COT6405 Analysis of Algorithms
Spring 16

MACIEJ MEDYK
Z-15197421

Assignment 2

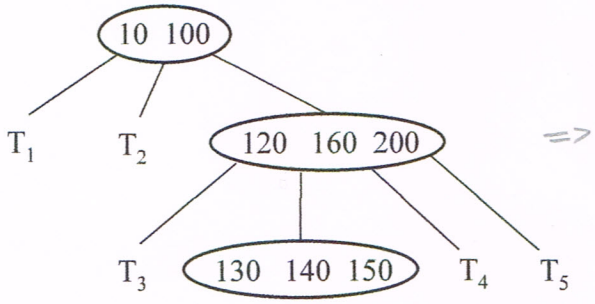
Posted on Feb 9, due on Feb 23

Maximum total of 35 points.

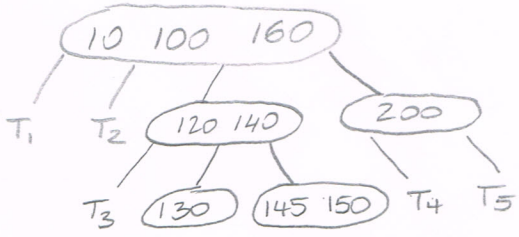
(*20 points*)

1. B-TREES
   For both parts the minimum degree t = 2.

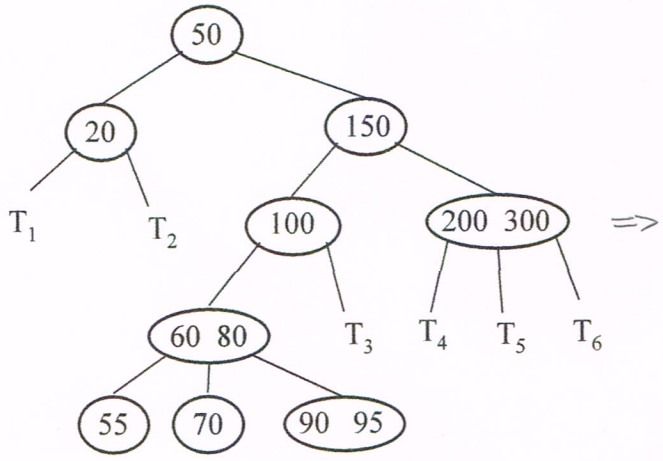
a). Consider the B-Tree below. Illustrate the operation:
B-Tree-Insert (T, 145)
Show your work, the tree obtained after applying each step.



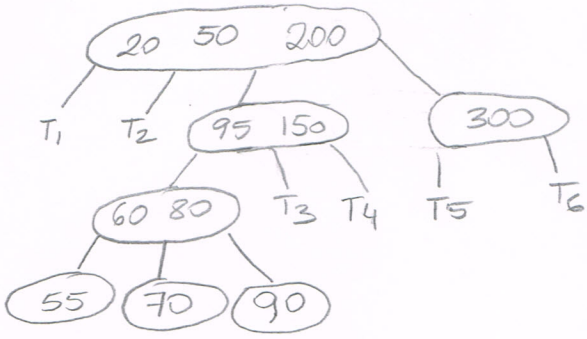
=>



b). Consider the B-Tree below. Illustrate the operation:
B-Tree-Delete (T, 100)
Show your work: the rule used and the tree obtained after each step.



