Brza iterativna varijanta algoritma za DFT_n, $n = 2^m$

U daljnjem stalno pretpostavljamo da je n potencija od 2, tj. $n = 2^m$, $m \in \mathbb{N}$.

Da bismo dobili iterativnu varijantu algoritma iz rekurzivne, treba "raspabirati" rekurziju, tj. prvo pogledati što se događa na pojedinim nivoima rekurzije.

Rekurzivni algoritam ima tipični oblik obilaska binarnog stabla i to u "post"-uredaju - prvo obilazimo oba djeteta, obavljamo neki posao i vraćamo se natrag.

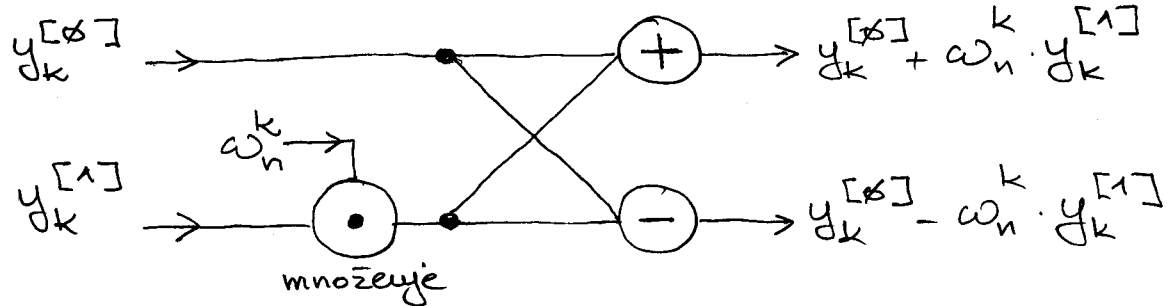
Naš "neki" posao je tzv. "butterfly" ili leptir operacija u unutarnjoj petlji, nakon rekurzivnih poziva.

Tu operaciju možemo shematski prikazati kao kombinatorni ili elektronički sklop (engl. "circuit") koji iz 2 ulaza $y_k^{[0]}$ i $y_k^{[1]}$, uz zadani faktor ω_n^k, generira 2 izlaza zadana (definirana) relacijama

$$y_k = y_k^{[0]} + \omega_n^k \cdot y_k^{[1]}$$

$$y_{k+n/2} = y_k^{[0]} - \omega_n^k \cdot y_k^{[1]}.$$

Standardna shema je:



Ove kutijice možemo zamišljati i kao vrlo jednostavne procesore (kutijice), koje možemo međusobno vezati da napravimo računalo ("veliki" krug) za računanje DFT_n.

