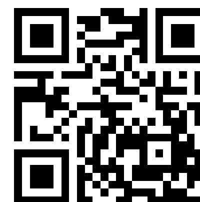
Korespondenční Seminář z Programování

34. ročník | **KSP** | **Listopad 2021**



Milí řešitelé, řešitelky a řešitelčata!

Dostává se k vám druhé číslo hlavní kategorie 34. ročníku KSP.

Opět se můžete těšit několik praktických i teoretických úloh, speciálně bychom vás chtěli upozornit na jednu **kompetitivní úlohu** s procházkou po Brně. V této sérii se také objeví pokračování Manimového seriálu, ale jeho zadání bude dostupné až po ukončení minulé série.

Odměny & na Matfyz bez přijímaček

Za úspěšné řešení KSP můžete být **přijati na MFF UK bez přijímacích zkoušek.** Úspěšným řešitelem se stává ten, kdo získá za celý ročník (této kategorie) alespoň 50 % bodů. Za letošní rok půjde získat maximálně 300 bodů, takže hranice pro úspěšné řešitele je 150. Maturanti pozor, pokud chcete prominutí využít letos, musíte to stihnout do konce čtvrté série, pátá už bude moc pozdě. Také každému řešiteli, který v tomto ročníku z každé série dostane alespoň 5 bodů, darujeme KSP propisku, blok, nálepku na notebook a možná i další překvapení.

Termín série: **20. prosince 2021 ve 32:00 (tedy další ráno v 8:00)**

Odevzdávání: Přes web na adrese *https://ksp.mff.cuni.cz/h/odevzdavatko/*

Značky úloh: Lehčí úloha (či její část) vhodná pro začátečníky; Praktická open-data úloha; Úloha, u které doporučujeme začíst se do kuchařky; Seriálová úloha

Odměna série: **Třem nejúspěšnějším řešitelům kompetitivní úlohy pošleme sladkou odměnu**.

Druhá série třicátého čtvrtého ročníku KSP

34-2-1 Skupinová jízdenka — 9 bodů

Alice a Bob plánují navštívit Carol. Alice bydlí v Adršpachu, Bob v Břeclavi a Carol v Chebu. Alice i Bob chtějí jet vlakem firmy HippoExpres, která právě vyhlásila slevu na skupinové jízdné. Proto se jim vyplácí sejít se v nějakém městě po cestě a odtamtud už jet společně.

Vymyslete algoritmus, který jim cestu naplánuje. Dostanete mapu železniční sítě: neorientovaný graf, jehož vrcholy jsou stanice a hrany tratě mezi nimi, ohodnocené vzdálenostmi v kilometrech. Dále dostanete cenu jednotlivého i skupinového jízdného v korunách za kilometr. Zjistěte, kde se mají Alice s Bobem sejít, aby je cesta stála co nejméně.

Toto je praktická open-data úloha. V odevzdávacím systému si necháte vygenerovat vstupy a odevzdáte příslušné výstupy. Záleží jen na vás, jak výstupy vyrobíte.

Formát vstupu: První řádek vstupu obsahuje 4 celá čísla: počet stanic n, počet tratí m, cenu jednotlivého jízdného v Kč za km, cenu skupinového jízdného v Kč za km. Následujících m řádků popisuje jednotlivé trati. Na každém z nich jsou 3 celá čísla: koncové stanice trati a délka trati v kilometrech. Stanice číslujeme od 1 do n, číslo 1 je Adršpach, 2 Břeclav a 3 je Cheb.

Formát výstupu: Vypište jediné číslo: nejmenší možnou cenu, kterou Alice s Bobem dohromady zaplatí za jízdenky.

Ukázkový vstup:

```
4 5 1 1
1 4 3
2 4 3
3 4 3
1 3 5
2 3 5
```

Ukázkový výstup:

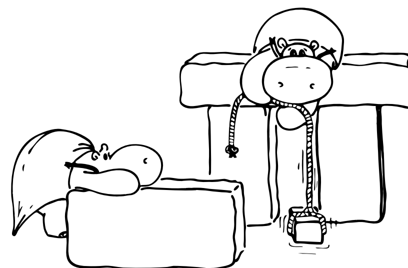
```
9
```

34-2-2 Lezení — 10 bodů

Sportovní lezení na olympijských hrách sestává ze tří disciplín: lezení na rychlost, bouldering a lezení na obtížnost. Počet trestných bodů, které lezec dostane za disciplínu, odpovídá jeho pořadí (počítáno od 1). Celkový počet trestných bodů je součin trestných bodů za jednotlivé disciplíny. Vítězem se stává lezec, který má nejmenší počet celkových trestných bodů, podobným způsobem se seřadí zbylí lezci.

