How does this C++ function use memoization?

Asked 16 years, 2 months ago Modified 11 years, 5 months ago Viewed 1k times

4

```
#include <vector>
std::vector<long int> as;

long int a(size_t n){
  if(n==1) return 1;
  if(n==2) return -2;
  if(as.size()<n+1)
    as.resize(n+1);
  if(as[n]<=0)
  {
    as[n]=-4*a(n-1)-4*a(n-2);
  }
  return mod(as[n], 65535);
}
```

The above code sample using memoization to calculate a recursive formula based on some input `n`. I know that this uses memoization, because I have written a purely recursive function that uses the same formula, but this one much, much faster for much larger values of `n`. I've never used vectors before, but I've done some research and I understand the concept of them. I understand that memoization is supposed to store each calculated value, so that instead of performing the same calculations over again, it can simply retrieve ones that have already been calculated.

My question is: how is this memoization, and how does it work? I can't seem to see in the code at which point it checks to see if a value for n already exists. Also, I don't understand the purpose of the `if(as[n]<=0)`. This formula can yield positive and negative values, so I'm not sure what this check is looking for.

Thank you, I think I'm close to understanding how this works, it's actually a bit more simple than I was thinking it was.

I do not think the values in the sequence can ever be 0, so this should work for me, as I think n has to start at 1.

However, if zero was a viable number in my sequence, what is another way I could solve it? For example, what if five could never appear? Would I just need to fill my vector with fives?

Edit: Wow, I got a lot of other responses while checking code and typing this one. Thanks for the help everyone, I think I understand it now.

c++ dynamic-programming memoization

Share
Improve this question
Follow

edited Jul 1, 2013 at 2:34

Andrew Barber
40.1k ● 20 ● 96 ● 124

asked Sep 29, 2008 at 3:07

Zachary Wright
24k ● 11 ● 45 ● 57

5 Answers

Sorted by: Highest score (default)

6

`if (as[n] <= 0)` is the check. If valid values can be negative like you say, then you need a different sentinel to check against. Can valid values ever be zero? If not, then just make the test `if (as[n] == 0)`. This makes your code easier to write, because by default vectors of `int`s are filled with zeroes.

Share Improve this answer Follow

answered Sep 29, 2008 at 3:08

C. K. Young
223k ● 47 ● 390 ● 443

1

The code appears to be incorrectly checking is (as[n] <= 0), and recalculates the negative values of the function(which appear to be approximately every other value). This makes the work scale linearly with n instead of 2^n with the recursive solution, so it runs a lot faster.

Still, a better check would be to test if (as[n] == 0), which appears to run 3x faster on my system. Even if the function can return 0, a 0 value just means it will take slightly longer to compute (although if 0 is a frequent return value, you might want to consider a separate vector that flags whether the value has been computed or not instead of using a single vector to store the function's value and whether it has been computed)

Share Improve this answer Follow

answered Sep 29, 2008 at 3:36

Plasmer
1,120 ● 5 ● 10

I think it still takes exponential time when the check is as[n]<=0, just with a smaller base. Indeed, try computing a(10000) -- it will never finish, even though it takes no time with the right check. – A. Rex Sep 29, 2008 at 3:45

0

If the formula can yield both positive and negative values then this function has a serious bug. The check `if(as[n]<=0)` is *supposed* to be checking if it had already cached this value of computation. But if the formula can be negative this function recalculates this cached value alot...

What it really probably wanted was a `vector<pair<bool, unsigned> >`, where the bool says if the value has been calculated or not.

Share Improve this answer Follow

answered Sep 29, 2008 at 3:30

Greg Rogers
36.4k ● 17 ● 69 ● 94

0

The code, as posted, only memoizes about 40% of the time (precisely when the remembered value is positive). As Chris Jester-Young pointed out, a correct implementation would instead check `if(as[n]==0)`. Alternatively, one can change the memoization code itself to read `as[n]=mod(-4*a(n-1)-4*a(n-2),65535);`

(Even the `==0` check would spend effort when the memoized value was 0. Luckily, in your case, this never happens!)

Share Improve this answer Follow

answered Sep 29, 2008 at 3:38

A. Rex
32k ● 21 ● 94 ● 97

0

There's a bug in this code. It will continue to recalculate the values of as[n] for as[n] <= 0. It will memoize the values of a that turn out to be positive. It works a lot faster than code without the memoization because there are enough positive values of as[] so that the recursion is terminated quickly. You could improve this by using a value of greater than 65535 as a sentinal. The new values of the vector are initialized to zero when the vector expands.

Share Improve this answer Follow

answered Sep 29, 2008 at 4:20

David Nehme
21.6k ● 8 ● 81 ● 121