_{n-1} + \frac{1}{n}$ or $a_n = a_{n-1} + \lg n$, because the functions $\frac{1}{n}$ and $\lg n$ do not have annihilators. Our tool, as it stands, is limited to linear recurrences.

1.4 Divide and Conquer Recurrences

Divide and conquer algorithms often give us running-time recurrences of the form

$$T(n) = aT(n/b) + f(n) \tag{1}$$

where a and b are constants and $f(n)$ is some other function. The so-called 'Master Theorem' gives us a general method for solving such recurrences when $f(n)$ is a simple polynomial.

Unfortunately, the Master Theorem doesn't work for all functions $f(n)$, and many useful recurrences don't look like (1) at all. Fortunately, there's a more general technique to solve most divide-and-conquer recurrences, even if they don't have this form. This technique is used to *prove* the Master Theorem, so if you remember this technique, you can forget the Master Theorem entirely (which is what I did). Throw off your chains!

We illustrate the technique using the generic recurrence (1). We start by drawing a *recursion tree*, shown in Figure 1. The root of the recursion tree is a box containing the value $f(n)$, it has a children, each of which is the root of a recursion tree for $T(n/b)$. Equivalently, a recursion tree is a complete a-ary tree where each node at depth i contains the value $a^i f(n/b^i)$. The recursion stops when we get to the base case(s) of the recurrence. Since we're looking for asymptotic bounds, it turns out not to matter much what we use for the base case; for purposes of illustration, assume that $T(1) = f(1)$.

Now $T(n)$ is just the sum of all values stored in the tree. Assuming that each level of the tree is full, we have

$$T(n) = f(n) + a\,f(n/b) + a^2\,f(n/b^2) + \cdots + a^i\,f(n/b^i) + \cdots + a^L\,f(n/b^L)$$

where L is the depth of the recursion tree. We easily see that $L = \log_b n$, since $n/b^L = 1$. Since $f(1) = \Theta(1)$, the last non-zero term in the summation is $\Theta(a^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$.

Now we can easily state and prove the Master Theorem, in a slightly different form than it's usually stated.

The Master Theorem. The recurrence $T(n) = aT(n/b) + f(n)$ can be solved as follows.

- If $af(n/b)/f(n) < 1$, then $T(n) = \Theta(f(n))$.
- If $af(n/b)/f(n) > 1$, then $T(n) = \Theta(n^{\log_b a})$.
- If $af(n/b)/f(n) = 1$, then $T(n) = \Theta(f(n)\log_b n)$.
- If none of these three cases apply, you're on your own.

Proof. If $f(n)$ is a *constant factor larger* than $af(n/b)$, then by induction, the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$.

If $f(n)$ is a *constant factor smaller* than $af(n/b)$, then by induction, the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$.

Finally, if $af(n/b) = f(n)$, then by induction, each of the $L+1$ terms in the summation is equal to $f(n)$. □

Here are a few canonical examples of the Master Theorem in action:

- **Randomized selection: $T(n) = T(3n/4) + n$**

  Here $a\,f(n/b) = 3n/4$ is smaller than $f(n) = n$ by a factor of $4/3$, so $\boxed{T(n) = \Theta(n)}$

- **Karatsuba's multiplication algorithm: $T(n) = 3T(n/2) + n$**

  Here $a\,f(n/b) = 3n/2$ is bigger than $f(n) = n$ by a factor of $3/2$, so $\boxed{T(n) = \Theta(n^{\log_2 3})}$

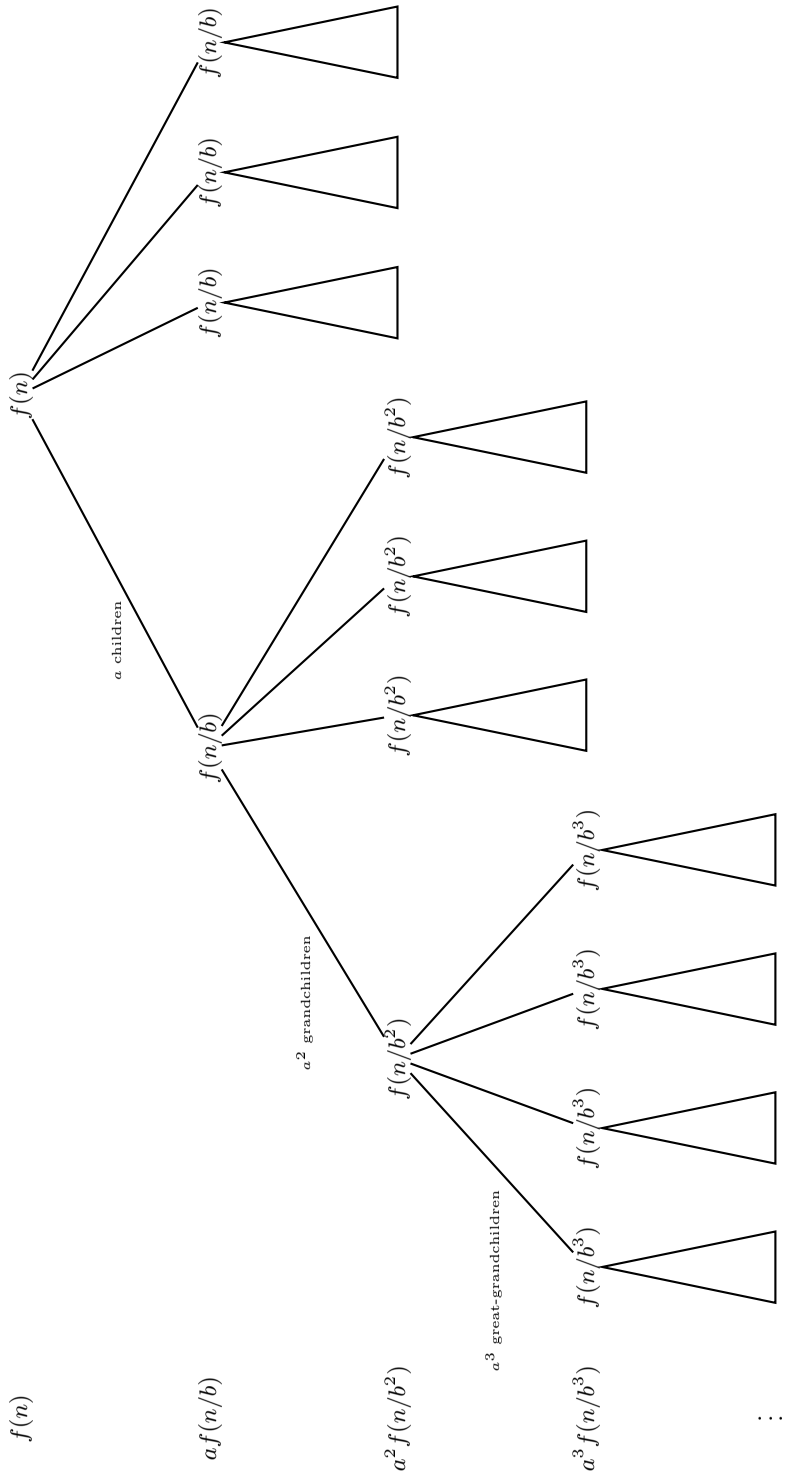


Figure 1: A recursion tree for the recurrence $T(n) = aT(n/b) + f(n)$. The tree has height $L = \log_b n$.

- **Mergesort: $T(n) = 2T(n/2) + n$**

  Here $a\,f(n/b) = f(n)$, so $\boxed{T(n) = \Theta(n \log n)}$

- **$T(n) = 4T(n/2) + n \lg n$**

  In this case, we have $a\,f(n/b) = 2n \lg n - 2n$, which is not quite twice $f(n) = n \lg n$. However, for sufficiently large n (which is all we care about with asymptotic bounds) we have $2f(n) > a\,f(n/b) > 1.9f(n)$. Since the level sums are bounded both above and below by ascending geometric series, the solution is $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$. (This trick will *not* work in the second or third cases of the Master Theorem!)

Using the same recursion-tree technique, we can also solve recurrences where the Master Theorem doesn't apply.

- **$T(n) = 2T(n/2) + n/\lg n$**

  We can't apply the Master Theorem here, because $a\,f(n/b) = n/(\lg n - 1)$ isn't equal to $f(n) = n/\lg n$, but the difference isn't a constant factor. So we need to compute each of the level sums and compute their total in some other way. It's not hard to see that the sum of all the nodes in the ith level is $n/(\lg n - i)$. In particular, this means the depth of the tree is at most $\lg n - 1$.

$$T(n) = \sum_{i=0}^{\lg n - 1} \frac{n}{\lg n - i} = \sum_{j=1}^{\lg n} \frac{n}{j} = nH_{\lg n} = \boxed{\Theta(n \lg \lg n)}$$

- **Randomized quicksort: $T(n) = T(3n/4) + T(n/4) + n$**

  In this case, nodes in the same level of the recursion tree have different values. This makes the tree lopsided; different leaves are at different levels. However, it's not to hard to see that the nodes in any *complete* level (that is, above any of the leaves) sum to n, so this is like the last case of the Master Theorem, and that every leaf has depth between $\log_4 n$ and $\log_{4/3} n$. To derive an upper bound, we overestimate $T(n)$ by ignoring the base cases and extending the tree downward to the level of the *deepest* leaf. Similarly, to derive a lower bound, we overestimate $T(n)$ by counting only nodes in the tree up to the level of the *shallowest* leaf. These observations give us the upper and lower bounds $n \log_4 n \le T(n) \le n \log_{4/3} n$. Since these bound differ by only a constant factor, we have $\boxed{T(n) = \Theta(n \log n)}$.

- **Deterministic selection: $T(n) = T(n/5) + T(7n/10) + n$**

  Again, we have a lopsided recursion tree. If we look only at complete levels of the tree, we find that the level sums form a descending geometric series $T(n) = n + 9n/10 + 81n/100 + \cdots$, so this is like the first case of the master theorem. We can get an upper bound by ignoring the base cases entirely and growing the tree out to infinity, and we can get a lower bound by only counting nodes in complete levels. Either way, the geometric series is dominated by its largest term, so $\boxed{T(n) = \Theta(n)}$.

- **$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$**

  In this case, we have a complete recursion tree, but the *degree* of the nodes is no longer constant, so we have to be a bit more careful. It's not hard to see that the nodes in any level sum to n, so this is like the third case of the Master Theorem. The depth L satisfies the identity $n^{2^{-L}} = 2$ (we can't get all the way down to 1 by taking square roots), so $L = \lg \lg n$ and $\boxed{T(n) = \Theta(n \lg \lg n)}$.

- $\boldsymbol{T(n) = 4\sqrt{n} \cdot T(\sqrt{n}) + n}$

  We still have at most $\lg \lg n$ levels, but now the nodes in level i sum to $4^i n$. We have an increasing geometric series of level sums, like the second Master case, so $T(n)$ is dominated by the sum over the deepest level: $T(n) = \Theta(4^{\lg \lg n} n) = \boxed{\Theta(n \log^2 n)}$

1.5 Transforming Recurrences

1.5.1 An analysis of mergesort: domain transformation

Previously we gave the recurrence for mergesort as $T(n) = 2T(n/2) + n$, and obtained the solution $T(n) = \Theta(n \log n)$ using the Master Theorem (or the recursion tree method if you, like me, can't remember the Master Theorem). This is fine is n is a power of two, but for other values values of n, this recurrence is incorrect. When n is odd, then the recurrence calls for us to sort a fractional number of elements! Worse yet, if n is not a power of two, we will *never* reach the base case $T(1) = 0$.

To get a recurrence that's valid for *all* integers n, we need to carefully add ceilings and floors:

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n.$$

We have almost no hope of getting an exact solution here; the floors and ceilings will eventually kill us. So instead, let's just try to get a tight asymptotic upper bound for $T(n)$ using a technique called *domain transformation*. A domain transformation rewrites a function $T(n)$ with a difficult recurrence as a nested function $S(f(n))$, where $f(n)$ is a simple function and $S()$ has an easier recurrence.

First we overestimate the time bound, once by pretending that the two subproblem sizes are equal, and again to eliminate the ceiling:

$$T(n) \le 2T\big(\lceil n/2 \rceil\big) + n \le 2T(n/2 + 1) + n.$$

Now we define a new function $S(n) = T(n + \alpha)$, where α is a unknown constant, chosen so that $S(n)$ satisfies the Master-ready recurrence $S(n) \le 2S(n/2) + O(n)$. To figure out the correct value of α, we compare two versions of the recurrence for the function $T(n + \alpha)$:

$$\begin{array}{rcll} S(n) & \le & 2S(n/2) + O(n) & \text{implies} \quad T(n+\alpha) \le 2T(n/2 + \alpha) + O(n) \\ T(n) & \le & 2T(n/2 + 1) + n & \text{implies} \quad T(n+\alpha) \le 2T((n+\alpha)/2 + 1) + n + \alpha \end{array}$$

For these two recurrences to be equal, we need $n/2 + \alpha = (n + \alpha)/2 + 1$, which implies that $\alpha = 2$. The Master Theorem now tells us that $S(n) = O(n \log n)$, so

$$T(n) = S(n - 2) = O((n - 2) \log(n - 2)) = O(n \log n).$$

A similar argument gives a matching lower bound $T(n) = \Omega(n \log n)$. So $\boxed{T(n) = \Theta(n \log n)}$ after all, just as though we had ignored the floors and ceilings from the beginning!

