SNUPC 2025 풀이

Official Solutions

by

SNUPC 2025 출제진

2025년 9월 13일

A. 예티와 주사위 던지기

- `expected-value, brute-force`
- 출제자: benedict0724

A. 예티와 주사위 던지기

- 한 번의 기회에서는 어떤 주사위를 다시 던질지만 결정하면 충분합니다. 이후 족보 선택은 최종 조합을 본 다음 최댓값을 고르면 됩니다.
- 최종 조합이 주어지면 획득 점수는

$$\max\left(\underbrace{50\ (\text{모두 같을 때})}_{\text{Yacht}},\ \max_{i=1}^{6} i \times \#\{i\} \right).$$

- 따라서, 각 재던지기 집합 $R \subseteq \{1, \ldots, 5\}$에 대해

$$\mathbb{E}[\text{최종 점수}]\text{을 계산하고 그중 최댓값을 택하면 됩니다.}$$

A. 예티와 주사위 던지기

- 선택한 집합 R에 대해, 벡터 v에서 다시 던질 위치를 0으로 표시합시다.
- 인덱스 $h = 0 \ldots 4$에 대해 다음을 재귀로 계산합니다.

$$S(h, v) = \begin{cases} \text{score}(v) & (h = 5) \\ \displaystyle\sum_{x=1}^{6} S(h{+}1, v[h]{\leftarrow}x) & (v[h] = 0) \\ 6 \cdot S(h{+}1, v) & (v[h] \neq 0) \end{cases}$$

- 여기서 $S(0, v)$는 해당 R에서의 기댓값 $\times\ 6^5$ 입니다. 모든 R에 대해 $S(0, v)$의 최댓값이 정답입니다.
- 모든 부분집합 R을 순회하는 방법의 수는 $2^5 = 32$으로 모두 다 시도해보아도 됩니다.

B. 여우 덧셈

- `digit-dp`
- 출제자: ksoosung77

B. 여우 덧셈

- N의 각 자릿수에는 최소 1개이상의 S의 자릿수가 들어감을 알 수 있고, 들어가는 자릿수는 연속되며 N의 다른자릿수에는 들어가지 않는다는것을 알 수 있습니다.
- 우리는 이러한 발상으로 dp에 대해 생각해 볼 수 있습니다. dp[S의 현 위치 (0 |S|-1)][N의 현 위치 (0 |N|-1)][N의 현위치에 S의 자릿수들을 더한상태 % 10] = 최소 0으로 변경한 횟수
- 이러한 구조로 3중 반복문을 돌면서 dp를 채울때 아래와 같은 4가지 조건을 고려하여 최솟값인지 확인하며 채우면 됩니다.

B. 여우 덧셈

1. 현재 S의 위치에 있는 자릿수를 더하기만 할때

`dp[S_i][N_i][digit] -> dp[S_i+1][N_i][(digit + S[S_i])%10]`

2. 현재 S의 위치에 있는 자릿수를 더했더니 N의 현재 자릿수와 같아 N의 다음 자릿수로 넘길때.

`dp[S_i][N_i][digit] -> dp[S_i+1][N_i+1][0]`

3. 현재 S의 위치에 있는 자릿수를 0으로 변경 후 더하기만 할때

`dp[S_i][N_i][digit] -> dp[S_i+1][N_i][digit] + 1`

4. 현재 S의 위치에 있는 자릿수를 0으로 변경 후 더했더니 N의 현재 자릿수와 같아 N의 다음 자릿수로 넘길때.

`dp[S_i][N_i][digit] -> dp[S_i+1][N_i+1][0] + 1`

- 그렇게 dp를 다 채운 후 dp[$|S|$][$|N|$][0]을 출력하면 우리가 원하는 0의 최소 변경 횟수를 알게 됩니다.

