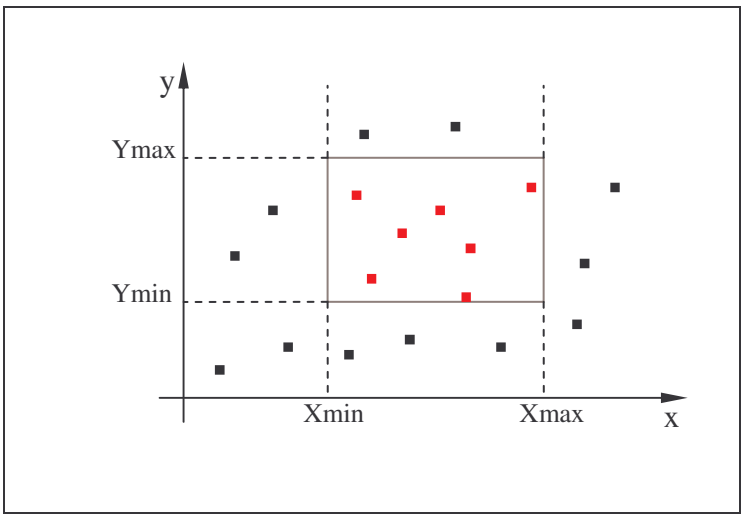
Zpráva k semestrální práci z předmětu
36PT – programovací techniky

Implementace algoritmu: Dvojrozměrný intervalový strom

jméno a příjmení: Michal Trs
studijní skupina: 12
ročník: 2
cvičení: pátek 7:30

1. Specifikace řešeného problému

Dvourozměrný intervalový strom je datová struktura vhodná pro reprezentaci vyhledávacího prostru s operací hledání na intervalovou shodu. Tento datový typ je vhodné použít pro velký počet neměnných prvků (bodů [x,y]), kde potřebujeme často vědět které body z prohledávaného prostoru padnou do obdélníku <Xmin,Xmax> x <Ymin,Ymax>. Viz *Obr **1.1***.



Obr 1.1 - řešený problém

2. Rešerše

Nejjednodušší algoritmus prohledávání na intervalovou shodu je Sekvenční vyhledávání (Popsáno viz použitá literatura kap. 4.2.1). Pokud je množina nad kterou vyhledáváme seřazena, stačí sekvenčně od začátku najít dolní mez a poté vypisovat prvky dokud je aktuální prvek menší než horní mez hledaného intervalu. Částečně lze hledání urychlit tím, že do určitých míst seznamu budeme přistupovat pomocí pole. Další metodou jak efektivněji najít minimum od kterého probíhá vyhledávání je binární půlení.

Další použitelnou strukturou pro vyhledávání na intervalovou shodu jsou D-BVS. Jsou to D-rozměrné binární vyhledávací stromy kde se v každém patře stromu cyklicky mění klíč podle kterého vyhledáváme. Například máme-li 2-BVS hledáme na první úrovni podle 1.klíče (souřadnice x), na druhé úrovni podle 2.klíče (souřadnice y), na 3. úrovni znovu podle 1.klíče, atd. Výhodou těchto stromů je, že prostor můžeme prohledávat také na úplnou a částečnou shodu.

3. Popis a vysvětlení algoritmů

3.1 Jednorozměrný intervalový strom

Nejprve vysvětlím konstrukci a prohledávání jednorozměrného stromu, protože tento strom je ve všech uzlech stromu dvourozměrného. Používá se pro vyhledávání intervalů v jednom rozměru.

3.1.1 Konstrukce stromu

Základem je obousměrně zřetězený seznam (dále jen *SEZNAM*) bodů se souřadnicemi x,y. Tento seznam musíme nejprve seřadit podle souřadnice podle které bude chtít vyhledávat. V našem případě podle souřadnice y. Nad takto seřazeným *Seznam*em stavíme odspodu binární vyhledávací strom tak, že prvky *SEZNAM*u jsou listy stromu.

Strom má tu vlastnost, že levý synové každého uzlu jsou menší nebo rovny uzlu a pravý synové mají větší hodnotu klíče než uzel. Kdyby byl seznam nesetříděn, toto by neplatilo a strom by nebyl použitelný pro vyhledávání. Tento strom je vidět na *Obr 5.3*.

3.1.2 Vyhledávání ve stromu

Nechť je hledaným intervalem interval <Ymin,Ymax>. Hledáme hodnotu p ≥ Ymin. Postupujeme od kořene, pokud je Ymin ≤ aktuální uzel pak jdi doleva, pokud Ymin> aktuální uzel pak jdi doprava. Takto postupuj dokud není uzel list (tzn. našel jsem prvek p). Pokud je p > Ymax je hledaná interval prázdný, jinak počínaje tímto prvkem p začni sekvenčně procházet *SEZNAM* směrem k větším prvků, dokud jsou prvky menší než Ymax. Tyto prvky jsou ukládány do výstupního seznamu.

3.2 Dvojrozměrný intervalový strom

Používá se pro prohledávání 2D prostoru na intervalovou shodu podle dvou souřadnic. Situace je znázorněna na *Obr **1.1 - řešený problém***. Opět popíšu konstrukci a prohledávání stromu jako v předchozím případě.

3.2.1 Konstrukce stromu

Základem je *SEZNAM*, tentokrát seřazený podle souřadnice x. Listy stromu nejsou prvky *SEZNAM*u (jako u 1D stomu), ale mají pouze hodnotu klíče a odkaz na jednorozměrný podstrom vytvořený ze seznamu. Tento seznam obsahuje pouze prvky kteří jsou synové tohoto uzlu (listu). Uzly vypadají stejně jako u 1D stromu, ale mají navíc ukazatel na 1D podstrom popsaný výše. Nad listy vygenerujeme opět vyvážený binární vyhledávací strom a každému vnitřnímu uzlu přiřadíme podstrom generovaný podle souřadnice y. Vlastní strom můžeme stavět odspodu od listů, nebo od kořene. Já jsem použil stavění od zdola které je podrobně popsáno v kapitole 4.3 u vysvětlování metody Create t2Stromu.

3.2.2 Vyhledávání ve stromu

Najdeme horní a dolní mez intervalu <Xmin,Xmax>. Označíme si uzel u ve kterém se dělí cesta k minimu a maximu. Je dobře patrné, že levý a pravý syn obsahují dohromady všechny prvky padnoucí do intervalu plus nějaké navíc, proto začneme minimu vyhledávat v levém synovy. Postupujeme takto:

- Pokud je hodnota Xmin ≤ hodnota uzlu jdeme doleva a tudíž musíme zpracovat (prohledáme výše popsaným způsobem 1D strom na shodu podle souřadnice y) podstrom pravého syna tohoto uzlu, protože všechny body jsou zaručeně mezi Xmin a Xmax a tudíž patří do hledaného intervalu.
- Pokud je Xmin > hodnota uzlu jdeme doprava a podstrom nezpracováváme.
- tímto způsobem pokračujeme dokud nenarazíme na list (tj. jsme na konci stromu).

Obdobně postupujeme při hledání maxima. Začneme u pravého syna. Pokud jdeme doprava, zpracováváme levé podstromy tohoto uzlu. Pokud jdeme doleva, podstrom nezpracováváme. Končíme opět narazíme-li na list.

4. Popis implementace a datových struktur

V této kapitole popíšu moji implementaci dvojrozměrného intervalového stromu. Programovací jazyk jsem zvolil Object Pascal – Delphi 6.

4.1 Pomocné datové struktury

Jsou to struktury používané jedno a dvojrozměrným intervalovým vyhledávacím stromem. Postupně zde uvedu deklarace všech datových typů a vysvětlím co jednotlivé metody dělají a u některých jak fungují.

tBod *(uSeznam)*

reprezentuje bod v rovině o souřadnicích x,y

```
tBod = record
   x,y: integer;
end;
```

tCell *(uSeznam)*

Buňka spojového seznamu.

```
ptrCell = ^tCell;
tCell = record
  prev, next: ptrCell;
  point: tBod;
end;
```

tKlic *(uSeznam)*

výčtový typ - používán v metodách při specifikaci o jakou souřadnici se jedná. Napři v proceduře Serad.

```
tKlic = (x,y);
```

tSeznam *(uSeznam)*

Vlastní spojový seznam.

```
tSeznam = class
 public
  constructor Create;
  destructor Destroy; {zrusi vsechna data v seznamu}
  procedure Prazdny;  {vyhodi vsechny bunky, ale nerusi data!!!}
  procedure Pridej(bod: tBod); {pridani noveho prvku na konec}
  procedure SetAktualni(prvek: ptrCell);
  procedure ZaradPrvek(Prvek: ptrCell);
  function Prvni: ptrCell;
  function Dalsi: ptrCell;
  function PrvniHodn: tBod;
  function DalsiHodn: tBod;
  procedure Serad(klic: tKlic);
  function QKonec: Boolean;
  function QPrazdny: Boolean;
  function PocPrvku: integer;
  procedure PripojSeznam(Sez: tSeznam); {pripoji Sez nakonec seznamu}
  function SpojSeznamy(Sez1,Sez2: tSeznam): tSeznam;
  function GetAktualni: tBod;
  {preskoci prvky stejne hodnoty jak Hod podle x nebo y}
  function PreskocStejne(Hod: integer; klic: tKlic): tSeznam;
 private
  pPrvku: integer;
  first, last, akt: ptrCell;
end;
```

Vysvětlení základních metod objektu:

Create	Vytvoří prázdný spojový seznam.
Destroy	Zruší instanci objektu všechny jeho data.
Pridej	Přidá nový Bod na konec seznamu.
SetAktualni	Nastaví prvek jako aktuální pozici v seznamu.
ZaradPrvek	Obdoba metody Pridej, ale přidává již vytvořený Prvek seznamu.
Prvni / **PrvniHodn**	Vrátí ukazatel na první buňku seznamu / přímo hodnotu bodu v buňce. Přitom nastaví aktuální pozici na začátek.
Dalsi / **DalsiHodn**	Posune se v seznamu o jednu pozici doprava a vrátí ukazatel na buňku seznamu / přímo hodnotu bodu v buňce.
Serad	Seřadí seznam bodle souřadnice odpovídající klíči. Řazení je realizováno algoritmem QUICKSORT. Probíhá takto: Nejprve se v paměti alokuje dynamické pole ukazatelů na buňky seznamu. Tyto ukazatele se setřídí. Dále je použita metoda Prázdný která vyprázdní seznam, ale zachová data v paměti. Do takovéhoto seznamu jsou přidány metodou ZaradPrvek již seřazené prvky z pole.
QKonec	true pokud je aktuální prvek poslední.
QPrazdny	true pokud počet prvků je roven nule.
PocPrvku	Vrací počet prvků v seznamu.
PripojSeznam	Připojí seznam Sez na konec seznamu a tím pádem Sez zruší.
SpojSeznamy	Vrací nový objekt seznam. Symbolicky se dá tato operace zapsat takto: result:=Sez1+Sez2; Seznamy Sez1 a Sez2 zůstanou nezměněné.
GetAktualni	Vrací hodnotu aktuálního bodu
PreskocStejne	Přeskočí buňky seznamu které podle zvoleného klíče mají stejnou hodnotu souřadnice jako právě aktuální. Zastaví se na poslední buče se stejnou hodnotou. Přeskočené buňky jsou ukládány do nového seznamu a ten je funkcí vrácen.

funkce porovnej *(uSeznam)*

Porovnají proměnné A a B a vrací 1 pokud A > B, 0 pokud A = B a -1 pokud A<B

```
function Porovnej(A,B: tBod; klic: tKlic): integer; overload;{body s
prioritou klice}
function Porovnej(A,B: integer): integer; overload;
```

tFronta *(iStrom)****, t2Fronta*** *(i2Strom)*

Fronta do které je možno ukládat Uzly strumu. Jsou to 2 rozdílné objekty které se liší pouze daty s kterými mohou pracovat. tFronta pracuje s objekty tUzel (příp. zděděným tList) a t2Fronta s objekty t2Uzel (příp. t2List). Vlastní fronta je implementována pomocí jednosměrně zřetězeného spojového seznamu. Uvedu zde pouze deklaraci objektu tFronta, protože objekt t2Fronta obsahuje zcela stejné metody.

```
tFronta = class
 public
  constructor Create;
  destructor Destroy;
  procedure Pridej(Uzel: tUzel; dalsiHodn:integer);
  procedure Odeber(var Uzel: tUzel; var dalsiHodn:integer);
  function QPrazdna: boolean;
private
  celo, tyl: tPrvekFronty;
  pprvku: integer;
end;
```

Samotnou frontu tvoří buňky – objekty tPrvekFronty. Nebudu zde uvádět jejich deklarace, protože je používán pouze samotná fronta.

4.2 Jednorozměrný intervalový strom

Vše dále popisované v této podkapitole je implementováno v unitě iStrom.pas. Využívá unitu uSeznam.

tUzel

jednotlivé uzly z kterých je stavěn strom

```
tUzel = class;
tUzel = class
 public
  constructor Create(levy: tUzel; pravy: tUzel; hodn: integer);
  function GetHodnota: integer;
  function GetLevy: tUzel;
  function GetPravy: tUzel;
 private
  l,p: tUzel;
  hod: integer;
end;
```

Vysvětlení jednotlivých metod:

Create Vytvoří nový uzel, její parametry jsou levý a pravý syn a hodnota uzlu.
GetHodnota Vrací hodnotu uzlu.
GetLevy/**GetPtavy** Vrátí levého / pravého syna.

tList

Je odvozen od tUzel. Formálně je s ním shodný, ale jeho synové jsou ukazatelé na tCell, tj. jsou to ukazatelé do spojového seznamu.

```
tList = class (tUzel)
 private
  l,p: ptrCell;
 public
  constructor Create(levy: ptrCell; pravy: ptrCell; hodn: integer);
  function GetLevy: ptrCell;
  function GetPravy: ptrCell;
end;
```

Create Vytvoří nový list, který bude ukazovat do seznamu a bude mít hodnotu hodn

tStrom

Vlastní jednorozměrný vyhledávací strom. Pracuje zcela nezávisle na dvojrozměrném stromu.

```
tStrom = class
   public
    constructor Create(Seznam: tSeznam);
    destructor Destroy;
    Function NajdiInterval(Ymin,Ymax: integer): tSeznam;
    Function NajdiMin(Ymin: integer): ptrCell;
    function GetSeznam: tSeznam;
    function GetKoren: tUzel;
   private
```

```
    koren: tUzel;
    S: tSeznam;
  end;
```

Vysvětlení implementace jednotlivých metod:

Create Vytvoří nový jednorozměrný strom nad seznamem. Stavba stromu probíhá odspodu. Postupuje v těchto krocích:

- seřadí seznam podle klíče vzestupně.
- načte první prvek seznamu do proměnné lCell
- přeskočí buňky se stejnou hodnotou klíče
- načte další prvek do proměné pCell
- z lCell a pCell vytvoří list a přidá jej do fronty1
- takto pokračujeme dokud nejsme na konci seznamu
- pokud jsme na konci seznamu a v lCell máme načtenou buňku musíme list také vytvořit, ale s tím rozdílem, že pCell bude nil.
- Dokud není fronta1 prázdná, vyber vždy 2 prvky a vytvoř z nich nový uzel a přidej ho do fronty2. Pokud ve frontě1 zbyl jeden prvek, nestavíme nad ním další uzel, ale rovnou jej přesuneme do fronty 2.
- fronta1:=fronta2; a vyprázdni frontu 2;
- Opakuj předchozí 2 bodu dokud fronta neobsahuje pouze jeden prvek, pak skonči a tento prvek je kořen stromu.

Destroy Zruší strom se všemi daty

NajdiMin Pomocí stromu najde prvek p, který je větší nebo roven Ymin. Hledání je popsáno v kapitole 3.1.2.

NajdiInterval Najde interval <Ymin,Ymax>. Výsledek který funkce vrací je SEZNAM. Nejprve volá funkci NajdiMin. Pokud vrácená hodnota této funkce je větší než Ymin vrátí tato funkce nil – tzn. hledaný interval je prázdný. Jinak počínaje tímto prvkem začne procházet prvky ze SEZNAMu a začne je ukládat do nového seznamu dokud je aktuální hodnota < Ymax. Tento nový seznam je výstup funkce.

GetSeznam Vrátí seznam nad kterým byl strom postaven. Využito při konstrukci dvojrozměrného stromu.

GetKoren Vrací kořen stromu.

4.3 Dvojrozměrný intervalový strom

Vše dále popsané je implementováno v unitě i2Strom.pas. Tato unita využívá uSeznam a iStrom. Opět uvedu deklarace objektů Uzlu a Stromu a slovně vysvětlím jak pracují jednotlivé metody.

t2Uzel

„Základní kameny“ z kterých se skládá strom. Jsou podobné Uzlům jednorozměrného stromu, ale mají navíc položku podstrom, ve které je uložen strom generovaný podle souřadnice y.

```
t2Uzel = class;
t2Uzel = class
 public
  constructor Create(levy: t2Uzel; pravy: t2Uzel; hodn: integer;
    podstrom: tStrom);
  destructor Destroy;  virtual;
  function GetHodnota: integer;
  function GetLevy: t2Uzel;
  function GetPravy: t2Uzel;
  function GetPodStrom: tStrom;
```

```
 private
  l,p: t2Uzel;
  hod: integer;
  Strom: tStrom;
end;
```

Vysvětlení metod:

Create	Vytvoří nový uzel. Proměnné levy a pravy jsou levý a pravý syn. PodStrom je jednorozměrný strom.
Destroy	Zruší Uzel a všechna data která obsahuje (i podstrom)
GetHodnota	Vrátí hodnotu uzlu.
GetLevy/GetPravy	Vrátí levého / pravého syna.
GetPodStrom	Vrátí objekt jednorozměrný strom.

t2List

Odvozen z t2Uzel. Vytvořil jsem si jej proto, že pomocí něj snadno zjistím, že jsem na konci stromu. Oproti t2Uzel nemá žádné metody navíc, pouze má jednoduší `constructor`, který pravého a levého syna nastaví na nil.

t2Strom

Rozhraní 2 rozměrného vyhledávacího intervalového stromu. Create postaví dvojrozměrný strom nad seznamem. Funkce najdi interval, hledá body padnoucí do zadaného intervalu.

```
t2Strom = class
 public
  constructor Create(seznam: tSeznam);
  destructor Destroy;
  Function NajdiRozcesti(Xmin,Xmax: integer): t2Uzel;
  Function NajdiInterval(Xmin,Xmax,Ymin,Ymax: integer): tSeznam;
 private
  koren: t2Uzel;
  Sez: tSeznam;
end;
```

Vysvětlení metod:

Create Pracuje v těchto krocích:

- Seřadí seznam podle souřadnice x
- Načte první hodnotu ze seznamu a z ní vytvoří list. Přeskočí body se stejnou hodnotou souřadnice x. Tyto hodnoty jsou uložené v seznamu. K tomu seznamu je přidána první hodnota a je poslán do konstruktoru jednorozměrného strumu. Tento strom je přiřazen tomuto listu. Vytvořený list uložíme do fronty1
- To samé proveď pro všechny další hodnoty x. Pokračuj tak dlouho, dokud nejsme na konci seznamu.
- Postupně vybírej listy po dvou z fronty1 a vytvářej nad nimi uzly. Spoj seznamy z kterých byl vytvořen podstrom prvního a druhého načteného listu a pošly do konstruktoru jednorozměrného stromu. Tento strom bude podstromem tohoto uzlu. Pokud jsme na konci a je načten pouze první list z dvojice, tak druhému přiřadíme nil a vytvoříme uzel. Uzly přidáváme do fronty2
- fronta1:=fronta2,
- pokud fronta1 obsahuje pouze 1 prvek , je tento prvek kořen stromu.

- jinak vybírej uzly z fronty1 po dvou a vytvářej nad nimi uzly výše popsaným způsobem. Pokud zbude na konci fronty1 uzel přesuň jej do fronty2.

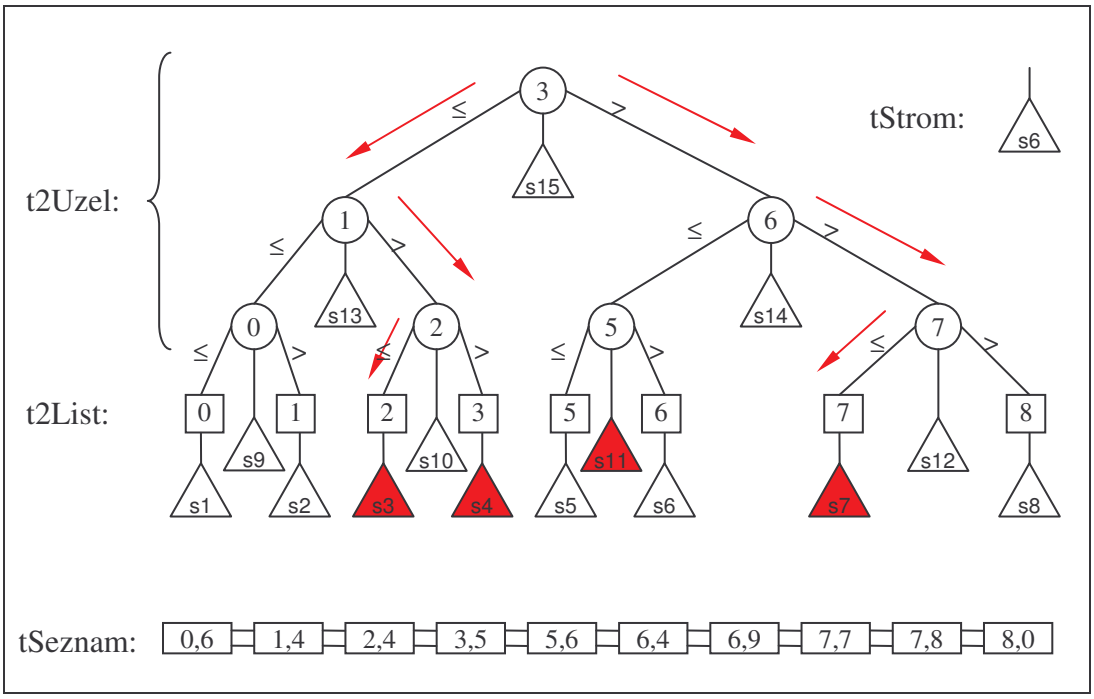
Destroy Zruší strom včetně všech dat.
NajdiRozcestí Najde uzel ve kterém se dělí cesta k maximu a minimu.
NajdiInterval popsáno v kapitole 3.2.2.

5. Ověření algoritmu

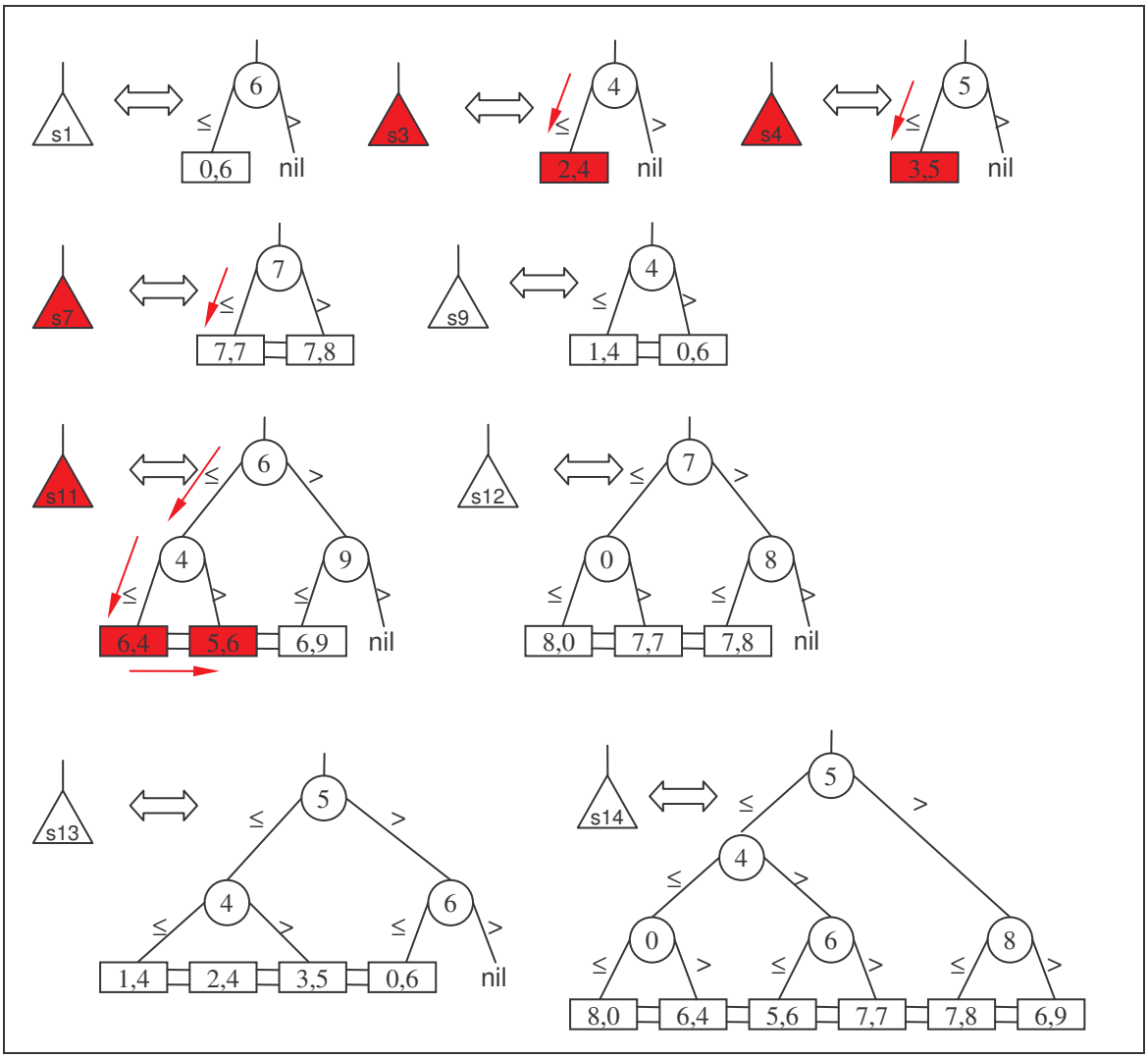
V následujících podkapitolách si na dvou příkladech ukážeme jak probíhá vyhledávání ve dvojrozměrném intervalovém stromě.

5.1 První příklad

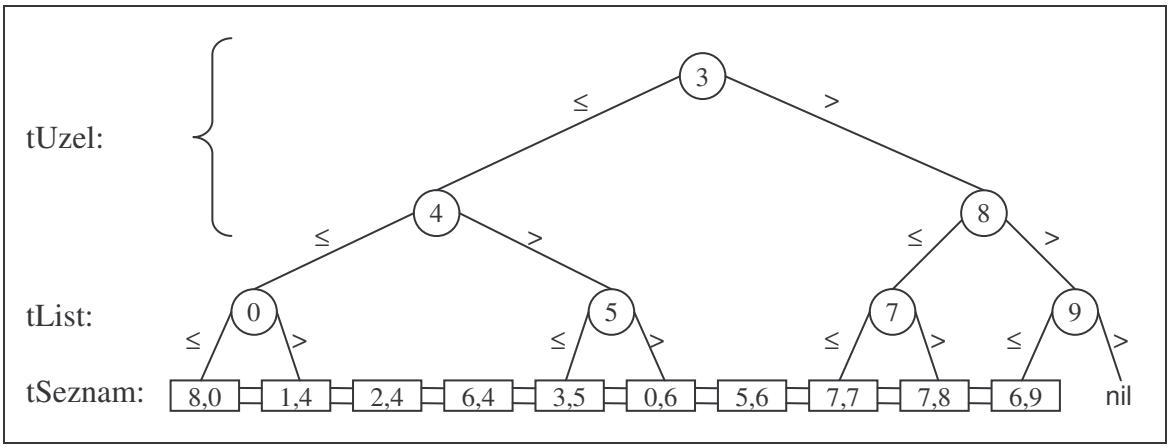
Mějme tyto body [0,6], [1,4], [2,4], [3,5], [5,6], [6,4], [6,9], [7,7], [7,8], [8,0]. Z nich je vytvořen dvojrozměrný intervalový strom (*Obr 5.1*, *Obr 5.2*, *Obr 5.3*). Hledejme například interval **<2,7>** x **<3,6>**. Nejprve hledáme uzel, kde se nám dělí cesta k horní a dolní mezi intervalu x. Tento bod je hned kořen stromu. Proto od tohoto bodu pokračuje jinou cestou při hledání minima a maxima. Nejprve si ukážeme hledání minima. Dobře je to vidět v *Obr 5.1*, cesta je vyznačena červenými šipkami. Dolní mez intervalu je 2, proto v uzlu 1 jdeme doprava a levý podstrom do intervalu nepatří. Uzel 2 je roven dolní mezi proto pokračujeme doleva a přitom musíme zpracovat pravý podstrom s4, protože podle souřadnice x padne do intervalu. V podstromu s4 hledáme na shodu podle y. Uzel = 5, Ymin = 3, jdeme doleva. Jsme v seznamu, nalezli jsme prvek **[3,5]**, 5 je menší než Ymax = 6 a proto tento prvek zařadíme do výsledného seznamu. Další prvek v seznamu není a proto pokračujeme dále v prohledávání hlavního stromu podle x. Uzel s hodnotou 2 je list a to znamená že jsme s hledáním minima skončili. Ještě musíme prohledat podstrom s3 na shodu podle y. Hledání probíhá naprosto stejně jako u podstromu s4. Nalezený prvek **[2,4]** do intervalu patří a proto jej přidáme do výsledného seznamu.
Nyní pokračujeme hledáním maxima. Uzel 6 je menší než Xmax = 7 a proto pokračujeme doprava, ale musíme zpracovat levý podstrom s11. Hledání v tomto podstromu je patrné z *Obr 5.2*. Hodnota kořene je 6, což je více než Ymin = 3 a proto pokračujeme doleva. Uzel 4 je větší než Ymin a proto pokračujeme opět doleva. Nalezli jsme prvek **[6,4]**. 4 je menší než Ymax = 6 a proto tento prvek do hledaného intervalu patří. Přidáme jej tedy do výstupního seznamu. Sekvenčně začneme procházet seznam dokud nebude hodnota větší než Ymax, nebo bude konec seznamu. Další je prvek **[5,6]**, 6 je menší nebo rovno Ymax = 6 a proto jej přidáme do seznamu. Další je prvek [6,9], 9 není menší než Ymax a proto končíme s procházením podstromu a vracíme se k hledání maxima podle souřadnice x. Uzel 7 je menší nebo roven Xmax = 7 a proto jdeme doleva. Uzel 7 je zároveň list a proto zde hledání maxima končí. Ještě musíme prohledat podstrom s7. Uzel 7 je větší než Ymin = 6 a proto pokračujeme doleva. Nalezli jsme prvek [7,7], 7 je větší než Ymax a proto tento prvek do intervalu nepatří.



Obr 5.1- t2Strom - dvojrozměrný intervalový strom (strom podle souřadnice x)



Obr 5.2 - ukázka podstromů dvojrozměrného stromu z Obr 5.1

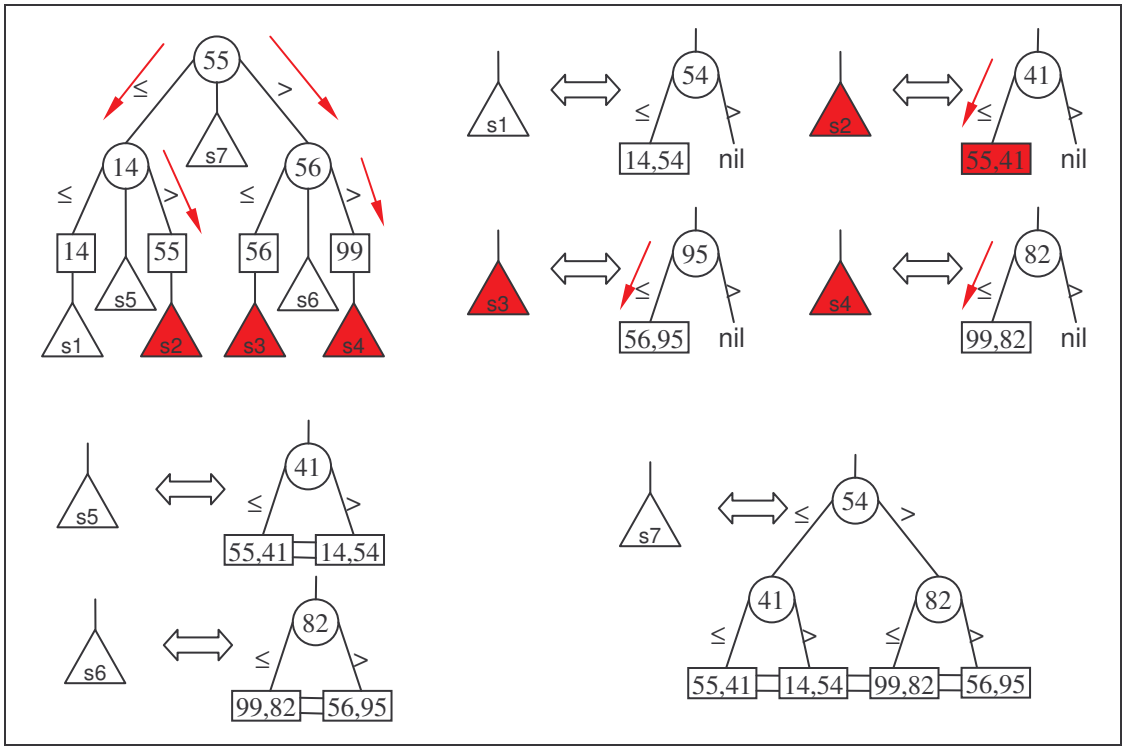


Obr 5.3 - Jednorozměrný strom podle souřadnice y (současně podstrom s15)

Obr 5.4 - výsledek testovací utility

5.2 Druhý příklad

Druhý příklad jsem zvolil jednoduší. Náhodně jsem zvolil 4 body a dal vyhledat ¼ intervalu. Hledání stejně jako je popsáno v prvním příkladu. Na *Obr 5.5* je situace rozkreslená do grafu. Je patrné, že vyjde stejný výsledek jako v testovací utilitě na *Obr 5.6*



Obr 5.5 - Kompletní dvojrozměrný intervalový strom

Obr 5.6 - výsledek testovací utility

6. Výpočet a ověření složitosti algoritmu

6.1 Jednorozměrný intervalový strom

6.1.1 Operační složitost vyhledávání

Při hledání intervalu musíme nejprve najít prvek p ≥ Ymin. Znamená to, že musíme jednou projít stromem od kořene k listu. Počet provedených porovnání je roven výšce stromu. U binárního stromu je rovna $\log_2 n$. Po nalezení prvku p začneme sekvenčně procházet seznam. Při každém průchodu musíme porovnat jestli je aktuální prvek menší než Ymax. Tento počet odpovídá počtu prvků odpovídající hledanému intervalu. Označíme ho m. Ve většině případů je $m \geq \log_2 n$, proto jej nemůžeme zanedbat. Celková složitost vyhledávání je tedy $O(m+\log_2 n)$.

