AJOU UNIV.
PROGRAMMING
CONTEST

2021
아주대학교
프로그래밍 경시대회

풀이 슬라이드

Div. 1	Div. 2	출제자
A. 코딩 바이오리듬	A. 코딩 바이오리듬	하주헌
B. 아주 서바이벌	B. 아주 서바이벌	신상민
C. 웹 브라우저 1	**C. 웹 브라우저 2**	하주헌
D. 스터디 시간 정하기 1	**D. 스터디 시간 정하기 2**	정의찬
E. 엘리베이터 조작	E. 엘리베이터 조작	정의찬
F. W3W (What 3 Words)	F. W3W (What 3 Words)	정세훈
G. 전파와 병합 1	**G. 전파와 병합 2**	하주헌
H. 화질 - 자동(480p)	H. 화질 - 자동(480p)	신상민

총 참가자 수 : 94명

1회 이상 제출자 수 : 39명

총 참가자 수 : 19명

1회 이상 제출자 수 : 6명

A. 코딩 바이오리듬

Div. 1		Div. 2	
전체 제출 :	129	전체 제출 :	12
맞은 사람 :	39	맞은 사람 :	6
정답률 :	33.06%	정답률 :	50.00%

A. 코딩 바이오리듬

출제자: 하주헌(hjhj97)

$$\sum_{i=1}^{4}(Y_i - y_i)^2 \times \sum_{j=1}^{2}(M_j - m_j)^2 \times \sum_{k=1}^{2}(D_k - d_k)^2$$

- 수식은 거창해 보이지만, 실제로는 두 문자열에서 각 자릿수에 대응되는 두 수의 차이를 제곱해서 모두 더하면 된다.
- 정수 (20211030)를 각 자릿수 별로 배열([2,0,2,1,1,0,3,0]) 에 담는 방법

  → 정수를 10의 나머지 연산의 값을 저장한 뒤에, 10으로 나눠주는 연산을 반복.
- 같은 값을 가지는 날짜가 여러개라면 가장 빠른 날짜를 찾아야 한다.
- 가장 빠른 날짜 == 가장 작은 값 이다.
- 따라서 처음부터 날짜를 크기순으로 정렬한 후에 찾거나,

  또는 [1,i-1]에서 가장 작은 날짜와 i 번째 날짜의 크기를 비교해서 가장 작은 값을 찾으면 된다.

  시간복잡도 : $O(N)$ 정렬할 경우 : $O(NlogN)$

B. 아주 서바이벌

Div. 1		Div. 2	
전체 제출 :	168	전체 제출 :	7
맞은 사람 :	19	맞은 사람 :	2
정답률 :	11.31%	정답률 :	28.57%

B. 아주 서바이벌

출제자: 신상민(qilip)

- 배열을 이용해 유저별 아이템 개수와 위치를 저장한다.
- item[a][b] – 유저 a의 b아이템 개수
- pos[a] – 유저 a의 현재 위치
- 로그를 순서대로 확인하면서 매 로그마다 $O(1)$의 시간복잡도로 해당 로그의 부정행위 유무를 확인할 수 있다.
- 찾아낸 부정행위 목록을 정렬하여 중복을 제거하고 출력한다.
- 총 시간복잡도: $O(N)$

아이템 번호	A	B	C	…
1	0	0	2	…
2	1	1	0	…
3	1	0	0	…
4	0	1	0	…
…	…	…	…	…

유저 번호	유저 위치
A	1
B	1
C	6
D	40
…	…

C. 웹 브라우저 1,2

Div. 1		Div. 2	
전체 제출 :	143	전체 제출 :	16
맞은 사람 :	11	맞은 사람 :	4
정답률 :	7.69%	정답률 :	25.00%

C. 웹 브라우저

출제자: 하주헌(hjhj97)

웹 브라우저 1과 2의 차이는 **최대 캐시 용량**이 제한되어 있는지 여부이다.

