# Kvantumáramkör-szimulációs rendszer gyorsításának vizsgálata

Kabódi László
konzulens: Dr. Friedl Katalin

## Tartalomjegyzék

**1. Bevezető** **2**

**2. A kvantumalgoritmusok áttekintése** **2**
2.1. A qubit . . . . . 2
2.2. A kvantumszámítás lépései . . . . . 4

**3. Grover kereső algoritmusa** **4**

**4. A QuIDD rendszer** **6**

**5. A Grover-féle keresés lépésszáma QuIDD használatával** **9**
5.1. Egy jó elem esetén . . . . . 9
5.2. Több jó elem esetén . . . . . 12

**6. További lehetőségek** **13**

**Hivatkozások** **13**

# 1. Bevezető

A kvantumszámítógépről nagy nyilvánosság előtt először Feynman beszélt [2]. Ebben az előadásában a fizika számítógépes szimulációjáról volt szó, és kifejtette, hogy klasszikus számítógépekkel nem lehet a kvantumfizikát hatékonyan szimulálni. Ezért felvetette, hogy ha lehetne olyan számítógépet építeni, ami a kvantumfizika elvei alapján működik, azon lehetséges lenne hatékonyan szimulálni a kvantumfizikai jelenségeket. Ám ekkor még nem figyeltek fel a témakörre. A nagy áttörésre egészen 1994-ig kellett várni, amikor Shor publikálta azóta híressé vált algoritmusát a számok faktorizálására. Ennek fontossága abban áll, hogy tetszőleges számot polinom időben tud faktorizálni. Ha létezne elég nagy kapacitású kvantumszámítógép, ezzel feltörhetővé válna az interneten legelterjedtebb aszimmetrikus titkosítás, az RSA. Ez után fellendült a kvantumalgoritmusok kutatása, annak reményében, hogy valamikor lesz működő kvantumszámítógép.

Erre ugyan még várni kell, de addig is szükséges a kvantumalgoritmusokat ellenőrizni, ezért különböző szimulációs módszereket találtak ki. Ez, mint Feynman is gondolta, nem kivitelezhető hatékonyan. De megfelelő adatszerkezeteket használva az exponenciális futásidő bizonyos algoritmusok esetén polinomiálisra csökkenthető. Egy ilyen adatszerkezet a Viamontes, Markov és Hayes által publikált QuIDD [6], ami a mátrixok struktúrájára alapozva tömörít, és magán a tömörített adaton tudja végrehajtani a szükséges műveleteket.

A dolgozat ennek az adatszerkezetnek vizsgálja a potenciális lépésszámát Grover keresőalgoritmusán. Ehhez először bemutatom a kvantumszámítás alapjait, majd Grover keresőalgoritmusának működését. Ez után a QuIDD adatszerkezet működését írom le nagy vonalakban [6] alapján. Majd az 5. fejezetben részletesen ismertetem észrevételeimet, hogy a már bemutatott keresőalgoritmus lépésszáma hogy függ az algoritmus lépéseinek végrehajtási sorrendjétől.

# 2. A kvantumalgoritmusok áttekintése

## 2.1. A qubit

Kvantumszámítógépnek olyan rendszert nevezünk, amiben az információt nem klasszikus bitek, hanem úgynevezett qubitek, azaz kvantum bitek tárolják. A qubitek különlegessége a hagyományos bitekkel szemben, hogy nem csak két állapotban lehetnek, hanem a két állapot tetszőleges szuperpozíciójában. Azaz nem csak 1 vagy 0 értéket vehetnek fel, hanem például 0.25 valószínűséggel 0 és 0.75 valószínűséggel 1. Egy másik érdekes tulajdonságuk a *kvantum összefonódás* jelensége. Ez egy olyan érdekes állapot, amikor két qubit egy rendszerként viselkedik, azaz az egyiken végzett változtatás pillanatszerűen végrehajtódik a másikon is, a két qubit távolságától függetlenül. Pontosabban akkor nevezünk egy állapotot *összefonódottnak*, ha nem áll elő két kisebb állapot tenzorszorzataként. Ilyen állapot például $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Ha pedig előállítható kisebb állapotokból egy állapot, azt *szeparábilisnek* nevezzük.

Mivel a qubit egyik meghatározó tulajdonsága, hogy két állapot szuperpozíciójában található, ezért jelölésére a szokásos 1 és 0 nem igazán alkalmas. Épp ezért a fizikusok által kedvelt bra-ket jelölésrendszert szokás alkalmazni, amit Paul Dirac vezetett be 1939-ben. Ennek lényege, hogy az oszlopvektorokat egy ket jelöli: $|x\rangle$. Ekkor egy $A$ operátor hatása

az $x$ vektorra $A|x\rangle$-ként írható fel. A sorvektorok jele a bra: $\langle y|$. Nagy előnye a jelölésnek, hogy a belső szorzat intuitíven adódik, ugyanis $\langle y|$ és $|x\rangle$ belső szorzata $\langle y|\,|x\rangle = \langle y|x\rangle$, a bra-ket.

A *szuperpozíció* azt jelenti, hogy egy qubit két bázisvektor, a nulla bázisvektor és az egy bázisvektor lineáris kombinációja. Ezek szokásos jelölése $|0\rangle$ és $|1\rangle$. Általános esetben a lineáris kombinációban komplex együtthatók vannak, de ez ritkán szükséges, tehát nyugodtan lehet valós számokként gondolni rájuk. A szokásos jelöléssel így egy qubit felírható $\alpha|0\rangle + \beta|1\rangle$ alakban. Látható, hogy egy qubit tárolása klasszikus számítógépeken a két együttható tárolásának felelne meg, ami legegyszerűbben (valós együtthatókat feltételezve) két float típusú számmal lehetséges.

