Korespondenční Seminář z Programování

33. ročník **KSP** **Prosinec 2020**

V předvánoční čas vám přinášíme vzorová řešení druhé série. Čtení vzorových řešení je skvělý způsob, jak se naučit nové způsoby řešení úloh, a tak neváhejte a začtěte se do nich.

Připomínáme, že **seriál můžete za menší bodové ohodnocení stále odevzdávat** a to až do konce ročníku, takže jeho vzorové řešení vám vydáme až po skončení celého ročníku.

Vzorová řešení druhé série třicátého třetího ročníku KSP

33-2-1 Ostrovní království

Začněme tím, co je to vůbec strom. Strom lze třeba definovat tak, že je to souvislý graf, ve kterém se nevyskytuje cyklus. Z tohoto se dá vyvodit vlastnost, že strom o n vrcholech bude mít $n-1$ hran. Správnost tvrzení lze nahlédnout třeba tím, že když budeme postupně odtrhávat listy, tak po $n-1$ odtržených listech nám bude zbývat jeden vrchol, který nebude mít hranu.

Pokud stromy neznáte, tak vám doporučujeme nahlédnout do naší základní kuchařky.[1]

Nyní již k řešení samotné úlohy. Nejdřív předpokládejme, že pokud známe počet měst (vrcholů) a cest (hran), tak řešení je již triviální zjistit. Bude to rozdíl těchto dvou hodnot. Proč? Každý ostrov má síť cest ve tvaru stromu, a víme, že rozdíl měst a cest v jednom stromě je roven jedné. Každý ostrov tak zvětší celkový rozdíl měst a cest o hodnotu jedna.

Teď nastává ta těžší část, jak zjistíme počet měst a cest? Začneme tím, že zjistíme počet měst pomocí druhého dotazu (jestli mezi městy existuje cesta). Stačí nám najít největší index města (města jsou číslována sekvenčně). Kdybychom znali vrchní odhad na počet měst, tak bychom mohli udělat obyčejné binární vyhledávání, ale ten bohužel neznáme.

Ukážeme trik, kterým lze nalézt počet měst na $\log(n)$ dotazů, kde n je počet měst. Začněme s rozmezím měst od 0 do 1. Zeptáme se, jestli mezi městy existuje cesta, ale zároveň tento dotaz nám říká, že vůbec města existují! Pokud města existují, tak zvětšíme počet uvažovaných měst na dvojnásobek. Tedy rozmezí měst bude od 0 do 2. Obecně toto rozmezí bude od 0 do 2^i. Takto budeme opakovat dokud v odpovědi bude, že města existují. Po odpovědi, že město 2^i neexistuje, tak víme, že počet měst je mezi 2^{i-1} a 2^i.

Abychom byli úplně korektní, tak rozmezí bude od 2^{i-1} do 2^i-1, protože určitě víme, že město 2^i neexistuje. Nicméně mínus jedničku můžeme vynechat, protože na asymptotický počet dotazů konstanta nemá vliv.

Nyní najdeme počet měst pomocí binárního vyhledávání od 2^{i-1} do 2^i. Poté stačí najít počet cest binárním vyhledáváním od 0 do n a máme vyhráno.

Kolik dotazů celkem položíme? Označme si číslem n počet měst. Při hledání prvního čísla města, které neexistuje, jsme počet měst „přestřelili“ maximálně o hodnotu n a udělali jsme přitom $\mathcal{O}(\log n)$ dotazů (vždy jsme zvětšovali na dvojnásobek). Obě binární vyhledávání pak zabrala také maximálně $\mathcal{O}(\log n)$ dotazů, takže celkový počet dotazů na královské ministerstvo bude $\mathcal{O}(\log n)$.

Úlohu připravili: Michal Kodad, Jirka Setnička

33-2-2 Hasičské stanice

Nejprve si povšimněme, že nemá smysl, aby byl řádek a současně sloupec, ve kterém není umístěná stanice. Evidentně totiž můžeme umístit stanice do všech budov, které leží v průsečíku neobsazených řádků a neobsazených sloupců. Tyto budovy totiž nebyly dostupné z ani jedné stanice, tedy když z nich uděláme stanice, tak nepřijdeme o žádné obyvatele. Takto upravené řešení tedy zaručeně nebude horší než původní a po úpravě již nemůže existovat řádek a současně sloupec, kde by nebyla stanice.

Z předchozího odstavce tedy plyne, že má smysl se zaobírat pouze rozestavěními, kde buď v každém řádku je alespoň jedna stanice nebo v každém sloupci je alespoň jedna stanice. O ostatních rozestavěních jsme ukázali, že nemůžou být lepší než některé z uvažovaných.

Dále se zamyslíme nad případem, kdy v každém řádku bude alespoň jedna stanice. Druhý případ vyřešíme pouze prohozením řádků a sloupců.

Každá z budov musí být dostupná alespoň jednou stanicí, protože na daném řádku musí být alespoň jedna stanice. Z toho plyne, že obyvatelé mohou bydlet všude kromě hasičských stanic. Nemá tedy smysl, aby v jednom řádku bylo více než jedna stanice. Když totiž odstraníme z daného řádku všechny stanice až na jednu, tak pořád budou všechny budovy dostupné a navíc uvolníme některé stanice k bydlení. V každém řádku je tedy potřeba vybrat právě jednu budovu, kam umístíme stanici.

