Im4u 봄방학 캠프
DAY 4; Discrete Optimization Problems #4
Dynamic Programming (II)

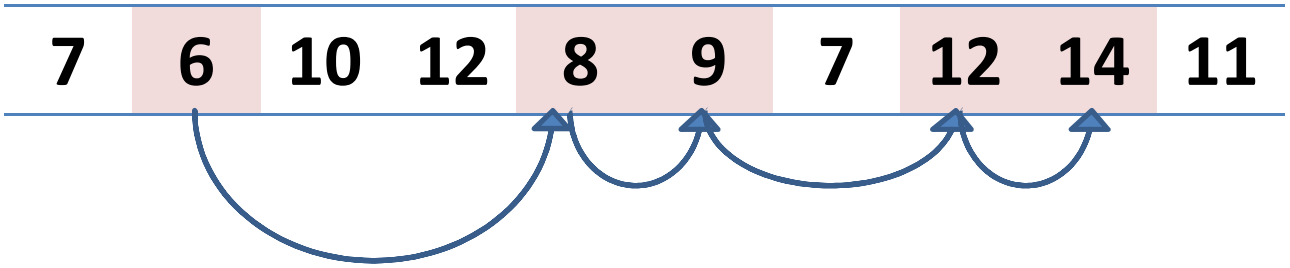
구종만
jongman@gmail.com

오늘 할 얘기

- 최적화 문제의 답안 생성하기
- 다른 형태의 동적 계획법 문제들
  - 계산 게임
  - 지수 크기 상태공간을 갖는 문제들
  - 선형변환 형태의 점화식을 갖는 문제들

- 이론적 배경
  - Optimal Substructure (.. 하지만 슬라이드 50장 돌파해서 이건 생략)

Longest Increasing Subsequence



- 단조 증가하는 부분 수열을 구하고 싶다
- 일단은 해당 수열의 '길이' 만을 찾아봅시다
  - $O(n^2)$ 의 답을 찾아봅시다

Think-Path

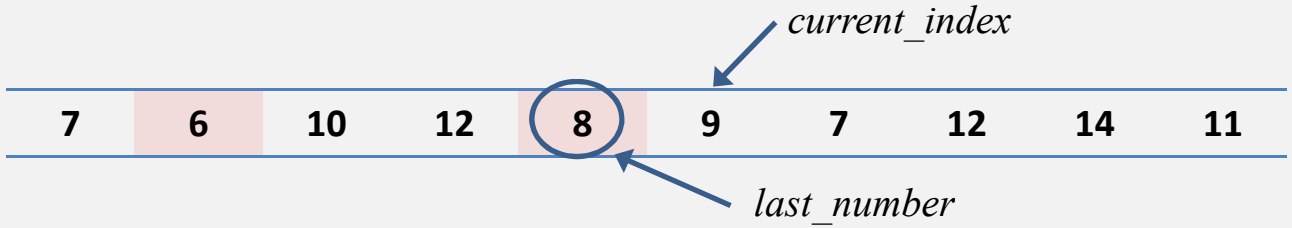
- 지금까지 본 문제와는 스타일이 약간 달라요~
  - 답이 어디서 시작할지, 끝날지를 모름
- 당장 동적 계획법으로 찾으려면 막막할 수 있음
- “늘 그렇듯이” Brute-Force 에서 시작해 봅시다

- 어떤 재귀호출 방법을 써야 모든 답안을 다 만들어 볼 수 있을까요?

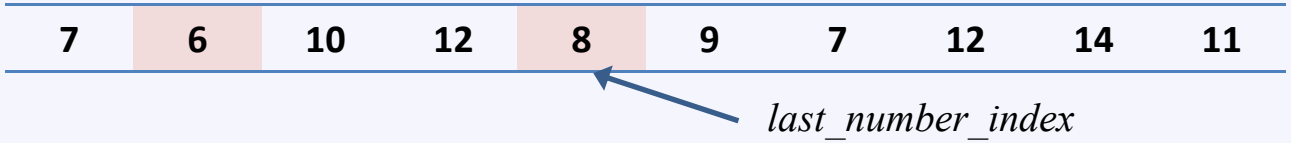
모든 답안을 만들어 보자

- 두 가지 방법이 있을 듯:

$try\_subsequence(last\_number, current\_index)$



$try\_subsequence(last\_number\_index)$



- 어느 쪽이 더 동적 계획법에 적합해 보이나? 왜?

당연히 두 번째죠

- 첫 번째 점화식을 동적 계획법으로 바꾸려면 주어지는 숫자의 크기에 비례하는 배열이 있어야 하죠
  - 다른 테크닉으로 이 방법을 사용할 수도 있지만 여전히 더 느립니다.
- 두 번째 점화식은 $O(n^2)$ 에 구현 가능 ^_^
  - $C_i = i$ 번째 수에서 시작하는 LIS 의 길이
  - $C_i = \max_{j=i+1, A_j > A_i}^{n-1} C_j + 1$
- 답은? $\max C_i$

구현해 봅시다

```
int n, A[100];
 int cache[100];

 int go(int here) {
    int& ret = cache[here];
    if(ret != -1) return ret;
    ret = 1;
    for(int there = here+1; there < n; ++there)
       if(A[here] < A[there])
          ret = max(ret, go(there) + 1);
    return ret;
 }

 int ret = 0;
 for(int i = 0; i < n; ++i) ret = max(ret, go(i));
```

- 처음으로 C++ 의 reference variable 을 쓴 코드
  - (이 테크닉 꽤나 유용합니다)

실제 해 계산하기

- 길이를 구하기는 쉬웠는데, 실제 수열을 출력하라면 어떻게 해야 하나?
- 방법1: 길이를 저장하는 대신, 수열을 직접 저장

```
int n, A[100];
vector<int> cache[100];

vector<int> go(int here) {
   vector<int>& ret = cache[here];
   if(!ret.empty()) return ret;
   ret.push_back(A[here]);
   for(int there = here+1; there < n; ++there)
      if(A[here] < A[there]) {
        vector<int>& cand = go(there);
         if(cand.size() + 1 > ret.size()) {
            ret.clear();
            ret.push_back(A[here]);
            ret.insert( cand.begin(), cand.end() );
         }
      }
   return ret;
}
```

재귀적으로 해 계산하기

- $C_{14} = \max C_i = 14$ 였다고 하자.
- 이 값은 어떻게 14 가 되었을까?
- 14 오른쪽에 $A_j > A_{14}$ 이고, $C_j = 14$ 인 j 가 존재

A	12	17	10	4	..	..	..	> 4	..	..
C	8	6	9	14	..	..	..	13	..	..