Například pokud by lezec A skončil v disciplínách na 1., 4. a 7. místě a lezec B na 2., 13. a 1. místě, pak dostane lezec A celkem $1 \cdot 4 \cdot 7 = 28$ trestných bodů a umístí se tak až za lezcem B s $2 \cdot 13 \cdot 1 = 26$ trestnými body.

První dvě disciplíny již proběhly a je známé pořadí všech n lezců v obou dvou disciplínách. Náš nejlepší lezec je mistrem v lezení na obtížnost a věří si na první místo v této disciplíně. I tak ale netuší, jaké bude výsledné pořadí. Pomůžete mu zjistit, na jaké nejhorší příčce se může umístit, pokud by se mu skutečně podařilo skončit první v lezení na obtížnost, ale ostatní by se mohli v lezení na obtížnost seřadit libovolně?

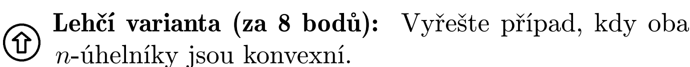


34-2-3 Na Hroších pláních — 12 bodů

Na Hroších pláních byla objevena obrovská naleziště čokolády! Všichni organizátoři KSPčka se tam hned rozjeli, aby si každý z nich vykolíkoval svůj claim – území, kde bude těžit.

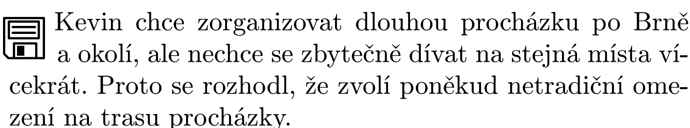
Každý claim má tvar n-úhelníku v rovině (ne nutně konvexního). V jeho vrcholech jsou zapíchané kolíky, po obvodu očíslované postupně od 1 do n. Každý organizátor má svou barvu kolíků, ale i tak je proklatě obtížné zjistit, jestli se nějaké dva claimy protínají. Vymyslete algoritmus, který to pozná.

Vymyslete algoritmus, který na vstupu dostane souřadnice kolíků určujících dva n-úhelníky claimů. Na výstup vypíše buď souřadnice nějakého bodu ležícího v průniku obou claimů, nebo sdělí, že claimy žádný společný bod nemají. Pokud se dva claimy jenom dotknou (vrcholem nebo hranou), za průnik to nepovažujeme.



Lehčí varianta (za 8 bodů): Vyřešte případ, kdy oba n-úhelníky jsou konvexní.

34-2-4 Brněnská procházka — 14 bodů



Kevin chce zorganizovat dlouhou procházku po Brně a okolí, ale nechce se zbytečně dívat na stejná místa vícekrát. Proto se rozhodl, že zvolí poněkud netradiční omezení na trasu procházky.

Brno je ohodnoceným neorientovaným grafem, kde vrcholy odpovídají křižovatkám a hrany ulicím. Ulice jsou celočíselně ohodnoceny jejich délkou v metrech. Vaším úkolem je najít co nejdelší cestu, ze které nezahlédnete žádnou jinou ulici dvakrát (tedy pokud vezmete libovolné dva vrcholy cesty, které po sobě těsně nenásledují, tak mezi nimi nebude v grafu existovat hrana). Začít i skončit můžete na libovolném vrcholu, jen od sebe také musí udržet vzdálenost dvou hran. Nejdelší cesta je ta, která má největší součet délek hran, na počtu hran Kevinovi nezáleží.

Toto je speciální **soutěžní úloha se statických vstupem**, kde všichni řešitelé dostanou stejný vstup a přes Odevzdávátko pak odevzdají nejlepší řešení, které se jim povede najít. Obodování úlohy provedeme až po konci série a to tak, že nejlepší řešení dostane plný počet bodů a ostatní řešení dostanou body odstupňovaně podle toho, jak byla dobrá. Zároveň slibujeme, že každé korektní řešení dostane alespoň jeden bod.

