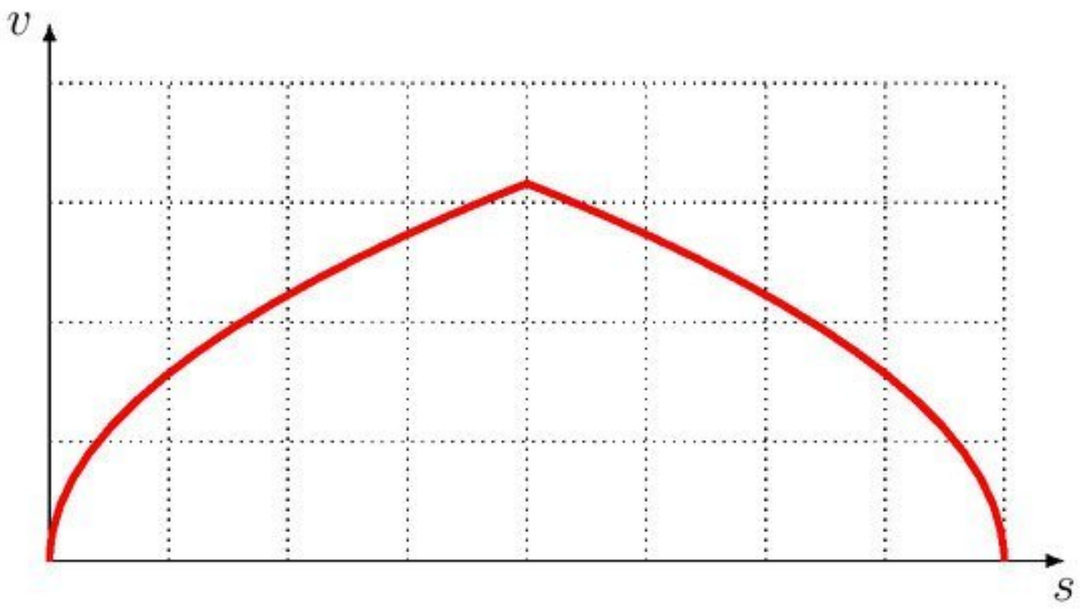
MWPZ 2019 : Szkice rozwiązań

Zadanie A - Autostrada

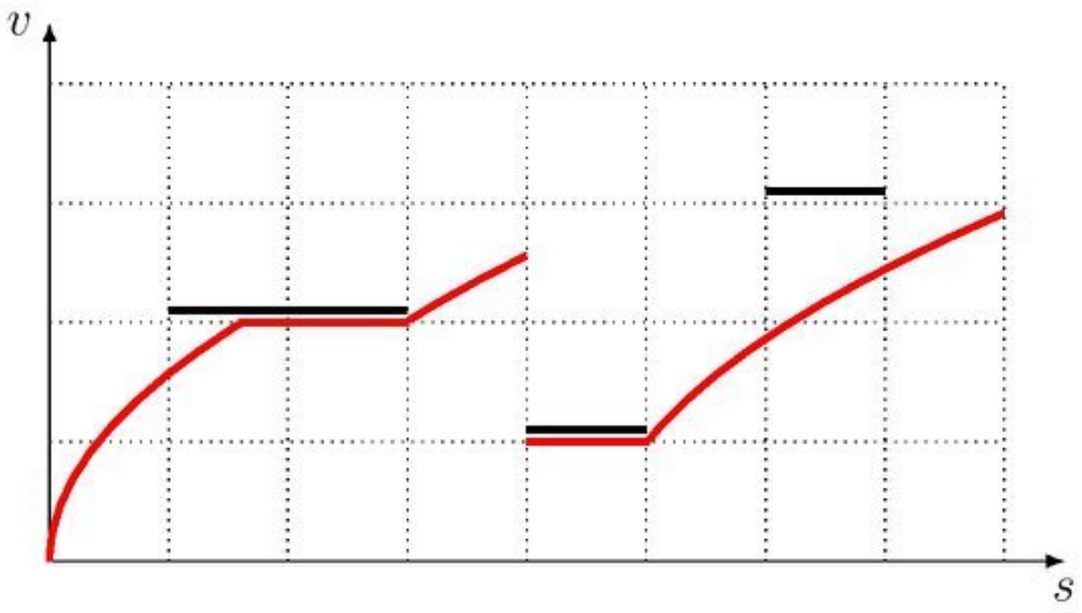
W każdej chwili opłaca nam się albo rozpędzać z maksymalnym przyspieszeniem a km/h^2, albo jechać z prędkością podaną na aktualnie obowiązującym ograniczeniu, albo hamować z maksymalnym przyspieszeniem a km/h^2.

Na początek sortujemy n ograniczeń i zastępujemy je zestawem $O(n)$ ograniczeń nie pokrywających się. Można to zrobić, przeglądając kolejno ograniczenia i trzymając wszystkie obowiązujące ograniczenia na kolejce priorytetowej.

Wygodnie jest narysować sobie wykres prędkości samochodu od pozycji na drodze $v(s)$. Przy braku ograniczeń połowę drogi przyspieszamy, a połowę hamujemy. Wykres składa się z dwóch kawałków odwróconej paraboli $v(s) = \sqrt{2as}$ i wygląda tak:



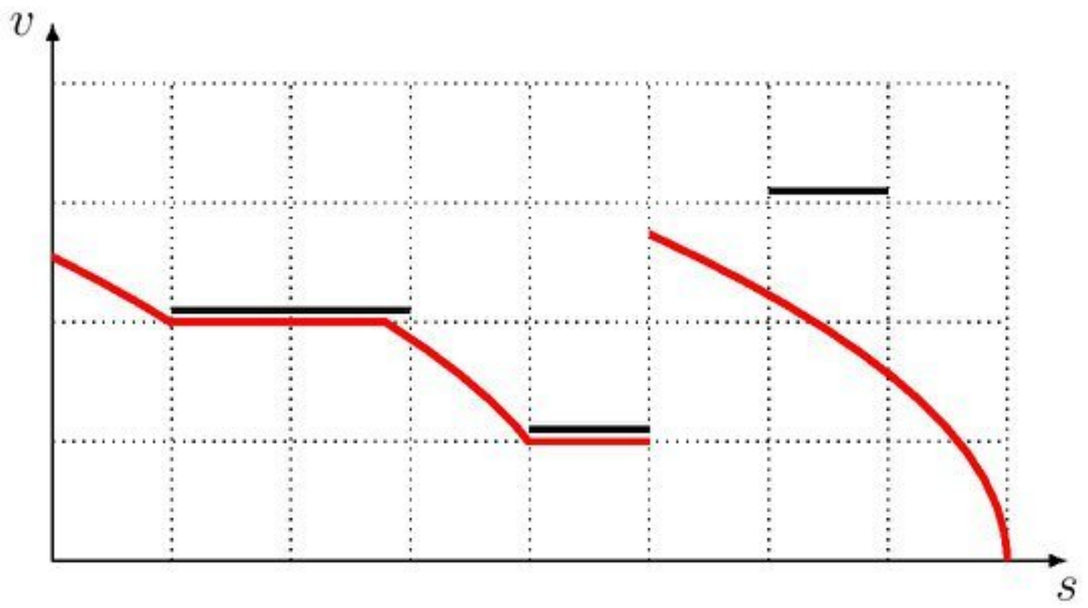
Dodajemy ograniczenia i tworzymy wykres dla samochodu, który rozpędza się z przyspieszeniem a km/h^2, ale może hamować dowolnie szybko. Wykres składa się z kawałków parabolicznych i kawałków poziomych:



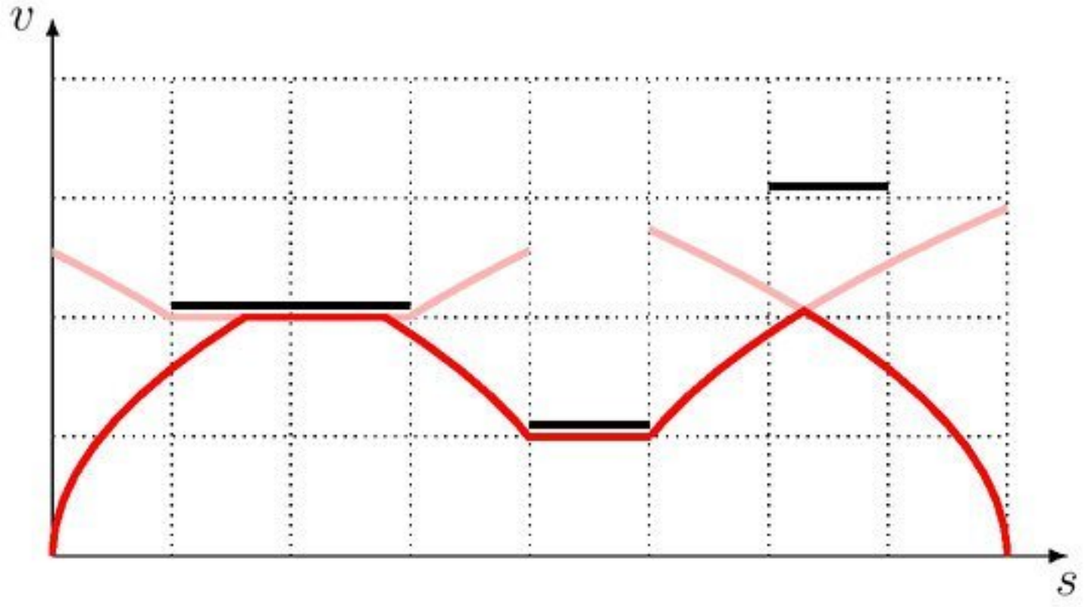
Wykres tworzymy następująco. Zaczynamy parabolę w zerowym punkcie i obliczamy jej przecięcia z kolejnymi ograniczeniami:

- Jeśli się nie przecinają, a ograniczenie jest powyżej paraboli, to kontynuujemy parabolę.
- Jeśli się przecinają, to kończymy parabolę w punkcie przecięcia i dalej poruszamy się poziomo, do końca ograniczenia, a od tego punktu zaczynamy nową parabolę.
- Jeśli się nie przecinają, a ograniczenie jest poniżej paraboli, to „w miejscu” wyhamowujemy do prędkości z ograniczenia i dalej jak w poprzednim przypadku.

Następnie tworzymy drugi wykres dla samochodu, który może rozpędzać się dowolnie szybko, ale hamuje z przyspieszeniem a km/h^2. Wykorzystujemy ten sam algorytm, uruchomiony od tyłu:



Teraz bierzemy minimum z obu wykresów, iterując się po kolejnych kawałkach. Wymaga to obsłużenia przecięcia dwóch parabol:



Na końcu, by uzyskać czas przejazdu, całkujemy odwrotność uzyskanego wykresu, czyli funkcję $1/v(s)$. Dla kawałków będących parabolami, funkcją pierwotną jest $\sqrt{2s/a}$.

Złożoność czasowa rozwiązania to $O(n \log n)$.

Zadanie B - Byki i krowy

Zadanie:

Dany jest N-wierzchołkowy graf G, w którym wyróżniony jest K-elementowy zbiór B. Sprawdzić, czy istnieje klika C w grafie $G[B]$ dla której nie istnieje skojarzenie nasycające zbiór B w grafie indukowanym przez krawędzie między zbiorami B i $G \setminus B$.

Rozwiązanie:

Na początku przekształcamy graf G usuwając wszystkie krawędzie należące do $G[B]$ i dodając wszystkie krawędzie nie należące do $G[B]$ (czyli bierzemy dopełnienie fragmentu grafu). Zadanie teraz polega na sprawdzeniu czy istnieje zbiór niezależny w $G[B]$ (a nie klika), dla którego nie istnieje opisane w treści skojarzenie nasycające. Jasnym jest równieżm że zbiór B stanowi pokrycie wierzchołkowe grafu G.

Łatwo udowodnić, że w grafie G istnieje szukany zbiór niezależny wtedy i tylko wtedy, gdy istnieje pokrycie wierzchołkowe grafu G o mocy nie większej niż $K-1$: w jedną stronę, zamieniając znaleziony zbiór niezależny I o szukanej własności na $N(I) \cap (G \setminus B)$ otrzymamy pokrycie o mocy mniejszej niż K, a w drugą stronę, mając pokrycie wierzchołkowe V_c o mocy nie większej niż $K-1$ szukanym zbiorem niezależnym jest $B \setminus V_c$.

