VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

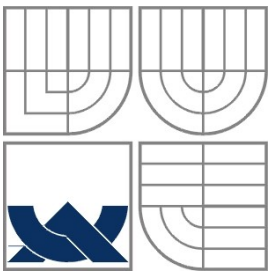
INFORMAČNÍ SYSTÉM PRO VYHLEDÁVÁNÍ PŘEPRAVNÍCH SPOJŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

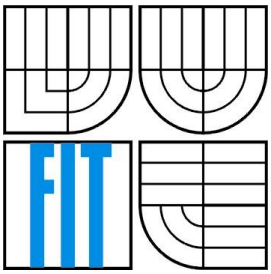
AUTOR PRÁCE
AUTHOR

MIROSLAV ČÍŽEK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO VYHLEDÁVÁNÍ PŘEPRAVNÍCH SPOJŮ

INFORMATION SYSTEM FOR SEARCHING TRAFFIC RELATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MIROSLAV ČÍŽEK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. ROMAN LUKÁŠ, PH.D.

BRNO 2008

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací webového informačního systému pro vyhledávání přepravních spojů. Hlavním cílem bylo vytvořit systém, který umožní vyhledávání spojů s přestupy na základě různých kritérií, jako „co nejkratší vzdálenost“, „co nejnižší cena cesty“ atd. Práce se zaměřuje především na návrh databáze, volbu vhodné vyhledávací metody a implementaci systému s ohledem na výkon a rychlost vyhledávání.

Klíčová slova

Informační systém, doprava, umělá inteligence, databáze, webové technologie, PHP, MySQL, XHTML, CSS, JavaScript, AJAX

Abstract

The thesis is occupied with analysis, design and implementation of web information system for searching traffic relations. The main objective of this thesis was to create system, which will provide searching traffic relations with changes based on various criteria such as „the lowest distance“, „the lowest price“ etc. The thesis is focusing principally on database design, selection of efficient searching technique and system implementation with regard to system performance and search speed.

Keywords

Information system, traffic, artificial intelligence, database, web technologies, PHP, MySQL, XHTML, CSS, JavaScript, AJAX

Citace

Miroslav Čížek: Informační systém pro vyhledávání přepravních spojů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Informační systém pro vyhledávání přepravních spojů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením
Ing. Romana Lukáše, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Čížek
10.5.2008

Poděkování

Poděkování patří především mému vedoucímu, Ing. Romanu Lukášovi, Ph.D. za vedení a pomoc při řešení některých problémů v průběhu práce.
Rovněž bych rád poděkoval panu Jaroslavu Čechovi za poskytnutí webhostingu a technické podpory.

Obsah

1 Úvod....3
2 Analýza a návrh....4
2.1 Specifikace požadavků....4
2.1.1 Základní funkce systému....4
2.1.2 Kritéria pro vyhledávání....4
2.1.3 Přestupy a typy spojů....5
2.1.4 Výsledky hledání....5
2.2 Návrh databáze....6
2.2.1 Uchovávané informace....6
2.2.2 ER Diagram....8
2.2.3 Popis ER diagramu....9
3 Metoda vyhledávání....13
3.1 Stavový prostor....13
3.2 Metoda UCS....14
3.2.1 Popis metody....14
3.2.2 Implementace pomocí metody UCS....14
3.3 Metoda A* search....16
3.3.1 Popis metody....16
3.3.2 Heuristika....16
3.3.3 Implementace pomocí metody A*....18
3.3.4 Rychlé vyhledávání....18
3.3.5 Zpětné vyhledávání....19
4 Implementace systému....21
4.1 Použité technologie....21
4.1.1 PHP....21
4.1.2 MySQL a databázová vrstva....21
4.1.3 XHTML, CSS, Javascript, AJAX....22
4.2 Webhosting....22
4.3 Implementace vyhledávání....23
4.3.1 Uzly stavového prostoru....23
4.3.2 Seznam OPEN....25
4.4 Testovací data....29
4.4.1 Získání testovacích dat....29

4.4.2 Nedokonalosti v testovacích datech.....29
4.4.3 Heuristika a průběžná aktualizace.....30
5 Srovnání s jinými podobnými systémy.....32
6 Návrhy na další rozšíření systému.....33
7 Závěr.....34

1 Úvod

Cílem této práce je popsat návrh a implementaci pokročilého informačního systému pro vyhledávání přepravních spojů na základě různých kritérií, jako je „co nejkratší ujetá vzdálenost“, „co nejnižší cena cesty“ apod. Takovéto možnosti totiž v současné době žádný existující systém nenabízí.

V první části práce se zabývám specifikací požadavků na funkčnost systému a následným návrhem systému a databáze. V následující kapitole se budu věnovat výběru vhodné metody umělé inteligence a návrhu vyhledávacího algoritmu. Poukážu na některá úskalí, na něž jsem narazil, ať už v průběhu návrhu nebo později při implementaci. Samotnou implementací systému se zabývám v další kapitole. Také se zmíním o použitých technologiích a na závěr se pokusím výsledný systém srovnat s jinými existujícími systémy pro vyhledávání přepravních spojů a navrhnout možnosti jeho případného dalšího rozšíření.

2 Analýza a návrh

Nejprve je nutné stanovit, co by měl systém obsahovat a jaké funkce a možnosti má nabízet. Poté je možné přistoupit k návrhu samotného systému. Zde se zaměřím především na návrh databáze, což je jedna z nejpodstatnějších částí celého systému. Dalším velice důležitým aspektem je volba vhodné vyhledávací metody (metody prohledávání stavového prostoru), kterou se ovšem zabývám až v další, samostatné kapitole.

2.1 Specifikace požadavků

2.1.1 Základní funkce systému

Systém bude umět vyhledávat autobusové, vlakové a kombinované spoje z místa A do místa B. Oním místem A (resp. B) může být buďto celé město, městská část, nebo konkrétní zastávka.

Kromě výchozího a cílového bodu specifikuje uživatel také datum a čas, kdy chce odjet z výchozího místa, případně dorazit do místa cílového. Tady je samozřejmě nutné počítat s tím, že jednotlivé spoje mohou mít různá omezení, např. že daný spoj jede pouze v pracovní dny, nejede v určitém období apod. Tato omezení budou detailněji rozebrána dále.

2.1.2 Kritéria pro vyhledávání

Uživateli bude umožněno zvolit, jaké spoje chce při vyhledávání preferovat. Možnosti jsou následující:

- Co nejdříve v cíli – při vyhledávání jsou preferovány ty spoje, jejichž čas příjezdu do cílového místa je co nejblíže výchozímu času zadanému uživatelem.
- Co nejkratší doba cesty – i když se na první pohled může zdát, že se jedná prakticky o stejnou možnost jako v předchozím případě, není tomu tak. Tento typ vyhledávání totiž bere v potaz skutečnou dobu cesty, tj. od odjezdu z výchozího místa do příjezdu do místa cílového. Do této doby se samozřejmě započítává i případný čas čekání na přípoje v přestupních stanicích.

  Příklad: Uživatel chce vyhledat spojení ze stanice Nový Jičín,, aut.nádr. do stanice Ostrava,, ÚAN ve všední den v časovém rozmezí 15:00 – 17:00 hod. Zvolí li kritérium „co nejdříve v cíli“, systém vyhledá takový spoj, jehož čas příjezdu do cílové stanice je co nejblíže času 15:00. V tomto případě bude tedy vyhledán spoj, který odjíždí z Nového Jičína v 15:12 hod. a do Ostravy přijíždí v 16:00 hod. (doba cesty je tedy 48 minut). Pokud ovšem uživatel zvolí

kritérium „co nejkratší doba cesty“, vyhledá systém takové spojení, kde rozdíl mezi časem odjezdu z výchozí stanice a časem příjezdu do stanice cílové je co nejmenší. V tomto případě bude tedy výsledkem spoj, který z Nového Jičína odjíždí v 16:35 hod. a do Ostravy přijíždí v 17:20 hod. Celková doba cesty je tedy 45 minut, což je méně než v předchozím případě.

- Co nejkratší ujetá vzdálenost – vyhledávač preferuje spoje s nejnižší vzdáleností mezi dvěma místy, bez ohledu na dobu jízdy, čas příjezdu, cenu apod.
- Co nejnižší cena cesty – tady je hlavním kritériem celková cena cesty. Vzhledem k tomu, že jednotlivé spoje mohou provozovat různí dopravci s rozličnými cenovými tarify a navíc na některé typy spojů může být příplatek apod., může být někdy delší trasa cenově výhodnější než kratší.
- Co nejmenší počet přestupů – systém se snaží najít cestu s co nejmenším počtem přestupů i za cenu delší vzdálenosti, vyšší ceny atd.

Uživatel může vždy zvolit pouze jedno kritérium.

2.1.3 Přestupy a typy spojů

Uživatel bude muset zvolit maximální možný počet přestupů, které je ochotný akceptovat. Tento počet může být v rozmezí 0 (tzn. hledají se jen přímá spojení) až 5.

Dále zde bude možnost zvolit minimální a maximální čas na přestup, tj. čas mezi příjezdem spoje do přestupní stanice a odjezdem navazujícího spoje.

Nezbytná je také volba typů spojů, které se mají při vyhledávání brát v potaz. U autobusové dopravy je to příměstský autobus, dálkový autobus, mezinárodní autobus a MHD, u vlakových spojů se jedná o osobní vlak, rychlík, Express, InterCity, EuroCity a SuperCity. Samozřejmě zde bude možnost rozšíření o další typy spojů, případně i druhy dopravy.

2.1.4 Výsledky hledání

Po vyhledání příslušných spojů se zobrazí výpis s výsledky hledání. Bude zobrazen celý název výchozí a cílové stanice a případných přestupních stanic, spolu s časem odjezdu z výchozí stanice, příjezdu do cílové a časy příjezdu a odjezdu v jednotlivých přestupních stanicích. Budou zde uvedeny také čísla příslušných linek a spojů s možností zobrazit kompletní trasu spoje. Následovat budou informace o celkové ceně, vzdálenosti a době jízdy.

U každé přestupní stanice bude rovněž možnost zobrazit alternativní přestupní body, tj. ostatní zastávky, ve kterých oba příslušné spoje zastavují a ve kterých je tedy možné uskutečnit přestup. Toto může být užitečná informace například v případech, kdy je doba čekání na navazující spoj relativně dlouhá a je tedy výhodnější použít pro přestup autobusové stanoviště s potřebným zázemím, než odlehlou zastávku na okraji města.

2.2 Návrh databáze

Před zahájením práce na návrhu databáze je nutné stanovit, jaké informace bude potřeba uchovávat. Databáze bude uchovávat poměrně velké množství dat – předpokládaný rozsah je v řádu milionů záznamů a stovek megabytů. Proto bude nezbytné navrhnout strukturu databáze především s ohledem na výkon, tzn. že budu preferovat rychlost prováděných dotazů i za cenu většího množství dat způsobeného případnou redundancí.