C. 잡아라 벌레 벌레!

- dp
- 출제자: young_out

C. 잡아라 벌레 벌레!

- 오른쪽 끝 r 을 고정합시다. 이때 정답에 더해지는 l 의 개수는 구간 $[1, r]$ 안에서 합이 100이 되는 어떤 부분집합 S 들의 최소 인덱스 $\min(S)$ 들의 최댓값 m_r 와 정확히 같습니다.
- 결국 $\sum_{r=1}^{N} m_r$ 를 한 번의 스캔으로 구하면 됩니다.

dp 정의

- $dp[s]$: 현재까지 본 원소들로 합이 s가 되는 부분집합들 중 $\min(\text{부분집합})$의 최댓값. 만들 수 없으면 0.
- 새 원소 $x = A_i$를 넣기 직전의 $dp[100 - x]$는 x를 더해 합 100을 만들 수 있는 부분집합의 최소 인덱스를 알려줍니다.
- prefix maximum psb를 유지합시다. $psb = \max(psb,\ dp[100 - x])$.
- 그러면 $m_i = psb$, 곧 $[1, i]$에서 가능한 l의 개수이며, 답에 psb를 더합니다.

전이

- 모든 $j = 100, 100-1, \ldots, x+1$에 대해

$$dp[j] \leftarrow \max\big(dp[j],\ dp[j-x]\big).$$

- 또한 단일 원소를 추가할 수 있으므로 $dp[x] \leftarrow \max(dp[x],\ i)$.

D. Apollonian Embedding

- `graph, constructive`
- 출제자: azberjibiou

D. Apollonian Embedding

- 볼록 N각형에 서로 교차하지 않는 $N-3$개의 대각선을 더한 그래프는 모든 내부면이 삼각형입니다.
- 이런 그래프에는 항상 차수 2의 정점이 존재하며, 그 두 이웃은 서로 인접합니다.
- 차수 2의 정점를 하나씩 제거하면 언젠가 삼각형만 남습니다.
- 제거 순서를 거꾸로 재생하면, 매번 어떤 삼각형 면에 새 정점을 얹는 과정과 정확히 일치합니다.
- 즉, 차수 2의 정점 제거의 역과정 = Apollonian network의 생성 과정.

단계 1: 차수 2의 정점 제거 순서 구하기

- 인접 집합 $g[u]$를 집합(set)으로 유지합니다. 초기에는 다각형의 변과 주어진 대각선을 모두 추가합니다.
- 아직 제거하지 않은 정점 중 $|g[i]| = 2$인 i를 하나 고릅니다.
- i의 두 이웃을 u, v로 기록하고, $g[u], g[v]$에서 i를 삭제합니다.
- 위 작업을 $N-3$번 반복하면 세 정점만 남고, 이들이 시작 삼각형이 됩니다.

단계 2: 역재생으로 Apollonian 생성

- 제거 순서를 뒤집어 각 차수 2의 정점 i를 다시 넣습니다. 이때 기록된 두 이웃을 u, v라 두면, 현재 그래프에서 uv는 여전히 간선입니다.
- u와 v의 공통 이웃 w를 하나 찾습니다. 그러면 삼각형 면 uvw가 존재합니다.
- 면 uvw를 제거하고 새 정점 i를 추가하며 간선 iu, iv, iw를 추가합니다.
- 이를 그대로 한 줄로 $u\ v\ w\ i$ 형태로 출력하면 요구 형식과 일치합니다.

정당성

- 차수 2의 정점 제거 시 uv는 항상 남습니다. 역재생 순간의 그래프는 귀납적으로 삼각분할이므로, uv는 어떤 삼각형 uvw의 변이 되어 공통 이웃 w가 반드시 존재합니다.
- 각 단계에서 i를 u, v에 연결하므로, 원래 그래프의 모든 간선은 시작 삼각형의 변이거나 어떤 단계에서 이미 등장한 간선으로 유지됩니다. 결과적으로 원 그래프는 최종 Apollonian network의 부분 그래프가 됩니다.

E. LIS 하나 빼기

- `dp, segtree, lis`
- 출제자: ncy09

- 증가 수열의 길이를 최우선, 그 다음 가치를 비교하도록 쌍 $(\text{len}, \text{weight})$ 로 다룹니다.
- 순서쌍에 max operation을 적용하면, 길이가 가장 긴 수열 중에서 가치가 가장 큰 경우를 구할 수 있습니다.
- 먼저 dp를 이용하여 LIS에 대한 정보를 구할 수 있습니다.
- $A[1:i]$ 에 포함되며 A_i 를 포함하는 최대 순서쌍을 L_i, $A[i:N]$ 에 포함되며 A_i 를 포함하는 최대 순서쌍을 R_i 라 정의합니다.

E. LIS 하나 빼기

- 두 값은 다음과 같은 점화식으로 계산할 수 있습니다.

$$L_i = \max_{j<i,\ A_j<A_i} L_j + (1, V_i)$$

$$R_i = \max_{j>i,\ A_j>A_i} R_j + (1, V_i)$$

- 이는 좌표압축을 이용한 세그먼트 트리로 $O(N \log N)$에 계산할 수 있습니다.
- 오른쪽으로 스위핑하며 L_i를 계산하고, 왼쪽으로 스위핑하며 R_i를 계산합니다.
- 전체 수열에서 A_i를 포함하는 최대 순서쌍 C_i는 다음과 같습니다.

$$C_i = (\mathrm{lenL}_i + \mathrm{lenR}_i - 1,\ \mathrm{weightL}_i + \mathrm{weightR}_i - V_i)$$

E. LIS 하나 빼기

- A_i를 제거한 수열에 존재하는 증가 수열은 다음 중 하나입니다.
  - 왼쪽에 있으면서 $A_j > A_i$인 어떤 A_j를 포함하는 기존의 증가 수열
  - 오른쪽에 있으면서 $A_j < A_i$인 어떤 A_j를 포함하는 기존의 증가 수열
  - A_i를 포함하는 기존의 증가 수열에서 A_i만 제거한 것
- 첫 두 경우는 구조적으로 A_i를 포함할 수 없습니다. 또한 두 경우가 모두 아니라 가정하면, 반드시 A_i를 삽입할 수 있는 형태임을 보일 수 있습니다.

E. LIS 하나 빼기

- 이 사실에 따라 A_i 를 제거한 경우의 정답은 다음과 같습니다.

$$res_i = \max\left(\max_{j<i,\ A_j>A_i} C_j,\ \max_{j>i,\ A_j<A_i} C_j,\ (\text{lenC}_i - 1,\ \text{weightC}_i - V_i)\right)$$

- 첫번째 항과 두번째 항은 이전과 유사하게 세그먼트 트리로 계산할 수 있습니다.
- 시간 복잡도는 $O(N \log N)$ 입니다.

F. 코코의 노래

- `suffix array`, `lcp`, `sqrt-decomp`, `sweeping`
- 출제자: kizen

- 목표는 문제에서 정의된 p를 고정해 주기적 패턴의 기여도를 합산하는 것입니다.
- $B = \sqrt{n}$을 잡아 $p < B, p \geq B$로 나눕니다.
- $\operatorname{lcp}(i, j)$: i, j에서 시작하는 접두사 공통 길이. $\operatorname{lcs}(i, j)$: i, j에서 끝나는 접미사 공통 길이.
- 선형 LCP 전처리를 합시다.

Case 1: $p < B$

- 각 p에 대해 뒤에서 앞으로 $dp[i]$를 계산합니다.
- $dp[i] = 1$, 그리고 $\text{lcp}(i, i+2p) \geq p$이면 $dp[i] \mathrel{+}= dp[i+2p]$.
- 위치 i의 기여도는 $a[i-1] \leq dp[i]$일 때 1을 더합니다.
- 한 p당 $\mathcal{O}(n)$, 따라서 $\displaystyle\sum_{p=1}^{B-1} \mathcal{O}(n) = \mathcal{O}(nB)$.

Case 2: $p \geq B$

- 수열을 길이 p 블록으로 나눠 $[0, p-1], [p, 2p-1], \ldots$처럼 본 뒤, 코코가 흉내낼 수 있는 연속 부분 수열의 시작 위치가 $ip \sim (i+1)p-1$에서 개수를 구해봅시다.
- a_s를 $2, 3, 4$처럼 늘리며 이어붙일 수 있는지 판단합니다.
- 이때 두 값의 최솟값을 유지합니다.
- bmn: $\min\left(\text{lcp}((i+1)p, (i+3)p),\ \text{lcp}((i+3)p, (i+5)p),\ \ldots\right)$.
- fmn: $\min\left(\text{lcs}((i+1)p-1, (i+3)p-1),\ \ldots\right)$.
- $\text{fmn} + \text{bmn} < p$가 되는 순간 더 이상의 a_s는 불가하므로 중단합니다.

F. 코코의 노래

- 각 a_s 에 대해 시작 인덱스의 허용 구간 $[st, ed]$ 를 구합시다.
- 이후 오프라인 쿼리로 구간에서 특정 값의 개수를 구하는 쿼리를 모두 처리합니다.
- 시간복잡도는 $\sum_{p=B}^{n} \mathcal{O}\left(\frac{n^2}{p^2}\right) = \mathcal{O}\left(\frac{n^2}{B}\right)$.

F. 코코의 노래

- 전체 시복은 $\mathcal{O}(nB) + \mathcal{O}\left(\frac{n^2}{B}\right)$. $B = \sqrt{n}$을 택해 $\mathcal{O}(n\sqrt{n})$.
- 오프라인 쿼리를 루트번 진행하여 공간복잡도가 $\mathcal{O}(n\sqrt{n})$가 되지 않게 할 수 있습니다.

G. 카드 게임

- `bitmasking, constructive`
- 출제자: pyb1031

- L 이하의 가장 큰 2의 거듭제곱을 2^a라 두고 $k = 2^a$로 표기합니다. 그러면 $k \le L < 2k$가 성립합니다.
- $R < 7k$이면 수비를 선택합니다.
- $R \ge 7k$이면 공격을 선택합니다.

- 수비 전략 ($R < 7k$)
  - $x \in [L, R]$에 대하여

$$\begin{cases} \text{R} & \text{if } 3k \le x < 5k, \\ \text{B} & \text{otherwise.} \end{cases}$$

  - 상위 3 비트 (즉 $\lfloor x/k \rfloor \in \{1, 2, 3, 4, 5, 6\}$)만 보면 색이 다음과 같이 결정됩니다.

$$001 : \text{B}, \quad 010 : \text{B}, \quad 011 : \text{R}, \quad 100 : \text{R}, \quad 101 : \text{B}, \quad 110 : \text{B}$$

수비 전략 ($R < 7k$)

- 같은 색만 고른 세 수의 상위 3 비트 xor가 절대 000 이 되지 않습니다.
  - B 집합 $\{001, 010, 101, 110\}$ 의 xor 전체가 0 이므로 임의의 세 개의 xor는 나머지 하나가 되어 0 이 되지 않습니다.
  - R 집합 $\{011, 100\}$ 에서는 어떤 조합도 000 이 되지 않습니다.

✓ 공격 전략 ($R \geq 7k$)

- 세 수 x, y, z에 대한 다음 7개의 수 집합을 생각해봅시다.

$$S = \{x,\ y,\ z,\ x \oplus y,\ x \oplus z,\ y \oplus z,\ x \oplus y \oplus z\}$$

- 브루트포스를 돌려보면 위 7개의 수를 어떻게 색칠해도 승리 조건을 만족하는 세 수가 존재함을 알 수 있습니다.
- 저 7개의 수가 $[L, R]$에 들어가게 하는 x, y, z를 찾으면 될것입니다.

✓ 공격 전략 ($R \geq 7k$)

- $x = 2k - 1,\ y = 2k,\ z = 7k$ 로 수를 정합니다.
- 이 때 $S = \{2k, 2k-1, 7k, 4k-1, 5k, 7k-1, 5k-1\}$ 가 됩니다.
- $L < 2k$ 이므로 $x = 2k - 1 \geq L$ 이고 $z = 7k \leq R$ 입니다. 따라서 위 7 개 모두 $[L, R]$ 에 들어갑니다.
- 7 개를 질의한 뒤 $\binom{7}{3} = 35$ 개를 전부 확인하여 조건을 만족하는 세 수를 출력하면 됩니다.

H. 공연 준비

- dp
- 출제자: dadas08

Lemma 1

- 항상 다음 꼴의 해가 존재합니다. $i = 1, 2, \ldots, N$ 을 순회하며 i 번 사람을 필요한 만큼 앞으로 당깁시다. 물론 키 조건을 만족해야 합니다.
- 이 Lemma 로 인해 다음과 같은 정의가 가능합니다. $\mathrm{DP}[i][j][\mathrm{state}]$ 를 $1 \sim i$ 명까지 배치했을 때 앞에서 j 명이 보이고, 앞쪽의 순서가 state 인 경우의 최소 스왑 수로 둡시다.
- 다만 state 가짓수가 큽니다. 상태를 직접 들고 가면 계산량이 커집니다.
- 관건은 state 를 줄이는 것입니다. 이를 위해 다음 Lemma 를 사용합니다.

Lemma 2

- $P_i > P_j$ 이고 $i < j$ 라 합시다. i 번이 안 보인다면 j 번도 안 보이게 하는 최적해가 존재합니다.
- j 가 앞으로 나와 수행하는 순간마다 그 동작을 취소하고 i 가 같은 역할을 수행하게 바꾸면 손해가 생기지 않습니다.
- 이 Lemma는 작은 키가 굳이 더 앞쪽으로 보이게 나올 이유가 없음을 보장합니다. 따라서 앞쪽 전역 순서를 전부 기억할 필요가 줄어듭니다.

- $\mathrm{DP}[i][j][k]$ 를 정의합시다.
- 의미는 다음과 같습니다.
  - $1 \sim i$ 번 사람을 Lemma 1 방식으로 배치했습니다.
  - 앞에서 j 명이 보입니다.
  - 뒤에서 연속한 k 명이 모두 보이고, 그 앞의 한 명은 보이지 않습니다.
- 뒤의 k 명이 모두 보이면 그들의 키 집합은 $1 \sim i$ 중 가장 큰 k 명이 오름차순으로 고정됩니다. 따라서 뒤쪽 k 명 정보만으로 충분합니다.

H. 공연 준비

- $i+1$ 번째 사람을 끼우는 선택을 둘로 나눕시다.
  - 보이지 않게 맨 앞이나 특정 위치 앞에 세웁니다. 필요한 스왑 수는 남은 사람들의 상대 위치로 계산합니다.
  - 보이도록 사이에 정확히 끼웁니다. 이때는 현재까지의 큰 키들 사이에 들어가는 자리 수를 세면 됩니다.
- prefix min을 활용하여 $O(N^3)$ 에 구현할 수 있습니다.
- 더 자세한 풀이는 `https://dadas08.tistory.com/21`에서 확인할 수 있습니다.

I. snupc 문자열 (Easy)

- `binary-search, greedy`
- 출제자: young_out

- 고정 패턴이므로 구간 안에서 s를 k개, 이어서 n을 k개, $\ldots$, c를 k개를 왼쪽부터 그리디로 집어넣으면 충분합니다.
- 임의 위치 pos와 문자종류 $a \in \{0, 1, 2, 3, 4\}$에 대해 pos 이후에서 a의 k번째 등장이 일어나는 최소 인덱스를 찾으면 검증이 끝납니다.
- 접두사 개수 $cnt[i][a]$를 두고 조건 $cnt[m][a] - cnt[pos-1][a] \geq k$를 만족하는 최소 m을 이분 탐색으로 찾습니다.

- 패턴 snupc를 $a = 0, 1, 2, 3, 4$로 매핑합니다.
- $cnt[i][a] = S[1..i]$에서 문자 a의 개수를 저장합니다.
- 다음 등장 인덱스를 찾는 $kth(pos, a, k)$는 구간 $[pos, |S|]$에서 이분 탐색으로 최소 m을 찾습니다.
- 만약 $cnt[|S|][a] - cnt[pos - 1][a] < k$이면 해당 단계에서 즉시 실패 판정입니다.

- $psb(l, r, k)$ 를 정의합니다.
- $pos \leftarrow l$ 로 두고 $a = 0$ 부터 4 까지 반복합니다.
- 각 단계에서 $m \leftarrow kth(pos, a, k)$ 를 구해 $pos \leftarrow m$ 으로 갱신합니다.
- 중간에 m 이 존재하지 않거나 $pos > r$ 가 되면 실패입니다. 다섯 문자를 모두 통과하면 성공입니다.
- 한 번의 psb 는 $O(5 \log N)$ 입니다.

