**Advanced Coding Practice HW Problem #1**

**20191599 송경호**

**A) Finding Celebrities**

**A-1) 조건 및 입력 형식 정의**

N명의 사람 중 celebrity C가 존재할 때, C를 제외한 모든 사람은 C를 알지만, C는 본인을 제외하고 단 한사람도 알지 않는다.

즉, N명의 사람 중 C는 단 한 명 존재하거나 존재하지 않는다. ∵ C1, C2가 존재할 때, 서로를 모르면서 서로를 안다는 것은 모순이다.

추가적으로, 사람 간의 아는지에 대한 관계를 나타내기 위해 이차원 배열 *know*를 사용한다. 아래 예시의 경우 4번째 사람이 celebrity이다.

|  |
| --- |
| bool know[N][N] = { {0,0,0,1}, {0,0,0,1}, {0,0,0,1}, {0,0,0,1} }; |

*know[i][j] = 1*은 사람 i가 사람 j를 안다는 것을 의미한다. 아래 함수를 이용한다.

|  |
| --- |
| bool knows(int i, int j) { return know[i][j]; } |

**A-2) 문제해결**

* ***know[X][Y] = 1*일 때, X는 Y를 알고 있으므로, X는 절대 celebrity가 될 수 없으며 Y는 celebrity가 될 수 있는 후보이다.**
* ***know[X][Y] = 0*일 때, X는 Y를 모르고 있으므로, Y는 절대 celebrity가 될 수 없으며 X는 celebrity가 될 수 있는 후보이다.**

위 두 항상 참인 명제를 이용한다.

또한 celebrity가 존재할 때 N명의 사람들 중에서 위의 조건을 이용해 celebrity가 될 수 없는 후보를 제거했을 때 마지막 남은 사람이 celebrity임을 이용한다.

Stack에 N명의 사람을 넣는다.

|  |
| --- |
| stack<int> st;  for (int i = 0; i < N; i++) st.push(i); |

stack에서 가장 위 두 사람 X, Y를 뽑는다. 위의 명제에 따르면 두 사람 중 하나만 celebrity의 후보가 될 수 있다. 후보인 사람을 *st*에 다시 넣고 후보가 아닌 사람은 제거한다. 이 과정을 마지막 후보만 남을 때 까지 반복한다.

|  |
| --- |
| while (st.size() != 1) {  X = st.top();  st.pop();  Y = st.top();  st.pop();  if (knows(X, Y)) st.push(Y);  else st.push(X);  } |

마지막 남은 후보 역시 celebrity가 아닐 수 있다. 마지막 남은 후보에 대해 *know*를 확인하며 검증한다.

|  |
| --- |
| int candidate = st.top();  for (int i = 0; i < N && candidate != -1; i++) {  if (i == candidate) continue;  if (knows(i, candidate) && !knows(candidate, i)) continue;  else candidate = -1;  } |

**A-3) 시간, 공간복잡도 및 고려사항**

최초 문제해결 시, *know*배열 전체를 순회하며 그래프나 특정 형식의 자료구조를 만드는 방식을 통해 해결하니 시간복잡도는 항상 *O(n2)*이 나왔다. 따라서 *know*배열 전체를 순회하지 않는 알고리즘을 구상했다.

위 과정의 시간복잡도는 *O(n)*이다. “당신은 X를 아는가?”라는 질문이 연산이라고 할 때 연산 횟수를 구하는 것은 *knows*함수가 호출되는 횟수와 동일하다.

stack에 한 명의 후보만 남을 때 까지 제거 연산을 반복하므로 *n-1*, 마지막 후보의 검증 과정 2*n*을 더해 *3n-1*의 연산이 수행된다.

또한 stack에 N명을 넣으므로 공간복잡도 역시 *O(n)이다.*

마지막 남은 후보를 검증해야 되는 예시는 1 -> 2 -> 3 -> 1과 같이 사람 간의 관계가 원을 이룰 때, 마지막 후보 역시 첫 후보를 알게 되므로 celebrity가 아니다. 따라서 마지막 후보 역시 검증 과정을 거쳐야 한다.

**B) Minimum Calculation**

**B-1) 문제해결**

**i) Naive Approach**

|  |
| --- |
| ll result = 1;  for (int i = 0; i < n; i++)  result \*= a; |

단순히 반복문을 통해서 제곱한다.

**ii) Divide & Conquer Approach**

*an*의 결과를 구하는 함수가 *pow(a, n)*이라고 했을 때,

* + **n이 짝수이면, *pow(a,n) = pow(a,n/2) \* pow(a,n/2)***
  + **n이 홀수이면, *pow(a,n) = pow(a,n/2) \* pow(a,n/2) \* a***

임을 이용한다.