=>

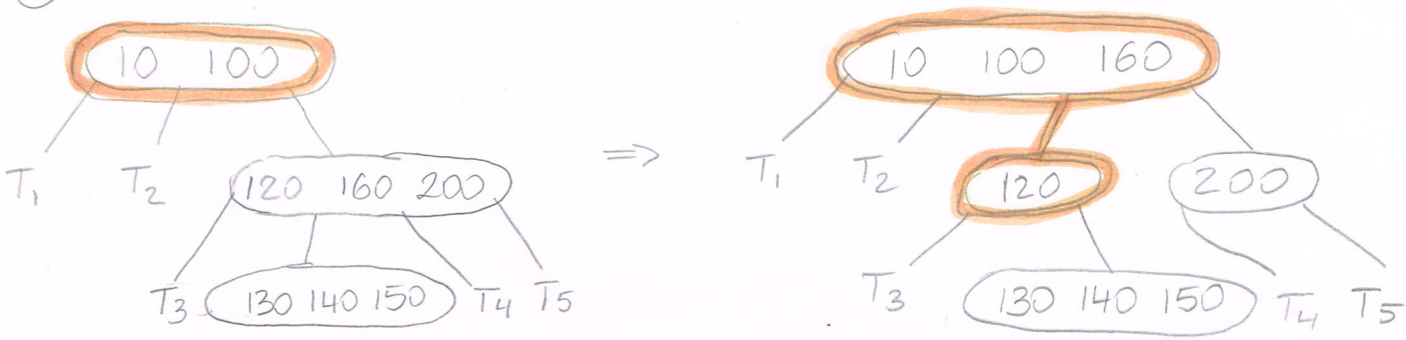


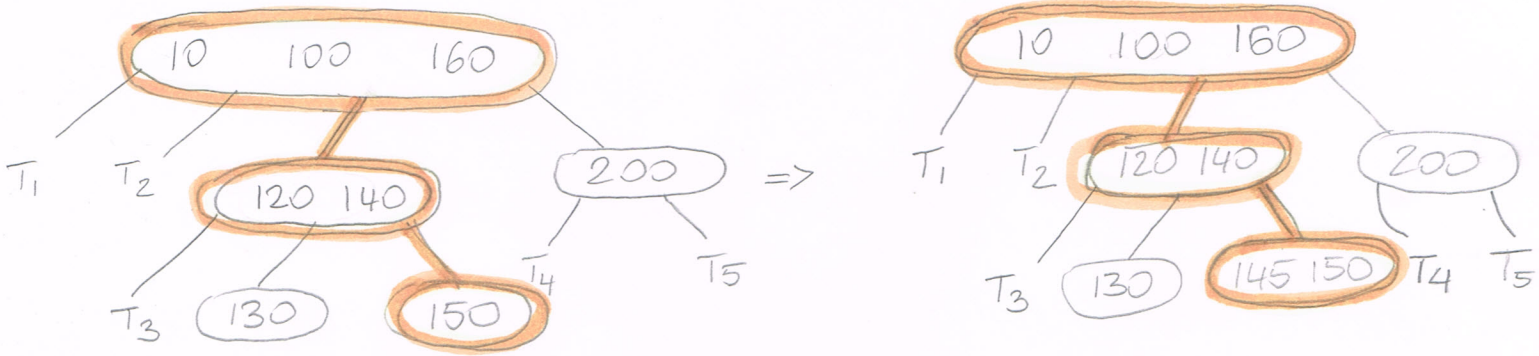
(continued next page)

(*15 points*)
2. BACKTRACKING
Solve the m-Independent Set Problem (defined below) using a backtracking algorithm. Write the pseudo-code and analyze its worst-case running time. **You have to use the general framework described in class. All other attempts will not be graded.**

