BCSE 204L- Design and Analysis of Algorithms

CAT-II F2-Slot Answer key

1. Illustrate the solution to the following 0/1 Knapsack problem by constructing the state space tree using LIFO and FIFO Branch and Bound techniques, and answer the questions given.

a) Determine the maximum profit obtained for the given knapsack problem? Do both the methods provide the optimal solution?

b) Which method explores fewer nodes for this problem?

Capacity=7

Item	**Profit**	**Weight**	p/w
1	20	2	10
2	14	4	3.5
3	18	3	6

Solution:

FIFO B&B:

Global UB=-38
Node 1: UB=-38, LB=-45
Enqueue Node 1
1←

1) Dequeue Node 1 and explore (generate child nodes 2 & 3)

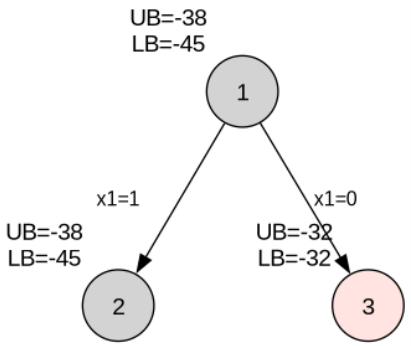
Node 2: UB=-38, LB=-45
Enqueue Node 2
2←

Node 3: UB=-32,LB=-32
As LB > Global UB, discard it

2) Dequeue Node 2 and explore (generate child nodes 4 & 5)

Node 4: UB = -38, LB= -45
Enqueue Node 4
4←

Node 5: UB=-34 & LB=-34
As LB > Global UB, discard it



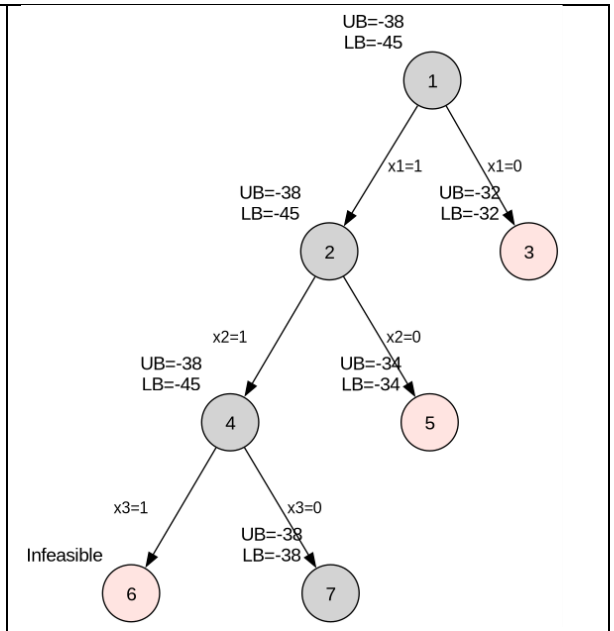
3) Dequeue Node 4 and explore (generate child nodes 6 & 7)

Node 6: Infeasible; so, discard

Node 7: UB=-38 & LB=-38
Enqueue Node 7.
7←

4) Dequeue Node 7; no more items to explore and Node 7 is the only live node whose UB=Global UB. Traverse the path from node 7 to the root to obtain the solution.

Solution ={x1,x2} with profit = 38



LIFO B&B:

Global UB=-38
Node 1: UB=-38, LB=-45
Push Node 1 into stack
→1

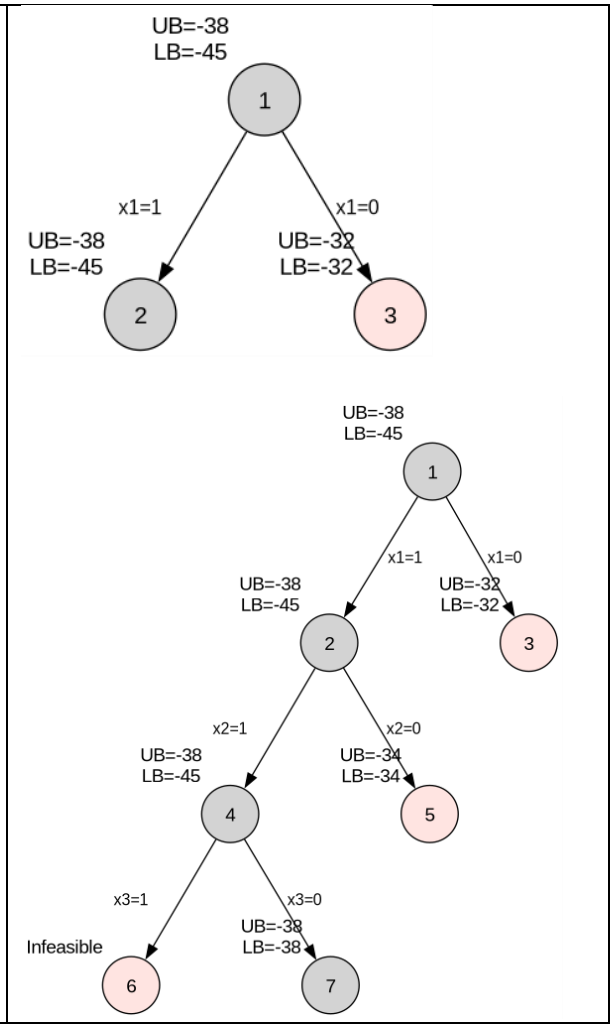
1) Pop Node 1 and explore (generate child nodes 2 & 3)
   Node 2: UB=-38, LB=-45
   Push Node 2
   →2

   Node 3: UB=-32,LB=-32
   As LB > Global UB, discard it

2) Pop Node 2 and explore (generate child nodes 4 & 5)
   Node 4: UB = -38, LB= -45
   Push Node 4
   →4

   Node 5: UB=-34 & LB=-34
   As LB > Global UB, discard it

3) Pop Node 4 and explore (generate child nodes 6 & 7)
   Node 6: Infeasible; so, discard



Node 7: UB=-38 & LB=-38 Push Node 7. →7 4) Pop Node 7; no more items to explore and Node 7 is the only live node whose UB=Global UB. Traverse the path from node 7 to the root to obtain the solution. Solution ={x1,x2} with profit = 38	

a) Maximum profit =38. Yes, both FIFO & LIFO BB provide the optimal solution

b) For this example, both explore the same number of nodes. In this example, at the very first step (the Root Node), the initial Greedy Upper Bound is calculated as **-38**, which is the exact optimal mathematical solution for this example. Because we started with the absolute best possible Global UB, the pruning rule was applicable for many nodes and hence both FIFO & LIFO BB prune the same set of nodes and so equally efficient.

2. a) Given below are the search text, pattern and the prefix function (π) calculated using the KMP algorithm for pattern matching.

Search text (s) = "12121213112123121231213" ; pattern (p) = "121231213"

Index	0	1	2	3	4	5	6	7	8
Prefix Function or LPS Table (π)	0	0	1	2	0	1	2	3	0

While applying the KMP algorithm to compare the search text and the pattern, for each of the first three mismatches, specify the next character from the search string and the corresponding character from the pattern that will be compared [Note: Continue the trace only up to three mismatches; Do not have to perform the checking until the pattern is found.]

b) The Rabin–Karp algorithm is efficient for multiple patterns searching. Illustrate this by generating multiple patterns from Document A formed with two consecutive words (such as words at indexes (0,1), (1,2), (2,3), …). Using these patterns, explain how the Rabin–Karp algorithm can be applied to detect possible plagiarism between the following two documents.

Doc A = "Today Artificial Intelligence is widely used"
Doc B = "Artificial Intelligence is widely used today".

