Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Kvantum rendező algoritmusok

TDK dolgozat, 2018

Mogyorósi Bálint

Konzulens: Dr. Imre Sándor

Hálózati Rendszerek és Szolgáltatások Tanszék

Tartalom jegyzék

Kivonat

Ahogy egyre csökkennek a számítógépekben alkalmazott alkatrészek méretei egyszer, elérnek egy bizonyos méretet mikor már nem a klasszikus fizika szabályai érvényesek és itt lép a képbe a kvantummechanika. Azokat a számítógépek, amelyek úgynevezett qbiteket használnak kvantumszámítógépeknek nevezzük.

Ezek a számítógépek nagyságrendekkel gyorsabban tudnak problémákat megoldani, exponenciális lépésszámot igénylő problémákat degradálnak polinomiális időjűekké. Ezért ilyen hatékonyak mert egy qbit lehet a 0 és az 1 szuperpozíciójába p valószínűséggel az első, míg 1-p valószínűséggel a második állapotban, így n qbiten 2^n állapotot tudunk előállítani egyszerre.

Példaként Grover algoritmusa rendezetlen adatbázisban $\sqrt{N}$ lépésben tud rákeresni egy megadott elemre.

A dolgozatomban általam kitalált kvantumos rendező algoritmusokat fogok bemutatni majd összehasonlítani klasszikus társaikkal.

Abstract

As the size of parts used in computers decreases once they reach a certain size when the rules of classical physics are no longer valid and quantum mechanics is introduced here. Computers that use so-called qbits are called quantum computers.

These computers can solve problems faster than classical ones, degrading problems that require exponential steps to polynomial time. Therefore, they are so efficient because a qbit may be in the superposition of 0 and 1 with p probability of the first, while 1-p probability in the second state, so with n qbit we can produce 2^n state simultaneously.

As an example, Grover's algorithm can search for an item in an unsorted database at $\sqrt{N}$ step.

I will present quantum sorting algorithm in my paper that I have invented and compare them to classics.

1. Bevezetés

Mai világban egyre nagyobb mennyiségű, egyre nagyobb adatot, egyre gyorsabban kell valós időben feldolgoznunk és tárolnunk. Ha egy hatékonyabb rendezéssel tudnánk rendezni, mint az eddigiek akkor egy nagyon hatékony eszköz lenne a kezünkben. Emellett az adatok kiértékelésében is egy igen erős eszköz lenne. Képzeljük el egy olyan bonyolult rendszert, mint a tőzsde. Sok átláthatatlan és korreláló adat, amiknek a kiértékelése bonyolult feladat valós időben, de egy hatékony rendezéssel az adatok elemzése egyszerűsödik, amivel egy jobban kiszámítható és stabilabb rendszerhez jutunk.

Vagy például vegyük a közlekedést, ami egy globális probléma a fejlődő nagyvárosokban, ahol nagy a népsűrűség. Ha közlekedésben résztvevő járművek és forgalomirányító rendszerek adatait könnyebben elemeznénk akkor az egész közlekedést jóval egyszerűbb lenne finom hangolni és ezzel elkerülni a torlódások és forgalmi dugók kialakulását. Esetleg kicsit tovább menve az időben az önvezető autókig, ahol a kereszteződések tele lesznek érzékelőkkel, lézerszkennerekkel és hasonlókkal, amelyek megértik, mi történik, hány ember megy át a zebrán, satöbbi. Majd ezt az információt megosztják a járművel, így az többet tud majd, mint amit a saját érzékelői közölnek vele. Ezen szenzorok adatmennyisége óriási és a rendezés mellett az optimalizálás is egy fontos probléma ha gépekre bízzuk a teljes közlekedést. Egy igen gyors rendezéssel még akár a maximum és minimum keresés is kiváltható lenne.

2. Kvantuminformációelméleti bevezetés

Az 1970es években született meg a kvantuminformatika és kvantum számítógépek ötlete. Az 1980as évekre lefektették az elméleti alapjait és az 1990es években sok jelentős eredmény született, mint például szupersűrű kódolás, ami felére csökkenti az átvivendő adat mennyiségét, Shor algoritmusa ami számokat könnyedén faktorizál vagy Grover algoritmusa mely $\sqrt{N}$ lépésben tud keresni rendezetlen adatbázisban.

2.1. Kvatummechanikai alapok

1900-ban Max Planck bevezette az energia kvantálását, hogy levezessen egy, a feketetest által kisugárzott energia frekvenciafüggését helyesen leíró képletet. 1905-ben Einstein a fotoelektromos hatást azzal a feltételezéssel tudta magyarázni, hogy a fény részecskékből, fotonokból áll. Az ötlet, miszerint a foton energiájának kvantumokból kell összeadódnia, jelentős eredmény volt, mivel megszüntette a lehetőségét annak, hogy a feketetest-sugárzás végtelen nagy energiát vigyen magával, ahhoz képest, ha kizárólag csak hullámokkal kellett volna a jelenséget magyarázni. 1913-ban Bohr megmagyarázta a hidrogénatomszínképvonalait, ismét a kvantumosság feltételezésével, 1913 júliusában megjelent *Az atomok és molekulák szerkezete* c. cikkében. 1924-ben terjesztette elő Louis de Broglie anyaghullám-elméletét, mely szerint minden anyag rendelkezik hullámtulajdonsággal és megfordítva. Ezek az elméletek, bár sikeresek, de szigorúan véve fenomenologikusak (jelenségszintűek) voltak, a kvantálásnak nem létezett precíz bizonyítása. Ezeket együtt a *régi kvantumelmélet* néven ismerik.

A „kvantumfizika" kifejezést először Johnston *Planck Univerzuma a modern fizika fényében* c. könyve alkalmazta.

A modern kvantummechanika 1925-ben született meg, amikor Heisenberg kifejlesztette a mátrixmechanikát, Schrödinger pedig a hullámmechanikát, majd felírta a Schrödinger-egyenletet. Schrödinger utána megmutatta, hogy a két megközelítés egyenértékű. (Valamivel Schrödinger előtt Lánczos Kornél Heisenberg egyenleteiből kiindulva integrálalakban fogalmazta meg a kvantummechanikát). Heisenberg határozatlansági relációját 1927-ben fogalmazta meg, és a koppenhágai értelmezés is nagyjából ekkor öltött formát. Az 1927-es évet követően Paul Dirac egyesítette a kvantummechanikát a speciális relativitáselmélettel, felfedezve az elektron Dirac-egyenletét. Ő volt az első abban is,

hogy operátorelméletet használt, és bevezette a nagy hatású braket-jelölést, amit 1930-as híres könyvében tett közzé. Ugyanebben az időben Neumann János lefektette a kvantummechanika precíz matematikai alapjait, mint a Hilbert-terek lineáris operátorainak elméletét, és ezt közzétette hasonlóképpen híres 1932-es könyvében. Ezek a munkák, mint sok más is az alapító időszakból, azóta is érvényesek és széles körben használják őket[1].

A legegyszerűbb kvantum rendszert reprezentálhatjuk úgy, mint egy két-dimenziós komplex együtthatós vektort, amit a két-dimenziós Hilbert tér felett értelmezünk. Ezeket hívhatjuk qbiteknek, melyek lehetnek elektronok, fotonok vagy bármely mikroszkopikus részecske, ami a kvantummechanika szabályai szerint viselkedik. A **v** vektor oszlopot ezentúl |v⟩-nek jelöljük és 'ket v'-nek ejtjük Dirac-féle jelölésmód szerint.

Egy qbitnek két bázis vektora van a Hilbert tér |0⟩ és |1⟩ vektorai, megfelelően a klasszikus 0 és 1 bit értékeknek. A qbit egy tetszőleges | φ ⟩ állapota nem más mint, egy lineárisan súlyozott kombinációja a bázis vektorainak

$$|\varphi\rangle = a|0\rangle + b|1\rangle = a\begin{bmatrix} 1 \\ 0 \end{bmatrix} + b\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \tag{2.1}$$

ahol az együtthatók $a,b \in C$ a valószínűségi amplitúdók, melyek kielégítik az $|a|^2 + |b|^2 = 1$ normálási feltételt. Ez teljes összhangban van a harmadik posztulátummal, ami azt mondja ki hogy ha megmérünk egy qbitet akkor az $|a|^2$ valószínűséggel lesz a |0⟩ állapotban és $|b|^2$ valószínűséggel lesz a |1⟩ állapotban.

A bázis állapotok ortogonálisak egymásra így részecskékre alkalmazva például a foton bázis vektorai reprezentálhatóak, mint a függőleges és vízszintes polarizációja vagy egy elektronnál a fel és le spinek tudják betölteni ezt a szerepet.

Úgy fogjuk jelölni a sorvektort megfelelően $|\varphi\rangle$ as $\langle\varphi|$ használatával, és 'bra ϕ'-nek ejtjük. A sor és oszlop vektorok közötti reláció a következő: $|\varphi\rangle = (\langle\varphi|)^\dagger$.

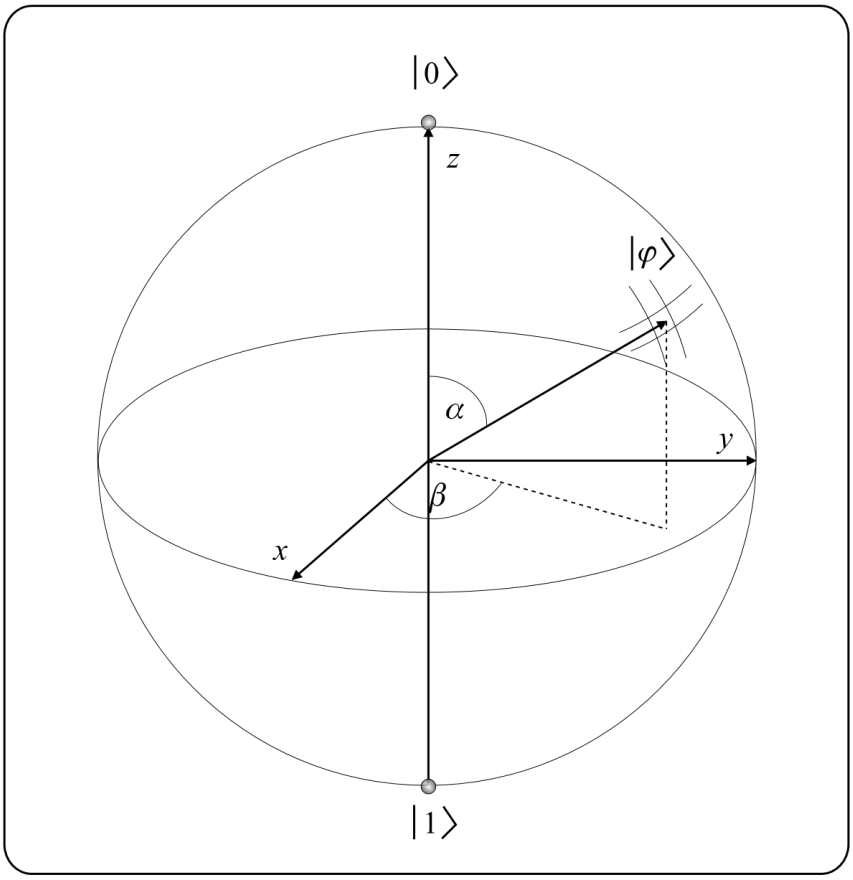
A belső (skalár) szorzata két vektornak |φ⟩ és |ψ⟩ választ ad a furcsa elnevezésekre. Így kell jelölni: ⟨φ|ψ⟩ és úgy kell ejteni, hogy bracket φ és ψ.

Mielőtt áttérnénk komplexebb rendszerekre a qbitekről előtte bemutatunk egy látványosabb jelölési módot mellyel egyetlen qbitet tudunk reprezentálni. Ehhez az előző |φ⟩ állapotot egy másabb formába kell átírnunk

$$|\varphi\rangle = e^{j\gamma}\left[\cos\left(\frac{\alpha}{2}\right)|0\rangle + e^{j\beta}\sin\left(\frac{\alpha}{2}\right)|1\rangle\right], \tag{2.2}$$

ahol $\alpha,\beta,\gamma \in R$. Az $e^{i\gamma}$ szorzót globális fázisnak hívjuk. Mivel az abszolút értéke 1 a globális fázisnak így nem befolyásolja a mérési statisztikát ami a valószínűségi amplitúdók $|\cdot|^2$ függvénye. Ennek következtében a globális fázist elhanyagoljuk amikor analizáljuk a kvantum algoritmusokat és áramköröket.

Míg az első $|\varphi\rangle$ állapot úgy jeleníthető meg egy két-dimenziós Descartes koordináta rendszerben hogy a tengelyek komplexek és így 4 geometriai tengelyen lehetne csak ábrázolni a vektort, ezzel ellentétben a második ábrázolási módnál az $e^{i\gamma}$ szorzót elhagyhatjuk így lehetővé válik az ábrázolása három-dimenziós Descartes koordináta rendszerben. Polár koordináta rendszerben való ábrázoláshoz két valós szöget α,β és a vektor hosszát kell megadni ami triviálisan 1 köszönhetően a normálási feltételnek. Ez a speciális ábrázolási mód Felix Blochnak köszönhető, ezért ezt a koordináta rendszert Bloch gömbnek nevezzük.



1.ábra Bloch-gömb reprezentáció

Most már átválthatjuk a polár koordinátáinkat három-dimenziós Descartes koordinátákká a következő módon

$$|\varphi\rangle = [x, y, z]^T = [\cos(\beta)\sin(\alpha), \sin(\beta)\sin(\alpha), \cos(\alpha)]^T \quad (2.3)$$

Végül kiemelendő, hogy a Bloch gömb a globális fázist is vizualizálja. Ha egy másik γ érték mellet döntünk akkor |φ⟩ máshová fog mutatni a felszínen.

2.2. Elemi kvantum kapuk

Felvázoltuk az analógiát a klasszikus és a kvantumos számítástechnika között, úgy, mint hogy írjuk le az információ-tároló egységeket amiket regisztereknek hívunk. A következő evidens kérdés hogy hogyan tudjuk általánosítani a klasszikus műveletek regisztereken (logikai kapuk és áramkörök) kvantumosan. A második posztulátum tisztán leírja az időbeli evolúcióját a kvantum regiszterek állapotának amiket unitér operátorokkal tudunk modellezni amikre gyakran hivatkozunk mint kvantum kapuk. Miután egy kvantum kapu hivatkozható, mint az elemi kvantum-számítástechnikai eszköz ami fellép mint egy fix unitér operátor a kiválasztott qbiteken egy adott periódus idő alatt. Egy-qbites kvantum kapukat *elemi kvantum kapuknak* hívjuk.

Mielőtt megismerkednénk a széles közben alkalmazott kvantum kapukkal tegyünk egy rövid kerülőt, hogy tisztázzunk egy látszólagos paradoxont. Tudjuk hogy az unitér operátorok reverzibilisek és egy távolság-tartó leképezést implementálnak. Továbbá megtanultuk, hogy a kvantumos leírása a természetnek általánosabb a klasszikusnál, ezért van az hogy van olyan kvantum jelenség amit a klasszikus elmélettel nem lehet megmagyarázni de kvantum mechanikai posztulátumoknak illeszkedniük kell a klasszikus szemléletbe. Azonban vannak olyan klasszikus logikai kapuk, melyek nem reverzibilisek mint az AND, XOR, stb. Például ha egy XOR kapu kimenetén egyes értéket látunk nem lehetünk biztosak benne hogy a bemenet (0,0) vagy (1,1) volt. Ez a megközelítés bonyodalmakhoz vezet, lehetséges hogy a kvantum mechanika nem teljes? Hogy eloszlassuk az ellentmondást segítségül hívjuk az egyik posztulátumot ami kimondja hogy, az időbeli evolúciója bármely zárt rendszernek leírható egy unitér transzformációval ami egyedül kezdeti és végső állapotoktól függ. Az AND és XOR kapuk nem zárt rendszerek.

Most eljött az ideje bemutatni néhány alap, egy-qbites kvantum kaput *U*. Egy teljesen általános egy-qbites állapoton |φ⟩ = a|0⟩ +b|1⟩ mutatom be az elemi kapuk hatását.

Kezdjük a kvantum megfelelőjével a klasszikus inverternek amit *bit-flip kapunak* vagy Pauli-X kapunak hívunk

$$|\psi\rangle = X|\varphi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = b|0\rangle + a|1\rangle \tag{2.4}$$

Könnyű látni, hogy a bit-flip kapu megcseréli a valószínűségi amplitúdóit a bázis vektoroknak. Képzeljük azt az esetet, amikor |φ⟩ = |0⟩ vagy |1⟩ akkor |ψ⟩ az inverze lesz |φ⟩-nek.

A következő kvantum-kapunknak nincs klasszikus megfelelője . A Pauli-Z kapu vagy *fázis-flip kapu* megcseréli a fázisait a bemeneti állapotnak

$$|\psi\rangle = Z|\varphi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a \\ -b \end{bmatrix} = a|0\rangle - b|1\rangle \tag{2.5}$$

Egy egyszerű szabályként észrevehetjük hogy a fázis-flip kapu csak az |1⟩ bázis állapotot szorozza mínusz eggyel.

Hogy a Pauli kapuk halmazát befejezzük ahhoz még egy kaput kell definiálnunk a Pauli-Y kaput a következő féleképpen

$$|\psi\rangle = Y|\varphi\rangle = \begin{bmatrix} 0 & -j \\ j & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} -jb \\ ja \end{bmatrix} = -jb|0\rangle + ja|1\rangle \tag{2.6}$$

ami azt eredményezte, hogy felcserélte és beszorozta j-vel a valószínűségi amplitúdókat.

