# Анализ алгоритмов Семинар 2

## Анализ трудоёмкости

Модель оценки трудоёмкости (модели вычислений):

1. Трудоёмкость базовой операции
   1. Пусть трудоёмкость следующих операций единична: +, -, =, +=, -=, ==, !=, <, >, -=, >=, <<, >>, []  
      Пусть трудоёмкость \*, /, \, %, /=, \*= - 2
   2. Можем ввести некоторые коэффициенты от базовой величины, например: ,
2. Трудоёмкость цикла  
   for (I = 0; i<n; i++)  
   {  
    /\*Тело цикла\*/ ()  
   }  
   пример for(k=0; k<N/2; k++)
3. Трудоёмкость условного оператора  
   if [условие] ()  
   {  
    [Блок1] ()  
   }  
   else  
   {  
    [Блок2]()  
   }  
   Пусть трудоёмкость условного перехода 0  
   Пример  
   if (N%2==1) (2) (1) =>

## Алгоритм умножения матриц

### Стандартный

for (i=0; i<M; i++0)  
 for (j=0; j<Q; j++)  
 for (k=0; k<N; k++)  
 C[i][j]=([i][j]+A[i][k]B[k][j])

Используем операцию += в последней строке

Доля умножений в итоговой оценке – 1\*MNQ   
Можно ли эту величину снизить?  
При итоговой оценке трудоёмкости обобщённой оценки даётся по наиболее быстро растущему слогаемому. Здесь это MNQ, поэтому говорят что кубический характер функции трудоёмкости  
/\* куб от линейного размера матриц \*/  
Обозначим знаком “<” как “растёт медленнее чем”.

### Алгоритм Винограда

Алгоритм Винограда умножения матриц

Пусть – это i-я строчка A

*– это j-й столбец B =>*

Можно использовать повторно

**Часть 1)** Формирование массива MulH – сумма произведений пар соседних элементов строк матрицы A  
for (i=0; i<M; i++)  
 for (k=0; k<N/2;k++)  
 MulH[i]=MulH[i]+A[i][k\*2]\*[i][k\*2+1]  
/\* значение MulH[i] будет повторно использовано при умножении i-й строки матрицы A на все столбцы матрицы B \*/

= [] \* +

Оптимизируем дальше:

1. +=
2. for : k<N, k+=2

+= [] \* +

**Часть 2)** Заполнение MulV аналогично

**Часть 3)** Заполнение ячеек матрицы C  
for (i=0; i<M; i++)  
 for (j=0; j<Q; j++)  
 C[i][j]=-MulH[i]-MulV[j]  
 for (k=0; k<N/2; k++)  
 C[i][j]=C[i][j]+(A[i][2\*k]+B[2\*k+1][j])(A[i][2\*k+1]+B[2\*k][j])

= [] + \*

Задействуем оптимизацию N-1

В ничего не изменится ровно, как и в , зато в экономия MQ

Задействуем третий перечень оптимизаций

+= [] \* +-

Из них умножения вдвое реже, чем в стандартном методе

И ещё:  
for  
{  
 Buf = MulH[i]+MulV[j];  
 for  
 Buf += …  
 C[i][j]=Buf;  
}  
 трата + MQ операций, экономия MNQ => ( vs 11MNQ)

**Часть 4)** Если (n%2 == 1), тогда   
for (i=0; i<M; i++)  
 for(j=0; j<Q; j++)  
 C[i][j]+=A[i][N-1]\*B[N-1][j]

Улучшим худший случай (за счёт ухудшения лучшего случая)

Задействуем флаг; сольём **3** и **4 =>**=> x.c.: долой л.с.: дополнительная работа   
Итоговую оценку считаем как

ЛР2:

* Стандартный метод
* Алгоритм Винограда
* И алгоритм Вин. оптимизированный нами (не менее чем 3-мя типами оптимизации)

Посчитать (Трудоёмкость) и замерить

* Лучший случай ()  
   ()  
   …  
   (0)
* Худший случай ()  
   ()  
   …  
   ()

ЛР3 Трудоёмкость из источника для одного

Для двух рассчитать

Рисунок вставить  
for i:=1 to N-1 do  
 for j:=1 to N-I do  
 if(a[j]>a[j+1]) then  
 begin  
 buf := a[j];  
 a[j] := a[j+1];  
 a[j+1] := buf;  
 end;

Гипотеза

Т.к. переменный предел

i++ i<N-i