MDL 10 30.11

Dominik Szczepaniak

December 21, 2023

1 Zadanie 1

Załóżmy, że istnieją dwa drzewa rozpinające S1, S2 oraz oba S1 oraz S2 są najmniejszymi drzewami rozpinającymi, które nie są tymi samymi drzewami.

W takim razie niech zbiorem wierzchołkow i krawędzi S1 będzie V1 oraz E1, a S2 będzie V2 i E2. Wiemy, że V1 = V2 oraz E1 != E2.

W takim razie weźmy krawędź o najmniejszej wadze w całym drzewie, niech będzie to e1. Bez straty ogólności niech występuje ona w S1. Wtedy S2 z e1 tworzy cykl i jedna z krawędzi tego cyklu, niech to będzie e2, nie jest w S1. Skoro e2 != e1 i znajduje się w S2 oraz waga e2 > waga e1.

No ale wtedy S = S2 + e1 / e2 jest drzewem rozpinającym i ma mniejszą wagę niż S2, więc S2 nie jest najmniejszym drzewem rozpinającym.

albo

Dowód przez sprzeczność:

Załóżmy, że graf posiada dwie minimalne drzewa rozpinające (MST) oznaczone jako MST1 i MST2. Niech E będzie zbiorem krawędzi obecnych w MST2, ale nieobecnych w MST1.

Rozważmy MST1. Jeśli to jest minimalne drzewo rozpinające, dodanie krawędzi do niego powinno stworzyć cykl. Rozważmy dodanie krawędzi 'e' z zbioru E. To spowoduje utworzenie cyklu. W związku z tym, nowe drzewo (oznaczone jako T) jest tylko 1 krawędzią od bycia minimalnym drzewem rozpinającym. Aby uzyskać MINIMALNE drzewo rozpinające, trzeba usunąć najdroższą krawędź w cyklu. Ponieważ wszystkie krawędzie mają różne wagi, najdroższa krawędź będzie jedyną swojego rodzaju. Jeśli 'e' jest najdroższą krawędzią, to nie otrzymujemy wielu minimalnych drzew rozpinających. Jeśli 'e' nie jest najdroższą krawędzią, to MST1 nie było MINIMALNYM drzewem rozpinającym.

2 Zadanie 2

Załóżmy przez sprzeczność, że najcięższa krawędź e należy do MST T1. Wtedy usunięcie e rozbije T1 na dwa drzewa, ale możemy je połączyć dowolną inną krawędzią która należała do cyklu, a ta ma na pewno mniejszą wagę, więc mamy mniejszą wagę drzewa rozpinającego, co jest sprzeczne z tym, że T1 jest MST.

Proof: Assume the contrary, i.e. that e belongs to an MST T1. Then deleting e will break T1 into two subtrees with the two ends of e in different subtrees. The remainder of C reconnects the subtrees, hence there is an edge f of C with ends in different subtrees, i.e., it reconnects the subtrees into a tree T2 with weight less than that of T1, because the weight of f is less than the weight of e.

3 Zadanie 3

tak

zalozmy ze ten algorytm znajduje jakies drzewo i nie jest one mst. oznaczmy przez M MST na tym grafie i przez T drzewo ktore znalezlismy. jesli T nie jest MST to istnieje jakis wierzchołek v, który jest połączony krawędzią o mniejszej wadze w M niż w T. No ale zauważmy, że jest to niemożliwe, ponieważ w tym algorytmie usuwamy wszystkie takie krawędzie, które są większe a nie rozspajają grafu, a wiemy, że bez tej krawędzi graf nie jest rozspojony, ponieważ M jest MST i nie ma tej krawędzi.

4 Zadanie 4

Dowód indukcyjny:

Co iterację drzewo T jest jakimś podgrafem MST M, dla ilości wierzchołków = 1 to jest prawda, bo nie zawiera żadnych krawędzi.

Załóżmy teraz indukcyjnie, że dla n wierzchołków T jest podgrafem MST M i algorytm Prima wybiera krawędź e do dodania do T. Jeśli $e \in M$ to skoro T jest podgrafem M to po dodaniu krawędzi e, T jest dalej podgrafem M.

Załóżmy więc, że e nie należy do M. Dodajmy więc e do M i tworzymy jakiś cykl. Skoro e ma jeden punkt końcowy w T i drugiego końca nie mamy (bo dodajemy tą krawędź algorytmem Prima) to musi być jakaś inna krawędź

e' w tym cyklu, która ma tylko jeden punkt końcowy w T (bo drzewo rozpinające jeszcze nie jest skończone). Więc algorytm Prima mógł dodać e', ale zamiast tego wybrał e, więc w(e) <= w(e'). Więc jeśli dodamy e do M i usuniemy krawędź e' to dostajemy nowe drzewo rozpinające M', gdzie w(M') <= w(M) i zawiera T z krawędzią e, co dalej podtrzymuje indukcję, bo T jest podgrafem jakiegoś MST.

