UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Vilanova i la Geltrú

PUBLICACIÓ DOCENT

COL·LECCIÓ DE PROBLEMES D'ESIN

AUTOR: Bernardino Casas, Jordi Esteve
ASSIGNATURA: Estructura de la Informació (ESIN)
CURS: Q3
TITULACIONS: Grau en Informàtica
DEPARTAMENT: Ciències de la Computació
ANY: 2018/2019

Vilanova i la Geltrú, 1 d'octubre de 2018

Índex

1

Programació orientada a objectes

1.1 Exercicis

1.1 Fer una classe *data* que permeti emmagatzemar la informació d'una data del calendari i que tingui les següents operacions:

- *crea()* retorna la data (1/1/1980).
- *crea(d, m, a)* retorna una data el valor de la qual està indicat pels dos naturals (dia i mes) i l'enter (any). Genera un error si la data no és vàlida.
- *següent()* retorna una nova data que ve a continuació de la que tenim.
- *anterior()* retorna una nova data immediatament anterior a la que tenim.
- *incrementa(d)* incrementa la data amb el nombre de dies indicat.
- *decrementa(d)* decrementa la data amb el nombre de dies indicat.
- *dia()* retorna el dia de la data.
- *mes()* retorna el mes de la data.
- *any()* retorna l'any de la data.
- *dies_transcorreguts(dat)* indica els dies en valor absolut que han transcorregut entre les dues dates.
- *any_traspas()* indica si aquesta data pertany a un any de traspàs. Un any és de traspàs si és divisible per 4, excepte l'últim de cada segle (aquell que és divisible per 100), exceptuant que aquest any sigui divisible per 400.

- *igualtat(d2)* indica si les dues dates indicades són iguals.
- *diferència(d2)* indica si les dues dates indicades són diferents.
- *menor(d2)* indica si la primera data és menor que la segona.
- *menor_igual(d2)* indica si la primera data és menor o igual que la segona.
- *major(d2)* indica si la primera data és major que la segona.
- *major_igual(d2)* indica si la primera data és major o igual que la segona.

Fes:

1. Mètodes de la classe.
   (a) Classifica les operacions de la classe *data* entre **creadores**, **modificadores** i **consultores**.
   (b) Tria les operacions generadores.
2. Especificació.
   Crea l'especificació de la classe *data* en C++ (capçalera o també anomenat fitxer .HPP).
   (a) Escriu la definició de la classe, és a dir, la part pública.
   (b) Escriu la representació de la classe, és a dir, la part privada.
3. Implementació.
   Implementa els mètodes de la classe.
   (a) El mètode constructor, incrementa o decrementa, any de traspàs.
   (b) Els mètodes consultors i algun mètode de comparació.

   NOTA: Per implementar aquests mètodes pots crear els mètodes privats que creguis necessaris.

2

Eficiència temporal i espacial

2.1 Exercicis

2.1 Agrupa les divuit funcions següents de manera que $f(n)$ i $g(n)$ pertanyin al mateix grup si i només si $f(n) \in O(g(n))$ i $g(n) \in O(f(n))$, i enumera els grups en ordre ascendent (de menys a més).

$\sqrt{n}$	n	2^n	$n \log n$	$n - n^3 + 7n^5$	$n^2 + \log n$
n^2	n^3	$\log n$	$n^{1/3} + \log n$	$(\log n)^2$	$n!$
$\ln n$	$n / \log n$	$\log \log n$	$(1/3)^n$	$(3/2)^n$	6

2.2 Indica, per a cada parell de funcions f i g següents, si és cert o fals que $f \in O(g)$ i que $f \in \Theta(g)$. Omple cada cel·la d'aquesta taula amb SI o NO segons correspongui.

f	g	$f \in O(g)$	$f \in \Theta(g)$
$n^2 + 3n + 4$	$7n^2$		
$n \log_2 n$	$n\sqrt{n}$		
$n^{0,1}$	$(\log_2 n)^5$		
$(\log_2 n)^2$	$\log_2 n + 1$		
$\log_2(n^2)$	$2 \log_2 n$		
2^n	3^n		

2.3 De cadascun dels fragments de codi següents, analitzeu el seu cost.

```
// (a) Fragment 1
int s = 0;
for (int i = 0; i < n; ++i) {
  ++s;
}

// (b) Fragment 2
int s = 0;
for (int i = 0; i < n; i += 2) {
  ++s;
}

// (c) Fragment 3
int s = 0;
for (int i = 0; i < n; ++i) {
  ++s;
}
for (int j = 0; j < n; ++j) {
  ++s;
}

// (d) Fragment 4
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    ++s;
  }
}

// (e) Fragment 5
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < i ; ++j) {
    ++s;
  }
}

// (f) Fragment 6
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = i ; j < n; ++j) {
    ++s;
  }
}

// (g) Fragment 7
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    for (int k = 0; k < n; ++k) {
      ++s;
    }
  }
}
```

```
// (h) Fragment 8
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < i ; ++j) {
    for (int k = 0; k < j ; ++k) {
      ++s;
    }
  }
}

// (i) Fragment 9
int s = 0;
for (int i = 1; i <= n; i *= 2) {
  ++s;
}

// (j) Fragment 10
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < i * i ; ++j) {
    for (int k = 0; k < n; ++k) {
      ++s;
    }
  }
}

// (k) Fragment 11
int s = 0;
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < i * i ; ++j) {
    if (j % i == 0) {
      for (int k = 0; k < n; ++k) {
        ++s;
      }
    }
  }
}
```

2.4 Donat el següent algorisme:

```
  //crida inicial: B(v, 0, n-1)}
  //Pre: i <= j}
  int B(int v[], int i, int j) {
    int m1, m2, m3, aux;

    aux = (j - i + 1) / 4;
    if (aux > 0) {
      m1 = i - 1 + aux;
      m2 = i - 1 + aux + aux;
      m3 = i - 1 + aux + aux + aux;
      return B(v, i, m1) + B(v, m1+1, m2) + B(v, m2+1, m3) + B(v, m3+1, j);
    }
```

```
  else {
    return FG(v, i, j);
  }
}
```

(a) Escriu l'equació de recurrència per l'algorisme anterior tenint en compte que la funció `FG` té cost constant.

(b) Calcula el cost asimptòtic temporal de l'algorisme.

2.5 Considera els següents algorismes `maxd` i `maxs`:

```
// crida inicial: maxd(v, 0, n-1)}
int maxd(int v[], int i, int j) {
  int m, max1, max2;
  if (i < j) {
    m = (i + j) / 2;
    max1 = maxd(v, i, m-1);
    max2 = maxd(v, m, j);
    if (max1 < max2) {
      return max2;
    }
    else {
      return max1;
    }
  }
  else {
    return v[i];
  }
}

// crida inicial: maxs(v, n-1)}
int maxs(int v[], int i) {
  if (i == 0) {
    return v[i];
  }
  else {
    int m = maxs(v, i-1);
    if (v[i] < m) {
      return m;
    }
    else {
      return v[i];
    }
  }
}
```

(a) Calcula el cost asimptòtic temporal dels dos algorismes (de les crides inicials) en funció de n.

(b) Quin d'ells escolliries? Per què?

2.6 Considera els següents algorismes A i B:

```
//Pre: i <= j
int A(int *v, int i, int j) {
  int m;
  if (i < j) {
    f(v, i, j);
    m = (i + j) / 2;
    return A(v, i, m - 1) + A(v, m, j) + A(v, i + 1, m);
  }
  else {
    return v[i];
  }
}

// Pre: i <= j
int B(int *v, int i, int j) {
  int m1, m2;
  if (i < j) {
    g(v, i, j);
    m1 = i + (j - i + 1) / 3;
    m2 = i + 2 * (j - i + 1) / 3;
    return  B(v, i, m1-1) + B(v, m1, m2-1) + B(v, m2, j);
  }
  else {
    return v[i];
  }
}
```

(a) Suposant que el cost de la funció $f \in \Theta(1)$ i que el cost de g és proporcional a la grandària del subvector a processar (és a dir, $\Theta(j - i + 1)$), quin és el cost asimptòtic temporal d'A i de B en funció de n? Per fer els càlculs escriu abans l'equació de recurrència de cadascun d'aquests algorismes.

(b) Quin d'ells escolliries, suposant que fan el mateix?

2.7 Donat el següent algorisme d'ordenació:

```
void ordena(int A[], int i, int j) {
  int k;
  if (i < j) {
    divideix(A, i, j, k);
    // Tots els elements del segment i..k són més petits o iguals que
    // qualsevol element del segment k+1..j de la taula A i k=(i+j)/2
    ordena(A, i, k);
    ordena(A, k+1, j);
```

```
    }
  }
```

Tenint en compte que la crida inicial és `ordena(A, 0, n-1)` :

(a) Escriu l'equació de recurrència d'aquest algorisme.

(b) Calcula el seu cost en cas mig i en cas pitjor en funció de n, suposant que el cost de divideix és $\Theta(m)$ en cas mig i $\Theta(m^2)$ en cas pitjor, on m=j-i+1 és la grandària del (sub)vector a dividir. Justifica la teva resposta.

(c) Consideres que aquest algorisme és un bon algorisme d'ordenació? Per què?

2.8 Per a tota n potència de 4, sigui la recurrència:

$$T(n) = \begin{cases} 3 & si\ n = 1 \\ 4T(n/4) + 3 & si\ n > 1 \end{cases}$$

Considera també una altra recurrència:

$$C(m) = \begin{cases} 3 & si\ m = 1 \\ 4C(m-1) + 3 & si\ m > 1 \end{cases}$$

(a) Troba una solució exacta per a $T(n)$ i demostra que és correcta.

(b) Troba una solució exacta per a $C(m)$ i demostra que és correcta.

(c) Quina relació hi ha entre les dues recurrències?