Egy egyszerű üveglap úgy, viselkedik mint egy elemi kvantum kapu. Ezt a kaput úgy hívjuk hogy *fázis kapu* ami a következőképpen viselkedik

$$|\psi\rangle = P(\alpha)|\varphi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{j\alpha} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} a \\ e^{j\alpha}b \end{bmatrix} = a|0\rangle + e^{j\alpha}b|1\rangle \tag{2.7}$$

Végül az úgy nevezett Hadamard kaput nézzük. Egy általános bemenetre a következő eredményt adja

$$|\psi\rangle = H|\varphi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \begin{bmatrix} \frac{a+b}{\sqrt{2}} \\ \frac{a-b}{\sqrt{2}} \end{bmatrix} = \frac{a+b}{\sqrt{2}}|0\rangle + \frac{a-b}{\sqrt{2}}|1\rangle \tag{2.8}$$

Könnyen belátható, hogy a Hadamard kapu mátrixa nem csak unitér hanem hermetikus is ($H^{\dagger} = H$). Pauli kapuk és a Hadamard kapu között speciális kapcsolat áll fenn, $HXH = Z$, $HYH = -Y$, $HZH = X$.

Az első és a harmadik reláció rávilágít a tényre hogy a bit-flip és a fázis-flip kapuk helyettesíthetik egymást ha van Hadamard kapunk azaz egymással ekvivalensek a Hadamard transzformációt tekintve

Mivel a Hadamard kaput gyakran inicializáljuk klasszikus inputtal ezért számos kvantum-számítástechnikai algoritmusban mi biztosítjuk a megfelelő outputokat

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \qquad (2.9)$$

Mindkét eredmény azt sugallja, hogy ha a Hadamard kaput klasszikus állapotokkal inicializáljuk akkor előáll az összes bázisvektor egyenletes szuperpozíciója, csak az amplitúdók előjeleiben lehet különbség. Ez az egyszerű megfigyelés könnyen általánosítható n-qbites regiszterekre ahol minden önálló qbit egy egy-qbites Hadamard kapura csatlakozik. Elsőnek a $|\varphi\rangle = |000\ldots0\rangle$ állapot kimenetelét mutatjuk be:

$$|\psi\rangle = H^{\otimes n}|\varphi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \qquad (2.10)$$

ahol $H^{\otimes n}$ az n-qbites Hadamard kaput jelöli[2].

3. Klasszikus rendező algoritmusok

3.1. Buborék rendezés

A buborék rendezés egy egyszerű algoritmus, ami ismételten átmegy a rendezendő listán, összehasonlít minden szomszédos párt és megcseréli őket, ha a rossz sorrendben vannak. Addig megy át újra és újra a listán amíg nincs szükség cserére, ami jelzi hogy a listánk rendezett. Az algoritmust, ami egy összehasonlító algoritmus a legkisebb vagy legnagyobb elemről kapta a nevét amik mint egy buborék felmegy a lista tetejére. Bár az algoritmus egyszerű cserébe lassú és a gyakorlatban nem túl hatásos. A buborék rendezés akkor használható ha a bemenetek nagyrésze már rendezett és csak néhány elem nincs a helyén de azok is közel vannak a megfelelő pozíciójukhoz.

A buborék rendezésnek a legrosszabb és az átlagos komplexitása is $O(n^2)$ ahol n az elemek száma a rendezendő listában. A legtöbb gyakorlati rendező algoritmus lényegesen jobb komplexitási paraméterekkel rendelkezik gyakran O(n log n).

Az egyetlen jelentős előnye a buborék rendezésnek még a gyorsrendezéssel szemben is hogy megvan a képessége arra, hogy hatékonyan jelezze hogy a listánk már rendezett, mely tulajdonsága be van építve az algoritmusba. Amikor a lista rendezett, a buborék rendezés komplexitása csupán O(n). Ellenben a legtöbb algoritmus, még a jobb átlagos-komplexitásúak is végrehajtják az teljes rendezési folyamatot a halmazon és így jóval komplexebbek. Ezenfelül jobban teljesít még az olyan listákon amiknek a nagyrésze már rendezett[3].

3.2. Gyors rendezés

A gyorsrendezés egy ún. nemdeterminisztikus algoritmus, amely a véletlent is felhasználja a működéséhez. A gyakorlatban egy determinizált változatát szokás alkalmazni, arra számítva, hogy a rekordok valamiféle "véletlen" sorrendben vannak, és elhanyagolható az esélye annak, hogy pont olyan sorrendből kiindulva kelljen a rendezést végrehajtani, amely túlságosan sok lépést igényel. Ha a gyorsrendezést véletlen algoritmusnak tekintjük, akkor a várható lépésszámáról tudjuk elmondani, hogy igen versenyképes, ha pedig a determinisztikus változatát tekintjük, akkor bár az néhány inputtal sok lépésben végez, ám az inputok döntő többségén rendkívül gyors.

A gyorsrendezés inputja egy A[1..n] rendezetlen tömb, outputja pedig a rekordok növekvő sorrendjében rendezett tömb. Az algoritmus alapja a PARTÍCIÓ(s) eljárás, ahol s az egyik (véletlenül választott) rekord. A gyakorlatban az s rekordot a tömb első (A[1]) elemének szokás választani. A PARTÍCIÓ(s) eljárás inputja egy X[1..k] tömb és egy benne tárolt s rekord, outputja pedig egy átrendezett tömb, amely három részből áll: tömb elejére, mondjuk az X[1..t] tömbbe gyűjtjük az s-nél kisebb rekordokat, középen, az X[t + 1..l] tömbben találhatók az s-sel egyenlő rekordok, míg az X tömb végén elhelyezkedő X[l + 1..k] tömbben s-nél nagyobb rekordok következnek. Ezt úgy szokás implementálni, hogy elkezdjük kiolvasni az X[1], x[2],…. rekordokat, amíg egy s-nél nem kisebb rekordot találunk, mondjuk X[i]-t. Ugyancsak addig olvassuk az X[k],X[k+1], …. rekordokat, egészen addig, amíg egy s-nél kisebb rekordot találunk, mondjuk X[j]-t. Ekkor kicseréljük X[i]-t és X[j]-t, majd folyatjuk az eljárást, az X[i + 1], X[i + 2], …. rekordok ill. az X[j + 1],X[j + 2],…. rekordok olvasásával. Akkor állunk meg, ha az első sorozatban s-nél nem kisebbet, ill. a második sorozatban s-nél kisebb rekordot találunk, amelyeket ismét felcserélünk, majd folytatjuk az eljárást. Ha a két olvasási sorozat összeér, akkor e kapott rekordtól balra s-nél kisebb rekordok vannak a tömbben, míg attól jobbra az s-nél nem kisebbek találhatók. Ezt követően az s-sel egyenlő rekordokat a tömb második részének elejére mozgatjuk.

