Omówienie zadań

Łamana 2

Najszybsze rozwiązanie: **Bartosz Kostka (0:09)**

Dane jest słowo s nad alfabetem wielkości $k = 16$.

Każdej literze alfabetu przypisz strzałkę $\uparrow$ lub $\rightarrow$, by zmaksymalizować pole powierzchni pod otrzymaną łamaną.

Rozwiązanie brutalne sprawdza liniowo każdą z 2^k możliwości.

Rozwiązanie brutalne sprawdza liniowo każdą z 2^k możliwości.

```
long long evaluate(int mask) {
  long long score = 0;
  int y = 0;
  for (int c : s) { // 0 <= c < k
    if (mask & (1 << c)) { // up
      y++;
    }
    else { // right
      score += y;
    }
  }
  return score;
}
```

Rozwiązanie brutalne sprawdza liniowo każdą z 2^k możliwości.

```
long long evaluate(int mask) {
  long long score = 0;
  int y = 0;
  for (int c : s) { // 0 <= c < k
    if (mask & (1 << c)) { // up
      y++;
    }
    else { // right
      score += y;
    }
  }
  return score;
}
```

Wygląda znajomo?

Rozwiązanie brutalne sprawdza liniowo każdą z 2^k możliwości.

```
long long evaluate(int mask) {
  long long score = 0;
  int y = 0;
  for (int c : s) { // 0 <= c < k
    if (mask & (1 << c)) { // up
      y++;
    }
    else { // right
      score += y;
    }
  }
  return score;
}
```

Wygląda znajomo? Zliczamy inwersje w ciągu binarnym!

Inwersja w ciągu binarnym to para pozycji (i, j) że:

- $i < j$
- $a_i = 1$
- $a_j = 0$

Inwersja w ciągu binarnym to para pozycji (i, j) że:

- $i < j$
- $a_i = 1$ — strzałka w górę
- $a_j = 0$ — strzałka w dół

... co ma sens też bez uwzględnienia rozwiązania brutalnego.

Dla każdej z k^2 par literek (t, v) zliczmy inwersje w przypadku przypisania t na $\uparrow$ oraz v na $\rightarrow$. Więcej o tym za chwilę.

Znanie takich wartości pozwala liczyć wynik każdej maski w złożoności $\mathcal{O}(k^2)$ lub nawet $\mathcal{O}(k)$ z rekurencją i wybieraniem każdego kolejnego bitu, co nie było wymagane.

Metoda I

Niezależne przejście po ciągu dla każdej pary daje wystarczającą złożoność $\mathcal{O}(n \cdot k^2)$.

Metoda I

Niezależne przejście po ciągu dla każdej pary daje wystarczającą złożoność $\mathcal{O}(n \cdot k^2)$.

Metoda II

Możemy zapamiętać pozycje wystąpowania literek i przerobić parę (t, v) w $\mathcal{O}(cnt_t + cnt_v)$, co daje łącznie $\mathcal{O}(n \cdot k)$.

Metoda I

Niezależne przejście po ciągu dla każdej pary daje wystarczającą złożoność $\mathcal{O}(n \cdot k^2)$.

Metoda II

Możemy zapamiętać pozycje wystąpowania literek i przerobić parę (t, v) w $\mathcal{O}(cnt_t + cnt_v)$, co daje łącznie $\mathcal{O}(n \cdot k)$.

Metoda III

```
for (int t = 0; t < k; t++) {
  int y = 0;
  for(int c : s) {
    if (c == t) y++;
    else inversions[t][c] += y;
  }
}
```

Dopuszczalna złożoność: $\mathcal{O}(n \cdot k^2 + 2^k \cdot k^2)$
Lepsza złożoność: $\mathcal{O}(n \cdot k + 2^k \cdot k)$

Dopuszczalna złożoność: $\mathcal{O}(n \cdot k^2 + 2^k \cdot k^2)$
Lepsza złożoność: $\mathcal{O}(n \cdot k + 2^k \cdot k)$

Bonusowanie pytanie za batonika:
Jaki jest wynik dla losowego palindromu długości 300 000 nad alfabetem 16? Dlaczego tak się dzieje?

Zaczarowany ołówek

Najszybsze rozwiązanie: **Konrad Paluszek (0:17)**

Dany jest zbiór odcinków na płaszczyźnie (podany dla niepoznaki jako boki trójkątów). Policz sumę długości odcinków tworzących część płaszczyzny należącą do nieparzystej liczby z nich.

Jak rozwiązać zadanie w 1D?

- Sortujemy $2k$ końców

Jak rozwiązać zadanie w 1D?

- Sortujemy $2k$ końców
- $\sum_{i=0}^{k} dist(P_{2i}, P_{2i+1})$

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

- $Ax_1 + By_1 + C = 0, Ax_2 + By_2 + C = 0$

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

- $Ax_1 + By_1 + C = 0, Ax_2 + By_2 + C = 0$
- $NWD(A, B) = 1$

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

- $Ax_1 + By_1 + C = 0, Ax_2 + By_2 + C = 0$
- $NWD(A, B) = 1$
- $A > 0 \vee (A = 0 \wedge B > 0)$.

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

- $Ax_1 + By_1 + C = 0, Ax_2 + By_2 + C = 0$
- $NWD(A, B) = 1$
- $A > 0 \vee (A = 0 \wedge B > 0)$.

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

- $Ax_1 + By_1 + C = 0, Ax_2 + By_2 + C = 0$
- $NWD(A, B) = 1$
- $A > 0 \vee (A = 0 \wedge B > 0)$.

Bardziej wizualnie:

- (A, B) - jednoznaczny kierunek

Jak rozwiązać zadanie w 2D?

Pogrupować wszystkie odcinki na proste $\Rightarrow$ rozwiązać wersję 1D.

Jak pogrupować odcinki po zawierających je prostych?

Równanie prostej!

- $Ax_1 + By_1 + C = 0, Ax_2 + By_2 + C = 0$
- $NWD(A, B) = 1$
- $A > 0 \vee (A = 0 \wedge B > 0)$.

Bardziej wizualnie:

- (A, B) - jednoznaczny kierunek
- C - iloczyn wektorowy kierunku i dowolnego punktu

Zdjęcie rodzinne

Najszybsze rozwiązanie: **Bartosz Kostka (0:20)**

Dane jest ukorzenione drzewo. Dorzucamy do niego wszystkie krawędzie przodek-potomek. Pytamy o najdłuższą rozłączną wierzchołkowo ścieżkę w powstałym w grafie.

Definicja

$DP[v][i]$ – maksymalna sumaryczna długość i (lub mniej) rozłącznych ścieżek wybranych w grafie ograniczonym tylko do poddrzewa ukorzenionego w v.

Definicja

$DP[v][i]$ – maksymalna sumaryczna długość i (lub mniej) rozłącznych ścieżek wybranych w grafie ograniczonym tylko do poddrzewa ukorzenionego w v.

Kluczową obserwacją o wartościach $DP[v]$ jest ich wypukłość – $DP[v][i+1] - DP[v][i] \leq DP[v][i] - DP[v][i-1]$. Wynika to wprost z przejść, za pomocą których obliczamy ich wartości, tzn. może być dowodzone indukcyjnie.

Łączenie wartości dla dwóch poddrzew to liczenie sumy Minkowskiego dla wykresów wyników poddrzew.

Łączenie wartości dla dwóch poddrzew to liczenie sumy Minkowskiego dla wykresów wyników poddrzew.

Dodawanie wierzchołka połączonego ze wszystkimi wierzchołkami ze zbioru (poddrzew) to zastąpienie wartości $DP[v][i]$ przez $DP[v][i+1]+1$ dla wszystkich dodatnich i.

Łączenie wartości dla dwóch poddrzew to liczenie sumy Minkowskiego dla wykresów wyników poddrzew.

Dodawanie wierzchołka połączonego ze wszystkimi wierzchołkami ze zbioru (poddrzew) to zastąpienie wartości $DP[v][i]$ przez $DP[v][i+1]+1$ dla wszystkich dodatnich i.

Powyższe operacje działają dzięki warunkowi indukcyjnemu i utrzymują go, co można łatwo sprawdzić.

Wydajną implementację można osiągnąć utrzymując dla każdego v multizbiór wartości $DP[v][i] - DP[v][i-1]$, które ze względu na wypukłość są nierosnące dla rosnących i.

Łączenie poddrzew to łączenie multizbiorów, dodawanie wspólnego przodka to drobna modyfikacja największych elementów.

Wydajną implementację można osiągnąć utrzymując dla każdego v multizbiór wartości $DP[v][i] - DP[v][i-1]$, które ze względu na wypukłość są nierosnące dla rosnących i.

Łączenie poddrzew to łączenie multizbiorów, dodawanie wspólnego przodka to drobna modyfikacja największych elementów.

Ostateczne rozwiązanie działa w złożoności $O(n \cdot \log^2(n))$ i przez głębszą analizę struktury problemu może zostać zoptymalizowane nawet do $O(n)$, co jednak nie było konieczne.

Gdzie jest jedynka? 2

Najszybsze rozwiązanie: **Maciej Wawro (1:52)**

Mamy ukrytą permutację n-elementową $(P_0, P_1, \ldots, P_{n-1})$, $n \geq 3$.

Możemy zadawać pytania o $\mathrm{NWD}(P_i, P_j)$ dla $i \neq j$.

Należy znaleźć pozycję liczby 1 w permutacji.

Możemy użyć maksymalnie $\lceil 2.5n \rceil$ zapytań.

Pomysł

Weźmy dowolny element permutacji, np. P_0, i zapytajmy o jego NWD z każdym innym elementem permutacji.

Pomysł

Weźmy dowolny element permutacji, np. P_0, i zapytajmy o jego NWD z każdym innym elementem permutacji.

Skutek

Zużywamy $n-1$ zapytań.

- Jeśli wszystkie NWD są równe 1, to $P_0 = 1$.
- Jeśli wszystkie NWD są różne, to $P_0 = 0$. Wtedy $\mathrm{NWD}(P_0, P_i) = 1 \Leftrightarrow P_i = 1$.
- Inaczej mamy $P_0 > 1$ oraz:

$$\mathrm{NWD}(P_0, 1) = 1 \quad \text{oraz} \quad \mathrm{NWD}(P_0, 0) = \max_{i \geq 1} \mathrm{NWD}(P_0, P_i).$$

Każde P_i może być kandydatem na 0 (maksymalne NWD) albo kandydatem na 1 (NWD równe 1), ale nie jednocześnie.

Skutek

- Każde P_i może być kandydatem na 0 albo kandydatem na 1.
- Kandydatów na 0 jest co najwyżej $\frac{n}{2}$.
- Zostało $\lceil 1.5n \rceil$ zapytań.

Skutek

- Każde P_i może być kandydatem na 0 albo kandydatem na 1.
- Kandydatów na 0 jest co najwyżej $\frac{n}{2}$.
- Zostało $\lceil 1.5n \rceil$ zapytań.

Szukanie wartości 0

Dany zbiór $X = \{i_0, \ldots, i_k\}$ kandydatów na wartość 0, $|X| \geq 3$.
Zredukujmy X do rozmiaru $\leq \max(2, |X|/2)$.

- Obliczmy $\mathrm{NWD}(P_{i_0}, P_{i_1}), \mathrm{NWD}(P_{i_0}, P_{i_2}), \ldots, \mathrm{NWD}(P_{i_0}, P_{i_k})$.
- $X' := \{x \in X \mid \mathrm{NWD}(P_{i_0}, P_x)$ jest maksymalne$\}$.
  - Jeśli $X' = \{x\}$, to $P_x = 0$ albo $P_{i_0} = 0$.
  - Jeśli $|X'| \geq 2$, to zero znajduje się w X'.

Pojedyncza redukcja zużywa $|X| - 1$ zapytań.

Skutek

- Każde P_i może być kandydatem na 0 albo kandydatem na 1.
- Kandydatów na 0 jest co najwyżej $\frac{n}{2}$.
- Zostało $\lceil 1.5n \rceil$ zapytań.

Szukanie wartości 0

Dany zbiór $X = \{i_0, \ldots, i_k\}$ kandydatów na wartość 0, $|X| \geq 3$.
Zredukujmy X do rozmiaru $\leq \max(2, |X|/2)$.

- Obliczmy $\mathrm{NWD}(P_{i_0}, P_{i_1}), \mathrm{NWD}(P_{i_0}, P_{i_2}), \ldots, \mathrm{NWD}(P_{i_0}, P_{i_k})$.
- $X' := \{x \in X \mid \mathrm{NWD}(P_{i_0}, P_x)$ jest maksymalne$\}$.
  - Jeśli $X' = \{x\}$, to $P_x = 0$ albo $P_{i_0} = 0$.
  - Jeśli $|X'| \geq 2$, to zero znajduje się w X'.

Pojedyncza redukcja zużywa $|X| - 1$ zapytań.

W co najwyżej $2|X|$ zapytaniach możemy ograniczyć zbiór kandydatów na 0 do dwóch elementów.

Szukanie wartości 1

Wiemy, że $P_a = 0$ lub $P_b = 0$. Znamy też zbiór Y kandydatów na jedynkę.

Szukanie wartości 1

Wiemy, że $P_a = 0$ lub $P_b = 0$. Znamy też zbiór Y kandydatów na jedynkę.

Algorytm

- Przechodzimy po kolejnych kandydatach $y_1, y_2, \ldots, y_k$ na jedynkę i liczymy kolejno $\mathrm{NWD}(P_a, P_{y_1}), \mathrm{NWD}(P_a, P_{y_2}), \ldots$. Przerywamy, gdy otrzymamy pierwszą odpowiedź 1: $\mathrm{NWD}(P_a, P_{y_j}) = 1$.
- Jeśli $\mathrm{NWD}(P_b, P_{y_j}) = 1$, to $P_{y_j} = 1$.
- Jeśli $\mathrm{NWD}(P_b, P_{y_j}) > 1$, to $P_b = 0$. Przechodzimy po pozostałych kandydatach $y_{j+1}, y_{j+2}, \ldots, y_k$ na jedynkę i przerywamy, gdy NWD kandydata z P_b wyniesie 1. Ten kandydat jest wynikiem.

Szukanie wartości 1

Wiemy, że $P_a = 0$ lub $P_b = 0$. Znamy też zbiór Y kandydatów na jedynkę.

Algorytm

- Przechodzimy po kolejnych kandydatach $y_1, y_2, \ldots, y_k$ na jedynkę i liczymy kolejno $\mathrm{NWD}(P_a, P_{y_1}), \mathrm{NWD}(P_a, P_{y_2}), \ldots$. Przerywamy, gdy otrzymamy pierwszą odpowiedź 1: $\mathrm{NWD}(P_a, P_{y_j}) = 1$.
- Jeśli $\mathrm{NWD}(P_b, P_{y_j}) = 1$, to $P_{y_j} = 1$.
- Jeśli $\mathrm{NWD}(P_b, P_{y_j}) > 1$, to $P_b = 0$. Przechodzimy po pozostałych kandydatach $y_{j+1}, y_{j+2}, \ldots, y_k$ na jedynkę i przerywamy, gdy NWD kandydata z P_b wyniesie 1. Ten kandydat jest wynikiem.

Powyższy algorytm znajduje ostatecznie jedynkę w $|Y| + 1$ zapytań.

Czas działania

- Pierwsza faza zużywa $n-1$ zapytań i tworzy zbiór X kandydatów na zera i zbiór Y kandydatów na jedynki.
  - $|X| + |Y| \leq n$
  - $|X| \leq \frac{n}{2}$
- Druga faza (szukanie dwóch kandydatów na zero) zużywa $\leq 2 \cdot |X|$ zapytań.
- Trzecia faza (szukanie jedynki) zużywa $|Y| + 1$ zapytań.
- Łącznie: $n + 2|X| + |Y| \leq \frac{5}{2}n$ zapytań.

Terytoria 2

Najszybsze rozwiązanie: **Bartosz Kostka (1:22)**

Dany prostokąt $X \times Y$, oraz S zwierząt podzielonych na n gatunków. Każde zwierze ustawiamy w polu poza zakazanym (dla jego gatunku) prostokątem. Dla $p_1, p_2, \ldots, p_k$ zwierząt w poszczególnych polach wynikiem jest

$$\sum_{i=0}^{k} \frac{p_i \cdot (p_i - 1)}{2}.$$

Lemat

Jeśli można ustawić zwierzę w p_i to nie ustawiamy go w p_j dla $p_i \geq p_j$.

Lemat

Jeśli można ustawić zwierzę w p_i to nie ustawiamy go w p_j dla $p_i \geq p_j$.

Dowód

Lepiej zrobić $p_i' := p_i + 1$, $p_j' := p_j - 1$.

$$\frac{p_i \cdot (p_i - 1)}{2} + \frac{p_j \cdot (p_j - 1)}{2} <$$

$$< \frac{p_i \cdot (p_i - 1)}{2} + p_i + \frac{p_j \cdot (p_j - 1)}{2} - (p_j - 1) =$$

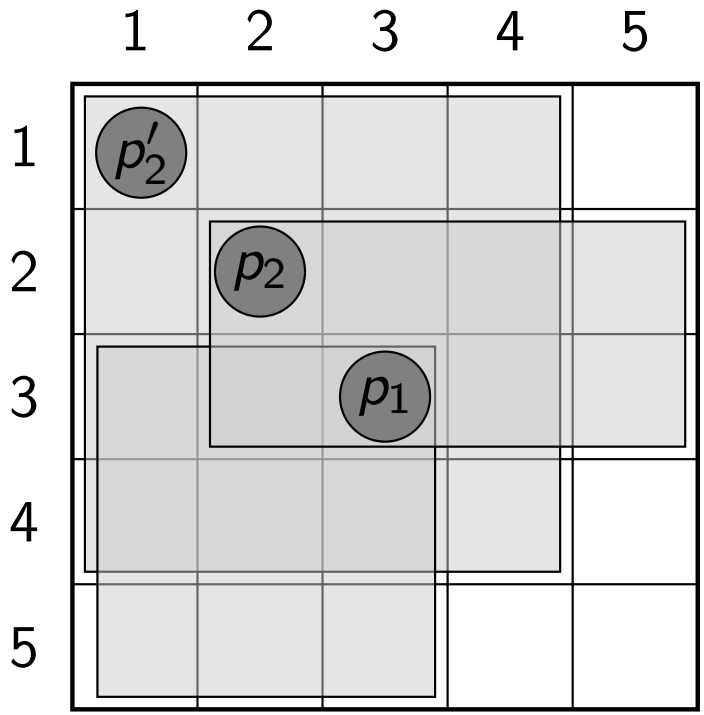
$$= \frac{(p_i + 1) \cdot p_i}{2} + \frac{(p_j - 1) \cdot (p_j - 2)}{2}$$

Lemat

Istnieje optymalne rozwiązanie z co najwyżej jednym niepustym polem nie w rogu planszy.

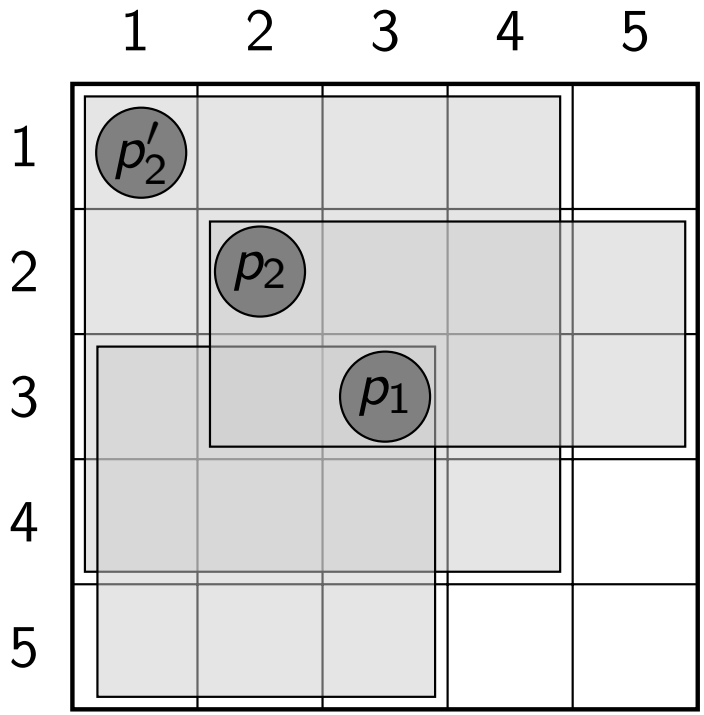
Lemat

Istnieje optymalne rozwiązanie z co najwyżej jednym niepustym polem nie w rogu planszy.



Lemat

Istnieje optymalne rozwiązanie z co najwyżej jednym niepustym polem nie w rogu planszy.



Pole p_2' jest ściśle lepsze niż p_2. Każdy prostokąt zawierający p_1 pokrywa oba, albo samo p_2.

Algorytm

1. Iterujemy po pierwszym polu p_1.

Algorytm

1. Iterujemy po pierwszym polu p_1.
2. Iterujemy po rogu p_2.

Algorytm

1. Iterujemy po pierwszym polu p_1.
2. Iterujemy po rogu p_2.
3. Wszystkie pozostałe zwierzęta umieszczamy w rogu przeciwnym do p_2 (żaden prostokąt nie zawiera dwóch przeciwnych rogów).

Algorytm

1. Iterujemy po pierwszym polu p_1.
2. Iterujemy po rogu p_2.
3. Wszystkie pozostałe zwierzęta umieszczamy w rogu przeciwnym do p_2 (żaden prostokąt nie zawiera dwóch przeciwnych rogów).

Po stablicowaniu sum prefiksowych jedną parę (p_1, p_2) można rozpatrzeć w czasie stałym.

Algorytm

1. Iterujemy po pierwszym polu p_1.
2. Iterujemy po rogu p_2.
3. Wszystkie pozostałe zwierzęta umieszczamy w rogu przeciwnym do p_2 (żaden prostokąt nie zawiera dwóch przeciwnych rogów).

Po stablicowaniu sum prefiksowych jedną parę (p_1, p_2) można rozpatrzeć w czasie stałym.

Złożoność

$\mathcal{O}(n + X \cdot Y)$

Floyd-Warshall

Najszybsze rozwiązanie: **Maciej Wawro (3:17)**

Poprawny Floyd-Warshall ma kolejność pętli:

(midpoint, skąd, dokąd).

Rozpatrujemy błędną implementację Floyda-Warshalla z kolejnością pętli:

(skąd, dokąd, midpoint).

Ile odległości zostanie błędnie obliczonych przez niepoprawną implementację?

Symulacja poprawnego algorytmu

Nie musimy symulować poprawnej implementacji Floyda-Warshalla – wystarczy z każdego źródła uruchomić algorytm najkrótszych ścieżek Dijkstry. Możemy to zrobić w czasie $O(nm \log n)$.

Symulacja poprawnego algorytmu

Nie musimy symulować poprawnej implementacji Floyda-Warshalla – wystarczy z każdego źródła uruchomić algorytm najkrótszych ścieżek Dijkstry. Możemy to zrobić w czasie $O(nm \log n)$.

Symulacja błędnego algorytmu

Błędny algorytm Dijkstry liczy wyniki dla kolejnych par wierzchołków. Spróbujmy więc dla każdej kolejnej pary (x, y) zobaczyć, czy odległość z x do y zostanie obliczona poprawnie.

Oryginalne krawędzie grafu

Jeśli odległość z x do y wynosi w_{xy} oraz istnieje krawędź $x \to y$ o wadze w_{xy}, to na początku działania błędnego Floyda-Warshalla odległość jest wyznaczona poprawnie.

Oryginalne krawędzie grafu

Jeśli odległość z x do y wynosi w_{xy} oraz istnieje krawędź $x \to y$ o wadze w_{xy}, to na początku działania błędnego Floyda-Warshalla odległość jest wyznaczona poprawnie.

Pozostałe pary wierzchołków

Błędny algorytm Floyda-Warshalla zapisze poprawnie odległość z x do y ($x \neq y$), gdy istnieje wierzchołek z taki, że:

- Odległość $x \to z$ jest wyznaczona poprawnie,
- Odległość $z \to y$ jest wyznaczona poprawnie,
- z leży na jakiejś najkrótszej ścieżce $x \to y$.

Pozostałe pary wierzchołków

Błędny algorytm Floyda-Warshalla zapisze poprawnie odległość z x do y ($x \neq y$), gdy istnieje wierzchołek z taki, że:

- Odległość $x \to z$ jest wyznaczona poprawnie,
- Odległość $z \to y$ jest wyznaczona poprawnie,
- z leży na jakiejś najkrótszej ścieżce $x \to y$.

Będziemy utrzymywać trzy tablice bitsetów:

- $correct(x, y)$: odległość $x \to y$ jest poprawna,
- $rev\_correct(x, y)$: odległość $y \to x$ jest poprawna,
- $on\_path(x, y, z)$: z leży na najkrótszej ścieżce $x \to y$.

Pozostałe pary wierzchołków

Błędny algorytm Floyda-Warshalla zapisze poprawnie odległość z x do y ($x \neq y$), gdy przecięcie trzech bitsetów jest niepuste:

- $correct(x, \star)$,
- $rev\_correct(y, \star)$,
- $on\_path(x, y, \star)$.

Będziemy utrzymywać trzy tablice bitsetów:

- $correct(x, y)$: odległość $x \to y$ jest poprawna,
- $rev\_correct(x, y)$: odległość $y \to x$ jest poprawna,
- $on\_path(x, y, z)$: z leży na najkrótszej ścieżce $x \to y$.

Pozostałe pary wierzchołków

Błędny algorytm Floyda-Warshalla zapisze poprawnie odległość z x do y ($x \neq y$), gdy przecięcie trzech bitsetów jest niepuste:

- $correct(x, \star)$,
- $rev\_correct(y, \star)$,
- $on\_path(x, y, \star)$.

Złożoność

Bitsety on_path można wyznaczyć w łącznym czasie $O\left(\frac{n^2 m}{64}\right)$.

Przecinanie bitsetów można zaimplementować w czasie $O\left(\frac{n^3}{64}\right)$.

Ostateczna złożoność czasowa: $O\left(nm \log n + \frac{n^2(n+m)}{64}\right)$.

Pionki

Najszybsze rozwiązanie: **Marek Sommer (3:48)**

Dana jest trójwymiarowa plansza. Dla każdej jej komórki wiemy ile pionków znajduje się w niej początkowo, a ile pionków powinno znajdować się w niej na koniec. Pionki można przesuwać jedynie w stronę rosnących współrzędnych. Pytamy, czy da się tak poprzesuwać pionki, aby osiągnąć wymagany stan planszy.

Zinterpretujmy problem grafowo.

Zinterpretujmy problem grafowo.

Tworzenie grafu

Stwórzmy graf dwudzielny. Wierzchołki tego grafu będziemy nazywać **lewymi** i **prawymi**, ze względu na to, do której części grafu należą.

Zinterpretujmy problem grafowo.

Tworzenie grafu

Stwórzmy graf dwudzielny. Wierzchołki tego grafu będziemy nazywać **lewymi** i **prawymi**, ze względu na to, do której części grafu należą.
W każdej komórce umieścimy tyle wierzchołków lewych, ile pionków znajduje się w niej początkowo, i tyle wierzchołków prawych, ile pionków powinno znajdować się w niej na koniec.

Zinterpretujmy problem grafowo.

Tworzenie grafu

Stwórzmy graf dwudzielny. Wierzchołki tego grafu będziemy nazywać **lewymi** i **prawymi**, ze względu na to, do której części grafu należą.
W każdej komórce umieścimy tyle wierzchołków lewych, ile pionków znajduje się w niej początkowo, i tyle wierzchołków prawych, ile pionków powinno znajdować się w niej na koniec.
Dwa wierzchołki łączymy krawędzią jeśli z komórki odpowiadającej lewemu wierzchołkowi można przesunąć pionek do komórki odpowiadającej prawemu wierzchołkowi.

Zinterpretujmy problem grafowo.

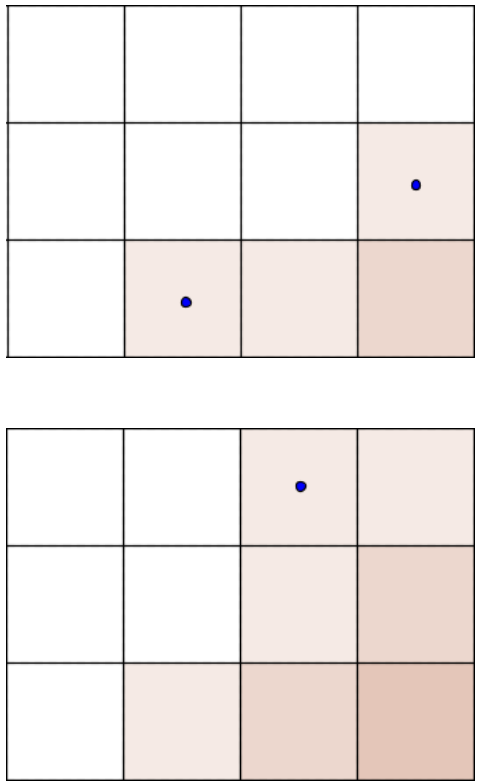
Tworzenie grafu

Stwórzmy graf dwudzielny. Wierzchołki tego grafu będziemy nazywać **lewymi** i **prawymi**, ze względu na to, do której części grafu należą.
W każdej komórce umieścimy tyle wierzchołków lewych, ile pionków znajduje się w niej początkowo, i tyle wierzchołków prawych, ile pionków powinno znajdować się w niej na koniec.
Dwa wierzchołki łączymy krawędzią jeśli z komórki odpowiadającej lewemu wierzchołkowi można przesunąć pionek do komórki odpowiadającej prawemu wierzchołkowi.

Jasnym jest, że odpowiedź jest pozytywna wtedy i tylko wtedy, gdy istnieje pełne skojarzenie w stworzonym grafie.

By stwierdzić czy skojarzenie istnieje, skorzystamy z twierdzenia Halla.

By stwierdzić czy skojarzenie istnieje, skorzystamy z twierdzenia Halla.
Wybierzmy podzbiór lewych wierzchołków i zastanówmy się jak wygląda ich sąsiedztwo. Przykładowo dla planszy $2 \times 3 \times 4$ i wybranych lewych punktów w komórkach $(1, 2, 4)$, $(1, 3, 2)$ oraz $(2, 1, 3)$, sąsiedztwo zawiera zacieniowane komórki:



Aby wykazać, że skojarzenie nie istnieje, należy znaleźć taki podzbiór lewych wierzchołków, że jego sąsiedztwo zawiera ściśle mniej prawych wierzchołków od niego. Jeśli nie da się znaleźć takiego podzbioru to zgodnie z twierdzeniem skojarzenie istnieje.

Aby wykazać, że skojarzenie nie istnieje, należy znaleźć taki podzbiór lewych wierzchołków, że jego sąsiedztwo zawiera ściśle mniej prawych wierzchołków od niego. Jeśli nie da się znaleźć takiego podzbioru to zgodnie z twierdzeniem skojarzenie istnieje.

Łatwo zauważyć, że jeśli wybierzemy podzbiór lewych wierzchołków i otrzymamy pewien zacieniowany kształt, to opłaca się dorzucić do owego podzbioru wszystkie lewe wierzchołki znajdujące się w tym kształcie – kształt się nie zmieni (zatem nie zmieni się również liczba prawych wierzchołków), a moc zbioru lewych wierzchołków tylko wzrośnie.

Wiemy zatem jak wyglądają optymalne do rozpatrywania podzbiory wierzchołków – wybieramy pewien *kształt* i rozważamy wszystkie lewe i prawe wierzchołki należące do tego kształtu. Owy kształt jest dowodem na nieistnienie skojarzenia jeśli zawiera ściśle więcej lewych wierzchołków niż prawych.

Wiemy zatem jak wyglądają optymalne do rozpatrywania podzbiory wierzchołków – wybieramy pewien *kształt* i rozważamy wszystkie lewe i prawe wierzchołki należące do tego kształtu. Owy kształt jest dowodem na nieistnienie skojarzenia jeśli zawiera ściśle więcej lewych wierzchołków niż prawych.

Oczywiście możemy wybierać jedynie kształty przyległe do wierzchołka o maksymalnych współrzędnych – jeśli kształt zawiera komórkę (i, j, k), to musi zawierać także komórki $(i + 1, j, k)$, $(i, j + 1, k)$ i $(i, j, k + 1)$.

Do znalezienia odpowiedniego kształtu skorzystamy z programowania dynamicznego. Niech $DP[i][m]$ oznacza maksymalną możliwą różnicę między liczbą lewych i prawych wierzchołków jeśli ustaliliśmy już co ma należeć do kształtu w poziomach od pierwszego do i-tego, a na i-tym poziomie do kształtu należą dokładnie komórki ze zbioru m.

Do znalezienia odpowiedniego kształtu skorzystamy z programowania dynamicznego. Niech $DP[i][m]$ oznacza maksymalną możliwą różnicę między liczbą lewych i prawych wierzchołków jeśli ustaliliśmy już co ma należeć do kształtu w poziomach od pierwszego do i-tego, a na i-tym poziomie do kształtu należą dokładnie komórki ze zbioru m.

Możliwych zbiorów m jest $\binom{B+C}{B}$ – jest to liczba ścieżek z lewego dolnego rogu do prawego górnego.

Do znalezienia odpowiedniego kształtu skorzystamy z programowania dynamicznego. Niech $DP[i][m]$ oznacza maksymalną możliwą różnicę między liczbą lewych i prawych wierzchołków jeśli ustaliliśmy już co ma należeć do kształtu w poziomach od pierwszego do i-tego, a na i-tym poziomie do kształtu należą dokładnie komórki ze zbioru m.

Możliwych zbiorów m jest $\binom{B+C}{B}$ – jest to liczba ścieżek z lewego dolnego rogu do prawego górnego.

Przejściami jest dołożenie do zbioru dodatkowego elementu (jest na to co najwyżej $min(B, C)$ opcji), lub stwierdzenie, że kształt na aktualnym poziomie zostaje jaki jest i przechodzimy do następnego poziomu (na co jest jedna opcja).

Skojarzenie nie istnieje jeśli możemy narysować kształt zawierający ściśle więcej wierzchołków lewych niż prawych. W przeciwnym przypadku odpowiedź jest pozytywna.

Skojarzenie nie istnieje jeśli możemy narysować kształt zawierający ściśle więcej wierzchołków lewych niż prawych. W przeciwnym przypadku odpowiedź jest pozytywna.

Końcowa złożoność wynosi zatem $O(A \cdot \binom{B+C}{B} \cdot (B + C))$.

Parzysty deszcz

Najszybsze rozwiązanie: ??? (?:??)

Ile sposobów usunięcia k z n danych kolumn skutkuje parzystą objętością deszczu, który zaraz spadnie z góry i nie będzie miał gdzie spłynąć? Wypisz wynik modulo $10^9 + 7$.

$$\text{water} = \sum_i \min(\mathit{prefMax}_{i-1}, \mathit{sufMax}_{i+1}) - \text{columns}$$

$$\text{water} = \sum_i \min(\mathit{prefMax}_{i-1}, \mathit{sufMax}_{i+1}) - \text{columns}$$

1. Pozbywamy się remisów.
2. Jest $k + 1$ możliwych pozycji największej kolumny.
3. Dwie niezależne części – lewo i prawo od największej kolumny.

$$\text{water} = \sum_i \min(prefMax_{i-1}, sufMax_{i+1}) - \text{columns}$$

1. Pozbywamy się remisów.
2. Jest $k + 1$ możliwych pozycji największej kolumny.
3. Dwie niezależne części – lewo i prawo od największej kolumny.

$$dp[i][removed][max\_so\_far]$$

$$dp[i][removed][max\_so\_far]$$

$$\mathcal{O}(n \cdot k^3 \cdot \log k)$$

Do AC wystarczy pozbyć się logarytmu z mapy i/lub nie iterować się po największej pozycji, skoro i tak liczymy to samo dp. W przypadku zastosowaniu obu tych optymalizacji otrzymamy:

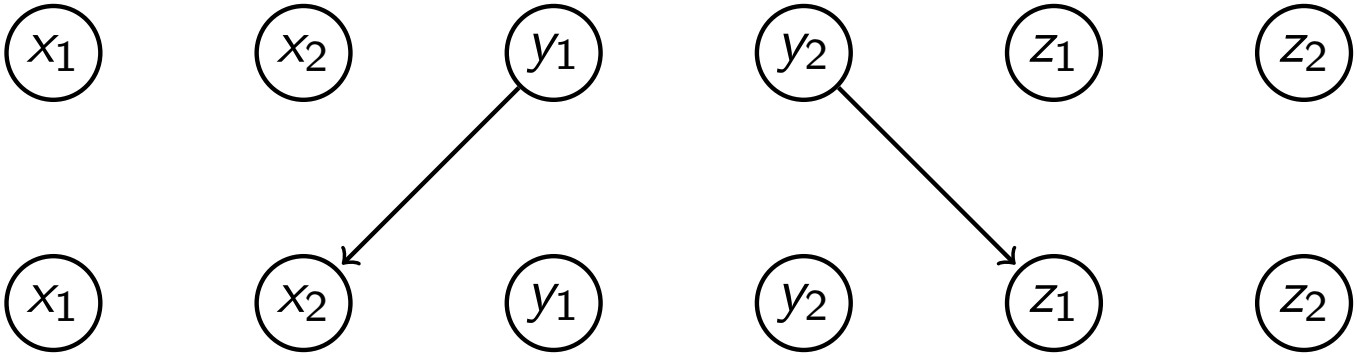
$$\mathcal{O}(n \cdot k^2)$$

Grafy

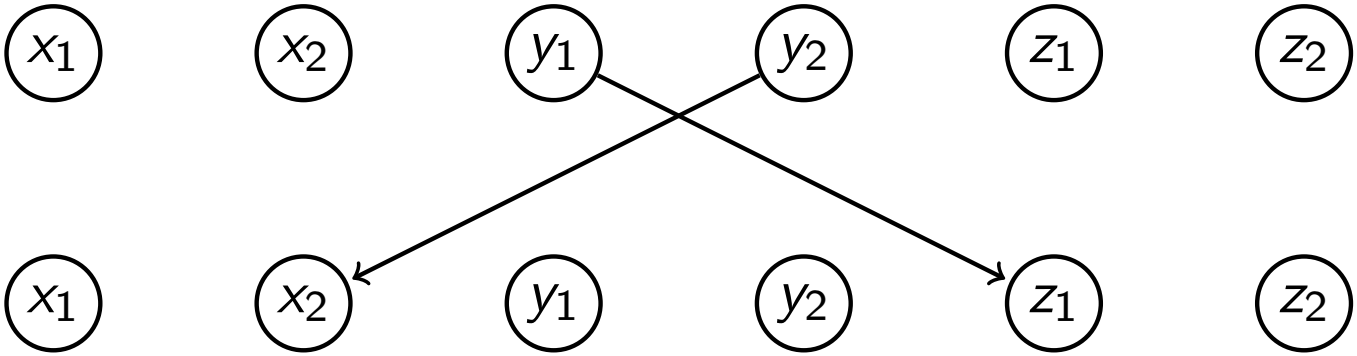
Najszybsze rozwiązanie: **??? (?:??)**

Ile jest grafów prostych o n wierzchołkach, w których każdy wierzchołek ma stopień wejściowy i wyjściowy 2.

Rozdwójmy każdy wierzchołek. Każdemu dobremu grafowi odpowiada 2^{2n} permutacji na nowych $2n$ wierzchołkach.

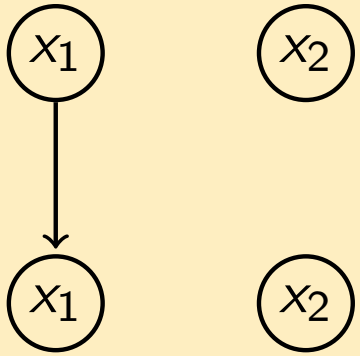


Możemy zamieniać krawędzie z kopii wierzchołka.



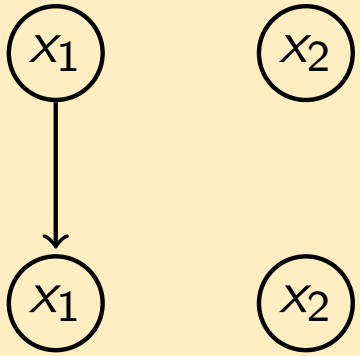
Złe zdarzenie

- Pętla - krawędź z kopii wierzchołka do kopii tego samego wierzchołka.

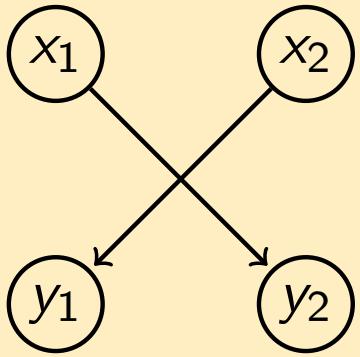


Złe zdarzenie

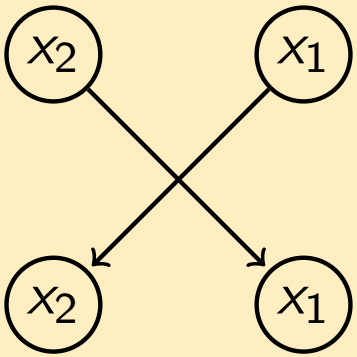
- Pętla - krawędź z kopii wierzchołka do kopii tego samego wierzchołka.



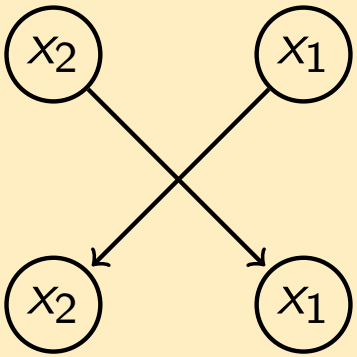
- Multikrawędź - dwie krawędzie z obu kopii wierzchołka do obu kopii (niekoniecznie innego) wierzchołka.