2.2.1 Uchovávané informace

V databázi bude potřeba evidovat tyto údaje:

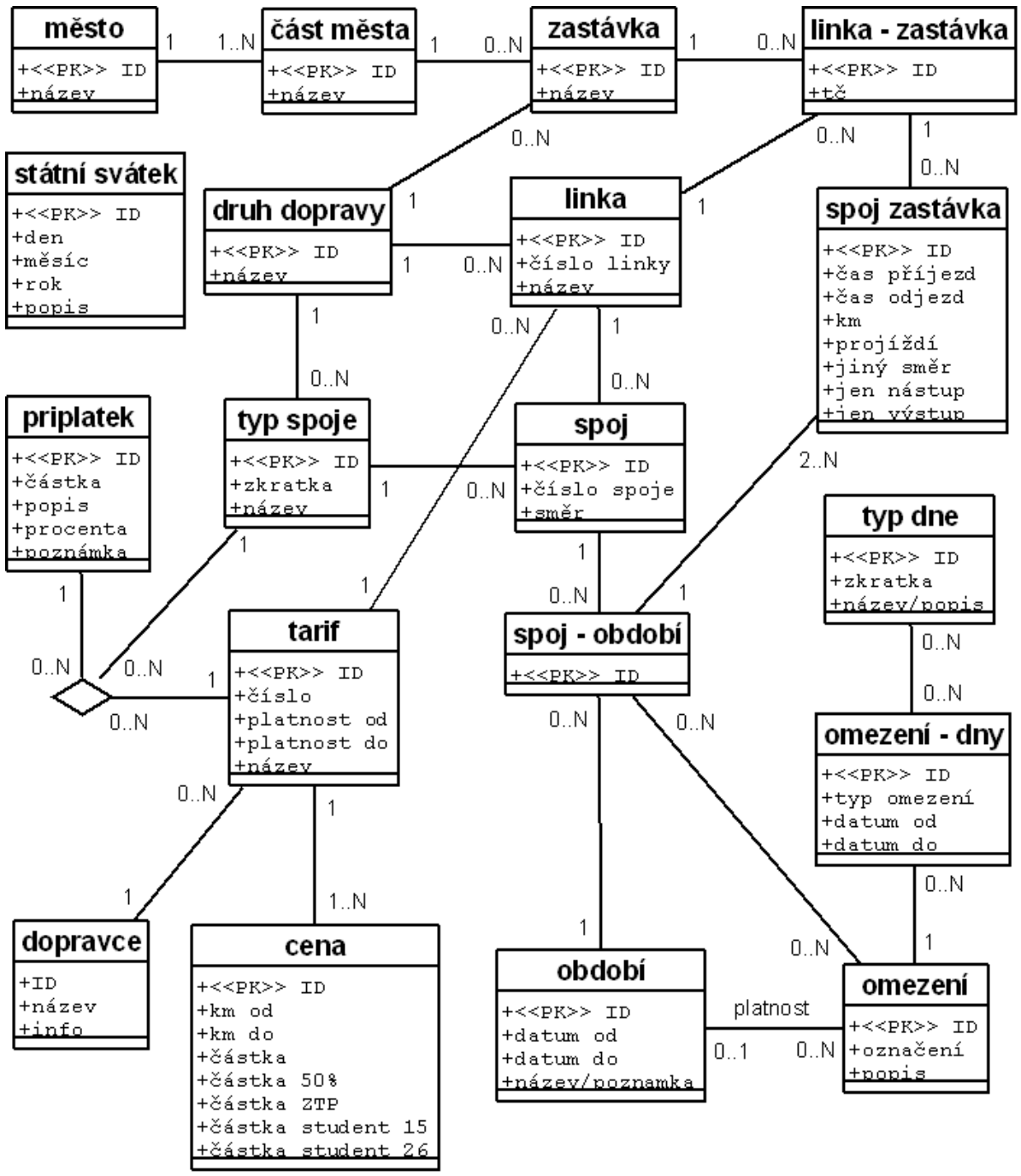
- **města**, příslušné **části měst** a **zastávky**.
- **druhy dopravy** (autobus, vlak) a k nim náležející **typy spojů** (příměstský autobus, osobní vlak, rychlík atd. – blíže popsáno v kapitole 2.1.3.
- **linky/tratě** – jedná se o trasu procházející danou posloupností dopravních uzlů (zastávek, stanic), která má pro každý druh dopravy unikátní označení (číslo). Ve vlakové dopravě je to tedy železniční trať spojující dvě koncové stanice, která může samozřejmě procházet určitým počtem jiných stanic. V autobusové dopravě se pak jedná o trasu, po které jezdí pravidelné autobusové spoje.
- **spoje** – spoj zajišťuje přepravu na konkrétní lince v daném směru (tj. ve směru „*tam*“ nebo „*zpět*“). Má přesně určen čas odjezdu z jednotlivých zastávek. Nemusí ovšem přesně kopírovat posloupnost zastávek dané linky – některé zastávky může projíždět bez zastavení, jiné míjet po jiné trase. Vzhledem k tomu, že jedním z možných kritérií je i vyhledávání podle ujeté vzdálenosti, je tento fakt velice důležitý. Pokud totiž spoj může z jedné zastávky do druhé jet po více trasách, znamená to také, že se bude lišit kilometrová vzdálenost mezi těmito dvěma zastávkami. Proto bude nutné evidovat počet kilometrů příslušný k jednotlivým zastávkám evidovat přímo u konkrétních spojů, nikoliv u linek. Mimo to je nutné zaznamenávat také čas odjezdu (popř. příjezdu) v jednotlivých zastávkách. Spoj má rovněž své unikátní označení (číslo) v rámci příslušné linky a je určitého typu (rychlík, příměstský autobus).
- **období** – další důležitou informací jsou období platnosti jednotlivých informací o spojích. Jízdní řády zpravidla platí po dobu jednoho roku, ovšem během tohoto období se může jízdní řád mírně obměňovat (zpravidla se tak děje v rozmezí tří měsíců). Pro každý spoj tedy může existovat několik jízdních řádů platných v různých obdobích. Tato období se navíc mohou překrývat.

Příklad: původní jízdní řád pro všechny spoje na autobusové lince č. 880608 platí od 9.12.2007 do 13.12.2008. Ovšem po čtvrt roce je jízdní řád pro spoje na této lince aktualizován a nová verze platí od 3.3.2008 do 13.12.2008. Původní verze ovšem musí zůstat v databázi zachována. Pokud potom bude uživatel hledat spojení např. s datem 2.3.2008, bude použita stará verze jízdního řádu, ale pro datum 3.3.2008 se již použije nová verze.

V jednotlivých verzích jízdních řádů pro konkrétní linku se mohou u některých spojů lišit časy příjezdu/odjezdu u jednotlivých zastávek. Některé spoje mohou být také úplně zrušeny nebo naopak mohou být přidány spoje nové.

- **omezení** – jedna z nejkomplikovanějších částí systému. Ke spojům mohou být přiřazena různá omezení vztahující se na jednotlivé dny v týdnu, typy dnů (pracovní den, svátek), ale také na různá období či konkrétní den v roce. Toto omezení může říkat buďto že příslušný spoj v daný den (resp. období) jede nebo naopak nejede. Navíc je možné omezení různě kombinovat, čili může vzniknout souhrn omezení říkající, že daný spoj např. „jede v pracovní dny, ale nejede v období od 1.7.2008 do 31.8.2008“, nebo „nejede v neděli a státem uznané svátky v období od 1.9.2008 do 30.9.2008“ apod. Tudíž je nutné při vyhledávání vždy určit, zda spoj v daný den jede či nejede a podle toho jej do vyhledávání zahrnout nebo nezahrnout. Samozřejmě i tato omezení se u konkrétního spoje mohou lišit v jednotlivých obdobích platnosti.
- **Dopravci, cenové tarify, příplatky** – dopravní obslužnost na každé lince je zajišťována konkrétním dopravcem. Linka má rovněž přiřazen tarif, podle něhož se řídí výpočet ceny dopravy na dané lince. U tarifů jsou evidovány ceny pro jednotlivá pásma určená rozsahem počtu ujetých kilometrů (Např. 5-7 km => 13,- Kč apod.). Pro každé pásmo je uvedena jak „běžná cena“, tak zlevněné jízdné (tj. jízdné s 50% slevou, jízdné pro osoby vlastnící průkaz ZTP, jízdné pro studenty do 15 a pro studenty od 15 do 26 let). Každý tarif má určené období platnosti, ovšem na rozdíl od výše zmiňovaného období platnosti jízdních řádů se zde jednotlivá období nesmějí překrývat. Ke každému tarifu mohou být rovněž přiřazeny příplatky vztahující se k jednotlivým typům spojů (např. příplatek za vlak EuroCity apod.). Příplatek je buďto pevně daný (tzn. nezávislý na počtu kilometrů), nebo procentní, tzn. že se cena jízdného vypočítaná na základě počtu ujetých kilometrů násobí určitým příplatkovým koeficientem.

2.2.2 ER Diagram



2.2.3 Popis ER diagramu

2.2.3.1 Město

Tato entita bude uchovávat názvy všech měst vyskytujících se v jízdních řádech.

2.2.3.2 Část města

Některá města mohou být rozdělena na více městských částí. Entita *část města* slouží k ukládání názvů těchto městských částí a kromě samotného názvu obsahuje odkaz na příslušnou nadřazenou entitu *město*. Název městské částí může být tvořen i prázdným řetězcem, a to kvůli propojení měst a zastávek. Taková městská část poté reprezentuje „centrální část“ města, případně celé město, pokud žádné další městské části vztahující se k danému městu neexistují.

2.2.3.3 Zastávka

Každá zastávka má svůj název a je přiřazena do určité městské části. Pokud dané město není rozděleno na městské části, nebo zastávka patří do „centrální části“ města je název městské části tvořen prázdným řetězcem. Tím je zaručeno, že ke každé entitě *zastávka* bude existovat příslušná entita *část města*. Sice bude nutné uchovávat prakticky pro každé město jednu městskou část navíc, ovšem na druhou stranu dojde k úspoře nároků na prostor tím, že nebude nutné u každé zastávky uchovávat odkaz na příslušné město, ale postačí odkaz na městskou část.

Entita *zastávka* dále nese informaci o druhu dopravy (dopravním prostředku), kterým je tato zastávka obsluhována. Název zastávky v každém městě a městské části musí být unikátní. Výjimkou jsou situace, kdy je v dané městské části více zastávek se stejným názvem, ale každá patří k jinému druhu dopravy.

