Top Ten Algorithms Class 13

John Burkardt
Department of Scientific Computing
Florida State University

.....

http://people.sc.fsu.edu/~jburkardt/classes/tta_2015/class13.pdf

23 November 2015

Our Current Algorithm List

1. Area of a triangle
2. Back Propagation algorithm
3. Bank routing number checksum for error detection
4. Barycentric coordinates of point in triangle
5. Bernoulli number calculation
6. Bootstrap algorithm
7. Collinearity of three points
8. Computational geometry (triangle area, containment, mapping)
9. Data stream: most common item
10. Discrete Cosine Transform

Our Current Algorithm List

1. Discrete Fourier Transform
2. Euclid's greatest common factor algorithm
3. Finite Element linear triangle basis function evaluation
4. Gram-Schmidt vector orthogonalization algorithm
5. Hamming error correcting codes
6. ISBN (International Standard Book Number) checksum
7. k-means clustering algorithm
8. Luhn/IBM checksum for error detection
9. Monte Carlo Sampling
10. PageRank algorithm for ranking web pages

Our Current Algorithm List

1. Pancake flipping algorithm for genome relations
2. Path counting with the adjacency matrix
3. Power method for eigenvector problems
4. Probability evolution with the transition matrix
5. Prototein model of protein folding
6. Quasirandom number generation
7. QR (Quick Response) images and error correction
8. QR matrix factorization
9. QR iteration for eigenvalues
10. Reed-Solomon error correcting codes

Our Current Algorithm List

1. Ripple Carry algorithm
2. Search engine indexing
3. Signal detection in noisy data
4. Trees for computational biology
5. Triangle-contains-point algorithm
6. Triangulation of a polygon
7. UPC (Universal Product Code) checksum for error detection
8. Zero Knowledge Proofs

Student Presentation

Anna Yannakopoulos, *"Recovering Solutions from Insufficient Data"*

This topic is based on two articles by Cleve Moler, "Mr Matlab", both of which try to solve a problem with insufficient data.

1) I'm thinking of two numbers whose average is 3. What numbers am I thinking of? Reference: www.mathworks.com/clevescorner/dec1990

2) A signal of millions of values was sent. But I only received a compressed signal, containing weighted averages. Can I recover the exact original signal? Reference: http://www.mathworks.com/tagteam/65074_91850v00_NN10_Cleve.pdf

Under the right conditions, the answer is yes.

Student Presentation

Isaac Lyngaas, *"Computing With Very Large Numbers"*

We are familiar with the limitations of computer arithmetic, including a biggest and smallest positive real number, and about 16 decimal digits of accuracy in arithmetic.

What do we do if we need to calculate π to a hundred ... or a million digits?

What if we need to factor a prime number that has a hundred digits?

reference: Brian Hayes, "The Higher Arithmetic", American Scientist, September/October 2010.

The McNuggets Problem

At one time, McDonalds sold Chicken McNuggets in packs of 6, 9, or 20. A customer, sent to buy 43 McNuggets, realized this was impossible. There did not exist nonnegative integers a, b, c so that:

$$43 = a * 6 + b * 9 + c * 10$$

This is one example of problems in which we are given objects with a certain value, and asked to construct a new object. We might be allowed to use each object several times, or no more than once.

The Balanced Partition Problem

Ten integers are about to play a game, and so it's time to choose up sides.

2 10 3 8 5 7 9 5 3 2

The goal of the balanced partition problem is to break the numbers into two groups with equal sum, or as nearly equal as possible.

Balanced Partition Solution Cost

Given N numbers, how much work will it generally take to solve the balanced partition problem?

No one has found a general procedure that can work much faster than brute force - in other words, trying every possible arrangement of the values. For N numbers, there are 2^N possible subsets. Each subset and its complement form a partition.

If we solve the problem by checking every subset, the problem is NP-hard...as N increases, the work 2^N exceeds any polynomial.

For small N or special cases, the problem may be solvable, but in general, this is an "impossible" problem.