De egy qubit nem igazán elég komolyabb algoritmusokhoz. Több qubitet jelölhetünk úgy, hogy a ket vektort egymás mögé írjuk, azaz például $|0\rangle|0\rangle$. Ami kicsit bonyodalmas, és a sok ket miatt gyorsan áttekinthetetlen lesz, ezért szokás a $|00\rangle$ jelölést is alkalmazni. Látható, hogy egy $n$ bites rendszerben ez a jelölés az $n$ bites bináris számokat fogja tartalmazni a ket belsejében. Hogy ne kelljen folyton $n$ darab egyest vagy nullást leírni, szokás a bináris szám tízes számrendszerbeli alakját használni, azaz $|0101\rangle = |5\rangle$. Ezek után könnyen belátható, hogy egy $n$ bites rendszer a következő alakban írható fel: $\alpha_0|0\rangle + \alpha_1|1\rangle + \ldots + \alpha_{2^n-1}|2^n-1\rangle$. Tehát $2^n$ darab együttható szükséges hozzá. És pont ebben rejlik a kvantumrendszerek szimulálásának nehézsége, legalábbis klasszikus számítógépeken. Egy $n$ bites kvantum rendszerhez ugyanis $2^n$ együtthatót kell eltárolni, tehát a rendszer méretével exponenciálisan növekszik az erőforrásigény klasszikus számítógép esetén.

Egy $n$ bites kvantum rendszer tehát láthatólag exponenciális információt tartalmaz. Ám ez az információ jelenlegi tudásunk szerint nem nyerhető ki belőle. Ugyanis a qubit állapotát nem tudjuk csak úgy nézegetni, mint a klasszikus bitét. Ahhoz, hogy egy qubit állapotát megvizsgálhassuk, egy *mérést* kell rajta végeznünk. Ez azonban destruktív hatással van rá, ugyanis összeomlik a hullámfüggvénye, és a két állapot szuperpozíciójából az egyik állapotba kerül. Az $\alpha|0\rangle + \beta|1\rangle$ qubit $|\alpha|^2$ valószínűséggel a $|0\rangle$ állapotban lesz, $|\beta|^2$ valószínűséggel pedig a $|1\rangle$ állapotban. Ehhez az kell, hogy $|\alpha|^2 + |\beta|^2 = 1$ teljesüljön. Valós esetben az abszolútérték elhagyható, és mivel a továbbiakban az együtthatók valósak lesznek, ezt meg is tesszük. A mérésről megfogalmazottak általában is elmondhatóak, az $\alpha_0|0\rangle + \alpha_1|1\rangle + \ldots + \alpha_{2^n-1}|2^n-1\rangle$ rendszer a mérés után $\alpha_0^2$ valószínűséggel a $|0\rangle$, $\alpha_1^2$ valószínűséggel a $|1\rangle$ állapotban lesz, és így tovább, minden állapotra. Természetesen itt is fenn áll, hogy $\alpha_0^2 + \alpha_1^2 + \ldots + \alpha_{2^n}^2 = 1$.

Mérni azonban nem csak a teljes rendszert lehet, hanem annak egy részét is. Ekkor a mért qubitek kerülnek ki a szuperpozícióból, a maradék állapota megmarad. Tehát például egy $\alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$ rendszernek ha megmérjük az első qubitjét, akkor $\alpha_0^2 + \alpha_1^2$ valószínűséggel $|0\rangle\left(\frac{\alpha_0}{\sqrt{\alpha_0^2+\alpha_1^2}}|0\rangle + \frac{\alpha_1}{\sqrt{\alpha_0^2+\alpha_1^2}}|1\rangle\right)$ állapotba kerülünk, $\alpha_2^2 + \alpha_3^2$ valószínűséggel pedig az $|1\rangle\left(\frac{\alpha_2}{\sqrt{\alpha_2^2+\alpha_3^2}}|0\rangle + \frac{\alpha_3}{\sqrt{\alpha_2^2+\alpha_3^2}}|1\rangle\right)$ állapotba.

Az összefonódott állapot a méréssel is igen érdekes kapcsolatban van. Vegyük a következő rendszert: $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Ez egy úgynevezett EPR pár, ami az Einstein, Podolski, Rosen hármasról kapta a nevét. Ha itt az egyik qubiten elvégzünk egy mérést, akkor a másik qubit állapota is az lesz, amit az elsőn mértünk. Még akkor is, ha a két qubit egymástól tetszőlegesen távol van, sőt, a második qubit azonnal az elsőn mért állapotba kerül.

## 2.2. A kvantumszámítás lépései

A qubitek nem sokat érnek, ha nem tudunk rajtuk különböző műveleteket, transzformációkat végezni. Erre az unitér mátrixokat használhatjuk. Egy mátrixot unitérnek nevezünk, ha az inverze megegyezik a transzponált konjugáltjával, azaz az adjungáltjával.

Ezek a mátrixok természetesen invertálhatóak, tehát qubiteken csak invertálható műveleteket végezhetünk. És mivel ezek a mátrixok négyzetesek, ezért $n$ qubitből csak $n$ qubitet képezhetünk.

A legfontosabb ilyen operátor a Walsh mátrix, Hadamard mátrix vagy Hadamard-Walsh mátrix néven ismert. Ez 2×2-es esetben a következőképp néz ki: $H_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Ennek hatása: $H_1 |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ és $H_1 |1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Továbbá $H_1(H_1 |0\rangle) = |0\rangle$, valamint $H_1(H_1 |1\rangle) = |1\rangle$. Itt érdemes megjegyezni, hogy szokásosan $H_n$ az $n$ qubiten ható Hadamard mátrixot jelöli, melynek mérete $2^n \times 2^n$.

