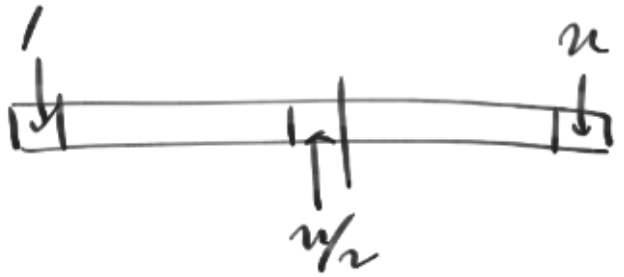
알고리즘 공부

[2017 - 2]

1. $n = 2^k$, $A[1 \cdots n]$ array.

(a)



$$T(n) = 4T\left(\frac{n}{2}\right) + O(n)$$

$$= 4\left(4T\left(\frac{n}{2^2}\right) + n\right) + n$$

$$\cdots$$

$$= 4^k T\left(\frac{n}{2^k}\right) + (1 + 4 + \cdots + 4^{k-1})n$$

$$= 4^k T(n/2^k) + n \sum_{i=0}^{k} 4^i$$

$$= 4^k T(n/2^k) + n \cdot \frac{4(4^{k+1} - 1)}{4 - 1}$$

when $n/2^k = 1$, or $k = \log_2 n$,

$$= n^2 T(1) + O(n \log n)$$

$$\therefore T(n) = O(n^2)$$

master's theorem 이용

$a \log_b 4 > n$

$n^{\log_2 3} > n$

$\therefore \; T(n) = \underline{\Theta(n^2)}$

asymtotic bound

(b) No. 자신 호출 필요.

2. $\langle A_1 \cdots A_n \rangle$

$A_1 \to \mathbb{R}^{p_0 \times p_1}$, $A_n \to \mathbb{R}^{p_{n-1} \times p_n}$

let m_{ij} : $\langle A_i \cdots A_j \rangle$ operation cost.

(a)

$$m_{ij} = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{ m_{ik} + m_{k+1 j} + p_{i-1} p_k p_j \} & \text{, if } i < j \end{cases}$$

(b) Given $p[0 \cdots n]$

Algorithm (p)

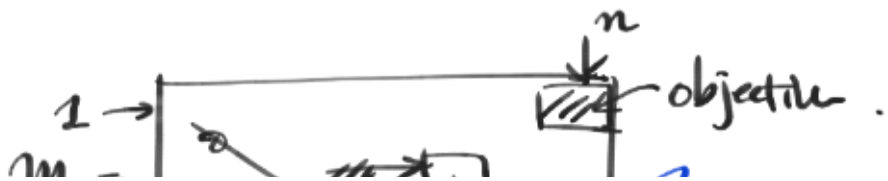
$n = |p| - 1$

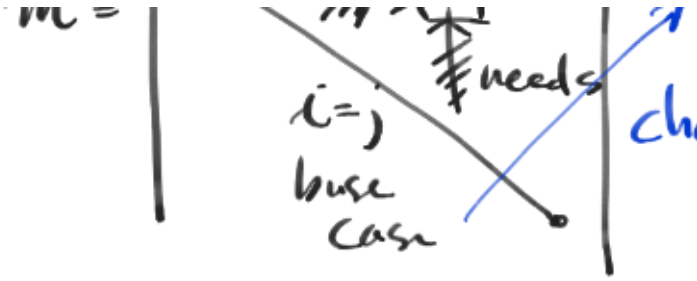
let $m[1 \cdots n, 1 \cdots n]$ be array.

call $m[:,:]$ initialized by ∞
return lookup$(p, m, 1, n)$

```
lookup(p, m, i, j)
    if m_ij < ∞
        return m_ij
    if i == j
        m_ij = 0
        return m_ij
    q = ∞
    for k = i to j-1
        q = min(q, lookup(p, m, i, k)
                   + lookup(p, m, k+1, j)
                   + p_{i-1} p_k p_j)
    m_ij = q
    return m_ij
```

