Geometria obliczeniowa
Wykład 5

Geometryczne struktury danych

1. Drzewa odcinków

   Badanie przecięć odcinków w modelu PRAM

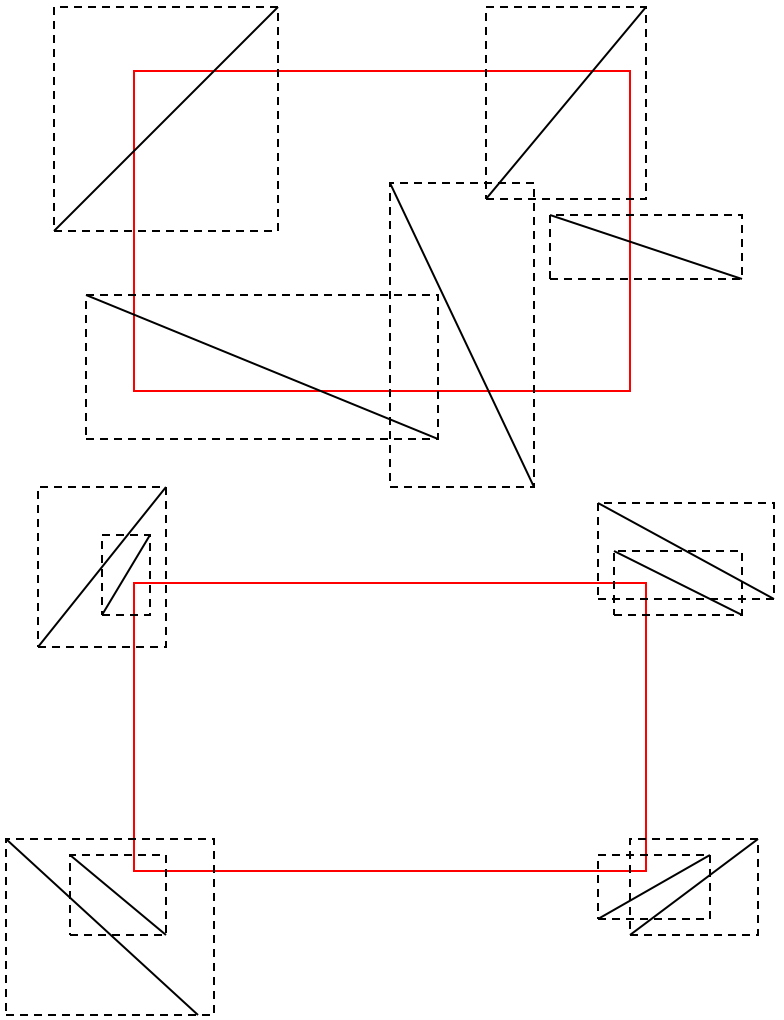
2. Drzewa czwórkowe

3. Drzewa BSP

Dotychczas zajmowaliśmy się problemem przecinania prostokąta odcinkami równoległymi do jego boków. Jednak znacznie bardziej realistyczne jest założenie, że odcinki mogą być położone dowolne.

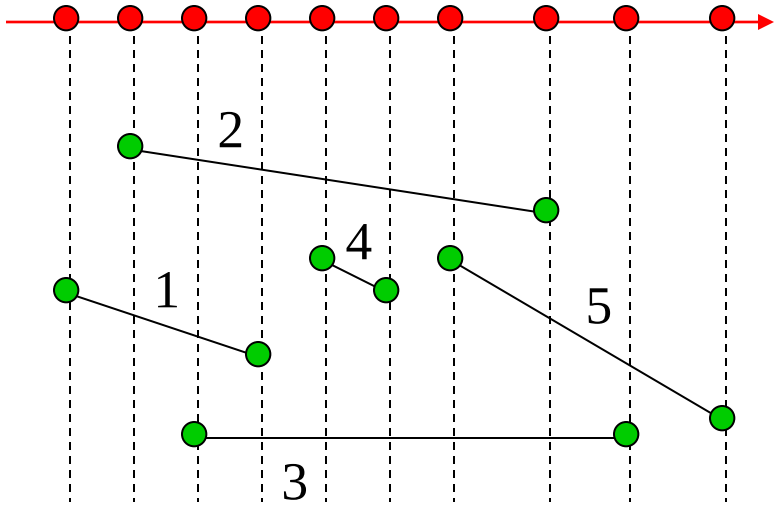
W takim przypadku możemy zamiast odcinków badać prostokąty, których przekątnymi są dane odcinki. Korzystając ze znanych już struktur możemy znaleźć wszystkie prostokąty przecinające dany prostokąt a następnie sprawdzić, które odcinki rzeczywiście go przecinają.

Jednak w szczególnych przypadkach ta metoda może okazać się bardzo nieefektywna.



Drzewo odcinków (segment tree).

W n-elementowym zbiorze S rozłącznych odcinków, chcemy znaleźć te z nich, które przecinają prostokąt R := $[x_1,x_2]\times[y_1,y_2]$.



Zrzutujmy zbiór S na oś x-ów. Niech I będzie zbiorem rzutów odcinków z S.

Rzuty końców odcinków wyznaczają podział osi na dwa rodzaje przedziałów: domknięte odpowiadające jednopunktowym rzutom końców oraz otwarte wypełniające resztę osi. Przedziały te nazywamy ***elementarnymi***.

Zbudujemy drzewo binarne, którego liśćmi będą przedziały elementarne. Przedział elementarny odpowiadający liściowi μ będziemy oznaczać przez Int(μ).

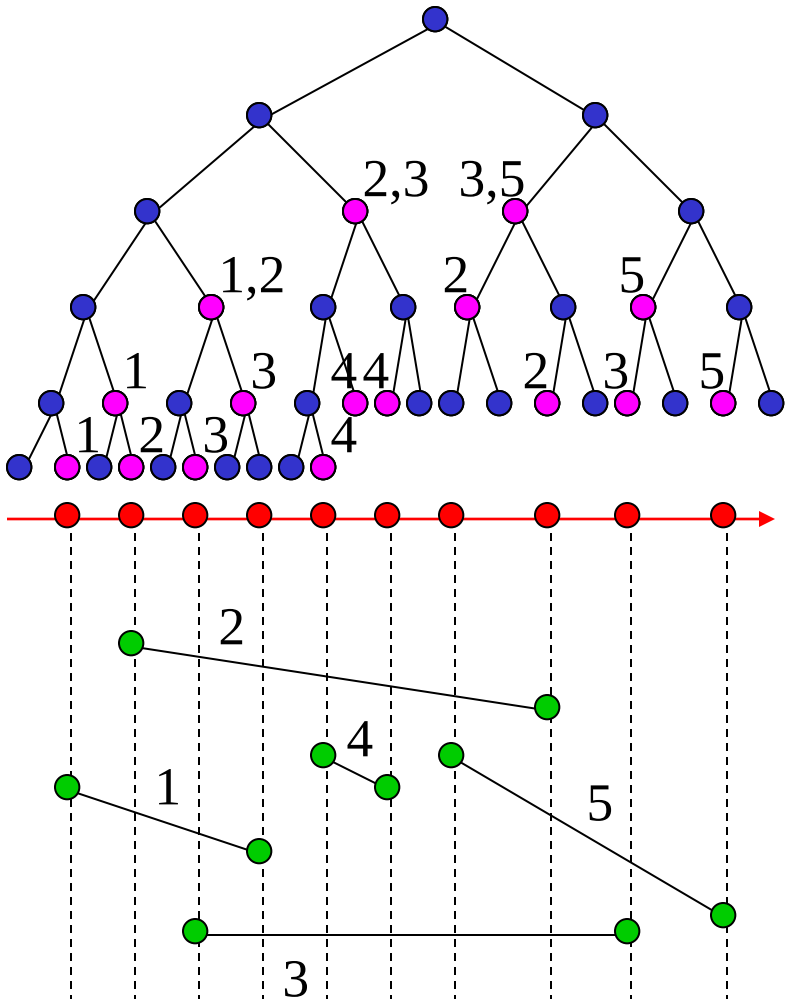
Niech v_l i v_r oznaczają synów v.

