2019

고려대학교 프로그래밍 경시대회

평생 새내기가 되고 싶은 유신이

- 타 전공/신입생 부문 A번 문제
- 출제자: 안수빈

평생 새내기가 되고 싶은 유신이

단순하게 반복문을 사용해도 되고, 수식으로 써도 됩니다.

N / 4 * 4

피카츄 라이츄

- 타 전공/신입생 부문 B번 문제
- 출제자: 김도재

피카츄 라이츄

대회가 열리는 날이 1 ~ 100 일 사이에 존재하기 때문에 이후에 추가적인 경험치 획득은 없을 것입니다.

따라서, **1 ~ 100 일 사이에 획득하는 경험치를 구하여 라이츄가 될 수 있는지 확인** 하면 됩니다.

주의할 점은, 낮에 경험치를 얻고 밤에 경험치를 잃기 때문에 대회가 끝나고 밤에 되기 전에 라이츄로 진화할 수 있는지 여부를 확인해야 합니다.

또, 보유중인 경험치량이 음수가 되지는 않았는지 확인해야 합니다.

KCPC는 무엇의 약자일까?

- 타 전공/신입생 부문 C번 문제
- 출제자: 박홍빈

KCPC는 무엇의 약자일까?

KCPC와 입력으로 받은 문자열A가 문자열 S를 축약해서 만들 수 있는지 묻는 문제

각각 문자열에 대해 S를 돌면서 순서대로 문자열 X를 만들 수 있는지 체크

초코칩 케이크

- 타 전공/신입생 부문 D번 문제
- 출제자: 이상헌

초코칩 케이크

0이 쓰여 있는 N*N 격자가 있고, 가로줄이나 세로줄 하나를 잡아서 1을 더하는 과정을 반복하자

매 과정이 끝날 때 가장 큰 수는 몇 개의 칸에 있을까?

초코칩 케이크

가로줄과 세로줄을 따로 생각해보자

최댓값은 어디에 있을까?

- 가장 많이 더해진 가로줄과
- 가장 많이 더해진 세로줄이 겹치는 칸에만 존재한다

초코칩 케이크

i번째 가로/세로줄에 몇 번 더했는지 저장하고,
i번 더해진 가로줄/세로줄의 개수를 저장하자

매번 초코칩을 뿌릴 때마다 위의 값을 갱신하고

가로/세로 최댓값 또한 갱신한 뒤 (0 또는 1 증가)

(가장 많이 더해진 가로줄) * (가장 많이 더해진 세로줄)을 출력

함정에 빠진 이동관

- 타 전공/신입생 부문 E번 문제
- 일반 부문 A번 문제
- 출제자: 이동관

함정에 빠진 이동관

BFS 알고리즘 문제입니다!

BFS 알고리즘은 가능한 모든 노드를 방문하기 때문에 -1인지 아닌지 파악할 수 있습니다.

또, 모든 거리의 가중치가 1이므로 BFS를 이용하여 최단 경로 또한 구할 수 있습니다.

대자보

- 타 전공/신입생 부문 F번 문제
- 일반 부문 B번 문제
- 출제자: 공인호

대자보

- dp(i , k) : i ~ n의 수열을 알파벳 k로 시작하여 해석할 수 있는 단어의 수

- dp(i, k)에서 t(=1~26)에 대해, k*t가 (a[i])이거나
  (a[i]*10+a[i+1])이거나 (a[i]*100+a[i]*10+a[i])인 경우에
  각각 dp(i+1, t), dp(i+2, t), dp(i+3, t)를 호출하면 된다.

신앙

- 타 전공/신입생 부문 G번 문제
- 일반 부문 C번 문제
- 출제자: 양유신

신앙

Naive)

가능한 모든 빨간약 분배 케이스에 대해 최댓값 구하기

$$_nH_R * nlogn$$

신앙

최적해는 빨간약을 한 사람에게 몰아서 먹인 경우이다.

n명이 각각 빨간약을 먹었을 때 가능한 최댓값 구하기

파란약은 (신앙-헌금)이 큰 사람순으로 먹인다
단, 신앙<헌금이면 파란약을 먹이면 안 된다

O(n*nlogn) (O(nlogn)으로 가능)

얼마나 예뻐?

- 타 전공/신입생 부문 H번 문제
- 일반 부문 D번 문제
- 출제자: 김도재

얼마나 예뻐?

<올바른 괄호 문자열 판별>

기본적으로 여는 괄호의 개수와 닫는 괄호의 개수가 같아야 합니다.

앞에서부터 보면서 닫는 괄호 개수가 여는 괄호의 개수보다 많아지는 순간이 있으면 안됩니다.

예시로 "(( ))) (" 의 경우 3번째 닫는 괄호에서 닫는 괄호의 개수가 더 많기 때문에 올바른 괄호 문자열이 아닙니다.

얼마나 예뻐?

<올바른 괄호 문자열의 일반화>

여는 괄호를 +1, 닫는 괄호를 −1로 대응하여 prefix_sum 중 최솟값을 구합시다.

Prefix_Sum 의 최솟값이 0이면서 total_sum 이 0인 것 노드를 '아름다운 노드' 라고 정의할 수 있습니다.

자손 노드들의 prefix_sum 과 sum 을 통해서 현재 노드의 prefix_sum 과 sum을 업데이트 할 경우, O(NlogN)의 시간복잡도로 해결 가능할 것입니다.

ㅇㅇ

- 타 전공/신입생 부문 I번 문제
- 출제자: 손민철

ㅇㅇ

<아이디어>

한 개의 점을 포함하는 원은 그냥 (x−1, y, 1) 혹은 (x+1, y, 1)을 통해 쉽게 만들 수 있다.
(−1,000,000~1,000,000 범위에 유의)

두 개의 점을 포함하는 원은 두 점의 거리가 1.1 이상이라는 조건이 있으므로 그냥 중점을 잡으면 된다.