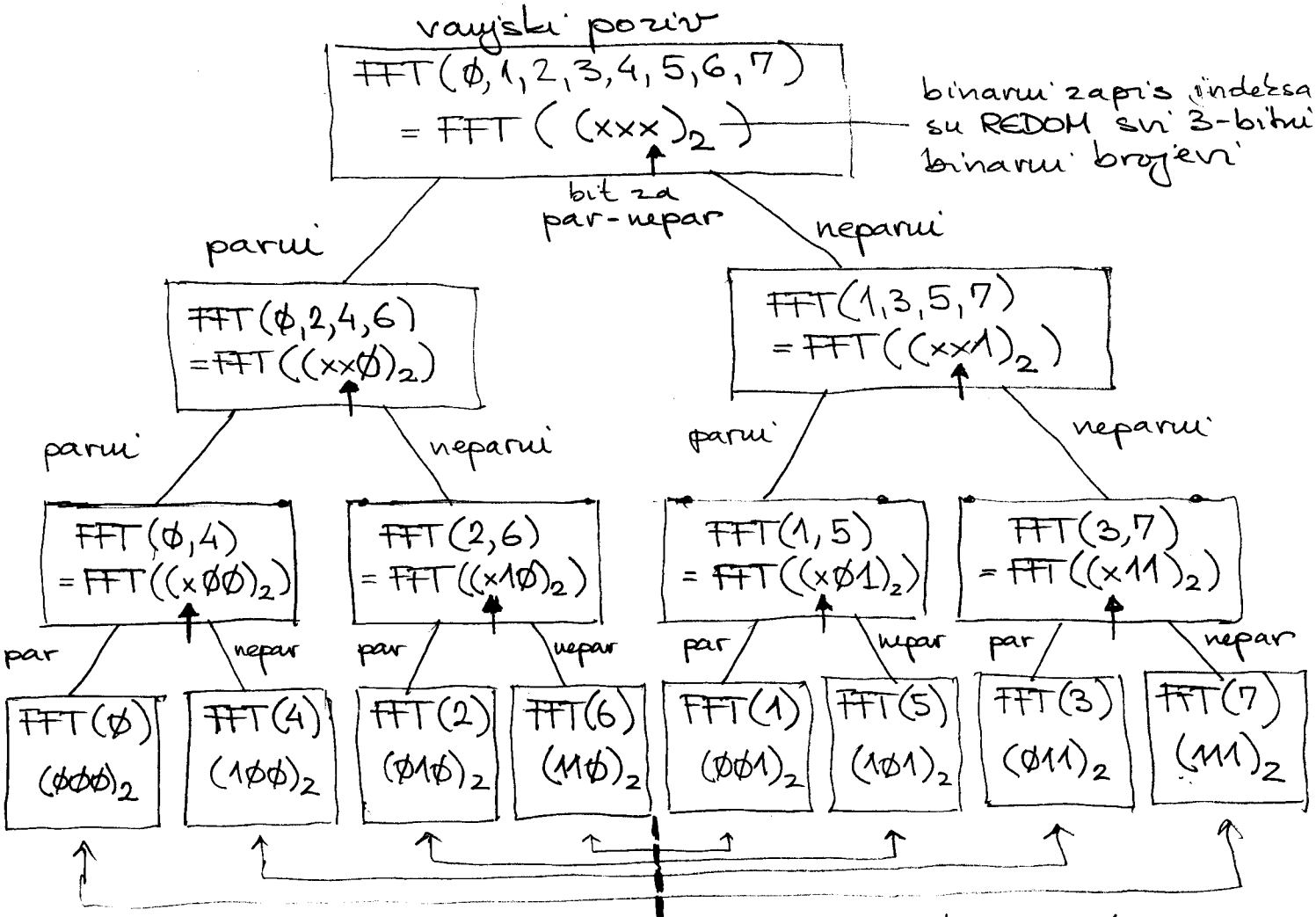
Pogledajmo sad kako izgleda stablo koje generiramo i obilazimo u rekurzivnoj FFT algoritmu za DFT_n.

Ključna operacija je rastav ulaznog vektora a na podvektore $a^{[0]}$, $a^{[1]}$ koji sadrže komponente parnih, odnosno, neparnih indeksa iz a.

Pamost indeksa je odredena zadnjim bitom u njegovom binarnom prikazu, pa je zgodno indekse prikazivati i kao m-bitne brojeve u bazi 2.

Nacrtajmo stablo poziva za vanjski poziv s $n=8=2^3$ tj. za računanje $DFT_8((a_0,a_1,a_2,a_3,a_4,a_5,a_6,a_7)^T)$

U čvorovima stabla pišemo, radi jednostavnosti, samo indekse polaznih komponenti vektora a, koje dolaze na ulaz u FFT rekurziji.



NE DIRATI

Uočimo binarne zapise na dnu - oni su simetrični $0 \Leftrightarrow 1$ na istom mjestu, oko centralne osi!

Na istoj udaljenosti lijevo i desno od te osi, binarni zapisi su komplementarni - imaju sve bitove obratne (mjesto, po mjesto $0 \Leftrightarrow 1$).

Ovo stablo prikazuje samo rekurzivni dio DFT_n algoritma. Cijeli rekurzivni algoritam radi ovako

- obidi lijevo i desno (parno i neparno) podstablo
- zatim, u čvoru izračunaj kombinaciju ta dva podstabla ($2 \times DFT_{pola} \rightarrow DFT_{cijeli}$)

Dakle, u svakom čvoru je još "sakriveno" ona petlja s dva rekurzivnog algoritma. Drugim riječima, rekurzivni DFT_n odgovara obilasku ovog stabla u post poretku (lijevo dijete, desno dijete, čvor).