V průběhu série se můžete s ostatními porovnávat pomocí průběžné online výsledkovky. Upozorňujeme, že se v ní mohou vyskytnout i řešení od organizátorů.

Jde o těžkou úlohu, pro kterou neslibujeme existenci efektivního algoritmu na nalezení optimálního řešení. Pro inspiraci, jak k této úloze přistoupit, se můžete podívat na vzorová řešení minulé soutěžní úlohy 33-3-4. Vezměte ale na vědomí, že jde o výrazně odlišnou úlohu, a tak budete muset zvolit jiný přístup a rozhodně se vám vyplatí experimentovat.

Tuto úlohu také doporučujeme začít řešit včas, vstupní data jsou relativně velká a bude se vám velmi hodit čas na postupné vylepšování vašeho řešení.

Formát vstupu: Na prvním řádku dostanete počet vrcholů N a počet ulic M. Následuje N řádků s definicemi vrcholů ve formátu *v lat lon*, kde *v* je číslo vrcholu a *lat* a *lon* popisují GPS souřadnice – ty se mohou hodit pro vizualizace, ale nijak je není potřeba použít pro počítání délek. Vstup zakončuje M řádků s definicemi hran. Na každém řádku je popsána jedna hrana trojicí čísel oddělenou mezerami: v_1, v_2 a D. Taková hrana spojuje vrcholy v_1 a v_2 a má celočíselnou délku D metrů. Vrcholy jsou indexované od nuly.

Formát výstupu: Na první řádek vypište celkovou délku nalezené cesty v metrech. Poté vypište čísla vrcholů v pořadí, ve kterém je cesta navštěvuje, každé na samostatný řádek.

Odevzdávátko spočítá délku cesty a přidá odevzdané řešení do průběžné výsledkovky.[1] Upozorňujeme, že od každého řešitele bereme v potaz vždy jeho poslední odevzdané řešení, i kdyby si tím měl zhoršit skóre. Proto vám doporučujeme si svá řešení ukládat, abyste je případně mohli odevzdat znovu.

Zdroje: Graf k této úloze jsme získali zpracováním dat z projektu OpenStreetMap.

Ukázkový vstup:

```
6 7
0 49.16298 16.56755
1 49.16316 16.56767
2 49.16319 16.56854
3 49.16328 16.56858
4 49.16328 16.56951
5 49.17429 16.55060
0 2 15
0 4 10
1 2 5
1 3 20
2 5 1
3 4 5
3 5 2
```

Ukázkový výstup:

```
40
3
1
2
0
```

[1] https://ksp.mff.cuni.cz/h/ulohy/34/34-2-4/vysledky

34-2-X1 Převod čísel — 10 bodů

Kevin začal řešit úkol do informatiky. Chtěli po něm, aby převáděl čísla mezi desítkovou a dvojkovou soustavou. Ale tento úkol ho po chvíli přestal bavit, a tak začal převádět čísla i mezi jinými soustavami. Vaším úkolem bude mu s těmito převody pomoct. Vymyslete program, který bude převádět čísla ze soustavy o základu A do soustavy o základu B.

Na vstupu dostanete nejprve dvojici čísel A a B následovanou N čísly mezi 0 a $A-1$ značící vstupní číslo v soustavě o základu A. Výstupem vašeho programu bude posloupnost čísel mezi 0 a $B-1$ reprezentující totéž číslo, ale v soustavě o základu B.

Ve vašem programu ovšem nemůžete pracovat s libovolně velkými čísly v konstantním čase. Nemůžete tedy převést číslo na vstupu do jednoho čísla a to pak dále převádět do výstupní soustavy. Můžete ale předpokládat, že v konstantním čase umíte pracovat v čísly velikosti $\mathcal{O}((A+B)^k)$ pro nějakou pevnou konstantu k.

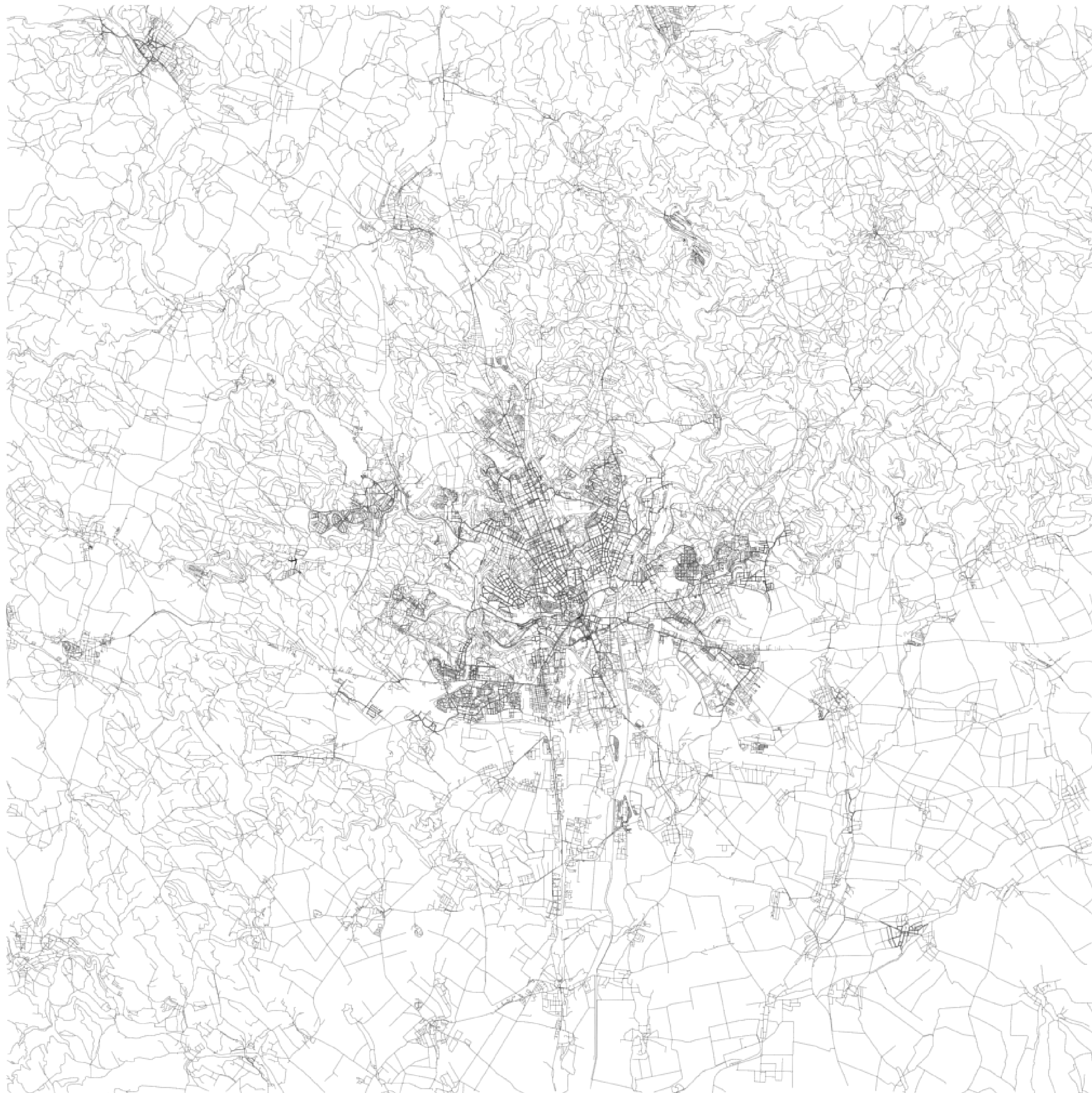
Nápověda: Existují algoritmy, které násobí n-ciferná čísla s časovou složitostí $\mathcal{O}(n^c)$ pro $1 \leq c < 2$. Najdete je například v Průvodci labyrintem algoritmů v kapitole Rozděl a panuj. Ve svém řešení můžete takový algoritmus používat jako podprogram, aniž byste ho museli sami popisovat.

34-2-S Manimujeme – updatery 15 bodů

Seriál je tento ročník zaměřen na generování animací pomocí Pythoní knihovny Manim a obsahuje tedy řadu animací, které se do formátu PDF nehodí. Proto je dostupný pouze na webu.[2]

Tomáš Sláma

Vizualizace dat pro úlohu 34-2-4 Brněnská procházka



[2] http://ksp.mff.cuni.cz/viz/34-2-S

Recepty z programátorské kuchařky: Haldy, heapsort a Dijkstrův algoritmus

Dnešní povídání o algoritmech a datových strukturách se bude zabývat jedním z nejznámějších algoritmů: Dijkstrovým algoritmem pro hledání nejkratších cest v grafech. K tomu (a nejenom k tomu) se nám bude hodit šikovná datová strukturka zvaná halda, tak si předvedeme nejdříve ji.

Halda

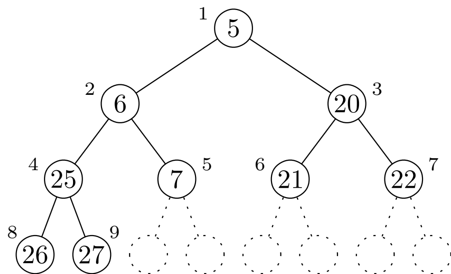
Halda je datová struktura pro uchovávání množiny čísel (či jakýchkoliv jiných prvků, na kterých máme definováno uspořádání, tj. umíme pro každou dvojici prvků říci, který z nich je menší). Tato datová struktura obvykle podporuje následující operace: přidání nového prvku, nalezení nejmenšího prvku a odebrání nejmenšího prvku. My si ukážeme jednoduchou implementaci haldy, která bude při uložení N prvků potřebovat čas $\mathcal{O}(\log N)$ na přidání či odebrání jednoho prvku a $\mathcal{O}(1)$ (tj. konstantní) na zjištění hodnoty nejmenšího prvku.

Naše implementace bude vypadat následovně: Pokud halda obsahuje N prvků, uložíme její prvky do pole na pozice 1 až N. Prvek na pozici k bude mít dva *následníky,* a to prvky na pozicích $2k$ a $2k+1$; samozřejmě, pokud je k velké, a tedy např. $2k+1 > N$, má takový prvek jen jednoho či dokonce žádného následníka. Naopak prvek na pozici $\lfloor k/2 \rfloor$ nazveme *předchůdcem* prvku na pozici k. Ti z vás, kteří znají binární stromy, v tomto jistě rozpoznali způsob, jak v poli uchovávat úplné binární stromy (následníci jsou synové a předchůdci otcové v obvyklé stromové terminologii, prvek č. 1 je kořen stromu).

Prvky haldy však v poli neuchováváme v úplně libovolném pořadí. Chceme, aby platilo, že každý prvek je menší nebo roven všem svým následníkům. Naše halda tedy může vypadat např. takto:

1	2	3	4	5	6	7	8	9
5	6	20	25	7	21	22	26	27

Tomu odpovídá tento strom:



Z toho, co jsme si právě popsali, je jasné, že nejmenší prvek je uložen na pozici s indexem 1, a tedy můžeme snadno v konstantním čase zjistit jeho hodnotu. Ještě prozradíme, jak lze prvky do haldy rychle přidávat a odebírat:

Jestliže halda obsahuje N prvků, pak nový prvek, říkejme mu třeba x, přidáme na konec pole, tj. na pozici s indexem $N+1$. Nyní x porovnáme s jeho předchůdcem. Pokud je jeho předchůdce menší, je vše v pořádku a jsme hotovi. V opačném případě x s jeho předchůdcem prohodíme. Tím jsme problém napravili, ale nyní může být x menší než jeho nový předchůdce. To lze napravit dalším prohozením a tak budeme pokračovat dále, než se buďto dostaneme do situace, kdy už je x větší nebo rovno svému předchůdci, nebo „vybublá“ až do kořene haldy, kde už žádného předchůdce nemá. Protože se v každém kroku pozice, na níž se prvek x právě nachází, zmenší alespoň na polovinu, provedeme dohromady nejvýše $\mathcal{O}(\log N)$ výměn, a tedy spotřebujeme čas $\mathcal{O}(\log N)$.

Odebírání nejmenšího prvku probíhá podobně: Prvek z poslední pozice (tj. z pozice N) přesuneme na pozici 1, tedy místo minima. Místo s předchůdci jej však porovnáme s jeho následníky, a v případě, že je větší než některý z jeho následníků, opět je prohodíme (pokud je větší než oba následníci, prohodíme ho s menším z nich). A protože se nám v každém kroku index „bublajícího“ prvku v poli alespoň zdvojnásobí, opět spotřebujeme čas $\mathcal{O}(\log N)$.

Jako cvičení si rozmyslete, že v čase $\mathcal{O}(\log N)$ lze z haldy smazat dokonce libovolný prvek, pokud si ovšem pamatujeme, kde se v haldě nachází. Také můžeme prvek ponechat a jen změnit jeho hodnotu.

Ještě si předvedeme program:

```
halda = []

def nejmensi():
  return halda[0]

def swap(a,b):
  (halda[a], halda[b]) = (halda[b], halda[a])

def vloz(prvek):
  # Vložíme prvek na konec
  halda.append(prvek)
  i = len(halda) - 1
  while (i > 0) and (halda[i//2] > halda[i]):
    swap(i, i//2)
    i = i//2

def smaz_nejmensi():
  N = len(halda)
  nejmensi = halda[0]
  halda[0] = halda[N - 1]
  # Odstraníme prvek z konce
  halda.pop()
  N -= 1
  i = 0
  while 2*i < N:
    j = i
    if halda[j] > halda[2*i]:
      j = 2*i
    if (2*i+1 < N) and (halda[j]>halda[2*i+1]):
      j = 2*i+1
    if i == j:
      break
    swap(i, j)
    i = j
  return nejmensi
```

HeapSort

Když už máme k dispozici haldu, můžeme pomocí ní například snadno třídit čísla. Máme-li N čísel, která chceme setřídit, vytvoříme si z nich nejprve haldu o N prvcích (například postupným vkládáním do prázdné haldy), načež z ní budeme postupně N-krát odebírat nejmenší prvek. Tím získáme prvky původního pole v rostoucím pořadí. Celkově provedeme N vložení, N nalezení minima a N smazání. To vše dohromady stihneme v čase $\mathcal{O}(N \log N)$.

Než si ukážeme program, přidáme ještě dva triky, které nám implementaci značně usnadní. Předně si vše uložíme do jed-

noho pole – to bude při plnění haldy obsahovat na svém začátku haldu a na konci zbytek vstupního pole, přitom zbytek pole se bude postupně zmenšovat a uvolňovat tak místo haldě; naopak v druhé polovině algoritmu budeme zmenšovat haldu a do volného prostoru ukládat setříděné prvky. K tomu se nám bude hodit získávat prvky v opačném pořadí, proto si upravíme haldu tak, aby udržovala nikoliv minimum, nýbrž maximum.

Druhý trik spočívá v tom, že nebudeme haldu vytvářet postupným vkládáním, nýbrž naopak zabubláváním prvků (podobným, jako děláme při mazání minima) od konce. Všimněte si, že takto také získáme správné nerovnosti mezi prvky a jejich následníky, a dokonce tak zvládneme celou haldu vytvořit v lineárním čase (proč to tak je, si zkuste dokázat sami, stačí si uvědomit, kolikrát zabubláváme které prvky). Zbytek třídění bohužel nadále zůstává $\mathcal{O}(N \log N)$.

Tomuto algoritmu se obvykle říká *HeapSort* (čili třídění haldou) a je jedním z mála známých rychlých třídících algoritmů, které nepotřebují pomocnou paměť.

```
# Zabublání prvku dolů:
#   N určuje část pole vyhrazenou haldě
#   i je index zabublávaného prvku
def bublej(pole, N, i):
  while 2*i < N:
    j = 2*i
    if (j+1 < N) and (pole[j+1] > pole[j]):
      j = j+1
    if pole[i] >= pole[j]:
      break
    (pole[i], pole[j]) = (pole[j], pole[i])
    i = j

def heapsort(pole):
  N = len(pole)
  for i in range(N//2, -1, -1):
    bublej(pole, N, i)
  # Výběr maxima a jeho přesun nakonec
  for i in range(N - 1, 0, -1):
    (pole[0], pole[i]) = (pole[i], pole[0])
    # Dál už bubláme jen na zbytku pole
    bublej(pole, i, 0)
  return pole
```

Dijkstrův algoritmus

Nyní již konečně k slíbenému *Dijkstrovu algoritmu.* Tento algoritmus dostane orientovaný graf s hranami ohodnocenými nezápornými čísly (viz kuchařka o grafech)[3] a nalezne v něm nejkratší cestu mezi dvěma zadanými vrcholy. Ve skutečnosti tento algoritmus dělá o malinko více: Najde totiž nejkratší cesty z jednoho zadaného vrcholu do všech ostatních.

