Coding contest trick: Meet in the middle

Cosmin Negruseri
10 august 2012

Meet in the middle (sometimes called split and merge) is a clever idea that uses caching to get efficient solutions. Much like divide et impera it splits the problem in two and then tries to merge the results. The benefit is that by using quite a bit of extra memory you can tackle problems of twice the size you could before.

Before we go on I want to mention that the additional problems are the best part of the article. Now let's go through a few applications of the trick.

4sum (popular interview question)

> *Given A, an array of integers, find out if there are any four numbers in the array that sum up to zero (the same element can be used multiple times). For example given A = [2, 3, 1, 0, -4, -1] a solution is 3 + 1 + 0 - 4 = 0 or 0 + 0 + 0 + 0 = 0.*

The naive algorithm checks all four number combinations. This solution takes $O(N^4)$ time.

A slightly improved algorithm brute forces through all n^3 three number combinations and efficiently checks if -(a + b + c) is in the original array using a hash table. This algorithm has $O(n^3)$ complexity.

By now, you're probably wondering how the meet in the middle technique can be applied here. The critical insight comes from rewriting `a + b + c + d = 0` as `a + b = -(c + d)`. Now we store all n^2 sums `a + b` in a hash set `S`. Then iterate through all n^2 combinations for c and d and check if `S` contains -(c + d).

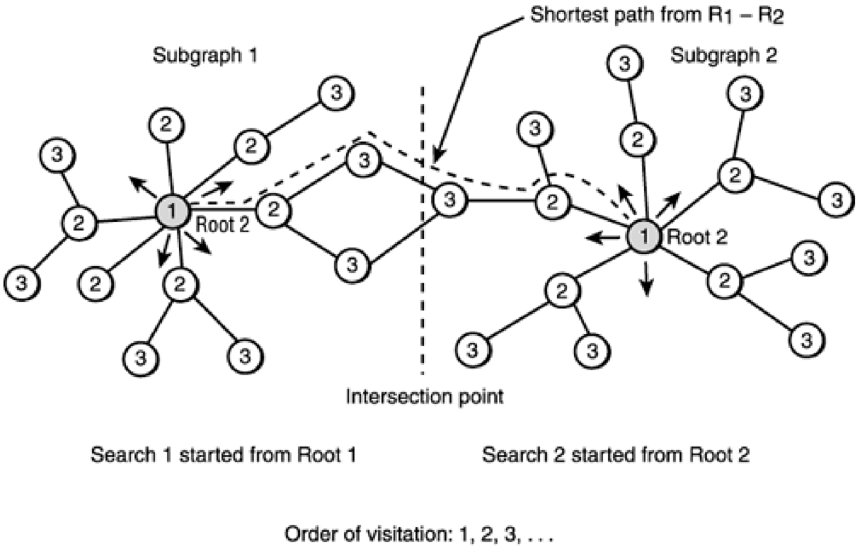
Here's how the code looks

```
def 4sum(A):
  sums = {}
  for a in A:
    for b in A:
      sums[a + b] = (a, b)

  for c in A:
    for d in A:
      if -(c + d) in sums:
        print (sums[-(c + d)][0], sums[-(c + d)][1], c, d)
        return

  print "No solution."
```

This algorithm has $O(n^2)$ time and space complexity. There's no known faster algorithm for this problem.

Bidirectional search

> *Find the shortest path between two nodes nodes in a large graph, for example the Facebook friendship graph.*


Shortest path from R1 – R2
Subgraph 1
Subgraph 2
Root 2
Root 2
Intersection point
Search 1 started from Root 1
Search 2 started from Root 2
Order of visitation: 1, 2, 3, . . .

img source

The breadth first search algorithm is a standard approach for this problem. If the distance between two nodes is `k` and the average degree in the network is `p` BFS explores $O(p^k)$ nodes.

A better solution starts from both nodes and sees when the two search frontiers meet. This reduces the number of states explored to $O(p^{k/2})$.

The approach works well with both path finding problems on explicit graphs and with implicit state graphs like the ones you find in games.

Despre Blog

Ultimele insemnari

» Algoritmiada Runda Finala 2017
» Surse libere
» Linear Algebra Resources
» infO(1) Cup
» Concursul National de Informatica "Adolescent Grigore Moisil"
» Mindcoding 2017
» Interview puzzle: Count distinct (2)
» Catalysts Coding Contest 2017 Bucuresti
» Retrospectiva anului 2016
» Starea natiunii 2016 - Bun venit la scoala

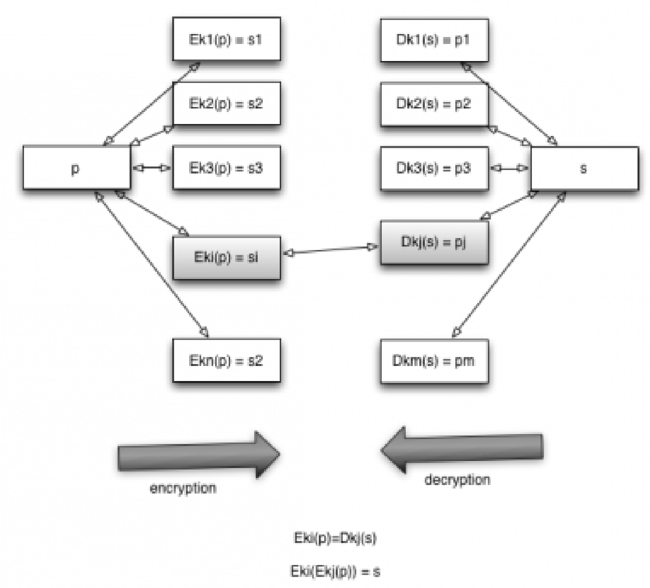
Categorii

» algoritmiada (7)
» algoritmica (1)
» concursuri (11)
» doi la suta (1)
» Evenimente (7)
» facebook (1)
» Features (3)
» girl camp (2)
» Girls (2)
» Girls camp (2)
» girls programming camp (3)
» google (1)
» infoarena (6)
» internships (1)
» interviu (12)
» life the universe and everything (7)
» organizare (3)
» potw (31)
» preoni 2008 (3)
» probleme (9)
» Selectie Girls Programming Camp (1)
» statistici (1)
» stiri (76)
» video (6)

