Cvičenie 1

Príklad 1.1.

Máme utriedené pole n čísel. Nájdite takú dvojicu týchto čísel, že ich súčet je aspoň k a zároveň je ku k čo najbližšie.

Náčrt riešenia 1.1.

Pokúsme sa nájsť riešenie lepšie ako $O(n^2)$. Ak si zafixujem jedno konkrétne číslo x, tak sa nám problém zmení – hľadáme najmenšie číslo väčšie alebo rovné ako $k - x$. Čo v utriedenom poli môžeme robiť pomocou binárneho vyhľadávania. Toto teda zopakujem pre každé možné x a vypíšeme najlepšiu nájdenú dvojicu, čím dostaneme riešenie s časovou zložitosťou $O(n \log n)$.

Môžeme si ale všimnúť nasledovnú vec. Ak skúšame x postupne od najmenších po najväčšie, tak sa hodnota $k - x$ bude postupne zmenšovať. Nemusíme preto binárne vyhľadávať v celom poli, stačí sa pozrieť kde sme našli predchádzajúce $k - x$ a hľadať už len v menších číslach.

Toto pozorovanie potom vedie k ešte jednoduchšiemu riešeniu. Budeme si pamätať dva indexy *zac* a *kon*, prvý nastavíme na začiatok a druhý na koniec. Oba ukazujú na nejaké dva prvky. Ak je súčet týchto dvoch prvkov väčší ako k, musíme tento súčet zmenšiť, teda zmenšíme hodnotu *kon* o 1. Ak je súčet menší, tak o jedna zväčšíme premennú *zac*. Takto postupne prejdeme viacero dvojíc, medzi ktorými je určite aj výsledok a vyberieme z nich tú, ktorá dávala najlepší výsledok. Naviac, každú premennú zmeníme najviac n krát, preto dostaneme riešenie so zložitosťou $O(n)$.

Príklad 1.2.

V rovine máme n bodov, ktoré sú na súradniciach (x_i, y_i). V rovine chceme nakresliť vodorovnú priamku. Navrhnite algoritmus, ktorý určí y-ovú súradnicu tejto priamky tak, aby sa minimalizoval súčet vzdialeností bodov od priamky.

Náčrt riešenia 1.2.

Je jasné, že vzdialenosť bodu i od priamky je $y - y_i$. Potrebujeme teda minimalizovať sumu takýchto rozdielov. Zvoľme si nejakú súradnicu y (BUNV na súradnici y neleží žiaden z bodov), na ktorej bude ležať naša priamka a súčet vzdialeností označme s. Ako sa zmení hodnota s ak priamku umiestnime na pozíciu $y + 1$?

Priamka sa vzdiali od všetkých bodov, ktoré ležali pod ňou, od každého z nich o 1. Ale o 1 sa zase priblíži ku všetkým bodov, ktoré ležali nad ňou. Takéto posunutie preto bude výhodné práve vtedy ak je nad priamkou leží viacej bodov, ako leží pod ňou. V takom prípade sa nám totiž hodnota s zmenší.

No a táto myšlienka platí aj všeobecne. Vždy keď nebude nad aj pod priamkou rovnaký počet bodov, bude existovať smer, v ktorom sa nám túto priamku oplatí posunúť. Optimálne umiestnenie bude také, pri ktorom bude nad aj pod priamkou rovnaký počet bodov. Hľadáme teda medián z y-ových súradníc zadaných bodov.

Príklad 1.3.

Máme pole A kladných aj záporných čísel dĺžky n. Nájdite súvislý podúsek, ktorý má najväčší súčet.

Náčrt riešenia 1.3.

Pri tejto úlohe využijeme prefixové súčty. Vytvoríme si pole $P[]$, kde na pozícii $P[i]$ bude súčet prvých i prvkov poľa $A[]$ (teda tie na pozíciách 0 až $i - 1$). Hodnota $P[0]$ bude rovná 0.

Uvedomme si, že ak chceme zistiť súčet nejakého súvislého úseku, ktorý začína na indexe zac a končí na indexe kon, vypočítame ho ako rozdiel $P[kon+1] - P[zac]$. Toto pole naviac vieme ľahko spočítať v lineárnom čase, lebo $P[i] = P[i-1] + A[i-1]$.

