알고리즘과 그림으로 이해하고 C로 구현하는

자료구조 입문

모범 답안

본 모범답안에 실린 소스는 프리렉 홈페이지에서 다운로드 받을 수 있습니다

내용

[1장 3](#_Toc488179653)

[2장 5](#_Toc488179654)

[3장 10](#_Toc488179655)

[4장 15](#_Toc488179656)

[5장 17](#_Toc488179657)

[6장 21](#_Toc488179658)

[7장 39](#_Toc488179659)

[8장 41](#_Toc488179660)

[9장 43](#_Toc488179661)

[9장 45](#_Toc488179662)

[11장 50](#_Toc488179663)

[12장 57](#_Toc488179664)

# 1장

01\_

정의: 자료구조는 컴퓨터에 자료를 효율적으로 저장하는 방식입니다

목적: 자료구조의 목적은 다음과 같이 3가지 입니다.

1. 프로그램에서 사용하는 '메모리를 절약'하기 위해 자료구조를 사용합니다.
2. 프로그램 실행 시간을 단축하기 위해 자료구조를 사용합니다.
3. 프로그램의 구현을 쉽게 하기 위해 자료구조를 사용합니다.

02\_

1. 선형구조에 속하는 자료구조: 리스트, 스택, 큐, 덱
2. 비선형구조에 속하는 자료구조: 트리, 그래프
3. 알고리즘의 정의: 주어진 문제를 해결하기 위한 절차

4) 알고리즘의 5가지 특성:

a. 입력: 외부에서 제공되는 자료가 0개 이상 있어야 한다

b. 출력: 적어도 1개 이상의 결과를 만들어야 한다

c. 명백성: 각 명령어는 의미가 모호하지 않고 명확해야 한다

d. 유한성: 한정된 수의 단계 뒤에는 반드시 종료된다. 무한히 동작해서는 안된다

e. 유효성: 모든 명령은 실행 가능한 연산이어야 한다.

예를 들어 0으로 나누는 연산과 같이 실행할 수 없는 연산을 포함해서는 안 된다.

03\_

1. 시간 복잡도 함수

n \* (1 + 1 + 1 + n \* 3 ) = 3n2 + 3n

설명: (바깥 루프가 n번) \* ( 비교 연산 1번 + 증가 연산 1 번 + 대입 연산 1 번

+ 내부 루프가 n번 \* ( 비교 연산 1번 + 증가 연산 1번 + 더하기 연산 1번) )

1. 빅-오 표기법

O( 3n2 + 3n ) = O( n2 )

04\_

1. O( 6n3 + n! + 5n + 4) = O( n! )
2. O( 4nlog n + 5n2 + 2 ) = O( 5n2 )
3. O( 2n + n3 + 5 ) = O( 2n  )

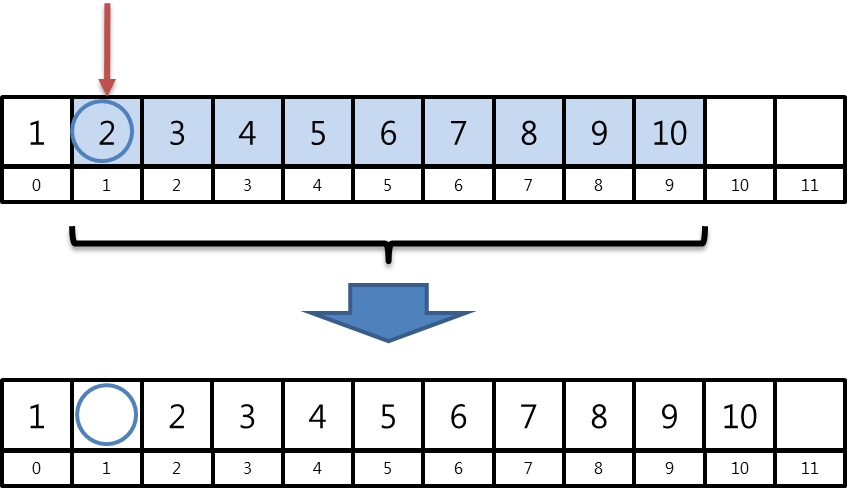
05\_

O( n ) < O( log n ) < O( n log n ) < O( n2 ) < O( n3 ) < O( 2n ) < O( n! )

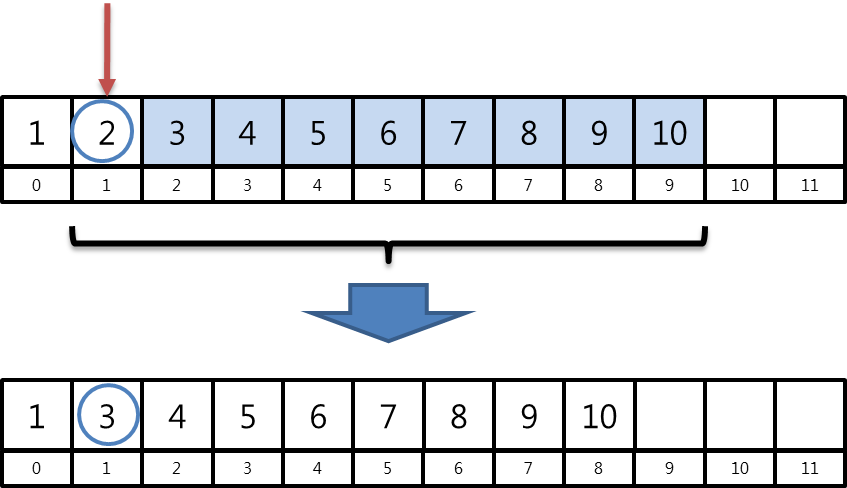
# 2장

01\_

1) 9개



2) 8개



02\_

|  |
| --- |
| int getListLength(ArrayList\* pList)  {  if (pList != NULL)  {  return pList->currentCount;  }  return 0;  } |

03\_

|  |
| --- |
| ArrayList \*createList(int count)  {  ArrayList \*pReturn = NULL;  if (count > 0) {  pReturn = (ArrayList \*)malloc(sizeof(ArrayList));  if (pReturn != NULL) {  pReturn->maxCount = count;  pReturn->currentCount = 0;  pReturn->pData = NULL;  }  else {  printf("오류, 메모리할당 createList()\n");  return NULL;  }  }  else {  printf("오류, 최대 원소 개수는 0이상이어야 합니다\n");  return NULL;  }  pReturn->pData = (ArrayListNode \*)malloc(sizeof(ArrayListNode) \* count);  if (pReturn->pData == NULL) {  printf("오류, 2번째 메모리할당 createArrayList()\n");  free(pReturn); return NULL;  }  memset(pReturn->pData, 0, sizeof(ArrayListNode) \* count);  return pReturn;  } |

04\_

|  |
| --- |
| int addListData(ArrayList\* pList, int position, int data)  {  int ret = 0;  int i = 0;  if (pList != NULL) {  if (pList->currentCount < pList->maxCount) {  if (position >= 0 && position <= pList->currentCount) {  for (i = pList->currentCount - 1; i >= position; i--) {  pList->pData[i + 1] = pList->pData[i];  }  pList->pData[position].data = data;  pList->currentCount++;  }  else {  printf("오류, 위치 인덱스-[%d] 범위 초과, addALElement()\n", position);  ret = 2;  }  }  else {  printf("오류, 리스트 용량 초과-[%d]/[%d]\n", position, pList->maxCount);  ret = 2;  }  }  else {  ret = 1;  }  return ret;  } |

05\_

함수 getListData()의 안정성을 보완한 함수 getListData2()는 아래와 같습니다. 다만, 함수의 반환 값이 “오류 코드”로 바뀌었기 때문에, 실제 리스트의 값을 반환하는 것은 포인터를 이용하였습니다. 자세한 사용 방법은 첨부되는 소스를 참고해 주세요

|  |
| --- |
| int getListData2(ArrayList\* pList, int position, int \*pResult) {  int ret = 0;  if (pList != NULL) {  if (position >= 0 && position < pList->currentCount) {  \*pResult = pList->pData[position].data;  }  else {  printf("오류, 위치 인덱스-[%d] 범위 초과, getListData2()\n", position);  ret = 2;  }  }  else {  ret = 1;  }  return ret;  } |
| < 함수 getListData2()의 사용 방법 예>  status = getListData2(pList, 4, &value);  if (0 == status) {  printf("위치: %d, 값: %d\n", 4, value);  } |

|  |
| --- |
| int removeListData(ArrayList\* pList, int position)  {  int ret = 0;  int i = 0;  if (pList != NULL) {  if (position >= 0 && position < pList->currentCount) {  for (i = position; i < pList->currentCount - 1; i++) {  pList->pData[i] = pList->pData[i + 1];  }  pList->currentCount--;  }  else {  printf("오류, 위치 인덱스-[%d] 범위 초과, removeListData()\n", position);  ret = 2;  }  }  else {  ret = 1;  }  return ret;  } |

06\_

|  |
| --- |
| void deleteList(ArrayList\* pList)  {  if (pList != NULL) {  if (pList->pData != NULL) {  free(pList->pData);  }  free(pList);  }  } |

# 3장

01\_

|  |
| --- |
| (1) 멤버 변수 currentCount를 이용하는 방법 |
| int getListLength(LinkedList\* pList) {  if (NULL != pList) {  return pList->currentCount;  }  return 0;  } |
| (2) 리스트에서 직접 자료의 개수를 구하는 방법 |
| int getListLength2(LinkedList\* pList) {  int ret = 0;  if (NULL != pList) {  LinkedListNode \*pPreNode = pList->headerNode.pLink;  while (pPreNode != NULL) {  ret++;  pPreNode = pPreNode->pLink;  }  }  return ret;  } |

02\_

|  |
| --- |
| int main(int argc, char \*argv[]) {  LinkedList \*pList = NULL;  double value = 0, sum = 0;  int position = 0, i = 0;  pList = createLinkedList();  do {  printf("%d번째 점수는(-1을 입력하면 끝납니다)? ", (position + 1));  scanf\_s("%lf", &value);  if (value != -1) {  addLinkedListData(pList, position, value);  position++;  }  } while (value != -1);  if (position > 0) {  for (i = 0; i < position; i++) {  sum += getLinkedListData(pList, i);  }  printf("%d 명의 평균: %.3f\n", position, (sum / position));  }  else {  printf("1명 이상의 점수를 입력해야 합니다\n");  }  deleteLinkedList(pList);  return 0;  } |

03\_

|  |
| --- |
| LinkedList \*createLinkedList() {  LinkedList \*pReturn = (LinkedList \*)malloc(sizeof(LinkedList));  if (NULL != pReturn) {  memset(pReturn, 0, sizeof(LinkedList));  }  return pReturn;  } |

04\_

|  |
| --- |
| int addLinkedListData(LinkedList\* pList, int position, int data) {  int ret = 0;  int i = 0;  LinkedListNode \*pNewNode = NULL;  LinkedListNode \*pPreNode = NULL;  if (pList != NULL) {  if (position >= 0 && position <= pList->currentCount) {  pNewNode = (LinkedListNode\*)malloc(sizeof(LinkedListNode));  if (pNewNode != NULL) {  pNewNode->data = data;  pPreNode = &(pList->headerNode);  for (i = 0; i < position; i++) {  pPreNode = pPreNode->pLink;  }  pNewNode->pLink = pPreNode->pLink;  pPreNode->pLink = pNewNode;  pList->currentCount++;  }  else {  ret = 1;  printf("오류, 메모리할당 addLinkedListData()\n");  }  }  else {  ret = 1;  printf("오류, 위치 인덱스-[%d], addLinkedListData()\n", position);  }  }  else {  ret = 1;  }  return ret;  } |
| int removeLinkedListData(LinkedList\* pList, int position) {  int ret = 0;  int i = 0;  LinkedListNode \*pDelNode = NULL;  LinkedListNode \*pPreNode = NULL;  if (pList != NULL) {  if (position >= 0 && position < pList->currentCount) {  pPreNode = &(pList->headerNode);  for (i = 0; i < position; i++) {  pPreNode = pPreNode->pLink;  }  pDelNode = pPreNode->pLink;  pPreNode->pLink = pDelNode->pLink;  free(pDelNode);  pList->currentCount--;  }  else {  ret = 1;  printf("오류, 위치 인덱스-[%d], removeLinkedListData()\n", position);  }  }  else {  ret = 1;  }  return ret;  } |

05\_

|  |
| --- |
| void concatLinkedList(LinkedList\* pListA, LinkedList\* pListB) {  LinkedListNode\* pNodeA = NULL;  if (pListA != NULL && pListB != NULL) {  pNodeA = pListA->headerNode.pLink;  **if (pNodeA != NULL) {**  while (pNodeA->pLink != NULL) {  pNodeA = pNodeA->pLink;  }  pNodeA->pLink = pListB->headerNode.pLink;  **}**  **else {**  **pListA->headerNode.pLink = pListB->headerNode.pLink;**  **}**  pListB->headerNode.pLink = NULL;  }  } |

06\_

|  |
| --- |
| void reverseLinkedList(LinkedList\* pList)  {  LinkedListNode \*pNode = NULL, \*pCurrentNode = NULL, \*pPrevNode = NULL;  if (pList != NULL) {  pNode = pList->headerNode.pLink;  while (pNode != NULL) {  pPrevNode = pCurrentNode;  pCurrentNode = pNode;  pNode = pNode->pLink;  pCurrentNode->pLink = pPrevNode;  }  pList->headerNode.pLink = pCurrentNode;  }  } |

# 4장

01\_

|  |  |  |
| --- | --- | --- |
|  | **공통점** | **차이점** |
| 단순 연결 리스트 | 포인터를 이용하여 구현함 | 1) 연결 리스트의 가장 기본이 되는 간단한 구조를 가짐 |
| 원형 연결 리스트 | 2) 연결 리스트의 가장 마지막 노드가 첫 번째 노드와 연결되어 원형을 이루는 구조임 |
| 이중 연결 리스트 | 3) 양방향 링크를 사용하기 때문에, 특정 노드의 이전 노드에 대한 직접 접근이 가능함 |

