category

전체보기 (2699)

공지사항
이벤트 당첨자

그림방
그림방(from)
설정놀음

C
C++
C/C++ 예제
대회알고리즘

학업

백준저지(풀이) N

한줄글
혼자놀기!
코무합니다만
쉬어가는 곳
2차 창작 음악
레진코믹스
일러스트모음
메이플스토리
메이플스토리2
블로그씨

블로그 음악
좋아하는 음악
좋아하는 음악(E)
좋아하는 음악(J)
좋아하는 음악(G)

수학수학열매
좋은 정보들
본문스크랩 창고
갖가지 게임플레이
갖가지 마료TAS
갖가지 게임음악

STEAM 게임 리뷰
바인딩 오브 아이작
아이작 리버스
프리덤 플래닛
리스크 오브 레인
STEAM 그 외
지갑의 무덤

마료카페

전체보기(2699) 목록열기

2-SAT 문제(2-Satisfiability Problem)

대회알고리즘 2016.09.02. 14:04

http://kks227.blog.me/220803009418 복사

번역하기 전용뷰어 보기

안녕하세요. 이번에 강의할 내용은 **2-SAT**(2-Satisfiability)이라는 좀 생소할 수 있는 내용입니다!
이건 **충족 가능성 문제**(satisfiability problem) 중 하나인데,
충족 가능성 문제는 여러 개의 boolean 변수들(true, false 중 하나의 값만 가질 수 있는)로 이루어진 boolean expression이 있을 때,
각 변수의 값을 true, false 중 하나로 설정하여 전체 식의 결과를 true로 만들 수 있느냐는 문제입니다.

▲ 사진 출처: 코드포스 튜토리얼 페이지

이번 글은 명제와 깊은 관련이 있기 때문에, 이산구조 중 명제에 대한 지식이 좀 필요합니다.

https://www.acmicpc.net/problem/11280

라이 (kks227)

요즘 많이 바쁘네요(대딩+男+마리오+컴퓨터과학+그림+음악)

프로필 쪽지 이웃추가

2017.03 월별보기

일	월	화	수	목	금	토
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

마료카페배너
마료카페 대충툰
Ries World(NTC)
SMW-FALL
Cyberworld2(중단)
SMGW(팀프로젝트)
Eight Devas
기타 단편작

슈마월드공략(구)
브루탈마리오공략(구)
DWTLC공략(구)
DWTLC공략
LA1공략
LA2공략
LA3공략
LA4공략
X-ser공략
마리오퀘스트공략
바우져의복수공략
TSRP2공략
기타 단편작

마리오슈터(GM)
Game Maker 8(기능)
Game Maker 8(코딩)
Game Maker 8(함수)
Game Maker 8(팁)
Game Maker 8(예제)
자작 라이브러리
GM폭풍강좌(슈팅편)

공튀기기 - 자작
공튀기기 - 연구소
공튀기기 - 답사
공튀기기 - 자료실
이지툰 - 심심풀이
이지툰 - 펌

슈퍼마리오월드공략
슈퍼마리오월드동영상
요시아일랜드공략
요시아일랜드동영상
마리오RPG공략
젤다트라이포스공략
별의카비3공략
카비슈퍼스타공략
슈퍼봄버맨5공략
팝픈 트윈비

동굴이야기공략
모바일 게임
닌텐도 DS

Lunar Magic 강좌

커스텀 블록
커스텀 스프라이트
커스텀 뮤직
커스텀 그래픽
커스텀 패치

루나매직(LM)
LM 유틸리티
SMW 작품(ips)

이 문제에서 SAT 문제가 무엇인지에 대해 대략 설명해주고 있는데요.

$$f = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2)$$

이러한 식 f가 있으면 x1 = x2 = false, x3 = true로 정하면 f가 true가 될 수 있지만,
(¬자 모양 연산자는 NOT, 위로 뾰족한 연산자는 AND, 아래로 뾰족한 연산자는 OR입니다.)

$$f = (x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1)$$

이런 식의 경우 x1에 무슨 값을 넣어도 f가 true가 될 수 없습니다.
이렇게 f가 어떻게 주어지느냐에 따라서 f를 true로 만들 수도, 없을 수도 있습니다.

이때 식 f는 항상 괄호로 둘러싸여 있고 안쪽은 두 변수 혹은 NOT 변수들의 OR 연산으로만 이루어져 있으며, 바깥엔 AND 연산으로만 이루어져 있는 2중 구조를 띄고 있는데(NOT까지 포함하면 3중)
이때 괄호 단위의, OR 연산으로만 이루어진 부분을 **절**(closure)이라 하며, 이렇게 절의 AND 연산으로만 표현된 식의 형태를 **CNF**(Closure Normal Form)라고 합니다.
각 절이 모두 true여야만 전체식도 true가 될 것이고, 따라서 각 절의 변수들 중 하나라도 true가 되도록 만드는 것이 문제를 푸는 방향입니다.
이때 **각 절의 변수 개수가 최대 2개인 경우를 2-SAT이라고 칭합니다.** 최대 k개는 k-SAT인 식.
3-SAT 이상의 식들은 모두 변형하여 3-SAT의 꼴로 나타낼 수 있으며, 3-SAT 문제를 푸는 것은 NP 문제이지만... **2-SAT 문제만은 다항 시간에 풀 수 있습니다.**

도대체 왜 뜬금없이 이런 난해한 개념이 나오느냐?
왜냐면, 2-SAT 문제를 푸는 방법이 **SCC**를 응용한 것이기 때문입니다;; ~~어?~~
네. 아직 그래프 챕터는 안 끝났습니다. 게다가, 그래프에서 굉장히 중요한 파트도 아직 안 했고...

두 변수 p, q가 있을 때, p가 참이면 q도 참이라는 뜻의 명제를 **p ⇒ q**로 표현 가능한데요.<xml>
<o:OfficeDocumentSettings> <o:AllowPNG /> </o:OfficeDocumentSettings> </xml>
p → q는 단순히 원래부터 주어져 있던 명제이고, 중간에 r이라는 또다른 변수를 거쳐서
p → r, r → q가 성립하면 삼단논법에 의해서 간접적으로 p가 참이면 q도 참이 되고 이런 관계를 2중 화살표로 나타내서 p ⇒ q로 표현합니다. 이런 과정을 추론이라고도 합니다.

여기서, 처음부터 있던 명제들로 인해 어떤 변수 x에 대해 **x ⇒ ¬x인 동시에** ¬x ⇒ x가 성립한다면, 이 명제들의 묶음에서는 **모순**이 발생한다고 할 수 있습니다. 한쪽 명제만 있다면야 좌변이 거짓이고 우변이 참이면 성립은 하는데, 양쪽 명제가 다 있다면... 이런 모순이 또 없죠.
그렇다면 처음부터 있던 명제들은 어떻게 고를까요? 이게 2-SAT 문제에서만 가능합니다.

2-SAT 문제에서 어떤 절 하나를 살펴봅시다. 각 절이 true여야 하니까, 절에 들어있는 두 항 중 적어도 하나는 참이어야 합니다.
(x1∨x2)라는 절이 있다면, 만약 x1이 거짓이라면 x2는 참이어야만 하고, x2가 거짓이면 반대로 x1이 참이어야 전체 식이 true가 될 가능성이 생깁니다. 아니면 둘 다 거짓이 되어 절대 전체 식이 true가 될 수가 없습니다.
(x1∨¬x2)라는 절이 있다면, x1이 거짓이면 ¬x2가 참이어야 하는데 이 말은 x2가 거짓이어야 한다는 의미이며, ¬x2가 거짓이면, 즉 x2가 참이면 x1이 참이어야 합니다.
이런 식으로 만약 두 항 중 하나가 거짓이라면 나머지 하나는 참이어야 한다는 명제들을 모두 모아보는 겁니다.

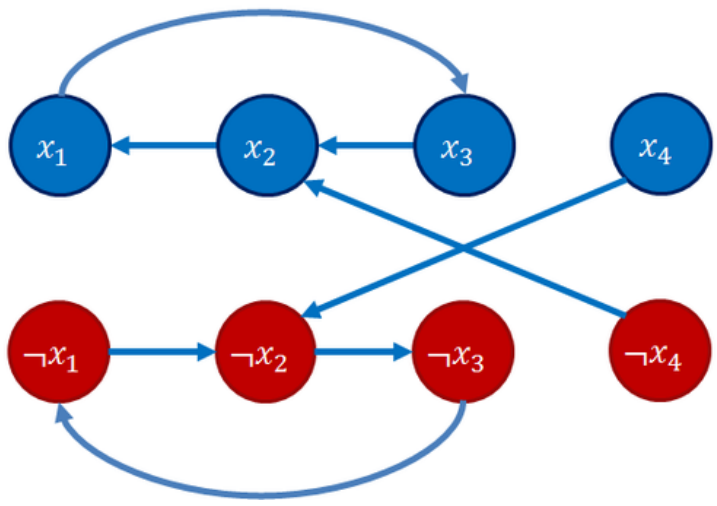
절 (x1∨x2)는 다음과 같은 2개의 명제로 나타낼 수 있습니다.
¬x1 → x2, ¬x2 → x1
절 (x1∨¬x2)는 다음과 같은 2개의 명제로 나타낼 수 있습니다.
¬x1 → ¬x2, ¬(¬x2) = x2 → x1
이런 식으로 모든 절에서 각각 두 개의 명제를 추출합니다. 예를 들어,

$$f = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_3 \vee x_2)$$

이 식에서는 이런 명제들을 끌어내야 합니다.
x1 → x2, ¬x2 → ¬x1, x2 → x3, ¬x3 → ¬x2, ¬x1 → x3, ¬x3 → x1, ¬x3 → x2, ¬x2 → x3

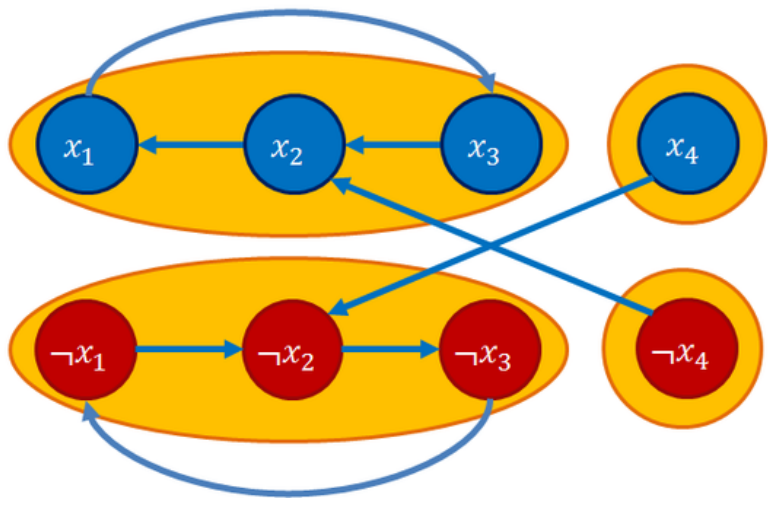
이제 각 변수와 NOT 변수를 하나의 정점이라 생각하고, 이런 → 관계들을 간선이라 생각한다면,
x1 → ¬x1과 ¬x1 → x1 꼴의, **자신의 NOT 형 변수로 가는 경로가 양쪽으로 존재하는 경우**가 하나라도 존재한다면 전체 식을 참으로 만들 방법이 없는 게 확실합니다.
변수의 개수가 적다면 이건 플로이드 알고리즘을 통해 확인할 수 있지만, 변수가 많다면?
네, 그렇습니다. 이 그래프를 SCC별로 분리한 후, **어떤 변수와 NOT 형 변수가 같은 SCC에 있는 경우가 있는지**를 체크하는 식으로 확인할 수가 있습니다!! 그렇다면 서로간에 경로가 존재한다는 의미니까요.

$$f = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_4 \vee \neg x_2)$$



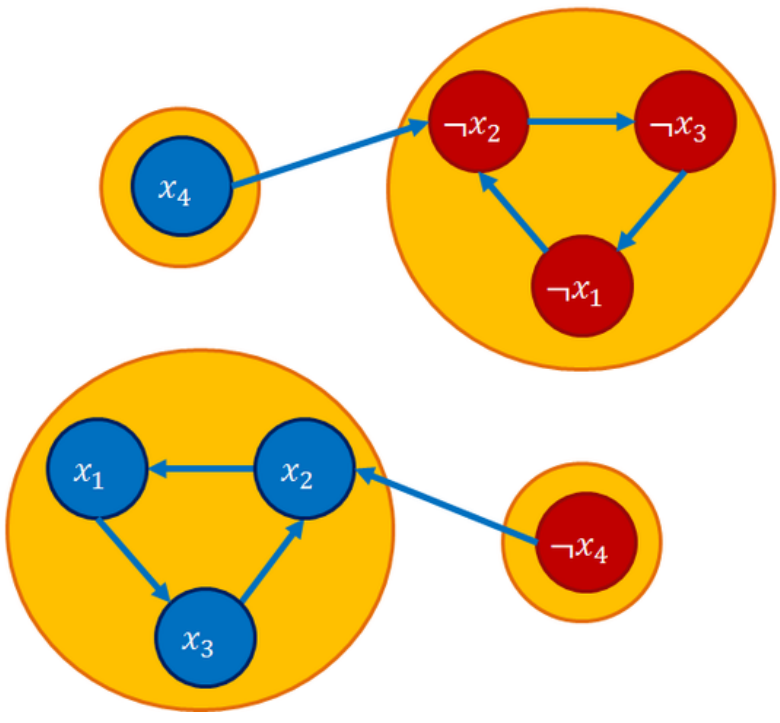
어떤 식 f가 저러할 때, 이걸 그래프 모델링하면 이렇게 됩니다.

$$f = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_4 \vee \neg x_2)$$



SCC별로 분리하면 이렇게 되고,

$$f = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_4 \vee \neg x_2)$$



보기 좋게 재배열하면 이렇게 됩니다.
indegree가 0인 SCC는 {x4}, {¬x4} 2개입니다.
이때, 항상 대칭형의 명제들만을 가지고 그래프를 만들었기 때문에 결과 그래프도 항상 대칭형이 됩니다.
여튼 여기서 주목해야 할 점은, p → q라는 명제가 있을 때 p가 거짓이면 q는 참이어도 거짓이어도 명제를 해치지 않지만, p가 참이면 q도 반드시 참이어야 합니다.
따라서 하나의 SCC 안에 있는 정점들은 그 중 하나라도 참이면 나머지도 모두 참이어야 하고,
따라서 SCC 안에 p와 ¬p가 동시에 존재한다면 둘 다 참일 수는 없으므로 모순이 됩니다.

그럼 일단 SCC 안에 p, ¬p가 동시에 존재하면 f 식을 참으로 만들 수 없는 건 알았는데요.
그럼 그런 경우가 하나도 없으면 f 식을 반드시 참으로 만들 수 있을까요? 그렇다고 합니다.
일단 그렇다고 믿고(?) "2-SAT 3" 문제를 풀어본다면 아래와 같습니다.

```cs
#include <cstdio>
#include <cstring>
#include <vector>
```

```cpp
#include <stack>
#include <utility>
#include <algorithm>
using namespace std;
typedef pair<int, int> P;
const int MAX = 10000;

int N, M, cnt, scc, dfsn[MAX*2], sn[MAX*2];
vector<int> adj[MAX*2];
bool finished[MAX*2];
stack<int> S;

// 자신의 not literal의 정점 번호 리턴
inline int oppo(int n){ return n%2 ? n-1 : n+1; }

// GetSCCsByDFS로 SCC 추출
int GetSCCsByDFS(int curr){
    // ...
}

int main(){
    // 그래프 구축
    scanf("%d %d", &N, &M);
    for(int i=0; i<M; i++){
        int A, B;
        scanf("%d %d", &A, &B);
        // 양수냐 음수냐에 따라 각 정점 번호를 새로 매김
        // x_k: (k-1)*2, not x_k: (k-1)*2-1
        A = (A<0 ? -(A+1)*2 : A*2-1);
        B = (B<0 ? -(B+1)*2 : B*2-1);
        // (A or B)에 대한 간선 추가
        adj[oppo(A)].push_back(B); // not A -> B
        adj[oppo(B)].push_back(A); // not B -> A
    }

    // SCC 추출
    for(int i=0; i<N*2; i++)
        if(dfsn[i] == 0) GetSCCsByDFS(i);

    // x_k와 not x_k가 한 SCC 안에 있으면 불가능
    for(int i=0; i<N; i++){
        if(sn[i*2] == sn[i*2+1]){
            puts("0");
            return 0;
        }
    }
    // 가능
    puts("1");
}
```

Colored by Color Scripter

CNF를 입력받아 그걸 토대로 그래프를 구축하고, SCC별로 분리한 후
각 변수에 대해 자신과 not 형 변수가 같은 SCC 안에 있는 경우가 있는지 다 훑어봅니다.
그런 게 하나라도 있으면 불가능, 없으면 가능합니다.

https://www.acmicpc.net/problem/11281

11281번: 2-SAT - 4

www.acmicpc.net

그럼 이제 나아가서, 각 변수에 어떤 값을 대입해야 f 식을 참으로 만들 수 있는지까지 알아봅시다.
이 솔루션을 얻어낼 수 있다면, f 식을 참으로 만들 수 있는지에 대한 조건이 정당하다는 것도 증명해낼 수 있겠죠.

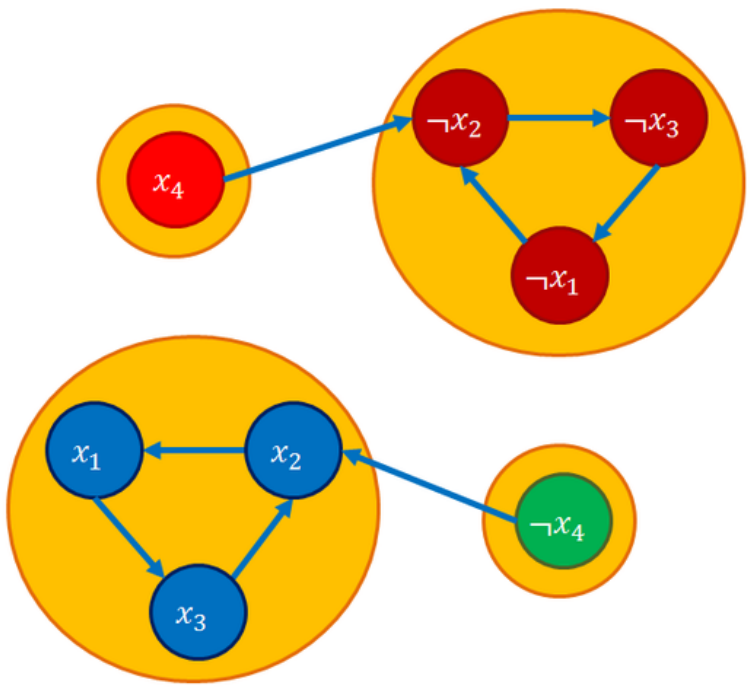
아이디어는 아까 언급한대로, 명제 p → q가 있을 때 p가 거짓이라면 절대 이 명제를 해칠 일이 없다는 것에서 나옵니다.
이제 SCC 단위로는 안의 정점들이 다 같은 값을 가져야 한다는 것을 알았으니, 각 SCC를 하나의 정점으로 본다면 이런 발상이 가능합니다.
SCC P, Q가 있을 때, P에서 Q로 가는 경로가 존재한다면, 만약 P의 정점들의 값이 거짓이라면 Q는 어찌되더라도 좋지만, P의 정점들의 값이 참이었다면 Q에 속한 정점들의 값은 무조건 참이어야만 합니다.

따라서 SCC 단위로 위상 정렬을 하여 훑어갈 때, 처음에 만나는 정점들의 값은 되도록 false로 설정해주고, 그 not 버젼의 정점을 true가 되게 하는 식으로 설정해 봅시다.

q = ¬p라 할 때, p가 먼저 방문되었을 경우 p를 false로, q를 true로 설정하는 식으로 하면,

p와 q는 서로 다른 SCC에 있고, p가 속한 SCC를 P, q가 속한 SCC를 Q라 하면 P에서 Q로 가는 경로야 있**을 수 있**지만 Q에서 P로 가는 경로는 없으므로 이런 방식이 먹힙니다. 즉, 이런 방식으로 값을 매기다가 참->거짓 꼴의 이동경로가 생기지 않습니다.

