11장. 우선순위 큐

1. 가
2. 다
3. b, c
4. 라
5. 라
6. 다

7.
가. 9 5 6 3 2　　나. 20 9 10 3 2 5 6

8.
가. 과정생략. 결과배열 38, 35, 33, 28, 25, 8, 2, 6, 16, 4
나. 과정생략. 결과배열 38, 28, 35, 25, 8, 33, 2, 16, 6, 4
다. 35 28 33 16 25 8 2 6 4

9.
가. 과정생략. 결과배열 93, 60, 12, 34, 35, 8, 11, 9, 7, 26
나. 과정생략. 결과배열 93, 60, 13, 35, 34, 8, 11, 26, 7, 9
다. 9 35 12 26 34 8 11 9 7

10. 과정생략
결과배열 X T P R S O N G E A A M I L E

11. 45 44 32 23 11 25 21 13

12. 결과배열 32 23 25 13 11 15 21

13.
가. 40 35 30 10 20 15 25
나. 80 40 30 35 20 15 25 10

14. 30 25 26 13 10 3 13 4 11

15.
13 4 25 26 10 3 30 11 14
13 4 30 26 10 3 25 11 14
13 26 30 14 10 3 25 11 4
30 26 25 14 10 3 13 11 4

16.
Level 0 (1) + Level 1(2) + Level 2(4) + Level 3(8) + Level 4(7)
높이는 최대 레벨 수인 3

17.
가. 77 59 35 55 1 8 11 40
나. 59 55 35 40 1 8 11

18. if ((i < MaxIndex) || ((Data[i] > Data[2i]) && (Data[i] > Data[2i+1])))

19. 0번

20.
가. 들어오는 순서대로 연결 리스트의 가장 첫 노드로 삽입
나.
```
node* PriorityQueue::RemoveMax( )
{ if (Head != NULL)
  {    int Maximum = -32767;
       for (node *Temp = Head; Temp != NULL; Temp=Temp->Next)
       {   if (Temp->Key > Maximum)
           {    Maximum = Temp->Key;
                node* MaxNode = Temp;
           }
       }
       ListRemove(MaxNode);
  }
}
```

21.
CAB
FDEACG
HGFDCBEA

22.
가. (15 + 19 + 31 + 19) / 4
나. (4 + 16 + 24 + 39) / 4

23.
```
void IntHeap::InsertItem(int Key)
{ HeapArray[Size] = Key;
  for (i = Size; i > 1; i = Floor(i/2))
       if (HeapArray[i] < HeapArray[i/2]
       {    int Temp = HeapArray[i];
            HeapArray[i] = HeapArray[i/2];
            HeapArray[i/2] = Temp;
       }
}
```

24.하향식 O(NlogN), 상향식 O(N)

25.
가. 100 90 80 35 8 50 65 15
나. 150 135 80 35 100 50 65 15 8 90

26.
가.
DeleteMin: O(N) 정렬 안되어 있으므로 선형검색 O(N). 삭제후 쉬프트에 O(N)
DeleteMax: O(N) 정렬 안되어 있으므로 선형검색 O(N). 삭제후 쉬프트에 O(N)

AddQueue: O(1) 정렬 안되어 있으므로 맨 앞이나 맨 뒤에 삽입

나.

DeleteMin: O(N) 처음 것 지운 뒤 나머지 모두 쉬프트 작업(N)
DeleteMax: O(1) 제일 마지막 것 삭제하므로
AddQueue: O(N) 삽입위치 찾기(N 또는 lgN), 이후 쉬프트 작업(N)

다.
DeleteMin: O(1) 헤드로 단번에 접근, 쉬프트 없음.
DeleteMax: O(1) 테일로 단번에 접근, 쉬프트 없음
AddQueue: O9N) 최악의 경우 제일 뒤에 삽입

12장. 탐색 알고리즘

1. 다
2. 라
3. 다
4. 다
5. 가
6. 가
7. 가
8. 다
9. 라

10. O(1)을 지향한다.

11. 클러스터 크기가 커질 수록 새로운 레코드가 그 내부 인덱스로 해쉬될 확률이 높아진다.

