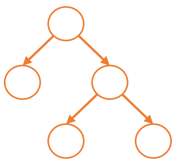
알고리즘 문제풀이

이진 트리의 순회

- 재귀를 사용한 순회



```
find(n)
{
    Visit(n);
    if Left
        find(Left);
    if Right
        find(Right);
}
```

```
find(n)
{
    if Left
        find(Left);
    Visit(n);
    if Right
        find(Right);
}
```

```
find(n)
{
    if Left
        find(Left);
    if Right
        find(Right);
    Visit(n);
}
```

```
find(n)
{
    if(n)
        Visit(n);
        find(Left);
        find(Right);
}
```

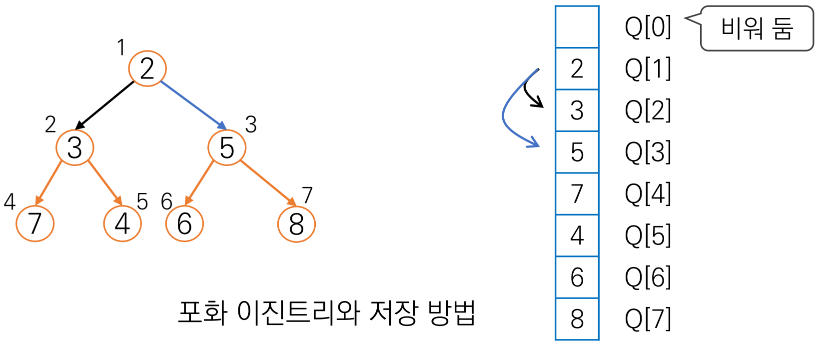
```
find(n)
{
    if(n)
        find(Left);
        Visit(n);
        find(Right);
}
```

```
find(n)
{
    if(n)
        find(Left);
        find(Right);
         Visit(n);
}
```

포화/완전 이진트리

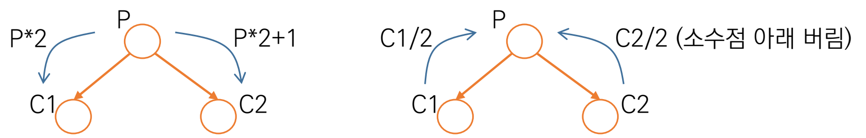
▪ 포화 이진트리

- 특정 높이까지 꽉 차 있는 이진 트리.
- 노드 번호는 위->아래, 왼쪽->오른쪽 순서.
- 노드 번호를 배열의 인덱스로 사용해 저장.



포화 이진트리와 저장 방법

▪ 부모-자식 노드 번호 계산



부모->자식

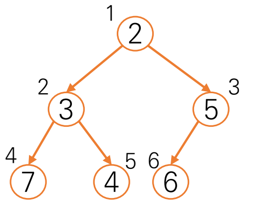
1->2, 1->3

자식->부모

2->1, 3->1

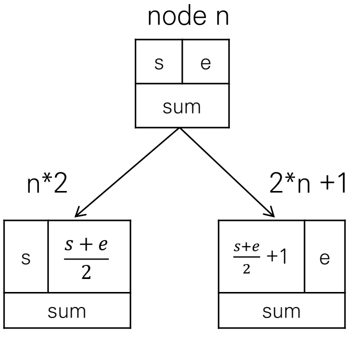
▪ 완전 이진트리

- 오른쪽 끝부터 노드가 비어있는 이진 트리.
- 포화 이진트리와 같은 방법으로 저장.
- 마지막 노드 번호를 별도로 관리.



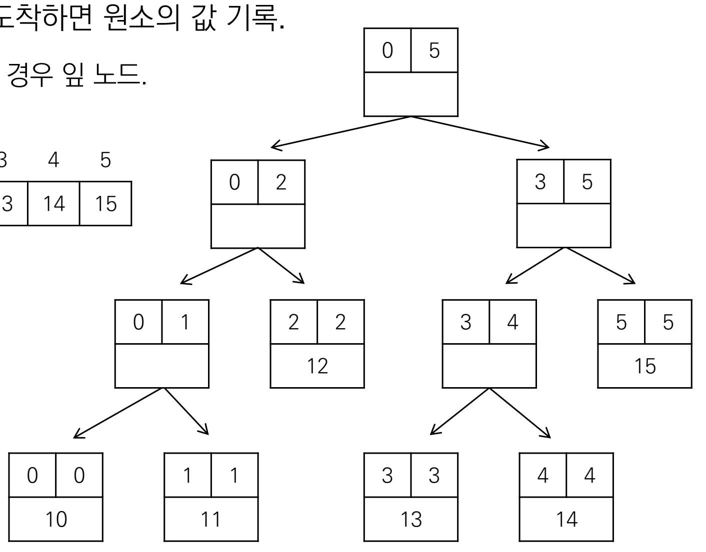
세그먼트 트리

- n개의 정수에 대한 일정 구간의 합을 포화이진 트리에 저장.
  - 각 노드에는 구간의 시작과 끝 인덱스와 구간의 합을 저장한다.
  - 루트 노드에 전 구간 인덱스와 합이 저장된다.
  - 자식 노드에는 부모가 가진 구간의 절반씩을 저장.
    - 부모 노드의 구간이 s, e인 경우
    - 왼쪽 자식 노드의 구간 s, (s+e)/2.
    - 오른쪽 자식 노드의 구간 (s+e)/2+1, e.
  - s == e인 경우 잎 노드.
    - 구간의 길이가 1.



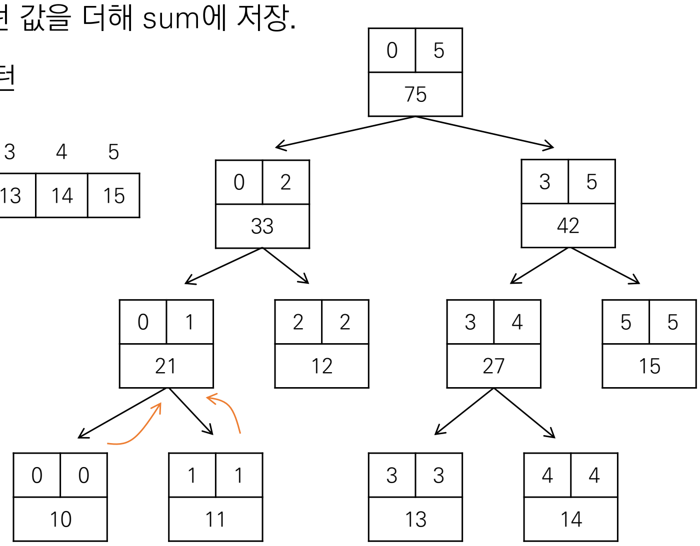
- 전위 순회로 구간을 나눔.
  - 잎 노드에 도착하면 원소의 값 기록.
    - s == e인 경우 잎 노드.

0	1	2	3	4	5
10	11	12	13	14	15



- 후위 순회로 자식 노드의 구간 합을 계산.
  - 양쪽의 리턴 값을 더해 sum에 저장.
  - sum을 리턴

0	1	2	3	4	5
10	11	12	13	14	15



▪L-R 구간의 합을 구하기 위한 탐색 과정.

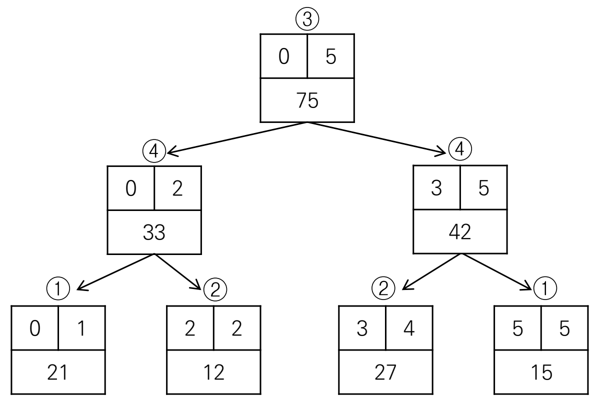
① if (R〈s || e〈L) return 0 : 범위 밖.

② if (L〈=s && e〈=R) return sum : 유효한 구간

③ if (s〈=L && R〈=e) 계속 탐색 후 리턴 값끼리 더함.

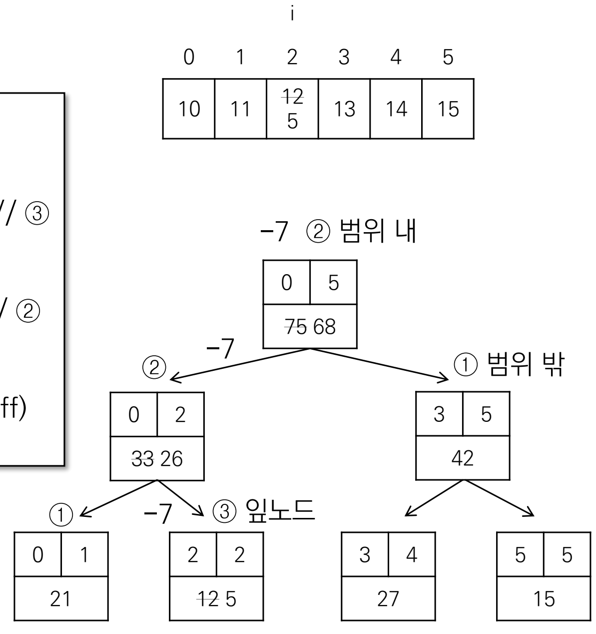
④ 일부 구간 겹침 : 계속 탐색 후 리턴 값끼리 더함.