Nechť v_0 je vrchol grafu, ze kterého chceme určit délky nejkratších cest. Budeme si udržovat pole délek zatím nalezených cest z vrcholu v_0 do všech ostatních vrcholů grafu. Navíc u některých vrcholů budeme mít poznamenáno, že cesta nalezená do nich je už ta nejkratší možná. Takovým vrcholům budeme říkat *definitivní.* Na začátku inicializujeme v poli všechny hodnoty na ∞ kromě hodnoty odpovídající vrcholu v_0, kterou inicializujeme na 0 (délka nejkratší cesty z v_0 do v_0 je 0). V každém *kroku* algoritmu pak provedeme následující: Vybereme vrchol w, který ještě není definitivní, a mezi všemi takovými vrcholy je délka zatím nalezené cesty do něj nejkratší možná. Vrchol w prohlásíme za definitivní. Dále otestujeme, zda pro nějaký vrchol v cesta z vrcholu v_0 do w a pak po hraně z w do v není kratší, než zatím nalezená cesta z v_0 do v, a je-li tomu tak, upravíme délku zatím nalezené cesty do v. Toto provedeme pro všechny takové vrcholy v. Celý algoritmus skončí, pokud jsou už všechny vrcholy definitivní nebo všechny vrcholy, které nejsou definitivní, mají délku cesty rovnou ∞ (v takovém případě se graf skládá z více nesouvislých částí).

Předtím, než dokážeme, že právě představený algoritmus opravdu nalezne délky nejkratších cest z vrcholu v_0, se zamysleme nad jeho časovou složitostí.

Pro každý z N vrcholů si délku dosud nalezené cesty uchováme v poli. Celý algoritmus provede nejvýše N kroků, protože v každém kroku nám přibyde jeden definitivní vrchol. Ten vybíráme jako minimum z délky aktuální cesty přes všechny dosud nedefinitivní vrcholy, kterých je $\mathcal{O}(N)$. V každém kroku musíme zkontrolovat tolik vrcholů v, kolik hran vede z vrcholu w. Počet takových změn pro všechny kroky dohromady je pak nejvýše $\mathcal{O}(M)$, kde M je počet hran vstupního grafu. Z toho vyjde časová složitost $\mathcal{O}(N^2 + M)$, čili $\mathcal{O}(N^2)$, jelikož M je nejvýše N^2. Tuto implementaci Dijkstrova algoritmu najdete na konci naší kuchařky.

K uchovávání délek dosud nalezených nejkratších cest můžeme ovšem použít haldu. Ta bude na začátku obsahovat N prvků a v každém kroku se počet jejích prvků sníží o jeden: Nalezneme a smažeme nejmenší prvek, to zvládneme v čase $\mathcal{O}(\log N)$, a případně upravíme délky nejkratších cest do sousedů právě zpracovávaného vrcholu. To pro každou hranu trvá rovněž $\mathcal{O}(\log N)$, celkově za všechny hrany tedy $\mathcal{O}(M \log N)$. Z toho vyjde celková časová složitost algoritmu $\mathcal{O}((N+M) \log N)$, a to je pro „řídké“ grafy (tedy grafy s $M \ll N^2$) výrazně lepší.

Vraťme se nyní k důkazu správnosti Dijkstrova algoritmu. Ukážeme, že po každém kroku algoritmu platí následující tvrzení: Nechť A je množina definitivních vrcholů. Pak délka dosud nalezené cesty z v_0 do v (v je libovolný vrchol grafu) je délka nejkratší cesty $v_0 v_1 \ldots v_k v$ takové, že všechny vrcholy $v_0, v_1, \ldots, v_k$ jsou v množině A. Tvrzení dokážeme indukcí dle počtu kroků algoritmu, které již proběhly. Tvrzení zřejmě platí před a po prvním kroku algoritmu. Nechť w je vrchol, který byl v předchozím kroku prohlášen za definitivní. Uvažme nejprve nějaký vrchol v, který je definitivní. Pokud $v = w$, tvrzení je triviální. V opačném případě ukážeme, že existuje nejkratší cesta z v_0 do v přes vrcholy z A, která nepoužívá vrchol w. Označme D délku cesty z v_0 do v přes vrcholy A bez vrcholu w. Protože v každém kroku vybíráme vrchol s nejmenším ohodnocením a ohodnocení vybraných vrcholů v jednotlivých krocích tvoří neklesající posloupnost (váhy hran jsou nezáporné!), tak délka cesty z v_0 do w přes vrcholy z A je alespoň D. Ale potom délka libovolné cesty z v_0 do v přes w používající vrcholy z A je alespoň D. Z volby D pak víme, že existuje nejkratší cesta z v_0 do v přes vrcholy z A, která nepoužívá vrchol w.