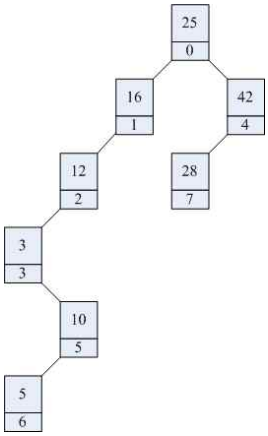
12.
가. 5 나. 19 다. 9

13.
가. 4 5 6 7 8
나. 4 5 8 13 6
다. 4 6 5 7 8

14. 헤더까지 해쉬하는데 O(1)이고 헤더로부터 충돌 키이를 가진 레코드를 찾는 데 O(lgN) 이므로 결과적으로 O(lgN)이다. 여기서 N은 헤더가 있는 인덱스로 충돌된 레코드들의 수를 의미한다.

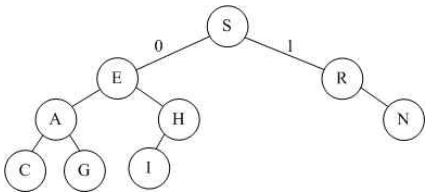
15.
가. 4회 정도 (0+17)/2, (1+7)/2, (5+7)/2, (7+7)/2
나. 2회 정도

16.
가.

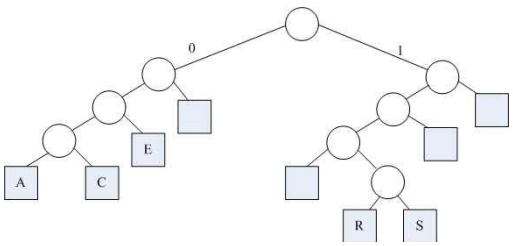


나. 18/8

17.
A = 00001, S = 10011, E = 00101, R = 10010, C = 00011, G = 00111, H = 01000, I = 01001, N = 01110
가. 디지털 탐색트리 (SEARCHING 순으로 입력)



나. 기수탐색트리 (SEARC 순으로 입력)



18.
가.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
	224563	234535			199646	140146		137457	214560	162144		214577	144468

나.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
	224563	234535	162144		199646	140146		137457	214560			214577	144468

19.
가. 1차 클러스터, 2차 클러스터. 2차 클러스터가 같은 인덱스로 해쉬되는 키이에 적용된다고 볼 때 1차 클러스터는 당연히 2차 클러스터이다.

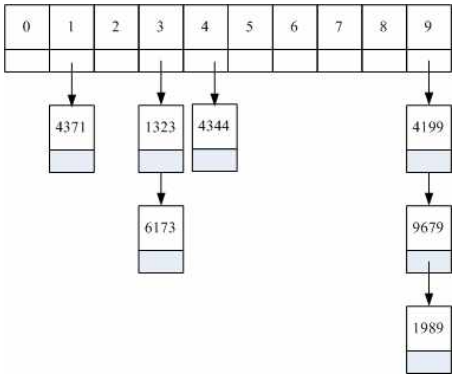
나. 비 클러스터

다. 1차 클러스터, 2차 클러스터. 1차 클러스터가 발생하는 이유는 클러스터 중간으로 해쉬된 키이도 모두 같은 순서로 진행하면서 클러스터 크기를 키우기 때문이다.

라. 비 클러스터

마. 비 클러스터

20.
가.



나.

0	1	2	3	4	5	6	7	8	9
9679	4371	1989	1323	6173	4344				4199

다.

0	1	2	3	4	5	6	7	8	9
9679	4371		1323	6173	4344			1989	4199

22. 키이가 11의 배수이면 간격이 0이 되므로 충돌 해결이 불가능함.

23.
```
int Modulo-119(stringClass Key)
{ int Sum = Key[0];
  int i = 1;
  while (i < Key.Length)
  {     Sum = (Sum + Key[i]) % 119;
        i++;
  }
  return Sum;
}
```

24.
가. 그림생략
나.

0	1	2	3	4	5	6	7	8	9	10
	23	46	12	59	78	2	3			87

다.

0	1	2	3	4	5	6	7	8	9	10
87	23	46	2	59	12		3			78

라.