Drzewo odcinków jest zrównoważonym drzewem binarnym T, w którym

- liście odpowiadają przedziałom elementarnym określonym przez końce odcinków z I,

- każdy węzeł wewnętrzny odpowiada przedziałom będącym sumą przedziałów elementarnych liści poddrzewa o korzeniu w tym węźle, $Int(v) := Int(v_l) \cup Int(v_r)$,

- każdy węzeł wewnętrzny lub liść v pamięta przedział Int(v) i zbiór przedziałów (rzutów odcinków) $I(v) \subseteq I$, który składa się z takich $[x_1,x_2] \in I$, że $Int(v) \subseteq [x_1,x_2]$ oraz $\neg(Int(ojciec(v)) \subseteq [x_1,x_2])$ (w takim przypadku mówimy, że ***przedział pokrywa węzeł***).



procedure BUILDST(I)

posortuj końce przedziałów z I;

stwórz zrównoważone drzewo binarne T
dla przedziałów elementarnych;

for każdy węzeł v ***do*** oblicz Int(v);

for każdy s $\in$ I ***do*** INSERTST(korzeń T,s);

return T;

procedure INSERTST(v,s)

if Int(v) $\subseteq$ s

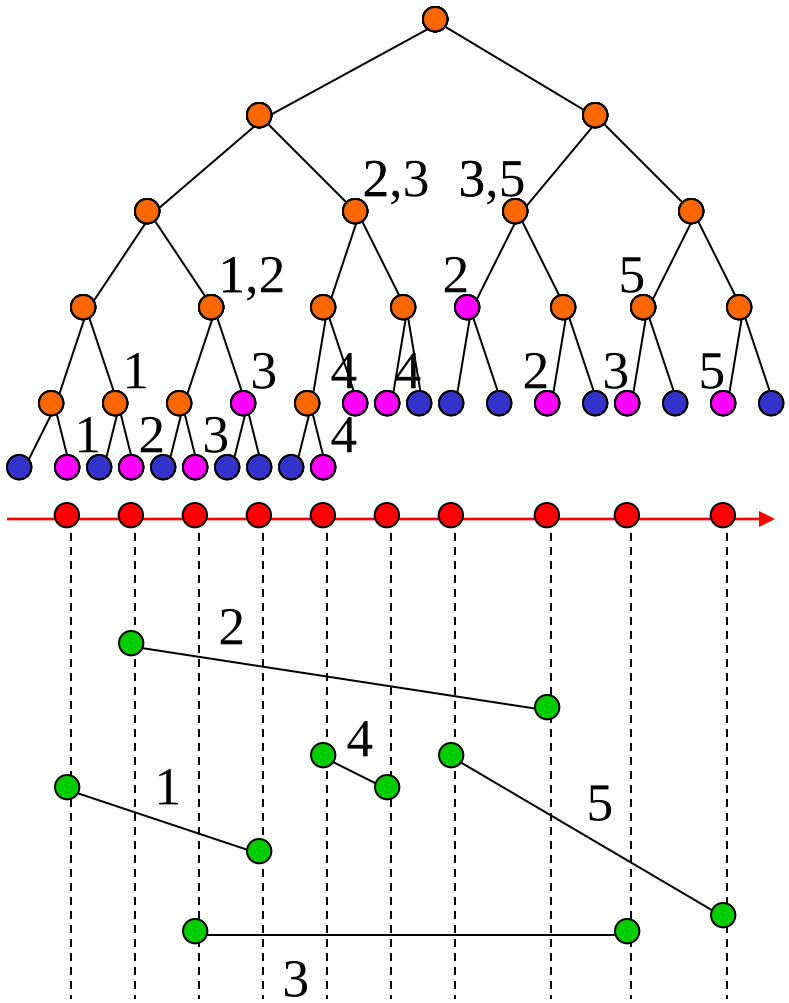
then zapisz s w v

else if Int(ls(v)) $\cap$ s $\neq \emptyset$

then INSERTST(ls(v),s)

if Int(rs(v)) $\cap$ s $\neq \emptyset$

then INSERTST(rs(v),s)



Spróbujmy znaleźć wszystkie odcinki przecinane przez daną prostą pionową o współrzędnej x-owej q_x.

procedure SEARCHST(v,q_x)

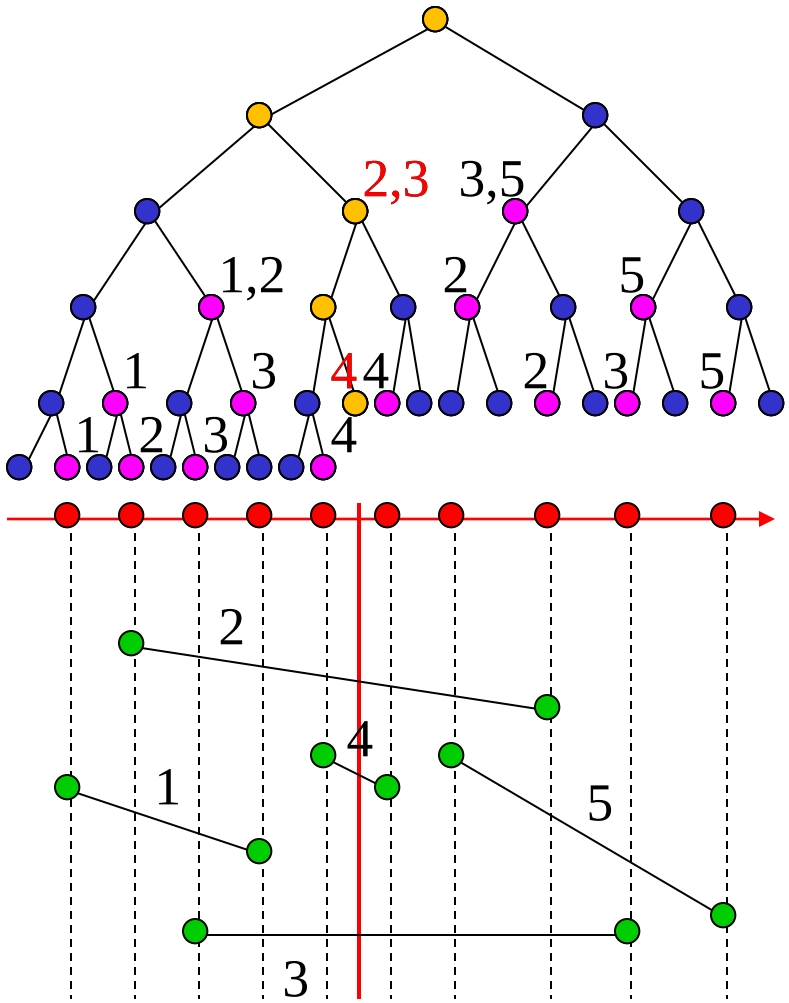
return wszystkie przedziały z I(v);

if v nie jest liściem

then if $q_x \in$ Int(ls(v))

then SEARCHST(ls(v),q_x)

else SEARCHST(rs(v),q_x)



Lemat.

Drzewo odcinków dla zbioru I zawierającego n przedziałów używa O(n log n) pamięci i można je zbudować w czasie O(n log n).

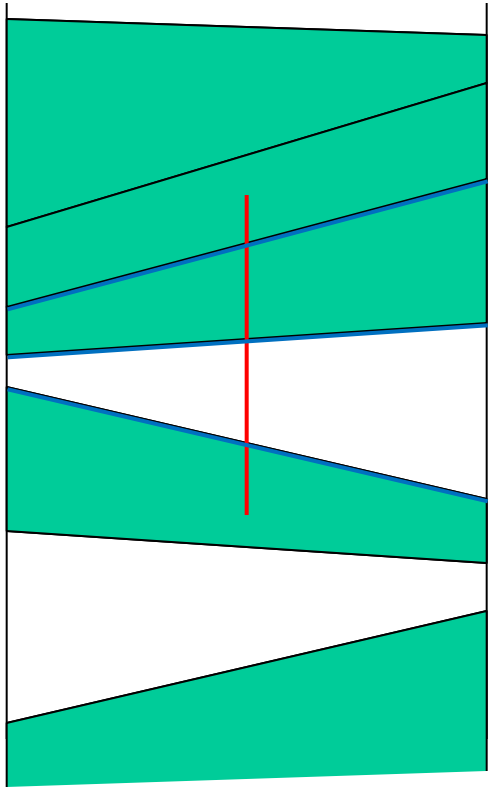
Dowód.

Każdy odcinek jest pamiętany w strukturze co najwyżej O(log n) razy (na każdym poziomie co najwyżej dwukrotnie). Zatem wszystkie zbiory I(v) aktualizujemy o dane kolejnego odcinka w czasie O(log n). Pozostałe operacje wykonujemy w czasie O(n log n).

Lemat.

Stosując drzewo odcinków, możemy podać wszystkie przedziały z I zawierające punkt zapytania w czasie O(k + log n), gdzie k jest liczbą znalezionych przedziałów.

Jeśli chcemy znaleźć przecięcia odcinków z prostokątem, to zamiast zbioru I(v) stosujemy np. zrównoważone drzewo poszukiwań binarnych lub uporządkowaną tablicę. Wykorzystujemy tu fakt, że odcinki ze zbioru I(v) przecinają cały pas odpowiadający v oraz są rozłączne.



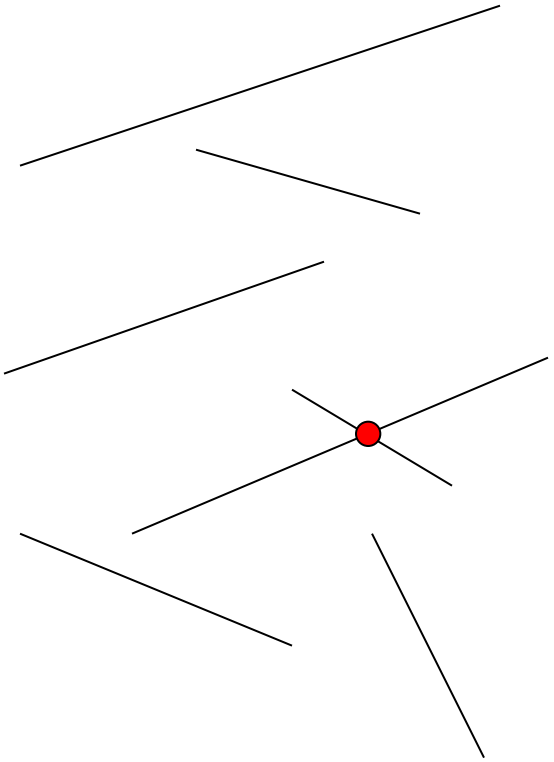
Lemat.

Niech S będzie zbiorem n rozłącznych odcinków na płaszczyźnie. Odcinki przecinające pionowy odcinek zapytania można znaleźć w czasie $O(k + \log^2 n)$, gdzie k jest liczbą znalezionych odcinków, stosując strukturę, którą można zbudować w czasie $O(n \log n)$ (musimy uporządkować odcinki przecinające pasy) używając $O(n \log n)$ pamięci.

Sprawdzanie istnienia pary przecinających się odcinków.

Problem.

Dany jest zbiór S zawierający n odcinków na płaszczyźnie. Sprawdź, czy istnieją dwa przecinające się odcinki.



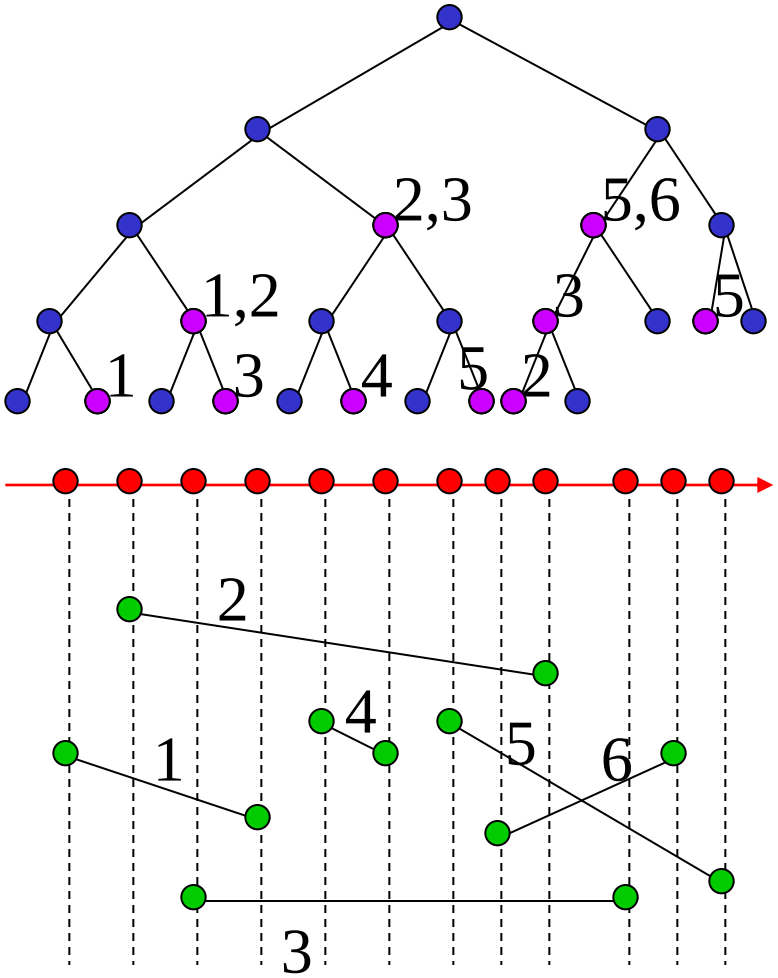
Rozpatrzmy model PRAM (Parallel Random Access Machine), w którym procesory komunikują się poprzez wspólną pamięć. Wyróżniamy różne rodzaje obliczeń w zależności od tego, czy procesory mogą jednocześnie czytać (CR - concurrent read) informacje z tej samej komórki pamięci czy nie (ER - exclusive read) oraz czy mogą jednocześnie zapisywać dane (CW - concurrent write) czy tylko osobno (EW - exclusive write). W przypadku CW określamy dodatkowo jaki sposób zapisu danych nie powoduje konfliktu.

Konstrukcja drzewa odcinków.

(a) Posortuj końce odcinków.

(b) Stwórz drzewo binarne zupełne o liściach odpowiadających przedziałom elementarnym (rysunek zawiera jeden poziom mniej).

(c) Procesory odpowiadające danym odcinkom z S przemieszczają się od korzenia w kierunku liści znajdując węzły (pokrywane), dla których $\mathrm{Int}(v) \subseteq [x_1,x_2]$ oraz $\neg(\mathrm{Int}(\mathrm{ojciec}(v)) \subseteq [x_1,x_2])$.



Lemat.

Wykorzystując O(n log n) procesorów CREW PRAM możemy stworzyć drzewo odcinków dla danego zbioru S w czasie O(log n).

Fakt (algorytm Cole'a).

Z pomocą O(n) procesorów CREW PRAM możemy posortować n liczb w optymalnym czasie O(log n).

Definicja.

