

UNIVERSITÀ DI PISA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

INGEGNERIA INFORMATICA

Algoritmi e strutture dati

Raccolta dei compiti passati

A.A 2019-2020 - Aggiornata al 23 aprile 2020

File realizzato da Gabriele Frassi, disponibile presso la copisteria *OneCent*

I Compiti scritti passati	3	16 Compito 16	39
1 Compito 1	5	17 Compito 17	41
2 Compito 2	8		
3 Compito 3	10	II Compiti pratici passati	43
4 Compito 4	12	18 Compito 1	45
5 Compito 5	14	19 Compito 2	46
6 Compito 6	16	20 Compito 3	47
7 Compito 7	18	21 Compito 4	48
8 Compito 8	20	22 Compito 5	49
9 Compito 9	23	23 Compito 6	50
10 Compito 10	25	24 Compito 7	51
11 Compito 11	28	25 Compito 8	52
12 Compito 12	30	26 Compito 9	53
13 Compito 13	32	27 Compito 10	54
14 Compito 14	34	28 Compito 11	55
15 Compito 15	36	29 Compito 12	56
		30 Compito 13	57

Test Files

I *Test Files*, che potrete utilizzare per verificare la validità delle vostre prove pratiche, sono disponibili al seguente link: <http://www.ifrax.it/materialeidattico/provepraticheASD.zip>

Parte I

Compiti scritti passati

1	2	3	4	5
7	6	7	7	6

Esercizio 1

Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione di n:

```

int a = 0;
for (int i=0; i <= g(n)/n; i++)
    a += f(n);
}

```

con le funzioni **f** e **g** definite come segue:

<pre> int f(int x) { if (x<=1) return 1; cout << f(x/3) + f(x/3); return f(x/3) + 1; } </pre>	<pre> int g(int x) { int a=0; for (int i=0; i <= f(x); i++) a++; for (i=0; i <= 2*f(x); i++) a+=i; return a; } </pre>
--	---

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità dell'iterazione singola.

Tf(0,1)=k1
Tf(n)=k2 + 3 Tf(n/3) Tf(n) è O(n)
Rf(0,1)=1
Rf(n)=1 + Rf(n/3) Rf(n) è O(logn)

Calcolo Tg(n)
Entrambi i for hanno
numero iterazioni: O(logn) Complessità singola iterazione: O(n)
complessità dei for: O(nlogn)

il valore di alla fine e' O((logn)^2) perché a contiene logn + 2 volte la somma dei primi logn numeri
Tg(n) è O(nlogn)
Rg(n) è O((logn)^2)

Calcolo totale del blocco:
numero iterazioni: Rg(n)/n = ((logn)^2/n)
Complessità singola iterazione: Tg(n)+ Tf(n)+= O(nlogn) + O(n)=O(nlogn)
Complessità totale blocco = O(((logn)^2)/n * nlogn) = O((logn)^3)

Esercizio 2

Indicare l'output del seguente programma e le istanze delle funzioni template generate dal compilatore.

```

class alpha {
protected:
    int a;
public:
    alpha() {a=8;
              cout << "nuovo alpha " << endl;
            }
    void virtual f()=0;
    void g() {cout << a << endl; }
};

class beta: public alpha {
protected:
    int a;
public:
    beta() {a=5;
              cout << "nuovo beta " << endl;
            }
    void virtual f() { }
    void virtual g() {cout << a << endl; }
};

class delta: public beta {
protected:
    beta ob;
public:
    delta() {
              cout << "nuovo delta " << endl;
            }
    void f() { cout << a << endl; }
    void g(){cout << a+1 << endl; }
};

template<class T>
void funzione( T *obj){
    static int a;
    obj->f();
    obj->g();
    a++;
    cout << a << endl;
}

template<class T1, class T2>
void funzione1( T1 *obj1, T2 *obj2){
    funzione(obj1);
    cout << endl;
    funzione(obj2);
    cout << endl;
}

```

```

    funzione<alpha> (obj2);
    cout << endl;
}

void main() {
    delta *obj1= new delta;
    alpha *obj2=obj1;
    beta *obj3=obj1;
    funzione1(obj3, obj2);
}

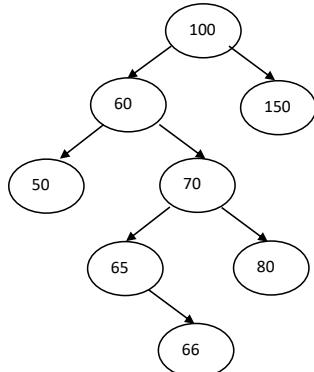
```

nuovo alpha
nuovo beta
nuovo alpha
nuovo beta
nuovo delta
5
6
1
5
8
1
5
8
2

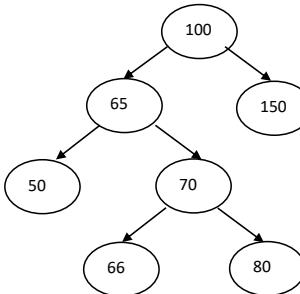
Istanze delle funzioni: funzione1<beta, alpha>, funzione<beta>, funzione<alpha>

Esercizio 3

Descrivere a parole il tipo di dato albero binario di ricerca: a) definizione e descrizione delle operazioni con relativa complessità. b) Dato l'albero in figura, indicare l'albero risultante dalla cancellazione del nodo 60. c) Indicare un esempio di albero binario di ricerca con 7 nodi per cui la complessità della ricerca è minima e uno per cui è massima.



b)



Esercizio 4

Scrivere una funzione **void potatura(Node* t)** che, dato un albero generico, cancella il secondo sottoalbero di ogni nodo, se esiste. Si supponga che l'albero generico sia non vuoto e memorizzato figlio-fratello. Indicare la complessità.

```

void potatura (Node* t) {
    if (!t) return;
    if (t->left) && (t->left->right) {
        Node* temp= t->left->right;
        t->left->right= t->left->right->right;
        temp->right=NULL;
        deltreetemp();
    }
    potatura (t->left);
    potatura (t->right);
}

void deltreetemp (Node* &t) {
    if (!t) return;
    deltreetemp (t->left);
    deltreetemp (t->right);

    delete (t);
    t=NULL;
}
  
```

Esercizio 5

Descrivere l'algoritmo di compressione di Huffmann: a cosa serve, qual è la sua complessità e come viene calcolata.

Applicarlo al seguente alfabeto con le frequenze indicate, indicando l'albero risultante con la convenzione che per ogni nodo con figli il nodo minore sta a sinistra e corrisponde a 0.

carettere	frequenza
A	9
B	37
C	21
D	20
E	8
F	5

carettere	codifica
A	100
B	11
C	01
D	00
E	1011
F	1010

FONDAMENTI DI INFORMATICA II –Algoritmi e Strutture Dati

a.a 2015/16 – 5 luglio 2016

Esercizio 1

- a) Descrivere il tipo di dato heap: definizione, operazioni e loro complessità, memorizzazione
- b) Un array ordinato in ordine decrescente è uno heap? SI
- c) Uno heap è un array ordinato in ordine decrescente? NO
- d) Sia dato lo heap:

[100 77 87 25 56 34 12]

mostrare lo stato dello stesso e le chiamate ad up e down

- 1) Dopo l'inserimento del valore 33
- 2) Dopo l'estrazione di un elemento (a partire dallo heap ottenuto al passo a)

1)

[100 77 87 33 56 34 12 25] up(7) up(3)

2)

[87 77 34 33 56 25 12] down(0) down(2) down(5)

Esercizio 2. Calcolare la complessità del comando (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione di n:

```
for (int i=0; i <=f(g(n)); i++) a += f(n);
```

con le funzioni **f** e **g** definite come segue:

```
int f(int x) {                                int g(int x) {
    if (x<=1)                                if (x<=1)
        return 1;                                return 1;
    cout << 2*f(x/2)+f(x/2);                  int a=0;
    return f(x/2)+f(x/2)+1;                    for (int i=0; i <= f(x); i++)
                                                a+=i;
                                                return a + g(x-1);
}
```

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

$T_f(0,1) = k_1$
 $T_f(n) = k_2 + 4 T_f(n/2)$ $T_f(n)$ è $O(n^2)$
 $R_f(0,1) = 1$
 $R_f(n) = 1 + 2R_f(n/2)$ $R_f(n)$ è $O(n)$

Calcolo $T_g(n)$
numero iterazioni: $O(n)$
complessità singola iterazione: $O(n^2)$
complessità dei for: $O(n^3)$

il valore di alla fine è $O(n^2)$

$T_g(0,1) = a$
 $T_g(n) = bn^3 + T_g(n-1)$ $T_g(n)$ è $O(n^4)$
 $R_g(0,1) = 1$
 $R_g(n) = n^2 + R_g(n-1)$ $R_g(n)$ è $O(n^3)$

Calcolo for del blocco:
numero iterazioni: $O(n^3)$
Complessità singola iterazione: $T_g(n) + T_f(n^3) + T_f(n) = O(n^3) + O(n^6) + O(n^2) = O(n^6)$
Complessità totale blocco = $O(n^3) * O(n^6) = O(n^9)$

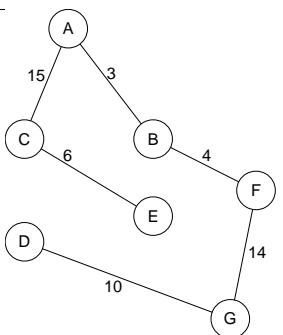
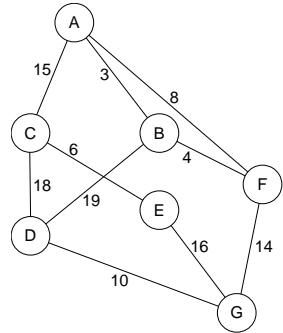
Esercizio 3

Scrivere una funzione che, dato un albero generico con etichette intere, aggiunge ad ogni nodo la somma delle etichette dei suoi figli. Si supponga che l'albero generico sia memorizzato figlio-fratello. Indicare la complessità.

```
void somma (Node* t) {
    if (!t) return;
    t->label= sommafigli(t->left);
    somma (t->left);
    somma (t->right);
}
int sommafigli (Node* t) {
    if (!t) return 0;
    return (t->label + sommafigli(t->right));
}
```

Esercizio 4

- a) Descrivere l'algoritmo di Kruskal: a cosa serve, come è implementato e qual è la sua complessità.
- b) Trovare l'albero di copertura di costo minimo del grafo in figura mediante l'algoritmo di Kruskal. Indicare l'ordine in cui vengono presi gli archi durante l'esecuzione dell'algoritmo.



b)

Esercizio 5

Indicare l'output del seguente programma c++. Per ogni linea di output, indicare quali costruttori vengono chiamati.

```
#include<iostream.h>
class uno {
protected:
    int a;
public:
    uno() { a=0; cout << "uno, a= " << a << endl; }
    uno(int z) { a=z; cout << "uno, a= " << a << endl; }
};
```

```
class due : public uno{
protected:
    int a;
public:
    due() { a=1; cout << "due, a= " << a << endl; }
    due(int y): uno(y){ a=1; cout << "due, a= " << a << endl; }
};

class tre: public due{
public:
    tre() { cout << "tre, a= " << a << endl; }
    tre(int x, int y): due(y){ a=x; cout << "tre, a= " << a << endl; }
    tre(int x): due(x){ a=x; cout << "tre, a= " << a << endl; }
};

void main(){
    tre obj1;
    tre obj2(4,6);
    tre obj3(5);
}

=====
uno, a= 0                                // uno::uno()
due, a= 1                                 // due:: due()
tre, a= 1                                 // tre::tre()
uno, a= 6                                // uno::uno(6)
due, a= 1                                 // due::due(6)
tre, a= 4                                 // tre::tre(4,6)
uno, a= 5                                // uno::uno(5)
due, a= 1                                 // due::due(5)
tre, a= 5                                 // tre::tre(5)
```

Fondamenti di Informatica II – Modulo di Algoritmi e Strutture dati

ANNO ACCADEMICO 2015/16 - 26 luglio 2016

Esercizio 1

- a. Dare la definizione di complessità, cioè quando una funzione è O di un'altra.
- b. Utilizzando la definizione, dimostrare che: se $f(n)$ è $O(g(n))$ e $g(n)$ è $O(h(n))$, allora $f(n)$ è $O(h(n))$.

b.
Se $f(n)$ è $O(g(n))$, allora esistono n_1 e c_1 tali che : per ogni $n \geq n_1$ $f(n) \leq c_1 * g(n)$

Se $g(n)$ è $O(h(n))$, allora esistono n_2 e c_2 tali che: per ogni $n \geq n_2$ $g(n) \leq c_2 * h(n)$

Se $n_3 = \max(n_1, n_2)$ e $c_3 = c_1 * c_2$ vale che : per ogni $n \geq n_3$, $f(n) \leq c_3 * h(n)$

e questo dimostra che $f(n)$ è $O(h(n))$.

Esercizio 2

Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione del numero di nodi dell'albero t , supponendo che sia quasi bilanciato e che Nodes sia la funzione definita a lezione.

```

int a = 0;
for (int i=0; i <= g(f(t)); i++)
    a += Nodes(t);
}

```

Le funzioni f e g sono definite come segue:

<pre>int f(Node* tree) { if (!tree) return 1; int a=Nodes(tree),j=a; for (int i=1;i<=j;i++) a+=j; int b = f(tree->left) +f(tree->right); return a+b; }</pre>	<pre>int g(int x) { if (x<=1) return 1; cout << g(x/2) + g(x/2) + g(x/2); int a=1+g(x/2); return a; }</pre>
---	--

$T_f(0)=\text{cost}$
 $T_f(n)=dn+2T_f(n/2)$ $O(n\log n)$

$R_f(0)=1$
 $R_f(n)=dn^2+2R_f(n/2)$ $O(n^2)$

$T_g(0)=\text{cost}$
 $T_g(n)=d+4T_g(n/2)$ $O(n^2)$

$R_g(0)=1$
 $R_g(n)=1+R_g(n/2)$ $O(\log n)$

Numero iterazioni for: $R_g(R_f(n)) = (O(\log(n^2))) = O(\log n)$

Complessità di una iterazione: $C[f(t)] + C[g(n^2)] + C[\text{Nodes}(t)] = O(n\log n) + O(n^4) + O(n) = O(n^4)$

Complessità del blocco: $O(n^4 * \log n)$

Esercizio 3

Si scriva una funzione che, dati un albero generico ad etichette intere (memorizzato figlio-fratello) con puntatore alla radice t e due interi a e b , ritorna true se nell'albero c'è un nodo con etichetta a che ha fra i suoi discendenti un nodo con etichetta b . Se ne calcoli la complessità in funzione del numero di nodi dell'albero.

