

ZAVOD ZA PRIMIJENJENO RAČUNARSTVO  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA  
SVEUČILIŠTE U ZAGREBU

## **PARALELIZACIJA BREADTH FIRST SEARCH ALGORITMA**

SEMINARSKI RADI IZ KOLEGIJA Programske paradigme i jezici

Zagreb Rujan, 2012

## Sadržaj

1. Uvod	3
2. Implementacija	4
1. Node	4
2. Breadth-First search	5
3. Analiza algoritama	7
4. Zaključak	11
5. Literatura	12

## 1.Uvod

U seminarskom radu obrađena je tema paralelizacije *Bredth-First search* algoritma u programskom jeziku C#. Implementirati ćemo algoritam Breadth-First search u C# u paralelnoj, koristeći TPL knjižnicu, i neparalelnoj verziji te izraditi vremenske analize spomenutih algoritama na stablu s više ili manje čvorova. Podrazumijeva se da je za razumijevanje ove tematike potrebno baratanje znanjem teorije grafova i praktične primjene programskog jezika C#.

## 2. Implementacija

### 2.1.Node

Čvorove grafa implementirali smo kao klasu *Node* koja sadrži 3 atributa : *Children*, *isVisited* i *name*. Atribut *Children* je tipa *List<>* i predstavlja listu djece čvora *Node*. *isVisited* je tipa *bool* i predstavlja da li je čvor posjećene, te *name* koji je tipa *string* i predstavlja ime čvora. Također klasa *node* implementira metodu *AddChild* koja čvoru tipa *Node* dodaje djecu tipa *Node*.

```
public class Node
{
    List<Node> children = new List<Node>();
    bool isVisited;
    string name;

    public Node(string Name)
    {
        name = Name;
        isVisited = false;
    }

    public void AddChild (Node node)
    {
        children.Add(node);
    }

    public bool IsVisited
    {
        get { return isVisited; }
        set { isVisited = value; }
    }

    public List<Node> Children
    {
        get{return children;}
    }

    public string Name
    {
        get{return name;}
    }
}
```

Kod 1 Klasa Node

## 2.2. Breadth-First search

Koristeći pseudo kod Breadth-First search algoritma u klasu *BreadthFirstSearch* implementirali smo dvije metode *BFS* te metodu *PBFS*. Metoda *BFS* tipa *void*. Kao argumente prima dva parametra tipa *Node*, *Node start* i *Node end* koje predstavljaju početni i krajnji čvor pretrage stabla. Atribut *nodeQueue* tipa *Queue<node>* koristimo za spremanje djece čvora koji se trenutno obrađuje.

```
static class BreadthFirstSearch
{
    public static void BFS(Node start, Node end)
    {
        Queue<Node> nodeQueue = new Queue<Node>();
        nodeQueue.Enqueue(start);

        while (nodeQueue.Count != 0)
        {
            Node CurrentNode = nodeQueue.Dequeue();
            CurrentNode.IsVisited = true;

            foreach (Node Child in CurrentNode.Children)
            {
                if (Child.IsVisited == false)
                {
                    Child.IsVisited = true;

                    if (Child == end)
                        return;
                }
                nodeQueue.Enqueue(Child);
            }
        }
        return;
    }
}
...
```