		L		R	
0	1	2	3	4	5
10	11	12	13	14	15



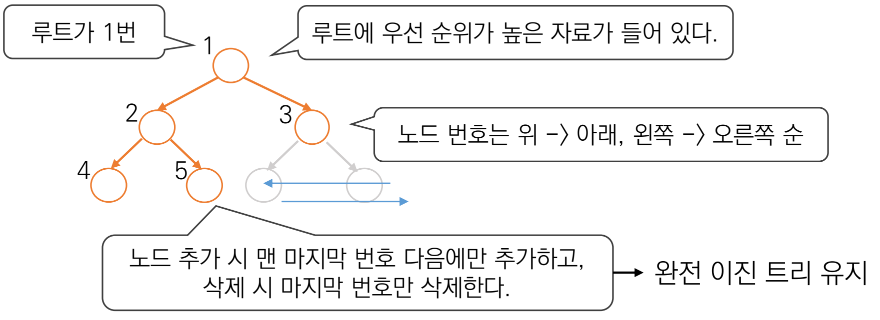
- i값을 m으로 변경.
  - diff = m − p[i]

```
change( n, i, diff)
    if (i<s || e<i) // ①
        return
    else if (s==i && i==e) // ③
        sum += diff
        return
    else if (s<=i && i<=e) // ②
        sum += diff
        change(n*2, i, diff)
        change(n*2+1, i, diff)
        return
```



이진 힙 (binary heap)

- 완전 이진트리 유지.
- 루트에 최소값이 저장되는 최소 힙.
- 루트에 최대값이 저장되는 최대 힙.

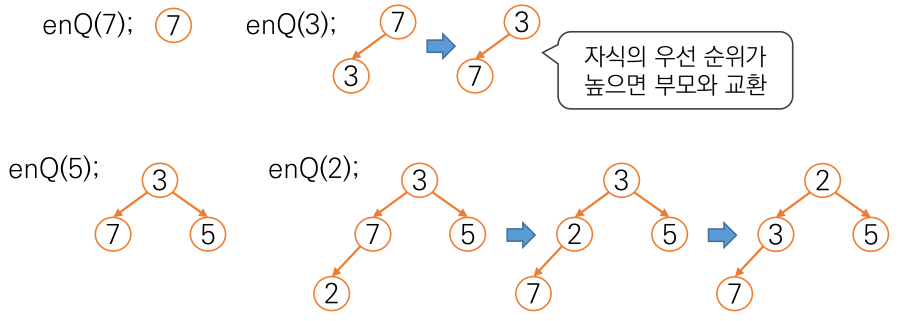


■ 최소 힙 생성

- 작은 값이 우선 순위가 높음. 루트에 가장 작은 값 저장.
- 이진 힙은 우선 순위 큐의 구현에 사용.

조건 : 완전 이진 트리 유지. 부모 〈 자식

{7, 3, 5, 2, 4, 6}

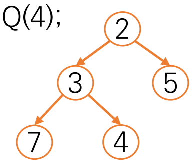


▪ 최소 힙 생성(계속)

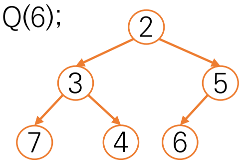
조건 : 완전 이진 트리 유지. 부모 < 자식

{7, 2, 5, 3, 4, 6}

enQ(4);



enQ(6);

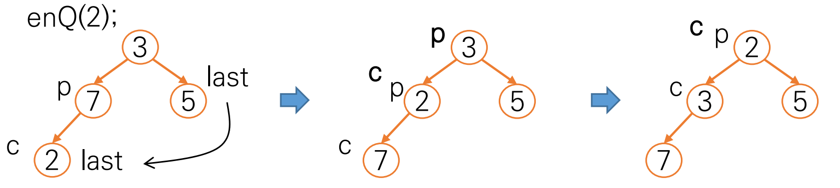


▪ 최소 힙 우선순위 큐의 연산

- enQ( )

조건 : 완전 이진 트리 유지. 부모 〈 자식

enQ(2);

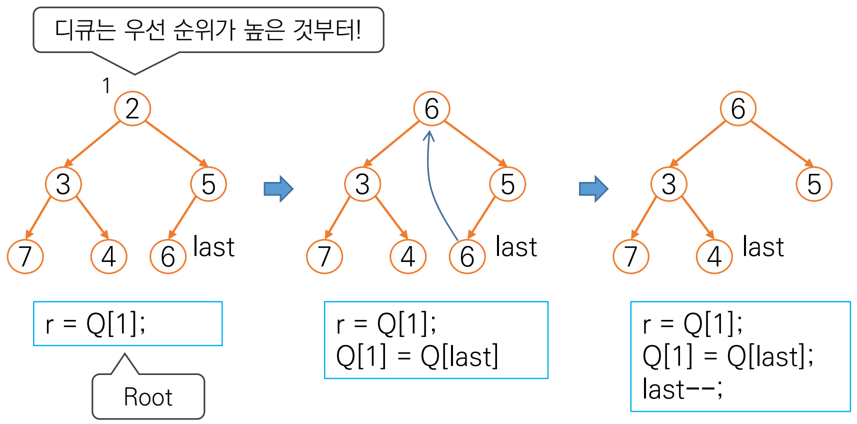



```
c = ++last;
p = c / 2;
Q[c] = data;
```

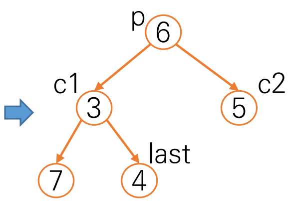
```
while(c 〉 1 && Q[p] 〉 Q[c])
{
      swap(Q[p], Q[c]);
      c= p;
      p = p/2;
}
```

- deQ( )

조건 : 완전 이진 트리 유지. 부모 < 자식

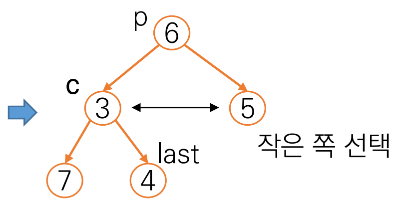


• deQ( )계속




```
p = 1;
while(p<last)
{
      c1 = p * 2;
      c2 = p * 2 + 1;
```

자식 노드 번호 계산



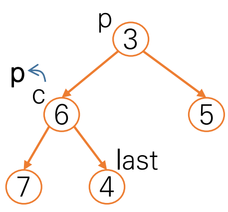

```
if(c2<=last)
{
      c = Q[c1]<Q[c2]?c1:c2;
      if( Q[c] < Q[p] )
            swap(Q[p], Q[c]);
```

양쪽 자식이 있는 경우

작은 쪽과 자리바꿈

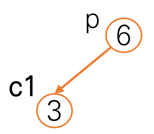
우선순위 큐

- deQ( )계속




```
if(c2<=Last)
    c = Q[c1]<Q[c2]?c1:c2;
    if( Q[c] < Q[p] )
        swap(Q[p], Q[c]);
        p = c;
    else
        break;
```

바꾼 쪽을 새로운 부모로

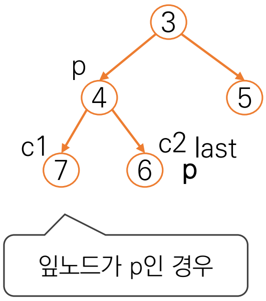



```
while(p<last)
    c1 = p * 2;
    c2 = p * 2 + 1;
    if(c2<=last)
    { … }
    else if(c1<=last)
        if(Q[c1]<Q[p])
            swap(Q[p], Q[c1]);
            p = c1;
        else
            break;
```

왼쪽 자식만 있는 경우

우선순위 큐

- deQ( )계속




```
while(p<last)
{
      c1 = p * 2;
      c2 = p * 2 + 1;
      if(c2<=last)
      { … }
      else if(c1<=last)
      { … }
      else
            break;
}
```

연습

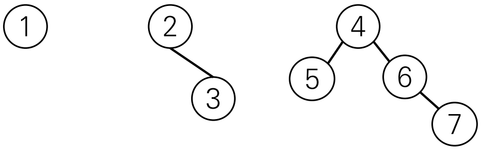
- N개의 정수가 입력된다. 앞에서 설명한 방식대로 최소힙을 구현해 저장하고, 마지막 노드의 조상 노드가 갖고 있는 숫자들의 합을 출력하라. N과 N개의 정수가 차례대로 주어진다.

```
5 5 4 3 2 1
```

```
7 2 6 10 8 5 11 7
```

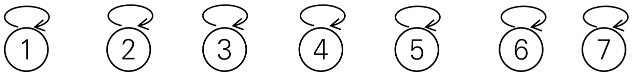
같은 트리에 속한 노드인지 확인하는 방법

- 트리의 대표 원소를 이용.
  - 간선으로 연결된 노드 중에서 대표 노드를 지정.
- 크루스칼(Kruskal) 알고리즘으로 최소비용신장트리(MST)를 찾을 때 필요한 기술.
- 서로 소 집합과 관련.
- 1, 6번 노드는 같은 트리에 속해있는가?