Egy $k$ dimenziós operátorból $k^2$ dimenziósat tudunk csinálni, ha vesszük az önmagával vett tenzorszorzatát. Így $H_4 = H_2 \otimes H_2$. Hasonló módon egy $k$ és egy $\ell$ dimenziós operátor tenzorszorzatából egy $k \cdot \ell$ dimenziósat kapunk.

Egy másik hasznos transzformáció a vezérelt nem, azaz a cnot. Legyen ennek a mátrixa $M_{cnot}$. Ez egy két qubites rendszeren végez el egy transzformációt. A lényege az, hogy az első bit vezérli, hogy a második negálva legyen-e, vagy sem. Tehát $M_{cnot} |00\rangle = |00\rangle, M_{cnot} |01\rangle = |01\rangle, M_{cnot} |10\rangle = |11\rangle, M_{cnot} |11\rangle = |10\rangle$. Ennek is felírhatjuk a mátrixát, ami a következő alakot ölti:

$$M_{cnot} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

# 3. Grover kereső algoritmusa

Az első híres, és nagy gyorsulást elérő kvantum algoritmus Shor faktorizációs algoritmusa volt [5]. Azonban ennek az algoritmusnak a felhasználhatósága erősen korlátozott. Ilyen szempontból sokkal általánosabb Lov Grover 1996-os keresőalgoritmusa [3]. Ugyanis a keresés nagyon sok problémában előfordul, mint részfeladat. Ráadásul az algoritmus által használt amplitúdó amplifikáció egy más problémákhoz is igen jól használható módszert adott. Az alábbiakban ezt a keresést fogom ismertetni. [4]

Legyen adott egy

$$f_y(x) = \begin{cases} 1 & \text{, ha } x = y \\ 0 & \text{, különben} \end{cases}$$

függvény, aminek segítségével $y$-t szeretnénk meghatározni. Ehhez először definiáljunk két kvantum operátort.

Legyen az első $R_n$, ami $n$ qubiten hat. Legyen $R_n |0\rangle = -|0\rangle$, és $R_n |x\rangle = |x\rangle$, ha $x \neq 0$. A másik pedig legyen $V_f$, ami egy előjelbe kódolja egy $f(x)$ függvény értékét. Tehát $V_f |x\rangle = (-1)^{f(x)} |x\rangle$.

Ezeket felhasználva elő is állíthatjuk a szükséges operátort: $G_n = -H_n R_n H_n V_f$, ahol $H_n$ az $n$ qubiten ható Hadamard operátor. Érdemes megfigyelni, hogy $H_n R_n H_n$ felír-

ható $I - 2P$ alakban, ahol $I$ a $2^n \times 2^n$-es egységmátrix, $P$ pedig egy $2^n \times 2^n$-es vetítési mátrix, aminek minden eleme $\frac{1}{2^n}$. Ez abba az egydimenziós altérbe vetít, melyet a $\psi = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$ vektor generál.

A $(-H_n R_n H_n)$ transzformációt az átlag körüli tükrözésnek is szokták hívni. Ugyanis vegyünk egy tetszőleges $\sum c_x |x\rangle$ állapott. Jelölje $A = \frac{1}{2^n} \sum c_x$ az átlagos amplitúdót. Ekkor $\sum c_x |x\rangle = A \sum |x\rangle + \sum (c_x - A) |x\rangle$ az eredeti állapotunk felbontása két merőleges altérbe eső vektorra.

Tehát $P \sum c_x |x\rangle = A \sum |x\rangle$, ahonnan $(-H_n R_n H_n) \sum c_x |x\rangle = (2P - I) \sum c_x |x\rangle = \sum (2A - c_x) |x\rangle$. Tehát minden amplitúdót a -1-szeresére változtat, majd az átlag kétszereséhez ad. Ennek a haszna a keresés során az, hogy a keresett elem amplitúdóját, azaz a mérésnél az előfordulásának valószínűségét, megnöveli, míg a többi elemét lecsökkenti.

Ezt a következő módon teszi. Kezdetben vegyük a $\frac{1}{\sqrt{2^n}} \sum |x\rangle$ állapotot, aminek egyenletes az amplitúdóeloszlása. $V_f$ ekkor a keresett elemnek megfordítja az előjelét. Így $A = \frac{1}{2^n}((2^n - 1)\frac{1}{\sqrt{2^n}} - \frac{1}{\sqrt{2^n}}) = \frac{1}{\sqrt{2^n}}(1 - \frac{2}{2^n})$, ami nagyjából megegyezik a kezdeti, $\frac{1}{\sqrt{2^n}}$-nel. Viszont ha ezen végrehajtjuk a $(-H_n R_n H_n)$ transzformációt, a keresett elem amplitúdója $3\frac{1}{\sqrt{2^n}}$ lesz, míg a többié marad nagyjából $\frac{1}{\sqrt{2^n}}$.

Most vizsgáljuk meg, mi van akkor, ha $f(x)$ több helyen is 1 értéket vesz fel. Legyen $T \in \{0,1\}^n$ a megoldások halmaza, és $F = \{0,1\}^n \setminus T$ pedig a nem-megoldásoké, $|T| = k$, és $|F| = 2^n - k$. Ekkor $(-H_n R_n H_n V_f)$ $r$-szeri alkalmazása után a következő állapotban leszünk: $t_r \sum_{x \in T} |x\rangle + f_r \sum_{x \in F} |x\rangle$. Ha most megint alkalmazzuk $V_f$-et: $-t_r \sum_{x \in T} |x\rangle + f_r \sum_{x \in F} |x\rangle$, ahol az átlagos amplitúdó $A = \frac{1}{2^n}(-tk + f(2^n - k))$. Ha most $(-H_n R_n H_n)$-et alkalmazzuk, $t_{r+1} \sum_{x \in T} |x\rangle + f_{r+1} \sum_{x \in F} |x\rangle$ állapotot kapjuk, ahol $t_{r+1} = 2A - t_r = (1 - \frac{2k}{2^n}) t_r + (2 - \frac{2k}{2^n}) f_r$, valamint $f_{r+1} = 2A - f_r = -\frac{2k}{2^n} t_r + (1 - \frac{2k}{2^n}) f_r$.

Mivel egyenletes amplitúdóeloszlásból indulunk ki, ezért $t_0 = r_0 = \frac{1}{\sqrt{2^n}}$ kezdeti feltételeket használjuk. Továbbá tudjuk, hogy $k t_r^2 + (2^n - k) f_r^2 = 1$-nek minden $r$-re teljesülnie kell. Vagyis $(t_r, f_r)$-et egy ellipszis pontjai határozzák meg. Azaz felírhatjuk, hogy

$$\begin{cases} t_r = \frac{1}{\sqrt{k}} \sin \theta_r \\ f_r = \frac{1}{\sqrt{2^n - k}} \cos \theta_r \end{cases}$$

valamilyen $\theta_r$-re. Ezt behelyettesítve a rekurzióba a következő összefüggéseket kapjuk:

$$\begin{cases} \sin \theta_{r+1} = (1 - \frac{2k}{2^n}) \sin \theta_r + \frac{2}{2^n} \sqrt{2^n - k} \sqrt{k} \cos \theta_r \\ \cos \theta_{r+1} = -\frac{2}{2^n} \sqrt{2^n - k} \sqrt{k} \sin \theta_r + (1 - \frac{2k}{2^n}) \cos \theta_r \end{cases}$$

Mivel $k \leq n$, ezért $1 - \frac{2k}{2^n} \in [-1, 1]$. Ezért választhatun k$\omega \in [0, \pi]$-t, hogy $\cos \omega = 1 - \frac{2k}{2^n}$. Ekkor $\sin \omega = \frac{2}{2^n} \sqrt{2^n - k} \sqrt{k}$. Ezzel az előző egyenletek egy kellemesebb alakba írhatóak:

$$\begin{cases} \sin \theta_{r+1} = \sin(\theta_r + \omega) \\ \cos \theta_{r+1} = \cos(\theta_r + \omega) \end{cases}$$

A kezdeti feltételekből megkapjuk, hogy $\sin^2 \theta_0 = \frac{k}{2^n}$, ami $\theta_0$ meghatározásához használható. Így a rekurzió megoldása a következő:

$$\begin{cases} t_r = \frac{1}{\sqrt{k}} \sin((2r+1)\theta_0) \\ f_r = \frac{1}{\sqrt{2^n - k}} \cos((2r+1)\theta_0) \end{cases}$$

Mivel $k$ megoldásunk van, ezért egy megoldás megtalálásának valószínűsége $kt_r^2 = \sin^2\theta_r = \sin^2((2r+1)\theta_0)$. Ehhez szeretnénk olyan, minél kisebb $r$-er találni, hogy a valószínűség maximális legyen, tehát $(2r+1)\theta_0$ legyen a legközelebb $\frac{\pi}{2}$-höz. Ez $\theta_0 \approx \sqrt{\frac{k}{2^n}}$-nél lesz. Ekkor annak a valószínűsége, hogy $\left\lfloor \frac{\pi}{4\theta_0} \right\rfloor$ iteráció után megtaláljunk egy megoldást, legalább $\frac{1}{4}$.

**Az algoritmus összefoglalva**:

1. Ha $k > \frac{3}{4}2^n$, akkor válasszunk egy véletlenszerű $y$-t, az jó megoldás lesz.
2. Különben számoljuk ki $r = \left\lfloor \frac{\pi}{4\theta_0} \right\rfloor$-t, ahol $\theta_0 \in [0, \pi/3]$ és $\sin^2\theta_0 = \frac{k}{2^n}$-ből számolható.
3. Állítsuk elő a következő állapotot: $\frac{1}{\sqrt{2^n}} \sum |x\rangle$, az Hadamard transzformációval.
4. Alkalmazzuk a $G_n = -H_n R_n H_n V_f$ operátort $r$-szer.
5. Mérjünk, hogy megkapjuk $y$-t.

Ekkor az eredményül kapott $y$ legalább $\frac{1}{4}$ valószínűséggel jó megoldás lesz. Ha ennél nagyobb valószínűséggel szeretnénk a helyes megoldást megkapni, az $r$-t nagyobbra választva a hiba tetszőlegesen kicsire csökkenthető.