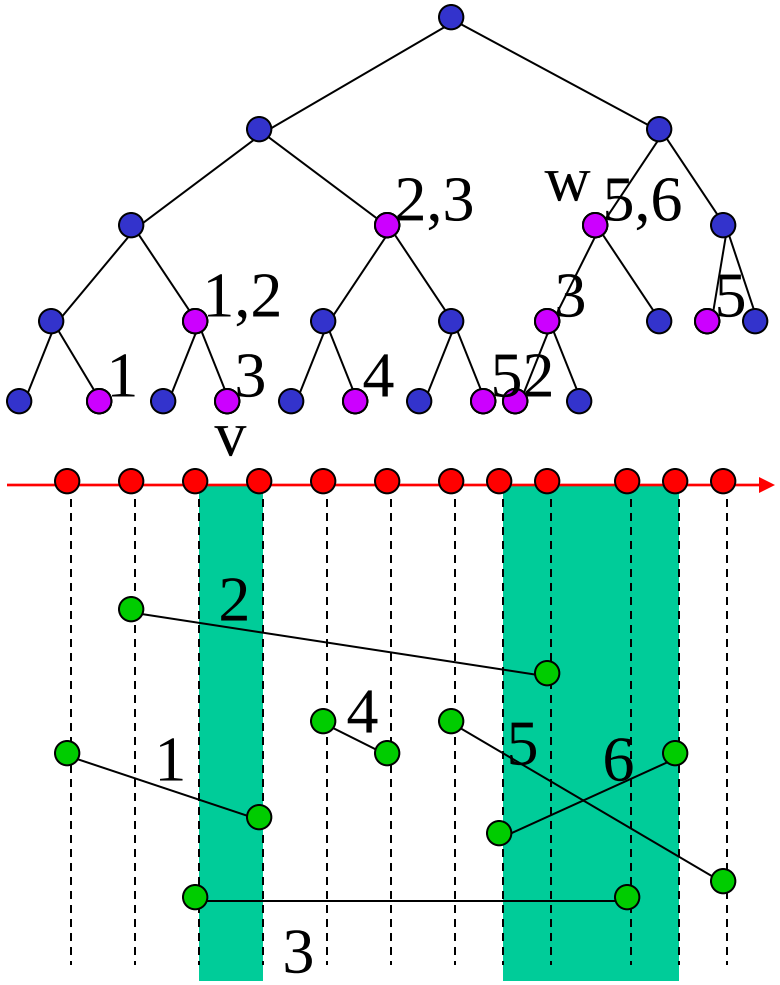
Niech Int(v) oznacza przedział na osi x-ów odpowiadający v, P(v) - pas Int(v) × (-∞,+∞) a Z(v) - zbiór odcinków pokrywających Int(v).

Niech H(v) oznacza uporządkowane listy odpowiadające punktom kolejnych przecięć Z(v) z brzegami P(v).

Dysponujemy O(n log n) procesorami CREW PRAM.

Każdej etykiecie pokrycia przypisany jest procesor. Zatem wszystkie listy H(v) możemy stworzyć w czasie logarytmicznym.

Dla każdego odcinka przypisujemy dodatkowe procesory do tych pasów, które nie są pokrywane przez ten odcinek, ale zawierają co najmniej jeden jego koniec.



Int(v) Z(v) = {3} Int(w)

Z(w) = {5,6}

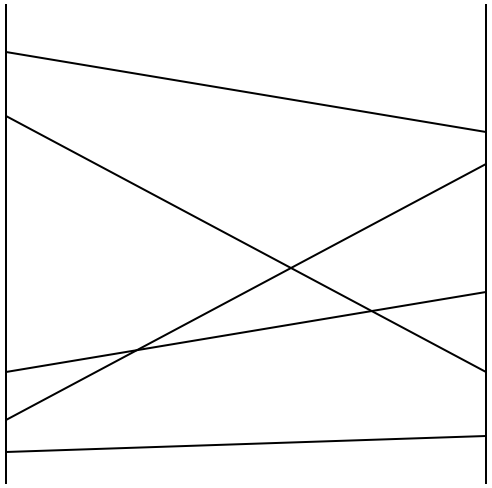
H(v) = {(3), (3)}

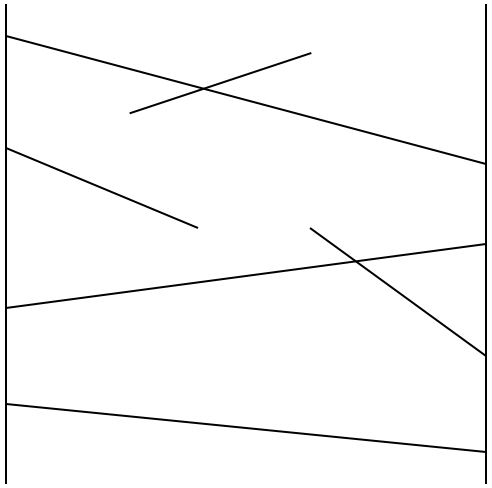
H(w) = {(5,6), (6,5)}

Mamy dwa przypadki.

(a) Istnieje przecięcie odcinków z Z(v). Możemy to stwierdzić badając kolejność odcinków na listach H(v).

(b) Odcinki z Z(v) przecinają się z odcinkami, które przecinały pas wyznaczany przez potomka v (niekoniecznie syna), a teraz mają co najmniej jeden koniec wewnątrz pasa P(v). Wtedy sprawdzamy, między którymi odcinkami z Z(v) znajdują się końce danego odcinka lub jego przecięcie z brzegiem pasa i czy jest to ten sam obszar.





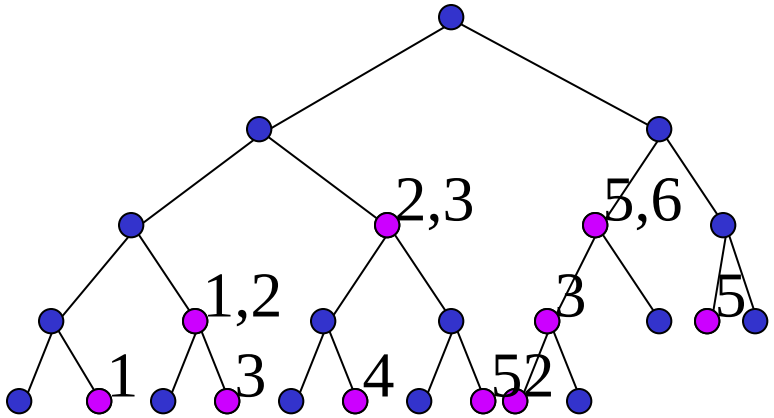
Lemat.

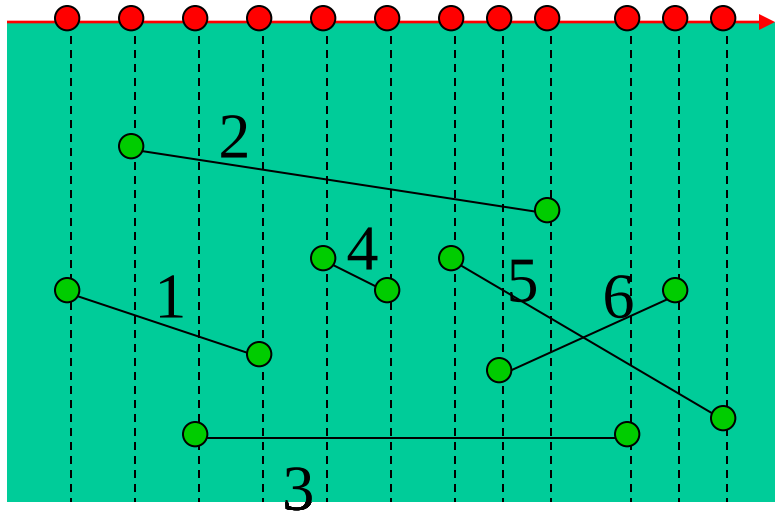
Wszystkie powyższe operacje można wykonać w czasie O(log n) równolegle w każdym węźle.

