2019 홍익대학교 프로그래밍 경진대회 풀이

2019.11.01

A 나의 학점은

제출 : 136회

정답 : 71명 (정답률 52%)

First Solve : 김X욱님 (3분)

출제자 : wonjaek36

A 나의 학점은?

```
int score[51], my_score;
for(int i=1; i<=50; ++i)
        scanf(“%d”, &score[i]);
scanf(“%d”, &my_score);

for(int i=1; i<=50; ++i){
        if(score[i] == my_score){
                //i등에 해당하는
                //학점을 출력
        }
}
```

이런 실수들이 있었습니다 :

- 인덱스를 0~49로 잡았는데, 등수는 1~50으로 확인한다.

- score[]안에 내 등수가 없다고 생각하고, 배열 내에서 같은 점수의 등수를 찾지 않는다.

B 공주님을 구해라!

제출 : 206회

정답 : 11명 (정답률 5%)

First Solve : 김X욱님 (17분)

출제자 : wonjaek36

B 공주님을 구해라!

그람을 얻었으면 벽을 무시하므로, 공주님까지 다이렉트로 갈 수 있습니다.

BFS를 이용해서, ①그람까지의 거리, ②공주님까지의 거리를 구하면
최단거리는 min(①+그람에서 공주님까지의 멘하탄 거리, ②)가 됩니다.
(멘하탄 거리 : |x1−x2|+|y1−y2|)

또는, 그람을 얻었는지의 여부를 기억해 놓으면서, 가지고 있다면 벽을 무시하고 지나도록 BFS를 돌리셔도 됩니다.

C 달팽이 리스트

제출 : 265회

정답 : 27명 (정답률 10%)

First Solve : 손X용님 (20분)

출제자 : degurii

C 달팽이 리스트

K가 주어질 때마다 반복문을 K번씩 돌리면 시간 복잡도가 O(MK)로 시간 초과를 받게 됩니다.

우선 싸이클에 포함된 노드와 그렇지 않은 노드를 따로 관리합니다.

1) K번 이동해서 도착했을 때 싸이클에 포함된 노드가 아니라면, 쉽게 구할 수 있습니다.
2) K번 이동해서 도착한 노드가 싸이클에 포함된다면,
   (K-싸이클에 포함되지 않은 노드 수)%(싸이클에 포함된 노드 수) 를 인덱스로 사용해서 쿼리당 O(1)로 구할 수 있습니다.

D 문자열 화폐

제출 : 150회

정답 : 26명 (정답률 17%)

First Solve : 배X반님 (60분)

출제자 : Green55

D 문자열 화폐

최대한 왼쪽에 앞서는 알파벳을 끼워 넣는게 최적입니다.

ex) A**A**ZZZZZZZ 보다는 A**B**AAAAAAAA이 앞선다.

따라서, 최대한 뒤쪽에 Z를 채워넣어서 문자열의 가치를 높이고,
가치가 충분히 커졌으면 앞쪽엔 A를 채워넣으면 됩니다.

즉, 답은 기본적으로 “AA..AA[?]ZZ..ZZ”의 형태입니다. ([?] = 어떤 알파벳)

D 문자열 화폐

1	2	…	i-1	i	i+1	…	N-1	N
		…		?	Z		Z	Z

i번째 알파벳이 무엇을 넣을지 결정해봅시다.

1~(i-1)을 A로 채워넣었을 때 문자열의 가치는 1×(i-1) + [i번째 알파벳의 가치] + 26×(N-i)

따라서 1 <= X - 1×(i-1) - 26×(N-i) <= 26이라면,

i에 X - 1×(i-1) - 26×(N-i)번째 알파벳을 넣고, 1~(i-1)을 A로 채워넣을 수 있습니다.

그렇지 않다면, i에는 Z를 채워야합니다.

D 문자열 화폐

예외 : 전부 A로 채워도 안 될만큼 X가 너무 작거나,
전부 Z로 채워도 안 될만큼 X가 너무 크면 불가능합니다.

(출력초과를 받으신 분들은, 대부분 작은 경우를 처리하지 않아서 입니다.)

E 222-풀링

제출 : 32회

정답 : 27명 (정답률 84%)

First Solve : 이X식님 (52분)

출제자 : degurii

E 222-풀링

본문에 나온 대로 구현만 하면 되는 문제입니다.
설명하기가 애매하니 코드로 대체합니다.

```
int n, p[1100][1100];
int snd_max(int x, int y) {
            vector<int> t;
            for (int i = x; i < x + 2; i++) {
                        for (int j = y; j < y + 2; j++) {
                                    t.push_back(p[i][j]);
                        }
            }
            sort(t.begin(), t.end());
            return t[2];
}
```

E 222-풀링

```
int main() {
	cin >> n;
	for (int i = 0; i < n; i++) {
		for (int j = 0; j < n; j++) {
			cin >> p[i][j];
		}
	}
	while (n > 1) {
		for (int i = 0; i < n / 2; i++) {
			for (int j = 0; j < n / 2; j++) {
				int x = i * 2, y = j * 2;
				p[i][j] = snd_max(x, y);
			}
		}
		n /= 2;
	}
	cout << p[0][0];
}
```

F 이진수씨의 하루 일과

제출 : 18회

정답 : 5명 (정답률 28%)

First Solve : 이X형님 (118분)

출제자 : Green55

F 이진수씨의 하루 일과

“ 진수씨는 그의 일생 동안의 경험으로
  큰 이진수를 빠르게 곱하는 것은 매우 어렵다는 것을 알고 있다. ”

평범하게 곱셈 알고리즘을 구현한다면, $O(N^2)$이 걸립니다.
이것보다 빠른 곱셈 알고리즘은 구현이 어렵습니다.
(카라츠바, FFT등으로 O(NlogN))

따라서 수를 직접 곱하지 말고,
곱했을 때 결과의 길이만 빠르게 구하는 방법을 생각해봅시다.

F 이진수씨의 하루 일과

?를 전부 1로 채웠을 때 두 곱의 값이 제일 크고,
?를 전부 0로 채웠을 때 두 곱의 값이 제일 작습니다.

따라서 그렇게 채운 후, A와 B의 곱의 길이만 판별해봅시다.

ex) B = 0?1??면, 최소값 : 11111×0?1?? → 11111×00100 → 11111×100

논의를 위해 B 앞에 쓸데 없는 0은 떼버립니다.
편의상 이런 B의 길이를 M이라고 해보겠습니다. (위의 예는 M = 3)

F 이진수씨의 하루 일과

관찰 1 : A×B는 아무리 커도 최대 (N+M)자리이다.

B가 가장 큰 경우는 111...11일 때 입니다. 이 경우,

$$A = 2^N - 1_{(10)}$$
$$B = 2^M - 1_{(10)}$$
$$A \times B = 2^{N+M} - (2^N + 2^M - 1) < 2^{N+M}$$

F 이진수씨의 하루 일과

관찰 2 : A×B는 아무리 작아도 최소 (N+M-1)자리이다.

B가 가장 작은 경우는 100...00일 때 입니다. 이 경우,

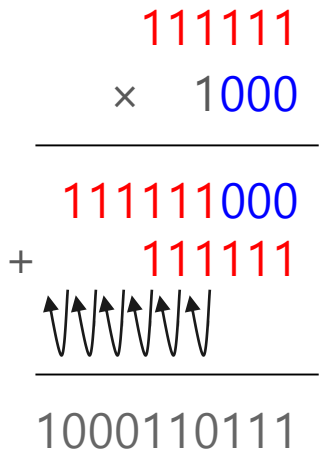
```
  111111
×   1000
--------
111111000
```

명백하게 (N+M-1)자리입니다.

F 이진수씨의 하루 일과

그렇다면 답은 N+M, N+M-1 중 하나입니다.

그런데 제일 작은 B=1000보다 딱 1 더 큰 B=1001일 때,
111111×1001 = (111111×1000) + 111111
올림이 일어나서 N+M-1자리를 넘어버립니다.
(111111000은, 1000만 더해도 올림이 일어납니다.)



따라서, B가 100...00 꼴이면 답은 N+M-1 이고,
맨 앞 비트 외에 1이 하나라도 있으면 답은 N+M 입니다.

예외 : B = 1을 주의하세요! B는 맨 앞 비트 외에 1인 비트가 없습니다.

F 이진수씨의 하루 일과

4404 ms	Python 3

하지만… 검수 때는 제가 파이썬으로 시간초과가 났는데, 본대회 중 학우분께서 두 이진수를 직접 곱하는 방법으로 풀어버렸습니다
(파이썬은 큰 수의 연산을 지원합니다)

제가 부족한 따름입니다. 죄송합니다 😢

G 면접보는 승범이네

제출 : 7회

정답 : 1명 (정답률 14%)

First Solve : 이X성님 (173분)

출제자 : degurii

G 면접보는 승범이네

문제를 해결하기 위해 그래프의 각 노드에서 가장 가까운 면접장까지의
최단 거리를 구해야 합니다.
따라서 최단 거리 알고리즘, 특히 다익스트라를 써야 한다는 것을 알 수 있습니다.

하지만 모든 노드에서 다익스트라를 돌려 답을 구할 경우,
O(VElogE)로 시간초과를 받게 됩니다.

이를 해결하기 위해 알아야 할 두 가지를 다음 페이지에서 설명합니다.

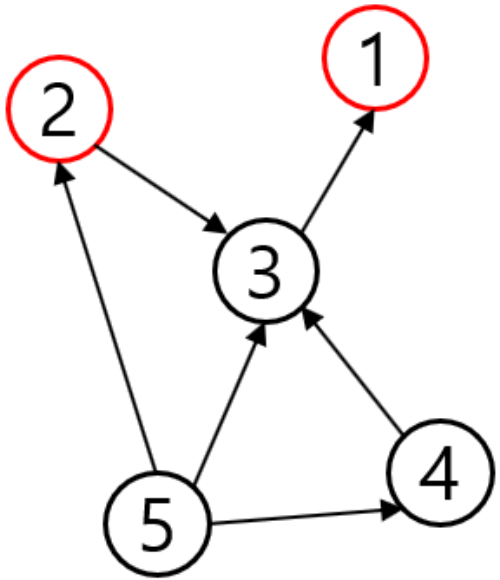
G 면접보는 승범이네

1) **역 그래프에서 돌린 다익스트라 한 번으로 원본 그래프의 모든 노드에서 특정 노드 x까지의 최단 경로를 구할 수 있습니다.**

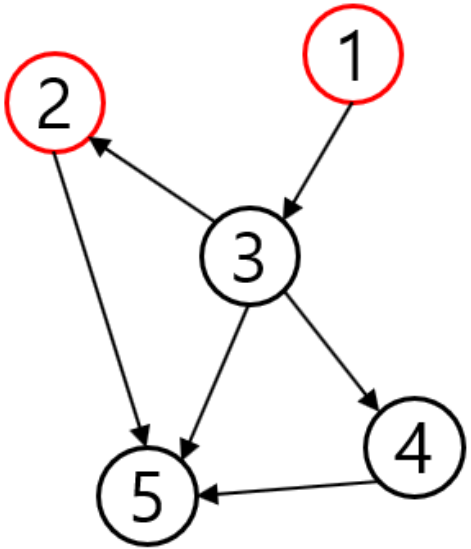
임의의 그래프와 그 그래프의 간선 방향을 뒤집은 역방향 그래프가 있다고 합시다.

이때 역방향 그래프에서 노드 x를 시작점으로 다익스트라를 돌려 거리배열 dst[]를 얻을 수 있는데, 원본 그래프와 역 그래프에서 해석이 달라집니다.

역 그래프에서 dst[i]는 x->i까지의 최단 거리이고, 놀랍게도 원본 그래프에서의 dst[i]는 i->x까지의 최단 거리를 뜻합니다.



[원본 그래프]



[역방향 그래프]

G 면접보는 승범이네

2) **다익스트라 알고리즘은 시작점이 여러 개여도 됩니다.**

보통 다익스트라 알고리즘에서 시작점은 하나만 있어야 한다고 생각할 수 있습니다.

하지만 다익스트라의 원리가 '현재까지 최단 거리임이 보장된 정점'들의 코스트를 기반으로 '아직 보장되지 않은 정점' 중 코스트가 가장 작은 정점을 확장해나가는 방식인 것을 생각해보면 시작점이 여러 개여도 상관없다는 것을 알 수 있습니다. 최단 거리임을 보장하기만 한다면요!

여러 개의 시작점 v_1, v_2, ⋯, v_n 이 있을 때 각 점의 최단 거리는 당연히 0일 것이니, 거리를 0으로 설정하고 시작점 모두를 큐에 넣어 다익스트라를 돌리면 됩니다.

그 결과로 얻는 최단 거리는 어떤 시작점에서 출발했는지는 알 수 없지만, 어쨌든 시작점에서 출발해 임의의 노드까지 가는 가장 짧은 거리라는 뜻입니다.

G 면접보는 승범이네

문제를 해결하기 위해 앞에서 설명한 두 가지를 한 번에 사용하면 됩니다.

우선 모든 간선의 방향을 뒤집습니다. 이 그래프에서 임의의 면접장 x를 시작점으로 다익스트라를 돌리면 원본 그래프의 (모든 노드들→x) 까지의 최단 거리를 구할 수 있습니다.

(1 ≤ 면접장의 개수 ≤ N) 이므로 각 면접장마다 다익스트라를 돌리면 시간 초과입니다.

그러지 않고 모든 면접장을 한 번에 큐에 넣어 다익스트라를 돌리면 시간내에 문제를 해결할 수 있습니다.

I 대기업 승범이네

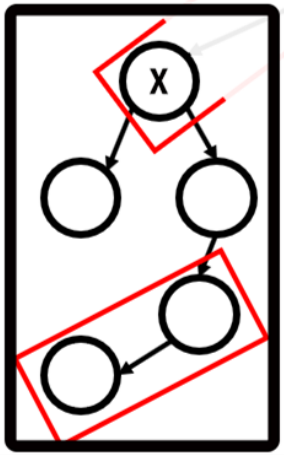
제출 : 0회

정답 : 0명

First Solve : -

출제자 : Green55

I 대기업 승범이네



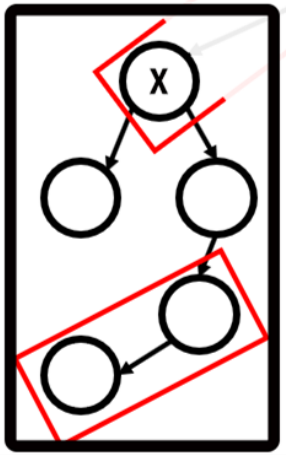
어떤 노드 X와, X의 자손들을 전부 선택하면
트리입니다. 이것을 X의 서브트리라고 합니다.

이런 성질을 이용해서,
트리에도 DP를 적용 시킬 수 있습니다.

트리 DP는 탑-다운(재귀 방식)으로 하는게 편합니다.

바텀-업(반복문)을 사용하려면,
트리의 높이를 파악해 아래부터 채워나가야 하기 때문에
어차피 DFS를 한번 돌려야합니다.

I 대기업 승범이네



X의 서브트리 안에서 최적의 답을 구할 때,
X 말고는 서브트리 외부와 멘토링을 맺을 가능성이 없습니다.

즉, X가 X의 사수와 멘토링을 하는지의 여부 외에는,
서브트리 외부의 멘토링 관계를 기억 할 필요가 없습니다.
따라서 다음과 같은 점화식을 세워봅시다.

dp[X][hasMento] =
X가 X의 사수와 멘토링을 맺을 때(hasMento=1), 혹은 맺지 않을 때(hasMento=0), X의 서브트리에서의 시너지의 합의 최대값

I 대기업 승범이네

케이스 1 : X가 X의 사수와 멘토링을 맺을 때

X는 이미 멘토가 있으니, X의 부사수들은 X를 멘토로 가질 수 없습니다.

그냥 X의 부사수들의 서브트리에서, 최적의 답을 구해 전부 더해주면 됩니다.

X의 부사수를 C1, C2, C3, … 라고 하면

dp[X][1] = dp[C1][0] + dp[C2][0] + dp[C3][0]…

I 대기업 승범이네

케이스 2 : X가 X의 사수와 멘토링을 맺지 않을 때

케이스 1에서 추가로, X와 X의 부사수 한명이 멘토링 하는 경우도 고려해야 합니다.

dp[X][1] = max{

dp[C1][0] + dp[C2][0] + … // X가 아무와도 멘토링 하지 않음

실력[X]*실력[C1] + dp[C1][**1**] + dp[C2][0] + … // X와 C1이 멘토링을 맺음

실력[X]*실력[C2] + dp[C1][0] + dp[C2][**1**] + … // X와 C1이 멘토링을 맺음

…

I 대기업 승범이네

실력[X]*실력[C1] + dp[C1][**1**] + dp[C2][0] + … ①
실력[X]*실력[C2] + dp[C1][0] + dp[C2][**1**] + … ②
…

을 전부 일일이 구하면, 시간이 너무 오래걸립니다. (자식의 수가 C면, $O(C^2)$)

따라서 (dp[C1][0] + dp[C2][0] + dp[C3][0]…)을 미리 O(C)로 구해놓고,

① = (dp[C1][0] + dp[C2][0] + dp[C3][0]…) - dp[C1][0] + dp[C1][1] + 실력[X]*실력[C1]
② = (dp[C1][0] + dp[C2][0] + dp[C3][0]…) - dp[C2][0] + dp[C2][1] + 실력[X]*실력[C2]

면 하나당 O(1)에 구할 수 있습니다.

J 사자와 토끼

제출 : 3회

정답 : 0명

First Solve : -

출제자 : Green55

J 사자와 토끼

왠만하면 게임이 끝납니다.

게임이 끝나지 않으려면, 토끼와 사자가 아무리 노력해도 만날 수 없는 특수한 케이스의 그래프여야합니다.

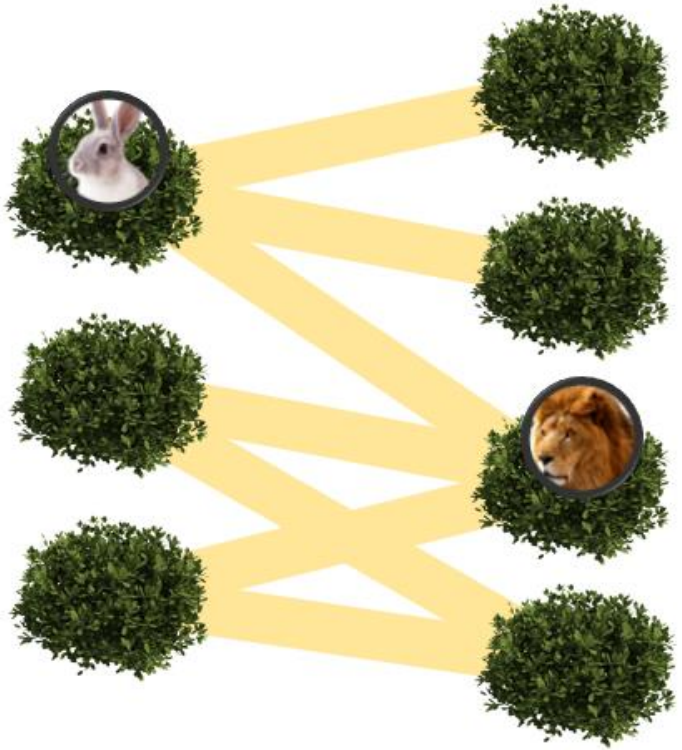
이런 그래프는 어떻게 생겼을까요?

J 사자와 토끼

홀수번째 턴에는
토끼는 항상 왼쪽 수풀에,
사자는 항상 오른쪽 수풀에 있습니다.

짝수번째 턴에는
사자는 항상 왼쪽 수풀에,
토끼는 항상 오른쪽 수풀에 있습니다.

이러면 둘이 항상 엇갈려서
절대 만날 수 없습니다.



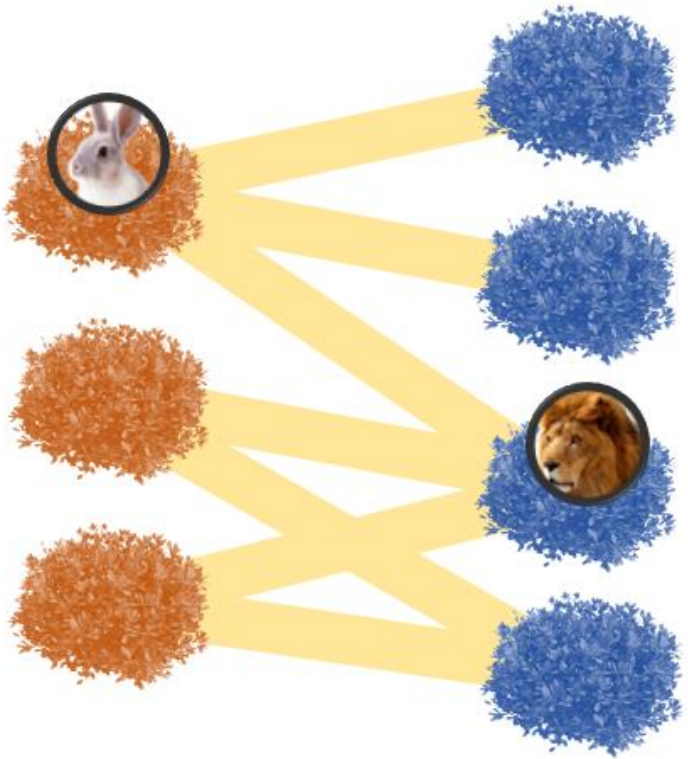
J 사자와 토끼

색을 칠해보겠습니다.

빨간 수풀과 인접한
빨간 수풀은 하나도 없습니다.

파란 수풀도 마찬가지입니다.

이런 그래프를
이분 그래프(Bipartite Graph) 라고 합니다.



J 사자와 토끼

아무 정점이나 잡고 빨간색으로 칠합시다.

DFS나 BFS로 그래프를 순회하며, 현재 정점과 인접한 정점 중
처음 방문하는 정점, 즉 색칠되지 않은 정점을 찾았으면
그 색을 현재 정점과 반대 색으로 칠합니다.

이렇게 칠한 후, 같은 색인 수풀을 연결하는 길이 하나라도 있다면,
이분 그래프가 아니므로 답은 0입니다.

(잘 구현하면 BFS/DFS 한 번으로도 가능합니다.)

J 사자와 토끼

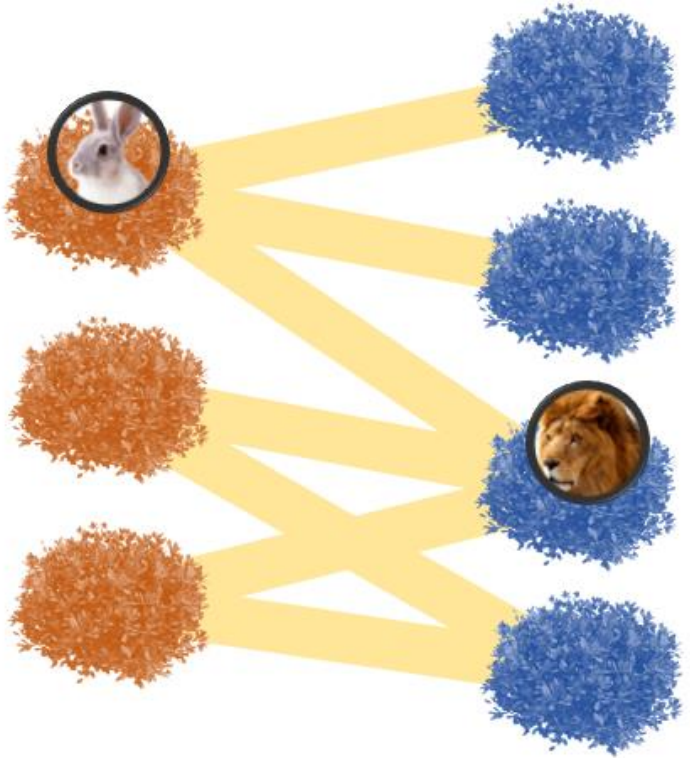
이분 그래프라면,

빨간색 정점의 개수 = R
파란색 정점의 개수 = B

일 때 답은 R * B * 2입니다.

(토끼가 왼쪽, 사자가 오른쪽에 시작하는 경우 R*B

+ 토끼가 오른쪽, 사자가 왼쪽에서 시작하는 경우 R*B)



K 홍익대학교 지하캠퍼스

제출 : 0회

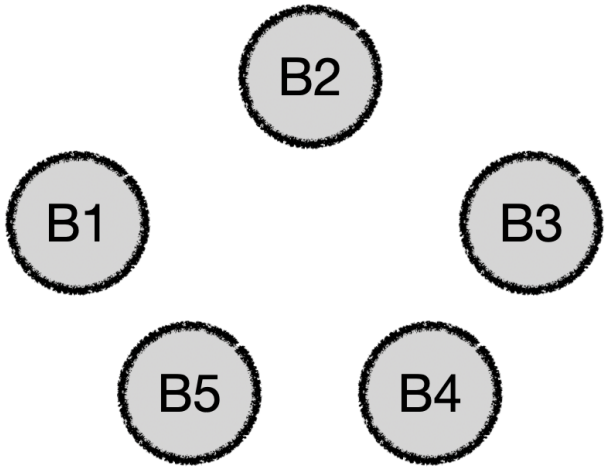
정답 : 0명

First Solve : -

출제자 : rltjqdl1138

K 홍익대학교 지하캠퍼스

1. 지하 1층부터 지하 N층까지를 각각의 층을 Vertex로 생각하는 그래프를 만듭니다.



K 홍익대학교 지하캠퍼스

2. 건물 모델을 한 개 입력받을 때 마다 가능한 모든 간선을 추가합니다.

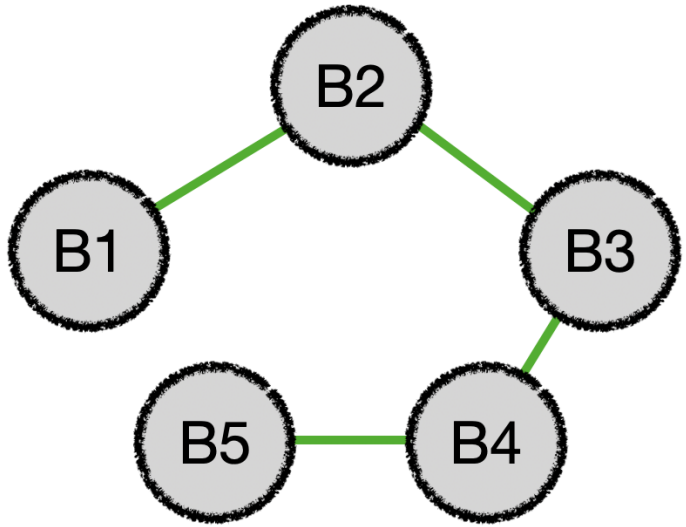
예를들어 H1=2, E11=1, E12=2인 모델을 입력하면

오른쪽 그림과 같이 초록 간선이 추가됩니다.

(해당 모델은 1층과 2층을 연결할 수 있고

2층과 3층을 연결할 수 있고, …

4층과 5층을 연결할 수 있기 때문입니다.)



K 홍익대학교 지하캠퍼스

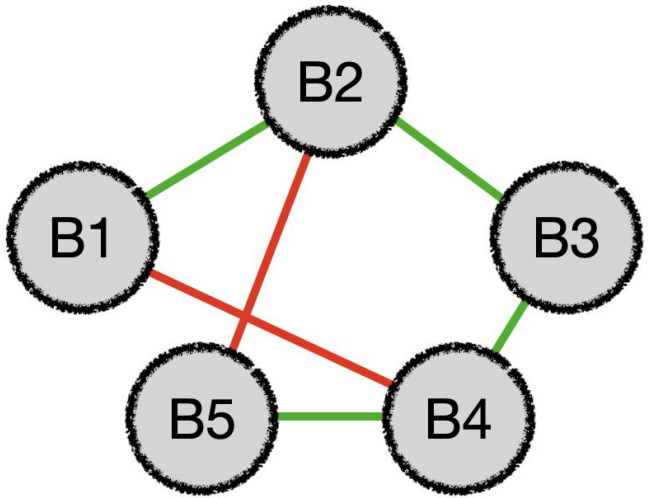
2. 건물 모델을 한 개 입력받을 때 마다 가능한 모든 간선을 추가합니다.

다음으로 H2=4, E21=4, E22=1인 모델을 입력하면

오른쪽 그림과 같이 빨간 간선이 추가됩니다.

(해당 모델은 1층과 4층을 연결할 수 있고

2층과 5층을 연결할 수 있기 때문입니다.)



K 홍익대학교 지하캠퍼스

2. 건물 모델을 한 개 입력받을 때 마다 가능한 모든 간선을 추가합니다.

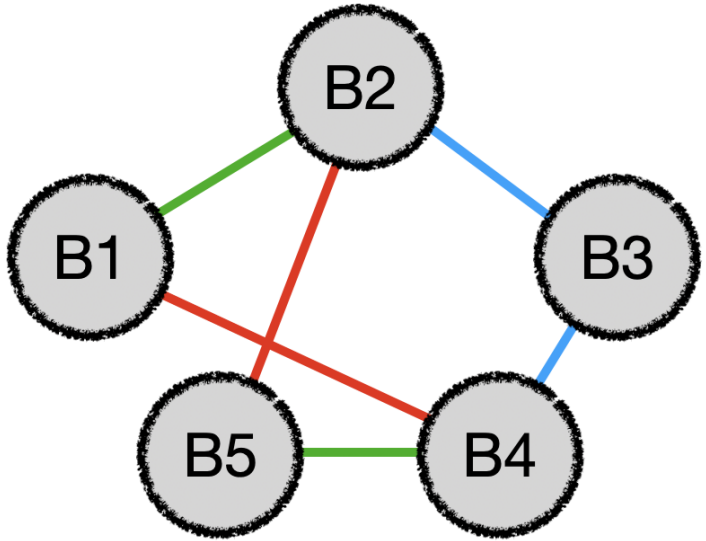
다음으로 H3=4, E31=2, E32=3인 모델을 입력하면

기존의 초록 간선과 T값을 비교합니다.

초록 간선의 Weight(T1)가 파란 간선의 Weight(T3) 보다

클 경우, 오른쪽 그림처럼 간선의 weight를 갱신합니다.

만약 T1 < T3 라면 weight를 갱신하지 않고 넘어갑니다.



K 홍익대학교 지하캠퍼스

3. 완성된 그래프를 이용해 최단거리 (최소 비용)을 구합니다.

이 때 시작점은 *R*, 끝 점은 *D*로 생각하여

최단거리 알고리즘을 구현합니다.

(최단거리 알고리즘에 대한 내용은 생략하겠습니다.

음수의 간선이 없으므로 Dijkstra를 사용하는 것이

무난한 풀이 방법으로 생각됩니다.)