Domain transformations are useful for removing floors, ceilings, and lower order terms from the arguments of any recurrence that otherwise looks like it ought to fit either the Master Theorem or the recursion tree method. But now that we know this, we don't need to bother grinding through the actual gory details!

1.5.2 A less trivial example

There is a data structure in computational geometry called *ham-sandwich trees*, where the cost of doing a certain search operation obeys the recurrence $\boldsymbol{T(n) = T(n/2) + T(n/4) + 1}$. This doesn't fit the Master

theorem, because the two subproblems have different sizes, and using the recursion tree method only gives us the loose bounds $\sqrt{n} \ll T(n) \ll n$.

Domain transformations save the day. If we define the new function $t(k) = T(2^k)$, we have a new recurrence

$$t(k) = t(k-1) + t(k-2) + 1$$

which should immediately remind you of Fibonacci numbers. Sure enough, after a bit of work, the annihilator method gives us the solution $t(k) = \Theta(\phi^k)$, where $\phi = (1+\sqrt{5})/2$ is the golden ratio. This implies that

$$T(n) = t(\lg n) = \Theta(\phi^{\lg n}) = \boxed{\Theta(n^{\lg \phi})} \approx \Theta(n^{0.69424}).$$

It's possible to solve this recurrence without domain transformations and annihilators—in fact, the inventors of ham-sandwich trees did so—but it's much more difficult.

1.5.3 Secondary recurrences

Consider the recurrence $\boldsymbol{T(n) = 2T(\frac{n}{3} - 1) + n}$ with the base case $T(1) = 1$. We already know how to use domain transformations to get the tight asymptotic bound $T(n) = \Theta(n)$, but how would we we obtain an *exact* solution?

First we need to figure out how the parameter n changes as we get deeper and deeper into the recurrence. For this we use a *secondary recurrence*. We define a sequence n_i so that

$$T(n_i) = 2T(n_{i-1}) + n_i,$$

So n_i is the argument of $T()$ when we are i recursion steps away from the base case $n_0 = 1$. The original recurrence gives us the following secondary recurrence for n_i:

$$n_{i-1} = \frac{n_i}{3} - 1 \text{ implies } n_i = 3n_{i-1} + 3.$$

The annihilator for this recurrence is $(\mathbf{E}-1)(\mathbf{E}-3)$, so the generic solution is $n_i = \alpha 3^i + \beta$. Plugging in the base cases $n_0 = 1$ and $n_1 = 6$, we get the exact solution

$$n_i = \frac{5}{2} \cdot 3^i - \frac{3}{2}.$$

Notice that our original function $T(n)$ is only well-defined if $n = n_i$ for some integer $i \geq 0$.

Now to solve the original recurrence, we do a range transformation. If we set $t_i = T(n_i)$, we have the recurrence $t_i = 2t_{i-1} + \frac{5}{2} \cdot 3^i - \frac{3}{2}$, which by now we can solve using the annihilator method. The annihilator of the recurrence is $(\mathbf{E}-2)(\mathbf{E}-3)(\mathbf{E}-1)$, so the generic solution is $\alpha' 3^i + \beta' 2^i + \gamma'$. Plugging in the base cases $t_0 = 1$, $t_1 = 8$, $t_2 = 37$, we get the exact solution

$$t_i = \frac{15}{2} \cdot 3^i - 8 \cdot 2^i + \frac{3}{2}$$

Finally, we need to substitute to get a solution for the original recurrence in terms of n, by inverting the solution of the secondary recurrence. If $n = n_i = \frac{5}{2} \cdot 3^i - \frac{3}{2}$, then (after a little algebra) we have

$$i = \log_3 \left(\frac{2}{5} n + \frac{3}{5} \right).$$

Substituting this into the expression for t_i, we get our exact, closed-form solution.

$$\begin{aligned} T(n) &= \frac{15}{2} \cdot 3^i - 8 \cdot 2^i + \frac{3}{2} \\ &= \frac{15}{2} \cdot 3^{\left(\frac{2}{5}n+\frac{3}{5}\right)} - 8 \cdot 2^{\log_3\left(\frac{2}{5}n+\frac{3}{5}\right)} + \frac{3}{2} \\ &= \frac{15}{2}\left(\frac{2}{5}n+\frac{3}{5}\right) - 8 \cdot \left(\frac{2}{5}n+\frac{3}{5}\right)^{\log_3 2} + \frac{3}{2} \\ &= 3n - 8 \cdot \left(\frac{2}{5}n+\frac{3}{5}\right)^{\log_3 2} + 6 \end{aligned}$$

Isn't that special? Now you know why we stick to asymptotic bounds for most recurrences.

1.6 The Ultimate Method: Guess and Confirm

Ultimately, there is one failsafe method to solve *any* recurrence:

Guess the answer, and then prove it correct by induction.

The annihilator method, the recursion-tree method, and transformations are good ways to generate guesses that are guaranteed to be correct, provided you use them correctly. But if you're faced with a recurrence that doesn't seem to fit any of these methods, or if you've forgotten how those techniques work, don't despair! If you guess a closed-form solution and then try to verify your guess inductively, usually either the proof succeeds and you're done, or the proof fails in a way that lets you refine your guess. Where you get your initial guess is utterly irrelevant[3]—from a classmate, from a textbook, on the web, from the answer to a different problem, scrawled on a bathroom wall in Siebel, dictated by the machine elves, whatever. If you can prove that the answer is correct, then it's correct!

1.6.1 Tower of Hanoi

The classical Tower of Hanoi problem gives us the recurrence $\boldsymbol{T(n) = 2T(n-1) + 1}$ with base case $\boldsymbol{T(0) = 0}$. Just looking at the recurrence we can guess that $T(n)$ is something like 2^n. If we write out the first few values of $T(n)$ all the values are one less than a power of two.

$$T(0) = 0,\ T(1) = 1,\ T(2) = 3,\ T(3) = 7,\ T(4) = 15,\ T(5) = 31,\ T(6) = 63, \ldots,$$

It looks like $\boxed{\boldsymbol{T(n) = 2^n - 1}}$ might be the right answer. Let's check.

$$\begin{aligned} T(0) &= 0 = 2^0 - 1 \\ T(n) &= 2T(n-1) + 1 \\ &= 2(2^{n-1} - 1) + 1 \quad \text{[induction hypothesis]} \\ &= 2^n - 1 \quad \text{[algebra]} \end{aligned}$$

We were right!

[3] . . . except of course during exams, where you aren't supposed to use *any* outside sources

1.6.2 Fibonacci numbers

Let's try a less trivial example: the Fibonacci numbers $\boldsymbol{F_n = F_{n-1} + F_{n-2}}$ with base cases $\boldsymbol{F_0 = 0}$ and $\boldsymbol{F_1 = 1}$. There is no obvious pattern (besides the obvious one) in the first several values, but we can reasonably guess that F_n is exponential in n. Let's try to prove that $F_n \le a \cdot c^n$ for some constants $a > 0$ and $c > 1$ and see how far we get.

$$F_n \;=\; F_{n-1} + F_{n-2} \;\le\; a \cdot c^{n-1} + a \cdot c^{n-2} \;\le\; a \cdot c^n \;???$$

The last inequality is satisfied if $c^n \ge c^{n-1} + c^{n-2}$, or more simply, if $c^2 - c - 1 \ge 0$. The smallest value of c that works is $\phi = (1 + \sqrt{5})/2 \approx 1.618034$; the other root of the quadratic equation is negative, so we can ignore it.

So we have *most* of an inductive proof that $F_n \le a \cdot \phi^n$ for *any* constant a. All that we're missing are the base cases, which (we can easily guess) must determine the value of the coefficient a. We quickly compute

$$\frac{F_0}{\phi^0} = 0 \quad \text{and} \quad \frac{F_1}{\phi^1} = \frac{1}{\phi} \approx 0.618034 > 0,$$

so the base cases of our induction proof are correct as long as $a \ge 1/\phi$. It follows that $\boldsymbol{F_n \le \phi^{n-1}}$ for all $n \ge 0$.

What about a matching lower bound? Well, the same inductive proof implies that $F_n \ge b \cdot \phi^n$ for some constant b, but the only value of b that works for *all* n is the trivial $b = 0$. We could try to find some lower-order term that makes the base case non-trivial, but an easier approach is to recall that $\Omega()$ bounds only have to work for sufficiently large n. So let's ignore the trivial base case $F_0 = 0$ and assume that $F_2 = 1$ is a base case instead. Some more calculation gives us

$$\frac{F_2}{\phi^2} = \frac{1}{\phi^2} \approx 0.381966 < \frac{1}{\phi}.$$

Thus, the new base cases of our induction proof are correct as long as $b \le 1/\phi^2$, which implies that $\boldsymbol{F_n \ge \phi^{n-2}}$ for all $n \ge 1$.

Putting the upper and lower bounds together, we correctly conclude that $\boxed{\boldsymbol{F_n = \Theta(\phi^n)}}$. It *is* possible to get a more exact solution by speculatively refining and conforming our current bounds, but it's not easy; you're better off just using annihilators.

1.6.3 A divide-and-conquer example

Consider the divide-and-conquer recurrence $\boldsymbol{T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n}$. It doesn't fit into the form required by the Master Theorem, but it still sort of resembles the Mergesort recurrence—the total size of the subproblems at the first level of recursion is n—so let's *guess* that $T(n) = O(n \log n)$, and then try to prove that our guess is correct. Specifically, let's conjecture that $T(n) \le a\, n \lg n$ for all sufficiently large n and some constant a to be determined later:

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\le \sqrt{n} \cdot a \sqrt{n} \lg \sqrt{n} + n \quad \text{[induction hypothesis]} \\ &= (a/2) n \lg n + n \quad \text{[algebra]} \\ &\le a n \lg n \end{aligned}$$

The last inequality assumes only that $1 \le (a/2) \log n$,or equivalently, that $n \ge 2^{2/a}$. In other words, the induction proof is correct if n is sufficiently large. So we were right!

But before you break out the champagne, what about the multiplicative constant a? The proof worked for *any* constant a, no matter how small. This strongly suggests that our upper bound $T(n) = O(n \log n)$ is not tight. Indeed, if we try to prove a matching lower bound $T(n) \geq b\, n \log n$ for sufficiently large n, we run into trouble.

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\geq \sqrt{n} \cdot b\sqrt{n} \log \sqrt{n} + n \quad \text{[induction hypothesis]} \\ &= (b/2) n \log n + n \\ &\not\geq bn \log n \end{aligned}$$

The last inequality would be correct only if $1 > (b/2) \log n$, but that inequality is false for large values of n, no matter which constant b we choose. Okay, so $\Theta(n \log n)$ is too big. How about $\Theta(n)$? The lower bound is easy to prove directly:

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \geq n$$

But an inductive proof of the lower bound fails.

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\leq \sqrt{n} \cdot a\sqrt{n} + n \quad \text{[induction hypothesis]} \\ &= (a+1)n \quad \text{[algebra]} \\ &\not\leq an \end{aligned}$$

Hmmm. So what's bigger than n and smaller than $n \lg n$? How about $n\sqrt{\lg n}$?

$$\begin{aligned} T(n) &= \sqrt{n} \cdot T(\sqrt{n}) + n \\ &\leq \sqrt{n} \cdot a\sqrt{n}\sqrt{\lg \sqrt{n}} + n \quad \text{[induction hypothesis]} \\ &= (a/\sqrt{2})\, n\sqrt{\lg n} + n \quad \text{[algebra]} \\ &\leq a\, n\sqrt{\lg n} \quad \text{for large enough } n \end{aligned}$$

Okay, the upper bound checks out; how about the lower bound?

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\geq \sqrt{n} \cdot b\sqrt{n}\sqrt{\lg \sqrt{n}} + n \quad \text{[induction hypothesis]} \\ &= (b/\sqrt{2})\, n\sqrt{\lg n} + n \quad \text{[algebra]} \\ &\not\geq b\, n\sqrt{\lg n} \end{aligned}$$

No, the last step doesn't work. So $\Theta(n\sqrt{\lg n})$ doesn't work. Hmmm... what else is between n and $n \lg n$? How about $n \lg \lg n$?

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\leq \sqrt{n} \cdot a\sqrt{n} \lg \lg \sqrt{n} + n \quad \text{[induction hypothesis]} \\ &= a\, n \lg \lg n - a\, n + n \quad \text{[algebra]} \\ &\leq a\, n \lg \lg n \qquad \textbf{if } \boldsymbol{a \geq 1} \end{aligned}$$

Hey look at that! For once, our upper bound proof requires a constraint on the hidden constant a. This is an good indication that we've found the right answer. Let's try the lower bound:

$$\begin{aligned} T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n &\geq \sqrt{n} \cdot b\sqrt{n} \lg \lg \sqrt{n} + n \quad \text{[induction hypothesis]} \\ &= b\, n \lg \lg n - b\, n + n \quad \text{[algebra]} \\ &\geq b\, n \lg \lg n \qquad \textbf{if } \boldsymbol{b \leq 1} \end{aligned}$$

Hey, it worked! We have most of an inductive proof that $T(n) \leq an \lg \lg n$ for any $a \geq 1$ and most of an inductive proof that $T(n) \geq bn \lg \lg n$ for any $b \leq 1$. Technically, we're still missing the base cases in both proofs, but we can be fairly confident at this point that $\boxed{\boldsymbol{T(n) = \Theta(n \log \log n)}}$.

