```
!apt-get --purge remove cuda nvidia* libnvidia-*
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
!apt-get remove cuda-*
!apt autoremove
!apt-get update


!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2



!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
```

```
    Collecting git+git://github.com/andreinechaev/nvcc4jupyter.git
      Cloning git://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-bui
      Running command git clone -q git://github.com/andreinechaev/nvcc4jupyter.gi
    Building wheels for collected packages: NVCCPlugin
      Building wheel for NVCCPlugin (setup.py) ... done
      Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl si
      Stored in directory: /tmp/pip-ephem-wheel-cache-tsrx91gz/wheels/c5/2b/c0/87
    Successfully built NVCCPlugin
    Installing collected packages: NVCCPlugin
    Successfully installed NVCCPlugin-0.0.2
```

```
%load_ext nvcc_plugin
```

```
    created output directory at /content/src
    Out bin /content/result.out
```

```
%%cu
#include <stdio.h>
#include <iostream>
#include <cuda.h>
#include <math.h>
#include <chrono>
#include <random>
#include <bits/stdc++.h>
using namespace std;

int random_in_range( int minimum, int maximum )
{
  thread_local std::ranlux48 rng(
    std::chrono::system_clock::now().time_since_epoch().count() );
  return std::uniform_int_distribution <int> ( minimum, maximum )( rng );
}

__global__ void add(int *i, int *j, int *k) {
    int bid = blockIdx.x;
    k[bid] = i[bid] + j[bid];
}
```

```
}

void random_init(int *arr, int n) {
    for(int i = 0 ; i < n ; i++) {
        arr[i] = random_in_range(100,400);
    }
}

void add_cpu(int *i, int *j, int *k, int n) {
    for(int p = 0 ; p < n ; p++) {
        k[p] = i[p] + j[p];
    }
}

int main() {
    int n = 20000;
    int *a, *b;
    int c[n];
    int *i, *j, *k;
    int size = n * sizeof(int);

    a = new int[n];
    b = new int[n];
    random_init(a,n);
    random_init(b,n);

    cout<<"First: ";
    for(int i = 0 ; i < n ; i++) {
        cout<<a[i]<<" ";
    }
    cout<<endl;

    cout<<"Second: ";
    for(int i = 0 ; i < n ; i++) {
        cout<<b[i]<<" ";
    }
    cout<<endl;

    cudaMalloc((void **)&i,size);
    cudaMalloc((void **)&j,size);
    cudaMalloc((void **)&k,size);

    cudaMemcpy(i,a,size,cudaMemcpyHostToDevice);
    cudaMemcpy(j,b,size,cudaMemcpyHostToDevice);

    float gpu_elapsed_time;
    cudaEvent_t gpu_start,gpu_stop;
    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start,0);
    add<<<n,1>>>(i,j,k);
    cudaEventRecord(gpu_stop, 0);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);
```

```
    cudaEventDestroy(gpu_stop);

    cudaMemcpy(c,k,size,cudaMemcpyDeviceToHost);

    cout<<endl;
    cout<<"GPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds";
    cout<<endl;
    cout<<"Parallel Result: ";
    for(int i = 0 ; i < n ; i++) {
        cout<<c[i]<<" ";
    }
    cout<<endl;
    cout<<endl;

    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start,0);
    add_cpu(a,b,c,n);
    cudaEventRecord(gpu_stop, 0);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);

    cout<<"CPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds";
    cout<<endl;

    cout<<"Serial Result: ";
    for(int i = 0 ; i < n ; i++) {
        cout<<c[i]<<" ";
    }
    cout<<endl;

    cudaFree(i);
    cudaFree(j);
    cudaFree(k);

    return 0;
}
```

```
First: 138 136 186 175 201 129 164 197 242 294 212 335 150 304 127 297 125 15
Second: 397 111 359 309 200 103 150 321 206 160 400 341 314 132 263 239 309 3

GPU Elapsed time is: 0.204768 milliseconds