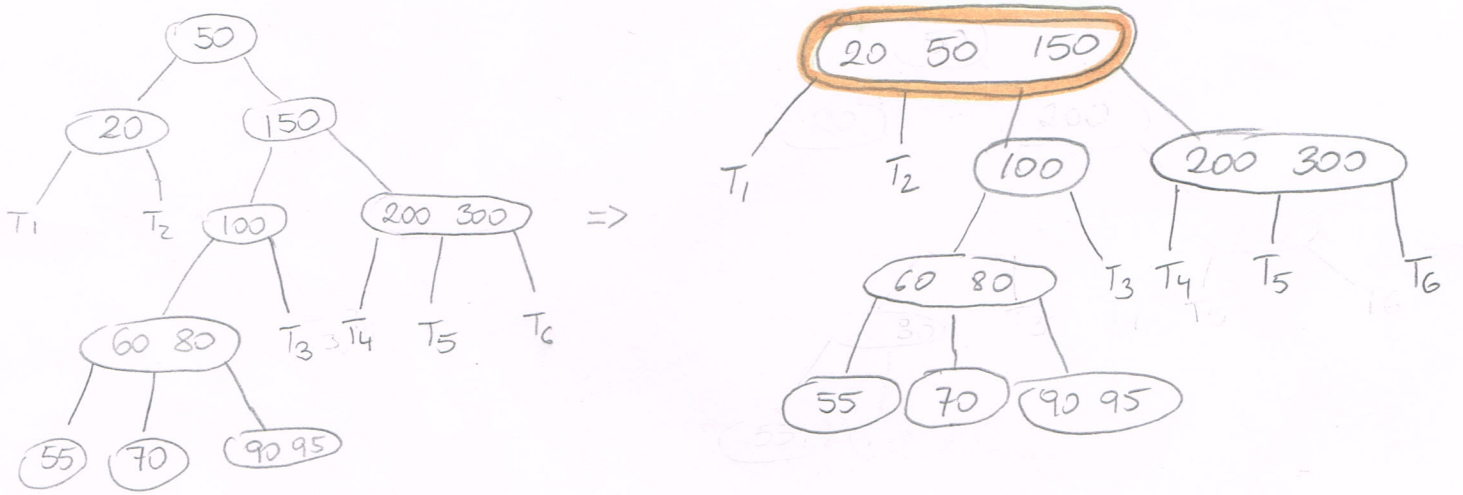
m-Independent Set Problem: Given a graph G with n nodes, where $n > 2$, and a value m such that $1 < m < n$, find whether G has an independent set of size m. Note that G, m, and n are given as input in this problem.