2.2.3.4 Druh dopravy

Entita reprezentuje jednotlivé druhy dopravy (autobus, vlak).

2.2.3.5 Typ spoje

Typ spoje rozděluje jednotlivé druhy dopravy na specifické kategorie, např. dálkový autobus, příměstský autobus, osobní vlak, rychlík apod. Obsahuje název a případně zkratku používanou ve výpisech jízdních řádů.

2.2.3.6 Dopravce

Uchovává informace o jednotlivých dopravcích, tedy název a případné text s bližšími informacemi o dané společnosti. K dopravci se pak váží jím provozované linky a také příslušné cenové tarify.

2.2.3.7 Tarif

Cenový tarif vždy obsahuje identifikaci dopravce, kterým byl vyhlášen. Může být také pojmenován. Tarif je jednoznačně určen kombinací čísla „označení“ a intervalu platnosti určeného počátečním a koncovým datem. Tyto intervaly se u tarifů se stejným označením nesmí překrývat a musí na sebe navazovat. Pro snadnější odkazování z jiných entitních množin na jednotlivé záznamy je navíc přidán unikátní identifikátor ID.

2.2.3.8 Cena

Entita cena udržuje informace o jednom cenovém pásmu. Obsahuje počáteční a koncový počet kilometrů ohraničující dané tarifní pásmo a dále pak cenu běžného jízdného pro toto pásmo, cenu s 50% slevou (tzv. poloviční jízdné), cenu pro držitele průkazu ZTP, cenu pro studenty ve věku do 15 let a cenu pro studenty ve věku 15-26 let. I když jedna informace o ceně může být společná pro více tarifů, je mezi tarifem a cenou pouze vazba 1.. N. Tím se sice teoreticky zvýší celkový počet entit „cena“, ovšem při následné implementaci to umožní snížit celkový počet spojovaných databázových tabulek při zjišťování ceny spojení. Vzhledem k tomu, že tato operace se bude provádět prakticky v každém kroku vyhledávání, může mít tato úprava určitý (i když ne zásadní) pozitivní vliv na rychlost vyhledávání.

2.2.3.9 Období

Tato entita najde využití především při evidenci období platnosti jízdních řádů u jednotlivých spojů. Obsahuje datum počátku a konce období a jednoznačný identifikátor ID. Vhledem k tomu, že období platnosti jsou vždy shodná u většího množství spojů, je z hlediska prostorové náročnosti výhodnější tato data uchovávat zvlášť a odkazovat na ně pomocí ID, než ukládat u každého spoje data ohraničující začátek a konec platnosti.

2.2.3.10 Typ dne

Obsahuje informace o jednotlivých typech dne používaných v omezeních spojů, např. pracovní den, neděle nebo státem uznaný svátek apod. Za „typ dne“ je považován i konkrétní den v týdnu (pondělí – neděle), jelikož i ten může figurovat v některých omezeních.

2.2.3.11 Omezení

Entita reprezentuje omezení, které se může vztahovat k jednotlivým spojům. Kromě *označení* (např. „X“ pro pracovní dny, „6“ pro sobotu, „+“ pro neděli a státem uznané svátky či číselné označení reprezentující souhrn jiných omezení) obsahuje ještě textový popis tohoto omezení a také odkaz na entitu typu *období*, která vymezuje období platnosti tohoto souhrnu omezení. Toto období

platnosti ve většině případů koresponduje s obdobím platnosti jízdních řádů spojů, ke kterým se dané omezení vztahuje. V případě, že období není určeno, platí toto omezení stále (např. omezení „X“)

2.2.3.12 Omezení – dny

Ke každé entitě typu *omezení* se může vztahovat jedna či více entit z entitní množiny omezení – dny. Každá z těchto entit se vztahuje na konkrétní typ dne a/nebo období a pomocí atributu *typ omezení* určuje, zda spoj, k němuž je omezení přiřazeno, v tento den (resp. období) jede či naopak nejede. Omezení typu *jede* znamená, že daný spoj jede pouze v tento den (resp. období) a ne jindy (pokud tak není určeno jinou entitou typu *omezení – dny*).

2.2.3.13 Příplatek

Tato entita uchovává informace o příplatcích, které se u některých tarifů mohou vztahovat k určitým typům spojů (např. příplatek za vlak EuroCity). Kromě unikátního identifikátoru ID má dva atributy: částka a procenta. Vždy musí být zadán právě jeden z nich. Je-li zadán atribut *částka* znamená to, že je příplatek vždy konstantní bez ohledu na celkovou ujetou vzdálenost. Naopak pokud je zadán atribut *procenta*, vypočítá se příplatek jako procentní přirážka k ceně cesty vypočítané dle příslušného tarifu.

2.2.3.14 St. svátek

Číselník obsahující všechny státem uznané svátky. Obsahuje název svátku, den, měsíc a případně rok. Pokud rok není uveden, pak je tento svátek každý rok ve stejný den. U svátků, jejichž datum se může v jednotlivých letech lišit (např. Velikonoční pondělí), je vždy uveden i rok.

2.2.3.15 Linka

Tato entita reprezentuje linku (trať) tak, jak byla popsána v kapitole 2.2.1. Kromě číselného označení a názvu linky nese informaci o příslušejícím druhu dopravy a o přiřazeném tarifu.

2.2.3.16 Linka – zastávka

Vazební množina *linka – zastávka* přiřazuje k linkám konkrétní zastávky, kterými tato linka prochází a které jsou obsluhovány spoji provozovanými na této lince. Pořadí zastávek lze určit z atributu *tč* (tarifní číslo).

2.2.3.17 Spoj

Entita reprezentující spoj zajišťující přepravu na konkrétní lince. Atribut *směr* nabývá jedné z hodnot *t* (tam), *z* (zpět) a určuje, v jakém pořadí spoj projíždí jednotlivé zastávky na dané trase.

2.2.3.18 Spoj – období

Jedná se o vazební entitní množinu reprezentující vztah mezi entitami typu *spoj* a *období*. Některé vlastnosti spoje, jako například omezení nebo čas průjezdu zastávkami, se totiž mohou v každém období lišit. Proto není možné tyto vlastnosti vztahovat přímo ke spoji, ale je nutné rozlišit jednotlivá období platnosti.

2.2.3.19 Spoj – zastávka

Další vazební entitní množina, která k entitě typu spoj – období přiřazuje zastávky, jimiž spoj (v daném období) projíždí. Entita současně nese další atributy této vazby, kterými jsou:

- *čas příjezdu*, *čas odjezdu* – tyto atributy určují čas příjezdu/odjezdu spoje do/ze zastávky. U výchozí zastávky spoje musí být vždy uveden čas odjezdu a u cílové zastávky čas příjezdu. U ostatních zastávek můžou být uvedeny buďto oba časy nebo pouze čas odjezdu (v případě, že se spoj v dané zastávce po svém příjezdu zdrží jen po dobu nutnou pro nástup a výstup cestujících)
- *další den* – tento atribut značí, že spoj přijíždí do dané zastávky v jiný den, než kdy odjel ze zastávky výchozí.

Příklad: pokud spoj vyjíždí ze zastávky A ve 23:40, v zastávce B zastavuje ve 23:55 a do zastávky C přijede až v 0:15 (tj. až v další den), bude u zastávek A a B tento atribut nabývat hodnoty 0, ale u zastávky C už to bude 1.

Tím se usnadní výpočet doby cesty, který se odvíjí právě od rozdílů časových údajů ve dvou mezních zastávkách.

- *km* – kilometrická vzdálenost od výchozí zastávky.
- *projíždí* – atribut říkající, že spoj zastávkou projíždí bez zastavení.
- *jiný směr* – značí, že spoj zastávku míjí po jiné trase
- *jen nástup*, *jen výstup* – tyto atributy říkají, že spoj v dané zastávce zastavuje pouze pro nastupování nebo naopak vystupování.

3 Metoda vyhledávání

Nejdůležitějším bodem celého systému je samotná vyhledávací metoda. Protože jedním z hlavních požadavků na systém je možnost vyhledávání podle různých kriterií (vzdálenost, cena…), je nutné zvolit metodu prohledávání stavového prostoru, která toto umožní. Dalšími podstatnými parametry metody jsou časová a paměťová náročnost. Je žádoucí, aby vyhledání požadovaného spojení proběhlo v co nejkratším čase a také aby nároky na paměť byly únosné (hranicí této únosnosti se zabývám v kapitole 4.2 Webhosting).

3.1 Stavový prostor

Než přistoupím k samotnému výběru metody prohledávání stavového prostoru, je vhodné nadefinovat, co bude pro potřeby této práce považováno za stavový prostor. Prvotní myšlenka byla považovat za stavový prostor graf, jehož uzly budou představovat jednotlivé zastávky a hrana (přechod) mezi dvěma konkrétními uzly bude reprezentovat nalezené přímé spojení mezi zastávkami, jež tyto uzly představují (samozřejmě pouze v případě, že mezi nimi přímé spojení existuje). Vzhledem k požadavku vyhledávat spojení podle různých kritérií, bude důležité i ohodnocení přechodů, které bude záviset na zvoleném kritériu. Pokud tedy mezi dvěma uzly existuje více možných přímých spojení, je nutné brát v potaz pouze to s nejlepším ohodnocením.

Problém ovšem nastane ve chvíli, kdy je potřeba zakomponovat do systému další požadované vlastnosti: minimální a maximální čas na přestup. Může totiž nastat situace, kdy z uzlu A do uzlu B (kde B není cílový) existují dvě různá spojení, každé v jiném čase. Jedno z nich má lepší ohodnocení, takže je zvoleno jako směrodatné pro ohodnocení dané hrany. Záhy se ovšem ukáže, že z uzlu B do cíle neexistuje v požadovaném čase žádné navazující spojení, nebo že navazující spojení existuje, ale výsledné celkové ohodnocení cesty je horší, než kdyby bylo z bodu A do bodu B použito druhé, sice hůře ohodnocené spojení z A do B, ale s výhodnějším navazujícím spojením z B do cíle.

Dalším požadavkem na systém byla možnost zvolit maximální akceptovatelný počet přestupů, což situaci ještě více zkomplikuje. Pokud totiž mezi dvěma uzly neexistuje přímé spojení, ale je nutné jet přes jeden či více přestupních uzlů, je pak celkové ohodnocení dáno součtem ohodnocení všech přechodu na této cestě. V případě, že z uzlu A do uzlu B (který není cílový) existuje více možných cest, je žádoucí zvolit pro další vyhledávání tu, která má nejnižší celkové ohodnocení. Ostatní cesty do uzlu B jsou tedy zahozeny a dále je brána v potaz pouze ta s nejnižším ohodnocením. Problémem ovšem je, že tato zvolená cesta může vést přes více uzlů, než jiná cesta, která má sice horší celkové ohodnocení, ale menší počet přestupů. V případě, že je tedy omezena hloubka vyhledávání, tj. je omezen maximální možný počet přestupů, může dojít k situaci, kdy v některém z následujících kroků

tato zvolená cesta přestane vyhovovat dané podmínce a je označena jako cesta, která nevede k cíli. Přitom pokud by z bodu A do B zvolena cesta jiná, s horším hodnocením ale menším počtem přestupů, mohlo by být následné hledání úspěšné.