(Zauważmy, że w(e') = w(e), bo w przeciwnym przypadku w(M') < w(M), więc M nie byłoby M)

5 Zadanie 5

Dopóki T nie jest drzewem rozpinającym wykonaj następujące:
dla każdej spojnej składowej Ci grafu T wykonaj następujące:
spośród krawędzi o jednym wierzchołku w Ci a drugim poza
wybierz tę o najmniejszej wadze i oznacz ją jako e(Ci)
dodaj wszystkie krawędzie e(Ci) do T

Załóżmy nie wprost, że powstaje cykl w jakimś kroku. Załóżmy, że mamy wierzchołki w1, w2, w3 które nie należą do wspólnych spójnych składowych. Aby powstał cykl to jakiegoś z tych wierzchołków muszą się łączyć dwa pozostałe oraz jednocześnie do siebie samych. No ale żeby tak się stało, to każdy z tych wierzchołków musi wybrać inną krawędź z tej trójki. Tj. np. w1 wybiera (w1, w2), w2 wybiera (w2, w3), a w3 wybiera (w3, w1). No ale skoro w1 wybrał (w1, w2), a w2 już nie wybrał tej krawędzi, to waga ((w1, w2)) <= waga ((w2, w3)). Analogicznie waga((w2, w3)) <= waga((w3, w1)) oraz waga((w3, w1)) <= waga((w1, w2)). Otrzymujemy z tego, że waga((w1, w2)) <= waga((w2, w3)) <= waga((w3, w1)) <= waga((w1, w2)), czyli wszystkie te krawędzie muszą mieć równą wagę, co jest niemożliwe z założenia zadania.

6 Zadanie 7

7 Zadanie 8

8 Zadanie 10

9 Zadanie 11

1. ukorzenmy drzewo w node 1

mamy dwa przypadki, albo jest jakas krawedz (1, x) ktora jest w maksymalnym matchingu albo nie ma takiej krawedzi.

jeśli nie ma takiej krawedzi to mozemy obliczyc maksymalny matching rekurencyjnie dla kazdego podrzewa dla 1

jesli jest taka krawdz to zalozmy ze jest krawedz (1, u). wtedy mozemy obliczyc rekurencyjnie max matching dla dzieci u oraz dzieci 1

oznaczmy przez dp[1][0] max matching jesli nie ma krawedzi (1, x) i dp[1][1] max matching jesli jest ta krawedz

wtedy nasza odpowiedz to max(dp[1][0], dp[1][1])

No ale musimy oczywiscie rekurencyjnie to policzyc.

dp[1][0] = $\sum_{w \in children(1)} max(dp[w][0], dp[w][1])$

(Po prostu liczymy maksymalne matchingi dla dzieci)

jesli istnieje jakas krawedz (1, u)

to naszym wynikiem jest oczywiscie wynik dla kazdego innego dziecka 1 oprocz u + dp[u][0] (czyli wynik jesli nie ma z u zadnej krawedzi) + 1 (bo mamy ta jedna krawedz)

dp[1][1] = $1 + \max(\sum_{w \in children(1)}(\sum_{u \in children(1), u!=w} dp[u][0]) + max(dp[w][0], dp[w][1]))$
// wybieramy w ktory wierzcholek

ale to jest wolne, przyspieszymy to sumami prefiksowymi

niech c1, c2, ..., ci, ..., cn to beda dzieci jakiegos wierzcholka v

wtedy wynik dp[v][1] = iteracja po dzieciach gdzie dla kazdego dziecka wynik: prefix[ci-1] + dp[ci][0] + prefix[cn] - prefix[ci] + 1

O(N)

algos:

```
dp[n+1][2] = 0;
def max_matching(start, parent):
    prefix[n+1], sufix[n+1] = 0
```

```
    dp[start][0] = dp[start][1] = 0
    leaf = True
    for child in adj[start]:
        if(child != parent):
            leaf = 0
            solve(child, start)
    if(leaf) return 0
    for child in adj[start]:
        if(child != parent):
            prefix.append(max(dp[child][0], dp[child][1]))
            suffix.append(max(dp[child][0], dp[child][1]))
    for i in range(1, len(prefix)+1):
        prefix[i] = prefix[i-1]
    for i in range(len(suffix)-2, -1, -1):
        suffix[i] = suffix[i+1]
    dp[start][0] = suffix[0]
    c_no = 0
    for child in adj[start]:
        if(child == parent): continue
        left = (c_no == 0)? 0 : prefix[c_no-1]
        right = (c_no == len(suffix)-1)? 0 : suffix[c_no+1]
        dp[start][1] = max(1+left + right + dp[child][0], dp[start][1])
        c_no+=1
max_matching(1, 1)
```