- Uočimo da je zapravo svejedno da li prvo obilazimo lijevo ili desno podstablo, bitno je samo da je čvor na kraju. Tj. prvo djeca, pa čvor (roditelj te djece).

- Objasnimo još što znače oznake u binarnom zapisu indeksa koji ulaze u FFT.

  $(xxx)_2$ u konjenu znači da su indeksi parametara od FFT svi 3-bitni binarni brojevi (svaki x može biti 0 ili 1) - dakle svih 8 takvih brojeva i to rastuće poredani (kao brojevi) ili leksikografski rastuće (kao 3-bitni binarni brojevi) Dakle, $0, 1, 2, \ldots, 7$.

  - svaki ulazak u parno ili neparno podstablo fiksira prvi nefiksirani tj. slobodni bit x na 0 ili 1, i to sa stražnje strane pokaza (od znamenke jedinica prema napnjeol: $\leftarrow$).
    Dakle, lijevo-parno dijete konjenu ima indekse čiji 3-bitni prikaz je $(xx0)_2$ - svi parni!
    Analogno, desno-neparno dijete ima indekse čiji 3-bitni prikaz je $(xx1)_2$ - svi neparni.

- Dakle, pri spuštanju od vrha prema dnu, tj. od konjenu prema listovima, fiksiranje bitova ide sa stražnje strane prema napnjeol:

  prvi slobodni x $\rightarrow$ 0 u lijevo-parno dijete
  $\rightarrow$ 1 u desno-neparno dijete

- Što znači blok x-ova sprijeda, ispred fiksnih bitova? (fiksiranih)

  Tu treba redom - leksikografski rastuće, pometati sve binarne zapise u kojima se svaki pojedini x može zamijeniti s Ø ili 1, tj. supstituirati redom $\emptyset\emptyset\ldots\emptyset, \emptyset\emptyset\ldots1, \ldots, 11\ldots1$

  (binarni prikazi brojeva od Ø do $2^{(\text{broj x-ova})}-1$)

- Naravno, to vrijedi u svakom čvoru:
  broj x-ova $\Leftrightarrow$ broj parametara je $2^{\text{broj x-ova}}$.

- Na samom dnu, svi bitovi su fiksirani $\Leftrightarrow$ zovemo FFT s jedinim jedinim parametrom, tj, preciznije, zovemo FFT na vektoru duljine 1 - na jednom elementu polaznog vektora.

  Taj dio posla je trivijalan - svodi se na golo kopiranje vrijednosti. Jedino je pitanje odakle-kamo? Otkud to?!

- U naše stablo smo ugradili sve elemente rekurzivnog algoritma osim jednog!

  Što imamo:

  2 rekurzivna poziva → 2 djeteta

  kombinacija $2\times DFT_{pola} \mapsto DFT_{cijeli}$ → petlja u čvoru.

  Što fali?

  Ono kopiranje vizova prije rekurzivnih poziva, koje smo elegantno izbjegli u stablu, jer eksplicitno pišemo parametre!

  No, općenito, takav ulazni parametar je zapravo <u>lokalna</u> kopija na <u>stacku</u>, kod izvršavanja rekurzije!

Vrlo zgodno bi bilo ako možemo izbjeći to gomilanje vizova na stacku i sve raditi na <u>jednom globalnom vektoru</u>.

Naime, ako želimo iterativni algoritam, onda:
- ili moramo simulirati stack za rekurziju
- ili moramo raditi na jednom dijelu
  (taj bi bio globalan u rekurziji!)

Iz ovog promatranja rada rekurzivnog FFT algoritma dobivamo sljedeće zaključke za konstrukciju iterativnog FFT algoritma.

1. Iz "djeca prije čvora-roditelja" $\Rightarrow$ (stvarno je $\Leftarrow$)
   mogli bismo to realizirati tako da bude
   "sva djeca prije (svih) čvorova-roditelja"
   [Sad je jasno zašto stvarno vrijedi $\Leftarrow$, a ne $\Rightarrow$].
   Naravno, to nije nužno, ali je zgodno napraviti baš takvu organizaciju posla.

- Stablo obrađujemo po slojevima-nivoima iste dubine/visine, i to od dna prema vrhu ($\uparrow$).