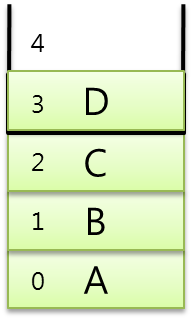
02\_

|  |
| --- |
| PolyList\* polySub(PolyList\* pListA, PolyList\* pListB)  {  PolyList\* pReturn = NULL;  LinkedListNode \*pNodeA = NULL, \*pNodeB = NULL;  double coefSum = 0;  if (pListA != NULL && pListB != NULL) {  pReturn = createLinkedList();  if (pReturn == NULL) {  printf("메모리 할당 오류, polyAdd()\n");  return NULL;  }  pNodeA = pListA->headerNode.pLink;  pNodeB = pListB->headerNode.pLink;  while (pNodeA != NULL && pNodeB != NULL) {  int degreeA = pNodeA->data.degree;  int degreeB = pNodeB->data.degree;  if (degreeA > degreeB) { // (1) A의 차수 > B의 차수  coefSum = pNodeA->data.coef;  addPolyNodeLast(pReturn, coefSum, degreeA);  pNodeA = pNodeA->pLink;  }  else if (degreeA == degreeB) { // (2) A의 차수(degree) == B의 차수  coefSum = pNodeA->data.coef - pNodeB->data.coef;  addPolyNodeLast(pReturn, coefSum, degreeA);  pNodeA = pNodeA->pLink;  pNodeB = pNodeB->pLink;  }  else { // (3) A의 차수 < B의 차수  coefSum = -1 \* pNodeB->data.coef;  addPolyNodeLast(pReturn, coefSum, degreeB);  pNodeB = pNodeB->pLink;  }  }  while (pNodeA != NULL) {  coefSum = pNodeA->data.coef;  addPolyNodeLast(pReturn, coefSum, pNodeA->data.degree);  pNodeA = pNodeA->pLink;  }  while (pNodeB != NULL) {  coefSum = -1 \* pNodeB->data.coef;  addPolyNodeLast(pReturn, coefSum, pNodeB->data.degree);  pNodeB = pNodeB->pLink;  }  }  else  {  printf("오류, NULL 다항식이 전달되었습니다, polyAdd()\n");  }  return pReturn;  } |

# 5장

01\_

(1)



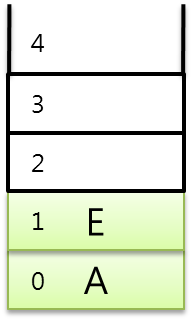
(2)

D → C → B → A

02\_

LIFO는 Last-In-First-Out의 줄임말로 가장 나중에 들어간 자료가 가장 먼저 나온다라는 의미임

03\_



04\_

대표적인 변경 내용은 다음과 같습니다. 전체 소스는 프리렉 홈페이지에서 다운로드 받을 수 있습니다.

|  |
| --- |
| typedef struct ArrayStackNodeType {  **int data;**  } ArrayStackNode;  ….. (중간 생략) ….  int pushAS(ArrayStack\* pStack**, int data)**  {  ….. (중간 생략) ….  void displayArrayStack(ArrayStack \*pStack)  {  ….. (중간 생략) ….  for (i = top - 1; i >= 0; i--) {  printf("[%d]-[%d]\n",  i, pStack->pData[i].data);  }  }  }  ….. (중간 생략) ….  int main(int argc, char \*argv[])  {  ArrayStack \*pStack = NULL;  ArrayStackNode \*pNode = NULL;  pStack = createArrayStack(6);  if (pStack != NULL) {  **pushAS(pStack, 10);**  **pushAS(pStack, 20);**  **pushAS(pStack, 30);**  **pushAS(pStack, 40);**  displayArrayStack(pStack);  pNode = popAS(pStack);  if (pNode != NULL) {  **printf("Pop 값-[%d]\n", pNode->data);**  free(pNode);  }  else {  printf("공백(Empty) 스택\n");  }  displayArrayStack(pStack);  pNode = peekAS(pStack);  if (pNode != NULL) {  **printf("Peek 값-[%d]\n", pNode->data);**  }  else {  printf("공백(Empty) 스택\n");  }  displayArrayStack(pStack);  deleteArrayStack(pStack);  }  return 0;  } |

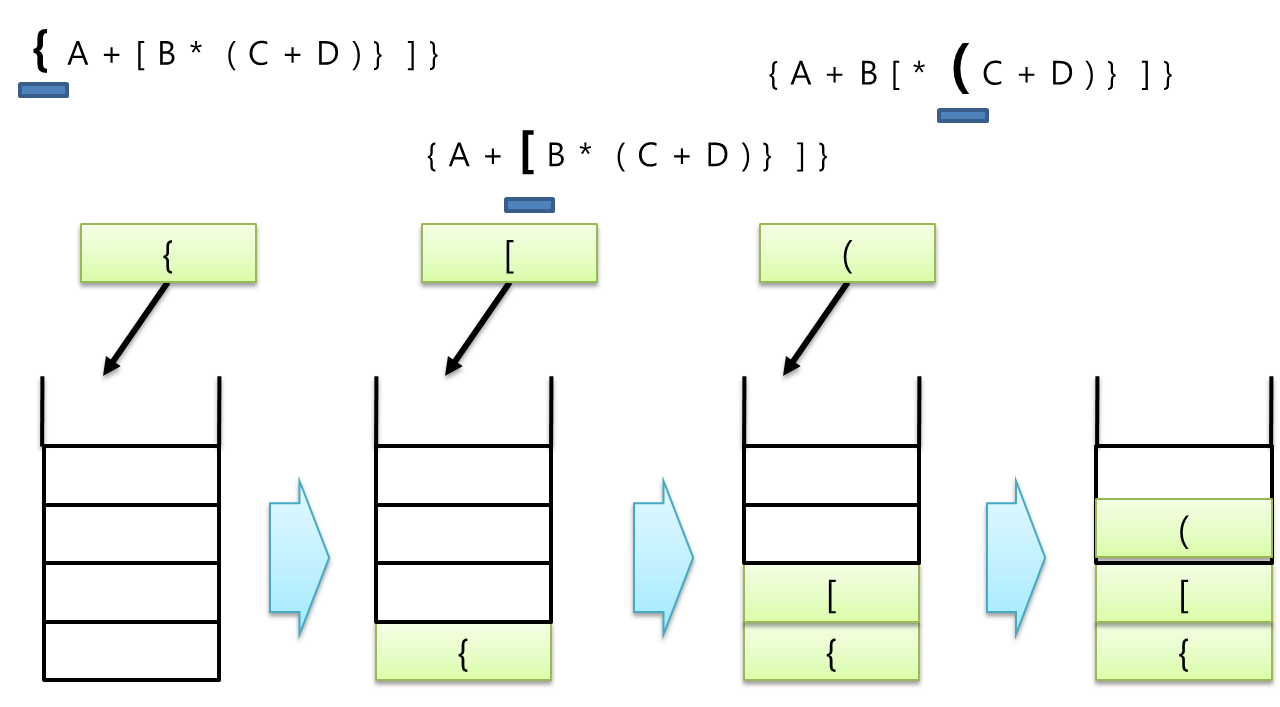
05\_

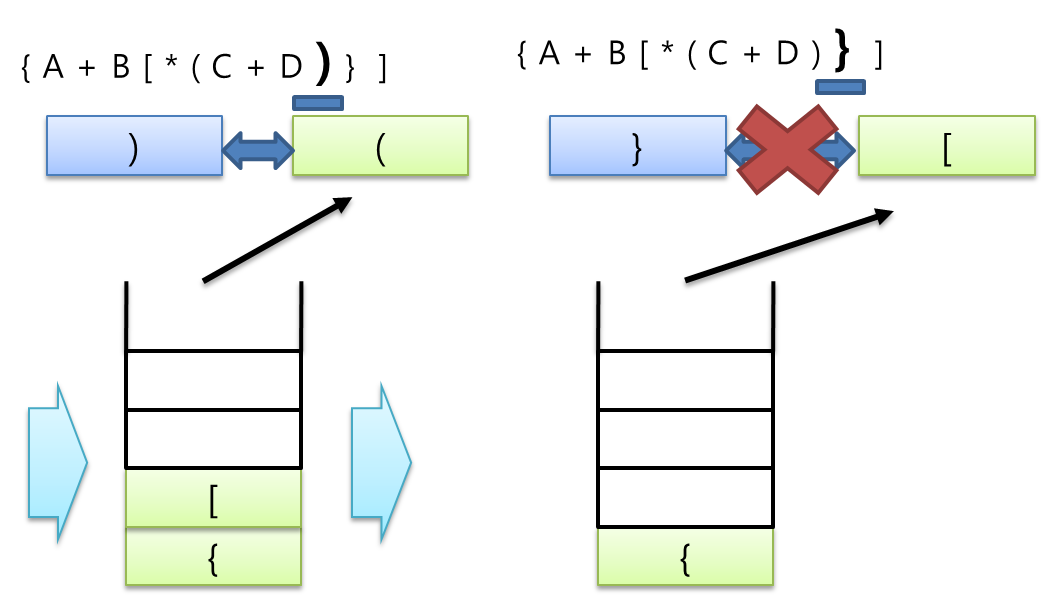
|  |
| --- |
| ArrayStack\* createArrayStack(int size)  {  ArrayStack \*pReturn = NULL;  if (size > 0) {  pReturn = (ArrayStack \*)malloc(sizeof(ArrayStack));  if (pReturn != NULL) {  memset(pReturn, 0, sizeof(ArrayStack));  pReturn->maxCount = size;  }  else {  printf("오류, 메모리 할당, createArrayStack()\n");  return NULL;  }  pReturn->pData = (ArrayStackNode \*)malloc(sizeof(ArrayStackNode) \* size);  if (pReturn->pData != NULL) {  memset(pReturn->pData, 0, sizeof(ArrayStackNode) \* size);  }  else {  printf("오류, 메모리 할당2, createArrayStack()\n");  free(pReturn); return NULL;  }  }  else {  printf("오류, 스택의 크기는 0이상이어야 합니다\n");  return NULL;  }  return pReturn;  } |

# 6장

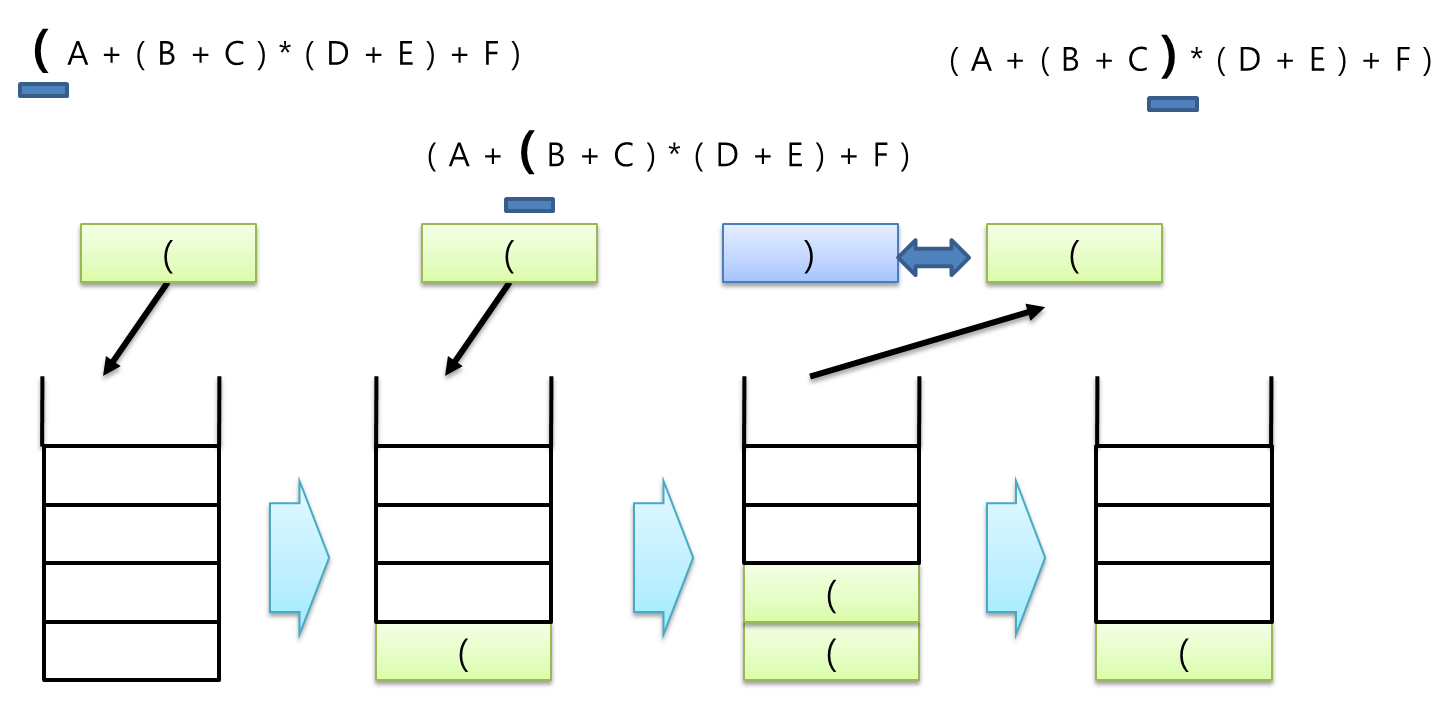
01\_

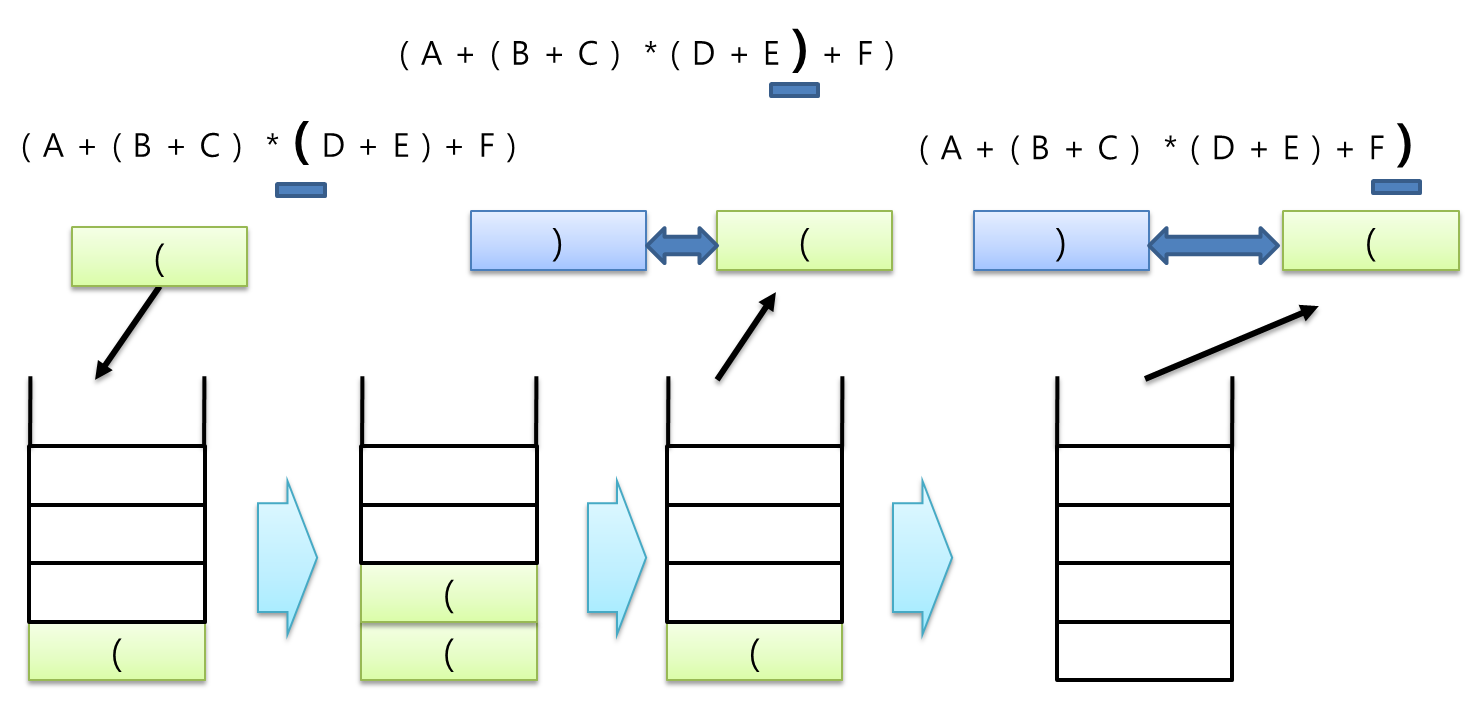
(1)