**Kod 2 Metoda BFS**

Metoda *PBFS*, kao i metoda *BFS*, prima dva argumenta tipa *Node* kao početni i krajnji čvor za pretragu stabla međutim metoda *PBFS* za spremanje djece čvora, koji se trenutno obrađuju, koristi *nodeQueue* tipa *ConcurrentQueue<Node>* koja je thread-safe first in-first out (FIFO) kolekcija.

```

...
public static void PBFS(Node start, Node end)
{
    ConcurrentQueue<Node> nodeQueue = new ConcurrentQueue<Node>();
    nodeQueue.Enqueue(start);

    while (nodeQueue.Count != 0)
    {
        Parallel.ForEach<Node>(nodeQueue, CurrentNode =>
        {
            if (nodeQueue.TryDequeue(out CurrentNode))
            {
                foreach (Node Child in CurrentNode.Children)
                {
                    if (Child.Visited == false)
                    {
                        Child.Visited = true;
                        if (Child == end) return;
                    }
                    nodeQueue.Enqueue(Child);
                }
            }
        });
    }
    return;
}
...

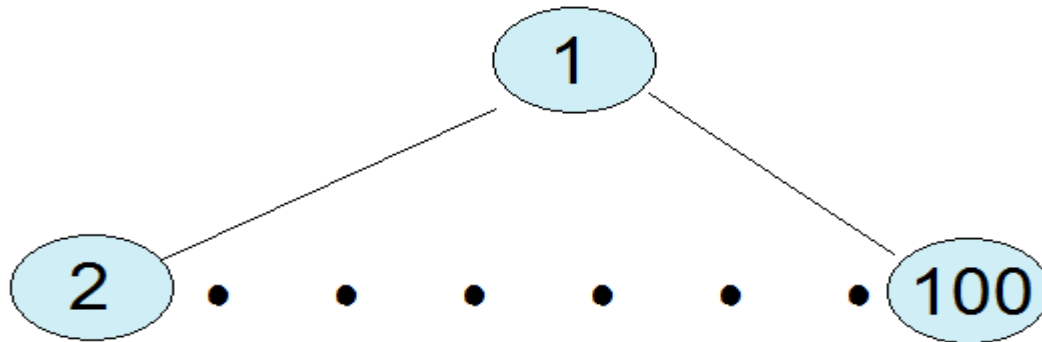
```

**Kod 3 Metoda PBFS**

### 3. Analiza algoritama

Analizu algoritama ćemo provesti nad 3 stabala.

Za potrebe prve analize stvorili smo stablo koje sadrži sto čvorova, jedan čvor roditelj kojem su preostali čvorovi djeca.



Stablo 1

Obilaskom Stabla 1 od čvora 1 do čvora 100 metodama *BFS* i *PBFS* dobili smo sljedeće vremenske rezultate.

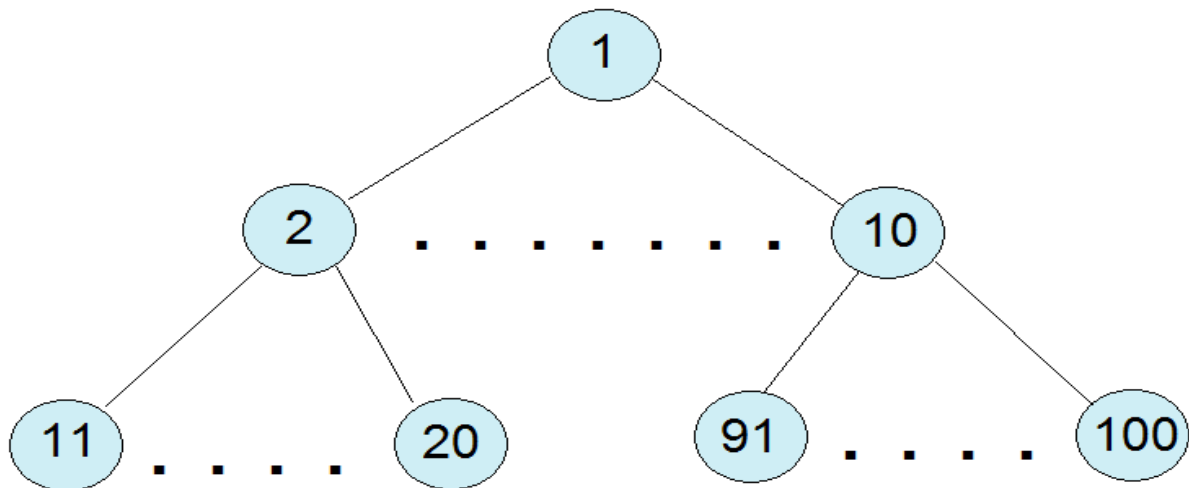
The screenshot shows a console window with the following text:

```
file:///C:/Users/Bojan/Documents/Visual Studio 2010/Projects/ConsoleApplication1/ConsoleApplic...
Time elapsed BFS 00:00:00.0011886
Time elapsed PBFS 00:00:00.0034556
```

Ispis 1 Rezultati prvog mjerenja

Suprotno od očekivanog, vidimo kako je vremensko trajanje obilaska ovog jednostavnog i „malog“ stabla gotovo tri puta brže koristeći neparalelnu metodu *BFS*. Ovakvo vremensko odstupanje rezultat je posljedica stvaranje novih procesa za pokretanje paralelne metode *PBFS* koje traje duže no obilazaka stabla na neparalelan način.

