AiSD L2

Maurycy Borkowski

17.03.2021

Zadanie 2

```
I <- lista par odcinkow
S = set()
sorted(A, key = lambda x: x.fk)
S.add(A[0])

last = A[0].k
for i in range(2,n):
    if A[i].p > last:
        S.add(A[i])
        last = A[i].k
```

Sortujemy odcinki rosnąco po ich końcach.

Bierzemy zachłannie odcinki jeżeli możemy tzn. nie nachodzą na siebie. Możemy go wziąć jeżeli początek jest większy niż koniec ostanio wziętego (ostatnio wzięty ma największy koniec w S, na początku je sortowaliśmy).

Oznaczmy jako I_0 odcinek o najmniejszym końcu. ze wszystkich odcinków i I_k odcinek o najmniejszym końcu w dowolnym rozwiązaniu optymalnym B. Zauważmy, że rozwiązanie $A = B \setminus \{I_k\} \cup \{I_0\}$ też jest optymalne bo dodany odcinek nie nachodzi się na żaden inny (B było poprawnym rozwiązaniem i $k_0 < k_k$) oraz $|A| = |B|$.

Argumentacje możemy indukcyjnie powtórzyć dla podproblemu ze zbiorem odcinków $I' = \{j \in I : p_j \geqslant k_1\}$, tam ponownie pokazujemy, że zachłanne wzięcie najmniejszego (względem końca) odcinka nie zepsuje nam optymalności rozwiązania.

Zadanie 3

```
M = a*b + 1
while aPb > 0:
    k = min([i if 1/i <= a/b for i in range(1,M)])
    aPb = aPb - 1/k
```

Nie prowadzi to do rozwiązania optymalnego, kontrprzykład $\frac{5}{121}$:
Zachłannie weźmiemy:

$$\frac{5}{121} = \frac{1}{25} + \frac{1}{757} + \frac{1}{763309} + \frac{1}{873\,960\,180\,913} + \frac{1}{1\,527\,612\,795\,642\,093\,418\,846\,225}$$

Natomiast optymalne rozwiązanie wynosi:

$$\frac{5}{121} = \frac{1}{33} + \frac{1}{121} + \frac{1}{363}$$

Musimy pokazać dwie rzeczy do znajdowania rozwiązania:

- suma ułamków wybranych przez algorytm będzie się sumowała do $\frac{a}{b}$
- ułamki wybrane przez algorytm będą unikalne

Dowód.

Sumowanie

Pokażemy, że licznik $\frac{a}{b}$ będzie zbiegał do 1.
Oznaczmy, przez $\frac{1}{u}$ ułamek zachłannie brany przez algorytm tj:

$$\frac{1}{u-1} < \frac{a}{b} \leqslant \frac{1}{u} \tag{1}$$

Po odjęciu go zostanie nam reszta:

$$\frac{a}{b} - \frac{1}{u} = \frac{au-b}{bu}, \tag{2}$$

dalej z (1):

$$\frac{1}{u-1} < \frac{a}{b} \implies a > au - b \tag{3}$$

Wiemy, że $a, u, b \in \mathbb{N}$ stąd wiemy, że liczniki będą zbiegały do 1.

Unikalność

Zakładamy niewprost, że dwa razy weźmiemy jakiś ułamek $\frac{1}{u}$, ale

$$1 \geqslant \frac{1}{u-1} \iff 2 \geqslant 1 + \frac{1}{u-1} \iff 2 \geqslant \frac{u-1+1}{u-1}$$
$$\iff 2 \geqslant \frac{u}{u-1} \iff \frac{2}{u} = \frac{1}{u} + \frac{1}{u} \geqslant \frac{1}{u-1}$$

otrzymujemy sprzeczność bo to oznacza, że mogliśmy odjąć $\frac{1}{u-1}$ czyli większy ułamek niż $\frac{1}{u}$ gdy odejmowaliśmy $\frac{1}{u}$ po raz pierwszy. $\square$

Zadanie 4

Lemat 1. *Dowolne optymalne kolorwanie możemy sprowadzić do optymalnego kolorwania, z pokolorwanymy liściami.*

