VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AUTOMATIZACE NÁVRHU ŠROUBOVANÉ MŘÍŽE Z POZINKOVANÝCH TRUBEK

AUTOMATED DESIGN OF SCREWED GRATING MADE OF GALVANIZED PIPES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RADEK HŮLKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Hůlka Radek**
Program: Informační technologie
Název: **Automatizace návrhu šroubované mříže z pozinkovaných trubek**
Automated Design of Screwed Grating Made of Galvanized Pipes
Kategorie: Algoritmy a datové struktury

Zadání:

1. Proveďte rešerši ohledně pasivních prostředků pro zabezpečení majetku a jejich použití v praxi - nejprve obecně, poté detailněji v souvislosti s mřížemi.
2. Zdokumentujte i) požadavky (konstrukční, funkční aj.) kladené na mříže v kontextu jejich použití v praxi a ii) detaily o dostupných pozinkovaných trubkách, a dalších dílech, spojitelných závity (dále jen "trubkách"). Navrhněte algoritmicky zpracovatelný způsob uložení výše zmíněných požadavků a detailů.
3. Navrhněte algoritmus, který na základě i) vstupních uživatelských požadavků kladených na mříž (např. údajů o otvoru pro zabudování mříže, o vzdálenosti mezi trubkami, mezi mříží a okolím, o umístění zámku, pantů či dalších doplňkových prvků) a ii) seznamu trubek využitelných pro konstrukci mříže vygeneruje, je-li to možné, konstrukci mříže maximálně zohledňující uživatelské požadavky a splňující kritérium nerozebratelnosti během umístění mříže v daném otvoru.
4. Algoritmus navržený v předchozím bodě implementujte pomocí běžně dostupných prostředků, do implementace zvažte zahrnout generování vizualizace mříže, seznamu potřebných dílů, montážního postupu atp.
5. Vhodně ověřte a kriticky zhodnoťte i) chování a výstupy algoritmu pro různé vstupy a ii) realizovatelnost výstupů generovaných algoritmem. Identifikujte případné problémy s algoritmem či jeho výstupy a navrhněte způsob jejich řešení.
6. Diskutujte možné směry pokračování v projektu.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

Práce se zabývá automatizovaným návrhem mříží z pozinkovaných trubek a snaží se tak rozšířit možnosti pasivní ochrany majetku. Cílem je prostudovat problematiku návrhu mříží a podobných problémů. Následně popisuje postup návrhu funkčního řešení, které je nakonec implementováno a otestováno.

Abstract

This bachelor's thesis is analyzing problematics of designing screwed grating made of galvanized pipes and fittings. The main goal is to study the problems of designing screwed grating and similar problems. Then continues with description of the algorithm design, its implementation and testing.

Klíčová slova

návrh mříží, algoritmus, NP-uplný problém, pasivní bezpečnost, javaScript

Keywords

screwed grating, algorithm, NP-Complete problems, passive safety, javaScript

Citace

HŮLKA, Radek. *Automatizace návrhu šroubované mříže z pozinkovaných trubek*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Automatizace návrhu šroubované mříže z pozinkovaných trubek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Josefa Strnadela Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Radek Hůlka
4. června 2020

Poděkování

Tímto bych rád poděkoval svému vedoucímu Ing. Josefovi Strnadelovi, Ph.D. za odborné vedení, vstřícnost při konzultacích a cenné rady, které mi pomohly tuto práci vyhotovit.

Obsah

Kapitola 1

Úvod

Pocit bezpečí je velmi důležitý, a proto je vhodné se o bezpečnost zajímat a využívat dostupných aktivních či pasivních prvků bezpečnosti, jak radí Policie České Republiky. [22]

Podle statistik je druhý nejčastější způsob vniknutí do objektu skrze nezabezpečené okno, tímto způsobem je uskutečněno 15% vloupání, zatímco přes mříž to jsou pouze 3% ze všechn vloupání [9]. Zabezpečení oken mřížemi tedy dramaticky snižuje pravděpodobnost vyloupení objektu a měli bychom vždy zvážit tuto formu pasivní ochrany majetku.

Tato práce řeší problematiku návrhu mříží z běžně dostupných pozinkovaných trubek a fitinek. Nalezení optimálního návrhu mříže zadaných rozměrů ze zadaných dílů je nedeterministický polynomiální úplný problém (NP-úplný problém). To znamená, že neexistuje jednoduchý efektivní spůsob, jak takové řešení najít a naivní algoritmus musí prohledat všechna potenciální řešení. Naopak ověření správnosti nalezeného řešení je jednoduché.

Cílem práce je navrhnout a implementovat algoritmus, který nalezne a vizualizuje všechny nejlepší návrhy mříží pro uživatelem zadané rozměry mříže a poskytnuté díly.

V první teoretické části se budu věnovat základním informacím o bezpečnosti se zaměřením na mříže. Zmíním se o dostupnosti a parametrech trubek a fitinek. Rozeberu NP-úplné problémy – co jsou zač, ukáži nejznámější NP-úplný problém – *0-1 Knapsack problem* česky *problém batohu* a porovnám problematiku návrhu mříží s již známým *Change making* NP-úplným problémem,

V další části se zaměřím na návrh algoritmu pro automatizovaný návrh mříží a následně se budu zabývat implementací navrženého algoritmu.

Nakonec provedu otestování implementovaného řešení a provedu i praktický test sešroubování mříže z reálných trubek.

V závěru bakalářské práce provedu shrnutí a kritické zhodnocení odvedené práce, nakonec zmíním možné směry pokračování práce.

Kapitola 2

Teorie - bezpečnost, trubky a fitinky, NP-úplné problémy

2.1 Pasivní prvky zabezečení majetku

Pasivními prvky zabezpečení majetku chápeme takové překážky, které případnému zloději nebo škůdci ztíží až znemožní zabezpečený majetek odcizit nebo poničit. V dnešní době existuje celá řada takových bezpečnostních prvků:

- Otvory – mříže, rolety
- Okna – tvrzená skla, bezpečnostní fólie
- Dveře – bezpečnostní dvěře a zámky, rozvorové systémy, řetízky, panoramatická kukátka, kryty zámků
- Objekty – ploty, ostnaté dráty, atrapy elektronických bezezpečnostních systémů
- Cennosti – trezory

2.1.1 Mříže

Mříž je jednou z nejstarších zábran používaných k prevenci proti vniknutí do zabezpečeného prostoru a řadí se mezi pasivní bezpečnostní prvky. Z pohledu bezpečnosti, patří mezi mechanické zábranné systémy. Mříže můžeme dělit podle [16]:

- Konstrukce – příklady jednotlivých konstrukcí jsou zobrazeny na obrázku 2.1
  - pevné nebo odnímatelné
  - otevírací – například nůžkové, které umožňují shrnutí do jedné strany
  - rolovací – vertikální pohyblivé spojení ok spolu s horizontálním vedením mříže umožňují nevíjení mříže
- Umístění – vnitřní, meziokenní, vnější
- Materiálu – ocel, hliníkové slitiny
- Ovládání – manuální, elektrické

Obrázek 2.1: Příklady různých konstrukcí mříží - zleva pevná mříž, nůžková mříž a rolovací mříž. (Převzato z [15], [21] a [12])

Požadavky na mříže jsou od roku 2012 specifikovány normou *ČSN EN 1627 – Dveře, okna, lehké obvodobé pláště, mříže, okenice – Odolnost proti vloupání – Požadavky a klasifikace*. Tato norma zavádí i nové bezpečnostní třídy označované RC1 až RC6, jejich charakteristika je uvedena v tabulce 2.1, kde v případě mříží je čas napadení, počítán jako doba potřebná k vytvoření průlezového otvoru 20×40 cm.

Konstrukční požadavky na pevné mříže:

- Konstrukce musí být tuhá, stabilní, v celé své ploše se nesmí prohýbat a pruty se nesmí dát roztáhnou.
- Spoje tyčí musí být svařeny nebo snýtovány do nerozebíratelného celku.
- Ukotvení musí odolat vytržení ze zdi, proto se doporučuje zapuštění alespoň 14cm do zdi.
  - Přímé – kotvící tyče jsou rovné a zapuštěné rovnoběžně do zdi.
  - Kolmé – kotvící tyče jsou ohnuté do pravého úhlu a zasazeny kolmo do zdi.
- Velikost ok mříží se doporučuje 10 cm mezi vertikálními příčníky a 20 cm mezi horizontálními příčníky.
- Průřez tyčí musí být minimálně 1 cm^2.

Mříž z pozinkovaných trubek a fitinek by podle konstrukce, po svaření spojů, spadala do kategorie pevných mřížích. Běžně dostupné pevné mříže se vyrábějí v bezpečnostních třídách RC2 nebo RC3, použitý materiál je nejčastěji $10 - 14$ mm čtvercová ocel a cena se pohybuje od 2200,- Kč/m^2.

RC	Čas napadení	Předpokládané metody a pokusy o vloupání
RC1	Neaplikuje se	Příležitostný zloděj se pokouší o vloupání s použitím malého jednoduchého nářadí a fyzickým násilím, např. kopáním, naražením ramenem, vytrháváním. Zloděj nemá znalosti o úrovni odolnosti MZS, má málo času a snaží se nezpůsobit hluk.
RC2	3 min	Příležitostný zloděj se navíc pokouší o vloupání s použitím jednoduchého nářadí a fyzickým násilím. Má malé znalosti o úrovni odolnosti MZS, má málo času a snaží se nezpůsobit hluk.
RC3	5 min	Zloděj se pokouší překonat MZS s použitím páčidla délky 710 mm, které umožňuje zvýšené fyzické násilí, a dalšího šroubováku, ručního nářadí, jako malé kladívko, důlčíky a mechanická ruční vrtačka. Zloděj má určité povědomí o systému uzávěru a s tímto nářadím je schopen těchto znalostí využít.
RC4	10 min	Zkušený zloděj používá navíc zámečnické kladivo, sekeru, dláta, sekáče, přenosnou akumulátorovou vrtačku atd. Toto další nářadí umožňuje zloději rozšířit počet způsobů napadení, případně jejich kombinace – vrtání, sekání, páčení, atd. Neznepokojuje se hlukem.
RC5	15 min	Velmi zkušený zloděj používá navíc jednoruční elektrické nářadí např. úhlovou brusku do průměru kotouče 125 mm, přímočarou pilu atd. Neznepokojuje se hlukem.
RC6	20 min	Velmi zkušený zloděj používá navíc dvouruční elektrické nářadí např. úhlovou brusku do průměru kotouče 230 mm, přímočarou pilu atd. Neznepokojuje se hlukem.

Tabulka 2.1: Tabulka bezpečnostních tříd definovaných normou *ČSN EN 1627* [14]

2.2 Trubky a fitinky

Trubky a fitinky, které slouží ke spojování trubek, jsou učené především pro transport kapalin a plynů. Ale občas své uplatnění naleznou i v jiných oblastech viz obrázek 2.2.

Obrázek 2.2: Trubky mají různé využití – zleva postel, zábradlí, garnýž, stůl. (Převzato z [11], [18], [23], [27])

Pozinkované trubky a fitinky se nejčastěji vyrábějí žárovým zinkováním ocelových kusů, jedná se o techniku pokovování ponorem do roztaveného kovu. Takto povrchově upravené ocelové trubky a fitinky získavají odolnost vůči korozi a větší odolnost proti mehcanickému poškození. Velikost trubek udává Jmenovitá Světlost JS neboli přibližný vnitřní průměr trubky. Nejčastěji se Jmenovitá Světlost označuje jako DN (Diameter Nominal) v milime-

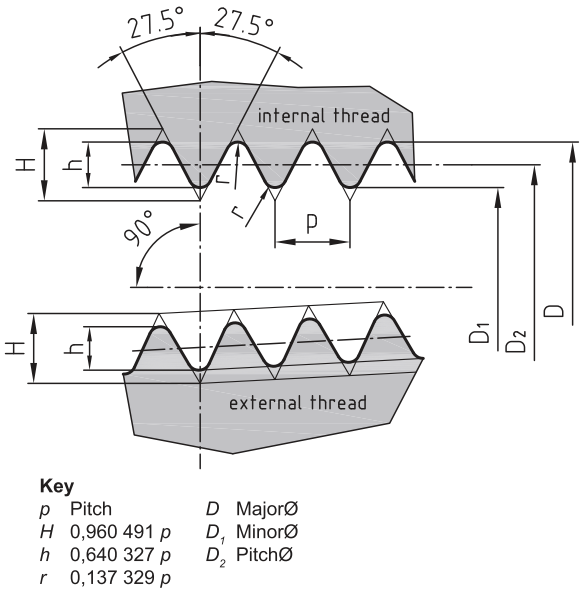
trech. Jmenovité světlosti jsou definovány normou *ČSN EN ISO 6708 (130015) – Potrubní části. Definice a výběr jmenovitých světlostí. DN*. V komerčním sektoru se velikosti trubek a fitinek běžně označují v palcích, převodní tabulku můžete vidět zde 2.2.

DN (mm)	” (palce)	DN (mm)	” (palce)	DN (mm)	” (palce)
6	1/8	32	5/4	125	5
8	1/4	40	6/4	150	6
10	3/8	50	2	200	8
15	1/2	65	2 1/2	250	10
20	3/4	80	3	300	12
25	1	100	4	350	14

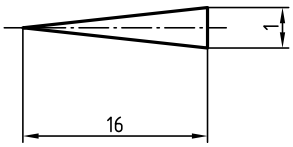
Tabulka 2.2: Trubky převodů DN na palce. (Převzato z [10])

2.2.1 Závity

Spojování šroubováním umožňuje mezinárodně standardizovaný závit pro šroubové spojování trubek s pomocí fitinek. Profil závitů vychází z Whitworthova závitu, parametry závitů jsou popsány normou *ČSN ISO 7 – Trubkové závity těsnící na závitech*. Hlavní rozměry závitů jsou znázorněny na obrázku 2.3. Vnější závit se zúžuje směrem ke konci trubky, to umožňuje jednoduché našroubování a dotažení, poměr zúžování je 1:16 a je znázorněn na obrázku 2.4.



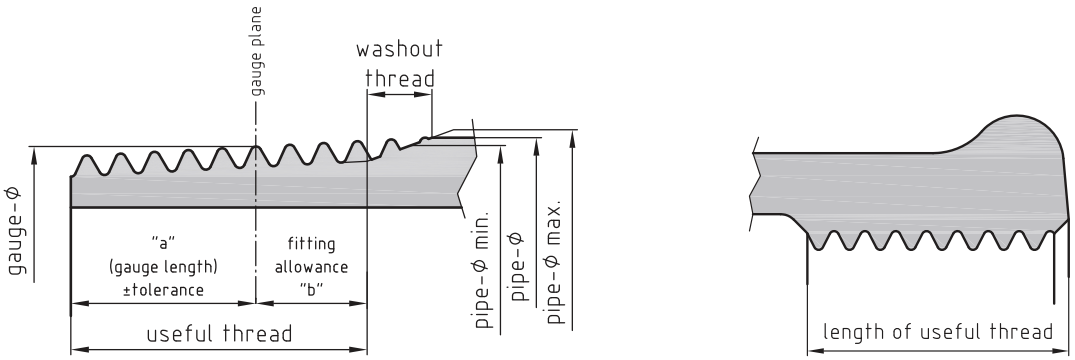
Obrázek 2.3: Hlavní rozměry vnitřního a vnějšího závitu. (Převzato z [1])



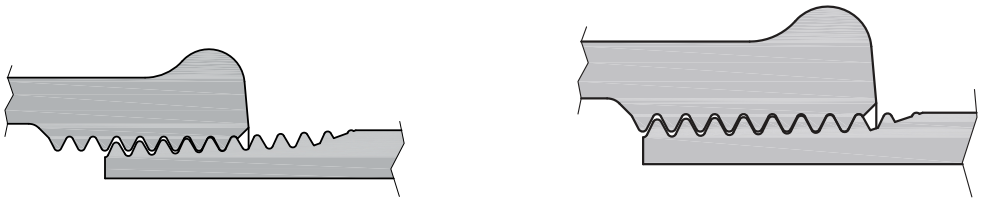
Obrázek 2.4: Poměr zúžení vnějšího závitu je 1:16. (Převzato z [1])

Vnější závit má 2 hlavní části, znázorněné v levé části obrázku 2.5. Část *"a"*, jejíž délka je dána normou, musí umožnit snadné zašroubování do vnitřního závitu, v levé části obrázku 2.6 je vidět dotažení části *"a"* rukou bez použití nástrojů. Část *"b"*, vnějšího závitu, je kritickou pro těsnost spoje. Pro její zašroubování a dotažení je nutné využít nářadí, v pravé části obrázku 2.6 je znázorněno dotažení s využitím nářadí. Takové žašroubování by mělo zajistit dostatečný tlak mezi závity, který zaručí těsnost spoje. Délky hlavních částí vnějších závitů jsou pro nejběžnější velikosti pozinkovaných trubek vypsány v tabulce 2.3.

Vnitřní závit, znázorněn v právé části obrázku 2.5, se žádným směrem nezúžuje a jeho délka musí umožnit plné zašroubování obou hlavních částí "*a*" a "*b*" vnějšího závitu.



Obrázek 2.5: Obrázek vlevo popisuje části vnějšího závitu a obrázek vpravo popisuje části vnitřního závitu. (Převzato z [1])



Obrázek 2.6: Obrázek vlevo znázorňuje dotažení závitu rukou a obrázek vpravo dotažení s použitím nářadí. (Převzato z [1])

Velikost trubky (palce)	1/2	3/4	1	5/4
Přibližná délka vnějšího závitu trubky (*mm*)	13,0	15,0	17,0	19,0
Délka části "a" vnějšího závitu trubky (*mm*)	8,2	9,5	10,4	12,7
Délka části "b" vnějšího závitu trubky (*mm*)	5,0	5,0	6,4	6,4
Vnější průměr trubky (*mm*)	21,3	26,9	33,7	42,4
Vnitřní průměr trubky (*mm*)	16,1	21,7	27,3	36,0
Tloušťka stěny trubky (*mm*)	2,6	2,6	3,2	3,2

Tabulka 2.3: Důležité rozměry běžně dostupných celikostí pozinkovaných trubek, podle norem *ČSN ISO 7* a *ČSN EN 10255*.