Snadno nahlédneme, že aby v daném řádku mohlo bydlet co nejvíce lidí, je nejlepší vybrat budovu s nejmenším počtem obyvatel.

Maximální hodnotu varianty, kdy je v každém řádku alespoň jedna stanice, tedy spočítáme jako součet všech budov bez součtu minim v každém řádku. Pro variantu, kdy je v každém sloupci alespoň jedna stanice, můžeme využít podobného algoritmu. Její hodnota tedy bude součet všech budov bez součtu minim v každém sloupci. O ostatních variantách jsme ukázali, že nemůžou být lepší. Finální výsledek tedy určíme jako maximum z předcházejících čísel.

Časová složitost je $\mathcal{O}(N \cdot M)$, protože musíme projít celý vstup.

V případě, že vstup budeme zpracovávat již při průchodu, paměťová složitost může být $\mathcal{O}(M)$. Budeme si počítat celkový součet všech budov a současně do dvou polí pro každý řádek a sloupec minimum v něm.

Úlohu připravili: Jirka Kalvoda, Standa Lukeš

[1] `http://ksp.mff.cuni.cz/viz/kucharky/zakladni-algoritmy`

33-2-3 Bludiště s turrety

Nebýt problémů s turrety střílejícími rakety, tak by tato úloha byla vlastně velmi jednoduchá. Máme bludiště se startovní a cílovou pozicí a políčka, na která můžeme vstoupit. Nejkratší cestu bychom nalezli *prohledáním do šířky* ze startu, jednoduchý algoritmus s použitím jedné fronty. Možná by nás chviličku potrápilo vypsání cesty potom, co se dostaneme do cíle. Ale i to bychom vyřešili tím, že bychom si na každé políčko jednoduše napsali, odkud jsme na něj přišli. Taková pěkná jednoduchá úloha… ale ouha, on tam doktor Zloun přidal turrety.

Tak co jen cestou kontrolovat, jestli náhodou nevrazíme do rakety? Během prohledávání do šířky si můžeme držet počet kroků, které jsme udělali od startu a s každým krokem bychom kontrolovali, jestli na daném políčku není raketa. Ale nám se někdy může vyplatit počkat (jako například v ukázkovém vstupu v zadání), nebo dokonce na jedno políčko vstoupit vícekrát (například když půjdeme proti turretu střílejícím skrz dlouhý tunel, který má co pár políček „odpočívadlo“, na které uhneme a počkáme, než raketa proletí, a pak se vrátíme do tunelu). Na takové případy je obyčejné prohledávání do šířky málo.

Stav políčka se totiž bude lišit v závislosti na tom, v jakou chvíli se na něj díváme. V čase t přes něj může prolétat raketa, ale v čase $t+2$ už bude volně průchozí. V podstatě bychom si mohli pro každý čas t nakreslit mapu bludiště se zakreslenými pozicemi raket. S každým krokem (nebo stáním na místě) se pak posuneme na odpovídající políčko v mapě $t+1$.

Tím jsme si vyrobili stavový prostor, který již můžeme zase prohledávat pomocí prohledávání do šířky – jenom naše políčka budou mít tři souřadnice a to řádek, sloupec a čas. Ale ještě není vyhráno, takový stavový prostor může být obrovský (protože průchod bludištěm může klidně trvat i více sekund, než je počet políček bludiště). Takový stavový prostor by bylo obtížné držet si v paměti.

Už na příkladu v zadání si ale můžeme všimnout, že se mapy bludiště po čase opakují. Například pokud by v bludišti byly jen turrety s intervalem 3, tak máme jen tři různé mapy. Pokud bychom měli dva typy turretů s intervaly 2 a 3, tak se stejné situace budou opakovat každé šesté kolo. A v obecnosti lze říci, že pro nějakou množinu turretů se budou situace opakovat po nejmenším společném násobku jejich intervalů. V zadání jsme slíbili, že turrety budou mít rozsah intervalů jen mezi čísly 1 až 6 včetně, což dává nejmenší společný násobek 60, což je pořád ještě rozumná hodnota.

Po načtení vstupu nám tak stačí spočítat si nejmenší společný násobek všech vyskytujících se intervalů, zduplikovat si mapu v tomto počtu a pro každý turret zakreslit políčka s raketou ve správné časy (pro políčko ve vzdálenosti d od turretu s intervalem k platí, že se na něm bude raketa nacházet v časech d, $d+k$, $d+2k$, … a tak dále, dokud nepřekročíme nejmenší společný násobek). Důležité je také nezapomenout na „ohnivý ocas“ rakety (ten jsme zavedli, aby nebylo validní běžet proti raketě a tím se jí vyhnout), který je opožděný o jedno políčko za samotnou raketou.

Na každé políčko s raketou si také poznamenáme, kdy poprvé přes toto políčko raketa přelétne. Díky tomu můžeme na začátku výpočtu vstoupit i na políčka, kde by sice raketa později byla, ale ta první tam ještě nestihla dolétnout. Měli jsme v úmyslu tuto vlastnost na některých vstupech testovat, ale nakonec jsme naše zákeřné plány neuskutečnili. V ukázkovém kódu se ale můžete podívat na to, jakým způsobem se taková podmínka dá implementovat.

Pak již máme připravený kompletní stavový prostor, na kterém již můžeme spustit prohledávání do šířky. Pokud si nejmenší společný násobek intervalů turretů označíme jako K, tak každý krok z mapy t povede na mapu $(t+1) \bmod K$. Zpětnou rekonstrukci cesty pak uděláme klasicky přes zpětné odkazy.