Greedy Algorithm

A simple algorithm works like two greedy team captains, alternating turns, and always choosing the largest remaining number for their team. For our data, this would yield the following teams

```
A: #9 #1 #7 #3              #5
    2 10  3  8               5          Sum = 28
    2 10  3  8  5  7  9  5  3  2
                5  7  9  5  3  2   Sum = 26
B:             #6 #4 #2     #8#10
```

Greedy but Fair Algorithm

To reduce the advantage of the first chooser, we can do two picks at a time after the first one.

```
A: #9 #1        #5 #4        #8
    2 10         5  7         3          Sum = 27
    2 10  3  8   5  7  9  5   3  2
          3  8         9  5      2       Sum = 27
B:       #7 #3        #2 #6    #10
```

Since we really have to sort the data to make choices, the amount of work here is $O(N \log(N))$.

Unfortunately, this method it not guaranteed to find the solution, hence, it is not a general solution algorithm, so we have not beaten the 2^N work requirement.

Actually, another problem with the greedy algorithm is that it chooses both subsets to be the same size.

Karmarkar-Karp Algorithm

The Karmarkar-Karp algorithm sorts the data, then replaces the two largest numbers by their absolute difference. This is equivalent to deciding that these two numbers must be assigned to different subsets. This process continues until only one number is left, which is the discrepancy between the two subsets. If it is zero, we have found a perfect partition.

```
Data                  Difference
6  9 13 17 19             2
2  6  9 13                4
2  4  6                    2
2  2                       0
0                     <-- Perfect solution has been found
2       | 2               0 = 2 - 2
6       | 2 4             2 = 6 - 4
6 9     | 2 13            4 = 13 - 9
6 9 17  | 13 19           2 = 19 - 17
```

The Subset Sum Problem

The balanced partition problem gave us N numbers to divide into subsets of equal sum.

If all the numbers add up to S, then we are looking for two subsets that add up to $S/2$, which is really the same as asking for one subset that adds up to $S/2$.

This is just a special case of the subset sum problem: select a subset of N numbers which add up to the target T.

A Pseudo-Fast Algorithm

Let M be the target value, and set vector T(0:M) to -1.

We can get a sum of 0 by using no elements at all, so set T(0) to 0. The next element of the set might be 2. Since 0 is "reachable", we know that $0 + 2$ is reachable: Set $T(2) = 2$.

The next value might be 3. Since 0 is reachable, so is $0 + 3$, set T(3) to 3. Since 2 was reachable, so is $2 + 3$, so set T(5) to 3.

If the next value is 9, set $T(9) = 9$, $T(12) = 9$ and $T(14) = 9$, because these became reachable using 9. (Actually, we can't use 9 twice in one sum, so the actual check will count **down**.)

Once we have considered all the numbers, then T(M) is either -1 (couldn't reach it) or it has a nonnegative value, indicating the last entry in the sum. By backtracking, we can recover the whole sum.

A Pseudo-Fast Algorithm

```
1  function t = subset_sum ( m, n, a )
2
3    t = zeros ( m, 1 );
4
5    for i = 1 : n
6      for j = m - a(i) : -1 : 0
7
8        if ( j == 0 )
9          if ( t(a(i)) == 0 )
10           t(a(i)) = a(i)
11         end
12       elseif ( t(j) ~= 0 && t(j+a(i)) == 0 )
13         t(j+a(i)) = a(i)
14       end
15
16     end
17
18   end
19
20   return
21 end
```

Listing 1: Subset Sum

The Subset Sum Problem

An example calculation might have a target value of 30, using the values $[2, 3, 9, 14, 23, 11, 2, 5]$. At the end of the calculation, the T vector would look like:

```
      0   1   2   3   4   5   6   7   8   9
-----------------------------------------------
00  [ 0, -1,  2,  3,  2,  3, -1,  2,  5,  9,
10    5,  9,  9, 11,  9,  2, 14, 14,  2, 14,
20   11,  2, 11, 14,  2, 14, 14, 11, 14,  2,
30   11 ]
```

meaning that $30 = 11 + 14 + 3 + 2$

This procedure is guaranteed to find an answer if it exists. Does this mean the problem is not NP-hard? No, because the amount of work depends on the value M, which is not controlled by N. Given any number of values N, we can choose a target M that is N^2, N^3 or any power of N we like. Thus, we can never give a power of N that caps the possible work.