A PARTÍCIÓ(s) eljárás segítségével a gyorsrendezés algoritmust a rekurzív QUICKSORT(A[1..n]) eljárással valósítjuk meg a következő módon. Végrehajtunk egy PARTÍCIÓ(s) eljárást az A[1..n] tömbre és egy benne tárolt véletlen s rekordra. A kapott A[1..k],A[k+1..l],A[l+1..n] partíción első tömbjén végrehajtunk egy QUICKSORT(A[1..k]) eljárást, míg a harmadik részen egy QUICKSORT(A[l + 1..n]) eljárást.

Az input tömb n mérete szerinti indukcióval könnyen látható, hogy a QUICKSORT eljárás helyesen működik. Nem triviális, de igazolható, hogy a QUICKSORT eljárás

átlagos lépésszáma az n * log2 n konstanszorosával felülről becsülhető. Az is könnyen látható, hogy a QUICKSORT lépésszáma legrosszabb esetben (amikoris a véletlenül választott s rekord mindig a legkisebb vagy a legnagyobb elem az adott tömbben) az n^2-nek konstansszorosa alkalmas pozitív konstansra.

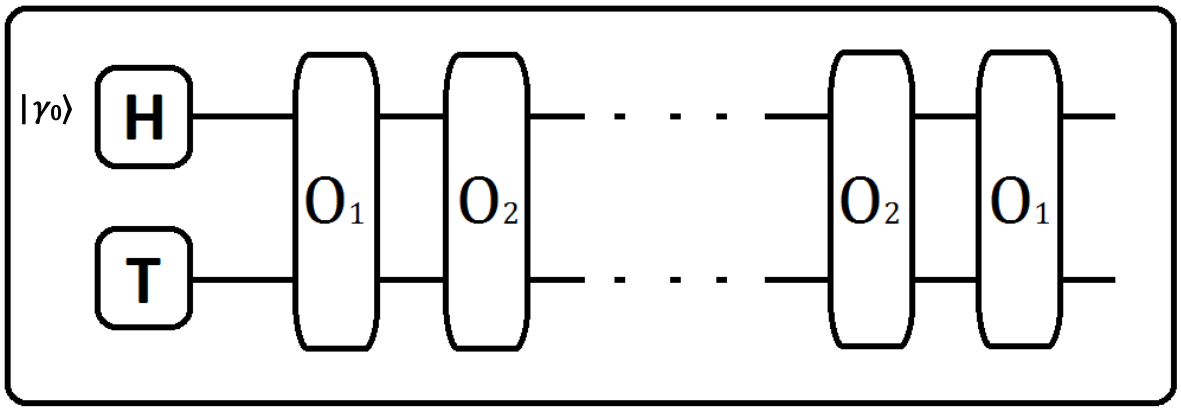
Az eddig ismertetett rendezési algoritmusok mindegyike összehasonlítás-alapú volt, és ezért teljesült rájuk a szakasz elején igazolt információelméleti felső korlát, azaz van olyan pozitív c konstans, hogy n rekord rendezésekor legrosszabb esetben legalább c * n * log2 n összehasonlításra van szükség. Az alábbiakban két kulcsmanipulációs rendezésialgoritmust tekintünk át, amelyek nem összehasonlítás-alapúak lévén akár c*n*log2 n-nél lényegesen kevesebb lépés után is végezhetnek[4].

4. Kvantum rendező algoritmusok

4.1. Imre-Mogyorósi rendezés

A célunk, hogy egy rendezetlen adatbázist(T) quantumosan rendezzünk. Legyen $N = 2^n$ elemünk, ezeket egy adatbázisban tároljuk, vegyünk egy n qbites Hadamard kaput melyre egy $|\gamma_0\rangle = |0\rangle$ állapotot engedünk ekkor a rendszerünk $N = 2^n$ állapotot vesz fel egyforma valószínűségi amplitúdókkal($1/\sqrt{2^n}$), amik a címei lesznek a tárolt adatoknak.

A rendezést úgy valósítjuk meg hogy első lépésben az egymás mellet lévő elemeket, kezdve az elsőtől összehasonlítjuk és ha a kisebb sorszámú nagyobb, mint a másik akkor megcseréljük őket, aztán az előzőlépéshez hasonlóan, de most a második elemtől kezdve hasonlítjuk össze őket az első elemet pedig az utolsóval hasonlítjuk és itt is felcseréljük őket, ha a kisebb sorszámú nagyobb, mint a másik.



2.ábra Kvantum áramkör ami a rendezést megvalósíjta

$$|\gamma_1\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \tag{4.1}$$

O_1: ha $[x_i] > [x_{i+1}]$ $\forall i \in \{ 0, 2, 4, \ldots N\text{-}2 \}$ akkor cserélünk, (4.2)

ha nem volt cserélés akkor egy regiszterbe(F_1) beírunk egy 1-es értéket amivel jelezzük a külvilág számára hogy az adatbázisunk lehet rendezett. Ezután elvégezzük az előbbi összehasonlításokat, de most a második elemtől kezdve, az elsőt és az utolsót egymással.

O_2: ha $[x_i] > [x_{i+1}]$ $\forall i \in \{ 1, 3, 5, \dots N\text{-}3 \}$ VAGY $[x_0] > [x_{N-1}]$, (4.3)

ha volt a feltételeknek megfelelő adatpár akkor az előzőeknek megfelelően cserélünk ha nem akkor egy regiszterbe(F_2) beírunk egy 1-es értéket, amivel jelezzük, hogy az adatbázisunk lehet rendezett.