Solution:

a) KMP

If *i and j* are used to index search text and pattern respectively,

i) First mismatch at i=4 &j=2

12121213112123121231213 i=4

12123 1213 j=4

So, j changes to LPS[3]=2

Therefore, characters at index i=4 & j=2 are compared

Which means characters 1 and 1 are compared; as they are equal, i and j are incremented to 5 &3

Characters 2 &2 are also equal and so, I and j are incremented to 6 & 4

1212121311212312123121 3 i=6

12123121 3 j=4

Here, the second mismatch occurs

So, j=LPS[3]=2

Next, characters at index i=6 & j=2 are compared

The characters 1 and 1 are same; So, i and j are incremented to 7 & 3 respectively

But, the third mismatch occurs at i=7 & j=3

1212121311212312123121 3 i=7

12123121 3 j=3

So, j=LPS[2]= 1

Next, characters at index i=7 & j=1 are compared; But, there is a mismatch here too.

1212121311212312123121 3

12123121 3

Stop with 3 mismatches, as per the question.

So, after the first mismatch, characters at i=4 & j=2 are compared

After the second mismatch, characters at i=6 & j=2 are compared

After the third mismatch, characters at i=7 & j=1 are compared

b) Rabin Karp

Doc A = “Today Artificial Intelligence is widely used”

Doc B = “Artificial Intelligence is widely used today”.

Multiple two-word patterns that can be generated from Doc A are:

“Today Artificial”

“Artificial Intelligence”

“Intelligence is”

“is widely”

“widely used”

There are a total of 5 patterns.

Document B is taken as the search text and Rabin-Karp algorithm can be run in parallel to match each of these 5 patterns with document B.

The hash values and subsequently the corresponding characters will match for 4 patterns namely “Artificial Intelligence”, “Intelligence is”, “is widely” and “widely used”, with document B

4 two-word patterns out of 5 match with document B.

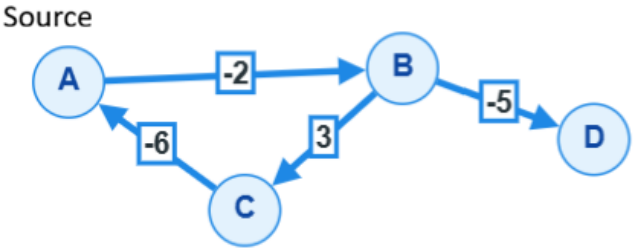
So, plagiarism =(4/5)*100 =80%

Document A is 80% similar to Document B

3. The theorem for the correctness of the Bellman–Ford algorithm follows from proving two properties: (i) it computes correct shortest-path distances when no negative-weight

cycle is reachable from the source, and (ii) it returns FALSE when a negative-weight cycle exists.

To prove the second property, we proceed by contradiction. Consider the graph below with node A as the source, that contains a negative-weight cycle that is reachable from source.



" Let this cycle be c = <v_0, v_1, …, v_k> where $v_0 = v_k$. Then,

$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$ ……*(1)*

Assume for contradiction, that Bellman–Ford returns TRUE.

Thus, $d(v_i) \leq d(v_{i-1}) + w(v_{i-1}, v_i)$ for i=1,2,…,k. Summing these inequalities around a cycle c gives:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$

$$\leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Since $v_0 = v_k$, each vertex appears exactly once in both summations of distances, and therefore: $\Sigma\, d(v_i) = \Sigma\, d(v_{i-1})$ which implies: $0 \leq \Sigma\, w(v_{i-1}, v_i)$

If the total weight of the cycle is non-negative, this contradicts the inequality in eqn(1). So, we conclude that the algorithm should return FALSE if the graph contains negative cycle."

Apply the proof steps provided above to the given graph and hence justify why the algorithm must return FALSE.

Solution:

Assume Bellman-Ford returns TRUE.

$d(v_i) \leq d(v_{i-1}) + w(v_{i-1}, v_i)$

Therefore, apply the aforementioned equation for all the nodes in the cycle

$d(B) \leq d(A) - 2$

$d(C) \leq d(B) + 3$

$d(A) \leq d(C) - 6$

Sum up the 3 equations.