Z toho plyne, že výše uvedená představa stavového prostoru je pro realizaci nevhodná. Uzel nemůže představovat pouze zastávku, ale musí zahrnout ještě další parametry. Kromě samotného identifikátoru zastávky musí být uzel určen ještě datem a časem příjezdu spoje do zastávky (kvůli minimálnímu a maximálnímu času na přestup) a počtem realizovaných přestupů. Kombinace těchto tří parametrů musí být unikátní pro každý uzel stavového prostoru.

Nyní je tedy možné přistoupit k samotné volbě vhodné metody prohledávání stavového prostoru.

3.2 Metoda UCS

3.2.1 Popis metody

Metoda stejných cen (Uniform Cost Search) se řadí k neinformovaným metodám prohledávání stavového prostoru. Neinformované (nebo také slepé) metody vyhledávání se vyznačují tím, že nemají v průběhu vyhledávání žádnou informaci o cílovém stavu. Nemohou tedy nijak rozhodnout, který uzel je z hlediska dosažení cíle nejvýhodnější pro následnou expanzi.

Metoda UCS ovšem neprohledává stavový prostor úplně náhodně. Na základě ceny již provedených přechodů dokáže ohodnotit aktuální stav a pro následnou expanzi vždy vybírá stav s nejlepším ohodnocením. Pro uložení expandovaných uzlů je tedy použit seznam OPEN, ze kterého je vždy pro následující expanzi vybrán uzel s nejlepším ohodnocením.

3.2.2 Implementace pomocí metody UCS

V první fázi jsem se tedy rozhodl implementovat vyhledávací algoritmus s využitím metody UCS. Postup byl následující:

1. Program vytvoří seznam OPEN a vloží do něj výchozí zastávku, popř. více zastávek.
2. Pokud je seznam prázdný, vyhledávání končí neúspěšně.
3. Ze seznamu OPEN je vybrán uzel s nejlepším ohodnocením. Toto ohodnocení je vypočítáno již v době vkládání uzlu do seznamu a závisí na zvoleném kritériu. Může jím být vzdálenost od výchozího bodu, doba cesty, cena apod.
4. Je-li tento uzel cílový (tj. daná zastávka patří do množiny cílových zastávek), funkce vrátí tento uzel jako cílový k dalšímu zpracování. Vyhledávání tímto končí.

5. Bylo-li v uzlu dosaženo maximálního povoleného počtu přestupů, je tento uzel vyřazen a běh programu se vrací na bod 2.

6. Jsou vyhledány všechny zastávky, do kterých existuje z tohoto uzlu v daný den a daný čas přímé spojení a které vyhovují dalším kritériím, např. že typ dopravního spoje, který toto spojení realizuje, patří do množiny povolených typů zvolených uživatelem apod. Jejich ohodnocení se určí součtem ohodnocení tohoto spojení a ohodnocení uzlu, ze kterého vycházíme. Takto vyhledané uzly jsou ohodnoceny dle zvoleného kritéria a vloženy do seznamu OPEN. Běh programu začíná znovu od bodu 2.

Pro snížení paměťové náročnosti jsem se rozhodl provést ještě mírné úpravy tohoto algoritmu.

První zásadní úpravou bylo zamezení několikanásobného vkládání téhož uzlu do seznamu OPEN. Jak již bylo řečeno, uzel je jednoznačně určen identifikátorem zastávky, časem příjezdu spoje do zastávky a počtem uskutečněných přestupů. Pokud by tedy došlo k situaci, že lze do jedné zastávky dojet více spoji ve stejný čas a se stejným počtem přestupů, je tento vzniklý uzel vložen do seznamu OPEN pouze jednou.

Dalším krokem vedoucím ke snížení paměťové náročnosti bylo zabránění návratu do již navštívených zastávek v rámci jedné cesty. Toto se děje bez ohledu na čas v zastávce a počet přestupů. Je to logické: pokud již spoj danou zastávkou jednou projede, nemá smysl se do této zastávky v rámci stejného spojení znovu vracet některým z navazujících spojů. Zároveň se tím zamezí „zacyklení“, tj. situaci, kdy by byly stále dokola vyhledávány spoje mezi dvěma blízko ležícími zastávkami s nízkým ohodnocením přechodu.

Také jsem upustil od původního plánu využít seznam CLOSED, do kterého by se ukládaly všechny uzly vyjmuté ze seznamu OPEN, a který by zamezil pozdějšímu navštívení tohoto uzlu v rámci kteréhokoliv spojení. Ukázalo se, že k situacím, kdy je jedna zastávka navštívena v rámci více spojení ve stejný čas a se stejným počtem uskutečněných přestupů, nedochází příliš často. Seznam CLOSED by tedy našel využití jen zřídka, ovšem výrazně by zvýšil paměťovou náročnost.

Během implementace a průběžného testování se ukázalo, že takto modifikovaná metoda stejných cen sice poskytuje správné a poměrně rychlé výsledky při vyhledávání spojení mezi dvěma méně vzdálenými zastávkami[1], kde celkové ohodnocení cesty je relativně nízké. Ovšem při pokusu vyhledat spojení mez dvěma velice vzdálenými místy dochází k neúměrnému nárůstu paměťové i časové náročnosti. Program postupně prochází nejbližší možné zastávky ve všech směrech od počátečního bodu a ke vzdálenějším místům se propracovává velice pomalu. Navíc po chvíli dojde k selhání z důvodu nedostatku paměti.

Příklad: Při pokusu vyhledat spojení mezi městy Nový Jičín a Kopřivnice, která jsou vzdálena cca 15 km najde program správný výsledek, a to v čase cca 10 sekund. Při vyhledávání na trase

1 Za předpokladu, že jako kritérium pro vyhledávání je zvolena co nejmenší vzdálenost

Nový Jičín – Praha už ovšem dojde k selhání kvůli nedostatku paměti, protože systém by byl nucen projít postupně všechny zastávky, u nichž existuje spojení z Nového Jičína a jejichž vzdálenost od Nového Jičína je menší než vzdálenost Nový Jičín – Praha (tj. cca 360 km). Nehledě na to, že některé zastávky mohou být z dříve uvedených důvodů (minimální a maximální čas na přestup apod. – viz. kapitola 3.1) v seznamu OPEN uloženy vícekrát.

Ukázalo se tedy, že metoda UCS je pro tento systém nevhodná a bude potřeba najít jinou, paměťově i časově méně náročnou metodu prohledávání stavového prostoru.

3.3 Metoda A* search

3.3.1 Popis metody

Metoda A* (A s hvězdičkou) patří do skupiny informovaných metod zvané Best First Search. Jak již název napovídá, jedná se o skupinu metod prohledávání stavového prostoru založených na výběru vždy nejlépe ohodnoceného stavu. Je zde tedy jistá podobnost s výše popsanou neinformovanou metodou UCS. Zásadním rozdílem však je, že informované metody (na rozdíl od neinformovaných) mají v průběhu prohledávání stavového prostoru určitou informaci o cílovém stavu, na základě čehož dokáží ohodnotit aktuální stavy.

Zatímco v metodě UCS bylo ohodnocení aktuálního uzlu dáno pouze hodnotou ceny cesty z počátečního do aktuálního uzlu, u metod Best First Search má toto ohodnocení ještě další složku, kterou je odhad ceny cesty z aktuálního do cílového uzlu. Ohodnocení uzlu n lze tedy vyjádřit jako $f(n)=g(n)+h(n)$, kde $g(n)$ je cena cesty z počátečního uzlu do uzlu n a $h(n)$ je hodnota heuristické funkce, neboli odhad ceny cesty z n do cílového uzlu.

Volba správné heuristiky je velice důležitá. Čím přesnější je odhad zbývající ceny cesty, tím méně uzlů je nutné při vyhledávání projít a tím nižší je paměťová náročnost. U metody A* musí být heuristická funkce tzv. spodním odhadem skutečné ceny cesty od aktuálního uzlu k cíli. Je-li tomu tak, pak lze takovou heuristickou funkci označit za přípustnou heuristickou funkci (přípustnou heuristiku). V případě použití přípustné heuristiky je metoda A* optimální, tzn. že nalezené řešení má vždy nejnižší možnou cenu. Stejně jako metoda UCS je A* rovněž úplná.

3.3.2 Heuristika

Jak již bylo řečeno, je potřeba zvolit takovou heuristiku, která bude dobrým spodním odhadem skutečné ceny zbývající cesty a tím zajistí, že část stavového prostoru kterou bude nutné prohledat bude co nejmenší.

Vzhledem k tomu, že výpočet hodnoty heuristické funkce se provádí v každém kroku procházení stavového prostoru, má rychlost tohoto výpočtu značný vliv na celkovou dobu vyhledávání. Jako nejpřijatelnější řešení se tedy jeví mít veškeré heuristické informace (např. vzdálenosti mezi jednotlivými dvojicemi míst) trvale uloženy v databázi a během samotného vyhledávání je pouze číst bez nutnosti dalších výpočtů. Toto řešení ovšem na druhou stranu povede k poměrně vysokým nárokům na diskový prostor. Proto bude nutné omezit množství uložených informací na úkor přesnosti heuristiky.

Jak bylo uvedeno v kapitole 2.1.2, systém musí umět vyhledávat na základě různých kritérií. Od toho se také odvíjí způsob ohodnocení uzlů při prohledávání stavového prostoru. Základem pro výpočet tohoto ohodnocení může být buďto vzdálenost, doba cesty nebo cena, v závislosti právě na zvoleném kritériu. Proto je nutné uchovávat heuristickou informaci pro všechny tyto možné varianty ohodnocení. Pro každou dvojici míst tedy musí být k dispozici informace o nejkratší možné vzdálenosti mezi těmito místy, o nejkratším čase, v jakém je možné realizovat cestu mezi těmito místy, a o nejnižší dosažitelné ceně této cesty.

Samozřejmě že tyto informace nebudou vždy zcela přesné. Přepravu mohou zajišťovat různí dopravci s různými cenovými tarify. Navíc spoje mají různá omezení a nemusí na sebe navazovat v požadovaném časovém intervalu, tudíž je možné, že v uživatelem daný den a hodinu nebude možné tuto cestu realizovat s nejnižší možnou cenou (resp. vzdáleností, časem) a výsledná cena se tedy může (v některých případech výrazně) lišit od uložené heuristické hodnoty.

Otázkou zůstává, pro jaké dvojice entit budou heuristické informace uchovávány. V ideálním případě by to měly být všechny dvojice zastávek, což by zajistilo poměrně dobrou přesnost heuristických hodnot. Problém je ovšem s již zmiňovanou náročností na diskový prostor. Zároveň platí, že čím obsáhlejší bude databáze heuristických údajů, tím delší bude čas potřebný k jejich vyhledání, což, jak již bylo řečeno, ovlivní i celkovou dobu běhu programu.

Nyní je tedy nutné zjistit, jak náročné by bylo uložení heuristických údajů pro všechny možné dvojice zastávek. Použiji k tomu vzorec pro výpočet všech možných kombinací čísel, přičemž nezáleží na pořadí:

$$K(k,n)=\frac{n!}{k!*(n-k)!}$$

V tomto případě $k=2$ (hledám dvojice čísel) a n je počet všech zastávek. Vzhledem k tomu, že testovací data v současné době obsahují 31 331 různých zastávek, výsledný počet záznamů s heuristickými údaji by byl:

$$K(2,\ 31\ 331) = \frac{31\ 331!}{2!*(31\ 331-2)!} = 490\ 800\ 115$$

Vezmu-li v potaz, že každý záznam musí obsahovat identifikátory obou zastávek, vzdálenost, čas a cenu, byly by prostorové nároky v tomto případě neúnosné. Navíc vyhledávání v takovém množství dat by bylo velice časově náročné, což je v rozporu z výše uvedenými požadavky kladenými na heuristickou funkci. Je také nutné brát v potaz, že v současné době jsou v databázi pouze autobusové zastávky. V případě rozšíření i o vlakové stanice by počet potřebných heuristických záznamů ještě výrazně narostl.

Nezbývá tedy než přistoupit k jiné variantě, a tou je uložení daných heuristických informací pro dvojice měst. Dojde tím sice k jisté ztrátě přesnosti (zvláště u rozlehlejších měst, kde je velká vzdálenost mezi zastávkami ležícími na opačných koncích města), ovšem na druhou stranu dojde k výraznému snížení prostorové náročnosti. V databázi se v současné době nachází 5 731 měst. Celkový počet heuristických záznamů tedy bude

$$K(2,\ 5\,731) = \frac{5\,731!}{2!*(5\,731-2)!} = 16\,419\,315$$

což je výrazně méně než v předchozím případě. Je to sice stále relativně vysoký počet záznamů, ovšem z hlediska prostorové náročnosti už je přijatelný. Dá se rovněž předpokládat, že na rozdíl od počtu zastávek se počet měst nebude v budoucnu výrazně měnit a proto lze toto výsledné množství heuristických záznamů považovat za konečné.

3.3.3 Implementace pomocí metody A*

Celý vyhledávací algoritmus je velice podobný algoritmu použitému u metody UCS (viz. kapitola 3.2.2) s jediným zásadním rozdílem. Ten spočívá ve výpočtu ohodnocení expandovaných uzlů, kdy je k aktuální ceně cesty připočtena hodnota heuristické funkce pro daný uzel. Stejně jako u implementace pomocí metody UCS zůstala zachována i rozšíření spočívající v zamezení vícenásobného vkládání téhož uzlu do seznamu OPEN a v zabránění návratu k dříve navštíveným uzlům v rámci jedné cesty.

V této podobě již systém dosahuje správných výsledků v relativně přijatelném čase.

3.3.4 Rychlé vyhledávání

Po implementaci a následném otestování systému s využitím metody A* jsem se rozhodl ještě pro jednu mírnou modifikaci. Uživatel bude mít možnost ještě před započetím samotného vyhledávacího procesu zvolit možnost tzv. rychlého vyhledávání. Tato volba způsobí, že test, zda je daný uzel cílový, nebude probíhat až při jeho vyjmutí ze seznamu OPEN, nýbrž ještě před jeho vložením do tohoto seznamu, tj. při vyhledávání následníků aktuálního uzlu.

Celý algoritmus pak bude vypadat následovně:

1. Program vytvoří seznam OPEN a vloží do něj výchozí zastávku (více zastávek).
2. Pokud je seznam OPEN prázdný, vyhledávání končí neúspěšně.
3. Ze seznamu OPEN je vybrán uzel s nejlepším ohodnocením.
4. Jsou vyhledány všechny zastávky, do kterých existuje z tohoto uzlu v daný den a daný čas přímé spojení a které vyhovují dalším kritériím (stejně jako u metody UCS). Vyhledané uzly jsou ohodnoceny na základě součtu ceny dosavadní cesty a hodnoty heuristické funkce.
5. Je-li jeden z takto vyhledaných uzlů uzlem cílovým, funkce jej vrátí jako výsledek hledání k dalšímu zpracování. V případě že je v tomto kroku nalezeno více cílových uzlů, vrátí funkce jeden z nich, a to ten s nejlepším ohodnocením. Vyhledávání tímto končí.
6. Pokud bylo u některých necílových uzlů dosaženo maximálního povoleného počtu přestupů, jsou vyřazeny.
7. Ostatní vyhledané uzly jsou vloženy do seznamu OPEN a běh programu se vrací na bod 2.

Prohledávání stavového prostoru je tedy ukončeno hned jakmile je dosaženo některého z cílových uzlů i přesto, že se v seznamu OPEN mohou v té době nacházet ještě jiné uzly s nižším ohodnocením, které rovněž mohou vést k cíli. Není tedy zaručeno, že výsledné řešení má nejnižší možné ohodnocení. Vyhledávací metoda tím tedy ztratí optimálnost, ovšem na druhou stranu dojde k výraznému urychlení celého vyhledávacího procesu.

3.3.5 Zpětné vyhledávání

Jak je uvedeno v kapitole 2.1.1, uživatel má možnost zadat buďto datum a čas, kdy chce odjet z výchozího místa, nebo datum a čas požadovaného příjezdu do místa cílového. V prvním případě je tedy stavový prostor prohledáván postupně od výchozího místa k místu cílovému (dále jako *klasické vyhledávání*). Ve druhém případě je ovšem nutné realizovat vyhledávání opačným směrem (dále jako *zpětné vyhledávání*).

To v prvé řadě znamená, že výchozí zastávka (resp. zastávky) bude označena za cílovou a naopak. V každém kroku vyhledávání se pak hledají zastávky, ze kterých existuje do aktuální zastávky přímé spojení.

Princip ohodnocení uzlů je ve většině případů (tzn. vzdálenost, cena, počet přestupů nebo doba cesty) stejný jako u klasického vyhledávání. Výjimkou je ovšem vyhledávání při zvoleném kritériu „co nejdříve v cíli“. V případě klasického vyhledávání je zde ohodnocení uzlu dáno předpokládaným datem a časem příjezdu do cíle, tzn. za nejlépe ohodnocený uzel je považován ten, jehož předpokládaný čas příjezdu do cíle je nejmenší. V případě zpětného vyhledávání bude ovšem tato podmínka opačná. Ohodnocení uzlu je zde totiž dáno předpokládaným datem a časem odjezdu

z výchozího místa. Za nejlépe ohodnocený uzel je tedy považován ten, jehož ohodnocení je nejvyšší, tj. předpokládaný čas odjezdu z výchozího místa je největší.

4 Implementace systému

4.1 Použité technologie

4.1.1 PHP

Pro implementaci jádra systému jsem zvolil skriptovací jazyk PHP. Systém byl vyvíjen a testován na serveru obsahujícím verzi PHP 5.2.6, ale je kompatibilní se všemi dosud existujícími verzemi PHP od verze 5.0. PHP 5[2] se vyznačuje především výrazně lepší podporou objektově orientovaného programování než předchozí verze.

Nově je zde možnost specifikovat viditelnost atributů a metod objektů, tzn. dle potřeby dělit atributy a metody na veřejné (public), chráněné (protected) a privátní (private). Výhodou je rovněž možnost používání statických atributů a metod. PHP 5 se od předchozích verzí liší rovněž předávání parametrů funkcím: je-li parametrem instance objektu, předává se funkci pouze odkaz na tento objekt, nikoliv kopie objektu jak tomu bylo v PHP v. 4. Tímto se PHP opět přiblížilo „klasickým“ objektově orientovaným jazykům.

Velkým pozitivem jazyka PHP je rovněž poměrně dobrá podpora práce s databázovým systémem MySQL.

4.1.2 MySQL a databázová vrstva

Pro uložení perzistentních dat jsem využil databázový systém MySQL ve verzi 5.0 nebo vyšší. MySQL od verze 4.1 umožňuje používat tzv. poddotazy (vnořené dotazy), využívá paměť cache pro uložení výsledků dotazů a jejich případné pozdější použití, čímž dojde k urychlení získávání dat. Ve verzi 5.0 navíc přibyla podpora uložených procedur a funkcí.

Pro práci s databází jsem použil vlastní PHP knihovnu. Jedná se vlastně o třídu zapouzdřující všechny databázové operace jako je provádění dotazů, získávání výsledků dotazů, zjištění počtu vrácených záznamů atd. Tato třída v současné době umí pracovat pouze s databázovým systémem MySQL, ovšem není problém ji rozšířit o práci s dalšími databázovými systémy (např. Oracle apod.). Tím by byl umožněn přechod celého systému na jiný databázový systém bez nutnosti zasahovat do zdrojových kódů. Tuto třídu pro práci s databází lze nalézt v modulu *sql.php*.

2 Označení PHP 5 zde používám pro PHP ve verzi 5.0 nebo vyšší

4.1.3 XHTML, CSS, Javascript, AJAX

Program generuje výstup v jazyce XTTML s využitím kaskádových stylů (CSS). Je tedy určen především ke zobrazení ve webových prohlížečích.

Pro kontrolu údajů zadaných do formuláře (např. kontrola správného formátu času či zadání všech povinných údajů) je využit Javascript, který je prováděn na straně klienta a tím pádem nedochází ke zbytečné komunikaci mezi klientem způsobené odesláním chybně zadaných údajů. Stejná kontrola je samozřejmě prováděna i na straně serveru, protože lze očekávat i uživatele, kteří používají prohlížeče nepodporující Javascript nebo mají podporu Javascriptu záměrně vypnutou.

Další zajímavou a v poslední době poměrně oblíbenou technologií je AJAX (Asynchronous JavaScript and XML), který umožňuje měnit část obsahu webové stránky bez nutnosti opětovného načtení celého XHTML dokumentu. Dojde tak k omezení množství dat přenášeného mezi klientem a serverem, protože server vždy posílá pouze část obsahu dokumentu, která se změnila, a nikoliv celý dokument. Pomocí AJAXu je realizováno např. vyhledání a zobrazení alternativních přestupních bodů ve výsledcích hledání nebo vyhledání dřívějších či pozdějších spojů oproti nalezenému výsledku.

4.2 Webhosting

Systém musí být umístěn na webovém serveru podporujícím technologie PHP 5.0 a vyšší a MySQL 5.0 a vyšší. Také je nutné počítat s poměrně velkými požadavky na diskový prostor. Především MySQL databáze obsahuje více než 1GB dat (z čehož cca 70% připadá na tabulku udržující heuristické informace). Na českém trhu sice působí několik firem, které poskytují webhosting zdarma, ovšem žádná z nich nenabízí dostatečný diskový prostor pro umístění tohoto systému. Jejich služeb tedy není možné využít ani pro testovací provoz systému.

Dalším problémem je, že většina webhostingových firem má v konfiguraci omezenu maximální možnou dobu běhu PHP skriptu (direktiva *max_execution_time* v konfiguračním souboru *php.ini*) na 30 sekund. To v některých případech nemusí stačit k vyhledání výsledného spojení. Proto je třeba systém umístit na server, kde bude možné toto nastavení upravit a prodloužit maximální povolenou dobu běhu skriptu.

Rozhodl jsem se tedy využít možnosti umístit systém v průběhu implementace a testování na server, který je umístěn v Brně, je prakticky nepřetržitě v provozu a patří mému známému, p. Jaroslavu Čechovi. Ten mi vyšel vstříc nejen v požadavku na umístění systému, ale i v požadavcích na změny v konfiguraci serveru (navýšení časového limitu apod.).

Interpret PHP skriptů ovšem může alokovat maximálně 128 MB paměti (direktiva *memory_limit* v konfiguračním souboru *php.ini*). Na toto omezení bylo nutné brát ohled nejen při

výběru vyhledávací metody s co nejnižší paměťovou náročností, ale také při volbě použitých datových struktur v průběhu implementace.

4.3 Implementace vyhledávání

Jádrem systému je třída pro samotné vyhledávání spojů. Ta se nachází v modulu *connection.php* a obsahuje veškeré atributy a metody potřebné pro vyhledání spojení. Po vyplnění vyhledávacího formuláře uživatelem se vytvoří instance této třídy a její příslušné atributy jsou nastaveny dle údajů z formuláře. Poté je spuštěn samotný vyhledávací proces.

Algoritmus použitý pro vyhledávání jsem popsal v kapitole 3.3.3 a proto se k němu již detailně vracet nebudu. Co bych ovšem rád zmínil je implementace seznamu OPEN a způsob uložení jednotlivých uzlů stavového prostoru v paměti.

4.3.1 Uzly stavového prostoru

Pro uložení jednotlivých uzlů stavového prostoru jsem využil vestavěného abstraktního datového typu, který jazyk PHP nabízí a jímž je asociativní pole. Asociativní pole se vyznačuje tím, že jeho prvky nejsou indexovány celými čísly, jak je tomu u klasického datového typu pole, nýbrž pomocí klíčů. Klíčem v jazyce PHP může být buďto celé číslo nebo řetězec.

Vzhledem k tomu, že každý uzel ponese více údajů a je vhodné mít tyto údaje pojmenované a snadno rozlišitelné, jeví se volba asociativního pole jako výhodná. Klíčem v poli tedy bude jméno daného atributu a hodnotou bude informace, jíž tento atribut nese. Struktura tohoto pole je znázorněna v tabulce 1:

Klíč	**Datový typ hodnoty**
id_station	integer
rating	integer nebo string
end_station	boolean (integer 0 nebo 1)
changes_cnt	integer
arrive_datetime	string
total_distance	integer
total_time	integer
total_price	float
route	array
visited_stations	array

Struktura uzlu stavového prostoru

Jednotlivé prvky pole ponesou následující informace:

- **id_station –** identifikátor příslušné zastávky.
- **rating** – ohodnocení daného uzlu v závislostí na zvoleném kritériu pro vyhledávání daná součtem dosavadní ceny cesty a předpokládané ceny zbytku cesty. Může nabývat buďto celočíselné hodnoty (vzdálenost v kilometrech, doba cesty v minutách, cena cesty zaokrouhlená na nejbližší nižší celé číslo) nebo hodnoty typu řetězec (předpokládané datum a čas příjezdu do cílové stanice ve tvaru „RRRR-MM-DD HH: mm“[3]). Vzhledem k tomu že datový typ ohodnocení je vždy stejný pro všechny uzly, je zajištěno, že bude možné ohodnocení jednotlivých uzlů mezi sebou porovnávat, bez ohledu na to, které kritérium je v daném případě zvoleno.
- **end_station** – nabývá-li tato položka hodnoty *true* (resp. 1) znamená to, že daný uzel patří do množiny cílových uzlů a má být při výběru ze seznamu upřednostněn před ostatními uzly se stejným ohodnocením. Pokud se totiž v seznamu OPEN nachází více uzlů se stejným ohodnocením, je jejich pořadí nedefinované. Ovšem v případě, že se mezi nimi nachází i uzel cílový, je výhodnější jako první vybrat ze seznamu tento cílový uzel, čímž vyhledávání skončí, než zbytečně expandovat další stejně ohodnocené necílové uzly. Tím dojde k mírnému urychlení vyhledávacího procesu.
- **changes_cnt** – obsahuje počet přestupů uskutečněných během cesty k aktuálnímu uzlu.
- **arrive_datetime** – datum a čas příjezdu spoje do dané zastávky ve tvaru „RRRR-MM-DD HH: mm“.
- **total_distance** – vzdálenost v kilometrech mezi výchozí a aktuální zastávkou.
- **total_time** – doba cesty z výchozí do aktuální zastávky (v celých minutách).
- **total_price** – dosavadní cena cesty
- **route** – pole záznamů informujících o dosavadním průběhu cesty (viz. dále)
- **visited_stations** – pole obsahující identifikátory zastávek ležících na dosud projeté trase (i když se nejedná o přestupní uzly). Využívá se pro zabránění návratů do takovýchto zastávek, což pomáhá snížit paměťovou náročnost.

Parametr *route* uchovává informace o průběhu cesty z výchozího do aktuálního uzlu a umožňuje tak později tuto cestu kompletně zrekonstruovat. Jedná se o klasicky indexované pole, jehož každý prvek tvoří asociativní pole s následujícími prvky:

- **id_station** – obsahuje identifikátor výchozí zastávky pro daný úsek cesty.
- **id_connection** – identifikátor spoje, kterým byl daný úsek realizován.

3 RRRR – čtyřciferné označení roku, MM – dvojciferná reprezentace měsíce, DD – dvojciferné označení dne v měsíci, HH – hodiny (dvojciferně), mm – minuty (dvojciferně)

Tyto dva údaje plně postačují k pozdější rekonstrukci cesty z výchozí do cílové zastávky. S jejich pomocí lze z databáze získat veškeré údaje potřebné k výpisu nalezených spojení, jako jsou názvy zastávek, časy příjezdů a odjezdů spojů apod.

4.3.2 Seznam OPEN

Vzhledem k tomu, že jazyk PHP neumožňuje definici vlastních datových typů, není zde možná „klasická“ implementace lineárního seznamu pomocí ukazatelů. Rozhodl jsem se tedy pro implementaci seznamu OPEN pomocí datového typu pole, případně asociativního pole. Funkce pro práci se seznamem musí splňovat dva základní požadavky:

1. Při vyjmutí vrátit vždy uzel s nejlepším ohodnocením.
2. Při vkládání do seznamu zamezit vložení duplicitních uzlů[4]

Protože vkládání a vyjímaní prvků ze seznamu OPEN jsou dvě nejčastěji prováděné operace při vyhledávání, bude mít časová náročnost těchto operací značný vliv na celkovou dobu potřebnou k dosažení výsledku. Je také nutné si uvědomit, že operace vkládání uzlu do seznamu se provádí častěji než operace vyjmutí uzlu. Tento rozdíl závisí na parametru nazývaném *faktor větvení*. Ten určuje průměrný počet bezprostředních následníků každého uzlu a závisí na zvolených parametrech vyhledávání, především na minimálním a maximálním času na přestup. Je jasné, že čím delší bude časový interval daný těmito dvěma parametry, tím více spojů z dané zastávky v tomto intervalu odjede a tím více následníků daného uzlu bude tedy vygenerováno. Faktor větvení je tedy pro každou kombinaci vyhledávacích parametrů jiný a může v extrémních případech dosahovat až hodnot v řádu několika desítek.

Postupně jsem tedy implementoval a testoval několik různých variant implementace seznamu OPEN, z nichž vybírám čtyři, které dále popisuji a srovnávám.

4.3.2.1 Varianta 1: Implementace pomocí klasicky indexovaného pole

První variantou je ukládat jednotlivé uzly stavového prostoru do klasicky indexovaného pole.

Vyhledání a vyjmutí nejlépe ohodnoceného uzlu je možné dvěma způsoby. Buďto projít celé pole a zjistit index nejlépe ohodnoceného prvku, nebo využít možnosti, kterou jazyk PHP nabízí a kterou je seřazení pole na základě uživatelsky definované funkce. V tomto případě by funkce porovnávala ohodnocení jednotlivých prvků pole a další faktory, jako například zda daný prvek představuje cílový uzel, což by jej v seznamu upřednostnilo před ostatními stejně ohodnocenými uzly. Po seřazení by se nejlépe ohodnocený uzel vždy nacházel v poli na prvním místě, tedy

4 Duplicitní uzly jsou takové, které mají shodný identifikátor zastávky, datum a čas příjezdu do zastávky a počet uskutečněných přestupů.

na indexu 0. I když řazení polí je v PHP poměrně rychlé, ukázala se jako výhodnější první varianta, tedy projít celé pole a vyhledat index nejlépe ohodnoceného prvku.

Při vkládání prvku do seznamu již nastává problém. Je totiž nutné zjistit, zda se v poli již nenachází duplicitní uzel. Je tedy nutné pole projít a v případě nalezení duplicitního uzlu ho již znovu nevkládat. Časová náročnost této operace ovšem narůstá se vzdáleností duplicitního uzlu od začátku pole (je nutné projít více prvků). V případě, že se v seznamu žádný duplicitní uzel nenachází nebo se nachází na poslední pozici, je nutné projít celé pole, což může při vysokém počtu uzlu v seznamu OPEN operaci vkládání velice zpomalit.

4.3.2.2 Varianta 2: Implementace pomocí dvojrozměrného pole

Cílem této varianty bylo omezit počet prvků pole, které je nutné projít při vyhledávání nejlépe ohodnoceného uzlu. Toho lze dosáhnout rozdělením pole představujícího seznam OPEN do několika menších polí. Základní strukturu tedy tvoří asociativní pole, kde klíčem je vždy ohodnocení (číselné nebo řetězcové, v závislosti na zvoleném kritériu) a hodnotou je klasicky indexované pole obsahující všechny uzly, jejichž ohodnocení odpovídá klíči. Počet prvků asociativního pole tedy bude odpovídat aktuálnímu počtu všech unikátních ohodnocení uzlů, které se v daném okamžiku nacházejí v seznamu OPEN. Počet prvků pole zpřístupněného pod daným klíčem bude dán počtem uzlů s odpovídajícím ohodnocením, jež se právě nacházejí v seznamu OPEN.

Pro vyjmutí nejlépe ohodnoceného uzlu ze seznamu OPEN je tedy potřeba provést následující kroky:

1. Vyhledat v asociativním poli klíč s nejmenší hodnotou. Tento klíč zpřístupní pole uzlů s nejlepším ohodnocením.
2. 2. V poli prvků s nejlepším ohodnocením vyhledat prvky, které mají hodnotu *end_station* nastavenu na *true* (resp. 1) a tím pádem mají být při vyjmutí ze seznamu OPEN upřednostněny před ostatními uzly se stejným ohodnocením (viz. kapitola 4.3.1). Pokud se tam minimálně jeden takový uzel nachází, je vrácen jako uzel určený k expanzi. V opačném případě je vrácen libovolný uzel z dané množiny uzlů s nejlepším ohodnocením.

Celková doba potřebná k provedení těchto dvou kroků je výrazně nižší než doba potřebná k průchodu celého seznamu OPEN tak, jako tomu bylo u předchozí varianty.

*Odůvodnění: M je počet unikátních ohodnocení prvků nacházejících se aktuálně v seznamu OPEN (tedy počet prvků asociativního pole, které bude nutné projít v bodě 1), N je průměrný počet prvků v seznamu OPEN se stejným ohodnocením (tj. průměrný počet prvků, jež je nutné projít v bodě 2). Celkový počet prvků, které bude nutné projít pro nalezení nejlépe ohodnoceného uzlu je tedy dán součtem M+N, zatímco v předchozí variantě byl tento počet dán součinem M*N (bylo nutné projít všechny prvky pole představujícího celý seznam OPEN).*

Stále ovšem přetrvává problém s vkládáním nových uzlů do seznamu OPEN, kdy je nutné odhalit duplicitní uzly. Tato kontrola může totiž v některých případech znamenat stejně jako u předchozí varianty průchod celým seznamem OPEN, tedy celým dvojrozměrným polem, což je velice časově náročné.

4.3.2.3 Varianta 3: Implementace pomocí asociativního pole

V předchozí variantě se sice podařilo snížit časovou náročnost vyhledání a vyjmutí nejlépe ohodnoceného uzlu ze seznamu OPEN, ovšem čas potřebný k vložení nového uzlu zůstává stále příliš vysoký. Příčinou této vysoké náročnosti je především průchod celým seznamem OPEN při každém vkládání nového uzlu kvůli odhalení duplicit. Je tedy žádoucí tuto operaci nějakým způsobem urychlit.

Nabízí se zde možnost využití asociativního pole. Všechny uzly nacházející se aktuálně v seznamu OPEN budou uloženy v jednorozměrném poli, podobně jako tomu bylo u první varianty. Toto pole ovšem nebude klasicky indexované, nýbrž asociativní. Klíč bude vždy tvořit řetězec vzniklý spojením tří hodnot: identifikátoru zastávky, počtu dosud uskutečněných přestupů a datem a časem příjezdu spoje do dané zastávky, tedy hodnot, jejichž souhrn jednoznačně identifikuje daný uzel.

Při vkládání nového uzlu poté není nutné projít celé pole a kontrolovat, zda některý uzel nemá tyto tři hodnoty shodné s nově vkládaným uzlem. Stačí pouze spojením jmenovaných hodnot nově vkládaného uzlu vytvořit příslušný klíč a ověřit, zda se v asociativním poli již nachází prvek se stejným klíčem. V případě, že je takový prvek nalezen, se na základě jeho ohodnocení a ohodnocení nově vkládaného prvku rozhodne, který z nich bude v poli pod příslušným klíčem ponechán (vždy zůstane ten s lepším ohodnocením).

Kontrola existence klíče v asociativním poli je výrazně rychlejší než průchod celým polem a vyhledávání duplicitních prvků. Proto zde dochází ke značnému urychlení vkládání nových uzlů do seznamu OPEN.

Pro vyhledání a vyjmutí nejlépe ohodnoceného uzlu je ovšem opět nutné projít celé pole tak, jako tomu bylo u první varianty. Časová náročnost této operace je tedy prakticky stejná jako u první varianty a tím pádem vyšší než u varianty druhé. Ovšem vzhledem k tomu, že operace vkládání uzlu do seznamu probíhá mnohem častěji než operace vyjmutí uzlu, dojde v konečném důsledku ke zkrácení doby potřebné k vyhledání výsledku.

4.3.2.4 Varianta 4: Implementace s využitím asociativního pole a pomocného pole

Varianta 2 nabízí velice rychlé vyhledání a vyjmutí nejlépe ohodnoceného uzlu ze seznamu OPEN, ale pomalé vkládání nových uzlů, zatímco u varianty 3 je tomu naopak. Cílem této varianty je tedy

spojit výhody předchozích dvou variant, i když to pravděpodobně povede k mírnému zvýšení nároků na paměťový prostor.

Základní struktura reprezentující seznam OPEN tedy zůstává stejná jako u varianty 3, tedy asociativní pole (dále označeno jako *„pole uzlů“*), kde klíčem k danému uzlu je vždy řetězec tvořený konkatenací tří hodnot, které společně jednoznačně určují daný uzel (viz. předchozí varianta). Je tím zajištěno rychlé odhalování duplicitních uzlů pouze otestováním existence daného klíče v poli, což zaručuje rychlé vkládání nových uzlů.

Zbývá tedy snížit časovou náročnost vyhledání a vyjmutí nejlépe ohodnoceného uzlu ze seznamu OPEN, k čemuž je možné využít poznatků z varianty 3. Přibude tedy další, pomocné, dvojrozměrné pole (dále označeno jako *„pole ohodnocení“*). Jeho základ tvoří asociativní pole, kde klíčem je vždy ohodnocení uzlu (číslo nebo řetězec, v závislosti na zvoleném kritériu) a hodnotou je klasicky indexované pole, které obsahuje řetězce – klíče, odkazující do *pole uzlů* na jednotlivé prvky s daným ohodnocením. Struktura *pole ohodnocení* je tedy velice podobná struktuře použité pro uložení seznamu OPEN ve variantě 2, pouze s tím rozdílem, že místo kompletních informací o daném uzlu nese každý prvek této struktury pouze hodnotu klíče, pod kterým lze příslušný uzel najít v *poli uzlů*. Tato struktura tedy najde využití při vyhledávání nejlépe ohodnoceného uzlu, kdy už není potřeba procházet celé pole uzlů, nýbrž stačí v *poli ohodnocení* vyhledat klíč nejlépe ohodnoceného uzlu (stejným způsobem jako ve variantě 3) a poté s pomocí tohoto klíče získat příslušný uzel z *pole uzlů*. Je ovšem nutné zajistit, aby při odstranění (vyjmutí) prvku z *pole uzlů* došlo také k odstranění příslušného klíče z *pole ohodnocení a* naopak při vkládání nového uzlu byl jeho klíč správně vložen do *pole ohodnocení*.

4.3.2.5 Srovnání variant 1-4

Následující tabulka ukazuje časovou náročnost jednotlivých operací u výše popsaných variant implementace seznamu OPEN. Uvedené údaje byly naměřeny po provedení 5.000 kroků vyhledávacího algoritmu (po expanzi 5.000 uzlů) při faktoru větvení 12,8. Hodnoty jsou v sekundách a představují celkový čas, který systém strávil prováděním daných operací. Operace vložení uzlu zahrnuje vložení nového uzlu do seznamu OPEN a kontrolu duplicitních uzlů, operace vyjmutí uzlu znamená vyhledání nejlépe ohodnoceného uzlu a jeho následné odstranění ze seznamu.

Operace	Varianta 1	Varianta 2	Varianta 3	Varianta 4
Vložení uzlu	360,36 s	382,35 s	3,38 s	3,4 s
Vyjmutí uzlu	65,62 s	2,28 s	63,11 s	2,42 s

Srovnání časové náročnosti jednotlivých variant implementace seznamu OPEN

Na tomto srovnání je vidět, že poslední, čtvrtá varianta je z hlediska časové náročnosti nejvýhodnější a proto je taky v systému použita.

4.4 Testovací data

Pro ověření správné funkčnosti systému je samozřejmě nutné mít k dispozici dostatečné množství testovacích dat. Zadávat tato data do databáze ručně by bylo velice časově náročné, proto jsem se rozhodl získávání dat a jejich vkládání do databáze zautomatizovat pomocí PHP a Bash skriptů.

4.4.1 Získání testovacích dat

Veškerá potřebná data jsou veřejně dostupná na portálu http://portal.jizdnirady.cz. Ten byl zřízen Ministerstvem dopravy ČR a společností CHAPS s r.o. a zpřístupňuje data uložená v Celostátním informačním systému o jízdních řádech (CIS JŘ). Jsou zde k dispozici aktuální jízdní řády všech autobusových a osobních vlakových linek v ČR. Bohužel, vlakové jízdní řády jsou zde k dispozici pouze ve formátu PDF, který je pro další zpracování poměrně nevhodný. Pro testování systému ovšem postačí autobusové jízdní řády, které jsou zde k dispozici kromě PDF také ve formátu XLS, který už lze dále zpracovávat pomocí skriptů.

Bylo tedy nutné nejprve stáhnout všechny dostupné XLS soubory s jízdními řády, z nich následně vydolovat potřebné informace a ty následně vložit do databáze tak, aby nedošlo ke ztrátě asociací mezi informacemi.

4.4.2 Nedokonalosti v testovacích datech

I přes veškerou snahu ovšem zůstaly v testovacích datech některé nesrovnalosti, jejichž oprava by vyžadovala „ruční“ zásahy do databáze a byla by velice časové náročná. Tyto nesrovnalosti ovšem nemají zásadní vliv na chod systému, proto je lze tolerovat.

Problém je především s duplicitními názvy měst a obcí. Některé názvy obcí se totiž v České Republice vyskytují vícekrát (např. *Kojetín*). Tyto „duplicitní“ obce ovšem nejsou ve získaných jízdních řádech nijak rozlišeny. Z příslušného XLS dokumentu tedy není patrné, zda se jedná o obec Kojetín na Přerovsku či na Havlíčkobrodsku. Z toho důvodu jsou tedy obě tato města uložena v databázi pod stejným ID a nelze je tedy rozlišit.