2.2.2 Trubky

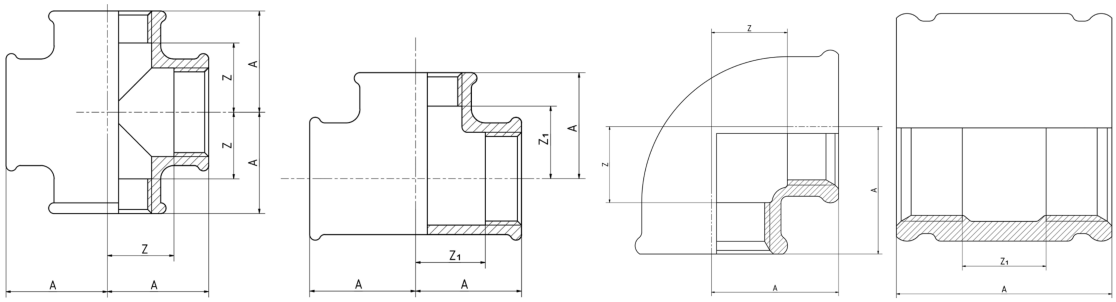
Parametry trubek jsou definované normou *ČSN EN 10255 – Trubky z nelegované oceli vhodné ke sváření a řezání závitů – Technické dodací podmínky*. Důležité rozměry běžných

velikostí pozikovaných trubek jsou uvedeny v tabulce 2.3. Běžně dostupné délky trubek jsou [26] [13]:

- 1/2" trubka – 60 mm, 80 mm, 100 mm, 120 mm, 150 mm, 180 mm, 200 mm, 300 mm, 500 mm, 1000 mm, 1500 mm, 2000 mm
- 3/4" trubka – 60 mm, 80 mm, 100 mm, 120 mm, 150 mm, 180 mm, 200 mm, 300 mm, 500 mm, 1000 mm, 1500 mm, 2000 mm
- 1" trubka – 60 mm, 80 mm, 100 mm, 120 mm, 150 mm, 180 mm, 200 mm, 300 mm, 500 mm, 1000 mm, 1500 mm, 2000 mm
- 5/4" trubka – 60 mm, 100 mm, 150 mm, 200 mm, 500 mm, 1000 mm, 1500 mm

2.2.3 Fitinky

Fitinky jsou armatury sloužící ke spojování trubek. Jejich parametry jsou dané normou *ČSN EN 10242 – Fitinky z temperované oceli s trubkovými závity*. Výkresy základních fitinek (kříž, T-kus, koleno a nátrubek), s označenými hlavními rozměry, jsou na obrázku 2.7. Tabulka 2.4 obsahuje rozměry základních fitinek v běžně dostupných velikostech.



Obrázek 2.7: Výkresy základních typů fitinek - zleva kříž, T-kus, koleno, nátrubek. (Převzato z [2])

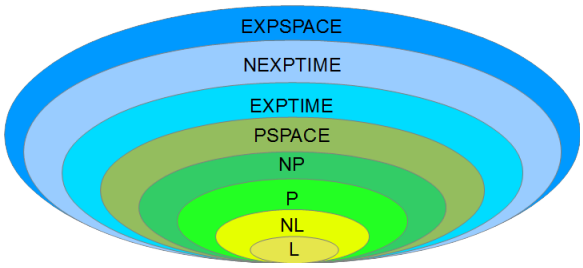
Velikost (palce)	Kříž, T-kus, Koleno		Nátrubek	
	A (mm)	Z/Z1 (mm)	A (mm)	Z1 (mm)
1/2	24,5	13,5	31	9
3/4	29	16,5	36	11
1	35	20,5	41	12
5/4	42	26	47	15

Tabulka 2.4: Rozměry základních typů fitinek pro běžně dostupné velikosti fitinek.

2.3 Třídy složitosti a NP-úplné problémy

Třídami složitosti se zabývá teorie složitosti, která se zaměřuje na výpočetní problémy v kontextu jejich vzájemné složitosti, nejčastěji parametry složitosti bývá čas nebo paměť, které počítač potřebuje pro vyřešení instance problému. Výpočetním problémem chápeme úlohu, kterou lze vyřešit pomocí algoritmu, počítač potom s využitím algoritmu řeší konkrétní instanci problému. Instance problému je konkrétní zadání daného problému neboli

problém je nekonečná množina instancí a jejich řešení. Středem zájmu zkoumání tříd složitosti jsou takzvané rozhodovací problémy, jejichž řešením je *"ano"* nebo *"ne"*. Základní třídy složitosti jsou znázorněny na obrázku 2.8.



Obrázek 2.8: Diagram základních tříd složitosti. (Převzato z [25])

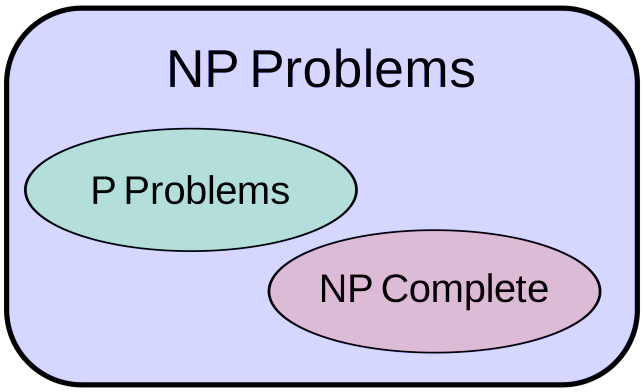
Problém návrhu mříže spadá do třídy složitosti NP. Pro jednodušší pochopení se nejprve zaměřím na třídu složitosti P. Ta obsahuje všechny problémy, které jsou řešitelné v polynomiálně omezeném čase na deterministickém Turingově stroji DTS. DTS je teoretický model počítače, který byl vymyšlen Alanem Turingem v roce 1936, DTS se skládá z neomezené lineární pásky složené z buněk, kdy každá buňka může v jakémkoliv okamžiku obsahovat přesně jeden znak přijmané abecedy. DTS potom provádí kroky, při kterých čte znak abecedy v aktuální buňce, případně ji přepíše, a posunout se o jednu buňku vpravo nebo vlevo. Jakou akci DTS provede je udáváno konečným automatem. To znamená, že doba řešení instance takového P problému je, v nejhorším případě, omezena asymptoticky shora polynomem – $O(p(n))$, kde p je polynom a n je velikost vstupních dat. Pro počítač je typicky jednoduché najít i ověřit správnost řešení.

NP potom obsahuje problémy, které jsou řešitelné v polynomiálně omezeném čase na nedeterministikém Turingove stroji NTS – ten si můžeme představit jako DTS, který se dokáže v jakémkoliv okamžiku naklonovat a každý klon dokáže pokračovat jinou výpočetní cestou. To znamená, že můžeme v polynomiálním čase ověřit správnost řešení instance takového NP problému, ale nalezení řešení na běžném počítaci s pomocí algoritmu může trvat delší dobu, za předpokladu, že $P \neq NP$. Pokud by se ovšem $P = NP$, znamelo by to, že pro všechny problémy z NP lze najít řešení v polynomiálně omezeném čase. Takové zjištění by mělo vělký dopad na spoustu oborů, například v kryptografii se často spoléháme na to, že optimální řešení některých NP problémů trvá dlouhou dobu, ale jejich ověření je rychlé. Prozatím rovnost $P = NP$ dokázána ani vyvrácena nebyla a je jednou ze sedmi matematických problémů tisíciletí[1].

Nejsložitější problémy NP jsou označovány jako NP-úplné, mezi které patři takové NP problémy, na které jsou polynomiálně redukovatelně všechny ostatní NP problémy. Vztah mezi třídami složitosti NP-úplnosti, P a NP je zobrazen na obrázku 2.9 Prvním NP-úplným problémem byl problém splnitelnosti booleovské formule[2], který se ptá, zdali existuje takové ohodnocení proměnných dané booleovské formule, aby byla formule pravdivá. Důkaz provedl Stephen A. Cook ve své práci *The complexity of theorem – proving procedures*[8]. Brzy poté bylo u spousty NP problémů dokázáno, pomocí polynomiální redukce, že patří také do třídy složitosti NP-úplnosti. Dnes jsou známy tisíce problémů

[1] anglicky Millenium Prize Problems [5]
[2] anglicky Satisfability – SAT

řadících se mezi NP-úplné. Mezi nejznámější patří problém batohu[3], problém obchodního cestujícího[4] nebo například hledání hamiltonovské kružnice [4].



Obrázek 2.9: Diagram znázorňující vtah mezi problémy P, NP a NP-úplnými (NP-Complete). (Převzato z [7])

2.3.1 Problém rozměnění

Problém návrhu pravidelných mříží je blízký problému rozměnění[5], protože se na něj dá redukovat.

Problém rozměnění vychází z problému batohu, který je následující – je dán batoh, který má nosnost t, a n předmětů, každý má hmotnost h a cenu c. Cílem je najít kombinaci předmětů, kterou lze dát do batohu, aby nebyla překročena jeho nosnost, a zároveň měla v součtu co nejvěší cenu. Předmět můžeme buďto celý dát do batohu nebo celý nedat do batohu, není možné dělení předmětů na menší části. Pokud předměty očíslujeme od 1 do n a přidáme vektor binárních hodnot $x_i, i \in \{1, .., n\}$, který má následující význam:

$$x_i = \begin{cases} 1 & \text{když je předmět } i \text{ dán do batohu;} \\ 0 & \text{jinak} \end{cases}$$

Můžeme potom problém batohu formálně zapsat takto[6]:

$$\begin{aligned} \text{maximalizuj} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{pro} \quad & \sum_{i=1}^{n} h_i x_i \leq t \end{aligned}$$

Problém rozměnění je potom následující – je dána částka t a n hodnot mincí h, od každé hodnoty je k dispozici neomezené množství mincí. Cílem je najít takovou kombinaci mincí, aby její cena byla v součtu rovna zadané částce a zároveň obsahovala co nejmenší množství

[3] anglicky Knapsack problem
[4] anglicky Travelling Salesman Problem
[5] anglicky Change making problem
[6] Inspirováno [19]

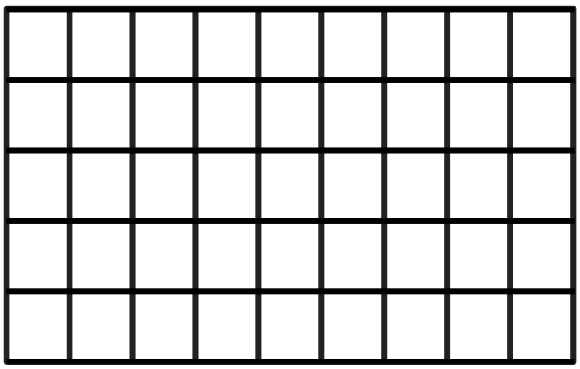
mincí. Pokud hodnoty mincí očíslujeme od 1 do n a přidáme vektor $x_i \in \mathbb{N}_0, i \in \{1, .., n\}$, kde hodnota x_i znamená počet mincí hodnoty h_i, můžeme problém formálně zapsat takto:

$$\begin{aligned} \text{minimalizuj} \quad & \sum_{i=1}^{n} x_i \\ \text{pro} \quad & \sum_{i=1}^{n} h_i x_i = t \end{aligned}$$

Pokud porovnáme problém rozměnění a problém batohu, tak zjistíme, že problém rozměnění je speciálním případem problému batohu, kdy máme od každého předmětu neomezené množství kusů a zároveň všechny předměty mají hodnotu 1 ($c_i = 1$) a místo maximální hodnoty hledáme hodnotu minimální.

Oba problémy jsou NP-úplné, důkaz provedl v roce 1975 George S. Lueker a je k nahlédnutí v knize *Two NP-complete Problems in Nonnegative Integer Programming* [17].

Problém návrhu mříží můžeme převést na problém rozměnění. Pokud konstrukci navrhovaných mříží omezíme pouze na takovou, která je pravidelná neboli trubky tvoří pravoúhlou síť a ve všech řadách i sloupcích vedou trubky po célé délce, viz obrázek 2.10, zanedbáme všechny rozměry dílů, kromě délky trubek a křížení sloupců s řádky bude na obou koncích každé trubky. Tak získáme konstrukci, kde se posloupnost trubek v každém sloupci i v každém řádku opakuje. Pro návrh mříže v takovém případě stačí nalézt takové trubky, jejichž celková delka je rovna výšce mříže a trubky, jejichž celková délka je rovna šířce mříže. Návrh konstrukce takové mříže potom bude mít počet sloupců trubek roven počtu trubek potřebných pro sestavení šířky mříže +1, obdobně počet řádků trubek bude roven počtu trubek potřebných pro výšku +1.



Obrázek 2.10: Pravidelná mříž, tvoří pravoúhlou síť, kde v každém řádku i v každém sloupci vedou trubky po celé délce.

Formálně můžeme problém nalezení trubek potřebných pro sestavení délky strany mříže zapsat takto – pokud délku strany mříže označíme t a máme dáno n délek trubek, kdy od každé délky h máme neomezený počet kusů použitelných pro návrh mříže, potom očíslujeme jednotlivé délky trubek od 1 do n a přidáme vektor $x_i \in \mathbb{N}_0, i \in \{1, .., n\}$, kde hodnota x_i znamená počet trubek délky h_i, potom je řešením vektor x_i pro který platí:

$$\sum_{i=1}^{n} h_i x_i = t$$

Pokud bychom chtěli nalézt řešení, které obsahuje nejmenší možné množství trubek, potom by formální zápis vypadal takto:

$$\begin{aligned} \text{minimalizuj} \quad & \sum_{i=1}^{n} x_i \\ \text{pro} \quad & \sum_{i=1}^{n} h_i x_i = t \end{aligned}$$

Nyní můžeme porovnat problém rozměnění s problémem nalezení trubek potřebných pro sestavení délky strany mříže. Zjistíme, že jsou ekvivalentní. Délky trubek odpovídají hodnotám mincí a délka strany mříže odpovídá částce, kterou je potřeba rozměnit. Z toho vyplývá, že problém návrhu pravidelné mříže je také NP-úplným problémem.

2.3.2 Řešení NP-úplných problémů

Jelikož nalezení nejlepšího nebo všech přesných (optimálních) řešení instance NP-úplného problému pomocí hrubé síly[7], to znamená ověřením všech potenciálně řešení, je použitelné pouze pro velmi malá vstupní data, protože by jinak hledání řešení trvalo přílis dlouho. Tímpádem se musí přistoupit k sofistikovanějším metodám výpočtu řešení. Ty například zmenší množinu prohledávaných řešení tím, že vyloučí řešení, která určitě nebudou optimálním řešením. Nebo se například spokojí s prvním řešením které splňuje nějakou podmínku (prahovou hodnotu) a je proto dostatečně dobré, abychom se s ním mohli spokojit. Takové řešení ovšem nemusí být optimální.

Algoritmy pro řešení NP-úplných problémů můžeme rozdělit do 2 skupin:

- Exaktní – to jsou takové algortimy, které naleznou globální optimální řešení, pokud existuje. Jsou tedy deterministické a můžeme mezi ně zařadit například *metodu větví a mezí*[8] nebo *dynamické programování*[9].
- Heuristické – ty se snaží v krátkém čase odhadnout co nepřesněji řešení instance problému. Správnost řešení není možné dokázat, ale i přes to jsou často používané pro problémy neřešitelné v polynomiálně omezeném čase. Mezi heuristické algoritmy patří například *hladový algoritmus*[10], *genetické algoritmy*[11] nebo *horolezecký algoritmus*[12].

Hladový algoritmus

Hladový algoritmus je příkladem heuristických algoritmů, který se v každém kroku snaží jít nejlepší možnou cestou pro daný krok. Důsledkem toho je nalezení řešení v polynomiálně omezeném čase, ale není zaručeno, že je nalezené řešení globálně optimální, často bývá pouze lokálně optimální.

Algoritmus pro problém rozměnění by byl následující:

1. Seřadíme hodnoty mincí sestupně.

[7] anglicky Brute force
[8] anglicky Branch and bound (B&B)
[9] anglicky Dynamic programming
[10] anglicky Greedy algorithm
[11] anglicky Genetic algorithms
[12] anglicky Hill-climbing

2. Postupně procházíme přes hodnoty sešazených mincí.

    - V každém kroku se snažíme od zbylé částky odečíst co největší počet aktuálních mincí tak, abychom získaly zbytek ≥ 0. Počet odečtených mincí a jejich hodnoty si zapamatujeme. Pokud je zbytek roven 0, tak jsme nalezli řešení, jinak pokračujeme dalším krokem, pokud již další hodnotu mince k dispozici nemáme, tak algoritmus řešení nenalezl.

- Asymptotická složitost tohoto algoritmu je $O(n)$, kde n je počet hodnot mincí, které máme k dispozici.

Příklad s konkrétními hodnotami – mějme dány mince hodnot {1,5,10} a částku 56:

1. Seřadíme hodnoty mincí – {10,5,1}
2. Projdeme seřazené mince:
    - Hodnota aktuální mince je 10 a zbytek 56, od 56 můžeme 10 odečíst 5x, nový zbytek je 6, $6 \neq 0$, takže pokračujeme na další hodnotu mince.
    - Hodnota aktuální mince je 5 a zbytek 6, od 6 můžeme 5 odečíst 1x, nový zbytek je 1, $4 \neq 0$, takže pokračujeme na další hodnotu mince.
    - Hodnota aktuální mince je 1 a zbytek 1, od 1 můžeme 1 odečíst 1x, nový zbytek je 0, $0 \neq 0$, našli jsme řešení.
3. Nalezené řešení je 7 mincí a jejich hodnoty jsou {10,10,10,10,10,5,1}.

V tomto příkladě hladový algoritmus nalezl optimální řešení, pokud bychom ovšem přidali další druh minci s hodnotou 9, hladový algoritmus by nalezl stejné řešení jako v předchozím příkladě, optimálním řešením by, ale bylo 6 mincí s hodnotou 9.

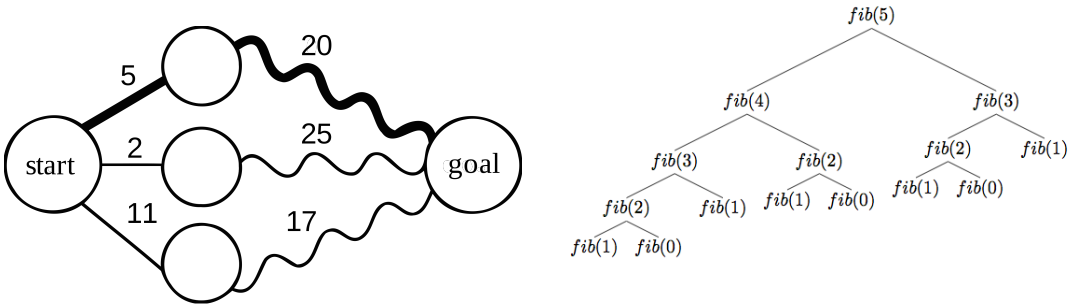
Dynamické programování

Dynamické programování je příkladam exaktního deterministického aalgoritmu. Jedná se o optimalizační techniku řešení problémů, které obsahují opakující se podproblémy. Zakladní princip je takový, že si algoritmus pamatuje řešení již vyřešených podproblémů a pokud je potřeba vyřešit problém skládající se z podproblémů, z nichž již byly některé vyřešeny předtím, tak se řešení těchto podproblému vezme z paměti a nehledá se znovu.

Problém, který je řešitený pomocí dynamického programování musí mít nálsedující vlastnosti:

- Optimální strukturu – to znamená, že optimální řešení problému může být vypočítáno z optimálních řešení podproblémů. Příkladem může být nalezení nějkratší cesty v grafu, viz levá část obrázku 2.11,kde známe řešení podproblémů (vlnité hrany) a chceme zjistit délku cesty z uzlu *start* do uzlu *goal*.

- Překrývající se podproblémy – problém může být rozdělen na podproblémy, kdy se některé z nich opakují. Příkladem může být Fibonacciho posloupnsost, viz pravá část obrázku 2.11, který znázorňuje vytovoření páteho člene Fibonacciho posloupnosti, můžeme vidět, že některé podproblémy (uzly) se opakují.



Obrázek 2.11: Levá část obrázku demonstruje optimální strukturu, znázorněním nalezení optimální cesty v grafu. Pravá část obrázku demonstruje překrývající se podproblémy, znázorněním stromové struktury vytvoření pátého člene Fibonacciho posloupnosti. (Převzato z [6]) a [24])

Algoritmus pro problém rozměnění, kdy hledáme nejmenší množství mincí, ze kterých lze částku sestavit, by byl následující:

1. Vytvoříme pole p o délce rovné částce $+1$, kde index pole i bude symbolizovat částku a hodnota prvku pole na daném indexu $p[i]$ bude počet mincí potřebných k sestavení částky.

2. Pole inicializujeme, první prvek pole $p[0]$ bude mít hodnotu 0, ostatní budou mít hodnotu ∞.

3. Následně pole p projdeme od indexu 0 až do konce.

4. V každém kroku i zkusíme od částky, kterou symbolizuje (index), postupně odečíst hodnoty všech mincí, vypočítanou hodnotu označme h, pokud $h \geq 0$, tak se podíváme na prvek pole na indexu h, pokud hodnota tohoto prvku pole $pole[h] + 1$ (označme ji x) je menší než hodnota prvku, který právě zpracováváme $pole[i]$, tak jeho hodnotu přepíšeme na x ($pole[i] = x$).

5. Nakonec přečteme hodnotu posledního prvku pole, pokud bude rovna ∞, tak řešení neexistuje, jinak je řešením hodnota tohoto prvku pole.

- Asymptotická složitost tohoto algoritmu je $O(Cn)$, kde C je částka a n je počet hodnot mincí, které máme k dispozici.

Příklad s konkrétními hodnotami – mějme mince o hodnotách 1 a 3, jaké je nejmenší množství mincí, ze kterých můžeme složit částku 4? Postup výpočtu algoritmu můžeme znázornit tabulkou 2.5. Nalezené řešení je 2 a můžeme o něm s jistotou prohlásit, že se jedná o optimální řešení.

krok (i)	$p[0]$	$p[1]$	$p[2]$	$p[3]$	$p[4]$	komentář
init	0	∞	∞	∞	∞	výchozí hodnoty pole po inicializaci
0	0	∞	∞	∞	∞	v kroku 0 od hodnoty 0 nemůže odečíst žádnou z hodnot mincí, abychom dostali číslo ≥ 0
1	0	1	∞	∞	∞	od 1 můžeme odečíst pouze 1, $1-1=0$, na indexu 0 máme v poli hodnotu 0, $0+1<\infty$, takže hodnotu aktuálního prvku $p[1]$ přepíšeme
2	0	1	2	∞	∞	od 2 můžeme odečíst pouze 1, $2-1=1$, na indexu 1 máme v poli hodnotu 1, $1+1<\infty$, takže hodnotu aktuálního prvku pole $p[2]$ přepíšeme
3	0	1	2	3	∞	od 3 můžeme odečíst 1 a 3, nejpreve zkusíme odečíst 1, $3-1=2$, na indexu 2 máme v poli hodnotu 2, $2+1<\infty$, takže hodnotu aktuálního prvku pole $p[3]$ přepíšeme
3	0	1	2	1	∞	nyní zkusíme odečíst 3, $3-3=0$, na indexu 0 máme v poli hodnotu 0, $0+1<3$, takže hodnotu aktuálního prvku pole $p[3]$ přepíšeme
4	0	1	2	1	2	od 4 můžeme odečíst 1 a 3, nejprve zkusíme odečíst 1, $4-1=3$, na indexu 3 máme v poli hodnotu 1, $1+1<\infty$, takže hodnotu aktuálního prvku pole $p[4]$ přepíšeme,
4	0	1	2	1	2	potom zkusíme odečíst 3, $4-3=1$, na indexu 1 máme v poli hodnotu 1, $1+1\not<2$, takže hodnotu aktuálního prvku pole $p[4]$ nezměníme
řešení	0	1	2	1	2	hodnota posledního prvku pole je 2, $2\neq\infty$, takže řešení existuje a je rovno 2

Tabulka 2.5: Příklad výpočtu problému rozměnění pomocí dynamického programování, kdy chceme zjistit nejmenší možný počet mincí, ze kterých lze složit částku 4, k dispozici máme mince s hodnotami 1 a 3.

Kapitola 3

Návrh algoritmu pro generování mříží

Shrnutí teorie
Během rešeršování se mi nepodařilo objevit žádnou práci zabývající se návrhem mříží ze zadaných dílů. Provedl jsem tedy průzkum podobných problémů a podařilo se mi zařadit problematiku návrhu mříží do třídy NP-úplných problémů. Toto zařazení jsem následně ověřil tranformováním problému návrhu mříží na problém rozměnění, který je známým NP-úplným problémem.

3.1 Definování přesného zadání

Nezbytným krokem před samotným návrhem algoritmu je definování přesného zadání, které určí, jaké budou vstupy a výstupy algoritmu.

Zadání v bodech

- Algoritmus, na základě vstupních uživatelských požadavků na mříž, vygeneruje, pokud je to konstrukčně možné, všechny nejlepší konstrukce pravidelných mříží (2.10) splňující uživatelské požadavky.
- Uživatelskými požadavky na mříž jsou:
  - Maximální výška a šířka mříže – algoritmus se vždy bude snažit navrhnout co největší mříž do zadaných rozměru.
  - Maximální výška a šířka otvorů mříže – otvory mříže nemusí být všechny stejně velké, ale nesmí překročit zadané rozměry.
  - Použití trubek s danými parametry – seznam délek a cen trubek, které je možné pro návrh mříže použít, vnější průměr a délka závitu "a" + "b", viz 2.5.
  - Použití fitinek s danými parametry – cena všech fitinek, rádius A, viz 2.7, a vnější průměr u fitinek kříž, T-kus a koleno (měly by být pro všechny typy fitinek stejné), délka A fitinky nátrubek, viz 2.7.
  - Zdali řešení musí nebo nemusí mít co nejmenší počet děr.
  - Preference nejnižší ceny nebo mejmenšího počtu dílů, případně ani jednoho – tento požadavek má menší prioritu než požadavek na nejmenší množství děr.

- Algoritmus předpokládá, že všechny trubky je možné zašroubovat do všech fitinek.
- Nejlepší řešení bude takové, které vyhoví uživatelským požadavkům.
- Uživatelské rozhranní umožní zadání požadavků na mříž a následně zobrazí nalezená řešení.

3.2 Odhalení jádra problému

Problém návrhu pravidelné mříže je NP-úplným problémem, jak již bylo zmíněno v teoretické části 2.3, pro návrh řešení, splňující zadání, je však potřeba problému porozumět více do hloubky, a proto ho nyní rozeberu.

V teoretické části 2.3 jsem ukázal, že problém návrhu pravidelné mříže je možné řešit nalezením nejmenší monžné množiny trubek potřebných k sestavení výšky mříže a nejmenší možné množiny trubek potřebné k sestavení šířky mříže. Následně můžeme návrh mříže vytvořit tak, že posloupnost trubek šířky mříže použijeme pro všechny řádky mříže a posloupnost trubek výšky mříže použijeme pro všechny sloupce mříže. Získáme tak mříž, kde na obou koncích každé trubky bude křížení řádku a sloupce.

Takové řešení není optimální z pohledu zadání, které vyžaduje, aby byl počet otvorů mříže co nejmenší. To znamená, že strany některých otvorů mohou být tvořeny více trubkami. Abychom takovou mříž získali s využitím stejného postupu, tak nejprve musíme zjistit, jaké všechny délky složených trubek[1] můžeme ze zadaných trubek vytvořit do délky dané maximální velikostí otvoru pro šířku a výšku mříže. Takové trubky následně použijeme pro hledání nejmenších možných množin trubek potřebných k sestavení šířky a výšky mříže (zdůraznění – pro sestavování šířky mříže potřebujeme jinou množinu složených trubek než pro sestavování výšky mříže, protože velikost otvoru může být v každém rozměru jiná). Při následném vytváření návrhu mříže bude křížení řádků a sloupců na konci každé složené trubky (místo na konci každé trubky, jak tomu bylo v předchozím případě).

Z toho vyplývá, že problém návrhu pravidelné mříže můžeme rozdělit na dva podproblémy:

1. Vytvoření seznamů složených trubek pro šířku a výšku mříže.
2. Vytvoření co nejmenších množin trubek potřebných pro sestavení výšky a šířky mříže.

Problém vytvoření seznamu trubek můžeme rozložit na více podproblémů, kdy každým podproblém bude nalezení nejmenší možné množiny trubek potřebných k sestavení složené trubky dané délky. Tyto problémy budeme mít pro všechny délky do maximální délky složené trubky (maximálního rozměru otvoru pro šířku a výšku mříže).

Všechny problémy, nalezení množin trubek pro složené trubky a nalezení množin složených trubek pro šířku a výšku mříže, lze řešit stejným algoritmem. Takže primárním cílem je vytvořit algoritmus, který dokáže nalézt nejmenší možnou množinu trubek, ze kterých lze sestavit složenou trubku zadané délky.

3.3 Návrh algoritmu

Na základě zadání bakalářské práce, kde je kladen důraz na maximální zohlednění uživatelských požadavků, jsem se rozhodl pro návrh exaktního algoritmu pro návrh pravidelných

[1] Složenou trubkou chápeme trubku, která je složena z 1 a více trubek spojených fitinkou nátrubek

mříží, aby byl schopen nalézt optimální řešení. Přestože takový algoritmus bude pro velká vstupní data v podstatě nepoužitelný, protože by výpočet trval příliš dlouho. Na druhou stranu je ale nutné vzít v potaz účel algoritmu, kterým je návrh mříží, a dá se tedy předpokládat, že vstupní data budou dostatečně rozumně malá, aby bylo možné nalézt optimální řešení pomocí exaktního algoritmu v přiměřeném čase.

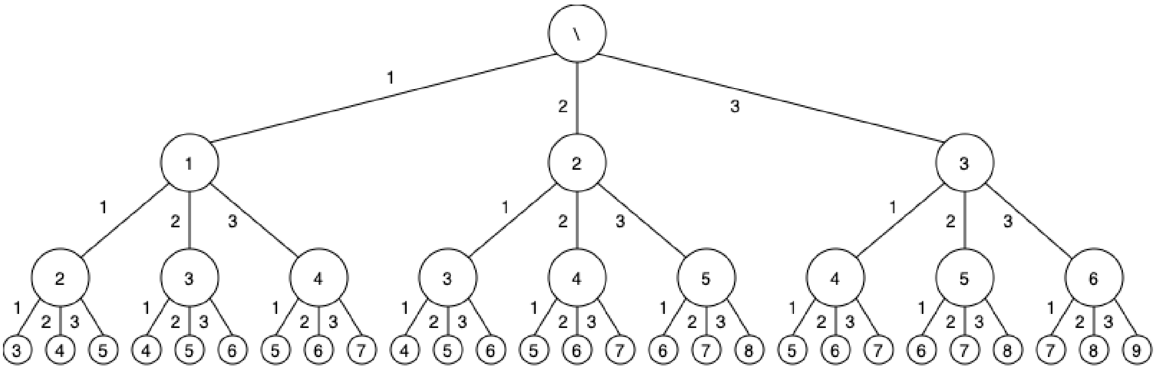
Naivní algoritmus – hrubou silou
Nejjednoduší a zároveň nejnaivnější algoritmus uvažuje pouze délku trubek, ostatní rozměry dílů zanedbává, a projde všechny možné variace trubek, jejižch součet délek je menší nebo roven hledané délce. Vrátí všechna nejlepší nalezená řešení. Takový algoritmus může vypadat následovně:

1. Na začátku existuje fronta nezpracovaných složených trubek, kde je pouze složená trubka délky 0
2. Algoritmus postupně zpracovává frontu nezpracovaných složených trubek:
   - Aktuální složenou trubku postupně zkusí prodloužit o všechny délky trubek, které má pro skládání složené trubky k dispozici.
   - Pokud je nově vzniklá prodloužená trubka menší než hledaná složená trubka, tak se zařadí mezi nezpracované složené trubky do fronty. Pokud je rovna délce hledané složené trubky, tak je přidána na seznam potenciálních řešení. Jinak se zahodí.
3. nakonec projdeme všechna potenciální řešení a vybereme ta, které jsou z nejmenšího počtu trubek, pokud žádné potenciální řešení neexituje, tak ani žádné optimální řešení neexistuje.

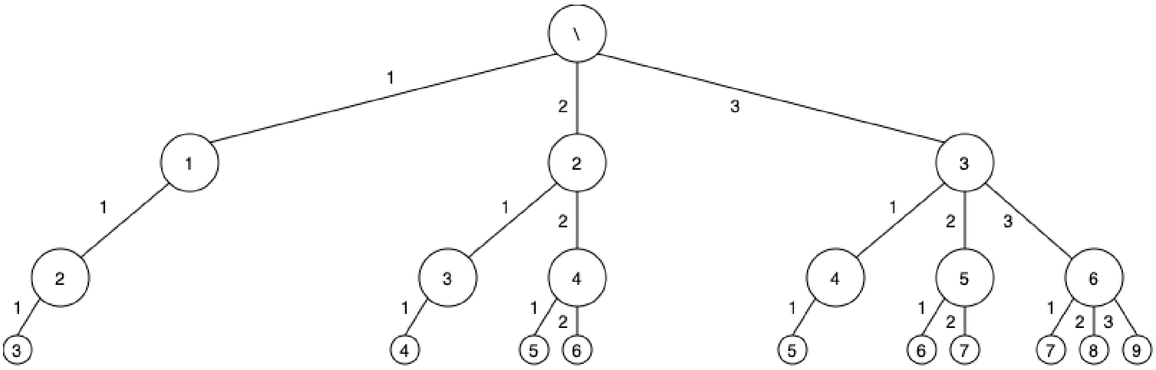
Odstranění variací
Jelikož naivní algoritmus prochází všechny variace, tak vzniká spoustu duplicitních podproblémů, které se musí opakovaně vyřešit. Například pokud budeme mít trubky délek 1,2 a 3, tak složené trubky, které naivní algoritmus vytvoří, by se daly vizualizovat pomocí stromu, jehož uzly reprezentují složenou trubku a nesou informaci o její délce, a kde seznam délek jednotlivých trubek, ze kterých je složená trubka stvořena, je reprezentován hodnotami hran cesty z uzlu složené trubky do kořenového uzlu. Na obrázku 3.1 je znázorněn takový strom do maximální hloubky 3.

Protože nám nezáleží na pořadí trubek, ze kterých jsou složené trubky sestavené, tak si můžeme rychle všimnout, že nekteré složené trubky vytvořené naivním algoritmem jsou stejné. Například $\{1, 1, 2\}, \{1, 2, 1\}$ a $\{1, 1, 2\}$.

Abychom se zbavili ekvivalentních řešení, tak stačí do algoritmu přidat podmínku, která omezí množinu trubek, kterými může algoritmus aktuální složenou trubku prodloužit. Tato podmínka říká, že aktuální složenou trubku lze prodloužit pouze trubkami, které jsou stejně dlouhé nebo menší než poslední trubka, kterou byla aktuální složená trubka vytvořena. Tímto zamezíme vytváření ekvivalentních cest stromu, každá cesta bude unikátní kombinací trubek. Strom se tedy změní a bude vypadat jako na obázku 3.2.



Obrázek 3.1: Strom reprezentující složené trubky vytvořené naivním algoritmem. hodnota uzlu je délka složené trubky a ohodnocení hran cesty z uzlu složené trubky do kořenového uzlu je seznamem délek trubek, ze kterých je složená trubka složena.



Obrázek 3.2: Strom reprezentující složené trubky vytvořené algoritmem, který vytváří pouze unikátní kombinace trubek.