- 각 쿼리마다 k를 이분 탐색하며 $psb(l, r, k)$로 검증합니다.
- 쿼리당 시간은 $O(\log N \cdot 5 \log N)$, 전체는 $O(N + Q \log^2 N)$ 수준입니다.

J. SNUPC 문자열 (Hard)

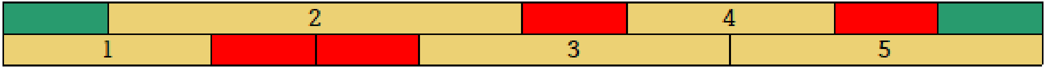
- `eulerian_path`, `linear_algebra`
- 출제자: lunarlity

J. SNUPC 문자열 (Hard)

- 편의상 종이 1에 적혀있는 문자열 조각들을 SN set, 종이 2에 적혀있는 문자열 조각들을 UP set이라고 하겠습니다. 중복된 원소가 있을 수 있음에 유의합시다.
- 관찰 1. SN set의 각 원소를 U, P를 기준으로 한번 더 분리시키고, UP set의 각 원소에 대해 S, N을 기준으로 한번 더 분리시켜봅시다.
- 예를 들면 'CUPCS'라는 SN set에 있는 문자열을 [CU,P,CS]로 쪼개서 생각한다는 것입니다. 이 때 쪼개지는 문자열을 $s_1, s_2, \cdots, s_k$라 합시다.
- $k \geq 2$인 경우에 대해 생각합니다. 여기서 $s_2, \cdots, s_{k-1}$은 앞뒤로 문자열이 붙어있으며 추가적인 변형을 가할 수 없기 때문에 그냥 UP set에서 해당 문자열을 제거한 후 s_1과 s_k만 붙어있다고 생각해도 됩니다.
- 앞으로 편의상 s_1을 **앞문자열**, s_k를 **뒤문자열**이라고 하겠습니다.

J. SNUPC 문자열 (Hard)

- 관찰 2. 위에서 $k = 1$이였다면 대부분 다른 문자열의 중간 부분을 지우는 과정에서 사라졌었겠지만, 정확히 2개는 사라지지 않습니다.
- 하나는 맨 앞쪽에 있는 문자열이며, 다른 하나는 맨 뒤쪽에 있는 문자열입니다.
- 이는 다음과 같은 그림을 생각하면 쉽게 이해할 수 있습니다.



- 여기서 하나의 구간이 한 문자열 조각이며, 첫 번째 행이 SN set이고 두 번째 행이 UP set입니다.
- 여기서 빨간색 부분은 관찰 1에 의해 삭제되며, 자명하게도 초록색 부분 2개가 남게 됩니다. 이는 한 위치에서 SN set과 UP set 모두에서 동시에 끊어질 수 없기에 자명합니다.

- 남은 초록색 문자열 중 앞에 오는것과 뒤에 오는 것을 구별하는 방법은 다음과 같습니다:
- – SN 문자열인데 U/P가 등장하지 않으면 앞에 오는 문자열이다.
  – UP 문자열인데 S/N이 등장하지 않으면 앞에 오는 문자열이다.
  – SN 문자열인데 S/N으로 끝나지 않으면 뒤에 오는 문자열이다.
  – UP 문자열인데 U/P로 끝나지 않으면 뒤에 오는 문자열이다.
- 이는 조금 생각해보면 당연하며, 정확히 앞의 하나와 뒤의 하나로 나뉘집니다.

J. SNUPC 문자열 (Hard)

- 관찰 3. 위에 봤던 그림에서 1, 3은 U, P로 끝나며 2, 4는 S, N으로 끝납니다. 또한 1, 3의 앞문자열은 S, N로 끝나며 2, 4의 앞문자열은 U, P로 끝납니다.
- 따라서 매우 자연스럽게 레고 블록을 끼듯이 이어집니다.
- 그렇다면 그냥 앞문자열에서 뒤문자열로 유향 간선을 잇는다고 생각하고 간선을 그냥 타다 보면 문자열이 나올 것 같습니다.
- 위 성질 때문에 SN set->UP set->SN set->··· 형태로 번갈아가면서 지나지기 때문입니다.
- 다만 한 가지 유의할 점이 있습니다. 초록색 부분에 한해서 SN set->SN set이나 UP set->UP set으로 이동하는 경우가 생길 수 있습니다.
- 따라서 이를 배제해줘야 하는데, 끝 부분은 유일하므로 끝 부분에서 E라는 가상 정점으로 가는 것이랑 E에서 시작 부분으로 가는 간선을 추가해주면 해결이 가능합니다.