$$f = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_4 \vee \neg x_2)$$

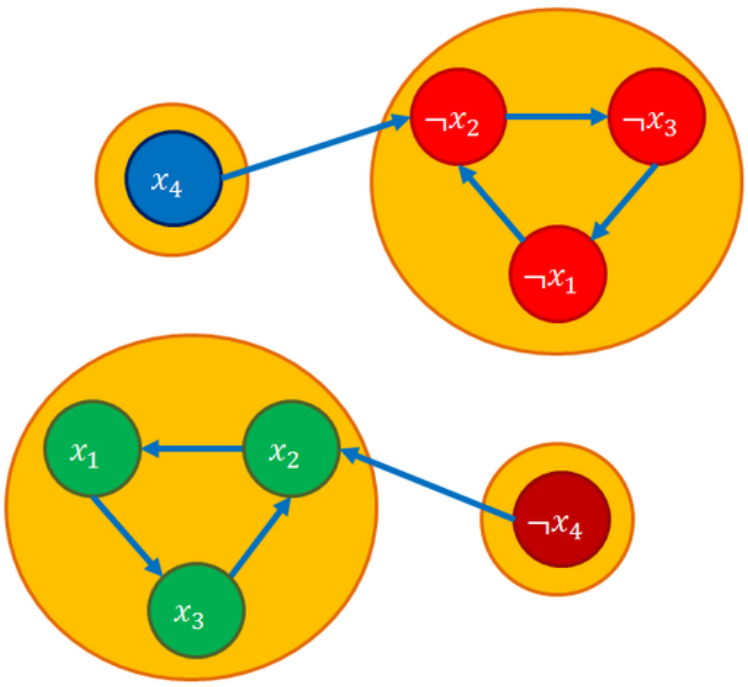


x_1	x_2	x_3	x_4
0	-	-	-

맨 처음에 x4와 ¬x4 중 어떤 것이 방문될지는 모르지만, x4가 먼저 방문되었다고 칩시다.

그렇다면 우리는 x4 = false가 되게 설정합니다.

$$f = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_4 \vee \neg x_2)$$



x_1	x_2	x_3	x_4
0	1	1	1

그 다음, ¬x4와 ¬x2 중 어떤 게 먼저 방문될지 역시 알 수 없습니다.

¬x4가 먼저 방문된다면 x4의 값은 이미 매겼으니 무시되고, 이제 ¬x2와 x2 중 하나를 이어서 방문하겠죠.

¬x2가 먼저 방문된다면 x1, x2, x3을 싸그리 true로 설정해 줄 겁니다.

반대로, x2가 먼저 방문된다면 셋은 false가 됩니다.

어떤 경우라도 f 식은 참이 됩니다. 이 식 자체가 상당히 헐거웠네요.

```cs
int result[MAX]; // 각 변수의 값 (0, 1 중 하나)
memset(result, -1, sizeof(result)); // 초기화
// 각 변수를 속해있는 SCC 번호순으로 정렬
P p[MAX*2];
for(int i=0; i<N*2; i++)
    p[i] = P(sn[i], i);
sort(p, p+N*2);

// 놀랍게도, SCC 번호가 크면 클수록 DAG상에서 앞에 있음
for(int i=N*2-1; i>=0; i--){
    int var = p[i].second;
    // 아직 해당 변수값이 설정되지 않았다면 지금 설정
    if(result[var/2] == -1) result[var/2] = !(var%2);
}
// 각 변수의 값 출력
for(int i=0; i<N; i++)
    printf("%d ", result[i]);
```

가능성 판별 후, 이런 추가적인 작업을 행해 주면 되는데,
여기서 SCC DAG의 위상 정렬 순서대로 정점을 방문하는 것을 놀랍도록 빠르게 할 수 있는데
종만북에서 언급한 위상 정렬 알고리즘이 DFS 방문 후 그 순서를 역순으로 뒤집으면 위상 정렬 순서가 되는 것이었습니다.
DFS 방문 순서대로 발견되는 SCC 번호를 오름차순으로 매겼으니, 그 번호를 역순으로 방문하면 SCC 단위 위상 정렬 순서대로 방문하는 꼴이 되는 셈입니다.
이제 만나는 변수마다 먼저 마주친 쪽을 false로 설정해주면 되는데, 정점 x_k를 먼저 마주쳤다면 x_k = false가 되고, 정점 not x_k를 먼저 마주쳤다면 x_k = true가 됩니다.
또한, SCC 번호 순으로 변수를 방문하므로 하나의 SCC 안에 속한 변수들은 모두 연속적으로 방문되므로 항상 같은 값으로만 설정될 겁니다.