▪ 초기 조건 : 7개의 노드

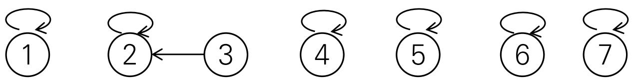
- 처음에는 자기 자신이 트리의 대표 노드.



노드	1	2	3	4	5	6	7
대표	1	2	3	4	5	6	7

- 간선 정보가 주어지면 대표 원소를 갱신.
  - 2 3
    - p[rep(3)] = rep(2)   // 3의 대표원소를 2의 대표원소로 대체

노드	1	2	3	4	5	6	7
대표	1	2	3->2	4	5	6	7



- 4 5, 4 6

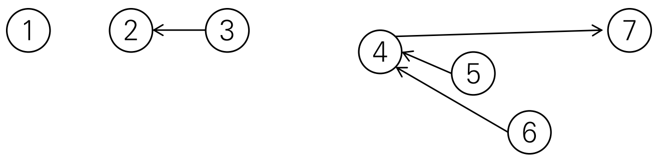
p[rep(5)] = rep(4), p[rep(6)] = rep(4)

노드	1	2	3	4	5	6	7
대표	1	2	2	4	**5->4**	**6->4**	7

- 7 6

p[rep(6)] = rep(7)

노드	1	2	3	4	5	6	7
대표	1	2	2	**4->7**	4	4	**7**



- 5와 7이 같은 트리에 속해 있는지 확인 하는 방법.
  - 5의 대표값과 7의 대표값을 비교.
  - 5의 대표값.

노드	1	2	3	4	5	6	7
대표	1	2	2	7	4	4	7

  - 7의 대표값.

노드	1	2	3	4	5	6	7
대표	1	2	2	4	4	4	7

  - 대표값이 같으므로 같은 트리에 속함.

▪ 대표값 찾기

```
rep( n )
    while( p[n] != n )
        n = p[n]
    return n
```

4	5	6	7
7	4	4	7

▪ 트리의 수

- 인덱스==대표 원소인 개수를 확인

노드	1	2	3	4	5	6	7
대표	1	2	2	7	4	4	7

연습

- 1번 부터 N번까지의 노드는 서로 다른 이진 트리에 속할 수 있다고 한다. 트리 정보가 주어지면 몇 개의 트리가 생성 되는지 출력하고, 주어진 노드가 같은 트리에 속하면 1, 아니면 0을 출력한다. 에지의 수와 부모, 자식 정보, 찾을 노드 번호가 주어진다.

입력

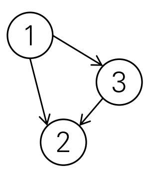
```
4
2 3 4 5 4 6  6 7
3 6
```

출력

```
3 0
```

위상 정렬

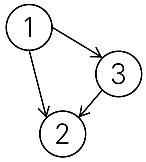
- 앞의 1, 3이 처리되어야 2번을 처리할 수 있는 경우.
  - 정점의 진입 차수를 활용.
  - 진입 차수가 0인 정점부터 시작.
  - 정점을 처리할 때 인접 정정에 처리되었음을 알림.
    - 인접 정점의 진입 차수를 하나 줄임.
    - 진입 차수가 0이 되면 다음 번에 처리할 차례가 됨.



A	1	2	3
1	0	1	1
2	0	0	0
3	0	1	0

I	0	2	1

A : 인접행렬
I : 진입 차수



A : 인접행렬
I : 진입 차수

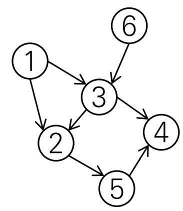
```
Sort( )
    for i : 0 -> N          // 진입차수가 0이면 enQ
        if( I[i] == 0 )
            enQ(i)
    while(is_not_emptyQ())
        n = deQ( )
        visit(n)              // 노드 n에서 해야할 일
        for i : 1 -> N
            if( A[n][i] == 1 )
                I[i]—    // n의 인접노드 진입차수 감소
                if( I[i] == 0 )  // 진입차수가 0이면 enQ
                    enQ(i)
```

Q	1								I	0	2	1
Q	~~1~~	3							I	0	2->1	1->0
Q	~~1~~	~~3~~	2						I	0	1->0	0
Q	~~1~~	~~3~~	~~2~~						I	0	0	0

연습

- 6명이 사람이 각자 갖고 있는 100원짜리 동전의 개수를 비교했더니 다음과 같은 조건이 되었다. 1번과 6번이 갖고 있는 금액이 100원 이었다면, 가장 많은 동전을 가진 사람은 최소한 몇 개의 동전을 갖고 있었는가?

1 〈 2
1 〈 3
3 〈 2
6 〈 3
3 〈 4
5 〈 4
2 〈 5



- 1~V까지 각 사람을 노드로 표시.
- 노드 별 동전 수를 저장하는 배열 coin[] 선언, 0으로 초기화.
- visit(n)

노드 n으로 진입하는 모든 노드 i에 대해, coin[i]의 최대값 + 1을 coin[n]으로 정함. 진입하는 노드가 없는 경우 동전수는 1이 됨.

$coin[n] = \max_{1 \le i \le V} coin\,[i] + 1, adj[i][n]! = 0$이 있으면

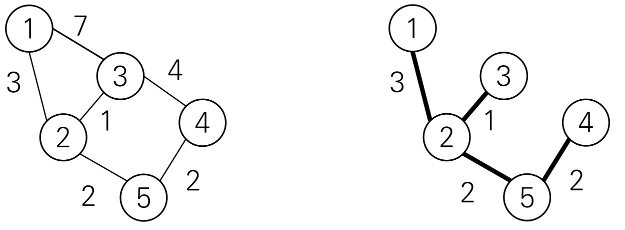
$coin[n] = 1, adj[i][n]! = 0$이 없으면

```
visit(n) :
max = 0;
for i : 1-> V
    if( adj[i][n] != 0 )
        if( max < coin[i] )
            max = coin[i]
coin[n] = max+1;
```

✓ 이 문제의 경우,
진입차수 0인 노드 i는 coin[i] = 1,
while에서는 visit(n)대신 enQ(i) 후에
coin[i] = coin[n] + 1로 처리해도 됨.

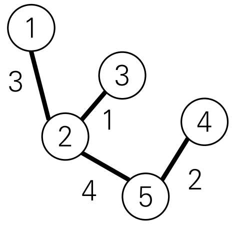
최소신장트리(MST)

- 가중치를 가진 무향그래프에서, 간선의 가중치의 합이 최소가 되도록 모든 노드를 연결한 트리.



- 한 그래프에 여러 개의 MST가 존재할 수 있다.

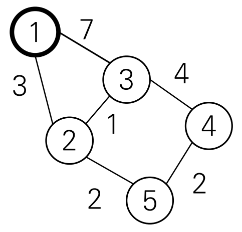
- MST에 속한 간선의 저장.
  - 자식을 인덱스로 부모를 저장하는 방식으로 저장하거나 간선의 배열 형태로 저장.



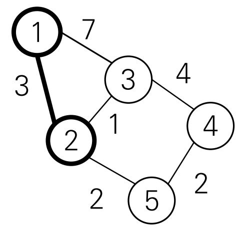
1	2	3	4	5
1	1	2	5	2

1	2
2	3
5	4
2	5

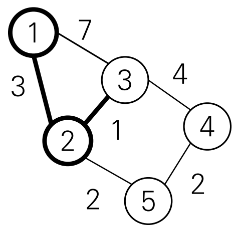
Prim 알고리즘

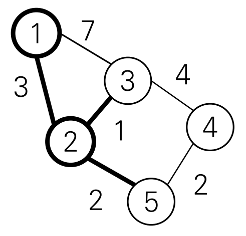


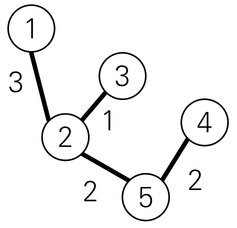
임의의 노드를 선택해
MST에 포함시킨다.



MST에 속한 노드와 인접하고, 아직
MST에 속하지 않은 노드 중 비용이
최소인 노드를 추가한다.







모든 노드가 MST에 포함되면 종료한다.

- MST에 속한 노드에 인접한 다른 노드를 찾는 과정이 중복되어 시간이 오래걸리는 단점이 있다.
  - 예) 1의 인접, 1-2의 인접, 1-2-3의 모든 인접
- 비용이 최소인 인접 노드를 찾아 노드 번호를 우선순위큐(또는 최소힙)에 넣어서 처리하면 시간이 단축된다. (Kruskal 알고리즘과 처리 시간이 비슷함.)

▪ Prim 알고리즘

비용을 표시한 인접 행렬 생성.

노드 한 개를 MST에 추가.

모든 노드가 MST에 포함 될 때까지 다음을 반복.

