Problems to Program

While working on a variety of software projects, I've often stumbled across minor tasks that were interesting problems by themselves.

These little problems may also give an idea of how theoretical mathematics, geometry, and computer science actually end up getting used in real programs.

Table of Contents

- The Fish on the Ceiling
- If It'S 3 O'Clock, This Must be Chicago
- The Sneaky Flight Plan
- The Careful Crash
- Building 3D Countries
- Word Golf
- Six Degrees of Kevin Bacon
- Exploding the Brain
- As the Worm Turns
- When Atoms Snuggle Up
- The Panic Button
- Breaking the Code
- Fast Graph Matching
- Spotting the Trees in the Forest
- The Knapsack Problem
- What Time is the Next Satellite?
- The Blinking Christmas Tree
- I Recognize the Face, But the Shape is Unfamiliar
- The Gerrymander

The Fish on the Ceiling

One part of a science show at the local planetarium involves displaying a computer-generated model of a fish on the planetarium's hemispherical screen. The fish was generated on a PC, and is described by a series of points and connections between them, in 3D. The laser projector at the planetarium is at the center of the hemisphere.

Given a data file that describes points and lines on a 3D fish,

- a data file that describes points and lines on a 3D fish;
- the dimensions of the hemispherical dome;

Find:

- the X, Y, Z coordinates of the points on the dome which should be illuminated so that an observer at the center of the dome would "see" the fish.

Imagine a light at the center of the planetarium, and place the fish ten feet away. Where would the shadows of the fish fall on the dome?

If your browser can handle VRML files, you might be able to view a sample set of 3D fish data. Be patient, and try to back away from the image if you're too close to see it.

This problem came up while I was doing work for the Studio for Creative Inquiry.

Back to Table of Contents.

If It's 3 O'Clock, This Must be Chicago

The Global Positioning System (GPS) is used to locate anybody, anywhere. To use it, you need a receiver, and a very accurate clock. There is a network of GPS satellites, and at any time, there are at least three of them within range. Each satellite sends out a signal, every 30 seconds, containing the current time, and the satellite's position. How does this tell you where you are?

Suppose you receive a signal from one satellite. Since the satellite is some distance from you, the signal will take a certain time (probably a fraction of a second) to reach you. So since your own clock is very accurate, the time on your clock when you receive the signal will differ from the time in the satellite signal, and this difference is the amount of time it took the signal to travel to you. Knowing the speed of light, (300,000 kilometers/second), you now know that you are so many kilometers away from the position (X,Y,Z) of the satellite.

Do you really need to have a very accurate watch?

Let's pretend for now that the (X,Y,Z) coordinate system used has its origin at the center of the earth, with the Z axis sticking out the North Pole, the X axis sticking through the 0 degree longitude line at the equator, and the Y axis through the 90 degree east longitude line at the equator.

If you assume you are on the earth's surface, this narrows your possible locations down to points on a circle. The signal from a second satellite will tell you enough information to narrow that position down to just two points on the earth's surface, and a third signal will finish the job.

(Actually, the GPS receiver does not assume you are on the earth's surface, so the three signals only narrow your position down to two points, one of which is likely to be out in space somewhere.)

Given:

- three signals, (T,X,Y,Z), from three different satellites, and the times T that you received each signal,

Find:

- your location (X,Y,Z), presumably on the earth's surface.
- your latitude and longitude, assuming you actually are on the earth.

It might help to think about the problem in one or two dimensions first.

Back to Table of Contents.

The Sneaky Flight Plan

A plane wants to get from one place to another without being detected. The location of detection sites is known. As long as the plane stays at least 5 miles away from all such sites, it will pass unnoticed.

Given:

- a flight plan for the plane, consisting of points on the ground between which it flies in a straight line,
- the locations of all the detection sites,

Find

- the minimum distance between the plane and the detectors.

One way to solve this problem is to compute the location of the plane at EVERY possible time (isn't this impossible?), and measure its distance to each detector. Well, if that's impossible, I suppose we could compute it every second. That ought to be good enough!

Another way would be to take a map, draw the flight plan, and then draw a strip around the flight plan, 5 miles on either side, and see if any detector falls within the strip. But a computer can't do this, can it? And it wouldn't even work on paper

if we allowed the detectors to move during the exercise.

Here's another approach. Pick a detection site. Compute the distance from the detection site to the line segment representing the first leg of the flight plan, and then for all the other line segments. Repeat this for all the detection sites. If you keep track of the minimum distance you encounter, that's the closest the plane gets to a detector.