Časová a paměťová složitost prohledávání do šířky je lineární k velikosti stavového prostoru. Ten bude odpovídat $\mathcal{O}(KN)$ (kde N je počet políček), takže celková časová i paměťová složitost budou také $\mathcal{O}(KN)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/33-2-3.py`

Úlohu připravili: Kuba Pelc, Jirka Sejkora, Jirka Setnička

33-2-4 Oprava satelitů

Řešení využívající čtyři značky je vcelku přímočaré. Na začátku každé K-tice satelitů položíme značku S, abychom věděli, kde začíná. Poté přepojíme K následujících satelitů tak, aby ukazovali opačným směrem. Nakonec správně zapojíme začátek a konec K-tice a celé opakujeme N/K-krát (počet skupin):

```
N, K = map(int, input().split())
p = lambda *x: print(*x, sep="\n")

p("P S")
for _ in range(n // k):
     # ..-> o] -> [o -> o -> ...
     #      S
    p("P A", "D", "P B", "T S")
     # ..-> o] -> [o -> o -> ...
     #      S      B
     #      A

    for _ in range(K):
         # ... <- o    o -> o -> ...
         #   C    A    B
        p("P C", "T B", "P A", "D", "P B",
          "T A", "N C")
         # ... <- o <- o    o -> ...
         #        C    A    B

     # ..-> o] <> [o <- o <- o]    [o ->
     #      S                C      N
    p("T S", "D", "N B", "P C", "T S", "N A",
      "T C", "P S")
     #             .------------.
     # ..-> o]    [o <- o <- o] '> [o ->
     #      |      S         ^
     #      '----------------'
```

Instrukční složitost je lineární, protože se jedná o vnořené smyčky, kde vnější se opakuje N/K-krát a vnitřní K-krát, dohromady tedy $\mathcal{O}(N/K \cdot K) = \mathcal{O}(N)$.

Tři značky

Při řešení úlohy pouze se třemi značkami musíme být opatrnější. Na otáčení směru satelitu tři značky potřebujeme (když nemůžeme ze satelitů vytvářet cyklus, v takovém případě by to šlo), proto potřebujeme na začátku otáčení každé K-tice dělat trochu více práce.

Přepojení začátku a konce K-tice, které jsme v řešení se čtyřmi značkami dělali až po jejím otočení, tentokrát uděláme předem. Díky tomu po dokončení otáčení (které uděláme úplně stejně jako při řešení 4 značek) už nebude potřeba nic dalšího dělat. Tento postup opakujeme K/N-krát (počet skupin):

```
N, K = map(int, input().split())
p = lambda *x: print(*x, sep="\n")

p("P A")
for _ in range(N // K):
     # ..-> o] -> [o -> o -> ... -> o] -> [o ->
     #      A
    p("D", "P B", *["D"] * (K - 1), "P C")
     # ..-> o] -> [o -> o -> ... -> o] -> [o ->
     #      A      B                C
    p("T A", "N C", "T C", "D", "P A")
     # ..-> o]    [o -> o -> ... -> o] -> [o ->
     #      |      B                C      A
     #      '-----------------------^
    p("T B", "D", "P C", "T B", "N A")
     #             .----------------------v
     # ..-> o]    [o    o -> ... -> o] -> [o ->
     #      |      B    C           ^      A
     #      '-----------------------'
    p("T C", *["D"] * (K - 2), "N B")
     #             .-----------------------.
     #             |  .-------------.      v
     # ..-> o]    [o <' o -> ... -> o]    [o ->
     #      |      B    C           ^      A
     #      '-----------------------'
    p("T C", *["D", "P A", "T C", "N B", "P B",
      "T A", "P C"] * (K - 1))
     #             .------------------------v
     # ..-> o]    [o <- o <- ... <- o]    [o ->
     #      |      A    B           C
     #      '------------------------'
```

Instrukční složitost je opět lineární, ze stejného důvodu jako při řešení se čtyřmi značkami.

Úlohu připravili: Kuba Pelc, Tom Sláma

33-2-X1 Závody formulí

Předehra: Každá funkce má svůj obvod

Začneme jednoduchým pozorováním: Mějme nějakou funkci $f(x_1, \ldots, x_k)$, která každé k-tici bitů přiřazuje jednobitový výsledek. Všimneme si, že tuto funkci jde spočítat formulí, a tím pádem i obvodem.

Proč tomu tak je? Funkci můžeme popsat tabulkou, která každé z 2^k možných vstupních k-tic přiřadí výsledek. Pro XOR vypadá takto:

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Kdyby pravý sloupec obsahoval samé nuly, stačila by konstantně nulová formule (třeba $x_1 \wedge \neg x_1$). Pokud v něm bude právě jedna jednička, postačí AND proměnných a jejich negací: například jedničce v řádku $x_1 = x_2 = 1$ odpovídá $x_1 \wedge x_2$, jedničce pro $x_1 = 0$, $x_2 = 1$ odpovídá $\neg x_1 \wedge x_2$ atd. A pokud je na pravé straně jedniček více, stačí uvážit OR formulí generujících jednotlivé jedničky. Pro XOR tedy vyjde $(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$, což už vlastně známe ze zadání.

Tím pádem umíme vytvořit obvod pro libovolnou funkci. Neslibujeme, že onen obvod je malý :) Nicméně funguje a jeho velikost je pro konkrétní funkci konstantní.