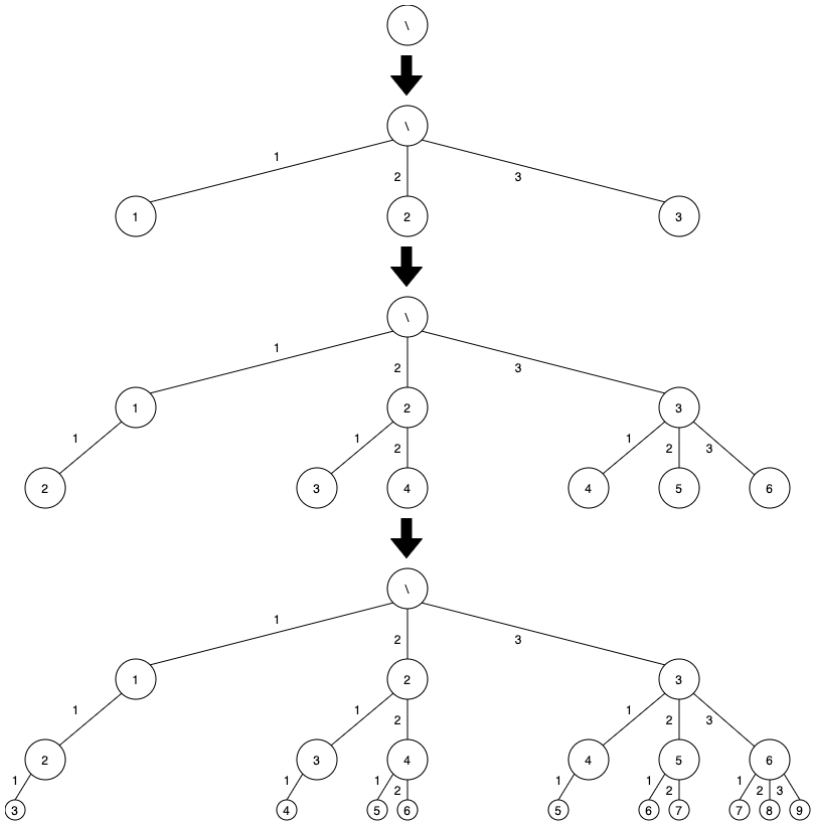
Odříznutí slepých větví

Když se zadíváme na strom na obrázku 3.2, tak vidíme, že některé uzly mají stejnou hodnotu, to znamená, že náš algoritmus vytváří různé složené trubky stejné délky, ale pouze některé z nich jsou součástí optimálních řešení pro délku hledané složené trubky, a i přes to je další zpracování (prodloužení) provedeno u vše. To znamená, že prodlužování některých trubek nemá smysl, protože z nich nikdy nebude optimální řešení, které by nevzniklo z jiné složené trubky. Abychom zpracování takových složených trubek mohli zabránit, tak nejprve algoritmus musí splnit následující dvě podmínky:

- Trubky, které máme k dispozici pro sestavení složené trubky, budou seřazeny sestupně a algoritmus bude aktuální složenou trubku prodlužovat trubkami od nejdelší po nejkratší. To nám zajistí, že budeme schopni s jistotou určit, který z uzlů se stejnou hodnotou již dál není potřeba a nevede k dalšímu optimálnímu řešení.
- Vytváření složených trubek musí probíhat do šířky, ne do hloubky, to znamená, že nejprve algoritmus poskládá složené trubky z 1 trubky, následně poskládá všechny složené trubky ze 2 trubek a tak dále. To můžeme zajistit postupným zpracováváním složených trubek, tak že použijeme FIFO frontu[2] pro nezpracované složené trubky

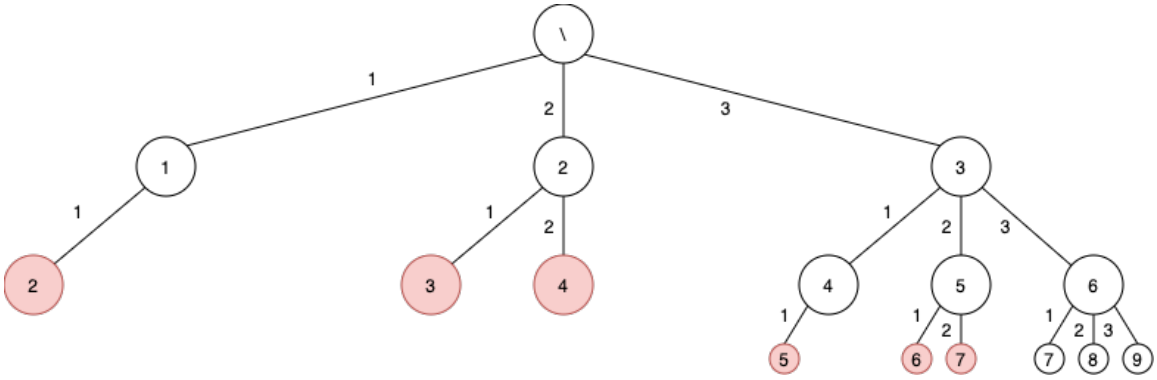
[2]z anglického first in first out neboli kdo dřív přišel, bude dřív obsloužen

nebo individuálními seznamy pro každý krok. Když se podíváme na obrázek stromu 3.2, tak v případě vytváření trubek metodou do šířky se bude strom generovat po úrovních, viz obrázek 3.3.



Obrázek 3.3: obrázek demonstruje proces vytváření stromu metodou do šířky.

Po splnění těchto podmínek můžeme přidat další podmínku, která zajistí ukončení prodlužování složených trubek, které nevedou k novým optimálním řešením – Pokud je prodloužením vytvořena složená trubka o stejné délce jako jakákoliv jiná doposud vytvořená složená trubka a pokud sama není optimálním řešením, tak nevede k novému optimálnímu řešení a dál se jí nemá smysl věnovat. Aplikace této podmínky v algoritmu bude mít za následek průchod složených trubek, které jsou interpretovány stromem na obrázku 3.4, kde jsou složené trubky zachycené touto podmínkou znázorněny červeně. (Poznámka – pokud bychom hledali složenou trubku délky 4, tak uzel délky 4, který je na obázku znázorněn červeně, by červený nebyl a byl by optimálním řešením.)



Obrázek 3.4: Obrázek červeně znázorňuje větve stromu, které nevedou k optimálnímu řešení a nemá smysl se nimi zabývat.

Sloučení dosavadních poznatků

Kombinace všech doposud zmíněných podmínek tvoří algoritmus, který prohledá nezbytný počet potenciálních optimálních řešení a nelezne všechna globální optimální řešení.

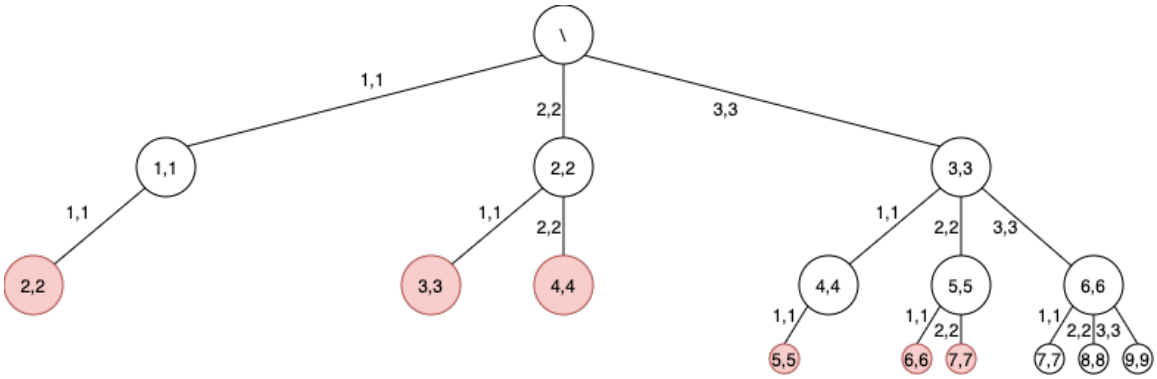
Algoritmus s kombinací všech podmínek vypadá takto:

1. Na začátku eixistuje seznam nezpracovaných složených trubek pro první krok, kde je pouze složená trubka délky 0, která vznikla přidáním trubky délky 0.
2. Trubky použitelné pro sestavení složené trubky se seřadí sestupně.
3. Algoritmus postupně zpracovává seznam nezpracovaných složených trubek pro daný krok:
   - Aktuální složenou trubku postupně zkusí prodloužit o všechny délky trubek, které má pro skládání složené trubky k dispozici a jsou stejně velké nebo menší než naposledy přidaná trubka, jejímž přidáním aktuální složená trubka vznikla.
   - Pokud je nově vzniklá prodloužená trubka menší než hledaná složená trubka a zároveň doposud nebyla složená trubka takové délky vytvořena, tak se zařadí na seznam nezpracovaných složených trubek pro následující krok. Pokud je rovna délce hledané složené trubky, tak ji zařadíme mezi optimální řešení. Jinak se zahodí.
   - Po prodloužení aktuální složené trubky, všemi přípustnými trubkami, algoritmus pokračuje na další trubku v seznamu nezpracovaných trubek pro aktuální krok. Pokud v seznamu další trubka není a již bylo nalezeno alespoň jedno optimální řešení, tak hledání končí a jsou nalezena všechna optimální řešení. Jinak pokud žádné takové řešení nalezeno nebylo, tak algoritmus pokračuje na zpracování seznamu nezpracovaných trubek pro další krok, pokud je tento seznam prazdný, tak řešení neexistuje.

Přidání ceny trubek

Důležitou roli může, při návrhu mříže, hrát cena. Uživatel kromě rozměrů dílů poskytne i jejich cenu a může si říci o nejlevnější návrh mříže splňující jeho požadavky. Cenu do stromu, kterým je vizualizováno hledání řešení pomocí navrhovaného algoritmu, přidáme jako druhou hodnotu hrany a uzlu, hrana potom bude nést informaci o délce přidávané

trubky a o její ceně, uzel potom bude tvořen celkovou délkou a cenou složené trubky. Strom se tedy změní a bude mít podobu jako na obrázku 3.5.



Obrázek 3.5: Obrázek zachycuje příklad stromu, který nese kromě informace o délce i informaci o ceně.

Pokud budeme hledat nejlevnější řešení, tak to nemusí být to, které je z nejméně dílů. Dokonce může existovat více nejlevnějších řešení s různým počtem dílů. Proto musíme změnit podmínku, která rozhoduje o tom, co se má stát po vytvoření nové prodloužené trubky. Na obrázku 3.6 jsou ukázány případy, kdy má nově vytvořený uzel stejnou délku, jako některý z dříve vytvořených uzlů. Tyto případy jsou nejkritičtější pro správné fungování algoritmu, proto je nyní popíši:

1. Nový uzel byl vytvořen ve stejném kroku a má menší cenu než již dříve vytvořený uzel se stejnou délkou. V takové případě je nový uzel lepší. To znamená, že dříve vytvořený uzel smažeme a nebudeme ho v přístím kroku zpracovávat. Nový uzel necháme a budeme ho v příštím kroku zpracovávat.

2. Nový uzel byl vytvořen ve stejném kroku a má větší cenu než již dříve vytvořený uzel se stejnou délkou. V takovém případě je nový uzel horší. To znamená, že nový uzel zahodíme. Starý uzel necháme a budeme ho v příštím kroku zpracovávat.

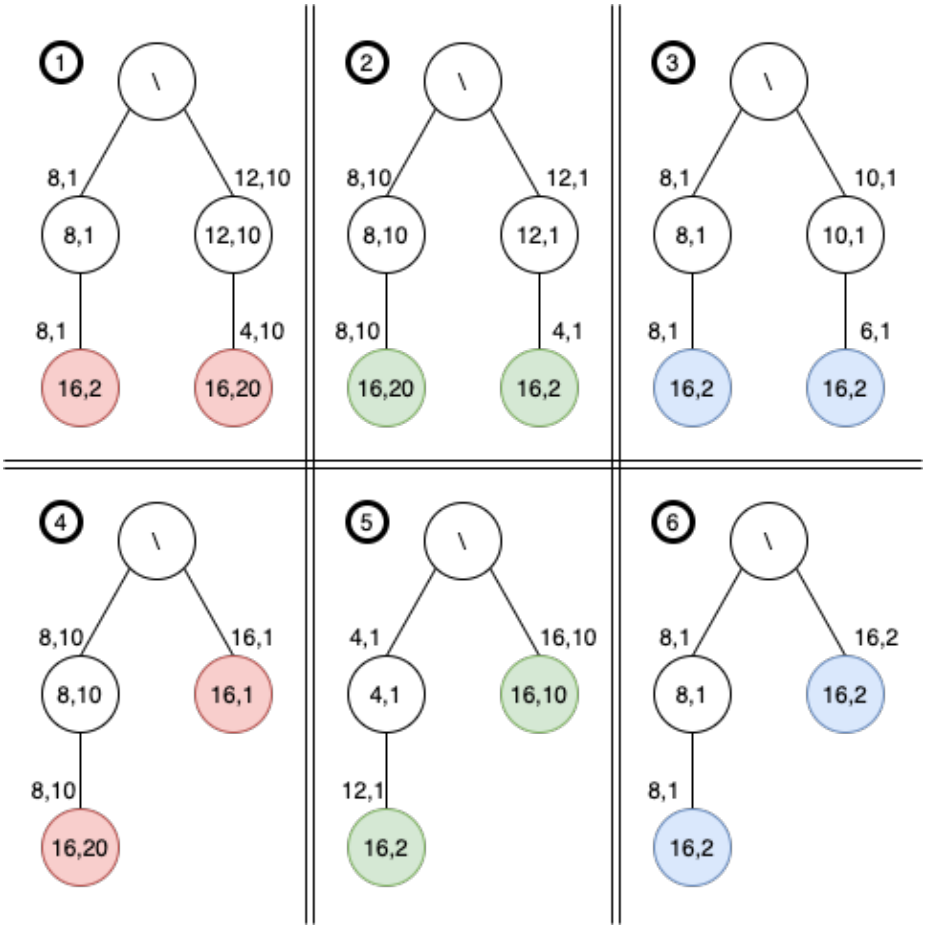
3. Nový uzel byl vytvořen ve stejném kroku a má stejnou cenu jako již dříve vytvořený uzel se stejnou délkou. V takovém případě je nový uzel stejně dobrý. To znamená, že necháme oba uzly a oba budeme v příštím kroku zpracovávat.

4. Nový uzel byl vytvořen v pozdějším kroku a má větší cenu než již dříve vytvořený uzel se stejnou délkou. V takovém případě je nový uzel horší. To znamená, že ho zahodíme. Se starým uzlem nic dělat nebudeme.

5. Nový uzel byl vytvořen v pozdějším kroku a má menší cenu než již dříve vytvořený uzel se stejnou délkou. V takovém případě je nový uzel lepší. To znamená, že ho necháme a budeme ho v příštím kroku zpracovávat. Se starým uzlem nic dělat nebudeme.

6. Nový uzel byl vytvořen v pozdějším kroku a má stejnou cenu jako již dříve vytvořený uzel se stejnou délkou. V takovém případě je nový uzel stejně dobrý. To znamená, že necháme oba uzly a nový uzel budeme v příštím kroku zpracovávat.

- Pokud ještě žádný z již dříve vytvořených uzlů neměl délku jako nově vytvořený uzel, tak nový uzel přídáme a v nasledujícím kroku ho zpracujeme..



Obrázek 3.6: Ukázky smyšlených podstromů pro nastínění kritických situací při vytváření nových uzlů a jejich zpracování.

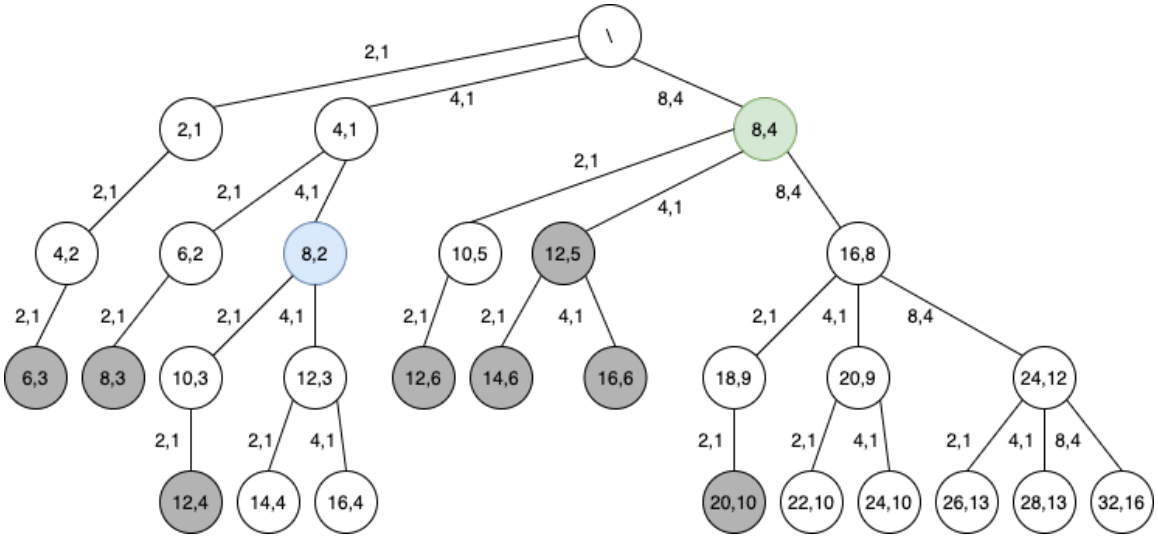
Pokud použijeme algoritmus vytvořený v minulém kroku, tak je nutné podotkou, že neoznačujeme za optimální řešení všechny uzly, jejichž délka je rovna zadané délce hledané složené trubky, ale pouze ty, které mají nejnižší cenu. To znamená, že průběžně řešení označujeme za potenciálně optimální řešení. Nekončíme v kroku, kdy bylo nalezeno první potenciálně optimální řešení, ale hledáme (prodlužujeme trubky) dokud ještě máme nějaký neprázdný seznam nezpracovaných složených trubek. Nakonec všechna potenciálně optimální řešení projdeme a porovnáme jejich cenu, následně za optimální prohlásíme ta nejlevnější.

Nyní díky parametru ceny můžeme jedním algoritmem vyhledávat ve 3 různých nastaveních:

- Nejlevnější řešení – hodnoty ceny jednotlivých dílů budou tak, jak byly zadány uživatelem, případně tak, jak byly vypočítány pro hledání složených trubek, pokud algoritmus navrhuje stranu mříže.
- Nejméně dílů – hodnoty ceny všech dílů budou rovny 1, tím pádem bude cena složené trubky sloužit jako počítadlo dílů, ze kterých je složena, a protože algoritmus hledá nejlevnější složené trubky, tak najde složené trubky z nejméně dílů.

- Je to jedno – zajímají nás všechna řešení. Hodnoty všech dílů budou rovny 0, algoritmus se tímpáden nebude řídit cenou ani počtem trubek.

Chování pro jednotlivá nastavení je ukázáno obrázkem 3.7, kde je znázorněna strom do hloubky 5 pro modelovou situace s trubkami délek 2, 4, 8 a s cenami 1, 1, 4. Pokud by algoritmus hledal nejlevnější složenou trubku délky 8, tak nalezne složenou trubky reprezentovanou modře označeným uzlem, pokud by hledal složenou trubku z nejméně dílů, tak nalezne zeleně označený uzel, pokud by na tom nezáleželo, tak nalezne obě dvě trubky (oba dva uzle) a prohlásí je oba za optimální řešení.



Obrázek 3.7: Strom pro modelovou situaci, kde je modře vyznačeno optimální nejlevnější řešení pro složenou trubku délky 8 a zeleně vyznačené optimální řešení pro složenou trubku z nejméně dílů.

Algoritmus by zohledňující cenu vypadal následovně:

1. Nejprve se nastaví ceny trubek.
2. Na začátku eixistuje seznam nezpracovaných složených trubek pro první krok, kde je pouze složená trubka délky 0 a ceny 0, která vznikla přidáním trubky délky 0 a ceny 0.
3. Trubky použitelné pro sestavení složené trubky se seřadí sestupně.
4. Algoritmus postupně zpracovává seznam nezpracovaných složených trubek pro daný krok:
    - Aktuální složenou trubku postupně zkusí prodloužit o všechny délky trubek, které má pro skládání složené trubky k dispozici a jsou stejně velké nebo menší než naposledy přidaná trubka, jejímž přidáním aktuální složená trubka vznikla.
    - Pokud je nově vzniklá prodloužená trubka menší než hledaná složená trubka a zároveň doposud nebyla složená trubka takové délky vytvořena, tak se zařadí na seznam nezpracovaných složených trubek pro následující krok. Pokud je rovna délce složené trubky, ktera již byla vytovřena, tak se postupuje podle

instrukcí zde 3.3. Pokud je rovna délce hledané složené trubky, tak ji zařadíme mezi potenciální optimální řešení.