2.9 Resol les recurrències subtractores següents:

(a) $T(n) = T(n-1) + \Theta(1)$

(b) $T(n) = T(n-2) + \Theta(1)$

(c) $T(n) = T(n-1) + \Theta(n)$

(d) $T(n) = 2T(n-1) + \Theta(1)$

2.10 Resoleu les recurrències divisores següents:

(a) $T(n) = 2T(n/2) + \Theta(1)$

(b) $T(n) = 2T(n/2) + \Theta(n)$

(c) $T(n) = 2T(n/2) + \Theta(n^2)$

(d) $T(n) = 2T(n/2) + O(1)$

(e) $T(n) = 2T(n/2) + O(n)$

(f) $T(n) = 2T(n/2) + O(n^2)$

(g) $T(n) = 4T(n/2) + n$

(h) $T(n) = 4T(n/2) + n^2$

(i) $T(n) = 4T(n/2) + n^3$

(j) $T(n) = 9T(n/3) + 3n + 2$

(k) $T(n) = T(9n/10) + \Theta(n)$

(l) $T(n) = 2T(n/4) + \sqrt{n}$

2.11 Calcula el cost asimptòtic temporal de la següent acció en funció de $m = p2 - p1 + 1$ que és el nombre de dades del vector V que es processen quan es crida l'acció.

```
void suavitza(float V[MAX], int p1, int p2) {
  int i, j, cont;
  float V2[MAX], sum;
  for (i=p1; i<=p2; i++) {
    sum=0; cont=0;
    for (j=i-3; j<=i+3; j++) {
      if (p1<=j && j<=p2) {
        sum = sum + V[j];
        cont = cont + 1;
      }
    }
    V2[i] = sum / cont;
  }
  for (i=p1; i<=p2; i++) {
    V[i] = V2[i];
  }
}
```

A partir del cost calculat anteriorment calcula el cost asimptòtic temporal de la següent acció en funció de n (la crida inicial és `SuavitzatGradual(V, 0, n-1)`):

```
void SuavitzatGradual(float V[MAX], int p1, int p2) {
  if (p1<p2-1) {
    SuavitzatGradual(V, p1+1, p2-1);
    suavitza(V, p1, p2);
  }
}
```

2.12 Escriu l'equació de recurrència (en funció de n) pels algorismes següents i calcula el cost asimptòtic temporal. Per simplificar l'anàlisi, suposa que n és una potència de 2.

```
int power_1(int x, int n) {
  if (n==1) {
    return x;
  }
  else {
    return x * power_1(x, n-1);
  }
}

int power_2(int x, int n) {
  if (n==1) {
    return x;
  }
  else {
    int half = n/2, halfpower = power_2(x, half);
    if (2*half == n) { // n és parell
      return halfpower * halfpower;
    }
    else { // n és senar
      return halfpower * halfpower * x;
    }
  }
}

int power_3(int x, int n) {
  if (n==1) {
    return x;
  }
  else {
    int half = n/2;
    if (2*half == n) { // n és parell
      return power_3(x, half) * power_3(x, half);
    }
    else { // n és senar
      return power_3(x, half) * power_3(x, half) * x;
    }
  }
}
```

2.13 El cost $T(n)$ d'un algorisme recursiu satisfà $T(n) = xT(n/4) + \Theta(\sqrt{n})$, amb x > 0. Digueu quin és el cost $T(n)$ tenint en compte que hi ha tres situacions possibles, segons l'interval en el qual cau la x.

2.14 Considera el procediment següent (el qual no se sap que fa):

```
void FesFeina(float v[MAX], int i, int j) {
  if (i < j) {
    m = (i + j) / 2;
    QuickSort(v, i, m); // Ordena amb quicksort de v[i] a v[m]
    MergeSort(v, m+1, j); // Ordena amb mergesort de v[m+1] a v[j]
    Processa(v, i, j);  // Processa d'alguna manera v[i] ... v[j]
    FesFeina(v, i, m);
    FesFeina(v, m+1, j);
    FesFeina(v, i, m);
    FesFeina(v, m+1, j);
  }
}
```

Tenint en compte que el cost de `Processa(v,i,j)` és $\Theta((j-i+1)^{1,5})$:

(a) Escriu l'equació de la recurrència de `FesFeina`.

(b) Calcula els costos asimptòtics temporals en els casos pitjor i mig tenint en compte que la crida inicial és `FesFeina(v,1,n)`.

3

Estructures linials estàtiques

3.1 Exercicis

3.1 Implementa la classe genèrica `dues_piles` que emmagatzema de manera conjunta dues piles amb vectors (memòria estàtica).
En la següent especificació no s'inclou el comportament dels mètodes per major brevetat del codi.

```
template <typename T>
class dues_piles {
public:
  dues_piles() throw(error);

  dues_piles(const dues_piles<T> &p) throw(error);
  dues_piles<T>& operator=(const dues_piles<T> &p)
  throw(error);
  ~dues_piles() throw();

  void apilar(nat p, const T &x) throw(error);
  void desapilar(nat p) throw(error);

  const T& cim(nat p) const throw(error);
  bool es_buida(nat p) const throw();
};
```

Com es pot comprovar l'especificació de la classe és l'habitual però s'ha afegit a cada mètode un natural que distingeix la pila de treball (pila 1 o pila 2).
Per implementar aquesta classe cal tenir en compte que degut a que només hi ha canvis en un extrem de la pila (el cim), es pot implementar amb facilitat dues piles en un únic vector si col·loquem les piles enfrontades.

D'aquesta manera creixaran en sentit contrari, l'espai s'aprofita millor i una pila no tindrà espai si realment no hi ha espai lliure dins el vector.

3.2 Teniu una classe llista genèrica amb punt d'interès. Escriviu la implementació en C++ d'una funció ***es_capicua*** que torna cert si i només si la llista es llegeix igual d'esquerra a dreta que de dreta a esquerra.

```
template <typename T>
bool es_capicua(const llista_pi<T>& l) throw();
```

Suposeu que l'operador d'igualtat (==) està definit per als elements de tipus T. Fes servir la següent especificació de la classe en C++:

```
template <typename T>
class llista_pi {
public:
  // Crea una llista buida amb el punt d'interès indefinit.
  llista_pi() throw(error);

  // Tres grans.
  llista_pi(const llista_pi& l) throw(error);
  llista_pi<T>& operator=(const llista_pi& l) throw(error);
  ~llista_pi() throw();

  // Insereix l'element x darrera de l'element apuntat pel PI; si el
  // PI és indefinit, insereix x com primer element de la llista.
  void inserir(const T& x) throw(error);

  // Elimina l'element apuntat pel PI; no fa res si el PI és
  // indefinit; el PI passa a apuntar al successor de l'element
  // eliminat o queda indefinit si l'element eliminat no tenia
  // successor.
  void esborrar() throw();

  // Situa el PI en el primer element de la llista o queda indefinit
  // si la llista és buida.
  void principi() throw();
  void final() throw();

  // Mou el PI al successor/predecessor de l'element apuntat pel PI,
  // quedant el PI indefinit si apuntava a l'últim/primer de la
  // llista; no fa res si el PI estava indefinit.
  void avancar() throw();
  void endarrerir() throw();

  // Retorna l'element apuntat pel PI; llança una excepció si el PI
  // estava indefinit.
  const T& actual() const throw(error);

  // Retorna cert si i només si el PI està indefinit (apunta a
  // l'element "final" fictici).
  bool indef() const throw();
```

```
  // Retorna la longitud de la llista.
  nat longitud() const throw();

  // Retorna cert si només si la llista és buida.
  bool es_buida() const throw();

  // Gestió d'errors.
  static const int PIIndef = 320

};
```

3.3 La classe doble cua (anglès: dequeue) és una classe que permet fer insercions, supressions i consultes en els dos extrems de la cua. És a dir, ha de disposar de les següents operacions:

- *crea()* crea una dequeue buida.
- *inject(e)* insereix un element pel darrera.
- *push(e)* insereix un element pel davant.
- *eject()* treu l'últim element de la dequeue.
- *pop()* treu el primer element de la dequeue.
- *front()* retorna el primer element de la dequeue.
- *rear()* retorna l'últim element de la dequeue.
- *es_buida()* indica si la dequeue és buida.

Dissenya i implementa (usant vectors estàtics) la classe doble cua tenint en compte que les operacions anteriors han de tenir cost constant.

4

Estructures linials dinàmiques

4.1 Exercicis

4.1 Donada la següent representació en C++ d'una llista de nombres enters encadenada:

```
class llista {
private :
  struct node {
    int elem;
    node* seg;
  };
  node* _head;
  ...
};
```

escriu la implementació del mètode `elimina_parells` que elimina de la llista els enters parells

```
void llista::elimina_parells() {
  // Aquí anirà el vostre codi
}
```

tenint en compte:

a) Hi ha element fantasma. L'atribut `_head` apunta al fantasma.

b) No hi ha element fantasma. L'atribut `_head` apunta al primer element de la llista.