# 4. A QuIDD rendszer

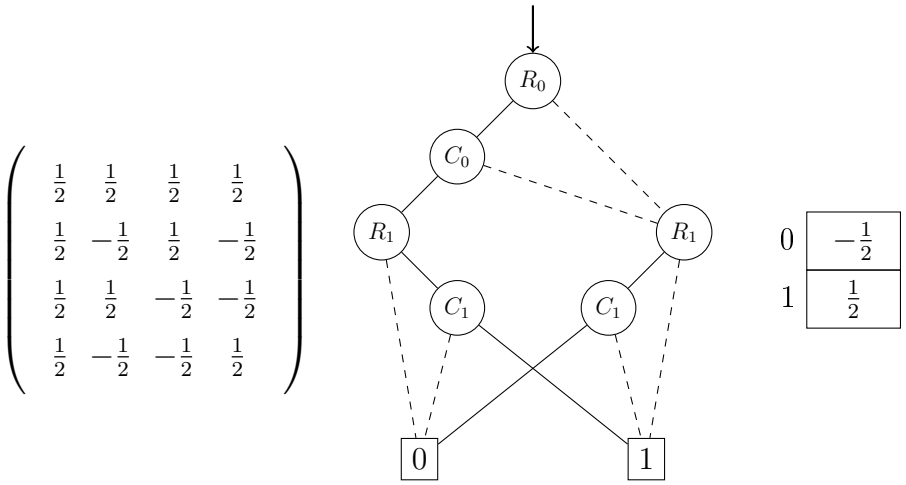
A kvantumszámítógépek klasszikus számítógépen szimulálásához a legkézenfekvőbb módszer az, ha az operátorokat mátrixokkal reprezentáljuk. Ekkor a kvantumszámítás lépései egyszerű mátrixszorzások lesznek. Azonban egy $n$ qubites állapot tárolásához $2^n$ klasszikus bit kell, és az ezen ható operátorok $2^n \times 2^n$-es mátrixok lesznek. Ezek a mátrixok általában mutatnak valami struktúrát, amit kihasználva lehetőség nyílik arra, hogy tömörebb formában tároljuk a mátrixokat, és a számolás is tömörebb lehessen. Erre ad egy adatszerkezetet a kvantuminformációs döntési diagram (Quantum Information Decision Diagram, QuIDD) [6].

Az adatszerkezet alapja egy bináris döntési diagram (Binary Decision Diagram, BDD). Ezt először logikai függvények ábrázolására használták, ahol a gráf csúcsai egy-egy változót jeleztek, és a két gyereke a változó igaz illetve hamis értéke esetén vizsgálandó következő változó volt. A fa levelei a függvény értékeit tárolják olyan változóértékek mellett, amilyenekkel oda eljutottunk. Azonban ez a szerkezet nem tömöríti az adatokat. Ezt a hiányosságát pótolja a csökkentett, rendezett bináris döntési diagram (Reduced Ordered BDD, ROBDD). Ez két egyszerűsítési szabályt vesz a BDD-hez, amik a következők:

1. Nincs olyan $v$ és $v'$ csomópont, hogy a $v$ és a $v'$ gyökerű részgráfok izomorfak.
2. Nincsenek belső csúcsok, melyeknek a két gyereke azonos.

Könnyen belátható, hogy a változók sorrendezése nagy hatással lehet a kialakult fa méretére. Ráadásul az optimális sorrend megtalálása NP-nehéz probléma. A gyakorlatban használt esetek egy részében ezzel a módszerrel a változóknak polinomiális függvénye lesz a fa mérete.

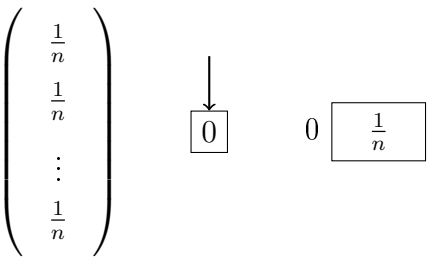
A QuIDD tárolási módszere ezen az ROBDD elrendezésen alapul. Azonban csak a tárolás nem elég ahhoz, hogy a számítások sebessége megnövekedhessen, szükség van ezeken a fákon végrehajtható műveletekre is. Erre való az *Apply* művelet. Ez bejárja mindkét fát, és a megfelelő csomópontok között végrehajtja a megadott műveletet.




1. ábra. $H_2$, a két qubiten ható Hadamard kapu, és QuIDD reprezentációja

Az *Apply* rekurzívan bejárja a két fát, és minden lépésnél megvizsgálja a csomópontot, ahol épp áll. Legyen az egyik fában az aktuálisan vizsgált csomópont $v_f$, a másikban pedig $v_g$. Ekkor ha $v_f$ előbb van a fákhoz használt sorrendezésben, mint $v_g$, az új fába $v_f$ kerül, és a művelet meghívja magát rekurzívan $v_f$ egyik gyerekére és $v_g$-re, valamint $v_f$ másik gyerekére és $v_g$-re. Azaz ha $T(v_f)$ jelöli az egyik gyereket, és $E(v_f)$ a másikat, akkor $Apply(v_f, v_g, op)$ az $Apply(T(v_f), v_g, op)$ és az $Apply(E(v_f), v_g, op)$ függvényeket hívja.

Hogyha a $v_g$ van előrébb, akkor ugyan ez történik, csak épp $v_g$ gyerekeivel. Míg ha a két csomópont azonos változóról szól, az $Apply(T(v_f), T(v_g), op)$ és az $Apply(E(v_f), E(v_g), op)$ függvények indulnak el. A rekurzió akkor ér véget, ha mindkét csomópont egy levél. Ekkor az *Apply* végrehajtja a két levélben tárolt értéken az *op* műveletet. Látható, hogy ennek a műveletnek a lépésszáma $O(ab)$, ahol $a$ az egyik mátrix reprezentációjának a csomópontjainak száma, $b$ pedig a másiké.