Blogroll

» Vivi's blog
» WebDiarios de Motocicleta
» Catalin Francu
» Florin Manea
» Stevey's Blog Rants
» Joel on Software
» Developing for developers

Breaking 2DES



DES is an encryption standard which uses 56 bit keys. Today computers can use a brute force approach to break the encryption. One simple approach to make the encryption more secure is to apply it twice, using two different keys. This approach is susceptible to the meet in the middle attack developed by Diffie-Hellman. 3DES works around this problem by encrypting the message 3 times using 2 keys.

Let’s see why 2DES is vulnerable. Let `Ek` be the encryption function using the secret key `k` and `Dk` the decryption function using the secret key `k`. 2DES uses two keys, k and K. Ek(EK(p)) = s does the encryption and DK(Dk(s)) = p does the decryption.

Diffie-Hellman’s meet in the middle attack trades off space for time to find out the two secret keys.
For the pattern p it tries all the possible keys to obtain a set of numbers corresponding Ek(p). Also for the pattern s it uses all the possible keys to decrypt s, Dk(s).
If we find any match in the two sets it means that Eki(p) = Dkj(s) so the secret keys are ki and kj.
The naive brute force algorithm does $2^{56} * 2^{56}$ iterations going through all possible values of k1 and k2 while this algorithm uses $2^{56} * 56$ memory to store all Eki(p) and does 2^{56} work to find a match.
This is quite a bit of space and quite a bit of computation time. But a for a large enough company or country it starts being within the realm of posibility.

The problem DOUBLE the International Olympiad in Informatics 2001 was basically asking to break 2DES for keys of size 24 which is quite feasible.

Discrete logarithm

> *Given n a prime number and p, q two integers between 0 and n-1, find k such that $p^k = q$ (mod n).*

The naive solution goes through all possible values of k and takes `O(n)` time.

The baby-step, giant-step algorithm solves the problem more efficiently using the meet in the middle trick.
Let's write k = `i[sqrt(n)] + j`
Notice that `i <= sqrt(n)` and `j <= sqrt(n)`.
Replacing k in the equality we get $p^{(i[sqrt(n)] + j)} = q \pmod{n}$.
Dividing by p^j we get $p^{i[sqrt(n)]} = qp^{-j} \pmod{n}$.
At this point we can brute force through the numbers on each side of the equality and find a colision.

The algorithm takes `O(sqrt(n))` space and `O(sqrt(n))` time.

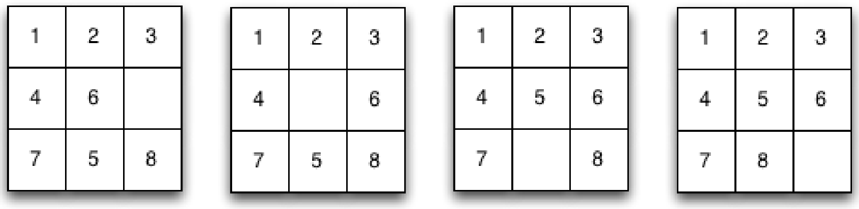
Caveats

Unlike divide and conquer meet in the middle can't be applied recursively because the sub problems don't have the same structure as the original problem.
Bidirectional search can be often replaced by some search algorithm that uses some heuristics like A*.

Additional problems

1. **Friend of a friend**(interview question) Given two user names in a social network design an efficient way to test if one is a friend of a friend of the other.
2. **6 degrees of separation** Given two user names in a social network design an efficient way to test if they are at most 6 friends apart.
3. **Equal partition** Given a set A of 40 real numbers, find out if there is any way to split A in two sets such that the sums of their elements are equal. (Hint: complexity $O(2^{n/2})$)
4. **Minimal vertex cover** Given a graph of n nodes (n <= 30), find out a set with the smallest number of vertices such that each edge in the graph has at least one node inside the set. (Hint: complexity $O(3^{n/2})$)
5. **Square fence** You're given an array L which represents the lengths of n planks. You have to answer if there's any way to form the edges of a square fence using all the planks without breaking or overlapping them. (Hint: complexity $O(4^{n/2})$)

6. **8 puzzle** The 8 puzzle is a sliding tile game of 3×3 slots with 8 tiles and one empty slot. At each step you can move one of the orthogonally neighbouring tiles to the empty slot. The game starts from a random initial configuration and the purpose is to get to the final sorted configuration in the fewest number of moves. Figure out an efficient algorithm that solves the 8 puzzle. (Hint: Each position is solvable in at most 31 moves) In the picture we see a sequence of moves that solves the puzzle.



7. **4 reversals** We are given two equal length strings S and T. Figure out if we can get string T starting from string S and applying 4 substring reversal operations. (Hint: complexity $O(n^5)$)

Try to solve them in the comment section.

Categorii: