Piotr Dąbrowski – Opracowanie Algorytmów z AISDE Lab2 – prof. Ogrodzki.

Poprzednim opracowaniu Lab1 umieściłem niepotrzebnie kod algorytmu poly ,który był w wykładzie , nie popełnię tego błędu jeszcze raz ,więc jeśli szukasz algorytmu rec_power wejdź na wykład i znajdziesz go na slajdach. Dodaję ten początek aby nikt nie narzekał.

Algorytm 1:

4. Zakodować algorytm rozwiązujący problem abstrakcyjny znajdowania NWD(a,b). Zastosować twierdzenie Euklidesa z odejmowaniem i metodę iteracyjną. Za operację dominującą przyjąć dzielenie odejmowanie ~~modulo~~. Zbadać złożoność optymistyczną $B(n)$ i pesymistyczną $W(n)$ w funkcji długości bitowej n większej z danych a, b. Liczbę operacji dominujących zliczać za pomocą wstawionego w kod licznika. Otrzymane wykresy aproksymować tak, żeby można było ocenić klasę złożoności. Na wspólnym wykresie podać przebieg aproksymujący i otrzymany z eksperymentu dla złożoności optymistycznej i pesymistycznej. Sformułować wnioski.

Algorytm rozwiązujący problem:

```
function result=Nwd(a,b)
if a==0|b==0
if a==0
result=b;
end
if b==0
result=a;
end
else
while a~=b
        if(a>b)
            a=a-b;
        else
            b=b-a;
        end
end
result=a;%równie dobrze może być result=b ponieważ obie
end       %zmienne przechowują informacje o NWD
end
```

Algorytm byłby dużo prostszy gdybyśmy mogli skorzystać z dzielenia modulo ,czyli standardowego algorytmu Euklidesowego. Niestety twierdzenie Euklidesa z odejmowaniem i za pomocą iteracji jest nieco bardziej skomplikowane.

Algorytm 2:

4. Zakodować algorytm rozwiązujący problem abstrakcyjny znajdowania NWD(a,b). Zastosować twierdzenie Euklidesa z dzieleniem modulo i metodę rekurencyjną. Za operację dominującą przyjąć dzielenie modulo. Zbadać złożoność oczekiwaną A(a) oraz A(n), n=db(a). Na wspólnym wykresie podać przebiegi eksperymentalne i teoretyczne. Sformułować wnioski.

Algorytm rozwiązujący problem:

```
function result=NWD2(a,b)
if b == 0
result=a;
else
result=NWD2(b,mod(a,b));
```

```
end
end
```

Algorytm v2-(ALE NIE REKURENCYJNY!)

```
function result=NWD2(a,b)
while b~=0
        temp = mod(a,b);
        a = b;
        b = temp;
    end
result=a;
end
```

Algorytm 3:

2. Opracować i zakodować algorytmy pierwszego rodzaju: rekurencyjny i iteracyjny rozwiązujący problem abstrakcyjny obliczania sumy s dla n wyrazów ciągu harmonicznego {a/i}, i=1,2,3,... o pierwszym wyrazie a. a) Wykreślić s(n) dla dwóch wartości a. Porównać wyniki obu algorytmów. b) Wykreślić dla obu algorytmów złożoność t(n) teoretyczną i eksperymentalną. Operacja dominująca: dodawanie i dzielenie. Podać wnioski.

Algorytm iteracyjny:

```
function sum=AdderH(a,i)
c=1;sum=0;
while c<=i
    sum=sum+(a/c);
    c=c+1;
end
end
```

Algorytm rekurencyjny:

```
function sum=AdderHR(a,i)

summof=0;c=a/i;
if i>0
   summof=c+AdderHR(a,i-1);
end
sum=summof;
end
```

Oba algorytmy są malutkie i proste do implementacji , także największym problemem w tym przypadku może być jedynie badanie złożoności algorytmów itd.

Algorytm 4:

Lab 2, zestaw 5, Ogrodzki

Opracować i zakodować algorytmy pierwszego rodzaju: rekurencyjny i iteracyjny rozwiązujący problem abstrakcyjny obliczania wyrazów ciągu {x(n)} opisanego wzorem rekurencyjnym x(n+1)=x(n)*(2-ln(x(n))). Przyjąć x(1)=y.

Algorytm iteracyjny:

```
function  sum=Calculate(n,x)
c=1;
if n==1
   sum=0;
else
   while c<n
       sum=x*(2-log(x));
       x=sum;
       c=c+1;
   end
end
end
```

Algorytm rekurencyjny:

```
function sum=CalculateRec(n,x)
   if n>1
      x=CalculateRec(n-1,x);     % <-zdobywamy informacje o x(n-1)
      sum=x*(2-log(x));          % czyli w zadanym wzorze x(n), a potem
   else                          % poprostu wyliczamy naszego x(n)
      sum=x*(2-log(x));
   end
end
```

W obu algorytmach wartość x oznacza wartość początkową jaką nadajemy x(1). Pozwala to na ustalenie wartości x(1)=y ,która najpewniej oznacza jakieś dowolne wartości jakie pomogą nam przy badaniu różnych parametrów. Na początku byłem nieco zmieszany wartościami które wychodziły , lecz potem sam obliczyłem wartości dla poszczególnych danych i wszystko się zgadza, przy wartościach większych od 5-6 zmiany zauważalne są dopiero p o 5-8 miejscach po przecinku i dalej dlatego nie widać zmian wartości w mathlabie.