6.1.2 Paměťová složitost

Strom se skládá ze spojového seznamu o n prvcích. Nad tímto seznamem je postaven strom o n - 1 uzlech. Celková složitost je S(n).

6.2 Dvojrozměrný intervalový strom

6.2.1 Operační složitost vyhledávání

Je patrné, že nalezení minima a maxima podle souřadnice x je úměrné $\log_2 n$. Dále musíme pro každou souřadnici x ležící v intervalu prohledat podstrom na shodu podle souřadnice y se složitostí $O(m+\log_2 n)$. To znamená, že složitosti musíme vynásobit. Dostaneme tedy $O((\log_2 n)^2+m)$.

6.2.2 Paměťová složitost

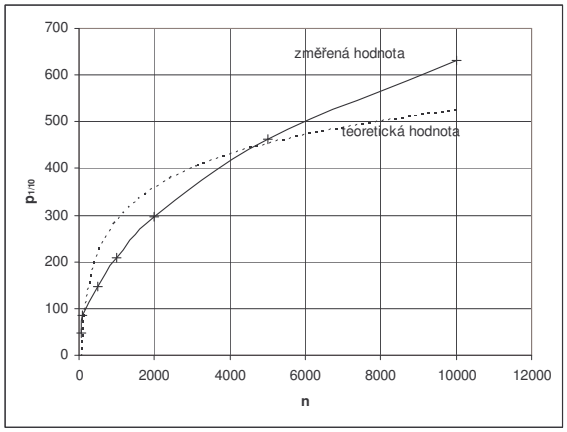
Strom se skládá z n-1 uzlů. Každý uzel obsahuje podstrom vystavěný na seznamu. Součet prvků seznamů v jedné úrovni je n a součet uzlů podstromů v každé úrovni je n-1. Abychom dostaly celkový počet vynásobíme toto číslo počtem úrovní stromu, kterých je $\log_2 n$. Vyjde nám, že celková paměťová složitost je $S(n+n.\log_2 n)$.

6.3 Měření počtu operací

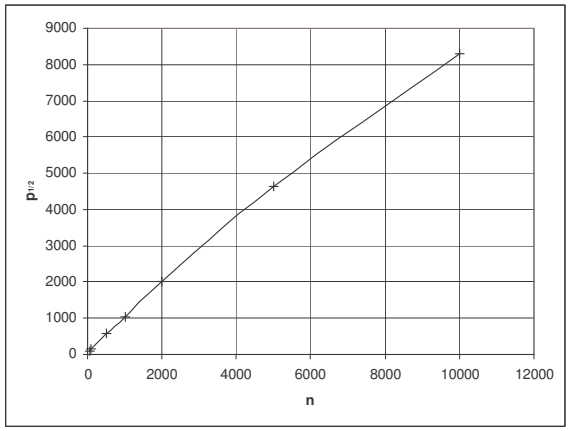
Pro ověření operační složitosti jsem do kódu umístil počítadla, která počítala počet porovnání. Vždy jsem vygeneroval n náhodných bodů z intervalu <0;99>x<0;99>. Poté jsem vyhledával body z intervalu <15;85>x<15;85>. To přibližně odpovídá ½ bodů ležící v tomto intervalu ze všech vygenerovaných bodů. To samé jsem provedla pro případ kdy hledaný interval splňuje přibližně 1/10 bodů. Hledal jsem interval <34;66>x<34;66>. Zde popsané měření je vyneseno v tabulce. Kde n – počet všech bodů, $p_{1/2}$ – hledaný interval je polovina všech bodů, $p_{1/10}$ – hledaný interval je 1/10 všech bodů.

n	50	100	500	1000	2000	5000	10000
$P_{1/10}$	48	85	146	208	296	462	632
$P_{1/2}$	88	143	581	1029	2002	4634	8312

Závislosti z tabulky jsou vyneseny do Grafu.



Graf 6.1 - 1/10 budů leží v hledaném inervalu



Graf 6.2 - 1/2 bodů leží v hledaném intervalu

Z grafů je velice dobře patrné, že pro menší počet bodů ležící v hledaném intervalu má algoritmus skoutečně logaritmickou operační složitost. Z Grafu 6.2 je patrné, že při velkém počtu prvků ležící v hledaném intervalu je složitost lineární. Je to způsobeno sekvenčním vyhledáváním po nalezení hodnoty Ymin v jednorozměrných podstromech. Znamená to, že $m << \log_2 n$.

7. Použitá literatura

Hudec, B.: Programovací techniky. Praha, ČVUT 2001.