- How do you compute the distance from a point to a point?
- How do you compute the distance from a point to a line?
- How do you compute the distance from a point to a line segment?

Back to Table of Contents.

The Careful Crash

One of the tests that the FAA requires a new helicopter to undergo is the "sudden loss of power" test. The helicopter takes off from the roof of a building, flies straight up 50 feet, and then cuts off power. Suprisingly, the helicopter will not plummet to the ground, although it will fall rapidly; the rotors turn during the fall, and actually provide a certain amount of lift and control. During the test, the pilot has to be able to use this minimal amount of control to guide the craft to a safe landing. Moreover, the helicopter must never come closer than 15 feet to the building at any time!

Given:

- a set of data points (T,X,Y,Z) describing the location of (the center of) the helicopter every second;
- the (X,Y,Z) coordinates of the corners of the building (which you can assume is shaped like a box);

Find:

- the time and position of the closest approach of the helicopter to the building.

You can start this problem by thinking about in two dimensions. Draw a curve near a square, mark a few points on the curve, and figure out the closest point. Could you tell a computer how to do this? The closest point will NEVER be inside the square, unless something strange happens. The closest point is usually one of the corners of the square, except in special cases. When is that? What points in the plane are closest to a corner, and what points closest to a side?

To work on the full problem, you need to know some tools from geometry, including how to compute the distance from a point (X,Y,Z) to:

- another point;
- a line;
- a line segment;
- a plane;
- a rectangle;
- a box (made of 6 rectangles).

This problem is like the Sneaky Flight Plan, but in three dimensions. I ran across this problem while working at Bell Helicopter.

Back to Table of Contents.

Building 3D Countries

We have divided the rectangular volume containing a human brain into a collection of 64 by 64 by 26 "voxels" or little volumes. Several regions of the brain are electrically active, and we are able to measure the amount of such activity in each voxel. We want to group the voxels together into "countries" or regions. We assume that every active region is exactly made up of a collection of voxels with nonzero activity that are neighbors (up or down, left or right, front or back).

Given:

- an array of 64 by 64 by 26 numbers, containing 0 (inactive) or 1 (active),

Find:

- how many separate regions are represented;
- a list of the voxels that belong to each separate region;
- the "center of mass" of each region.

When you find the first nonzero data item, that's a piece of "country 1". How do you find all the other pieces of that country, and how do you resume your search for the next country?

For a look at some representative brain data, see the PSC FMRI page.

Back to Table of Contents.

Word Golf

Lewis Carroll invented a game called "Word Golf" or "Word Ladders" or "Doublets". He tried to change one word into another, by changing a single letter at a time. For instance, you can change "WORD" to "GOLF" by this sequence:

```
WORD
WARD
HARD
CARD
CORD
COLD
GOLD
GOLF
```

Can you change:

- HATE to LOVE?
- GOOD to EVIL?
- HORSE to WAGON?
- POUND to OUNCE?

I got from POUND to OUNCE in 1967, with the help of a few Scottish dialect words. When I tried to do it again this year, it took me a month of browsing in (paper) dictionaries before I came up with my solution.

Can any five letter word be changed into any other five letter word or are some words "aloof"?

You might try making a table of two letter words. On a piece of graph paper, mark 26 columns "A" through "Z", and do the same for rows. Put a check mark in every box that represents a word.

What would a computer need, in order to find a path from one word to another?

How is this problem related to the Building 3D Countries problem?

This is a hard, but interesting problem. One way to approach it is to imagine a graph of all the five letter words, with a link between pairs of words which are one change apart. Now start at the first word on the list, and do a "breadth-first" traversal of the graph, until you reach the target word, or have no more choices.

Software to handle this problem is available in the program "LADDERS", which is part of the Stanford Graph Base.

Back to Table of Contents.

Six Degrees of Kevin Bacon

Given:

- any movie star,

Find:

- a sequence of pairs of movie stars who acted in the same movie, so that the last movie star is Kevin Bacon.

For example, given Matthew Broderick, we can find a chain of movies linking him to Kevin Bacon as follows:

- Matthew Broderick was in *The Cable Guy* with Jim Carrey;
- Jim Carrey was in *Batman Forever* with Val Kilmer;
- Val Kilmer was in *Top Gun* with Tom Cruise;
- Tom Cruise was in *A Few Good Men* with Kevin Bacon.