Dowód. K - dowolne optymalne kolorwanie, tż. liść u nie jest pokolorwany. Niech w oznacza najbliższy pokolorwany wierzchołek u. Zamieniamy je kolorowaniem pokażemy, że nowe kolorwanie K' dalej jest optymalne. Wystarczy, więc pokazać, że jest poprawne. Załóżmy niewprost, że nie jest:
Istenieje ścieżka S z u tż. ma ponad k kolorowych wierzchołków. Oznaczmy jeszcze przez P ścieżkę z u do w, tam jest tylko jeden pokolorwany wierzchołek (w najbliższy kolorowy u). Rozważmy ścieżkę $L = S + P - (S \cap P)$. Ta ścieżka ma dokładnie tyle samo kolorowych wierzchołków z kolorwaniem K jak i z K' (zarówno, u i w są w tej ścieżce). Zatem istnieje, ścieżka L po kolorwaniu K, tż. ma ponad k kolorowych wierzchołków. Sprzeczność. □

Lemat 2. *Dodanie dowolnych pokolorwanych liści do G nie psuje optymalności w $G' = G + liscie$ (G z dodanymi liśćmi) z $k' = k + 2$.*

Dowód. Oznaczmy przez K kolorwanie optymalne w G, pokażemy, że
$K' = K + liscie$ jest optymalne w $G' = G + liscie$ z $k' = k + 2$.
Z Lematu 2. BSO można założyć, że optymalne kolorwanie K', będzie miało pokolorwane wszystkie liście).
Dodanie pokolorowanych liści może zwiększyć liczbę pokolorowanych wierzchołków w dowolnej ścieżce o co najwyżej 2. Jedyne nowo powstałe ścieżki to takie z conajmniej jednym <u>nowym</u> wierzchołkiem na końcu. Więc każda będzie miała co najwyżej $k + 2 = k'$ (nowe wierzchołki są kolorowane).

Nie możemy zwiększyć K' kolorując <u>nowe</u> wierzchołki (kolorujemy już wszystkie), więc jeżeli K' nie jest optymalne, oznacza to, że możemy jeszcze pokolorwać wierzchołek w $G' - liscie = G$, ale to przeczy optymalności kolorwania K w G z k. □

Czyli chcemy wziąć takie poddrzewo $\widetilde{G} \subset G$, że dodając $k \div 2$ razy (tyle możemy dodać startując od $k = 0$ lub $k = 1$) synów (w G) liści obecnego poddrzewa, otrzymamy G i kolorwać za każdym razem liście obecnego poddrzewa.

Zauważmy, że możemy to robić od końca (i tak pokolorujemy wszystkie potrzebne wierzchołki), czyli z G usunąć liście, pokolorwać je itd. Na końcu jeszcze jak zostaje nam $k = 1$ kolorujemy dowolny wierzchołek z pozostałego $\widetilde{G}$ po *strzyżeniu topologicznym* G.

```
while k > 1 and G:
    koloruj_liscie(G)
    usun_liscie(G)
    k-=2
if k == 1 and G:
    koloruj_dowolny(G)
```

$\mathcal{O}(n)$

Zadanie 5

Dowód podzielimy da dwie części:

1. W każdym momencie działania algorytmu nie będzie cyklu
2. Dla każdej składowej będzie MST

Dowód.

Brak cyklu

Załóżmy niewprost, że podczas działania algorytmu, w jakiejś spójnej składowej, powstał cykl C. Oznacza to, że powstał on na skutek połączenia n (super) wierzchołków $v_0, v_1, \ldots v_n$ będącymi kolejnymi wierzchołkami w C. Niech $e_0, e_1, \ldots e_n$ będą kolejnymi krawędziami takimi, że e_k łączy v_k i $v_{k+1 \ mod \ n+1}$. Z tego jak działa algorytm wynika, że

$$Cost(e_0) < Cost(e_1) < \ldots < Cost(e_n) < Cost(e_0)$$

sprzeczność

Minimalność

Załóżmy, nie wprost, że w algorytmie dodajemy krawędź e łącząc super wierzchołki V, W i e psuje minimalność (już w wynikowym MST).
Wtedy jeżeli do rozwiązania optymalnego dodamy e otrzymamy cykl C. Na tym cyklu leży conajmniej jedna krawędź e' (różna od e) łącząca pewne wierzchołki z V i W (inaczej OPT nie byłby spójny). Rozpatrzmy przypadki:

- $Cost(e) < Cost(e')$
  Sprzeczność. OPT nie jest OPT-em.
- $Cost(e) > Cost(e')$
  Sprzeczność. Zgodnie z działaniem algorytmu powinniśmy wziąć krawędź e' jako minimalną między V i W,

□

Zadanie 6

Lemat 1. *C - dowolny cykl, e - dowolna krawędź w* C*. Jeżeli* $c(e) = \max_{\bar{e} \in C} c(\bar{e})$ *to e nie należy do żadnego MST.*

Dowód. Załóżmy, że tak nie jest. Usuńmy e z MST, nasze drzewo spinające zostało podzielone na dwie składowe T_1 i T_2. Skoro e leży na cyklu C to istnieje takie e', że łączy ono T_1, T_2. Sprzeczność z minimalnością MST. Możemy zbudować *lepsze* drzewo spinające używając e' zamiast e
bo $c(e) = \max_{\bar{e} \in C} c(\bar{e}) > c(e')$ □

Z lematu wystarczy, że sprawdzimy czy zadana krawędź jest krawędzią o maxymalnym koszcie w pewnym cyklu.