Parallel Result: 535 247 545 484 401 232 314 518 448 454 612 676 464 436 390

CPU Elapsed time is: 0.086304 milliseconds
Serial Result: 535 247 545 484 401 232 314 518 448 454 612 676 464 436 390 53
```

```
%%cu
#include <stdio.h>
#include <iostream>
#include <cuda.h>
#include <chrono>
```

```
#include <random>
using namespace std;


int random_in_range( int minimum, int maximum )
{
  thread_local std::ranlux48 rng(
    std::chrono::system_clock::now().time_since_epoch().count() );
  return std::uniform_int_distribution <int> ( minimum, maximum )( rng );
}

__global__
void matrixVector(int *vec, int *mat, int *result, int n, int m)
{
    int tid = blockIdx.x*blockDim.x + threadIdx.x;
    int sum=0;

    if(tid <= n) {
        for(int i=0; i<n; i++) {
            sum += vec[i]*mat[(i*m) + tid];
        }
        result[tid] = sum;
    }
}

void maxtrixVector_cpu(int *vec, int *mat, int *result, int n, int m) {
    for(int i = 0 ; i < n ; i++) {
        long sum = 0;
        for(int j = 0 ; j < m ; j++) {
            sum = sum + mat[j*m+i] * vec[j];
        }
        result[i] = sum;
    }
}

void init_array(int *a, int n) {
    for(int i=0; i<n; i++)
      a[i] = random_in_range(10,40);
}

void init_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++) {
            a[i*m + j] = random_in_range(10, 40);
        }
    }
}

void print_array(int *a, int n) {
    for(int i=0; i<n; i++) {
        cout<<a[i]<<" ";
    }
    cout<<endl;
}
```

```
void print_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
        for(int j=0; j<m; j++)
          cout<<"  "<<a[i*m + j];
        cout<<endl;
    }
}

int main() {
    int *a, *b, *c;
    int *a_dev, *b_dev, *c_dev;

    int n = 100;
    int m = 100;

    a = new int[n];
    b = new int[n*m];
    c = new int[m];

    init_array(a, n);
    init_matrix(b, n, m);

    cout<<"Initial vector array : "<<endl;
    print_array(a, n);
    cout<<endl;
    cout<<"Initial matrix : "<<endl;
    print_matrix(b, n, m);
    cout<<endl;

    cudaMalloc(&a_dev, sizeof(int)*n);
    cudaMalloc(&b_dev, sizeof(int)*n*m);
    cudaMalloc(&c_dev, sizeof(int)*m);

    cudaMemcpy(a_dev, a, sizeof(int)*n, cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev, b, sizeof(int)*n*m, cudaMemcpyHostToDevice);

    float gpu_elapsed_time;
    cudaEvent_t gpu_start,gpu_stop;
    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start,0);
    matrixVector<<<m, 1>>>(a_dev, b_dev, c_dev, n, m);
    cudaEventRecord(gpu_stop, 0);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);
    cout<<"GPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

    cudaMemcpy(c, c_dev, sizeof(int)*m, cudaMemcpyDeviceToHost);

    cout<<"GPU Resultant vector : ";
    print_array(c, m);
    cout<<endl;
```

```
    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start,0);
    maxtrixVector_cpu(a, b, c, n, m);
    cudaEventRecord(gpu_stop, 0);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);
    cout<<"CPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

    cout<<"CPU Resultant vector : ";
    for(int i = 0 ; i < n ; i++) {
        cout<<c[i]<<" ";
    }
    cout<<endl;

    cudaFree(a_dev);
    cudaFree(b_dev);
    cudaFree(c_dev);

    delete[] a;
    delete[] b;
    delete[] c;

    return 0;
}
```

```
 Initial vector array :
 16 11 13 20 10 33 13 15 10 39 28 30 19 14 35 22 21 13 10 21 27 14 21 27 30 1

 Initial matrix :
   34  38  35  24  40  19  23  24  23  26  19  21  17  11  23  16  26  31  1
   34  26  10  38  27  11  10  35  36  19  13  17  31  27  36  39  11  21  2
   19  22  16  15  38  30  24  10  39  22  32  25  32  16  27  20  37  25  2