Beláthatjuk, hogy az algoritmus helyesen működik az első és második lépéseket egymás után ismételgetve. Mivel egy elem mindaddig mozogni fog az adatbázisban amíg az a helyére nem kerül.

Az algoritmus az adatbázist N-1 lépésben tudja rendezni. Képzeljük el az adatbázist mintha egy gráf lenne minden adat egy csúcs és két csúcs között akkor fut él, ha egymás mellettiek az adatbázisban, az első és utolsó elemet is egymás mellettinek tekintjük. Így kapunk egy N csúcsú N élű gráfot, ami jelen esetben egy kör, egy elemet ezen kör mentén mozgathatunk. Legyen a gráfban lévő utak halamaza L, $\max(L) = N\text{-}1$ ez a legnagyobb elem helyére juttatásához szükséges maximális lépésszám, ez akkor fordul elő, ha a legnagyobb elem az első helyen van és mivel az első lépésben ez elsőtől kezdve hasonlítunk így csak egyetlen úton tud a helyére kerülni. Mire a helyére kerül a többi elem is a helyére kerül. Így az algoritmus maximum N-1 lépésben rendezi az adatbázist.

A rendezésnek a legrosszabb és az átlagos komplexitása is $O(N)$ ahol Naz elemek száma a rendezendő listában. A legjobb klasszikus rendező algoritmus minimális komplexitása is csak O(N log N) így a rendezésnek lineáris lépésszámú megoldása nagy előny az eddigiekkel szemben. Az algoritmus nagy előnye, hogy könnyen észrevehetjük ha már a lista rendezett ugyanis ilyenkor nincs olyan elempár amiket megkéne cserélnünk, ezt pedig a $V = F_1 \,\&\, F_2$ logikai függvénnyel vizsgáljuk, $V = 1$ esetén az adatbázis rendezett.

CIKLUS i = 0 TÓL i < N-1 IG{

HA i ≡ 0 (mod 2)

AKKOR{ ∀j ∈ { 0, 2, 4, … N-2 }

HA TOMB[j] > TOMB[j+i]

AKKOR { CSERÉLJÜK AMELYIK ADATPÁRRA IGAZ }

EGYÉBKÉNT {$F_1 = 1$} }

EGYÉBKÉNT { ∀j ∈ { 1, 3, 5, … N-3 }

HA TOMB[j] > TOMB[j+i]

AKKOR { CSERÉLJÜK AMELYIK ADATPÁRRA IGAZ }

EGYÉBKÉNT {$F_2 = 1$}

HA TOMB[0] > TOMB[N-1]

AKKOR { CSERÉLÜNK }

EGYÉBKÉNT {$F_2 = 1$} }

HA F_1&$F_2 = 1$

AKKOR VÉGE }

Nézzünk egy példát az rendezésünkre:

66	66	66	15	15	1	1	1
88	88	15	66	1	15	15	15
15	15	88	1	66	30	30	30
30	30	1	88	30	66	52	52
1	1	30	30	88	52	66	66
70	70	52	52	52	88	70	70
80	52	70	70	70	70	88	80
52	80	80	80	80	80	80	88
1.	2.	3.	4.	5.	6.	7.	

4.2. Mogyorósi Rendezés

4.2.1. Az algoritmus

Most egy másik általam kitalált algoritmust fogok bemutatni, ami bármely adatbázist mindössze log N lépésben képes rendezni. Ahogy az előző algoritmusban $N = 2^n$ elemünk van, amiket egy adatbázisban(T) tárolunk, vegyünk egy n qbites Hadamard kaput melyre egy $|\phi_0\rangle = |0\rangle$ állapotot engedünk ekkor a rendszerünk $N = 2^n$ állapotot vesz fel egyforma valószínűségi amplitúdókkal($1/\sqrt{2^n}$), amikkel az adatok indexelését fogjuk megvalósítani.

Ez az algoritmusunk valamivel komplikáltabb lesz, mint az előző, és nem determinisztikus, vagyis a véletlent is kihasználja a működése során. Első lépésben egy viszonyítási elemet kell választanunk amihez képest az adatbázisunkat rendezni akarjuk. Ez az a pont, ahol úgymond kihasználjuk a véletlent mivel nem ismerjük az adatbázisban lévő elemeink értékét. Ahhoz, hogy optimális legyen a futásidő a mediánt kéne választanunk ugyanis ez mindig felezné az adatbázisunkat, de mivel igen kicsi annak az esélye, hogy az algoritmus elszálljon ami azt jelentené hogy minden lépésben a lista rendezetlen részének a legnagyobb vagy legkisebb elemét válasszuk ki ezért mindegy hogy melyiket válasszuk. A választásnál figyelembe vesszük az algoritmus egyik fontos tulajdonságát, amit lentebb közlünk egyelőre éljünk azzal a választással, hogy a viszonyítási elemünk a $\lfloor N/2 \rfloor$. lista elem. Az elem indexének az értékét beírjuk egy regiszterbe(P).

Következő lépésben a viszonyítási adathoz képest meg kell adnunk minden elemre, hogy nagyobb nála az adott elem vagy sem. Ezek az adatokat megint egy másik regiszterben(S) fogjuk tárolni. $|S\rangle = |s_0 s_1 \dots s_{(N-1)}\rangle$, ahol

$$s_i = \begin{cases} 1, \text{ ha } [x_i] > [x_p] \\ 0, \text{ ha } [x_i] \leq [x_p] \end{cases}$$

,ahol $\forall i \in \{1, 2, 3, \dots N-1\}$. A harmadik lépésben pedig a viszonyítási elemünkhöz($[x_p]$) képesti rendezést hajtjuk végre azt szeretnénk elérni, hogy a nála nagyobb elemek előtte a nála kisebb-egyenlők pedig mögötte legyenek. Ezt két lépésben fogjuk végrehajtani kihasználva a kvantumpárhuzamosságot. Azért van két lépésre szükségünk mert előfordulhat, hogy a viszonyítási elemünk nagyság szerinti helyén egy nála nagyobb elem van és ilyenkor nem teljesülne az előírt kritérium miszerint a nála nagyobb elemek előtte helyezkedjenek el. Első mozzanatként az $[x_p]$ elemünket rakjuk a

helyére, mivel cseréltünk így a P regiszterben tárol cím és az S-ben tárolt értékek is változnak:

$$a := P \text{ , } b := S^T S$$

$$P' = b$$

S': s_a-t és s_b-t felcseréljük

Ebben a lépésben már csak az a dolgunk, hogy a viszonyítási elemünkhöz képest rendezzük a többi elemet és egy n-bites regiszterbe(B) beírunk egy egyest a P' helyre, B kezdetben csupa 0-val volt inicializálva.

