Информатика

Самойленко Денис Николаевич
Кандидат физико-математичексих наук, заместитель директора по учебно-методической работе Николаевского филиала частного высшего учебного заведения «Европейский университет» (Украина).

Магический квадрат снова удивляет

Любая задача имеет несколько способов решения. Чем интереснее задача, тем интереснее искать новые экзотические способы её решения. Но на определённом этапе возникает необходимость сделать выбор из множества решений одного – лучшего. Обычно эта проблема проявляется в момент перехода к составлению программы для решения задачи. Причём красота или необычность решения отходит на задний план, а ведущую роль начинает играть скорость получения ответа.

Чтобы сравнивать различные решения, а точнее, алгоритмы решений, нужно приписать алгоритмам определённые числа-характеристики. Разработкой таких характеристик занимается теория сложности алгоритмов.

Очевидно, читатель уже догадался, что основной характеристикой алгоритма выступает его сложность. Однако в зависимости от области применения алгоритма выделяют разные виды сложности:

- сложность описания (по сути, длина программы);
- функциональная сложность (количество функциональных инструкций);
- конструктивная сложность (объединяющая технические требования к исполнителю алгоритма);
- энергетическая сложность (количество энергии, потраченной на выполнение);
- экологическая сложность (мера загрязнения среды, ведь при нагреве процессора и проводов выделяются различные вредные вещества) и так далее.

Наличие такого перечня характеристик делает задачу сравнения алгоритмов довольно непростой. Может оказаться, что один алгоритм проще по функциональности, но требует бóльших затрат энергии, чем другой алгоритм. Предоставить единый критерий отбора невозможно – всё зависит от области применения алгоритма. В одних случаях решающую роль играет время выполнения, в других – экономия энергии.

Задача немного упрощается, когда мы говорим о программных реализациях алгоритмов на персональных компьютерах (ПК). Современные ПК имеют более-менее одинаковые наборы команд, время исполнения которых на

разных компьютерах также мало отличается друг от друга. Они затрачивают за одинаковое время примерно одинаковую энергию и выделяют одинаковое количество вредных веществ. На первое место выходит именно время выполнения программы, все остальные характеристики находятся в некоторой зависимости от этого времени.

Итак, мы хотим определить время выполнения программ, решающих одну и ту же задачу, и выбрать ту программу, которая делает это быстрее всех. Это относительно несложно сделать, если говорить об определённой конкретной задаче с известным условием. Сложнее, если мы хотим выбрать решение, которое будет наилучшим для любых входных данных задачи. Ведь одни алгоритмы быстрее справляются с малыми объёмами данных, а другие – с большими. Можно проводить измерения при различных условиях и составлять таблицы результатов, но это позапрошлый век!

Попробуем поступить по-другому. Решим дополнительную задачу – выявим зависимость времени выполнения программы от так называемой размерности входа – количественной характеристики объёма данных, подлежащих обработке. Имея такие зависимости, достаточно провести одно измерение времени для данного ПК и конкретного условия, а для всех остальных условий воспользоваться полученными зависимостями.

Попробуем построить указанную зависимость для следующей задачи. Выберем в качестве объекта исследования алгоритм, проверяющий, является ли двумерный массив магическим квадратом.

Напомним, что такое магический квадрат. Магический, или волшебный квадрат – это квадратная таблица $N \times N$, заполненная числами таким образом, что сумма чисел в каждой строке, каждом столбце и на обеих диагоналях одинакова. Если в квадрате равны суммы чисел только в строках и столбцах, то он называется полумагическим.

Составить программу, проверяющую массив на «магичность», – задача несложная. Приведём фрагмент программы на C++, проверяющий массив **kv** на «полумагичность». В программе используется дополнительный массив **sum**.

```
const N = 10;
double *kv,*sum;
int i,j;
kv=new double [N*N];
sum=new double [N+N];
 /* блок заполнения массива здесь пропущен */
  for(i=0;i<N;i++)
  {sum[i]=0;
   for(j=0;j<N;j++)
     sum[i]+=kv[i*N+j];
```

```
 }
for(i=0;i<N;i++)
{sum[i+N]=0;
  for(j=0;j<N;j++)
 sum[i+N]+=kv[j*N+i];
}

i=0;
do  i++; while(sum[0]==sum[i] && i<N);
if(i==N){/* магический квадрат */}
```

Уточним задачу. Необходимо исследовать зависимость времени работы составленной программы от размера массива, используемого для хранения матрицы. Размер матрицы, напомним, равен $N \times N$.

Замечание. Раскроем небольшой секрет – задача выбрана не случайно. Естественно, ожидается достижение определённого эффекта. Ведь при увеличении N в два раза размерность массива-матрицы увеличится в $2 \times 2 = 4$ раза. Поскольку обрабатываются все элементы матрицы, время обработки должно быть пропорционально $N \times N = N^2$. Ожидается квадратичная зависимость времени от N.

Для заполнения массивов используем случайные числа. Конечно, получить магический квадрат из случайных чисел практически невозможно, но задача не столько в поиске таких квадратов, сколько в изучении времени выполнения программы.

Блок измерения времени построим по рекомендованной документацией Borland схеме:

```
start=clock();
измеряемый блок;
end=clock();
расчёт времени.
```

Поскольку для измерения будут использоваться системные часы-таймер, опрашиваемые функцией **clock()**, предельная точность измерений (без перепрограммирования таймера) будет составлять 1/18 сек. А это значит, что быстро выполняющиеся фрагменты либо не поддадутся измерениям, либо будут измеряться с большой погрешностью. Современные компьютеры за 1/18 сек. обработают массив с размерностью N около 1000.