Why is THIS problem related to the Building 3D Countries problem?

What would a program look like that could play this game? To see one example, try The Oracle of Bacon at Virginia .

Back to Table of Contents.

Exploding the Brain

In the Building 3D Countries problem, we divided the rectangular volume containing a human brain into a collection of voxels. Now let's suppose that someone has already set up the regions; that is, each voxel now contains either "0", because it belongs in no region, or a number between "1" and, let's say, "8", because that voxel is in that region.

Now we're going to make a computer animation of the brain's behavior. We only want to display the shape of the head, not the interior. However, we want the various internal regions to "show up" on the surface of the head. We can pretend that an elf is inside the head, shining a flashlight, and each region casts a shadow on the surface. Assume for now that the shadows don't cause us problems by overlapping in some strange way.

To start with, this problem might sound a little like The Fish on the Ceiling. After all, we are looking at the "shadow" cast by an object on a surface.

Unfortunately, if we stick literally to the idea of shadow casting, then objects closer to the center of projection will appear larger than objects which are the same size, but further away. Moreover, the shape of an object can be distorted. For this particular work, we want to try to preserve the size and shape of the internal objects, so far as possible. We want each shadow to appear roughly the same size as the region it represents. How can we do this?

Even if you don't stretch the region out, you could still take two voxels that are neighbors and make them separate as they move to the boundary. We don't want that. Can you avoid this somehow? One good start is to decide that ALL the voxels in a region will move in the exact same direction, rather than each moving independently. It might be good to try to compute a "center point" of the region, from which you could set the direction.

Given:

- A 64 by 64 by 26 array of voxel data. An entry in the array is 0 if it is part of no region, 1 through 8 if it is part of some particular region, and 9 if it represents the surface of the head.
- The coordinates of a special voxel, the projection point; the data in the other voxels is going to move away from this point, along a line towards the head surface.

Find:

- The places on the head surface which will be intersected by the regions as they move. If a place on the head is intersected by a voxel from region 7, say, then its value should change from "9" to "7". This will help us when we display the head, because that patch of head will be drawn with color 7 rather than color 9.

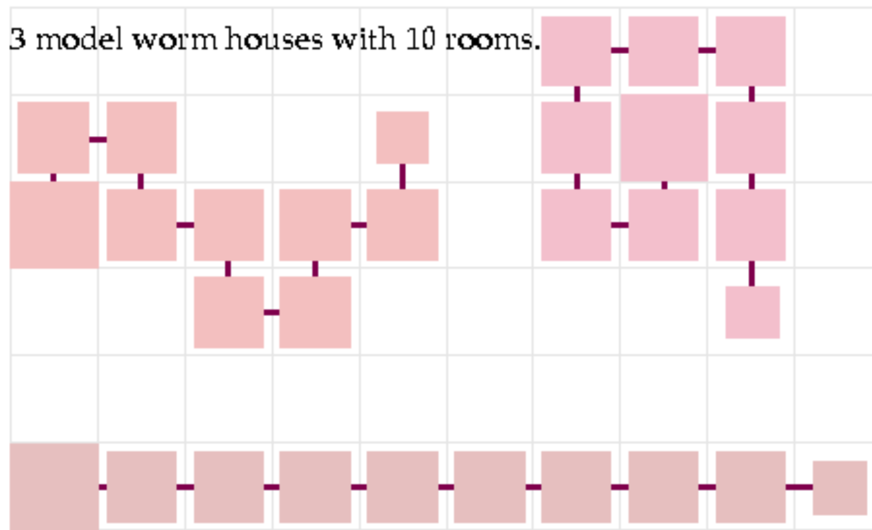
It might help to think of a similar problem on a checkerboard. Think of rules for moving a group of checkers away from the center, in such a way that the group moves together, without dispersing. To choose the direction of motion, you might try to find the "center of mass" of the group. Once you have the direction, you really only need to be able to list the squares that the checkers will pass over on their flight to the boundary.

Back to Table of Contents.

As the Worm Turns

A very fussy two dimensional worm decides to have his burrow built by an architect. The worm draws a sketch of his ideal burrow, which consists of 10 square rooms arranged in a straight line. "I want something like this," he says, "with 10 rooms, no more, no less." The architect feels put off by this boring design. "Must your burrow actually be straight?", she asks. "Come to think of it, that's not important," says the worm, "but the rooms must be square, and arranged so that you can enter the front room and walk from one room to the next until you reach the last room."