<pre>bool findabG (Node* t, int a, int b) { if (!t) return false; if (t->label == a) { if (findNode(t->left,b)) return true; } return (findabG(t->left) findabG(t->right)); }</pre>	<pre>bool findab (Node* t, int a, int b, bool a_found=false) { if (!t) return false; bool a_old = a_found; if ((t->label == b) && a_found) return true; if (t->label == a) a_found = true; return (findab(t->left,a, b, a_found) findab(t->right,a, b, a_old)); }</pre>
---	---

Esercizio 4

- a) Descrivere l'algoritmo PLSC: a cosa serve, su quale ragionamento è basato, come funziona, qual è la sua complessità.
- b) Applicarlo alle due sequenze: BBACB e BABBC e indicare la lunghezza della PLSC e la/le PLSC.

b) B B A C B

	0	0	0	0	0	0
B	0	1	1	1	1	1
A	0	1	1	2	2	2
B	0	1	2	2	2	2
B	0	1	2	2	2	3
C	0	1	2	2	3	3

PLSC: BBB, BBC, BAC, BAB

Esercizio 5

Indicare l'output del programma seguente

```
#include <iostream.h>
template <class T>
class A{
T a;
public:
A(T x) {
    a=x;
    cout << 'A' << endl;
    cout << a << endl;
}
void fun(T x) {
    cout << a << endl;
    cout << x << endl;
}
};

void main(){
    A<int> obj(8);
    obj.fun(3.5);
}
```

A
8
8
3

Il risultato è lo stesso del programma seguente? Spiegare.

```
#include <iostream.h>
template <class T>
class A{
T a;
public:
A(T x) {
    a=x;
    cout << 'A' << endl;
    cout << a << endl;
}
void fun(T1 x) {
    cout << a << endl;
    cout << x << endl;
}
};

void main(){
    A<int> obj(8);
    obj.fun(3.5);
}

template<class T1>
```

Nel primo caso viene instanziata la funzione fun<int> perché fun ha lo stesso template della classe che viene instanziata con int (con conseguente conversione), nel secondo la funzione fun<double> perché il template della funzione è indipendente da quello della classe. Di conseguenza l'output del secondo programma è:

A
8
8
3,5

Punteggio esercizi: 6,7,7,7,6

FONDAMENTI DI INFORMATICA II – Algoritmi e Strutture dati

19 settembre 2016 - ANNO ACCADEMICO 2015/16

Esercizio 1

1	2	3	4	5
5	7	7	7	7

Sia dato il seguente min-heap (un min-heap è uno heap in cui ogni nodo è minore o uguale dei suoi figli e le operazioni di up e down cambiano di conseguenza):

[10 20 15 30 20 40 17]

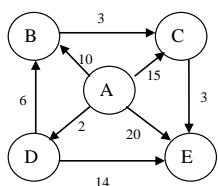
mostrare lo stato dello stesso e le chiamate ad up e down:

- A) dopo l'inserzione dell'intero 9
- B) dopo l'estrazione di un elemento dal min-heap ottenuto al passo A
- C) dopo l'estrazione di un elemento dal min-heap ottenuto al passo B

A 9 10 15 20 20 40 17 30	up(7), up(3), up(1), up(0)
B 10 20 15 30 20 40 17	down(0), down(1), down(3)
C 15 20 17 30 20 40	down(0), down(2)

Esercizio 2

- a) Descrivere l'algoritmo di Dijkstra: a cosa serve, il suo funzionamento, la sua complessità. (3)
- b) Applicarlo al grafo in figura con il nodo A come nodo di partenza. (4)



Q	A	B	C	D	E
A, B, C, D, E	0 -	inf -	inf -	inf -	inf -
B, C, D E	0 -	10 A	15 A	2 A	20 A
B, C, E	0 -	8 D	15 A	2 A	16 D
C, E	0 -	8 D	11 B	2 A	16 D
E	0 -	8 D	11 B	2 A	14 C

Esercizio 3

Calcolare la complessità in funzione di $n > 0$ dell'istruzione

$y=g(f(n));$

con le funzioni f e g definite come segue:

```
int f(int x) {
    if (x<=1) return 1;
    int b=0, i, j, c;
    for (i=1; i<=x; i++) b+=i;
    c = b*b;
    for (j=1; j<=c; j++) b+=j;
    return b + f(x-1);
}
```

```
int g(int x) {
    if (x<=1) return 10;
    int a=0;
    for (int i=0; i<f(x); i++)
        a++;
    return a+2*g(x/2);
}
```

Indicare le eventuali relazioni di ricorrenza e spiegare brevemente il calcolo della complessità dei cicli.

Stima del tempo di f

Primo for
numero iterazioni = $O(n)$
complessità di un'iterazione = costante
tempo del for = $O(n)$

Secondo for
numero iterazioni for = $O(n^4)$
complessità di un'iterazione = costante
tempo del for = $O(n^4)$

$T_f(1) = a$
 $T_f(n) = b n^4 + T_f(n-1) \quad O(n^5)$

$R_f(1)=a$
 $R_f(n) = n^8 + R_f(n-1) \quad O(n^9)$

Stima del tempo di g :
numero iterazioni del for: $R_g(m) = O(m^9)$
complessità di un'iterazione: $T_g(m) = O(m^5)$
tempo del for: $O(m^{14})$

tempo di g
 $T_g(1) = \text{cost}$
 $T_g(m) = c \cdot m^{14} + T_g(m/2)$
 $T_g \in O(m^{14})$

Tempo di $y=g(f(n))$:

$T_f(n) + T_g(n^9) = O(n^5) + O(n^{(9*14)}) = O(n^{126})$

Esercizio 4

Sia dato un albero binario ad etichette intere. Scrivere una funzione che, per ogni nodo somma all'etichetta la differenza fra il numero di discendenti di sinistra e il numero di foglie di destra. La complessità della funzione deve essere $O(n)$, con n numero di nodi dell'albero.

```
int somma(Node* t, int & foglie) {
    if (!t)
        {foglie=0; return 0; }
    if ( !t->left && !t->right)
        {foglie=1; return 1; }

    int nodi_l, nodi_r, foglie_l, foglie_r ;
    nodi_l = somma(t->left, foglie_l);
    nodi_r = somma(t->right, foglie_d);
    t->label+=nodi_l-foglie_r ;
    foglie= foglie_l+foglie_r ;
    return nodi_l+nodi_r+1 ;
}
```

Esercizio 5 Sia dato il seguente programma c++.

```
class A {
protected:
    int a;
public:
    A(){a=8;}
    void stampa (){ cout << a; }
};

class B: public A {
protected:
    int a;
public:
    B(){a=9;}
    void stampa (){ cout << a; }
};

class C: public A {
protected:
    int a;
public:
    C(){a=10;}
    void stampa (){ cout << a; }
};

class D: public C {
public:
    D(){a=11;}
};

int main(){
    B *obj1= new B;
    D *obj2= new D;
    C *obj3= new C;
    obj1->stampa();
    obj2->stampa();
    obj3->stampa();
}
```

1. Indicare l'uscita del programma (3)

- a) così come è scritto
b) eliminando la linea asteriscata

a) 9 11 10
b) 9 8 8

2. Spiegare la eventuale differenza fra i due casi. (2)

Nel secondo caso la funzione stampa che viene chiamate da obj2 e da obj3 è quella della classe A, poiché né la classe C né la classe D ridefiniscono questa funzione e quindi viene chiamata la funzione ereditata da A.

3. Spiegare cosa vuol dire "classe astratta". (2)

E' una classe che contiene almeno una funzione virtuale pura e per questo non può essere instanziata.

ANNO ACCADEMICO 2015/16

Algoritmi e Basi di dati – Modulo di Algoritmi e Strutture dati 17 gennaio 2017

1	2	3	4	5
6	7	7	7	6

Esercizio 1

1. Descrivere le seguenti memorizzazioni di un grafo orientato etichettato: a) liste di adiacenza;
b) matrici di adiacenza
2. Se il grafo ha n nodi e m archi, calcolare la complessità, per ognuna delle memorizzazioni indicate, dell'operazione di:
 - i. Inserimento di un arco che congiunge i nodi i e j
 - ii. Cancellazione di un arco che congiunge i nodi i e j

a-i O(1)
a-ii O(m)
b-i O(1)
b-ii O(1)

Esercizio 2

Calcolare la complessità dell'istruzione

```
for (int i=0; i<=f(t); i++) cout << g(t->left) + g(t->right);
```

in funzione del numero di nodi di t . Indicare per esteso le relazioni di ricorrenza.

Supporre che t sia un albero binario bilanciato e le funzioni f e g siano definite come segue:

```
int f(Node* t) {
    if (!t)
        return 1;
    return g(t->left) + f(t->left) +
f(t->right);
}
int g(Node * t) {
    if (!t)
        return 0;
    int a = g(t->left);
    int b = g(t->right);
    return 1 + 2a + 2b;
}
```

$$Tg(0) = d$$

$$Tg(n) = c + 2 Tg(n/2) \quad Tg(n) \text{ è } O(n)$$

$$Rg(0) = d$$

$$Rg(n) = c + 4 Rg(n/2) \quad Rg(n) \text{ è } O(n^2)$$

$$Tf(0) = d$$

$$Tf(n) = cn + 2 Tf(n/2) \quad Tf(n) \text{ è } O(n\log n)$$

$$Rf(0) = d$$

$$Rf(n) = cn^2 + 2 Rf(n/2) \quad Rf(n) \text{ è } O(n^2)$$

Numero di iterazioni del for: $O(n^2)$

Complessità di una iterazione: $Tf(n) + 2Tg(n/2) = O(n\log n) + O(n) = O(n\log n)$

Complessità del for: $O(n^3\log n)$

Esercizio 3

Sia dato un albero generico ad etichette intere memorizzato figlio-fratello. Si scriva una funzione con complessità $O(n)$ che prende in ingresso un intero positivo k e somma 1 ad ogni foglia di livello maggiore di k .

```
void sum (Node* t, int k, int level) {
    if (!t) return;
    if (level>k && !t->left) t->label++;
    sum level(t->left, k, level+1);
    sum level(t->right, k, level);
}
```

Esercizio 4

- a) Algoritmo di Huffman per la codifica dei caratteri di un alfabeto: dire quale problema risolve, quali sono i suoi input e output, descriverlo dettagliatamente indicando la sua complessità.
- b) Fare un esempio con un alfabeto di 4 caratteri per il quale l'algoritmo non presenta miglioramenti rispetto ad una codifica fissa

Esercizio 5

- a) Spiegare il meccanismo della derivazione in C++, con riferimento alla visibilità delle variabili e dei metodi delle classi nella gerarchia.
- b) Dato il seguente programma, indicarne la gerarchia di classi e l'output indicando riga per riga qual è la funzione coinvolta.

```
class A {
protected:
int x;
public:
A() { x=10; cout << x << endl;};
void stampa(){ cout << x << endl;};
};
class B: public A {
int x;
public:
B() { x=20; cout << x << endl;};
void stampa(){ cout << x << endl;};
};
class C: public B {
int x;
public:
C() { x=30; cout << x << endl;};
void stampa(){ cout << x << endl;};
};
class D: public A {
```

```

public:
D() { x=50; cout << x << endl;};
void stampa(){ cout << x << endl;};
};

class E: public D {
int x;
public:
E() { x=40; cout << x << endl;};
void stampa(){ cout << x << endl;};
};

int main(){

A* objA=new A;
B* objB=new B;
C* objC=new C;
D* objD=new D;
E* objE=new E;

B* objH = objC;
objH->stampa();

A* objL=objE;
objL->stampa();
}

```

10	A()
10	A()
20	B()
10	A()
20	B()
30	C()
10	A()
50	D()
10	A()
50	D()
40	E()
20	B.stamp()
50	A.stamp()

ANNO ACCADEMICO 2015/16**Algoritmi e Basi di dati – Modulo di Algoritmi e Strutture dati****6 febbraio 2017**

1	2	3	4	5
5	7	7	7	7

Esercizio 1

Sia dato lo heap:

[80 70 60 60 50 10 15]

mostrare lo stato dello stesso e le chiamate ad up e down

a) Dopo l'estrazione di un elemento

b) Dopo l'inserimento del valore 65(a partire dallo heap ottenuto al passo a)

a) [70 60 60 15 50 10] down(0), down(1), down(3)

b) [70 60 65 15 50 10 60] up(6), up(2)

Esercizio 5

- a) Cosa sono gli alberi di decisione e a cosa servono.
- b) Spiegare come con gli alberi di decisione si arriva ad affermare che mergesort è un algoritmo ottimo.
- c) Disegnare l'albero di decisione per il problema di trovare il minimo in una sequenza di tre elementi. Qual è la complessità minima per questo problema che si ricava dall'albero di decisione?

Esercizio 2Calcolare la complessità dell'espressione $g(f(n)) + f(g(n))$ in funzione di n (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni f e g definite come segue:

<pre>int f(int x) { if (x<=1) return 2; int a = f(x/2)+f(x/2); cout << a; return 1+ x + 4*a; }</pre>	<pre>int g(int x) { int b=0; if (x<=1) return 5; for (int i=1, i<=2*f(x);i++) b++; return 10 + b +g(x-2); }</pre>
---	---

$T_f(1)=k_1$
 $T_f(n)=k_2 + 2T_f(n/2)$ $T_f(n) \in O(n)$

$R_f(1)=k$
 $R_f(n)=n + 8T_f(n/2)$ $T_f(n) \in O(n^3)$

$R_f(1) \in O(n^3)$

Calcolo $T_g(n)$

for:
numero iterazioni: $R_g(x) = O(n^3)$
Complessità singola iterazione: $T_g(n) = O(n)$
complessità del for: $O(n^4)$

$T_g(1)=k_1$
 $T_g(n)=k_2 n^4 + T_g(n-2)$ $T_g(n) \in O(n^5)$

$R_g(1)=5$
 $R_g(n)=k n^3 + T_g(n-2)$ $T_g(n) \in O(n^4)$

Tempo di $g(f(n))$ = Tempo per il calcolo di $f(n)$ + tempo per il calcolo di $g(n^3)$ =

$O(n) + O(n^15) = O(n^{16})$

Tempo di $f(g(n))$ = Tempo per il calcolo di $g(n)$ + tempo per il calcolo di $f(n^4)$ =

$O(n^5) + O(n^4) = O(n^9)$

Tempo per l'esecuzione di $g(f(n)) * f(g(n)) = O(n^{16}) + O(n^9) = O(n^{25})$

Esercizio 3

Dato un albero generico memorizzato figlio-fratello con etichette intere, cancellare il primo figlio di ogni nodo se è una foglia e ha una etichetta maggiore o uguale della somma delle etichette degli altri figli del nodo (se è l'unico figlio si assume che tale somma sia 0).

```
int sommafratelli(Node*t) {
    if (! t) return 0;
    return t->label+sommafratelli(t->right);
}

void cancella (Node* t) {
    if (! t) return;
    if (! t->left) { cancella(t->right); return; }
    if (! t->left->left &&
        t->left->label >=sommafratelli (t->left->right)) {
        Node* t1=t->left;
        t->left=t->left->right;
        t1->right=0;
        delete t1;
    }
    cancella(t->left);
    cancella(t->right);
}
```