$d(A)+d(B)+d(C) \leq [d(A)-2]+[d(B)+3]+[d(C)-6]$

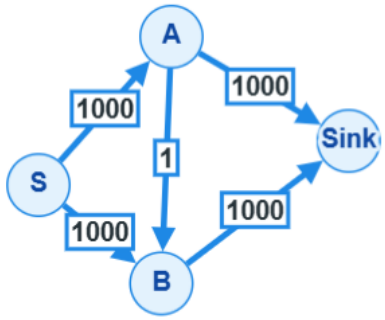
$d(A)+d(B)+d(C) \leq d(A)+d(B)+d(C)-2+3-6$

Taking all the terms to the right yields

$0 \leq -2+3-6$

$0 \leq -5$ this contradicts the inequality in (1). Thus, the algorithm should return false if there is a negative cycle.

4. a) Assume that, for the graph shown below, the Ford–Fulkerson algorithm uses the following DFS approach to compute the augmenting path.




```
DFS (G,u,sink)
{
visited[u] ← true
if u = sink then
  return true
for each vertex v ∈ V[G] {
  if visited[v] = false and
   (capacity[u][v] >0 || flow[u][v] > 0) then
    predecessor[v] ← u
    if DFS(G, v, sink) = true then
        return true
  }
return false
}
```

Find out the different augmenting paths and the maxflow.

b) Use the Edmonds–Karp algorithm for the same graph and compare the number of augmentations and the overall performance with that of the DFS-based method. State whether the computational efficiencies of these 2 methods are equal.

Solution:

a) Ford Fulkerson

Initial Residual matrix

	S	A	B	Sink
S	0	1000	1000	0
A	0	0	1	1000
B	0	0	0	1000
Sink	0	0	0	0

i) iteration-1

DFS(Source)

|

DFS(A)

|

DFS(B)

|

DFS(Sink)

So, **Augmenting path= S-> A->B->Sink** and flow=1 So, maxflow=1

Updated residual matrix

	S	A	B	Sink
S	0	999	1000	0
A	1	0	0	1000
B	0	1	0	999
Sink	0	0	1	0

ii) iteration -2

Adjacent nodes are found as per the updated residual matrix

DFS(Source)

|

DFS(A)

|

DFS(Sink)

So, **Augmenting path= S-> A->Sink** and flow=999 So, maxflow=1+999=1000

Updated residual matrix

	S	A	B	Sink
S	0	0	1000	0
A	1000	0	0	1
B	0	1	0	999
Sink	0	999	1	0

iii) iteration-3

Adjacent nodes are found as per the updated residual matrix

DFS(Source)

|

DFS(B)

|

DFS(A)

|

DFS(Sink)

So, **Augmenting path= S->B-> A->Sink** and flow=1 So, maxflow=1000+1=1001

Updated residual matrix

	S	A	B	Sink
S	0	0	999	0
A	1000	0	1	0
B	1	0	0	999
Sink	0	1000	1	0

iv) iteration-4

Adjacent nodes are found as per the updated residual matrix

DFS(Source)

|

DFS(B)

|

DFS(Sink)

So, Augmenting path= **S->B->Sink** and flow=999 So, maxflow=1001+999=2000
Updated residual matrix

	S	A	B	Sink
S	0	0	0	0
A	1000	0	1	0
B	1000	0	0	0
Sink	0	1000	1000	0

v) No adjacent nodes for S; Therefore, algorithm terminates

Augmenting paths:
Source->A->B->Sink; Flow=1
Source->A->Sink; Flow=999
Source->B->A->Sink; Flow=1
Source->B->Sink; Flow=999
Maxflow: 2000

b)Edmond Karp

Initially Residual matrix

	S	A	B	Sink
S	0	1000	1000	0
A	0	0	1	1000
B	0	0	0	1000
Sink	0	0	0	0

i) iteration-1

	S	**A**	**B**	**Sink**
Predecessor	-1	S	S	A
Visited	1	1	1	1

Augmenting path= S-> A->Sink and flow=1000 So, maxflow=1000
Updated residual matrix