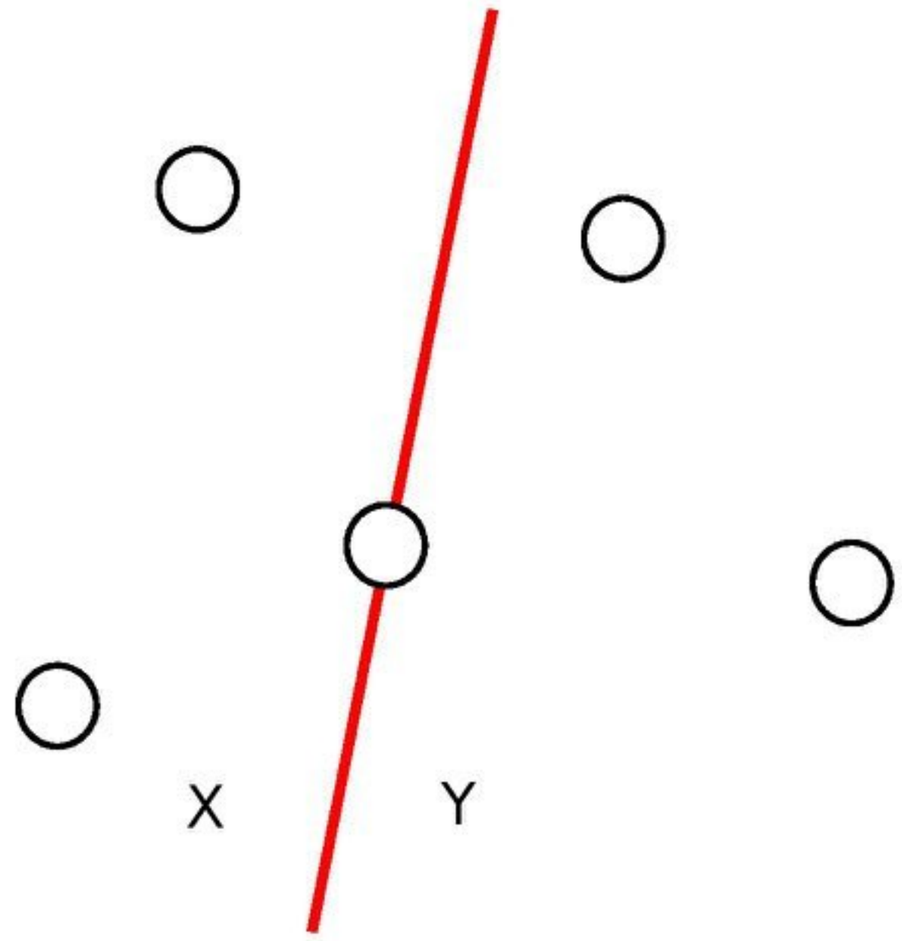
W celu wyznaczenia najmniejszego pokrycia wierzchołkowego V_c grafu G na początku dodajemy do V_c wszystkie wierzchołki o stopniu co najmniej K i usuwamy je z grafu (jeżeli któryś z tych wierzchołków nie należałby do szukanego pokrycia wierzchołkowego, wówczas do pokrycia tego musieliby należeć wszyscy sąsiedzi - co dało by pokrycie mocy co najmniej K). Następnie algorytmem branch-and-bound sprawdzamy, czy istnieje pokrycie rozmiaru $K-1$ - rozpatrując wszystkie wierzchołki $v \in G$ i dodając v do V_c oraz usuwając v z G (w jednej gałęzi) lub dodając zbiór $N(v)$ do V_c i usuwając $N(v)$ z G (w drugiej gałęzi, jednkże tą sytuację rozpatrujemy tylko w przypadku, gdy wierzchołek v nie zawiera sąsiada o stopniu 1). Wierzchołki rozpatrujemy biorąc za każdym razem w pierwszej kolejności sąsiada wierzchołka o stopniu 1 (jeżeli taki istnieje) lub wierzchołek o największym stopniu, pod warunkiem, że stopień ten jest większy niż 2. Jeżeli stopień jest nie większy niż 2, wówczas rozpatrywany graf jest sumą cykli i/lub ścieżek i najmniejsze pokrycie wierzchołkowe można znaleźć w czasie liniowym (w zależności od liczby wierzchołkow) przy użyciu algorytmu dfs. Jeżeli w którymkolwiek momencie działania algorytmu moc zbioru V_c przekroczy $K-1$, powracamy do poprzedniego kroku (węzła w drzewie) w rekurencji.

Przy każdym rozgałęzieniu zwiększamy liczbę wierzchołków należących do pokrycia wierzchołkowego o 1 (pierwsza gałąź) lub co najmniej o 3 (druga gałąź), skąd wynika ograniczenie górne na liczbę kroków rekurencji postaci $T(k) = T(k-1) + T(k-3)$. Złożoność czasowa tego algorytmu wynosi więc $O(K \cdot N + K \cdot 1.4656^K)$ (wartość 1.4656 jest pierwiastkiem wielomianu charakterystycznego $x^3 = x^2 + 1$ dla uzyskanej rekurencji).

Zadanie C - Cyrkulacja śmigła

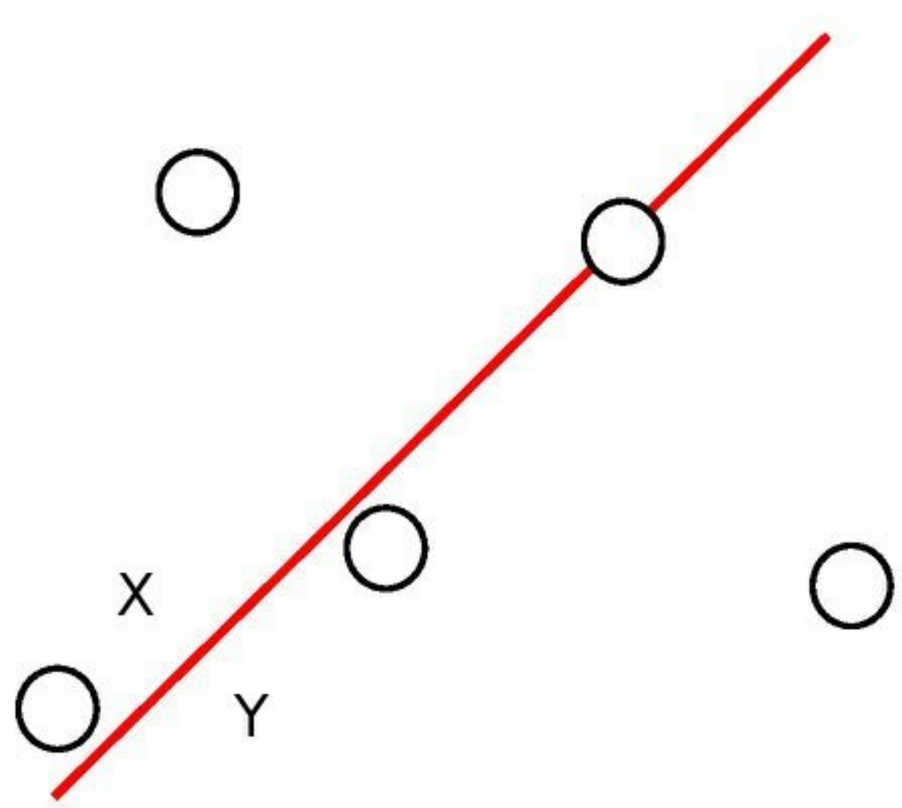
W celu rozwiązania zadania C potrzebne są dwie obserwacje:

- Jeżeli śmigło przecina tylko 1 punkt płaszczyzny, to dzieli pozostałe punkty płaszczyzny w jakimś stosunku X punktów po jednej stronie prostej i Y punktów po drugiej stronie prostej.



Obserwacja 1 *Podczas zmiany punktu obrotu śmigła, nie zmienia się wyznaczony przez prostą podział punktów płaszczyzny.*

Rzeczywiście, w przypadku, gdy nowy punkt był po jakiejś stronie prostej, to przy zmianie środka poprzedni środek znajdzie się po tej samej stronie, w związku z tym suma punktów w X i Y się nie zmieni.