Fakt.

Sytuacja, w której przecinają się dwa odcinki nieprzecinające całego pasa nie ma miejsca. Takie przecięcie zostałoby wykryte na niższym poziomie drzewa.

for każdy wierzchołek v ***do***
  stwórz zbiory Z(v) i H(v);
  if odcinki z H(v) przecinają się
    then return „przecinają się”
    else porównaj H(v) z odcinkami
      mającymi co najmniej jeden
      koniec wewnątrz pasa P(v)
      if odcinki przecinają się
        then return „przecinają się”;
return „nie przecinają się”





Twierdzenie.
Z pomocą O(n log n) procesorów CREW PRAM możemy sprawdzić w czasie O(log n), czy w danym zbiorze S zawierającym n odcinków na płaszczyźnie istnieją co najmniej dwa, które się przecinają.

Drzewo czwórkowe (quadtree).

Drzewo czwórkowe dla n-elementowego zbioru punktów P (card(P) oznacza liczność zbioru P) definiujemy w następujący sposób. Niech $Q := [x_1,x_2]\times[y_1,y_2]$ będzie kwadratem.

- Jeśli $card(P) \le 1$, to drzewo czwórkowe zawiera pojedynczy liść, w którym pamiętamy zbiór P i kwadrat Q.

- W przeciwnym przypadku dzielimy kwadrat Q na ćwiartki Q_{NE}, Q_{NW}, Q_{SW}, Q_{SE} względem $x_m := (x_1+x_2)/2$ i $y_m := (y_1+y_2)/2$,

gdzie $P_{NE} := \{p \in P: p_x > x_m , p_y > y_m \}$,

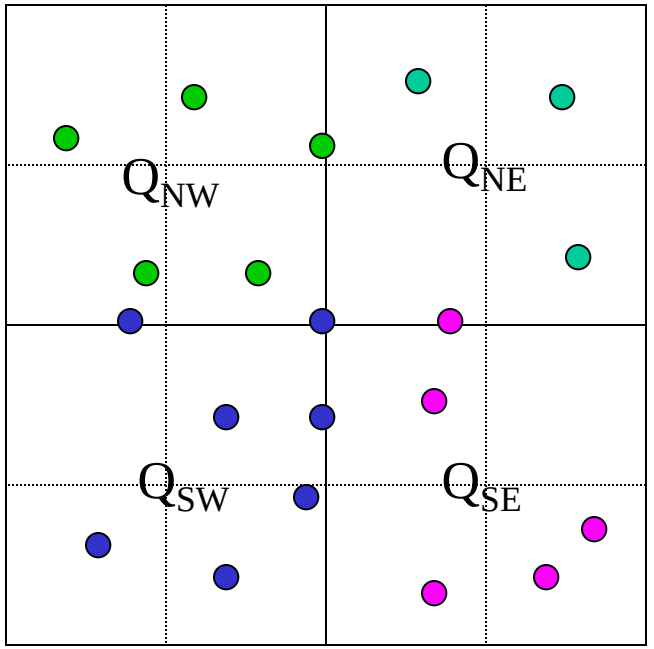
$P_{NW} := \{p \in P: p_x \le x_m , p_y > y_m \}$,

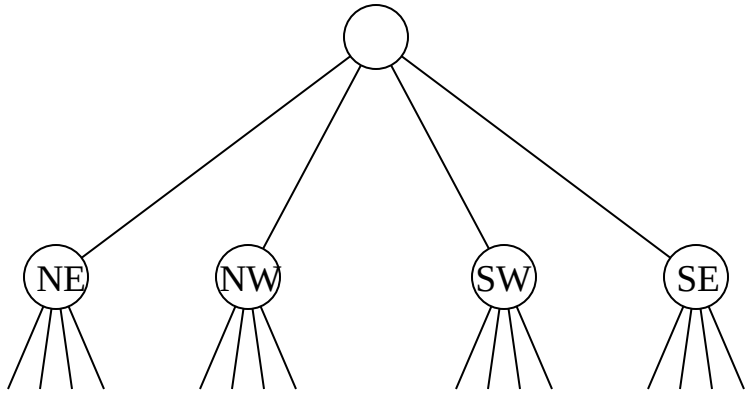
$P_{SW} := \{p \in P: p_x \le x_m , p_y \le y_m \}$,

$P_{SE} := \{p \in P: p_x > x_m , p_y \le y_m \}$.

Korzeniowi drzewa odpowiada kwadrat Q a jego synom - Q_{NE}, Q_{NW}, Q_{SW}, Q_{SE} .

W wierzchołku v trzymamy kwadrat Q(v).





Drzewa czwórkowe mogą być wykorzystane np. w celu kompresji jednotonalnych obrazów bitmapowych.

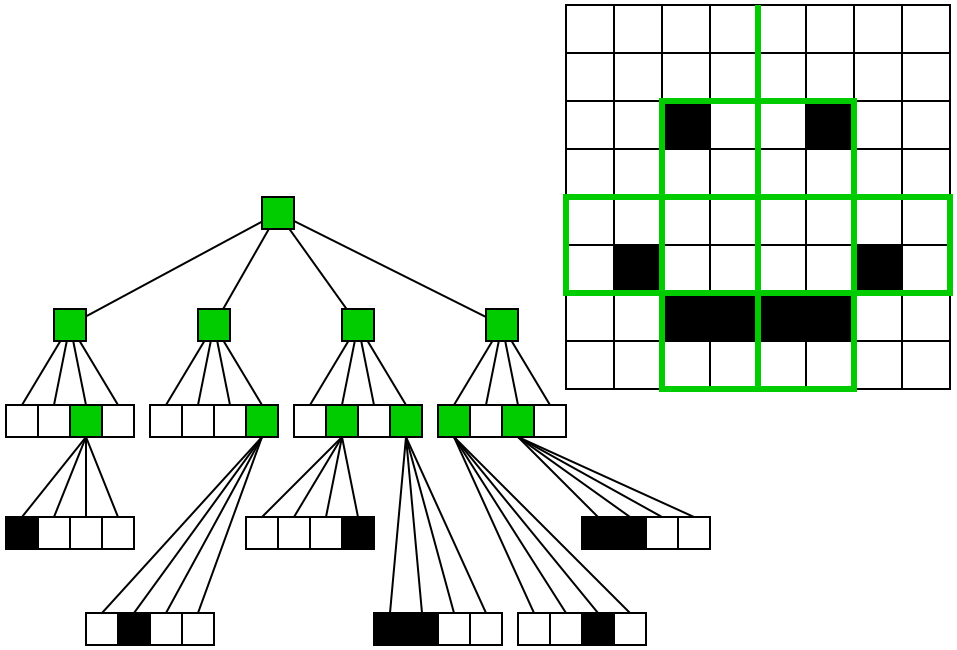
Można też skorzystać z nich do tworzenia sieci trójkątów dla efektywnych obliczeń numerycznych w szczególnych przypadkach płytek obwodów drukowanych (kierunki ścieżek różnią się o wielokrotność $\pi/4$). Siatki muszą :

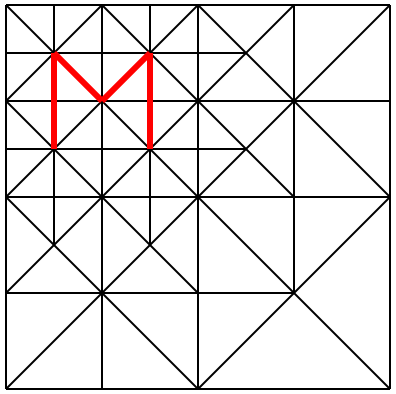
- być dopasowane (nie ma wierzchołków trójkątów na krawędziach innych trójkątów),

- uwzględniać dane (ścieżki są zawarte w krawędziach siatki),

- być dobrze ukształtowana (trójkąty muszą mieć określony kształt),

- być niejednolite (małe trójkąty blisko ścieżek, a duże – daleko).