- 이것을 찾는 일을 반복하자

반복적 (Iterative) 으로 찾기

```
int n, A[100];
int cache[100];
int bestIndex = 0;

for(int i = 1; i < n; ++i)
    if(go(bestIndex) < go(i)) bestIndex = i;

do {
    printf("%d ", A[bestIndex]);
    for(int i = bestIndex+1; i < n; ++i)
        if(go(i) + 1 == go(bestIdex) && A[bestIndex] < A[i]) {
            bestIndex = i;
            break;
        }
}
while(A[bestIndex] > 1);

printf("\n");
```

- 반복적으로 다음 위치를 찾아 나간다

재귀적으로 찾기

```
int n, A[100];
int cache[100];
void printBest(int here) {
   printf("%d ", A[here]);
   if(cache[here] > 1)
      for(int there = here+1; there < N; ++there)
         if(cache[there]+1 == cache[here] && A[there] > A[here]) {
            printBest(there);
            break;
         }
}

for(int i = 1; i < n; ++i)
     if(go(bestIndex) < go(i)) bestIndex = i;
printBest(bestIndex);
```

- 경험적으로는 재귀적으로 짜는 편이 좀 더 좋다

Lexicographically Smallest Solution

- “주어진 수열의 LIS 중, 사전 순으로 가장 앞에 있는 것을 계산하라”

5	6	7	8	9	1	2	3	4	5

Lexicographically Smallest Solution

- "주어진 중복이 없는 수열의 LIS 중, 사전 순으로 가장 앞에 있는 것을 계산하라"

6	7	8	9	10	1	2	3	4	5

- $C_i = \max C$ 인 i 가 하나라면 문제없다.
- 여러 개라면 어떻게 할까?
  - 숫자가 (클/작을)수록 좋다!

Lexicographically Smallest Solution

- "주어진 중복이 없는 수열의 LIS 중, 사전 순으로 가장 앞에 있는 것을 계산하라"

6	7	8	9	10	1	2	3	4	5

- $C_i = \max C$ 인 i 가 하나라면 문제없다.
- 여러 개라면 어떻게 할까?
  - 숫자가 (클/작을)수록 좋다!

K-th Smallest Solution

- “주어진 수열의 LIS 중, 사전 순으로 k번째에 있는 것을 계산하라”

K=3

1	5	2	6	3	7	4	8	5	9

- 답의 수는 감당하기 힘들 정도로 커질 수 있다.

2	1	4	3	6	5	8	7	10	9

  - 따라서, Brute-Force 로는 안 된다!

또 한 번의 동적 계획법

- 가장 좋은 답의 '길이' 말고도, 가장 좋은 답을 만들 수 있는 방법의 수 또한 별도로 계산해서 저장

- $C_i = i$ 번째 수에서 시작하는 LIS 의 길이

  $D_i = i$ 번째 수에서 시작하는 LIS 의 수

$$= \sum_{j=i+1, A_j > A_i, C_j + 1 == C_i}^{n-1} D_j$$

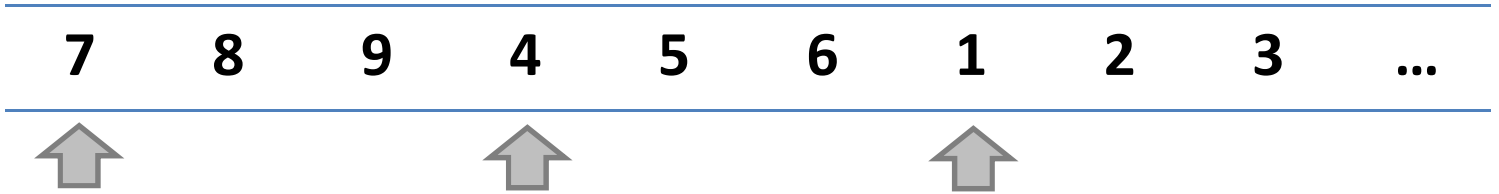
- 이걸 가지고 어떻게 답을 만들 수 있을까?

생각하기 나름이겠지만

- 전 이 방법이 가장 쉽더군요
  - K번째 답 (1-based)
  - 그 앞에 k-1개의 답이 있는 것 중 사전순으로 가장 앞에 있는 답 (0-based)

- int skip = k – 1;

그렇게 생각하면…



- $C_i = \max C$ 인 i 가 여러 개 있다고 합시다
- 이들을 줄 세워 보면…
- 1 에서 시작하는 것들이 맨 앞에
- 7 에서 시작하는 것들이 맨 뒤에
  - 가겠지요..

```
1 .. .. .. .. ..
1 .. .. .. .. ..
...
4 .. .. .. .. ..
4 .. .. .. .. ..
...
7 .. .. .. .. ..
7 .. .. .. .. ..
```

첫 번째 숫자만 맞춰 봅시다

A[]	7	8	9	4	5	6	1	2	3	…
D[]	7			12			15			…

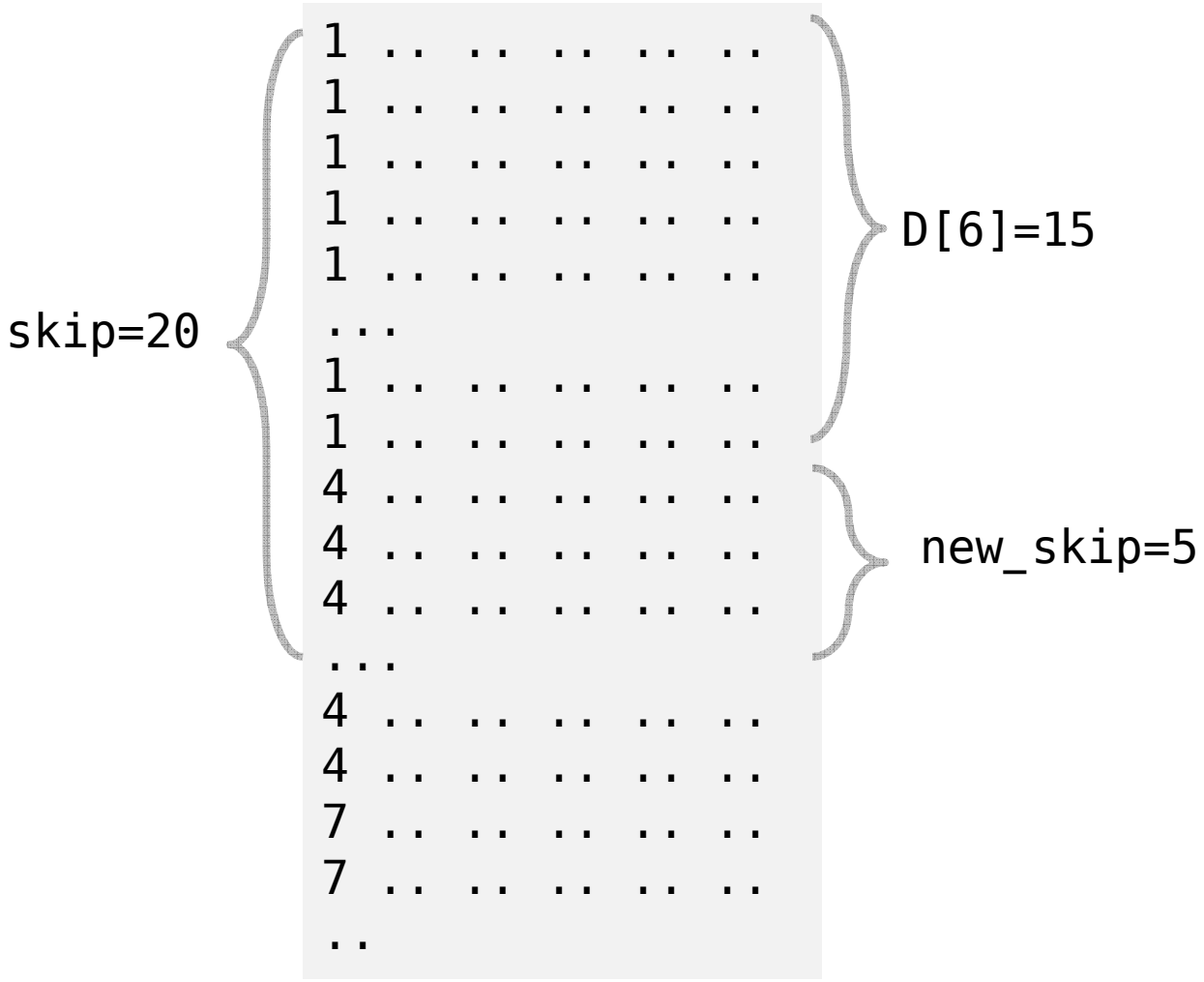
- If skip = 0? ____
- If skip = 20? ____
- If skip = 30? ____
- If skip = 15 ? ____

첫 번째 숫자만 맞춰 봅시다

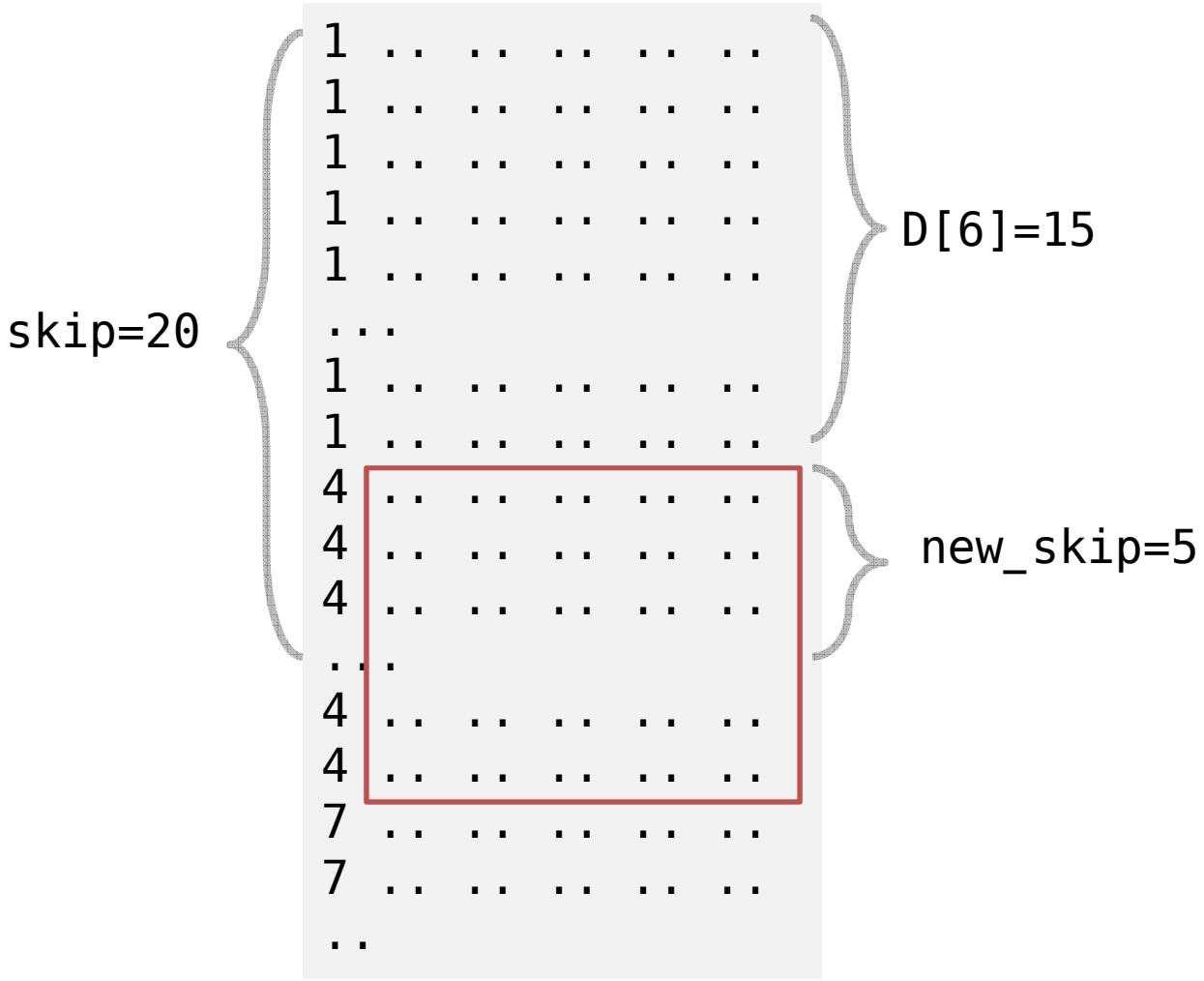
A[]	7	8	9	4	5	6	1	2	3	…
D[]	7			12			15			…

- If skip = 0? 1
- If skip = 20? 4
- If skip = 30? 7
- If skip = 15 ? 4

Skip=20 의 예



Skip=20 의 예



코드

```
int n, A[100], C[100], D[100];

void printAfterSkip(int begin, int skip) {
   printf("%d ", A[begin]);
   if(C[begin] == 1) return;

   // (A[index], index) 의 쌍
   vector<pair<int, int> > nextCandidates;
   for(int i = begin+1; i < n; ++i)
      if(A[begin] < A[i] && C[i]+1 == C[begin])
         nextCandidates.push_back(make_pair(A[i], i));
   sort(nextCandidates.begin(), nextCandidates.end());

   for(int i = 0; i < nextCandidates.size(); ++i)
      if(skip >= D[nextCandidates[i].second)
         skip -= D[nextCandidates[i].second];
      else {
         printAfterSkip(nextCandidates[i].second, skip);
         break;
      }
}
```

TheDictionary

- 알파벳 a 와 z 만으로 구성된 문자열들
  - a 의 개수는 n 개, z 의 개수는 m 개라고 하자.
  - 이들을 사전순으로 정렬했을 경우, k번째에 오는 문자열은 무엇일까?

