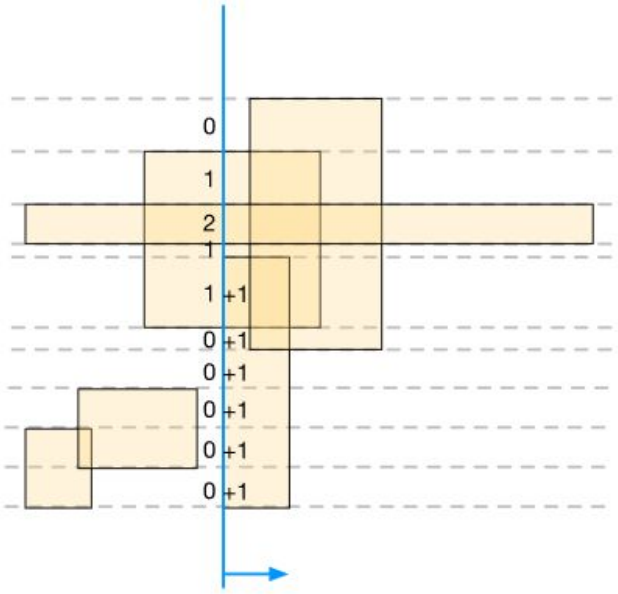
Unija pravougaonika (Union of rectangles)

Ana Vuksić



Opis problema

Kod ovog problema cilj je da nađemo koliko iznosi površina svih pravougaonika koji su nam dati. Prirodno je da će biti slučajeva kada ima preseka među njima, ne želimo da računamo više puta nego što je potrebno takve površine. Pravougaonici mogu biti zadati preko dokumenta ili nasumično napravljeni (mi biramo samo željeni broj pravougaonika).

Ulaz: *n pravougaonika / dokument sa n pravougaonika*

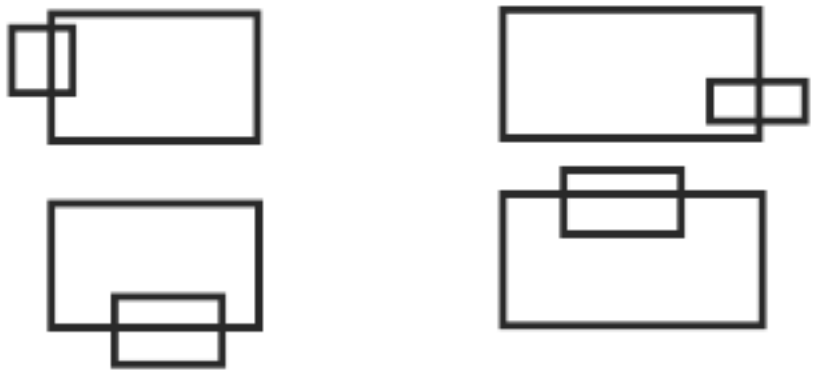
Izlaz: *veličina površine unije ovih pravougaonika*

Naivno rešenje problema

Za izradu naivnog algoritma potrebna nam je lista u kojoj ćemo čuvati sve pravougaonike čije površi treba da računamo. Ovaj algoritam se izvršava tako što redom uzimamo

pravougaonik po pravougaonik od učitanih pravougaonika. Svaki pravougaonik proverimo sa listom gotovih (one za koje smo već proverili imaju li preseke i dodali u listu pravougaonika za računanje površi). Ako ima preseka sa nekim pravougaonikom šaljemo oba pravougaonika funkciji koja će izvršiti potrebno deljenje pravougaonika na manje pravougaonike. I zatim svaki deo dodajemo u listu gotovih pravougaonika sa kojom ćemo dalje porediti druge pravougaonike.

Slika nekih mogućih preseka:



Funkcija koja će da podeli ove pravougaonike ako ima preseka je u nastavku.

```
/* If there is intersaction, split rectangle and add parts of rectangle to the list */
void ga24_unionofrectangles::SplitRectangleAndAddToList(Rectangle first, Rectangle* second)
{

    /*This is the case of completely overlapping rectangles, we can ignore this new ...*/
    if (first.x1 >= second->x1 && first.x2 <= second->x2
        && first.y1 >= second->y1 && first.y2 <= second->y2)
    {
        return;
    }

    /* In case we have left part of rectangle to add */
    if (first.x1 < second->x1 && first.x2 > second->x1)
    {
        AddRectangleToList(Rectangle(first.x1, max(first.y1,second->y1), second->x1, min(first.y2,second->y2)));
    }

    /* In case we have right part of rectangle to add */
    if (first.x2 > second->x2 && first.x1 < second->x2)
    {
        AddRectangleToList(Rectangle(second->x2, max(first.y1,second->y1), first.x2, min(first.y2,second->y2)));
    }

    /* In case we have top part of rectangle to add */
    if (first.y1 < second->y2 && first.y2 > second->y2)
    {
        AddRectangleToList(Rectangle(first.x1, second->y2, first.x2, first.y2));
    }

    /* In case we have bottom part of rectangle to add */
    if (first.y2 > second->y1 && first.y1 < second->y1)
    {
        AddRectangleToList(Rectangle(first.x1, first.y1, first.x2, second->y1));
    }

}
```

Na kraju imamo jedan prolaz kroz listu kako bi izračunali površinu koju zauzimaju.

```
/* Calculates area by going trough list of rectangles and adding there area */
int ga24_unionofrectangles::CalculateArea()
{
    Rectangle* tmp = First;
    int area = 0;
    while (tmp)
    {
        area += (tmp->x2 - tmp->x1) * (tmp->y2 - tmp->y1);
        tmp = tmp->next;
    }

    /* We need to free the memory taken by the list */
    FreeList(First);

    return area;
}
```

Algoritam sa pomerajućom pravom i segmentnim drvetom

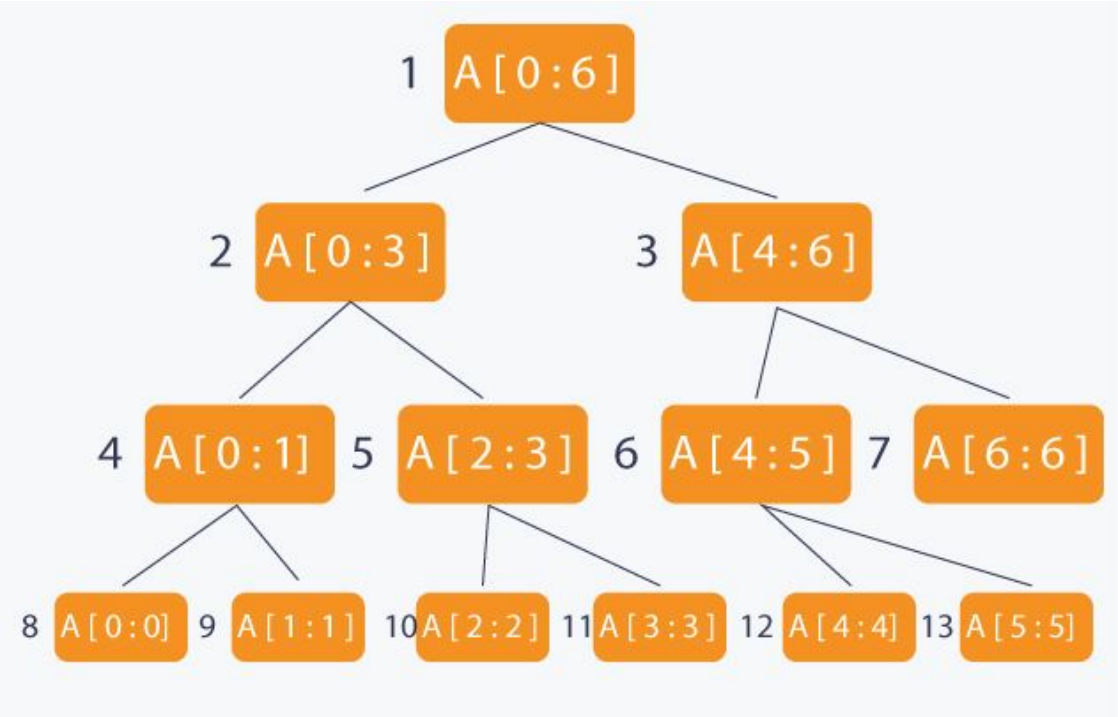
Efikasnost kod obrade ovog problema se može poboljšati korišćenjem pomerajuće prave. Želimo da izbegnemo sva poređenja koja smo izvršavali u naivnom algoritmu. Pomerajuća prava ide s leva na desno. A istovremeno posmatramo kao da je sve podeljeno u osnovne y intervale. Podelu na njih pravimo uzimajući y koordinate od unetih pravougaonika. Potreban nam je i brojač za svaki od njih, koji nam govori o tome koliko pravougaonika prekriva trenutno polje sa tom vrednošću x. Povećavamo brojač kada smo na levoj stranici pravougaonika (pomerajuća prava ide preko manje x koordinate pravouganika), a smanjujemo kada prođemu desnu stranu pravougaonika (pomerajuća prava ide preko veće x koordinate pravouganika).

Kako bismo računali površinu koju prekrivaju, treba da dodamo deo koji je između susednih x tačaka pored kojih je prošla prava i y koji su aktivni.

Algoritam ima za ideju da smanji potrebu sa prolazkom kroz sve x i y tačke pravougaonika kako bi se došlo do površina koju po delovima ovi pravougaonici pokrivaju. Imamo segmentno drvo u kome čuvamo brojače za y i time smo poboljšali izvršavanje jer je cena operacija nad segmentnim drvetom manja.

Segmentno drvo je specifično po tome što ima uvek fiksan broj “listova”, tako da se sama struktura drveta u praksi pravi od niza sa nama specifičnim čvorovima.

Izgled intervala u nekom segmentnom drvetu.



Prikaz dodavanja tačaka x.

```
void AddPointToX(int p, int index, int* xPoints, int* indexOfRectangles)
{
    int i = 0;
    while (i <= last && xPoints[i] <= p)
    {
        i++;
    }

    if (i <= last)
    {
        for (int j = last + 1; j > i; j--)
        {
            xPoints[j] = xPoints[j - 1];
            indexOfRectangles[j] = indexOfRectangles[j-1];
        }
    }

    xPoints[i] = p;
    indexOfRectangles[i] = index;
    last++;
}
```

Kako bi efikasno prolazili kroz stablo i mogli da vidimo koliko je y-ilona aktivno i samim tim da dođemo do vrednosti sa kojom trebamo da ponožimo vrednost između trenutnih x

koordinata koje su na razmatranju. Potrebno nam je da znamo koji je najlevlji list, sledeća funkcija pokazuje način za pronalaženje baš njega. Zašto najlevlji? Ako nađemo njega lako možemo iterirati kroz druge listove i tako pokupiti gde je brojač od y veći od nula. Gore prikazana slika segmentnog drveta nam može pomoći u razumevanju ovoga.

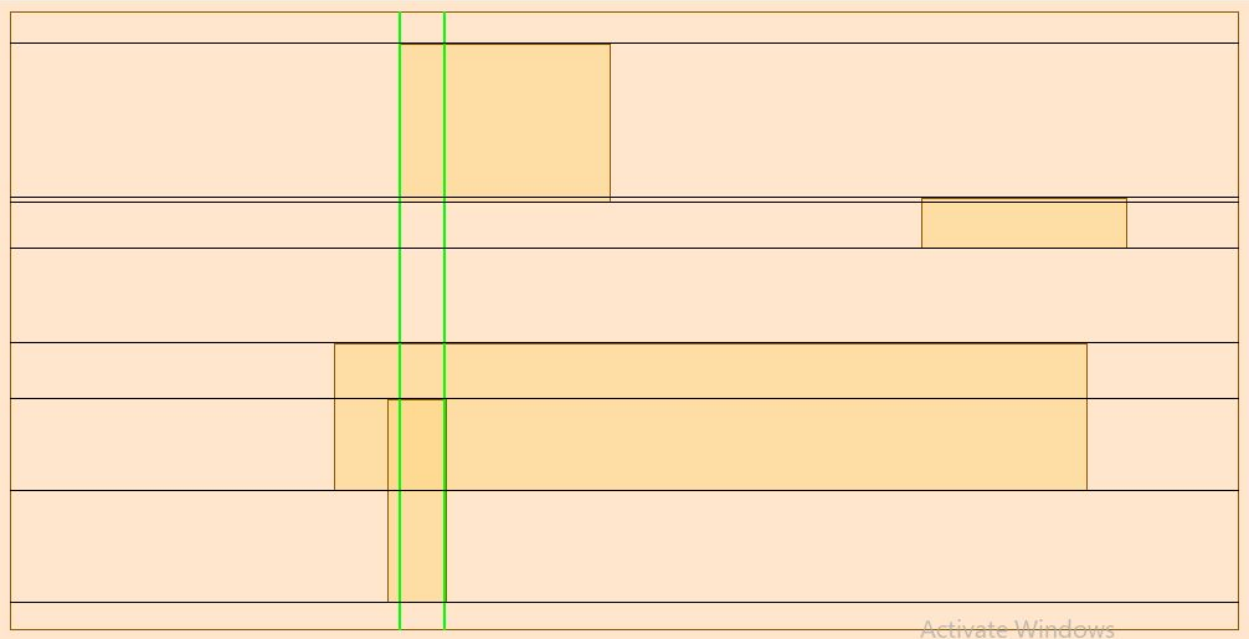
```
bool IsLeftMost(int i)
{
    bool isLeftMost = false;
    int pom =1;

    while(!((pow(2,pom)/2)>=(2*numberOfRect)))
    {
        if(i == (pow(2,pom)-1))
            isLeftMost = true;
        pom++;
    }

    if (i == 0)
    {
        isLeftMost = true;
    }

    return isLeftMost;
}
```