(2)





02\_

(1) A \* B + C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **단계** | | **처리** | **출력 내용** | **스택의 상태** |
| 1 | **A** \* B + C | 피연산자 A를 출력 | A |  |
| 2 | A **\*** B + C | 연산자 \*를 스택에 푸시 | A |  |
| 3 | A \* **B** + C | 피연산자 B를 출력 | A B |  |
| 4 | A \* B **+** C | 연산자 \*를 출력, 연산자 +를 푸시 | A B \* |  |
| 5 | A \* B + **C** | 피연산자 C를 출력 | A B \* C |  |
| 6 | <종료> | 스택을 팝하여, 연산자 + 출력 | A B \* C + |  |

(2) A + B \* C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **단계** | | **처리** | **출력 내용** | **스택의 상태** |
| 1 | **A** + B \* C | 피연산자 A를 출력 | A |  |
| 2 | A **+** B \* C | 연산자 +를 스택에 푸시 | A |  |
| 3 | A + **B** \* C | 피연산자 B를 출력 | A B |  |
| 4 | A + B **\*** C | 연산자 \*를 푸시 | A B \* |  |
| 5 | A + B \* **C** | 피연산자 C를 출력 | A B C |  |
| 6 | <종료> | 스택을 팝하여, 연산자 \* + 출력 | A B C \* + |  |

(3) A \* B / C

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **단계** | | **처리** | **출력 내용** | **스택의 상태** |
| 1 | **A** \* B / C | 피연산자 A를 출력 | A |  |
| 2 | A **\*** B / C | 연산자 \*를 스택에 푸시 | A |  |
| 3 | A \* **B** / C | 피연산자 B를 출력 | A B |  |
| 4 | A \* B **/** C | 연산자 / 를 푸시 | A B |  |
| 5 | A \* B / **C** | 피연산자 C를 출력 | A B C |  |
| 6 | <종료> | 스택을 팝하여, 연산자 / + 출력 | A B C / \* |  |

(4) A \* ( B + C )

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **단계** | | **처리** | | **출력 내용** | | **스택의 상태** |
| 1 | **A** \* ( B + C ) | **피연산자 A**를 출력 | | A | |  |
| 2) | A **\*** ( B + C ) | **연산자 \***를 스택에 push | | A | |  |
| 3 | A \* **(** B + C ) | **연산자 (**를 스택에 push | | A | |  |
| 4 | A \* ( **B** + C ) | | **피연산자 B**를 출력 | | A B |  |
| 5 | A \* ( B **+** C ) | | **연산자 +**를 스택에 push | | A B |  |
| 6 | A \* ( B + **C** ) | | **피연산자 C**를 출력 | | A B C |  |
| 7 | A \* ( B + C **)** | | **연산자 (를 만날 때까지 pop** | | A B C + |  |
| 8 | <종료> | | **스택에 남은 토큰들 pop** | | A B C + \* |  |

(5) ( A + B ) \* C

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **단계** | | **처리** | | **출력 내용** | | **스택의 상태** |
| 1 | **(** A + B ) \* C | **연산자 (**를 스택에 push | |  | |  |
| 2) | ( **A** + B ) \* C | **피연산자 A**를 출력 | | A | |  |
| 3 | ( A **+** B ) \* C | **연산자 +**를 스택에 push | | A | |  |
| 4 | ( A + **B** ) \* C | | **피연산자 B**를 출력 | | A B |  |
| 5 | ( A + B **)** \* C | | **연산자 (를 만날 때까지 pop** | | A B + |  |
| 6 | ( A + B ) **\*** C | | **연산자 \***를 스택에 push | | A B + |  |
| 7 | ( A + B ) \* **C** | | **피연산자 C**를 출력 | | A B + C |  |
| 8 | <종료> | | **스택에 남은 토큰들 pop** | | A B + C \* |  |

(6) A / B + D \* E – F \* G

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **단계** | | **처리** | **출력 내용** | **스택의 상태** |
| 1 | **A** / B + D \* E – F \* G | 피연산자 A를 출력 | A |  |
| 2 | A **/** B + D \* E – F \* G | 연산자 /를 스택에 푸시 | A |  |
| 3 | A / **B** + D \* E – F \* G | 피연산자 B를 출력 | A B |  |
| 4 | A / B **+** D \* E – F \* G | 연산자 / 를 출력, 연산자 +를 푸시 | A B / |  |
| 5 | A / B + **D** \* E – F \* G | 피연산자 D를 출력 | A B / D |  |
| 6 | A / B + D **\*** E – F \* G | 연산자 \* 스택에 푸시 | A B / D |  |
| 7 | A / B + D \* **E** – F \* G | 피연산자 E를 출력 | A B / D E |  |
| 8 | A / B + D \* E **–** F \* G | 연산자 \*를 출력, 연산자 – 를 스택에 푸시 | A B / D E \* |  |
| 9 | A / B + D \* E – **F** \* G | 피연산자 F를 출력 | A B / D E \* F |  |
| 10 | A / B + D \* E – F \* G | 연산자 \*를 푸시 | A B / D E \* F |  |
| 11 | A / B + D \* E – F \* **G** | 피연산자 G를 출력 | A B / D E \* F G |  |
| 12 | <종료> | 스택을 팝하여, 연산자 \* - + 출력 | A B / D E \* F G \* - + |  |

(7) ( A + ( B \* ( C + D ) ) )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 단계 | | 처리 | 출력 내용 | 스택의 상태 |
| 1 | **(** A + ( B \* ( C + D ) ) ) | **연산자 (**를 스택에 push |  |  |
| 2) | ( **A** + ( B \* ( C + D ) ) ) | **피연산자 A**를 출력 | A |  |
| 3 | ( A **+** ( B \* ( C + D ) ) ) | **연산자 +**를 스택에 push | A |  |
| 4 | ( A + **(** B \* ( C + D ) ) ) | **연산자 (**를 스택에 push | A |  |
| 5 | ( A + ( **B** \* ( C + D ) ) ) | **피연산자 B**를 출력 | A B |  |
| 6 | ( A + ( B **\*** ( C + D ) ) ) | **연산자 \***를 스택에 push | A B |  |
| 7 | ( A + ( B \* **(** C + D ) ) ) | **연산자 (**를 스택에 push | A B |  |
| 8 | ( A + ( B \* ( **C** + D ) ) ) | **피연산자 C**를 출력 | A B C |  |
| 9 | ( A + ( B \* ( C **+** D ) ) ) | **연산자 +**를 스택에 push | A B C |  |
| 10 | ( A + ( B \* ( C + **D** ) ) ) | **피연산자 D**를 출력 | A B C D |  |
| 11 | ( A + ( B \* ( C + D **)** ) ) | **연산자 (를 만날 때까지 pop** | A B C D + |  |
| 12 | ( A + ( B \* ( C + D ) **)** ) | **연산자 (를 만날 때까지 pop** | A B C D + \* |  |
| 13 | ( A + ( B \* ( C + D ) ) **)** | **연산자 (를 만날 때까지 pop** | A B C D + \* + |  |
| 14 | <종료> | **스택에 남은 토큰들 pop** | A B C D + \* + |  |

(8) ( A + ( B + C ) \* ( D + E ) + F )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 단계 | | 처리 | 출력 내용 | 스택의 상태 |
| 1 | **(** A + ( B \* C ) \* ( D + E ) + F ) | **연산자 (**를 스택에 push |  |  |
| 2) | ( **A** + ( B + C ) \* ( D + E ) + F ) | **피연산자 A**를 출력 | A |  |
| 3 | ( A **+** ( B + C ) \* ( D + E ) + F ) | **연산자 +**를 스택에 push | A |  |
| 4 | ( A + **(** B + C ) \* ( D + E ) + F ) | **연산자 (**를 스택에 push | A |  |
| 5 | ( A + ( **B** + C ) \* ( D + E ) + F ) | **피연산자 B**를 출력 | A B |  |
| 6 | ( A + ( B **+** C ) \* ( D + E ) + F ) | **연산자 +**를 스택에 push | A B |  |
| 7 | ( A + ( B \* **C** ) \* ( D + E ) + F ) | **피연산자 C**를 출력 | A B C |  |
| 8 | ( A + ( B + C **)** \* ( D + E ) + F ) | **피연산자 (를 만날 때까지 pop** | A B C + |  |
| 9 | ( A + ( B + C ) **\*** ( D + E ) + F ) | **연산자 \***를 스택에 push | A B C + |  |
| 10 | ( A + ( B + C ) \* **(** D + E ) + F ) | **연산자 (**를 스택에 push | A B C + |  |
| 11 | ( A + ( B + C ) \* ( **D** + E ) + F ) | **피연산자 D**를 출력 | A B C + D |  |
| 12 | ( A + ( B + C ) \* ( D **+** E ) + F ) | **연산자 +**를 스택에 push | A B C + D |  |
| 13 | ( A + ( B \* C ) \* ( D + **E** ) + F ) | **피연산자 E**를 출력 | A B C + D E |  |
| 14 | ( A + ( B \* C ) \* ( D + E **)** + F ) | **연산자 (를 만날 때까지 pop** | A B C + D E + |  |
| 15 | ( A + ( B \* C ) \* ( D + E ) **+** F ) | **연산자 \***를 출력  **연산자 +**를 출력 | A B C + D E + \* + |  |
| 16 | ( A + ( B \* C ) \* ( D + E ) + **F** ) | **피연산자 F**를 출력 | A B C + D E + \* + F |  |
| 17 | ( A + ( B \* C ) \* ( D + E ) + F **)** | **연산자 (를 만날 때까지 pop** | A B C + D E + \* + F + |  |
| 18 | <종료> | **스택에 남은 토큰들 pop** | A B C + D E + \* + F + |  |

(9) ( ( A / ( B + C \* D ) - E )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 단계 | | 처리 | 출력 내용 | 스택의 상태 |
| 1 | **(** ( A / ( B + C \* D ) - E ) | **연산자 (**를 스택에 push |  |  |
| 2 | ( **(** A / ( B + C \* D ) - E ) | **연산자 (**를 스택에 push |  |  |
| 3 | ( ( **A** / ( B + C \* D ) - E ) | **피연산자 A**를 출력 | A |  |
| 4 | ( ( A **/** ( B + C \* D ) - E ) | **연산자 /** 를 push | A |  |
| 5 | ( ( A / **(** B + C \* D ) - E ) | **연산자 (** 를 push | A |  |
| 6 | ( ( A / ( **B** + C \* D ) - E ) | **피연산자 B**를 출력 | A B |  |
| 7 | ( ( A / ( B **+** C \* D ) - E ) | **연산자 +**를 push | A B |  |
| 8 | ( ( A / ( B + **C** \* D ) - E ) | **피연산자 C**를 출력 | A B C |  |
| 9 | ( ( A / ( B + C **\*** D ) - E ) | **연산자 \***를 push | A B C |  |
| 10 | ( ( A / ( B + C \* **D** ) - E ) | **피연산자 D**를 출력 | A B C D |  |
| 11 | ( ( A / ( B + C \* D **)** - E ) | **피연산자 (를 만날 때까지 pop** | A B C D \* + |  |
| 12 | ( ( A / ( B + C \* D ) **-** E ) | **연산자 /**를 출력, 연산자 – 를 스택에 push | A B C D \* + / |  |
| 13 | ( ( A / ( B + C \* D ) - **E** ) | **피연산자 E**를 출력 | A B C D \* + / E |  |
| 14 | ( ( A / ( B + C \* D ) - E **)** | **피연산자 (를 만날 때까지 pop** | A B C D \* + / E - |  |
| 15 | <종료> | **스택에 남은 토큰들 pop** | A B C D \* + / E - |  |

(10) ( A / ( B – C + D ) \* ( E + F ) ) + G

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 단계 | | 처리 | 출력 내용 | 스택의 상태 |
| 1 | **(** A / ( B – C + D ) \* ( E + F ) ) + G | **연산자 (**를 스택에 push |  |  |
| 2 | ( **A** / ( B – C + D ) \* ( E + F ) ) + G | **피연산자 A**를 출력 | A |  |
| 3 | ( A **/** ( B – C + D ) \* ( E + F ) ) + G | **연산자 /**를 스택에 push | A |  |
| 4 | ( A / **(** B – C + D ) \* ( E + F ) ) + G | **피연산자 (** 를 push | A |  |
| 5 | ( A / ( **B** – C + D ) \* ( E + F ) ) + G | **피연산자 B**를 출력 | A B |  |
| 6 | ( A / ( B **–** C + D ) \* ( E + F ) ) + G | **피연산자 -** 를 push | A B |  |
| 7 | ( A / ( B – **C** + D ) \* ( E + F ) ) + G | **피연산자 C**를 출력 | A B C |  |
| 8 | ( A / ( B – C **+** D ) \* ( E + F ) ) + G | **연산자 +**를 스택에 push | A B C |  |
| 9 | ( A / ( B – C + **D** ) \* ( E + F ) ) + G | **피연산자 D**를 출력 | A B C D |  |
| 10 | ( A / ( B – C + D **)** \* ( E + F ) ) + G | **피연산자 (를 만날 때까지 pop** | A B C D + - |  |
| 11 | ( A / ( B – C + D ) **\*** ( E + F ) ) + G | **연산자 \***를 스택에 push | A B C D + - |  |
| 12 | ( A / ( B – C + D ) \* **(** E + F ) ) + G | **연산자 (** 를 스택에 push | A B C D + - |  |
| 13 | ( A / ( B – C + D ) \* ( **E** + F ) ) + G | **피연산자 E**를 출력 | A B C D + - E |  |
| 14 | ( A / ( B – C + D ) \* ( E **+** F ) ) + G | **연산자 +** 를 스택에 push | A B C D + - E |  |
| 15 | ( A / ( B – C + D ) \* ( E + **F** ) ) + G | **피연산자 F**를 출력 | A B C D + - E F |  |
| 16 | ( A / ( B – C + D ) \* ( E + F **)** ) + G | **피연산자 (를 만날 때까지 pop** | A B C D + - E F + |  |
| 17 | ( A / ( B – C + D ) \* ( E + F ) **)** + G | **피연산자 (를 만날 때까지 pop** | A B C D + - E F + \* / |  |
| 18 | ( A / ( B – C + D ) \* ( E + F ) ) **+** G | **연산자 +** 를 스택에 push | A B C D + - E F + \* / |  |
| 19 | ( A / ( B – C + D ) \* ( E + F ) ) + **G** | **피연산자 G**를 출력 | A B C D + - E F + \* / G |  |
| 20 | <종료> | **스택에 남은 토큰들 pop** | A B C D + - E F + \* / G + |  |