The architect pulls out a checkerboard, and begins shuffling around 10 checkers. After a while, she says, "You know, I think there will be quite a *lot* of possibilities to choose from now!"



Can you determine how many possible burrow designs there will be? Can you determine an orderly method of listing every one of them, exactly once? (In an "orderly" method, you produce the designs one after the other, without having to go back and see if your latest design was already listed earlier.)

A *walk* is a sequence of neighboring positions on a grid. A *self avoiding walk* is a walk in which no position is used more than once. We can think of the worm's house as similar to a self avoiding walk.

Back to Table of Contents.

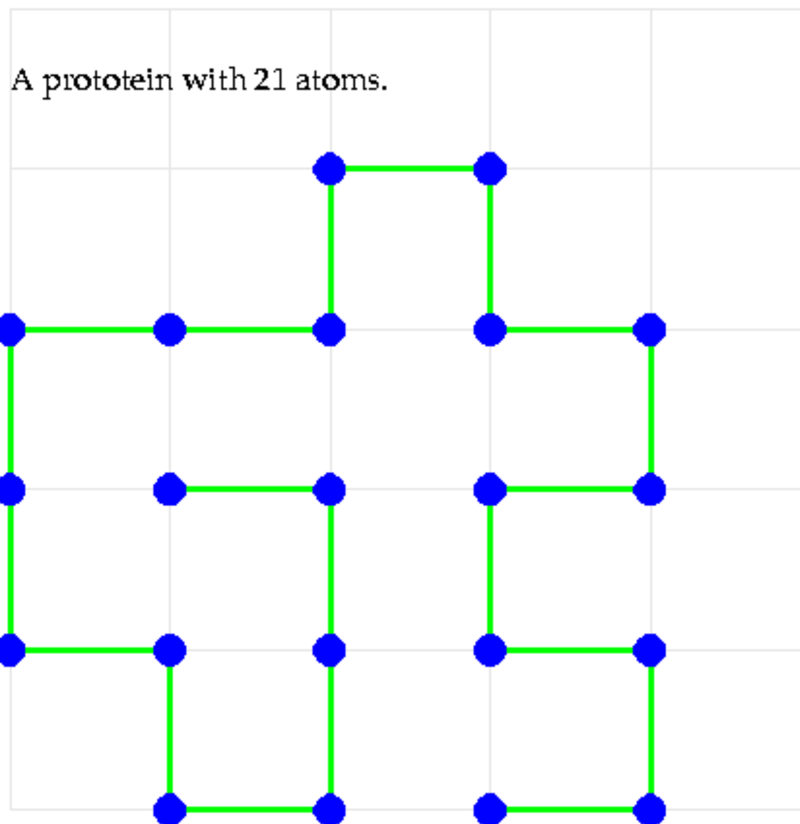
When Atoms Snuggle Up

Proteins are enormous molecules, made up of chains of amino acids, water, and other components. The simplest way of describing a protein is as a ribbon or string, and a very useful description is simply the list or sequence of amino acids that make it up.

However, in living things, proteins tend to "curl up", with parts of the ribbon forming a helix, or two parts lining up like a pair of railings on stairs. This curling up happens because of the unequal distribution of electrical charge. Many areas of the protein have a weak positive or negative charge. Areas of opposite charge attract each other. The protein tends to bend in ways that bring oppositely charged areas closer. Computational biologists spend a great deal of effort trying to notice, catalog, analyze and predict the pattern of protein folding.

We will look at a greatly simplified problem. Our model for a protein will be a string of numbers, 0, 1, and -1, representing neutral, positive, and negative regions. The "formula" for a protein will be its particular string of numbers. Our string proteins will "live" on sheets of graph paper, where they can bend and twist in very limited ways. We will assume that the first "atom" in a protein is associated with one cell of the graph paper. The next atom goes in one of the four unoccupied neighboring cells, and so on.

We can imagine the protein initially being "written" out straight on the paper. Now we expect to see some folding. Now let's suppose that the negative and positive atoms prefer to be near each other, and that the protein is "willing" to bend in various ways to allow this to happen. What is the configuration that is most "comfortable" for the protein?



In order to compare configurations, we'll have to come up with a scoring system. A crude but simple score is as follows: every charged atom that is not the neighbor of an oppositely charged atom adds 1 to the score.