0	1	2	3	4	5	6	7	8	9	10
3	23	46	12	59		2	78			87

25. 테이블 크기 29

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
		350	90					95 617	38					14					19 164							26	56	

26.
디지털 탐색트리: lgN
기수 탐색 트라이: lgN
다중 링크 탐색 트라이: $\log_M N$

27.
가. 1, 2(앞에서부터 탐색) 또는 14, 15(뒤에서부터 탐색)
나. 8, 4, 12

28.

```
bool Have42(const int Data[ ], int Size)
{ int First = 0; Last = (Size-1);
  while(First <= Last)
  { Middle = (First + Last) /2;
    if (Data[Middle].Key = = 42)
        return TRUE;
    else if (Data[Middle].Key > 42)
        Last = Middle - 1;
    else
        First = Middle + 1;
   }
   return FALSE;
}
```

29.

0	1	2	3	4	5	6	7	8
0	27	29	20	18	5			

30.

0	1	2	3	4	5	6	7	8
0		29 20 18			5			

31. 81/811

32.
가. 부하율 = 2/3 테이블 크기 1500
나. 부하율 = 2 테이블 크기 500

33. 클러스터 크기가 클 수록 더 커질 확률이 높아지기 때문이다.

34.

가.

0	1	2	3	4	5	6	7	8	9
		1326		4344	9679	6175	4371		4199

나.

0	1	2	3	4	5	6	7	8	9
	9679	1326		4344			4371	6175	4199

다.

0	1	2	3	4	5	6	7	8	9
		1326	9679	4344			4371	6175	4199

35.

0	1	2	3	4	5	6	7	8	9
4689	7841		4589	6355	7485	9415		5124	1999

36. 트리의 균형상태를 최대한 오래도록 유지하기 위해서

37. 7개

38. (여러가지 답변이 가능함)
가. 문자열 아닌 정수 키이이므로 기수 탐색보다 해쉬를 사용하는 것이 유리.
나. 균형 탐색트리가 최악의 경우에도 lgN의 효율을 보인다. 또 이진 탐색트리이므로 중위순회하면 정렬된 순서로 방문할 수 있다.

39.
가. 배열. lgN 효율의 이진탐색이 가능
나. 연결 리스트. 삽입 및 삭제에 따른 쉬프트가 불필요하다.
다. 이진 탐색트리. 배열은 이진탐색은 가능하지만 쉬프트가 필요하다. 반대로 연결리스트는 쉬프트는 불필요하지만 이진탐색을 가하기 어렵다. 쉬프트 없이 이진탐색을 가할 수 있는 것이 이진 탐색트리이다.

40. (Temp + Name[i++]) % Size

41.
가. lg 150,000 ≈ 17
나. 최악의 높이 7 * 20 = 140, 평균높이는 lg 150,000 ≈ 17
다. 가: 단어(키이) 나: 비트(키이)

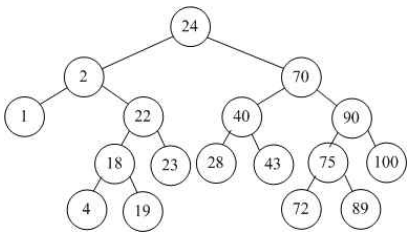
42.

0	1	2	3	4	5	6	7	8	9	10
Chung	Kim		Lee	Yoo				Youn	Choi	Park

13장. 균형 탐색트리

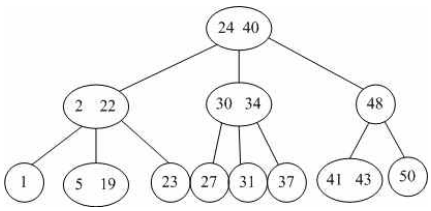
1. False
2. False
3. 나
4. 다
5. 다
6. 라
7. 라
8. 라
9. 다
10. 다

11.

