

ZADANIE 2.

Exercise 1. Udowodnij, że algorytm Kruskala znajduje minimalne drzewa spinające poprzez przyrównanie tych drzew do drzew optymalnych

Po pierwsze, rozpiszmy co robi ten algorytm:

```
E' <- [] pusty zbior
C <- E

while |E'| < n+1
  e <- min(C)
  if (E' + e nie ma cyklu)
    E' <- E' + e
  C <- C - e
```

Niech T będzie drzewem minimalnym, a E' będzie wynikiem algorytmu. Chcę pokazać, że E' jest minimalnym spanning tree. To że jest wogóle tree, to widać, elo.

Co, jeśli istnieje $e \in E' \setminus T$? Wtedy $e \cup T$ będzie miało cykle, bo T zachaczało o wszystkie krawędzie, jara jara jara. Czyli w ten sposób tworzymy sobie pewien cykl. Super. To teraz w T musiałam mieć jakieś inne przejście w tym cyklu, niech to będzie f . No ale miara e była mniejsza, bo inaczej to był f próbowała dołączyć przed e do E' i by nie stwierdziło, że się zacykli. Czyli graf $E' \setminus e \cup f$ będzie miał troszkę większą sumę niż E' .

Robiąc tak indukcyjnie aż nam się wszystkie wierzchołki pokryją, dojdziemy do grafu pokrywającego się z T , ale o większej mocy niż E' . Czyli to nie mogło być tak, że takie usuwanie krawędzi faktycznie zwiększało sumę, tylko to wszystko musiało zostawać tak samo, więc suma z E' to to samo co te sumy w międzyczasie, a one z kolei równały się sumie T .

ZADANIE 3.

Exercise 2. Danych jest n odcinków $I_j = [p_j, k_j]$ leżących na osi OX , $j = 1, \dots, n$. Ułóż algorytm znajdujący zbiór $S \subseteq \{I_1, \dots, I_n\}$ nieprzecinających się przecinków, o największej mocy.

Mam listę P i K , odpowiednio początków i końców tych pyśków. Może jakoś od razu sobie założę, że mam dwójkę? Indeks i wartość początku/końca? wtedy mogę łatwo wiedzieć gdzie był czyj koniec?

Dobra, to teraz biorę pierwszego pyśka z końców i wkładam go sobie do S . W ten sposób gwarantuję sobie, że śmignie, bo nawet jeśli jakiś z dalszym końcem da mi ten sam wynik, to nie wiem startowo że tak będzie, więc po prostu wybieram to, co niszczy mi najmniej innych wyborów.

W następnej kolejności idę dalej przez K aż znajdę pierwszy koniec, który ma początek ostro większy niż to co jako pierwsze wybrałam. Tutaj jest ta sama historia. No i tak dalej aż do końca listy K .

ZADANIE 4.

Exercise 3. Rozważ następującą wersję problemu wydawania reszty: dla danych liczb naturalnych a, b , ($a \leq b$) chcemy przedstawić ułamek $\frac{a}{b}$ jako sumę różnych ułamków o licznikach równych 1. Udowodnij, że algorytm zachłanny zawsze daje rozwiązanie. Czy zawsze jest to rozwiązanie optymalne (tj. o najmniejszej liczbie składników?)

Rozumiem, że algorytm zachłanny po prostu leci przez kolejne liczby naturalne i sprawdza, czy dodając je dostaję coś większego, czy się mieszczę? I jeszcze sprawdzam, czy jest sens iść dalej, to znaczy czy mam już dokładnie ten ułamek który chciałam.

Otóż nie daje najlepszego, bo mi podusia powiedziała, że istnieje

$$\frac{9}{20} = \frac{1}{3} + \frac{1}{9} + \frac{1}{180}$$

według zachłana, a można też zrobić

$$\frac{9}{20} = \frac{1}{4} + \frac{1}{5}$$

Niech k będzie najmniejszą taką liczbą naturalną, że

$$\frac{a}{b} - \frac{1}{k} = \frac{ak - b}{bk} > 0.$$

Chcę pokazać, że $a > ak - b$. Wtedy zakończenie algorytmu wynika z nieistnienia nieskończonego, malejącego ciągu liczb naturalnych.

$$a > ak - b$$

$$b > ak - a = a(k - 1)$$

$$\frac{b}{a} > k - 1$$

$$\frac{1}{k-1} > \frac{a}{b}$$

co jest prawdą, bo k było najmniejsze takie, że $\frac{a}{b} \geq \frac{1}{k}$

ZADANIE 5.

Exercise 4. *Ułóż algorytm, który dla danego n -wierzchołkowego drzewa i liczby k , pokoloruje jak najwięcej wierzchołków tak, by na każdej ścieżce prostej było nie więcej niż k pokolorowanych wierzchołków.*

Czy ja chcę pokolorować wszystkie wierzchołki, uciąć je i powtórzyć to samo dla $k - 2$ na tym nowym drzewie? A jeśli $k = 1$, to wtedy chyba losowy jeden mogę pomalować.

ZADANIE 6.

Exercise 5. *Ułóż algorytm, który dla danego spójnego grafu G oraz krawędzi e sprawdza w czasie $O(n + m)$, czy krawędź e należy do jakiegoś minimalnego drzewa spinającego grafu G . Możesz założyć, że wszystkie wagi krawędzi są różne.*

Exercise 9. Operacja $\text{swap}(i, j)$ na permutacji powoduje przestawienie elementów znajdujących się na pozycjach i oraz j . Koszt takiej operacji określamy na $|i - j|$. Kosztem ciągu operacji swap jest suma kosztów poszczególnych operacji.

U mnie indeks tablicy to aktualna liczba którą przestawiamy, a zawartość tablicy mówi gdzie ona stoi

```
P <- permutacja pi
S <- permutacja sigma
left <- [] * n
right <- [] * n

op = []

for i in {0, 1, ..., n-1}:
    if P[i] > S[i]: # jesli element stoi bardziej w prawo w pi
        left += [i]
    if P[i] < S[i]: # element w pi jest bardziej na lewo
        right += [i]

for i in right:
    for j in left:
        # zamieniam to co chce w lewo z tym co chce w prawo
        pom = P[i]
```

```

P[i] = P[j]
P[j] = pom

if P[j] == S[j]:
    left.remove(j)

if P[i] == S[i]:
    right.remove(i)
    break

```

Exercise 2. Z LISTY 3

To z prostymi

```
proste <- lista par (a, b)
```

```

merge_sort (L):
    if |L| == 1:
        return L
    if |L| == 2:
        if L[0][0] == L[1][0]:
            return [[L[0][0], max(L[0][1], L[1][1])]
        else:
            if L[0][0] < L[1][0]:
                return [L[0], L[1]]
            else:
                return [L[1], L[0]]
    A = L[0::|L|/2]
    B = L[|L|/2 + 1, |L|-1]
    if (A[0][0] < B[0][0]):
        return A + B
    else:
        return B + A

```

```

P = merge_sort(proste)
n = |P|

```

```

RET = [P[0], P[1]]
for i in {2, ..., n-1}:
    j = i-1
    przec = (RET[j-1][1] - P[i][1]) / (P[i][0]-RET[j-1][0])
    while i >= 0 && (P[i][0] * przec + P[i][1]) >= (RET[j][0] * przec + RET[j][1]):
        RET.pop_back()
        przec = (RET[j-1][1] - P[i][1]) / (P[i][0]-RET[j-1][0])
        j = j-1
    RET += P[i]

```