4.2 Escriu una funció split en C++ tal que, donada una llista d'enters `L` simplement enllaçada, sense element fantasma i ordenada de forma creixent, i un element `x`, retorna dues llistes `la` i `lb` (inicialment buides), amb els elements estrictament menors que `x` i estrictament majors que `x`, respectivament. La funció deixa la llista `L` original buida. Fes servir les següents definicions:

```
struct node {
  int info;
  node* seg;  // Punter null si el node no té successor
};
typedef node* llista; // Una llista es representa amb un punter al primer element

void split(llista& L, int x, llista& la, llista& lb);
```

4.3 Tens una classe llista genèrica amb iteradors. Escriu la implementació en C++ d'una funció ***es_capicua*** que torna cert si i només si la llista es llegeix igual d'esquerra a dreta que de dreta a esquerra.

```
template <typename T>
bool es_capicua(const llista_itr<T>& l) throw();
```

Suposa que l'operador d'igualtat (==) està definit per als elements de tipus T. Fes servir la següent especificació de la classe `llista_itr`:

```
template <class T>
class llista_itr {
public:
  // Constructora. Crea una llista buida.
  llista_itr() throw(error);

  // Tres grans.
  llista_itr(const llista_itr& l) throw(error);
  llista_itr<T>& operator=(const llista_itr& l) throw(error);
  ~llista_itr() throw();

  // Iteradors sobre llistes. Un objecte iterador sempre està
  // associat a una llista particular; només poden ser creats
  // mitjançant els mètodes Llista<T>::primer, Llista<T>::ultim i
  // Llista<T>::indef.
  // Cada llista té el seu iterador indefinit propi:
  //       L1.indef() != L2.indef()
  friend class iterador;

  class iterador {
  public:
    friend class llista_itr;
    // Accedeix a l'element apuntat per l'iterador o llança una
    // excepció si l'iterador és indefinit.
    const T& operator*() const throw(error);

    // Operadors per avançar (pre i postincrement) i per retrocedir
    // (pre i postdecrement); no fan res si l'iterador és indefinit.
```

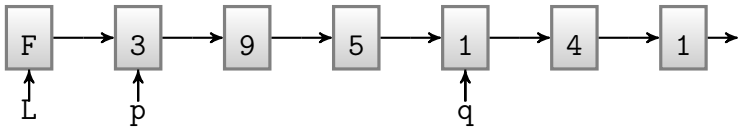
```
    iterador& operator++() throw();
    iterador operator++(int) throw();
    iterador& operator--() throw();
    iterador operator--(int) throw();

    // Operadors d'igualtat i desigualtat entre iteradors.
    bool operator==(const iterador& it) const throw();
    bool operator!=(const iterador& it) const throw();

    static const int IteradorIndef = 330;
  };

  // Insereix l'element x darrera/davant de l'element apuntat per
  // l'iterador; si l'iterador it és indefinit, insereix_darrera
  // afegeix x com primer de la llista, i insereix_davant afegeix x
  // com últim de la llista; en abstracte un iterador indefinit
  // "apunta" a un element fictici que és al mateix temps
  // predecessor del primer i successor de l'últim.
  void inserir_darrera(const T& x, iterador it) throw(error);
  void inserir_davant(const T& x, iterador it) throw(error);

  // Tant esborra_avnc com esborra_darr eliminen l'element apuntat
  // per l'iterador, menys en el cas que it és indefinit (llavors no
  // fan res); amb esborra_avnc l'iterador passa a apuntar al
  // successor de l'element eliminat o queda indefinit si l'element
  // eliminat és l'últim; amb esborra_darr l'iterador passa a apuntar
  // al predecessor de l'element eliminat o queda indefinit si
  // l'eliminat és el primer element.
  void esborrar_avnc(iterador& it) throw();
  void esborrar_darr(iterador& it) throw();

  // Retorna la longitud de la llista.
  nat longitud() const throw();

  // Retorna cert si i només si la llista és buida.
  bool es_buida() const throw();

  // Retorna un iterador al primer/últim element o un iterador
  // indefinit si la llista és buida.
  iterador primer() const throw();
  iterador ultim() const throw();

  // Retorna un iterador indefinit.
  iterador indef() const throw();
};
```

4.4 Invertir

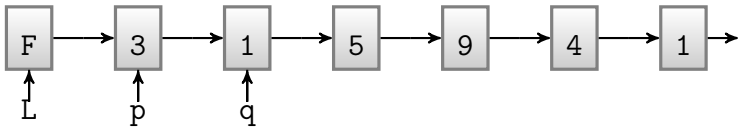
a) Dissenya i codifica una acció `invertir` que donada una llista simplement enllaçada L amb un element fantasma inicial i dos apuntadors p i q, inverteix la subllista formada pels elements compresos entre el següent de l'element apuntat per p i l'element apuntat per q. Si $p = q$ llavors la subllista és buida i l'acció no fa res. Pots assumir com a

precondició que ni p ni q són nuls, que si L no és buida llavors p no apunta a l'últim element de L i que q no apunta a un element anterior a l'apuntat per p.

Per exemple, donada la llista L següent:



el resultat de l'operació `invertir(p, q)` és el següent:



Treballa amb la definició:

```
struct node {
  int info;
  node* seg;  // Punter nul si el node no té successor
};
typedef node* llista; // La llista es representa amb un punter al fantasma
```

b) Dissenya i codifica el mètode `invertir` per una classe llista. que emmagatzema una llista doblement enllaçada sense element fantasma. Treballa amb la definició:

```
template <typename T>
class llista {
private:
  struct node {
     T info;
     node* seg; // Punter nul si el node no té successor
     node* ant; // Punter nul si el node no té antecessor
  };
  node* _head;  // la llista es representa amb un punter al primer element

  void invertir(node* p, node* q);
};
```

4.5 Escriu una funció `promote` en C++ tal que, donada una llista L amb elements diferents, un element $x \in L$ i un enter $k > 0$, mou l'element x k posicions cap al front de la llista, o mou x al principi de la llista si x té menys de k elements al davant.
Per exemple, donats $x = 3$, $k = 4$ i $L = [2,5,1,6,8,3,7,4,10,9]$ la llista resultant és $L' = [2,3,5,1,6,8,7,4,10,9]$. Si tornéssim a aplicar l'algorisme sobre L' amb els mateixos x i k obtindríem $L'' = [3,2,5,1,6,8,7,4,10,9]$. La teva implementació no pot fer servir cap llista o vector auxiliar. Resolt l'exercici tenint en compte les següents definicions:

a) llista doblement enllaçada amb element fantasma inicial.

```
struct node {
  int info;
  node* ant;
  node* seg;
};
typedef node* llista; // Una llista es representa amb un punter al primer
            // element

void promote(llista& L, int x, int k);
```

b) llista simplement enllaçada amb element fantasma inicial.

```
struct node {
  int info;
  node* seg;  // Punter nul si el node no té successor
};
typedef node* llista; // Una llista es representa amb un punter al primer
            // element

void promote(llista& L, int x, int k);
```

4.6 Tens una classe llista genèrica implementada amb llistes doblement enllaçades, tancades circularment i sense element fantasma. Escriu la implementació en C++ d'un mètode `es_capicua` que torna cert si i només si la llista es llegeix igual d'esquerra a dreta que de dreta a esquerra. Suposa que l'operador d'igualtat (==) està definit per als elements de tipus T. Fes servir les següents definicions en C++:

```
template<typename T>
class llista {
public:
   ...
   bool es_capicua() const;
   ...
private:
   struct node {
      T info;
      node* seg;
      node* prev;
   };
   node* _primer; // Apunta al primer element o és NULL si la llista és buida
}

template<typename T>
bool llista<T>::es_capicua() const {
   // aquí anirà el vostre codi
}
```

4.7 Tens una llista L simplement enllaçada, sense element fantasma, els elements de la qual contenen un camp color que pot ser 0 (blanc), 1 (vermell), ó 2 (blau), a més d'altres camps, d'acord amb la següent definició:

```
struct node {
  int color;
  ...   // Altres camps
  node* seg;  // El punter es nul si el node no te successor
};
typedef node* llista; // Una llista es representa amb un punter al primer element
```

Escriu un procediment en C++ anomenat `bandera_holandesa` que donada una llista L, la reorganitza de tal manera que primer estiguin tots els elements blaus, després tots els elements blancs i finalment tots els elements vermells. Òbviament, poden no existir elements d'un o més dels colors. La teva implementació no pot crear ni destruir nodes, i no es poden copiar els camps d'un node en els d'un altre: cal reorganitzar la llista, modificant els enllaços.

4.8 Escriu en C++ un mètode consultor ***tria*** d'una classe llista d'enters que donat un enter x, retorni una altra llista amb tots els elements més grans que x de la llista en curs. La llista retornada ha d'estar ordenada creixentment, i ha d'incloure els elements repetits un sol cop. L'especificació d'aquesta classe llista és la següent:

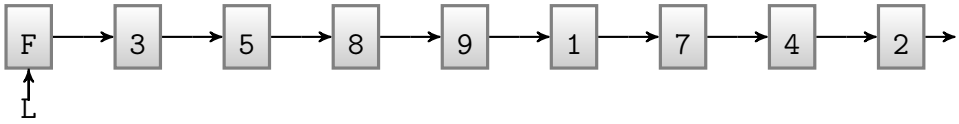
```
class llista {
private:
  struct node {
    int info;
    node* seg;
  };

  node* _primer;

public:
  ...
  llista tria(int x) const throw(error);
  ...
};
```

Les llistes són simplement enllaçades i tenen un atribut privat que apunta al primer element (les llistes no tenen element fanstama). Podeu usar (implementant-los, és clar) mètodes privats auxiliars si us calen.

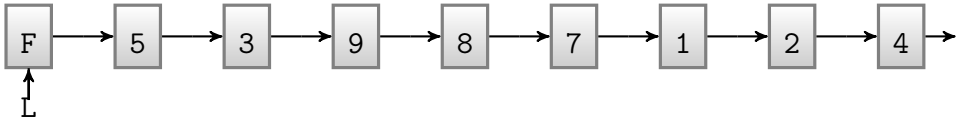
4.9 Dissenya i codifica una acció `invertir_parelles` que donada una llista simplement enllaçada L (dóna la representació del tipus) amb un element fantasma inicial, inverteix els elements dos a dos. Pots assumir com a precondició que la llista o bé és buida o bé conté un nombre parell d'elements.
Hi ha una sèrie de restriccions a tenir en compte a l'hora de codificar l'acció:

- No es pot demanar memòria dinàmica, així doncs la llista resultant no és una llista nova, sinó que sobre la llista donada hem de deixar el resultat.
- No es pot modificar la informació del nodes. Això vol dir que no es pot intercanviar la informació dels elements.

Exemple: Donada la llista següent (L apunta al fantasma F):



El resultat de l'acció `invertir_parelles` és:



4.10 Implementa la classe taules de freqüències mitjançant llistes enllaçades autoorganitzades. Aquesta classe tindrà les següents operacions:

- *crea()* crea una taula de freqüències buida.
- *incr(cl)* insereix la clau amb frequència 1. Si ja hi era incrementa la freqüència.
- *freq(cl)* retorna la freqüència de la clau. Si no hi era retorna 0.