세 점이 주어질 때, 삼각형의 외심은 유일하다.(세 점이 일직선일 경우 문제가 발생하나 그런 입력은 존재하지 않음이 보장된다.)

n(>3)개의 점이 주어질 때 해당 점들이 한 원에 올려질 수 있는지를 판단하려면 임의의 세 점을 잡아 외심을 찾고, 그 외심으로 만든 원에 모든 점을 올릴 수 있는가 확인하면 된다.

ㅇㅇ

외심 구하는 법 1 : 좌표기하, 솔직히 선 넘음;;

※ 참고사항

세 점 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$을 지나는 외접원의 중심(외심) (x, y)의 좌표는 아래와 같다.

$$x = -\frac{(x_2^2 - x_1^2 + y_2^2 - y_1^2)(y_3 - y_2) - (x_2^2 - x_3^2 + y_2^2 - y_3^2)(y_1 - y_2)}{2(x_1 - x_2)(y_3 - y_2) - 2(x_3 - x_2)(y_1 - y_2)}$$

$$y = -\frac{(y_2^2 - y_1^2 + x_2^2 - x_1^2)(x_3 - x_2) - (y_2^2 - y_3^2 + x_2^2 - x_3^2)(x_1 - x_2)}{2(y_1 - y_2)(x_3 - x_2) - 2(y_3 - y_2)(x_1 - x_2)}$$

ㅇㅇ

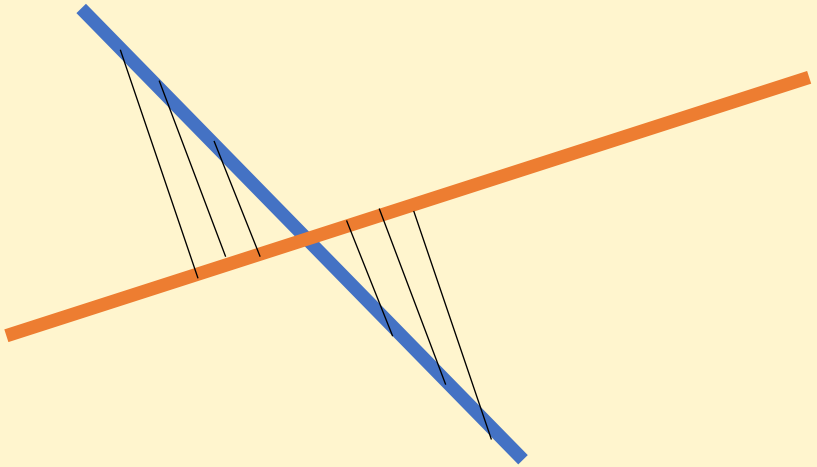
외심 구하는 법 2 : 두 수직이등분선의 교점 O(1)에 구하기

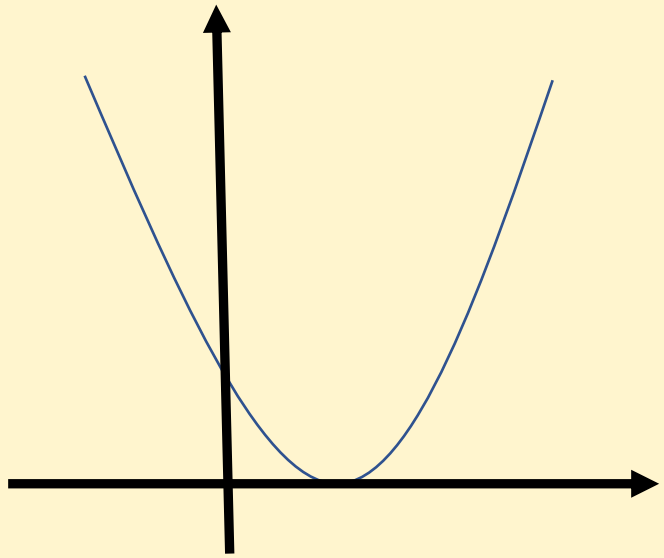
```
   double A, B  E;   // ax + by = e
double C, D, F;   // cx + dy = f

A = y2 - y1;
B = x1 - x2;
E = (y2 - y1)*x1 + (x1 - x2)*y1;

C = y4 - y3;
D = x3 - x4;
F = (y4 - y3)*x3 + (x3 - x4)*y3;

double DE = (A*D) - (B*C);

if(DE==0) {
    System.out.println("Parallel");
} else {
    double X = ((E * D) - (B * F)) / DE;
    double Y = ((A * F) - (E * C)) / DE ;
}
```

```
T sq(pt p){ return p.x*p.x + p.y*p.y; }

T cross(pt v, pt w){ return v.x*w.y - v.y*w.x; } // 외적

pt perp(pt p){return {-p.y, p.x}; }

cir outer_circle(pt a, pt b, pt c){
  b = b-a, c= c-a;
  if(cross(b,c) == 0) return {{-1e7l,-1e7l},-1e7l};
  pt O{a + perp(b*sq(c) - c*sq(b))/cross(b,c)/2.0f};
  return {O, abs(O-a)};
}
```

ㅇㅇ

외심 구하는 법 3 : 두 수직이등분선의 교점 O(log)에 구하기

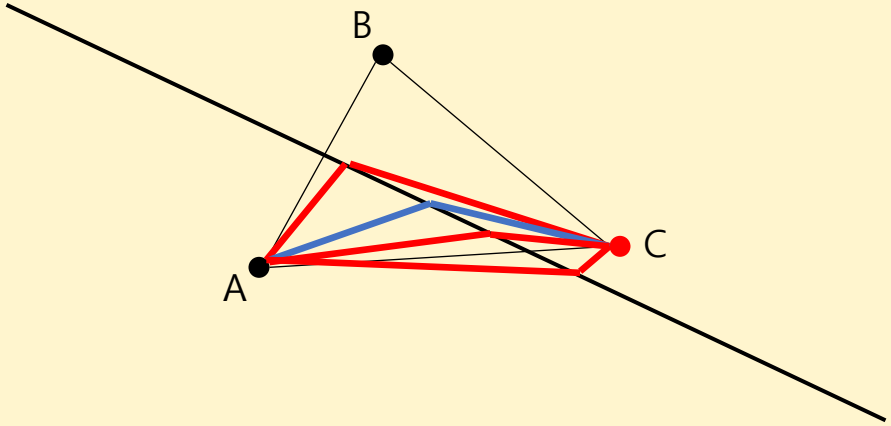




어느 한 선 위의 점들에 대해 다른 선까지의 거리는 아래로 볼록한 그래프이므로 삼분탐색을 이용해 교점을 구할 수도 있다. 팀노트가 없으니 이것도 괜찮은 방법이다.

ㅇㅇ

외심 구하는 법 3 : 이분탐색으로 외심을 찾기



AB의 수직이등분선 위에서 A까지의 거리 / C까지의 거리의 대소를 가지고 이분탐색을 진행할 수 있다. 마찬가지로 팀노트 없이 충분히 시도할 수 있는 방법이다.

ㅇㅇ

<O(2N)에 푸는 방법>

n개의 점을 2개의 그룹으로 쪼갠다.

각 그룹이 한 원에 올려지는지 판단한다.

ㅇㅇ

<O(N)에 푸는 방법>

비둘기집의 원리에 의해 5개의 점을 잡으면 적어도 3개의 점은 한 원에 속한다.

임의의 5개의 점을 잡아서 5 combination 3 조합을 다 보면서 일단 원 한 개를 정한다. 이후 그 원에 포함되지 않는 나머지 점들을 한 개의 원에 다 올릴 수 있는지 확인한다.

ㅇㅇ

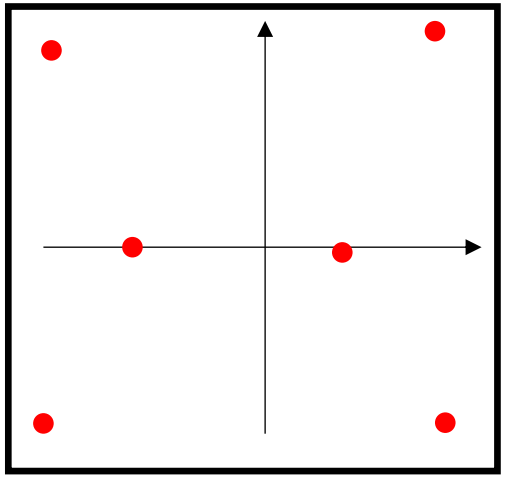
<주의사항>

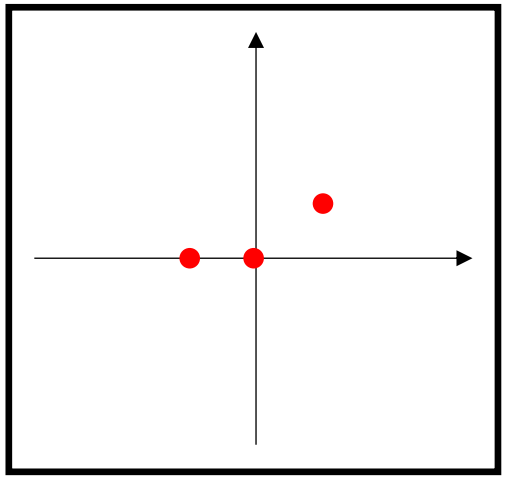
세 점의 외심이 −1,000,000 ~ 1,000,000 범위에 들어오는지 확인해야 한다.

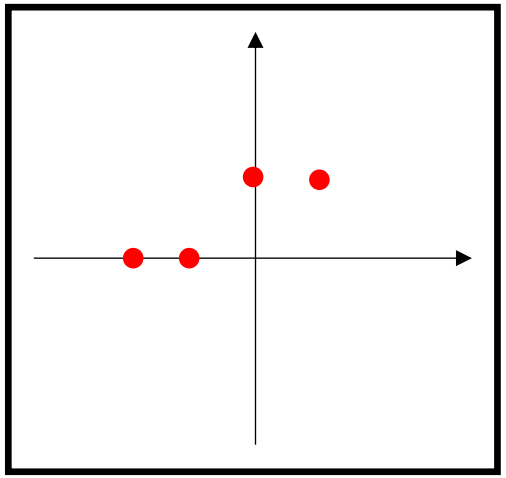
한 개의 점을 포함하는 원을 만들 때 −1,000,000~1,000,000 범위를 벗어나지 않게 주의해야 한다.

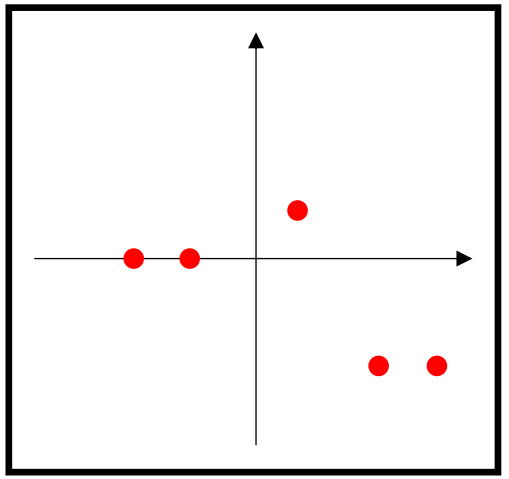
유효숫자 자리수가 작은 자료형(Ex : float)을 사용할 경우 실수오차로 인해 틀릴 가능성이 크다.

ㅇㅇ

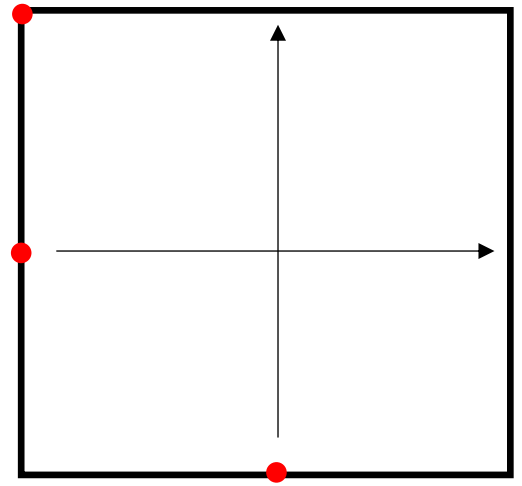


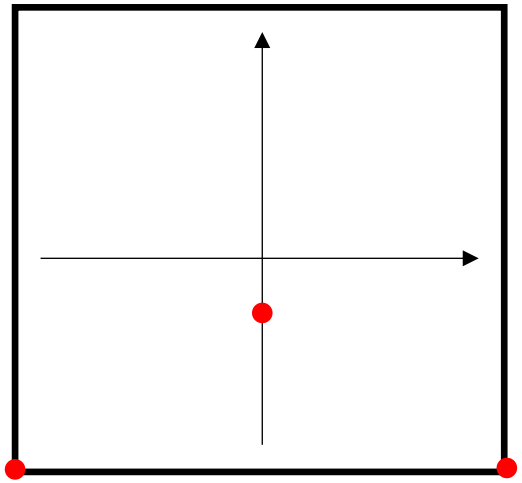


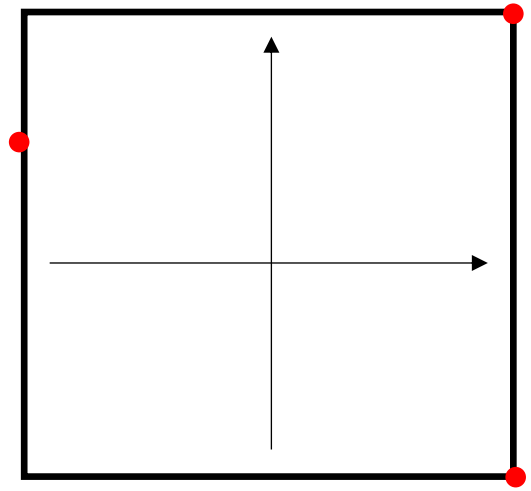


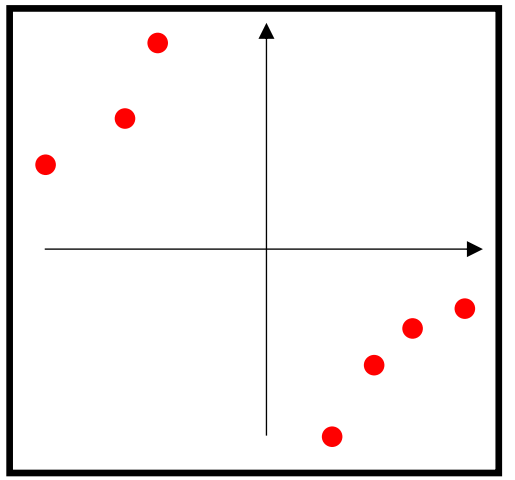


ㅇㅇ









ㅇㅇ

<TMI>

원래 문제는 원이 3개, 점이 10만개인 문제이고 점의 각도 조건도 없어서 일직선에 대한 예외처리가 필요했다.

Sum without Carries

- 일반 부문 E번 문제
- 출제자: 김민성

Sum without Carries

핵심 키워드

- Offline Queries
- Prefix Sum
- Trie

Sum without Carries

Offline Queries

-쿼리 간의 dependency가 없기 때문에, 어느 쿼리를 먼저 해결하더라도 상관이 없다. 해당 쿼리가 몇번째로 들어온 질문이었는지 기억하기만 하면 된다.

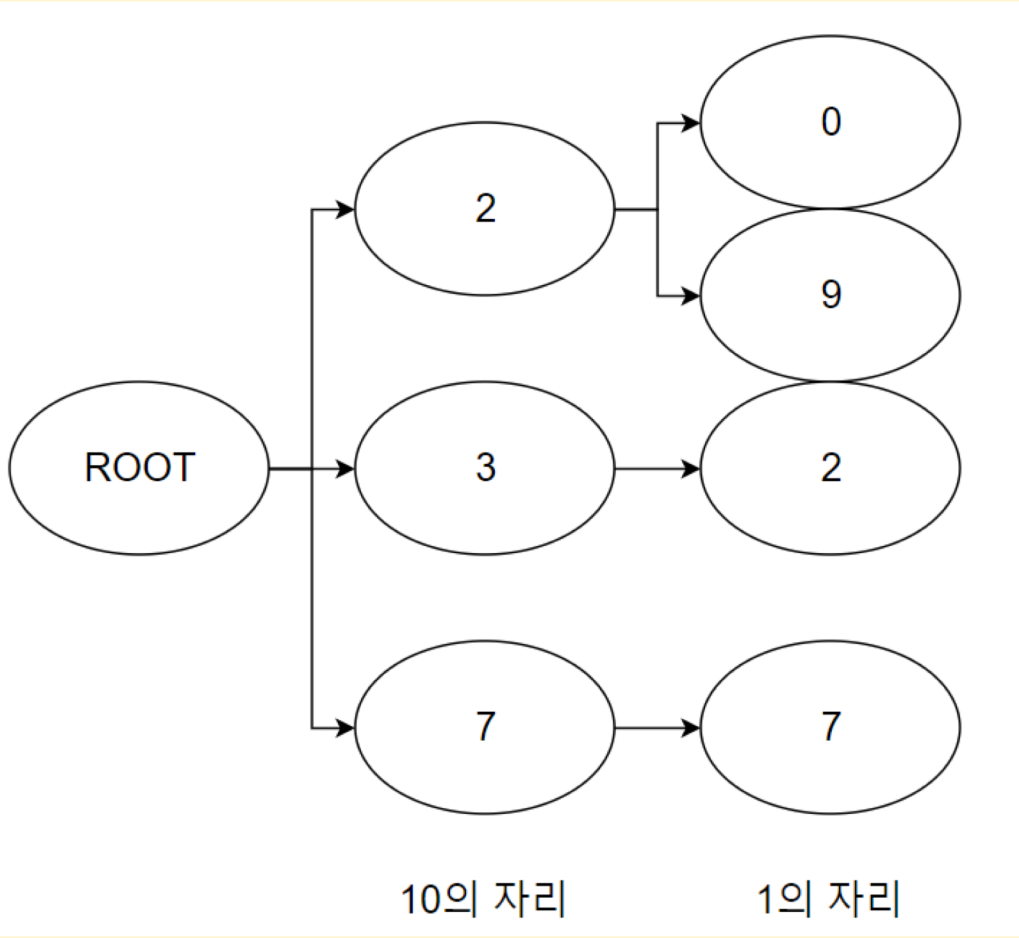
-각 쿼리는 L 하한선과 R 상한선을 가지고 있다. L을 기준으로 하든, R을 기준으로 하든 정렬을 해서 생각해보자. 정해는 R을 기준으로 정렬하였는데, R이 작은 쿼리부터 처리하면 어떤 일이 일어날까?

Sum without Carries

Trie

-@ 연산의 특성상, 각 자릿수를 독립적으로 판단하는 게 가능하다. 좀 다른 관점에서 @ 연산을 바라보면, 이건 일종의 문자열 연산 비슷하게 생각할 수도 있다.

-오른쪽 예시는 10진법 숫자 [32, 29, 77, 20]을 가지고 Trie를 만들었을 때 생기는 구조이다.



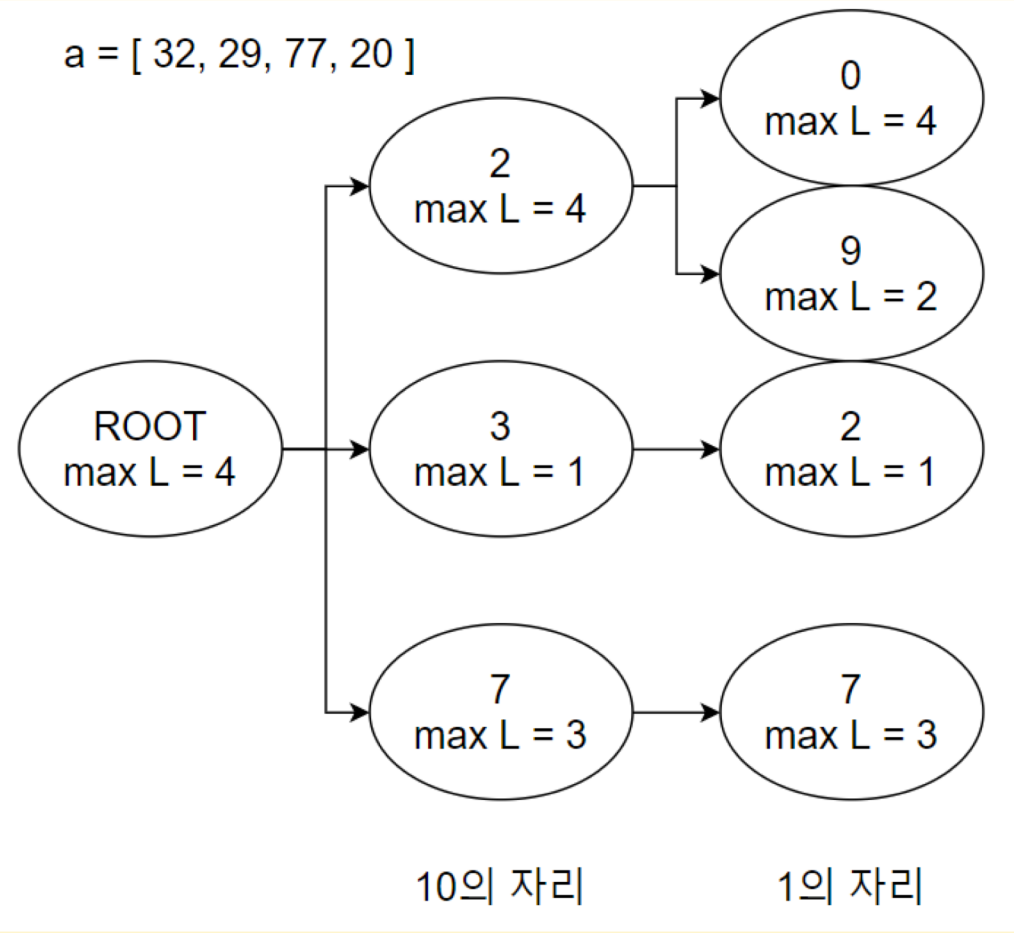
Sum without Carries

Trie를 응용해보자

-Trie를 정적으로 두지 말고, 다이나믹하게 관리하자.

-해당 노드에 추가적인 정보(max L)를 기입하자. 어떤 노드에 x[i1], x[i2], …, x[ik]가 들어가있다면, 해당 노드의 max L = max(i1, i2, …, ik)로 하자.

-그렇다면, 우리가 전체 트라이에 어떤 k개의 정보 x[1], x[2], …, x[k]를 넣었을 때, 각 노드들이 가지고 있는 x에서의 최신 인덱스들을 알 수 있다!



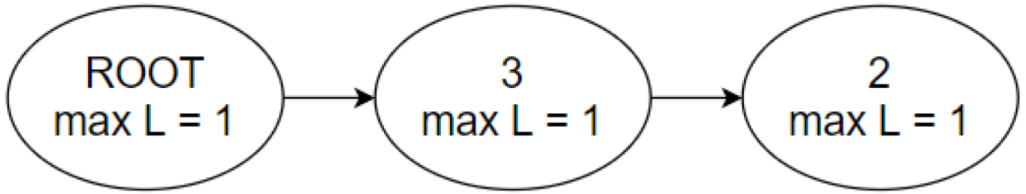
Sum without Carries

Progressive Update Example

-편의상 d=10 (10진법) 이라고 가정하자. 그리고 이 트라이에서는 그냥 a를 넣는다고 가정하자.

-처음에 32가 들어갔다.

a = [32]

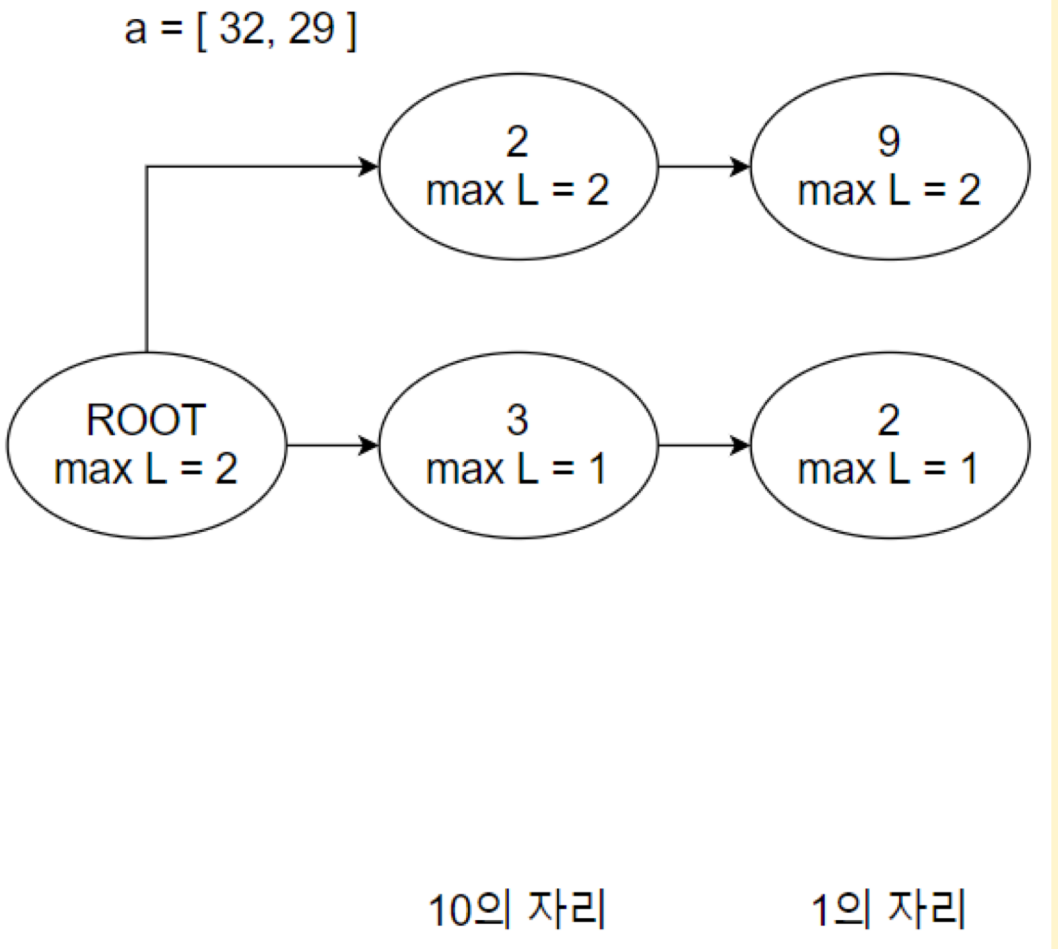


10의 자리 1의 자리

Sum without Carries

Progressive Update Example

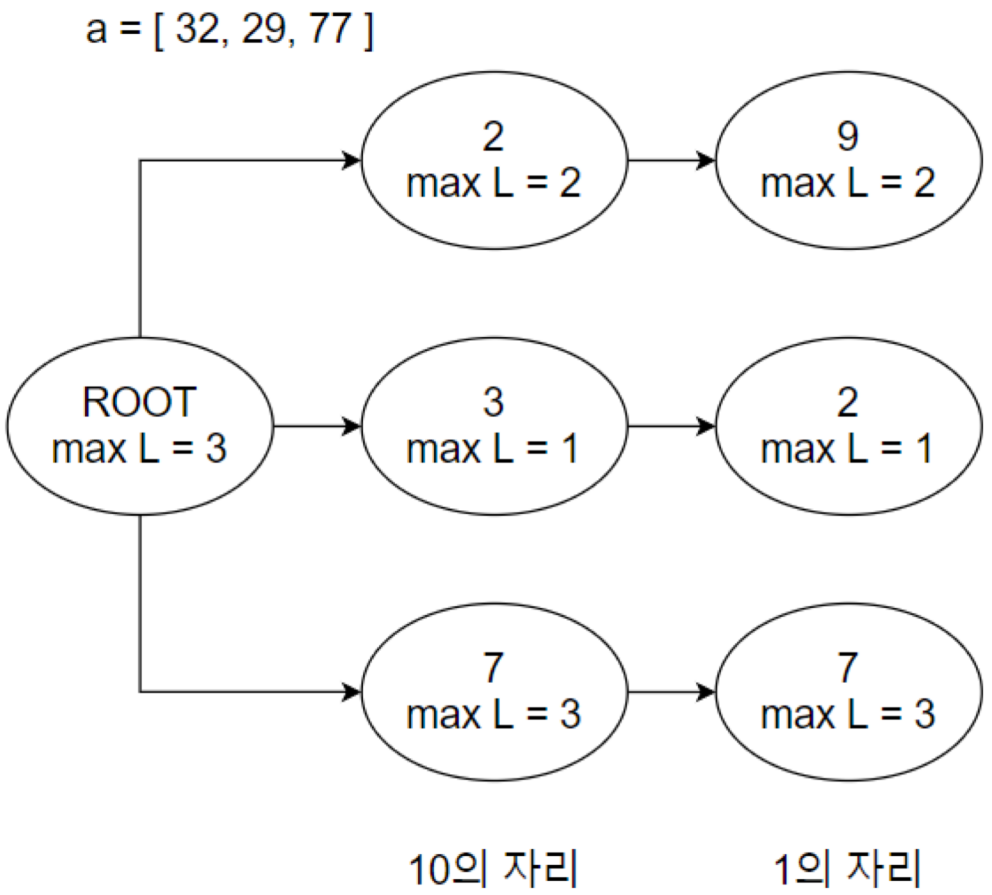
-그 다음에 29가 들어가서 Root 노드의 max L이 갱신되었다.



