Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio 3. [2019-10-31]

PASSWD PARA EL ZIP: 2GQ5 3M67 RG27

Ejercicios

[Ej. 1] [spaceship] (AB)

Recordemos que una relación de orden puede definirse a través de un predicado binario `bool operator<(T a,T b);`. Vimos también que, en realidad, puede definirse a partir de cualquiera de los operadores <, <=, >, >=. Alternativamente puede definirse el operador "nave espacial" (*spaceship* operator) introducido en la norma **C++20**, definido de la siguiente forma: **a<=>b** debe retornar **+1** si **a>b**, 0 si **a==b**, -1 si **a<b**. La ventaja del operador <=> es que cualquiera de los otros operadores puede escribirse en términos de <=> con una sola llamada al mismo.
En realidad no es necesario que spaceship retorne exactamente **-1,0,1** sino que basta con que retorne un número positivo o negativo. Por lo tanto para enteros basta con que **spacehip(a,b)** retorne **a-b**.
Consigna: Escribir una función `int spaceship(btree<int> &T1,btree<int>&T2);` que define el operador **spaceship** para árboles binarios (AB).
[**Nota:** Como el operador <=> no puede ser sobrecargado en las versiones actuales del compilador lo implementaremos en términos de una función **spaceship(.,.)**.]
Para el árbol binario podemos definir **spaceship** en forma recursiva de la siguiente forma. Básicamente se compara primero la raíz, después los hijos izquierdos, despues los derechos. En cada caso si el resultado de **spaceship** de la comparación no es nulo, ya se puede retornar ese valor. Básicamente es lo siguiente,

- Si **spaceship(*n1,*n2)!=0** retornar **spaceship(*n1,*n2)**
- Si **spaceship(l1,l2)!=0** retornar **spaceship(l1,l2)**
- Si **spaceship(r1,r2)!=0** retornar **spaceship(r1,r2)**
- Retornar 0

Sin embargo, a estas instrucciones hay que modificarlas para considerar el caso de que los nodos **n1** o **n2** sean **Lambda**. Lo más simple es considerar que, si un nodo es **Lambda** entonces debe considerarse como $-\infty$ (como hacemos en la comparación lexicográfica). Entonces,

- Si **n1==n2==Lambda** entonces retorna **0**.
- Si **n1==Lambda** y **n2!=Lambda**, retorna **-1**.
- Si **n1!=Lambda** y **n2==Lambda**, retorna **+1**.
- Si **spaceship(*n1,*n2)!=0** retornar **spaceship(*n1,*n2)**
- Si **spaceship(l1,l2)!=0** retornar **spaceship(l1,l2)**
- Si **spaceship(r1,r2)!=0** retornar **spaceship(r1,r2)**
- Si llegamos aquí los árboles son iguales y por lo tanto debe retornar **0**

Notar que en la línea donde dice **spaceship(*n1,*n2)** debe aplicarse el spaceship de enteros, como se describió más arriba.

[Ej. 2] [mkhuffman] (AB)

Implemente la función **btree<char>** `makeHuffmannTree(list<pair<btree<char>,float>> & bosque);` que reciba una lista de pares **pair<btree<char>, float> bosque** con los árboles iniciales y su peso respectivo (procentaje de ocurrencia) del proceso de Huffmann y retorne el árbol de la codificación resultante.

Ayuda:

- Mientras haya más de un árbol en el bosque:
  - Seleccionar los dos árboles de peso mínimo `Tl` y `Tr`. Para ello deberá utilizar la función auxiliar **`getMin`** provista que retorna un iterador al árbol de peso mínimo.
  - Remover del bosque los pares que involucran dichos árboles
  - Generar un par con los datos `T = ("-" Tl Tr)` y `w = wl+wr` y almacenarlo en la última posición del bosque. Notar que se inserta un char guión "`-`" como raíz.
- Retornar el árbol resultante

[Ej. 3] [min-cost-sets] (Conjuntos)

Dado un conjunto universal `U` de n elementos y una lista de subconjuntos de `U` denominada `S = (S1, S2,...,Sm)` en donde cada subconjunto `Si` tiene un costo asociado, se pide implementar una función `int minCostSets(set<int>& U, vector<set<int>>& S, map<set<int>, int>& costos);` que retorna el costo de la sublista de `S` de costo mínimo y que cubra todos los elementos de `U`. Si no se puede cubrir el conjunto `U` se debe retornar `INT_MAX`.