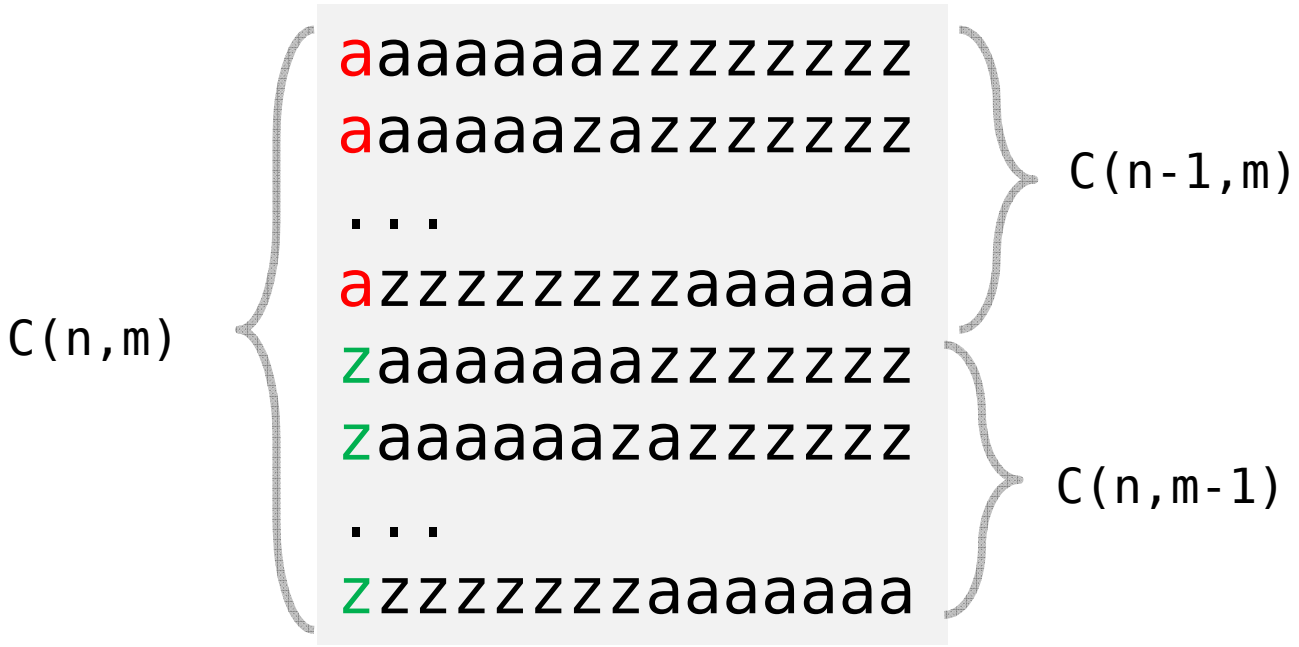
Counting

- ~~이산구조 시간에 배우셨겠죠?~~
- C(n,m) = nCm = 이항계수 (binomial coefficient)

```
aazz
azaz
azza
zaaz
zaza
zzaa
```

- C(n,m) = C(n-1,m-1) + C(n-1,m)

첫 글자를 결정해보자


C(n,m)
aaaaaaazzzzzzzz
aaaaaazazzzzzzz
...
azzzzzzzzaaaaaa
zaaaaaaazzzzzzz
zaaaaaazazzzzzz
...
zzzzzzzzaaaaaaa
C(n-1,m)
C(n,m-1)

배스킨 라빈스 31

- 두 명의 사람이 배스킨 라빈스 31을 한다.
  - 한 사람은 한 개~여섯 개의 숫자를 부를 수 있다.
  - 한 게임에서 같은 숫자는 최대 네 번만 부를 수 있다.
- 게임의 현재 진행 상황이 주어진다.
  - 1123656
- 양쪽이 '완벽하게' 플레이한다고 할 때, 이 게임의 승자는?

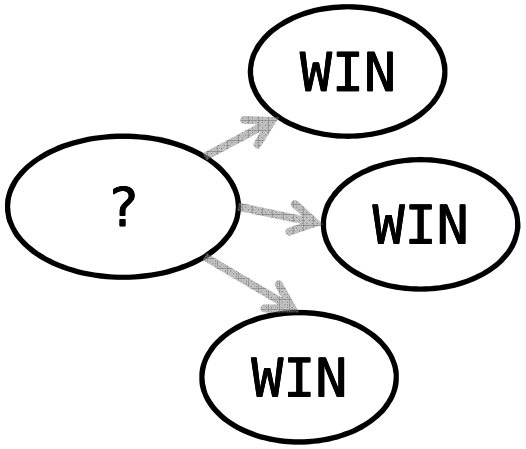
‘완벽하게’ 의 정의

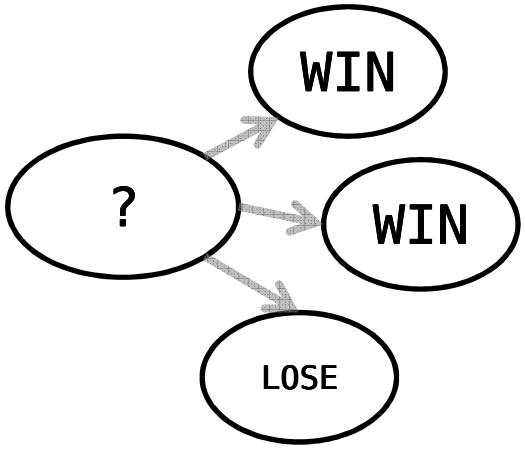
- 상대방이 어떻게 플레이하더라도, 내가 적절히 반응하면 승리할 수 있다 => 승리