    MST에 포함된 모든 노드에 대해, MST에 미포함된 인접 노드와의 비용을 비교.

    최소 비용인 인접 노드를 MST에 추가.

MST에 포함된 모든 노드에 대해 조사하는 부분이 반복되어 노드가 많은 경우 시간이 오래 걸림.

▪ 우선순위 큐를 사용한 Prim 알고리즘

```
비용을 표시한 인접 행렬 생성.
노드 n1을 MST에 추가.
n1과 인접이면서 MST가 아닌 모든 노드 n2와의 에지 e(n1,n2)를 큐에 추가.
모든 노드가 MST에 포함 될 때까지 다음을 반복.
    t(n1, n2) <- 디큐 (비용이 최소인 에지)
    n2가 MST에 없으면
        n2를 MST에 추가.
        n2와 인접이면서 MST가 아닌 모드 노드 n3와의 에지 e(n2, n3)를 큐에 추가.
```

Kruskal 알고리즘

1	2	3
1	3	7
3	2	1
3	4	4
2	5	2
5	4	2

⇨

3	2	1
5	4	2
2	5	2
1	2	3
3	4	4
1	3	7

가중치에 대해 오름차순 정렬

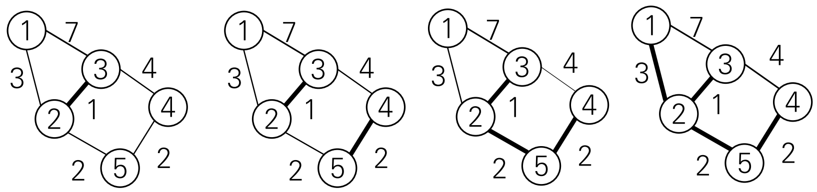
3	**2**	1
5	4	2
2	5	2
1	2	3
3	4	4
1	3	7

3	**2**	1
5	**4**	2
2	5	2
1	2	3
3	4	4
1	3	7

3	**2**	1
5	**4**	2
2	**5**	2
1	2	3
3	4	4
1	3	7

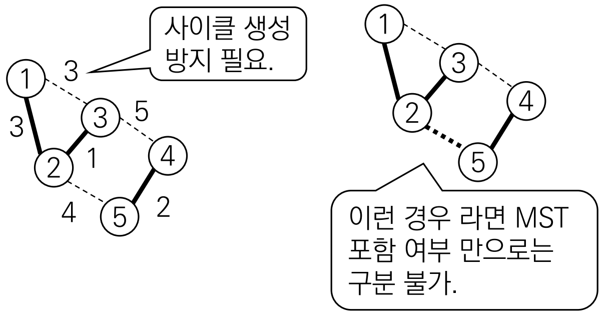
3	**2**	1
5	**4**	2
2	**5**	2
1	**2**	3
3	4	4
1	3	7

차례대로 선택



- 추가로 고려할 부분.

3 2 1
5 4 2
1 2 3
1 3 3
2 5 4
3 4 5



Kruskal의 경우 대표 원소 관리가 필요.

노드	1	2	3	4	5
대표	**1**	1	1	**4**	4

1의 대표 원소와 3의 대표원소가 같으므로 연결 불가.

2의 대표 원소와 5의 대표원소가 다르므로 연결 가능.

연습

- 1번부터 N번까지의 노드로 구성된 그래프에서 MST의 가중치의 합을 출력하시오. N과 간선의 수, 간선의 양쪽 노드 번호와 가중치가 주어진다.

입력

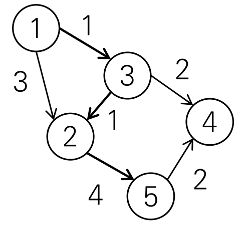
```
5 6
1 2 3
1 3 3
2 3 1
2 5 4
3 4 5
5 4 2
```

출력

```
9
```

최소 비용

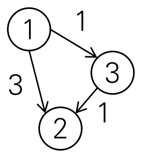
- 한 노드에서 다른 노드에 도착하는 비용 계산.
  - 모든 노드를 방문할 필요는 없는 경우.
  - 이미 비용이 계산된 노드도 다른 경로로 접근하면 비용이 감소할 수 있음.
  - 비용이 계산된 노드에 대한 중복 연산을 줄여야 함.



1에서 5로 가는 비용
3, 2를 거쳐가면 최소.
2만 거치면 오히려 비용이 증가.

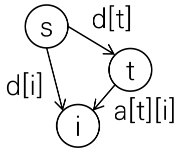
▪ 다익스트라 (Dijkstra) 알고리즘

- 조건
  - 모든 노드를 거쳐갈 필요는 없음.
  - 모든 비용이 양수.
- 출발지에서 가까운 노드부터 경유지로 선택하고, 경유지를 거쳐 갈 수 있는 노드에 대해 기존의 비용과 경유하는 비용을 비교해 새 비용을 정한다.
- 모든 노드가 경유지로 고려되면 종료한다.
- 계산이 끝나면 모든 노드에 대한 최소 비용이 계산되어 있다.
- 원하는 노드까지의 비용을 출력한다.



경유지를 고려 하기 전 1에서 2로 가는 비용은 3
경유지를 거쳐 1에서 2로 가는 비용은 2

▪ 다익스트라 (Dijkstra) 알고리즘



출발 s.
인접행렬 a[][].
경유지로 고려되었음을 표시하는 u[].
출발에서 노드까지의 비용 d[].
경유지로 고려 안된 노드가 있으면 다음을 반복.
    고려안된 노드 중 d[t]가 최소인 노드 t를 찾아 고려됨으로 표시.
    t와 인접인 노드 i에 대해,
    출발지에서 t를 거쳐 i로 가는 비용 d[t]+a[t][i]과
    현재 i까지 가는 비용 중 작은 쪽을 작은 쪽을 새로운 d[i]로 선택.

    d[i] = min(d[i], d[t] + a[t][i])

▪ 다익스트라 (Dijkstra) 알고리즘

- 출발 s = 1, 도착 g = 5

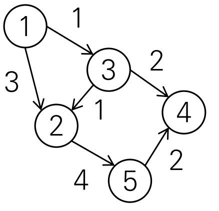
경유지로 고려 여부

초기화

	1	2	3	4	5
u	1	0	0	0	0

최소 거리

	1	2	3	4	5
d	0	3	1	∞	∞

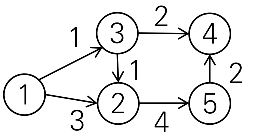



```
u[s] = 1
d[] 초기화
while( u[]에 0이 남아 있으면 )
    u[t] == 0 이고 d[t]가 최소인 노드 t를 찾는다.
    u[t] = 1
    t의 모든 인접 i에 대해
        d[i] = min(d[i], d[t] + a[t][i])
return d[g]
```

s는 경유지로 고려된 것으로 표시

i는 경유지로 고려됨

출발 s = 1, 도착 g = 5



경유지로 고려 여부 / 최소 거리

초기화

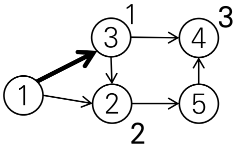
	1	2	3	4	5
u	1	0	0	0	0

	1	2	3	4	5
d	0	3	1	∞	∞

t=3

	1	2	3	4	5
u	1	0	1	0	0

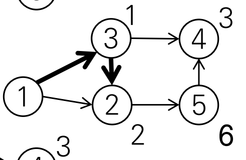
	1	2	3	4	5
d	0	**2**	1	**3**	∞



t=2

	1	2	3	4	5
u	1	1	1	0	0

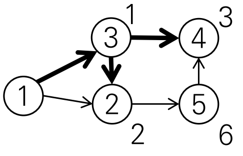
	1	2	3	4	5
d	0	2	1	3	**6**



t=4

	1	2	3	4	5
u	1	1	1	1	0

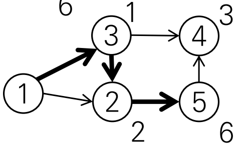
	1	2	3	4	5
d	0	2	1	3	6



t=5

	1	2	3	4	5
u	1	1	1	1	1

	1	2	3	4	5
d	0	2	1	3	6



- u[t] == 0인 모든 노드를 반복해서 검색하면 시간이 오래 걸림.
- 우선순위 큐를 사용하면 속도를 높일 수 있음.

```
u[s] = 1
s에 인접한 모든 노드 i enqueue(i).
while( 큐가 비어있지 않으면 )
    t = dequeue()
    u[t] = 1
    t의 모든 인접 i에 대해
        d[i]보다 (d[t] + a[t][i])가 작으면
            d[i] = d[t]+a[t][i]
            enqueue(i)
return d[g]
```

- 앞의 문제를 BFS를 변형해서 계산할 수 있다.
  - 다른 경로에 의해 비용이 갱신된 노드를 큐에 넣는다.
  - 갱신된 노드의 인접 노드가 갱신되는지 확인한다.
  - 더 이상 비용이 갱신되는 노드가 없으면 완료.
  - 중복이 많이 발생하지만 노드의 수가 많지 않으면 속도가 빠르다.

```
BFS(s)
    시작점의 모든 인접 노드 enqueue
    s에서 다른 노드로 가는 비용으로 d[] 초기화
    while( 큐가 비어 있지 않으면 )
        t = dequeue
        t의 모든 인접 노드 i에 대해
            t를 거쳐 i로 가는 비용이 d[i]보다 작으면
                d[i] 갱신
                enqueue(i) // 갱신된 i의 인접 노드를 검사하기 위해.
```

동적 계획법 (DP, Dynamic Programming)

- 작은 부분부터 문제를 풀어 전체 문제를 해결하는 방법.
- 점화식을 찾은 후 반복 구조로 구현.
- 빠른 속도로 해를 구할 수 있다.
- 매우 다양한 문제에 적용되므로 쉬운 것부터 많은 연습이 필요.