Możemy to zrobić w czasie $\mathcal{O}(n+m)$ idąc algorytmem BFS po grafie $G \setminus \{e\}$ od u ($e = \{u, v\}$) przy czym wchodzimy do jakiegoś wierzchołka tylko wtedy gdy krawędź do niego ma mniejszy koszt niż e. Jeżeli będziemy w stanie dojść do v to znaczy, że istniała ścieżka pomiędzy tymi wierzchołkami nie używająca e więc z e powstanie cykl. Dodatkowo, rozpatrywaliśmy tylko krawędzie o mniejszym koszcie więc e ma maksymalny koszt na tym cyklu, więc na pewno nie ma jej w żadnym MST.
Jeżeli nie znajdziemy ścieżki łączącej u z v idąc tym zmodyfikowanym BFS'em to znaczy, że nie ma cyklu, gdzie e byłaby maksymalną krawędzią.

Usuwając z grafu krawędzię będące maksymalnymi w swoich cyklach otrzymamy w końcu MST.

```
q = [u]
visited = [False] * n
while q:
    w = q.pop()
    if w == v:
        return False
    visited[w] = True
    for (wp,c) in G[w]:
        if !visited[wp] and c < e.c:
            visited[wp] = True
            q.add(wp)
return True
```

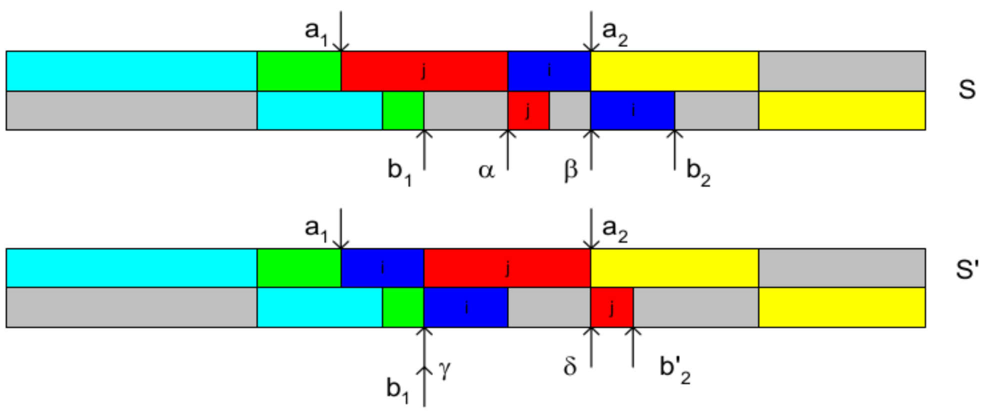
Zadanie 7

Podzielmy zbiór zadań na: $S_1 = \{i : A_i < B_i\}$ i $S_2 = \{i : A_i \geqslant B_i\}$.
Posortujmy S_1 rosnąco po A_i a S_2 malejąco o B_i. Odpowiedź $S_1 \cup S_2$.

Dowód. Weźmy dowolne rozwiązanie optymalne S jeżeli S nie jest *powyższej* postaci to musi być spełniony jeden z przypadków dla dwóch kolejnych zadań i, j w S:

1. $j \in S_2 \wedge i \in S_1$
2. $i, j \in S_1 \wedge A_j > A_i$
3. $i, j \in S_2 \wedge B_j < B_i$

Pokażemy, że w każdym z tych przypadków zmiana zadań i, j da nam rozwiązanie niegorsze niż S. Oznaczenia:



S' - S z zamienionymi zadaniami i, j
b_1 - zakończenie poprzedniego (przed i) zadania w S
b_1' - zakończenie poprzedniego (przed j) zadania w S'
b_2 - zakończenie obu zadań w S
b_2' - zakończenie obu zadań w S'

