Ultima revisione: mercoledì 23 giugno 2021

ESAME DI ALGORITMI E STRUTTURE DATI

Parte delle domande proposte al quiz e all'esame orale tenuto il 9 giugno 2021 (primo appello estivo)

Documento creato da Alessio Meini, studente al primo anno di Ingegneria Informatica AA. 2020/2021.

Si ringrazia Federico Nardi per la collaborazione e la pazienza!

NON mi assumo alcuna responsabilità riguardo la completezza e/o correttezza delle informazioni qui riportate.

Mi auguro che questo documento possa essere di aiuto a chi si preparerà a tenere l'esame di Algoritmi e Strutture Dati.

In bocca al lupo!

LA RISPOSTA ALLE DOMANDE DEI QUIZ È EVIDENZIATA

QUIZ

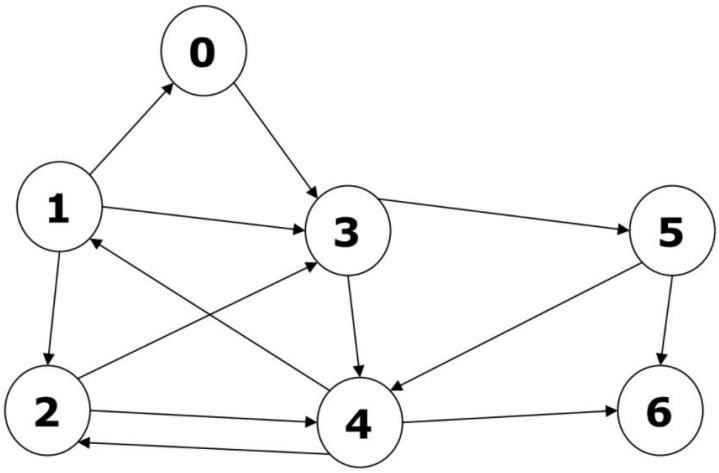
1. Date le seguenti stringhe (xyzzy – xxyzx) quanto è lunga la PLSC?

   ✓ 3(xyz)

2. Mostrare l'output al secondo step del radix sort (a, b, c, d, e) sulla lista [ace ceb bec abc eba bba]

   ✓ eba bba abc ace ceb bec

3. Visita in profondità di un grafico salvato in lista di adiacenza



   ✓ 0,3,4,1,2,6,5

4. Qual è la complessità dell'algoritmo di Dijkstra?

   ✓ O(nlogn+mlogn)

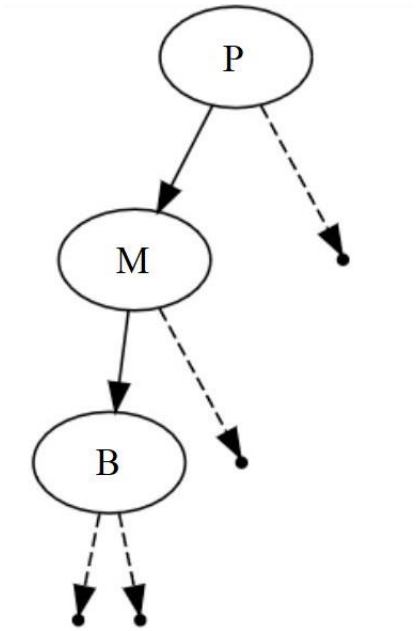
5. A quale categoria di problemi (P,NP, NP-Completi) appartiene SAT-I?

   ✓ SAT nella logica del I ordine non è risolvibile con un algoritmo

6. Un problema si definisce appartenente a NP:

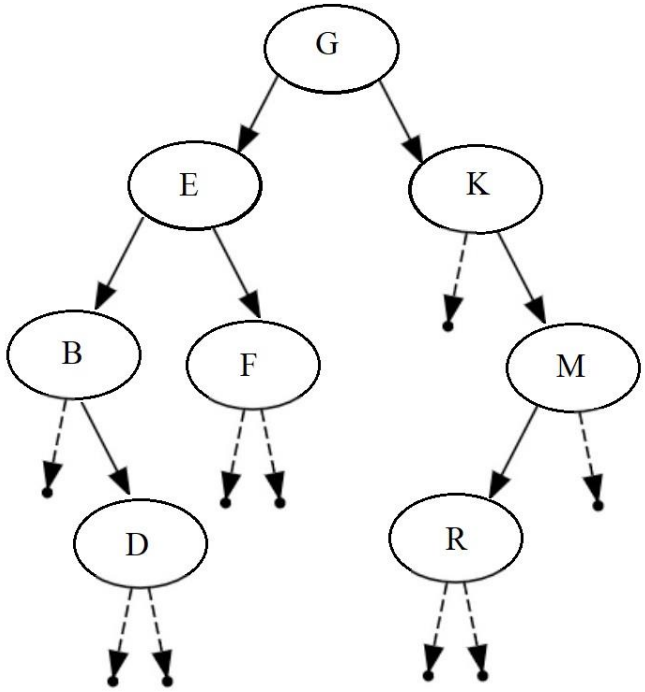
   ✓ Se esiste un algoritmo che verifica una sua soluzione in tempo polinomiale

7. Quale definizione è corretta per il seguente albero? (nodi in ordine alfabetico con A<Z, per chiarire la differenza tra figlio sinistro e destro, sono riportati anche i figli vuoti dei nodi)