Az előzőben definiált algoritmusunk tökéletesen alkalmas arra, hogy vizsgáljuk az adatbázisunk rendezett. Ez két mozzanatból fog állni elsőnek

ha $[x_i] > [x_{i+1}]$ $\forall i \in \{ 0, 2, 4, \dots N-2 \}$, akkor cserélünk csak úgy mint az előző rendezésnél, ha nem volt cserélés akkor egy egybitesregiszterbe(F_1) beírunk egy 1-es értéket amivel jelezzük a külvilág számára hogy az adatbázisunk lehet rendezett. Ezután elvégezzük az előbbi összehasonlításokat, de most a második elemtől kezdve, az elsőt és az utolsót egymással:

$$[x_i] > [x_{i+1}] \; \forall i \in \{ 1, 3, 5, \dots N-3 \} \text{ vagy } [x_0] > [x_{N-1}]$$

ha volt a feltételeknek megfelelő adatpár akkor az előzőeknek megfelelően cserélünk, ha nem akkor egy egybitesregiszterbe(F_2) beírunk egy 1-es értéket, amivel jelezzük, hogy az adatbázisunk lehet rendezett.

Ha meg volt ez a lépéssorozat akkor elmondhatjuk, hogy az $[x_p]$ elemünk már a helyén van, ezt egy regiszterben fogjuk jelezni egyes értékkel, ami kezdetben csak nullákkal volt inicializálva. Ennek a segítségével nyomon tudjuk követni, hogy hány részre is van már osztva a viszonyítási elemek által az adatbázisunk.

Az algoritmus eddigi lépéseit ismételjük viszont most már külön kell végrehajtanunk a viszonyítás elemünk által kétrészre osztott részre, de a kvantum párhuzamosságnak köszönhetően nem kell külön-külön megcsinálunk, hanem képesek vagyunk egyszerre végrehajtani. Viszont mielőtt folytatnánk az algoritmust meg kell vizsgálnunk az $V = (F_1 \,\&\, F_2)$ logikai függvényt ha $V = 1$, akkor az adatbázisunk rendezett és leállíthatjuk a rendezést $V = 0$ esetén folytatnunk kell de F_1 és F_2 értékét is 0-ra kell állítani elkerülve a nemkívánatos működést.

$P[0] = \lfloor N/2 \rfloor - 1$

$B[i] = 0\ \forall i \in \{ 1, 2, 3, \ldots N-1 \}$

CIKLUS $i = 0$ TÓL P DARABSZÁMÁIG

    $S[i][j] > P[i], \forall j \in \{ 1, 2, 3, \ldots N-1 \}$
    CSERE TÖMB[P[i]] ÉS TÖMB[||S[i]||][1]
    $a[i] = P[i]$
    $b[i] = ||S[i]||$
    $P[i] = b[i]$
    $B[i] = P[i]$
    CSERE S[i][a] ÉS S[i][b]
    TÖMB[P[i]] KÉPESTI RENDEZÉS

CIKLUS $i = 0$ TÓL $i < 2$ {

    HA $i \equiv 0 \pmod 2$
    AKKOR{
        HA TÖMB[j] > TÖMB[j+1], $\forall j \in \{ 0, 2, 4, \ldots N-2 \}$
        AKKOR { CSERÉLJÜK AMELYIK ADATPÁRRA IGAZ }
        EGYÉBKÉNT $\{F_1 = 1\}$ }
    EGYÉBKÉNT { $\forall j \in \{ 1, 3, 5, \ldots N-3 \}$
        HA TÖMB[j] > TÖMB[j+1]
        AKKOR { CSERÉLJÜK AMELYIK ADATPÁRRA IGAZ }
        EGYÉBKÉNT $\{F_2 = 1\}$
        HA TÖMB[0] > TOMB[N-1]
        AKKOR { CSERÉLÜNK }
        EGYÉBKÉNT $\{F_2 = 1\}$ }
    HA $F_1 \& F_2 = 1$
    AKKOR VÉGE -
    EGYÉBKÉNT {
        CIKLUS $i = 0${
            CIKLUS $y = 0$ TÓL N-1 IG{
                HA $B[y] = 1$
                AKKOR{}
            $P[i] = (x + y) / 2$
        UGORJ HARMADIK SOR} }

[1] ||S|| = S-ben lévő egyesek száma

Egy mintapélda a rendezésre:

4	4	4	2	2	1	1	1	1	1	1	1	1
2	2	2	4	1	2	2	2	2	2	2	2	2
8	8	3	1	4	4	4	3	3	3	3	3	3
5	7	1	3	3	3	3	4	4	4	4	4	4
7	5	5	5	5	5	5	5	5	5	5	5	5
3	3	8	8	6	6	6	6	6	6	6	6	6
6	6	6	6	8	7	7	7	7	7	7	7	7
1	1	7	7	7	8	8	8	8	8	8	8	8
	1.lépés				2.lépés				3.lépés			