Centroidy stromů

Ještě chvíli strpení, prosím. Potřebujeme vybudovat jeden užitečný nástroj pro práci se stromy. Popisujeme ho zvlášť, protože se hodí i v jiných úlohách.

Mějme nějaký strom T o n vrcholech. *Centroid* říkáme libovolnému vrcholu, jehož odebráním strom rozebereme na komponenty, z nichž každá má nejvýše $n/2$ vrcholů. V těchto komponentách můžeme zase najít centroidy a tak dále, čímž vytvoříme hierarchii čím dál menších stromečků. Jelikož počet vrcholů se pokaždé zmenší aspoň dvakrát, po maximálně $\log n$ krocích musíme dostat stromečky konstantní velikosti. (Logaritmem bez uvedeného základu myslíme vždy dvojkový.)

Zbývá dokázat, že v každém stromu existuje aspoň jeden centroid. Strom libovolně zakořeníme a pro každý vrchol v označíme $p(v)$ počet jeho potomků (včetně v samotného). Pak najdeme nejhlubší vrchol u, pro nějž je $p(u) > n/2$. Dokážeme, že u je centroid. Podstromy ležící pod u mají nejvýše $n/2$ vrcholů (jinak by $p(u)$ nebyl nejhlubší takový vrchol), zbytek stromu má $n - p(u) < n/2$ vrcholů.

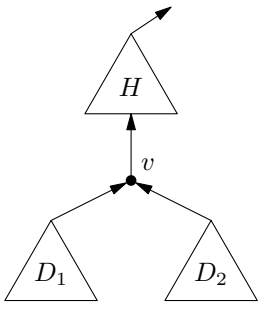
Náš důkaz dokonce dává lineární algoritmus pro nalezení centroidu pomocí DFS. A když ho budeme volat rekurzivně, sestrojíme celou dekompozici v čase $\mathcal{O}(n \log n)$. Neví se, zda to jde rychleji.

Řešení pomocí centroidů

Teď centroidovou dekompozici použijeme k vyřešení úlohy. Představme si stromovou strukturu formule – už v zadání jsme ukazovali, že tento strom odpovídá nějakému obvodu, který formuli spočítá, byť neefektivně. Označme n počet vrcholů stromu, což je asymptoticky totéž jako délka formule.

Najdeme centroid stromu. To je nějaké hradlo. Pod ním jsou nejvýše dva „dolní“ podstromy, které počítají vstupy pro naše hradlo. Hradlo pak svůj výstup předá „hornímu“ podstromu, který vydá konečný výsledek.

Situaci sledujme na následujícím obrázku: v je centroid, H horní podstrom, D_1 a D_2 dolní podstromy. Každý podstrom se navíc může odkazovat na vstupní proměnné formule.



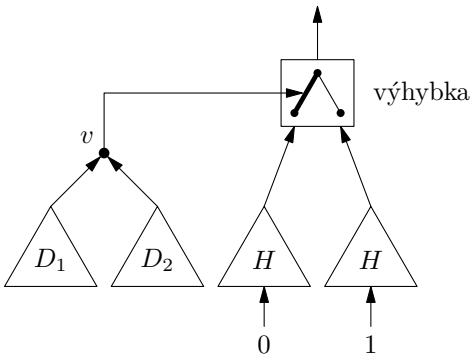
Kdyby horní podstrom byl prázdný, bylo by řešení triviální. Paralelně bychom spočítali oba dolní podstromy (ty závisí jen na vstupních proměnných formule). A pak bychom použili naše hradlo. Za jednu úroveň dekompozice bychom tedy přidali jediný takt výpočtu. Úrovní dekompozice je $\log n$, takže obvod počítá $\log n$ taktů.

Jenže horní podstrom obvykle prázdný nebude a musí počkat na výstup našeho hradla. Mohli bychom nejdřív paralelně spočítat oba dolní podstromy, pak naše hradlo, a nakonec horní podstrom. V nejhorším případě bychom tedy za každou úroveň dekompozice počet taktů cca zdvojnásobili. Z toho by vyšel lineární počet taktů, protože bychom vlastně počítali původní strom.

Použijeme proto trik: horní podstrom spočítáme ve dvou verzích: jednu pro nulový výsledek našeho hradla, druhou pro jedničkový. To můžeme udělat předem a až budeme znát výsledek hradla, dodatečně si z verzí vybereme tu správnou.

Potřebujeme tedy sestrojit „výhybku“ – obvod, který dostane výsledek našeho hradla y a výsledky h_0 a h_1 obou verzí horního stromu. Jeho výsledkem pak bude h_y. Výhybka je nějaká booleovská funkce tří proměnných, takže podle předehry existuje obvod, který ji počítá.

Celé to bude vypadat takto:



Konstrukci obvodu tedy můžeme popsat rekurzivně. Strom formule rozdělíme centroidem na dva dolní podstromy a jeden horní. Rekurzivně sestrojíme obvody pro oba dolní podstromy a pro obě možné verze horního podstromu. Výstupy těchto obvodů pak propojíme konstantně mnoha novými hradly (původní centroidové hradlo a výhybka).

Jelikož s každým krokem rekurze se velikost podstromů vydělí aspoň dvěma, hloubka rekurze činí nejvýše $\log n$. Každá úroveň rekurze k délce výpočtu vytvořeného obvodu přispěje konstantním počtem taktů. Obvod tedy pracuje v $\mathcal{O}(\log n)$ taktech.