Multipętle

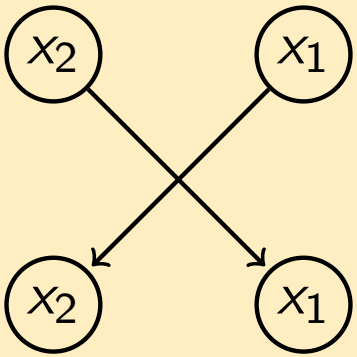


Multipętle



Multipętli odpowiada zbiór trzech zdarzeń:

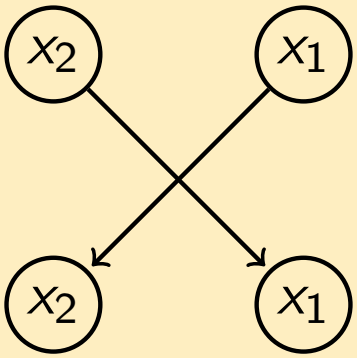
Multipętle



Multipętli odpowiada zbiór trzech zdarzeń:

- Pętla z x_1 do x_2.
- Pętla z x_2 do x_1.
- Multikrawędź z x do x.

Multipętle



Multipętli odpowiada zbiór trzech zdarzeń:

- Pętla z x_1 do x_2.
- Pętla z x_2 do x_1.
- Multikrawędź z x do x.

Obserwacja

Każde 2 z tych 3 zdarzeń wymuszają zajście trzeciego.

Zasada Wł-Wył

X - podzbiór złych zdarzeń
$p(X)$ - liczba permutacji pasujących do X

$$\sum_X (-1)^{|X|} \cdot p(X)$$

Zasada Wł-Wył

X - podzbiór złych zdarzeń
$p(X)$ - liczba permutacji pasujących do X

$$\sum_X (-1)^{|X|} \cdot p(X)$$

Grupowanie zbiorów zdarzeń

Zbiorowi zdarzeń X przyporządkujmy jego profil – trójkę liczb

- MP - liczba multipętli, które wynikają z co najmniej dwóch zdarzeń w X.

Zasada Wł-Wył

X - podzbiór złych zdarzeń
$p(X)$ - liczba permutacji pasujących do X

$$\sum_X (-1)^{|X|} \cdot p(X)$$

Grupowanie zbiorów zdarzeń

Zbiorowi zdarzeń X przyporządkujmy jego profil – trójkę liczb

- MP - liczba multipętli, które wynikają z co najmniej dwóch zdarzeń w X.
- P - liczba pętli z X, nie zawartych w MP.

Zasada Wł-Wył

X - podzbiór złych zdarzeń
$p(X)$ - liczba permutacji pasujących do X

$$\sum_X (-1)^{|X|} \cdot p(X)$$

Grupowanie zbiorów zdarzeń

Zbiorowi zdarzeń X przyporządkujmy jego profil – trójkę liczb

- MP - liczba multipętli, które wynikają z co najmniej dwóch zdarzeń w X.
- P - liczba pętli z X, nie zawartych w MP.
- M - liczba multikrawędzi, nie zawartych w MP.

Permutacje pasujące do profilu

- $\binom{n}{MP} \cdot 2^{MP}$ - wybory multipętli
- $\binom{n-MP}{P} \cdot 4^{P}$ - wybory pętli
- $\binom{n-MP-P}{M}^2 \cdot M! \cdot 2^{M}$ - wybory multikrawędzi
- $(2n - 2MP - 2M - P)!$ - dowolna permutacja na pozostałych wierzchołkach

Permutacje pasujące do profilu

- $\binom{n}{MP} \cdot 2^{MP}$ - wybory multipętli
- $\binom{n-MP}{P} \cdot 4^{P}$ - wybory pętli
- $\binom{n-MP-P}{M}^{2} \cdot M! \cdot 2^{M}$ - wybory multikrawędzi
- $(2n - 2MP - 2M - P)!$ - dowolna permutacja na pozostałych wierzchołkach

X pasujące do profilu

Dla każdej multipętli są 4 opcje:

- X zawiera dokładnie 2 zdarzenia (na 3 różne sposoby).
- X zawiera wszystkie 3 zdarzenia.

Permutacje pasujące do profilu

- $\binom{n}{MP} \cdot 2^{MP}$ - wybory multipętli
- $\binom{n-MP}{P} \cdot 4^{P}$ - wybory pętli
- $\binom{n-MP-P}{M}^2 \cdot M! \cdot 2^{M}$ - wybory multikrawędzi
- $(2n - 2MP - 2M - P)!$ - dowolna permutacja na pozostałych wierzchołkach

X pasujące do profilu

Dla każdej multipętli są 4 opcje:

- X zawiera dokładnie 2 zdarzenia (na 3 różne sposoby).
- X zawiera wszystkie 3 zdarzenia.

Trzy nie zmieniają parzystość $|X|$, ostatnia zmienia i skraca się we wł-wył z jednym z 3 pozostałych.

Permutacje pasujące do profilu

- $\binom{n}{MP} \cdot 2^{MP}$ - wybory multipętli
- $\binom{n-MP}{P} \cdot 4^{P}$ - wybory pętli
- $\binom{n-MP-P}{M}^{2} \cdot M! \cdot 2^{M}$ - wybory multikrawędzi
- $(2n - 2MP - 2M - P)!$ - dowolna permutacja na pozostałych wierzchołkach

X pasujące do profilu

Dla każdej multipętli są 4 opcje:

- X zawiera dokładnie 2 zdarzenia (na 3 różne sposoby).
- X zawiera wszystkie 3 zdarzenia.

Trzy nie zmieniają parzystość $|X|$, ostatnia zmienia i skraca się we wł-wył z jednym z 3 pozostałych. Wynik danego profilu mnożymy przez:

$$(-1)^{P+M} \cdot 2^{MP}$$

Tablicujemy silnie, dwumiany oraz potęgi 2. Każdy profil (MP, P, M) rozpatrujemy w czasie stałym.

Złożoność

$$\mathcal{O}(n^3)$$