   12  23  32  12  12  17  31  23  37  21  11  33  28  24  22  19  39  25  1
   23  26  23  22  19  30  15  13  20  35  21  10  10  40  17  11  30  32  1
   14  23  23  33  27  37  19  13  34  22  34  23  11  34  21  40  10  11  1
   11  27  10  39  16  11  18  35  30  20  11  33  20  10  19  27  20  29  1
   37  33  35  29  39  28  23  13  14  39  30  29  16  38  22  22  25  26  2
   30  30  40  10  23  17  32  10  29  20  33  36  26  26  13  18  27  26  2
   39  17  33  31  15  33  19  15  28  35  10  28  25  10  15  31  29  13  3
   27  31  33  15  30  28  36  11  31  17  19  37  12  25  33  25  34  28  4
   26  24  38  34  29  28  10  27  27  39  28  36  16  23  17  11  11  40  3
   22  10  26  31  24  24  12  21  33  25  17  12  10  14  13  28  14  19  3
   15  30  29  10  11  20  38  34  38  36  22  25  20  28  39  36  11  36  1
   12  27  39  24  15  36  12  20  30  33  18  21  24  39  32  24  20  40  1
   31  37  18  21  38  12  38  35  14  32  20  37  24  16  28  40  13  16  3
   17  19  39  38  12  30  26  37  28  11  20  26  39  20  21  18  28  19  3
   25  26  14  27  12  10  15  20  31  28  34  15  16  12  40  22  15  15  1
   28  15  15  37  12  27  33  14  25  30  36  12  21  33  25  36  20  30  2
   29  34  22  16  30  34  26  17  28  40  11  33  28  10  37  28  38  15  1
   24  16  19  27  13  38  22  23  28  21  19  32  28  20  13  20  11  17  1
   29  30  28  30  38  24  19  14  30  17  23  26  33  39  27  17  39  37  2
   38  23  37  29  32  25  35  30  18  12  25  38  31  39  14  29  15  29  4
   32  16  27  30  32  13  16  14  32  10  31  32  26  25  28  21  35  14  2
   19  30  20  20  36  35  22  20  36  23  10  39  11  31  33  39  32  13  1
   27  38  10  19  12  28  26  19  23  23  12  22  30  33  31  20  15  30  1
```

38	24	22	23	31	16	18	15	33	22	33	38	35	40	30	12	29	29	1
13	36	21	30	23	40	27	16	28	27	25	19	31	34	31	23	40	14	1
32	14	16	24	30	38	35	38	12	10	33	10	21	40	40	33	39	18	4
18	23	11	15	24	36	31	40	21	22	34	39	31	21	22	13	31	18	2
17	19	38	37	38	37	10	19	17	13	35	24	36	25	34	24	22	40	2
30	39	39	38	30	36	22	27	10	11	13	10	25	17	17	26	26	12	1
20	25	20	34	30	22	20	22	29	20	32	29	36	32	39	25	20	28	3
10	13	26	32	11	32	12	27	25	16	33	32	35	25	30	26	36	20	1
38	33	10	12	23	38	39	37	19	23	39	11	30	25	31	25	27	31	3
24	37	32	14	16	18	35	24	35	35	11	17	23	21	19	22	21	20	2
36	22	40	32	30	14	34	32	35	27	28	33	11	36	10	26	14	23	2
10	10	10	25	17	36	32	35	34	31	39	37	28	16	14	28	18	25	2
28	15	17	19	40	25	24	14	18	35	18	33	39	20	26	39	18	34	3
23	21	23	29	32	13	24	11	26	23	27	15	10	27	22	31	30	19	1
19	28	16	16	30	10	35	10	32	38	40	24	16	23	18	32	31	29	3
32	31	17	12	11	27	23	19	20	39	36	26	32	22	32	30	34	11	3
21	37	12	19	20	37	12	23	11	23	18	28	20	18	32	22	24	33	2
20	35	16	16	10	36	34	23	32	40	11	35	26	31	18	21	32	33	1
11	39	20	25	33	35	23	22	40	14	24	35	19	17	35	40	10	30	2
38	17	19	24	34	33	21	34	21	17	13	26	14	14	24	37	26	39	3
36	31	18	40	37	38	13	30	30	11	17	31	18	34	23	11	15	35	2
34	22	28	37	11	24	38	13	39	35	18	17	11	32	17	28	22	36	3
18	39	25	15	35	26	15	16	12	34	11	27	13	31	33	21	31	38	1
33	32	14	29	37	31	12	15	17	29	19	18	26	17	11	40	20	40	3
15	36	34	19	33	21	18	22	30	40	40	25	10	15	33	24	24	33	2
19	24	33	22	32	32	30	29	23	20	28	11	23	34	16	25	31	36	4
37	24	11	11	21	20	18	24	29	36	29	30	28	13	22	10	11	26	3
26	29	40	24	21	16	35	19	13	23	10	24	38	26	11	22	22	10	3

```
%%cu

#include <stdio.h>
#include <iostream>
#include <cuda.h>
#include <chrono>
#include <random>
using namespace std;


int random_in_range( int minimum, int maximum )
{
  thread_local std::ranlux48 rng(
    std::chrono::system_clock::now().time_since_epoch().count() );