Trochu pracnější bude odhadnout, z kolika hradel se vytvořený obvod skládá. Uvažujme strom rekurze našeho algoritmu – pozor, to je úplně jiný strom než ten popisující formuli! Strom má výšku $v \leq \log n$, v každém vrcholu bydlí nějaký podproblém, při jehož řešení vznikne konstantní počet hradel. Počet hradel je tedy $\mathcal{O}$(počet vrcholů stromu rekurze), což je $\mathcal{O}$(počet listů stromu rekurze), jelikož každý vnitřní vrchol má aspoň 2 syny. Proto nám stačí spočítat listy.

Nechť S_i značí součet velikostí všech podproblémů na i-té hladině stromu rekurze. V kořeni je zjevně $S_0 = n$. Dokážeme, že $S_{i+1} \leq (3/2) \cdot S_i$. Mějme nějaký podproblém na i-té hladině. Označme t jeho velikost. Centroid rozdělí podproblém na dolní stromy velikostí d_1 a d_2 a horní strom velikosti h. Z vlastností centroidu víme, že $d_1, d_2, h \leq t/2$ a $d_1 + d_2 + h < t$. Na $(i+1)$-ní hladině jsme vytvořili podproblémy velikostí d_1, d_2, h a ještě jednou h. Jejich celková velikost tedy je $d_1 + d_2 + 2h \leq t + h \leq 3/2 \cdot t$. Sečtením přes všechny podproblémy na hladině dostaneme $S_{i-1} \leq (3/2) \cdot S_i$.

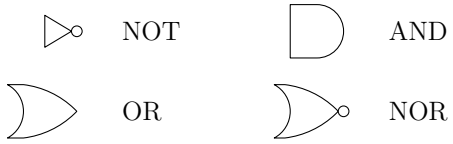
Na poslední hladině (té s listy, tedy v-té) tudíž máme $S_v \leq n \cdot (3/2)^{\log n}$. A jelikož každý podproblém v listu je konstantně velký, musí být celkový počet listů, a tím pádem i všech hradel $\mathcal{O}(n \cdot (3/2)^{\log n}) = \mathcal{O}(n \cdot 3^{\log n}/2^{\log n}) = \mathcal{O}(n \cdot 3^{\log n}/n) = \mathcal{O}(3^{\log n}) = \mathcal{O}((2^{\log 3})^{\log n}) = \mathcal{O}(2^{\log 3 \cdot \log n}) = \mathcal{O}((2^{\log n})^{\log 3}) = \mathcal{O}(n^{\log 3}) \approx \mathcal{O}(n^{1.59})$. (Zde trochu švindlujeme – všechny listy nemusí ležet na poslední hladině. Ale pokud jsou některé výše, uvedené nerovnosti platí tím spíš.)

Kdyby nás ještě zajímala časová složitost algoritmu, kterým jsme obvod vytvořili, můžeme ji spočítat podobnou úvahou. V každém podproblému strávíme čas lineární ve velikosti podproblému hledáním centroidu a předáváním vstupů do rekurze. Celkový čas tedy bude lineární v součtu velikostí hladin $\sum_i S_i$. Jelikož S_i rostou exponenciálně, sčítáme geometrickou řadu a její součet je asymptoticky roven nejvyššímu členu S_v. Časová složitost je tedy také $\mathcal{O}(n^{\log 3})$.

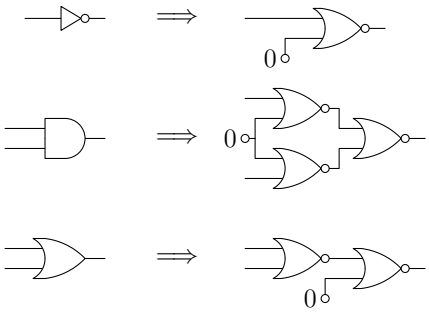
Řešení pomocí heavy-light dekompozice

Takhle zajímavou úlohu přeci nemůžeme odbýt jediným řešením :) Ukážeme si proto další možný přístup. Z něj také nakonec vyjdou logaritmicky hluboké obvody, které budou navíc jen lineárně velké.

Nadále budeme používat následující značení hradel:



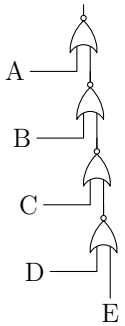
Nejprve si uvědomíme, že všechna uvažovaná hradla lze sestrojit pomocí jediného druhu hradel – NORu. To je hradlo, které vznikne zřetězením negace za OR. Vrací tedy pravdu pouze v případě, že oba jeho vstupy jsou nepravdivé.



Všimneme si, že těmito náhradami se počet hradel zvýší jen konstantně krát. Navíc se nikde nerozděluje signál. Vznikne nám tedy binární zakořeněný strom. Jeho listy jsou buď vstupu nebo konstanty a vnitřní vrcholy jsou NORy.

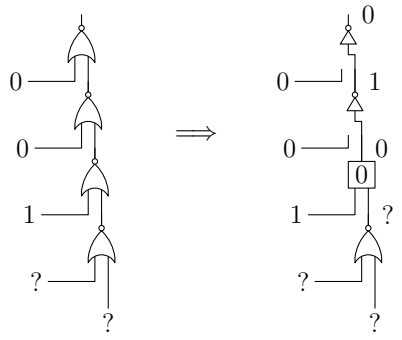
Na něm si uděláme heavy-light dekompozici. Pokud ji neznáte, podívejte se na řešení úlohy 29-4-7. Pak se každou cestičku z těžkých hran pokusíme nahradit tak, aby se celková hloubka stromu snížila.