1.7 References

Methods for solving recurrences by annihilators, domain transformations, and secondary recurrences are nicely described in George Lueker's paper "Some techniques for solving recurrences" (*ACM Computing Surveys* 12(4):419–436, 1980). The Master Theorem is presented in Chapter 4 of CLRS. Sections 1–3 and 5 of this handout were based on notes taken by Ari Trachtenberg of lectures by Ed Reingold; the notes were then revised by Jeff Erickson. Sections 4 and 6 by Erickson.

Illinois Institute of Technology
Department of Computer Science

Lecture 17: October 23, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Divide-and-conquer algorithms

1.1 Mergesort

Sorting a sequence of n values efficiently can be done using the divide-and-conquer idea. Split the n values arbitrarily into two piles of $n/2$ values each, sort each of the piles separately, and then merge the two piles into a single sorted pile. This sorting technique, pictured in Figure 1, is called *merge sort*. Let $T(n)$ be the time required by merge sort for sorting n values. The time needed to do the merging is proportional to the number of elements being merged, so that

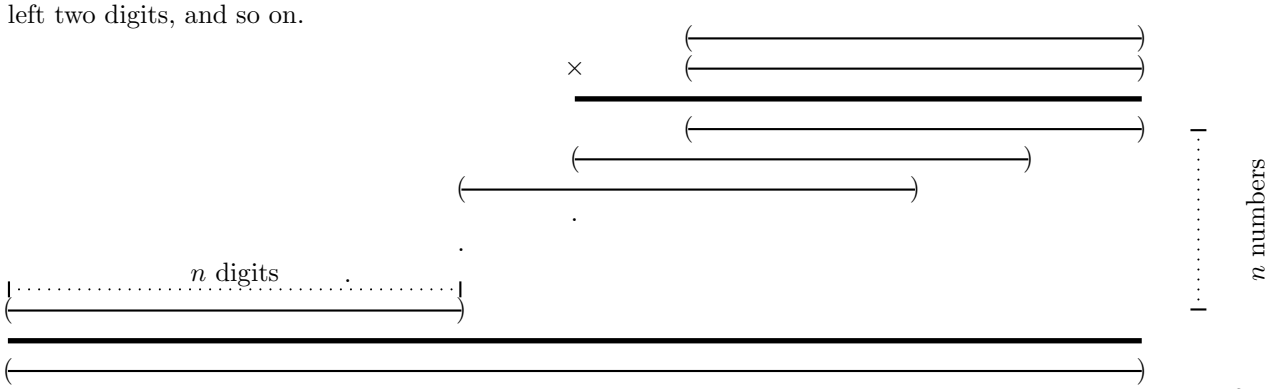
$$T(n) = cn + 2T(n/2),$$

because we must sort the two halves (time $T(n/2)$ for each half) and then merge (time proportional to n). Let $n = 2^i$ and let $t_i = T(2^i) + c2^i$. So $t_i = t_{i-1} + c2^i$, which is annihilated by $(\mathbf{E} - 2)^2$. Thus $t_k = O(2^k k)$ so that $T(n)$ is $\Theta(n \log n)$.

1.2 Multiplying numbers

This is example 4 in Rosen, pages 528–529, continued in example 10 on page 532.

How do we multiply two n-digit numbers? First we multiply the first number by the rightmost digit of the second number. Then we multiply the first number by the second digit of the second number, and so on. Finally, we add the first result to the second result shifted left one digit, add that to the third result shifted left two digits, and so on.



Treating additions and multiplications of digits as our atomic operations, we can accomplish this in $\Theta(n^2)$ operations.

Let us consider a divide-and-conquer algorithm: divide each n-digit number into two $\frac{n}{2}$-digit numbers:

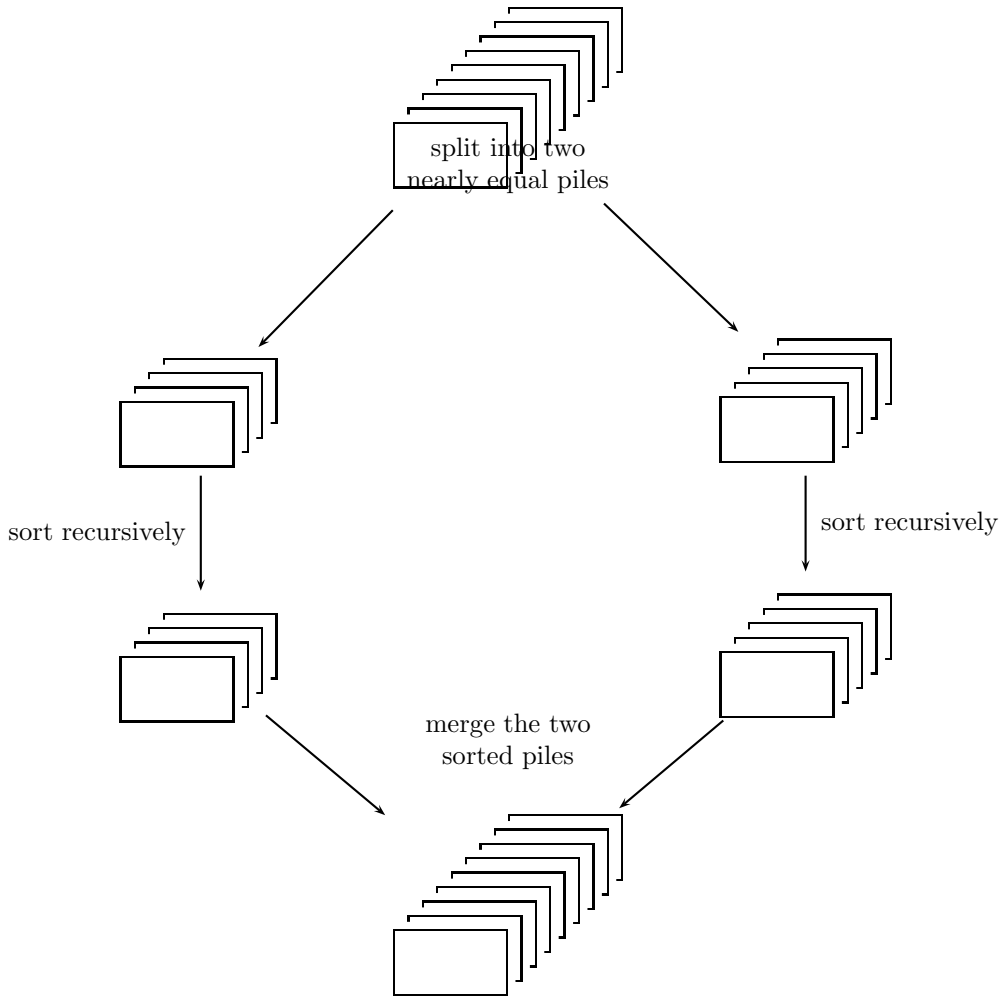


Figure 1: Schematic description of merge sort.

Since we are dividing A and B in half, we know that $A = 10^{\frac{n}{2}}A_1 + A_0$ and $B = 10^{\frac{n}{2}}B_1 + B_0$. So:

$$\begin{aligned} AB &= (10^{\frac{n}{2}}A_1 + A_0)(10^{\frac{n}{2}}B_1 + B_0) \\ &= 10^n A_1B_1 + 10^{\frac{n}{2}}(A_1B_0 + A_0B_1) + A_0B_0 \end{aligned}$$

This yields four multiplications of $\frac{n}{2}$-digit numbers, as well as $\Theta(n)$ additions and shifts. This leads us to the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T\left(\frac{n}{2}\right) + kn \end{aligned}$$

As before, let $n = 2^i$ and let $t_i = T(2^i)$. So $t_i = 4t_{i-1} + k2^i$, which is annihilated by $(\mathbf{E} - 2)(\mathbf{E} - 4)$. Thus $T(n) = cn + \hat{c}(n^2) = \Theta(n^2)$, which is no better than our original algorithm.

We can do something about this, however. Notice that $A_1B_0 + A_0B_1 = (A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0$. We can perform the calculation with only three multiplications. This leads to the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T\left(\frac{n}{2}\right) + kn \end{aligned}$$

Let $n = 2^i$ and let $t_i = T(2^i)$. So $t_i = 3t_{i-1} + k2^i$, which is annihilated by $(\mathbf{E} - 2)(\mathbf{E} - 3)$. Thus $T(n) = cn + \hat{c}n^{\lg 3} = \Theta(n^{\lg 3}) \approx \Theta(n^{1.57})$. This is a substantial improvement over our earlier divide-and-conquer algorithm as well as the naive algorithm.

1.3 Matrix multiplication and Strassen's algorithm

This is example 5 on page 529 of Rosen.

We define the product of two matrices A and B as follows, where A is an $m \times n$ matrix and B is an $n \times r$ matrix:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nr} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n} a_{1i}b_{i1} & \sum_{i=0}^{n} a_{1i}b_{i2} & \cdots & \sum_{i=0}^{n} a_{1i}b_{ir} \\ \sum_{i=0}^{n} a_{2i}b_{i1} & \sum_{i=0}^{n} a_{2i}b_{i2} & \cdots & \sum_{i=0}^{n} a_{2i}b_{ir} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^{n} a_{mi}b_{i1} & \sum_{i=0}^{n} a_{mi}b_{i2} & \cdots & \sum_{i=0}^{n} a_{mi}b_{ir} \end{pmatrix}$$

That is, to find $(AB)_{ij}$, first locate the ith row of A and the jth column of B. Multiply the first element of the row by the first element of the column, the second element of the row by the second element of the column, and so on, and finally add these products.

Finding one element of AB thus requires $\Theta(n)$ time. Since there are mr elements in AB, multiplying A and B in this fashion requires $\Theta(mnr)$ time, or, if we consider the special case in which all matrices are $n \times n$, it requires $\Theta(n^3)$ time.

Let us consider a simple divide-and-conquer approach. We can divide A, B, and AB each into four $\frac{n}{2} \times \frac{n}{2}$ matrices:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

This corresponds to the four equations:

$$\begin{aligned} r &= ae + bf \\ s &= ag + bh \\ t &= ce + df \\ u &= cg + dh \end{aligned}$$

To find AB, then, we must multiply eight pairs of $\frac{n}{2} \times \frac{n}{2}$ matrices and add four pairs of matrices. This leads us to the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 8T\left(\frac{n}{2}\right) + \Theta(n^2) \end{aligned}$$

Solving this recurrence, unfortunately, yields $T(n) = \Theta(n^3)$, identical to the naive algorithm above.

As before, though, a divide-and-conquer approach allows us to take advantage of clever observations about the structure of matrices. In this case, the actual algorithm is quite complicated and is beyond the scope of this lecture, but it will suffice to state that Strassen's algorithm requires only seven multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices rather than the eight multiplications we required earlier. Here's how it's done: Define the seven matrix (sub)products

$$\begin{aligned} P_1 &= a(f - h) \\ P_2 &= (a + b)h \\ P_3 &= (c + d)e \\ P_4 &= d(g - e) \\ P_5 &= (a + d)(e + h) \\ P_6 &= (b - d)(g + h) \\ P_7 &= (a - c)(e + f) \end{aligned}$$

Then

$$\begin{aligned} r &= -P_2 + P_4 + P_5 + P_6 \\ s &= P_1 + P_2 \\ t &= P_3 + P_4 \\ u &= P_1 - P_3 + P_5 - P_7 \end{aligned}$$

(For a suggestion of how this particular workable combination of sums/products might have been discovered, see Cormen, Leiserson, Rivest, and Stein, section 28.2.)

Thus we have the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 7T\left(\frac{n}{2}\right) + \Theta(n^2) \end{aligned}$$

Solving this recurrence gives us better news: $T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})$. We have found a divide-and-conquer algorithm that is asymptotically faster than the naive algorithm.

The constant factor hidden in our analysis of Strassen's algorithm is large, so the naive algorithm is in fact faster on small ($n < 45$ or so) matrices. Faster algorithms than Strassen's have been found, the best one running in $O(n^{2.3727})$ time (Virginia Vassilevska Williams, 2011), and the question of whether there are even faster algorithms is still open. The largest lower bound known is $\Omega(n^2)$.

1.4 Closest Pair of Points

See Example 12 on pages 532–534 of Rosen.