- **Obserwacja 2** *Każda prosta mająca środek obrotu w punkcie P i dzieląca punkty płaszczyzny w stosunku X do Y zostanie odwiedzona przez obrót prostej, której startowy podział płaszczyzny wynosi X do Y.*

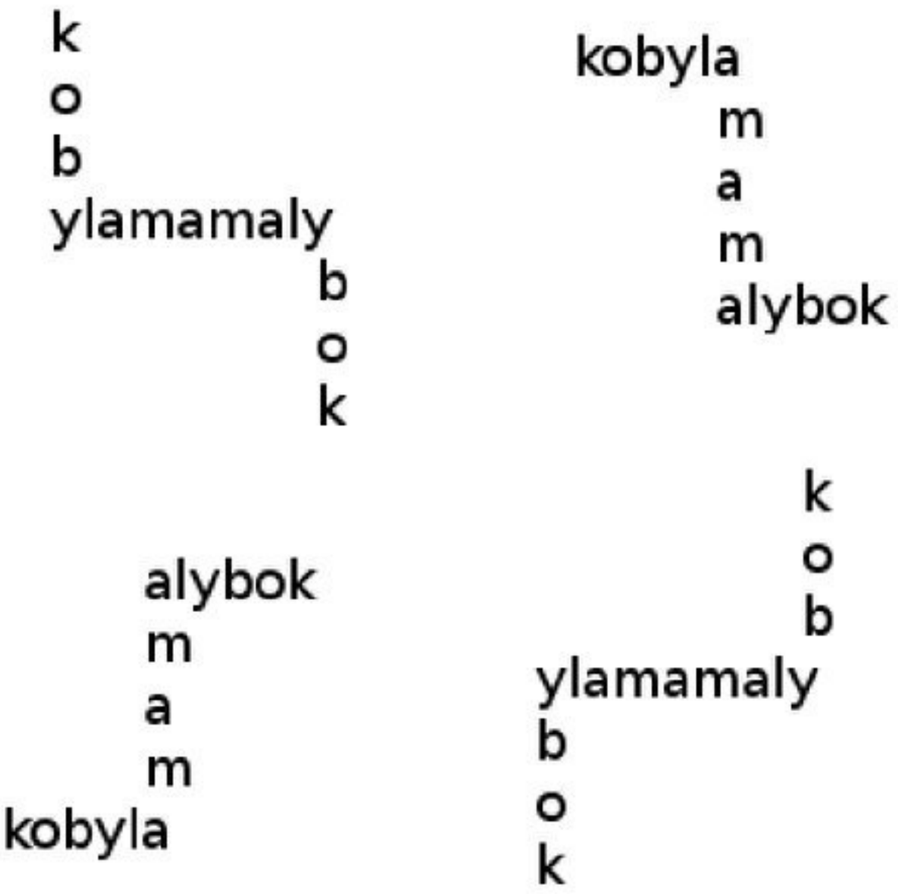
  Istnieje tylko jedna prosta spełniająca warunki : prosta przechodzi przez płaszczyznę pod zadanym kątem K i dzieli jej punkty na dwie części X punktów po jednej stronie prostej i Y punktów po drugiej. W związku z tym przy obrocie o 360 stopni prosta ta musi być częścią rozwiązania.

  Zauważmy, że po obrocie o 180 stopni część X znajdzie się po prawej stronie prostej a część Y po lewej. W związku z czym nie interesuje nas, czy punkty znajdują się na prawo, czy na lewo od prostej, tylko same wartości tego podziału.

Algorytm: Dla każdego punktu p posortuj kątowo pozostałe punkty płaszczyzny. Liniowo wyznacz podziały prostymi punktów, gdy p jest środkiem. Jeżeli dla punktu p więcej niż jedna prosta wyznacza taki sam podział zsumuj je. Zbierz dla wszystkich punktów te same podziały we wspólne zbiory. Wyznaczają nam one przypadki, dla których odpowiedź w zadaniu brzmi "YES".

Złożoność $O(n^2 \log n)$.

Zadanie D - Dziwne palindromy



Mamy cztery możliwe ustawienia kształtu wężyka (wszystkie widoczne na obrazku). Jeśli będziemy umieli rozwiązać zadanie dla jednego z nich, to wystarczy cztery razy uruchomić nasz algorytm na odpowiednio obróconej lub odbitej symetrycznie tablicy. Ustalmy więc, że rozwiązujemy zadanie dla kształtu pokazanego w lewym górnym rogu rysunku.

Zauważmy, że mamy po $O(m^2)$ kandydatów na centralną część wężyka w każdym wierszu, czyli łącznie $O(nm^2)$ w całym tekście. Możemy w łatwy sposób sprawdzić, którzy z kandydatów są palindromami w łącznym czasie $O(nm^2)$ przeglądając wszystkie możliwe środki centralnej części i zachłannie rozszerzając palindrom w obie strony (przetwarzamy przy tym parzyste i nieparzyste palindromy osobno).

Gdybyśmy teraz dla każdej możliwej centralnej części znali maksymalną możliwą długość ogonków, to potrafilibyśmy podać odpowiedź. Naiwne sprawdzanie ogonków daje nam więc rozwiązanie działające w czasie $O(n^2m^2)$, które jest nieco za wolne.

Jednym ze sposobów przyspieszenia rozwiązania jest policzenie haszy wszystkich kolumn w obie strony. Dzięki temu możemy w czasie stałym sprawdzać czy dana długość ogonka jest możliwa. Dzięki temu, wykorzystując wyszukiwanie binarne, możemy policzyć maksymalną długość ogonka dla konkretnego centralnego fragmentu w czasie $O(\log n)$ uzyskując rozwiązanie w czasie $O(nm^2 \log n)$ (czyli $O(nm^2 \log n + n^2 m \log m)$ po rozpatrzeniu wszystkich kształtów). Optymalna implementacja takiego podejścia powinna mieścić się w limicie czasu.

Praktyczna uwaga implementacyjna: autorowi nie jest znana żadna metoda obalania haszowania modulo 2^{64} w tym zadaniu. Szeroko stosowana metoda opiera się na użyciu słów Thue'go-Morse'a. Jeśli haszujemy modulując przez potęgę dwójki, to niezależnie od wybranej podstawy, odpowiednio długie słowo Thue'go-Morse'a będzie generowało dużo kolizji haszy wśród swoich podsłów. Jednak przy haszowaniu modulo 2^k potrzebujemy w tym celu wziąć słowo długości rzędu $2^{\sqrt{2k}}$. Dla $k = 64$ oznacza to słowo długości większej niż 2500, czyli za długie, aby użyć go w testach. W praktyce, licząc hasze modulo potęga dwójki nie musimy w ogóle wykonywać kosztownego czasowo działania modulo, co znacznie przyspiesza program.

Można jednak rozwiązać zadanie w jeszcze lepszej złożoności czasowej. Ustalmy wiersz r, w którym będziemy rozpatrywać kandydatów na centralne fragmenty i popatrzmy na wszystkie słowa, które zaczynają się w wierszu r i kończą na pierwszym wierszu (idące w górę) oraz te, które zaczynają się w wierszu r i kończą na ostatnim wierszu (idące w dół). Wszystkich takich pionowych słów mamy $2m$.

Oznaczmy jako $T[i][j]$ długość najdłuższego wspólnego prefiksu słowa idącego w kolumnie i w górę i słowa idącego w kolumnie j w dół. Oczywistym jest, że znając wartości tablicy T potrafimy w czasie

stałym podawać długości ogonków. Pokażemy, że potrafimy wyliczyć wszystkie wartości tej tablicy w łącznym czasie $O(nm)$.

Mamy łącznie $2m$ pionowych słów. Możemy je posortować w czasie $O(nm)$ używając radix sorta. Następnie dla takiego posortowanego ciągu wyliczamy brutalnie wartości tablicy lcp (długości najdłuższych prefiksów dla sąsiednich par słów). Zauważmy teraz, że $T[i][j] = \min_{pos(i) \leqslant k < pos(j)} lcp[k]$, gdzie $pos(i), pos(j)$ oznaczają pozycje i-tego słowa w górę i j-tego słowa w dół w posortowanym ciągu (zakładamy tutaj bez straty ogólności, że $pos(i) < pos(j)$). Możemy teraz łatwo spamiętać wszystkie takie minima na przedziałach w czasie $O(nm)$.