  return std::uniform_int_distribution <int> ( minimum, maximum )( rng );
}

__global__
void matrixMultiplication(int *a, int *b, int *c, int m, int n, int k)
{
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;
    int sum=0;

    if(col<k && row<m) {
      for(int j=0;j<n;j++)
      {
          sum += a[row*n+j] * b[j*k+col];
```

```
      }
      c[k*row+col]=sum;
    }

}

void matrix_multiplication_cpu(int *a, int *b, int *c, int m, int n, int k) {
    for(int i = 0 ; i < m ; i++) {
        for(int j = 0 ; j < n ; j++) {
            long result = 0;
            for(int p = 0 ; p < k ; p++) {
                result=result+a[i*k+p]*b[p*k+j];
            }
            c[k*i+j] = result;
        }
    }
}

void init_result(int *a, int m, int k) {
    for(int i=0; i<m; i++) {
      for(int j=0; j<k; j++) {
        a[i*k + j] = 0;
      }
    }
}

void init_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
      for(int j=0; j<m; j++) {
        a[i*m + j] = random_in_range(10,30);
      }
    }
}

void print_matrix(int *a, int n, int m) {
    for(int i=0; i<n; i++) {
      for(int j=0; j<m; j++) {
        cout<<"  "<<a[i*m + j];
      }
      cout<<endl;
    }
    cout<<endl;
}

int main()
{

    int *a,*b,*c;
    int *a_dev,*b_dev,*c_dev;
    int m=30, n=30, k=30;

    a = new int[m*n];
    b = new int[n*k];
    c = new int[m*k];
```

```
    init_matrix(a, m, n);
    init_matrix(b, n ,k);
    init_result(c, m, k);

    cout<<"First matrix : "<<endl;
    print_matrix(a, m, n);
    cout<<"Second matrix : "<<endl;
    print_matrix(b, n, k);

    cudaMalloc(&a_dev, sizeof(int)*m*n);
    cudaMalloc(&b_dev, sizeof(int)*n*k);
    cudaMalloc(&c_dev, sizeof(int)*m*k);

    cudaMemcpy(a_dev, a, sizeof(int)*m*n, cudaMemcpyHostToDevice);
    cudaMemcpy(b_dev, b, sizeof(int)*n*k, cudaMemcpyHostToDevice);

    dim3 dimGrid(1,1);
    dim3 dimBlock(n,n);

    float gpu_elapsed_time;
    cudaEvent_t gpu_start,gpu_stop;
    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start,0);
    matrixMultiplication<<<dimGrid, dimBlock>>>(a_dev,b_dev,c_dev, m, n, k);
    cudaEventRecord(gpu_stop, 0);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);
    cout<<"GPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

    cudaMemcpy(c, c_dev, sizeof(int)*m*k, cudaMemcpyDeviceToHost);

    cout<<"GPU Result : "<<endl;
    print_matrix(c, m, k);
    cout<<endl;

    cudaEventCreate(&gpu_start);
    cudaEventCreate(&gpu_stop);
    cudaEventRecord(gpu_start,0);
    matrix_multiplication_cpu(a, b, c, m, n, k);
    cudaEventRecord(gpu_stop, 0);
    cudaEventSynchronize(gpu_stop);
    cudaEventElapsedTime(&gpu_elapsed_time, gpu_start, gpu_stop);
    cudaEventDestroy(gpu_start);
    cudaEventDestroy(gpu_stop);
    cout<<"CPU Elapsed time is: "<<gpu_elapsed_time<<" milliseconds"<<endl;

    cout<<"CPU Result : "<<endl;
    print_matrix(c, m, k);

    cudaFree(a_dev);
    cudaFree(b_dev);
    cudaFree(c_dev);
```

```
    delete[] a;
    delete[] b;
    delete[] c;

    return 0;
}
```

```
 First matrix :
   21  11  19  12  18  20  18  23  13  26  20  19  23  19  17  17  10  14  30
   18  27  15  27  23  29  24  23  21  14  28  12  28  10  16  21  17  18  22
   14  11  24  23  23  18  17  11  16  10  10  10  11  19  27  12  21  29  13
   16  14  30  19  21  27  10  23  21  23  15  25  21  19  24  11  10  30  25
   19  21  24  29  26  13  19  25  19  10  10  11  27  23  26  14  25  17  19
   23  27  16  30  18  27  23  17  23  15  10  21  11  20  21  17  13  29  29
   14  29  27  24  24  15  17  24  17  20  10  30  25  25  12  10  17  17  29