Drugim riječima, prvo obradimo sve listove - tj. naotemo sve DFT_1, pa onda sve čvorove iznad njih (svi DFT_2), i tako redom, do korijena, gdje na kraju naotemo traženi DFT_n, $n = 2^m$.

- Ovime dobivamo vanjsku petlju iterativnog algoritma koja prolazi sve slojeve odozdo prema gore:

```
for s := 0 to m do
    obradi sloj s;
```

Varijabla s = stage = stadij = sloj.
Brojanje od 0 do m odgovara visini od dna (listovi su na visini 0). Možemo odmah uočiti da s odgovara i broju x-ova, tj. broju slobodnih bitova na tom sloju.

Svi čvorovi na sloju s imaju s slobodnih bitova, tj. pripadni FFT na ulazu ima niz duljine

$$2^s, \quad s = 0, \ldots, m.$$

- Osim toga, listovi ($s = 0$) su posebno jednostavni. Pripadni DFT_1 u čvoru nema aritmetičkih operacija.

Ako sav posao želimo obaviti u jednom polju y, koje će, na kraju, biti i izlazni rezultat, onda na ovom mjestu - u listovima prebacujemo a-ove u y-e. Kasnije, sve radimo na polju y.

Sve operacije u listovima su oblika

$$y_{neki\_y} := DFT_1(a_{neki\_a}) = a_{neki\_a}.$$

Uočimo odmah da element od a ne moramo prebaciti na isto mjesto u vektor y, tj. može biti

$$neki\_y \neq neki\_a.$$

Bitno je da listova ima točno $n = 2^m$ i da svaki ima svoj element od a. I, ovako, dva različita elementa od a ne smiju na isto mjesto u vektor y (jednog od njih - ranijeg - bismo izgubili).

Dalje, obrada svih listova se svodi na operaciju kopiranja vektora a u vektor y, ali ne nužno u istom poretku indeksa, već ih smijemo i permutirati; ako nam to više odgovara za ostatak algoritma!

Neka je p_n ta izabrana permutacija indeksa $0, \ldots, n-1$.

$$j \longmapsto p_n(j), \quad j = 0, \ldots, n-1.$$

Operacija "obradi sloj 0 (listove)" ima onda oblik

```
for j := 0 to n-1 do
    y[pn(j)] := aj;
```

ili

```
y[j] := a[pn^-1(j)]
```

Ovdje je y radno polje, a ne izlazni vektor, pa indeksiranje pišemo Pascalski - u [ ], da ne dođe do zabune.

Kako treba izabrati permutaciju p_n? O tome malo kasnije, kad razradimo ostatak iterativnog algoritma.

Ostatak algoritma (nakon ovog kopiranja s permutiranjem) kojeg moramo još razraditi je obrada svih ostalih slojeva:

```
for s := 1 to m do
    obradi sloj s;
```

2. Čvorovi na istom sloju, očito, ne ovise jedan o drugom $\Rightarrow$ potpuno je svejedno kojim redom obrađujemo čvorove na istom sloju.

Tih čvorova na "visini" s ima točno $n/2^s$, tj. 2^{m-s}. Dakle, sljedeća petlja bi mogla imati ovih 2^{m-s} prolaza za obradu čvor-po-čvor, u nekom zgodnom poretku. (Nećemo još napisati tu petlju!)

Na kraju, obrada u svakom čvoru je kombinacija

$$2 \times DFT_{2^{s-1}}(\ldots) \longmapsto DFT_{2^s}(\ldots)$$

a to se svodi na točno 2^{s-1} "leptir" operacija (treća - zadnja petlja).

Odmah možemo uočiti da ove dvije petlje zajedno imaju

$$2^{m-s} \cdot 2^{s-1} = 2^{m-1} = \frac{n}{2}$$

leptir operacija. Te leptir operacije iz starog stanja polja y proizvode novo stanje tog istog radnog polja y

stari y sadrži 2^{m-s+1} podvektora oblika $DFT_{2^{s-1}}$
tj. 2^{m-s+1} "vektorčića" koji su već izračunati DFT-ovi svih podnizova duljine 2^{s-1}.

novi y sadrži samo 2^{m-s} podvektora koji su DFT-ovi svih podnizova duljine 2^s

Naravno, podnizovi su oni DFT-ovi iz stabla

$$FFT\Big(\underbrace{(x\ldots x}_{s-1}\overbrace{\underset{\text{fix}}{\square}\square\ldots\square}^{\text{duljina } 2^{s-1}})_2\Big) \longrightarrow FFT\Big(\underbrace{(x\ldots x\underset{\text{fix}}{x}}_{s}\overbrace{\square\ldots\square}^{\text{duljina } 2^s})_2\Big).$$

i do

$$\left.\begin{matrix}(x\ldots x\,0\,\square\ldots\square)_2\\ (x\ldots x\,1\,\square\ldots\square)_2\end{matrix}\right\} \longrightarrow (x\ldots x\,x\,\square\ldots\square)_2$$

Naravno, jedino je pitanje gdje unutar polja y se nalaze potrebna 2 vektora koje treba kombinirati i gdje treba smjestiti njihovu kombinaciju.

- Prirodno je da kombinaciju preporučemo preko istih onih mjesta gdje su ranije bili kraći "ulazni" nizovi za kombinaciju.

Čak malo jače od toga!

Osnovna operacija unutar svih petlji je jedna leptir operacija. Nju grubo možemo interpretirati u obliku

$$\text{2 stara } y\text{-a } (\text{2 elementa}) \longmapsto \text{2 nova } y\text{-a } (\text{2 elementa})$$

Naravno, bilo bi zgodno nova 2 elementa prepisati preko stara 2 i tako to $n/2$ puta!

Na taj način osiguravamo da sve operacije možemo obaviti u istom polju y, bez dodatnih polja i to neovisno o realizaciji obrade pojedinog sloja — tj. 2 unutarnje petlje - po čvorovima i leptirima u čvoru mogu u bilo kom poretku (!) ili čak kao jedna petlja za $n/2$ leptira (!).

Dakle, moramo odlučiti gdje će ti elementi biti u polju y. Naravno, ta odluka ima veze i s tim kako je finalni vektor $y = DFT_n(a)$ poredan u polju y - na izlazu.

Prirodno je izabrati da na izlazu y ima očekivani uredeni poredak

$$y_k = y[k], \quad k = 0, \ldots, n-1.$$

U protivnom, još na kraju moramo permutirati y da ga dobijemo u prirodnom poretku.

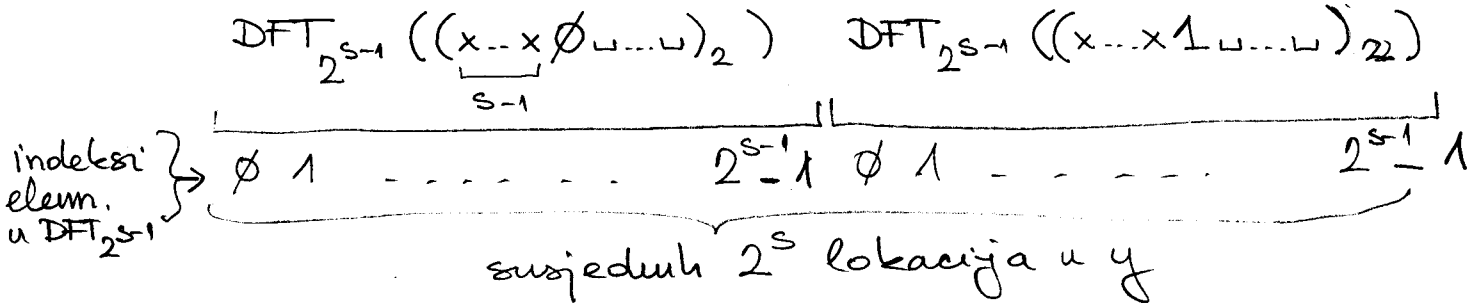
[Baš na ovo ćemo se još vratiti!]

Idemo onda, u skladu s tim, organizirati polje y tako da:

(a) svi DFT(...) uzori imaju prirodan poredak indeksa i to u bloku susjednih lokacija u polju y

(b) prema stablu, DFT-ovi iz lijevih podstabala dolaze ispred onih iz desnih podstabala.