W sumie otrzymaliśmy rozwiązanie działające w czasie $O(nm^2 + n^2m)$.

Zadanie E - E300

Zadanie polegało na podziale 3-regularnego grafu na dwa zbiory A i B, tak, by w żadnym nie było wierzchołka stopnia większego niż 1.

Należy zauważyć, że bardzo trudno znaleźć taki podział w jednym kroku. Jednak mając dany podział, który nie jest poprawny możemy w wydajny sposób go ulepszyć. Mianowicie w przypadku, gdy w jakimś ze zbiorów występuje wierzchołek stopnia 2 lub 3 to, gdy przeniesiemy go do drugiego zbioru dwupodziału otrzymamy "lepszy" podział. Przez lepszy podział rozumiemy taki, w którym liczba krawędzi pomiędzy zbiorami A i B jest większa. W każdej iteracji naszego algorytmu zwiększamy więc liczbę krawędzi między A a B o jeden. Jako, że liczba krawędzi w całym grafie jest ograniczona przez $3n$, nasz algorytm działa w czasie $O(n)$.

Możliwym błędem w tym zadaniu było stosowanie czasochłonnych struktur do przetrzymywania wierzchołków np. w *hashsetach* i wyszukiwanie sąsiadów z wykorzystaniem metody *find*. Rozwiązania takie otrzymywały TLE, gdyż o wiele wydajniej jest trzymać podział wierzchołków w statycznej tablic i aktualizować jest stan po każdej zamianie.

Zadanie F - Fontanna

Symulacja

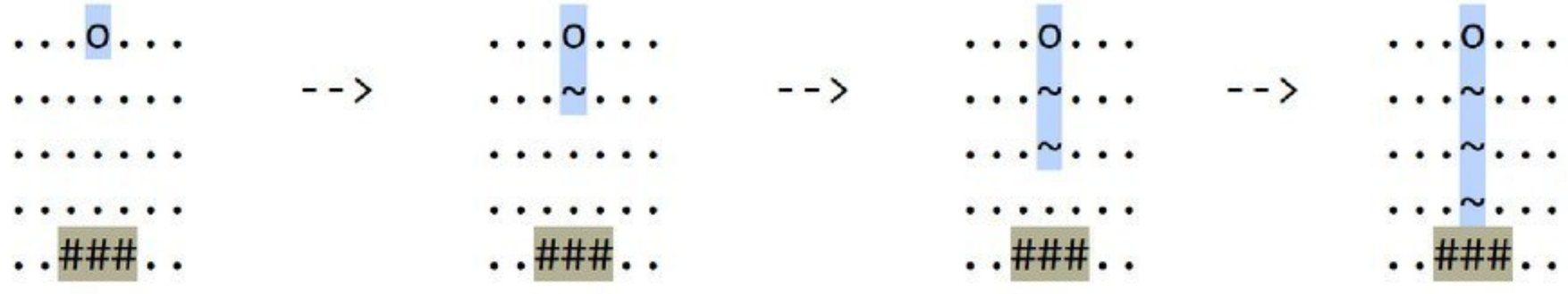
Należy napisać symulację, wykonując w każdym kroku następujące operacje:

1. Wydłużyć strumienie (o jedno pole w dół)
2. Rozpropagować powierzchnie (rozlać w poziomie)
3. Podnieść ograniczone powierzchnie (o jedno pole do góry)
4. Rozpropagować powierzchnie jeszcze raz
5. Scalić powierzchnie, które się wyrównały i stykają się

Na początku do symulacji dodajemy strumienie w każdym polu 'o'. Zaczynamy z pustą listą wyników. Jeżeli w którymkolwiek punkcie kroku symulacji zostanie zalane pole '?', dodajemy współrzędne tego pola do listy wyników (odpowiednio posortowane w obrębie jednego kroku symulacji, zgodnie z treścią zadania). Jeżeli nie można już wykonać kolejnego kroku (nie pojawia się żadne nowe pole z wodą), kończymy symulację i wypisujemy listę wyników.

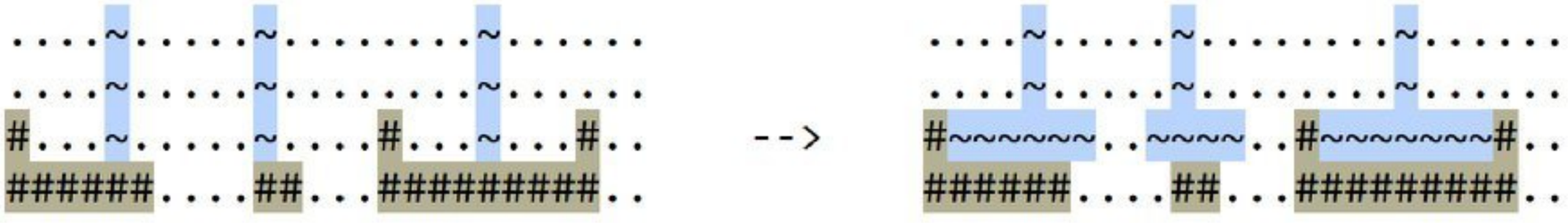
Wydłużanie strumieni

Każdy strumień wydłużamy o jedno pole w dół, jeżeli to pole jest puste. Jeżeli strumień dotknie wówczas kamienia, to od razu tworzymy w tym miejscu powierzchnię (wystarczy jedno pole, które w punkcie 2. rozpropagujemy).



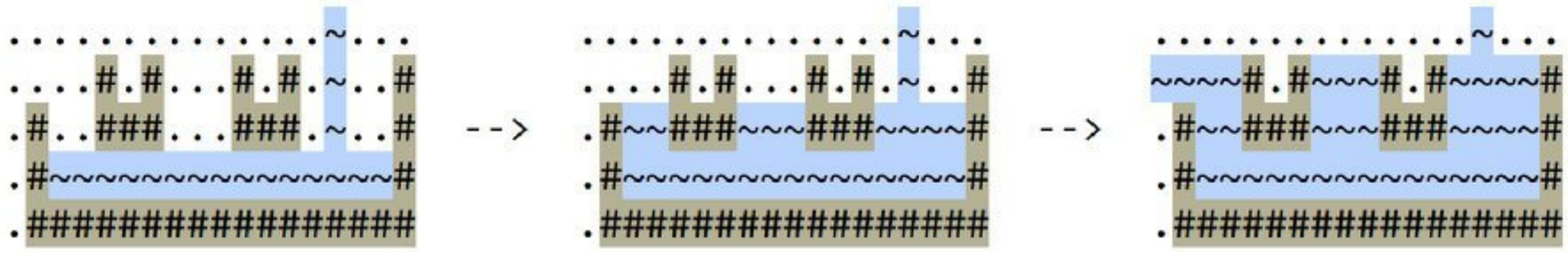
Rozpropagowanie powierzchni

Każdą z powierzchni, rozszerzamy w poziomie, o ile to możliwe. Jeżeli dojdziemy do kamienia, kończymy rozszerzanie. Jeżeli dojdziemy do pola, pod którym nie ma kamienia, kończymy rozszerzanie i dodatkowo tworzymy w tym miejscu nowy strumień.