Given:

- The "formula" for a string protein;

Find:

- a configuration of the protein that has the lowest score.

To determine the lowest score, you might simply generate every possible configuration. To do that, you should have thought about the problem mentioned in As the Worm Turns.

Now suppose you are given 10 atoms, and told their types, and you are asked to construct a string protein and design a configuration for it that has the lowest score. How would you do it? Now you will also need to generate permutations (with repetitions), as in Breaking the Code.

Suppose you were given 100 atoms, and you know it is impossible to solve the problem by listing all possible configurations. How could you try to estimate a solution?

(It is often the case that an algorithm that seems natural and efficient is actually unsuitable for large problems. In scientific computing, it is common to reject an algorithm, not because it is incorrect, but because it is too slow, or requires too much storage, to be feasible.)

Back to Table of Contents.

The Panic Button

One of the displays in an aircraft shows the current nearby "threats", that is, objects, moving or stationary, which the pilot should watch or avoid. There isn't room for a big detailed report; the display is a simple circle, divided into four rings, with each ring split into 12 zones. It looks a little like a dart board.

The center ring is always empty. The three remaining rings are for "near", "intermediate" and "far away" threats. It would be nice if the symbol for a threat would be displayed in the zone which roughly corresponds to its direction from the aircraft.

Each zone can display at most one threat. What if there are two threats in the same area, say of types "A" and "C"? The idea is that you would shift the less important threat ("C") clockwise one zone, so it still shows up, slightly out of place.

The problem is, what happens if THAT zone is occupied? Let's suppose that in that case, the zone will be filled by whichever of the two threats is the most important. If the zone was currently occupied by a "B" threat, then "C" would keep shifting clockwise. Otherwise, if there was, say, a "D" there, then "C" would take over the zone, and "D" would start hunting for a place to sit.

Before we can really set up this problem, there's at least one question we have to answer: assuming there MIGHT be a lot of threats, what happens if we try to cram 13 threats into one ring of 12 zones? Presumably, we'd bounce the least important threat out of that ring, and into the further away ring. And what happens if there are no more rings? Presumably, that threat would just be discarded.

Given:

- The aircraft's position over time, (X(T), Y(T)) (assume that height is unimportant);
- The number of threats, NT, their location (XT(T), YT(T)) over time, and their type ("A" = deadly, "B" = serious, "C" = mild, "D = minor",),
- Ranges for near, intermediate and far threats (0 to 5 miles, 5 to 20, 20 to 40),

Find:

- What the warning display should look like, using shifting rules to solve conflicts when two threats should occupy the same position.

Back to Table of Contents.

Breaking the Code

Some homes and cars use a keyless system involving a pad with the ten digits 0 through 9. To open the lock, you need to push a certain number of digits (typically three or four) in the correct sequence. Let's say you can't use any digit more than once.

How many three digit codes are there? Well, you have 10 choices for the first digit, 9 for the second, and 8 for the third, making 720 codes. These include (1,2,3) and (9,5,8). But (1,2,1) for instance is not a legal code, since "1" is repeated. Also,

(1,2,3) is a different code from (3,2,1).

Now suppose you want to generate every possible such code. Isn't it time to call in a computer?

Given:

- The number of digits to choose from (10 in this case);
- The number of digits in the code (let's say "3").

Find:

- A list of all possible codes.

To think about this problem, you might look at a simpler problem, using 2 digits out of 5, and write out all the possible codes. See if you can put these codes into order, and then see if you can figure out a way to determine how to go from one code to the next one.

Suppose you had a bag of M and M's containing 12 blue's and 3 reds. If you reach into the bag without looking, and take 4 candies, one at a time, then:

- What are the possible outcomes?
  What are the probabilities of these outcomes?
- Suppose you just reach in and grab 4 at once. How would this change your answers?

Back to Table of Contents.

Fast Graph Matching

For this problem, a "graph" is a collection of points which may or may not be connected to each other. The points have labels (1, 2, 3, etc) but these are purely for our convenience, and don't represent anything important about the graph. You can think of such a graph as being a collection of buttons, connected by a certain number of strings.

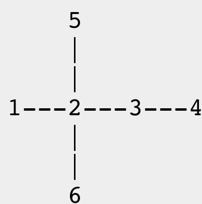
Now suppose we have two graphs. We want to know if they are the "same". By that, we mean, could we find some way to pair the respective points and lines, so that (ignoring the labels), the two graphs are the same?