Sum without Carries

Progressive Update Example

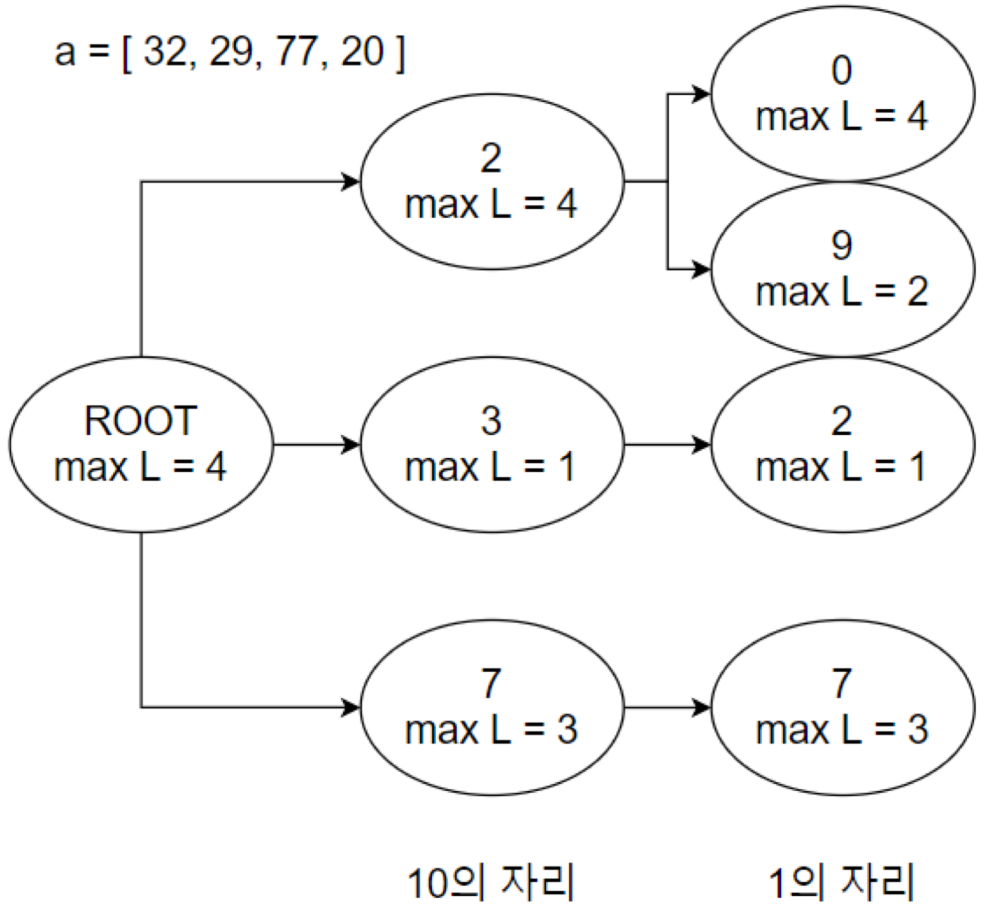
-그 다음에 77이 들어가서 Root 노드의 max L이 갱신되었다.



Sum without Carries

Progressive Update Example

-그 다음에 20가 들어가서 Root 노드,
그리고 20에 해당하는 노드의 max L
이 갱신되었다.



Sum without Carries

이렇게 Trie를 빌드하면..

-매 순간순간마다 특정 값 이하인 x값 중에서, L 값이 특정 값 이상인 수를 찾는 것이 O(depth * d) 안에 가능하다. (여기서 depth = log(max(a))로 설정할 수 있다)

-다시 말하면, 아까와 같은 구현 방식을 이용하여 원하는 값을 서서히 증가하는 R의 상한에 따라 원하는 범위 내에서 추적하는 것이 자유로워졌다.

Sum without Carries

그런데 여기서 Prefix Sum을 사용해야 한다.

-아까의 설명에서는 편의 상 a를 넣어주었다. 이 문제를 풀기 위해서는 a 그 자체를 넣는 대신에 a의 prefix sum을 Trie 안에 넣어주어야 한다. 단순히 Prefix sum을 넣어도 되는 이유는, **@ 연산의 뺄셈 버젼은 @ 연산의 덧셈 버젼의 역방향 작용을 하기 때문이다.**

Sum without Carries

시간복잡도 = O(n*d*log(max(a)))

서버 증축

- 일반 부문 F번 문제
- 출제자: 이상헌

서버 증축

	0	1	2	…	K-2	K-1
1	1	2	4	…	2^{K-2}	2^{K-1}
2	1	2	4	…	2^{K-2}	2^{K-1}
…	…	…	…	…	…	…
a	1	2	4	…	2^{K-2}	2^{K-1}

합이 n이 되게 고르는 가짓수 mod $2^{20} - 3$?

(a는 10^6 이하, k는 40 이하, n은 10^{15} 이하)

전처리로 $x!, (x!)^{-1} \, mod \, p$를 $O(p)$ 정도에 계산

서버 증축

	0	1	2	...	K-2	K-1
1	1	2	4	...	2^{K-2}	2^{K-1}
2	1	2	4	...	2^{K-2}	2^{K-1}
...	...	...	...	...	...	...
a	1	2	4	...	2^{K-2}	2^{K-1}

k가 작으니까 2^i 동전들 기준으로 생각해볼까?

$O(ka^2)$ DP, FFT를 쓰면 $O(ka \lg a)$ 시간복잡도 가능

하지만 충분해보이진 않는다...

서버 증축

	0	1	2	…	K-1	…
1	1	2	4	…	2^{K-1}	…
2	1	2	4	…	2^{K-1}	…
…	…	…	…	…	…	…
a	1	2	4	…	2^{K-1}	…

k가 무한대라고 생각하고, a 기준으로 생각

이 때 답은 어떻게 될까?