Podnoszenie powierzchni

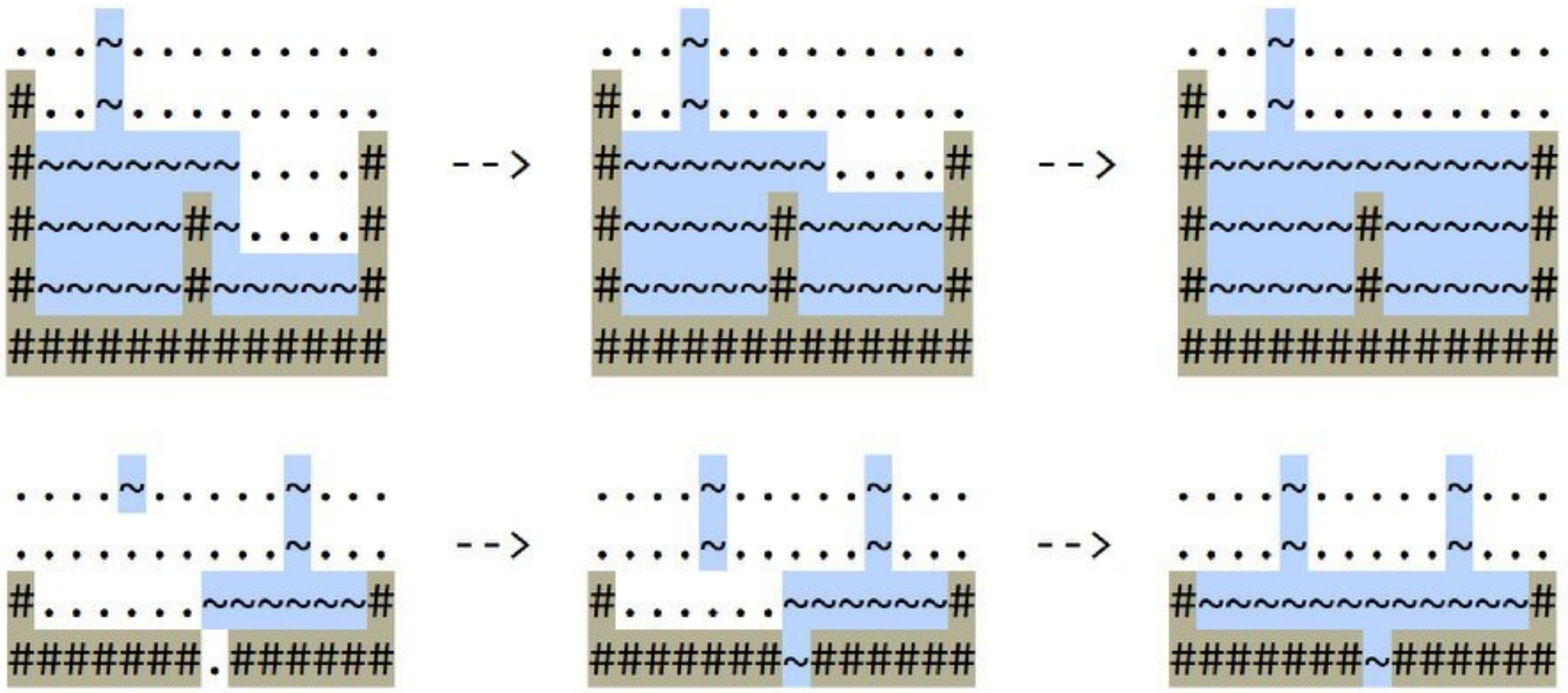
Każdą z powierzchni, która powinna się podnieść, uaktualniamy, wypełniając wodą wszystkie niezajęte przez kamień pola bezpośrednio powyżej tej powierzchni.



Aby wiedzieć, które powierzchnie należy podnieść, trzeba sprawdzić, czy należą do jednego zbioru naczyń połączonych (istnieje pomiędzy nimi jakieś połączenie ograniczone ścianami. Ponadto, powierzchnie podnosimy tylko wówczas, gdy wszystkie powierzchnie, należące do tego samego zbioru naczyń połączonych, są ograniczone ścianami (tzn. woda nie wylewa się z tychże naczyń połączonych żadnym strumieniem). W tym celu należy utrzymywać listę naczyń połączonych. Można do tego wykorzystać strukturę zbiorów rozłącznych (efektywna implementacja Find Union z kompresją ścieżek i rangami). Zbiór naczyń połączonych opisujemy za pomocą listy powierzchni, które do niego należą. Każda powierzchnia początkowo tworzy jednoelementowy zbiór naczyń połączonych. W czasie symulacji łączenie naczyń następuje poprzez scalanie zbiorów powierzchni, co opisane jest w kolejnym punkcie.

Scalanie powierzchni

Powierzchnie mogą się wyrównać i zetknąć na skutek podnoszenia lub rozpropagowania w poziomie (w punktach 2-4). W takim przypadku łączymy ze sobą zbiory naczyń połączonych, do których należały obie powierzchnie (uaktualniamy strukturę Find Union).



Złożoność

Rozwiązanie można zaimplementować w czasie $O(n^2\alpha(n))$ (gdzie $\alpha(n)$ jest odwrotnością tzw. funkcji Ackermanna), jeżeli wykorzysta się szybką implementację struktury zbiorów rozłącznych.

Zadanie G - Graf Jasia

Rozwiązanie zadania polegało na wygenerowaniu ciągu $2M + 1$ liczb według wzoru podanego w treści zadania, utworzeniu list sąsiedztw tego grafu na podstawie tego ciągu oraz wyznaczeniu wartości zależnych od kolejności sąsiadów na posortowanych listach sąsiedztw. W celu posortowania list sąsiedztw należało stworzyć graf odwrotny (ropatrując kolejno wierzchołki v od 0 do N-1, dla każdego sąsiada d wierzchołka v dodać do nowo tworzonego grafu odwrotnego krawędź (d, v)), chociaż posortowanie każdej listy sąsiedztwa z osobna również było akceptowane.

Złożoność czasowa wynosi $O(N + M)$.

Zadanie H - Hurtownia z liczbami

Zadanie

Podaj k-tą liczbę o iloczynie cyfr równym n.

Rozwiązanie

Obliczamy na początku dynamika DP$[d][n]$ – ile jest liczb d-cyfrowych (być może z zerami wiodącymi) o iloczynie cyfr n. Stanów w tym dynamiku nie ma zbyt wiele, bo sensowne n są postaci $2^\alpha \times 3^\beta \times 5^\gamma \times 7^\delta$.

Każde zapytanie będziemy teraz rozwiązywać ustalając najpierw ile cyfr ma mieć wynik, a potem po kolei ustalając cyfry wyniku od najbardziej znaczącej.

Przypadek $n \neq 0$

Żeby ustalić liczbę cyfr d, sumujemy dla kolejno rosnących d wartości DP$[d][n]$, aż przekroczymy k. Wtedy wiadomo, że jest $d-1$ cyfr.