Dalje, parni i neparni koje treba kombinirati dolaze prirodno jedan iza drugog - u bloku



To znači da polje y na svakom sloju s sadrži točno redom vektore iz čvorova na tom sloju u binarnom stablu i to <u>slijeva udesno</u> (→).

U tom slučaju, vanjske dvije petlje iterativnog algoritma imaju oblik

```
for s := 1 to m do
  for l := 0 to 2^{m-s} - 1 do
      p := l · 2^s ; {početni indeks u y}
      kombiniraj 2 DFT-a duljine 2^{s-1} koji
      se nalaze u
          y[p .. p + 2^{s-1} - 1] i
          y[p + 2^{s-1} .. p + 2^s - 1]
      u jedan DFT duljine 2^s u
          y[p .. p + 2^s - 1]
```

Ova kombinacija ima 2^{s-1} leptira. Svi ti leptiri koriste potencije od ω_{2^s}, jer se računa $DFT_{2^s}(\dots)$.

Kad uvrstimo pripadne leptir operacije dobivamo kompletan algoritam za prijanje - od prvog sloja $s=1$ do konjena.

Fali nam još samo permutacija p_n za cijeli algoritam.

Da bi "prijanje" radilo, na dnu - u listovima mora vrijediti isto pravilo o poretku DFT$_1$ u polju y. To znači da u polju y, na početku moraju biti elementi iz vektora a onim redom kojim se pojavljuju u listovima stabla - slijeva udesno.

Za $n=8$, taj poredak je

$$a_0\ a_4\ a_2\ a_6\ a_1\ a_5\ a_3\ a_7.$$

Ako pogledamo binarne zapise indeksa u y i indeksa u a, dobivamo tablicu

indeks u y	indeks u a
$\emptyset = \emptyset\emptyset\emptyset_2$	$\emptyset\emptyset\emptyset_2 = 0$
$1 = \emptyset\emptyset 1_2$	$1\emptyset\emptyset_2 = 4$
$2 = \emptyset 1\emptyset_2$	$\emptyset 1\emptyset_2 = 2$
$3 = \emptyset 11_2$	$11\emptyset_2 = 6$
$4 = 1\emptyset\emptyset_2$	$\emptyset\emptyset 1_2 = 1$
$5 = 1\emptyset 1_2$	$1\emptyset 1_2 = 5$
$6 = 11\emptyset_2$	$\emptyset 11_2 = 2$
$7 = 111_2$	$111_2 = 7$

Možemo zaključiti da se binarni zapis indeksa u a dobiva obratnim poretkom bitova u binarnom zapisu indeksa u y. Naravno, vrijedi i obratno - dva puta obrnemo redoslijed - ništa se ne mijenja.

Dakle, permutacija p_n radi ovako, za $n=2^m$ ako je binarni zapis indeksa j u a (tj. u y)

$$j = (j_{m-1}\, j_{m-2} \cdots j_1\, j_0)_2$$

(j_r su bitovi, $r=0,\dots,m-1$), onda je binarni zapis indeksa $p_n(j)$ u y (tj. u a) oblika

$$p_n(j) = (j_0\, j_1 \cdots j_{m-2}\, j_{m-1})_2.$$

Ova permutacija p_n koja okreće bitove u obratnu poredak zove se bit_reverse (okreni_bitove) permutacija i označava s

$$bit\_rev_n. \qquad (n = 2^m)$$

Pripadni poredak $bit\_rev_n(j)$, $j = 0, \dots, n-1$ zove se obratni bit poredak (bit_reverse order).

Odmah vidimo da je ova sama sebi inverz

$$(bit\_rev_n)^{-1} = bit\_rev_n.$$

- Precizan dokaz da je poredak u listovima upravo obratni bit poredak ide ovako.

Pogledajmo korijen stabla. Svi parni indeksi od a idu lijevo - ispred svih neparnih indeksa od a, koji idu desno - straga.

Dakle, ako je zadnji bit j_0 indeksa j za a jednak $\emptyset$ (j paran), onda pripadni a_j ide u prvu polovinu polja y, a svi indeksi u prvoj polovini imaju vodeći bit jednak $\emptyset$

$$j = (j_{m-1} \cdots j_1 \overset{0}{1})_2 \iff p_n(j) = (\overset{\emptyset}{1} k_{m-2} \cdots k_0)_2$$

Isti princip se rekurzivno ponavlja u svakom čvoru koji nije list, pa indukcijom lako izlazi tvrdnja.