- **뒤로 가기**와 **앞으로 가기** 모두 공통적으로 먼저 들어온 정보는 늦게 나가고, 늦게 들어온 정보는 먼저 나간다 → 자료구조 stack의 작동 방식
- 압축 연산이 실행되면, 뒤로 가기 공간에 저장되어 있던 정보들은 모두 꺼내서 연속해서 중복된 번호가 등장할 경우, 하나만 남기고 삭제한 뒤에 다시 공간에 넣어야 함.
- 하지만 이때 stack을 사용한다면 원소의 순서가 역전된다. 예를 들어 스택에 [1,2,3 이 들어있는 상태에서 압축 연산이 실행되어 원소를 꺼냈다가 다시 집어 넣으면 [3,2,1 이 된다.

C. 웹 브라우저

출제자: 하주헌(hjhj97)

- 따라서 뒤로 가기 공간에는 압축 연산이 실행되었을 때는 먼저 들어간 원소는 먼저 나오고, 늦게 들어간 원소는 늦게 나오도록 해야함

  → 자료구조 queue의 작동 방식

- 자료구조에서 stack과 queue의 작동 방식을 동시에 갖고 있는 자료구조

  → deque

- 따라서 뒤로 가기 공간에는 deque, 앞으로 가기 공간에는 stack을 사용하면 된다.
- 만약 자료구조 deque을 쓰지 않고 푼다면?

  → 압축연산에서 원소를 빼낸 뒤에 다시 집어넣을 때, 본인이 직접 원소의 순서를 역전시켜서 집어 넣으면 원본의 순서를 유지할 수 있다.

시간복잡도 : $O(N \times Q)$

D. 스터디 시간 정하기 1,2

Div. 1		Div. 2	
전체 제출 :	108	전체 제출 :	11
맞은 사람 :	3	맞은 사람 :	6
정답률 :	2.77%	정답률 :	54.54%

D. 스터디 시간 정하기 1, 2

출제자: 정의찬(define_chan)

- 스터디 가능한 시간들을 입력 받아 시간 만족도가 최대인 구간을 찾아 출력하는 문제
- 스터디 시간 정하기 1과 2의 차이는 입력 값 범위 차이이다.

  스터디 시간 정하기 1 ($0 \le N \le 100,000$, $1 \le T \le 100,000$, $0 \le Si < Ei \le 100,000$)

  스터디 시간 정하기 2 ($0 \le N \le 1,000$, $1 \le T \le 1,000$, $0 \le Si < Ei \le 1,000$)

D. 스터디 시간 정하기 2

출제자: 정의찬(define_chan)

- 스터디 시간 정하기 2의 경우 입력 값 최대 범위가 작아 naive하게 다중반복문으로도 풀 수 있다.
- 먼저 각 가능한 시간의 시작 시각 s와 끝나는 시각 e를 입력받으면 아래와 같이 반복문으로 구간을 체크해줄 수 있다.

```
scanf(" %d %d", &s, &e);
for (int i = s; i < e; i++)
    time[i]++;
```

- T시간 만큼의 시간만족도 합또한 오른쪽과 같이 다중반복문으로 구할 수 있다.

```
for (int i = 0; i <= 1000 - t; i++) // 스터디 시작 시간
{
    sum = 0;
    for (int j = 0; j < t; j++) // t 시간동안의 시간만족도
        sum += time[i + j];
    if (_max < sum) // 최대값 비교
    {
        _max = sum;
        _maxi = i;
    }
}
```

시간복잡도 : $O(N \times T)$

D. 스터디 시간 정하기 1

출제자: 정의찬(define_chan)

- 스터디 시간 정하기 1의 경우 **누적합** 또는 **슬라이딩 윈도우**를 적용하여 풀어야 한다.
- 먼저 각 시간들이 얼마나 겹치는지를 구해야한다.
  - 가능한 시간의 시작 시각 s와 끝나는 시각 e를 아래와 같이 표시하고,
  - **누적합**을 구하면 각 시간에 대해서 얼마나 겹치는지를 알 수 있다.

ex) 1번사람 가능한 시간 1 ~ 7, 2번사람 가능한 시간 3 ~ 5 이라고 할때

time[1] = 1; time[7] = -1; time[3] = 1; time[5] = -1;

Index	0	1	2	3	4	5	6	7
time[]	0	1	0	1	0	-1	0	-1

D. 스터디 시간 정하기 1

출제자: 정의찬(define_chan)

- **누적합**

```
for (int i = 1; i < 8; i++)
    time[i] = time[i - 1] + time[i];
```

Index	0	1	2	3	4	5	6	7
time[]	0	1	1	2	2	1	1	0

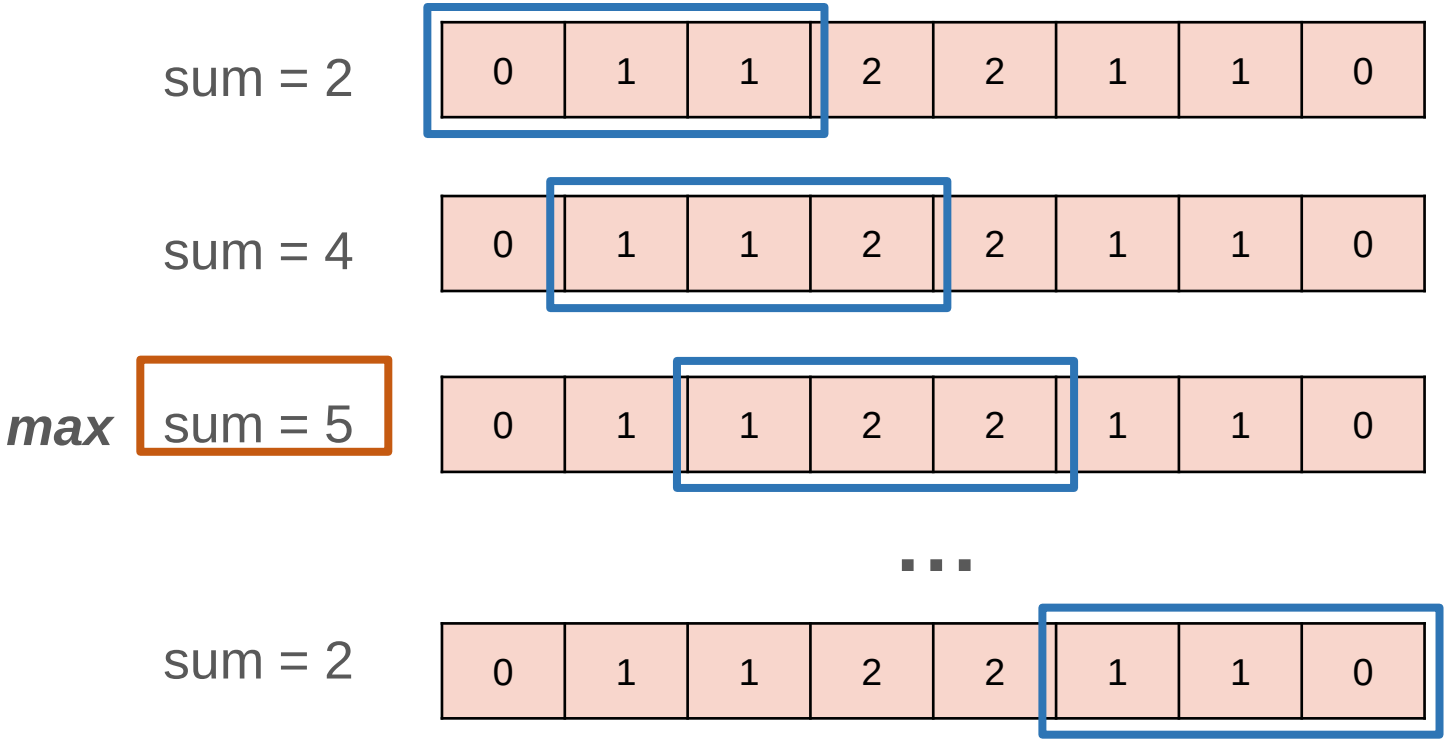
각 시간에 대하여 참여가능한 참가자 수를 알 수 있다.

- 그 다음은 최대 구간 합을 찾으면 된다. 구간 합은 **슬라이딩 윈도우** 알고리즘을 사용하여 구할 수 있다.

D. 스터디 시간 정하기 1

출제자: 정의찬(define_chan)

- **슬라이딩 윈도우** (스터디 시간 T = 3)



- 시간복잡도 : $O(N)$

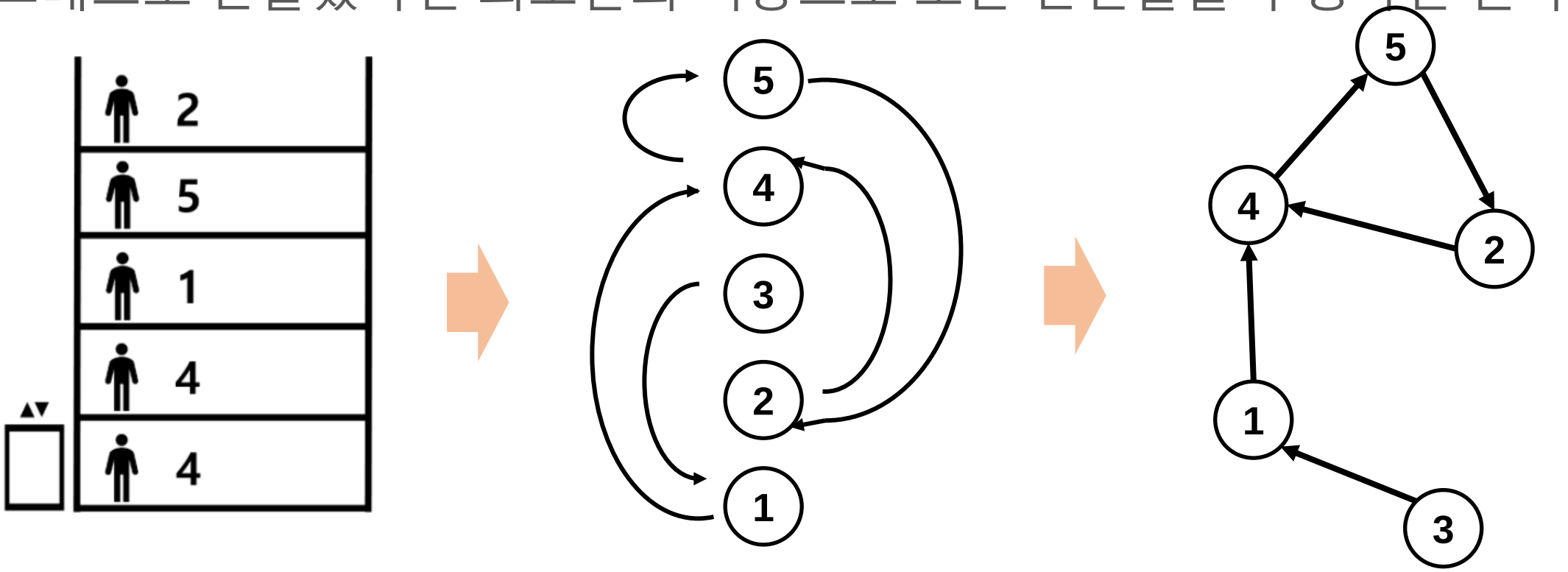
E. 엘리베이터 조작

Div. 1		Div. 2	
전체 제출 :	43	전체 제출 :	11
맞은 사람 :	2	맞은 사람 :	1
정답률 :	4.66%	정답률 :	9.09%

E. 엘리베이터 조작

출제자: 정의찬(define_chan)

- 최대한 사람을 연속해서 이동시켜 주는 것이 최적이다.
- 입력받은 데이터를 그래프화 시켜서 보면 접근하기 쉽다.
- 그래프로 만들었다면 최소한의 이동으로 모든 간선들을 수행하면 된다.



E. 엘리베이터 조작

출제자: 정의찬(define_chan)

- **DFS** 알고리즘을 사용
- 1번 정점(1층)부터 **DFS** 탐색을 시작한다.
- 더 이상 갈 수 있는 곳이 없다면 in-degree(진입차수)가 0인 곳을 찾아 이어서 **DFS** 탐색을 한다.
- in-degree(진입차수)가 0인 곳이 더 이상 없다면 아직 방문안한 정점을 찾아 이어서 **DFS** 탐색을 한다.
- **DFS**탐색 순서가 버튼을 눌러야 하는 순서이다.
- 시간복잡도 : $O(N)$

F. W3W (What 3 words)

Div. 1		Div. 2	
전체 제출 :	8	전체 제출 :	9
맞은 사람 :	0	맞은 사람 :	0
정답률 :	0%	정답률 :	0%

F. W3W(What 3 Words)

출제자: 정세훈(smsm8111)

- ‘.’으로 구분되어 나타날 수 있는 문자열들의 개수를 구하는 문제
- ‘.’으로 구분했을 때 나눠지는 3개의 단어 중 첫번째 단어를 prefix로, 세 번째 단어를 suffix로 생각하여 가능한 개수를 찾는다.

F. W3W(What 3 Words)

출제자: 정세훈(smsm8111)

- 먼저 길이가 L인 문자열의 prefix개수를 구해보자
  - 길이가 L인 문자열의 prefix개수는 L개 이다.
  - **서로 다른 prefix의 길이는 같을 수 없다.**
  - 결국 L개의 prefix의 최소 길이의 합은 1~L까지의 합이 될 것이다.
  - 길이가 1부터 1413인 문자열의 총 길이의 합은

    1413 * (1413 + 1) / 2 = 998991이다.
  - 즉, 단어들의 길이의 총 합이 100만인 단어 목록에서 찾을 수 있는 prefix의 개수는최대 1413개이다.

F. W3W(What 3 Words)

출제자: 정세훈(smsm8111)

- 결국 '.'으로 구분되는 3개의 단어 중 첫번째 단어로 가능한 단어는 단어 목록에서 1413개 보다 더 많이 찾을 수 없음이 보장된다.
- 세번째 단어도 suffix의 개념으로 prefix와 동일한 방법으로 구한다.
- 검색할 문자열 S가 주어졌을 때 반복문을 통해 suffix, prefix 목록을 만들 수 있다.
- 문자열 S에서 prefix와 suffix를 제거하면 두번째 단어를 구할 수 있다.
- 두번째 단어의 개수는 주어진 N개의 단어 목록들을 map/set에 저장한 후 검색하는 방법으로 구할 수 있다.
- 단, 문자열을 map/set에 저장하고 검색하는 과정의 시간을 줄이기 위해 문자열 hashing과정을 거쳐야 한다.

시간복잡도: $O(L \times logN)$

F. 전파와 병합 1,2

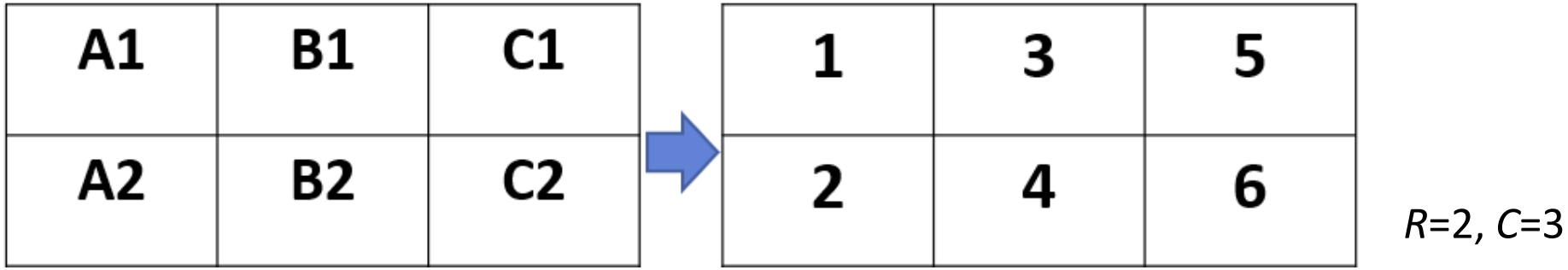
Div. 1		Div. 2	
전체 제출 :	1	전체 제출 :	3
맞은 사람 :	0	맞은 사람 :	1
정답률 :	0%	정답률 :	33.33%

G. 전파와 병합 2

출제자: 하주헌(hjhj97)

전파와 병합 2의 풀이는 1의 풀이의 부분집합에 속한다.

- 이 문제의 관건은 아래와 같이 grid 형태로 되어 있는 셀들을 어떻게 인덱싱하느냐이다



R=2, *C*=3

- 같은 알파벳은 같은 열에, 같은 숫자는 같은 행에 존재한다.