팩토리얼

- 점화식
  - f(n) = n * f(n-1)
  - f(1) = 1, f(0) = 1

```
f(n)
    if(n<2)
        return n
    else
        return n*f(n-1)
```

⇨

```
f[0] = 1
f[1] = 1
for i : 2 -> n
    f[i] = i * f[i-1]
```

- n!을 1,000,000,007로 나눈 나머지를 출력하는 프로그램을 작성하시오.

$$\boxed{ab \bmod n = [(a \bmod n)(b \bmod n)] \bmod n}$$

- f(n) = n * f(n-1) % 1000000007

피보나치 수 (Fibonacci number)

- f(n) = f(n−1) + f(n−2)
- f(0) = 1, f(1) = 1
  - 1, 1, 2, 3, 5, 8, 13, 21, …

✓f(0) = 0, f(1) = 1인 경우

✓0, 1, 1, 2, 3, 5, 8, 13, 21, …

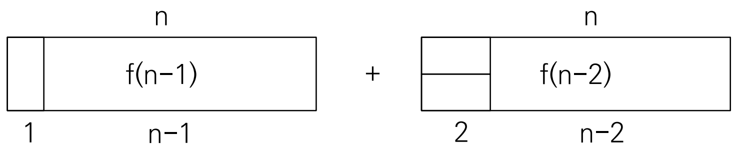
```
f(n)
    if(n<2)
        return 1
    else
        return f(n-1) + f(n-2)
```

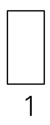
⇨

```
f[0] = 1
f[1] = 1
for i : 2 -> n
    f[i] = f[i-1] + f[i-2]
```

- 크기가 2x1인 타일을 2xn인 공간에 붙이는 경우의 수를 구하시오.

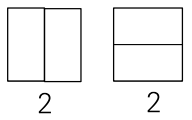
점화식을 구하기 위해 두 경우로 나눠서 생각한다.





$f(1) = 1$

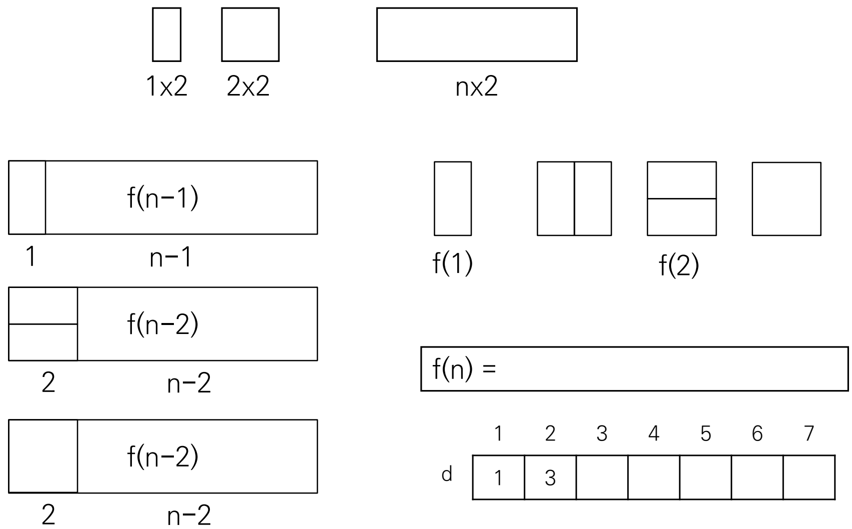
$\boxed{f(n) = \qquad\qquad\qquad\qquad}$



$f(2) = 2$

	1	2	3	4	5	6	7
d	1	2					

- 1x2, 2x2 크기의 타일을 nx2 크기의 공간에 붙이는 경우의 수를 구하시오. 1x2 타일은 가로, 세로로 모두 붙일 수 있음.



여러 개의 값 활용

- 1 2 3의 부분 집합으로 만든 중복 순열의 합이 N이 되는 경우의 수를 구하시오.

입력

4

4를 만들 수 있는 경우

1 + 1 + 1 + 1
2 + 1 + 1
1 + 2 + 1
3 + 1
1 + 1 + 2
2 + 2
1 + 3

4는 어떤 수에 1, 2, 3을 더해 만들 수 있으므로,

3 + 1 = 4
2 + 2 = 4
1 + 3 = 4

(4-1=3을 만드는 경우의 수)
+ (4-2=2를 만드는 경우의 수)
+ (4-3=1을 만드는 경우의 수)

▪ N이 10인 경우

n	1	2	3	4	5	6	7	8	9	10
d	1	2	4							

+

f(n) =

초기값

1	2	3
1	1+1 2	1+1+1 2+1 1+2 3

최장 증가 부분 수열

- 어떤 수열에서 숫자가 증가하는 순서로 숫자를 골라 만든 부분 수열 중, 숫자의 개수가 가장 많은 것을 최장 증가 부분 수열이라고 한다. 주어진 수열에서 최장 증가수열의 길이를 구하시오.

$\boxed{3\ 1\ 2\ 7\ 6\ 4\ 5} \longrightarrow \boxed{1\ 2\ 4\ 5}$

i	0	1	2	3	4	5	6
a	3	1	2	7	6	4	5
증가 수열의 길이 d	1	1	2	3	3	3	4

■ a[i]까지의 증가 수열의 길이 d[i]

- 최소값 1로 초기화 (이후 숫자가 계속 작아져도 최소한 1이 된다.)

i	0	1	2	3	4	5	6
a	3	1	2	7	6	4	5
증가 수열의 길이 d	1	1	1	1	1	1	1

0<=j<i인 j에 대해 a[j]<a[i]를 만족하는 경우 중, d[j]가 가장 큰 값을 골라 1을 더한다.

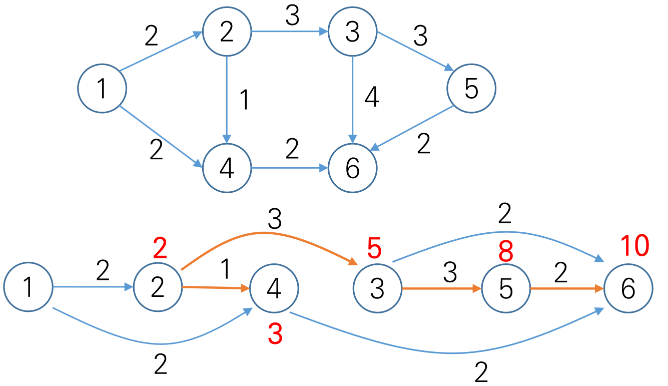
$\boxed{d[i] = \left(\max_{0 \leqq j < i} d[j]\right) + 1,\ a[j] < a[i]\text{인 경우}}$

a	3	1	2	7	6	4	5
증가 수열의 길이 d	1						

■ n개의 수에서 최장 증가 수열의 길이 : $\max_{0 \leqq i < n} d[i]$

최장거리 구하기

- 조건 : 사이클이 없는 방향성 그래프(DAG)
- 시작 노드 1, 도착 노드 6.
- 위상 정렬을 사용해 거리 계산 순서를 결정한다.

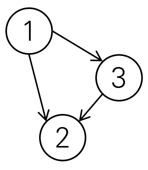


- 각 노드까지의 최대 거리 dis[], 인접행렬 adj[][].
  - n으로 진입하는 모든 노드 i에 대해, dis[i]+adj[i][n]이 최대인 값을 dis[n]으로 정함.

$$dis[n] = \max_{1 \le i \le V}(dis\,[i] + adj[i][n])\,, adj[i][n]! = 0\text{인 노드 } i\text{에 대해}$$

```
max = 0
for i : 1->V
    if( adj[i][n] != 0 )
        if( max < dis[i] + adj[i][n] )
            max = dis[i] + adj[i][n]
dis[n] = max
```

✓위상 정렬



```
Sort( )
    for i : 0 -> N          // 진입차수가 0이면 enQ
        if( ind[i] == 0 )
            enQ(i)
    while(is_not_emptyQ())
        n = deQ( )
        visit(n)             // 노드 n에서 해야할 일
        for i : 1 -> N
            if( adj[n][i] == 1 )
                ind[i]--     // n의 인접노드 진입차수 감소
                if( ind[i] == 0 )  // 진입차수가 0이면 enQ
                    enQ(i)
```

adj[] : 인접행렬
ind[] : 진입 차수

오른쪽과 아래로 이동

- 각 칸에서는 오른쪽이나 아래로만 이동할 수 있다.
- 출발은 맨 왼쪽 위, 도착은 맨 오른쪽 아래이다.
- 출발부터 도착까지 지나는 각 칸의 합계가 최대가 되도록 움직였을 때, 합계를 계산하라.
- 각 칸은 1에서 9사이의 숫자.