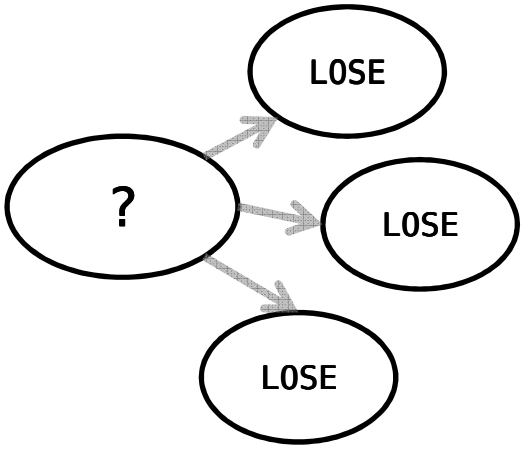
- 내가 어떻게 플레이하더라도, 상대방이 적절히 반응해 나를 지게 만들 수 있다 => 패배

Recurrence

- 게임의 "상태"가 주어질 때, 게임의 승패를 결정하는 함수를 정의하고, 이 함수를 점화식으로 나타낸다
  - F(state) = 현재 상태에서, 이번 차례인 사람이 반드시 승리할 수 있는 방법이 있는가?







배스킨 라빈스 31: 상태공간

- Key Observation:
  - 각 숫자가 어떤 _____ 로 나왔는지는 상관이 없다
  - 각 숫자가 나온 _____ 만이 중요하다

배스킨 라빈스 31: 상태공간

- Key Observation:
  - 각 숫자가 어떤 _순서_ 로 나왔는지는 상관이 없다
  - 각 숫자가 나온 _횟수_ 만이 중요하다
- 따라서, 각 숫자마다 0,1,2,3,4 의 상태:
  - 5^6 = 15,625 개의 상태
  - Quite manageable!
- 각 상태마다 할 수 있는 행동을 다 해 본다
  - 어떤 행동을 해도 내가 진다면 패배, 하나라도 이긴다면 승리

배스킨 라빈스 31: 코드

```
map<vector<int>, int> cache;

int willWin(vector<int>& used, int total) {
   if(total >= 31) return 1;
   if(cache.count(used)) return cache[used];
   int& ret = cache[used];
   for(int i = 1; i <= 6; ++i)
      if(used[i-1] < 4) {
         used[i-1]++;
         if(!willWin(used, total+i))
            ret = 1;
         used[i-1]--;
      }
   return ret;
}
```

- std::map 을 이용한 간단한 memoization!

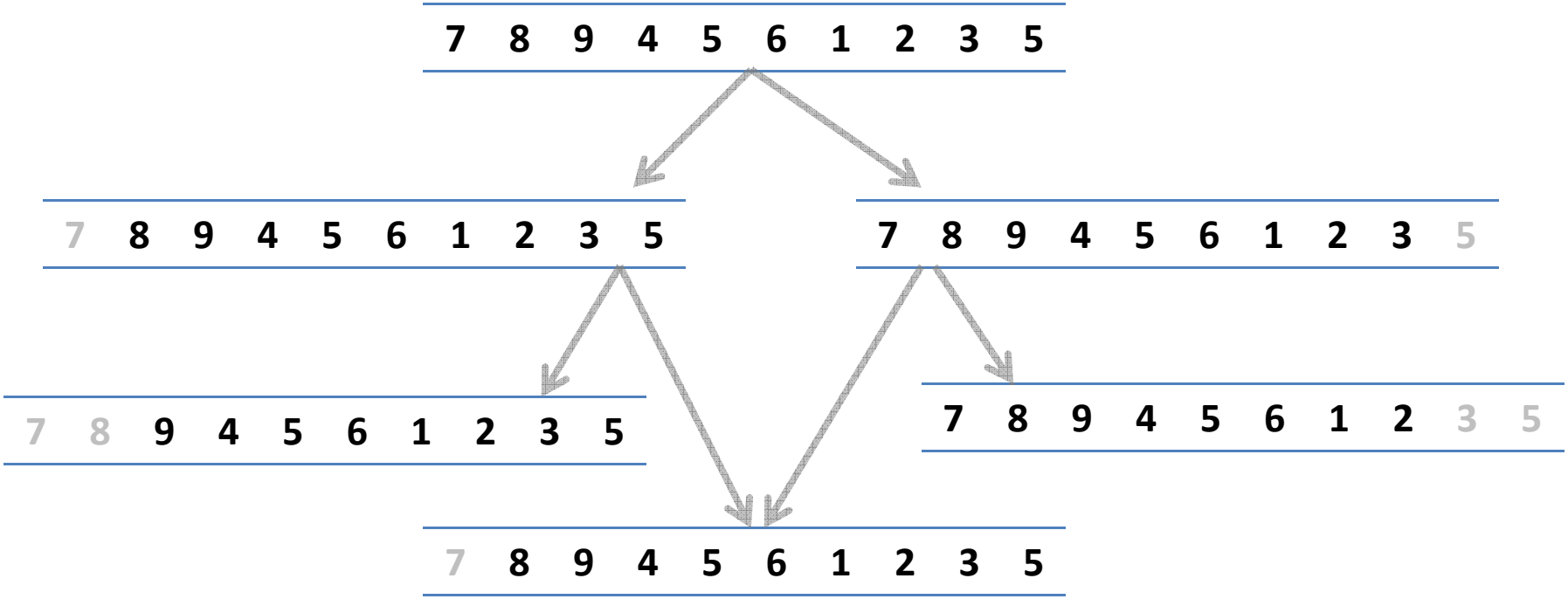
A Game

- 짝수 길이의 게임판, 두 사람의 플레이어

7	8	9	4	5	6	1	2	3	5

- 왼쪽 끝이나 오른쪽 끝의 숫자를 번갈아 가며 먹는다
  - 먹은 수의 총합이 해당 플레이어의 점수
  - (내 점수) – (상대 점수) 를 최대화할 수 있는 방법은?