Funkcija koja poziva funkcije koje će da inkrementiraju potrebne y koordinate u segmentom drvetu ili dekrementiraju.

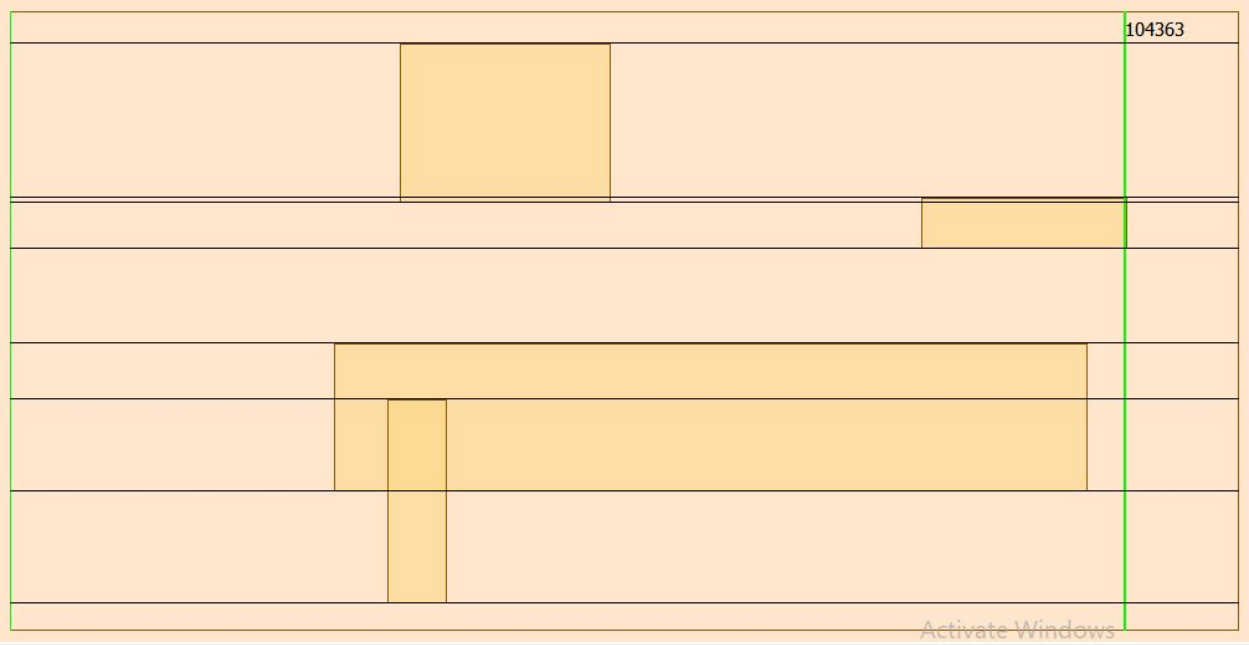
```
int CalculateArea(Node* tree, Rectangle* rectangles, int* xPoints, int* indexOfRectangles)
{
    int area = 0;
    for (int i = 0; i < 2*numberOfRect; i++)
    {
        if (i > 0)
        {
            area += CalculateMultiplier(tree) * (xPoints[i] - xPoints[i-1]);
        }

        IncrementTree(tree, xPoints[i], rectangles[indexOfRectangles[i]]);
    }

    return area;
}
```

Vizuelizacija algoritma

Imamo pomerajuću pravu koja ide sa leva na desno i prolazi kroz sve značajne tačke. Mi smo ovo predstavili sa dve prave, kako bi se bolje uvideo deo između x koordinata koji posmatramo i tako mogli lakše da shvatimo algoritam. Imamo iscrtane sve pravougaonike tokom celog izvršavanja, koji imaju određenu transparentnost kako bi mogli da uvidimo preseke. Imamo i paralelne y linije za sve y tačke među pravougaonicima. One nam pomažu da uvidimo rupu i intervala sa kojim bi množili udaljenost između trenutno aktivnih x koordinata. I na kraju kad se izvrši algoritam, imamo ispis ukupne površine algoritma.