Per tal d'implementar aquesta classe segueix la següent estratègia sobre llistes autoorganitzades:

- quan es fa una inserció situa l'element al principi de la llista.
- quan es fa una consulta desplaça l'element fins al principi de la llista.

En cap dels dos casos s'altera l'ordre relatiu dels altres elements.

4.11 Diu la llegenda que al segle I, l'historiador Josephus es va trobar, juntament amb altres 40 soldats jueus, assetjat per l'exèrcit romà. La situació era tan desesperada que van preferir treure's la vida abans de ser esclaus de Roma. Van decidir posar-se en cercle i anar eliminant una de cada tres persones fins que només en quedés una (que suposadament s'hauria de suïcidar). En Josephus, que volia viure, va calcular ràpidament on posar-se i va ser l'únic supervivent.
Sabent el nombre n de persones i el nombre k del comptador el nostre objectiu és calcular quina serà la persona supervivent. Per resoldre aquest problema utilitzarem una llista circular, simplement encadenada i sense element fantasma, inicialitzada amb els elements 1,2,3,...,n. Escriu en C++ la funció llista tal que, donat el nombre n de persones, construeixi la llista inicial (una llista amb n elements on que cada element conté un enter que enumera la seva posició) i retorni el punter al primer element de la llista.

```
typedef node* punter;
struct node {
  int el;
  punter seg;
};

punter llista(int n) {
  ...
}
```

Després implementa la funció `Josephus` que, donats el nombre n de persones i el nombre k del comptador, crei la llista inicial amb n elements i després elimini un element de cada k elements fins que en quedi només un. Aquest és el supervivent.
Per exemple, si n=5 i k=2 la llista tindria els següents elements

```
1,2,3,4,5
1,3,4,5
1,3,5
3,5
3
```

i el resultat de la funció `Josephus` seria 3.

```
int Josephus(int n, int k) {
  punter p = llista(n);
  ...
}
```

5

Arbres

5.1 Exercicis

5.1 Implementa una funció que calculi l'alçada d'un arbre binari donat. L'alçada d'un arbre és el màxim de les alçades dels seus subarbres més 1.
NOTA: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.2 Implementa una funció que compti el nombre de nodes d'un arbre binari donat.
NOTA: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.3 Implementa una funció que calculi el nombre de nodes amb tots dos subarbres buits d'un arbre binari donat.
NOTA: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.4 Implementeu una funció que donats arbres binaris d'enters, retorna si el primer arbre té més elements que el segon.

```
bool mes_gran(const Abin<int>& a, const Abin<int>& b) throw();
```

NOTA: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.5 Dissenya una funció que, donat un enter n i un arbre general en el que els seus elements són enters, retorni el nombre d'elements de l'arbre que són més petits que n.
NOTA: L'especificació de la classe *Arbre general* és la que apareix en els apunts de teoria d'ESIN.

5.6 Dissenya una funció que a partir d'un arbre general retorni el nombre de fulles (nombre de nodes de grau 0) que té.
NOTA: L'especificació de la classe *Arbre general* és la que apareix en els apunts de teoria d'ESIN.

5.7 Reconstrucció arbre binari (dibuix)

a) Dibuixa l'arbre binari que té com a recorreguts les següents seqüències:

- Preordre: 8, 4, 5, 3, 7, 10, 9, 6
- Inordre: 3, 5, 7, 4, 8, 9, 10, 6

b) Hi ha alguna manera de reconstruir un arbre binari si els recorreguts donats són el preordre i el postordre? Raona la teva resposta.

5.8 Dissenya un algorisme que reconstrueixi un arbre binari donats els seus recorreguts (en forma de llista d'elements) en preordre i en inordre. Tots els elements són diferents. La llista d'elements és la llista de STL.
Podeu suposar que les llistes són sempre correctes, és a dir, permeten reconstruir un arbre binari i per tant no hi ha situacions d'error.

```
template <typename T>
Abin<T> reconstruir(const list<T>& pre, const list<T>& in) throw(error);
```

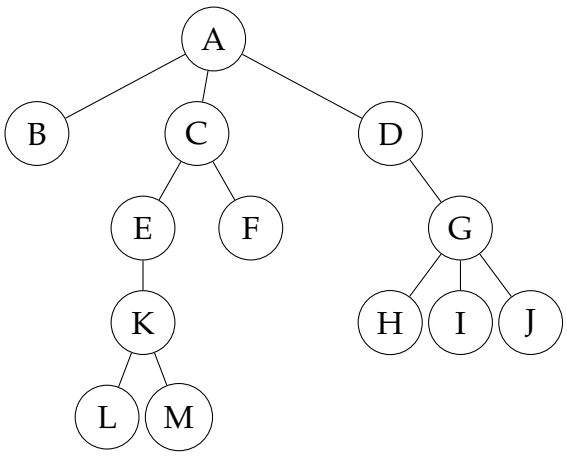
NOTA: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.9 Escriu una funció que donat un arbre binari d'enters indiqui si tots els seus nodes compleixen que el seu valor és major o igual que la suma dels elements dels seus dos subarbres.

```
bool nodes_mes_grans_que_suma(const Abin<int> &a);
```

NOTA: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.10 Dibuixa l'arbre binari que s'obté al representar el següent arbre *n*-ari mitjançant un arbre binari.



5.11 Enriquiu la classe `Abin` amb un mètode que indiqui l'alçada d'un arbre binari,

```
template <class T>
class Abin {
  public:
    ...
    int alcada() const throw();
    ...
};
```

NOTA: La representació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.12 Enriquiu la classe `Abin` amb un mètode que indiqui el nombre de nodes d'un arbre binari.
NOTA: La representació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.13 Enriquiu la classe `Abin` amb un mètode que indiqui el nombre amb tots dos subarbres buits d'un arbre binari.
NOTA: La representació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.14 Enriquiu la classe `Abin` amb un mètode que indiqui si donats dos arbres binaris el primer arbre té més elements que el segon.

```
template <typename T>
class Abin {
  public:
```

```
    ...
    bool operator>(const Abin<T>& b) const throw();
    ...
};
```

NOTA: La representació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

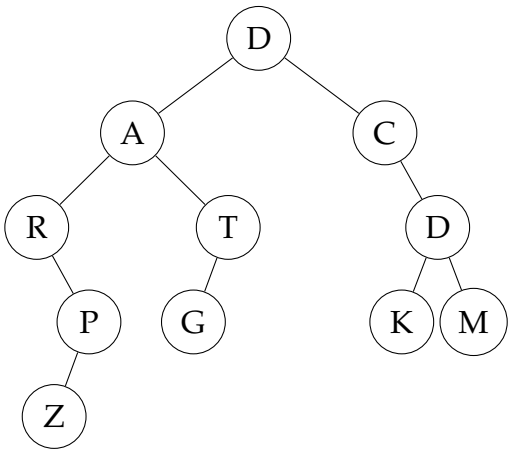
5.15 Dissenya un algorisme que donat un arbre general retorni una llista amb tots els camins des de l'arrel fins a les fulles. Un camí des de l'arrel fins a una fulla és una seqüència de tots els nodes que el formen.
Un camí el definirem com:

```
template <typename T>
typedef list<T> cami;
```

NOTA: L'especificació de la classe *Arbre general* és la que apareix en els apunts de teoria d'ESIN.

5.16 Arbres enfilats

a) Enfila el següent arbre binari.

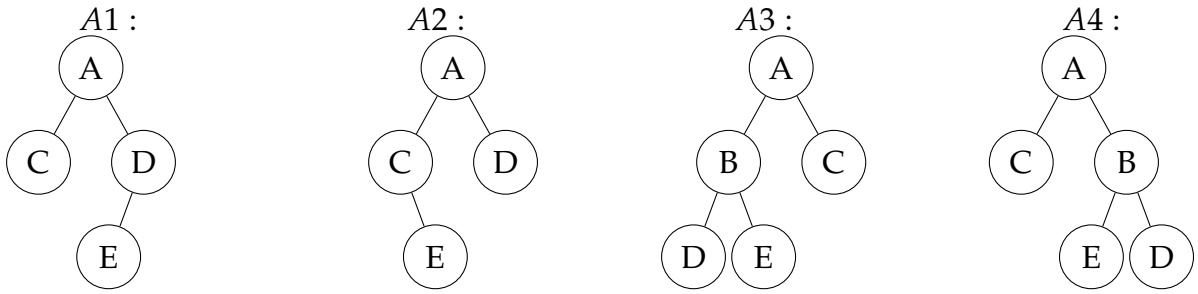


b) Perquè serveix enfilar-ho? És necessari memòria addicional per enfilar-ho? Raona les respostes.