$\alpha = \max(b_1, (A_j + a_1))$ rozpoczęcie pierwszego zadania b w S
$\beta = \max(a_2, (B_j + \alpha))$ rozpoczęcie drugiego zadania b w S
$b_2 = B_i + \beta = \max(a_1 + A_j + A_i + B_i, B_j + B_i + b_1, B_j + B_i + A_j + a_1)$
$\gamma = \max(b_1, (A_i + a_1))$ rozpoczęcie pierwszego zadania b w S'
$\delta = \max(a_2, (B_i + \gamma))$ rozpoczęcie drugiego zadania b w S'
$b_2' = B_j + \delta = \max(a_1 + A_j + A_i + B_j, B_j + B_i + b_1, B_j + B_i + A_i + a_1)$

1. $j \in S_2 \wedge i \in S_1 \implies A_j \geqslant B_j \wedge A_i < B_i$:

$$\overbrace{a_1 + A_j + A_i + B_i}^{1.\quad w \quad b_2} \geqslant \underbrace{B_i + B_j + A_i + a_1}_{3.\quad w \quad b_2'}$$

$$\overbrace{B_j + B_i + A_j + a_1}^{3.\quad w \quad b_2} \geqslant \underbrace{a_1 + A_j + A_i + B_j}_{1.\quad w \quad b_2'}$$

2. $i, j \in S_1 \wedge A_j > A_i \implies A_j > A_i \wedge A_i < B_i$

$$\overbrace{B_j + B_i + A_j + a_1}^{3.\quad w \quad b_2} > \underbrace{B_i + B_j + A_i + a_1}_{3.\quad w \quad b_2'}$$

$$\overbrace{B_j + B_i + A_j + a_1}^{3.\quad w \quad b_2} > \underbrace{a_1 + A_j + A_i + B_j}_{1.\quad w \quad b_2'}$$

3. $i, j \in S_2 \wedge B_j < B_i \implies A_j \geqslant B_j \wedge B_j < B_i$

$$\overbrace{a_1 + A_j + A_i + B_i}^{1.\quad w \quad b_2} \geqslant \underbrace{B_i + B_j + A_i + a_1}_{3.\quad w \quad b_2'}$$

$$\overbrace{a_1 + A_j + A_i + B_i}^{1.\quad w \quad b_2} > \underbrace{a_1 + A_j + A_i + B_j}_{1.\quad w \quad b_2'}$$

W każdym przypadku mamy: $b_2 \geqslant b_2'$. Zmiana nie pogarsza rozwiązania. $\square$

źródło

Zadanie 8

Lemat 1. *Jeżeli strategi zachłanna nie zwraca OPT'a to najmniejszy kontrprzykład x jest w przedziale:*

$$b + 1 < x < a + b$$

Dowód.

$b + 1 < x$

Rozważmy przypadki:

1. $x < b$ dobieramy jak OPT, same a z 1 lub same 1.
2. $x = b$ (+1) dobieramy jak OPT, tylko b (z 1)

$x < a + b$

Weźmy dowolny $x \geqslant a + b$ i załóżmy, że $\forall_{y<x} OPT(y) = GRD(y)$:

1. b jest w OPT $OPT(x) = OPT(x - b) + 1 = GRD(x - b) + 1 = GRD(x)$
2. a jest w OPT

$$GRD(x) = GRD(x - b) + 1 = OPT(x - b) + 1 \leqslant OPT(x - b - a) + 2 = \\ GRD(x - b - a) + 2 = GRD(x - a) + 1 = GRD(x)$$

3. 1 jest w OPT

$$GRD(x) = GRD(x - b) + 1 = OPT(x - b) + 1 \leqslant OPT(x - b - 1) + 2 = \\ GRD(x - b - 1) + 2 = GRD(x - 1) + 1 = OPT(x - 1) + 1 = OPT(x) \leqslant GRD(x)$$

Z dowolności wyboru x pokazaliśmy, że jeżeli istnieje kontrprzykład to musi też istnieć kontrprzykład $< a + b$ $\square$

Twierdzenie 1. *Strategia zachłanna z nominałami* $\{1, a, b\}$ *nie zwraca OPT'a wtedy i tylko wtedy gdy:*

$$0 < r < a - q$$

gdzie: $b = qa + r$ *oraz* $r \in [0, a-1]$

Dowód.