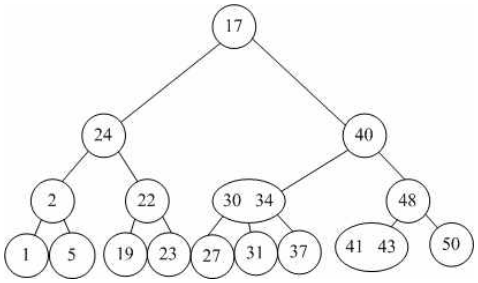


12. Leftmost Child Node (RB, 2-3-4)

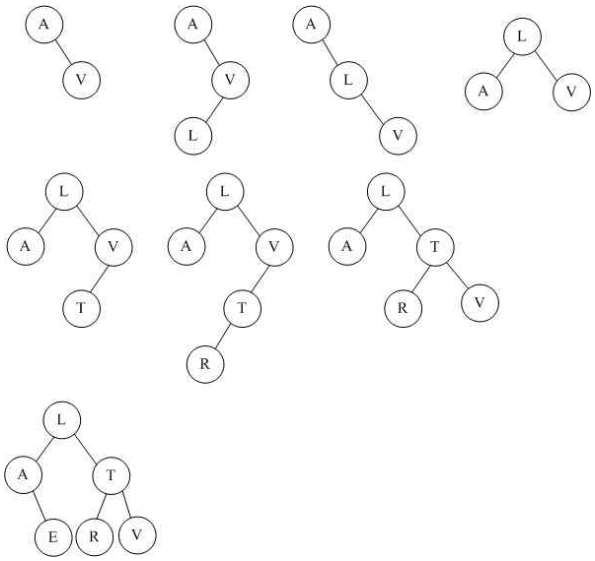
13.
가.



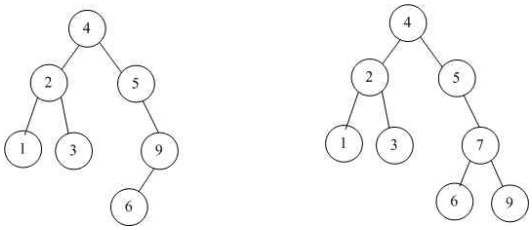
나.



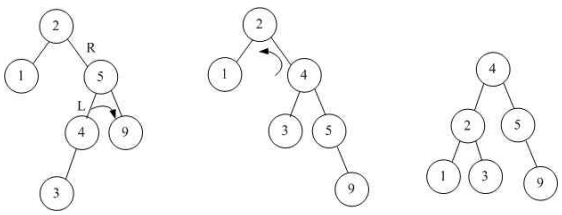
14.



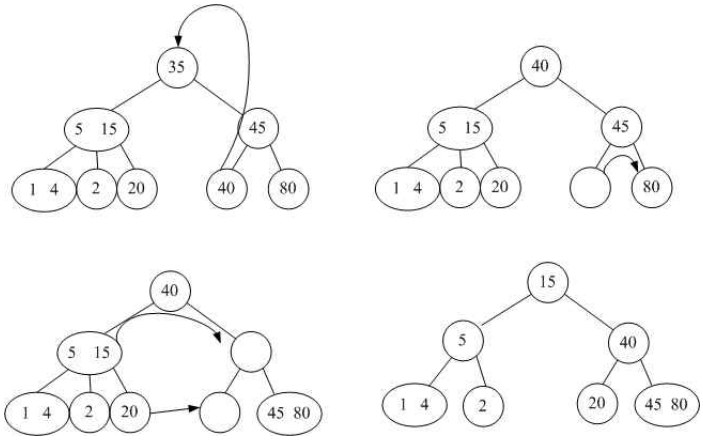
15. 중간, 최종결과



16.

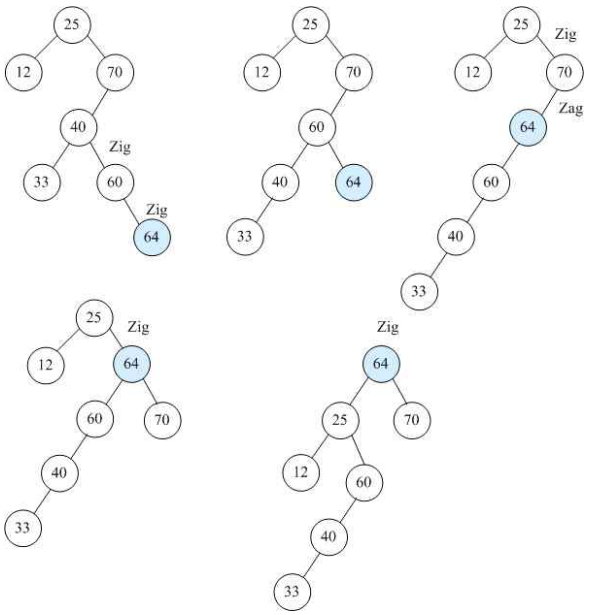


17.