[Precizni zapis koraka indukcije je malo tehnički kompliciran, ali je očito

j_s par/nepar $\quad \gamma_s = \overset{0}{1} \iff p_n(j)$ s $j_s = 0$ ispred (<)
$p_n(j)$ s $j_s = 1$]

- Da zaključimo. Uz dogovorenu organizaciju polja y prvom slogu ($s = \emptyset$), tj. listovima odgovara operacija

Bit_Reverse_Copy (a) ili (a, y)
ili (n, a, y)

koja radi slijedeće:

```
function Bit_Reverse_Copy(a);
    n := length(a);
    for j := 0 to n-1 do
        y[bit_rev_n(j)] := a_j;
    { može i obratno: y[j] := a_bit_rev_n(j). }
    endfor;
    return y;
```

Kako ćemo točno realizirati bit_rev_n, o tome malo kasnije (možemo računati za svaki j, ili spremiti unaprijed u vektor).

Prva varijanta iterativnog FFT algoritma je:

```
function FFT_Base(a); {kompleksni vektor}
    y := Bit_Reverse_Copy(a);   {listovi s = 0}
    n := length(a); m := lg(n);   {n = 2^m}
    for s := 1 to m do   {slojevi}
        q := 2^s;
        ω_q := e^(2πi/q);
        for l := 0 to n/q - 1 do   {ide do 2^(m-s) - 1}
            p := l · q;   {start bloka}
            { ω := 1;   - ako računamo sve potencije }
            for k := 0 to q/2 - 1 do   {ide do 2^(s-1) - 1}
                {leptir operacija}
                t := ω_q^k · y[p + k + q/2];
                    {ili t := ω * y[p+k+q/2]; }
                u := y[p+k];
                y[p+k] := u+t;
                y[p+k+q/2] := u-t;
            { ω := ω * ω_q }
            endfor; {k}
        endfor; {l}
    endfor; {s}
    return y;
```

Odje se lijepo vidi da 2^{m-s} puta računamo iste potencije od $\omega_g = \omega_{2^s}$ (ω_g^k, za $k = 0, -, 2^{s-1}-1$). [Naravno, ako zaista računamo te potencije, a ne čitamo iz tablice].

Međutim, to nije neki problem, jer unutarnje dvije petlje možemo bez problema okrenuti - "transponirati" s idejom da za fiksni k obavimo sve leptire koji trebaju ω_g^k.

Ta poboljšana varijanta algoritma je:

```
function FFT_Iter (a); { kompleksni vektor }
    y := Bit_Reverse_Copy (a); { listom' s=0 }
    n := length(a); m := lg(n); { n = 2^m }
    for s := 1 to m do { slojevi }
      g := 2^s;
      ω_g := e^(2πi/g);
    { ω := 1;  - ako računamo potencije. }
      for k := 0 to g/2 - 1 do { ide do 2^(s-1) - 1 }
        for l := 0 to n/g - 1 do { ide do 2^(m-s) - 1 }
          p := l·g + k; { umjesto leptira ! }
          t := ω_g^k · y[p+g/2]; { ili t := ω * y[p+g/2]; }
              { Okrenem redosljed operacija i eliminiram u! }
          y[p+g/2] := y[p] - t;
          y[p] := y[p] + t;
        endfor; { l }
    { ω := ω * ω_g;  - ako računamo potencije. }
      endfor; { k }
    endfor; { s }
  return y;
```

Daljnje uštede možemo napraviti samo efikasnijim indeksiranjem. Na primjer, zapamtim $g/2 = 2^{s-1}$ u vanjskoj petlji.

Ako mogu pisati korak u petlji (tj. ne mora biti 1 ili -1), onda još možemo skratiti zapis.

Blok u okolini unutarnje dvije petlje bi tada izgledao ovako:

```
q2 := 2^(s-1);   { q/2 }
for k := 0 to q2-1 do
  for p := k to n-1 step q do
    :
```

tako da p prolazi točno vrijednostima

$$k,\ q+k,\ 2q+k,\ \dots,\ (n/q-1)\cdot q+k.$$

Naime, zadnja vrijednost je $n-q+k$, pa ako dodam još jednom q, dobijem $n+k \geq \{k\geq 0\} \geq n > n-1$, tj. strogo više od zadnje dozvoljene vrijednosti $n-1$ za p, a to znači da se petlja prekinula prije toga!