Toto působí problémy především v heuristických datech. Jak již bylo řečeno, heuristická data uchovávají vždy nejkratší možnou vzdálenost, nejkratší možný čas cesty a nejnižší možnou cenu cesty mezi jednotlivými městy. Tedy například heuristický záznam pro města Kojetín a Olomouc říká, že nejkratší možná vzdálenost mezi nimi je 29 km. Toto odpovídá vzdálenosti Olomouce a Kojetína na Přerovsku. Ovšem při vyhledávání spojení mezi Olomoucí a Kojetínem na Havlíčkobrodsku už bude tato heuristická informace zavádějící, protože skutečná vzdálenost mezi těmito dvěma městy je cca 140km. Toto tedy může v některých případech vést k nepřesnému výpočtu ohodnocení uzlů stavového prostoru při vyhledávání a tím pádem k prodloužení doby vyhledávání.

Při získávání testovacích dat se vyskytly ještě další, méně závažné problémy. U některých spojů se například nepodařilo zjistit počet kilometrů mezi jednotlivými zastávkami apod. Takovéto spoje nemohou být zahrnuty do vyhledávání, protože chybějící údaje by mohly vést k nepřesným informacím (např. o ujeté vzdálenosti) a tím pádem ke špatnému ohodnocení uzlů a chybným výsledkům vyhledávání.

4.4.3 Heuristika a průběžná aktualizace

Jak již bylo řečeno, v databázi jsou uchovávána heuristická data nesoucí informace o nejkratší dosažitelné vzdálenosti, nejkratší možné době cesty a nejnižší možné ceně cesty mezi každými dvěma městy. Výpočet těchto údajů pro všechny dvojice měst je ovšem poměrně časově náročný, a to především u měst, mezi kterými neexistuje přímé spojení a je tedy nutné provést výpočet v několika krocích.

Testovací data sice v současné době obsahují heuristické informace pro cca 30% všech dvojic měst. To je sice relativně málo, ale i přesto již došlo k viditelnému urychlení vyhledávání oproti metodě UCS, která neměla k dispozici žádné heuristické informace. Pokud během vyhledávání dojde k situaci, že není k dispozici příslušný heuristický záznam pro daná dvě města, je v tomto případě heuristika nulová. To způsobí, že i když daný uzel neleží na optimální cestě, může mít nižší ohodnocení než uzly ležící na optimální cestě, pro které je ale heuristická hodnota známá. Tím pádem bude tento uzel při výběru z fronty OPEN upřednostněn před uzlem ležícím na optimální cestě, což sice negativně ovlivní časovou náročnost vyhledávání, ale na nalezení správného výsledku to vliv mít nebude.

Kromě množství heuristických informací je ovšem důležitá i jejich přesnost. Jak jsem již uvedl v kapitole 3.3, čím přesnější budou heuristické informace, tím menší část stavového prostoru bude potřeba prohledávat a tím nižší bude časová náročnost celého vyhledávacího procesu. Heuristické údaje, které jsou v současné době k dispozici ovšem nemusí být zcela přesné. Při výpočtu totiž nebyla brána v potaz žádná omezení, minimální a maximální čas na přestup atd. Proto v reálné situaci nemusí být možné danou cestu s takovým ohodnocením realizovat například proto, že spoje, které byly pro výpočet heuristických údajů použity na sebe v daném čase nenavazují apod. Výsledná skutečná cena cesty (ať už je za cenu považována vzdálenost, čas nebo částka v Kč) je tedy ve většině případu vyšší než udává heuristika. To ovšem vyhovuje požadavku, že hodnota heuristiky má být spodním odhadem skutečné ceny, a tak lze prohlásit tato heuristická data za přípustná.

Problém ovšem nastává v případě, že mezi danými dvěma městy existuje přímé spojení. Pak je totiž za hodnotu heuristiky považována skutečná cena tohoto přímého spojení, přestože je možné (i když k tomu nedochází často), že tuto cestu lze realizovat nepřímo (s přestupem) ale s nižším ohodnocením. V takovém případě už tuto heuristickou informaci nelze prohlásit za spodní odhad

skutečné ceny. Jediným důsledkem této nepřesnosti ovšem může být pouze mírné prodloužení doby potřebné k vyhledání výsledné cesty, na správnost výsledků toto žádný vliv nemá.

I tak je ovšem žádoucí, aby heuristická data byla co nejpřesnější. Proto jsem do systému zakomponoval funkci průběžné aktualizace heuristických dat. Ta zajistí postupné zpřesňování heuristických informací v průběhu používání systému. Princip je poměrně jednoduchý:

1. Uživatel zadá požadavek na vyhledání spojení z města A do města B (případně z kterékoliv městské části nebo zastávky ve městě A do městské části či zastávky ve městě B).
2. Spojení je vyhledáno a je zjištěna celková vzdálenost, doba a cena této cesty.
3. Pokud pro dvojici měst A a B neexistuje žádný heuristický záznam, jsou do databáze vloženy tyto tři vyhledané údaje (i když se nemusí jednat o nejnižší dosažitelné hodnoty) a ty jsou od této chvíle považovány za heuristickou informaci pro daná dvě města
4. Pokud pro tuto dvojici měst heuristický záznam již existuje, ale některá ze tří existujících hodnot je vyšší než nově nalezená hodnota, je původní hodnota nahrazena tou novou, nižší.

Takto tedy dochází k postupnému vkládání chybějících a zpřesňování existujících heuristických informací.

5 Srovnání s jinými podobnými systémy

V současné době nejpoužívanějším systémem pro vyhledávání přepravních spojů v celostátním měřítku je jednoznačně systém IDOS. Proto se budu v této kapitole věnovat srovnání mnou vyvinutého systému právě se systémem IDOS.

Hlavní předností mého systému je možnost zvolit kritérium pro vyhledávání. Na výběr je celkem pět různých možností[5]:

- Co nejmenší počet přestupů
- Co nejdříve v cíli
- Co nejkratší doba cesty
- Co nejkratší ujetá vzdálenost
- Co nejnižší cena cesty

Takovéto možnosti systém IDOS ani jiné podobné systémy nenabízejí.

Další výhodou je možnost vyhledat alternativní přestupní body u výsledných nalezených spojení. Pokud se trasa daných dvou spojů kříží na více stanicích, má uživatel možnost zobrazit všechny tyto stanice s příslušnými časy příjezdu a odjezdu obou spojů (samozřejmě pouze v případě, že je možné v dané stanici realizovat přestup). Toto může být výhodné především v případech, kdy je cestující nucen čekat delší dobu na navazující přípoj a je pro něj tedy lepší na tento přípoj čekat raději na větším autobusovém nádraží s náležitým zázemím, než na odlehlé zastávce na okraji města.

Za zmínku rovněž stojí pokročilé možnosti při vyhledávání měst, částí měst a zastávek. Pokud uživatel např. nezná přesný název zastávky, může si otevřít pop-up okno z formulářem, kde zadá název (nebo část názvu) města, městské části a/nebo zastávky a jsou mu vyhledány všechny odpovídající údaje. Vybraný údaj poté může jediným kliknutím přenést do vyhledávacího formuláře (což zároveň způsobí uzavření pop-up okna).

Na druhou stranu ovšem systém IDOS ve většině případů realizuje vyhledávání rychleji než můj systém. To je ovšem dáno především použitými technologiemi (skriptovací jazyk PHP v kombinaci s databázovým systémem MySQL vs. platforma .NET, na které je založen IDOS) a rovněž technickým zázemím (výkon serverů).

5 bližší specifikaci těchto možností lze nalézt v kapitole 2.1.2

6 Návrhy na další rozšíření systému

Co se týká možných rozšíření systému, naskýtá se zde samozřejmě spousta možností.

V prvé řadě by bylo vhodné systém rozšířit o administrační rozhraní, které by umožňovalo správu údajů uložených v databázi, možnost přidávání, editace a mazání zastávek, linek, spojů, omezení a prakticky všech dat souvisejících s jízdními řády.

Výrazným přínosem by rovněž byla možnost automatického importu jízdních řádů od jednotlivých dopravců, např. ve formátu XML apod.

Dále se naskýtá možnost upravit uživatelské rozhraní po grafické stránce a přidat další funkce, jako například generování jízdních řádů pro jednotlivé linky v tabulkovém formátu (podobném „papírovým“ jízdním řádům).

7 Závěr

Cílem této bakalářské práce bylo vytvoření informačního systému pro vyhledávání přepravních spojů s přestupy na základě různých kritérií.

Je zde popsán celý proces vývoje tohoto systému počínaje stanovením požadavků a návrhem systému a databáze, přes volbu vhodné metody prohledávání stavového prostoru až po samotnou implementaci systému. Dále jsem zde popsal několik způsobů implementace a na základě jejich srovnání jsem vybral jednu z nich, která z hlediska výkonu vyhovovala nejlépe.

Dále jsem se stručně zmínil o použitých technologiích. Následně jsem provedl porovnání systému s dalšími existujícími systémy pro vyhledávání přepravních spojů, zejména se systémem IDOS, který je v současné době nejrozšířenější. V závěru práce jsem uvedl několik návrhů na případné další rozšíření systému.

Hlavním přínosem této práce je již zmíněná možnost volby kritéria pro vyhledávání (na výběr je pět různých kritérií), což žádný jiný existující systém v současné době nenabízí, a další zajímavosti jako například alternativní přestupní body, snadné vyhledávání měst, částí měst a zastávek (i v případě, že uživatel nezná přesný název), či celkové zpříjemnění práce se systémem využitím technologie AJAX.

Pro mě osobně byla tato práce přínosem z hlediska prohloubení znalostí z oblasti návrhu a implementace informačních systémů a návrhu struktury rozsáhlých databází. Největším přínosem pro mě ovšem bylo rozšíření znalostí z hlediska programování a optimalizace výkonu systémů s vysokými výpočetními nároky postavených na kombinaci skriptovacího jazyka PHP a databázového systému MySQL.

Literatura

[1] Welling, L., Thomson, L. *MySQL: Průvodce základy databázového systému*. 1. vyd. Brno: CP Books. 2005. ISBN 80-251-0671-3.

[2] Zbořil, F., Zbořil, F. *Základy umělé inteligence (IZU), studijní opora*. Verze 3. 2006

[3] *PHP: Manuál PHP – Manual* [online]. poslední úpravy 24.3.2007. [cit. 2008-05-01]. URL: <http://www.php.net/manual/cs>.

[4] *Mysql 5.0 Reference Manual* [online]. [cit. 2008-05-01]. URL: <http://dev.mysql.com/doc/refman/5.0/en/>.

Seznam příloh

Příloha 1. DVD se zdrojovými texty, testovacími daty a návodem na instalaci

Systém je možné vyzkoušet na adrese http://eliska.jarcec.net/cizek/.