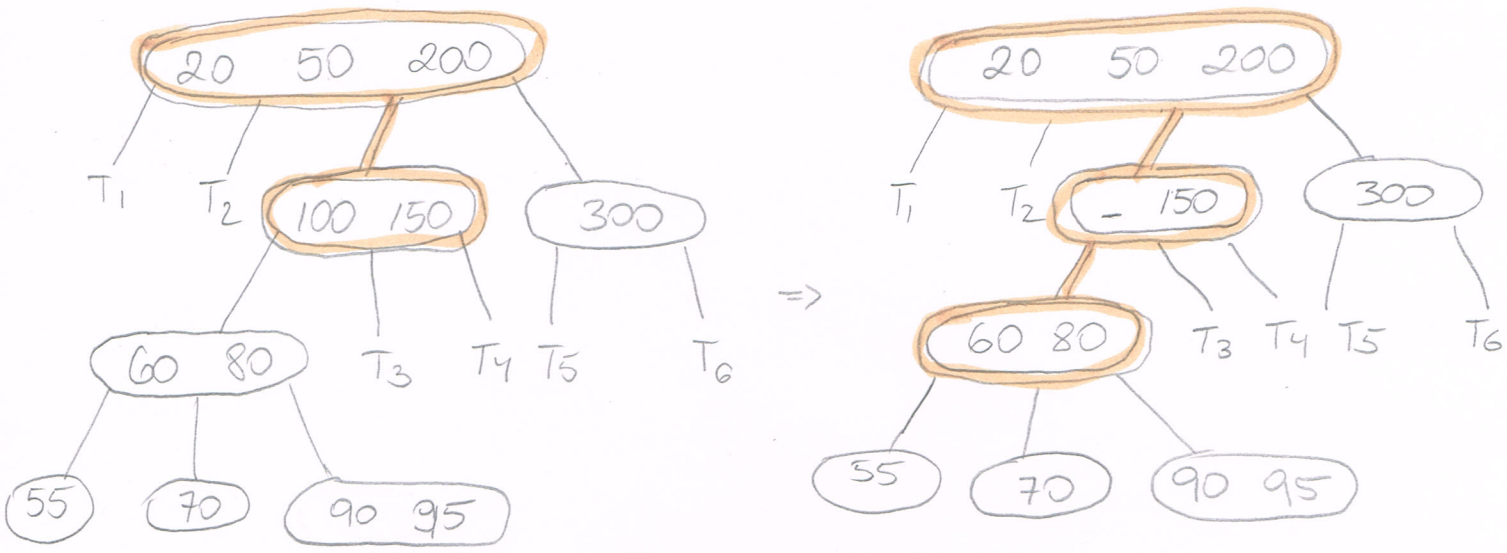
1 (A) INSERT 145

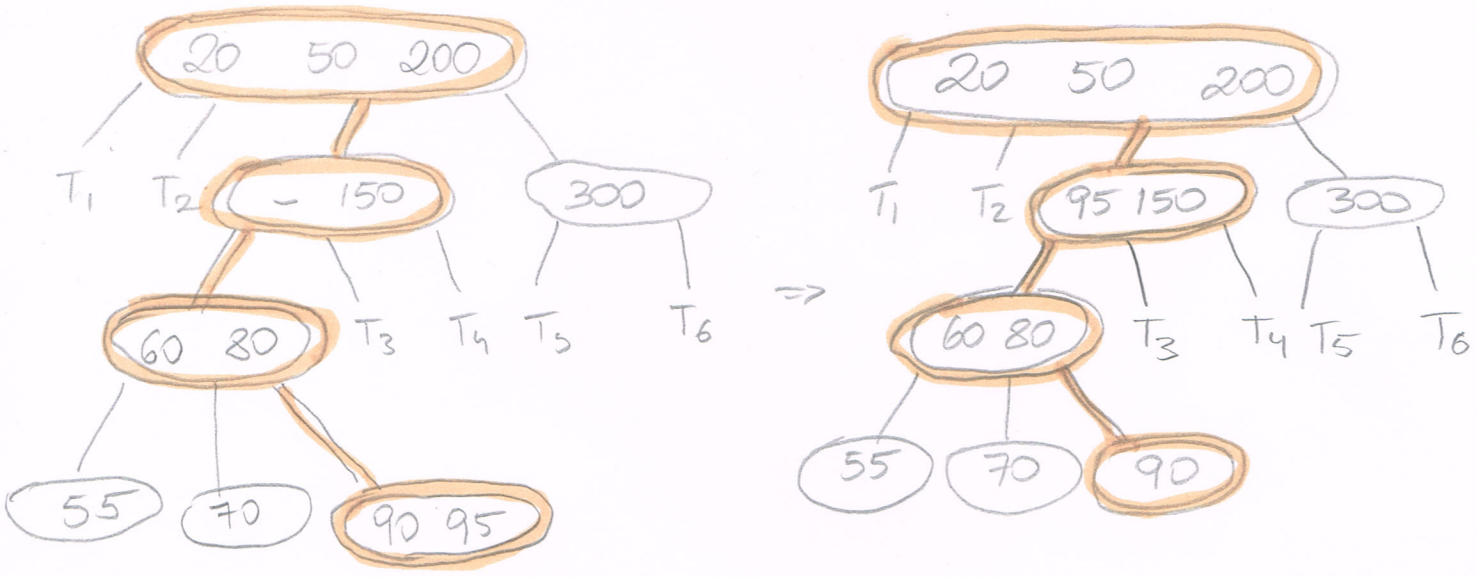




(B) DELETE 100




20 50 200
T1 T2 100 150 300
60 80 T3 T4 T5 T6
55 70 90 95
=>
20 50 200
T1 T2 - 150 300
60 80 T3 T4 T5 T6
55 70 90 95


20 50 200
T1 T2 - 150 300
60 80 T3 T4 T5 T6
55 70 90 95
=>
20 50 200
T1 T2 95 150 300
60 80 T3 T4 T5 T6
55 70 90

(2)

```
backtrack (G, m, n)
{
  for i from 0 to n
  {  for k from 0 to n
     { RESULTARRAY[k] = 0
     RESULTARRAY[i] = 1
     for j from 0 to n
     {  if G[i][j] = 0
        { ARRAYONE[j] = 1
        else
        { ARRAYONE[j] = 0
     }
     if ( rbacktrack ( G, RESULTARRAY, ARRAYONE, index, m, n) == TRUE)
     { RETURN TRUE
  }
  RETURN FALSE
}
```

```
rbacktrack ( G, RESULTARRAY, ARRAYONE, index, m, n)
{ nextindex = index
  for j in range index to n
  {  if ARRAYONE[j] == 1
     {nextindex = j
     break
  if nextindex == n
  { return FALSE
  for k in range 0 to n
  {  if G[nextindex][k] == 0
     { ARRAYTWO = 1
     else
     { ARRAYTWO = 0
  }
  IF ( BOUND (ARRAYONE, ARRAYTWO, RESULTARRAY, nextindex, n) == m)
  { RETURN TRUE
  ELSE
  { RETURN RBACKTRACK (G, RESULTARRAY, ARRAYONE, NEXTARRAY+1, m, n)
}
```

(2) CONTINUES

```
BOUND ( ARRAYONE, ARRAYTWO, RESULTARRAY, INDEX, n)
{
  RESULTARRAY[index] = 1
  for i in range from 0 to n
  {
    IF ( ARRAYONE[i] == 1 AND ARRAYTWO[i] == 1)
    { ARRAYONE = 1
    else
    { ARRAYONE = 0
  }
  COUNT = 0
  for k in range from 0 to length(RESULTARRAY)
  {
    if (RESULTARRAY[k] == 1)
    { COUNT = COUNT + 1
  }
  RETURN COUNT
}
```

BACKTRACK -> RT $\leq O(n)$

RBACKTRACK

- $k = 1$ -> 0 times
- $k = 2$ -> n times
- $k = 3$ -> $\leq n(n-1)$ times
- $k = 4$ -> $\leq n(n-1)(n-2)$ times
- $k = n$ -> $\leq n(n-1)(n-2) \ldots 2 = n!$ times