2. ábra. Egy egyenletesen elosztott kezdővektor, és QuIDD reprezentációja

A QuIDD is ezt az *Apply* algoritmust használja a műveletek végrehajtásához. Valamint a levelekben nem a konkrét értékeket tárolja, hanem egy mutatót egy tömb megfelelő elemére, ami az értéket tárolja. Továbbá szükséges, hogy ezek az értékek komplex számok legyenek. A változók sorrendje általában olyan, hogy sorok és oszlopok egymást felváltva

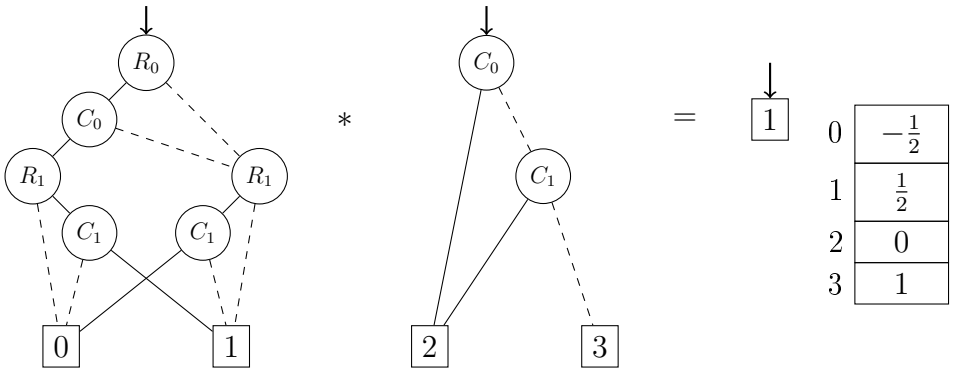
követik, ugyanis a kvantum algoritmusokban használt operátorokat így lehet hatékonyan tárolni. Ugyanis ez a blokkos szerkezeteket helyezi előtérbe, ami igen gyakori a tenzorszorzással előállított mátrixoknál.

A tenzorszorzáshoz először át kell alakítani a sorrendezést. Pontosabban a második mátrix változóit el kell tolni az első változói után. Így az első fa minden levele után fejtődik ki a második fa, ami pont a tenzorszorzásnak felel meg. Ugyanis Így a második mátrix minden elemét megszorozza az első mátrix elemeivel.

A tenzorszorzás akár nagy méretnövekedést is okozhat. Ugyanis egy $n \times n$-es mátrixot egy másik $n \times n$-es mátrixszal szorozva a keletkező mátrix $n^2 \times n^2$-es lesz. Azonban ez bizonyos esetben nem fog bekövetkezni.

**Lemma**: Adott $Q_i$ QuIDD-ek esetén ezek $\otimes_{i=1}^{n} Q_i$ tenzorszorzatának QuIDD-je $|In(Q_i)| + \sum_{i=2}^{n} |In(Q_i)| \left|Term(\otimes_{j=1}^{i-1} Q_j)\right| + |Term(\otimes_{i=1}^{n} Q_i)|$ csomópontot tartalmaz. Itt $In(Q_i)$ a QuIDD belső csúcsait jelenti, $Term(Q_i)$ pedig a leveleket.

Ha a tenzorszorzat végeredményében a levelek, azaz az előforduló különböző elemek száma nem növekszik, akkor az egész reprezentáció mérete $n$-ben lineáris marad. Ezért ha egy operátor előáll kisebb operátorok tenzorszorzataként, és nem tartalmaz sok különböző elemet, annak mérete $O(n)$ lesz.




3. ábra. $H_2 |00\rangle = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$, mátrixszorzás QuIDD formában

A mátrixszorzás végrehajtása felírható skalárszorzások sorozataként is. Azonban a skaláris szorzáshoz nem elég csak simán az *Apply*-t használni, ezért Bahar et al. mátrixszorzó algoritmusát [1] használja a QuIDD. Aminek a legnagyobb előnye, hogy magán a tömörített QuIDD szerkezeten is működik, nem kell visszaállítani belőle az eredeti mátrixot. Ennek az algoritmusnak a lépésszáma $O((ab)^2)$. Valamint, mint az összes ilyen fán végrehajtott algoritmus esetében, a létrehozott új fa mérete is ebben a nagyságrendben van, azaz $O((ab)^2)$. Azonban, ahogy a 3. ábrán is látható, a szorzás után az új mátrix reprezentációjának mérete akár csökkenhet is.

# 5. A Grover-féle keresés lépésszáma QuIDD használatával

## 5.1. Egy jó elem esetén

A QuIDD adatszerkezetet leíró cikkben a polinomiális lépésszám megmutatására Grover híres keresőalgoritmusát használták. Erre egy igen durva, $O(|A|^{16} n^{14})$ felső becslést adtak, ahol $|A|$ a $V_f$ mátrix, azaz az orákulum QuIDD reprezentációja csomópontjainak száma, $n$ pedig az állapotvektor mérete, azaz $2^n$ darab elem között folyik a keresés. A cikkben nem részletezik, hogy hogyan jött ki ez a becslés, ezért itt levezetem.

A mátrixszorzás lépésszáma QuIDD reprezentációval $O(|A|^2 |B|^2)$, ahol $|A|$ illetve $|B|$ a két mátrix reprezentációjának csomópontszáma. Az algoritmus első lépése az állapotvektor inicializálása, azaz $n$ qubit nullába állítása. Ez után a $H_n |x\rangle$ szorzást kell végrehajtani, hogy $|x\rangle$ az összes állapot szuperpozíciójában legyen. Mivel az Hadamard mátrix $O(n)$ csomóponttal ábrázolható, $|x\rangle$ pedig egy csomópont, ennek a lépésszáma $O(n^2)$. Eddig tartott az algoritmus inicializáló része, ami $O(n^2)$ lépés, viszont az eredmény tárolható egy csomópontban, mivel az állapotvektor minden eleme azonos.

Az algoritmus ismétlendő része leírható $(-H_n R_n H_n V_f |x\rangle)$ alakban, ahol $|x\rangle$ az előző lépésben előállított vektor, azaz egy csomóponttal tárolható. $V_f$ mérete legyen $|A| = |V_f|$, hogy a végeredmény a cikkben levővel azonos alakban legyen. Ekkor az első szorzás lépésszáma $O(|A|^2)$. Ez lesz a következő szorzás jobb oldalán levő elem mérete. A $H_n$-nel való szorzás $O((n |A|^2)^2) = O(|A|^4 n^2)$ lépéssel megoldható.