재귀호출을 통해, n이 1일 때를 base case로 두고 n을 계속해서 2로 나눈다.

|  |
| --- |
| ll pow(int base, int powN) {  if (powN == 0) return 1;  ll halfPow = pow(base, powN / 2);  if (powN % 2 == 0)  return halfPow \* halfPow;  else  return halfPow \* halfPow \* base;  } |

**iii) Bitwise Operator Approach**

an에 대하여, n을 이진수로 나타내어 분해할 수 있음을 이용한다. 예를 들어 210=28 x 22이며 10 = 1010b임을 알 수 있다.

n을 >> 연산과 & 연산을 통해 모든 비트를 확인한다. 이때 특정 자리의 비트가 1인 경우 그에 맞는 수를 곱한다. 그에 맞는 수는 >> 연산을 수행할 때마다, base인 a를 제곱하여 구할 수 있다.

해당 연산을 n이 0이 될 때까지 수행한다.

|  |
| --- |
| ll pow(int base, int powN) {  ll result = 1;  while (powN != 0) {  if ((powN & 1) == 1) result \*= base;  base = base \* base;  powN >>= 1;  }  return result;  } |

**B-2) 시간, 공간복잡도 및 고려사항**

**i) Naive Approach**

n에 대하여 O(n)의 시간복잡도를 갖는다.

**ii) Divide & Conquer Approach**

n이 0이 될 때 까지 n/2 연산을 수행하므로 O(logn)의 시간복잡도를 갖는다. 이 때 주의할 점은

|  |
| --- |
| ll halfPow = pow(base, powN / 2); |

을 통해 미리 *halfPow*를 구해서 중복된 연산을 줄여야 한다.

추가적으로, 해당 방식은 재귀 호출을 사용하기 때문에 추가적인 호출 스택이 사용되어 성능에 악영향을 미칠 수 있다.

**iii) Bitwise Operator Approach**

n에 대하여 모든 비트를 확인는 연산 횟수는 logn이므로 O(logn)의 시간복잡도를 갖는다. 비트 연산을 이용해 연산 속도도 상대적으로 빠르며 추가적인 호출 스택을 이용하지 않는다.

**C) Source Code**

**편의를 위해 첨부한 소스코드입니다. 채점 시 확인하실 필요없습니다.**

**A)**

|  |
| --- |
| #include <iostream>  #include <stack>  #include <string>  using namespace std;  #define N 4  bool know[N][N] = { {0,1,0,1}, {0,1,0,0}, {0,1,1,1}, {0,1,0,0} };  bool knows(int i, int j) { return know[i][j]; }  int main() {  ios\_base::sync\_with\_stdio(false);  cin.tie(NULL);  stack<int> st;  for (int i = 0; i < N; i++) st.push(i);  int X, Y;  while (st.size() != 1) {  X = st.top();  st.pop();  Y = st.top();  st.pop();  if (knows(X, Y)) st.push(Y);  else st.push(X);  }    int candidate = st.top();  for (int i = 0; i < N && candidate != -1; i++) {  if (i == candidate) continue;  if (knows(i, candidate) && !knows(candidate, i)) continue;  else candidate = -1;  }  if (candidate == -1) cout << "No Celebrity" << "\n";  else cout << "Celebrity Index : " + to\_string(candidate) << "\n";  return 0;  } |

**B-ii)**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  typedef long long ll;  ll pow(int base, int powN) {  if (powN == 0) return 1;  ll halfPow = pow(base, powN / 2);  if (powN % 2 == 0)  return halfPow \* halfPow;  else  return halfPow \* halfPow \* base;  }  int main() {  ios\_base::sync\_with\_stdio(false);  cin.tie(NULL);  int a = 2;  int n = 10;  ll result = pow(a,n);  cout << to\_string(a) + "^" + to\_string(n) + " = " + to\_string(result);  return 0;  } |

**B-iii)**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  typedef long long ll;  ll pow(int base, int powN) {  ll result = 1;  while (powN != 0) {  if ((powN & 1) == 1) result \*= base;  base = base \* base;  powN >>= 1;  }  return result;  }  int main() {  ios\_base::sync\_with\_stdio(false);  cin.tie(NULL);  int a = 2;  int n = 10;  ll result = pow(a,n);  cout << to\_string(a) + "^" + to\_string(n) + " = " + to\_string(result);  return 0;  } |