Algorytm 5:

3. Opracować i zakodować algorytmy pierwszego rodzaju: rekurencyjny i iteracyjny rozwiązujący problem abstrakcyjny obliczania sumy s dla n wyrazów ciągu geometrycznego o pierwszym wyrazie a=1 i ilorazie q=1/3. a) wykreślić s(n). Porównać wyniki obu algorytmów. b) Wykreślić dla obu algorytmów złożoność t(n) teoretyczną i eksperymentalną. Operacja dominująca: mnożenie. Podać wnioski.

Tutaj wdg. mnie najłatwiej byłoby wstawić algorytm wyliczający sumę ciągu geometrycznego z definicji ,ale w poleceniu jesteśmy zmuszeni do skorzystania z obliczania krok po kroku.

Algorytm wdg. mnie:

```
function sum=SumOfGeo(a,q,n)
sum=(a*(1-q^n))/(1-q);
end
```

-gdzie argumenty wejściowe możemy dowolnie modyfikować.

Algorytm Iteracyjny:

```
function sum=SumOfGeoIt(a,q,n)
i=0;sum=0;
while i<n
    sum=sum+(a*q^i);
    i=i+1;
end
end
```

Algorytm Rekurencyjny:

```
function sum=SumOfGeoRec(a,q,n)
   if n>1
      x=SumOfGeoRec(a,q,n-1);
      sum=x+a*(q^(n-1));
   else
      sum=a;
   end
end
```

Algotytm 6:

5. Zakodować algorytm rozwiązujący problem abstrakcyjny znajdowania jednocześnie maksimum i minimum w zbiorze n-elementowym. Za operację dominującą przyjąć porównanie. Wstawić wydruki pozwalające na podejrzenie danych wejściowych i wyjściowych na każdym poziomie rekurencji i narysować drzewo rekurencji dla n=6. Zbadać złożoność w funkcji n. Otrzymaną zależność wykreślić na wspólnym wykresie razem z teoretyczną. Sformułować wnioski.

W tym algorytmie za bezsens uważam korzystanie z rekurencji ,ale na siłę to można jechać z Warszawy do Gdańska przez Wrocław ,ale jak kto woli. Zamieszczę tutaj dwa algorytmy rozwiązujące ten problem abstrakcyjny. 1-Normalną metodę , 2-Metodę rekurencyjną

Algorytm iteracyjny „wdg mnie":

```
function [max,min]=FINDME(n)
max=0;min=0;x=length(n);i=1;
while i<=x
  if n(i)>max
      max=n(i);
  elseif  n(i)<min
      min=n(i);
  end
  i=i+1;
end
end
```

Algorytm rekurencyjny:

```
function [max,min]=FindMinMax(n)
max=0;min=0;x=length(n);
  if x>1
      [max,min]=FindMinMax(n(1:(x-1)));
      if n(x)>max
         max=n(x);
      elseif  n(x)<min
         min=n(x);
      end
  else
      if n(1)>max
         max=n(1);
      elseif  n(1)<min
         min=n(1);
      end
  end
end
```

-„Myślę ,że ten algorytm dało się zrobić jakoś krócej".

Uwagi(Ważne): Tutaj należy pamiętać o tym ,że przy wywoływaniu owej funkcji należy skorzystać z 2 wyjść ,bo A=FINDME(n) zwróci tylko jedną wartość ->pierwszą z lewej czyli w tym wypadku max. Należy skorzystać z takiej formułki: [A,B]=FINDME(n). Drugą uwagą jest to że na wejście należy podać wektor złożony z n liczb. Natomiast ostatnią uwagą jest to ,że nie rozumiem o co chodzi z opcją podejrzenia danych wejściowych na każdym poziomie rekurencji w każdym razie jak trzeba dodać jedną linijkę kodu to myślę ,że żaden problem. Jak ktoś wie i rozumie to niech doda do dysku odpowiednie sprostowanie!

Algorytm 8:

Opracować i zakodować algorytmy pierwszego rodzaju: rekurencyjny i iteracyjny rozwiązujący problem abstrakcyjny obliczania sumy s dla n wyrazów następującego ciągu {(-1)^i *(1/(2i)!)*x^(2i), i=0,1,2,..., x=π/2. a) Wykreślić s(n). Porównać wyniki obu algorytmów. Zastanowić się, do jakiej wartości dąży ten ciąg. b) Wykreślić dla obu algorytmów złożoność t(n) teoretyczną i eksperymentalną. Operacje dominujące: dodawanie, mnożenie, dzielenie. Podać wnioski.

Algorytm ten to po prostu edycja algorytmu 5. Należy w obu przypadkach po prostu zmienić formułę przy „sum"

Algorytm iteracyjny dla podglądu:

```
function sum=SumOfIt(x,n)
i=0;sum=0;
while i<=n
    sum=sum +( (-1)^i *(1/factorial(2*i))* x^(2*i) );
    i=i+1;
end
end
```

Algorytm rekurencyjny:

```
function sum=SumOfRec(x,n)
   if n>=1
       sum=SumOfRec(x,n-1)+( (-1)^n *(1/factorial(2*n))* x^(2*n) );
   else
      sum=( (-1)^n*(1/factorial(2*n))*x^(2*0) );
   end
end
```

Mam nadzieję ,że pomogłem ..