Esercizio 4

Spiegare il meccanismo delle funzioni virtuali.
Indicare l'uscita del seguente programma con le funzioni chiamate per ogni linea di output.

```
#include <iostream>
using namespace std;

class A {
protected:
int x;
public:
A(){ x=10; cout << 15 << endl;};
void stampa(){ cout << x << endl;};
};

class B: public A {
public:
B(){ x=x+20; cout << 16+x << endl;};
virtual void stampa(){ cout << x << endl;};
};

class C: public B {
public:
C(){ cout << 17 << endl;};
}
```

```
void stampa(){ cout << 70 + x << endl;};
};

class D: public A {
public:
D(){ x+=9; cout << x << endl;};
void stampa(){ cout << x << endl;};
};

int main(){
A* objB=new B;
C* objC=new C;
A* objC1=objC;
A* objD=new D;
B* objB1= objC;
objB1->stampa();
objB->stampa();
objC1->stampa();
objD->stampa();
}
```

15	A()
46	B()
15	A()
46	B()
17	C()
15	A()
19	D()
100	C.stampa()
30	A.stampa()
30	A.stampa()
19	A.stampa()

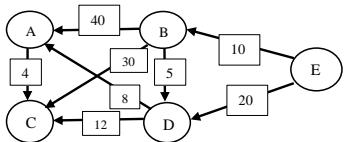
ANNO ACCADEMICO 2015/16

Algoritmi e Basi di dati – Modulo di Algoritmi e Strutture dati
23 febbraio 2017

1	2	3	4	5
7	6	7	7	6

Esercizio 1

- a) Descrivere l'algoritmo di Dijkstra.
- b) Applicarlo al grafo seguente con nodo di partenza E e indicare i cammini minimi



Q	A	B	C	D	E
A, B, C, D, E	Inf -	Inf -	Inf -	Inf -	0 -
A, B, C, D	Inf -	10 E	Inf -	20 E	0 -
A, C, D	50 B	10 E	40 B	15 B	0 -
A, C	23 D	10 E	27 D	15 B	0 -
C	23 D	10 E	27 D	15 B	0 -
Cammini minimi	E B D A	E B	E B D C	E B D	

Esercizio 3

Calcolare la complessità dell'espressione $g(f(n)) + f(g(n))$ in funzione di n (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni f e g definite come segue:

```

int f(int x) {
    if (x<=1) return 2;
    int a = g(x) + f(x/2)+f(x/2);
    return 1+ x + 2*a;
}

int g(int x) {
    int b=0;
    if (x<=1) return 5;
    for (int i=1, i<=x*x;i++)
        b+=i;
    return b + g(x-1);
}
  
```

$$T_f(1)=k_1 \\ T_f(n)=k_2 n^3 + 2T_f(n/2) \quad T_f(n) \in O(n^3)$$

$$R_g(1)=k \\ R_g(n)=kn^5 + 4R_g(n/2) \quad T_g(n) \in O(n^5)$$

Calcolo $T_g(n)$
for:
numero iterazioni: = $O(n^2)$
Complessità singola iterazione: $T_g(n) = O(1)$
complessità del for: $O(n^2)$

$$T_g(1)=k_1 \\ T_g(n)=k_2 n^2 + T_g(n-1) \quad T_g(n) \in O(n^3)$$

$$R_g(1)=5 \\ R_g(n)=k n^4 + R_g(n-1) \quad T_g(n) \in O(n^5)$$

Tempo di $g(f(n))$ = Tempo per il calcolo di $f(n)$ + tempo per il calcolo di $g(n^5)$ =

$$O(n^3)+O(n^15)=O(n^{18})$$

Tempo di $f(g(n))$ = Tempo per il calcolo di $g(n)$ + tempo per il calcolo di $f(n^5)$ =

$$O(n^3)+O(n^15)=O(n^{18})$$

Tempo per l'esecuzione di $g(f(n)) * f(g(n)) = O(n^{18}) + O(n^{18}) = O(n^{36})$

Esercizio 4

Dato un albero generico memorizzato figlio-fratello con etichette intere, scrivere una funzione che, dati tre interi p, q e r, aggiunge al nodo con etichetta r un primo figlio con etichetta p e un ultimo figlio con etichetta q.

```
void aggiungi (Node* t, int p, int q, int r) {
    Node* a=findNode (r,t);
    if (! a) return;
    Node * b=a->left;
    a->left= new Node;
    a->left->right=b;
    a->left->label=p;
    for (Node* c=a->left; c->right; c=c->right);
    c->right= new Node;
    c->right->label=q;
    c->right->left=0;
    c->right->right=0;
}
```

Esercizio 5

Spiegare il meccanismo delle funzioni e classi modello.
Indicare l'uscita del seguente programma.

```
class uno {
    int a;
public:
    uno(){ a=1; cout << a << endl;}
    void valore(){ a++; cout << a << endl;}
};
class due {
    int a;
public:
    due(){ a=2; cout << a << endl;}
    void valore(){ a++; cout << a << endl;}
};
template<class tipol, class tipo2>
class tre {
    static int b;
    tipol x;
    tipo2 y;
public:
    tre (){ cout << 3 << endl << b << endl; };
    void valore(){
        x.valore();
        y.valore();
    }
}
```

```
};

template<class tipol, class tipo2>
int tre<tipol, tipo2>::b;
```

```
int main (){
    tre<uno,uno> obj1;
    tre<uno, due> obj2;
    obj1.valore();
    obj2.valore();
}
```

```
1  
1  
3  
0  
1  
2  
3  
0  
2  
2  
3
```

Esercizio 2

Descrivere gli algoritmi di ordinamento fatti a lezione e confrontarli fra loro.

13 giugno 2017

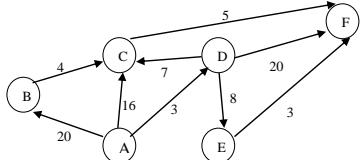
I	2	3	4	5	6
6	5	6	5	6	5

COGNOME E NOME :

MATRICOLA:

Esercizio 1

- a) Descrivere l'algoritmo di Dijkstra: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità.
- b) Applicarlo al grafo seguente a partire dal nodo A.



	A	B	C	D	E	F
A, B, C, D, E, F	0 -	inf -				
B, C, D, E, F	0 -	20 A	16 A	3 A	inf -	inf -
B, C, E, F	0 -	20 A	10 D	3 A	11 D	23 D
B, E, F	0 -	20 A	10 D	3 A	11 D	15 C
B, F	0 -	20 A	10 D	3 A	11 D	14 E
B	0 -	20 A	10 D	3 A	11 D	14 E

Cammini:

AB, ADC, AD, ADE, ADEF

Esercizio 2

- c) Descrivere l'algoritmo di Huffman: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità.

- a) Applicarlo all'alfabeto seguente in cui ad ogni carattere corrisponde la sua frequenza percentuale nel testo, indicando l'albero risultante.

A	12
B	16
C	7
D	10
E	9
F	30
G	5
H	11

Esercizio 3

Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione del numero di nodi dell'albero t,

- a) supponendo che t sia quasi bilanciato.
- b) supponendo che t sia completamente sbilanciato

Indicare per esteso numero di iterazioni e complessità di ogni iterazione per i comandi ripetitivi.

```

{
    int a = 0;
    for (int i=0; i <= g(t)*f(t); i++)
        a += Nodes(t);
}
  
```

Le funzioni **f** e **g** sono definite come segue:

<pre> int f(Node * tree) { if (!tree) return 1; int x=0; for (int i=1;i<= Nodes(tree);i++) x+=i; int b = 4*f(tree->left); return x+b; } </pre>	<pre> int g(Node * tree) { if (!tree) return 1; int a=0; for (int i=1;i<=f(tree)/Nodes(tree);i++) { a++; cout << a; } return 3; } </pre>
--	---

- a) Funzione f

Numero iterazioni del for: O(n)

Complessità di una iterazione: O(n)

Complessità del for: O(n^2)

Tf(0)=a

Tf(n)=dn^2+Tf(n/2)

O(n^2)

$Rf(0)=1$
 $Rf(n)=dn^2+4Rf(n/2)$ $O(n^2 \log n)$

Funzione g

Numero iterazioni del for: $O(n\log n)$
 Complessità di una iterazione: $O(n^2)$
 Complessità del for: $O(n^3 \log n)$

$Tg(n)= O(n^3 \log n)$

$Rg = O(1)$

For esterno

Numero iterazioni del for: $O(n^2 \log n)$
 Complessità di una iterazione: $O(n) + O(n^2) + O(n^3 \log n) = O(n^3 \log n)$
 Complessità del for: $O(n^5 \log^2 n)$

b) consideriamo il caso peggiore

Funzione f

Numero iterazioni del for: $O(n)$
 Complessità di una iterazione: $O(n)$
 Complessità del for: $O(n^2)$

$Tf(0)=a$
 $Tf(n)=dn^2+Tf(n-1)$ $O(n^3)$

$Rf(0)=1$
 $Rf(n)=dn^2+4Rf(n-1)$ $O(4^n)$ esponenziale

Funzione g

Numero iterazioni del for: $O(4^n/n)$
 Complessità di una iterazione: $O(n^3)$
 Complessità del for: $O(n^2 4^n)$

$Tg(n)= O(n^2 4^n)$

$Rg = O(1)$

For esterno

Numero iterazioni del for: $O(n^3)$
 Complessità di una iterazione: $O(n^2 4^n)$
 Complessità del for: $O(n^5 4^n)$

Esercizio 4 Scrivere un algoritmo in c++ che, dato un array, costruisce un albero binario quasi bilanciato che contiene tutti gli elementi dell'array. Descrivere a parole l'algoritmo e implementarlo in c++. Calcolare la complessità.

```
Node * build(int A, int i, int j) {
    if (i>j) return NULL;
    Node * t=new Node;
    int k=(i+j)/2;
    t->label=A[k];
    t->left=build(A,i,k-1);
    t->right=build(A,k+1,j);
    return t;
}

void costruisci (int* A, int n) {
    build (A, 0, n-1);
}
```

Complessità: $O(n)$

Esercizio 5 dato il seguente programma c++.

- a) Indicare la sua uscita
- b) Indicare l'uscita togliendo la parola "virtual"
- c) Indicare l'uscita togliendo la parola "protected"

Spiegare le eventuali differenze fra i casi sopra citati.

```
class C {
public:
    C(){cout << "nuovo C" << endl; };
    void f(){cout << "f di C" << endl;};
};

class A {
protected:
    C obj_x;
public:
    A(){cout << "nuovo A" << endl;
        obj_x.f();
    };
    void virtual f(){cout << "f di A"
        << endl; };
};

class B: public A {
    A obj_y;
public:
    B(){cout << "nuovo B" << endl;
        obj_x.f();
    };
    void f(){cout << "f di B" << endl; };

int main(){
    int i;
    A* obj1= new B;
    obj1->f();
    cin >> i;
}
```

a) Esecuzione del programma

Nuovo C
 Nuovo A
 F di C
 Nuovo C
 Nuovo A
 F di C
 Nuovo B
 F di C
 F di B

b) Togliendo il virtual

Nuovo C
 Nuovo A
 F di C
 Nuovo C
 Nuovo A
 F di C
 Nuovo B
 F di C
 F di A

c) togliendo la linea protected: errore di compilazione perché B usa obj_x che è privato in A

Esercizio 6

Scrivere una funzione `int conta (Node*tree)` che , dato un albero generico memorizzato figlio-fratello, conta quanti nodi hanno un numero pari di figli.

```
int contafigli (Node*tree) {
    if (! tree) return 0;
    return 1 + contafigli( tree->right);
}

int conta (Node*tree) {
    if (! tree) return 0;
    return (contafigli (tree->left)%2==0)
        + conta( tree->left) + conta( tree->right);
}
```

Algoritmi e Strutture dati - ANNO ACCADEMICO 2016/17
4 luglio 2017

Esercizio 1

1	2	3	4	5	6

- a) Descrivere l'algoritmo PLSC: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità e come viene calcolata (scrivere sul retro del foglio).
 b) Applicarlo per trovare la/le PLSC fra le due sequenze: xyzx yxyzy.

		x	y	z	z	y	x
0	0	0	0	0	0	0	0
x	0	1	1	1	1	1	1
x	0	1	1	1	1	1	2
y	0	1	2	2	2	2	2
z	0	1	2	3	3	3	3
x	0	1	2	3	3	3	4
y	0	1	2	3	3	4	4

PLSC: xyzx yxyzy

Esercizio 2

Scrivere una funzione `void sort (Node *tree1, Node *tree2, Elemt * & list)` che, dati due alberi binari di ricerca, costruisce una lista semplice ordinata in ordine crescente contenente tutti gli elementi dei due alberi.

```
void costruisci ( Node *tree, Elemt * & list){
    if (tree) {
        costruisci(tree->right, list);
        Elemt * list1= new Elemt;
        list1->inf=tree->label;
        list1->next=list;
        list=list1;
        costruisci(tree->left, list);
    }
}
void sort (Node *tree1, Node *tree2, Elemt * & list){
    list=NULL;
    Elemt * list2=NULL;
    costruisci(tree1, list);      // O(n/2)
    costruisci(tree2, list2);    // O(n/2)
    merge(list, list2);         // O(n)
}
```

O(n)

Esercizio 3

Scrivere una funzione che, dato un albero generico con etichette intere memorizzato figlio-fratello, conta il numero delle foglie che sono secondi figli di un nodo.

```
int check(Node* tree) {
    if (!tree || !tree->right) return 0;
    if (tree->right->left ==0 ) return 1;
    else return 0;
}
```

```
int conta ( Node *tree){
    if (!tree) return 0;
    return (check(tree->left) + conta(tree->left)+conta(tree->right));
}
```

Esercizio 4

Calcolare la complessità in funzione di $n > 0$ delle istruzioni `y=g(f(n))` e `y= f(g(n))`,

con le funzioni `f` e `g` definite come segue:

<pre>int f(int x) { if (x<=1) return 1; int b=0, a=0, i; for (i=1; i<=3; i++) b+=i; for (i=1; i<=x; i++) a+=i; cout << b*a; return f(x-1) + 4 + b; }</pre>	<pre>int g(int x) { if (x<=1) return 10; int a=0; for (int i=0; i<f(x)*f(x); i++) a++; cout << a; a=0; for (int i=0; i< x; i++) a++; return a+g(x/2)+g(x/2); }</pre>
---	---

Indicare le eventuali relazioni di ricorrenza e spiegare brevemente il calcolo della complessità dei cicli.