  즉, 숫자가 증가할 때마다 인덱스는 1씩 증가하고 알파벳이 커질 때마다 인덱스는 *R*씩 증가한다.

- 따라서 셀의 좌표가 **(*i*, *j*)의 인덱스 = (*j* - 1) × *R* + *i*** 로 구할 수 있다.

G. 전파와 병합 2

출제자: 하주헌(hjhj97)

- 인덱싱을 완료했으면 순환참조(cycle)를 찾아야 한다. cycle을 찾기 위해서 SCC를 검출할 수 있는 알고리즘(Tarjan's 또는 Kosaraju's algorithm)을 사용하거나, DFS로 순회하면서 visited 배열로 체크하는 방법도 가능하다.
- 시간복잡도 : $O(N)$ $(N = R \times C)$

G. 전파와 병합 1

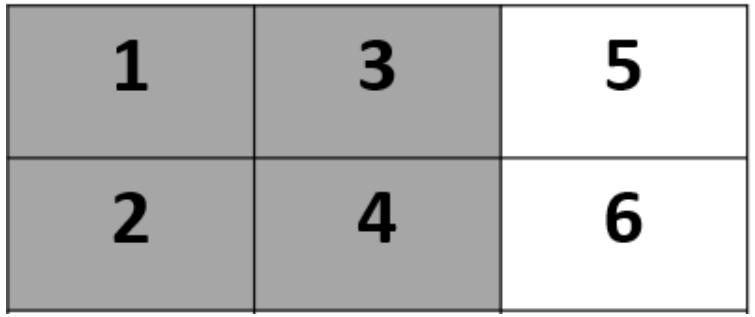
출제자: 하주헌(hjhj97)

- 전파와 병합 1을 풀기 위해서는 SCC를 필수적으로 찾아야 한다.
- SCC까지 찾았으면 해당 노드를 참조하는 다른 노드들에게 모두 전파시킨다.
- 전파시킨 뒤에, SCC가 직사각형인지 검증해야 한다. 그러기 위해서 SCC를 이루는 원소 중에서 가장 작은 인덱스와 가장 큰 인덱스를 찾은 다음, 두 인덱스가 시트 상에서 몇 행 몇 열 인지 찾는다.
- 가장 작은 인덱스는 시트 상에서 직사각형의 가장 왼쪽 위, 가장 큰 인덱스는 가장 오른쪽 아래에 위치해야 한다.

G. 전파와 병합 1

출제자: 하주헌(hjhj97)

- 예를 들어 아래와 같은 그림의 경우, 가장 작은 인덱스는 1이고 위치는 (1,1), 가장 큰 인덱스는 4이고 위치는(2,2)이다.



- 이제 가장 왼쪽 위 ~ 가장 오른쪽 아래로 2중 for문을 돌면서 해당 위치의 인덱스가 실제 SCC를 이루는 노드의 인덱스에도 존재하는지 확인한다.

  위 그림의 예시대로라면 [1,1], [2,1], [1,2], [2,2]를 돌면서 (1,2,3,4)가 모두 존재하는지 확인한다.

- 만약 모두 존재한다면 '완전한 직사각형'을 이루는 것이다.

G. 전파와 병합 1

출제자: 하주헌(hjhj97)

- 이제 병합을 한 뒤에, 병합된 셀이 유효하지 않은 셀을 하나도 참조하지 않는다면 자신은 유효한 셀로 변하게 된다.
- 하지만 이를 확인하는 과정이 매우 번거롭다. 과정의 순서를 바꿔서 단순하게 생각해보자. 만약 병합을 먼저 한 뒤에, 전파를 하면 어떻게 될까?
- 전파하기 전에 병합하면, 모든 직사각형 순환참조는 유효한 셀이다.
- 그 뒤에 전파를 하면, 유효하지 않은 셀이 자신을 참조하고 있던 직사각형들을 모두 전파시킨다. 전파시킨 뒤에도 유효한 상태로 남아있는 직사각형은 유효하지 않은 셀을 참조하지 않았다는 뜻이다.
- 이제 유효한 셀을 다시 초기에 입력받았던 알파벳+정수 형태로 변환하여 출력한다. 변환과정은 맨 처음 인덱싱 과정의 역연산이다.