Ejemplo 1: Retorna costo mínimo 13 correspondiente a la sublista `(S2, S3)`.

```
U = {1,2,3,4,5}
S = {S1,S2,S3}

S1 = {4,1,3},    Cost(S1) = 5
S2 = {2,5},      Cost(S2) = 10
S3 = {1,4,3,2},  Cost(S3) = 3
```

Ejemplo 2: Retorna costo mínimo `INT_MAX` ya que no se puede cubrir `U`.

```
U = {1,6}
S = {S1,S2}
S1 = {1},    Cost(S1) = 2
S2 = {},      Cost(S2) = 4
```

Ayuda:

- Escribir una función auxiliar recursiva `int minCostSets(set<int>& U, vector<set<int>> S, map<set<int>, int>& costos, list<set<int>>& L, int pos);` donde `L` es la lista de conjuntos que participan de la solución actual.
- Utilizar el índice `pos` para procesar el vector de conjuntos `S` y poder desplazarse sobre él en cada llamada a la recursión.
- Cuando se inserta un conjunto en la lista de conjuntos `L`, quitar todos sus elementos del conjunto `U` para reducir el problema y llegar a una solución. Recordar que se busca el listado de subconjuntos tal que su costo sea mínimo y que su unión sea igual al conjunto `U`.
- Por ejemplo, dado el conjunto `U = {1, 2, 3, 4, 5}`, si se insertó el conjunto `Si = {1, 2, 3}` en `L` entonces, al quitar los elementos de `Si`, el conjunto queda como `U = {4, 5}`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.

- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum `H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

mstorti@galileo/tpl2-2019-52-g9b8020c4/Thu Oct 31 17:34:09 2019 -0300

TPL3. Trabajo Práctico de Laboratorio 3. [2019-10-31]. TABLA SEED/HASH

```
S=123 -> H1=431 H2=882 H3=507
S=577 -> H1=703 H2=452 H3=981
S=393 -> H1=183 H2=648 H3=795
S=686 -> H1=165 H2=115 H3=421
S=349 -> H1=309 H2=104 H3=045
S=762 -> H1=681 H2=933 H3=407
S=690 -> H1=622 H2=926 H3=371
S=663 -> H1=176 H2=497 H3=689
S=340 -> H1=884 H2=139 H3=155
S=872 -> H1=430 H2=041 H3=076
S=711 -> H1=360 H2=758 H3=705
S=367 -> H1=162 H2=811 H3=411
S=882 -> H1=855 H2=298 H3=671
S=162 -> H1=434 H2=944 H3=769
S=767 -> H1=300 H2=392 H3=493
S=429 -> H1=446 H2=836 H3=099
S=622 -> H1=584 H2=228 H3=197
S=969 -> H1=554 H2=510 H3=520
S=893 -> H1=366 H2=895 H3=759
S=311 -> H1=554 H2=851 H3=330
S=529 -> H1=125 H2=475 H3=419
S=721 -> H1=070 H2=396 H3=487
S=384 -> H1=501 H2=393 H3=726
S=798 -> H1=730 H2=466 H3=485
S=615 -> H1=824 H2=988 H3=486

S=386 -> H1=878 H2=876 H3=477
S=215 -> H1=540 H2=226 H3=694
S=935 -> H1=777 H2=905 H3=245
S=292 -> H1=852 H2=693 H3=696
S=821 -> H1=393 H2=287 H3=556
S=527 -> H1=319 H2=266 H3=109
S=359 -> H1=280 H2=544 H3=855
S=626 -> H1=808 H2=371 H3=060
S=226 -> H1=429 H2=546 H3=495
S=236 -> H1=372 H2=104 H3=780
S=468 -> H1=481 H2=461 H3=560
S=529 -> H1=125 H2=475 H3=419
S=630 -> H1=596 H2=481 H3=101
S=923 -> H1=898 H2=785 H3=687
S=335 -> H1=204 H2=685 H3=523
S=802 -> H1=636 H2=183 H3=963
S=958 -> H1=230 H2=942 H3=389
S=967 -> H1=847 H2=207 H3=352
S=656 -> H1=468 H2=611 H3=503
S=242 -> H1=799 H2=586 H3=588
S=973 -> H1=215 H2=540 H3=045
S=219 -> H1=343 H2=977 H3=413
S=437 -> H1=194 H2=187 H3=319
S=624 -> H1=294 H2=040 H3=985
S=670 -> H1=985 H2=307 H3=605
```