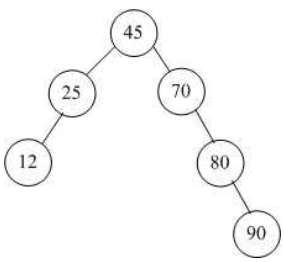


18. 노드 70과 노드 80 사이 링크를 레드로 바꾼다.

19.

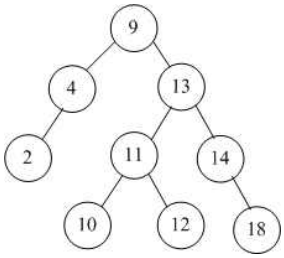


20.

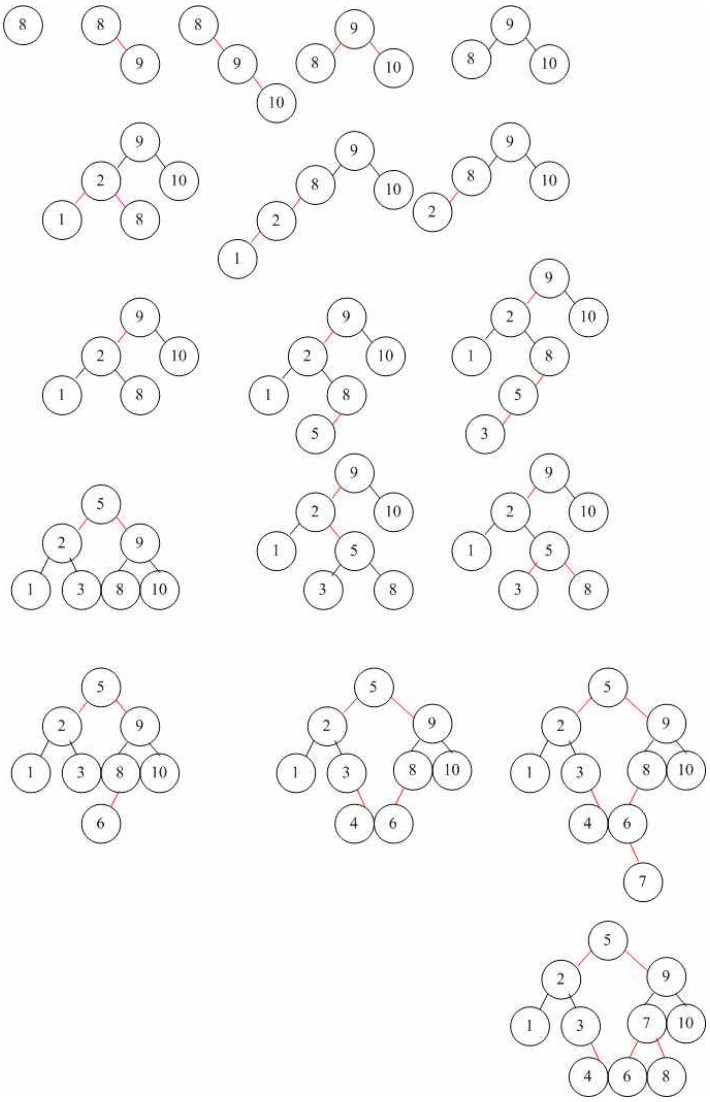


45, 25, 12, 70, 80, 90

21.
나.

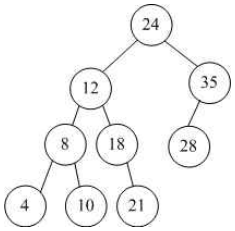


22.

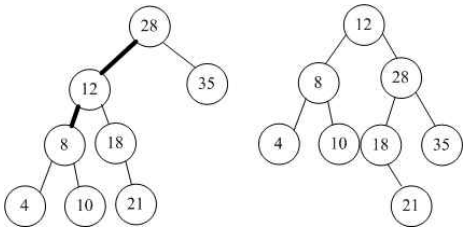


Rotation, Color Change, Color Flip을 연속적으로 가해야 함. 회전은 단일회전 또는 경우에 따라서는 연속회전.