Ponovimo: unutarnje dvije petlje zajedno rade točno $n/2$ leptir operacija. Te operacije su potpuno nezavisne, ako uzmemo da se

$$\omega_{2^s}^k$$

čita iz tablice. Time odmah dobivamo i paralelnu implementaciju za DFT_{2^m} [Slika].

- Možemo još "napasti" dva problema:

(1) - Kako se zgodno čitaju potencije $\omega_{2^s}^k$ iz unaprijed pripremljene tablice?

Pretpostavimo da smo unaprijed izračunali vektor ili polje omega, duljine n, tako da je

$$\text{omega}[j] = \omega_n^j,\quad j = 0,\dots,n-1$$

(tj. spremili lijepo sve n-te korijene iz jedinice).

Kako ga treba iskoristiti da izbjegnemo bilo kakvo računanje s $\omega_q^k = \omega_{2^s}^{k?}$

[Napomena: složenost računanja polja omega je linearna u n, preciznije = n kompleksnih množenja (= 4n real. množenja + 2n real. zbrajanja)]

Sjetimo se tzv. formule kraćenja

$$\omega_q^k = \omega_{q\cdot d}^{k\cdot d}, \quad \forall d \in \mathbb{N}$$

i uvažimo $q = 2^s$. Želimo $q \cdot d = 2^m = n$, pa treba uzeti

$$d = \frac{n}{q} = 2^{m-s}.$$

Dakle:

$$\omega_q^k = \omega_n^{k\cdot 2^{m-s}} = \text{omega}[k \cdot 2^{m-s}],$$

tj. u naredbi koja računa t iskoristimo omega $[k \cdot 2^{m-s}]$, odnosno omega $[k \cdot n/q]$, jer n/q isto tako koristimo u gornjoj granici unutarnje petlje.

Ovo je bilo lako! I još je cijena linearna u n. (uz dodatno polje duljine n)

(2) – Kako se računa bit-rev_n, odnosno kako realizirati operaciju Bit_Reverse_Copy?

To više nije tako jednostavno!

Naravno, ako znamo n unaprijed (i) pretpostavimo da smo unaprijed izračunali vektor Bit_Rev, duljine n

$$\text{Bit\_Rev}[j] = \text{bit-rev}_n(j), \quad j = 0, \dots, n-1,$$

onda je Bit_Reverse_Copy trivijalan. Samo čitamo potrebne indekse iz polja Bit_Rev i gotovo. Tada čitav Bit_Reverse_Copy ide u vremenu $O(n)$ – tj. opet linearno u n.

Čak ne bismo morali kopirati u novo polje y. Mogli bismo napraviti i operaciju

Bit_Reverse_Perm (a)

koja vši odgovarajuću permutaciju unutar istog polja a, bez dodatnog vektora – sa samo jednom pomoćnom varijablom. I opet je strar linearna u n.

Sasvim je drugačije ako Bit_Reverse_{Copy / Perm} (a) treba realizirati bez unaprijed izračunatog polja Bit_Rev.

Što onda? Uostalom, a kako bi se izračunao vektor Bit_Rev?

Odgovor bitno ovisi o tome koje operacije imamo na raspolaganju!

Ako koristimo samo cjelobrojne aritmetičke operacije, onda nam nema spasa. Stvar traje $O(n \log n)$ vremena ili operacija (sekvencijalno).

Prvi očiti način je tako da svaki j dijelujemo s 2 rastavljamo u bitove, koje onda umnožavanjem u obratnom redu (recimo Hornerom) akumuliramo u $bit\text{-}rev_n(j)$. To traje $c \cdot \lg n$ operacija, jer j ima $\lg n$ bitova. I tako to za svaki j, što je

$c' n \cdot \lg n$ operacija

(ima ponešto mogućnosti za malo uštede, ali ne bitno).

Drugi način je rekurzivan, a nalikuje na naše FFT-stablo. Nažalost, složenost je istog reda veličine kao i ajeli FFT!