V původním stavu je každá cestička posloupností NORů. Na každý z nich (kromě posledního) je přiveden jeden signál, muže to být i konstanta:



Dále si povšimneme, že záleží pouze na poloze prvního vstupu (počítáno od kořene), který má hodnotu 1. NOR, který má jako jeden vstup 1, totiž vždy vrátí nulu. Nezáleží tedy na tom, jaká hodnota přijde z druhého vstupu, z cestičky těžkých hran.

Od tohoto hradla směrem ke kořeni už jsou jen NORy s jedním nulovým vstupem (za předpokladu, že jsem pracovali s nejbližší jedničkou). Ty se chovají pouze jako negace. Tedy když nad prvním hradlem se vstupem 1 je sudý počet NORů, tak je celkový výstup z této části obvodu 0, v opačném případě 1.



Pro každý vstup je tedy jasně dané, jaký bude výsledek, když se jedná o první vstup s hodnotou 1.

V případě, že všechny vstupy budou 0, tak je výsledek 1 právě tehdy když je počet hradel lichý.

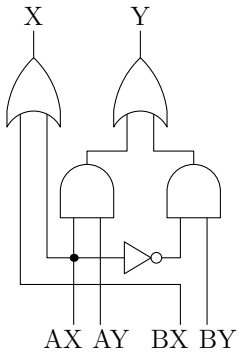
Sestrojíme si binární stromeček, kterým nahradíme uvažovanou cestu z těžkých hran. Jeho listy budou vstupy dané části obvodu seřazené podle vzdálenosti od kořene.

Hrany v daném stromu ve skutečnosti budou dva signály. První říká, jestli v daném podstromu existuje vstup s hodnotou 1 (označme X). Pokud ano, tak druhý (označme Y) určuje, jaký by byl výstup z našeho obvodu v případě, že by daný podstrom obsahoval první jedničku.

Vnitřní vrcholy budou jen tyto dvojice signálů spojovat. Složku X spočítáme jednoduše jako OR vstupních X.

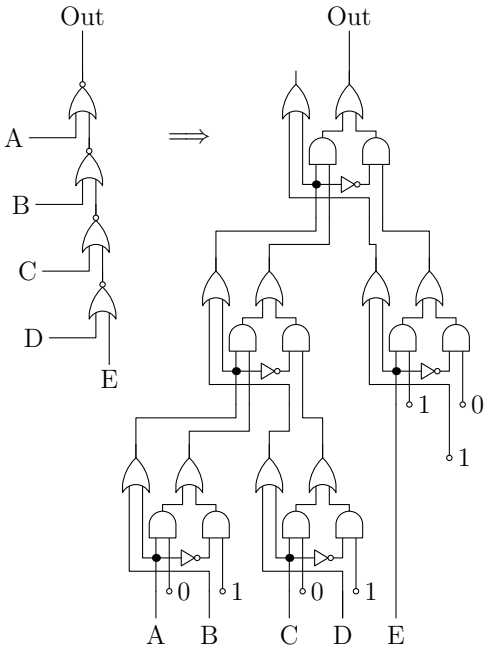
Složku Y spočítáme takto: V případě, že první podstrom obsahuje jedničkový vstup, tak se použije hodnota z něho, v opačném případě se použije hodnota z druhého podstromu. V případě, že ani v jednom z podstromů není jednička, tak nás nezajímá návratová hodnota Y.

Evidentně toto jsou jen dvě logické funkce a ty dle úvodního pozorování umíme sestrojit. Pro zajímavost si ale můžeme ukázat konkrétní obvod:



Z listu povede jako existence jedničky v podstromu původní vstup a výsledek v případě první jedničky bude konstanta.

Abychom ošetřili případ, kdy všechny vstupy jsou nulové, tak na konec přidáme ještě jeden list. Ten bude vždy aktivní a jeho výsledek bude konstanta dle parity počtu NORů v cestičce. Celé to bude vypadat takto:



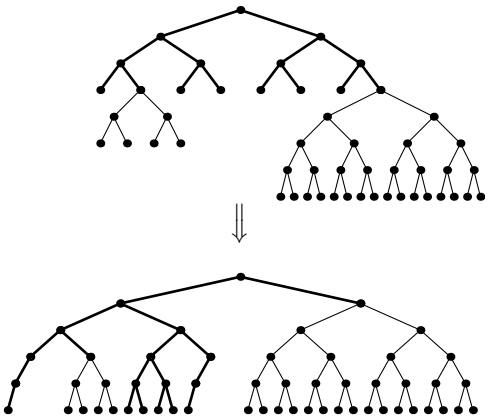
Každou těžkou cestu jsme tedy nahradili obvodem hloubky $\mathcal{O}(\log n)$. Z vlastností heavy-light dekompozice víme, že každá cesta z kořene do listu obsahuje $\mathcal{O}(\log n)$ lehkých hran, mezi kterými jsou kusy těžkých cest. Celý obvod má proto hloubku $\mathcal{O}(\log^2 n)$.

Co se prostoru týče, každý vrchol stromu jsme nahradili $\mathcal{O}(1)$ hradly, takže sestrojený obvod má $\mathcal{O}(n)$ hradel.