   12  21  26  23  17  14  29  13  24  25  23  24  17  18  21  30  18  21  17
   22  24  14  28  16  24  22  13  17  19  23  16  10  15  30  25  27  22  27
   29  17  24  29  20  29  12  18  14  20  27  27  22  20  19  22  23  30  28
   24  21  16  10  22  10  28  29  24  10  22  28  20  12  27  13  22  15  30
   17  22  18  25  18  23  15  11  28  11  15  22  14  19  20  15  21  18  10
   20  27  12  29  15  20  16  29  11  29  26  18  18  11  22  14  10  25  28
   10  17  16  24  27  22  21  28  24  30  28  25  21  26  22  10  20  10  27
   21  25  11  21  17  28  26  28  25  15  15  24  16  17  16  15  26  27  23
   24  15  20  12  26  28  23  13  25  22  16  13  13  25  26  13  17  30  10
   23  19  27  10  19  19  26  26  11  21  21  26  26  23  25  18  27  11  12
   23  14  28  17  11  23  26  11  26  23  17  30  27  10  21  15  25  18  14
   23  25  29  25  10  20  15  30  15  22  14  21  24  26  15  24  16  19  10
   11  21  23  11  11  23  12  13  30  10  17  15  26  28  10  22  29  16  20
   12  23  30  20  19  11  10  14  26  15  23  28  23  28  24  10  22  26  18
   15  12  21  17  22  26  11  28  18  27  22  18  22  15  12  14  25  24  17
   24  29  23  30  21  17  14  30  19  13  17  17  24  10  22  10  25  29  20
   12  30  16  17  29  26  10  23  21  17  13  21  20  20  16  26  18  13  27
   15  17  27  12  11  12  29  10  30  10  25  15  24  20  21  23  24  29  20
   13  14  25  27  25  23  21  27  19  19  23  11  16  19  20  11  13  25  12
   11  25  29  10  24  15  30  18  16  21  17  18  10  18  19  28  24  17  10
   26  11  18  17  22  17  21  23  30  16  15  28  17  12  23  22  28  18  17
   20  21  19  18  11  13  13  25  26  29  22  13  18  26  10  24  26  14  12
   12  14  25  18  21  25  27  11  16  13  29  22  30  13  21  20  13  15  29

 Second matrix :
   20  20  10  14  17  18  26  22  15  17  16  13  20  28  15  27  24  22  10
   23  20  27  21  13  21  11  15  19  16  10  21  23  29  21  12  17  15  10
   15  22  23  17  25  30  21  28  27  18  10  16  13  18  18  17  25  11  15
   23  15  29  13  14  14  21  28  27  13  15  25  23  16  20  29  24  18  27
   20  26  21  17  16  10  22  22  30  24  24  11  24  24  13  27  11  15  14
   16  22  25  19  12  17  30  15  24  23  29  29  19  16  19  10  13  19  19
   15  11  27  30  15  12  25  26  12  20  26  20  12  26  24  15  17  27  15
   26  27  16  26  25  25  18  19  26  11  18  28  14  26  12  18  14  18  28
   24  21  12  12  14  15  19  10  20  30  17  26  22  12  29  13  13  12  10
   16  19  19  28  12  18  19  11  28  12  24  22  11  18  28  15  18  18  10
   12  14  26  23  25  14  20  20  14  30  21  13  13  20  14  26  14  25  23
   12  28  25  30  13  23  16  19  22  23  16  22  16  11  17  23  21  29  27
   15  28  16  24  30  17  23  10  10  25  24  29  12  17  22  24  13  17  12
   23  17  12  26  15  21  24  24  20  23  20  17  15  21  27  11  22  10  10
   17  28  30  17  19  10  17  19  18  23  26  14  19  19  10  27  21  26  24
   22  12  14  24  25  22  14  23  13  27  29  22  28  26  23  12  20  28  10
   27  29  19  29  19  27  25  21  17  29  26  21  16  10  26  26  28  22  13
   29  29  14  26  19  12  19  21  25  30  14  15  20  15  27  19  16  10  20
   17  25  26  18  26  13  13  15  24  10  19  23  12  22  14  27  20  21  19
```

21	24	29	28	19	19	19	13	20	16	11	19	21	27	17	24	29	28	3
16	12	20	16	10	11	30	28	24	19	13	28	27	27	21	12	16	16	1
29	27	19	26	16	26	21	29	21	20	24	27	25	24	12	16	27	29	2
18	26	21	16	24	24	30	15	30	27	25	13	10	19	19	10	13	20	2
20	30	13	22	16	24	26	28	21	28	22	18	25	28	10	19	18	29	2
24	25	24	26	27	24	24	10	24	14	24	22	19	15	29	25	24	12	2