출발

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

도착

- 각 칸까지의 최대 합계 d.
  - 각 칸에는 위와 왼쪽에서만 진입 가능.
  - 윗쪽 칸과 왼쪽 칸 중 합계가 큰 곳 + 현재 칸의 값.

m	1	2	3	4	5	j
1	5	2	8	8	8	
2	4	4	8	5	8	
3	10	9	6	3	5	
i						

d	1	2	3	4	5	j
1						
2						
3						
i						

(i, j)칸에 진입은 위(i-1,j)나 왼쪽(i, j-1)에서만 가능.
각 칸까지의 최대 합계를 d[i][[j]라고 하면,
d[i][j] = max(d[i-1][j], d[i][j-1]) + m[i][j], i>1, j>1
d[i][j] = d[i][j-1] + m[i][j], i = 1
d[i][j] = d[i-1][j] + m[i][j], j = 1

d	0	1	2	3	4	5	j
0	0	0	0	0	0	0	
1	0						
2	0						
3	0						
i							

d[i][j] = 0, i==0 또는 j==0
d[i][j] = max(d[i-1][j], d[i][j-1]) + m[i][j], i>0, j>0

이항계수

- (a+b)^{n}에서 a^{n-k}b^{k}의 계수를 구하는 문제.$\binom{n}{k}$
- (a+b)(a+b)...(a+b)처럼 n개의 (a+b)가 있을 때, k개의 b와 n−k개의 a를 고르는 경우의 수. 또는 k개의 a와 n−k개의 b를 고르는 경우의 수.
- 파스칼의 삼각형

```
      1
    1   1
  1   2   1
1   3   3   1
```

```
1
1   1
1   2   1
1   3   3   1
```

	0	1	2	3	k
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
n					

▪ DP 배열

```
C[n][k] = 1, k = 0 or n == k
C[n][k] = C[n-1][k-1] + C[n-1][k]
```

	0	1	2	3	k
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
n					

계산하려는 값의 오른쪽 부분은 없어도 됨.

```
for i : 0 -> n
    for j : 0 -> i
```

```
for i : 0 -> n
    for j : 0 -> min(i, k)
```

중복 조합의 합

- 1 2 3으로 만든 중복 조합의 합이 N이 되는 경우의 수를 구하시오.

N 입력	4를 만들 수 있는 경우	1만 사용해 만들 수 있는 합 1, 2, 3, 4, …
		2 : 2, 4, 6, 8…
4	1 + 1 + 1 + 1	3 : 3, 6, 9, 12, …
	1 + 1 + 2	1, 2 : 3, 4, 5, …
	1 + 3	1, 3 : 4, 5, 6, …
	2 + 2	2, 3 : 5, 7, 8, …

- N이 10인 경우

	0	1	2	3	4	5	6	7	8	9	10	j
∅	0	0	0	0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	1	1	1	1	1	
2	1	1	2	2	3	3	4	4	5	5	6	
3	1	1	2	3	4	5	7	8	10	12	14	
i												

1만 사용, 1과 2 사용, 2만 사용

```
for i : 1 -> 3
    for j : 1 -> 10
        if j >= i
            d[i][j] = d[i-1][j] + d[i][j-i]
        else
            d[i][j] = d[i-1][j]
```

⇨

```
for i : 1 -> 3
    for j : i -> 10
        d[j] += d[j-i]
```

최장 공통 부분 수열

- LCS (Longest Common Subsequence)
  - 부분 수열
    - 주어진 수열에서 순서가 바뀌지 않게 일부 숫자를 고른 것.
  - 최장 공통 부분 수열
    - 두 개의 수열에서 고른 부분 수열이 일치할 때, 가장 긴 수열의 길이 구하기.

3 2 5 7 4

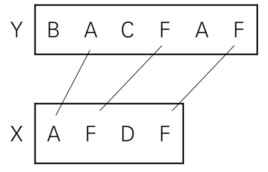
1 2 4 2 5 6 8 4

⇨ 2 5 4

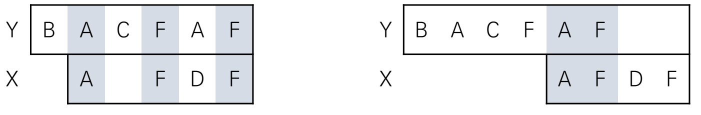
✓ 글자의 간격을 조절해 아래 위 글자를 일치시켜 본다.

▪ LCS 적용

- 양쪽에서 같은 글자끼리 선으로 연결 할 때, 교차되지 않는 선분의 최대 개수는?



글자를 맞추는 여러가지 방법



▪ LCS 테이블

- 각각에 속한 글자를 Xi, Yj라고 하면, 이 경우 1<=i<=4, 1<=j<=6.
- i 또는 j가 0인 경우는 글자가 없는 경우를 나타냄.
- M[i][j]는 Xi와 Yj까지 고려했을 때의 LCS 길이를 저장.

	1	2	3	4	5	6
Y	B	A	C	F	A	F
X	A	F	D	F		

	j-1	j				
Y	B	A	C	F	A	F
X		A		F	D	F
	i-1	i				

Xi == Yj인 경우
M[i][j] = M[i−1][j−1]+1

글자가 일치하는 경우, i−1과 j−1까지
고려한 수열의 길이 +1

▪ LCS 테이블

		j-1	j			
Y	B	A	C	F	A	F
X	A	F	D	F		
		i-1	i			

Xi != Yj인 경우
M[i][j] = max(M[i][j-1], M[i-1][j])

글자가 일치하지 않는 경우, i와 j-1까지 또는 i-1과 j까지 고려한 길이 중 큰 값

i == 0 또는 j == 0인 경우
M[i][j] = 0

M[i][j]

	∅	B	A	C	F	A	F	j
∅	0	0	0	0	0	0	0	
A	0	0	1	1	1	1	1	
F	0							
D	0							
F	0							
i								

부분 집합의 합

- True/False를 사용.
  - { 1, 1, 2, 2, 1 }을 원소로 갖는 집합 a의 부분 집합에서, 합이 5가 되는 경우가 있는가?
  - 부분집합으로 고려한 원소 a_i, 0 <= i <= 5, a_0는 공집합.
  - 부분집합의 합 j, 0 <= j <= 5
  - m[i][j]는 i원소까지 고려했을 때, 부분 합 j를 만들 수 있으면 1 아니면 0.
  - j가 존재하려면 m[i-1][j]가 존재하거나 m[i-1][j-a_i]가 존재해야 함.
    - m[i-1][j]==1 : a_i를 고려하기 전에 이미 합이 j인 경우가 존재함.
    - m[i-1][j-a_i]==1 : a_{i-1}까지 고려한 합 중에 j-a_i가 있으면, a_i를 더해서 j를 만들 수 있음.

▪ 부분 집합의 합 DP

		0	1	2	3	4	5	j
0	Φ	1	0	0	0	0	0	
1	1	1	1	0	0	0	0	
2	1	1	1	1				
3	2	1						
4	2	1						
5	1	1						
i	a_i							

어떤 원소든 고려해도
포함하지 않으면 합이 0

$$m[i][j] = m[i-1][j] \mid\mid m[i-1][j-a_i],\ i>0,\ j>0$$
$$m[i][j] = 1,\ j=0$$
$$m[i][j] = 0,\ i=0,\ j>0$$

✓부분 집합의 합이 5가되는 경우의 수는?

- {a1, a3} {a2, a3} 모두 {1, 2}이다. 이 둘을 다른 경우로 보는 경우.

	0	1	2	3	4	5	j
Φ	1	0	0	0	0	0	
1	1	1	0	0	0	0	
1	1	2	1	0	0	0	
2	1	2	2	2	1	0	
2	1	2	3	4	3	2	
1	1	3	5	7	7	5	
a_i							

m[i][j] = 1, j=0
m[i][j] = 0, i=0, j>0
m[i][j] = m[i−1][j] + m[i−1][j−a_i], i>0, j>0, ai<=j
m[i][j] = m[i−1][j], i>0, j>0, ai>j

✓부분 집합의 합이 5가 되는 경우의 수는?

- {a1, a3} {a2, a3} 모두 {1, 2}이다. 이 둘을 같은 경우로 보는 경우.
- 원소를 정렬한다.