**Állítás**: $R_n$ szintén megadható $O(n)$ csomóponttal.

*Bizonyítás.* $R_n$ egy diagonális mátrix, aminek a bal felső eleme $(-1)$, a főátló többi eleme pedig 1. Könnyen látható, hogy egy azonos méretű egységmátrix kisebb egységmátrixok tenzorszorzataként előállítható. Ráadásul mindegyik QuIDD-nek két levele van. Valamint egy olyan mátrix, ami a jobb felső sarokban $(-2)$-t tartalmaz, a többi helyen pedig 0-t, négy csomóponttal tárolható. Egy a sort, egy az oszlopot kódolja, a további kettő pedig a 0 és a $(-2)$ értékeket tárolja. Majd az egységmátrixhoz hozzáadva ezt a mátrixot megkapjuk $R_n$-et. Az összeadás lépésszáma $O(ab)$, ahol $a$ az egységmátrix csomópontjainak száma, ami $O(n)$, $b$ pedig a $(-2)$-t tartalmazó mátrixé, ami 4. Ugyanis az *Apply*-t kell használni *op* helyére +-t téve. Tehát $R_n$ mérete $O(4n) = O(n)$. □

A kereső algoritmus következő lépése az $R_n$-nel való szorzás, amihez emiatt elég $O((n |A|^4 n^2)^2) = O(|A|^8 n^6)$ lépés.

Végül ismét $H_n$-nel kell szorozni, ami továbbra is $O(n)$ méretű, így a lépésszáma az ismétlődő részben $O((n |A|^8 n^6)^2) = O(|A|^{16} n^{14})$ lesz, ahogy a cikkben is írták.

**Állítás**: A Grover algoritmus egy iterációjának lépésszáma $O(|A|^{16} n^{14})$, azaz polinomiális a qubitek számában.

*Bizonyítás.* Az algoritmus inicializálása tehát $O(1)$ lépéssel megoldható az eredmény struktúrája miatt. Ezután az egy iterációhoz szükséges $(-H_n R_n H_n V_f |x\rangle)$ kiszámítása az előbbieket felhasználva $O(|A|^{16} n^{14})$. □

**Állítás**: Az $|x\rangle$ állapotvektor leírásához szükséges állapotok száma nem változik egy Grover-iteráció után.

*Bizonyítás.* Az algoritmus működése soirán a keresett elemekhez tartozó értéket növeli, a többihez tartozót pedig ennek megfelelően csökkenti minden iteráció. Azonban minden keresetthez tartozó érték megegyezik, és minden nem keresetthez tartozó is megegyezik. □

Az $O(|A|^{16}\, n^{14})$-es becslés nagyon tág, ezért megvizsgálom, hogy lehetne finomítani. Egy kézenfekvő ötlet, amit sok helyen, például a számítógépes grafikában már rég óta széleskörűen használnak, hogy a mátrixszorzásokat elég egyszer, előre elvégezni. Ugyanis a mátrixok szorzása asszociatív, így nem számít, hogy hogyan zárójelezzük őket. Továbbá mivel a keresés folyamán a szükséges operátorok nem változnak, ezt megtehetjük. $H_n$ egy bemenettől független transzformáció, tehát az biztosan nem fog változni. $R_n$ szintén változatlan, ugyanis a $|0\rangle$ állapotot negálja, függetlenül a keresett elemtől. Az egyedüli mátrix, ami függ a konkrét kereséstől, az a $V_f$. De az is csak magától a keresendő elemtől függ, ami a keresés folyamán nem változik, ezért az algoritmus indításakor lekérdezhető az orákulumtól, és a végrehajtás folyamán nem fog változni. Tehát a $H_nR_nH_nV_f$ szorzat előre kiszámolható.

**Állítás**: A $H_nR_nH_n$ mátrix $O(n)$ csomóponttal tárolható.

*Bizonyítás.* Mint azt már korábban láthattuk, $H_nR_nH_n = I - 2P$, azaz egy olyan mátrix, ahol a főátló elemei azonosak, és a főátlón kívüli elemek is megegyeznek. Tehát összesen két féle elem található a mátrixban, méghozzá egy diagonális elrendezésben. A főátlóban levő elemek $1 - \frac{2}{2^n}$, a többi elem pedig $-\frac{2}{2^n}$. Legegyszerűbben ez a mátrix $I - 2P$ alakban állítható elő. Egy $n \times n$-es egységmátrixot elő tudunk állítani tenzorszorzatként, így az $O(n)$ csomóponttal tárolható. A $P$ mátrix előállításához olyan $2\times2$-es mátrixból indulunk ki, aminek minden eleme $\frac{1}{2}$. Ezt önmagával tenzorszorozzuk addig, míg a megfelelő méretű mátrixot nem kapjuk, aminek az elemei pont $\frac{1}{2^n}$ lesznek. Ez a mátrix egy csomóponttal tárolható, ugyanis minden eleme azonos. Két mátrix különbségét az *Apply* algoritmus minusz operátorral való meghívásával számolhatjuk ki, aminek a lépésszáma $O(ab)$, azaz ebben az esetben $O(n \cdot 1)$. Mivel $I - 2P$-t számolunk, kétszer kell a második konstruált mátrixot kivonna az egységmátrixból, ami így $2O(n) = O(n)$ lépés. □

**Állítás**: $V_f$ mátrix eltárolható $O(n)$ csomóponttal.

*Bizonyítás.* $V_f$ egy olyan mátrix, ami csak a főátlóban tartalmaz elemeket. Azon belül is mindenhol 1, kivéve a keresett elem helyén, ugyanis ott $(-1)$. Az egységmátrix továbbra is tárolható $O(n)$ csomóponttal, továbbá egy olyan mátrix, aminek egy eleme $(-2)$, a többi pedig nulla, négy csomóponttal tárolható. Így $V_f$ mérete $O(n \cdot 4) = O(n)$ lesz. □

Ekkor $H_nR_nH_n$-t jobbról megszorozzuk a $V_f$ mátrixszal, ami $O((ab)^2)$ lépés, ami jelen esetben $O((n\cdot n)^2) = O(n^4)$ lépés, ami azt jelenti, hogy $H_nR_nH_nV_f$ tárolásához is elegendő ennyi csomópont.