5.17 **Arbres mirall**. Implementa una funció que, donat dos arbres binaris, indiqui si el primer arbre és l'arbre mirall del segon. La capçalera d'aquesta funció és la següent:

```
template <typename T>
bool mirall (const Abin<T> &a, const Abin<T> &b) throw();
```

Exemple: Donats els següents arbres binaris:



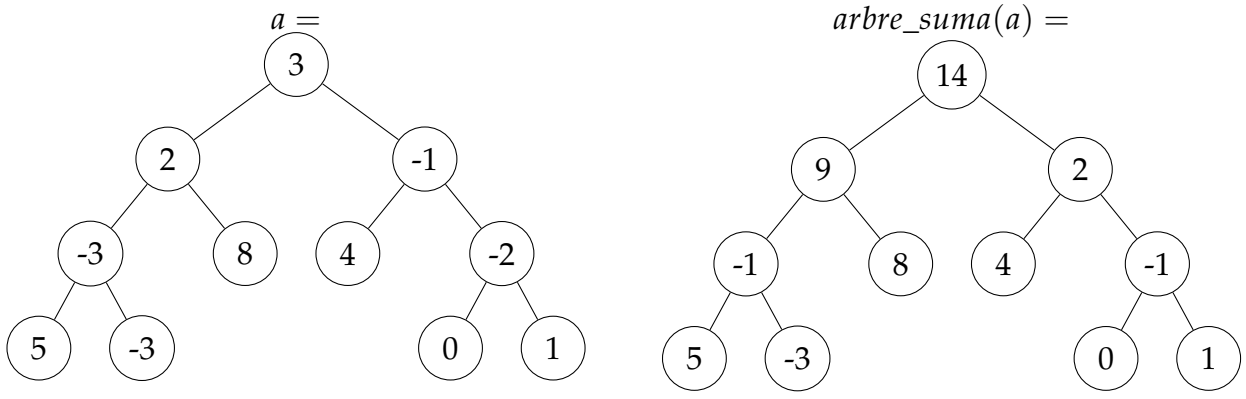
El resultat de les següents crides de la funció mirall amb els diferents arbres seria:

```
mirall(A1, A2) == false
mirall(A1, A3) == false
mirall(A1, A4) == false
mirall(A2, A1) == false
mirall(A2, A3) == false
mirall(A2, A4) == false
```

```
mirall(A3, A1) == false
mirall(A3, A2) == false
mirall(A3, A4) == true
mirall(A4, A1) == false
mirall(A4, A2) == false
mirall(A4, A3) == true
```

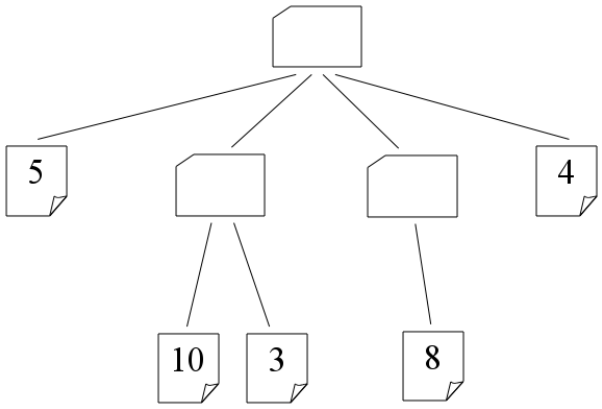
NOTA: Disposes de les classes `Abin<T>` i iterador amb els mètodes públics vistos a classe.

5.18 L'arbre de sumes d'un arbre *a* és un altre arbre amb la mateixa estructura, tal que cada node del nou arbre conté la suma dels seus descendents del node corresponent en l'arbre *a*, incloent aquest mateix. Per exemple:



Implementa un mètode de la classe `Abin` que retorni l'arbre suma corresponent.
NOTA: La representació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.

5.19 Sistema de fitxers.
Considera que s'utilitza un arbre general amb accés per primer fill/següent germà per emmagatzemar l'estructura jeràrquica d'un sistema de fitxers com el d'aquesta figura:



Cada carpeta o subdirectori pot contenir (tenir com a fills) un nombre indeterminat d'altres carpetes i fitxers. Els fitxers no contenen res (són les fulles de l'arbre, per simplicitat suposem que les carpetes no poden ser buides).
Disposes de la classe `acafi` (arbre de carpetes i fitxers) amb els mètodes públics llistats a continuació:

```
class acafi {
public:
  // Construeix un arbre a partir d'una carpeta/fitxer i una llista d'arbres
  acafi (elem e, const llista_acafi &l);

  // Retorna la carpeta o fitxer de l'arrel
  elem arrel () const;

  // Retorna la grandària de la carpeta o fitxer de l'arrel
  int get_size () const;

  // Fixa la grandària de la carpeta o fitxer de l'arrel amb el valor s
  void set_size (int s);

  // Retorna el primer fill de l'arrel
  acafi prim_fill () const;

  // Retorna el següent germà de l'arrel
  acafi seg_germà () const;

  // Indica si existeix primer fill
  bool hi_ha_prim_fill () const;

  // Indica si existeix següent germà
  bool hi_ha_seg_germà () const;
};
```

a) Implementa la següent acció que calcula al mateix temps el nombre de carpetes i de fitxers que conté l'arbre:

```
void quants (const acafi &a, int &nc, int &nf)
```

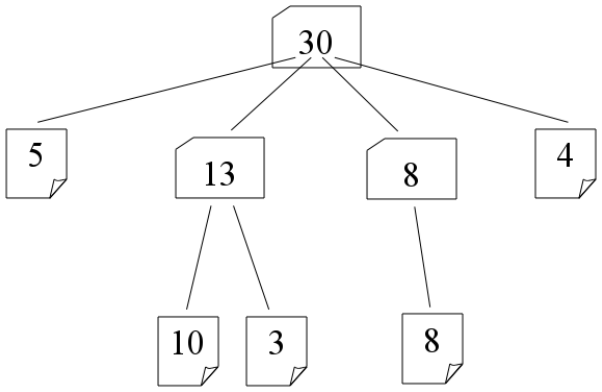
La crida inicial a aquesta acció seria:

```
int nc=0, nf=0;
quants (a, nc, nf);
```

b) Implementa la següent funció que retorna l'ocupació de memòria de l'arbre:

```
int size (acafi &a)
```

Considera que les carpetes individualment ocupen 0 bytes i els fitxers el valor que retorni el mètode `get_size`. A més a més aquesta funció ha d'actualitzar (amb el mètode `set_size`) l'ocupació de memòria de cada carpeta (que és la suma de tots els fitxers que pengen d'ella). Per exemple, en el sistema de fitxers de la figura anterior el resultat seria 30 i l'ocupació de les carpetes s'haurien actualitzat tal com mostra a continuació:



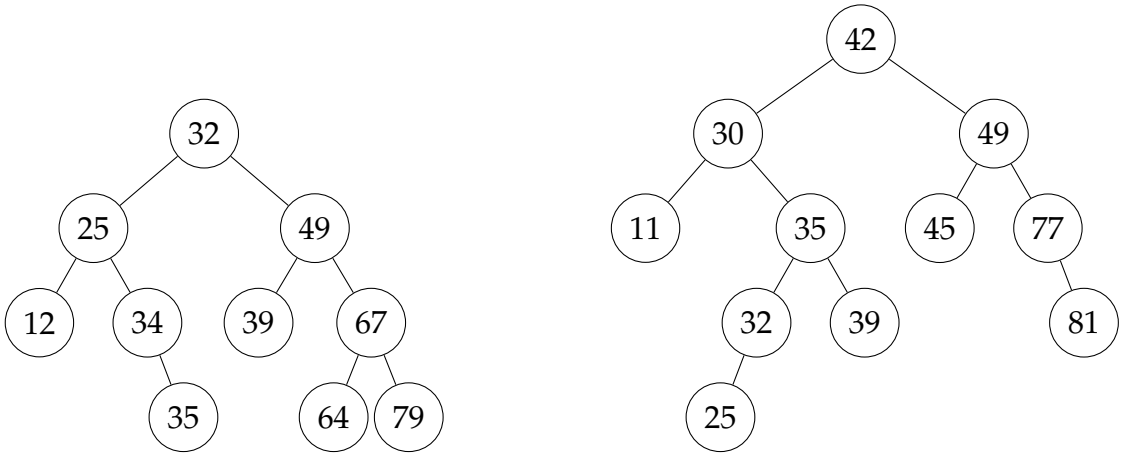
c) Quins tipus de recorreguts has utilitzat per implementar els apartats anteriors? Quins costos tenen en funció del nombre de nodes de l'arbre?

6

Diccionaris

6.1 Arbres binaris de cerca

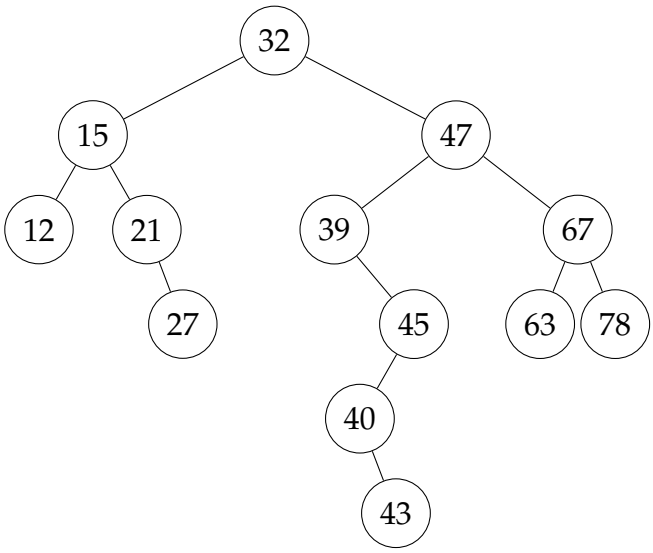
6.1 Digues si els arbres següents són arbres binaris de cerca (ABC) o no. Raona la teva resposta.



6.2 Partint d'un arbre binari de cerca buit, mostra l'ABC resultat inserir amb l'algorisme clàssic, l'una rera l'altra, la seqüència de claus 32, 15, 47, 67, 78, 39, 63, 45, 21, 12, 40, 27, 43.

6.3 Partint d'un arbre binari de cerca buit, mostra l'ABC resultat d'inserir amb l'algorisme clàssic, l'una rera l'altra, la seqüència de claus 12, 15, 21, 27, 32, 39, 47, 63, 67, 78.

6.4 Partint de l'arbre binari de cerca següent, mostra l'ABC resultat d'eliminar les claus 63, 21, 15, 32 l'una rera l'altra. Usa la política del succesor per eliminar les claus.



6.5 Partint d'un ABC buit, mostra l'ABC resultant després de fer cadascuna de les insercions/eliminacions següents:

a) Insereix de manera incremental les següents claus: 18, 4, 11, 2, 19, 7, 50, 15, 30, 20, 14, 13, 34

b) A partir de l'últim arbre de l'apartat anterior elimina de manera consecutiva les següents claus: 2, 30, 18.

NOTA: Per eliminar claus usa la política del predecesor.