Stima del tempo di f

numero iterazioni del 1° for = $O(1)$
 complessità di un'iterazione = costante
 tempo del 1° for = $O(1)$

numero iterazioni del 2° for = $O(n)$
 complessità di un'iterazione = costante
 tempo del for = $O(n)$

$T_f(1)=d$ $T_f \in O(n^2)$
 $T_f(n)=c n + T_f(n-1)$

$R_f(1)=1$ $R_f(n) \in O(n)$
 $R_f(n)=c + R_f(n-1)$

Stima del tempo di g:

numero iterazioni del 1° for = $O(m^2)$
 complessità di un'iterazione = $O(m^2)$
 tempo del for = $O(n^4)$

numero iterazioni del 2° for = $O(m)$
 complessità di un'iterazione = $O(1)$
 tempo del for = $O(m)$

tempo di g

$$T_g(1) = d \quad T_g \text{ è } O(m^4)$$

$$T_g(m) = c \cdot m^4 + 2T_g(m/2)$$

stima del risultato di g

$$R_g(1) = \text{cost} \quad R_g(m) = O(m \log m)$$

$$R_g(m) = cm + 2R_g(m/2)$$

Complessità dell'istruzione $y=g(f(n))$:

$$C[f(n)] + C[g(R_g(n))] = O(n^2) + C[g(n)] = O(n^2) + O(n^4) = O(n^4)$$

Complessità dell'istruzione $y=f(g(n))$:

$$C[g(n)] + C[f(R_g(n))] = O(n^4) + C[f(n \log n)] = O(n^4) + O(n^2 \log^2 n) = o(n^4)$$

Esercizio 5

- Descrivere il tipo di dato heap con le sue operazioni e relative complessità (scrivere sul retro del foglio).
- Dato lo heap

[88, 74, 57, 15, 30, 33, 55, 10]

Indicare

- il contenuto dello heap dopo l'inserimento del valore 78
- il contenuto dello heap dopo una successiva estrazione

Indicare le chiamate a up e down che vengono eseguite in questi casi.

heap	chiamate
88, 74, 57, 15, 30, 33, 55, 10	
i. 88, 78, 57, 74, 30, 33, 55, 10, 15	up(8), up(3), up(1)
ii. 78, 74, 57, 15, 30, 33, 55, 10	down(0), down(1), down(3)

Esercizio 6 Dato il programma seguente:

```

class A {
public:
int x=0;
A(){cout << "new A" << endl;};
};

class B: public A {
public:
int x=1;
B(){cout << "new B" << endl;};
};

class C: public A {
public:
int x=2;
C(){cout << "new C" << endl;};
};

template<class T1, class T2>
class D {
T1* oggetto1= new T1;
T2* oggetto2= new T2;
public:
D(){ cout << "new D" << endl;};
};

void f() {try
{
    T1* oggetto3= oggetto1;
    cout << "messaggio1" <<
endl;
    throw 0;
    T2* oggetto4= oggetto2;
    cout << "messaggio2" <<
endl;
}
catch(int){cout << "diverso
da 0" << endl;}
cout << "fine f" << endl;
};

int main(){
int i;
D<B, C>* obj1 = new D<B, C>;
obj1->f();
D<B, B>* obj2 = new D<B, B>;
obj2->f();
cin >> i;
}

```

- indicare il suo output
- indicare il suo output se si sostituisce l'istruzione asteriscata con :
- dire cosa succede se si sostituisce l'istruzione asteriscata con :

Spiegare brevemente le differenze.

a)

```

new B
new A
new C
new D
messaggio1
diverso da 0
fine f
new A
new B
new A
new B
new D
messaggio1
diverso da 0
fine f

```

```

new A
new B
new A
new C
new D
messaggio1
messaggio2
fine f
new A
new B
new A
new B
new D
messaggio1
messaggio2
fine f

```

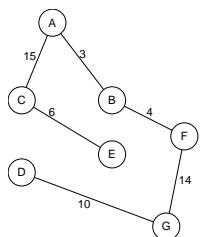
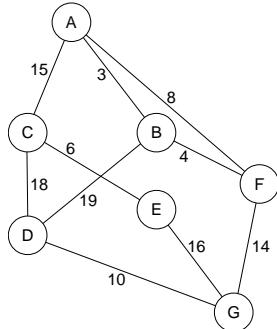
b)

- Il compilatore segnala un errore perchè per incompatibilità di tipo (B con C). nel caso b) invece c'è compatibilità perchè il tipo della variabile a cui si assegna il puntatore all'oggetto (A) è un supertipo del tipo dell'oggetto (B).

1	2	3	4	5	6

Esercizio 1

- a) Descrivere l'algoritmo di Kruskal: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità e come viene calcolata (scrivere sul retro del foglio).
- b) Applicarlo al grafo in figura indicando il grafo risultante.
- c) Indicare un albero di copertura non minimo.



Esercizio 2

Scrivere una funzione c++ che, dato un albero binario con etichette intere e un intero x, conti il numero di nodi la cui etichetta è uguale al numero di etichette uguali a x che compaiono nel suo sottoalbero.

```
int conta(Node* tree, int x, int & quanti_x) {
    if (!tree) { quanti_x=0; return 0; }
    int cl, cr, ql, qr;
    cl=conta(tree->left, x, ql);
    cr=conta(tree->right, x, qr);
    quanti_x=ql+qr+(tree->label==x);
    return cl+cr+(tree->label==quanti_x);
}
```

Esercizio 3

Scrivere una funzione c++ che, dato un albero generico memorizzato figlio-fratello con etichette intere, aggiunga ad ogni nodo un ultimo figlio con etichetta uguale a quella del padre.

```
void aggiungi(Node* & tree, int x) {
    if (!tree)
        {tree=new Node;
         tree->label=x;
         tree->left=tree->right=0;
        }
    aggiungi(tree->right, x);
}

void f(Node* tree) {
    if (!tree) return;
    f(tree->left);
    f(tree->right);
    aggiungi(tree->left, tree->label);
}
```

Esercizio 4

Calcolare la complessità del **for** in funzione di $n > 0$.

```
for (int i=0; i <= f(n); i++) cout << f(n)+g(n);
```

con le funzioni **f** e **g** definite come segue. Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

<pre>int g(int x) { if (x<=0) return 1; int a = 0; for (int i=0; i<=x;i++) a+= i; int b = 2*g(x/2); cout << a + g(x/2); return 1 + 2*b; }</pre>	<pre>int f(int x) { if (x<=0) return 1; int a = g(x); for (int i=0; i <= x*x*x; i++) cout << i; return a + f(x-1); }</pre>
---	--

Funzione g

Calcolo for:

numero iterazioni: $O(n)$

Complessità della singola iterazione: $O(1)$

Complessità del for: $O(n)$

$Tg(0)=d$

$Tg(n)= cn + 2Tg(n/2)$ $Tg(n) \in O(n\log n)$

$Rg(n)= 1$ $Rg(n) \in O(n^2)$

$Rg(n)= 1+4 Rg(n/2)$

Funzione f

Calcolo for:

numero iterazioni: $O(n^3)$

Complessità della singola iterazione: $O(1)$

Complessità del for: $O(n^3)$

$Tf(0)=d$ $Tf(n) \in O(n^4)$

$Tf(n)= n^3+ Tf(n-1)$

$Rf(0)= 1$ $Rf(n) \in O(n^3)$

$Rf(n)= n^2+ Rf(n-1)$

Calcolo for del blocco:

numero iterazioni: $O(n^3)$

Complessità della singola iterazione: $Tf(n) + Tg(n) = O(n^4) + O(n\log n) = O(n^4)$

Complessità del for: $= O(n^7)$

Esercizio 5

Descrivere la teoria della NP-completezza: indicare il significato degli insiemi P e NP, la relazione fra I due insiemi, il teorema di Cook, la riducibilità fra problemi, I problemi NP-completi.

Esercizio 6

a) Indicare l'output del programma seguente

```
class alpha {
protected:
    int a;
public:
    alpha(){a=8;
    cout << "nuovo alpha " << endl;
}
void virtual f()=0;
void g(){cout << a << endl; }
};

class beta: public alpha {
protected:
    int a;
public:
    beta(){a=5;
    cout << "nuovo beta " << endl;
}
void virtual f() { cout << a << endl; }
void virtual g(){cout << a << endl; }
};

class delta: public beta {
protected:
    beta ob;
public:
    delta(){
    cout << "nuovo delta " << endl;
}
void f() { cout << a << endl; }
void g(){cout << a+1 << endl; }
};

template<class T>
void funzione(T *obj){
    static int a;
    obj->f();
    obj->g();
    a++;
}

void funzione1(T1 *obj1, T2 *obj2){
    funzione(obj1);
    cout << endl;
    funzione(obj2);
    cout << endl;
    funzione<alpha>(obj2);
    cout << endl;
}

void main(){
    delta *obj1= new delta;
    alpha *obj2=obj1;
    beta *obj3=obj1;
    funzione1(obj3, obj2);
}

nuovo alpha
nuovo beta
nuovo alpha
nuovo beta
nuovo delta
5
6
1
5
8
1
5
8
2
```

b) Con le classi definite nell'esercizio 1, date le seguenti istruzioni:

```
delta *p1=new delta;  
delta obj1;  
alpha *p2=new delta;
```

dire quali sono giuste e quali errate fra le seguenti istruzioni, dandone la motivazione:

1. alpha *p3=p1; ok
2. beta *p3=p2; no: non c'è conversione implicita da puntatore a superclasse a puntatore a sottoclasse
3. beta *p4=&obj1; ok
4. beta obj2=obj1; ok
5. alpha obj3; no: non si può instanziare una classe astratta
6. cout << obj1.ob.f(); no: obj1 e' protetto e f è void
7. funzione1<alpha,alpha>(p1,p2); ok
8. funzione1<beta,beta>(p1,p2); no: non c'è conversione implicita da puntatore a superclasse a puntatore a sottoclasse

Algoritmi e Strutture dati - ANNO ACCADEMICO 2016/17
15 settembre 2017

1	2	3	4	5	6
6	5	6	5	6	5

Esercizio 1

- a) Descrivere l'algoritmo PLSC: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità e come viene calcolata (scrivere sul retro del foglio).
- b) Applicarlo alle due sequenze di caratteri : MUMMIA e MIAMIMA, indicando il contenuto della matrice in figura e la/le PLSC.

		M	U	M	M	I	A
0	0	0	0	0	0	0	0
M	0	1	1	1	1	1	1
I	0	1	1	1	1	2	2
A	0	1	1	1	1	2	3
M	0	1	1	2	2	2	3
I	0	1	1	2	2	3	3
M	0	1	1	2	3	3	3
A	0	1	1	2	3	3	4

Lunghezza massima : 4

PLSC : MMIA MMMA

- c) Fare un esempio di 2 sequenze di 3 caratteri ciascuna in cui la matrice contiene 1 in tutte le righe e le colonne a parte la prima riga e la prima colonna:

	a	d	e
a	0	0	0
b	0	1	1
c	0	1	1

Esercizio 2 Scrivere una funzione c++ che, dato un albero binario con etichette intere e un intero x, conti il numero di nodi che hanno fra i discendenti una e una sola etichetta uguale a x. Calcolare la complessità.

```
int conta(Node* tree, int x, int & quanti_x) {
    if (!tree) { quanti_x=0; return 0; }
    int cl, cr, ql, qr;
    cl=conta(tree->left, x, ql);
    cr=conta(tree->right, x, qr);
    quanti_x=ql+qr+(tree->label==x);
    return cl+cr+(ql+qr==1);
}
```

Esercizio 3 Scrivere una funzione c++ che, dato un albero generico memorizzato figlio-fratello, per ogni nodo sposta l'ultimo sottoalbero del nodo al primo posto lasciando inalterati gli altri sottoalberi (per ogni nodo l'ultimo sottoalbero diventa il primo).

```
void modifica(Node* & tree) {
    if (!tree) return;
    if (tree->left) scambia(tree->left);
    modifica(tree->left);
    modifica(tree->right);
}
```

```
void scambia(Node* & tree) {
    if (!tree->right) return;
    Node* a=tree;
    for(Node * penultimo=tree; penultimo->right->right;
        penultimo=penultimo->right);
    Node* ultimo=penultimo->right;
    penultimo->right=0;
    tree=ultimo;
    ultimo->right=a;
}
```

Esercizio 4

Calcolare la complessità in funzione di $n>0$ delle espressioni $f(g(n))$ e $g(f(n))$.

con le funzioni f e g definite come segue. Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

<pre>int g(int x) { if (x<=0) return 1; int a = 0; int b = 4*g(x/2); cout << a + 3*g(x/2); return 1 + 2*b; }</pre>	<pre>int f(int x) { if (x<=0) return 1; for (int i=0; i <= g(x); i++) cout << i; return f(x-1); }</pre>
---	---

$Tg(0) = d$
 $Tg(n) = c + 2Tg(n/2)$ $Tg(n)$ è $O(n)$

$Rg(n) = 1$ $Rg(n)$ è $O(n^3)$
 $Rg(n) = 1 + 8Rg(n/2)$

Funzione f

Calcolo for:
numero iterazioni: $O(n^3)$
Complessità della singola iterazione: $O(n)$
Complessità del for: $O(n^4)$

$Tf(0) = d$ $Tf(n)$ è $O(n^5)$
 $Tf(n) = n^4 + Tf(n-1)$

$Rf(n)$ è $O(1)$

Calcolo $f(g(n))$: $Tg(n) + Tf(n^3) = O(n) + O(n^{15}) = O(n^{15})$

Calcolo $g(f(n))$: $Tf(n) + Tg(1) = O(n^5) + O(1) = O(n^5)$

Esercizio 5

- a) Considerare gli algoritmi di ordinamento visti a lezione, indicando per ciascuno di essi una descrizione a parole e la complessità.
- b) Applicare il quicksort all'array con il contenuto seguente, indicando tutte le chiamate a quicksort generate dalla chiamata iniziale e, per ciascuna di esse, il contenuto dell'array e il perno.

4	15	2	3	21	7
---	----	---	---	----	---

						inf	sup	perno
4	15	2	3	21	7	0	5	2
2	15	4	3	21	7	1	5	3
2	3	4	15	21	7	2	5	15
2	3	4	7	21	15	2	3	4
2	3	4	7	21	15	4	5	21

- a) Indicare un array di 3 elementi contenente i numeri 1, 2, 3 per la quale il quicksort raggiunge la massima efficienza

1	2	3
---	---	---

- b) Indicare un array di 3 elementi contenente i numeri 1, 2, 3 per la quale il quicksort raggiunge la minima efficienza.

