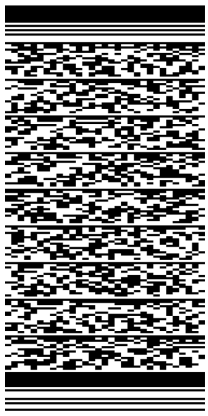
Day 2 Task 4: 저장 (Saveit)



Xedef는 여러 도시들간의 항공 화물을 운송하는 회사이다. Xedef의 항공기들은 각 항공기마다 정해진 두 도시 사이에서 화물을 양방향으로 운송한다. 도시들 중에서 어떤 도시들은 허브라고 불리며 특별한 처리 기능을 갖추고 있다.

한 도시에서 다른 도시로 화물을 운송할 때는 여러 도시를 거쳐서 운송하는데 이 때 하나의 hop은 Xedef의 항공기로 연결된 두 도시 사이의 화물 운송을 의미한다. 그리고 출발 도시에서 목적 도시로 화물을 운송할 때 경로상에 허브가 적어도 1번은 포함되어야 한다.

Xedef는 화물운송을 손쉽게 하기 위해 모든 도시에서 모든 허브 도시로 가는 최소hop의 수를 encoding해서 화물마다 붙여놓으려 한다. (한 허브에서 자기자신으로 가는 최소 hop의 수는 0이다.) 당연히, encoding된 결과는 간결할수록 좋다.

당신은 다음 2개의 함수 encode(N,H,P,A,B)와 decode(N,H)를 구현해야 한다. 여기서 N 은 도시의 수이고, H는 허브의 수이다. 도시는 0 번부터 N-1번까지 번호가 붙여져 있고 그 중에 0 번부터 H-1번까지의 도시를 허브라고 가정하라. 또, N ≤ 1000 이고 H ≤ 36 라고 가정하라. P는 항공기로 연결된 도시 쌍의 개수이다. 모든 도시 쌍들은 (순서가 바뀌는 것을 고려해도) 다르게 주어질 것이다. A와 B 는 연결된 도시 정보를 가지고 있는데 (A[0],B[0])가 연결된 도시의 쌍이고 (A[1],B[1])가 연결된 도시의 쌍이며 이런 방식으로 저장되어 있다.

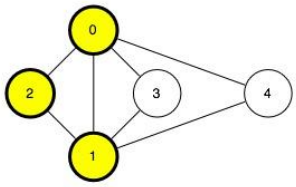
encode 함수는 하나의 비트열을 생성하는데 나중에 decode 함수가 이 비트열로부터 각 도시에서 각 허브까지의 최소 hop의 수를 알아 낼 수 있어야 한다. encode함수는 생성한 비트열을 채점서버로 전송해야 하는데 이를 위해서 encode_bit(b) 함수호출을 여러 번 사용한다. 여기서 b 는 0 또는 1이다. decode함수는 채점서버로부터 비트열을 전송 받아야 하는데 이를 위해 decode_bit 함수호출을 사용한다. *i*번째 decode_bit 함수 호출은 *i*번째 encode_bit(b) 함수를 호출할 때 사용한 b값을 가져올 것이다. decode함수가 decode_bit 함수를 호출하는 횟수가 encode 함수가 encode_bit(b) 함수를 호출하는 횟수와 동일해야 함을 명심하라.

decode 함수가 최소 hop의 수를 알아낸 후에 모든 허브 h와 모든 도시 c (허브 도시도 포함한다.)에 대해서 알아낸 최소 hop의 수 d를 가지고 반드시 hops(h,c,d) 함수를 호출해야 한다. 다시 말해서 총 N*H 번만큼 hops(h,c,d)함수를 호출해야 한다. 호출하는 순서는 중요하지 않다. 모든 허브와 모든 도시 사이에는 직접 연결되지 않아도 몇 개의 도시를 거치면 항상 항공 운송이 가능하다.

주의사항: encode함수와 decode함수는 주어진 인터페이스만을 사용해서 데이터를 주고 받아야 한다. 전역변수(Shared variables)나 파일접근(file access)이나 네트워크 접근 (network access)은 금지된다. C나 C++에서는 encode 나 decode함수 내에서 static 변수를 사용하여 영속적으로 정보를 보존할 수 있다. 파스칼에서는 implementation 부분에서 영속변수를 선언할 수 있다.

예제

오른쪽의 다이어그램은 5 개의 도시 (N=5) 가 7 개의 항공기 (P=7) 로 연결된 예를 보여주고 있다. 도시 0, 1, 2 는 허브다 (H=3). 허브 0에서 도시 3사이에는 1개의 hop이 필요하고 허브 2와 도시 3사이에는 2번의 hop이 필요하다. 이 예의 data는 grader.in.1 에 들어있다.



아래 테이블에는 decode 함수가 hops(h,c,d) 함수를 호출할 때 사용해야 하는 d 값들이 들어있다.

D		도시 c				
		0	**1**	**2**	**3**	**4**
허브 h	**0**	0	1	1	1	1
	1	1	0	1	1	1
	2	1	1	0	2	2

서브태스크 1 [25 점]

encode 함수는 16 000 000 번 이하로 **encode_bit(b)** 를 호출해야 한다.

서브태스크 2 [25 점]

encode 함수는 360 000 번 이하로 **encode_bit(b)** 를 호출해야 한다.

서브태스크 3 [25 점]

encode 함수는 80 000 번 이하로 **encode_bit(b)** 를 호출해야 한다.

서브태스크 4 [25 점]

encode 함수는 70 000 번 이하로 **encode_bit(b)** 를 호출해야 한다.

구현 시 주의사항

- [redacted]

- 참가자가 작성할 파일:
  - encoder.c 또는 encoder.cpp 또는 encoder.pas
  - decoder.c 또는 decoder.cpp 또는 decoder.pas
- 참가자 인터페이스:
  - encoder.h 또는 encoder.pas
  - decoder.h 또는 decoder.pas
- 채점프로그램(grader) 인터페이스: grader.h 또는 graderlib.pas
- 견본 채점프로그램(sample grader): grader.c 또는 grader.cpp 또는 grader.pas 그리고 graderlib.pas
- 견본 채점프로그램 입력(sample grader input): grader.in.1 grader.in.2 등

- *유의사항 : 각 파일의 첫 줄은 N P H 를 포함한다. 그 다음 P 개의 줄은 도시의 쌍 A[0] B[0], A[1] B[1], 등을 포함한다. 그 다음 H*N 개의 줄은 각 허브에서 각 도시까지의 hop의 수를 포함한다. (자기 자신과 모든 다른 허브를 포함한다.), 즉, 허브 i에서 도시 j 까지의 hop의 수는 H*N 개의 줄 중에서 i*N+j+1 번째 줄에 있다.*

• 견본 채점프로그램 입력(sample grader input)의 예상결과(expected output): 채점프로그램은 grader.out.1 grader.out.2 등을 다음과 같이 생성한다.

- 서브태스크 1에 대하여 바르게 구현되었으면, 출력에 OK 1이 포함될 것이다.
- 서브태스크 2에 대하여 바르게 구현되었으면, 출력에 OK 2가 포함될 것이다.
- 서브태스크 3에 대하여 바르게 구현되었으면, 출력에 OK 3이 포함될 것이다.
- 서브태스크 4에 대하여 바르게 구현되었으면, 출력에 OK 4가 포함될 것이다.