- Po prodloužení aktuální složené trubky, všemi přípustnými trubkami, algoritmus pokračuje na další trubku v seznamu nezpracovaných trubek pro aktuální krok. Pokud v seznamu další trubka není tak pokračuje zpracováváním seznamu nezpracovaných trubek pro další krok. Pokud je tento seznam prázdný, tak zpracovávání končí, porovnají se potenciální optimální řešení a určí se, která z nich jsou optimální oproti uživatelským požadavkům na řešení. Pokud žádné potenciální optimální řešení neexistuje, tak ani žádné optimální řešení neexistuje.

Pouze mříže s nejméně otvory
Pokud si uživatel přeje najít mříže s nejmenším počtem otvorů, tak díky prohledávání do šířky, můžeme nezávisle na ostatních požadavcích, ukončit prohledávání (prodlužování trubek) po prodloužení poslední nezpracované trubky z aktuálně zpracovávaného seznamu nezpracovaných trubek, pokud v tomto kroku byla nalezena alespoň jedna složená trubka požadaované délky. A následně pouze porovnáme potenciálně optimální řešení podle uživatelem zadaných parametrů a určíme, která z nich jsou skutečnými optimálními řešeními. Jelikož prohledávání do šířky jde po úrovních stromu a úroveň stromu, ve které se nalézá uzel, je zároveň rovna počtu složených trubek, ze kterých je strana mříže, reprezentována uzlem, sestavena.

Přidání ceny fitinek
Pro přidání ceny fitinek musíme rozlišit návrh strany mříže a návrh složené trubky, která bude následně použita pro návrh strany mříže:

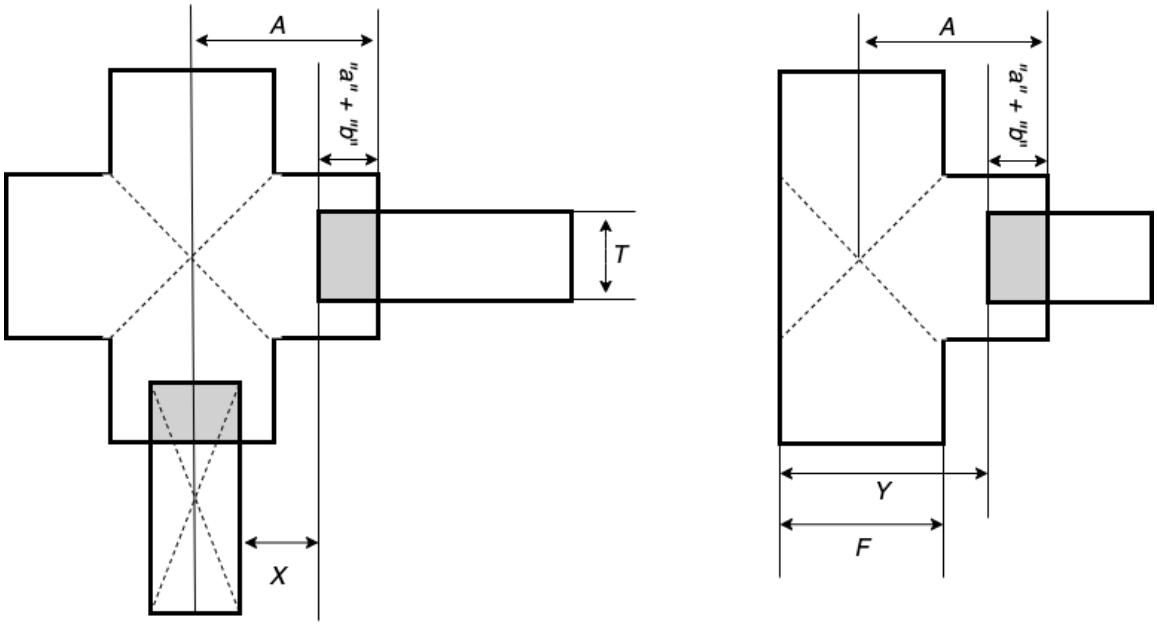
- V případě návrhu složených trubek – každá složená trubka obsahuje pouze fitinky typu nátrubek a jejich počet je o jednu menší než počet trubek složené trubky. Proto stačí, abychom od druhého kroku (tzn. od zpracovávání druhého seznamu nezpracovaných trubek) při každém prodloužení aktuálně zpracovávané složené trubky přičetli cenu fitinky nátrubek.

- V případě návrhu strany mříže – můžeme brát v úvahu pouze řešení, které je na obou koncích zakončena fitinkou T-kus a jednotlivé složené trubky strany mříže jsou spojeny fitinkou kříž. Krajní řešení, kdy jsou na koncích fitinky koleno a složené trubky jsou spojeny fitinkami T-kus, můžeme později jednoduše dopočítat. Důležité je nezapomenout na fakt, že každá fitinká typu kříž, T-kus nebo koleno je ve výsledné mříži sdílena právě dvěmi složenými trubkami (řádkem a sloupcem mříže), proto nesmíme přičítat celou cenu fitinky, ale pouze polovinu ceny. Pro algoritmus to potom znamená, že v prvním kroku pokaždé přičte přičte 2×polovinu ceny fitinky kříž. Od druhého kroku, při každém prodloužení aktuálně zpracovávané složené truby přičte polovinu ceny fitinky kříž.

Díly mají všechny rozměry
Pro přidání rozměrů fitinek a trubek musíme opět rozlišit mezi návrhem složené trubky a strany mříže:

- Pro návrh složené trubky – musíme od druhého kroku, při každém prodloužení aktuálně zpracovávané složené trubky přičíst hodnotu $A - 2 * (\text{"}a\text{"} + \text{"}b\text{"})$, A je délka fitinky nátrubek, viz 2.7, a $\text{"}a\text{"} + \text{"}b\text{"}$ je délka závitu trubky, viz 2.5.

- Pro návrh strany mříže – musíme nejpreve správně omezit délku použitelných složených trubek, protože fitinky na jejich koncích zapříčiní, že otvor bude mít větší rozměr, než jsou délky složených trubek, které ho ohraničují. Pokud použijeme proměnné použité v levé části obrázku 3.8 a maximální délku složené trubky označíme l a o maximální velikost otvoru, pro rozměr, pro který stranu mříže navrhujeme, potom $l \leq o - 2 * X$, kde $X = A - T/2 - (\text{"}a\text{"} + \text{"}b\text{"})$. Pokud máme vybrané správné složené trubky pro návrh strany mříže, tak algoritmus v prvním kroku po každém prodloužení aktuálně zpracovávané složené trubky přičte ještě velikost $2 * Y$, kde $Y = A + F/2 - (\text{"}a\text{"} + \text{"}b\text{"})$, viz proměnné v pravé části obrázku 3.8. A od druhého kroku, při každém prodloužení aktuálně zpracovávané složené truby přičte $2 * (A - (\text{"}a\text{"} + \text{"}b\text{"}))$, vize levá část obrázku 3.8.



Obrázek 3.8: V lévé části jsou znázorněny rozměry potřebné pro spočítání prodloužení X složené trubky fitinkou kříž. V pravé části jsou znázorněny rozměry potřebné pro spočítání prodloužení Y složenétrubky fitinkou T-kus).

3.4 Složitost navrženého algoritmu

Časovou složitost tohoto algoritmu je v njehorším případě exponencionální a můžeme ji zapsat jako $O(n2^n)$. Protože musí vypočítat optimální řešení, tak v nejhorším případě musí projít všechna potenciální řešení neboli vygenerovat všechny kombinace trubek do zadané délky. Pokud by všechny kombinace obsahovali stejný počet trubek, tak by počet kombinací byl $\binom{n}{k}$, ale protože počet trubek v kombinacích může být od 1 do t/m, kde t je délka skládané trubky a m je délka nejkratší trubky použitelné k návrhu složené trubky, tak potom s využitím binomické věty dostáváme $(n2^n)$.

3.5 zhodnocení návrhu řešení

Navržený algoritmus dokáže nalézt optimální řešení pro přesnou délku hledané složené trubky, respektive řádku nebo sloupce pravidelné mříže.

Prvním problémem je, že v případě neexistence řešení pro zadaný rozměr, v případe hledání návrhu délku strany mříže, musí algoritmus nalézt optimální řešení pro nejbližší menší zadání (menší rozměr hledané strany mříže). S využitím aktuálního algoritmu toho můžeme jednodušše docílit jeho opakovaným spouštění, pokaždé s nebližším nižším možný rozměrem, dokud nenalezne první optimální řešení. Absenci řešení, tak aby bylo uspokojeno zadání, lze potvrdit až pokud nebude nalezeno žádné optimální řešení při spuštění algoritmu pro všechny přípustné délky. Druhou možností jak tohoto docílit je, že si algoritmus bude pamatovat všechna potenciální optimální řešení pro jednotlivé délky, které byly vypočítány při hledání optimálního řešení pro zadanou délku. Potom, v případě nenalezení řešení pro zadanou délku, algoritmus našel všechny potenciální optimální řešení pro všechny menší délky, pokud existují, než byla zadaná délka a je tímpádem možné, prostým porovnáním jednotlivých nalezených řešeních pro menší zadání, nalézt optimální řešení pro nejbližší menší zadání. Tohoto lze s výhodou využít i pro hledání složených trubek pro návrh strany mříže, kdy stačí hledat optimální řešení pro nejdelší přípustnou složenou trubku dokud nezpracujeme všechny nezpracované uzly, tím zajistíme nalezení všech existujících optimálních řešení pro všechny délky složených trubek $\leq$ než byla délka hledané složené trubky.

Durým problémem je, že optimální řešení návrhu strany mříže nemusí být součátí optimálního řešením návrhu celé mříže. Z tohoto důvodu je potřeba porovna jednotlivé dvojice takové, kde je prvním členem nalezené optimální řešení pro šířku mříže a druhým členem je nalezené optimální řešení pro výšku mříže. A až potom je možné zhodnotit, které dvojice, těchto optimálních řešení pro jednotliv strany, jsou optimálním řešením pro celou mříž.

Kapitola 4

Implementace navrženého algoritmu

4.1 Výběr programovacího jazyka a knihovny

S ohledem na možnost využitím implementace jsem se rozhodl pro implementaci řešení problému návrhu mříží v javaScriptu s použitím knihovny p5.js [20], kvůli jednoduché použitelnosti na webových stránkách.

JavaScript je objektově orientovaný, událostmi řízený jazyk, který v roce 1995 navrhl Brendan Eich. V dnešní době je velice populární na webových stránkách, kde je často vkládán přímo do HTML kódu [3].

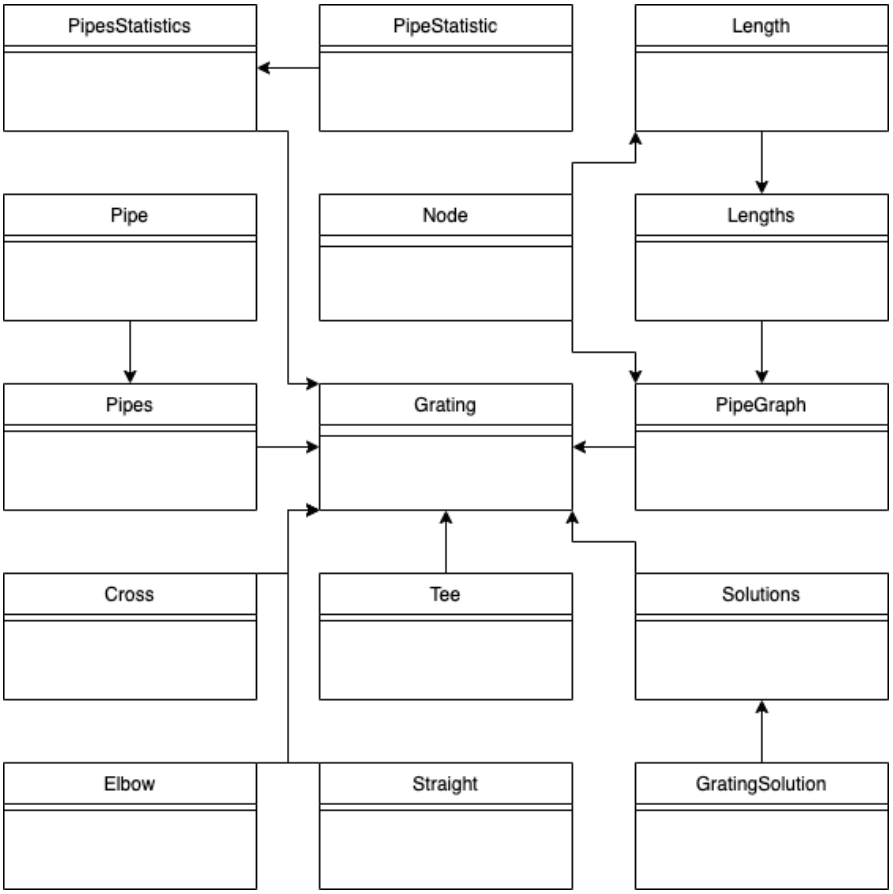
Grafická knihovka p5.js je snado použitelná knihovna pro grafické vizualizování výstupu algoritmů, její hlavní funkce jsou `function setup() {nastavení}` a `function draw() {kresli}`. Funkce `setup()` slouží k nastavení parametrů kreslení – například nastavení velikosti plátna, barvy pozadí nebo typu barevného schématu. Funkce `draw()` potom slouží k vykreslování plátna, kdy je v nekonečné smyčce volána a po každém průchodu je na plátno vykresleno vše, co bylo v průběhu této funkce nakresleno, pomocí `noLoop() a loop()`, je možné pozastavit nebo pokračovat v nekonečné smyčce a tím například zabránit zbytečnému překreslování neměnné scény a šetřit procesor.

4.2 Základní struktura a uživatelské rozhranní

Nejprve jsem navrhl strukturu základní části programu – mříže – mříž bude hlavní třídou, se kterou bude pracovat uživatelské rozhranní. Na obrázku 4.1 je zobrazen návrh jednotlivých tříd, jejich význam je následující:

- `Cross, Tee, Elbow, Straight` – třídy pro fitinky typu kříž, T-kus, koleno a nátrubek. Budou uchovávat parametry fitinek zadaných uživatelem. Pro implementaci pomocí tříd jsem se rozhodl z důvodu jednodušší rozšiřitelnosti řešení, pro případ kbyby algoritmus podporoval další, nestandardní, typy fitinek.
- `Pipe` a `Pipes` – třída `Pipes` rozšiřuje třídu `Array` a je polem instancí tříd `Pipe`, slouží k uchování množiny trubek a zjednodušuje práci s nimi. Třída `Pipe` potom slouží k uchování informací o jednotlivých trubkách, trubkou může být jednoduchá trubka zadaná uživatelem nebo složená trubka.

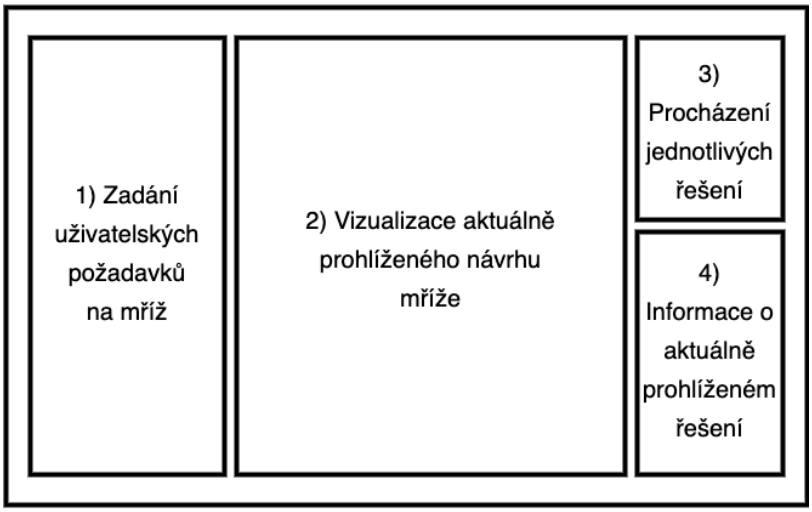
- `PipeStatistic` a `PipesStatistics` – třída `PipesStatistics` rozšiřuje třídu `Array` a je polem instancí tříd `PipeStatistic`, slouží k uchování statistik o trukách. Třída `PipeStatistic` je potom pro statistiky pro konkrétní trubku zadanou uživatelem.
- `Node` – reprezentuje uzel pro algoritmus návrhu složené trubky respektive strany mříže, viz 3.3.
- `Length` – uchovává seznam uzlů, instancí třídy `Node`, se stejnou délkou.
- `Lengths` – rozšiřuje třídu `Array` a uchovává seznamy jestnotlivých délek složených trubek, to znamená, že je polem instancí třídy `Length`.
- `pipeGraph` – je třída pro tvorbu a uchování stromové struktury potenciálně optimálních řešení, ty jsou reprezentovány uzly. Struktura je uspořádána do seznamů podle délky uzlů, které obsahují. Zároveň jsou mezi sebou propojeny uzly ze stejného počtu podtrubek. Využívá algoritmus navržený v 3.3.
- `GratingSolution` a `Solutions` – třída `GratingSolution` slouží k uchování informací o jednom optimálním řešení návrhu mříže, třída `Solutions` potom uchovává seznamy všech optimálních řešení pro návrh mříže.
- `Grating` – je centrální třídou, se kterou pracuje uživatelské rozhranní, zařídí vygenerování navrhů mříže, jejich uchování, umožní jejich procházení a vizualizaci.



Obrázek 4.1: Zobrazení jednotlivých tříd a jejich vzájemných vztahů.

Následně jsem se zaměřil na uživatelské prostředí a postupně jsem dospěl k rozvržení na 4 hlavní oblasti, viz 4.2, význam jednotlivých řešení je následující:

1. Levá část bude obsahovat zaškrtávací políčka pro nastavení preferencí pro návrh mříže, vstupní pole pro zadání rozměrů dílů, mříže a jejích otvorů. Vytvoření návrhu mříže se provede kliknutím na tlačítko.
2. Středová oblast bude sloužit k vykreslení vizualizace aktuálně zvoleného řešení.
3. Pravá horné oblast bude obsahovat tlačítka pro překlikávání mezi řešeními, počty řešení a číslo aktuálního řešení.)
4. Pravá spodní oblast bude obsahovat statistiky o počtu dílů použitých pro aktuálně zvolený návrh mříže a celkovou cenu mříže.