3	1	2
---	---	---

Esercizio 6

Indicare l'output del programma seguente:

```

class ecc1{
protected:
int x;
public:
ecc1(){ x= 5; cout << "ecc1" << endl;};
int h(){ return x; };
};

class ecc2: public ecc1{
protected:
int x;
public:
ecc2(){ x= 6; cout << "ecc2" << endl;};
int h(){ return x; };
};

int g(int x) {
try {
cout << 100 << endl;
if (x==2) { ecc2* e=new ecc2; throw e;};
if (x==3) throw 3;
}
catch (ecc2* e) { cout << e->h() +1
<< endl; }
catch (...) { throw; }
cout << "fine g" << endl;
};

int f(int x) {
try {
if (x==1) { ecc1* e=new ecc1; throw e;};
cout << 600 << endl;
g(x);
if (x==2) { ecc2* e=new ecc2; throw e;};
}
catch (ecc2* e) { cout << e->h()<< endl; }
catch (ecc1* e) { cout << 90 << endl; }
catch(int) { cout << 80 << endl; }
cout << "fine f" << endl;
};

int main () {
f(1);
cout << endl;
f(2);
cout << endl;
f(3);
}

```

ANNO ACCADEMICO 20016/2017- 16 gennaio 2018
Algoritmi e Strutture Dati

1	2	3	4	5	6
6	5	6	6	5	5

Esercizio 1

- a) Descrivere l'algoritmo di Kruskal: a cosa serve, come funziona, qual è la sua complessità e come viene calcolata.
- b) Si consideri la memorizzazione di un albero generico (senza etichette) mediante un array come nell'algoritmo di Kruskal, in cui cioè ogni elemento dell'array è un nodo e contiene l'indice del padre. I nodi sono numerati da 0 a n-1 e t[n]=-1 se n è la radice. Supponendo di avere un array `tree` con n elementi, scrivere dei frammenti di codice per le seguenti richieste.
 1. Stampare il padre del nodo x
 2. Stampare tutti figli del nodo x
 3. Stampare tutti gli antenati del nodo x
 4. Stampare tutti i fratelli del nodo x

SOLUZIONE

```

1. if (t[x] != -1) cout << t[x];
2. for (int i=0; i<n; i++) if (t[i] == x) cout << i;
3. for (int i=t[x]; i!= -1; i= t[i]) cout << i;
4. for (int i=0; i<n; i++) if (t[i] == t[x] && i!= x) cout << i;

```

Esercizio 2

Calcolare la complessità del blocco in funzione del numero di nodi dell'albero binario t:

```

{
  int a = 0;
  for (int i=0; i <= Nodes(t); i++)
    {a += f(t);
     es(t);
    }
}

```

con le funzioni `f` e `es` definite come segue. Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

```

void es (Node* t ) {
  if (!t) return;
  t->label+= f (t);
  es (t->left);
  es (t->left);
  es (t->right);
  es (t->right);
}

int f(Node* t) {
  if (!t) return 1;
  for (int i=0; i <= Nodes(t); i++)
    cout << Nodes(t);
  return 1 + f(t->left);
}

```

Funzione f

Numero di iterazioni del for: n
 Complessità della singola iterazione: O(n)
 Complessità del for: =O(n^2)

$T_f(0)= d$
 $T_f(n)= cn^2+ T_f(n/2)$ $T_f(n)$ è $O(n^2)$

Funzione es

$T_{es}(0)=d$
 $T_{es}(n)= cn^2+ 4 T_{es}(n/2)$ $T_{es}(n)$ è $O(n^2 \log n)$

Calcolo for del blocco:
 numero iterazioni: n
 Complessità della singola iterazione: $T_{nodes}(n) + T_f(n) + T_{es}(n) = O(n) + O(n^2) + O(n^2 \log n)$
 = $O(n^2 \log n)$
 Complessità del for: $O(n^3 \log n)$

Esercizio 3

Scrivere una funzione che, dato un albero binario con etichette intere, restituisce il numero di nodi dell'albero che hanno almeno un nodo con etichetta 100 in entrambi i sottoalberi.

```

int conta (Node* t, bool & cento) {
  if (!t) { cento=false; return 0; }
  int conta_l, conta_r;
  bool cento_l, cento_r;
  conta_l=conta(t->left, cento_l);
  conta_r=conta(t->right, cento_r);
  cento = (cento_l || cento_R || t->label == 100);
  return conta_l + conta_r + (cento_l && cento_r)
}

```

Esercizio 4

Scrivere una funzione **void esau(Node* t)** che, dato un albero generico memorizzato con la memorizzazione figlio-fratello, scambia il primo con il secondo sottoalbero di ogni nodo che ha almeno due sottoalberi.

```
void esau (Node* t) {
    if (!t) return;
    if (t->left && t->left->right) {
        Node* temp= t->left->right;
        t->left->right= temp->right;
        temp->right=t->left;
        t->left=temp;
    }
    esau (t->left);
    esau (t->right);
}
```

Esercizio 5

a) Eseguire il seguente programma e indicare le istanze della funzione f generate

```
#include<iostream>
using namespace std;
template<class tipo1, class tipo2>
class uno {
    tipo1 a;
    tipo2 b;
public:
    uno(tipo1 x, tipo2 y) { a=x; b=y; }
    tipo1 valore_a() { return a; }
    tipo2 valore_b() { return b; }
};
template<class T1, class T2>
void f(uno<T1,T2> obj){
    cout << obj.valore_a();
    cout << obj.valore_b();
}
int main() {
    uno<int, int> obj1(5,6);
    uno<char, char> obj2('m', 'n');
    uno<char, int> obj3('a', 7);
    f(obj1);           // L1
    f(obj2);           // L2
    f(obj3);           // L3
}
```

5 6 m n a 7

Istanze di f:

L1: void f<int,int>
L2: void f<char, char>
L3: void f<char, int>

b) Cosa cambia se la funzione f viene sostituita con la seguente funzione?

```
template<class T>
void f(uno<T,T> obj){
    cout << obj.valore_a();
    cout << obj.valore_b();
}
```

Errore a tempo di compilazione nell'istruzione L3 (chiamata f(obj3)); non si ricava il tipo T (non c'è conversione di tipo)

Esercizio 6

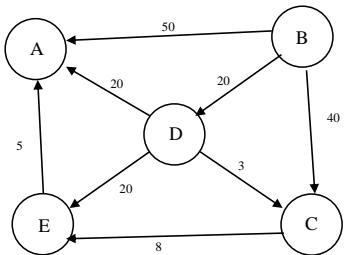
Descrivere la teoria della NP-completezza: cosa sono P e NP, il teorema di Cook, cosa vuol dire che un problema è NP-completo. Fare un esempio di problema NP-completo.

ANNO ACCADEMICO 20016/2017- 6 febbraio 2018
Algoritmi e Strutture Dati

1	2	3	4	5	6
5	6	5	6	6	5

Esercizio 1

- a) Descrivere l'algoritmo di Dijkstra: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità e come viene calcolata (scrivere sul retro del foglio).
- b) Applicarlo al grafo seguente a partire dal nodo B.



	A	B	C	D	E
A, B, C, D, E	inf -	0 -	inf -	inf -	inf -
A, C, D, E	50 B	0 -	40 B	20 B	inf -
A, C, E	40 D	0 -	23 D	20 B	40 D
A, E	40 D	0 -	23 D	20 B	31 C
A	36 E	0 -	23 D	20 B	31 C

Esercizio 2

- a) Descrivere il metodo di ricerca hash a indirizzamento aperto e con concatenazione: come funziona, cosa sono le collisioni, gli agglomerati, le leggi di scansione, da quali parametri dipende la velocità di ricerca.
- b) Sia data la seguente tabella hash a indirizzamento aperto con funzione hash modulare e scansione lineare. Si supponga che la tabella sia inizialmente vuota. Mostrare il contenuto della tabella dopo le operazioni indicate sulle colonne:

	contenuto iniziale	inserimento 356	inserimento 41	inserimento 287	eliminazione 41	inserimento 69
0	-1	-1	41	41	-2	69
1	-1	-1	-1	287	287	287
2	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	-1
6	-1	356	356	356	356	356

Esercizio 3

Calcolare la complessità in funzione di $n > 0$ dell'istruzione
 $y = g(f(n))$;

con le funzioni f e g definite come segue:

<pre>int f(int x) { if (x<=1) return 1; int b=0, i; for (i=1; i<=x; i++) b+=i; cout << b*b*b; return f(x-1)+ 4 + b; }</pre>	<pre>int g(int x) { if (x<=1) return 10; int a=0; for (int i=0; i<f(x)*f(x); i++) a++; return 10+g(x/2)+g(x/2); }</pre>
---	---

Indicare le eventuali relazioni di ricorrenza e spiegare brevemente il calcolo della complessità dei cicli.

Stima del tempo di f

numero iterazioni del for = $O(n)$
 complessità di un'iterazione = costante
 tempo del for = $O(n)$

$T_f(1)=d$
 $T_f(n)=c \cdot n + T_f(n-1)$

$T_f \in O(n^2)$

$R_f(n) = cn^2 + R_f(n-1)$
 $R_f(n) \in O(n^3)$

Stima del tempo di g :
 numero iterazioni del for: $R_f(m)*R_f(m) = O(m^6)$
 complessità di un'iterazione: $T_f(m) = O(m^2)$
 tempo del for: $O(m^8)$

tempo di g
 $T_g(1)=\text{cost}$
 $T_g(m)=c \cdot m^8 + 2T_g(m/2)$
 $T_g \in O(m^8)$

stima del risultato di g
 $R_g(1)=\text{cost}$
 $R_g(m)=\text{cost} + 2R_g(m/2)$
 $R_g(m)=O(m)$

Complessità dell'istruzione:
 $C[f(n)] + C[g(R_f(n))] = O(n^2) + C[g(n^3)] = O(n^2) + O(n^24) = O(n^{24})$

Esercizio 4

Scrivere una funzione booleana che, dato un albero generico non vuoto ad etichette di tipo int, memorizzato figlio-fratello, restituisce true se nell'albero c'è almeno un nodo tale che il suo primo e il suo ultimo figlio hanno la stessa etichetta. N.B: un nodo con un solo figlio verifica la condizione.

```
bool p_u(Node* t) {
    if (!t) return 0;
    if (!t->left) return p_u(t->right);
    if (t->left->label==ultimo(t->left)) return true;
    return (p_u(t->left)||p_u(t->right));
}

int ultimo(Node* t) {
    if (!t->right) return t->label;
    ultimo(t->right);
}
```

Esercizio 5

Siano dati due alberi binari ad etichette intere, con puntatori alla radice t1 e t2. Supponendo che gli alberi siano identici in quanto a struttura (ma non necessariamente rispetto al contenuto delle etichette), scrivere una funzione Node* sottrai(Node* t1, Node* t2) che prende in ingresso t1 e t2 e restituisce un puntatore alla radice di un terzo albero, anch'esso identico per struttura a t1 e t2, ma dove ogni nodo ha come etichetta la differenza delle etichette dei nodi nella stessa posizione in t1 e t2.

```
Node* sottrai(Node* t1, Node* t2) {
    if (!t1)
        return NULL;
    Node* n = new Node();
    n->label = t1->label - t2->label;
    n->left = sottrai(t1->left, t2->left);
    n->right = sottrai(t1->right, t2->right);
    return n;
}
```

Esercizio 6

Indicare l'uscita del seguente programma

- a) Così come è scritto
- b) Sostituendo l'istruzione indicata con ** con l'istruzione: alpha *o=new alpha(80);
- c) Spiegare la differenza

```
template<class tipo>
void funzione(tipo * obj){
    obj->g();
}

class alpha {
protected:
    int a;
public:
    alpha(){a=10; cout << a << " nuovo alpha " << endl; a++;}
    alpha(int x){a=x; cout << a << " nuovo alpha " << endl; a++;}
    void g(){cout << a+1 << endl; }
};

class beta: public alpha {
int a;
alpha *o=new alpha();    **
public:
    beta(){a=24; cout << a << " nuovo beta " << endl;}
    void g(){cout << a+2 << endl;}
    void h(){funzione<alpha>(o); cout << a+2 << endl;}
};

int main(){
    beta *obj1= new beta;
    alpha *obj2=obj1;
    funzione(obj1);
    funzione(obj2);
    obj1->h();
}
```

a)

```
10 nuovo alpha
10 nuovo alpha
24 nuovo beta
26
12
12
26
```

b)

```
10 nuovo alpha
80 nuovo alpha
24 nuovo beta
26
12
82
26
```

- c) Vengono chiamati due diversi costruttori della classe alpha.

Algoritmi e Strutture Dati
Anno accademico 20016/2017- 22 febbraio 2018

1	2	3	4	5	6
5	6	5	6	6	5

Esercizio1

- a) Definire quando una funzione è O di un'altra.
- b) Confrontare le due funzioni F e G dal punto di vista della complessità: dire se una è O dell'altra e viceversa. In caso affermativo, indicare una coppia (n0,c), in caso negativo, giustificare la risposta.

$$\begin{array}{ll} 3x^2 & x \text{ pari} \\ F(x)= & G(x)= \\ 50x^3 & x \text{ dispari} \end{array} \quad \begin{array}{ll} 9x^2 & x \text{ divisore di 255} \\ & \\ x^3 & \text{altrimenti} \end{array}$$

F è O(G) n0=256, c=51

G non è O(F) perché ci sono infiniti numeri pari dove G vale x^3 e F $3x^2$.

- c) Fare un esempio di funzioni incommensurabili.

Esercizio2

- a) Descrivere il tipo di dato heap: definizione, operazioni e loro complessità, memorizzazione.
- b) Un array ordinato in ordine decrescente è sempre uno heap? Se sì, dimostrarlo, se no, mostrare un contro esempio. **SI, perché ogni elemento è maggiore o uguale dei suoi due figli.**
- c) Uno heap è sempre un array ordinato in ordine decrescente? Se sì, dimostrarlo, se no, mostrare un contro esempio. **NO, vedi ad esempio lo heap [9 3 4]**
- d) Sia dato lo heap [50 28 15 22 9 10 8]; indicare lo heap dopo una estrazione e il successivo inserimento del valore 16. Indicare per ogni operazione le chiamate a up e down

28 22 15 8 9 10	down(0), down(1), down(3)	Dopo l'estrazione
28 22 16 8 9 10 15	up(6), up(2)	Dopo l'inserimento di 16

Esercizio3

Calcolare la complessità del for in funzione di n > 0.

```
for (int i=0; i <= g(n)+f(n); i++) cout << i;
```

con le funzioni f e g definite come segue. Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

```
int g (int x ) {
    if (x<=0) return 1;
    int a=0; b=0;
    for (int i=0; i <= x; i++) a+=i;
    for (int j=a; j >= 0;j--) b +=j;
    return b/a + g (x-1);
}
```

```
int f(int x) {
    if (x<=0) return 1;
    int a=g(x);
    int b= 5 + 2*f(x/2);
    for (int i=0; i <= a; i++)
        cout << i;
    return a*a + 2*f(x/2);
}
```

Funzione g

Primo for :

Complessità della singola iterazione: O(1)
 Numero di iterazioni : O(n)
 Complessità del for: =O(n)

Secondo for :

Complessità della singola iterazione: O(1)
 Numero di iterazioni : O(n^2)
 Complessità del for: =O(n^2)

Tg(0)=d
 Tg(n)= cn^2+ Tg(n-1) Tg(n) è O(n^2)

Rg(0)=d
 Rg(n)= cn^2+ Rf(n-1) Rg(n) è O(n^3)

Funzione f

for :

Complessità della singola iterazione: O(1)
 Numero di iterazioni : O(n^3)
 Complessità del for: =O(n^3)

Tf(0)=d
 Tf(n)= cn^3+ 2Tf(n/2) Tf(n) è O(n^3)

Rf(0)=d
 Rf(n)= cn^6+ 2Rf(n/2) Rf(n) è O(n^6)

Calcolo for del blocco:

numero iterazioni: g(n) + f(n)= O(n^3) + O(n^6)= O(n^6)
 Complessità della singola iterazione: Tg(n) + Tf(n) = O(n^3) + O(n^3) = O(n^3)
 Complessità del for: =O(n^9)

Esercizio 4

- a) Definire il tipo di dato “albero binario”.
- b) Scrivere una funzione che, dato un albero binario t con etichette intere e due etichette x e y, conta i nodi che hanno più nodi con etichetta x che nodi con etichetta y nel proprio sottoalbero.

```
int conta (Node* t, int x, int y, int & quantix, int & quantiy) {
    if (!t) {quantix=0; quantiy=0; return 0; }
    int quantix_l, quantix_r, quantiy_l, quantiy_r;
    int l = conta(t->left, quantix_l, quantiy_l);
    int r = conta(t->right, quantix_r, quantiy_r);
    quantix= quantix_l + quantix_r + (t->label==x);
    quantiy= quantiy_l + quantiy_r + (t->label==y);
    return l + r + (quantix > quantiy);
}
```

