Trabajo práctico 3:
"Algoritmica"

Normativa

Límite de entrega: Martes 21 de Noviembre, 23:59hs. Enviar el código fuente al mail: `algo2.dc+TP3@gmail.com`

Normas de entrega: Ver "Información sobre la cursada" en el sitio Web de la materia.
(`http://www.dc.uba.ar/materias/aed2/2017/2c/??/cursada`)

Versión: 1.2 del 17 de noviembre de 2017 (ver CHANGELOG.md)

Enunciado

Este Trabajo Práctico consiste en una ejercitación de implementación de técnicas algorítmicas en C++. Está permitido utilizar cualquier clase de la `std` para la resolución de este TP. El código base junto con este enunciado están disponibles tanto en la página de la materia como en el siguiente repositorio GIT: `https://git.exactas.uba.ar/gdperez/aed2-2c2017-tp3`.

Ejercicio 1 - Bucket Sort Genérico

El objetivo de este ejercicio consiste en implementar una rutina de *BucketSort* genérica. Para tal fin se pide implementar las siguentes dos operaciones que, combinadas entre sí, implementan un *BucketSort* tradicional.

```
  template<typename iterator, typename bucket>
vector<bucket> generar_buckets(iterator input_begin, iterator input_end)

template <typename bucket>
vector<typename bucket::value_type> aplanar_buckets(const std::vector<bucket> & B)
```

La función `generar_buckets` recorre un intervalo de valores delimitado por dos iteradores genéricos y los agrupa en buckets que serán contenedores del tipo paramétrico `bucket`. Finalmente, retorna un vector con los buckets generados. La función `aplanar_buckets` colapsa un vector de buckets que almacenan valores en un vector de valores.

Se puede asumir lo siguiente sobre los tipos involucrados.

- `iterator` es compatible con la convención de `std` de `forward_iterator`.
- `iterator::value_type` (el tipo de los elementos a ordenar) tiene al menos definidos `operator==`, `operator<` y `operator int()`. Esto último significa que todo `value_type` a usar tiene definida una conversión a `int`: si tengo un valor `v` de tipo `value_type`, la expresión `int(v)` devolverá algún valor entero que dependerá del tipo[1].
- `bucket` es un contenedor que al menos tiene las operaciones `begin()`, `end()`, `count(const value_type & v)` e `insert(iterator pos, const value_type & v)`

Ejercicio 2 - Clasificación de billetes

Desde el Banco Central de la Nación nos contactaron para resolverles eficientemente un problema que tienen a menudo: quieren detectar si un fajo de billetes entregado por un cliente puede tener billetes falsos.

Se sabe que los billetes se numeran consecutivamente y que por año se imprimen m billetes. Además, el banco tiene un listado de tamaño n de todos los billetes falsos hasta el día de la fecha. No todos los billetes que existen son falsos pero en el listado del banco hay al menos un billete falso por año.

Se pide:
Dado un fajo (arreglo) de billetes de tamaño p, se lo quiere devolver ordenado de mayor a menor según la probabilidad de que el billete sea falso o no. La probabilidad de que un billete emitido en el año a sea falso, es la cantidad de billetes falsos que el banco tiene en su listado para el año a. Los billete falsos, es decir, aquellos que figuran en el listado del banco, tienen la misma probabilidad entre ellos y mayor a todos los no falsos. En caso de misma probabilidad, los billetes deben ordenarse decrecientemente según su número de serie.

[1] Notar que los tipos numéricos básicos de C++ son convertibles a `int`

Resolver el problema en: $O(n\ log(m) + p\ log(m \cdot p))$.

Esto es, implemente la operación

```
inline fajo ordenar_por_probabilidad(const fajo& falsos_conocidos,
const fajo & a_ordenar);
```

donde `fajo` es un vector de billetes, y el tipo `billete` consiste en un número de serie y en una probabilidad de ser falso, además de las operaciones `operator==`, `operator<`, `operator int()`, devolviendo esta última el año de emisión de un billete. Los billetes de falsedad confirmada tienen una probabilidad igual a la constante `probabilidad_max` definida en `tp3.h`.

Ejercicio 3 - Multiplicación de matrices

La complejidad del algoritmo básico de multiplicación de matrices es $O(n^3)$ para matrices de $n \times n$. Se desea implementar una multiplicación de matrices de complejidad asintóticamente mejor.

Si aplicamos la técnica de Divide & Conquer de manera directa para particionar las matrices y multiplicamos de la siguiente manera:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

el algoritmo resultante sigue teniendo complejidad $O(n^3)$. Para mejorarlo, se pide implementar el *algoritmo de Strassen* de multiplicación de matrices. Dada la misma partición de A y B propuesta arriba, el algoritmo de Strassen propone la siguiente manera de calcular el producto:

$$\begin{aligned} M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) & M_5 &= (A_{11} + A_{12}) \times B_{22} \\ M_2 &= (A_{21} + A_{22}) \times B_{11} & M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}) \\ M_3 &= A_{11} \times (B_{12} - B_{22}) & M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ M_4 &= A_{22} \times (B_{21} - B_{11}) \end{aligned}$$

Finalmente, los cuatro cuadrantes del resultado de la multiplicación se definen como:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}$$

Esta compleja manera de estructurar las operaciones ahorra una multiplicación de matrices con respecto al algoritmo directo de Divide & Conquer. De todas maneras, Strassen agrega varias operaciones de suma/resta al algoritmo, y por ello no es eficiente para matrices demasiado pequeñas. La implementación debe utilizar al algoritmo tradicional iterativo (definido en `tp3.h`) de multiplicación para matrices de tamaño $K \times K$ o menor, con un K dado como parámetro.

Implemente la operación

```
inline Matriz multiplicar_strassen(const Matriz& A, const Matriz& B, int K)
```

donde el tipo `Matriz` es un renombre de `vector<vector<double> >`. Defina y utilice las operaciones auxiliares que crea necesarias.

Entrega

El trabajo práctico se entregará por mail a `algo2.dc+tp3@gmail.com` independientemente del turno de cursada. El código desarrollado se entregará adjuntando **únicamente el archivo tp3_impl.h** hasta el día **Martes 21 de Noviembre a las 23:59hs**. El mail deberá tener como **Asunto** los números de libreta separados por ;. Por ejemplo:

To: algo2.dc+tp3@gmail.com
From: alumno-algo2@dc.uba.ar
Subject: 102/09; 98/10
Adjunto: tp3_impl.h

Habrá las siguientes fechas de reentrega del TP3: **Martes 28/11**, **Martes 5/12** y **Viernes 8/12**. Todas las fechas de entrega y reentrega del TP3 también serán fechas de reentrega de **cuaquier taller adeudado**.

El **Viernes 8/12** es la última fecha posible para (re)entregas de cualquier índole.