Za potrebe druge analize koristi ćemo stablo koje, kao u prethodnom slučaju, sadrži sto čvorova međutim ovdje će korijen stabla roditelj sadržavati deset djece i svako dijete će sadržavati još deset djece.



Stablo 2

Obilaskom Stabla 2 od čvora 1 do čvora 100 metodama *BFS* i *PBFS* dobili smo sljedeće vremenske rezultate.

```

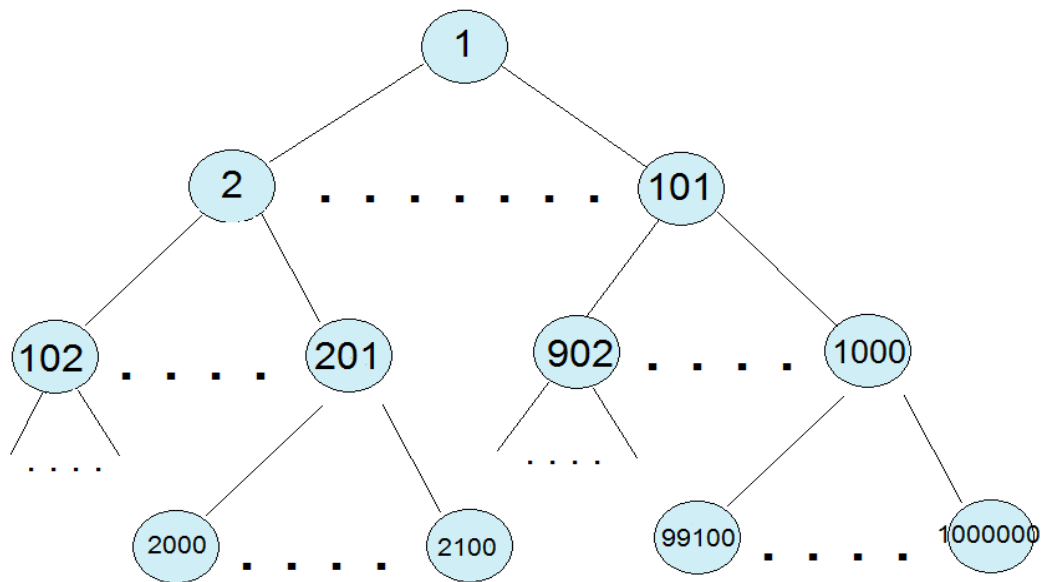
file:///C:/Users/Bojan/Documents/Visual Studio 2010/Projects/ConsoleApplication1/ConsoleApplic...
Time elapsed BFS  00:00:00.0011841
Time elapsed PBFS 00:00:00.0033857
    
```

Ispis 2 Rezultati drugog mjerenja

Slično kao u prvom primjeru ne paralelna metoda obilaska stabla je dala tri puta brže rezultate nego paralelna. Očito je da dubina stabla ne ovisi o brzini izvođenja algoritma.



Obzirom da prijašnje analize s „malim“ stablima daju otprilike jednake rezultate u ovoj analizi ćemo napraviti stablo od milijun čvorova gdje čvor roditelj ima sto djece, tih sto djece imaju djecu koja također imaju sto djece.