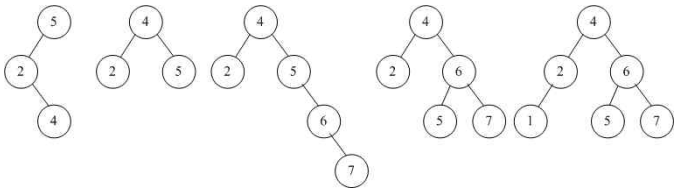
23.
가.



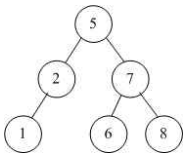
나.



24.
가.



나.

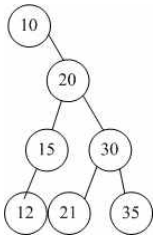


25. (최종결과 참조)

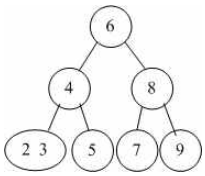
26. 중위순회 선행자(Predecessor)인 29를 루트노드로 복사하하고, 부모노드인 30의 링크를 끊으면 된다.

27.
가. abcd 나. bc 다. bc 라. cd 마. ad

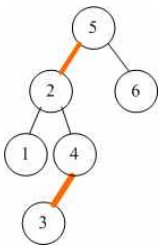
28.
가. 2-1 = 1 나. -1 다. 0
라.



29.



30.
가. 4-2-3으로 내려오는 경로가 연속적으로 레드 링크이다.
나.



31. 43개
32. 인터넷 조사발표 문제

14장. 그래프 알고리즘

1. True
2. False
3. False
4. False
5. False
6. 나
7. 나
8. 다
9. 가
10. 나
11. 마
12. 다
13. 나
14. 나
15. 라
16. 가
17. 가

18. m[i]를 시작주소로 하는 인접행렬의 변수공간이 반납되지 않는다.

19. 필요시 참조호출을 사용하기 위하여

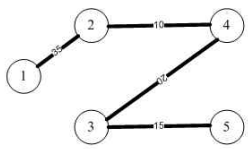
20. 연결리스트의 헤더를 집어넣기 위하여. 인접행렬에서는 첫 행렬을 잡을 때 그 크기가 동적으로 생성. 인접 리스트에서는 간선이 추가될 때마다 새 노드가 동적으로 생성됨.

21.
가.
```
graphPtr DeleteEdge(graphPtr g, int V1, int V2)
{ if (g->M[V1][V2] = = 1)
   { g->E--;                    전체 간선의 수를 줄임
     g->M[V1][V2] = 0;
   }
}
```

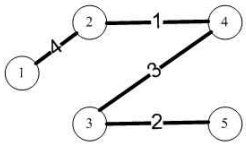
나.
```
graphPtr DeleteEdge(graphPtr g, int V1, int V2)
{ Nptr Temp = g->L[V1];
   Assert (Temp != NULL);
   if (Temp->Data = = V2)
        g->L[V1] = Temp->Nest;
   else
        ListDelete(g->L[V1], V2);   리스트 삭제함수 참조
}
```

22.

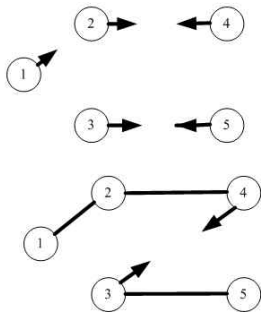
1에서 시작하는 프림



크루스칼 선택순서

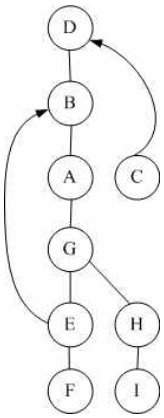


솔린



23.

가. 교재참조
나. 교재참조
다. Yes
라. No
마.



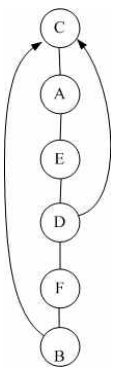
따라서 BE와 CD를 없애면 된다.
바. B, G, E, H

24.

```
typedef struct {
    int Data;
    int Weight;
    node* Next;
} node;
```

25.
가. Yes
나. 2
다. DCABJGEHFI (화살표 표시 추가)
라. 탐색순서는 DABJGEHFIC
그런데 이건 위상정렬 순서가 아님
위상정렬 순서는 백트랙 순서 DCAGHIFEBJ (화살표 표시 추가)

26. 교재참조
27.
가. CAEDFB
나.