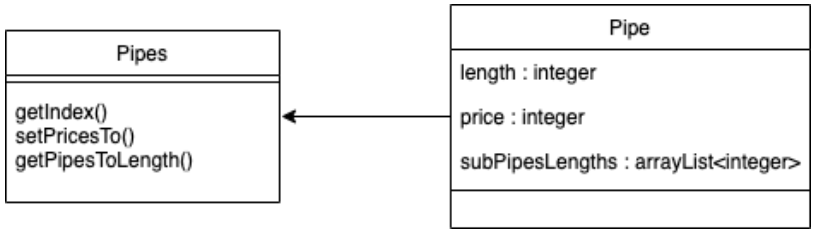
Obrázek 4.2: Zobrazení návrhu uživatelského rozhranní.

Technické fungování uživatelského rozhranní bude následující:

- Při spuštění se inicializuje ukázkovými hodnotami uživatelských požadavků.
- Následně vytvoří instanci třídy `Grating` s ukázkovými uživatelskými požadavky.
- Vykreslí uživatelské rozhranní – pro vykreslení mříže a výsledků použije metodu `draw()` třídy `Grating`, pro získání informací o řešeních použije metody třídy `Soltuions`, jejiž instanci třída `Grating` obsahuje.
- Potom bude čekat na uživatelskou interakci – kterou může být buďto přepnutí na jiné řešení pro aktuální mříž – k tomu využije příslušnou metodu třídy `Grating` a následně proběhne překreslení plátna – nebo zadání nových požadavků na mříž a pokyn k nalezení návrhů nové mříže, která je bude splňovat – v tomto případě se nové požadavky načtou, vytvoří se nová instance třídy `Grating` a následně se překreslí plátno.

4.3 Popis tříd algoritmu

4.3.1 Třídy Pipes a Pipe



Obrázek 4.3: Zobrazení tříd `Pipes` a `Pipe` a jejich vzájemného vztahu.

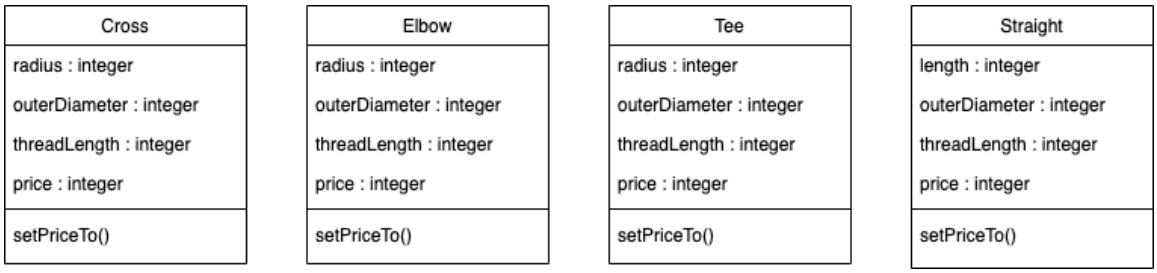
Třída `Pipe` slouží k reprezentaci trubek, jednoduchých i složených. Obsahuje následující atributy:

- length – celková délka trubky.
- price – celková cena truby, složenou trubku bereme jako jeden celek a do ceny se tedy počítá i cena fitinek potřebných k jejímu složení.
- subPipesLengths – pole délek trubek, ze kterých je trubka složena. Pokud není složena z trubek, má toto pole délku 1 a tento jediný prvek pole obsahuju délku trubky samotné.

Třída `Pipes`, která rozšiřuje třídu `Array`, potom slouží k manipulaci s větším množstvím trubek neboli instance třídy `Pipes` je pole instancí třídy `Pipe`. Má několik užitečných metod pro manipulaci s trubkami:

- `getIndex()` – vrátí index trubky o zadané délce.
- `getPipesToLength()` – vrátí trubky do zadané délky.

4.3.2 Třídy Cross, Tee, Elbow a Straight



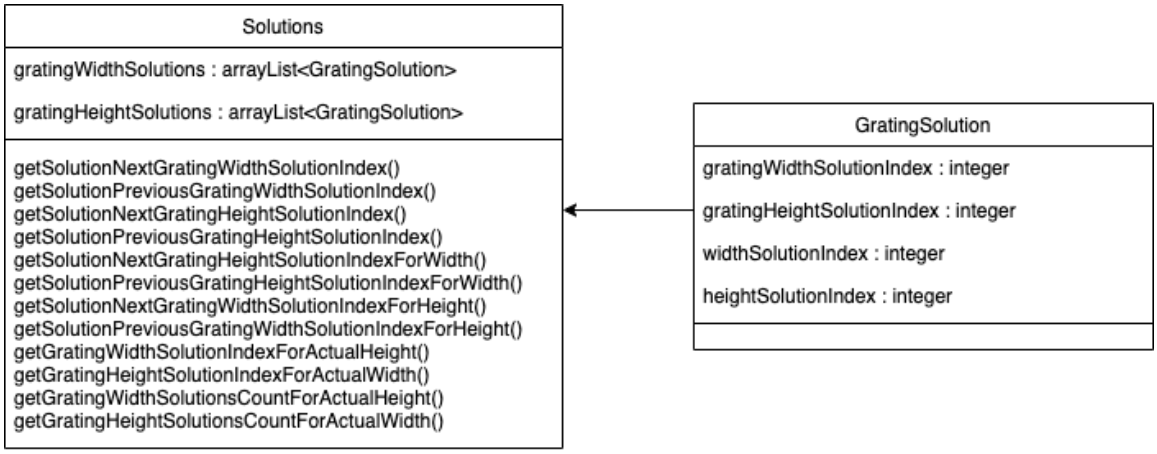
Obrázek 4.4: Zobrazení tříd `Cross`, `Tee`, `Elbow` a `Straight`.

Třídy pro fitinky – `Cross`, `Tee`, `Elbow` a `Straight` – slouží k uchování parametrů fitinek a obsahují následující atrubuty:

- `price` – cena fitinky.

- `threadLength` – délku závitu fitinky.
- `outerDiameter` – vnější průmer fitinky.
- `radius` nebo `length` – rozměr A fitinky, viz 2.7.

4.3.3 Třídy Solutions a GratingSolution



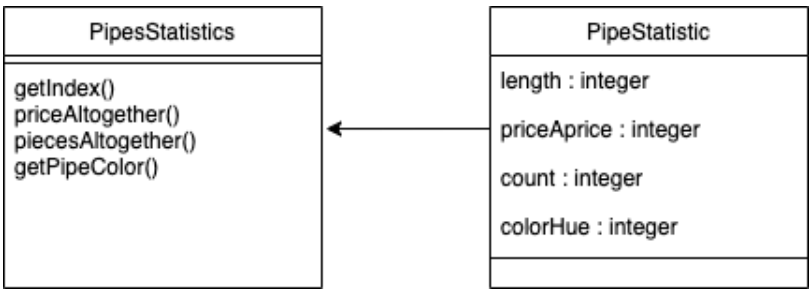
Obrázek 4.5: Zobrazení tříd `Solutions` a `GratingSolution` a jejich vzájemného vztahu.

Třída `GratingSolution` slouží k zaznamenání jednotlivých řešení. Každá instance obsahuje index řešení pro výšku a index řešení pro šířku mříže, které jsou dohromady optimálním řešením návrhu celé mříže. Její atributy mají následující význam:

- `gratingWidthSolutionIndex` – index řešení šířky mříže, z řešení šířek mříže, která jsou součástí optimálního řešení návrhu mříže.
- `widthSolutionIndex` – index řešení šířky mříže, ze všech optimálních řešení pro šířku mříže, které odpovídá hodnotě `gratingWidthSolutionIndex`.
- `gratingHeightSolutionIndex` – index řešení výšky mříže, z řešení výšek mříže, která jsou součástí optimálního řešení návrhu mříže.
- `heightSolutionIndex` – index řešení výšky mříže, ze všech optimálních řešení pro výšky mříže, která odpovídá hodnotě `gratingHeightSolutionIndex`..

Třída `Solutions` potom obsahuje dvě pole řešení – jedno pro šířku mříže a druhé pro její výšku. Zároveň obsahuje metody pro procházení řešení, jedná se o ty které začínají na `getSolution...()`.

4.3.4 Třídy PipesStatistics a PipeStatistic

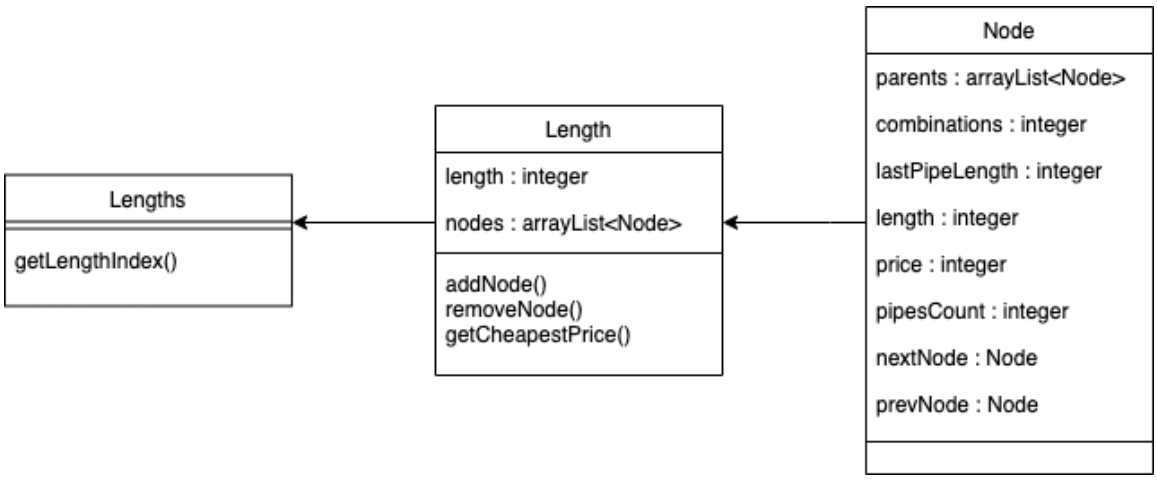


Obrázek 4.6: Zobrazení tříd `PipesStatistics` a `PipeStatistic` a jejich vztahu.

Třída `PipeStatistic` slouží k uchování statistiky o trubce zadané uživatelem. Její Atributy jsou následující:

- `length` – délka trubky.
- `priceApiece` – cena za kus trubky.
- `count` – kolikrát je použita v aktuálním řešení mříže.
- `colorHue` – jakým odstínem barvy má být vykreslena při vizualizaci řešení návrhu mříže.

4.3.5 Třídy Lengths, Length a Node



Obrázek 4.7: Zobrazení tříd `Lengths`, `Length` a `Node` a jejich vztahu.

Na dně hierarchie, zobrazených tříd, se nachází třída `Node`, instance této třídy reprezentují jednotlivé uzly stromu a uchovávají následující informace:

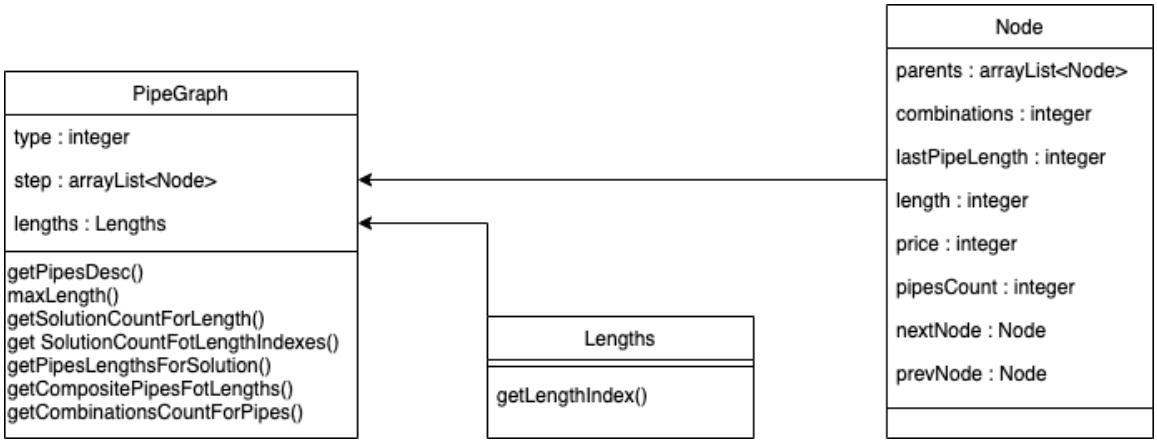
- `length` – délku složené trubky reprezentované uzlem.
- `price` – cenu složené trubky reprezentované uzlem.

- `pipesCount` – počet trubek, ze kterých je složena trubka reprezentována uzlem.
- `lastPipeLength` – délka poslední trubky, kterou byl uzel vytvořen.
- `nextNode` a `prevNode` ukazatele na následující a předešlý uzel vytvořený ze stejného počtu trubek – tím je v podstatě vytvořen dvousměrně svázaný seznam uzlů ze stejného počtu trubek.
- `parents` – seznam všech rodičů, ze kterých lze tento uzel s takovouto délkou poslední trubkou a z tohoto počtu trubek vytvořit – tím je omezeno množství vytvářených uzlů a ušetří se paměť, můsíme ovšem přidat počet kombinací – atribut `combinations` – který uchovává hodnotu rovnou počtu cest vedoucích do tohoto uzlu, tato hodnota je rovna součtu všech kombinací, kterými můžeme vytvořit jeho rodiče. Uzel bez rodičů má počet kombinací roven 1, uzel s rodiči má počet kombinací roven součtu kombinací všechn jeho rodičů.

Nad třídou `Node` je třída `Length`, která je polem všech uzlů – atribut `nodes` – které mají stejnou délku. Informaci o délce uzlů v poli `nodes` má v atributu `length`.

Třída `Lengths` je potom seřazeným pole tříd `Length` podle délky uzlů, které instance třídy `Length`, obsahuje (tzn. podle hodnoty jejich atributu `length`).

4.3.6 Třída PipeGraph



Obrázek 4.8: Zobrazení třídy `PipeGraph` a vtahu tříd `lengths` a `Node` k ní.

Třída `PipeGraph` je základem celého algoritmu. její atributy mají následující význam:

- `type` – obsahuje informaci, zdali se jedná o strom pro složené trubky nebo pro stranu mříže.
- `step` – pole ukazatelů na první uzel z počtu trubek rovnému $i + 1$, kde i je index prvku pole `step`, ve kterém je ukazatel.
- `lengths` – instance třídy `Lengths`, která uchovává uzly, které jsou optimálním řešením pro svou délku. Přidávání délek uzlů se povádí použitím metody `push()`. Přidání uzlu do existující délky se provádí metodou `addNode()` třídy `Length`, která ho umístí na správné místo a nastaví mu ukazatele na předchozí a následující uzel ze stejného počtu trubek. Naopak odebrání uzlu se provede metodou `removeNode()` třídy `Length`.

Konstruktor této třídy na základě požadavků na mříž vytvoří s pomocí metody `createGraph()` stromovou strukturu s použitím algoritmu, který byl navržen v přechozí kapitole 3.3. Instance této třídy potom obsahuje vytvořený strom, ze kterého můžeme pomocí metod číst optimální řešení pro danou délku.

Hlavní z těchto metod je `getPipesLengthsForSolution()`, která vrátí trubky optimálního řešení pro danou délku podle indexu řešení – index řešení může být v rozsahu 0 až $c-1$, kde c je počet kombinací trubek pro délku uzlu neboli počet cest do uzlu – tato hodnota je uložena v atributu `combinations` instance třídy uzlu. Při hledání trubek optimálního řešení je postup následující:

1. Index řešení označíme i.
2. Mezi uzly, délky hledaného řešení, nalezneme uzel, který obsahuje řešení s daným indexem následovně:
   - Budeme postupně procházet pole uzlů (třída `Length`) a budeme od i odečítat hodnotu jejich atributu `combinations` (počet cest, které z nich vedou) dokud nedostaneme záporné číslo.
   - Až dostaneme záporné číslo, tak jsme nelezli uzel, ze kterého vede cesta řešení s daným indexem.
3. K i přičteme zpět hodnotu `combinations` nalezeného uzlu.
4. A do seznam trubek potřebných k řešení s indexem i, přidáme délku poslední trubky `lastPipeLength` nalezeného uzlu.
5. Nyní musíme obdobným způsobem určit jeho rodiče, který je součástí řešení s daným indexem i:
   - Budeme v cyklu procházet pole jeho rodičů `parents`.
   - Od i budeme odečítat hodnotu jejich kombinací `combinations`, dokud nedostaneme zápornou hodnotu, v tom případě jsme našli správného rodiče.
   - Přičteme zpět k i počet jeho kombinací `combinations`, na seznam trubek pro řešení přidáme jeho poslední trubku `lastPipeLength` a posuneme se na jeho rodiče.
   - Tímto zpusobem budeme pokračovat dokud nenarazíme na uzel bez rodičů, v tom případě jsme došli na konec cesty řešení a vrátíme seznam trubek.

Druhou zajímavou metodou je `getCompositePipesForLengths()`, která pro dané délky, vrátí složené trubky, včetně délek jejich podtrubek, jako instanci třídy `Pipes`. Každá trubka může mít více než jeden způsob složení (různé kombinace podtrubek), proto metoda ještě vyžaduje index kombinace trubek `combinationIndex`, který pro danou kombinaci délek jednoznačně určuje, které řešení se má pro kterou délku složené trubky vybrat. Index řešení pro první délku složené trubky je hodnota výrazu $c \mod k$, kde c je `combinationIndex` a k počet všech kombinací trubek pro danou délku. Pro získání indexu řešení pro druhou délku potom musíme `combinationIndex` nepreve vydělit počtem všech kombinací pro první délku a následně provést operaci *modulo* s počtem všech kombinací délek pro druhou trubku, podobně pak pro další složené trubky. Pro vypočítání řešení s indexem pro danou délku již lze využít metodu `getCompositePipesForLengths()` k získání délek podtrubek pro dané řešení.

