4. domácí úkol | Vilém Zouhar

1

1.1 Popis

Nejvíce naivní by bylo seřadit si pole, vyextrahovat z něj vrchol nejmenšího stupně, snížit sousedům stupeň a uspořádání opravit. Zároveň bychom si při extrakci pamatovali velikost nejvyššího stupně, což by pak bylo hledané k. Ve skutečnosti to zas tak naivní není, pokud řazení budeme dělat rozumně a dobře odargumentujeme složitost. Na řazení budeme používat radix sort s dvousměrným spojovým seznamem (v poli R). Také si budeme uchovávat pole P délky n, které nám vrcholy (čísla 0..n-1) přeloží na pointery do spojoveného seznamu.

1.2 Korektnost

Pokud máme vrcholy seřazené podle jejich stupně v každém podgrafu, ze kterého jsme sebrali ten nejmenší, pak maximum z extrakcí bude hledané k.

1.3 Pseudokód

Něco mezi C++ a Python ☺

```
def remove_twll(ptr):
    (^ptr->prev).next = ptr->next
    (^ptr->next).prev = ptr->prev
    # fails on empty list tho

def add_twll(ptr, base):
    ptr->prev = base->next
    base->next = ptr

def decrease(ptr, R):
    remove_twll(ptr)
    ptr->deg--;
    add_twl(ptr, R[ptr->deg])

def extract_min(P, R):
    for i in range(m):
        if R[i] != null:
            v = R[i]
            remove_twll(v)
            for vertex w in neighbours(v):
                decrease(P[w], R)
            kill_from_neighbours(v) # removes from graph
            return i
    return -1

def main():
    (P, R) = radix_sort_by_deg(verticies)
    k = 0
    for i in range(n):
        k = max(k, extract_min(P, R))
    output(k)
```

1.4 Složitost

Inicializace polí R, P nám zcela jistě trvá $O(n+m)$, stejně tak i radix sort. Ještě předtím ale musíme graf nějak projít, abychom vůbec stupně zjistili, což můžeme třeba BFS a bude nám to trvat stejně tak lineárně. Pak ovšem $n\times$ voláme $extract_min$, které uvnitř má smyčku délky m. Mohlo by se tedy zdát, že je z toho složitost $O(nm)$. To ale není pravda, neboť $n\times$ odstraňujeme vrchol nejmenšího stupně a snižujeme hrany sousedům. Tedy na každou takovou extrakci potřebujeme $min(d_v^{G'})$ kroků. Ale součet všech těchto extrakcí nemůže být víc než součet všech hran (neboť počítáme hrany nejmenšího vrcholu). Na každou extrakci ještě použijeme konstantu přepojování, ale to nás netrápí. Tedy celé toto procházení je $O(m+n)$ a v důsledku toho je celý algoritmus lineární.

2

2.1 Popis

Jelikož máme zajištěnou acykličnost grafu, můžeme jej topologicky seřadit. Pak budeme postupovat zespodu směrem nahoru dle principů dynamického programování. Do vrcholu topologicky nejnižšího uložíme, že aktuálně nejdelší cesta končící v nich je 0. To platí, neboť pokud v nich skončíme, dál už jít nemůžeme. Pak přejdeme do další vrstvy (proti směru hran), kde přičteme cenu hrany z předchozí vrstvy. Pokud do nějakého takto nového vrcholu vedou hrany z více vrcholů nulté vrstvy, vybereme tu nejdražší a uložíme ji do aktuálního vrcholu. Přejdeme do druhé vrstvy, kde pro každý vrchol opět vybere nejdražší součet (cena ve vrcholu + hrana). Takto pokračujeme dokud nedojdeme až k poslednímu vrcholu. Maximum z hodnot, které jsme si ukládali do vrcholu je cena nejdražší cesty.
Zároveň si můžeme ve vrcholech vést pointery, abychom zjistili, jakou cestu jsme vlastně našli, ale to je jen implementační detail.

2.2 Korektnost

Celá rozebrána v popisu. Postupujeme od konce dle dynamického programování a vybíráme nejlepší variantu.

2.3 Pseudokód

```
f = queue_from_topological_sort(G)
k = 0
while not f.empty():
        p = f.pop()
        if p.deg_minus = 0:
                p.anckvn = 0
        else
                best = 0
                for vertex v in outgoing neighbours(p):
                        edge e = edge(p, v)
                        best = max(best, e.cost+v.anckvn)
                        # pointering would go somewhere here
                p.anckvn = best
                k = max(k, p.anckvn)
output(k)
# here output is the price, not the actual path (implementation detail)
```

2.4 Složitost

Topologické setřízení trvá $O(n+m)$, pak následný průchod a celkové zpracování ne víc jak $O(n+m)$. Potřebujeme také převrátit hrany, ale to trvá stejně jako celý algoritmus lineárně.