Esercizio 5

- a) Definire il tipo di dato “albero generico”.
- b) Scrivere una funzione che, dato un albero generico (memorizzato figlio-fratello) e una etichetta x, elimina il primo figlio di x, se esiste, e inserisce i suoi eventuali figli come figli di x.

```
void elimina (Node* t, int x) {
    Node * a= find(x,t);
    if (!a || !a->left) return;
    Node * b=a->left;
    if (!b->left) {
        a->left = b->right;
        b->right=0;
        delete b;
        return;
    }
    a->left = b->left;
    for (c=b->left; c->right; c=c->right);
    c->right = b->right;
    b->right=b->left=0;
    delete b;
}
```

Esercizio 6

- a) Spiegare il funzionamento delle funzioni virtuali in c++;
- b) Indicare l'output del seguente programma
 - 1) Così come è scritto
 - 2) Togliendo tutte le occorrenze dalla parola `virtual`

```
#include <iostream>
using namespace std;

class A {
public:
    int i;
    A(int i) : i(i) {
        cout << "A(" << i << ")" <<
endl;
    }
    virtual int f() { return i; }
    virtual ~A() {
        cout << "delA(" << i << ")"
<< endl;
    }
};

class B : public A {
public:
    int j;
    B(int f) : A(f+2) { j=90;
        cout << "B(" << j << ")" <<
endl;
    }
    int f() { return 50; }
    virtual ~B() { cout << "delB()" <<
endl; }
};

int main() {
    C* c = new C(10);
    A* ref = c;
    cout << ref->i << endl;
    cout << c->i << endl;
    cout << c->f() << endl;
    cout << ref->f() << endl;
    delete ref;
}

1)
A(11)
B(90)
C(10)
11
10
1010
1010
delC()
delB()
delA(11)

2)
A(11)
B(90)
C(10)
11
10
1010
1010
11
delA(11)
```

12 giugno 2018

I	2	3	4	5	6
6	6	6	5	5	5

Esercizio 1

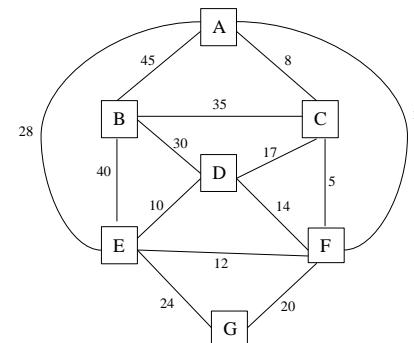
- a) Descrivere l'algoritmo di Huffman: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità.
- b) Applicarlo all'alfabeto in figura in cui ad ogni carattere corrisponde la sua frequenza percentuale nel testo, indicando l'albero risultante e la codifica di ogni carattere (per ogni nodo dell'albero etichettare con 0 l'arco che lo congiunge al figlio con etichetta minore)
- c) Se un alfabeto ha 2^n simboli e tutti hanno la stessa frequenza, che caratteristica hanno le relative codifiche? **Sono tutte di lunghezza uguale a n**

A	13
B	16
C	7
D	15
E	17
F	20
G	4
H	8

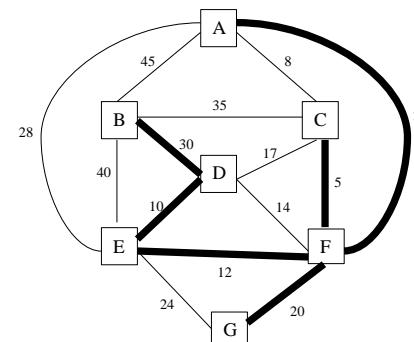
A	100
B	110
C	0011
D	101
E	111
F	01
G	0010
H	000

Esercizio 2

- a) Dare la definizione di minimo albero di copertura di un grafo non orientato.
- b) Descrivere a parole l'algoritmo di Kruskal per trovare il minimo albero di copertura. Descrivere come è implementato, indicare e spiegare la sua complessità.
- c) Applicarlo al grafo seguente indicando l'ordine in cui sono presi gli archi appartenenti al minimo albero di copertura.
- d) Dato un minimo albero di copertura per un grafo G questo contiene un cammino minimo per ogni coppia di nodi in G? Dimostrare la risposta. **NO**
- e) Tutti gli alberi di copertura (sia minimi che non) di un grafo G hanno lo stesso numero di archi? Dimostrare la risposta. **Tutti gli alberi di copertura hanno (n-1) archi**



(A, F) (F, C) (E, D) (E, F) (G,F) (B,D)



Esercizio 3

Calcolare in funzione di $n \geq 0$ la complessità del blocco

```
for (int i=0; i <= f(n)*g(n); i++) a+= 1;
```

con le funzioni **f** e **g** definite come segue:

int f(int x) {	int g(int x) {
if (x==0) return 1;	if (x==0) return 1;
int y=0;	int a=0;
for (int i=1;i<= g(x) ;i++) y+=1;	for (int i=1;i<=x*x;i++)
int b = 1 + 4*f(x/2);	{ a++;
return y+b;	cout << a;
}	}
	return a/x + g(x-2) ;

Indicare le relazioni di ricorrenza e il numero di iterazioni e la complessità di ogni iterazione per i comandi ripetitivi.

Funzione g

Numero iterazioni del for: $O(n^2)$

Complessità di una iterazione: $O(1)$

Complessità del for: $O(n^2)$

$Tg(0)= a$

$Tg(n)= bn^2+Tg(n-2) \quad O(n^3)$

$Rg(0)= 1$

$Rg(n)= bn+Rg(n-2) \quad O(n^2)$

Funzione f

Numero iterazioni del for: $O(n^2)$

Complessità di una iterazione: $O(n^3)$

Complessità del for: $O(n^5)$

$Tf(0)= a$

$Tf(n)= bn^5+Tf(n/2) \quad O(n^5)$

$Rf(0)= 1$

$Rf(n)= bn^2+4Tf(n/2) \quad O(n^2 \log n)$

blocco

Numero iterazioni del for: $Rf(n)*Rg(n) = O(n^2 \log n) * O(n^2) = O(n^4 \log n)$

Complessità di una iterazione: $Tff(n)+Tg(n) = (n^5) + O(n^3) = O(n^5)$

Complessità del for: $O(n^9 \log n)$

Esercizio 4

a) Dare la definizione di albero binario.

b) Scrivere una funzione in c++ con complessità $O(n)$ che, dato un albero binario con etichette intere, conta il numero di nodi la cui etichetta è uguale al numero di foglie che ci sono nel suo sottoalbero.

```
int conta (Node* t, int & leaves) {
    if (!t) {leaves=0; return 0;}
    int leaves_l, leaves_r, conta_l, conta_r;
    conta_l=conta(t->left,leaves_l);
    conta_r=conta(t->right,leaves_r);
    leaves=leaves_l+leaves_r + (t->left==0 && t->right==0);
    return (t->label==leaves)+ conta_l + conta_r;
}
```

Esercizio 5

a) Dare la definizione di albero generico.

b) Scrivere una funzione **int conta (Node*tree)** che, dato un albero generico memorizzato figlio-fratello, conta quanti nodi hanno un numero pari di nipoti (figli dei figli). Indicare la relazione di ricorrenza di **conta**.

```
int contafigli (Node*tree) {
    if (! tree) return 0;
    return 1 + contafigli( tree->right);
}

int containipoti (Node*tree) {
    if (! tree) return 0;
    return contafigli(tree->left) + containipoti( tree->right);
}

int conta (Node*tree) {
    if (! tree) return 0;
    return (containipoti (tree->left)%2==0)
        + conta( tree->left) + conta( tree->right);
}
```

Esercizio 6 Sia dato il seguente programma c++.

```

class A {
public:
    A(){cout << "nuovo A" << endl; };
    void virtual f()=0;
    void g() {cout << "g di A" << endl; }
};

class B: public A {
public:
    B(){cout << "nuovo B" << endl; };
    void f() {cout << "f di B" << endl; }
    void g() {cout << "g di B" << endl; }
};

class C: public A {
public:
    C(){cout << "nuovo C" << endl; };
    void f(){cout << "f di C" << endl; }
    void g() {cout << "g di C" << endl; }
};

class D: public C {
public:
    D(){cout << "nuovo D" << endl; };
    void f(int x){cout << x << endl; }
};

```

- Indicare la sua uscita
- Indicare l'uscita aggiungendo “virtual” alla g() di A
- Spiegare le eventuali differenze fra i casi sopra citati.
- Spiegare perché aggiungendo una qualsiasi delle seguenti istruzioni alla fine del main il compilatore dà errore: vet[1]= new A; vet[2]->f(3); D* obj= vet[2];

a)
 nuovo A
 nuovo B
 nuovo A
 nuovo C
 nuovo A
 nuovo C
 nuovo D
 f di B
 g di A
 f di C
 g di A
 f di C
 g di A

b)
 nuovo A
 nuovo B
 nuovo A
 nuovo C
 nuovo A
 nuovo C
 nuovo D
 f di B
 g di B
 f di C
 g di C
 f di C
 g di C

d)
 vet[1]= new A;
 vet[2]->f(3);
 D* obj= vet[2]

non si può istanziare una classe astratta
 vet[2] è di tipo A che è una classe che non ha la funzione f(int)
 conversione da supertipo a sottotipo non ammessa

3 luglio 2018

I	2	3	4	5	6	7
5	5	5	6	5	5	2

Esercizio 1

- a) Dare la definizione di albero binario bilanciato e quasi bilanciato.
- b) Scrivere una funzione in c++ con complessità O(n), dove n è il numero di nodi, che, dato un albero binario e un intero x, conta i nodi che hanno esattamente x sottoalberi vuoti nel proprio sottoalbero.

```
int conta (Node* t, int x, int & vuoti) {
    if (!t) {vuoti=1; return 0;}
    int vuoti_l, vuoti_r, conta_l,conta_r;
    conta_l=conta(t->left, vuoti_l);
    conta_r=conta(t->right, vuoti_r);
    vuoti= vuoti_l + vuoti_r;
    return (vuoti==x) + conta_l + conta_r;
}
```

Esercizio 2

- a) Descrivere la memorizzazione figlio-fratello dell'albero generico.
- b) Che relazioni ci sono fra le visite di un albero generico e la visite del suo “trasformato”?
- c) Scrivere una funzione in c++ che, dato un albero generico, scambia il primo con l'ultimo sottoalbero di ogni nodo.

```
void scambia_Figli (Node* & t) {
    if (!t) return;
    if (!t->right) return;
    for (Node * a=t; a->right->right; a=a->right);
    Node * b= t;
    t=a->right;
    t->right=b->right;
    a->right=b;
    b->right=0;
}

void scambia (Node* t) {
    if (!t) return;
    scambia_figli(t->left);
    scambia (t->left);
    scambia (t->right);
}
```

Esercizio 3

- a) Dare la definizione del tipo di dato heap con le operazioni e relative complessità
- b) Dato lo heap [80, 75, 75, 60, 20, 10, 40] indicarne il contenuto dopo l'inserimento del valore 78 e una successiva estrazione.

heap	Chiamate a up e down	
80, 78, 75, 75, 20, 10, 40, 60	up(7), up(3), up(1)	Dopo l'inserimento di 78
78, 75, 75, 60, 20, 10, 40	down(0), down(1), down(3)	Dopo un'estrazione

- c) Le rappresentazioni tramite array di due heap che hanno gli stessi elementi sono uguali? Dimostrare la risposta. NO (es [20 10 13] e [20 13 10])
- d) Dove si trova l'elemento più piccolo di un heap ? nelle foglie: a partire da da n/2

Esercizio 4

- a) Descrivere brevemente l'algoritmo di ordinamento quicksort e indicare la sua complessità.
- b) Applicare il quicksort all'array seguente indicando tutte le chiamate e i il valore del perno per ogni chiamata.
- c) La complessità del quicksort è sempre la stessa qualsiasi sia la configurazione dei dati nell'array? Spiegare.
- d) Cosa vuol dire che un algoritmo è ottimo? Quicksort è un algoritmo di ordinamento ottimo? Spiegare la risposta.

Array	inf	sup	perno
8, 3, 15 , 4, 6	0	4	15
8, 3, 6 , 4, 15	0	3	3
3, 8, 6, 4, 15	1	3	6
3, 4, 6, 8, 15			

Esercizio 5

Sia *i* il puntatore ad una lista semplice di interi. Calcolare la complessità dell'istruzione
`for(int i=0; i<g(f(1)); i++) cout << "!";`
in funzione della lunghezza *n* di *i* (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni *f* e *g* definite come segue:

<pre>int f(Elem* h) { if (!h) return 1; int a=0; for(Elem* e=h; e; e=e->next) { e->inf++; a++; } int b=a*a; for(int i=1; i<=b; i++) { a++; } return a*a+f(h->next); }</pre>	<pre>int g(int x) { if (x<=1) return 1; int b = 1+2*g(x/2); return 2*b; }</pre>
---	--

Tempo di *f* (in funzione della lunghezza *n* di *h*)

1° per $O(n)$ n° iterazioni: *n*; complessità iterazione: $O(1)$
2° per $O(n^2)$ n° iterazioni: n^2 ; complessità iterazione: $O(1)$

$$\begin{aligned} T_f(n) &= a && \text{per } n=0 \\ T_f(n) &= b*n^2 + T_f(n-1) && \text{altrimenti} \\ T_f(n) &\in O(n^5) \end{aligned}$$

Risultato di *f*

$$\begin{aligned} R_f(n) &= a && \text{per } n\leq 1 \\ R_f(n) &= b*n^4 + R_f(n-1) && \text{altrimenti} \\ R_f(n) &\in O(n^5) \end{aligned}$$

Tempo di *g*

$$\begin{aligned} T_g(x) &= a && \text{per } x\leq 1 \\ T_g(x) &= b+T_g(x/2) && \text{altrimenti} \\ T_g(x) &\in O(\log x) \end{aligned}$$

Risultato di *g*

$$\begin{aligned} R_g(x) &= a && \text{per } x\leq 1 \\ R_g(x) &= b+4*R_g(x/2) && \text{altrimenti} \\ R_g(x) &\in O(x^2) \end{aligned}$$

Calcolo del for

Numeri iterazioni: $R_g(R_f(n)) = R_g(n^5) = O(n^{10})$
Complessità singola iterazione = $T_f(n) + T_g(n^5) = O(n^3) + O(\log n) = O(n^3)$

Complessità del for: $O(n^{10})*O(n^3)=O(n^{13})$

Esercizio 6

- Indicare l'output del seguente programma c++;
- Le istanze delle classi e funzioni modello vengono create a tempo di compilazione o di esecuzione? **Di compilazione**
- Quali istanze delle classi e funzioni modello vengono create?
`A<int>, A<double>, B<int>, B<double>, f<A<int>, f<A<double>>`

```
template <class T>
class B {
T x;
public:
B() { x = 2.3;
cout << "nuovo B" << endl;
}
void print () { cout << x << endl; }
};
template <class T>
class A {
B<T>* a;
public:
A() { a = new B<T>;
cout << "nuovo A" << endl;
}
int main()
{
A<int> obj1;
A<double> obj2;
f(obj1);
f(obj2);
}
```

a)
nuovo B
nuovo A
nuovo B
nuovo A
2
2.3

Esercizio 7

Indicare per sommi capi un algoritmo che controlla se due sequenze sono l'uno l'anagramma dell'altra.