6.6 Escriu un mètode de la classe arbre binari que comprovi si un arbre binari és un arbre binari de cerca (ABC).
NOTA: Per resoldre aquest exercici heu de fer servir la representació de la classe `Abin<T>` amb memòria que apareix en els apunts de teoria d'ESIN.

6.7 Mostra l'arbre binari de cerca d'enters el recorregut en preordre del qual és:
8, 5, 2, 1, 4, 3, 6, 7, 11, 9, 13, 12

6.8 Implementa una funció que donada una llista d'elements que conté el recorregut en preordre d'un arbre binari de cerca retorni l'arbre binari corresponent.
NOTA 1: L'especificació de la classe `Abin<T>` és la que apareix en els apunts de teoria d'ESIN.
NOTA 2: Considera que la llista d'enters és la classe `list` de la biblioteca STL.
NOTA 3: Considera que l'element del que està formada la llista té implementats els mètodes de comparació (==, <, ...).

- Pels següents exercicis la classe `dicc` està implementada usant un ABC. Aquesta és la representació:

```
template <typename Elem>
class dicc {
private:
  struct node {
    Elem _k;
    node* _esq;   // fill esquerre
    node* _dret;  // fill dret
  };
  node *_arrel;
  ...
}
```

6.9 Implementa el mètode constructor de la classe `dicc` que donada una llista d'elements que conté el recorregut en preordre d'un arbre binari de cerca construeixi l'ABC corresponent.
NOTA 1: Considera que la llista d'enters és la classe `list` de la biblioteca STL.
NOTA 2: Considera que l'element del que està formada la llista té implementats els mètodes de comparació (==, <, ...).

6.10

a) Implementa un mètode de la classe `dicc` que donat un element *e* retorna el nombre d'elements de la classe que són més petits que *e*. Podeu suposar que el tipus genèric `Elem` disposa de l'operador menor.

b) Quin és el cost d'aquest mètode?

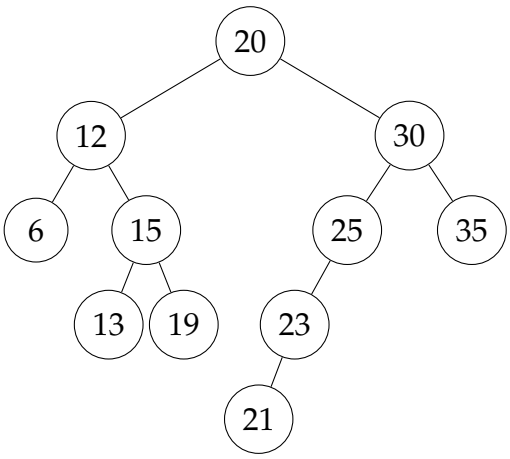
6.11 La fusió de dos arbres binaris de cerca és un arbre binari de cerca que conté tots els elements dels dos arbres.

a) Implementa un mètode de la classe `dicc` que a partir de dos `dicc` retorni la fusió dels dos arbres binaris de cerca. Els dos arbres originals han de quedar buits després de cridar el mètode.

b) Quin és el cost d'aquest mètode?

6.12

a) Dissenya un mètode de la classe que, donats dos elements $x1$ i $x2$ sent $x1 \leq x2$, retorni una llista amb tots els elements de `dicc` que es trobin entre $x1$ i $x2$ en ordre decreixent, ambdós inclosos. Per exemple, donat l'arbre:



i els valors 18 i 25, cal retornar la llista $< 25, 23, 21, 20, 19 >$.

NOTA: Podeu suposar que el tipus genèric `Elem` disposa dels operadors de comparació.

b) Quin és el cost d'aquest mètode en el cas pitjor?

6.13 En els arbres binaris de cerca (ABC o en anglès BST) existeixen moltes operacions (inserció, esborrat, mínim, màxim, existeix?, ...) que tenen un cost que depèn de l'alçada de l'arbre, que en el cas pitjor és igual al número d'elements. Podríem resoldre aquest problema del cost lineal en els ABC amb una altra aproximació: usar un nou tipus, ABCC (arbres binaris de cerca complets). Els ABCC serien com els ABC però a més amb la condició de que siguin arbres binaris complets. Així s'aconseguiria:

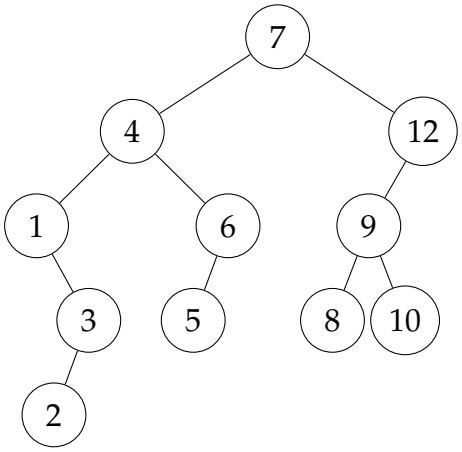
a) Despesa d'espai bona: al ser complets es poden emmagatzemar en un vector sense usar cap encadenament i sense perdre espai.

b) Les operacions anteriors tindrien cost logarítmic, ja que tenim l'ordenació total de l'ABC i alçada logarítmica.

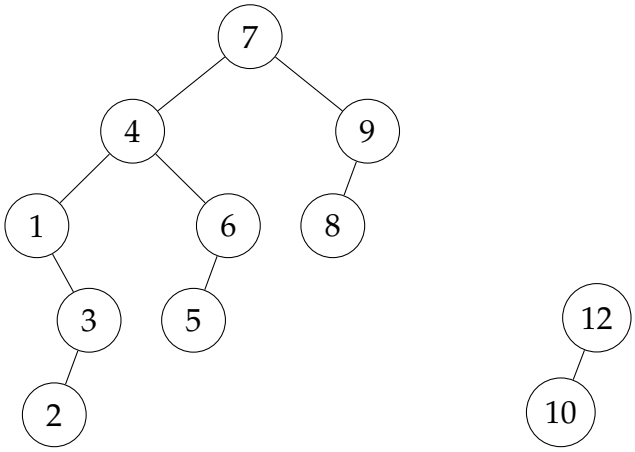
Són certes les afirmacions a) i b) i per tant els ABCC ens podrien ser útils? Justifica la teva resposta.

6.14

a) Implementa un mètode que donat un element (és una clau) x, retorni els dos `diccs` d^- i d^+, on d^- conté totes les claus menors o igual que x i d^+ conté totes les claus majors que x. Per exemple donat l'arbre següent:



i x=*9*, els arbres d^- i d^+ són els següents:



6.15

a) Modifica la representació dels ABC per tal de suportar eficientment cerques i esborrats per rang; és a dir, donat un natural i, localitzar o esborrar l'element i-èssim en ordre ascendent. Per exemple, si $i = 1$, es tractaria de localitzar o esborrar el mínim; si $i = n$, s'hauria de localitzar o esborrar el màxim, essent n el nombre d'elements de l'ABC.

b) Implementa l'algorisme de cerca per rang.

c) Modifica l'algorisme d'inserció perquè es mantingui correctament la nova representació.

Pista: En els apunts de teoria s'explica com es pot aconseguir de forma eficient, guardant dins de cada node el nombre d'elements que conté el subarbre corresponent.

6.2 AVL

6.16 Partint d'un AVL buit, mostra l'AVL resultat d'inserir amb l'algorisme clàssic, l'una rera l'altra, la seqüència de claus 32, 15, 47, 67, 78, 39, 63, 21, 12, 27.

6.17 Partint d'un AVL buit, mostra l'AVL resultant d'inserir amb l'algorisme clàssic, l'una rera l'altra, la seqüència de claus 12, 15, 21, 27, 32, 39, 47, 63, 67, 78.

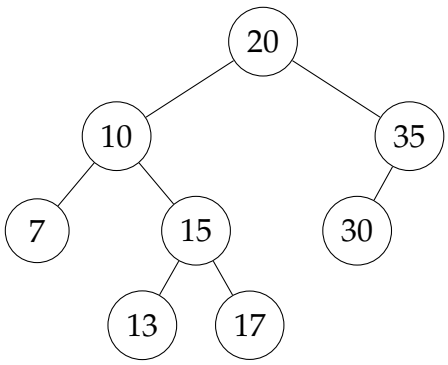
6.18 Partint d'un AVL buit, mostra l'AVL resultant després de fer cadascuna de les insercions/eliminacions següents:

a) Insereix de manera incremental les següents claus: 18, 4, 11, 2, 19, 7, 50, 15, 30, 20, 14, 13, 34

b) A partir de l'últim arbre de l'apartat anterior elimina de manera consecutiva les següents claus: 2, 30, 18.

NOTA: Per eliminar claus usa la política del predecesor.

6.19 Donat l'arbre binari de cerca equilibrat (AVL) de la figura, insereix els elements 18 i 12 i esborra els elements 7 i 30 sempre partint de l'arbre de la figura. Mostra l'evolució de l'arbre pas a pas a cada inserció/esborrat.



6.20 Implementa un mètode de la classe arbre binari que retorna cert si i només si l'arbre binari donat satisfà la propietat d'equilibrat característica dels AVLs.
La representació consta únicament d'un punter al node arrel. Els nodes de l'arbre no contenen informació d'equilibri ni es pot modificar la seva representació: cada node conté només un element (de tipus T) i dos apuntadors als fills esquerre i dret, respectivament (node* fesq, fdret). Es valorarà que el cost de la funció sigui $\Theta(n)$ on n és la grandària de l'arbre donat.

6.3 Taules de dispersió

6.21 Considera una taula de dispersió d'encadenament indirectes amb $M = 10$ posicions per a claus enteres i funció de dispersió *h(x) = x mod M*.

a) Començant amb una taula buida, mostra com queda després d'inserir les claus 3441, 3412, 498, 1983, 4893, 3874, 3722, 3313, 4830, 2001, 3202, 365, 128181, 7812, 1299, 999 i 18267.

b) Mostra com queda la taula de dispersió anterior quan s'aplica una redispersió per enmagatzemar 20 posicions.