We can tell right away that two graphs can't match if:

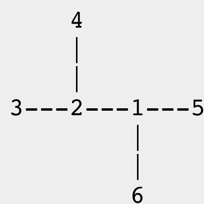
- they have a different number of points;
- they have a different number of lines.

But there are lots of graphs with the same number of points and lines that just aren't the same.

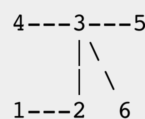
For instance, in the following diagram, two of the graphs are "the same" and one is different:



Graph #1



Graph #2



Graph #3

One reason graph #2 can't match the other two is that it doesn't have any point with four neighbors.

This kind of problem might seem impossible for a computer to handle. It's almost like you absolutely need a picture to see what's going on. Is there even any way we can tell the computer what the graphs look like?

A useful tool for graphs is called the "adjacency matrix". Number the points of the graph any way you want to. Then make a square table, with a row and column for each point of the graph. Put a "1" in row I, column J, if points I and J are connected, and "0" otherwise. Two graphs are definitely the same if their adjacency matrices are the same.

However, two graphs that are the same could have different adjacency matrices if we numbered the points differently. Even though graphs #1 and #3 are "the same", their adjacency matrices are quite different:

```
      1 2 3 4 5 6          1 2 3 4 5 6

   1  0 1 0 0 0 0       1  0 1 0 0 0 0
   2  1 0 1 0 1 1       2  1 0 1 0 0 0
   3  0 1 0 1 0 0       3  0 1 0 1 1 1
   4  0 0 1 0 0 0       4  0 0 1 0 0 0
   5  0 1 0 0 0 0       5  0 0 1 0 0 0
   6  0 1 0 0 0 0       6  0 0 1 0 0 0

   Adjacency #1         Adjacency #3
```

Can we still discover that graphs #1 and #3 are the same? There are many ways to do this. A simple, though inefficient method, uses "brute force", and tries all possible relabelings of one graph to match the other:

- Compute the adjacency matrix for the first graph, and leave it alone;
- Now try any labeling of the second graph, compute the corresponding adjacency matrix, and see if it is matches the first one.
- If there's a match, you're done.
- If there are no more ways to relabel the second graph, you're done, and they can't match.
- Otherwise, try another relabeling.

How do we generate all the labelings? This is just another form of Breaking the Code, where we have to use all the numbers each time.

For our problem, the brute force method should eventually report that if the graphs will match if we relabel graph #3 as follows (there are actually several relabelings that will work):

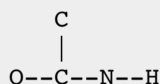
```
    Old label  New label
    ---------  ---------
       1          6
       2          3
       3          2
       4          1
       5          4
       6          5
```

For a quick introduction to graph theory, adjacency matrices, and the problem of determining when two graphs are the same, see the Graph Theory page .

Back to Table of Contents.

Spotting the Trees in the Forest

In studying proteins and other biological structures, it is necessary to identify parts of the protein. Essentially, we have a collection of special graphs, and we are looking to see if any of them occur in the protein. One such graph might be something like this:



```
     C
     |
  O--C--N--H
```

Now when we look for matches, it is NOT enough for the points to be connected in the same way. We also want the atom types associated with the points to match.

Now we can think of a graph as a collection of COLORED buttons connected by string. To say two graphs match is to say we can match buttons, strings and the colors as well.

Can you see a simple way of modifying the adjacency matrix so that if two adjacency matrices match, then the graphs look the same, AND they are also guaranteed to have the same atoms in the same places?

Hint: there's some spare room in the adjacency matrix that we're not really using.

Back to Table of Contents.

The Knapsack Problem

Suppose we are running away from home. We have a knapsack which can hold 50 pounds of supplies, and we have a pile of cans of supplies, of different weights. In order to maximize our chances of making it, we want to take the most amount of supplies we can.

Given:

- A knapsack of a known total weight capacity;
- Objects of different weights;

Find:

- A way to pack the knapsack with the most amount of weight.

It might be hard to see how to even begin this problem. But let's just take a few steps, at least:

- First of all, any object that by itself weighs more than 50 pounds is useless, and can be discarded.
- If any object weighs exactly 50 pounds, then we can pack that, and we're done. That's easy to check.
- If any *two* objects weight exactly 50 pounds, we can pack them, and we're done. That's pretty easy to check.
- I suppose we could check all sets of three, four, five objects, and so on, but isn't this starting to get complicated?