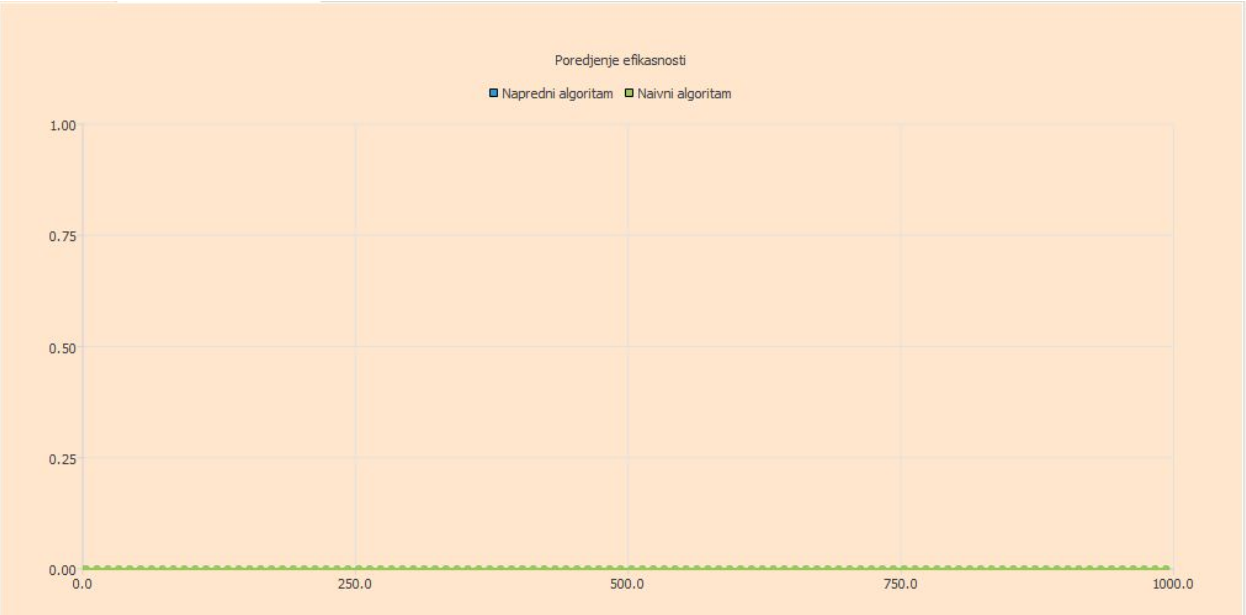
Posle prolaska kroz sve pravougaonike.



Poredjenje efikasnosti naivnog i naprednog algoritma

Naivni algoritma ima složenost O(n^2), može se brže izvršiti u slučaju da smo dobili pravougaonike bez preseka ili sa malim brojem preseka, ali u slučaju velikog broja preseka složenost se vidno povećava.

Kod poboljšanog algoritma imamo složenost O($nlogn$). Ubrzanje dobijamo zbog strukture segmentnog drveta koju koristimo. Cena izvršavanja upita na segmentnom drvetu je O(1), a cena ažuriranja O($logn$).



Testiranje ispravnosti algoritma

Ispod možemo naći test primere kojima smo se vodili tokom testiranja ovih algoritama.

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
Negativan ulaz	Ako je unet negativan broj kao vrednost za pravljenje pravougaonika	Neki broj manji od nule	Stavljanje površine na nulu
Bez pravougaonika	Posmatramo šta je izlaz ako nema na ulazu nikakvih pravougaonika	Odabiramo 0 kao ulaz	Stavljanje površine na nulu
Jedan nasumični	Posmatramo da li za isti nasumični pravougaonik imamo isti izlaz	Ulaz je jednak 1	Poklapanje rezultata naivnog i naprednog algoritma
Dva nasumična	Posmatramo da li za dva nasumična pravougaonika imamo isti izlaz	Ulaz je jednak 2	Poklapanje rezultata naivnog i naprednog algoritma

Pravougaonik sa sigurnim presekom	Izčitava primer iz datoteke koji ima presek pravougaonika	Ulaz je test.txt	Poklapanje rezultata naivnog i naprednog algoritma, vrednost 25
Veliki broj nasumičnih pravougaonika	Zadavanje 20 nasumičnih pravougaonika	Ulaz je jednak 20	Poklapanje rezultata naivnog i naprednog algoritma