6.22 Construeix la taula de dispersió d'encadenament indirectes per a les claus 30, 20, 56, 75, 31 i 19, amb 11 posicions i funció de dispersió *h(x) = x mod 11*.

a) Calcula el nombre esperat de comparacions en una cerca amb èxit en aquesta taula.

b) Calcula el nombre màxim de comparacions en una cerca amb èxit en aquesta taula.

6.23 Considera una taula de dispersió d'encadenament directes amb $MAX = 10$ posicions per a claus enteres i funció de dispersió *h(x) = x mod M* (on M és la mida de la zona principal). De l'espai total de la taula es reserva un 20% per la zona d'excedents.
Començant amb una taula buida, mostra com queda després d'inserir les claus 3441, 3412, 498, 1983, 4893, 3874, 3722, 3313, 4830, 2001, 64 i 3202.
En cas que la zona d'excedents s'ompli cal fer redispersió sobre la taula duplicant la mida de la taula.

6.24 Tenim implementat un diccionari amb una taula de dispersió amb n valors de dispersió implementada amb la tècnica d'adreçament obert usant l'estratègia del sondeig lineal. Fes un esquema de l'estructura de dades i indica la seva evolució si, partint del diccionari buit, s'insereixen i es suprimeixen claus (les claus són cadenes de caràcters) en aquest ordre:

```
insereix("OPEL")
insereix("FORD")
insereix("SEAT")
esborra("FORD")
insereix("FIAT")
```

Els valors de dispersió que dóna la funció de dispersió sobre les claus anteriors són:

```
h("OPEL") = 0        h("FORD") = 1
h("SEAT") = 0        h("FIAT") = 0
```

6.25 Donada una taula de dispersió inicialment buida en la qual s'insereixen les següents claus 4376, 1323, 6178, 4193, 14, 238, 2115 en aquest ordre. La mida de la taula és de $M = 5$ posicions i la funció de dispersió $h(x) = x \bmod M$
Mostra el contingut de la taula després de cada inserció tenint en compte que la tècnica de resolució de col·lisions és:

a) Encadenaments indirectes.

b) Adreçament obert amb sondeig lineal.

NOTA: En cas necessari fes redispersió sobre la taula.

6.26 Implementa l'operació `elimina` per les taules de dispersió encadenades indirectes que elimina la parella <clau, valor> del diccionari:

```
template <typename Clau, typename Valor>
void dicc<Clau, Valor>::elimina (const Clau &k) throw();
```

Suposa que ja tens definida la funció de dispersió `h()` que retorna valors de dispersió dins del rang `[0, M-1]`. La representació de les taules de dispersió encadenades indirectes és la següent:

```
template <typename Clau, typename Valor>
class dicc {
  ...
private:
  struct node_hash {
    Clau _k;
    Valor _v;
    node_hash* _seg;
```

```
    };
    node_hash **_taula;  // taula amb punters a les llistes
    nat _M;  // mida de la taula
    nat _quants;  // nº d'elements guardats al diccionari
    ...
  };
```

6.27 Taules de dispersió amb encadenaments directes.

a) Escriu la representació de la classe `dicc<Clau,Valor>` implementada amb una taula de dispersió amb encadenaments directes.

b) Implementa els mètodes `insereix` i `consulta` d'aquesta classe `dicc`.

NOTA: Suposa que ja tens definida la funció de dispersió `h()` que retorna valors de dispersió dins del rang `[0, M-1]`.

6.28 Implementa un mètode d'una classe llista d'`string`'s que faci la intersecció de dues llistes desordenades amb **cost màxim linial**.

```
class llista {
public:
   ...
   void intersectar(const llista& lst) throw(error);
   ...

private:
   // Llista simplement enllaçada amb element fantasma
   struct node {
      node* seg;
      string info;
   };
   node* _head;
   nat _size;
};
```

Pista: Per resoldre l'exercici podries usar un diccionari implementat amb una taula de dispersió amb adreçament obert i sondeig lineal.

6.4 Algorismes d'ordenació

6.29 Dibuixa quicksort.
Es vol ordenar el següent vector de menor a major amb l'algorisme de Quicksort.

3	4	7	8	5	2

Dibuixa l'evolució del vector pas a pas. Agafa com a pivot el primer element del vector (subvector) desordenat.

6.30 Dibuixa i cost Quicksort.

a) Ordenar de menor a major aquests 8 elements amb l'algorisme d'ordenació QuickSort, indicant la seva evolució pas a pas (agafa com a pivot el primer element del subvector a ordenar): `5 7 8 3 4 6 9 1`

b) Indica quin cost té aquest algorisme en el cas mig i en el cas pitjor en funció dels n elements a ordenar.

6.31 Escriu en notació algorísmica que fa l'acció principal de l'algorisme de Quicksort; *ini* i *fi* són respectivament les posicions inicials i finals del subvector a ordenar.

```
template <typename T>
void quicksort(T A[], nat ini, nat fi);
```

6.32 Donat aquest vector:

384	170	45	75	90	802	2	24	66	321

es vol ordenar de menor a major seguint l'algorisme d'ordenació:

a) Mergesort

b) Quicksort (agafa com a pivot el primer element del vector (subvector) desordenat).

c) Radixsort LSD (Least Significant Digit).

Per cada algorisme dibuixa l'evolució del vector pas a pas.

6.33 Implementa l'algorisme d'ordenacio de Radixsort LSD per un array d'un tipus genèric `T`.
NOTA: Pots suposar que el tipus `T` disposa del constructor per còpia, operador assignació, destructor i els operadors de comparació.

6.5 Tries i TST

6.34 Donades les claus següents: CEP, CASA, CAPA, COPA, PA, DIT, DOL, AU, BOL, CAP

a) Dibuixa el trie resultat d'inserir les claus anteriors en un arbre inicialment buit. Fes-ho en el cas que el trie es representi amb un arbre general segons la tècnica de primer fill–següent germà.

b) Dibuixa l'arbre ternari de cerca resultat d'inserir les claus anteriors en un arbre inicialment buit.

6.35 Dibuixa el resultat d'inserir els strings `'MARIA'`, `'MANEL'` i `'MAR'` en aquest ordre en un arbre ternari de cerca inicialment buit. Dibuixa cadascun d'aquests tres arbres:

a) amb `'MARIA'`

b) amb `'MARIA'` i `'MANEL'`

c) amb `'MARIA'`, `'MANEL'` i `'MAR'`

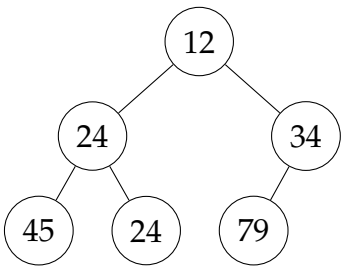
6.36 Implementa l'operació `elimina` d'una classe diccionari digital implementada amb un arbre ternari de cerca. Aquesta operació ha d'eliminar la parella <clau, valor> del diccionari. NOTA: Podeu suposar que la classe genèrica Clau té definit l'**operator[]**. Aquest operador retorna el símbol de la clau que es troba en la posició n.

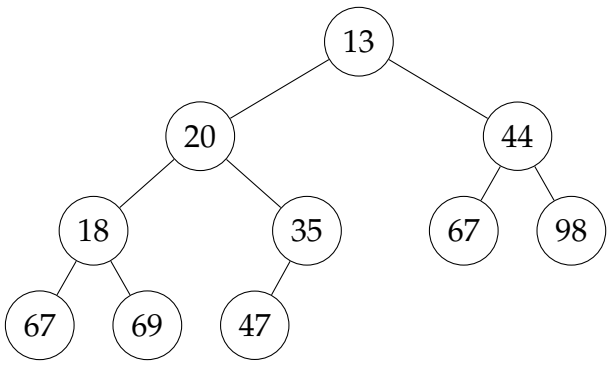
7

Cues de prioritat

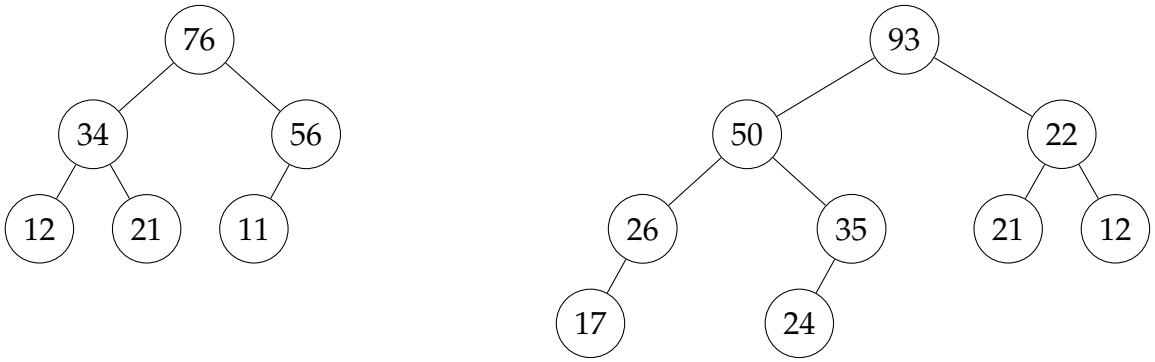
7.1 Monticles

7.1 Indica si els arbres binaris següents són `min-heaps` o no i perquè.





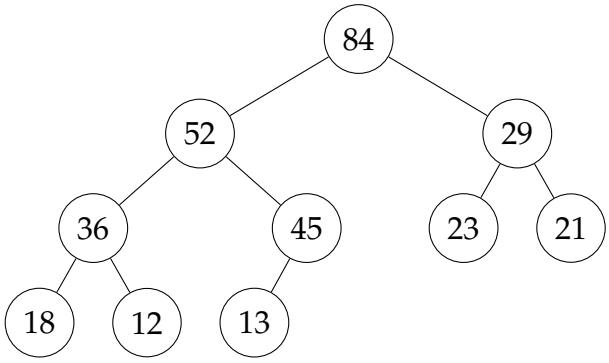
7.2 Indica si els arbres binaris següents són `max-heaps` o no i perquè.