$$RT = O(n) \cdot O\left(n!\left(\frac{1}{n!} + \frac{1}{(n-1)!} + \frac{1}{(n-3)!} + \ldots + \frac{1}{1!}\right)\right)$$

$$O(n) \cdot O(n!(e-1))$$

$$RT = O(n \times n!)$$

```
def backtrack(GRAPH, m, n):
    print("M parameter : " + str(m))
    for i in range(0, n):
        IARR = [0] * n
        IARR[i] = 1
        ARR1 = [0] * n
        for j in range(0, n):
            if(GRAPH[i][j] == 0):
                ARR1[j] = 1
        if(rbacktrack(GRAPH, IARR, ARR1, i+1, m, n) == True):
            print("Result : true - " + str(IARR))
            for k in range(0, n):
                if(IARR[k] == 1):
                    print("> Node : " + str(k + 1))
            return
    print("Result : false")

def rbacktrack(GRAPH, IARR, ARR1, index, m, n):
    nextindex = n
    for j in range(index, n):
            if(ARR1[j] == 1):
                nextindex = j
                break
    if(nextindex == n):
        return False
    ARR2 = [0] * n
    for j in range(0, n):
            if(GRAPH[nextindex][j] == 0):
                ARR2[j] = 1
    if(bound(ARR1, ARR2, IARR, nextindex, n) == m):
        return True
    else:
        return rbacktrack(GRAPH, IARR, ARR1, nextindex + 1, m, n)


def bound(ARR1, ARR2, IARR, index, n):
    IARR[index] = 1
    for j in range(0, n):
        if(ARR1[j] == 1 and ARR2[j] == 1):
            ARR1[j] = 1
        else:
            ARR1[j] = 0
    x = 0
    for i in range(0, n):
            if(IARR[i] == 1):
                x = x + 1
    return x
```

```
G = [[0, 1, 0, 1, 0, 0],
     [1, 0, 1, 0, 1, 0],
     [0, 1, 0, 0, 1, 0],
     [1, 0, 0, 0, 1, 1],
     [0, 1, 1, 1, 0, 1],
     [0, 0, 0, 1, 1, 0]]

m = 3
n = len(G)
backtrack(G, m, n)
```

```
M parameter : 3
Result : true - [1, 0, 1, 0, 0, 1]
> Node : 1
> Node : 3
> Node : 6
```