   ✓ Si definisce albero binario di ricerca DEGENERE

8. Fare la visita anticipata del seguente albero (pre-order) (nodi in ordine alfabetico con A<Z, per chiarire la differenza tra figlio sinistro e destro, sono riportati anche i figli vuoti dei nodi)



✓ G E B D F K M R

ALTRE DOMANDE:

Domande sulla gerarchia delle classi

Domande sulle funzioni template

ESEMPIO:

Data una funzione template<class tipo> con un valore statico incrementato ad ogni chiamata, alla modifica del “tipo” in input alla funzione veniva creata una nuova istanza che, alla chiamata della funzione, stampava un valore della variabile statica differente a seconda del “tipo” (esempio con static a=0; a++; : chiamata con<int> a=1, chiamata <double> a= 1, chiamata <int> a= 2)

PARTE PRATICA

ALFEO:

-**AlberiBinariDiRicerca**

INPUT DI ESEMPIO: 8 5 3 8 2 7 12 11 14

Un output di esempio verrà mostrato basandosi su l'input sopra riportato.

1. Somma dei nodi concordi, dove per concordi si definisce il nodo il cui valore e pari (o dispari) come anche il padre, la radice è concorde.

Output atteso: **34**

Una soluzione possibile:

```
int sommaConcordi(Node* tree, int dad = -1){
    if(!tree)return 0;
    int value=0;
    if((dad == -1)||((tree->value%2==0) == (dad%2==0))) value = tree->value;
    return sommaConcordi(tree->left, tree->value) + sommaConcordi(tree->right, tree-
>value) + value;
}
```

2. Sommatoria dei nodi con altezza dispari meno la sommatoria dei nodi con altezza pari.

Output atteso: **10**

Una soluzione possibile:

```
int diffPariDispari(Node* tree, int level = 0){
    if(!tree)return 0;
    return diffPariDispari(tree->left, level+1) + diffPariDispari(tree->right,
level+1) + (level%2!=0?tree->value:(-tree->value));
}
```

3. Sommatoria dei soli nodi pari dove si definisce pari il nodo la cui altezza è pari, solo nodi, niente foglie (la radice per convezione non si considera)

Output atteso: **12**

Una soluzione possibile:

```
int sommaNodiPari(Node* tree, int level = 0){
    if(!tree)return 0;
    if(level != 0 && level%2==0) {
        if (tree->left == nullptr && tree->right == nullptr)
            return tree->value + sommaNodiPari(tree->left, level + 1) +
sommaNodiPari(tree->right, level + 1);
```

```
    }
    return sommaNodiPari(tree->left, level + 1) + sommaNodiPari(tree->right, level +
1);
}
```

4. Prodotto dell'altezza del sottoalbero destro per il l'altezza del sottoalbero sinistro

Output atteso: **6**

Una soluzione possibile:

```
int altezza(Node* tree, int level = 0){
    if(!tree)return level-1;
    return max(altezza(tree->right, level+1), altezza(tree->left,level+1));
}

int prodotto(Node *tree){
    return altezza(tree->left, 1)*altezza(tree->right, 1);
}
```

(faccio la chiamata SOLO alla funzione "prodotto(albero.getRoot())" nel main)

5. Sommatoria dei nodi concordi dove concorde è il nodo con etichetta pari (o dispari) ed altezza pari (o dispari)

Output atteso: **28**

Una soluzione possibile:

```
int sommatoriaConcordi(Node* tree, int level = 0){
    if(!tree)return 0;
    return (((tree->value%2==0) == (level%2==0))?tree->value:0)
    +sommatoriaCocordi(tree->left, level+1)+sommatoriaCocordi(tree->right, level+1);
}
```

6. Implementare la ricerca in un albero binario

7. Sommatoria tra il minimo ed il massimo di un albero

8. Sommatori di nodi completi di altezza dispari, per completo si definisce il nodo che ha due figli (radice altezza 1)

-Heap

1. Implementare la stampa dello heap (dove i nodi sono disposti graficamente in modo da rappresentare lo heap come un albero), come da esempio:

10

9 6

8 4 3 2

Dove gli elementi appartenenti al sottoalbero di una etichetta K sono tra le parentesi quadre

10 (K=6)

9 6

8 4 [3] [2]

2. Stampa uno heap dove il figlio destro di un determinato nodo K è tra parentesi quadre

13 (k=9)

9 8

4 [3] 1

Una soluzione possibile:

```
void printRight(int elem, int i = 0){
    if(i >= lista_.size())return;
    if(isFirstChild(i+1)) std::cout<<std::endl;
    if(2*i+1 < lista_.size()&& lista_[2*i+2] == elem)
        std::cout<<'['<<lista_[i]<<']'<<'\t';
    else
        std::cout<<' '<<lista_[i]<<' '<<'\t';
    printRight(elem, i+1);
}
```

DUCANGE:

- Implementare lo HeapSort
  Quale caratteristica sfrutta?
  (Si tratta della caratteristica dello Heap di avere la radice con valore maggiore rispetto al resto dei nodi dell'albero)

- Implementare il mergeSort

- Creare una struttura dati albero e creare la funzione di inserimento

- Implementare l'algoritmo di Huffman

- Implementare il bubbleSort
- Implementare il selectionSort
- Implementare il quickSort
- Implementare un miniHeap