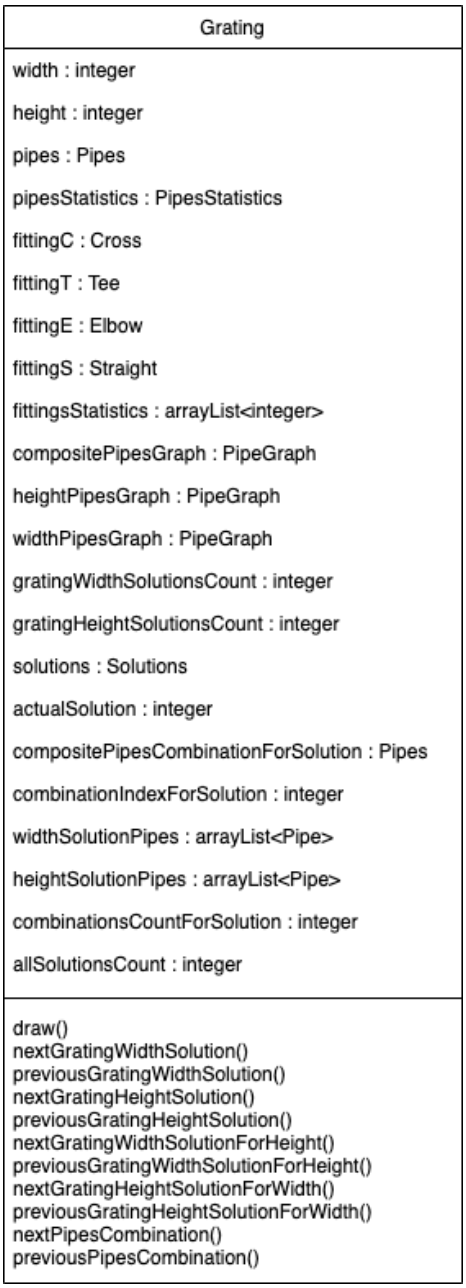
Poslední důležitou metodou je `getPipesDesc()`, která vrátí co možná nejlevnější trubku pro každou délku uzlu. Z těchto trubek potom můžeme vybrat délky, které jsou vhodné pro řešení návrhu strany mříže s danou velikostí otvoru.

Postup vytvoření stromu metodou createGraph je následující:

- Metoda počítá s tím, že pro vytváření uzlů může použít všechny trubky, které ji byly dány.
- Vytvoření uzlů skládajících se z 1 trubky, tím se vytvoří uzly, které budou zpracované v prvním kroku.
- Následně jde po krocích, kdy v každém dalším kroku zpracuje uzly vytvořené v minulém kroku, končí když již nejsou žádné další uzly ke zpracování.

    1. Ke zpracování kroku používá metodu `processStep()`, která postupně prochází uzly vytvořené v minulém kroku.
    2. Pro každý uzel volá metodu `processNode()`, která aktuálně zpracovávaný uzel postupně prodlouží o všechny přípustné trubky do délky trubky, kterou byl zpracovávaný uzel vytvořen.
    3. Každý takto nově vytvořený uzel je zpracován metodou `insertNewNode()`, která nejprve zjistí, zdali se jedná o první uzel s touto délkou nebo jestli už uzel takové délky existuje. Pokud je prvním uzlem této délky, tak se přidá do struktury instance třídy `Lengths`, jinak je zpracován metodou `checkLength()`, která ho porovná s existujícími uzly stejné délky a provedy akce podle podmínek dle návrhu algoritmu v minulé kapitole 3.3.

4.3.7 Třída Grating



Obrázek 4.9: Zobrazení třídy `Grating`.

Třída `Grating` slouží pro zpracování návrhu mříže na základě poskytnutých požadavků, jejími hlavními atributy jsou:

- `width` – šířka nejširší nalezené mříže.
- `height` – výška nejvyšší nalezené mříže.
- `pipeStatistics` – instance třídy `PipesStatistics`, informace o použitých trubkách pro návrh mříže a jejich počet pro aktuální řešení.

- `fittingsStatistics` – pole pro počet fitinek aktuálního řešení.
- `compositePipeGraph` – instance třídy `PipeGraph` neboli stromová struktura uzlů, které jsou optimálními řešeními, pro složené trubky použitelné pro návrh šířky a výšky mříže.
- `heightPipesGraph` – instance třídy `PipeGraph`, optimální řešení, pro výšku strany mříže.
- `widthPipesGraph` – instance třídy `PipeGraph`, optimální řešení, pro sířku strany mříže.
- `solutions` – instance třídy `Solutions`, struktura všech dvojic řešení šířky a výšky mříže, které dohromady tvoří optimální řešení návrhu mříže.
- `actualSolution` – instance třídy `GratingSolution`, obsahuje aktuální řešení.
- `compositePipesCombinationForSolution` – instance třídy `Pipes`, seznam složených trubek, ze kterých se skládá aktuální řešení návrhu mříže.
- `widthSolutionPipes` – pole obsahující posloupnost trubek, která je aktuálním řešením pro šířku návrhu míže.
- `heightSolutionPipes` – pole obsahující posloupnost trubek, která je aktuálním řešením pro výšku návrhu mříže.

Důležité metody třídy `Grating` jsou:

- `draw()` – vykresli vizualizaci mříže a vypíše informace o řešení, jakou jsou počet jednotlivých dílů a celková cena řešení.
- metody nazývající se `nextGrating...()` a `previousGrating..()` – slouží ke změně aktuálního řešení v atributu `actualSolution`.

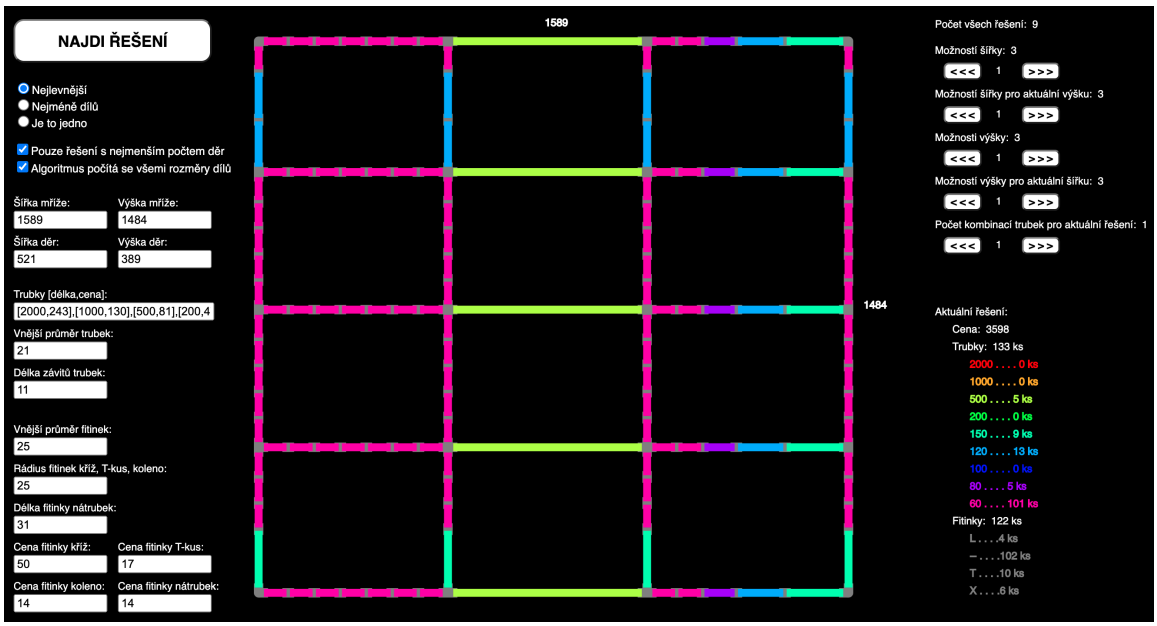
Postup konstruktoru při vytváření instance třídy `Grating` je následující:

1. Vybere použitelné trubky pro návrh mříže.
2. Inicializuje statistiky sbírané o trubkách.
3. metodou `createPipesGraphs()` vytvoří stromové struktury pro složené trubky (`compositePipeGraph`), šířku (`widthPipesGraph`) a výšku (`heightPipesGraph`) mříže. – nejprve vytvoří instanci třídy `PipesGraph` pro složené trubky, kdy maximální hodnotu uzlu stanoví na větší rozměr otvoru mříže. Tuto instanci uloží do `compositePipesGraph`. Následně vytvoří instance třídy `PipesGraph` pro šířku a výšku, trubky pro vytváření těchto stromových struktur zjistí ze stromové struktury pro složené trubky uložené v `compositePipesGraph`, využije k tomu metod `getPipesDesc()`, která vrátí všechny délky složených trubek a následně metodou `getPipesToLength()` vybere pouze ty, které jsou vhodné pro návrh šířky nebo výšky strany mříže.
4. Nalezne všechna optimální řešení návrhu mříže následovně – nejprve stanoví maximální cenu řešení použitím metody `finSolutionsMaxPrice()`, která projde všechny dvojice optimálních řešení pro šířku a výšku mříže. Potom s využitím metody `findSolutions()` projde všechny dvojice podruhé a ty, které mají menší nebo stejnou cenu oproti stanovené maximální ceně řešení, tak uloží do `solutions`.

5. Ze všech nalezených optimálních řešení vybere to první, prohlásí ho za aktuální řešení a nastaví hodnoty atributů `widthSolutionPipes`, `heightSolutionPipes` a `compositePipesCombinationForSoluion`, které jsou nezbytné k vykreslení aktuálního řešení.

4.4 Ukázka hotové implementace

Příklad vygenerování mříže pomocí algoritmu implementovaného v této kapitole můžete vidět na obrázku 4.10.



Obrázek 4.10: Ukázka hotové implementace.

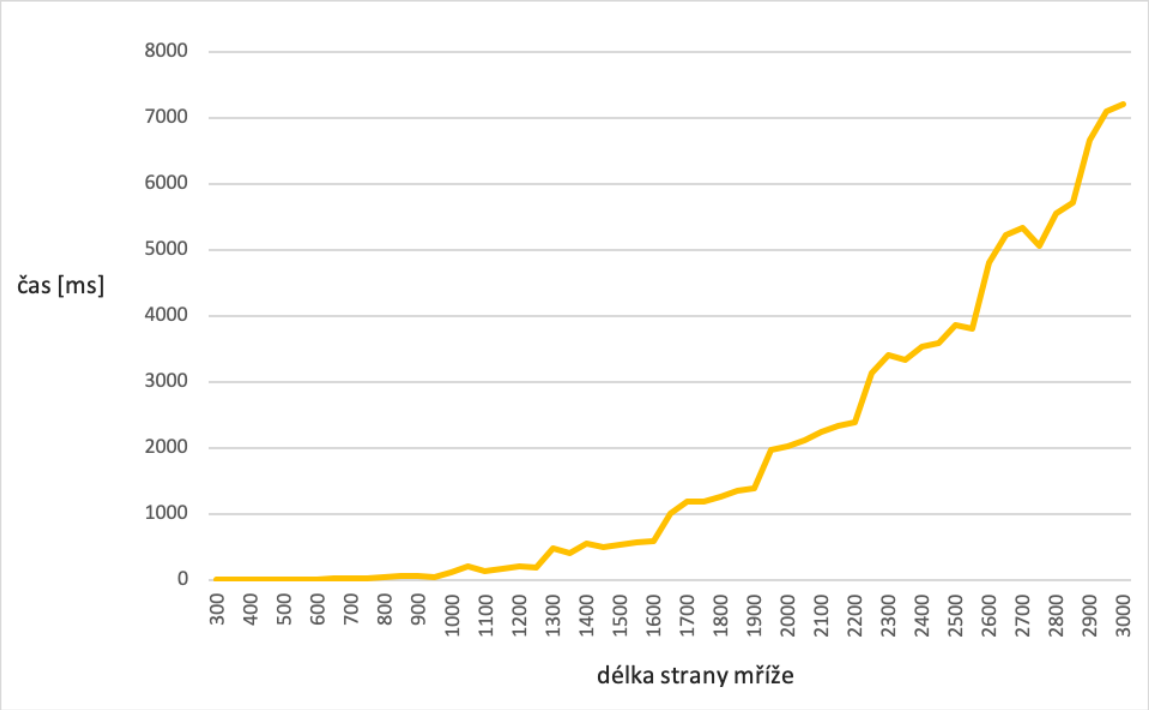
Kapitola 5

Testování implementovaného řešení

Testování jsem provedl dvěmi způsoby – nejprve jsem otestoval rychlost algoritmu a následně jsem se pokusil sestavit fyzickou mříž z reálných trubek podle návrhu algoritmu.

5.1 Testy časové složitosti

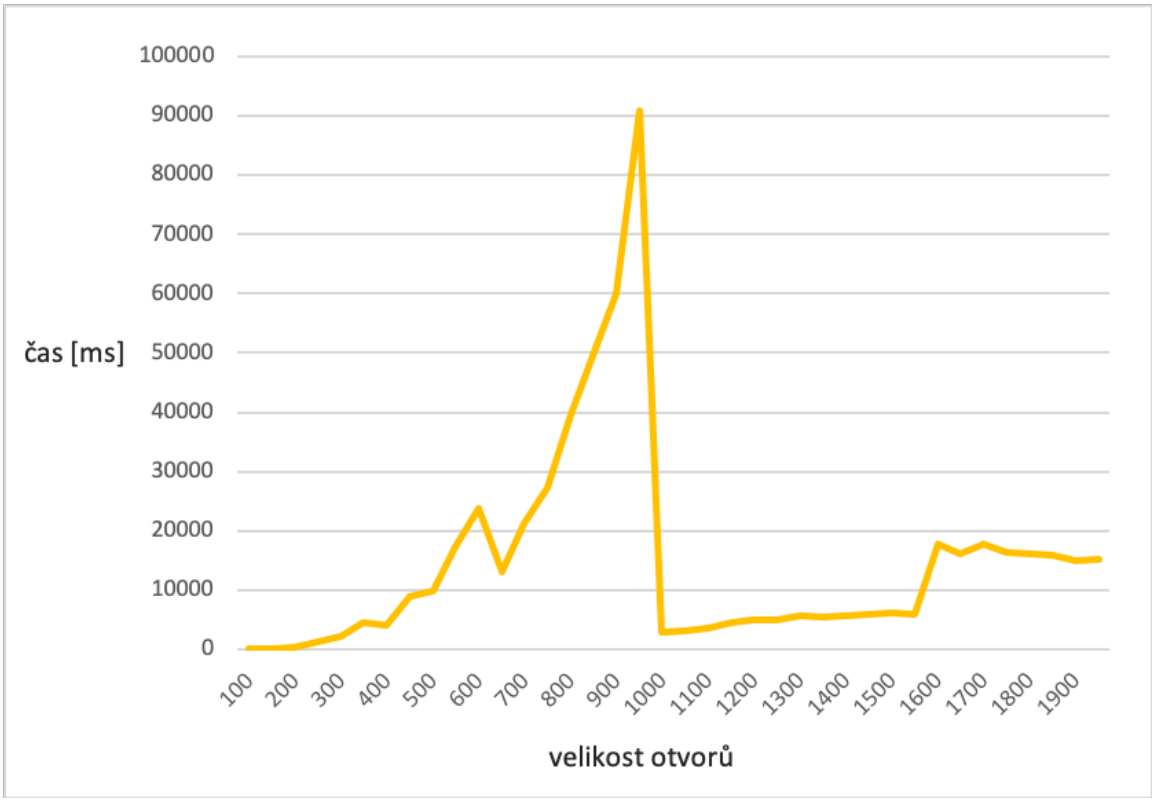
Test 1 – vliv velikosti strany mříže
V prvním testu jsem zkusil otestovat změnu rychlosti výpočtu při změně velikosti mříže. Všechny vstupní požadavky na mříž byly neměnné, kromě velikosti mříže, kterou jsem postupně zvětšoval. Aby byl test co nejvíce objektivní, tak jsem uvažoval pouze čtvercovou mříž, která má otvory v obou dvou rozměrech stejně velké. Navíc jsem meření pro každou velikost mříže provedl pětkrát a nameřené hodnoty jsem zprůměroval a zpracoval do grafu na obrázku 5.1.



Obrázek 5.1: Graf znázorňuje závislost rychlosti výpočtu na délce strany mříže.

Můžeme vidět, že rychlost stoupání se zvyšuje, výsledný graf bychom mohli proložit exponenciální funkcí. To odpovídá teoreticky vypočítané časové složitosti, kdy bylo prokázáno, že algoritmus má exponenciální časovou složitost.

Test 2 – vliv velikosti otvorů mříže
Ve druhém testu jsem zkusil otestovat změnu rychlosti výpočtu při změně velikosti otvorů mříže. Stejně jako v prvním testu, tak všechny vstupní požadavky na mříž byly neměnné, až na velikost otvorů mříže, které jsem postupně zvětšoval. Opět jsem zvolil čtvercovou mříž se čtvercovými otvory. Pro každou velikost otvoru jsem provedl pět měření, která jsem nakonec zprůměroval a zobrazil pomocí grafu na brázku 5.2.

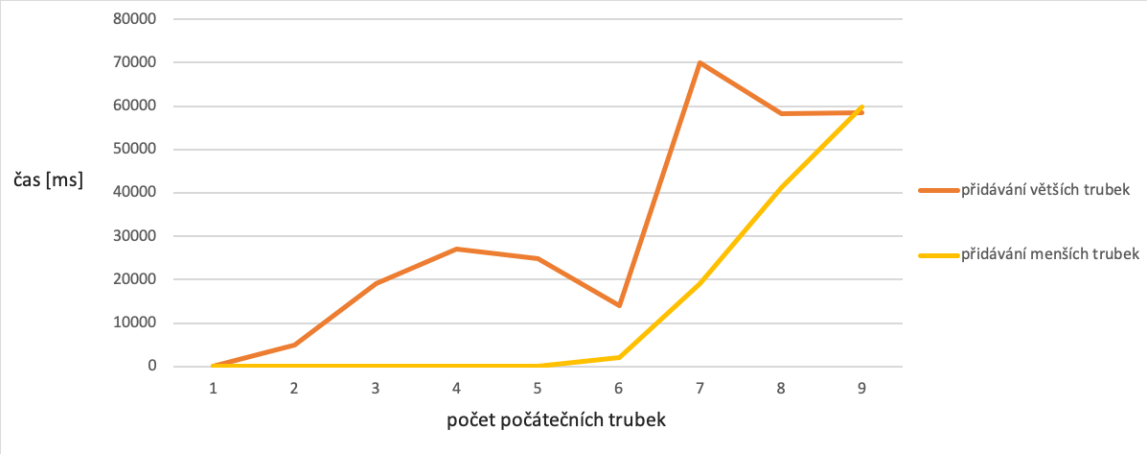


Obrázek 5.2: Graf znázorňuje závislost rychlosti výpočtu na velikosti otvorů mříže.