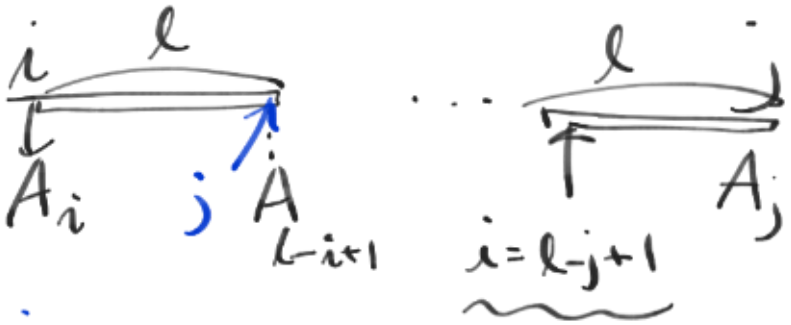
#iteration version





chain length $l = 1, 2, \dots, n$

base case



$j = i + l - 1$

```
Algorithm (p)
    n = |p| - 1
    let m[1...n, 1...n] be 2x2 array
    all m[:,:] initialized by ∞
    for i = 1 to n
        m_ii = 0
    for l = 2 to n
        q = ∞
        for i = 1 to l-n+1
            j = i+l-1

            q = min(q, m_ik + m_k+1j
                        + p_i-1 p_k p_j)
        m_ij = q
```

return m_{1n}

(c) $T(n) = O(n^3)$

n^2 entries,

each entry takes $O(n)$ time

[2017-1]

1. $T(n) = 4T(\frac{n}{4}) + O(n)$

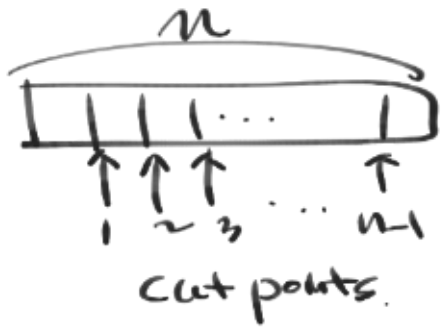
$n^{\log_4 4} = n$

$\therefore T(n) = \Theta(n \log n)$

2. given $\underbrace{p[1 \dots n]}_{\text{price information}}$

let r_n : optimal cost by cutting.

(a) $\exists\ 2^{n-1}$ cases.



(b)

(b)

$$r_n = \max\left(p_n, \max_{1 \le i < n} (r_i + r_{n-i})\right) \quad \text{if } n \ge 1$$

simplify is possible.

$$= \begin{cases} \max_{1 \le i \le n} (p_i + r_{n-i}) & \text{if } n \ge 1 \\ 0. & \text{if } n = 0 \end{cases}$$

(c)

```
rodcut(p)
    n = |p|
    let r[0 ... n] be array.
    all r[...] initialized by -∞
    return lookup(p, r, n)

lookup(p, r, i)
    if r_i ≥ 0   return r_i
    if i == 0
        r_i = 0
        return r_i
    q = -∞
    for j = 1 to i
        q = max(q, p_i + lookup(p, r, i-j))
```

$r_i = q$.
return r_i

iteration version.

```
rodcut(p)
    n = |p|
    let r[0 ... n] be new array.
    all initialized by -∞

    r[0] = 0
    for j = 1 to n
        q = -∞
        for i = 1 to j
            q = max(q, p_i + r_{j-i})
        r_j = q
    return r_n
```

(d) $T(n) = O(n^2)$

n entries required.

each entry takes $O(n)$ time.

3. Dijkstra(G, s)

```
v.d = ∞ for all v ∈ G.V
s.d = 0
Create priority queue Q.
Create set A.
Q ← all v ∈ G.V
while Q != ∅
        u ← Q.pop()
        A ← u
        for v in G.adj[u]
            if v ∉ A and
                v.d > u.d + w(u,v)
                v.d = u.d + w(u,v)
                Q.update(v, v.d)
```

$T(n) = O((|V|+|E|)\log|V|)$

마지막 수정: 오후 5:15