Una possibile soluzione:

Step 1. Si ordinano le due sequenze separatamente $O(n \log n)$

Step 2. Si controlla se sono uguali confrontando element per element $O(n)$

Complessità: $O(n \log n)$

Algoritmi e Strutture dati - ANNO ACCADEMICO 2017/18

24 luglio 2018

I	2	3	4	5	6	7
5	5	5	6	5	3	4

Esercizio 1. Scrivere una funzione in c++, che, dato un albero binario, conta i nodi tali per cui il sottoalbero sinistro è di livello maggiore del sottoalbero destro.

```
int conta (Node* t, int $ level) {
    if (!t) {level=-1; return 0;}
    int l, r;
    int conta_l =conta(t->left, 1);
    int conta_r =conta(t->right,r);
    level=max(l,r)+1;
    return conta_l+conta_r + (l>r);
}
```

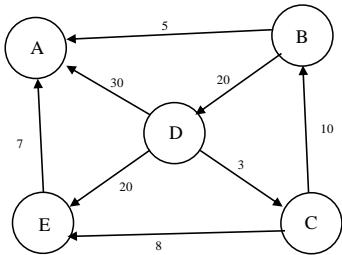
Esercizio 2

- Descrivere la differenza fra albero binario e albero generico. Con due nodi A e B quanti alberi binari si possono fare? 4 E quanti generici? 2 Disegnarli.
- Scrivere una funzione in c++ che, dato un albero generico memorizzato figlio-fratello, restituisce il numero dei nodi che hanno etichetta uguale a quella del padre.

```
int conta (Node* & t) {
    if (!t) return 0;
    return check(t->label, t->left)
        + conta(t->left) + conta (t->right);
}
int check (int padre, Node* t) {
    if (!t) return 0;
    return (t->label == padre)+ check(padre, t->right);
}
```

Esercizio 3

- Dare la definizione di grafo orientato.
- Descrivere i metodi di memorizzazione di un grafo orientato visti a lezione.
- Descrivere l'algoritmo di Dijkstra: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità e come viene calcolata (scrivere sul retro del foglio).
- Applicarlo al grafo seguente a partire dal nodo D:



	A	B	C	D	E
A, B, C, D, E	inf -	inf -	inf -	0 -	inf -
A, B, C, E	30 D	inf -	3 D	0 -	20 D
A, B, E	30 D	13 C	3 D	0 -	11 C
A, B	18 E	13 C	3 D	0 -	11 C
A	18 E	13 C	3 D	0 -	11 C

Cammini minimi: DCEA (o DCBA), DCB, DC, DCE

Esercizio 4

a) Dimostrare, utilizzando le regole di semplificazione delle espressioni O-grande, che la funzione $f(n)=7n^2+5n+4$ è $O(n^2)$

b) Calcolare la complessità in funzione di n dell'istruzione

$y=g(f(n));$

con le seguenti definizioni di funzione. Indicare le eventuali relazioni di ricorrenza e spiegare brevemente il calcolo della complessità dei cicli.

<pre>int f(int x) { if (x<=1) return 1; int a=0; int b=0; for (int i=1; i<= g(x); i++) a++; cout >> f(x-1)+a; return a; }</pre>	<pre>int g(int x) { if (x<=1) return 10; int b=0; for (int i=1; i<=x*x; i++) b++; return b+g(x/2)+g(x/2); }</pre>
---	---

Funzione g

numero iterazioni del for: $O(n^2)$
complessità di una iterazione: $O(1)$
tempo del for: $O(n^2)$

tempo di g
 $T(1)=d$
 $T(m)=c n^2 + 2T(n/2)$ $O(n^2)$

Risultato di g
 $R(1)=d$
 $R(m)=c n^4 + 2T(n/2)$ $O(n^4)$

Funzione f

numero iterazioni del for: $O(n^4)$
complessità di una iterazione: $O(n^2)$
tempo del for: $O(n^6)$

tempo di f
 $T(1)=d$
 $T(n)=cn^6 + T(n-1)$ $O(n^7)$

Risultato di f $O(n^4)$

Complessità dell'istruzione:
 $C[f(n)] + C[g(n^4)] = O(n^7) + O(n^8) = O(n^8)$

Esercizio 5

a) Indicare l'output del seguente programma c++;

```

class R {
protected:
    int z; int y;
public:
R() { z = 5; y=2;
    cout << "nuovo R" << endl;
    };
void print(){
    cout << z << endl;
    cout << y << endl;
    z++;
    y++;
}
};

class S : public R {
protected:
int y;
public:
S() { z++; y=3;
    cout << "nuovo S" << endl;
    };
void print (){
    cout << y << endl;
    y++;
    z++;
}
void virtual print1()=0;
};

class U: public S {
public:
U() { z=11; y=7;
    cout << "nuovo U" << endl;
    };
void print (){
    cout << z << endl;
    cout << y << endl;
    y++;
    z++;
}
void print1(){

}

```

nuovo R
nuovo S
nuovo U
11
7
8
13
2
14
9

b) Quale di queste istruzioni può essere aggiunta in coda al main? Spiegare.

S* obj3 = new S;	NO: non si può istanziare una classe astratta
R* obj4= new U;	OK: conversione da sottoclasse a superclasse
cout << obj2->z;	NO: z è protetta
obj1=obj2;	NO: conversione non permessa da superclasse a sottoclasse
obj2=obj1;	OK: conversione da sottoclasse a superclasse

Esercizio 6

Indicare per sommi capi un algoritmo con complessità minore di $O(n^2)$ che cancella da una lista semplice i doppioni.

Una possibile soluzione:

Step 1. Si ordina la lista $O(n\log n)$

Step 2. Si scorre la lista cancellando ogni elemento uguale al precedente $O(n)$

Complessità: $O(n\log n) + O(n) = O(n\log n)$

Esercizio 7

Spiegare brevemente la teoria della NP-completezza: gli insiemi P e NP, il teorema di Cook, i problemi NP-completi.

Parte II

Compiti pratici passati

Algoritmi e Strutture Dati – Prova di Laboratorio

11/01/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuati tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: **input0.txt** **output0.txt** **input1.txt** **output1.txt** ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file **output0.txt**. Per effettuare un controllo automatico sul primo file input **input0.txt** potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

[leggere il testo prestando particolare attenzione alle definizioni]
Si consideri un sistema di memorizzazione che legga una sequenza di N interi non negativi e li inserisca dentro una particolare *tabella hash*, in cui a ciascun indirizzo è associato un albero binario di ricerca (ABR).

Siano date le seguenti definizioni:

- una foglia si dice *destra* (*sinistra*) se è figlio destro (sinistro) di un nodo padre;
- per ogni albero si definisce *ndx* (*nsx*) il numero delle sue foglie destre (sinistre);

Scrivere un programma che legga da tastiera una sequenza di N interi x e per ciascuno di essi

- individui l'indirizzo corrispondente utilizzando la seguente funzione hash:

$$h(x) = \{[(a \times x) + b] \% p\} \% S$$

dove $p=999149$, $a=1000$ e $b=2000$;

- lo inserisca nell'ABR associato all'indirizzo calcolato al punto precedente. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette \leq vanno inserite a sinistra);

Il programma dovrà poi stampare:

- l'indirizzo dell'albero con valore di *ndx* più alto. A parità di tale valore, scegliere l'indirizzo maggiore.
- l'indirizzo dell'albero con valore di *nsx* più alto. A parità di tale valore, scegliere l'indirizzo maggiore.

L'**input** è formattato nel seguente modo: la prima riga contiene gli interi N e S . Seguono N righe contenenti un'etichetta ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Algoritmi e Strutture Dati – Prova di Laboratorio

01/02/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione `main`. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream `cin` e `cout` rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: `input0.txt` `output0.txt` `input1.txt` `output1.txt` ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

[leggere il testo prestando particolare attenzione alle definizioni]
Si consideri un sistema di memorizzazione che legga una sequenza di N interi unici e non negativi e li inserisca dentro un albero binario di ricerca (ABR). Siano date le seguenti definizioni:

- una foglia si dice *concorde* se ha etichetta *pari* (*dispari*) ed è figlia di un padre con etichetta *pari* (*dispari*);
- una foglia si dice *discorde* se ha etichetta *pari* (*dispari*) ed è figlia di un padre con etichetta *dispari* (*pari*);
- Per ciascun nodo x , si indica con $nC(x)$ e $nD(x)$ il numero di foglie rispettivamente concordi e discordi del sottoalbero radicato in x . Per una foglia tali valori sono entrambi nulli.

Scrivere un programma che:

- legga da tastiera N etichette e le inserisca all'interno dell'ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti;
- stampi le etichette dei nodi tali per cui $nC(x) - nD(x) \geq 0$, ordinati per etichetta non decrescente. (complessità al più $\mathcal{O}(n)$)

L'**input** è formattato nel seguente modo: la prima riga contiene l'intero N . Seguono N righe contenenti un'etichetta ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

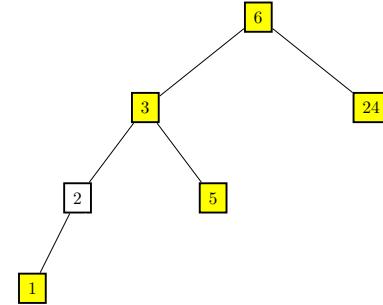
Esempio

Input

```
6
6
3
2
5
24
1
```

Output

```
1
3
5
6
24
```



Algoritmi e Strutture Dati – Prova di Laboratorio

19/02/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: *input0.txt* *output0.txt* *input1.txt* *output1.txt* ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file *output0.txt*. Per effettuare un controllo automatico sul primo file input *input0.txt* potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere. Dato un nodo x , si dicono suoi *antenati* tutti i nodi che si trovano sul cammino tra x e la radice dell'albero, radice inclusa. Si definisce inoltre $D(x)$ come la più lunga distanza tra x e un suo antenato con etichetta pari. Nel caso nessun antenato abbia etichetta pari, $D(x)=-1$.

Si scriva un programma che

- legga da tastiera N etichette e le inserisca all'interno dell'ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette \leq vanno inserite a sinistra);
- calcoli $D(x)$ per ogni nodo (complessità al più $\mathcal{O}(n)$);
- stampi al più le etichette dei primi K nodi con $D(x) > 0$ ordinate per valore di $D(x)$ decrescente. A parità di $D(x)$ si considerino le etichette in ordine non decrescente (complessità al più $\mathcal{O}(n \log(n))$).

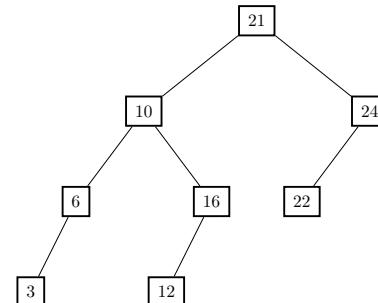
L'**input** è formattato nel seguente modo: la prima riga contiene gli interi N e K . Seguono N righe contenenti un'etichetta ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
8 3  
21  
10  
6  
16  
3  
24  
12  
22
```



Output

```
3  
12  
6
```

Algoritmi e Strutture Dati – Prova di Laboratorio

07/06/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: *input0.txt* *output0.txt* *input1.txt* *output1.txt* ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file *output0.txt*. Per effettuare un controllo automatico sul primo file input *input0.txt* potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

1

2

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere. Ogni nodo x dell'albero è associato ad un colore c . I colori sono identificati da un intero tra 0 e $C - 1$.

Si definisce **percorso di colore c** , un cammino che va da un nodo antenato verso un suo nodo discendente, attraversando tutti nodi con lo stesso colore c . Si definisce la **lunghezza** di un percorso come il numero di nodi che ne fanno parte.

Si scriva un programma che

- legga da tastiera N coppie $\{etichetta, colore\}$ e le inserisca all'interno dell'ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti (le etichette \leq vanno inserite a sinistra);
- per ogni c dei C colori possibili, calcoli $L(c)$ come la lunghezza del percorso più lungo relativo a detto colore (complessità al più $O(n)$)
- consideri i colori in ordine crescente di identificatore, e per ciascuno stampi la lunghezza del percorso più lungo $L(c)$ trovata al punto precedente. (complessità al più $O(C)$)

L'**input** è formattato nel seguente modo: la prima riga contiene gli interi N e C separati da uno spazio. Seguono N righe contenenti una coppia $\{etichetta, colore\}$ ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

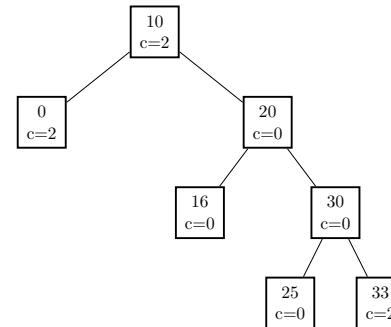
Esempio

Input

```
7 3  
10 2  
0 2  
20 0  
16 0  
30 0  
25 0  
33 2
```

Output

```
3  
0  
2
```



Algoritmi e Strutture Dati – Prova di Laboratorio

28/06/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione `main`. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuati tramite gli stream `cin` e `cout` rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: `input0.txt` `output0.txt` `input1.txt` `output1.txt` ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere. Ogni nodo dell'albero è associato ad una stringa.

Si definisce la seguente proprietà **P**: sia dato un nodo x ; sia d la distanza tra x e la radice dell'albero; sia infine f il numero di foglie appartenenti al sottoutilbero radicato in x . Si dice che il nodo x soddisfa la proprietà **P** se $d=f$.

NOTA: La distanza tra il nodo radice e se stesso è **0**.

Scrivere un programma che:

- legga da tastiera una sequenza di N coppie $[intero, stringa]$ e le inserisca in un ABR utilizzando il valore intero come etichetta. Le coppie devono essere inserite nello stesso ordine con cui vengono lette. (le etichette \leq vanno inserite a sinistra).
- identificare i nodi dell'ABR che soddisfano la proprietà P (complessità al più $\mathcal{O}(n)$).
- stampare le stringhe associate ai nodi che soddisfano la proprietà **P**, considerate in ordine lessicografico (complessità al più $\mathcal{O}(n\log n)$).

L'**input** è formato nel seguente modo: la prima riga contiene l'intero N . Seguono N righe contenenti una coppia $[intero, stringa]$ ciascuna, con gli elementi della coppia separati da uno spazio.

L'**output** contiene gli elementi della soluzione, uno per riga.

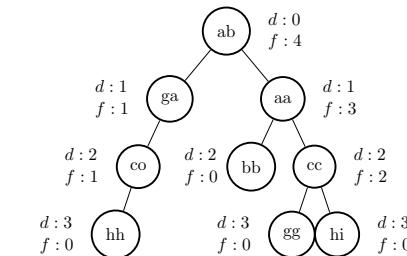
Esempio

Input

```
9
100 ab
120 aa
90 ga
115 bb
140 cc
130 gg
200 hi
10 co
5 hh
```

Output

```
cc
ga
```



Algoritmi e Strutture Dati – Prova di Laboratorio

19/07/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione `main`. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuati tramite gli stream `cin` e `cout` rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: `input0.txt` `output0.txt` `input1.txt` `output1.txt` ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema che memorizza le informazioni relative a dei lavoratori. Ciascun lavoratore è identificato da un **ID** univoco e intero, e da un **Cognome** di tipo stringa. Il sistema mantiene questi dati dentro una tabella hash con liste di trabocco (metodo di concatenazione). Scrivere un programma che

- legga da tastiera una sequenza di N coppie (**ID,Cognome**) ciascuna rappresentante un lavoratore;
 - salvi dentro una *tabella Hash* le informazioni relative ai lavoratori, utilizzando la seguente funzione hash:
- $$h(ID) = \{[(a \times ID) + b] \% p\} \% 2N$$
- dove $p=999149$, $a=1000$ e $b=2000$;
- identifichi i primi K indirizzi della *tabella Hash* in ordine decrescente di collisioni generate. A parità di numero di collisioni considerare l'indirizzo minore (complessità al più $\mathcal{O}(n\log n)$);
 - dati gli indirizzi ottenuti e ordinati al passo precedente, considerarli nello stesso ordine e stampare per ciascuno di essi l'**ID** del primo elemento secondo l'ordine lessicografico per **Cognome**. A parità di **Cognome**, scegliere l'elemento con **ID** più basso. In caso di lista vuota, stampare -1 (complessità al più $\mathcal{O}(n)$).

L'**input** è formattato nel seguente modo: le prime due righe contengono gli interi N e K . Seguono N righe contenenti una coppia (**intero,stringa**) ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
5 5
1111 Green
2222 Fontana
5555 Lupo
1456 Bernard
777 McCoy
```

Output

```
1456
777
2222
5555
-1
```

Algoritmi e Strutture Dati – Prova di Laboratorio

13/09/2018

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuati tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: **input0.txt** **output0.txt** **input1.txt** **output1.txt** ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file **output0.txt**. Per effettuare un controllo automatico sul primo file input **input0.txt** potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere. Un nodo si dice di *tipo* pari (dispari) se ha un'etichetta di valore pari (dispari).