시간복잡도 : $O(NlogN)$ $(N = R \times C)$

H. 화질 - 자동 (480p)

Div. 1		Div. 2	
전체 제출 :	6	전체 제출 :	0
맞은 사람 :	0	맞은 사람 :	0
정답률 :	0%	정답률 :	0%

H. 화질 – 자동 (480p)

출제자: 신상민(qilip)

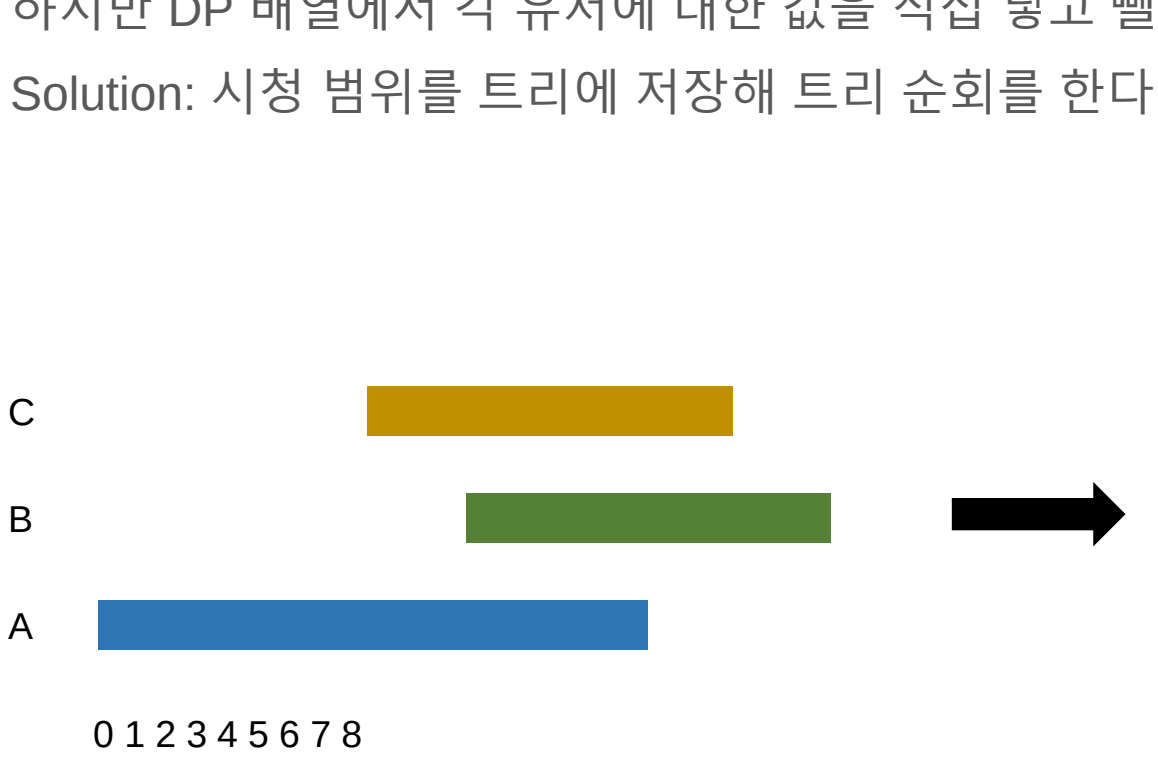
- 매 시간마다 시청중인 시청자들의 만족도 합을 최대화 시켜야 한다.
- 시청자가 시청을 끝내거나 새로 시작할 때만 최적의 만족도 합을 다시 구하면 된다.
- 0-1 Knapsack problem으로 볼 수 있으므로 최적의 만족도 합은 DP로 구할 수 있다.
- DP[i] = 서버의 대역폭이 i일 때 시청중인 시청자들의 최대 만족도
- 각 시청자 $1 \sim N$, 서버 대역폭 $1 \sim W$, 화질 종류 $1 \sim 6$에 대해 반복한다.
- 해당 DP를 모든 시청자가 시청을 시작하거나 끝낼 때마다 해야 하므로 $P(=2N)$ 회 수행한다.
- 이 방법으로는 시간복잡도가 $O(NW6P) \approx O(N^3)$ 이므로 시간초과가 발생한다!