04\_

(1)

|  |
| --- |
| char\* reverseString(char \*pSource)  {  char\* pReturn = NULL;  int i = 0, size = 0;  LinkedStack \*pStack = NULL;  LinkedStackNode \*pNode = NULL;  if (pSource == NULL || pSource[0] == 0) {  return NULL;  }  size = (int)strlen(pSource);  pReturn = (char \*)malloc(sizeof(char) \* (size + 1));  if (pReturn == NULL) {  printf("메모리 할당이 실패하였습니다,reverseString()\n");  return NULL;  }  memset(pReturn, 0, sizeof(char) \* (size + 1));  pStack = createLinkedStack();  if (pStack != NULL) {  for (i = 0; i < size; i++) {  pushLS(pStack, pSource[i]);  }  for (i = 0; i < size; i++) {  pNode = popLS(pStack);  if (pNode != NULL) {  pReturn[i] = pNode->data;  free(pNode);  }  }  deleteLinkedStack(pStack);  }  else {  if (pReturn == NULL) {  free(pReturn);  pReturn = NULL;  }  }  return pReturn;  } |

(2)

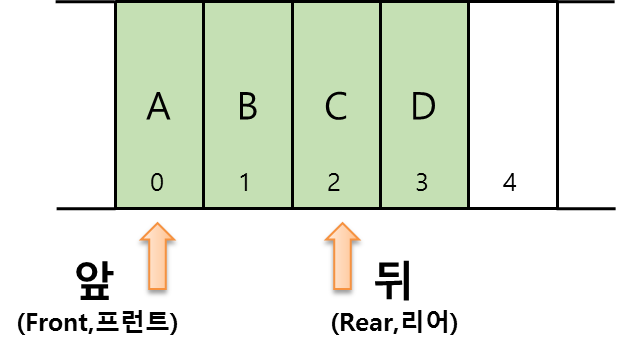
|  |
| --- |
| int checkBracketMatching(char \*pSource)  {  int ret = 0, i = 0;  char symbol = '\0', checkSymbol = '\0';  LinkedStack \*pStack = NULL;  LinkedStackNode \*pNode = NULL;  if (pSource == NULL) {  return -1;  }  pStack = createLinkedStack();  if (pStack != NULL) {  while (0 != pSource[i] && 0 == ret) {  symbol = pSource[i];  switch (symbol) {  case '(':  case '[':  case '{':  pushLS(pStack, symbol);  break;  case ')':  case ']':  case '}':  pNode = popLS(pStack);  if (pNode == NULL) {  // 닫는 괄호가 부족한 경우.  ret = 1;  }  else {  checkSymbol = pNode->data;  if ((symbol == ')' && checkSymbol == '(')  || (symbol == ']' && checkSymbol == '[')  || (symbol == '}' && checkSymbol == '{')) {  // Right case. do nothing.  }  else {  // 괄호의 쌍이 맞지 않는 경우.  ret = 2;  }  free(pNode);  }  break;  } //end-of-switch  i++;  } //end-of-while  if (0 == ret && isLinkedStackEmpty(pStack) == 0) {  // 닫는 괄호가 남은 경우.  ret = 3;  }  deleteLinkedStack(pStack);  }  else {  printf("오류!!! LinkedStack 생성에 실패하였습니다, checkBracketMatching()\n");  ret = -1;  }  return ret;  } |

04\_

: 별도 소스를 참고해 주세요 ( 프리렉 홈페이지에서 다운로드 받을 수 있습니다 )

# 7장

01\_

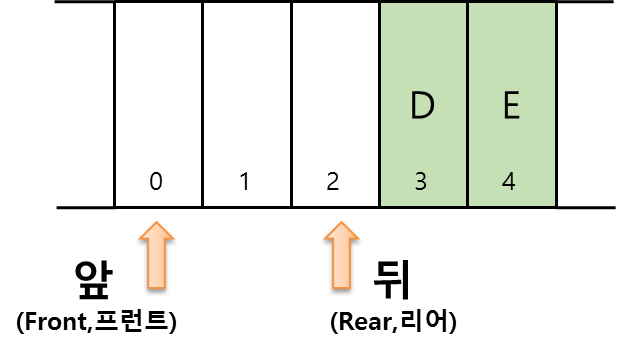
 (1)

(2)

A → B → C → D

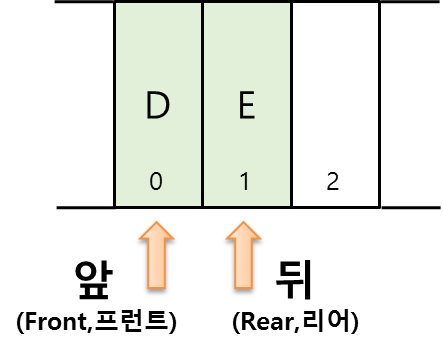
02\_

FIFO는 First-In-First-Out의 줄임말로 먼저 저장된 데이터가 나중 저장된 데이터 보다 항상 앞서 나오는 특성을 이야기 합니다. 이를 다른 말로 선입선출(先入先出)이라고도 합니다.

03\_

크기가 5인 선형 큐 기준

04\_

 크기가 3인 원형 큐 기준

05\_

|  |
| --- |
| ArrayQueue\* createArrayQueue(int size)  {  ArrayQueue \*pReturn = NULL;  if (size > 0) {  pReturn = (ArrayQueue \*)malloc(sizeof(ArrayQueue));  if (pReturn != NULL) {  memset(pReturn, 0, sizeof(ArrayQueue));  pReturn->maxCount = size;  pReturn->front = -1;  pReturn->rear = -1;  }  else {  printf("오류,메모리할당,createArrayQueue()\n");  return NULL;  }  }  else {  printf("오류,큐 크기는 0보다 커야 합니다\n");  return NULL;  }  pReturn->pData = (ArrayQueueNode \*)malloc(sizeof(ArrayQueueNode) \* size);  if (pReturn->pData != NULL) {  memset(pReturn->pData, 0, sizeof(ArrayQueueNode) \* size);  }  else {  printf("오류, 메모리 할당2,createArrayQueue()\n");  free(pReturn);  return NULL;  }  return pReturn;  } |

06\_

: 별도 소스를 참고해 주세요 ( 프리렉 홈페이지에서 다운로드 받을 수 있습니다 )

# 8장

01\_

재귀 호출을 할 때에, 문제의 범위가 줄어들지 않기 때문에 무한 루프에 빠지게 된다.

ret = n \* factorial( n );

02\_

재귀 호출에 종료 조건(기본 경우 혹은 최소 한계)이 없기 때문에 무한 루프에 빠지게 된다.

if (n >= 1) { …. 과 같은 조건이 없음

03\_

1

1

2

1

1

2

5

04\_

|  |
| --- |
| int sum(int n)  {  if (n > 1) {  return sum(n - 1) + n;  }  return n;  } |

05\_

|  |
| --- |
| double sum2(double n)  {  if (n > 1) {  return sum2(n – 1) + (1.0 / n);  }  return 1;  } |

06\_

|  |
| --- |
| int sum\_iter(int n)  {  int result = 0;  for (int i = 1; i <= n; i++) {  result = result + i;  }  return result;  } |

# 9장

01\_

(1) 루트 노드 A

(2) 단말 노드 H, I, J, K, L, M

(3) 내부 노드 A, B, C, D, E, F, G

(4) 전위 순회 과정 A 🡪 B 🡪 D 🡪 H 🡪 I 🡪 E 🡪 J 🡪 K 🡪 C 🡪 F 🡪 G 🡪 L 🡪 M

(5) 중위 순회 과정 H 🡪 D 🡪 I 🡪 B 🡪 J 🡪 E 🡪 K 🡪 A 🡪 F 🡪 C 🡪 L 🡪 G 🡪 M

(6) 후위 순회 과정 H 🡪 I 🡪 D 🡪 J 🡪 K 🡪 E 🡪 B 🡪 F 🡪 L 🡪 M 🡪 G 🡪 C 🡪 A

02\_

(1) 루트 노드 A

(2) 단말 노드 F, G, H, K, J

(3) 내부 노드 A, B, C, D, E, I

(4) F의 후손 노드 없음

B의 후손 노드 D, F, G

(5) I의 선조 노드 E, C, A

(6) H의 형제 노드 I, J

(7) 노드 F의 차수 0

(8) 노드 G의 높이 1

노드 A의 높이 5

(9) 노드 A의 레벨 1

노드 G의 레벨 4

03\_

(1) 전위 순회 과정

+ \* A - \* B C D \* E + F G

(2) 중위 순회 과정

A \* B \* C - D + E \* F + G

(3) 후위 순회 과정

A B C \* D - \* E F G + \* +

04\_

노드 개수 n = 40개

(1) 간선의 개수: 39개

(2) 최대 높이( h ) = 40 ( h ≤ n )

(3) 최소 높이( h ) = 6 ( n ≤ 2h – 1 )

25 – 1 = 31 < 40

26 – 1 = 63 > 40

05\_

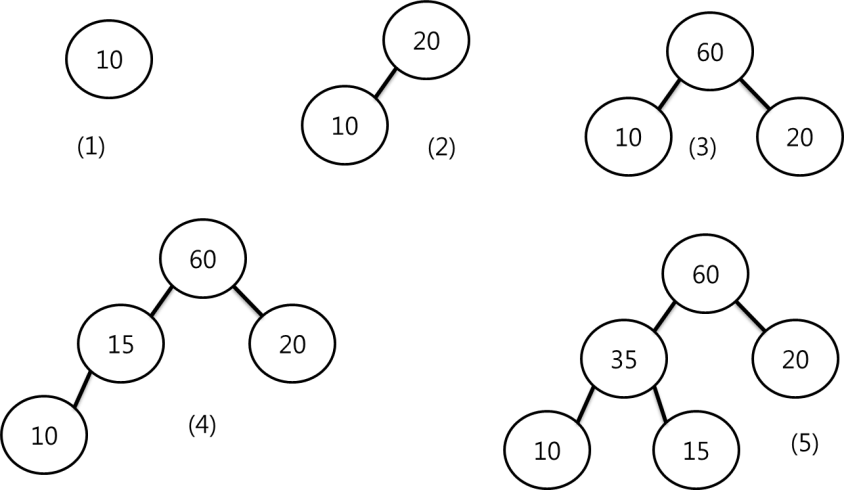
전체 소스는 프리렉 홈페이지에서 다운로드 받을 수 있습니다

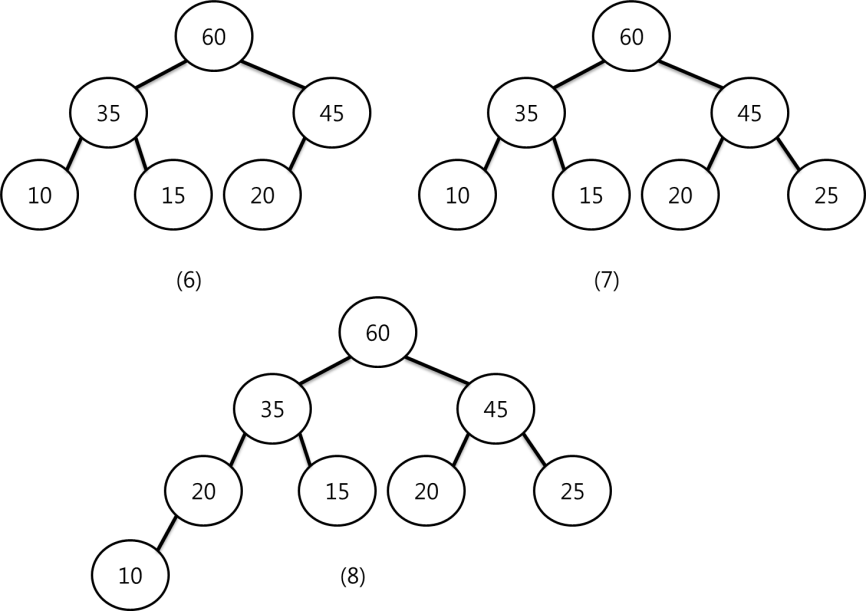
|  |
| --- |
| if (pCurrentNode != NULL) {  printf("%c ", pCurrentNode->data);  }  if (pCurrentNode->pLeftChild != NULL) {  enqueueLQ(pQueue, pCurrentNode->pLeftChild);  }  if (pCurrentNode->pRightChild != NULL) {  enqueueLQ(pQueue, pCurrentNode->pRightChild);  } |

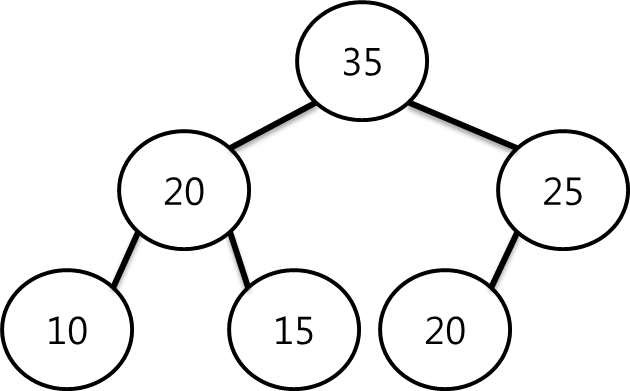
# 9장

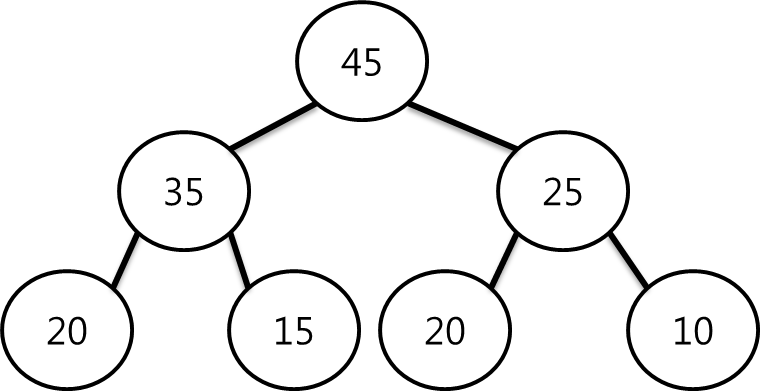
01\_

(1)





(2)