Indeed, there's no guarantee that we can ever completely fill the knapsack, and if we have an awful lot of little objects of different weights, we could never check all the combinations. Perhaps we ought to consider just taking a stab at the problem.

For instance, here's one way to get a rough solution:

- Take the heaviest object that will fit in the knapsack.
- Add the heaviest remaining object that will still keep the weight below 50.
- Keep adding objects until you reach 50 (fat chance!) or can't add any more objects without going over the weight.

This method has the advantage that you always know what to do next on each step. This is an example of a "greedy" algorithm, because when picking a step, it always takes the biggest one possible.

There's no guarantee that this will produce the best solution. In fact, if, for example, our objects weigh 26, 25, and 25 pounds, then the greedy algorithm will grab the 26 pound weight and halt, whereas the two 25 pound weights give a perfect solution.

This suggests that we might start our solution using the greedy algorithm, but then try to improve it, perhaps by deciding to recompute the solution without one of the heaviest weights. Can you think of a way to carry this out?

Now, suppose that the objects each has a dollar value, and that your goal is to maximize your haul. You still can only carry away at most 50 pounds of loot, but you're going to sell it afterwards, and 1 pound of gold would be a better thing to take than 50 pounds of mud. How would you think of trying to solve this problem?

Back to Table of Contents.

What Time is the Next Satellite?

Suppose you have a pile of messages to transmit in the next 24 hours. All the messages must be transferred via a satellite, which passes overhead, and is in range, for 20 minutes out of every hour. Each message has to be transmitted entirely, not in parts, and we can work out how long each message would take to be transmitted.

Given:

- A set of messages, containing varying amounts of text;
- A set of priorities ("URGENT", "ROUTINE", "LOW");
- The time it takes to transmit a single character of text;
- The length of time the satellite is available during one pass, and the number of passes it makes in 24 hours;

Find:

- Can we send all the messages?
- If not, what is the most NUMBER of messages we can send?
- What is the most AMOUNT of text (measured in characters) that we can send;
- If we are paid one cent per character for each LOW priority message completely transmitted, two cents for ROUTINE, and three cents for URGENT, how can we maximize our income?

This is a variation of The Knapsack Problem.

Back to Table of Contents.

The Blinking Christmas Tree

People risk electrocution stringing together lines of light bulbs on Christmas trees. Sometimes the networks can be quite complex. If one bulb goes out, all the bulbs on its strand go out, or perhaps just the bulbs "downwind" from it. Suppose we were watching the Christmas tree in the dark, and someone turned on and off various parts of the circuit. Could we tell which bulbs were connected?

To make this problem a little simpler, suppose we have a sort of circuit board. On the side we can see, there are light bulbs, and some on/off switches. With all the switches off, none of the bulbs are lit. When we turn any one switch, some of the lights go on. We guess that the bulbs are connected to each other and the switches by some crazy system of wires. Any bulb may be completely unconnected, or connected to several other bulbs. Can we figure out the connections?

Given:

- A set of light bulbs and switches;
- An "initial state" (ON or OFF) for each light bulb;
- The behavior of all the light bulbs when any one switch is turned;

Find:

- As much as possible about the wiring connections of the bulbs;

This problem is actually inspired by biology. The light bulbs are really proteins. Increasing the quantity of a "biological switch" can cause some types of proteins to increase in quantity, others to decrease, and most to remain unchanged. It is assumed that a few proteins are directly affected by the stimulus, but that the others that change are doing so because of indirect effects of the stimulus; that is, the stimulus only increases protein A directly, but increasing protein "A" might cause protein "B" to decrease. In other words, there's a sort of "wire" from "A" to "B". Scientists want to know where all these wires are.

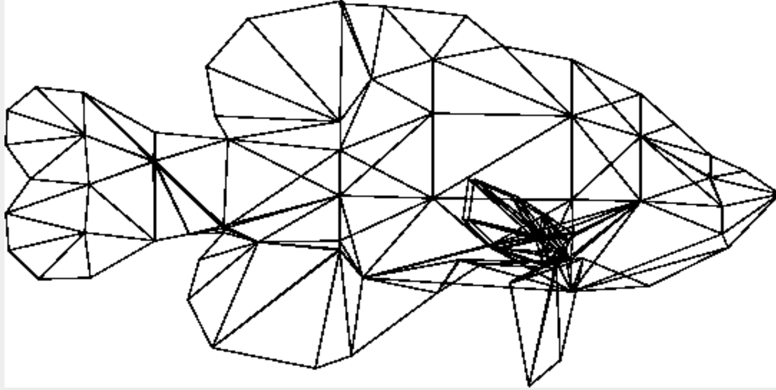
Back to Table of Contents.