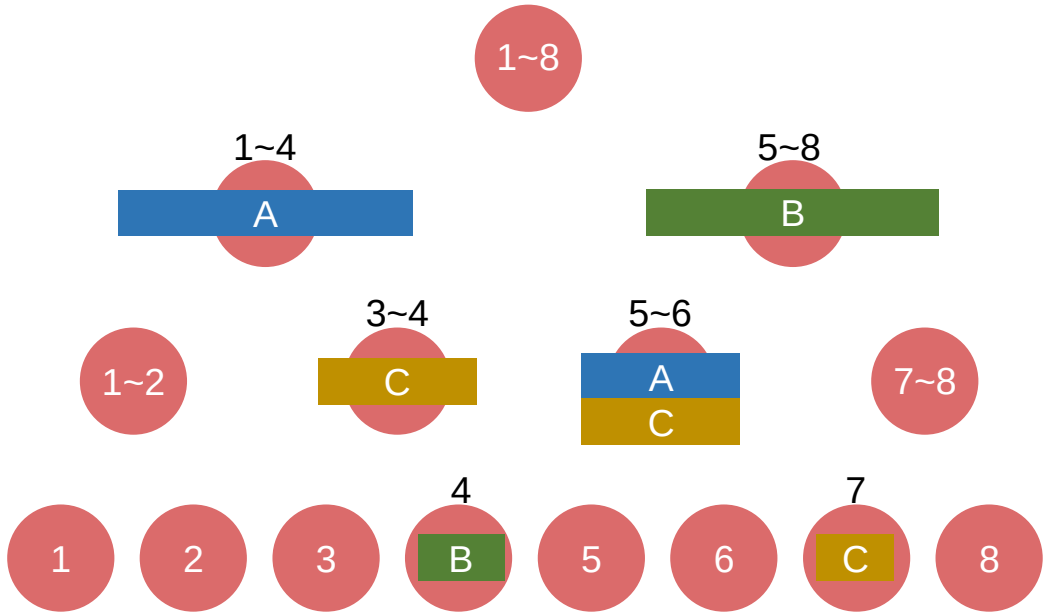
```
for(auto 시청자 : 현재_시청중인_시청자들){ // 전체 시청자에 대해 반복
    for(int i=w;i>=0;i--){ // dp[i] = 대역폭 i일때 최대 만족도
        for(int j=1;j<=6;j++){ // 화질[j] = 화질 j의 대역폭 사용량
            if(i+화질[j]<=w && dp[i]>-1){
                dp[i+화질[j]] = max(dp[i+화질[j]], dp[i] + 만족도[시청자][j]);
                전체_최대값 = max(전체_최대값, dp[i+화질[j]]);
            }
        }
    }
}
```

H. 화질 – 자동 (480p)

출제자: 신상민(qilip)

- Idea: 시청을 시작하거나 끝내는 지점에서는 시청중인 시청자들의 일부만 변한다. 이럴 때마다 전체 시청자에 대해 DP를 수행 할 필요가 있을까?
- 변화하는 시청자에 대해서만 DP를 수행하면 시간을 아낄 수 있다.
- 하지만 DP 배열에서 각 유저에 대한 값을 직접 넣고 뺄 효율적인 방법이 필요하다.
- Solution: 시청 범위를 트리에 저장해 트리 순회를 한다.





H. 화질 – 자동 (480p)

출제자: 신상민(qilip)

- 세그먼트 트리와 유사한 아이디어로 트리의 부모가 자식의 범위를 가지게 한다.
- 각 트리는 자신 범위에 완전히 속하는 시청자 목록을 가진다.
- 각 시청 범위는 최대 log만큼 나뉘어 저장된다.
- 루트부터 왼쪽을 우선한 DFS 순회한다.
- 트리의 자식으로 내려갈 때는 새 유저를 DP에 더한다.
- 트리의 부모로 돌아갈 때는 부모의 DP값으로 되돌린다.
- 트리에 DP 배열을 저장하거나, 스택을 이용.
- 총 시간복잡도: $O(NW6\log P) \approx O(N^2 \log N)$