Illinois Institute of Technology
Department of Computer Science

Lecture 18: October 30, 2013

CS 330 Discrete Structures
Fall Semester, 2013

Greedy Algorithms

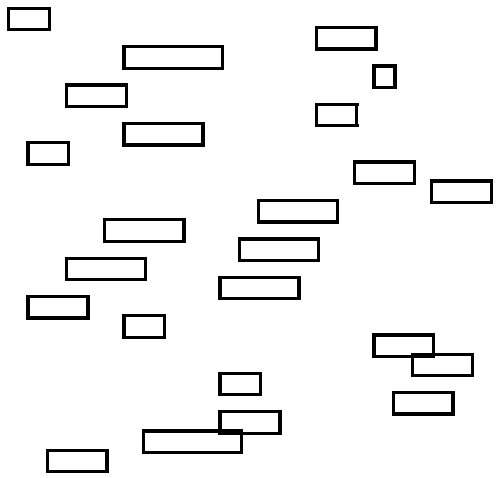
We have discussed Divide-and-Conquer algorithms as a general paradigm; now we discuss another paradigm, **Greedy Algorithms**. Greedy Algorithms are appropriate for many types of problems. However, there are still problems (such as the Traveling Salesman problem that in which a greedy algorithm is not optimal. We will now look at examples where the greedy algorithm works well.

1 Activity Selection

In this sub-section we will study scheduling. Here's the problem:

You are given a list of programs to run on a single princessor; each program has a start time and finish time when it should run. However, the processor can only run one program at any given moment. Your task is to schedule the maximum number of programs to run on the processor.

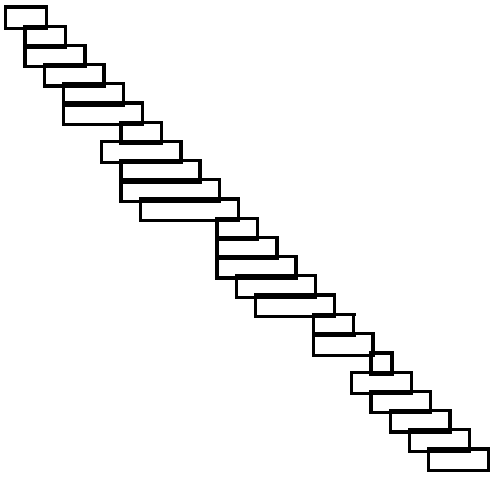
This problem can also be cast as an activity-scheduling problem, where activies take the place of programs. For example, you could be given the following list of activities:



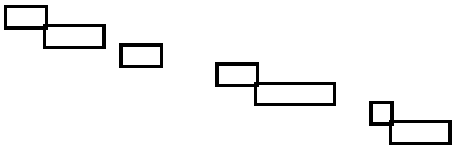
This actually looks too simplified to have any practical use in scheduling jobs for real computers. Besides, in real life it's hard to give exact start and finish time for jobs. But this problem can happen in real life. Suppose a guy with bad taste surprisingly won a Tech News award for guessing Oscar winners and got a one-day pass at Beverly. The cinema has a lot of halls, each of them with different schedules. The guy, of course, can only view one at a time and he only watches complete movies. The goal of this guy is to view as many free movies as possible even if he need to view the same one twice. This is exactly our simplified job scheduling problem with the movies as activities and this greedy guy as the computer.

There are many ways to do optimally schedule these activities. In the exhaustive method, we could just examine every subset of the elements and see which is the best one. There are 2^n ways to make a subset of a set of size n, so out algorithm will require exponential time.

If we allow for some preprocessing, we could significantly improve this algorithm. Specifically, we can first sort the items to be scheduled by their finish time:



In this algorithm, we start at the top of our sorted list and include the first activity. We then go through the list in order including activities whose start time is after (or the same as) the last added activity. This is called the `GREEDY ACTIVITY SELECTOR`, because we are picking the next activity to add based on a "greedy" criterion, that is the activity that looks best at the time we have to choose. For this particular set of events, we will end up with the following selection of activities:



This algorithm visits each item to be scheduled once, therefore the selection part requires time proportional to n. Remember that we first sorted the list, which required $O(n \log n)$ time, so the entire algorithm has a complexity of $O(n + n \log n)$ which is $O(n \lg n)$. This is much better than 2^n, but there is a potential problem. The exhaustive algorithm guaranteed us an optimal scheduling; does this algorithm give us an optimal solution. In other words, does making locally optimal ("greedy") decisions give a globally optimal solution in this case?

The answer is "yes" and we prove the optimality of our greedy algorithm by comparing the set selected with the greedy algorithm (called A) with a true optimal set (called B). If A is not optimal, there must be a place where the greedy algorithm selected a non-globally-optimal choice. If a non-optimal choice was made, there must be a first non-optimal choice. Let's see where that first non-optimal choice was.

Proof of the Optimality of the Greedy Algorithm

Let $A = \{a_{x_0}, a_{x_1}, a_{x_2}, a_{x_3}, \cdots\}$ be the greedily chosen set of activities. Suppose that after picking $j-1$ activities correctly, the greedy algorithm makes a mistake. It picks an activity a_{x_j} that could not possibly lead to an optimal solution. Let us say that a correct choice of activities was b_{x_j} that would lead to an optimal solution $B = \{a_{x_0}, a_{x_1}, a_{x_2}, a_{x_3}, \cdots, a_{x_{j-1}}, b_{x_j}, b_{x_{j+1}}, \cdots\}$.

Clearly, a_{x_j} must end before b_{x_j} or else our greedy selection would have picked b_{x_j}. Thus, switching a_{x_j} and b_{x_j} in the optimal solution B (to get a selection B') must still give a valid activity selection (think about this: remember that a_{x_j} was an acceptable choice for activities after $j-1$ iterations). However, B' contains the same number of elements as B and is, hence, optimal. Thus, the greedy choice does indeed provide the possibility for an optimal solution. Since we each choice of activities by the greedy strategy allows for an optimal solution, and we continue choosing activities until exhaustion, the greedy activity selection must be optimal.

2 Greedy algorithms and suboptimal solutions

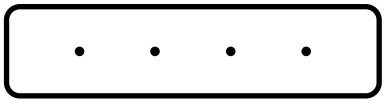
Greedy algorithms tend to be simple and efficient, as the locally optimal solution to a problem is usually easy to find. Unfortunately, greedy algorithms do not always yield optimal solutions.

Consider a set of points on the plane. We would like to connect each point to exactly one other point in such a way that the cost, the sum of the lengths of the edges, is minimized.

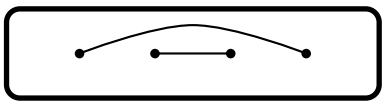
A brute force approach to this problem is to generate each possible matching of points and to find the corresponding costs. The matching with the minimum cost is thus the solution to the problem. While this approach will inarguably produce correct results, it requires exponential time, so we look for a more efficient algorithm.

One simple algorithm that comes to mind is a greedy algorithm. Find the two nearest points that are not yet connected (to other points) and connect them. Repeat this process exhaustively. In the case of a tie, select a pair of points arbitrarily. How efficient is this algorithm? If there are n points, it will take time $\binom{n}{2} + \binom{n-2}{2} + \binom{n-4}{2} + \cdots = \Theta(n^3)$.

Before declaring success, we must ask if our algorithm is correct: does it, in fact, give optimal results? Unfortunately it does not. Consider four points equally spaced along a line:

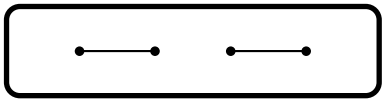


In the first iteration, there are three optimal pairs of points, so the algorithm chooses a pair arbitrarily. Say it chooses the two points in the middle:[1]

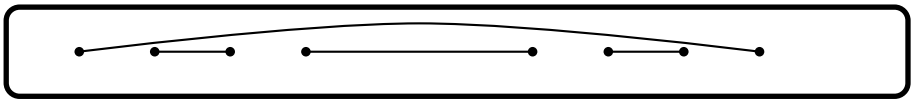


[1] Alternatively, let the points be separated in turn by distances 1, $1-\epsilon$, and 1, where ϵ is some small positive number. Then the algorithm is forced to select the two points in the middle.

This is clearly not the optimal solution, however. The optimal solution connects the two leftmost points and the two rightmost points instead:



If we separate two of the above case by three positions, we have an even worse case:



Again, the optimal solution does much better:

We can repeat this process to derive even worse results.

Consider the optimal solution to the problem with $N = 2^n$ points. The minimum cost is $OPT_n = 2^{n-1}$, where $OPT_1 = 1$.

Compare this to the solution given by the greedy algorithm. Let L_n be the length from the leftmost point to the rightmost point if there are 2^n points. This leads to the recurrence:

$GREEDY_n$
$= 2GREEDY_{n-1} + L_n - 2L_{n-1} + L_{n-1}$
$= 2GREEDY_{n-1} + L_n - L_{n-1}$
$= 2GREEDY_{n-1} + 3^{n-1} - 3^{n-2}$

This is annihilated by $(\mathbf{E} - 2)(\mathbf{E} - 3)$, leading to the solution:

$$GREEDY_n = 2 \cdot 3^{n-1} - 2^{n-1}$$

The error in the greedy algorithm relative to the optimal solution, then, is:

$GREEDY_n / OPT_n$
$= \frac{2 \cdot 3^{n-1} - 2^{n-1}}{2^{n-1}}$
$= 2 \cdot \left(\frac{3}{2}\right)^{n-1} - 1$
$= \frac{4}{3}\left(\frac{3}{2}\right)^{n} - 1$
$= \frac{4}{3} N^{\lg \frac{3}{2}} - 1$
$\approx O(\sqrt{N})$

Not only is the solution given by the greedy algorithm not optimal, as the size of the problem grows, the error grows as well. Clearly for certain problems a greedy approach is inappropriate.

Illinois Institute of Technology
Department of Computer Science

Lectures 19–20: November 4–6, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Graph theory

1.1 Terminology

A graph G is defined as a pair (V, E), where V is a set of *vertices* and E is a set of *edges*. A vertex can be thought of as a point and an edge as a line connecting the two points. An edge $e \in E$ is typically represented as a pair (u, v) where the edge is from u to v, where $u, v \in V$.

Graphs are simple structures with many useful applications. Depending on the application slight variations on the structure are useful. For instance, graphs can be *directed* or *undirected*. In a directed graph edges are unidirectional: for instance, an edge e might be from v_1 to v_2 but not from v_2 to v_1. A directed graph could model, say, the water pipes in a building or the network of (potentially one-way) roads in a city. In an undirected graph edges are bidirectional: an edge e might connect v_1 with v_2 in both directions. An undirected graph could model, for instance, the network of walkways around campus.[1]

Each edge in a graph may also have an associated *weight*. The weight on an edge could represent the capacity of a pipe or of a road. In rare cases, it is useful to associate weights with vertices rather than (or as well as) edges: for instance, the capacity through a pipe connector or of a road intersection.

Questions about graphs can be of a *structural* or of an *algorithmic* nature. Given a graph, a structural question would be whether it can be drawn with no crossing edges.[2] A related algorithmic question would be how to draw the graph with the fewest crossing edges. In this course we will be most concerned with algorithmic questions.

1.2 Representations

While a graph has a simple mathematical definition, we have yet to examine representations of graphs that are algorithmically useful. Three representations are presented here, adjacency structures (or adjacency lists), adjacency matrices, and incidence matrices.

1.2.1 Adjacency structures

The *adjacency structure* or *adjacency list* representation of a graph $G = (V, E)$ is an array of $|V|$ lists, one for each vertex in V. The list corresponding to the vertex v_i consists of the vertices in V that have edges from v_i. An adjacency structure requires $O(|V| + |E|)$ space.

[1] Note that an undirected graph can be defined in terms of a directed graph: if $(u, v) \in E$, then $(v, u) \in E$ as well.
[2] Such a graph is termed a *planar* graph.

1.2.2 Adjacency matrices

The *adjacency matrix* corresponding to a graph $G = (V, E)$, where $V = v_1, v_2, \ldots, v_{|V|}$, is a $|V| \times |V|$ binary matrix A defined by:

$$A_{ij} = \begin{cases} 1 \text{ if } (v_i, v_j) \in E \\ 0 \text{ otherwise} \end{cases}$$

An adjacency matrix requires $|V|^2$ bits of space.

1.2.3 Incidence matrices

If, for a graph $G = (V, E)$, $V = v_1, v_2, \ldots, v_{|V|}$ and $E = e_1, e_2, \ldots, e_{|E|}$, the *incidence matrix* associated with G is a $|V| \times |E|$ matrix B defined by:

$$B_{ij} = \begin{cases} 1 \text{ if edge } e_j \text{ touches vertex } v_i \\ 0 \text{ otherwise} \end{cases}$$

An incidence matrix requires $O(|V|\ |E|)$ space.

2 Breadth-first search

2.1 The breadth-first search algorithm

The goal of breadth-first search is to explore a graph. The technique used is to start at an arbitrary vertex[3] and to visit its neighbors. The neighbor's neighbors are then examined in turn, and so on until all vertices have been visited.