Další optimalizace

Předchozí algoritmus fungoval tak, že vytvořil vyvážené stromy a zavěsil je pod sebe. Ovšem pod každý strom věšíme různě velké stromy, takže výsledný obvod už může vyjít nevyvážený, a tudíž má hloubku $\mathcal{O}(\log^2 n)$ místo $\mathcal{O}(\log n)$.

Pojďme vymyslet, jak obvod vyvážit. Bude to fungovat tak, že velké části obvodu zasuneme do stromečku, pod kterým jsou pověšené.



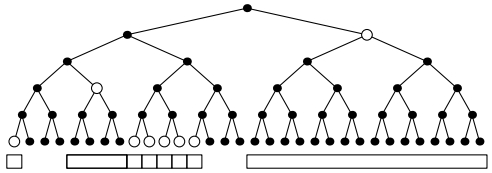
Vyvažovat budeme od listů obvodu. Tedy když budeme věšet podobvody pod nějaký stromeček, tyto podobvody už budou vyvážené.

Vyvažování stromečku bude probíhat následovně: Nejprve si vytvoříme posloupnost listů a pak nad ní vybudujeme samotný strom. (Všimněte si, že operace kombinující X a Y je asociativní, takže výsledek nezávisí na tvaru stromu, pouze na pořadí listů.)

Pro každý podobvod, který chceme pod aktuální stromeček zavěsit (ve správném pořadí), si v posloupnosti zabereme 2^h políček, kde h značí hloubku podobvodu. Navíc budeme chtít, aby zabraný úsek začínal na indexu, který

je násobkem 2^h. Proto před něj ještě přidáme nějaký počet prázdných políček, nejvýše však $2^h - 1$.

Na konec posloupnosti listů doplníme prázdné místo tak, aby délka posloupnosti (označme S) byla mocninou dvojky. Tím ji nejvýše dvakrát prodloužíme.



Každému podobvodu na pověšení jsem tedy přiřadili interval po sobě jdoucích listů. Ovšem díky tomu, že první z nich je na kulatém indexu 2^h, podstrom jejich nejbližšího předka obsahuje pouze tyto listy.

Vytvoříme si tedy strom s počtem vrcholů odpovídajícím délce posloupnosti. Každý obvod, který chceme pověsit, pak připojíme místo podstromu pod nejbližším předchůdcem příslušného intervalu listů.

Při implementaci pak samozřejmě nemusíme vytvářet pole listů. Stačí si jen počítat indexy. Nakonec přeskočíme všechny vrcholy, které mají jen jednoho potomka (nepočítáme-li podstromy obsahující pouze přeskočené listy).

Celý výsledek tedy bude mít hloubku $\log S$.

Takto postupně vyvážíme každý stromeček.

Odhad hloubky řešení. Nahlédneme, že délka posloupnosti při vyvažování stromečku je nejvýše čtyřikrát větší než součet délek intervalů vrcholů, které pod něj věšíme. Mezery před úseky přidají maximálně jednou tolik a doplnění na mocninu dvojky pak přidá ještě maximálně jednou tolik.

Označme celkový počet listů původního obvodu NORů jako N. Z heavy-light dekompozice víme, že každý list projde jen nejvýše $\log N$ spojeními. Což ovšem znamená, že součet délek původních posloupností (tedy N) se nejvýše $(\log N)$-krát vynásobí čtyřmi. Finální délka posloupnosti proto bude nejvýše $N \cdot 4^{\log N} = N^3$. Z toho vyjde hloubka obvodu $\mathcal{O}(\log N^3) = \mathcal{O}(\log N)$.

Počet hradel v řešení je $\mathcal{O}(n)$. Náš strom v každém vrcholu obsahuje pouze konstantní počet hradel. Jelikož počet listů je $\mathcal{O}(n)$ a každý vnitřní vrchol obsahuje dva potomky, celkový počet vrcholů je $\mathcal{O}(n)$. Heavy-light dekompozice běží v čase $\mathcal{O}(n)$. Každou těžkou cestu zpracujeme v čase lineárním s její délkou, celkem tedy také $\mathcal{O}(n)$. Celý algoritmus proto běží v čase $\mathcal{O}(n)$ a spotřebuje $\mathcal{O}(n)$ prostoru.

Úlohu připravili: Jirka Kalvoda, Martin „Medvěd“ Mareš

Výsledková listina KSP-X po druhé sérii třicátého třetího ročníku

	řešitel	*škola*	*ročník*	*sérií*	*1-X1*	*2-X1*	*celkem*
0.					10	10	20,0
1.	Viktor Fukala	GKepleraPH	4	6	9	10	19,0
2.	Jan Adámek	GKepleraPH	4	7	7	6	13,0
3.	Eliška Macáková	CENADA BA	1	2	9	1	10,0
4.	Ondřej Sladký	GMikulášPL	4	9	9		9,0
5.	Václav Janáček	GJarošeBO	4	5	8		8,0
6.	Daniel Skýpala	GTomkovaOL	3	17	4	1	5,0
7.	Vladimír Chudý	G Chrudim	4	17	4,5		4,5
8.	Lukáš Veškrna	GKepleraPH	3	2	4		4,0
9.	Jiří Kvapil	GTomkovaOL	3	16	2	1	3,0
10.	Robert Jaworski	GÚstavníPH	3	4	1	1	2,0

Bonusové úlohy označené „X“ mají svou vlastní výsledkovou listinu a nepočítají se do normálního bodování ročníku.