02\_

|  |
| --- |
| insertMaxHeap ( heap, data ) {  // Step-1) 트리의 마지막 자리에 임시 저장  heap->currentElementCount ← heap->currentElementCount + 1  i ← heap.currentElementCount  // Step-2) 부모 노드와 키 값 비교 및 이동  while ((i != 1) && (data.key > heap->pElement[i/2].key) ) {  heap->pElement[ i ] ← heap->pElement[i/2]  i ← i /2  }  heap->pElement[ i ] ← data  } |

|  |
| --- |
| deleteMaxHeap ( heap ) {  // Step-1: 루트 노드의 삭제  result ← heap->pElement[ 1 ]  //Step-2) 트리 마지막 자리 노드의 임시 이동  i ← heap->currentElementCount  temp ← heap->pElement[ i ]  heap->currentElementCount ← heap->currentElementCount - 1  //Step-3) 자식 노드와 키 값 비교 및 이동  parent ← 1  child ← 2  while(child <= heap->currentElementCount) {  if ( (child < heap->currentElementCount)  && heap->pElement[ child ].key < heap->pElement[ child+1 ].key) {  child ← child+1;  }  if (temp->key >= heap->pElement[ child ].key) {  break;  }  heap->pElement[ parent ] ← heap->pElement[ child ]  parent ← child  child ← child \* 2  }  heap->pElement[ parent ] ← temp  return result  } |

03\_

|  |
| --- |
| ArrayMaxHeap\* createArrayMaxHeap(int maxCount)  {  ArrayMaxHeap \*pReturn = NULL;  if (maxCount > 0) {  pReturn = (ArrayMaxHeap \*)malloc(sizeof(ArrayMaxHeap));  if (pReturn != NULL) {  pReturn->maxCount = maxCount;  pReturn->currentCount = 0;  pReturn->pData = (HeapNode \*)malloc(sizeof(HeapNode) \* (maxCount + 1));  if (pReturn->pData != NULL) {  memset(pReturn->pData, 0, sizeof(HeapNode) \* (maxCount + 1));  }  else {  printf("오류, 2번째 메모리 할당, createArrayMaxHeap()\n");  free(pReturn);  return NULL;  }  }  else {  printf("오류, 메모리 할당, createArrayMaxHeap()\n");  }  }  else {  printf("최대 원소 개수는 0보다 커야 합니다\n");  }  return pReturn;  } |

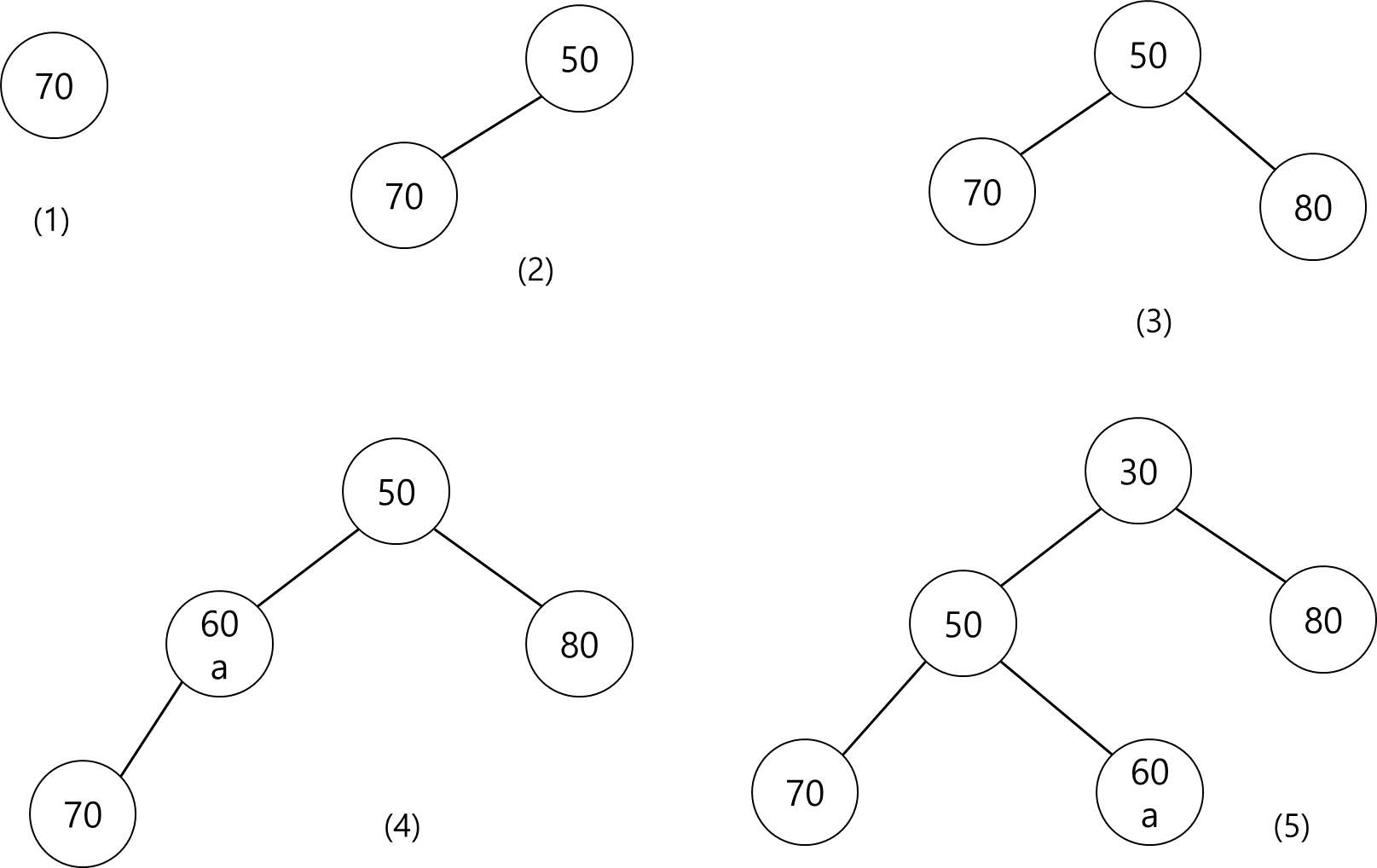
04\_

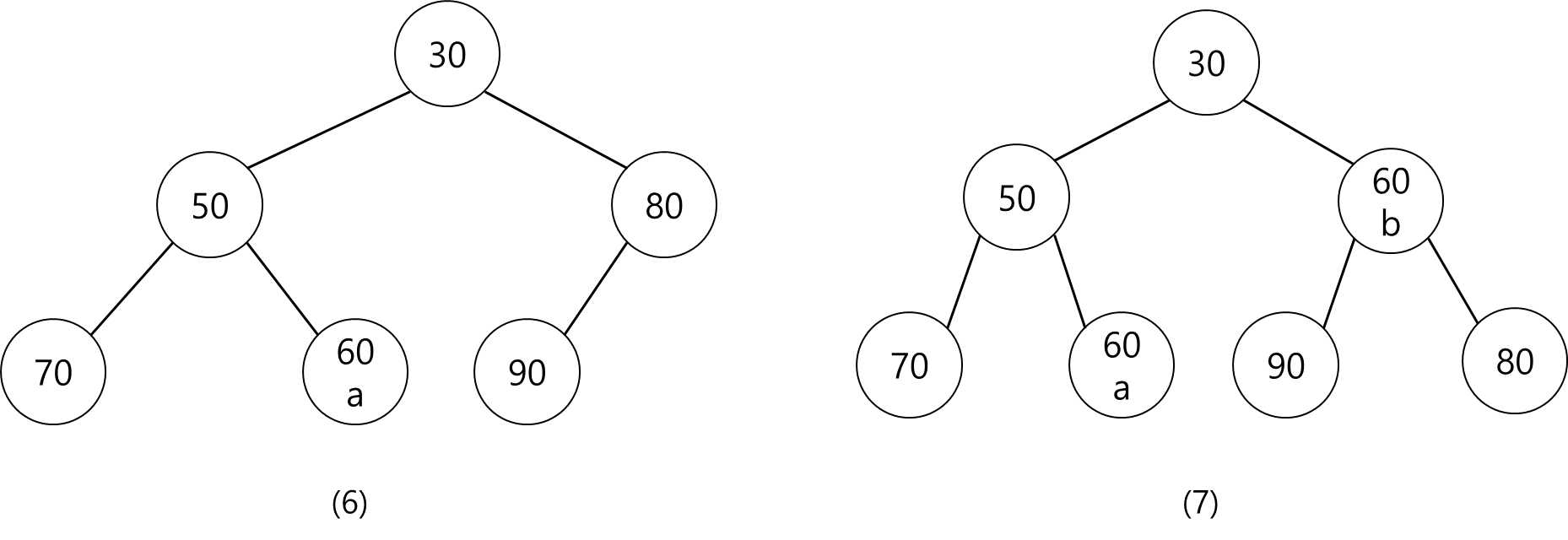
: 별도 소스를 참고해 주세요 ( 프리렉 홈페이지에서 다운로드 받을 수 있습니다 )