In breadth-first search, we color vertices we have not yet visited white, vertices we have visited black, and vertices we are in the process of visiting gray. As well, we keep track of the in-process (gray) vertices in a queue. Thus the algorithm must ensure that any vertex colored gray is also on the queue and vice versa. Below is the algorithm of BFS of a graph G starting at the vertex $s \in V[G]$:

function BFS(G, s)
1: **for all** $u \in V[G] - \{s\}$ **do**
2: $color[u] \leftarrow$ WHITE
3: $d[u] \leftarrow \infty$
4: $\pi[u] \leftarrow$ NIL
5: **end for**
6: $color[s] \leftarrow$ GRAY
7: $d[s] \leftarrow 0$
8: $\pi[s] \leftarrow$ NIL
9: $Q \leftarrow \{s\}$
10: **while** $Q \neq \emptyset$ **do**
11: $u \leftarrow$ DEQUEUE (Q)
12: **for all** $v \in Adj[u]$ **do**
13: **if** $color[v] =$ WHITE **then**
14: $color[v] \leftarrow$ GRAY

[3] If the graph is not connected (that is, if the vertices of the graph can be divided into two partitions with no edges between vertices in one partition and vertices in the other), it is actually quite relevant which vertex the algorithm begins with, as only one partition of the graph will be reached by breadth-first search.

```
15:         d[v] ← d[u] + 1
16:         π[v] ← u
17:         ENQUEUE (Q, v)
18:       end if
19:     end for
20:     color[u] ← BLACK
21: end while
```

This algorithm maintains two additional variables associated with each vertex. One of these is a time stamp, d, that is incremented each time a vertex is visited. The other is π, which identifies the predecessor to the vertex—that is, the vertex from which the current vertex was found. These variables are useful in some of the applications of breadth-first search.

The edges are of the following types:

Black vertex→black vertex	Completely explored territory
Black vertex→gray vertex	Haven't yet finished with latter vertex
Gray vertex→gray vertex	Haven't yet finished with either vertex
Gray vertex→white vertex	Haven't yet finished with former, haven't seen latter
White vertex→white vertex	Haven't seen either vertex

2.2 Time complexity of breadth-first search

What is the time complexity of BFS? Each vertex is added to and removed from the queue at most once, when it is colored gray and when it is colored black, respectively. These queue operations take constant time, so the total work involved is $O(|V|)$.

Each edge is examined no more than twice, once for each vertex. Again, this involves constant time for each vertex, $O(|E|)$ in total.

Thus BFS requires $O(|V| + |E|)$ time.

2.3 An example of Greedy Strategy

Notice that if the graph is disconnected, BFS will only search the component of the graph containing s. Thus BFS is a useful algorithm for determining if a graph is connected: select an arbitrary vertex s and run BFS from that vertex. If, after the algorithm is complete, any of the vertices are still colored white, they were not reached by BFS and thus the graph is not connected.

We can use BFS to compute the **shortest path** from one vertex to every other vertex in an unweighted graph.[4] In particular, for any vertex $v \in G$, $d[v]$ is the length of the shortest path from s to v. The path itself can be constructed by following the π pointers "in reverse" from v back to s.

Two observations are crucial to prove the correctness of the BFS algorithm in finding the shortest paths from one vertex to every other vertex in an unweighted graph:

1. Suppose that during the execution of BFS on a graph $G = \langle V, E\rangle$, the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r\rangle$, where v_1 is the head of Q and v_r is the tail. Then, $d[v_r] \leq d[v_1] + 1$ and $d[v_i] \leq d[v_{i+1}]$ for $i = 1, 2, \ldots, r - 1$.

2. When any new element v is put into the queue, $d[v] = d[u] + 1$, where $d[u]$ is the head of the queue.

[4] When weights are introduced, this problem becomes significantly more complex. Typically, for an efficient implementation, priority queues must be used.

These give us some very useful information about the distance values of those vertices in the queue (gray vertices).

Suppose our BFS algorithm is not correct and while traversing the graph, it makes its first mistake in finding the distance from s to v. The distance value of v is set by executing $d[v] \leftarrow d[u] + 1$, where u is the head of the queue. If this is wrong, we must have " distance from s to $v < d[u] + 1$ " and there is another path from s to v which is the shortest path. Since v is a white vertex before $d[v]$ is set, the shortest path must contain an edge, say, x—y, where x is a gray vertex and y is a white vertex. $d[x]$ and $d[u]$ are both set and we know the values are correct since our algorithm made its FIRST mistake when computing $d[v]$. x and u are both gray vertices and u is the first element in the queue, therefore by the above observations we have $d[x] \geq d[u]$. Thus

$$\text{shortest distance from } s \text{ to } v \geq d[x] + 1 \geq d[u] + 1 = d[v]$$

which contradicts with our assumption about "the first mistake".

3 Depth-first search

As we have seen, breadth-first search offers us a method to visit the vertices of a graph. In particular, given a starting vertex s, BFS first visits s, then visits all vertices adjacent to s, then visits all vertices adjacent to those vertices, and so on. An alternative approach, and that used by depth-first search, is, intuitively, to "plunge in" — as each node is visited, visit its children before continuing the search at the same depth, and only back out when no unvisited children remain.

3.1 The depth-first search algorithm

If we modify the breadth-first search algorithm by changing the queue Q to a stack, we have derived depth-first search. Alternatively, we may use recursion to implement the stack; this is the approach typically taken. One small but useful change in the algorithm is a different treatment of time stamps: in DFS, each vertex has two associated time stamps, one, d, holding the discovery time, and the other, f, holding the completion time. The time is incremented at each d or f assignment. Here is the recursive version:

function DFS(G)
1: **for all** $u \in V[G]$ **do**
2: $color[u] \leftarrow$ WHITE
3: $\pi[u] \leftarrow$ NIL
4: **end for**
5: $time \leftarrow 0$
6: **for all** $u \in V[G]$ **do**
7: **if** $color[u] =$ WHITE **then**
8: DFS-visit(u)
9: **end if**
10: **end for**

function DFS-visit(u)
1: $color[u] \leftarrow$ GRAY
2: $d[u] \leftarrow time \leftarrow time + 1$
3: **for all** $v \in Adj[u]$ **do**
4: **if** $color[v] =$ WHITE **then**
5: $\pi[v] \leftarrow u$

```
 6:       DFS-visit(v)
 7:    end if
 8: end for
 9: color[u] ← BLACK
10: f[u] ← time ← time + 1
```

3.2 Classification of edges

We can classify the edges in the graph G based on when the DFS algorithm traverses them:

A **tree edge** is an edge from a gray vertex to a white vertex. A tree edge brings the algorithm into deeper territory, as of yet undiscovered.

A **back edge** is an edge from a gray vertex to another gray vertex. Back edges form cycles in the graph, as they indicate that the algorithm has discovered a vertex further back in the path it is exploring.

Edges from gray vertices to black vertices fall into two classes. A **forward edge** connects the current vertex to a vertex in the subtree rooted at the current vertex (its "descendant"). A **cross edge** connects the current vertex to a vertex in another subtree (for example, it's "cousin", "uncle" or "nephew").

3.3 Time complexity of depth-first search

Depth-first search examines each vertex exactly once. However, it must consider each edge to determine if it leads to an undiscovered (white) vertex. This leads to a running time of $\Theta(|V| + |E|)$.

3.4 The parenthesis theorem

The parenthesis theorem tells us that, for two vertices $u, v \in V$, it cannot be the case that $d[u] < d[v] < f[u] < f[v]$; that is, the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are either disjoint or nested. This is a simple consequence of the depth-first nature of DFS. If the algorithm discovers u and then discovers v, it cannot later back out of u without first backing out of v.

3.5 Applications of depth-first search

3.5.1 Topological sort

One use of a directed acyclic graph (dag) is for what civil engineers term a PERT[5] chart. A PERT chart indicates dependencies in a large-scale building project. For example, the walls can't be painted before there are walls to be painted. A dependency of v on u would be indicated by an edge from u to v — so, in our example, there would be an edge from an "erect walls" vertex to a "paint walls" vertex.

A **topological sort** of a dag is a linear ordering of its vertices such that all edges point in the same (planar) direction. A topological sort of a PERT chart would produce one possible valid ordering of the components of the project. Note that for a general dag, there are many valid topological sorts. Also note that if the graph contains a cycle, topological sort is not possible, as at least one edge would have to point in the wrong direction.

How can we find the topological sort of a dag? We can simply run the depth-first search algorithm and sort the vertices in order of decreasing finish time. Alternatively, as DFS backs out of a vertex, it adds it to the

[5]Program evaluation and review technique.

front of a list. These two approaches produce identical results, although the second is more efficient as it does not need to sort the results of the DFS. The efficient algorithm runs in $\Theta(|V| + |E|)$ time, like DFS itself.

Why does this work? First, notice that a directed graph G has no cycles if and only if DFS produces no back edges in G. We need only show that for any pair of distinct vertices $u, v \in V$, if $(u, v) \in E$, then $f[v] < f[u]$. In a DFS exploration of G, an edge u to v means v cannot be gray, since it would then be a back edge (and hence there would be a cycle). If v is white, it is a descendant of u, so $f[v] < f[u]$. If v is black, $f[v] < f[u]$ as well. Thus for any edge $(u, v) \in E$, $f[v] < f[u]$.

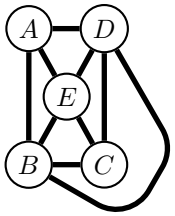
Illinois Institute of Technology
Department of Computer Science

Lecture 21: November 11, 2013

CS 330 Discrete Structures
Fall Semester, 2013

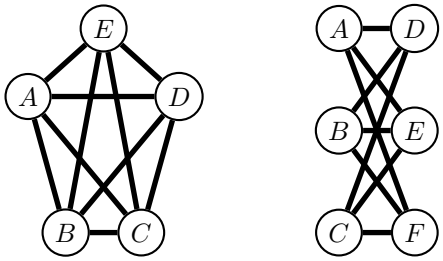
1 Planar graphs

A **planar graph** is defined as a graph that can be drawn in the plane so that no edges cross. For example the graph on the right is planar, while there is no way to add the edge from (A, C) and still have it planar. Note that any graph of less than five nodes must be planar.



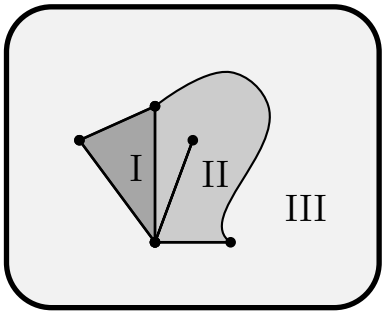
1.1 Kuratowski's theorem

The fully connected graph with five vertices is called K_5 and is isomorphic to the graph on the near right. There is no way that this can be drawn as in the plane with no crossing edges — that is, it is a nonplanar graph. Another nonplanar graph, called $K_{3,3}$, the complete bipartite graph, is on the far right. As it turns out, **Kuratowski's theorem** states that a graph is nonplanar iff it contains a **homeomorphic image** of $K_{3,3}$ or K_5. The proof of this theorem is beyond the scope of the course.



1.2 Euler's formula

Notice that when we draw a planar graph, it divides the space (plane) into faces. These faces are regions delimited by edges. On the right are three faces. When edges cross, faces are not well-defined. Also notice that a cycle in a graph determines a face. If there are no cycles, there are no bounded faces (we consider the region outside any cycles to be an unbounded face). This example has five vertices, six edges, and three faces.



It turns out that there is a relationship between the number of vertices $|V|$, the number of edges $|E|$, and the number of faces $|F|$ in any planar graph. This result is:

Euler's formula: for any simple, connected planar graph $|V| - |E| + |F| = 2$

This is proved by induction:

Base case: $|E| = 1$. A graph with no cycles has only one face. A graph with only one edge must have two vertices (since this is a connected simple graph), so $|V| - |E| + |F| = 2 - 1 + 1 = 2$.

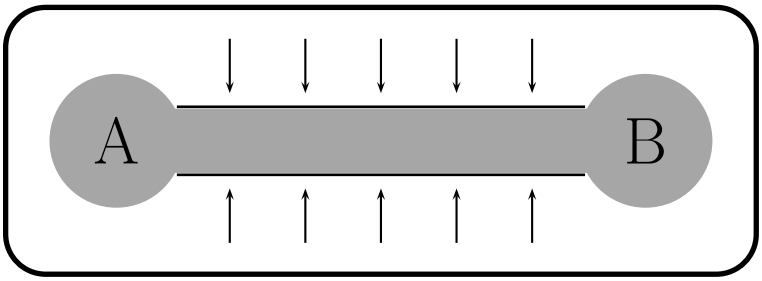
Inductive step: We do induction on the number of edges. So $|E| > 1$. There are two cases: either the graph has a cycle or it doesn't.

In a graph without a cycle, there is at least one vertex of degree one. You can see this if you consider DFS; it will not stop until it reaches a vertex without any exits. That vertex cannot have back edges, as there are no cycles. If we remove this vertex, $|V|$ decreases by 1, $|E|$ decreases by 1 and $|F|$ remains unchanged, so $(|V| - 1) - (|E| - 1) + |F| = |V| - |E| + |F| = 2$ and Euler's formula still holds.

In a graph with a cycle, we remove one edge from the cycle. $|E|$ decreases by 1, $|F|$ decreases by 1 and $|V|$ remains unchanged so $|V| - (|E| - 1) + (|F| - 1) = |V| - |E| + |F| = 2$ and Euler's formula still holds.

Corollary: In a planar graph, $|E| \leq 3|V| - 6$.

Proof: Consider the "sides" of a blown-up edge:



Each edge has exact two sides. At least three edges are required to make a face, so each face has at least three edge sides.[1] We can then derive the inequality $2|E| \geq 3|F|$, which can be substituted into Euler's formula to give us the corollary:

$$|V| - |E| + |F| = 2,$$

so

$$-|V| + |E| + 2 = |F| \leq \frac{2}{3}|E|,$$

and the corollary follows.