J. SNUPC 문자열 (Hard)

- 이렇게 하면 euler circuit의 개수를 구하는 문제로 환원됩니다.
- 앞문자열과 뒷문자열들을 적절히 해싱해서 정점으로 만들어주면 됩니다.
- 다만 유의해야할 조건이 하나 있는데, "이 때 문자열 조각들을 이어 붙인 순서는 고려하지 않고, 복원된 문자열이 같으면 같은 문자열이다." 라는 것입니다.
- 이에 대한 Lemma를 적절히 세워봅시다.

- $Lemma.$ SN set(or UP set)에 있는 문자열들을 $A = []$ 와 $B = []$ 형태의 두 가지 방법으로 나열하였을 때, 어떤 i 에 대해 $A_i \neq B_i$ 면 A, B 는 서로 다른 문자열이다. (A_i, B_i 는 문자열 조각)
- $Proof.$ $A_{1..(x-1)}$ 과 $B_{1..(x-1)}$ 은 모두 S 또는 N으로 끝나고 문자열 조각 중간에 S 또는 N이 없어 A, B 문자열이 주어졌을 때 ordered SN set으로 쪼개는 방법이 유일합니다. 따라서 A, B 가 같은 문자열이면 $A_i = B_i$ 입니다. 대우 증명법에 의해 $Lemma$ 가 성립합니다. SN set이라는 조건이 없다고 생각한다면 $ab/a/b, a/b/ab$ 와 같은 반례가 있습니다.

- 이제 Lemma를 이용해봅시다.
- 중간 문자열에 해당하는 문자열들을 다른 set에서 없애는 관찰 1을 진행한 후 단순히 같은 문자열의 대해서만 동자순열 느낌으로 나눠서 계산해주면 됩니다.
- 이 때 유의할 점은 $s_1, s_2, \cdots, s_k$를 모두 이어붙인 상태에서 중복인 개수를 세야한다는 것이며, 따라서 s_1Ps_k와 s_1Qs_k는 다른 문자열입니다.
- $k = 1$이여서 없어진 문자열에 대해선 계산하지 않아야 합니다.

J. SNUPC 문자열 (Hard)

- euler circuit의 개수는 어떻게 셀까요? 위키피디아를 참고하면 유향 그래프에서의 euler circuit을 세는 BEST theorem을 찾을 수 있습니다.
- 이에 대한 공식은 다음과 같이 나타납니다.

$$\text{개수} = t_w(G) \cdot \prod_{v \in V} (outdeg(v) - 1)!$$

- $t_w(G)$는 arborescences의 개수로, matrix-tree theorem에 의해 구할 수 있습니다.
- 결론은 [차수 행렬]-[인접 행렬]로 나타내지는 라플라시안 행렬의 minor matrix M의 determinant가 $t_w(G)$입니다.
- 어떠한 행/열을 없애서 minor(=$det(M)$)를 계산해도 결과는 같음이 알려져 있습니다. 이 때 행렬식은 M을 가우스 소거 시킨 후 주대각 원소의 곱과 같으므로 시간복잡도는 가우스 소거를 시키는 데 필요한 $O(V^3)$입니다. 현재로썬 최악의 경우 $V \simeq N$이므로 $O(N^3)$입니다.

J. SNUPC 문자열 (Hard)

- 관찰 4. 정말 $O(N^3)$ 일까요? 뒤문자열로는 뭐가 가능할까요?
- SN set이라고 편의상 생각합시다. U, P가 중간에 있을 수 없으므로, $CC \cdots CS$ 혹은 $CC \cdots CN$ 형태만 가능합니다.
- 이 때 중복되는 경우 그냥 하나의 정점에 넣어도 됩니다.
- 따라서 $S, CS, CCS, \cdots, CC \cdots CS$ 형태로 되어야 정점이 가장 많아질 수 있습니다.
- 이 때 사용하는 문자 개수는 $O(p^2)$ 수준으로 증가합니다. 즉 실제로 뒤문자열로 가능한 것은 $O(\sqrt{N})$ 수준입니다.