A Game: 상태공간



- 게임의 상태는 왼쪽 끝 칸과 오른쪽 끝 칸으로 정의할 수 있다: $O(n^2)$ 개의 상태

Minimax Algorithm

- $f(state) =$ 현재 상태에서 첫 번째 플레이어의 점수
  - 두 번째 플레이어의 점수 = $-f(state)$

- 따라서, 우리 점수를 최대화하려면 상대의 점수를 최소화해야 한다

$$f(state) = \left[\max_{nxt \in next(state)} -f(nxt) + Cost(nxt) \right]$$

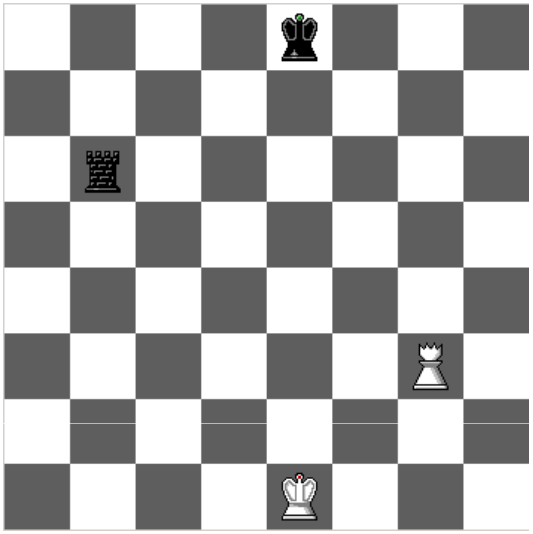
A Game: 코드

```
int n, A[100];
int cache[100][100];

int go(int left, int right) {
   if(left > right) return 0;
   int& ret = c[left][right]
   if(ret != -1) return ret;
   return ret = max(
      -go(left+1, right) + A[left],
      -go(left, right-1) + A[right]
   );
}
```

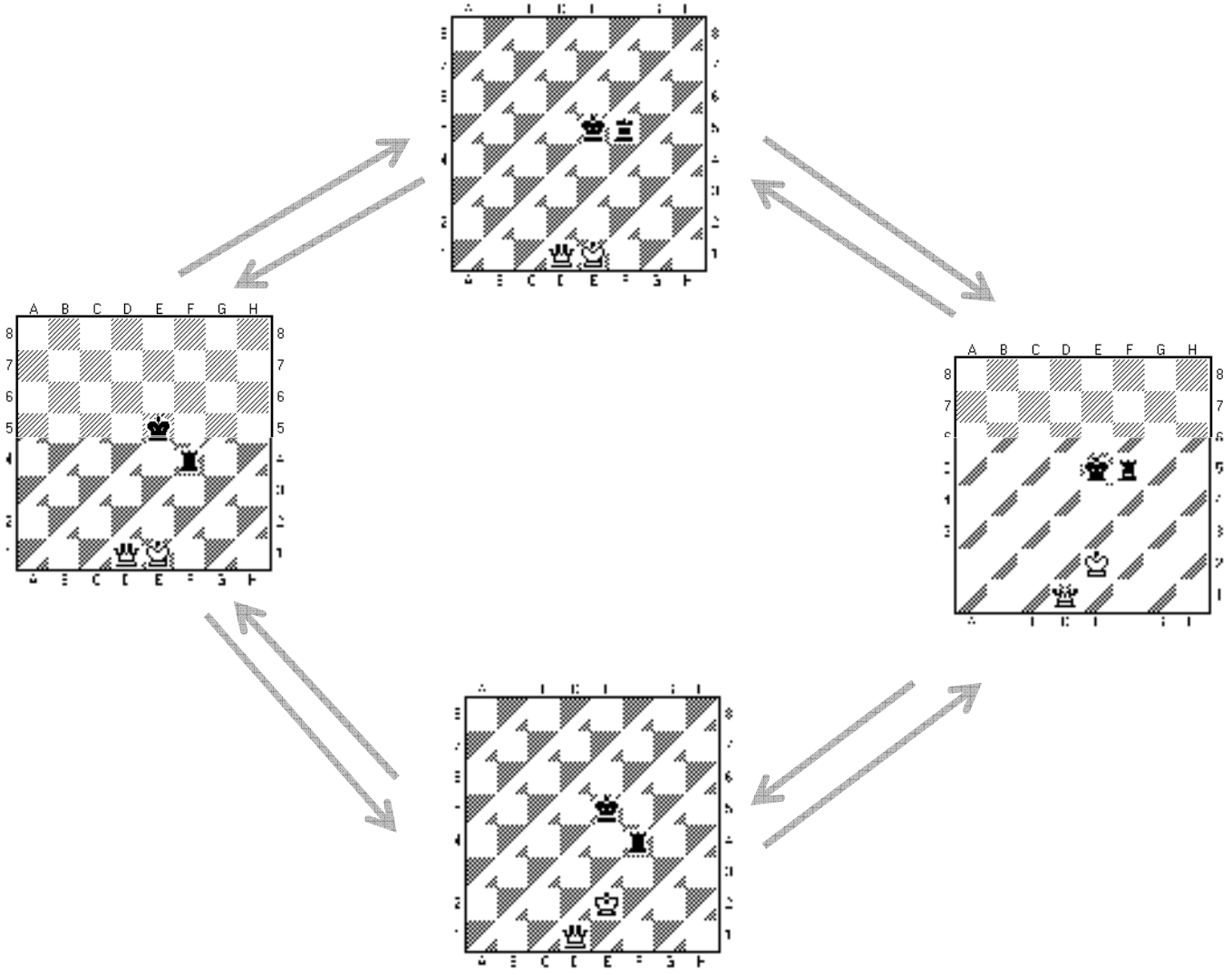
- go(left+1, right), go(left, right+1) 는 상대방의 점수

DP 가 만능은 아니다



- Chess Endgame analysis
  - King + Rook vs. King + Queen
- 어느 쪽이 이길 것인가?

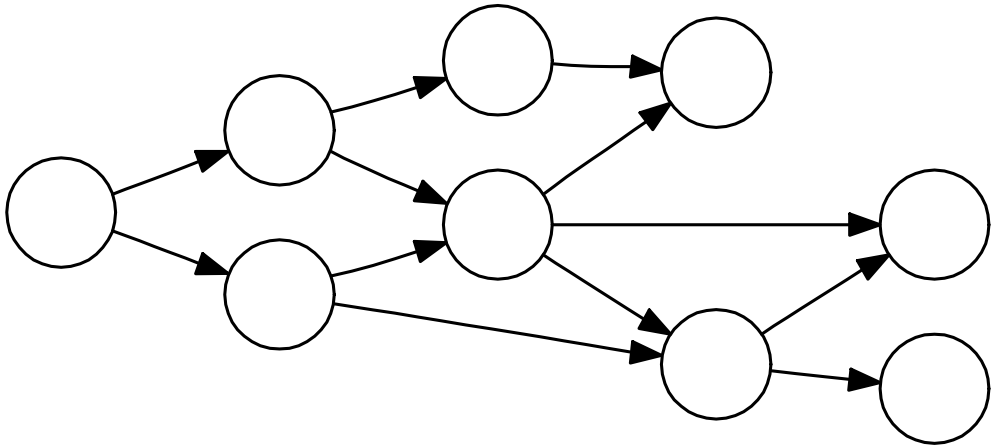
Cycles in the Game State



State Space MUST Be A DAG

- 게임의 상태들로 구성된 공간에 사이클이 있으면 memoization 을 사용할 수 없다
  - 이 경우에는 다른 방법을 사용해야 함 (생략)