Výsledková listina druhé série třicátého třetího ročníku KSP

	řešitel	*škola*	*ročník*	*sérií*	*2-1*	*2-2*	*2-3*	*2-4*	*2-S*	*2-X1*	*série*	*KSP-X*	*celkem*
0.					9	11	12	13	15	10	60,0	20,0	120,0
1.	Filip Hejsek	GPísnickáPH	4	4	9	11	12	13	15		60,0	0,0	122,0
2.	Kristýna Petrlíková	SPŠJičín	3	12	10	10	12	13	15		60,0	0,0	121,0
3.	Jan Adámek	GKepleraPH	4	7	9	11	12	13	15	6	60,0	13,0	119,0
4.	Viktor Fukala	GKepleraPH	4	6	9	11	12	13	15	10	60,0	19,0	118,0
5.–6.	Jiří Kvapil	GTomkovaOL	3	16	9	9	12	13	15	1	58,0	3,0	116,0
	Eliška Macáková	CENADA BA	1	2	9	11	12	9	15	1	56,0	10,0	116,0
7.	Daniel Skýpala	GTomkovaOL	3	17	9	11	12	13	15	1	60,0	5,0	111,0
8.	Robert Jaworski	GÚstavníPH	3	4	9	11	12	13	15	1	60,0	2,0	104,0
9.–10.	Vladimír Chudý	G Chrudim	4	17	2	11	12	13	6		44,0	4,5	103,0
	Ondřej Skácel	GTomkovaOL	2	2	9	10	12	13	14		58,0	0,0	103,0
11.	Vít Skalický	GPísnickáPH	3	16	8		12	13	15		48,0	0,0	94,0
12.	Jan Kotovský	GPísnickáPH	2	3	2	11	12	13	15		53,0	0,0	92,0
13.	Jakub Surga	ParkLane	3	2	9	11	5	13	11		49,0	0,0	87,0
14.	Ondřej Sladký	GMikulášPL	4	9	9	11			14		34,0	9,0	81,0
15.	Jakub Ondroušek	GTomkovaOL	1	2	2		12	13	15		42,0	0,0	79,0
16.	Lukáš Veškrna	GKepleraPH	3	2	5	11		13	11		40,0	4,0	78,0
17.	Matej Štencel	GPošKošice	4	5			12	13	15		40,0	0,0	77,0
18.	Patrik Herman	GTomkovaOL	2	3	9	11		13	10		43,0	0,0	69,0
19.	Pavel Jordán	GPOA Znojmo	2	2	9				6		15,0	0,0	66,0
20.	Janek Hlavatý	GJirsíkaČB	2	6	2			9	15		26,0	0,0	60,0
21.	Dominik Farhan	GMikulášPL	4	6	9				9		18,0	0,0	54,0
22.	Prokop Randáček	GFXŠaldyLI	2	4	1	10			15		26,0	0,0	53,0
23.	Václav Janáček	GJarošeBO	4	5	8	11		13			32,0	8,0	52,0
24.	Adam Kolník	SSŠVTPraha	2	2			5	0	10		15,0	0,0	44,0
25.	Šimon Genčur	GBBr	1	4					6		6,0	0,0	34,0
26.	Martin Havelka	Gym Třeboň	3	3							0,0	0,0	28,0
27.	Jiří Bartošík	SUHr	3	2							0,0	0,0	22,0
28.	Albert Kučera	GNadŠtolPH	4	4							0,0	0,0	21,0
29.	Kristýna Umlaufová	SPŠOstrov	4	2							0,0	0,0	20,0
30.–31.	Klára Grinerová	GZborovPH	4	2			0	8			8,0	0,0	19,0
	Jáchym Tuma	G FrýdlNOs	0	2				7			7,0	0,0	19,0
32.	Andrej Thomas Dobrev	GJHroncaBA	4	1							0,0	0,0	18,0
33.	Michal Žáček	MensaG	4	1	1	6		8			15,0	0,0	15,0
34.	Tomáš Kašpárek	G FrýdlNOs	3	1							0,0	0,0	12,0
35.–41.	Vojtěch Březina	GCoubTábor	4	4							0,0	0,0	10,0
	Petr Filip	GLovosice	2	1							0,0	0,0	10,0
	Vojtěch Gaďurek	PORGPha	4	1							0,0	0,0	10,0
	Štěpán Kovář	GNadKavaPH	4	1							0,0	0,0	10,0
	Michal Pavlíček	MendelGOP	3	1							0,0	0,0	10,0
	Daniel Šoltýs	GTřeKošice	3	2							0,0	0,0	10,0
	Filip Úradník	GyMimoň	4	1							0,0	0,0	10,0
42.	Jan Ráček	SPŠEMasLI	1	1							0,0	0,0	7,0
43.–44.	Bohumil Kulvejt	G Sokolov	3	1							0,0	0,0	6,0
	Josef Malý	GPísnickáPH	2	1							0,0	0,0	6,0
45.–46.	Veronika Jůzková	MensaG	3	1							0,0	0,0	2,0
	Matěj Strnad	ZŠRiegraSM	0	1							0,0	0,0	2,0

Bonusové úlohy z jednotlivých sérií se nepočítají do bodování ročníku. Mají svou vlastní výsledkovou listinu a za jejich úspěšné vyřešení (alespoň polovina bodů za úlohu) udělujeme speciální odměny.