7.3 Partint d'un `min-heap` buit, insereix successivament les claus 45, 67, 23, 46, 89, 65, 12, 34, 98, 76.

7.4 Partint d'un `max-heap` buit, insereix successivament les claus 45, 67, 23, 46, 89, 65, 12, 34, 98, 76.

7.5 Partint del `max-heap` següent, dibuixa el `max-heap` cada cop que eliminem l'element màxim. Fes això successivament fins que quedi buit.



7.6 Donat el següent `min-heap` implementat en vector

7	11	9	23	41	27	12	29

dibuixa l'arbre representat pel vector, i a continuació pinta l'evolució del contingut del vector i de l'arbre representat, en aplicar successivament les operacions següents: `insereix(3)`, `elim_min`.

7.7 Donat el següent vector que emmagatzema un `min-heap`

1	5	2	6	8	4	10	7	12	9	...

n = 10

dibuixa dos `min-heaps` (vector) després d'aplicar `elim_min` i després d'aplicar, sobre el heap resultant de l'eliminació anterior, l'operació `insereix(3)`.

7.8 Converteix la taula següent en un `min-heap` tot aplicant l'algorisme de construcció de heaps de dalt cap a baix.

45	53	27	21	11	97	34	78

7.9 Convertiu la taula següent en un `min-heap` tot aplicant l'algorisme de construcció de heaps de baix cap a dalt.

45	53	27	21	11	97	34	78

7.10 Considera la següent afirmació: *"Eliminar el màxim d'una cua de prioritats de* n *elements amb prioritats diferents implementada amb un max-heap en vector té cost* $\Theta(1)$ *en cas millor, inclòs si* n *fos molt gran"*. Si l'afirmació és certa, mostra un exemple. En cas contrari, justifica perquè no ho és.

7.2 Heapsort

7.11 Es vol ordenar el següent vector de menor a major amb l'algorisme de `Heapsort`. Dibuixa l'evolució del vector pas a pas i també del heap que el vector representa.

2	5	4	9	8	7

7.12 Implementa l'acció heapsort que ordena un vector de n enters mitjançant un heap implementat en el mateix vector a ordenar.

```
template <typename T>
void heapsort (T A[], nat n);
```

Suposa que l'acció enfonsar que ens enfonsa el node j-èssim fins a restablir l'ordre del heap de n elements ja està implementada i, per tant, la pots utilitzar lliurement.

```
template <typename T>
void enfonsar (T A[], nat n nat j) throw(error);
```

Comenta el cost temporal que té cada part de l'acció heapsort i el seu cost total.

8

Grafs

8.1 Implementació grafs

8.1 Donada una classe graf que implementa un graf dirigit i no etiquetat:

a) Escriu la representació d'aquesta classe usant matrius d'adjacència.

b) Implementa el constructor per defecte d'aquesta classe usant la representació anterior.

c) Implementa el mètode `adjacents` usant la representació anterior.

8.2 Donada una classe graf que implementa un graf no dirigit i etiquetat:

a) Escriu la representació d'aquesta classe usant llistes d'adjacència.

b) Implementa el constructor per defecte d'aquesta classe usant la representació anterior.

c) Implementa el mètode `adjacents` usant la representació anterior.

8.3 Donada una classe graf que implementa un graf dirigit i etiquetat:

a) Escriu la representació d'aquesta classe usant multillistes d'adjacència.

b) Implementa el constructor per defecte d'aquesta classe usant la representació anterior.

c) Implementa el mètode `adjacents` usant la representació anterior.

8.2 Recorregut sobre grafs

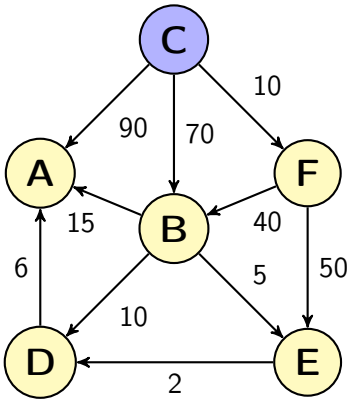
8.4

a) Donat el següent graf fes un esquema de la seva implementació amb:

i) matrius d'adjacència

ii) llistes d'adjacència

iii) multillistes d'adjacència.



b) Descriu breument en que consisteix un recorregut

i) en profunditat prioritària (DFS)

ii) en amplada (BFS)

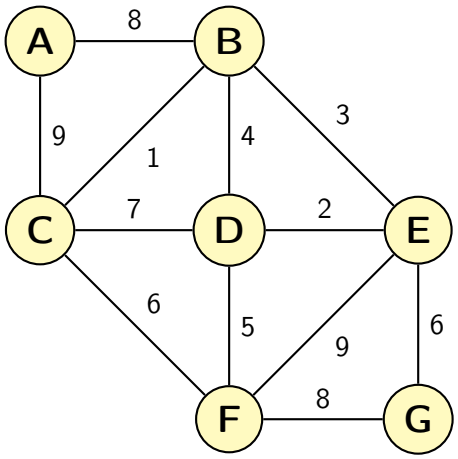
iii) en ordenació topològica

i per cadascun d'ells escriu la llista de nodes resultat d'aplicar-lo en el graf anterior començant pel vèrtex C (suposa que l'operació successor d'un vèrtex retorna els seus successors en ordre alfabètic).

8.3 Arbre d'expansió mínima

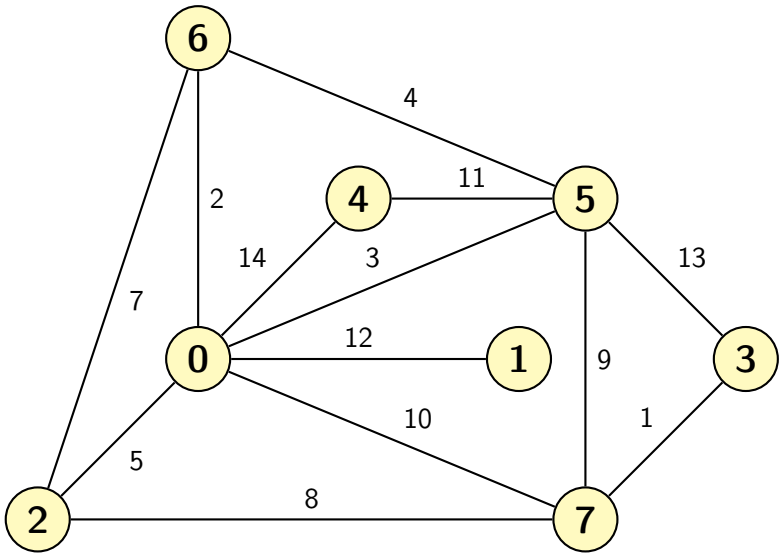
8.5

a) Donat el següent graf no dirigit, connex i etiquetat amb valors naturals fes un esquema de la seva implementació amb llistes d'adjacència. Té sentit usar multillistes d'adjacència per aquest graf? Raona la resposta.



b) Aplica l'algorisme de Kruskal per calcular l'arbre d'expansió mínima (Minimum Spanning Tree). Indica l'evolució de l'algorisme pas a pas. Quin classe auxiliar és necessària per implementar Kruskal?

8.6 Donat el següent graf:



respon a aquestes preguntes:

a) Quines arestes del següent graf són examinades per l'algorisme de Kruskal i rebutjades?

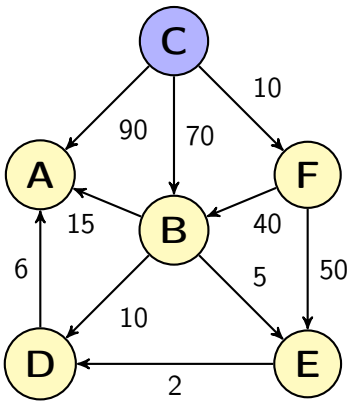
b) Quines són les arestes que formen l'arbre d'expansió mínim?

Suposa que l'algorisme no continua examinant arestes un cop ha construït l'arbre d'expansió mínim. Per llistar una aresta dóna els números dels vèrtexs que uneix, el de menor número primer, p.ex. 3-7. Escriu les arestes en el mateix ordre que són considerades per l'algorisme de Kruskal.

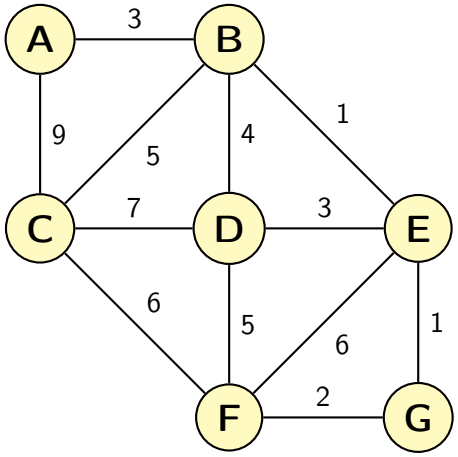
8.4 Camins mínim

8.7 Descriu amb detall com cal modificar l'algorisme de Dijkstra per tal que no només calculi el pes dels camins mínims entre un vèrtex donat s i la resta, sinó que també calculi quants camins de pes mínim hi ha entre s i u, per a tot vèrtex u del graf. Addicionalment, dibuixa un exemple i feu-lo servir per il·lustrar el funcionament de l'algorisme que proposeu.

8.8 Aplica l'algorisme de Dijkstra al següent graf per a trobar el camí més curt del node C a la resta de nodes. Indica l'evolució de l'algorisme pas a pas.



8.9 Donat el següent graf no dirigit, connex i etiquetat:



aplica l'algorisme de Dijkstra per calcular el camí més curt entre el vèrtex A i la resta. Indica l'evolució de l'algorisme pas a pas, mostrant el vector de distàncies i el conjunt de vèrtexs visitats.