Do My Coding

Do my Coding

Sometimes it is required to compare the speed of the execution of several algorithms.Often, for this purpose use code profilizers.But sometimes it happens that the profiler is not at hand, or work with it is complex, and you just need to compare the speed of the fragment of the code A at the speed of the fragment of the code b in a small test program.

Program template

If you fell into such a situation, then here is the advice of the type "hostess".

#include <iostream> #include <CTime> int Main () {clock_t the_time;Double elapsed_time;the_time = clock ();// Tileable code elapsed_time = double (clock () - the_time) / clocks_per_sec;STD :: COUT << "ELAPSED TIME" << ELAPSED_TIME << "SEC."<< Std :: Endl;Return 0;}

This "fish" allows you to find out the execution time of the code fragment.

Turning on the <iostream> header is necessary to display messages to the standard output (i.e. on the screen).

Turning on the <CTIME> header allows you to use the clock () function and related types and constants.What we actually need to be taming.

The only feature that we will use to tap has the following signature:

clock_t clock (void);

The function returns the time that has passed since the start of the process (i.e., actually since the start of this application) in ticks timer.Type of return value - Clock_t is usually a pseudonym type LONG (defined in <CTIME>).If the function cannot receive the time system from the beginning of the process, the function returns the value -1 given to the type clock_t.(With such an error I never met, so I will omit the check of the return value to -1.)

To convert Tikov Timer in seconds, the CLOCKS_PER_SEC constant is also defined in <CTIME>.

The work algorithm is transparent: before performing the timed code, we interfere with the time after performing the timed code once again we interconverse the time, their difference will give us the desired time for which the code was performed.It remains only to translate it from Tikov per second.

In the program template, I would like to draw attention to the last step.Since clock_t is an integer

type, and clocks_per_sec is an integer constant, then at least one of the parts of the expression should be obtained to obtain fractional parts of seconds.Several Timeing Tips

Between the first and second CLOCK () calls should not be:

1) I / O operations.If, of course, you do not want to test the speed of your hard drive or console convection subsystem.

2) Data request operations for the user (for example, from the keyboard).

It may turn out that the processor used in your computer is too fast to tap a small code fragment.In this case, the difference between the values between the CLOCK () calls will be 0.

The output, as usual, is simple: conclude a timed fragment into the cycle and turn it out there 1'000 or 1'000'000 times.In this case, an additional restriction is superimposed:

3) The timed code should not have side effects.For example, change the values of variables external with respect to this code fragment, or take memory from the heap.

If there is a side effect, then before the next iteration, it must be somehow compensated.This, of course, affects the accuracy of taming, but usually insignificantly.And if the two compared algorithms have the same side effects, this error can be neglected at all.Example

The example below compares two algorithms that perform the same task: the generation of squares of numbers from 0 to 999 and the search among these values of a certain number.In the first case, the vector container is used for storing values, in the second - the MAP container.(An example is a pure demonstration. It is clear that in the real program there will be such nonsense to write anyone!)

note that

The timorable code is too simple for my processor, so the cycle is used for it;

The timed code has a side effect that is compensated by calling the Clear () method for the container.

For the result of a call of the second function, a variable is organized, which is displayed on the screen after tamping.It was done specifically that a smart optimizing compiler would find that the result of the function of the function is not used, did not comply with the name of the function to the protects of the dog.

#include <iostream> #include <vector> #include <vector> #include <map> TypeDef Std :: Vector <Long> VECL; TypeDef Std :: Map <INT, LONG> MAPIL; Const int max_elem = 1000; // Number of squares of numbers const Long Pattern = 999 * 999; // This value is searched among the squares of the CONST INT MAX_TIMES = 1000 numbers; // Number of cycles when taming Void

Generator_Vector (VECL VEC) {for (int i = 0; i <max_elem; i ++) {vec.push_back (I * i); }} BOOL FIND_Vector (Const VECL VEC, LONG NUM) {for (Auto it = vec.begin (); it! = vec.end (); it ++) {if (* it == num) Return True; } Return False; } void generator_map (Mapil MP) {for (int i = 0; i <max_elem; i ++) {mp [i] = i * i; }} BOOL FIND_MAP (Const Mapil MP, Long Num) {for (auto it = mp.begin (); it! = mp.end (); IT ++) {if-> second == num) Return True; } Return False; } int Main () {VECL V; Mapil M; clock_t the_time; BOOL RES; Double elapsed_time; STD :: COUT << "Vector version" << std :: endl; the_time = clock (); For (int t = 0; t <max_times; t ++) {v.clear (); Generator_Vector (V); Res = Find_Vector (V, Pattern); } elapsed_time = double (clock () - the_time) / clocks_per_sec; STD :: COUT << "ELAPSED TIME" << ELAPSED_TIME << "SEC." << Std :: Endl; STD :: COUT << "RESULT =" << RES << "\ N" << STD :: ENDL; STD :: COUT << "MAP VERSION" << STD :: ENDL; the_time = clock (); for (int t = 0; t <max_times; t ++) {M.Clear (); generator_map (m); res = find_map (M, Pattern); } elapsed_time = double (clock () - the_time) / clocks_per_sec; STD :: COUT << "ELAPSED TIME" << ELAPSED_TIME << "SEC." << Std :: Endl; STD :: COUT << "RESULT =" << RES << "\ N" << STD :: ENDL; Return 0; }

Conclusion

The above-described method for testing the speed of the execution of the code fragment is only suitable in the simplest cases.For more difficult cases, it is necessary to use a profiler.

I hope that someone's article will be useful.