Lemat.

Głębokość drzewa czwórkowego dla zbioru punktów P na płaszczyźnie wynosi co najwyżej $\log(s/c)+3/2$, gdzie c jest najmniejszą odległością między dowolnymi dwoma punktami z P, a s jest długością boku początkowego kwadratu Q zawierającego P.

Dowód.

Długość boku kwadratu odpowiadającego węzłowi wewnętrznemu na głębokości i wynosi $s/2^i$. Maksymalna odległość między dwoma punktami w takim kwadracie wynosi $s/2^{(i-1/2)}$. Zatem $s/2^{(i-1/2)} \geq c$, czyli $\log(s/c) + ½ \geq i$. Głębokość drzewa jest o jeden większa niż maksymalna głębokość węzła wewnętrznego.

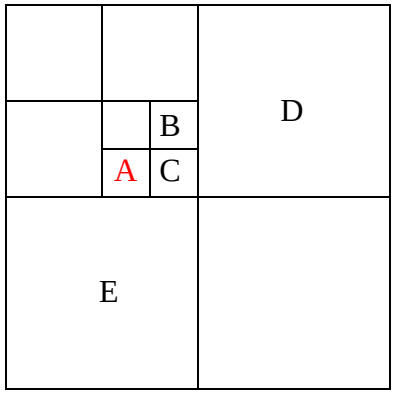
Lemat.

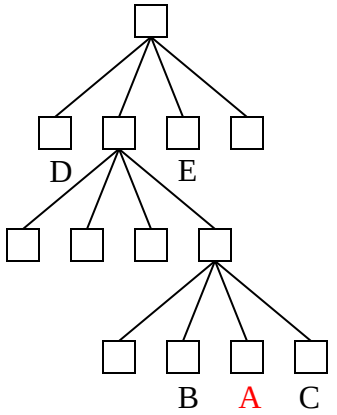
Drzewo czwórkowe o głębokości d przechowujące zbiór n punktów ma $O((d+1)n)$ węzłów i można je zbudować w czasie $O((d+1)n)$.

Dowód.

Liczba węzłów wewnętrznych na każdym poziomie szacuje się przez liczbę przechowywanych w nich punktów, czyli n. Liczba liści jest równa $3\times$(liczba węzłów wewnętrznych)+1 (dowód przez indukcję).

Podział kwadratu jest ***zrównoważony***, gdy długości boków dowolnych dwóch sąsiednich kwadratów różnią się co najwyżej dwukrotnie. Drzewo czwórkowe odpowiadające takiemu podziałowi nazywamy ***zrównoważonym***.



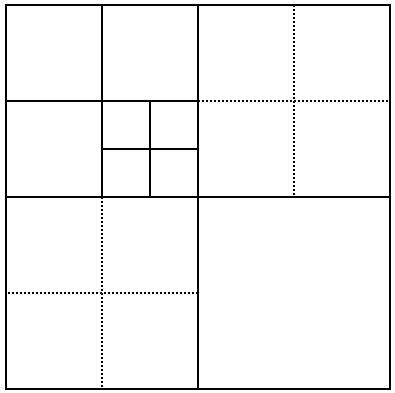


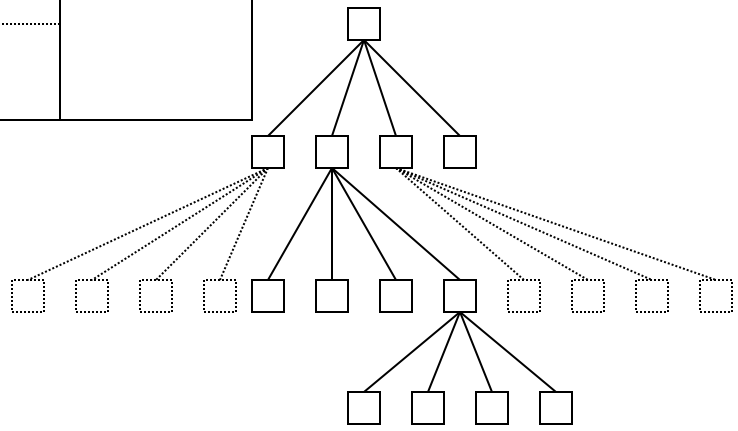
Lemat.

Niech T będzie drzewem czwórkowym o głębokości d. Sąsiada danego węzła v w T w dowolnym kierunku można znaleźć w czasie O(d+1).

Dowód.

Przeszukujemy drzewo czwórkowe w poszukiwaniu sąsiada (idąc w górę i w dół).





Algorytm równoważenia drzewa czwórkowego.

wstaw wszystkie liście z T do kolejki L;
while L nie jest pusta ***do***
    usuń liść μ z L
    if Q(μ) powinien zostać podzielony
        then przekształć μ w węzeł wewnętrzny i dodaj cztery liście;
            jeśli μ przechowuje punkt, to przepisz go do odpowiedniego liścia;
            wstaw cztery nowe liście do L;
            znajdź sąsiadów Q(μ), którzy powinni zostać podzieleni i wstaw ich do L;
return T;

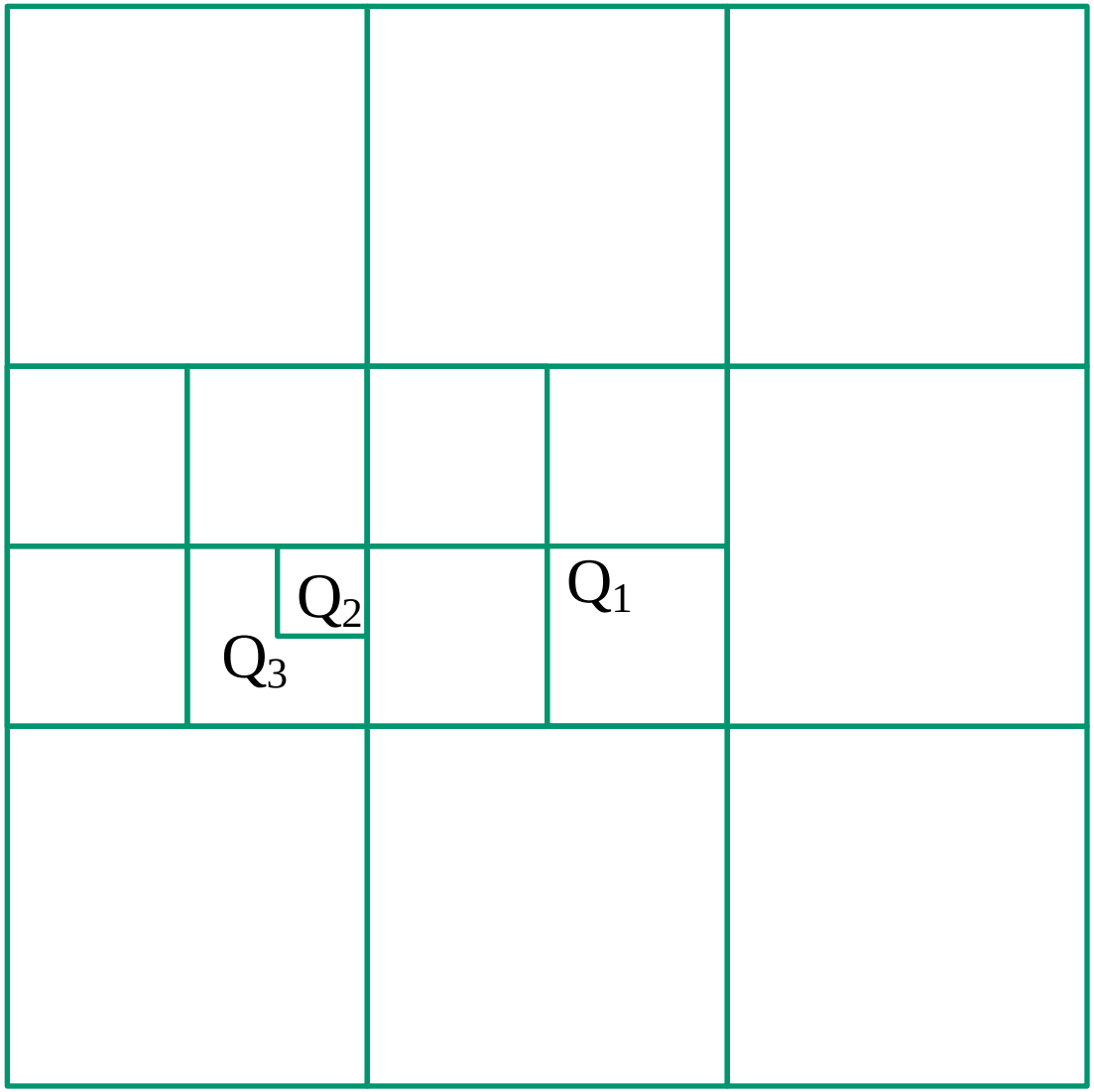
Lemat.