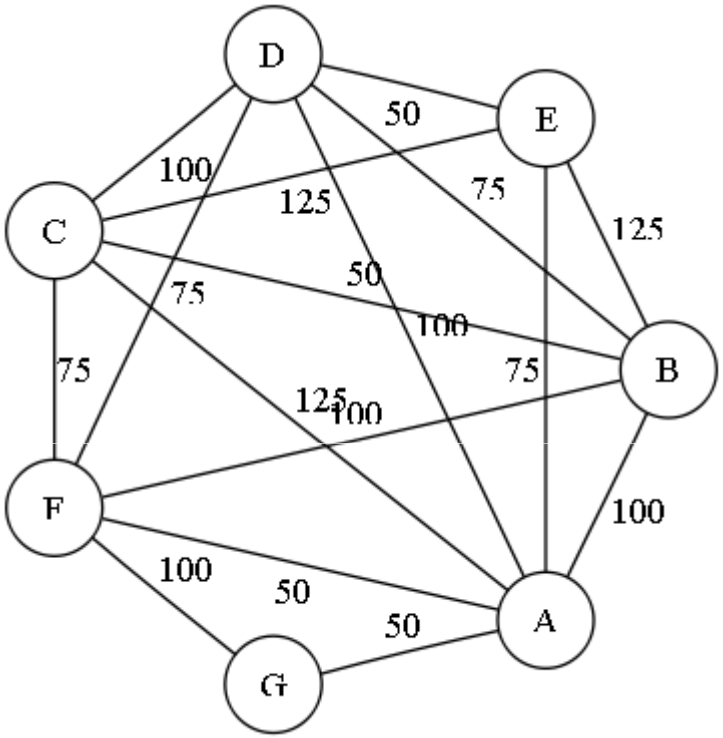
- 사이클이 없는 그래프 (Directed Acyclic Graph)



Bit of Combinatorial Game Theory

- Impartial game
  - 두 플레이어가 할 수 있는 일은 완전히 똑같고, 현재 게임판의 상태에만 좌우된다
  - 체스나 바둑은 Impartial Game 이 아니다
- Normal game
  - 마지막으로 수를 둘 수 있는 플레이어가 이기는 경우
  - Game of NIM
- Impartial Normal Game 은 수학적으로 훨씬 효율적으로 풀 수 있다 (여기선 생략)

Traveling Salesman Problem

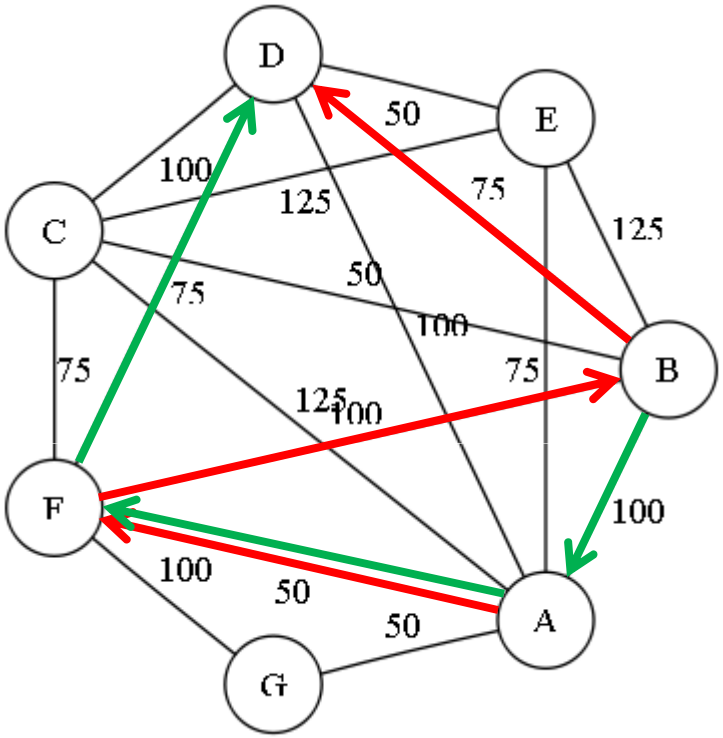


- 모든 도시를 1번씩 방문하는 경로 중 가장 짧은 것은?

Can We Brute-Force It?

- N 개의 도시에 대해 $N!$ 개의 경로가 있다.
- 20! = 2432902008176640000

Key Observation



- B-A-F-D, A-F-B-D 경로의 차이점은?
  - 최소한 앞으로는 없다!

Exponential State Space

- 어떤 경로의 현재 상태를
  - $state = \{\text{set of visited vertices}, \text{current vertex}\}$
  - 로 표현하자

- Current vertex: $O(V)$
- Set of visited vertex: $O(2^V)$

- $C(visited, here) = \min_{next \notin visited} C(visited \cup \{next\}, next) + Cost(here, next)$

Implementing Set of Visited Vertices

- std::bitset<20>
  - 20개의 비트 (0, 1) 을 저장하는 컨테이너
  - bitsetVariable[i] = i번 정점을 방문했는가?

- Integers w/ Bitmasks (far more popular)
  - 32비트 정수를 사용한다
  - i번 비트가 켜져 있는가 == i번 정점을 방문했는가?

$$372_{10} = \begin{array}{cccccccccc} 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ \hline \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{array}_2$$

C++ Bitmask Trick of Trades

```
int bitset;

// i 번째 비트가 켜져 있다면?
if(bitset & (1 << i))

// i 번째 비트를 켠다
bitset |= (1 << i);

// i 번째 비트를 끈다
bitset&= ~(1 << i);

// i 번째 비트를 토글
bitset ^= (1 << i);

// 모든 비트를 왼쪽으로 한 칸씩 옮긴다
bitset <<= 1;

// 마지막 비트를 끈다
bitset = (bitset - 1) & bitset;
```

- & : Bitwise AND
- | : Bitwise OR
- ^ : Bitwise XOR
- << : Shift left
- >> : Shift right