다. Yes. 관절점 없음
라. 행렬 엔트리 모두가 1임
마. 그래프 정점 사이가 모두 직접 연결된 완전 그래프 형태

28. 사이클이 존재하므로 위상정렬이 불가능함.

29.
가. 8: 1, 7 1: 8, 2 3: 6, 4
나. 교재참조
다. 63, 6543, 67123, 678123
라.

	1	2	3	4	5	6	7	8
8		6						**3**
2		**6**					7	
7			8				**7**	
3			**8**			10		
6				15	18	**10**		
4				**15**	18			
5					**18**			

30.
```
if (A[i,j] > A[i,k]+A[k,j])
      A[i,j] = A[i,k]+A[k.j];
```
O(N cubic)

31.
가. DE: O(1), DIE: O(N) = O(V) V는 Vertex 개수
나. DE: O(N) = O(V), DIE: O(V+2E)

32.
가. 교재참조
나. 0145236789
다. 0147562389
라. 제일 왼쪽칼럼은 단계별 최소비용 정점선택

	0	1	2	3	4	5	6	7	8	9
1		**1**			2			5		
2			**2**		2			5		
4				4	**2**	4		5		
5				4		**3**		5		
3				**4**			6	5		
6							**5**	5		
7								**5**	6	5
9									6	**5**
6									**6**	

33.

	A	B	C	D	E	F
A		3E	7F	9B	1	3
B			6	4		
C						
D			2			
E		2	8B	6B		
F			4			

34. (nextV <= V)

35.

	A	B	C	D	E
BE					B
CA	C				
EA			B		
AD				B	

36
가.

	A	B	C	D	E
	F	F	A	B	D

나.

	A	B	C	D	E
	F	F	A	F	F

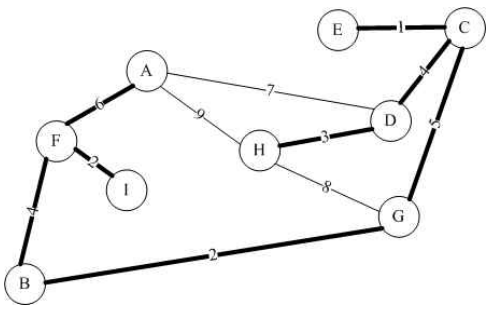
다.

	A	B	C	D	E
	F	F	A	F	B

37. 현재 갈 수 있는 노드 중 가중치 비용이 최소인 노드를 선택한다.

38. 그대로 계산하되, 최종결과에서 최소비용이 없으면 갈 수 있는 길이 없는 것으로 간주하면 된다.

39. 방법에 따라서 다르지만 세 방법 모두 다음 그림과 같은 결론에 이를 수도 있다.



40.

	A	B	C	D	E
A		8		9	4
B		8	5	9	
C		7		8	
D				8	
E					

41.
가. GADBEFC
나. GDEFC. 이 방식은 DFS가 전체 탐색경로를 드러내지 못한다. 즉, 시작하는 정점에 따라서 포리스트가 될 수 있다. 따라서 이 경우에 위상정렬에 깊이우선 탐색을 가하는 것은 바람직하지 않다.

42.
가.
S-4 = 4/4
4-5= 4/4
2-5= 4/4
5-1= 4/4
S-2=10/10
2-3=13/6
301=10/10
나. Mincut = 14, (S-4, S-2를 잘랐을 때)

15장. 알고리즘의 설계

1. True
2. False
3. False
4. False
5. False
6. True
7. True
8. 라
9. 다
10. 4
11. 나
12. Yes

13. A, B 모두 저명인사라면 B가 A를 알고 A가 B를 알아야 한다. 그런데 저명인사는 다른 사람을 몰라야 한다.

14. A=01 B=011이면 A는 B의 접두사

15. 100원, 75원, 50원, 10원, 1원의 화폐체계. 125원을 거슬러 달라할 때 탐욕 알고리즘은 8개이지만 최소 동전수는 2개(75원 + 50원)이다.

16. 결정트리를 기준으로 말할 때, 깊이우선 탐색은 모든 노드를 방문하지만 백 트랙 알고리즘은 그 노드에서 출발하는 정답이 없다고 판단될 때에는 그 노드 아래로 가지 않는다.