Aký najväčší súčet môže mať podúsek, ktorý má koniec na pozícii i? No predsa $P[i+1]-P[j]$, kde $P[j]$ je najmenšia predchádzajúca hodnota. Môžeme teda postupne ísť poľom $P[]$, pamätať si najmenšiu hodnotu, ktorú sme zatiaľ videli, počítať súčet úseku končiaceho na aktuálne spracovávanej pozícii a nakoniec vypísať najväčší súčet, ktorý sme počas behu programu videli.

Príklad 1.4.

Máme mesto s n križovatkami, medzi ktorými vedie m ciest rôznych dĺžok. Na každej križovatke je semafór, ktorý sa v pravidelných intervaloch prepína. Našou úlohou je prejsť z križovatky a na križovatku b v čo najkratšom čase. Pritom však chceme najviac 5 krát prejsť na križovatke na červenú, inak musíme počkať kým sa semafór prepne.

Nemusíte riešiť ako presne semafóry fungujú, predstavte si, že máte funkciu, ktorá vám pre danú križovatku a čas kedy ste na ňu prišli povie, kedy najbližšie z tejto križovatky budete môcť odísť na zelenú.

Náčrt riešenia 1.4.

Dijkstrov algoritmus slúži na hľadanie najkratšej cesty v ohodnotenom grafe. Ak ho však chceme použiť, nemusíme mať priamo zadaný graf. Niekedy ho chceme vybudovať nanovo.

Predstavme si, že každý vrchol nášho nového grafu bude predstavovať stav (x, y) – som na križovatke x a na červenú som zatiaľ prešiel y krát. Je jasné, že našou úlohou je spočítať najkratšiu cestu medzi vrcholmi $(a, 0)$ a $(b, 5)$. Ostáva pridať do tohto grafu hrany.

Ak existuje cesta medzi vrcholmi x a y, do nášho grafu pridáme hrany z (x, i) do (y, i) pre všetky i. Tieto hrany znamenajú, že sme si na križovatke x počkali na zelenú a potom bez porušenia zákazu prešli na križovatku y. Takisto však pridáme hrany z (x, i) do $(y, i+1)$ (pre všetky vhodné i), ktoré budú hovoriť o tom, že sme porušili zákaz a prešli na červenú (dokonca ani nemusím overovať, či sme na červenú skutočne prešli).

Keď teda budeme spracovávať nejaký vrchol x a budeme sa pozerať po jeho susedoch, dáme si pozor na to, akého typu je daná hrana. Ak je to hrana, kde sme zákaz neporušili, musíme si vypočítať koľko na tejto križovatke musíme čakať a aj to pripočítať do vzdialenosti. Ak je to nezákonná hrana, nemusíme nič počítať, proste ju použijeme.

Dostali sme teda nový graf, na ktorom môžeme spustiť Dijkstrov algoritmus a ten sa už postará o správne vyriešenie.

Príklad 1.5.

Máme neorienotavý ohodnotený graf. Nájdite najkratšiu cestu medzi vrcholmi a, b, ktorá prechádza cez párny počet hrán (teda nemusí mať párnu dĺžku).

Náčrt riešenia 1.5.

Uvedomme si, že Dijkstrov algoritmus vo všeobecnosti neprehľadáva graf, ale nejakú množinu stavov. Pri klasickom algoritme sú tieto stavy: Äko najrýchlejšie sa vieme dostať z vrchola a do vrchola b?"

V tomto prípade sa nám stavy zmenia: Äko najrýchlejšie sa vieme dostať z vrchola a do vrchola b s paritou p?"(teda párny alebo nepárny počet hrán, ktoré prejdeme.) Tieto stavy si vieme zakresliť ako vrcholy nového grafu, v podstate každý vrchol zdvojíme a musíme zdvojiť aj hrany. Každá hrana totiž zmení paritu. Následne v tomto grafe spustíme zo stavu (vrchol a, párny) Dijkstru do stavu (vrchol b, párny).