$p_1=3$ $p_1' = s_1 * s_1 = 4$ $B=|00001000\rangle$

$s_1=|11000101\rangle$ $s_1'=|11000101\rangle$

$F_1 = 0$ $F_2 = 0$

$p_2^1=1$ $p_2^{1'} = s_2^1 * s_2^1 = 0$

$p_2^2=6$ $p_2^{2'} = s_2^2 * s_2^2 = 7$ $B=|10001001\rangle$

$s_2^1=|00000000\rangle$ $s_2^{1'}=|00000000\rangle$

$s_2^2=|11111101\rangle$ $s_2^{2'}=|11111110\rangle$

$F_1 = 1$ $F_2 = 0$

$p_3^1=2$ $p_3^{1'} = s_3^1 * s_3^1 = 2$

$p_3^2=5$ $p_3^{2'} = s_3^2 * s_3^2 = 5$ $B=|10101101\rangle$

$s_3^1=|11000000\rangle$ $s_3^{1'}=|11000000\rangle$

$s_3^2=|11111000\rangle$ $s_3^{2'}=|11111000\rangle$

$F_1 = 1$ $F_2 = 1$, F_1 & $F_2 = 1$ vagyis az adatbázisunk rendezett.

4.2.2. Komplexitás vizsgálat

Tegyük fel, hogy az algoritmus minden lépésnél a mediánt választja viszonyítási elemnek ekkor minden lépésben feleződik az adatbázis és az algoritmusunk pontosan log N lépéssel véget ér. Most éljünk azzal a választással hogy mindig egy olyan elem a viszonyítási adat ami az adott résznek csak az ezredénél nagyobb így már 1000 * log N lépés kell a rendezéshez és minden lépés hat allépésből áll így a teljes lépésszám 6000 * log N $<$ c * log N, elmondhatjuk hogy bármely kis hányadát is rendezzük az adatbázisnak a futásidő felülbecsülhető c * log N-nel, így ez lesz az átlagos futási idő. Egy eset kivételt képez mikor a viszonyítási adatunkhoz képest nincs nagyobb vagy kisebb elem ekkor mindig csak egy elemet tesz a helyére és így elszáll az algoritmus a legrosszabb eseti lépésszáma N. Annak érdekében hogy minél kisebb legyen az esély arra hogy elszálljon, ezért van a viszonyítási elem szerinti rendezés után két plusz lépés. Elegendő lenne nézni hogy van e nagyobb elem és ennek eredményét beírni a megfelelő regiszterekbe de ha nem csak figyeljük hanem cseréljük is az elemeket elősegítjük hogy a szélső elemek ne az középső részeken legyenek ami javítja az esélyeinket.

5. Rendezés további alkalmazása

Mérnöki probléma megoldások során általában egy optimum meghatározása a fő feladatunk. Az optimum egy szélső érték maximumot vagy minimumot keresünk a feladat típusától függően.

Klasszikusan maximumot, illetve minimumot N lépésben tudtunk keresni. Kvantumos világban ez mindössze $O(\log^3(\sqrt{N})*\log(\Delta G))$[2] futásidővel teljesíthető, ahol ΔG az adatbázisban tárolt adatok intervallumhossza.

Az előbb vázolt algoritmussal ezt a lépésszámot tovább csökkenthetjük log N-re, ugyanis log N lépésben rendezzük az adathalmazunkat és így pontosan tudjuk, hogy hol a maximum és a minimum ráadásul elég egyszer lefuttatnunk az algoritmust, hogy megkapjuk a szélsőértékeket. A minimum adat $[x_0]$ a maximumé $[x_{N-1}]$.

6. Algoritmusok szimulációja

Minden szimuláció 1000 random generált adatbázisra futtatam le és az átlagokat és szórásokat ezekből az adatokból számoltam.

5.1. Imre-Mogyorósi rendezés szimulációja

Első adatsorban 1024 elemre futatott rendezés adatai a másodikban pedig 64 elemre, a random generált számok 0-10000ig kerültek ki.

Átlag	Szórás
993,74	0,3535
56,02	0,1253

Megfigyelhetjük, hogy ha az adatbázis mérete jóval kisebb mint az intervallum ahonnan sorsoljuk az adatokat akkor az algoritmusnak jobb átlagos futási ideje lesz.

5.2. Mogyorósi rendezés szimulációja

A rendezésnél fen áll az az opciónk is hogy amikor a viszonyítási elemünkhöz képesti rendezést elvégeztük akkor csak belenézzünk és a meghatározott feltételt vizsgáljuk, de nem cserélünk. Szimulációban összehasonlítom, hogy így a lépésszám mennyiben különbözik és hogy indokolja a cserét. Első adatsorban 1024 elemre futatott rendezés adatai a másodikban pedig 64 elemre, a számok 0-10000ig kerültek ki.

Cserével		Csere nélkül	
Átlag	Szórás	Átlag	Szórás
10,67	0,3535	13,89	0,4427
6,15	0,1253	9,9	0,2788

Első adatsorban azt az esetet láthatjuk amikor nagy az elemszám, másodikban kisebb. Jól látszik, hogy amikor cserélünk az átlag lépésszám és a szórás is kisebb ezzel igazolva a cserélések optimalizáló hatását.

7. Összegzés

Az algoritmus sok potenciált rejt magában bár még javítható lenne a legrosszabb eseti futás idő, hogy ne szálljon el az algoritmus. Ha el tudnánk érni valamilyen módon, hogy mielőtt az elemet kiválasztanánk egy átlagot számolni és az átlaghoz legközelebb eső eredményt választani így elkerülve a szélső értékek választását ami a futásidő javulását eredményezné. A legideálisabb az lenne ha minden lépésnél megtudnánk határozni a mediánt így az algoritmusunk legrosszabb és átlagos futás ideje is O(log N) lenne feltéve hogy konstans lépésben megtudnánk határozni a mediánt.

Irodalom jegyzék

[1] Wikipédia Kvantummechanika: https://hu.wikipedia.org/wiki/Kvantummechanika

[2] Quantum Computing and Communications: An Engineering Approach

Dr. Imre Sándor és Balázs Ferenc

[3] https://en.wikipedia.org/wiki/Bubble_sort

[4] http://www.cs.bme.hu/~fleiner/jegyzet/NESZ.pdf

Dr. Fleiner Tamás