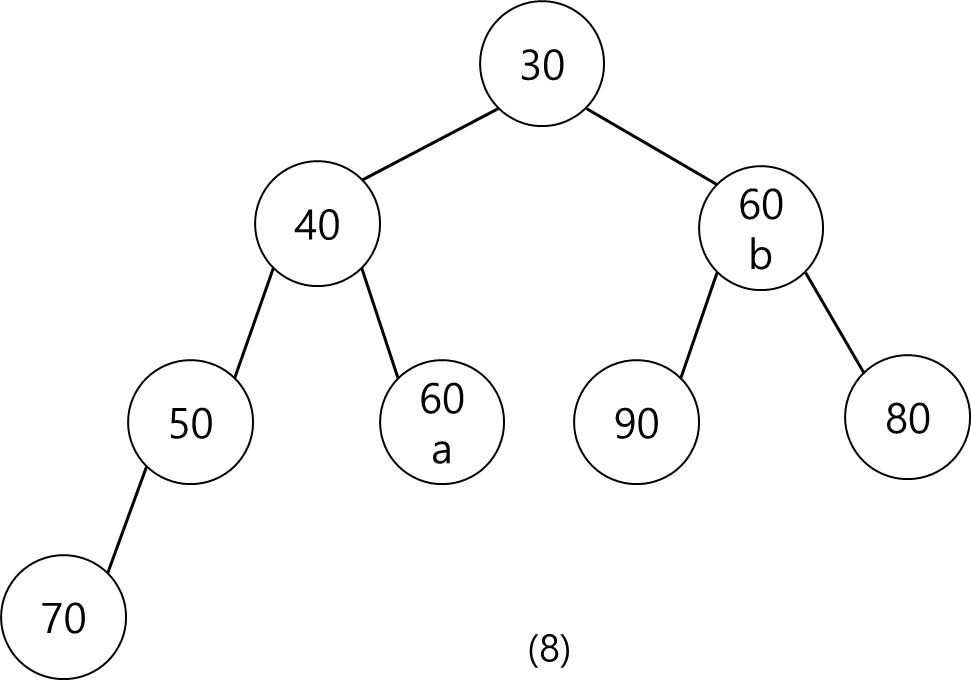
05\_

70 50 80 60(a) 30 90 60(b) 40

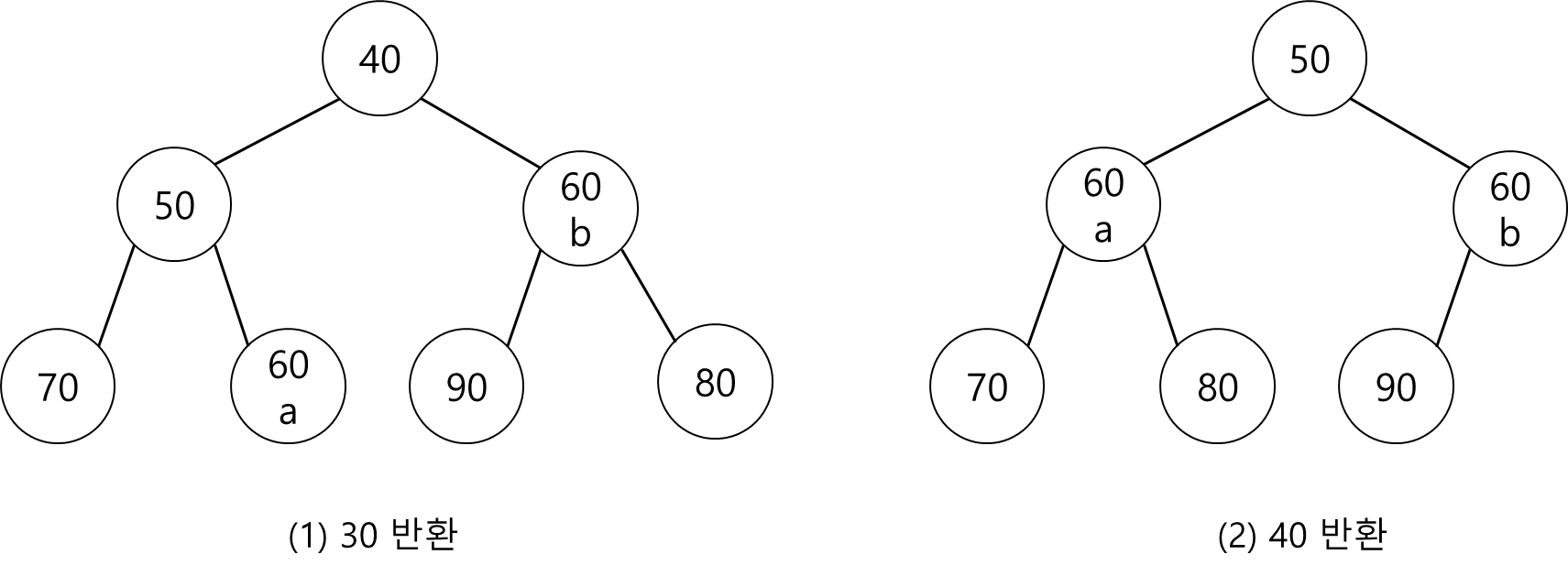
(1) 삽입 단계

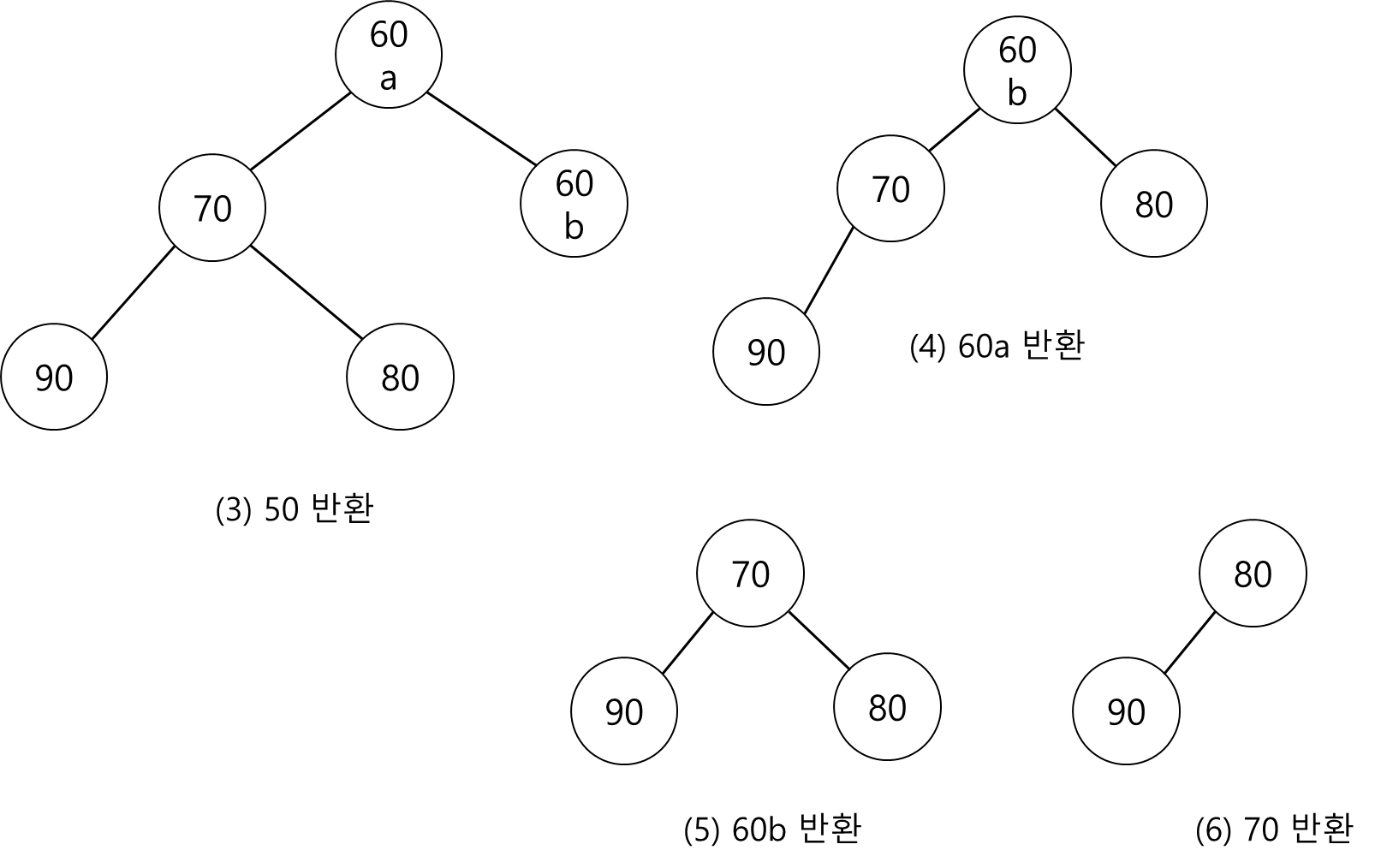






(2) 삭제 단계



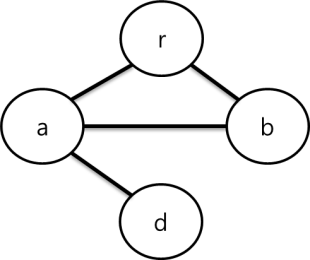




* 최종 정렬 결과: 30 🡪 40 🡪 50 🡪 60(a) 🡪 60(b) 🡪 70 🡪 80 🡪 90

# 11장

01\_



02\_

G = (V , E)

V(G) = { 0, 1, 2, 3, 4, 5, 6, 7 }

E(G) = {<0,3>, <1,0>, <7,1>, <6,1>, <6,4>, <6,7>, <2,0>, <2,3>, <2,6>, <2,4>,

<4,3>, <5,7>, <5,4>}

03\_

1) 각 노드(정점)에서의 인접 노드들

|  |  |
| --- | --- |
| 노드(정점) | 인접 노드들 |
| 0 | 1, 2, 3 |
| 1 | 0, 6, 7 |
| 2 | 0, 3, 4, 6 |
| 3 | 0, 2, 4 |
| 4 | 2, 3, 5, 6 |
| 5 | 4, 7 |
| 6 | 1, 2, 4, 7 |
| 7 | 1, 5, 6 |

2) 각 노드에서의 진입차수/진출차수

|  |  |  |
| --- | --- | --- |
| 노드(정점) | 진입 차수 | 진출 차수 |
| 0 | 2 | 1 |
| 1 | 2 | 1 |
| 2 | 0 | 4 |
| 3 | 3 | 0 |
| 4 | 3 | 1 |
| 5 | 0 | 2 |
| 6 | 1 | 3 |
| 7 | 2 | 1 |

3) 인접 행렬 표현

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 |
| 1 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | **1** | 0 | 0 | **1** | **1** | 0 | **1** | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | **1** |
| 6 | 0 | **1** | 0 | 0 | **1** | 0 | 0 | **1** |
| 7 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 |

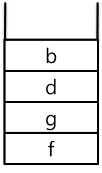
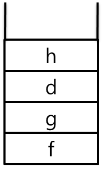
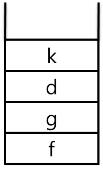
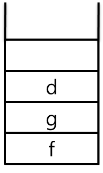
(4) 인접 리스트 표현 (인접 및 가중치 표현)

|  |  |
| --- | --- |
| 0 | → [3|1] |
| 1 | → [0|1] |
| 2 | → [0|1] → [3|1] → [4|1] → [6|1] |
| 3 |  |
| 4 | → [3|1] |
| 5 | → [4|1] → [7|1] |
| 6 | → [1|1] → [4|1] → [7|1] |
| 7 | → [1|1] |

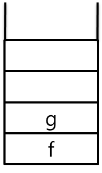
04\_

(1) 깊이 우선 탐색

a → b → h → k →

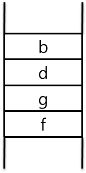
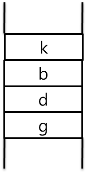
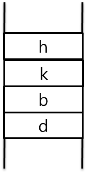
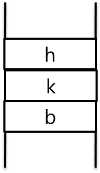
   

d → g → f

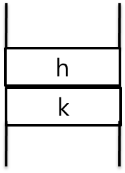
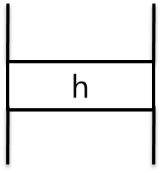
 