17. 최적분기 알고리즘은 목적함수 값의 타당성을 기준으로 노드를 방문할 것인가를 판단하지만 백 트랙 알고리즘은 단지 정답의 유무에 의해 판단한다.

18. 라스베가스 알고리즘은 정답을 보장하지만 몬테카를로 알고리즘은 정답을 보장하지는 않는다.

19. (Move (N-1, Z, Y, X); 지수함수적 복잡도, 링을 옮기는 횟수는 2^N. 사람이 초당 1번씩 옮길 수 있다면 64 개의 링을 모두 옮기는데에는 50만년이 소요됨.

20. 탐욕 알고리즘을 사용하여 p(5)=4, p(6)=8 까지 진행했을 때 그 다음 놓을 자리가 없게 된다. 즉, 백 트랙 없이 탐욕 알고리즘으로는 더 이상 진행할 수 없게 되어버린다.

21. 가. 3 6 8 2 4 1 7 5 나. 4 6 1 5 2 8 3 7

22. 94: A-1, B-3, C-4, D-2

23. 135: A-4, B-2, C-3, D-1

24.
가.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1원	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
4원	0	1	2	3	1	2	3	4	2	3	4	5	3	4	5
7원	0	1	2	3	1	2	3	1	2	3	4	2	3	4	2
8원	0	1	2	3	1	2	3	1	1	2	3	2	2	3	2

표 24 거스름돈 답안

나. 7원짜리 2개
다. 동적 알고리즘: 7원, 7원 탐욕 알고리즘: 8원, 4원, 1원, 1원

25.
a: 10
b: 001
c: 00000
d: 0001
e: 11
f: 00001
g: 01

평균길이 23/7 비트

26. 700번

27. 가. 20(P_1: 9, 6; P_2: 9, 6; P_3: 7, 7, 6). 또는 18(P_1: 9, 9; P_2: 7, 7; P_3: 6, 6, 6.)
나. 18(P_1: 8, 7; P_2: 8, 5, 5; P_3: 7, 7, 4)이 최적

28. 교재설명 참조

29. (A(BC))D
ABC = A(BC) = 28
ABC = (AB)C = 48
BCD = B(CD) = 60
BCD = (BC)D = 32
ABCD = A(BCD) = 72
ABCD = (AB)(CD) = 114
ABCD = (ABC)D = 68

	A	B	C	D	
A		24 B	28 B	68 D	
B			12 C	32 D	
C				30 D	
D					

30.
A:000
E:01
R:001
D:11
C:10

31.
C:0001
D:001
E:1
F:00001
K:000000
L:010
U:011
Z:000001

32. 분할정복은 부분문제 사이에 연관성을 두지 않고 따라서 하나의 부분문제 해결책을 다른 부분문제의 해결에 재사용하지 않는 것이다. 동적 프로그래밍은 반대로 부분문제 사이에 연관성을 두고 하나의 부분문제 해결책을 다른 부분문제 해결에 재사용하는 것이다.

33. 상향식 동적 프로그래밍은 작은 문제로의 해결책을 점차 쌓아나가서 크기 N의 문제를 해결하는 것이고, 하향식 동적 프로그래밍은 크기 N의 문제를 푸는 과정에서 이전에 계산된 부분문제의 해결책을 재사용한다.

34. 2 4 5 2 3 2 7에서 2-4를 제거, 5-2를 제거, 3-2를 제거하면 7이 남지만 7은 과반수가 아니다.

35. 여기다 놓을까 말까를 결정할 때, 만약 동전이 있어서 그걸 던져서 앞면이 나오면 거기다 놓고 아니면 다른 곳을 시도하게 할 수 있다. 그런데 그 동전이 마술 동전이라면 그 동전이 시키는 대로만 하면 항상 문제의 해결에 이르게 되므로 상식적인 시간 내에 문제를 해결할 수 있다.

36.

가. 361 ！　　나. NP의 문제　　다. 백 트랙킹

37. (예) 크기 15의 배낭에 최대가치 40을 담을 수 있는가 없는가?

38. 그 문제를 P 시간에 해결하는 알고리즘이 개발 되었을 때

39. NP 문제 중에서도 그 문제만 풀리면 다른 모둔 NP의 문제가 동시에 풀리는 그런 NP 문제를 NPC의 문제라 한다.

40. 교재 요약 마지막 부분 참조