První polovina grafu opět připomíná exponenciálu, což se dalo očekávat, ale druhá polovina grafu připomíná spíše konstatní funkci. Tato nepřesnost je zapříčiněna velikostí mříže, která byla po celou dobu testu 2000 a požadavkem na mříž s co nejmenším pořtem otvorů. A jelikož velikost otvoru přímo ovlivňuje velikost složených tubek, ze kterých se bude vytvářet návrh strany mříže, tak pokud máme k dispozici dlouhé trubky, tak může nalezení návrhu mříže být velice rychlé, protože stranu mříže složíme z malého množství trubek a požadavek na nejměnší počet trubek mříže hledání dálších návrhů ukončí.

Test 3 – vliv počtu trubek
Třetím testem jsem otestoval změnu rychlosti v závisloti na počtu vstupních trubek. Stejně jako v prvním a v druhém testu, tak všechny vstupní požadavky na mříž byly neměnné, až na počet počátečních trubek, jejichž počet jsem postupně zvyšoval. Zkusil jem dvě varianty

přidávání trubek – nejdříve jsem přidával vždy o něco větší trubky, podruhé jsem pokáždé přidal o něco měnší trubku (pro obě varianty jsem použil stejnou množinu trubek).Opět jsem uvažoval čtvercovou mříž se čtvercovými otvory a každé měření jsem provedl pětkrtá, ty jsem nakonec zprůměroval a výsledný graf můžeme vidět na obrázku 5.3.

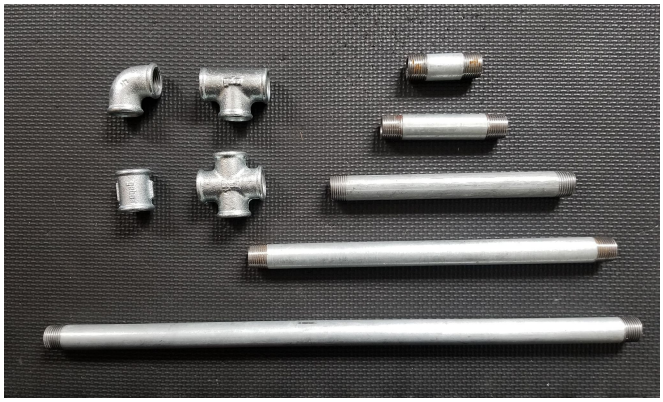


Obrázek 5.3: Graf znázorňuje závislost rychlosti výpočtu na počtu použitelných trubek pro návrh mříže.

Můžeme vidět, že s počtem trubek se doba výpočtu opět exponenciálně zvyšuje. Zároveň můžeme porovnat rozdíl mezi přidáváním kratších a delších trubek. Při přidávání kratších trubek začne graf stoupat rychleji spíše později, protože nejvíce nových možností složených trubek vzniká z krátkých trubek. Když tedy máme k dispozici pouze dlouhé trubky, tak algoritmus nevytvoří takové množtví složených trubek, a proto hledání návrhu mříže bude rychlejší, protože nebude takové množství kombinací složených trubek, které je nutné prověřit.

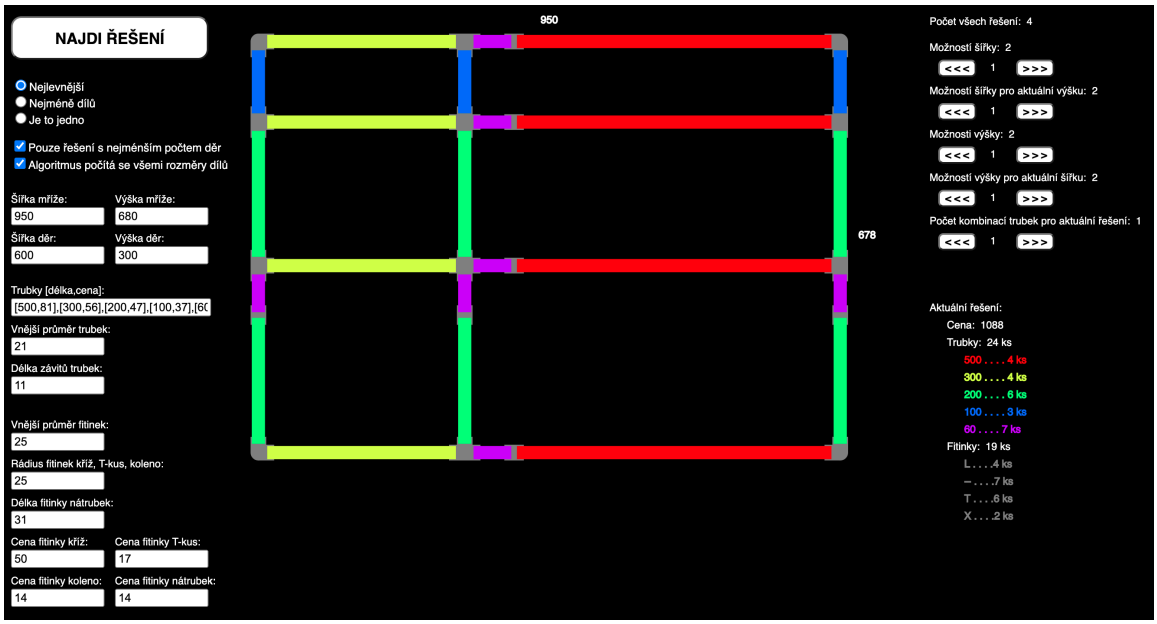
5.2 Praktický test sešroubování mříže

K praktickému testu jsem využil díly, které jsem sehnal při prvotním zkoumání problematiky návrhu mříže. Měl jsem tedy k dispozici 1/2” trubky o délkách $60mm, 100mm, 200mm, 300mm, 500mm$ a fitinky typu kříž, T-kus, koleno a nátrubek viz obrázek 5.4.



Obrázek 5.4: Ukázka dostupných dílů pro praktický test sešroubování mříže.

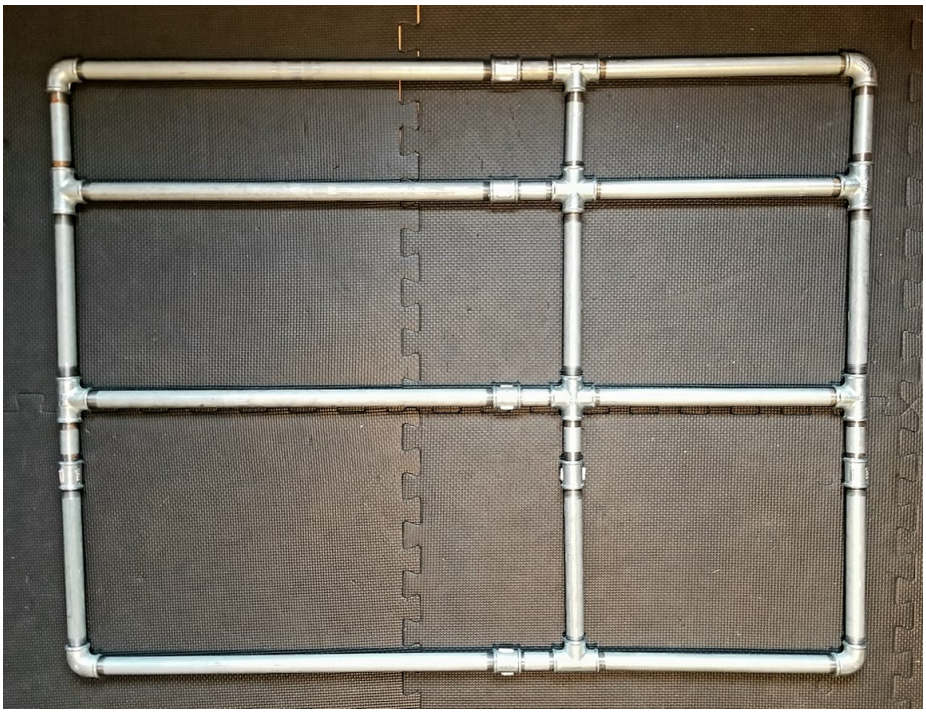
Zkusil jsem rozměry dílů spolu s požadavkem na rozměry mříže a otvorů zadat algoritmu. Nejprve mi algoritmus nabízel návrhy mříží, na které jsem neměl dostatečné množství dílů, ale po pár pokusech, s ruznými rozměry mříže a otvorů, mi algoritmus nabídl návrh mříže, kterou jsem mohl sestavit. Algoritmem navržená a vizualizovaná mříž je na obrázku 5.5.



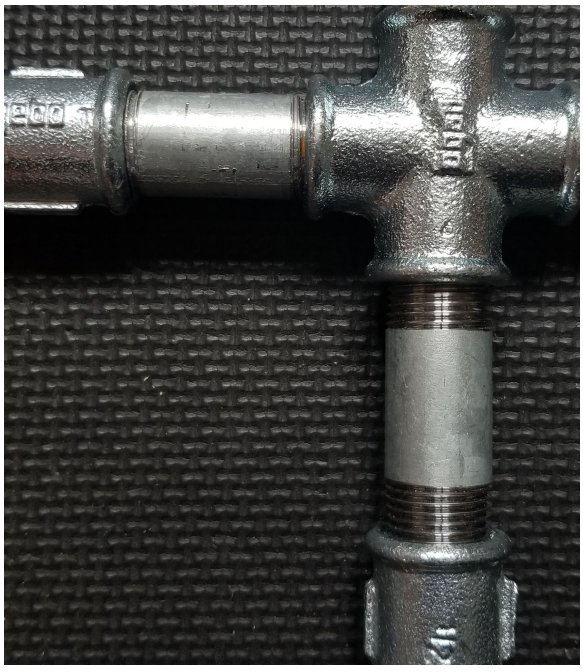
Obrázek 5.5: Algoritmem navržená mříž, kterou jsem se pokusil prakticky sešroubovat z reálných trubek.

Mříž jsem zkusil sešroubovat a výsledek je možné vidět na obrázku 5.6. Velikost mříže, kterou algoritmus navrhl je $950 \times 678mm$, mnou sestavená mříž má rozměry $975 \times 760mm$. V jednom rozměrů je tedy větší o $25mm$ a ve druhém dokonce o $82mm$. Prvním důvodem je, že jsem mříž sešrouboval bez použití nářadí a nebyl jsem tedy schopen úplně dotáhnout závity. Podle tabulky 2.3, je část závitů "a", kterou lze dotáhnout rukou $8,2mm$ u trubky velikosti 1/2", já pro návrh mříže použil délku závitu $11mm$, to je rozdíl skoro $3mm$ u každého závitu. Pokud se podíváme na rozměr, ve kterém byla mnou vytvořená mříž o $25mm$ delší, tak zjistíme že obsazuje 6 vnějších závitů trubky, to je celkem $18mm$ rozdíl, pokud tedy od $25mm$ odečteme $18mm$, tak získáme $7mm$, a to už není tak špatný výsledek, určitě by s využitím nářadí bylo možné takový rozdíl vyrovnat. Pokud se podíváme na problém druhého rozměru, který jsem oproti návrhu algoritmu měl o $82mm$ větší, tak i při odečtení $18mm$ ($3mm$ za každý vnejší závit v tomto rozměru strany mříže) se dostaneme na $64mm$. To je způsobeno stejnou orientací všech závitů a není proto možné dotáhnout všechny závity uzavřeného okruhu, viz obrázek 5.7. Pokud tedy zvolíme optimální strategii při sešroubování mříže, tak v jednom směru budou vždy některé trubky nedotažené, protože když je zašroubujeme na jedné straně, tak při zašroubovávání na druhé, se začnou vyšroubovávat z té první. Kvůli tomu bude v tomto směrů mříž o počet otvorů krát délka vnějšího závitu trubky delší. Ve mnou sešroubované mříží jsou v tomto směru 3 otvoru, takže realná mříž bude v tomto rozmětu oproti návrhu algoritmu minimálně o $3 \times 11mm = 33mm$ delší. Pokud tedy od $64mm$ odečteme ještě 33, tak se dotaneme na $31mm$. To sice není zanedbatelná

hodnota, ale pokud bych k sestavení mříže použil nářadí, tak by se mi ji jistě podařilo ještě o něco přiblížit rozměru algoritmem navrhované mříže.



Obrázek 5.6: Ukázka sešroubované mříže podle návrhu algoritmu.



Obrázek 5.7: Ukázka nedotažitelnosti všech trubek v mříži.

Kapitola 6

Závěr

Cílem práce bylo pochopit problém návrhu mříží z pozinkovaných trubek, následně navrhnout vhodný algoritmus, který na základě uživatelských požadavků na mříž vygeneruje řešení. Tento algoritmus naimplementovat a otestovat. To se mi podařilo.

Při řešení práce jsem se seznámil s oblastí pasivní bezpečnosti, dozvěděl jsem se spoustu informací o trubkách, fitinkácha a jejich závitech. Poté jsem pronikl do problematiky NP-úplných problémů a vyzkoušel jsem si některé existující algoritmy pro jejich řešení. prokračoval jsem k návrhu algoritmu pro návrh mříž, kde jsem se rozhodl pro řešení exaktním algoritmem a musel jsem řešit několik optimalizačních výzev. Navržený algoritmus se mi podařilo naimplementovat a navíc jsem se naučil používat novou grafickou knihovnu. Nakonec jsem provedl sérií testů včetně praktického sešroubování mříže navržené algoritmem. Během kterého jsem objevil několik nedostatků, největším z nich je nemožnost dotáhnout všechny závity. Na durhou stranu problém návrhu nepravidelné mříže nebo mříže velkých rozměrů je řešitelný a mohl by být podmětem k pokračování této práce.

Výsledkem práce je rešení, které dokáže navrhovat nejlepší možné mříže, rozumných velikostí, podle zadaných požadavků.

Literatura

[1] *Georg Fischer Fittings GmbH*. A-3160 Traisen/Austria: Georg Fischer Fittings, 2015.

[2] *Manver katalog litinové fitinky*. Pečky, Chvalovická 693: Manver, 2020.

[3] CONTRIBUTORS, W. *JavaScript* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=JavaScript&oldid=18389955.

[4] CONTRIBUTORS, W. *List of NP-complete problems* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://en.wikipedia.org/w/index.php?title=List_of_NP-complete_problems&oldid=957698266.

[5] CONTRIBUTORS, W. *Millennium Prize Problems* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://en.wikipedia.org/wiki/Millennium_Prize_Problems.

[6] CONTRIBUTORS, W. *Optimal substructure* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Optimal_substructure&oldid=956404218.

[7] CONTRIBUTORS, W. *P versus NP problem* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://en.wikipedia.org/w/index.php?title=P_versus_NP_problem&oldid=959884762.

[8] COOK, S. A. *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*. Association for Computing Machinery, New York, NY, United States, 1971. 151-158 s. ISBN 978-1-4503-7464-4.

[9] DVEŘE24. *Statistika: Kudy nejčastěji chodí pachatelé vloupání?* [online]. 2018 [cit. 2020-28-05]. Dostupné z: https://dvere24.cz/statistika-kudy-nejcasteji-chodi-pachatele-vloupani/.

[10] FITINKA, P. *DN (Diameter Nominal) - jmenovitá světlost potrubí* [online]. 2016 [cit. 2020-28-05]. Dostupné z: https://www.panfitinka.cz/post/dn-diameter-nominal-jmenovita-svetlost-potrubi.

[11] HARDWARE, R. *Industrial steel pipe bunk bed* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://www.rhbabyandchild.com/catalog/product/product.jsp?productId=rhbc_prod373128&src=rel.

[12] HERZEN. *Rolovací mříže [obrázek]* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://www.herzen.cz/rolovaci-mrize/.

[13] HORNBACH. *Trubka pozink* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://www.hornbach.cz/shop/vyhledavani/sortiment/trubka--_-%7Cpozink.

[14] Koktan, P. Nové označení bezpečnostních tříd MZS. *Zámkař* [online]. 1. vyd. Brno: [b.n.]. leden 2014, s. 4, [cit. 2020-28-05]. Dostupné z: http://www.azks.cz/data/clanky/files/000025.pdf.

[15] KovoKrab. *Kované mříže [obrázek]* [online]. 2020 [cit. 2020-28-05]. Dostupné z: http://www.pivni-souprava.cz/kovove-mrize-a-zabradli/.

[16] Liška, J. *Mechanické zabranné systémy ve věznici*. Zlín, 2018. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně.

[17] Lueker, G. S. *Two NP-complete Problems in Nonnegative Integer Programming*. Princeton University, Department of Electrical Engineering, 1975. Dostupné z: https://books.google.cz/books?id=x_YwrgEACAAJ.

[18] Madness, C. *Industrial Silver Curtain Rail made from 3/4"galvanized pipe* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://craftmadness.co.uk/products/industrial-curtain-rail-galvanized-iron-silver.

[19] Martello, S. *Knapsack problems*. Biddles Ltd, Guildford, Great Britain, 1990. ISBN 0-471-92420-2.

[20] McCarthy, L. *Libraries* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://p5js.org/libraries/.

[21] ProLock. *Nůžkové mříže [obrázek]* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://prolock.cz/produkt/nuzkova-mriz.

[22] Rubášová, H. *PREVENCE – CHRAŇTE SI SVŮJ MAJETEK* [online]. Duben 2018 [cit. 2020-28-05]. Dostupné z: https://www.policie.cz/clanek/prevence-chrante-si-svuj-majetek.aspx.

[23] Salvage, A. *Coffee Tables* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://arcadiasalvage.com/products/furniture/reclaimed-wood-coffee-tables/.

[24] Schwartz, H. R. *Memoization using Closures* [online]. 2011 [cit. 2020-28-05]. Dostupné z: https://harryrschwartz.com/2011/01/06/memoization-using-closures.

[25] Szalwinski, C. *Complexity theory* [online]. 2016 [cit. 2020-28-05]. Dostupné z: https://ict.senecacollege.ca/~gpu621/pages/content/intro_p.html.

[26] Triker. *Trubka pozinkovaná s oboustranným závitem* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://triker.cz/p-319900200020/Trubka-pozinkovana-s-oboustrannym-zavitem-3-4-x-20-cm/.

[27] uniquecity1. *Stair Rail, Industrial Staircase, Pipe Iron fittings , Hallway Decor Custom Made* [online]. 2020 [cit. 2020-28-05]. Dostupné z: https://www.ebay.co.uk/itm/Stair-Rail-Industrial-Staircase-Pipe-Iron-fittings-Hallway-Decor-Custom-Made-/312831789785.

Příloha A

Obsah CD

Adresářová struktura přiloženého CD:

/readme.txt – obsahuje popis obsahu CD
/finalni_implementace/ – obsahuje finální implementaci navrženého algoritmu
/latex/ – obsahuje zdrojové soubory technické zprávy
/xhulka02.pdf – je technickou zprávou
/prubezne/ – obsahuje dvě verze prubezné implementace