I Recognize the Face, but the Shape is Unfamiliar

Imagine your surprise when you get home from the store with your brand new Gizmo, only to find that it is in pieces. The pieces are something like TinkerToys: long sticks, and wooden drums which hold the ends of one or more sticks. The

drums and sticks are numbered, and all you have to do, say the instructions, is "connect drums 7 and 12 with stick 4, then ...". Now I suppose we'd admit that we have all the information about what the final structure looks like, but we actually can't put it together in our minds.

Now let's make the problem a bit more abstract. Suppose I have a list of numbered points, and a list of lines between them, which can be represented simply as the pairs of points they connect. If I look over the data carefully, I might be able to spot that, say, three particular lines actually form a triangle. (I don't know the shape of the triangle, since I don't get the actually coordinates of the points, but I can still tell that it is a triangle.)



Suppose that this information actually represents a three dimensional object. The points are actually points on the surface, and the lines connecting the points are approximations to lines along the surface. Then the triangle we saw earlier would correspond to a triangular patch of surface. Other surface patches might have four, five, or many more sides. These patches are called *faces*.

Just from the point and line data alone, can you figure out the faces?

Since we are assuming the data comes from a simple decomposition of the surface of a 3D object, we can actually assume that the points and lines can be drawn on a piece of paper, without any lines having to cross other lines. In this representation, any child could instantly see all the faces in the shape (except perhaps one, made by the outermost lines). How hard is it to tell a computer how to do this?

Hint: it's harder than you think. But it's a fair amount of fun just trying to think of an algorithm for coming up with faces, making sure your face can't be broken down into two simpler faces, and that you get all the faces.

This problem also came my way while I was doing work for the Studio for Creative Inquiry. Part of this work involved converting graphics data from one format to another. Most formats store information using faces, but a few record points and lines. Trying to get point and line data into face format is a real task.

Back to Table of Contents.

The Gerrymander

In a group of two people, the saying goes, there are *three* opinions. When large groups of people need to agree on a single choice, an election can be held to make the decision. But it has been known for ages that the election **process** can be arranged to prefer, or even to guarantee, a particular election **result**.

Consider the case of representative democracy, where a country or province is divided into smaller regions, each of which elects a representative to a common council. One might think it a desirable property of this system that the constitution of the council be similar to that of the people who elected them. On the other hand, if you were a politician, this might *not* be how you think!

In that case, you might hope to pack more of your party onto the council. One way to do this would occur if you were allowed to decide the boundaries of the districts. We might suppose that each district has to be connected, and perhaps needs to have roughly the same number of inhabitants as all the others. But if we have any idea of the voting preferences of the inhabitants, we can surely find a way to cleverly draw the boundaries of the districts to help our party.

This can be done by trying to all your own party members live in districts where they form a respectable majority; the

members of the opposing party live either as minorities in such a mixed district, or else in "pure" districts where there are none of your party. If things work out for you, every member of your party has a vote that is worth double that of the other party.

One of the early researchers in this field was Elbridge Gerry, signer of the Declaration of Independence, vice President under James Madison, but in 1811, the Governor of the state of Massachusetts. His party happened to be the Jeffersonian Republicans. He arranged for the voting district of Essex to be constructed in a way that encircled the city of Boston. A cartoonist made fun of its snakey shape by drawing on a pair of wings and calling the result a "gerrymander", a name which has been applied to this practice ever since.

Now to make this into a computational problem, suppose we have a graph of "wards", which are the "atomic units" of our voting system. Suppose for each ward we are given the numbers D of Democrats and R of Republicans. Moreover, suppose we are given a graph that records the connectedness of each district to its neighbors. We are required to create 10 voting districts, each of which is constructed by conglomerating adjacent wards. Each district should have "roughly" the same population, but we want to find arrangements that

- maximize the strength of the Democrats;
- maximize the strength of the Republicans;
- have the same percentage of "Republican" districts as there is a percentage of Republicans in the population;
- give every district roughly the same ratio of Democrats and Republicans;

Can you think of a procedure to approach this problem?

Back to Table of Contents.

You can return to the HTML web page.

Last revised on 27 February 2006.