Testujemy możliwości na pierwszą cyfrę c od 1 do 9, sprawdzając czy n jest podzielne przez c i wtedy odejmując od k wartość DP$[d-1][n/c]$. Gdy k spadnie poniżej 0, cofamy ostatni ruch, a c dla którego spadło jest pierwszą cyfrą wyniku. Kolejne cyfry odzyskujemy tak samo.

Przypadek $n = 0$

Przypadek z zerem jest podobny, ale trzeba uważać na dwie rzeczy:

- zera wiodące zapamiętane w DP$[d][n]$,
- dzielenie przez zero gdyby $c = 0$,

Generalnie, jeśli już w wypisanych cyfrach pojawiło się 0 to wtedy jest łatwo (bo dowolna końcówka pasuje). A jak nie pojawiło się to wcześniejszy pomysł działa, bo nie dzielimy przez 0.

Zadanie I - Iloczyn Mikołaja

Na wejściu mamy dany ciąg n liczb nieparzystych dodatnich: $a_1, a_2, ..., a_n$ oraz liczbę całkowitą dodatnią M. Naszym zadaniem jest znaleźć rozkład tych liczb na iloczyn dwóch liczb całkowitych dodatnich $a_k = p_k * q_k$ tak, aby $p_k \leqslant q_k \leqslant p_k + M$ oraz by ciąg p_k był niemalejący. Ze wszystkich takich rozkładów wybieramy ten o najmniejszym możliwym iloczynie liczb p_k.

Niech D będzie dowolną liczbą całkowitą dodatnią. Możemy wówczas w pre-processingu znaleźć żądane rozkłady dla wszystkich liczb od 1 do D w złożoności $O(\sqrt{D} * M)$, gdyż dla wszystkich liczb $1 \leqslant a \leqslant D$ oraz szukanych rozkładów $a = p * q$, zachodzą nierówności: $1 \leqslant p \leqslant \sqrt{a} <= \sqrt{D}$ oraz $p \leqslant q \leqslant p + M$. Dla każdego a zapamiętujemy je w kolejności rosnących wartości p.

Rozwiązanie wzorcowe polega na znalezieniu dla kolejnych wartości a_k rozkładu $a_k = p_k * q_k$, spełniającego dane warunki, o najmniejszej możliwej wartości p_k. Jeżeli $a_k \leqslant D$, to przeglądamy kolejne rozkłady a_k (wyznaczone w pre-processingu) w kolejności rosnących p_k. Jeżeli $p_k < p_{k-1}$, to usuwamy ten rozkład i wybieramy kolejny aż w końcu $p_k \geqslant p_{k-1}$. Jeżeli natomiast $a_k > D$, to wyznaczamy wszystkie żądane rozkłady a_k za pomocą algorytmu Fermata. Algorytm ten wykorzystuje następującą tożsamość:

$$a_k = p * q = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 = A^2 - B^2$$

i szuka żądanych rozkładów, wykorzystując przedstawienie danej liczby jako różnicy kwadratów. Dla kolejnych wartości $A = \lceil\sqrt{a_k}\rceil, \lceil\sqrt{a_k}\rceil + 1, \lceil\sqrt{a_k}\rceil + 2, ...$ sprawdza, czy liczba $A^2 - a$ jest kwadratem liczby całkowitej. Jeśli tak, to wyznacza ona pewien rozkład liczby a_k. Wykonując obliczenia, możemy jednak pokazać, że jeżeli $A \geqslant \lceil\sqrt{a_k}\rceil + 1$ oraz $0 \leqslant q - p \leqslant M$, to

$$a_k \leqslant \left(\frac{M^2}{8} - \frac{1}{2}\right)^2$$

Zatem, dla a_k większych od tej wartości, jedyny możliwy rozkład wyznacza już pierwszy krok algorytmu Fermata. Wybieramy więc jako D wartość $\left(\frac{M^2}{8} - \frac{1}{2}\right)^2$. Zakładając dla uproszczenia, że możemy w czasie stałym skorzystać z pamięci dostępnej dla rozkładów liczb niewiększych od D oraz obliczyć pierwiastek danej liczby, powyższy algorytm działa w złożoności $O(M^3 + n)$.

Zadanie J - Jak zdobyć królestwo

Rozwiązanie

Parzyste wymiary

Rozważmy na początek układankę, która ma parzystą liczbę kolumn i wierszy.

Ustalmy, że niebieskiej ściance przypisujemy 0, a zielonej 1.

Rozważmy wtedy klocki o współrzędnych $a_1 = (i, j)$, $a_2 = (n - i + 1, j)$, $a_3 = (i, m - j + 1)$, $a_4 = (n - i + 1, m - j + 1)$. Zauważmy, że każda możliwa operacja nie zmienia parzystości tych pól, zatem, aby układanka była możliwa do ułożenia to liczba pól zielonych i niebieskich musi być parzysta.

Dodatkowo, możemy zauważyć że dwie dodatkowa sytuacje są niemożliwe. Gdy pola zielone i niebieskie są w przeciwległych narożnikach.

Możemy to udowodnić rozważając sumę $a_1 - a_2 + a_3 - a_4$, wtedy możemy rozważyć wszystkie możliwości a_1 i a_2

- $a_1 = 0, a_2 = 0$, wtedy po obrocie będzie $a_1 = 1, a_2 = 1$, co nie zmienia sumy
- $a_1 = 1, a_2 = 0$, wtedy po obrocie będzie $a_1 = 1, a_2 = 0$, co nie zmienia sumy
- $a_1 = 0, a_2 = 1$, wtedy po obrocie będzie $a_1 = 0, a_2 = 1$, co nie zmienia sumy
- $a_1 = 1, a_2 = 1$, wtedy po obrocie będzie $a_1 = 0, a_2 = 0$, co nie zmienia sumy

Zatem musimy rozwiązać sytuację, gdy wartość 0 mają wszystkie 4 pola lub 2 sąsiednie.

Rozważmy 4 pola o wartości 0. Wtedy sekwencja ruchów:

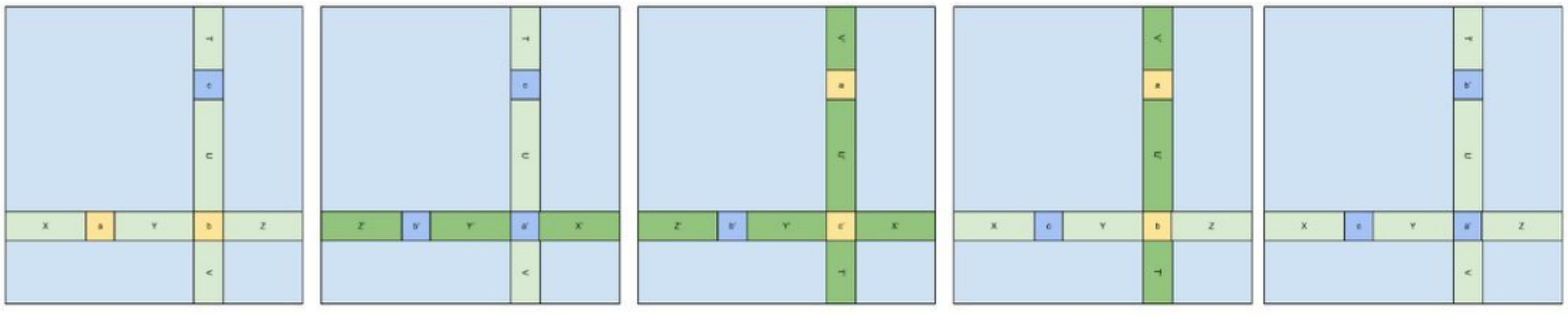
```
W i
W n - i + 1
K m - j + 1
W i
W n - i + 1
K m - j + 1
```

Zmienia wartość wszystkich pól na jeden bez zmiany pozostałych pól.