Easy Subset Sum Problems

Although the general subset sum problem is NP hard, it becomes easy if the values are superincreasing, that is, if they can be listed in such a way that each element is greater than the sum of all the previous ones.

The classic example of a superincreasing set is

$$\{1, 2, 4, 8, 16, 32, 64, ...\}$$

but we could use a set such as

$$\{2, 7, 11, 21, 42, 89, 180, 354\}$$

The greedy algorithm to form a sum s selects the largest element no greater than s, then the largest elementno greater than the remainder, and so on. This takes at most $O(n)$ work.

For example, the sum 208 is formed by $180 + 21 + 7$, and we can also see, just as fast, that there is no subset that will sum to 207.

Public Key Cryptography by Subset Sum

Alice wants to send Bob a message over a public line.

Bob uses a public key encryption scheme, with two keys. He has posted his public key so that anyone can send him a message, but he keeps the private key to himself.

Alice converts her message to a sequence of 0's and 1's, and then uses Bob's public key to encrypt her message, and send it over the insecure transmission line.

Bob receives the message, and uses his private key to decrypt it.

If a stranger already knew Alice's message, he could use Bob's public key and generate the same encrypted message. But having the encrypted message and the public key does not allow the stranger to recover the original message.

Encryption is like a "trapdoor", easy one way, hard the other.

Public Key Cryptography by Subset Sum

The system works by linking a pair of mathematical problems **X** and **Y**, both of which are subset sum problems.

Problem X uses a set of N values $\{x_1, x_2, ..., x_N\}$ and problem Y uses $\{y_1, y_2, ..., y_N\}$. The Y values are superincreasing, but the X values are randomly chosen, so problem Y is easy to solve and X is NP hard.

Suppose the first N bits of Alice's message are $a_1, a_2, ..., a_N$. She computes the corresponding code word w using problem X (the public key):

$$w = \sum_{i=1}^{N} a_i x_i$$

If her message is longer than N bits, she repeats this process as often as necessary, yielding a sequence of code words.

The Stranger Has a Hard Problem to Solve

Suppose a stranger sees the encoded word w. Because the public key is available, he also knows the values $\{x_1, x_2, ..., x_N\}$.

But in order to recover the original message bits, he must solve the NP hard subset sum problem:

$$\textit{Find the 0/1 values } a_1, a_2, ..., a_N \textit{ so that } w = \sum_{i=1}^{N} a_i x_i$$

While not impossible, this computation can be made extremely time consuming with a moderate value of N.

Bob Has an Easy Problem to Solve

However, when Bob receives Alice's coded message, his decoding task is easy. Using his private key, he applies a simple transformation to Alice's message word w, and solves problem Y, the easy subset sum problem.

The technical details involve a modulus m, a multiplier k, and k^{-1}, the inverse of k with respect to the modulus m. The private key values y are chosen first, and the public key values x are:

$$x_i = k * y_i \mod m$$

Upon receiving w from Alice, Bob computes

$$w' = k^{-1} * w \mod m$$

And, "magically", if Bob solves the simple subset problem Y for w', he gets back the original bits $a_1, a_2, ..., a_N$ that Alice encoded.

Diffie Hellman Merkle

The public key cryptography system outlined here is known as the Diffie-Hellman or Diffie-Hellman-Merkle scheme.

This scheme was invented because encrypted communication typically requires that the two correspondents first agree on a common secret key. But how could the key be kept secret if we haven't set up our encryption yet?

The public-key system allowed the encryption key to be transmitted safely. Subsequent communication would be done with a more efficient symmetric key system.

The example implementation using the subset sum problem was broken by one of the authors. However, similar encryption schemes have been developed, using modular multiplication.

Student Volunteer