- 지수시간 상태공간 동적 계획법은 매우 자주 출현하는 주제이므로 유의합시다~

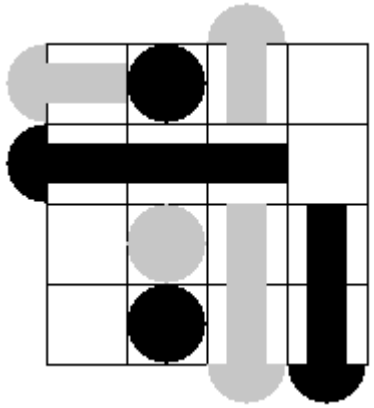
TSP: 코드

```
int n, graph[20][20];
int cache[20][1<<20];

int go(int here, int visited) {
   if(visited == (1<<n) - 1) return 0;
   int& ret = cache[here][visited];
   if(ret != -1) return ret;
   ret = 987654321;
   for(int next = 0; next < n; ++next) if((visited & (1 << next)) == 0)
      ret = min(ret, graph[here][next] + go(next, visited | (1 << next)));
   return ret;
}
```

Game Of Euler

- 4x4 판에 번갈아가며 길이 1~3의 핀을 꽂는다
  - 위에서 꽂을 수도 있고, 측면에서 꽂을 수도 있다
  - 더 꽂을 수 없을 때까지 계속
  - 마지막 플레이어가 진다 (misere form)
- 게임 중간의 상황이 주어질 때, 승자는?

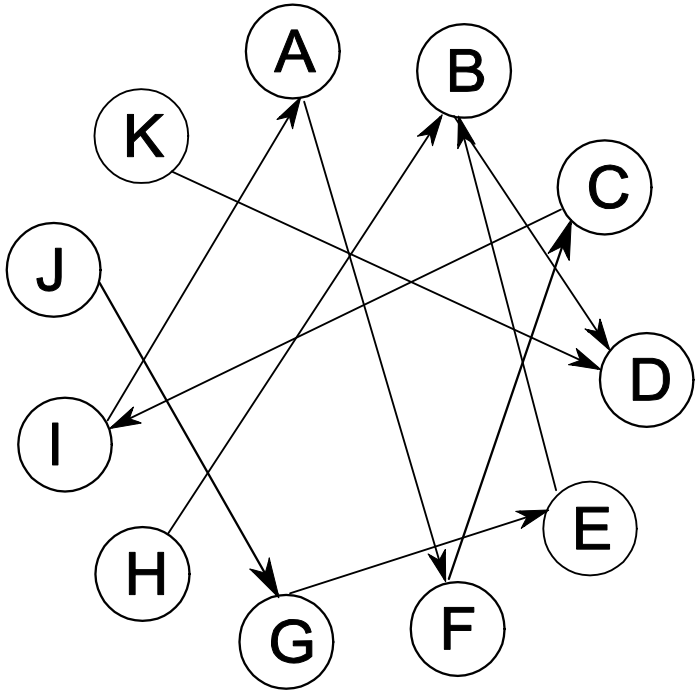


Key Observation

- 어떤 칸에 핀이 꽂혀 있다면, 어떤 형태의 핀으로 차 있던지 상관이 없다
  - 길이 3인 핀이 위에서 꽂혔던지
  - 길이 2인 핀이 옆에서 꽂혔던지
  - 길이 1인 핀이 위에서 꽂혔던지

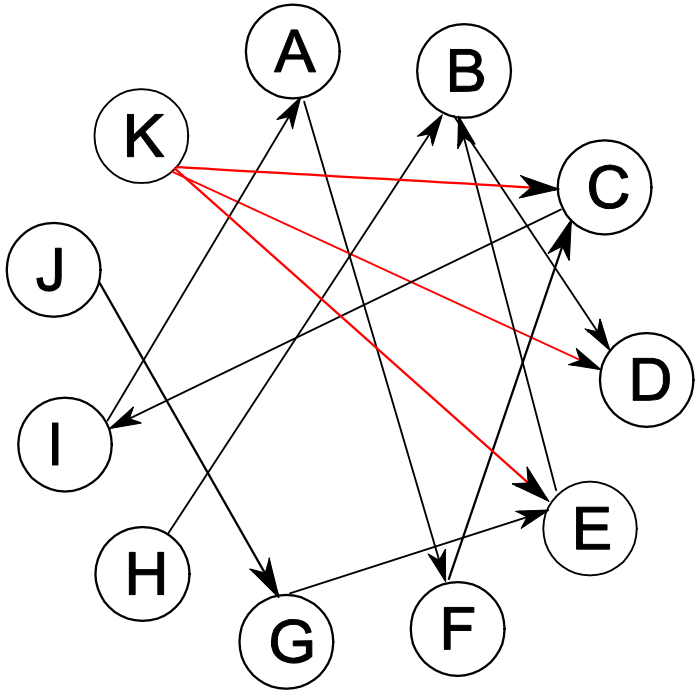
- 결국 $2^{4\times4} = 65536$ 개의 상태가 된다

The Game Of Death



- A 가 “일곱!” 을 외쳤다. 술을 먹을 사람은?

The Game Of Death



- A 도 꽤 취했기 때문에, A 가 본 것을 믿을 수는 없다
  - 왼쪽으로 한명/오른쪽으로 한명 (각각 1/3 확률)
  - G 가 술을 마실 확률은?

Recurrence

- $C(i, j) =$ i 개의 화살표를 따라간 뒤에 j 번째 사람일 확률

$$= \sum_{k \in \text{Points}(j)} \frac{C(i-1, k)}{3}$$

- O(NM) 동적 계획법..
- 하지만, $M \approx 2^{32}$ 라면 어떨까?

The Recurrence Is A Linear Transform

- $C_i(0) = \frac{C_{i-1}(3) + C_{i-1}(7) + C_{i-1}(8)}{3}$ $\quad C_i(1) = \frac{C_{i-1}(0) + C_{i-1}(3)}{3}$
- $C_i(2) = \frac{C_{i-1}(0) + C_{i-1}(1) + C_{i-1}(5) + C_{i-1}(7)}{3}$ ……
- 행렬로 바꿔서 써 보자!

$$C_i = [A] C_{i-1} \qquad C_M = [A]^M C_0$$

- 우와, Fast Matrix Exponentiation 이 되네요!

Lessons Learned

- 최적화 문제의 답안 계산하기
  - K-번째 답안 계산하기
- 계산 게임
  - 현재 상태에 대한 f(state) 의 점화식을 푼다
  - Minimax algorithm
  - 상태에 사이클이 있는 경우에는 DP 로 풀 수 없다
- Exponential State Space
  - Bitmask trick 들을 잘 알면 도움이 된다
- Problems w/ Linear Recurrence
  - Matrix exponentiation 으로 바꿔서 풀 수 있다