	0	1	2	3	4	5	j
Φ	1	0	0	0	0	0	
1	1	1	0	0	0	0	
1	1	1	1	0	0	0	
1	1	1	1	1	0	0	
2	1	1	2	2	1	1	
2	1	1	2	2	2	2	
a_i							

m[i][j] = 1, j=0
m[i][j] = 0, i=0, j>0
m[i][j] = m[i-1][j], a_i > j
m[i][j] = max(m[i-1][j − a_i] , m[i-1][j]), a_i == a_{i-1} , ai<=j , i>0, j>0
m[i][j] = m[i-1][j] + m[i-1][j-a_i], a_i != a_{i-1} , i>0, j>0, ai<=j

연속한 1이 없는 이진수

- 1로 시작하는 n자리 이진수에서 연속한 1이 없는 경우의 수를 구하시오.
- n=4인 경우 1000, 1001, 1010이 가능.

i번째 자리	1	2	3	4	5	6
0인 경우	0	1	1	2		
1인 경우	1	0	1	1		
경우의 수	1	1	2	3		

- i번 째 자리가 0인 경우와 1인 경우를 나눠서 생각한다.
  - 0인 경우는 i-1이 0과 1인 경우 모두 가능하다.
  - 1인 경우는 i-1이 0인 경우만 가능하다.
  - 첫 번째 자리는 0인 경우는 불가 하므로 0, 1인 경우는 1가지 경우이다.

```
d[1][0] = 0
d[1][1] = 1
for i : 2 -> n
    d[i][0] = d[i-1][0] + d[i-1][1]
    d[i][1] = d[i-1][0]
dn = d[n][0] + d[n][1]
```

연속한 0이 없는 이진수

- 1로 시작하는 n자리 이진수에서 0이 3번 이상 연속하지 않는 경우의 수를 구하시오.
- n=4인 경우 1001, 1010, 1011, 1100, 1101, 1110, 1111.

i번째 자리	1	2	3	4	5	6
첫 번째 0인 경우	0	1	1	2		
두 번째 0인 경우	0	0	1	1		
1인 경우	1	1	2	4		
경우의 수	1	1	3	7		

- i번째 자리는 세 가지 경우가 가능 (i〉2인 경우)
  - 첫 번째 0인 경우 : d[i][0] (이전 자리가 1인 경우)
  - 두 번째 0인 경우: d[i][1] (이전 자리가 첫 번째 0인 경우)
  - 1인 경우 : d[i][2] (이전 자리가 0이거나 1인 경우)

```
d[1][0] = 0
d[1][1] = 0
d[1][2] = 1
for i : 2 -> n
      d[i][0] = d[i-1][2]
      d[i][1] = d[i-1][0]
      d[i][2] = d[i-1][0] + d[i-1][1] + d[i-1][2]
dn = d[n][0] + d[n][1] + d[n][2]
```

3가지 색으로 칠하기

- 축제 행렬이 움직이는 길가의 집을 3가지 색으로 칠하려고 한다.
  - 이웃한 집은 서로 다른 색을 칠한다.
  - 각 집마다 색상 별로 소요 비용이 다르다.
  - 첫번째 집부터 순서대로 칠해 나가야 한다.
  - 총 n개의 집을 칠하려고 할 때 전체를 칠하는 최소 비용은?
  - 3개의 집을 칠하는 경우
    - 첫번째 집부터 각 색으로 칠할 때의 비용이 주어진다.

m	1	2	3	색
1	5	7	6	
2	2	4	8	
3	5	2	3	
집				

▪ i번 집까지 칠하는 비용

- i번 집을 각 색으로 칠하는 비용을 계산한다.
- 단, i번 집을 칠할 색상은 i-1에서 사용하지 않은 색이어야 한다.
- 예를 들어 i번 집을 1번으로 칠하는 경우, i-1은 2 또는 3번으로 칠해져 있어야 한다.

m (집 \ 색)	1	2	3
1	5	7	6
2	2	4	8
3	5	2	3

d (집 \ 색)	1	2	3
1	5	7	6
2			
3			

```
d[1][1] = m[1][1]
d[1][2] = m[1][2]
d[1][3] = m[1][3]
for i : 2 -> n
      d[i][1] = min(d[i-1][2], d[i-1][3]) + m[i][1]
      d[i][2] = min(d[i-1][1], d[i-1][3]) + m[i][2]
      d[i][3] = min(d[i-1][1], d[i-1][2]) + m[i][3]
dn = min(d[n][1], d[n][2], d[n][3])
```

건너뛰기

- 2차원 배열에 표시된 숫자만큼 오른쪽이나 아래로 이동해 도착지에 도달하는 경로의 수를 찾는 문제.
  - 3이내의 자연수로 채워진 NxM 배열.
  - 맨 왼쪽 위 출발, 맨 오른쪽 아래 도착.

2	3	1	3	1
2	1	1	1	1
3	1	2	1	3
3	1	2	3	2
3	1	2	3	2

- i, j 칸에 도착하는 경우의 수 d[i][j]
  - 왼쪽의 모든 칸 k에 대해 i, j로 올 수 있으면 d[i][j] += d[i][k]
  - 위쪽의 모든 칸 k에 대해 i, j로 올 수 있는면 d[i][j] += d[k][j]

2	3	1	3	1
2	1	1	1	1
3	1	2	1	3
3	1	2	3	2
3	1	2	3	2

d (columns: j, rows: i)

1				

```
for i : 1 -> N
    for j : 1 -> M
        for k : 1 -> j-1
        {
            if(A[i][k] == j-k)
                D[i][j] += D[i][k];
        }
        for k : 1 -> i-1
        {
            if(A[k][j] == i-k)
                D[i][j] += D[k][j];
        }
    }
}
```

거스름 동전의 최소 개수

- 동전의 단위가 1, 3, 5인 나라가 있다. 9원의 거스름돈을 최소한의 동전을 사용해 거슬러 줄 때, 동전의 개수를 구하라.
  - d[i][j] : i 동전까지 고려해 j원을 만들 때 동전의 최소 개수
  - a[i] = {1, 3, 5} : 동전의 액면가
  - 같은 금액의 동전은 충분히 있다.
  - 예를 들어 4원을 만들려면, 1원만 사용하거나, 3원까지 사용해 4원을 만들 수 있다. 이 중 개수가 적은 쪽을 택한다.
    - d[(3)][4−3] : 3원까지 사용하고, 1원을 만들 때 동전의 최소 개수.
    - d[(3)][4] = min(d[(3)][4−3] + 1, d[(1)][4])

- i 동전까지 고려해 j원을 만들 때 동전의 최소 개수

	d	0	1	2	3	4	5	6	7	8	9	
1원	0	0	1	2	3	4	5	6	7	8	9	j
3원	1	0										
5원	2	0										
	i											

```
// 1원 짜리만 고려한 경우
d[0][j] = j

// 1원보다 큰 동전까지 고려한 경우
d[i][j] = d[i-1][j]   // j < a[i] (거스름이 i동전보다 작은 경우)
d[i][j] = min (d[i][j-a[i]] + 1, d[i-1][j])
```

0-1 배낭 짐싸기

- 크기가 정해진 박스가 가득 찰 때까지 물건을 담을 수 있는 해피 박스 이벤트가 열린다고 한다. 물건의 크기와 가치가 주어질 때 박스에 들어간 물건의 최대 가치는 얼마인가? 각 물건은 1개씩 있다.
  - 박스의 크기 10. 물건 크기의 합이 상자 크기를 넘지 않으면 담을 수 있다.
  - 준비된 물건의 크기와 가치

물건	크기	가치
1	6	12
2	5	10
3	5	15
4	4	6

▪ 박스의 크기를 늘려가며 담아 본다.

- 1번 물건만 고려, 2번까지 고려, 3번까지 고려하는 식으로 생각해본다.
- i번 물건을 넣을 수 있는 상자의 용량을 생각한다.

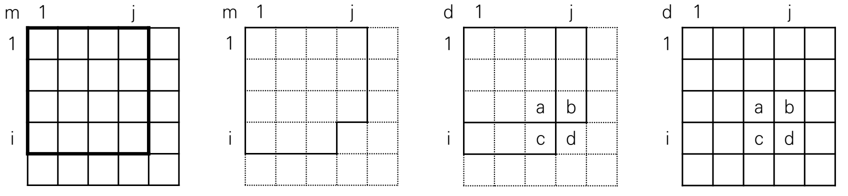
d[i][j] : 상자의 크기가 j이고 i번 물건까지 고려한 경우
d[i][j] = d[i-1][j] , j < w[i] (상자보다 물건이 크면)
d[i][j] = max(d[i-1][j-w[i]] + v[i], d[i-1][j])

물건	크기 w	가치 v
1	6	12
2	5	10
3	5	15
4	4	6

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	12	12	12	12	12
2	0	0	0	0	0	10	12	12	12	12	12
3	0	0	0	0	0	15	15	15	15	15	25
4	0	0	0	0	6	15	15	15	15	21	25

배열 원소의 합 구하기

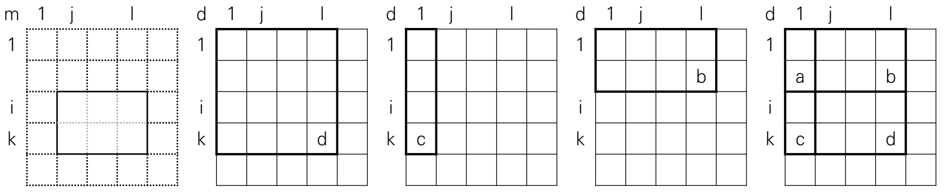
- 왼쪽 위 부터 i, j까지의 합 구하기.
  - 모든 i, j에 대해 구하는 경우.
  - d[i][j]에 i, j까지의 합을 저장한다.
  - d[i][j]는 i, j칸을 제외한 나머지 칸의 합에 i, j칸을 더한다.
    - d[i][j] = d[i−1][j] + d[i][j−1] − d[i−1][j−1] + m[i][j]



i, j를 제외한 부분의 합은 두 사각형 내부의 합을 더한 후 겹치는 부분을 뺀다.
b + c − a

▪2차원 배열 내부 사각 영역의 합 구하기.