자 이쯤되면 슬슬 이런 의문이 나올 겁니다. **우리가 도대체 왜 이딴 짓을 했을까?**
2-SAT 문제로 모델링하여 풀 수 있는 문제들이 존재하기 때문입니다.

https://www.acmicpc.net/problem/2207

2207번: 가위바위보

www.acmicpc.net

~~천더x천더~~가위바위보라는 문제가 대표적인 2-SAT 문제입니다.
원장선생님이 3가지 선택지 중 보를 버리고, 가위와 바위 중 하나만 내겠다고 합니다.
또한 각 학생은 2번의 추측 중 한 개라도 맞히면 살 수 있고, 문제에서 묻는 것은 모든 학생이 살 수 있는 가능성이 존재하느냐입니다.
따라서 **원장선생님이 k번째에 가위를 낼 경우 참이 되는 변수를 x_k라 합시다.** x_k가 거짓이면 k번째에 바위를 냈다는 겁니다.
각 학생은 두 개의 예측 중 하나라도 맞혀야 하므로, 각 절을 각 학생에 대응시켜서 만들고 전체를 AND 연산하면 2-SAT 식이 완성됩니다.
예를 들면, 어떤 학생이 "원장선생님은 4번째에 바위를 내고 7번째에 가위를 내실 것이다"라고 예측했다면, 해당하는 절은 (¬x4 ∨ x7)로 표현할 수 있습니다.
각 절마다 하나 이상의 항이 참이 되어야 전체 식이 참이 되어서 모든 학생이 살 수 있게 되죠! 그럴 수 있는 경우가 절대 없다면 답이 OTL입니다.

https://www.acmicpc.net/problem/3648

3648번: 아이돌

www.acmicpc.net

이 문제도 유사합니다. 각 심사위원이 2개의 의견을 내는데, 둘 중 최소한 하나는 반영이 되어야 한다고 합니다. 각 심사위원의 의견을 하나의 절에 대응시킬 수 있다는 게 보입니다.
헌데 이 문제의 경우 상근이에 해당하는 1번 참가자, 즉 x1 변수는 반드시 참이어야만 하는데,
이렇게 어떤 변수가 반드시 참이어야만 한다는 조건을 2-SAT 식에 잘 녹아들게 하려면 이런 절을 추가하면 됩니다.

(x1 ∨ x1)

두 항 중 하나 이상이 참이어야 하는데, 둘이 똑같으니까 결국 x1이 참이어야만 한다는 소리죠.

이렇게 2-SAT 문제는 겉으로는 굉장히 알아채기 어렵게 꽁꽁 싸여 있는데,
보통 여러 사람이나 사물이 **2개의 선택지를 가지고, 둘 중 최소한 하나는 만족해야 하는 형태의 조건**들이 주어질 때 2-SAT을 생각해 볼 수 있습니다.

다른 SAT 문제들에 대해 짤막하게 말씀드리자면, 일단 1-SAT 문제는 매우 쉽습니다.
그냥 식에서 x1과 ¬x1이라던지, 자신과 not 형이 동시에 존재하지만 않으면 됩니다.
그러나 3-SAT 이상의 문제는 **NP 영역**에 속해 있는데요. 즉, 다항 시간에 푸는 방법이 아직은 발견되지 않았습니다.
그런데 3 이상의 k에 대해, k-SAT 문제는 항상 3-SAT 문제로 변형 가능하다는 것이 또 증명되어 있습니다.
현재로써는 2-SAT과 그 너머의 문제들의 난이도가 아주 확연히 차이가 나고 있죠.

추천 문제

> 11280번: 2-SAT - 3

> 11281번: 2-SAT - 4

> 2207번: 가위바위보

> 3648번: 아이돌

> 3747번: 완벽한 선거!

> 7535번: A Bug's Life

> 1739번: 도로 정비하기 (★)

> 3153번: 타워 디펜스 (★)

#알고리즘

'전체' 카테고리의 다른 글 전체 포스트 보기

[5012번] 불만 정렬(★★★☆☆) (12) 2016.09.07.
네트워크 유량(Network Flow) (17) 2016.09.05.
2-SAT 문제(2-Satisfiability Problem) (2) 2016.09.02.
강한 연결 요소(Strongly Connected Component) (수정: 2017-01-16) (13) 2016.09.01.
[STEAM 게임 리뷰] Starbound (4) 2016.08.30.

◂ 이전 다음 ▸

▲ top