Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 1A. [27 de abril de 2004]

[Ej. 1] **[Clases (20 puntos)]** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores. Las funciones a implementar son `insert(p,x)`, `erase(p)`, `erase(p,q)`, `clear()`, `begin()`, `end()`, `next(p)`, `retrieve(p)`. Observaciones:

- Escribir tanto las declaraciones como las funciones (archivos `.h` y `.cpp`).
- Incluir las definiciones de tipo (`typedef`) y clases auxiliares necesarias.
- Se puede escribir la interface avanzada (con templates, clases anidadas, sobrecarga de operadores).

[Ej. 2] **[Programación (total = 60 puntos)]**

a) **[anagrama (40 puntos)]** Escribir una función `bool anagrama_p(list<int> &L1,list<int> &L2);` que retorna verdadero sólo si `L1` y `L2` tienen la misma cantidad de elementos y los elementos de `L1` son una permutación de los de `L2`. Por ejemplo, si `L1`=$(1, 23, 21, 4, 2, 3, 0)$ y `L2`=$(21, 1, 3, 2, 4, 23, 0)$ entonces `anagrama_p(L1,L2)` debe retornar `true`, mientras que si `L1`=$(1, 3, 5)$ y `L2`=$(4, 5, 4)$ debe retornar `false`. En el caso de que haya elementos repetidos estos tienen que estar el mismo número de veces en las dos listas. Se sugiere el siguiente algoritmo: Para cada elemento de `L1` se recorre `L2`. Si el elemento está en `L2` entonces se suprime y se continúa con el siguiente elemento, hasta recorrer todos los elementos de `L1`. Si el elemento no está en `L2` entonces directamente se retorna `false`.
Restricciones:

- Usar la interfase STL para listas.
- No usar el operador `--`.
- No usar ninguna estructura auxiliar ni otros algoritmos de STL.
- El algoritmo debe ser $O(n^2)$.
- El algoritmo puede ser *"destructivo"* (elimina elementos de los contenedores).
- Prestar a no usar posiciones inválidas al iterar sobre las listas.

b) **[flota-pares (15 puntos)]** Escribir una función `void flota_pares(stack<int> &P);` que reordena los elementos de una pila `P` de tal manera que los pares quedan arriba de los impares. Los elementos pares deben quedar en el mismo orden entre sí, y lo mismo para los impares. Por ejemplo, si `P`=$(4, 17, 9, 7, 4, 2, 0, 9, 2, 17)$ entonces después de hacer `flota_pares(P)` debe quedar `P`=$(4, 4, 2, 0, 2, 17, 9, 7, 9, 17)$. Usar dos pilas auxiliares.
Restricciones:

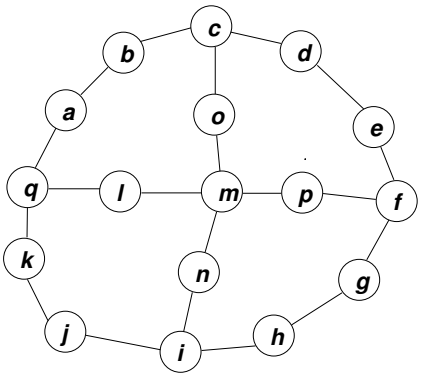
- Usar la interfase STL para pilas.
- No usar más estructuras auxiliares que los indicados ni otros algoritmos de STL.
- El algoritmo debe ser $O(n)$.

c) **[cum-sum-cola (15 puntos)]** Escribir una función `void cum_sum(queue<int> &Q)` que modifica a `Q` dejando la suma acumulada de los elementos, es decir, si los elementos de `Q` antes de llamar a `cum_sum(Q)` son $(a_0, a_1, \ldots, a_{n-1})$, entonces después de llamar a `cum_sum(Q)` debe quedar `Q`=$(a_0, a_0 + a_1, \ldots, a_0 + a_1 + \ldots + a_n)$. Por ejemplo, si `Q`=$(1, 3, 2, 4, 2)$ entonces después de hacer `cumsum(Q)` debe quedar `Q`=$(1, 4, 6, 10, 12)$. Usar una cola auxiliar.
Restricciones:

- Usar la interfase STL para colas (`clear()`, `front()`, `pop()`, `push(T x)`, `size()`, `empty()`).
- No usar más estructuras auxiliares que la indicada ni otros algoritmos de STL.

- El algoritmo debe ser $O(n)$.

[Ej. 3] **[color-grafo (10 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido.



[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) El tiempo de ejecución del operador `--` para listas *simplemente* enlazadas es ... (n es el número de elementos en la lista)

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$

---

b) El tiempo de ejecución de la función `clear()` para listas implementadas por punteros es ... (n es el número de elementos en la cola)

- ☐ ... $O(\log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(n)$

---

c) Sea `L` una lista conteniendo los elementos $(1, 3, 5)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q,r;
p=L.begin();
q=p++;
r=L.erase(p);
```

¿Cuál de las opciones es verdadera?

- ☐ `*p` es 3, `*r` es 5, `*q` es 1.
- ☐ `p` es inválido, `*r` es 5, `*q` es 3.
- ☐ `p` es inválido, `*r` es 5, `*q` es 1.
- ☐ `p` es inválido, `*r` es 5, `q` es inválido.

---

d) Considere el siguiente código:

```
map<int,int>M;
M[3] = 7;
map<int,int>::iterator p = M.find(3);
p->second = 9;
```

¿Cuál es el valor de `M[3]`?

- ☐ 0. (El valor por defecto generado por una asignación accidental).
- ☐ Da un error.
- ☐ 9
- ☐ 7