- 왼쪽 위 i, j. 오른쪽 아래 k, l.
- 앞에서 구한 d[][]를 활용한다.
  - 1, 1 부터 k, l 까지 면적이 d.
  - 1, 1 부터 k, j−1 까지 면적이 c.
  - 1, 1 부터 i−1, l 까지 면적이 b.
  - 1, 1 부터 i−1, j−1 까지 면적이 a.
  - r = d − c − b + a



r = d[k][l] − d[k][j−1] − d[i−1][l] + d[i−1][j−1]

연속 행렬 곱셈

- 행렬 곱셈에서 결합법칙을 적절히 사용하면 연산 횟수를 줄일 수 있음.
  - 행렬의 곱셈 A1 x A2

$$\underset{\mathbf{2}\times 2}{\begin{bmatrix} a1 & a2 \\ a3 & a4 \end{bmatrix}} \times \underset{2\times \mathbf{3}}{\begin{bmatrix} b1 & b2 & b3 \\ b4 & b5 & b6 \end{bmatrix}} = \underset{\mathbf{2}\times \mathbf{3}}{\begin{bmatrix} a1b1+a2b4 & a1b2+a2b5 & a1b3+a2b6 \\ a3b1+a4b4 & a3b2+a4b5 & a3b3+a4b6 \end{bmatrix}}$$

    - 곱셈 연산의 횟수 : 2 * 2 * 3 = 12
  - 곱하는 행렬 크기 저장
    - 중복은 제거
    - A1의 크기 p[0]xp[1], A2의 크기 p[1]*p[2]
    - Ai의 크기 p[i-1]xp[i]

	0	1	2
p	2	2	3

▪ 연속 행렬 곱셈

- A1(2x2), A2 (2x3), A3 (3x4) 행렬을 곱하는 경우
- A1A2의 크기 2x3
- A1A2A3의 크기 2x4
- Ai부터 Aj까지 곱하는 경우의 크기 p[i-1] x p[j]

	Ai				Aj	
	A1		A2		A3	
	2	2	2	3	3	4
	p[0]	p[1]	p[1]	p[2]	p[2]	p[3]
A1A2			p[0] x p[2]			
A1A2A3			p[0] x p[3]			
Ai…Aj			p[i-1] x p[j]			

▪ 연속 행렬 곱셈

- A1A2A3의 곱셈 횟수
  - (A1)(A2A3)와 (A1A2)(A3)의 곱셈 횟수 중 작은 쪽을 택함.
  - (왼쪽)(오른쪽)과 같이 결합 법칙이 적용 된 경우의 전체 곱셈 횟수.
    - (왼쪽) 내부 곱셈횟수 + (오른쪽) 내부 곱셈 횟수 + (왼쪽)(오른쪽) 사이 곱셈 횟수
  - (Ai…Ak)(A_{k+1}…Aj)로 표현. (i <= k < j)

	Ai		Ak		Aj	
	A1		A2		A3	
	2	2	2	3	3	4
	p[0]	p[1]	p[1]	p[2]	p[2]	p[3]
A1A2	p[0]*p[1]*p[2]					
(A1A2)(A3)	(p[0]*p[1]*p[2]) + (0) + p[0]*p[2]*p[3]					
	(Ai…Ak)곱셈횟수 + (A_{k+1}…Aj) 곱셈횟수 + (곱한 결과)(곱한 결과) 사이의 곱셈 횟수					

▪ 연속 행렬 곱셈

- D[i][j]는 Ai부터 Aj까지 최소 곱셈의 횟수.
  - (Ai⋯Ak)(A_{k+1}⋯Aj)인 경우
    - D[i][j] = D[i][k] + D[k+1][j] + p[i−1]p[k]p[j]
    - i<=k<j이므로 모든 k에 대해 계산해서 최소값을 택한다.

D[i][j] = min(D[i][k] + D[k+1][j] + p[i−1]p[k]p[j]), i<=k<j인 모든 k에 대해
D[i][j] = 0, i==j 인 경우

행렬 A1⋯A5에 대한 곱셈

D	1	2	3	4	5	j
1	0					
2		0				
3			0			
4				0		
5					0	
i						

A1A2A3은 A1A2와 A2A3가 필요.
A1A2A3A4는 A1A2A3, A2A3A4, A1A2, A2A3가 필요.
2개, 3개⋯ 식으로 곱해지는 행렬의 개수를 늘려감

▪ A1…A_N에 대한 연속 행렬 곱셈

D	1	2	3	4	5	j
1	0					
2		0				
3			0			
4				0		
5					0	
i						

l	행렬 수	계산 순서	i	j
1	2	(1, 2) (2, 3) (3, 4) (4, 5)	1~4	i+1
2	3	(1, 3) (2, 4) (3, 5)	1~3	i+2
3	4	(1, 4) (2, 5)	1~2	i+3
4	5	(1, 5)	1~1	i+4
			1~(N−l)	i+l

```
for l : 1 -> N-1
    for i : 1 -> N-l
        j = i + l
        for k : i -> j-1
            if(min > D[i][k] + D[k+1][j] + p[i-1]p[k]p[j])
                min = D[i][k] + D[k+1][j] + p[i-1]p[k]p[j]
        D[i][j] = min
```

카드 게임

- 두 사람이 카드를 집는 게임.
  - 숫자 카드를 한 줄로 늘어 놓고 교대로 가져간다.
  - 순서마다 양끝의 카드 중 하나를 선택해 가질 수 있음.
  - 한 사람이 가져갈 수 있는 카드의 숫자 합계 최대값은?
  - 단, 두 사람은 같은 전략을 써서 카드를 가져간다.

5	3	2	1	8	4
i					j

남은 카드의 왼쪽 인덱스 i, 오른쪽이 j라고 했을 때,
이번 순서에 가져갈 수 있는 카드는 i 또는 j 이다.
한 사람이 가져갈 수 있는 카드의 최대 값이 m[i][j]라고 한다.

▪ 카드 게임

✓ m[i][j]는 i부터 j까지 남은 카드에서 가질 수 있는 카드의 최대 합계.

5	3	2	1	8	4	
i					j	이번 차례에 남은 구간
	i+1				j	i를 택하면 다음 사람에 남는 구간
		i+2			j	다음 사람이 i+1을 택하면 남는 구간
	i+1			j-1		다음 사람이 j를 택하면 남는 구간

다음 사람은 m[i+2][j]와 m[i+1][j-1] 중 작은 쪽을 남길 것임.

i				j-1		j를 택하면 다음 사람에 남는 구간
	i+1			j-1		다음 사람이 i를 택하면 남는 구간
i			j-2			다음 사람이 j-1을 택하면 남는 구간

다음 사람은 m[i+1][j-1]과 m[i][j-2] 중 작은 쪽을 남길 것임.

a[]는 카드 숫자가 저장된 배열.
min[i][j] = max(a[i] + min(m[i+2][j], m[i+1][j-1]),   // i를 택한 경우
                      a[j]+ min(m[i+1][j-1], m[i][j-2]) // j를 택한 경우

TSP

- n개의 도시.
  - 1번 도시에서 출발해 나머지 도시를 1번씩 거쳐 1번으로 되돌아 올 때 최소 비용을 구하는 문제.
  - 도시 i에서 다른 도시 j로 갈 수 있고, 각각의 비용은 m[i][j].
  - 모든 도시를 원소로 갖는 집합을 S, 그 부분집합을 s라고 정의한다.
  - S의 원소 i에서 경유지 없이 1로 가는 최소 비용. 경유지는 공집합.
    - d[i][∅] = m[i][1] , 1 〈 i 〈=n
  - i에서 2를 거쳐 1로 가는 최소 비용.
    - d[i][{2}] = m[i][2] + d[2][∅], 1 〈 i 〈= n && i != 2
  - i에서 2와 3을 거쳐 1로 가는 최소 비용
    - d[i][{2, 3}] = min(m[i][2]+d[2][{3}], m[i][3]+d[3][{2}]), 1 〈 i 〈=n, $i \notin \{2,3\}$

- i에서 2,3,4를 거쳐 1로 가는 최소 비용. i $\notin$ {2,3,4}
  - d[i][{2,3,4}] = min(m[i][2] + d[2][{3,4}],
    m[i][3] + d[3][{2,4}],
    m[i][4] + d[4][{2,3}])
- 1에서 모든 도시를 거쳐 1로 되돌아 오는 최소 비용
  - d[1][S-{1}]
- 경유지인 부분 집합의 원소를 1개 부터 n-1개 까지 늘림.
- 경유지 집합을 이진수로 표시.
  - bit-n이 n번 도시 포함여부 표시.
  - {1, 2, 3, 4, 5}에서 1을 제외한 모든 도시가 경유지인 경우.
    - d[1][S-{1}] -> d[1][111100]