Consider K_5. There are five vertices and ten edges. Is $10 \leq 3(5) - 6$? No.

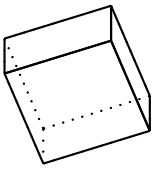
Consider $K_{3,3}$, There are six vertices and nine edges. Is $9 \leq 3(6) - 6$? Yes! Does this mean that $K_{3,3}$ is in fact planar? Not necessarily, since Euler's formula gives a condition that must hold for all planar graphs but may also hold for some nonplanar graphs. This discrepancy comes from the assumption that the minimum number of edge sides around a face is three. In a bipartite graph, each face has at least two vertices from each partition of the graph, so the inequality is $2|E| \geq 4|F|$, yielding the result $2|V| - 4 \geq |E|$ for a planar bipartite graphs. $K_{3,3}$ fails this test, as we expect.

[1] Read this over until the distinction between "edges" and "edge sides" is clear.

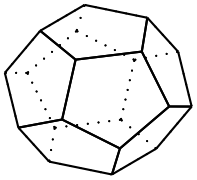
1.2.1 Platonic solids

Platonic solids are regular solids: every vertex has the same number of edges, and every face has the same number of edges. This was know to the ancient Greeks. There are only five Platonic solids:

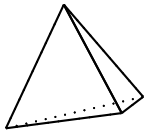
Cube



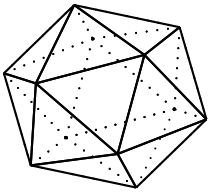
Dodecahedron



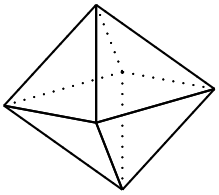
Tetrahedron



Icosahedron



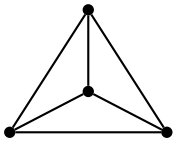
Octahedron



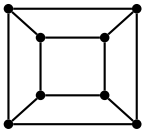
Let p be the number of edges that surround a face and q be the degree of each vertex.

Why are there only five Platonic solids? Consider another property of the planar solid: it can be represented as a planar graph. If you could put a hole in one face of the solid and stretch it until it was flat, you would end up with a planar graph where each vertex has a degree q. The number of regions would equal the number of faces. "Stretched out," the Platonic solids look like:

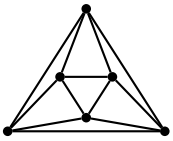
Tetrahedron



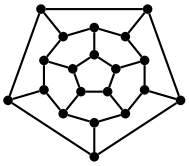
Cube



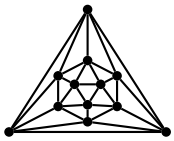
Octahedron



Dodecahedron



Icosahedron



Now we are going to count "tips of edges." There are two edge tips per edge, and there are q edge tips per vertex, so we know that

$$\begin{aligned} 2|E| &= q|V| \\ |V| &= \frac{2}{q}|E| \end{aligned}$$

Now consider the "sides of the edges." There are (still) two edge-sides per edge, and there are p edge-sides per face.

$$\begin{aligned} 2|E| &= p|F| \\ |F| &= \frac{2}{p}|E| \end{aligned}$$

Since this is a planar graph, Euler's formula holds, so we have

$$\begin{aligned} |V| - |E| + |F| &= 2 \\ \frac{2}{q}|E| - |E| + \frac{2}{p}|E| &= 2 \\ |E|\left(\frac{2p + 2q - pq}{pq}\right) &= 2 \end{aligned}$$

We notice that $|E|$ is positive, 2 is positive and pq is positive. This implies that $2p + 2q - pq$ is positive. So:

$$\begin{aligned} 2p + 2q - pq &> 0 \\ -2p - 2q + pq &< 0 \\ 4 - 2p - 2q + pq &< 4 \\ (p-2)(q-2) &< 4 \end{aligned}$$

But we have some more constraints; p must be greater than 2, or we would not have a face, and q must also be greater than 2, or we could not create a 3-dimensional solid.

This leaves us with only 5 possible integer results for the inequality: $(1)(1) < 4$, $(1)(2) < 4$, $(2)(1) < 4$, $(1)(3) < 4$, and $(3)(1) < 4$, giving the five solutions: $p = 3$, $q = 3$ (tetrahedron) $p = 3$, $q = 4$ (octahedron) $p = 4$, $q = 3$ (cube) $p = 5$, $q = 3$ (dodecahedron) and $p = 3$, $q = 5$ (icosahedron).

1.2.2 Graph coloring

Consider a map of the Continental United States. The mapmaker would like to color the states on the map in such a way that no two adjacent states have the same color. With that in mind, he would like to minimize the number of colors he uses. What is the fewest number of colors he can use?

This can be easily transformed into a graph-theoretic problem: states are vertices and state borders are edges. Is there a maximum number of colors that will suffice to color all graphs?

Clearly three colors will not suffice: K_4 is a planar graph but it cannot be colored in fewer than four colors.

It is not difficult to prove the five-color theorem:

Theorem: Given a planar graph, five colors are sufficient to color the graph such that no two connected vertices share the same color.

We will use induction on the number of vertices.

Base case: When $|V| \leq 5$, we only have five vertices, so five colors are certainly sufficient.

Inductive step: When $|V| \geq 6$, there must be a vertex of degree ≤ 5. This follows from the corollary to Euler's formula: $|E| \leq 3|V| - 6$. If all vertices had degree ≥ 6 then by counting the number of edge tips, we have the inequality $2|E| \geq 6|V|$, or $|E| \geq 3|V|$, which contradicts the corollary to Euler's formula.

Let's examine the vertex with degree ≤ 5. There are two cases, that the degree is less than five or the degree is five. If the degree ≤ 4, remove that vertex from the graph, color the rest of the graph recursively

(using the inductive hypothesis), and you will be able to color the graph with whatever color is not used by the four connected vertices. When the degree is 5, we know that there have to be two of those 5 vertices connected to the vertex that are not connected (otherwise there would be a K_5 subgraph and it would not be planar). We will cut out the two nonconnected vertices and the vertex with degree 5, replace them with one "mega-vertex," and color the graph recursively. We will then expand the vertex that we shrunk, giving the nonconnected vertices the color of the shrunken vertex, and then color the vertex with degree 5 with the remaining color not used by the five vertices connected to it, two of which have the same color.

Illinois Institute of Technology
Department of Computer Science

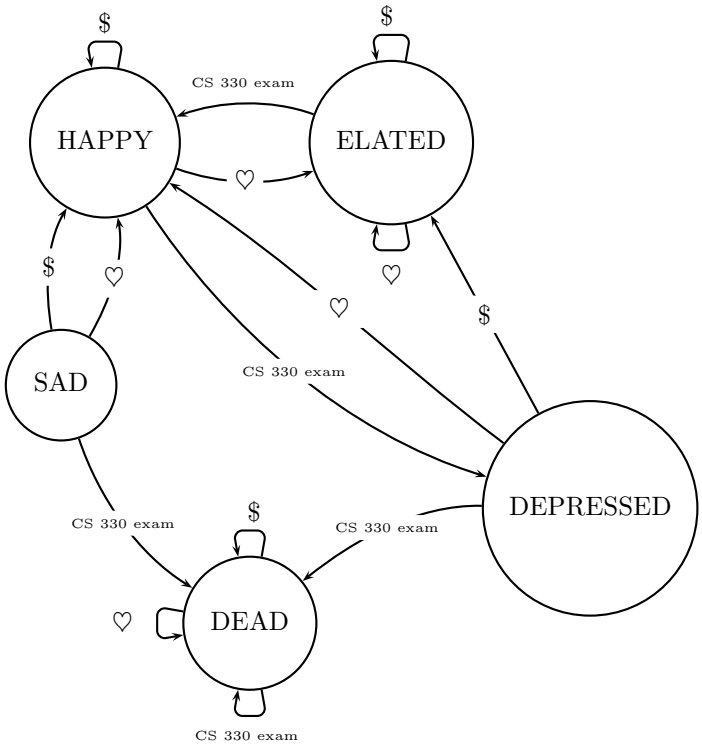
Lecture 22: November 13, 2013

CS 330 Discrete Structures
Fall Semester, 2013

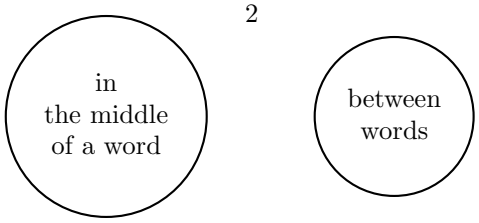
1 Finite state machines

1.1 Some examples of their use

First of all, let's take a look of a state diagram that a typical CS 330 student might follow.



One simple problem that will serve to introduce the notion of state is that of counting words: given an input string, scan the string from beginning to end and count the number of words in the string. Simply counting the number of spaces (and other word separators, such as punctuation marks) will not do, as words may be separated by more than one such separators. Rather, we need to distinguish between two *states*:

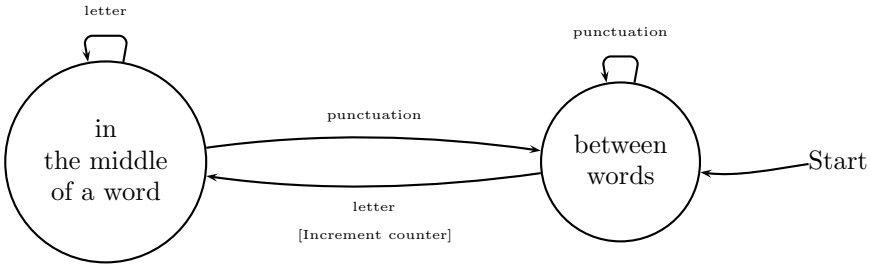


From each of these two states we can define appropriate *transitions*: that is, if we are in a state and we see an input character, what state do we emerge in? If we are in the middle of a word and we see a letter, we remain in the middle of a word. If we are between words and we see a letter, we are now in the middle of a word. Likewise, if we are in the middle of a word and we see punctuation, we are now between words, and if we are already between words and we see punctuation, we are still between words.

Where do we start? That is, before we begin processing material, which state should we be in? We should start in the "between words" state.

Finally, since we would like to count words and not just flutter between two states, when do we increment our counter? We'd like to claim that we've counted another word when we follow the transition from "between words" to "in the middle if a word."

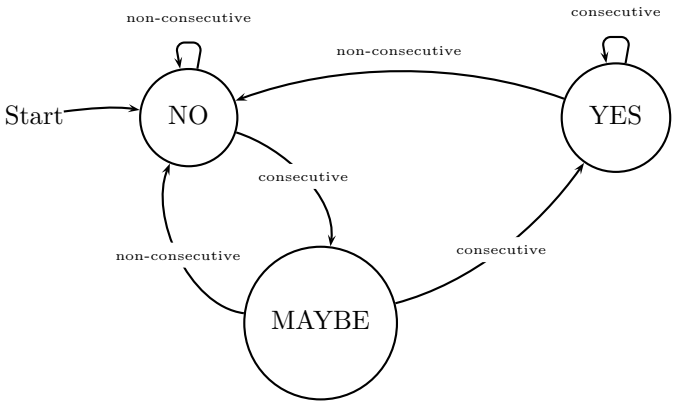
This leads to the following *state transition diagram*:



Let us consider a slightly more complicated problem. We are given a sorted list of page numbers and would like to generate an appropriate index listing — that is, runs of three or more consecutive page numbers should be collapsed. For instance, given the input 5, 6, 7, 8, 11, 12, 14, 17, 18, 19, 21, 27, 28, we would like to generate the listing 5–8, 11, 12, 14, 17–19, 21, 27, 28.

This can be modeled with three states: NO, YES and MAYBE. We are in NO when we are not in the middle of a run, in YES when we are in the middle of a run, and in MAYBE when we may be in the middle of a run.