Niech T będzie drzewem czwórkowym o m węzłach i głębokości d. Drzewo T_B powstałe w wyniku zrównoważenia T będzie mieć O(m) węzłów i można je zbudować w czasie O((d+1)m).

Dowód.

Nazwijmy kwadraty odpowiadające węzłom drzewa T starymi, a dodawane w T_B – nowymi. Pokażemy, że wokół kwadratu (starego lub nowego) dzielonego w procesie równoważenia, co najmniej jeden z ośmiu otaczających go kwadratów tego samego rozmiaru jest stary. Załóżmy, że tak nie jest. Wybierzmy najmniejszy kwadrat Q_1, który nie ma danej własności. Skoro kwadrat ten jest dzielony, to przylega do niego kwadrat Q_2 o boku co najmniej czterokrotnie mniejszym. Weźmy kwadrat Q_3 mający bok dwukrotnie mniejszy od Q_1, sąsiadujący z nim i zawierający Q_2. Q_3 jest dzielony i wokół niego są same nowe kwadraty (bo należą do nowych kwadratów sąsiadujących z Q_1). Zatem dochodzimy do sprzeczności z założeniem o minimalności Q_1. Koszt tworzenia nowych węzłów możemy przypisać węzłom odpowiadającym starym kwadratom sąsiadującym z kwadratami dzielonymi – ich liczba jest co najwyżej 8 razy większa od rozmiaru T. Zatem rozmiar T_B jest co najwyżej 41 razy (8x(nowy dzielony+jego dzieci)+1) =(8x(1+4)+1) większy od T.

Czas potrzebny do obsługi jednego węzła wynosi O(d+1). Zatem złożoność algorytmu równoważenia wynosi O((d+1)m).


Q_1
Q_2
Q_3

Algorytm generowania siatek dla zbioru S odcinków na płaszczyźnie o kierunkach będących wielokrotnościami $\pi/4$.

zbuduj drzewo czwórkowe T na zbiorze S wewnątrz kwadratu Q;

stwórz drzewo zrównoważone T_B ;

for każdy liść μ drzewa T_B ***do***

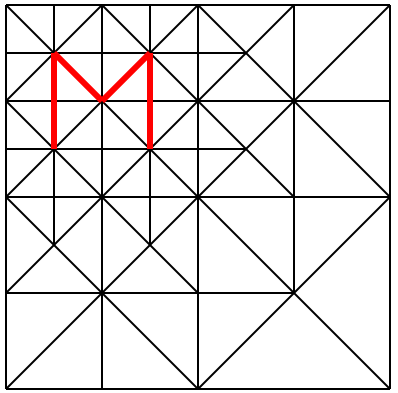
if odcinek z S przecina wnętrze $Q(\mu)$

then dodaj to przecięcie jako nową krawędź

else ***if*** wierzchołki trójkątów są tylko w rogach $Q(\mu)$

then dodaj przekątną $Q(\mu)$ jako nową krawędź

else dodaj punkt w środku $Q(\mu)$ i połącz go nowymi krawędziami ze wszystkimi wierzchołkami trójkątów na brzegu $Q(\mu)$;



Drzewo BSP (Binary Space Partition).

Binarny podział przestrzeni, w której znajduje się dany zbiór obiektów S, polega na podziale przestrzeni hiperpłaszczyznami (w przypadku R^2 – prostymi) tak, aby po zakończeniu podziału w każdym obszarze znajdował się fragment tylko jednego obiektu (porównaj z kd drzewem). Jeśli hiperpłaszczyzny są wyznaczane przez obiekty z S, to taki podział nazywamy ***autopodziałem***.

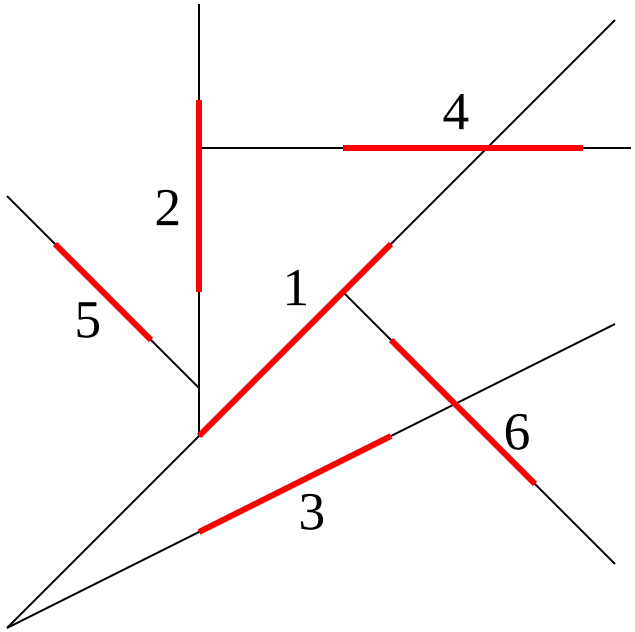
Strukturę opisującą podział przestrzeni nazywać będziemy ***drzewem BSP***, które jest drzewem binarnym T o następujących własnościach:

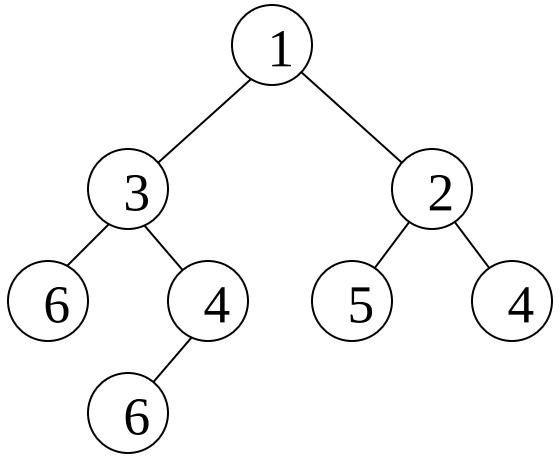
- Jeśli $card(S) \leq 1$, to T jest liściem. W liściu jest pamiętany odpowiedni fragment obiektu (o ile istnieje).

- Jeśli $card(S) > 1$, to korzeń v drzewa T pamięta hiperpłaszczyznę h_v wraz ze zbiorem S(v) obiektów, które są całkowicie zawarte w h_v. Lewym synem v jest korzeń drzewa BSP T^- dla zbioru $S^- := \{h^-_v \cap s: s \in S\}$, a prawym – korzeń drzewa BSP T^+ dla zbioru $S^+ := \{h^+_v \cap s: s \in S\}$, gdzie h^-_v i h^+_v oznaczają odpowiednio półprzestrzenie poniżej i powyżej hiperpłaszczyzny h_v.