Per ciascun nodo x che non sia una foglia, si definisce D come la massima distanza tra x e una sua foglia dello stesso tipo. Nel caso nel sottoalbero radicato in x non sia presente una foglia dello stesso tipo, $D = -1$.

Scrivere un programma che:

- legga da tastiera una sequenza di N interi e li inserisca in un ABR utilizzando il valore intero come etichetta. I valori devono essere inseriti nello stesso ordine con cui vengono letti. (le etichette \leq vanno inserite a sinistra).
- calcoli D per ogni nodo non foglia (complessità al più $\mathcal{O}(n)$).
- stampare in ordine decrescente e senza duplicati i valori di D (complessità al più $\mathcal{O}(n \log n)$).

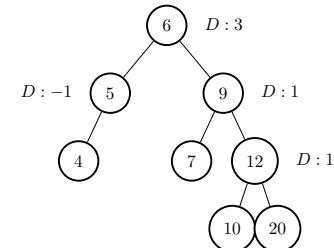
L'**input** è formato nel seguente modo: la prima riga contiene l'intero N . Seguono N righe contenenti un *intero* ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
8
6
5
9
4
12
7
10
20
```



Output

```
3
1
-1
```

Algoritmi e Strutture Dati – Prova di Laboratorio

10/01/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuati tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: **input0.txt** **output0.txt** **input1.txt** **output1.txt** ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file **output0.txt**. Per effettuare un controllo automatico sul primo file input **input0.txt** potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema di memorizzazione dei dati relativi agli utenti. Ogni utente è rappresentato da un *ID* univoco, intero e positivo, e da un'età. Il sistema legge i dati relativi a *N* utenti e li inserisce dentro una *tabella hash*, utilizzando l'*ID* come etichetta. La tabella hash è realizzata con il metodo di concatenazione.

Si definisce *I* come l'insieme dei primi *K* indirizzi della tabella hash che hanno generato più collisioni. A parità di numero di collisioni, si considerino gli indirizzi in ordine crescente. Si definisce *U* come l'insieme degli utenti memorizzati negli indirizzi appartenenti all'insieme *I*.

Scrivere un programma che:

- legga da tastiera una sequenza di *N* coppie di interi {*ID*,età} e le inserisca nella tabella hash all'indirizzo dato dalla seguente funzione hash:

$$h(x) = \{[(a \times x) + b] \% p\} \% (S)$$

dove $p=999149$, $a=1000$ e $b=2000$;
- Individui l'insieme *I*. (complessità al più $\mathcal{O}(n \log n)$).
- Stampa a video *ID* ed età dell'utente più anziano tra quelli appartenenti all'insieme *U*. A parità di età, si considerino gli utenti in ordine di *ID* crescente. (complessità al più $\mathcal{O}(n)$).

L'**input** è formattato nel seguente modo: la prima riga contiene gli interi *N*, *K* e *S* separati da uno spazio. Seguono *N* righe contenenti una coppia di interi ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input	Output
11 3 7	170
170 88	88
96 68	
577 79	
692 99	
873 75	
446 88	
732 79	
394 78	
845 58	
715 9	
455 11	

Algoritmi e Strutture Dati – Prova di Laboratorio

31/01/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione `main`. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuati tramite gli stream `cin` e `cout` rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: `input0.txt` `output0.txt` `input1.txt` `output1.txt` ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere. Un nodo si dice di *equilibrato* se

- non è una foglia;
 - ha entrambi i figli che sono foglie.
- Un nodo si dice invece di *non-equilibrato* se
- non è una foglia;
 - ha un solo figlio e quest'ultimo è una foglia.

Per ciascun nodo x si indica con eq e neq rispettivamente il numero di nodi *equilibrati* e *non-equilibrati* che si trovano nel sottoalbero radicato in x , x incluso. Infine, si definisce per ogni nodo x la funzione $f = neq - eq$. Scrivere un programma che:

- legga da tastiera una sequenza di N interi e li inserisca in un ABR utilizzando il valore intero come etichetta. I valori devono essere inseriti nello stesso ordine con cui vengono letti. (le etichette \leq vanno inserite a sinistra).
- calcoli f per ogni nodo x (complessità al più $\mathcal{O}(n)$).
- stampi l'etichetta e il valore di f del nodo con il valore di f più alto. A parità di f si considerino le etichette in ordine crescente (complessità al più $\mathcal{O}(n)$).

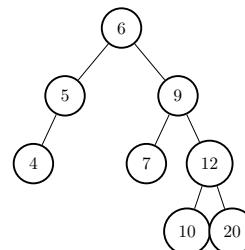
L'**input** è formato nel seguente modo: la prima riga contiene l'intero N . Seguono N righe contenenti un *intero* ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
8
6
5
4
9
12
7
10
20
```



Output

```
5
1
```

Algoritmi e Strutture Dati – Prova di Laboratorio

18/02/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione `main`. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream `cin` e `cout` rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: `input0.txt` `output0.txt` `input1.txt` `output1.txt` ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che `compilato` contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file `output0.txt`. Per effettuare un controllo automatico sul primo file input `input0.txt` potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere, in cui ciascun nodo è caratterizzato da una intero M , detta **massa**. Per ogni nodo x si indica con D la sua distanza dalla radice dell'albero. Per ogni nodo x può essere calcolato il suo **peso** P nella seguente maniera:

- Se x è foglia: $P = (M \times 2) - D$;
- In tutti gli altri casi: $P = M - D$;

Si definisce **carico** di uno nodo x , la somma dei pesi di tutti i nodi facenti parte del sottoalbero radicato in x , x escluso. Scrivere un programma che:

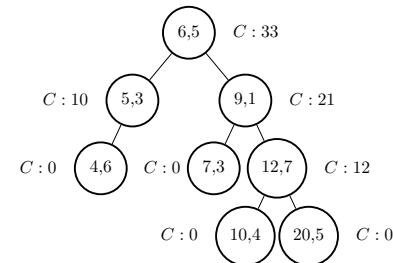
- legga da tastiera una sequenza di N coppie etichetta,massa e li inserisca in un ABR. I valori devono essere inseriti nello stesso ordine con cui vengono letti. (le etichette \leq vanno inserite a sinistra).
- stampi al più i primi K valori di P ordinati in maniera decrescente (complessità al più $\mathcal{O}(n\log n)$).

L'**input** è formato nel seguente modo: la prima riga contiene gli interi N e K . Seguono N righe contenenti una coppia $\{\text{etichetta}, \text{massa}\}$ ciascuna. L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
8 4  
6 5  
5 3  
4 6  
9 1  
12 7  
7 3  
10 4  
20 5
```



Output

```
33  
21  
12  
10
```

Algoritmi e Strutture Dati – Prova di Laboratorio

06/06/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità.

Nota Bene: La consegna del compito conclude la prova.

Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: *input0.txt* *output0.txt* *input1.txt* *output1.txt* ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file *output0.txt*. Per effettuare un controllo automatico sul primo file input *input0.txt* potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di alberi binari di ricerca (ABR) aventi etichette intere. Per ogni nodo x si indica con $h(x)$ la massima distanza tra x e le foglie del suo sottoalbero. Per la radice r si definisce H come $H = h(r)$, e per ogni foglia f vale $h(f) = 1$.

Scrivere un programma che stampi in maniera non decrescente al più le prime K etichette tali per cui $h(x) = H/2$ (complessità al più $\mathcal{O}(n)$).

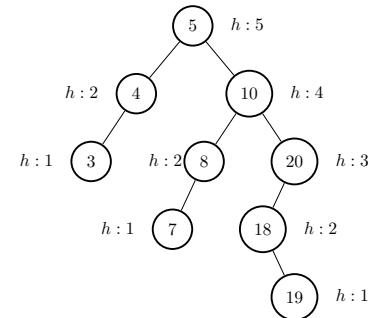
L'input è formato nel seguente modo: la prima riga contiene gli interi N e K . Seguono N righe contenenti un'etichetta ciascuna.

L'output contiene gli elementi della soluzione, uno per riga.

Esempio

Input

```
9 2  
5  
4  
10  
3  
8  
20  
7  
18  
19
```



Output

```
4  
8
```

Algoritmi e Strutture Dati – Prova di Laboratorio

27/06/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuate tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità.

Nota Bene: La consegna del compito conclude la prova.

Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: **input0.txt** **output0.txt** **input1.txt** **output1.txt** ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file **output0.txt**. Per effettuare un controllo automatico sul primo file input **input0.txt** potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema per la gestione di una rete informatica composta da N nodi. Ciascun nodo è caratterizzato da un *ID* intero e positivo, e da un tipo tra *Server*, *Client*, *Filtro* e *Router*, rappresentati dai caratteri S, C, F e R rispettivamente. I nodi della rete sono memorizzati tramite un albero binario di ricerca (ABR) usando l'*ID* come etichetta.

Un cammino che raggiunge un *Client* si dice *completo* se ha origine da un nodo *Server*, attraversa almeno un nodo *Filtro* e nessun altro *Server*.

Per ciascun *Server*, si definisce il numero g di *Clienti* serviti come il numero di cammini completi che hanno origine dal server stesso.

Scrivere un programma che consideri i *Server* in ordine di ID non decrescente, e stampi per ciascuno di essi il suo ID e il numero di *Client* da lui serviti. (complessità al più $\mathcal{O}(n)$).

L'**input** è formato nel seguente modo: la prima riga contiene l'intero N . Seguono N righe contenenti una coppia *ID, tipo* ciascuna, con i valori separati da uno spazio.

L'**output** contiene una coppia *ID, g* ciascuna, con i valori separati da uno spazio.

Esempio

Input

```
10
5 S
4 F
10 S
3 C
8 F
20 S
7 C
18 S
19 C
17 R
17 R
19 C
```

Output

```
5 1
10 1
18 0
```

Algoritmi e Strutture Dati – Prova di Laboratorio

18/07/2019

Istruzioni

Risolvere il seguente esercizio implementando un programma in un singolo file .cpp, completo di funzione *main*. Si presti particolare attenzione alla formattazione dell'input e dell'output, e alla complessità target indicata per ciascuna funzionalità. Nel caso la complessità target non sia specificata, si richiede che sia la migliore possibile. La lettura dell'input e la scrittura dell'output **DEVONO** essere effettuato tramite gli stream **cin** e **cout** rispettivamente. La correzione avverrà prima in maniera automatica inviando il file .cpp al server indicato in aula. Quest'ultimo esegue dei test confrontando l'output prodotto dalla vostra soluzione con l'output atteso. In caso la verifica abbia esito positivo sarà possibile consegnare il compito, il quale verrà valutato dai docenti in termini di complessità. Si ricorda che è possibile testare la correttezza del vostro programma in locale su un sottoinsieme dei input/output utilizzati nella seguente maniera. I file di input e output per i test sono nominati secondo lo schema: **input0.txt** **output0.txt** **input1.txt** **output1.txt** ... Per effettuare le vostre prove potete utilizzare il comando del terminale per la redirezione dell'input. Ad esempio

```
./compilato < input0.txt
```

effettua il test del vostro codice sui dati contenuti nel primo file di input, assumendo che **compilato** contenga la compilazione della vostra soluzione e che si trovi nella vostra home directory. Dovete aspettarvi che l'output coincida con quello contenuto nel file **output0.txt**. Per effettuare un controllo automatico sul primo file input **input0.txt** potete eseguire la sequenza di comandi

```
./compilato < input0.txt | diff - output0.txt
```

Questa esegue la vostra soluzione e controlla le differenze fra l'output prodotto e quello corretto.

Esercizio

Si consideri un sistema di memorizzazione dei dati relativi a dei veicoli. Ogni veicolo è rappresentato da un valore *targa* univoco, intero e positivo, e da un valore intero *categoria* compreso tra 0 e $C - 1$. Il sistema legge i dati relativi a N veicoli e li inserisce dentro una *tabella hash*, utilizzando la *targa* come etichetta. La tabella hash è realizzata con il metodo di concatenazione.

Per ogni indice i della tabella hash si definisce $M(i)$ come la categoria con più veicoli in i , e $V(i)$ come il numero di veicoli in i appartenenti a $M(i)$. A parità di numero di veicoli, si considerino le *categorie* in ordine crescente. Scrivere un programma che:

- legga da tastiera una sequenza di N coppie di interi $\{targa, categoria\}$ e le inserisca nella tabella hash all'indirizzo dato dalla seguente funzione hash:

$$h(x) = \{[(a \times x) + b] \% p\} \% (C)$$

dove $p=999149$, $a=1000$ e $b=2000$;

- Stampa a video i primi K indirizzi i della tabella hash in ordine di $V(i)$ decrescente. A parità di $V(i)$, si considerino gli indirizzi in ordine crescente.

L'**input** è formato nel seguente modo: la prima riga contiene gli interi N , K e C separati da uno spazio. Seguono N righe contenenti una coppia di interi ciascuna.

L'**output** contiene gli elementi della soluzione, uno per riga.

Esempio

Input	Output
11 6 7	0
170 1	3
96 2	1
577 0	2
692 0	4
873 6	6
446 6	
732 1	
394 1	
845 1	
715 1	
456 3	