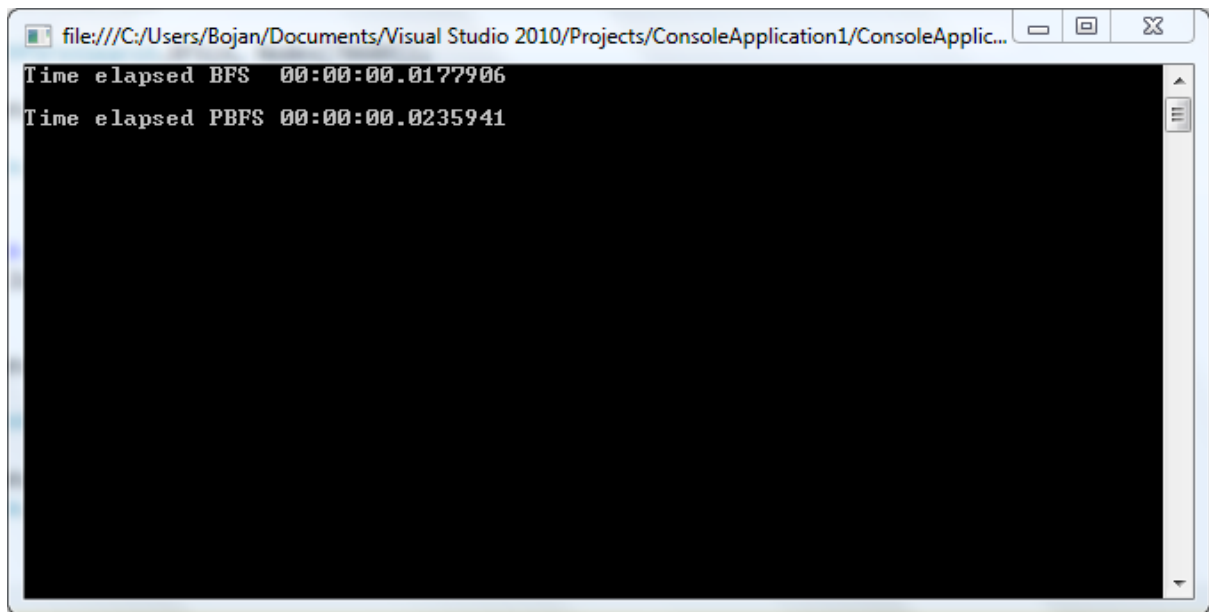
Stablo 3

Uzastopnim pokretanjima algoritama za ovo stablo od čvora 1 do čvora 1000000 dobili smo sljedeće rezultate.

```

file:///C:/Users/Bojan/Documents/Visual Studio 2010/Projects/ConsoleApplication1/ConsoleApplic...
Time elapsed BFS 00:00:00.0153716
Time elapsed PBFS 00:00:00.0141201
    
```

Ispis 3



```
file:///C:/Users/Bojan/Documents/Visual Studio 2010/Projects/ConsoleApplication1/ConsoleApplic...
Time elapsed BFS 00:00:00.0177906
Time elapsed PBFS 00:00:00.0235941
```

Ispis 4

U prvom mjerenju vrijeme pretraživanja paralelnim algoritmom je manje nego neparalelnim, međutim u drugom mjerenju to vrijeme je veće. Daljnja mjerenja, sa „većim“ grafovima nisu moguća zbog memorijskog ograničenja.

## 4. Zaključak

Za implementirani Breadth-First search algoritma na paralelan i neparalelan način, u programskom jeziku C#, ne možemo sa sigurnošću zaključiti da je paralelno izvođenje, što je za očekivati, brže od neparalelnog. Prvim i drugim mjerenjem zaključili smo da izvođenje algoritama nad „malim“ stablima je značajno brže neparalelno dok u trećem mjerenju to nije slučaj. Obzirom na dobivene rezultate možemo samo pretpostaviti da za veća stabla od testiranih dobivamo konkretnije rezultate u korist paralelnog izvođenja algoritma.

## 5. Literatura

1. [http://en.wikipedia.org/wiki/Breadth-first\\_search](http://en.wikipedia.org/wiki/Breadth-first_search)
2. <http://www.leniel.net/2008/01/breadth-and-depth-first-search.html>
3. <http://msdn.microsoft.com/en-us/library/dd267265.aspx>
4. <http://www.dzone.com/snippets/breadth-first-search-c>
5. <http://stackoverflow.com/questions/5111645/breadth-first-traversal-using-c-sharp>