	S	A	B	Sink
S	0	0	1000	0
A	1000	0	1	0
B	0	0	0	1000
Sink	0	1000	0	0

ii) iteration -2

Adjacent nodes are found as per the updated residual matrix

	S	**A**	**B**	**Sink**
Predecessor	-1		S	B
Visited	1	0	1	1

Augmenting path= S-> B->Sink and flow=1000 So, maxflow=1000+1000=2000

Updated residual matrix

	S	A	B	Sink
S	0	0	0	0
A	1000	0	1	0
B	1000	0	0	0
Sink	0	1000	1000	0

iii) iteration-3

there is no adjacent node to Source… This is the end of the algorithm.

Augmenting paths:

S->A->Sink

S->B->Sink

Maxflow=2000

Ford-Fulkerson: 4 augmenting paths

Edmond Karp: 2 augmenting paths

So, Edmond Karp is computationally efficient, and this is because BFS approach always tries to find the path with minimum number of edges first and this will avoid multiple small augmenting paths.

5. Apply the ANY-SEGMENTS-INTERSECT algorithm that uses the sweep-line status structure T for the following line segments and answer the questions below.

   S1: (1,0) → (11,0)

   S2: (1,4) → (7,3)

   S3: (2,2) → (5,3)

   S4: (8,3) → (12,2)

   S5: (10,1) →(13,4)

a) Will the two line-segments $S_1$and S_2 be checked for intersection at any point of time? If yes, identify the event point at which they will be checked. Else, justify your answer. [2]

b) Identify the event point at which the line segments $S_2$and S_4will be checked for intersection. Justify your answer. [2]

c) Line segments S1 and S4 will not be checked for intersection, as S1 and S4 are not neighbours with S5 placed in between them in the sweep-line status structure, T. Verify the correctness of this statement. [2]

d) Identify the event point at which the left-most intersection will be detected. Show the cross-product calculations to prove the intersection. [4]

Solution:

Event ordering:

(1,0)
(1,4)
(2,2)
(5,3)
(7,3)
(8,3)
(10,1)
(11,0)
(12,2)
(13,4)

a) S1 & S2 will be checked for intersection at the event (1,4) when the S2 will be placed above S1 in the total preorder
   S2
   |
   S1
b) S2 & S4 will not be checked for intersection because S2 will be removed from the total preorder even before S4 is inserted
c) "Line segments S1 and S4 will not be checked for intersection, as S1 and S4 are not neighbours with S5 placed in between them in the sweep-line status structure, T"
   This statement is wrong because at the event point (8,3) S4 will be inserted above S1 in the total preorder and at that point no other segments are in the preorder. **So, S1 and S4 will be the neighbours at that event point and so they will be checked for intersection** even before S5 is inserted in the preorder at the point (10,1)
d) The left-most intersection will be detected at the event point (10,1) when S5 is inserted into the preorder.
   At this point, the preorder will be as follows
   S4
   |
   S5
   |
   S1
   So, S5 should be checked with both S4 and S1
   i) S4= (8,3), (12,2)
   ii) S5= (10,1), (13,4)

   Let P1=(8,3), P2= (12,2), P3=(10,1) and P4=(13,4)

   D1=direction(P3,P4,P1)= (p1-p3)X(P4-P3) = [-2 2] X [3 3] = -12

   D2=direction(P3,P4,P2)=(p2-p3)X(P4-P3) = [2 1]X[3 3]=3

   D3=direction(P1,P2,P3)=(P3-P1)X(P2-P1) = [2 -2] X[4 -1]=6

D4=direction(P1,P2,P4)=(P4-P1)X(P2-P1) = [5 1]X[4 -1] = -9

Because (D1 < 0 and D2 > 0) and (D3 > 0 and D4 < 0) is true, S4 and S5 intersect.

After reporting this left-most intersection (S4 and S5), algorithm will terminate. It will not check for S5 and S1.

These can be verified with the visual representation given below.