J. SNUPC 문자열 (Hard)

- 엄밀하게는 뒤문자열로는 $C\cdots CS, C\cdots CN, C\cdots CU, C\cdots CP$ 형태가 가능하므로 최대한 길이가 작은것부터 채워나가는 것이 최적입니다.
- 즉 $\frac{k(k+1)}{2} \simeq \frac{k^2}{2} = \frac{N}{4}$ 여야 하고 $k = \sqrt{N/2}$ 여야 최적입니다.
- 총 $4k$ 개가 가능하므로 정점은 최악의 경우 $V = 4\sqrt{N/2}$ 개 생깁니다.
- 따라서 시간복잡도를 다시 계산해본다면 $O((4\sqrt{N/2})^3) = O(16\sqrt{2}\cdot N\sqrt{N})$ 입니다.
- $N = 10^5$ 일 때 대강 7억정도 되며 5초면 충분히 돌 수 있음을 알 수 있습니다.
- 실제로는 $O(N\sqrt{N})$ 치곤 이론적인 상수가 $16\sqrt{2}$ 로 매우 크지만 라플라시안 행렬이 sparse한 탓인지 매우 빠르게 돌아갑니다. (mcs 기준 60ms)

K. 대각선

- ad-hoc
- 출제자: kizen

K. 대각선

- 선택한 행을 r 이라 하면, 이 행이 주대각선과 만나는 칸은 (r, r) 입니다.
- 같은 행이 부대각선과 만나는 칸은 $(r, N+1-r)$ 입니다.
- 두 칸이 겹치는 경우는 $r = \dfrac{N+1}{2}$ 인 경우뿐입니다. 이는 N 이 홀수이고 r 이 가운데 행인 경우입니다.
- 그 외에는 (r, r) 와 $(r, N+1-r)$ 가 서로 다르므로 두 칸 모두 칠해야 합니다.

K. 대각선

- N 이 홀수인 경우, 가운데 행 $r = \dfrac{N+1}{2}$ 를 선택하여 (r, r) 한 칸만 칠하면 두 대각선을 동시에 만족합니다. 답은 1 입니다.
- N 이 짝수인 경우, 어떤 행을 골라도 두 지점이 항상 다릅니다. 두 칸을 칠해야 하므로 답은 2 입니다.
- 따라서 정답은

$$\text{ans} = \begin{cases} 1 & (N\text{이 홀수}) \\ 2 & (N\text{이 짝수}) \end{cases}$$

L. 제곱수 순열

- `number theory, parity`
- 출제자: lunarlity

L. 제곱수 순열

- 길이 N의 순열 A, B가 있을 때 모든 $1 \le i < N$에 대해 $A_i^{B_i} \times A_{i+1}^{B_{i+1}}$가 제곱수여야 합니다.
- Lemma: 해당 수열이 문제의 조건을 만족하려면 모든 수가 제곱수이거나 모든 수가 제곱수가 아니여야 한다.
- 제곱수와 제곱수가 아닌 수의 곱은 항상 제곱수가 아니므로, 위 Lemma가 항상 성립합니다.

- 하지만 조건에 따라 모든 수가 제곱수가 아닐 수는 없으므로, 모든 수가 제곱수여야 합니다.
- 각 숫자가 제곱수이려면, 지수가 짝수이거나 밑이 제곱수이면 됩니다.
- 그러면 제곱수를 최대한 만드려 할 때 $\lfloor\sqrt{N}\rfloor + \left\lfloor\frac{N}{2}\right\rfloor$ 개까지 만들 수 있고, 이것이 N 이상이여야 하므로 $N = 2, 4$ 만 가능합니다.
- $N = 2$ 인 경우 $A = (1, 2), B = (1, 2)$ 가 가능합니다.
- $N = 4$ 인 경우 $A = (1, 2, 3, 4), B = (1, 2, 4, 3)$ 이 가능합니다.