$(\implies)$

Niech x będzie najmniejszym kontrprzykładem (zakładamy, że zachłan nie zwraca OPT'a). Z lematu 1. wiemy, że $x \in [b+2, a+b-1]$. BSO (z dokładnością do jedynek, gdzie różnią się rozwiązania) GRD $= (e, 0, 1)$ OPT $= (0, k, 0)$ oczywiście $e \in [2, a-1]$, dalej przyrównujemy rozkłady x:

$$x = b + e = k \cdot a$$

$$b = k \cdot a - e = \underbrace{(k-1)}_{q} \cdot a + \underbrace{(a-e)}_{r}$$

z powyższych rozważan:

$$r = a - e \geqslant a - (a-1) = 1 > 0$$

założyliśmy, że zachłan zwraca gorszy wynik więc $\underbrace{e+1}_{GRD} > \overbrace{k}^{}_{OPT} \implies e > k-1$:

$$a - q = a - (k-1) > a - e = r$$

$(\impliedby)$

Rozważmy $x = a + b - 1$, zachłan zwróci $(a-1, 0, 1)$, ale

$$x = a + \underbrace{qa + r}_{b} - 1 = (q+1) \cdot a + (r-1)$$

czyli $x = (r-1, q+1, 0)$, stąd OPT $\leqslant r - 1 + q + 1 = r + q$
Z założenia:

$$0 < r < a - q \implies r + q < a$$

wobec powyższego, zachłan zwróci rozwiązanie gorsze od OPT'a:

$$OPT \leqslant r + q < a = a - 1 + 1 = GRD$$

□

```
r = b%a
q = (b-r)/a
return not (0 < r and r < a-q)
```

Zadanie 9

```
q = priority_queue() # minimow
for w in W:
    q.push(Node(w,(NULL,NULL)))
while q.size() >= 2:
    u = q.pop()
    v = q.pop()
    q.push(Node(u.w+v.w,(u,v))) # (waga,(dzieci))
return q.pop() # root
```

Lemat 1. *Niech w, w' najmniejsze wagi z $\{w_1, \dots, w_n\}$. Istnieje optymalne drzewo T tż. liście o wagach w, w', mają wspólnego rodzica.*

Dowód. Oczywiście wierzchołek o maksymalnej głębokości ma *brata* w OPT T, w przeciwnym przypadku możemy go *przechnąć* w górę i tym zmniejszymy zewnętrzną długość drzewa.
Rozważmy drzewo T', z zamienionymi maksymalnie głębokimi u, v liśćmi z minimalnymi wagami $V_w, V_{w'}$.

$$
\begin{aligned}
EL(T) - EL(T') \geqslant c(V_w)d_T(V_w) + c(u)d_T(u) - c(V_w)d_T(u) - c(u)d_T(V_w) = \\
c(V_w) \cdot (d_T(V_w) - d_T(u)) + c(u) \cdot (d_T(u) - d_T(V_w)) = \\
\underbrace{(c(u) - c(V_w))}_{\geqslant 0} \cdot \underbrace{(d_T(u) - d_T(V_w))}_{\geqslant 0} \geqslant 0
\end{aligned}
$$

Analogicznie dla drugiej pary zmienianych ze sobą wierzchołków $v, V_{w'}$ $\square$

Dowód.

Poprawność

Baza

Dla $n \in \{1, 2\}$ OK

Krok

Załóżmy, że dla n algorytm zwraca optymalne drzewo T. Dla $n + 1$:
w, w', najmniejsze wagi z W, $|W| = n + 1$, $W' = W \setminus \{w, w'\} \cup \{w + w'\}$

P - drzewo zwrócone przez algorytm dla danych wejściowych W
P' - drzewo zwrócone przez algorytm dla danych wejściowych W' (z założenia indukcyjnego, $P' = \text{OPT}$)
Pokażemy, że P też jest optymalne:

Oznaczmy T jako drzewo optymalne dla wejścia W, z lematu 1. BSO zakładamy, że $V_w, V_{w'}$ są najniższymi liśćmi i *braćmi*. Usuńmy zatem $V_w, V_{w'}$ z T i za wagę ich ojca ustawmy $w + w'$. Otrzymamy drzewo T' z wagami liści równymi zbiorowi W'. Oczywiście $EL(T') \geqslant EL(P')$ (P' to OPT).
Pamiętamy jeszcze $d(V_w) = d(V_{w'})$ bo to *bracia*, zatem:

$$EL(T) = EL(T') - (d(V_w) - 1)(c(V_w) + c(V_{w'})) + d(V_w)(c(V_w) + c(V_{w'}))$$

$$EL(P) = EL(P') - (d(V_w) - 1)(c(V_w) + c(V_{w'})) + d(V_w)(c(V_w) + c(V_{w'}))$$

Dalej:

$$EL(T') - EL(T) = EL(P') - EL(P)$$
$$EL(T') - EL(P') = EL(T) - EL(P)$$

wnioskujemy:

$$EL(T) \geqslant EL(P)$$

Stąd P - OPT bo T było OPT'em. □