Nyní uvažme takový vrchol v, který není definitivní. Nechť $v_0 v_1 \ldots v_k v$ je nejkratší cesta z v_0 do v taková, že všechny vrcholy $v_0, v_1, \ldots, v_k$ jsou v množině A. Pokud $v_k = w$, pak jsme ohodnocení v změnili na délku této cesty v právě proběhlém kroku. Pokud $v_k \neq w$, pak $v_0 v_1, \ldots, v_k$ je nejkratší

[3] http://ksp.mff.cuni.cz/viz/kucharky/grafy

cesta z v_0 do v_k přes vrcholy z množiny A, a tedy můžeme předpokládat, že žádný z vrcholů $v_1, \ldots, v_k$ není w (podle toho, co jsme si rozmysleli na konci minulého odstavce). Potom se ale délka cesty do v rovnala správné hodnotě už před právě proběhlým krokem.

Vzhledem k tomu, že po posledním kroku množina A obsahuje právě ty vrcholy, do kterých existuje cesta z vrcholu v_0, dokázali jsme, že náš algoritmus funguje správně.

Na závěr ještě poznamenejme, že Dijkstrův algoritmus je možné snadno upravit tak, aby nám kromě určení délky nejkratší cesty i takovou cestu našel: U každého vrcholu si v okamžiku, kdy mu měníme ohodnocení, poznamenáme, ze kterého vrcholu do něj přicházíme. Nejkratší cestu do nějakého vrcholu pak zrekonstruujeme tak, že u posledního vrcholu této cesty zjistíme, který vrchol je předposlední, u předposledního, který je předpředposlední, atd.

Poznámka pro zvídavé: existují i jiné druhy hald, například k-regulární haldy, v nichž má každý prvek k následníků (rozmyslete si, jaká je v takové haldě časová složitost operací a jak nastavit k v závislosti na M a N, aby byl Dijkstrův algoritmus co nejrychlejší), nebo tzv. Fibonacciho halda, která dokáže upravit hodnotu prvku v konstantním čase. S tou pak umíme hledat nejkratší cesty v čase $\mathcal{O}(M + N \log N)$.

Dnešní menu Vám servírovali

Dan Kráľ, Martin Mareš a Petr Škoda

Implementace Dijkstrova algoritmu s haldou

```
# Struktura pro vrchol jako dvojici (index, vzdálenost)
# (definuje vlastní porovnávání, aby šlo použít haldu výše)
class vrchol():
  def __init__(self, index, vzdalenost):
    self.index = index
    self.vzdalenost = vzdalenost
  # Předefinování operátoru >
  def __gt__(self, obj):
    return self.vzdalenost.__gt__(obj.vzdalenost)

halda = []
def dijkstra(N, cesty, start):
  final = [False] * N
  vzdalenost = [None] * N
  # Inicializace startu:
  vloz(vrchol(start, 0))
  vzdalenost[start] = 0
  while halda:
    # Vytáhneme z haldy nejmenší nezpracované místo
    v = smaz_nejmensi()
    if final[v.index]:
      continue
    # Označíme místo za použité
    final[v.index] = True
    # Projdeme všechny sousedy
    for (i, delka) in cesty[v.index]:
      if vzdalenost[i] is None or v.vzdalenost + delka < vzdalenost[i]:
        vzdalenost[i] = v.vzdalenost + delka
        vloz(vrchol(i, vzdalenost[i]))
  return vzdalenost
```

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.

Webové stránky: *https://ksp.mff.cuni.cz/*

E-mail: *ksp@mff.cuni.cz*

Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*