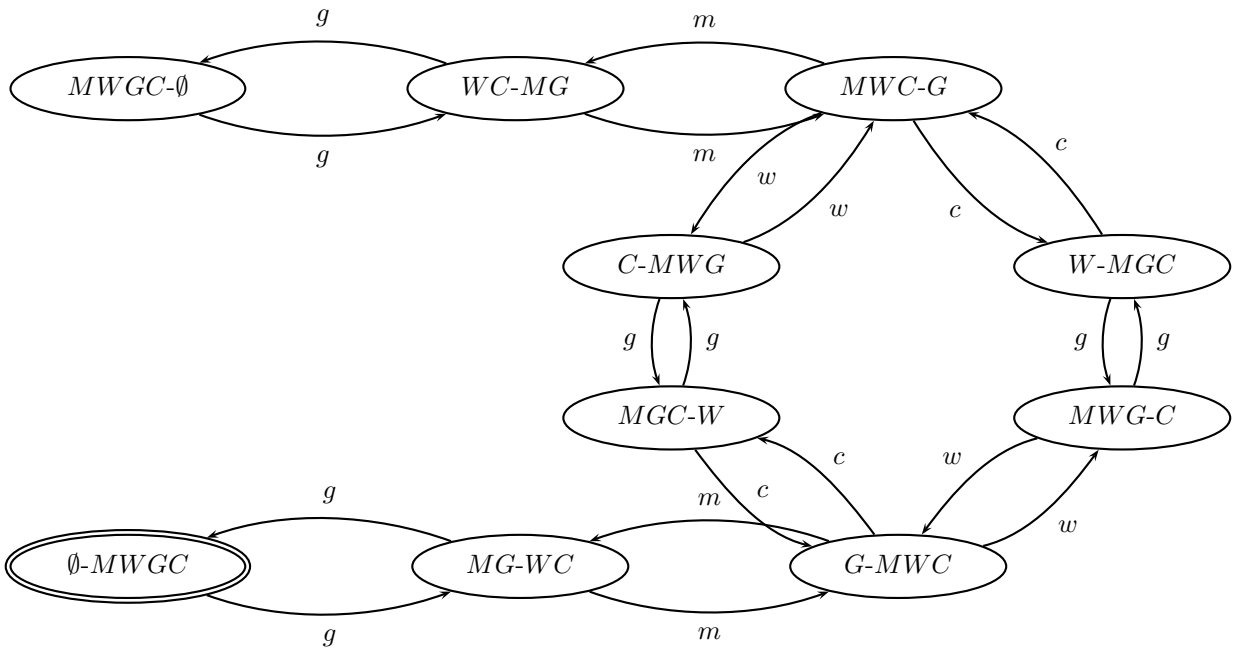
Appropriate transitions yield the following state diagram:



Let's consider another example. A man with a wolf, goat, and cabbage is on the left bank of a river. There is a boat large enough to carry the man and only one of the other three. The man and his entourage wish to cross to the right bank, and the man can ferry each across, one at a time. However, if the man leaves the wolf and goat unattended on either shore, the wolf will surely eat the goat. Similarly, if the goat and cabbage are left unattened, the goat will eat the cabbage. Is it possible to cross the river without the goat or cabbage being eaten?[1]

The problem is modeled by observing that the pertinent information is the occupants of each bank after a crossing. There are 16 subsets of the man (M), wolf (W), goat (G), and cabbage (C). A state corresponds to the subset that is on the left bank. States are labeled by hyphenated pairs such as $MG - WC$, where the symbols to the left of the hyphen denote the subset on the left bank; symbols to the right of the hyphen denote the subset on the right bank. Some of the 16 states, such as $GC - MW$, are fatal and may never entered by the system.

The "inputs" to the system are the actions the man takes. He may cross alone (input m), with the wolf(input w), the goat (input g), or cabbage (input c). The initial state is $MWGC$-$\emptyset$ and the final state is $\emptyset$-$MWGC$. Here is the transition diagram:



There are two equally short solutions to the problem, as can be seen by searching for paths from the initial state to the final state(which is doubly circled). There are infinitely many different solutions to the problem, all but two involving useless sycles.

[1]This problem appears in a medieval Latin manuscript by Alcuin of York. It was used in "Gone Maggie Gone," a 2009 episode of *The Simpsons* in which Homer is trapped on one side of a river with his baby Maggie, his dog, and a bottle of poison capsules. He has only a flimsy boat so he can carry only one item at a time. If he takes the dog, Maggie might swallow some poison; if he takes the poison, the dog might bite Maggie.

1.2 Finite state machines and juggling

A great application! See `http://en.wikipedia.org/wiki/Siteswap`

1.3 Finite state machines as mathematical objects

We need five entities to describe a finite state machine:

- Σ — a finite input alphabet
- S — a set of states
- $s_0 \in S$ — an initial state
- $F \subseteq S$ — a set of accept states
- $\delta : S \times \Sigma \to S$ — a transition function

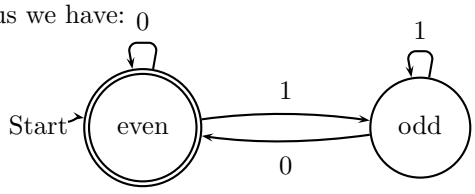
Then $M = (\Sigma, S, s_0, F, \delta)$ specifies a finite state machine.
We define a *regular language* to be a language recognized by some finite state machine.[2]

1.4 Finite state machines as a computational model

As we have seen, finite state machines are a useful programming tool. They are also useful for exploring the notion of computation. A computer has a finite amount of memory, and thus can be in any of a finite (quite large, but still finite) number of states. As the computer receives input, it changes state.

Let us refine our notion of a finite state machine.[3] A finite state machine receives input symbol-by-symbol; as each symbol comes in, it changes state according to the appropriate transition. As well, certain states are declared as *accept* or *final* states. If, at the end of an input string, the machine is in an accept state, it is said to *accept* the input; otherwise it *rejects* the input. Thus a finite state machine recognizes a *language*, some subset of all finite strings over an alphabet.

Let us design a finite state machine to recognize even numbers in binary, where the input has the most significant digit first. As the machine processes the input string, it fluctuates between two states, one if the part of the input string it has seen so far is even, and the other if it is odd. The transitions are not hard to find, as a number in binary is even if it ends in a 0 and odd if it ends in a 1. Finally, we want this machine to accept if the input is even. Thus we have:



This technique can be expanded to slightly more interesting problems, for instance recognizing multiples of three or five (still in binary).

[2]Rosen follows a different treatment by defining regular languages differently and then deriving this as a theorem. This is not the approach we will be following in lecture. You have been warned.

[3]Finite state machines are also known as finite automata. In particular, the variety we are developing here is a deterministic (or definite) finite automaton.

Illinois Institute of Technology
Department of Computer Science

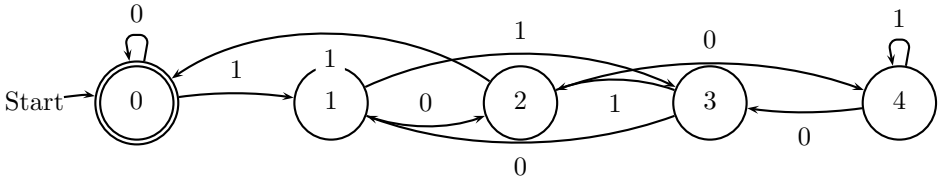
Lecture 23: November 18, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Finite state machines (continued)

1.1 Finite state machines as a computational model

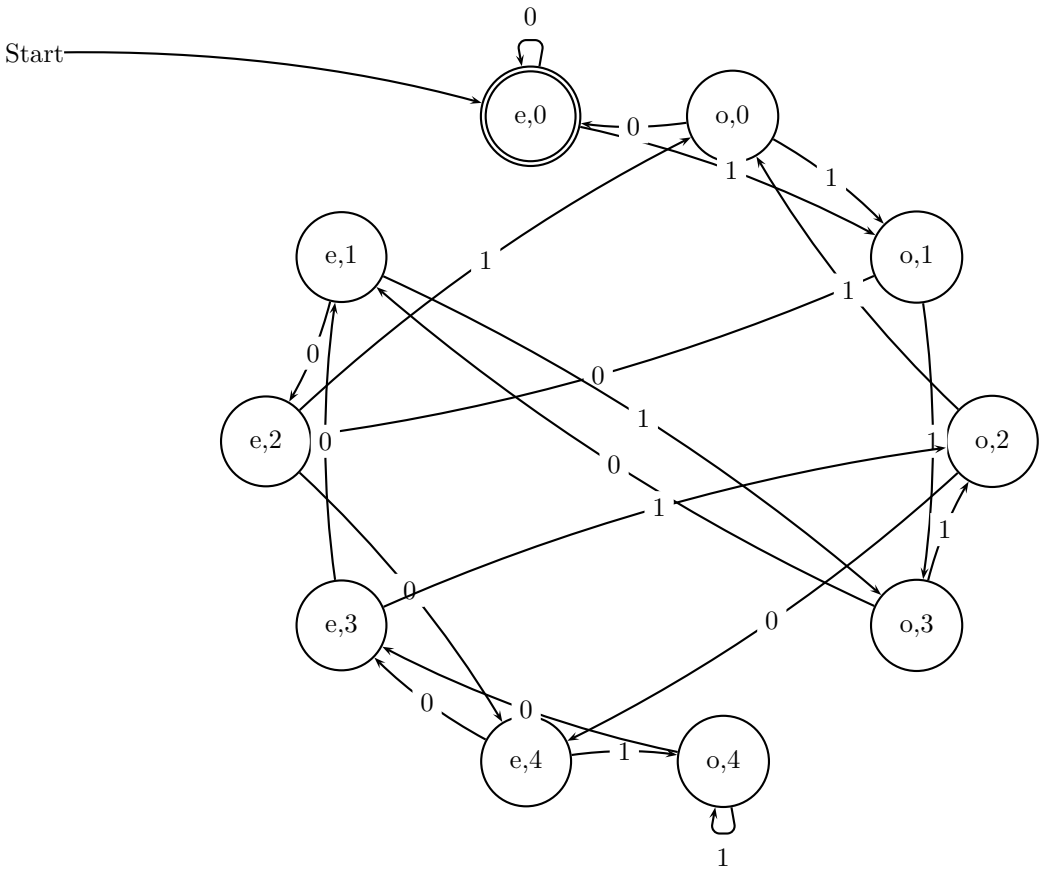
Consider a multiple of five. If a 0 is appended to the end, the number is doubled and is still a multiple of five. If a 1 is appended to the end, the number is doubled and 1 is added, so we should move to state 1. Similar reasoning for the other four states gives us:



What about multiples of ten? We can follow a similar approach and find the transitions for ten states labeled 0–9. Alternatively, since the multiples of ten are the numbers that are both even and multiples of five, and we have machines that recognize even numbers and multiples of five, it would be nice if we could somehow combine these two machines. In essence, we want to run these two machines in parallel, applying the input to both and accepting the input if both machines accept.

We can collapse these two machines into one by keeping track of the state of each machine in our new machine. This gives us the (somewhat messier) machine:[1]

[1] In fact, in this case, this is the identical machine, with the states relabeled, that we would derive if we used the same approach we did for the multiples of five on this problem. There is a simple way to reduce this machine to one with only six states: do you see how to do it?



2 Closure properties of regular languages

We can prove that a language L is regular by constructing a finite state machine that recognizes L. We use this observation to prove a number of results.

Lemma: The set of regular languages is closed under intersection.[2]

Proof:[3] Let L_1 and L_2 be regular languages. Say that L_1 is recognized by $M_1 = (\Sigma, S, s_0, F, \delta)$ and L_2 is recognized by $M_2 = (\Sigma, T, t_0, G, \lambda)$. Then $L_1 \cap L_2$ is recognized by $M = (\Sigma, S \times T, (s_0, t_0), F \times G, \alpha)$, where $\alpha : \Sigma \times S \times T \to S \times T$ is defined as $\alpha(x, s, t) = (\delta(x, s), \lambda(x, t))$.

Lemma: The set of regular languages is closed under union.

Proof: Let L_1 and L_2 be regular languages. Say that L_1 is recognized by $M_1 = (\Sigma, S, s_0, F, \delta)$ and L_2 is recognized by $M_2 = (\Sigma, T, t_0, G, \lambda)$. Then $L_1 \cup L_2$ is recognized by $M = (\Sigma, S \times T, (s_0, t_0), (F \times T) \cup (S \times G), \alpha)$, where $\alpha : \Sigma \times S \times T \to S \times T$ is defined as $\alpha(x, s, t) = (\delta(x, s), \lambda(x, t))$.

Lemma: The set of regular languages is closed under complementation.

Proof: Let L be a regular language. Say that L is recognized by $M = (\Sigma, S, s_0, F, \delta)$. Then $\overline{L}$ is recognized by $M' = (\Sigma, S, s_0, S - F, \delta)$.

[2] That is, the intersection of two regular languages is still a regular language.

[3] This is precisely what we did to solve the problem of multiples of ten.

These three lemmas directly lead to the following closure theorem:

Theorem: The set of regular languages is closed under intersection, union, and complement.

We move to other properties of regular languages.

Theorem: All finite sets are regular.

Proof sketch: Consider the language $L = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$. We prove this by induction on k.

Base case: Here $k = 1$. Assume σ_1 is not empty;[4] let $\sigma_1 = a_1 a_2 \cdots a_n$. We can construct a finite state machine that recognizes σ_1. Σ is the appropriate alphabet. Let $S = \{0, 1, 2, \ldots, n, \text{FAIL}\}$, $s_0 = 0$, and $F = \{n\}$.

Define the transition function δ as follows:

$$\begin{aligned} \delta(\text{FAIL}, a_j) &= \text{FAIL} \\ \delta(i, a_j) &= \begin{cases} j & \text{if } i+1 = j \\ \text{FAIL} & \text{otherwise} \end{cases} \end{aligned}$$

Then $M = (\Sigma, S, s_0, F, \delta)$ recognizes σ_1 and no other strings.

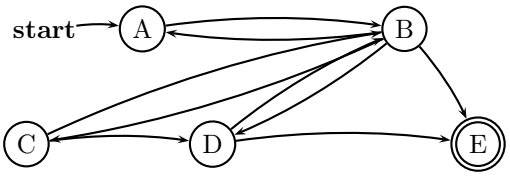
Inductive step: For $k > 1$, build a machine M_k that recognizes σ_k. By induction there is a machine $M_{1\ldots k-1}$ that recognizes the set $\{\sigma_1, \sigma_2, \ldots, \sigma_{k-1}\}$. By the "union lemma" above, the union of $\{\sigma_k\}$ and $\{\sigma_1, \sigma_2, \ldots, \sigma_{k-1}\}$ is therefore regular.