Для возможности проведения измерений с небольшими N используем приём искусственного увеличения времени работы программы путём её многократного повтора. Количество повторов следует подбирать для каждого компьютера отдельно в зависимости от его быстродействия. Поскольку количество повторов одинаково для всех N, зависимость искажена не будет.

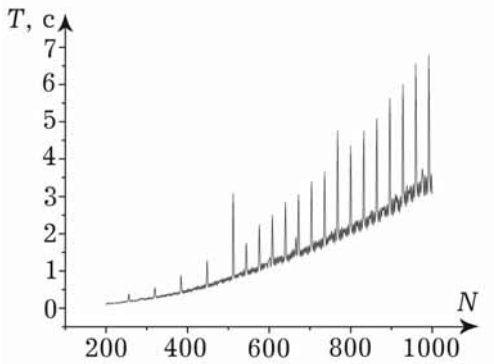
Для возможности работы с $N > 100$ программу необходимо компилировать как проект для операционной системы Windows. Под управлением DOS для программы выделяется не больше 1 Мб памяти, и размещение в памяти массивов с такими размерами невозможно.

Итоговую программу создадим в виде двух функций. Основная функция (**main**) обеспечивает циклический запуск измеряющей время функции (**T**) и запись результатов в файл для последующей обработки внешними программами. Измеряющая функция выполняет заполнение массива случайными числами и проверяет его на «магический» признак 50 раз. Процесс выполнения программы сопровождается выводом на экран точек после каждого измерения (**putch('.');**). Это позволяет следить за ходом работы, который длится около 10 минут. Результаты сохраняются в файле data.dat.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>
#include <time.h>

double T(int N)
{ double *kv,*sum,ret;
  int i,j;
  clock_t start,end;
  kv=new double [N*N];
  sum=new double [N+N];

  randomize();
  for(i=0;i<N;i++)
   for(j=0;j<N;j++)
         kv[i*N+j]=rand()% N*N;

  start=clock();
  for(int c=0;c<50;c++)
  {
    for(i=0;i<N;i++)
    {sum[i]=0;
     for(j=0;j<N;j++)
        sum[i]+=kv[i*N+j];
    }
   for(i=0;i<N;i++)
   {sum[i+N]=0;
     for(j=0;j<N;j++)
    sum[i+N]+=kv[j*N+i];
   }

   i=0;
   do  i++; while(sum[0]==sum[i] && i<N);
   if(i==N){/*реакция на магичность*/}
  }
  end=clock();
  ret=(double)(end - start) / CLK_TCK;
  delete[]sum; delete[]kv;
  putch('.');
  return ret;
}

int main()
{
 FILE *f;
 f = fopen("data.dat","wt");
 if(!f){printf("File operation error");return getch();}

 int N;
 for(N=200;N<=1000;N+=2)
```

```
    fprintf(f,"%d %.3f\n",N,T(N));

  fclose(f);
  printf("Done.\nPress a key\n");
  return getch();
}
```

По окончании работы программы файл data.dat был обработан в пакете Origin, с помощью которого был построен график зависимости времени работы программы от размерности массива.



Как видно из рисунка, ожидаемая квадратичная зависимость (график квадратичной функции имеет вид параболы) образовала лишь общий фон, на котором отчётливо видны всплески времени работы программы, приводящие к задержкам выполнения в 1,5–3 раза.

Самое простое объяснение обнаруженных отклонений – выполнение фоновых служб многозадачной системы Windows. Но это объяснение элементарно опровергается перезапуском программы. При всех повторениях положение экстремумов на графике не изменяется, в то время как моменты запуска фоновых служб относительно случайны. Можно провести измерения лишь вблизи отдельных экстремумов и убедиться, что они повторяются и в этом случае.

Следует также отметить, что в поисках объяснения полученных результатов было проведено множество экспериментов на разных ПК, под управлением разных систем, с использованием различных языков программирования, для различных матричных алгоритмов. Во всех случаях наблюдался эффект возрастания времени с периодическими всплесками. Различия проявлялись в величине пика, плотности пиков. Общее положение пиков всегда оставалось неизменным, хотя при разной плотности пропадали промежуточные пики.

Что вызывает особое удивление, так это восстановление времени выполнения после прохождения пика. То есть если при размерности $N=512$ время выполнения около 3 секунд, то при $N=513$ – около одной. Иными словами, квадрат 513×513 проверяется примерно в 3 раза быстрее, чем квадрат 512×512.

Существуют две составляющие в объяснении описанного эффекта. Одна связана с принципом работы микросхем памяти, вторая – с архитектурой памяти ПК. Детальное объяснение потребует от читателя знакомства с электроникой и микросхемотехникой.

Что же следует из описанного? Ожидаемая зависимость, логически предполагаемая из анализа задачи, не подтвердилась на практике. Существуют особенности строения ПК, которые могут проявиться возрастанием времени работы программы при «неудачном» выборе количества операций. Принципиальность эффекта возрастания времени при работе с массивами определённой размерности позволяет дать совет программистам быть предусмотрительными при разработке подобных программ. Вместо использования массива выбранной размерности, возможно, следует использовать больший на единицу массив и получить выигрыш во времени до 3-х раз!