서버 증축

	0	1	2	...		K-1	...
1	1	2	4	...		2^{K-1}	...
2	1	2	4	...		2^{K-1}	...
...	...	...	...	...		...	...
a	1	2	4	...		2^{K-1}	...

i번째 행의 합이 x_i가 되게 하는 방법은 1가지 뿐

$x_1 + x_2 + x_3 + \cdots + x_a = n$???

중복 조합식이 나온다! $\frac{(a+n-1)!}{(a-1)!n!}$

다만, 2^k를 넘어가는 값들도 포함된다

서버 증축

	0	1	2	...	K-2	K-1
1	1	2	4	...	2^{K-2}	2^{K-1}
2	1	2	4	...	2^{K-2}	2^{K-1}
...	...	...	...	...	...	...
a	1	2	4	...	2^{K-2}	2^{K-1}

포함 배제의 원리를 이용하자

0 – 1 + 2 – 3 + 4 – ...를 반복

답은 $\sum_{i=0}^{a}(-1)^i\binom{a}{i}\binom{a+(n-i\cdot 2^k)-1}{a-1}$, 시간복잡도 $O(a \lg n/p)$

생성함수를 이용해도 동일한 식 유도 가능

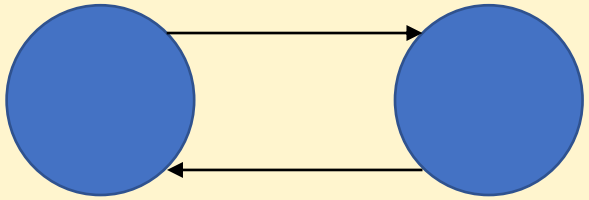
기묘한 여행계획 2

- 일반 부문 G번 문제
- 출제자: 강인구

기묘한 여행계획 2

- <트리일 때의 접근>
- 그래프가 아니라 트리라는 특수한 형태일 때를 고려해보자. 이 때 임의의 두 경로 상의 비용은 항상 일정하다.

- 이는 같은 경로를 두번 이용할 경우 그 도로의 사용 비용이 취소되기 때문이고, 두 번 사용했다는 것은 a->b->…->b->a와 같은 꼴로 원위치로 복귀했다는 의미이다.



기묘한 여행계획 2

- <일반적인 그래프에서의 풀이>
- 주어진 그래프에서 임의의 spanning tree를 구하자. 이 때 이 tree에서 사용된 edge를 tree edge라고 하고, 그 외의 edge를 non-tree edge라고 하자.

- tree edge만을 이용했을때의 비용은 트리일때의 접근과 동일하므로 비용이 정해진다. 이 때의 비용을 V라고 하자.

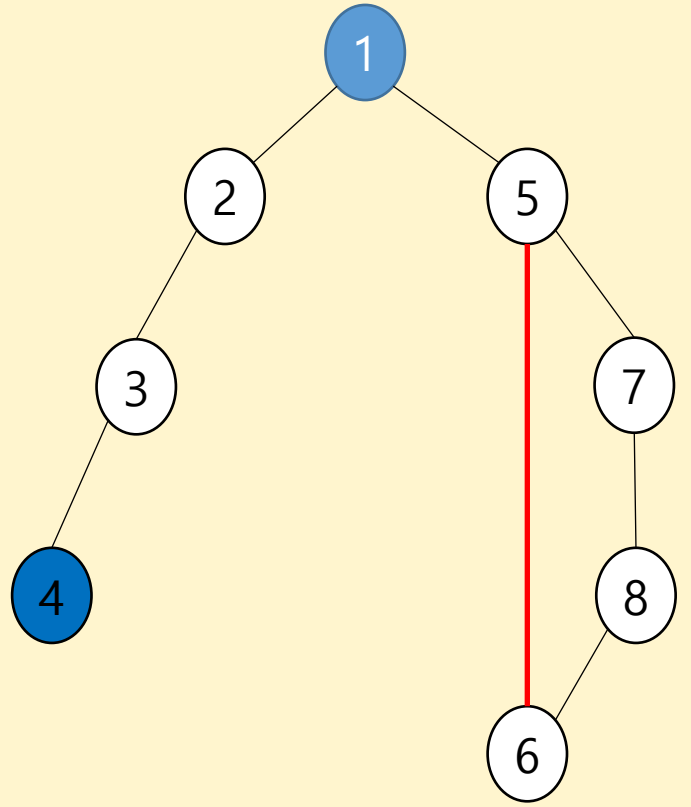
- non-tree edge를 이용하는 비용은 해당 non-tree edge랑 tree edge들로 구성된 사이클을 V에 xor해주는 것과 동일하다.(그림 다음 페이지)

기묘한 여행계획 2

- <일반적인 그래프에서의 풀이>
- 오른쪽 그래프에서 가능한 cost(1,4)의 경우

- cost(1,4) = 1~4까지 연결하는 edge

- cost(1,4) = edge(1,5) ^ (5,7,8,6)으로 이루어진 사이클에 포함된 edge ^ edge(5,1) ^ 1~4까지 연결하는 edge
  = 사이클 ^ 1~4까지 연결하는 edge



기묘한 여행계획 2

- <일반적인 그래프에서의 풀이>
- 즉, tree edge만으로 이루어진 path의 비용에 non-tree edge로 만들어진 사이클의 비용을 자유롭게 xor했을 때 얻을 수 있는 최소 비용을 구하는 문제가 된다.

- 사이클의 비용을 x1, … xk라고 했을때 이를 이진수로 나타낸 vector로 생각해서 택 basis를 구하는 문제로 변환할 수 있다.

- 총 시간복잡도는 O(NlogMAX)이다.

수고하셨습니다.

- 풀이 자료는 추후 github 페이지에 공유하겠습니다.
- https://github.com/KCPC-info/2019KCPC

- 시상식 이후 햄버거가 지급될 예정입니다.