Tareas 4, 5, 6
Algoritmos y Complejidad

«*Entregar platas*»

Algorithm Knaves

2021-11-07

En la perdida colonia antártica de Nadiria usaban billetes de $1, $4, $7, $13, $28, $52, $91 y $365. Como el papel era extremadamente escaso, por ley siempre debía entregarse el mínimo número de billetes en cada transacción.

Resulta que el algoritmo voraz obvio (entregue el máximo que pueda de la máxima denominación sin usar, y continúe con la denominación siguiente) para las denominaciones de Nadiria no siempre da el mínimo número de billetes. El algoritmo voraz entrega un óptimo con los billetes chilenos ($1 000, $2 000, $5,000, $10 000 y $20 000) y con las monedas en Euros (1¢, 2¢, 5¢, 10¢, 20¢, 50¢, 1€ y 2€). Una pregunta interesante es si la moneda belga de 2,50€ que conmemora la derrota de Napoleón en Waterloo rompe esto.

Las tareas siguientes son independientes, solo tienen en común el tema general.

Tarea 5: Escriba una función C++ que calcule las cantidades a entregar usando el algoritmo voraz, con prototipo (suplido por el encabezado `greedy.hh`):

```
std::vector<unsigned int>
greedy(unsigned int m,
       std::vector<unsigned int> d);
```

Acá `q` es la cantidad, `d` son las denominaciones (en orden ascendente, `d[0] == 1`). Retorna el vector de números de cada denominación a entregar.

Su archivo fuente debe llamarse `greedy.cc`, y debe incluir únicamente el encabezado `greedy.hh` provisto (además de posibles encabezados estándar).

Tarea 6: Dadas denominaciones enteras $d_1, d_2, \ldots, d_n$, se sabe que determinar si se puede entregar la cantidad q con b billetes o menos es NP-completo [1]. Sin embargo, Pearson [2, 3] describe un algoritmo polinomial para determinar la mínima cantidad para la que el algoritmo voraz obvio da más billetes de lo necesario. Escriba una función en C++ con prototipo (suplido por el encabezado `pearson.hh`):

```
std::vector<unsigned int>
pearson(std::vector<unsigned int> d);
```

que entregue el vector óptimo para el mínimo valor para el cual con las denominaciones d (vienen ordenadas de menor a mayor, d[0] == 1) el algoritmo voraz no da un óptimo, o un vector vacío si el algoritmo voraz siempre da un óptimo.

Su archivo fuente debe llamarse pearson.cc, y debe incluir únicamente el encabezado pearson.hh provisto (además de posibles encabezados estándar). Si las requiere, puede usar la función greedy y su respectivo encabezado greedy.hh de la tarea 5 (proveeremos la función de la pauta de esa tarea) y las funciones utilitarias declaradas en utility.hh.

Tarea 7: Usando el esquema de diseño de programación dinámica del apunte, escriba un programa eficiente para obtener el mínimo número de monedas para entregar una cantidad dada q. Su función debe tener prototipo (suplido por el encabezado change.hh):

```
std::vector<unsigned int>
change(unsigned int q, std::vector<unsigned int> d);
```

Acá q es la cantidad, d son las denominaciones (en orden ascendente, con d[0] == 1), retorna el vector de números de billetes de cada denominación a entregar.

Su archivo fuente final debe llamarse change.cc, y debe incluir únicamente el encabezado change.hh provisto (además de posibles encabezados estándar).

En pasos preliminares desarrollará funciones que únicamente retornen el número de billetes en la solución óptima, llame count() estas funciones, con prototipo dado por count.hh:

```
unsigned int
count(unsigned int q, std::vector<unsigned int> d);
```

Se proveen esqueletos de las funciones solicitadas (no hacen nada útil), funciones utilitarias (utility.hh, utility.cc) y programas de prueba ejemplo (tst_greedy.cc, tst_pearson.cc, tst_count.cc, tst_change.cc).

Referencias

[1] G. S. Lueker: *Two* NP*-complete problems in nonnegative integer programming.* Technical Report TR-178, Princeton University, Department of Computer Science, March 1975.

[2] David Pearson: *A polynomial-time algorithm for the change-making problem.* Technical Report TR94-1433, Department of Computer Science, Cornell University, 1994.

[3] David Pearson: *A polynomial-time algorithm for the change-making problem.* Operation Research Letters, 33(3):231–234, May 2005.

Condiciones de entrega

- Las tareas se realizarán *individualmente* (esto es grupos de una persona), sin excepciones.
- Las entregas deben realizarse vía Aula en un *tarball* en el área designada al efecto, en el formato `tarea-<num>.tar.gz`, conteniendo su archivo fuente (`greedy.cc`, `pearson.cc` o `change.cc`, según la tarea), fuentes LaTeX de su discusión del desarrollo y el PDF del caso.

  Cada tarea se entrega por separado. La entrega debe realizarse dentro del plazo indicado en Aula.
- Asegúrese que todas sus entregas tengan sus datos completos: número de la tarea, ramo, semestre, nombre y rol. Puede incluirlas como comentarios en sus fuentes LaTeX (en TeX comentarios son desde % hasta el final de la línea) o en posibles programas. Anótese como autor de los textos.
- En la portada de su texto deberá incluir una tabla como la siguiente:

Concepto	**Tiempo [min]**
Revisión	
Desarrollo	
Informe	

  Acá *revisión* incluye revisión de apuntes, búsquedas en Internet, lectura de otras referencias; *desarrollo* es el tiempo invertido en la solución pedida; *informe* se refiere al tiempo requerido para confeccionar los entregables.
- Si usa material adicional al discutido en clases, detállelo. Agregue información suficiente para ubicar ese material (en caso de no tratarse de discusiones con compañeros de curso u otras personas).
- Los programas se evalúan según que tan claros (bien escritos) son, si se compilan y ejecutan sin errores o advertencias según corresponda. Parte del puntaje es por ejecución correcta con casos de prueba. Si el programa no se ciñe a los requerimientos de entrada y salida, la nota respectiva es cero.
- La entrega debe realizarse dentro del plazo indicado en Aula.
- Nos reservamos el derecho de llamar a interrogación sobre algunas de las tareas entregadas. En tal caso, la nota base (antes de descuentos por atraso y otros) es la de la interrogación. No presentarse a la interrogación sin justificación previa significa automáticamente nota cero.