(2) 넓이 우선 탐색

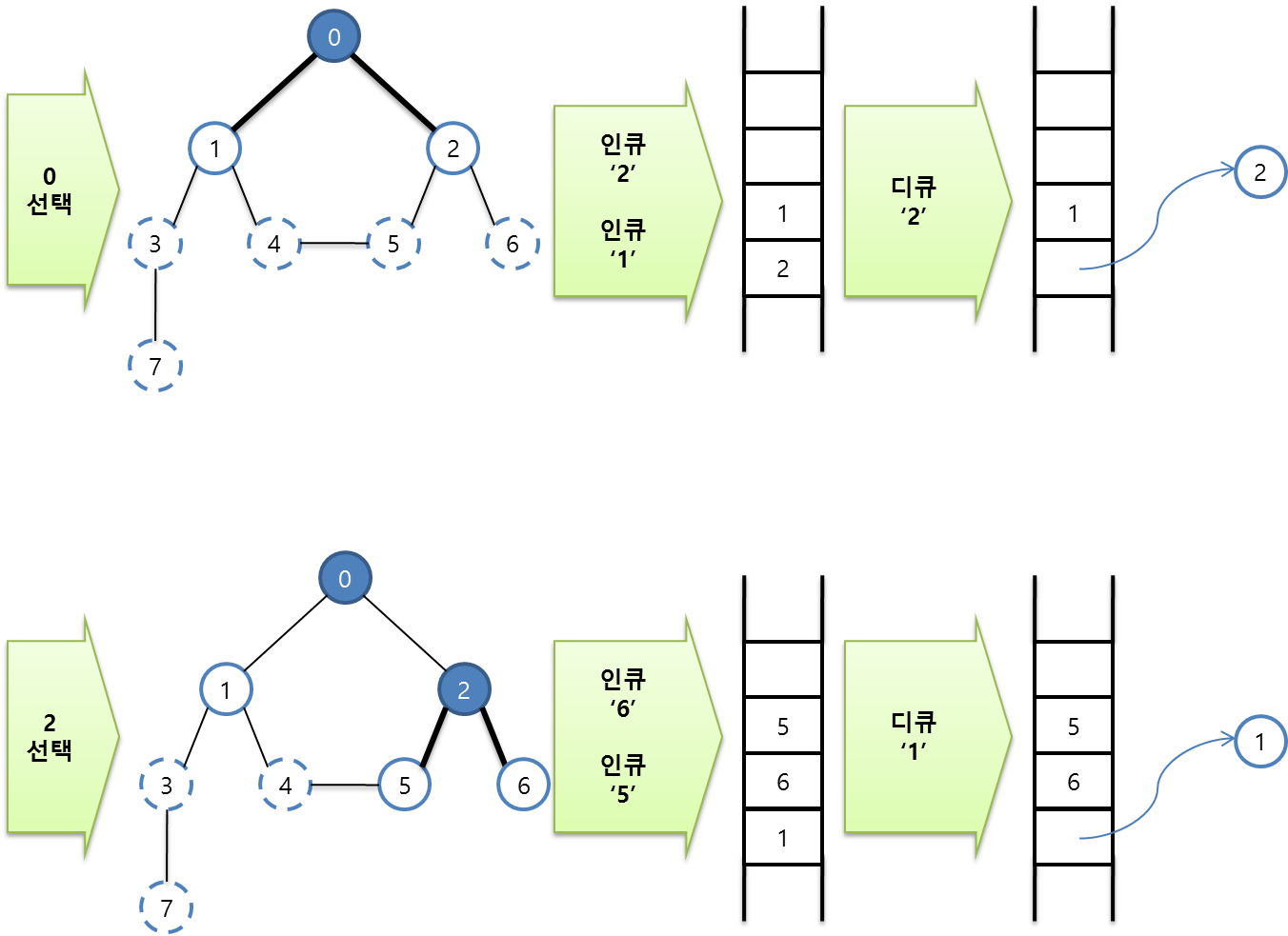
a → f → g → d →

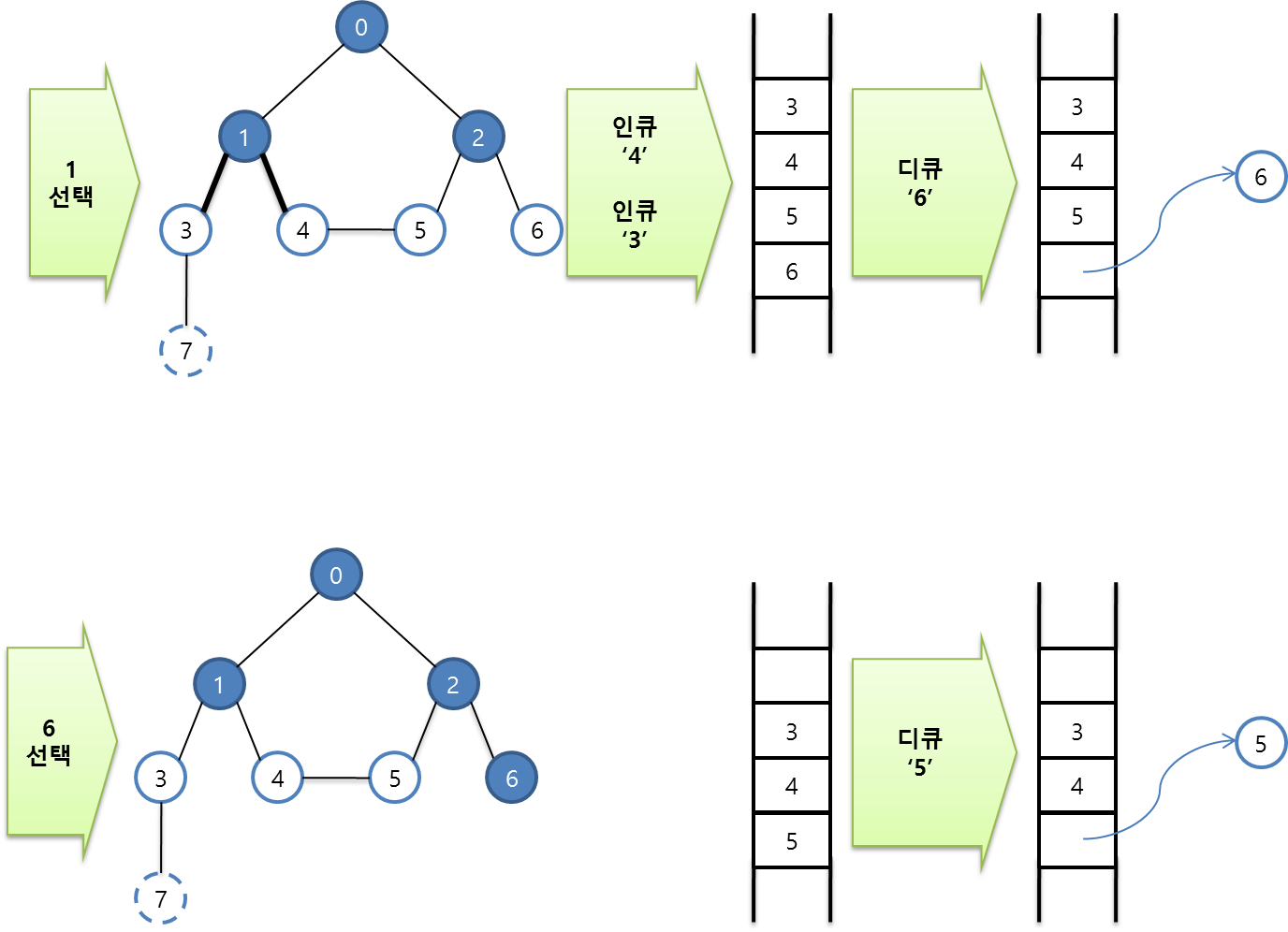
   

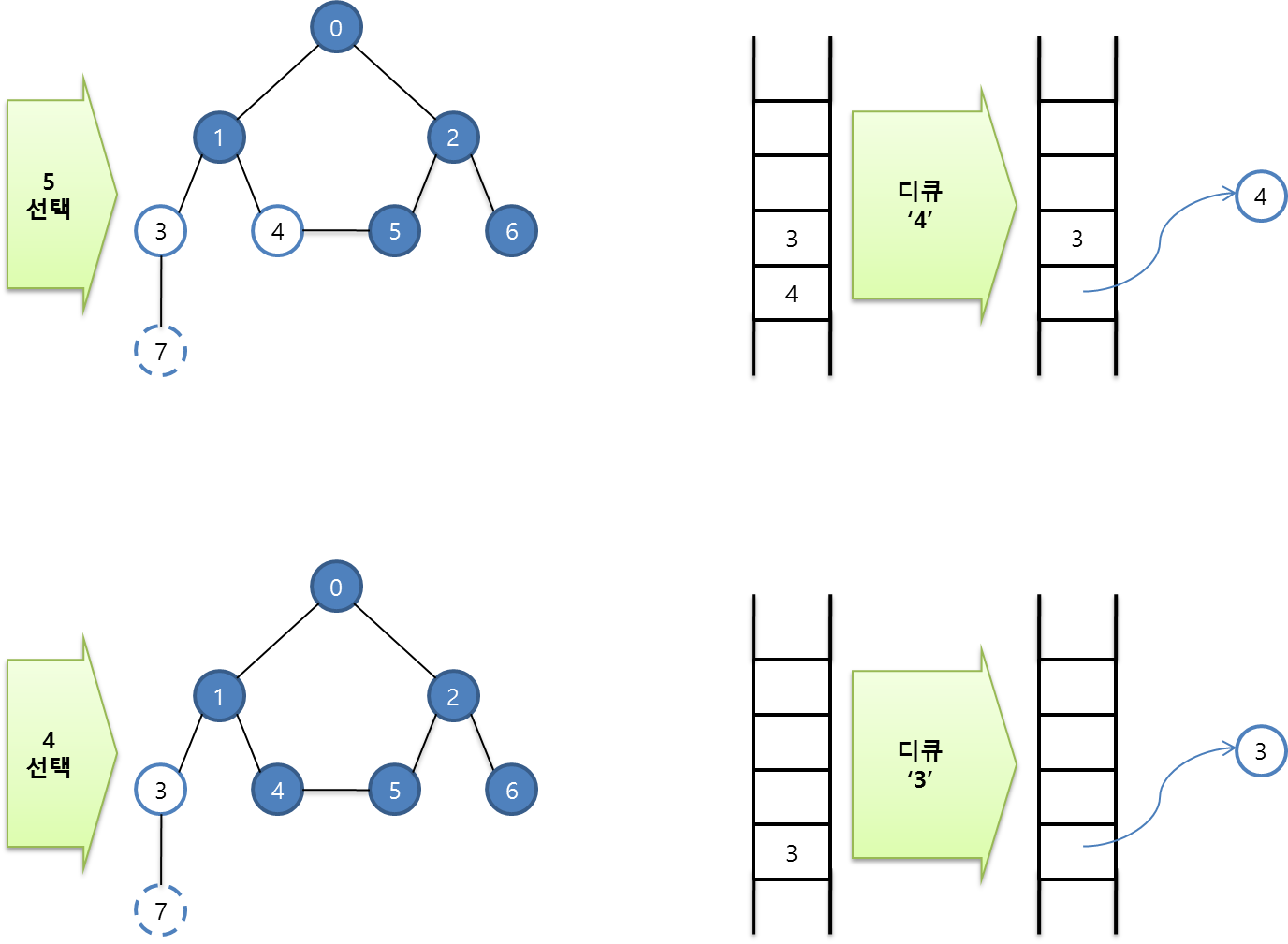
b → k → h

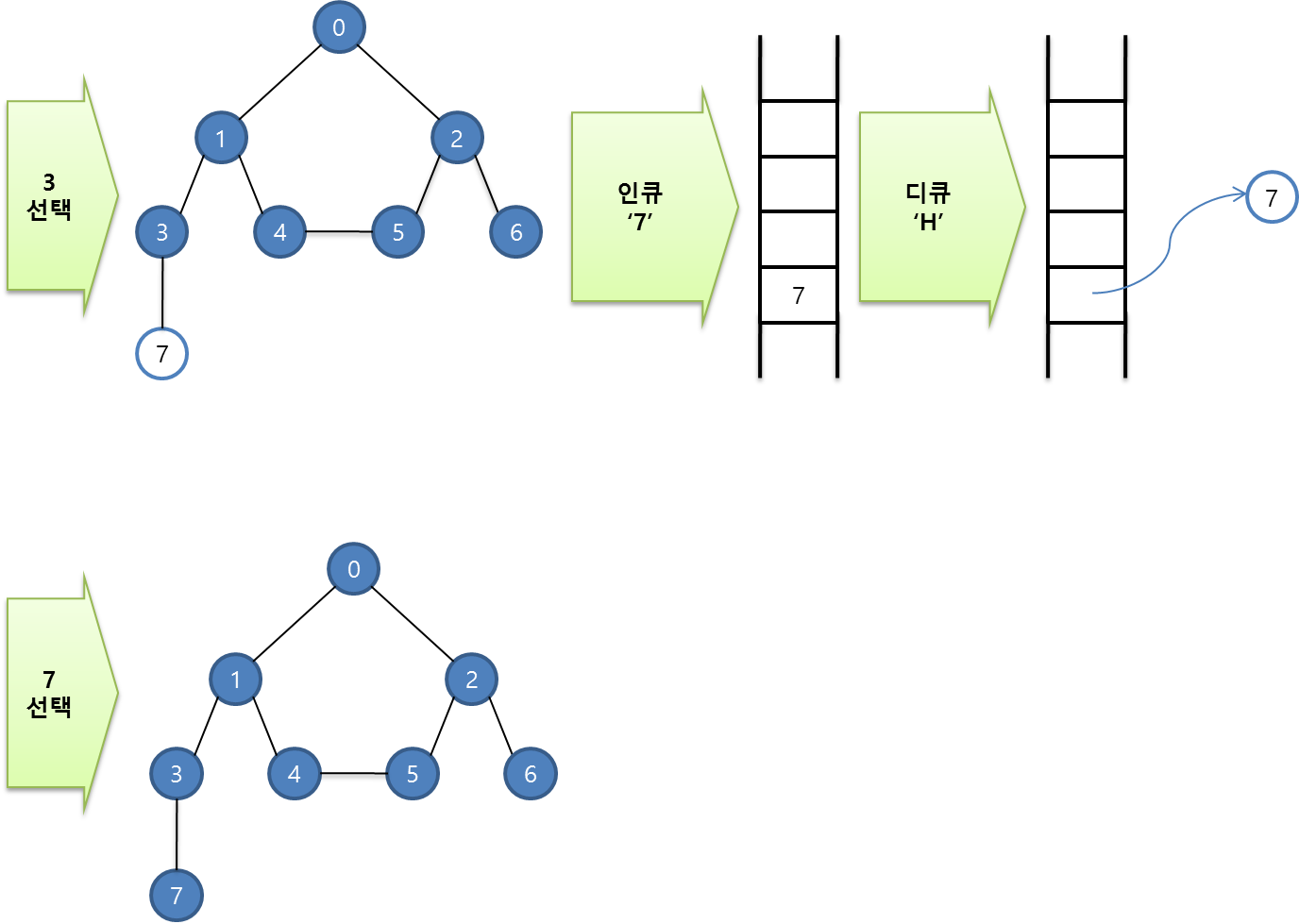
 

05\_









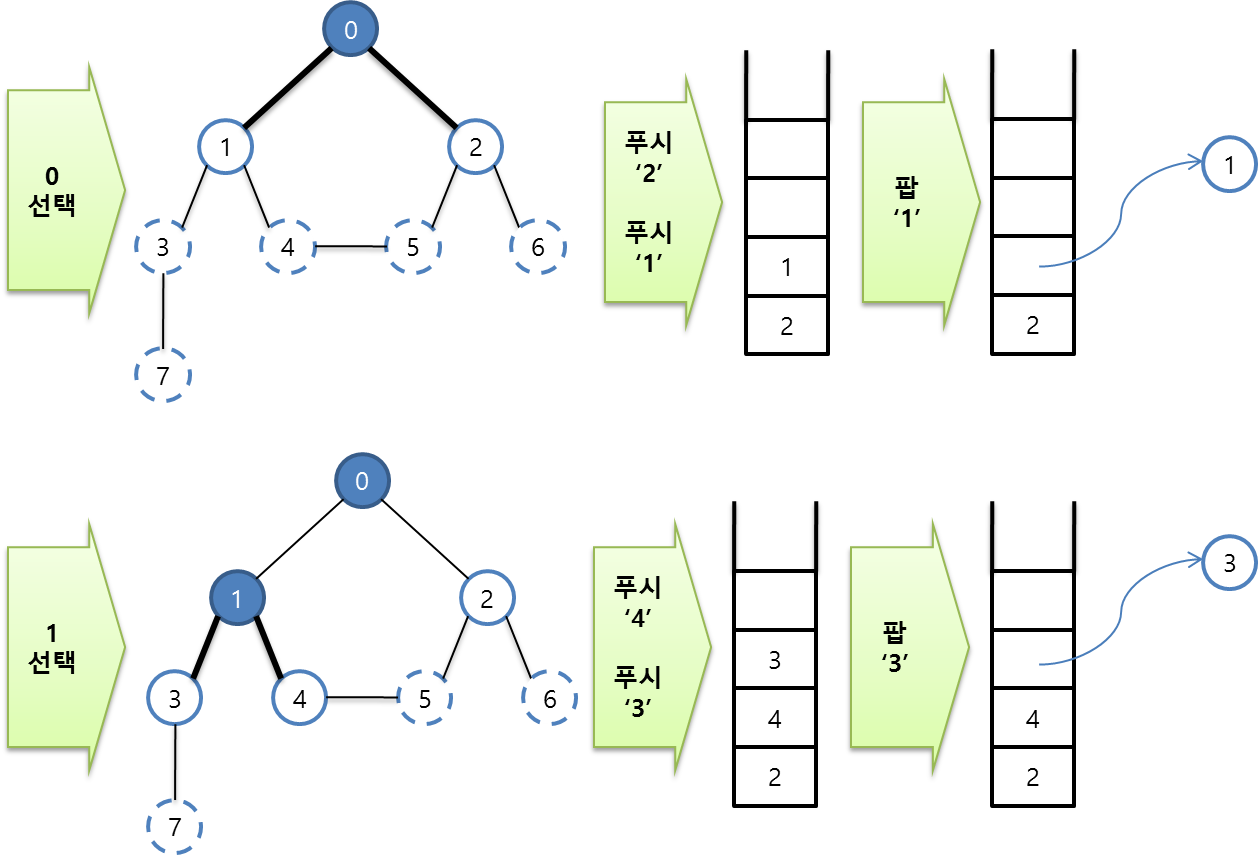
06\_

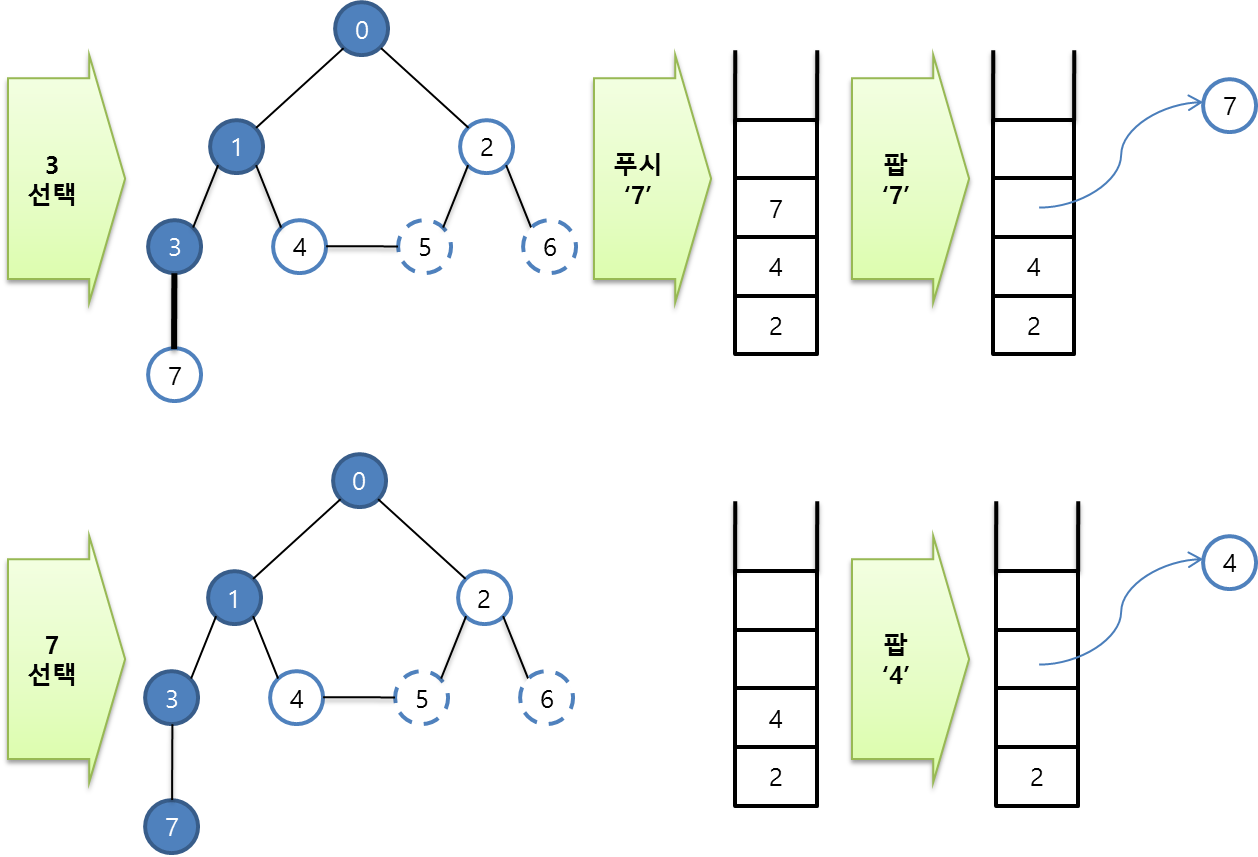
: 별도 소스를 참고해 주세요 ( 프리렉 홈페이지에서 다운로드 받을 수 있습니다 )

