Problemes LP – Curs 2018-19
Tema 6: Arbres

Exercici 2 (nivell mig) – EXERCICI AVALUABLE

Una de les aplicacions més habituals i directes dels *binary heaps* és la implementació d'una cua amb prioritats. Una cua amb prioritats funciona de forma similar a una cua ordinària, però amb la diferència que a l'hora de treure un element de la cua no traiem el més antic (el primer que ha entrat a la cua), sinó els més prioritari. En el nostre cas suposarem que un element és més prioritari que un altre simplement si és més petit (segons retorni l'operador <). Per tant, si guardem els elements en un *binary heap* retornar el primer element de la cua serà equivalent a retornar l'element mínim (el més prioritari), que sabem que sempre està a l'arrel del *binary heap*.

a) Per poder implementar la cua amb prioritats primer haurem de modificar la classe Heap que hem utilitzat a les sessions de classe per afegir un mètode que permeti eliminar un element donat del *heap*. Aquest mètode tindrà aquesta capçalera:

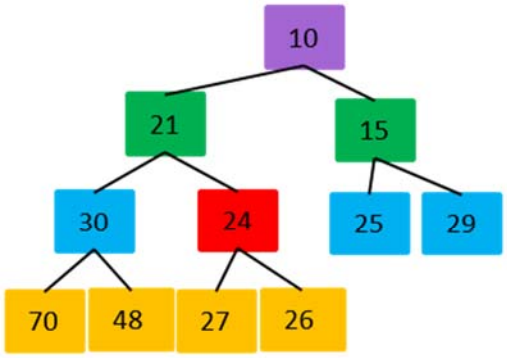
```
void esborra(const T& el);
```

El mètode ha d'eliminar del *heap* l'element que sigui igual al valor que es passa com a paràmetre. Per fer-ho, haurem de seguir aquests passos:

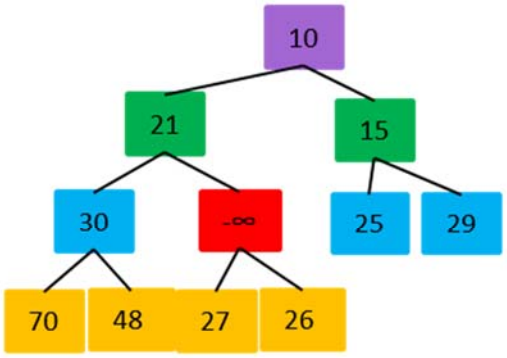
1. Buscarem l'element dins del vector que ens serveix per guardar tots els elements.
2. Un cop l'haguem trobat, substituirem el seu valor pel mínim valor possible del tipus dels elements que guardem al *heap*. Suposarem que el constructor per defecte ens retorna sempre el mínim valor possible per tots els objectes de la classe.
3. Ascendirem aquest valor mínim per l'arbre, igual que es fa quan afegim un element qualsevol al *heap*. Com que aquest valor és el mínim possible acabarà ocupant la posició de l'arrel de l'arbre.
4. Eliminarem l'element mínim del *heap* (l'arrel) cridant al mètode per esborrar el mínim que ja vam implementar a les sessions de classe.

A la figura que hi ha a sota podem veure un exemple dels 4 passos que hem de seguir.

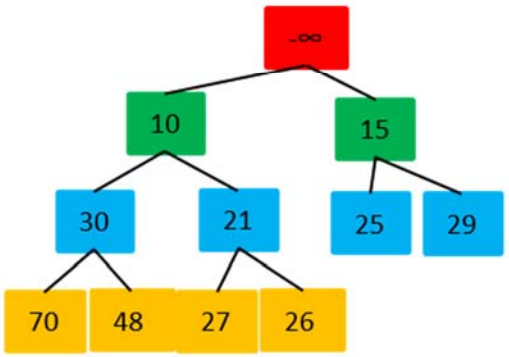
Eliminar 24



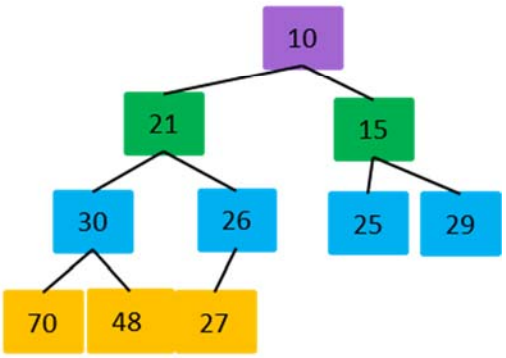
Buscar element



Substituir per mínim del tipus



Ascendir-lo fins l'arrel



Eliminar mínim (arrel)

b) Implementar la classe `CuaPrioritat` prioritat utilitzant la classe `Heap` modificada de l'apartat anterior. La interfície pública d'aquesta classe ha de ser la següent:

- `CuaPrioritat(const CuaPrioritat<T>& cua)`

  Constructor de còpia.
- `CuaPrioritat& operator=(const CuaPrioritat<T>& cua)`

  Operador s'assignació.
- `bool esBuida() const`

  Comprova si la cua està buida.
- `int getNElements() const`

  Retorna el nombre d'elements guardats a la cua.
- `T& top()`

  Recupera l'element més prioritari (el més petit) de tots els que están guardata a la cua.
- `void push(const T& element)`

Afegeix un element a la cua, col·locant-lo en ordre segons la seva prioritat.

- `void pop()`

  Elimina l'element més prioritari de la cua.

- `void remove(const T& element)`

  Elimina de la cua l'element que sigui igual al valor que es passa com a paràmetre.

Tingueu en compte que la implementació de tots aquests mètodes de la cua es trasllada de forma gairebé directa en crides a mètodes de la classe Heap. Tal com ja hem comentat suposarem que la prioritat dins la cua ve donada per l'ordre natural dels elements (un element més petit és un element més prioritari).

Per testejar la classe `CuaPrioritat` a Caronte us donem una classe `Tasca` ja implementada amb els mètodes necessaris per crear objectes de la classe i recuperar el valor dels atributs. Aquesta classe té implementats també els operadors < i == es necessiten per poder guardar i buscar elements a la cua per prioritat. Si l'operador < retorna que una tasca és més petita que una altra voldrà dir també que és més prioritària.
Us donem també la classe `Heap` que hem fet a les sessions de classe perquè la feu servir com a punt de partida a l'apartat a).

Exercici 3 (nivell mig) – EXERCICI AVALUABLE

En un dels exercicis del tema 1 de *Templates* us vam demanar la implementació d'una classe Map similar a la de la llibreria estàndard (http://www.cplusplus.com/reference/map/map/). Aquesta classe permet guardar un conjunt d'elements <clau, valor> i permet accedir als elements directament per la clau amb un cost petit.

En aquell exercici vam implementar la classe Map amb un vector ordenat per la clau. Ara volem canviar la implementació de la classe i enlloc d'un vector, utilitzar un *arbre red-black* com els que hem vist a les sessions de classe. L'arbre ha de guardar parells de clau i valor i els elements dins l'arbre de cerca han d'estar organitzats segons la clau de cada element.

Per poder implementar la classe Map us donem el codi complet de la classe TreeRB que hem anat desenvolupant a classe per implementar els *arbres red-black* i que està en el material de la sessió 20 actualitzat amb les solucions.

També us donem (dins del fitxer Map.h) el codi d'una classe PairMap que podeu fer servir per guardar els parells <clau, valor> dins de l'arbre. Aquesta classe funciona igual que la classe pair de la llibreria estàndard, però té redefinits els operadors < i == perquè a l'hora de comparar valors només es tingui en compte el primer element del parell (la clau). D'aquesta forma es poden fer servir per buscar elements dins l'arbre utilitzant només la clau com a valor de cerca.

La interfície pública de la classe Map serà la següent:

- `Map()`

  Constructor per defecte que inicialitza tots els atributs de forma segura.

- `Map(const Map<TClau, TValor>& m)`

  Constructor de còpia.

- `~Map()`

  Destructor.

- `bool esBuit() const`

  Comprovar si el conjunt està buit

- `TValor& operator[](const TClau& clau)`

  L'operador [] permet accedir a un element qualsevol del conjunt a partir de la seva clau. Si la clau no existeix s'ha de retornar el valor per defecte del tipus del valor que es guarda al conjunt (el valor per defecte és el valor que s'obté després de cridar al constructor per defecte). Tingueu en compte que com que s'ha de retornar per referència, aquest valor s'haurà de guardar en un atribut de la classe (no podrà ser una variable local del mètode). TClau és el tipus de la clau i TValor el tipus dels valors que es guarden al conjunt

- `void afegeix(const TClau& clau, const TValor& valor);`

  Afegeix un nou parell <clau, valor> al conjunt. Tingueu en compte que els elements han de quedar sempre organitzats segons la clau. Si la clau ja existia al conjunt es substitueix el valor que tingués pel nou valor que es passa com a paràmetre.

- `friend std::ostream& operator<<(std::ostream& out, const Map<TClau, TValor>& m)`

Sobrecàrrega de l'operador << per mostrar el contingut del conjunct per pantalla. Simplement ha de cridar a l'operador << de *l'arbre red-black* que guardar tots els valors del conjunt per mostrar per pantalla l'arbre complet.