Natomiast, gdy dwa pola mają wartość 0 (załóżmy że a_3 i a_4), wtedy sekwencja:

```
W n - i + 1
K m - j + 1
W n - i + 1
K m - j + 1
```

Zmienia wartości pól z 0 na 1.



Nieparzyste wymiary

Co zrobić z elementami w środkowej kolumnie lub wierszu, te elementy można bardzo łatwo naprawić przed przystąpieniem do rozwiązywania pozostałych elementów. W pierwszej kolejności, należy obróć klocek środkowy (jeżeli tego wymaga), a następnie w wierszu (obracając odpowiednie kolumny), i kolumnie (obracając wiersze).

Na koniec tego etapu powinniśmy uzyskać krzyż ułożony z 1 na środkowej kolumnie i wierszu.

Szacowanie ilości ruchów

Łatwo zauważyć, że powyższy algorytm nie generuje więcej niż $2 * n * m$ ruchów.

W pierwszym etapie na każdy element ze środkowego wiersza i środkowej kolumny wykonujemy jeden ruch. Następnie, aby naprawić każdą czwórkę wykonujemy nie więcej niż 6 ruchów. Zatem górnych oszacowaniem liczby ruchów jest $1.5 * n * m$

Złożoność

Zauważmy, że aby rozwiązać parzystą układankę nie musimy nawet wykonywać symulacji obrotów, zatem ten etap możemy wykonać w czasie $O(n * m)$, podczas pierwszego etapu (w przypadku nieparzystej układanki) mamy tylko maksymalnie $n + m - 1$ obrotów, zatem tutaj możemy wykonać symulację, bez straty w złożoności czasowej. Ten etap również będzie działał w czasie $O(n * m)$

Zatem całe rozwiązanie będzie działać w czasie $O(n * m)$

Zadanie K - Korporacja Świętego Mikołaja

Niech $F(x)$ oznacza liczbę poprawnych ciągów zaczynających się 1 i kończących liczbą nie większą niż x. Ciągów zaczynających się liczbą a i nie przekraczających x jest $F\left(\left\lfloor\frac{x}{a}\right\rfloor\right)$, ponieważ każdy ciąg taki ciąg po podzieleniu wszystkich wyrazów przez a staje się ciągiem zaczynającym się jedynką i nie przekraczającym $\left\lfloor\frac{x}{a}\right\rfloor$. Wynik dla przedziału $[l, r]$ jest więc równy $\sum_{k=l}^{r} F\left(\left\lfloor\frac{r}{k}\right\rfloor\right)$.

Kluczowy dla rozwiązania będzie następujący lemat: Dla dowolnego k całkowitego dodatniego wśród liczb $\left\lfloor\frac{k}{1}\right\rfloor, \left\lfloor\frac{k}{2}\right\rfloor, \ldots, \left\lfloor\frac{k}{k}\right\rfloor$ występuje $O(\sqrt{k})$ różnych liczb. Korzystając z tego możemy sprowadzić policzenie odpowiedzi do policzenia wartości funkcji F w $O(\sqrt{n})$ punktach.

Ciąg zaczynający się jedynką może być jednoelementowy, lub po jedynce mieć poprawny ciąg zaczynający się dowolną większą liczbą. Dla ustalonej d takich ciągów jest $F\left(\left\lfloor\frac{x}{d}\right\rfloor\right)$, stąd $F(x) = 1 + \sum_{d=2}^{x} F\left(\left\lfloor\frac{x}{d}\right\rfloor\right)$. Ponownie korzystając z lematu możemy przekształcić tę sumę tak, żeby odwoływała się do $O(\sqrt{x})$ mniejszych wartości. Mając ten wzór możemy rekurencyjnie wyznaczyć wszystkie wartości F. Korzystając z tożsamości $\left\lfloor\frac{\left\lfloor\frac{n}{a}\right\rfloor}{b}\right\rfloor = \left\lfloor\frac{n}{ab}\right\rfloor$ otrzymujemy, że rozwiązanie obliczające wynik dla przedziału $[l, r]$ przejrzy jedynie liczby postaci $\left\lfloor\frac{r}{x}\right\rfloor$, czyli $O(\sqrt{r})$ różnych liczb. Łączna liczba zejść rekurencyjnych dla liczb mniejszych od $\sqrt{r}$ jest równa $\sum_{k=1}^{\lfloor\sqrt{n}\rfloor} \sqrt{i}$, czyli $O(n^{\frac{3}{4}})$, a dla większych $\sum_{k=1}^{\lfloor\sqrt{n}\rfloor} \sqrt{\left\lfloor\frac{n}{k}\right\rfloor}$, czyli też $O(n^{\frac{3}{4}})$, stąd rozwiązanie obliczające wartości F rekurencyjnie ze spamiętywaniem działa w złożoności $O(n^{\frac{3}{4}})$. Intuicyjnie jednak widać, że dla wielu zapytań argumenty dla których algorytm oblicza wartości funkcji F będą się pokrywać, stąd dla większej liczby testów średnia złożoność będzie mniejsza. Dla dowolnego M łączny czas działania jest co najwyżej rzędu $\sum_{i=1}^{M} \sqrt{i} + t \sum_{i=1}^{\lfloor n/M \rfloor} \sqrt{\frac{n}{i}}$ czyli $O(M\sqrt{M} + t\sqrt{N}\sqrt{\frac{N}{M}})$. Funkcja ta osiąga minimum dla $M = O(\sqrt{nt})$, czas działania dla takiego M jest równy $O((nt)^{\frac{3}{4}})$.

Zadanie L - Liga tenisa

W zadaniu należało z sukcesem zaimplementować podane reguły obliczania wyniku. Do trudnych przypadków, dla zawodników należały:

```
SSSSSRRSSASRRSRSSSDRSSSRASRSSSSRSASASSSSRSSRDSSRSDSARRSADSSARRASSRRSSSRRSSSRRSSRSSSS
RSRRRSSSRSSRRSARASSSRSRSRRRSSDSSSSSASSSSRSASRRDSRSSSRSAAARRSSSASSSS
```

7-6(0) 6-3

oraz

```
RSSRASSSRSSRSSSSSSRSSSSSSSSSSRSASSSSRSSSSSSSSSSARRDSSSRRRSARSSRSSSSRRRRSSARRRSRSSSSSARSASDS
SSRSSSSSARRSRSSSSSRSSRSRSSSSSRSRASSRRSSSSSRSRSSASSRSSSSSRSRSSSRSSSSDSSARRSSSSSARSSARSASDR
RSSASDSRSSRSRRSSSSRSASSSDRSSRSSRSASSSSSSSRSSDSRSRSRSRSSSSRSSSSSRRR
```

5-7 7-6(4) 7-6(4)