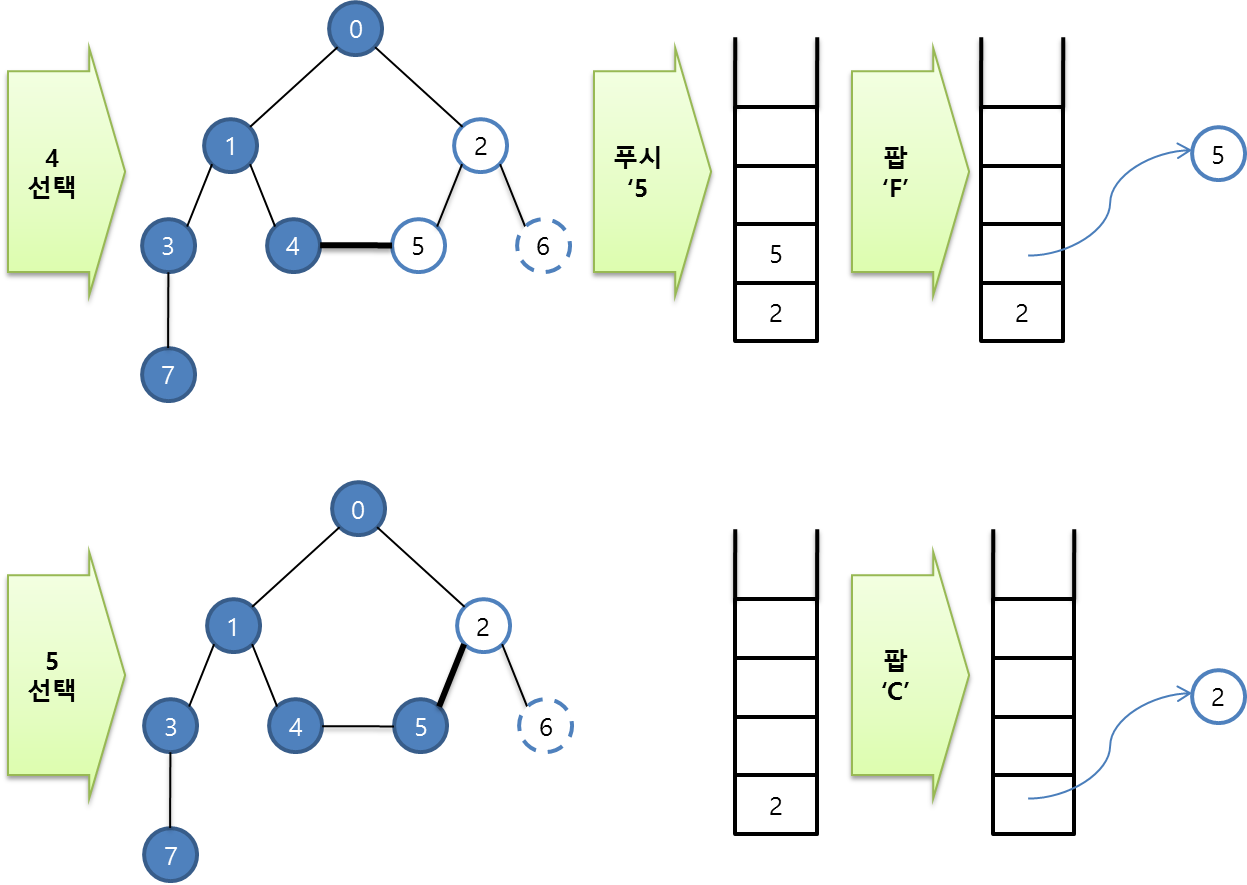
07\_

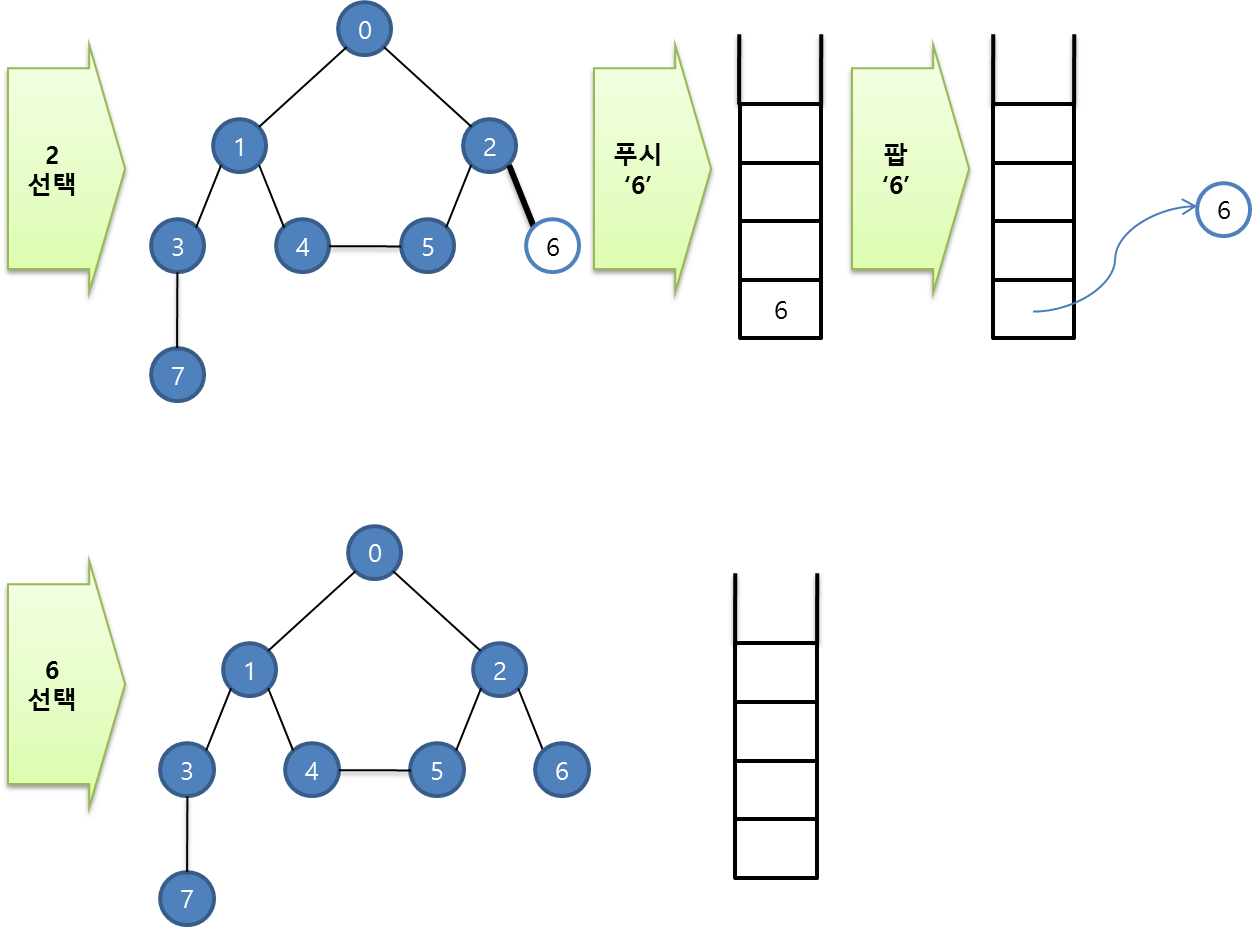
: 별도 소스를 참고해 주세요 ( 프리렉 홈페이지에서 다운로드 받을 수 있습니다 )

08\_









# 12장

01\_

4번의 함수 호출이 발생

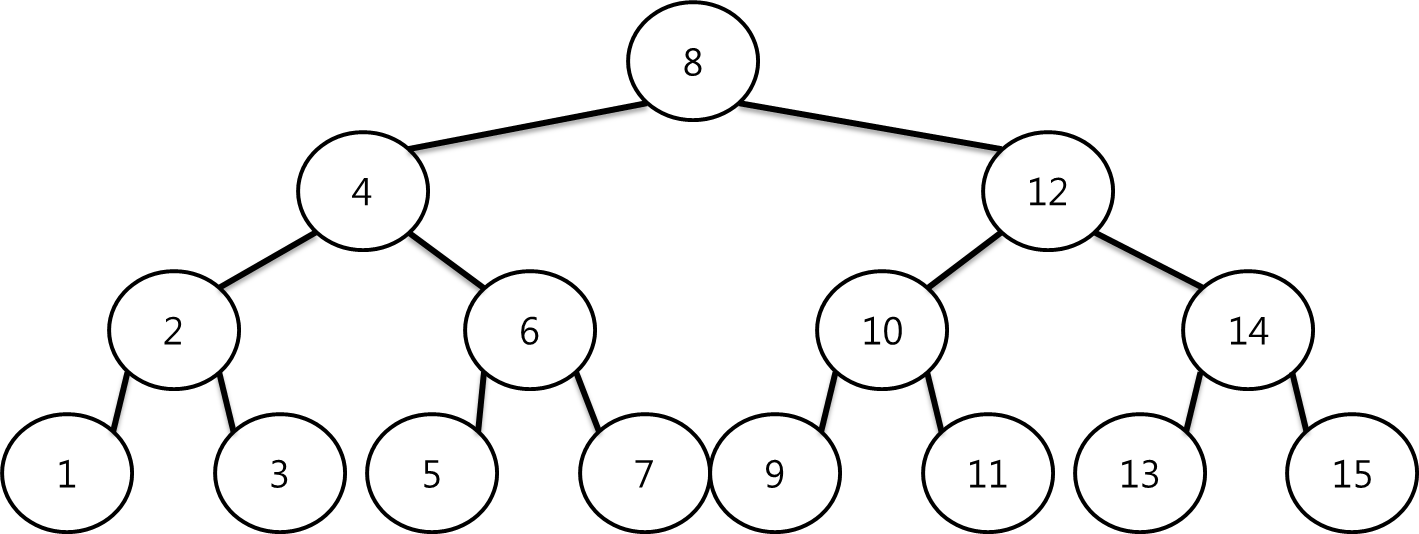
1번째, (위치 0 ~ 16)의 중간 위치 8의 값 9와 비교

* 2번째, (위치 9 ~ 16)의 중간 위치 12의 값 13과 비교
* 3번째, (위치 13 ~ 16)의 중간 위치 14의 값 15와 비교
* 4번째, (위치 13 ~ 13)의 중간 위치 13의 값 14와 비교

02\_

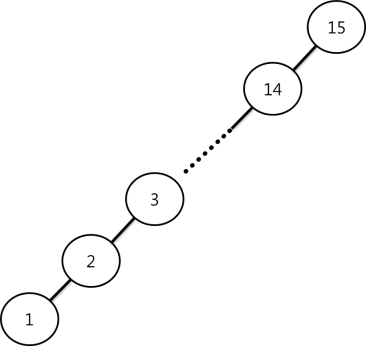
(1)

최소 높이: 4



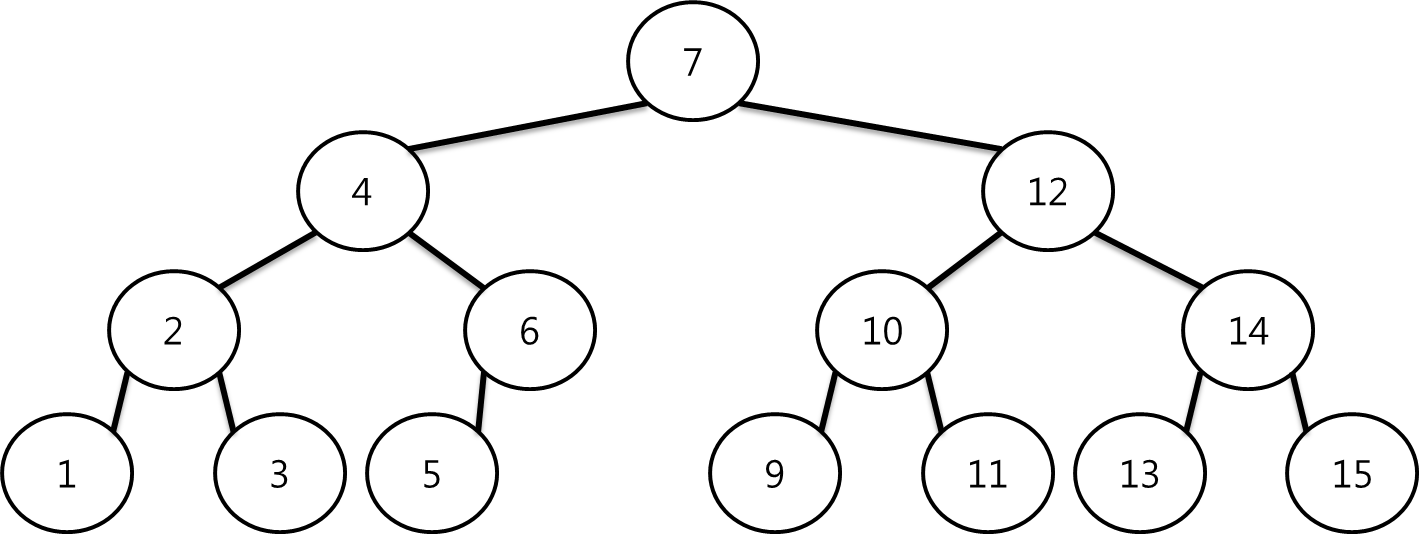
(2)

최대 높이: 15

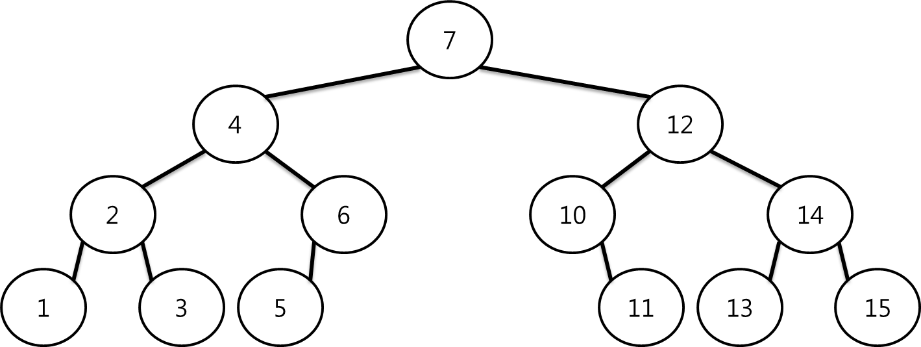


03\_

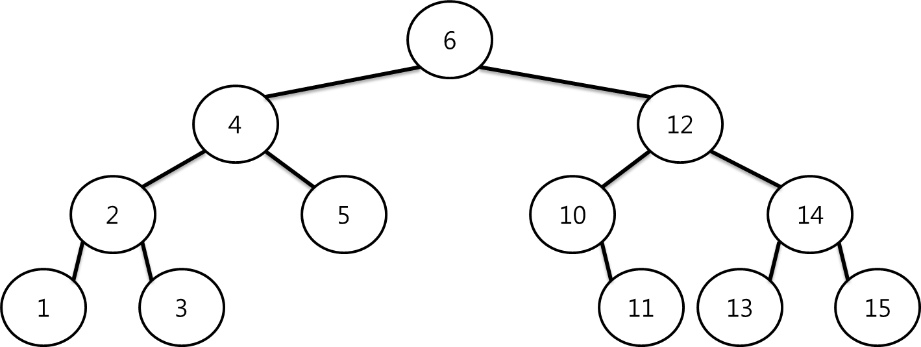
1) 8삭제 후



2) 9삭제후



3) 7삭제후



04\_

|  |
| --- |
| int binarySearch(int value[], int start, int end, int key)  {  int ret = -1;  int tempStart = 0, tempEnd = 0;  int middle = 0;  tempStart = start;  tempEnd = end;  while (tempStart <= tempEnd) {  middle = (tempStart + tempEnd) / 2;  if (key == value[middle]) {  ret = middle;  break;  }  else if (key < value[middle]) {  tempEnd = middle - 1;  }  else {  tempStart = middle + 1;  }  }  return ret;  } |