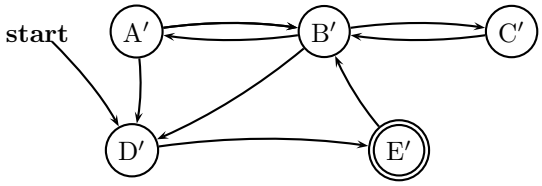
We now consider some slightly more complicated questions of regularity. Is concatenation closed over regular languages? The concatenation of languages L_1 and L_2 is defined as

$$L_3 = L_1 L_2 = \{l_1 l_2 \,|\, l_1 \in L_1,\, l_2 \in L_2\}$$

For example, if $L_1 =$\{ all strings with at least two zeroes\} and $L_2 =$ all strings with at least two ones then $L_3 = L_1 L_2 =$ \{all strings that can be divided into a first section with at least two zeroes and a last section with at least two ones\}.

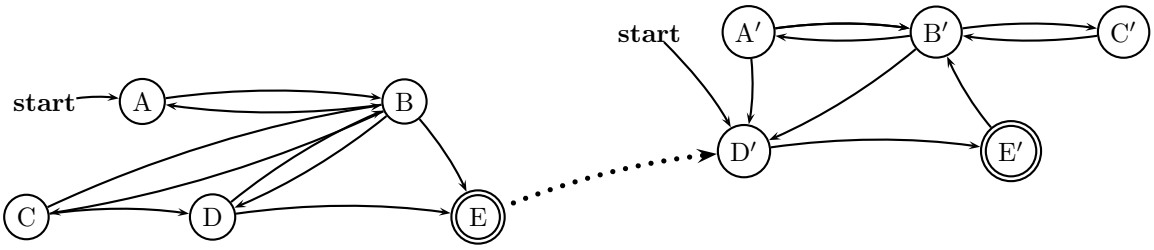
So far we have connected machines in "parallel", meaning that we simulate running two machines at the same time, For concatenation we want to hook up machines in "series". Thus, suppose we have 2 arbitrary machines M_1 and M_2 accepting L_1 and L_2:





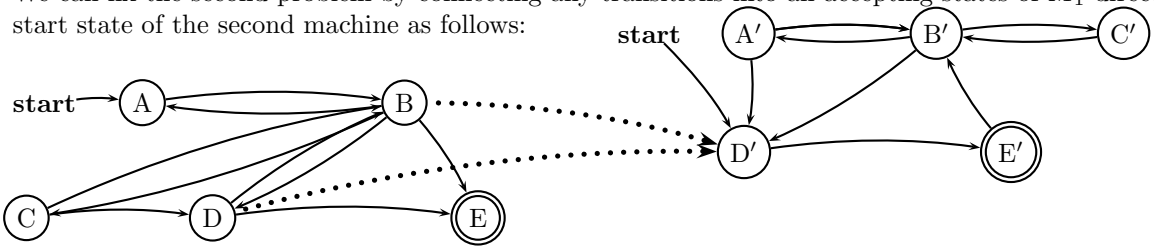
[4] Exercise: Design a finite state machine that recognizes only the empty string.

One attempt at making a machine M_3 to accept L_3 involves linking the final states of M_1 to the start state of M_2:



However, we have some problems here. What is the boundary between the two machines? If we are given a string in L_3, how do we decide which part of it should be checked for membership in L_1 and which part should be checked for membership in L_2. Moreover, the transition from machine M_1 to M_2 does not use up an input character, which violates the definition of a finite state machine.

We can fix the second problem by connecting any transitions into an accepting states of M_1 directly to the start state of the second machine as follows:



The first problem with our construction, however, is more complicated. In some sense, we want to break up a string in all possible ways $x = uv$, checking if $u \in L_1$ and $v \in L_2$. The basis for such a technique is nondeterminism.

Now we have

Theorem: If L_1 and L_2 are regular, then L_1L_2 is regular.

We define L^* to be $\{\varepsilon\} \cup L \cup LL \cup LLL \cup \cdots$. With the above theorem and the fact that regular language is closed under union operation, it would wrong to conclude that if L is regular, then L^* is regular because that is an infinite union and the union closure property holds only for finite unions. However L^* is regular if L is; the next section shows us how to approach the problem.

3 Non-determinism

When you come to a fork in the road, take it.

—Yogi Berra

A non-deterministic finite state machine transitions from one state to a whole set of states. In other words, it tries out several possibilities at the same time, and accepts if any of the possibilities lead to an accepting state.

More precisely, a non-deterministic finite state machine is a quintuple $(\Sigma, S, s_0, F, \delta)$ where:

Σ is the alphabet of the machine

S is the finite set of states of the machine

s_0 is the start state

F is the set of final states of the machine

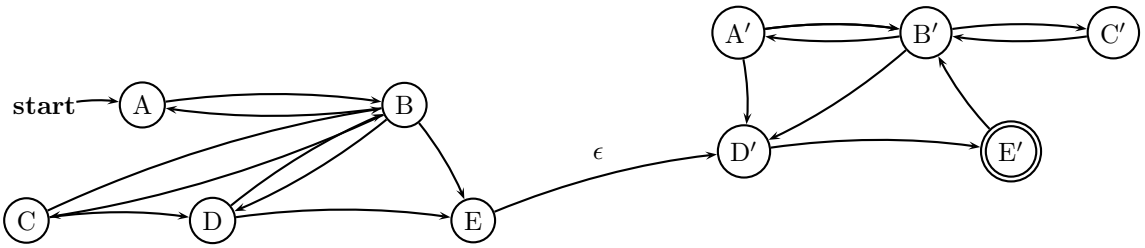
δ is a mapping from $S \times \Sigma \longrightarrow 2^S$. Where 2^S is the set of all subsets of S; it is sometimes called the power set of S and denoted $P(S)$. For example;

$$2^{\{1,2,3\}} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{1,3\}, \{1,2,3\}\}$$

Thus, this transition function maps states and input characters to a set of states.

A non-deterministic finite state machine accepts if any of the states in which it ends up at the end of the input is an accepting state.

In a non-deterministic machine, we may also allow transitions from state to state without gobbling up input; these transitions are called epsilon transitions (ϵ is the null string). Neither nondeterminism nor epsilon transitions really increase the power of our finite state machines, because they can all be modeled with a regular deterministic finite state machine. Nevertheless, they do make life easier. For example, we may now represent the concatenation of two finite state machines as the following nondeterministic machine:



Using nondeterminism, we can also show that for any given a regular language L, the L^{reverse} is also regular. L^{reverse} is simple the language consisting of the reverses of all the strings in L. For example, the reverse of the string "I love CS330" is "033SC evol I".

To prove closure under reversing, we start with the machine M that accepts L. Then we reverse the direction of all the transitions, and switch starting states with final states. If M has more than one final state, then we add a new starting state to the new machine, and add epsilon transitions from this new state to each of the final states of M.

Note that reversing the directions of transitions might mean that we have two transitions on the same symbol coming out of a state, so that the new machine needs to be nondeterministic.

Now, as an exercise, show that L^* is regular when L is by connecting final states to the start state with an epsilon move.

3.1 An example of non-regular language

Consider the language L=$\{0^n1^n|n = 0, 1, 2, \cdots\}$. This is not a regular language. If it was, we should have an finite state automaton which accepts L. Suppose this automaton has N states, then if we input a string 0^i1^i of length $2i, i > N$, by the pigeon hole principle, when we move around and reach the first "1", we move i times and must hit a certain state at least twice. So there will be a loop starting and ending at this state. Cutting the substring which labels the loop or repeating this substring for any times will give us another string which is also accepted by the automaton. But the string can't satify that we have the same number of "0"s and "1"s!

You can find a more detailed proof in Rosen. The same analysis gives us

pumping lemma: Let L be a regular set. Then there is a constant n such that if z is any word in L, and $|z| \geq n$, we may write z = uvw in such a way that $|uv| \leq n$, $|v| \geq 1$, and for all $i \geq 0$, uv^iw is in L.

Pumping lemma is a very useful tool to prove that a language is *not* regular.

Illinois Institute of Technology
Department of Computer Science

Lecture 24: November 20, 2013

CS 330 Discrete Structures
Fall Semester, 2013

1 Regular Expressions

Now we define a *regular expression*, which turns out to be equivalent to finite automata, but can be a nicer way to express some regular languages (those that are accepted by regular expressions and/or finite automata). The empty set ($\emptyset$) is a regular expression, corresponding to the language containing no words. Epsilon (ϵ) is a regular expression corresponding to the language contining only the empty word. (Notice that there is a (somewhat subtle) difference between these two.) Also, each letter x in our alphabet is a regular expression corresponding to the language containing that letter as its only word. We also define regular expressions to be closed under union, concatenation, and Kleene star. Using these properties together, it is possible to define quite complicated languages.

Since we have shown that regular languages have the same closure properties, it is perhaps not surprising that our "new" class of languages turns out to be exactly equivalent to the old one:

Theorem: Every regular expression represents a regular language, and every regular language can be represented by a regular expression.

Proof: By the rules of construction of regular expressions, the first part of this theorem is clear.

To prove the second part of the theorem, we use a variant on the Floyd-Warshall algorithm[1] to construct a regular expression given an arbitrary finite state machine.

Note that this algorithm assumes that the machine has exactly one final state. If it has multiple final states, the algorithm can be run once for each and the results joined with the + (union) operator on regular expressions.

Say the machine has n states. Give each state a unique integer between 1 and n, where state 1 is the initial state and state n is the final state. Define the expression R_{ij}^k to be the regular expression describing strings that carry the FSM from state i to state j, using only states between 1 and k as intermediate states. Then the expression that corresponds to the machine as a whole is R_{1n}^n.

R_{ij}^k is defined recursively:

$$\begin{aligned} R_{ij}^0 &= \{x \in \Sigma | \delta(i,x) = j\} \\ R_{ij}^k &= R_{ij}^{k-1} + R_{ij}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} \end{aligned}$$

As a regular expression can be constructed given any FSM, any regular language can be expressed by a regular expression.

[1] The Floyd-Warshall algorithm, presented in section 25.2 of CLRS, is a $\Theta(|V|^3)$ dynamic-programming algorithm for solving the all-pairs shortest-paths problem on a directed graph. Here we apply it to an FSM, treated as a graph.

2 Languages that are not regular

We talked about pumping lemma which can help determine that a language is not regular. A classic example of these languages is $\{0^n1^n|n = 0, 1, 2, \cdots\}$. There is not a finite state machine to recognize this language since you need infinite many states to keep track of "Where am I" during the computation. This requires infinite memory. For languages like the one above, we can recognize them using a finite state machine with a stack (PDA: push down automata)+: the machine just pushes every 0 into the stack until it sees the first 1. From that point it pops a 0 everytime it sees a 1. If it sees a 0 again, the string is rejected. Otherwise, if the stack is empty when the input ends, the machine accepts the input string. If a language can be recognized by a PDA, we call it context free language. (The name "context free" comes from the property of grammers generating this kind of language).

Can PDA recognize every language? No. The language $\{0^n1^n2^n|n = 0, 1, 2, \cdots\}$ can not be recognized by any PDA. With a slightly advanced version of pumping lemma we can prove that this language is not context free.

How about a PDA with two stacks? It can recognize the above language. Actually a PDA with two stacks is equivilent to a Turing machine, which can recognize any language that is decidable and compute any problem that is computable.

Are there any language with is not decidable (that means, given the input string, no machine can tell you whether the input belongs to that language)? Yes. One example is that we can't have a machine which, given input strings encoding a student's program, test input and standard output, can tell you whether the students program generates the correct output, because there might be dead loops in the student's code and the machine never "dares" to cut off the running!

3 Graphs and finite state machines

There are many questions about FSMs and regular languages that can be solved by treating the FSM as a graph. For instance, given an FSM, does it accept anything?[2] Is a language infinite? Are two languages equal?

4 An application: string matching

One useful application of FSMs is in string matching.[3] The basic idea is to construct an FSM that recognizes the string being searched for and to run it with the text as input.

Of course, this brute force approach is inefficient. In particular, as stated, it will only find the string if it occurs at the very beginning of the text. If the FSM rejects, we could then run it on the text beginning at the second character, and then beginning at the third character, and so on, but that would require $O(mn)$ time, and we can do better. For instance, if the search string is `ababaca` and the text is `abababacaba`, after reaching the third `b` in the text, rather than announcing failure and giving up, we can make use of the fact that the text, so far, still matches the beginning of the search string, as long as we begin looking two characters into the text.

How can this idea be encoded in an FSM? The idea is simple: *on failure, move to the latest state in the FSM that still matches some prefix of the text.* The algorithm is given in detail in CLR, and yields an $O(m + n)$ string matching algorithm. This algorithm and its variants are commonly used in text editors.

[2] In terms of graphs, then, this question would be whether there are any paths from the start state to any of the accept state.
[3] This is discussed in Cormen, Leiserson, Rivest, and Stein, chapter 32