Załóżmy, że zbiór S zawiera n nieprzecinających sie odcinków na płaszczyźnie.

procedure 2DBSP(S)
if card(S) ≤ 1
then stwórz drzewo T składające się z liścia, w którym jest pamiętany S;
return T
else $S^+ \leftarrow \{l^+(s_1) \cap s: s \in S\}$,
$T^+ \leftarrow$ 2DBSP(S^+);
$S^- \leftarrow \{l^-(s_1) \cap s: s \in S\}$,
$T^- \leftarrow$ 2DBSP(S^-);
stwórz drzewo BSP z korzeniem v,
lewym poddrzewem T^-, prawym –
T^+ i $S(v) = \{s \in S: s \subset l(s_1)\}$;
return T





Twierdzenie.

Przy założeniu losowych danych, oczekiwana liczba fragmentów tworzonych przez algorytm wynosi O(n log n). Odpowiednie drzewo BSP można obliczyć w oczekiwanym czasie $O(n^2 \log n)$.

Dowód.

Niech s_i będzie odcinkiem ze zbioru S. Spróbujemy obliczyć oczekiwaną liczbę przecięć odcinków o indeksach większych od i prostą $l(s_i)$ zawierającą s_i. Niech $dist_i(s_j)$ będzie równe liczbie odcinków przecinających $l(s_i)$ między s_i a s_j, jeśli $l(s_i)$ przecina s_j oraz $+\infty$ w przeciwnym przypadku.

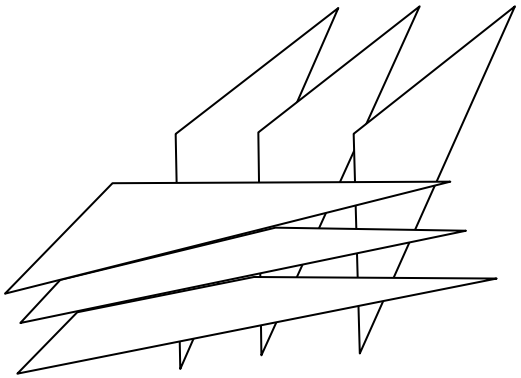
Wtedy prawdopodobieństwo, że prosta $l(s_i)$ przecina s_j możemy oszacować przez $P(l(s_i) \text{ przecina } s_j) \leq 1/(dist_i(s_j)+2)$, gdyż każdy z odcinków leżących między s_i a s_j musi zostać wybrany później niż s_i oraz wybór s_i musi poprzedzać s_j (im więcej odcinków tym oszacowanie jest grubsze).

Stąd E(liczba przecięć generowanych przez s_i (w obu kierunkach)) $\leq$ $\Sigma_{i \neq j} 1/(dist_i(s_j)+2) \leq 2 \Sigma^{n-2}_{k=0} 1/(k+2) \leq 2 (\ln n + \gamma) = O(\ln n)$.

Skoro dla jednego odcinka oczekiwana liczba przecięć innych odcinków wynosi O(ln n), więc dla całego zbioru S otrzymujemy oszacowanie O(n ln n). Ponieważ zbiór S zawiera n odcinków, więc oczekiwana liczba wszystkich fragmentów, jakie mogą powstać w trakcie działania algorytmu szacuje się przez n + O(n ln n), czyli jest O(n log n). Każdy krok podziału wymaga czasu O(n), więc oczekiwany czas konstrukcji drzewa wynosi $O(n^2 \log n)$.

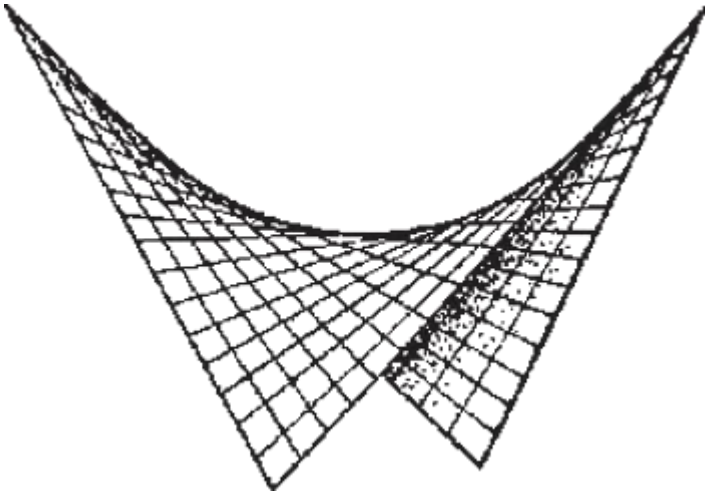
Lemat.

Przy założeniu losowych danych, oczekiwana liczba fragmentów obiektów tworzonych przez algorytm w R^3 wynosi $O(n^2)$.



Lemat.

Istnieją zbiory n nieprzecinających się trójkątów w R^3, dla których każdy autopodział (ale niekoniecznie każdy podział) ma rozmiar $\Omega(n^2)$.



Lemat.

Dla dowolnego zbioru n nieprzecinających się trójkątów w R^3 istnieje drzewo BSP rozmiaru $O(n^2)$.

Istnieje konfiguracja n nieprzecinających się trójkątów w R^3, dla której rozmiar każdego drzewa BSP jest $\Omega(n^2)$.

Dziekuję za uwagę.

Ćwiczenia 5.

1. Niech I będzie zbiorem przedziałów na prostej. Chcemy pamiętać przedziały tak, aby móc efektywnie określać te przedziały, które są całkowicie zawarte w danym przedziale $[x_1,x_2]$. opisz strukturę danych, która używa O(n log n) pamięci i daje odpowiedź na takie zapytanie w czasie O(log n + k), gdzie k jest liczbą odpowiedzi.

2. Dany jest zbiór S zawierający n rozłącznych odcinków na płaszczyźnie. Znajdź te odcinki, które przecinają pionowy promień biegnący ku górze z punktu (q_x,q_y) do nieskończoności.

a) Opisz strukturę danych dla tego problemu, która używa O(n log n) pamięci i ma czas odpowiedzi na zapytanie O(log n + k), gdzie k jest liczbą podawanych odpowiedzi.

b) Jak zbudować taką strukturę w czasie O(n log n) ?

3. Niech S będzie zbiorem n rozłącznych odcinków na płaszczyźnie. W czasie O(n log n) i używając O(n log n) pamięci skonstruuj strukturę, z pomocą której można znaleźć odcinki przecinające pionowy odcinek zapytania w czasie $O(k + \log^2 n)$, gdzie k jest liczbą znalezionych odcinków.

4. Wykorzystując O(n) procesorów CRCW PRAM (z jednoznacznym zapisem) znajdź w czasie stałym pierwsze wystąpienie jedynki w n-elementowym ciągu 0-1.

5. Wykorzystując O(n log n) procesorów CREW PRAM stwórz drzewo odcinków dla danego zbioru n odcinków S w czasie O(log n).

6. Załóżmy, że mamy drzewa czwórkowe na obrazach pikselowych I_1 i I_2. Oba obrazy mają rozmiar $2^k \times 2^k$ i zawierają tylko dwie intensywności 0 i 1. Podaj algorytm operacji boolowskich na tych obrazach, tzn. $I_1 \vee I_2$ i $I_1 \wedge I_2$.

7. Podaj przykład zbioru S zawierającego n rozłącznych odcinków na płaszczyźnie takiego, że dowolny autopodział S ma głębokość $\Omega(n)$.

8. Niech C będzie zbiorem n rozłącznych kół o promieniu 1 na płaszczyźnie. Pokaż, że dla C istnieje drzewo BSP rozmiaru O(n).