**Állítás**: $H_nR_nH_nV_f$ eltárolható $O(n)$ csomóponttal.

*Bizonyítás.* A négy mátrix szorzata egy olyan mátrix lesz, ami abban különbözik az $I-2P$ mátrixtól, hogy a keresett elem oszlopa $(-1)$-el meg van szorozva. Tehát ha veszünk egy $X$ mátrixot, amiben a keresett elem oszlopa $\frac{4}{2^n}$, és ezt kivonjuk az $I-2P$-ből, az már majdnem jó. Ekkor ugyanis a főátlóbeli elem még nem lesz megfelelő, de a többi már igen. A főátlónak a keresett elem oszlopában levő elemének $(-1+\frac{2}{2^n})$-nek kell lennie, de így $1-\frac{2}{2^n}-\frac{4}{2^n}$ lesz. Tehát egy olyan mátrixot adunk hozzá, amiben a főátló megfelelő eleme $(-2+\frac{8}{2^n})$, mindenhol máshol nulla. Ez négy csomóponttal leírható. Az $X$ mátrix három csomóponttal adható meg, ugyanis ki kell választani a megfelelő oszlopot, az egy értéket vesz fel, az összes többi elem értéke nulla. Ezekből a mátrixokból $V_f$ előállítható $O(n\cdot 4)$ és $O(n\cdot 3)$ lépéssel, tehát $O(n)$ méretű lesz az eredmény. □

Tehát az iteráció egy lépése egy $O(n)$ méretű mátrixszal való szorzás, ami $O((n\cdot 3)^2)$ lépés, ugyanis az állapotvektor egy kivételével minden eleme megegyezik. A keresett elem helyén álló érték különbözik csak, így egy belső csúcs és két levél elég a tárolásához.

Érdekes lehet megvizsgálni, hogy ennek a szorzatnak az előállítása mekkora számításigényű. Ehhez a $H_nR_nH_nV_f$ szorzat kiszámításának lépésszámát kell vizsgálni. Mindegyik mátrix $O(n)$ méretű, így a művelet lépésszáma $O((((n\cdot n)^2\cdot n)^2\cdot n)^2)=O(n^{22})$. Innen adódik a következő állítás.

**Állítás**: Ha a Grover-féle keresőalgoritmusban az iterációs lépésben a $H_nR_nH_nV_f$ mátrixot előre kiszámoljuk, és csak ezzel szorzunk, QuIDD reprezentáció használata esetén $O(n^{22})$ lépés az inicializálás, valamint $O(n^2)$ lépés az iterációs szakasz. □

Azonban a mátrixszorzás asszociativitása miatt megvizsgálhatunk más felbontást is. Vizsgáljuk a $(H_nR_nH_n)\cdot(V_f)\cdot|x\rangle$ alakot. Ekkor az inicializáló szakaszban csak a $H_nR_nH_n$ szorzatot számoljuk ki. Ez $O(((n\cdot n)^2\cdot n)^2)=O(n^{10})$ lépés. Viszont az algoritmus magjában ekkor két szorzást kell végrehajtani. Először a $V_f|x\rangle$ szorzást kell végrehajtani, ami $O(n^2)$ lépés.

**Állítás**: A $V_f|x\rangle$ szorzat három csomóponttal tárolható.

*Bizonyítás.* $|x\rangle$ állapotvektor három csomóponttal leírható, ugyanis csak a keresett elem helyén álló érték különbözik a többitől. $V_f$ pedig ennek az elemnek fordítja meg az előjelét, vagyis a vektor felépítése nem változik. Ezért továbbra is három csomópont elegendő a tárolásához, amiből egy belső csomópont, kettő pedig levél.. □

**Állítás**: Ha a Grover-féle keresőalgoritmusban az iterációs lépésben a $(H_nR_nH_n)$ mátrixot előre kiszámoljuk, és $(H_nR_nH_n)\cdot V_f|x\rangle$ zárójelezéssel számoljuk, QuIDD reprezentáció használata esetén $O(n^{10})$ lépés az inicializálás, valamint $2\cdot O(n^2)$ lépés az iterációs szakasz.

*Bizonyítás.* Az inicializáló szakaszban most csak a $H_nR_nH_n$ szorzást kell elvégeznünk, ami $O(((n\cdot n)^2\cdot n)^2)=O(n^{10})$ lépéssel elvégezhető.

Az iteráció lépésszáma a két szorzás lépésszámának összege lesz. Az első a $V_f|X\rangle$, ami $O((n\cdot 3)^2$ lépés, a második pedig ennek megszorzása a $H_nR_nH_n$ mátrixszal. Mivel az előző szorzás eredménye három csomóponttal tárolható, a második szorzás lépésszáma szintén $O((n\cdot 3)^2)$ lépés lesz. Ezek összege $2\cdot O(n^2)$ lépés. □

Megnézhetjük azt is, hogy milyen eredményt kapnánk, ha az eredeti becslést finomítanánk a szorzások után kapott struktúra elemzésével, azaz ha a négy mátrixszal egymás után szoroznánk. Azonban ez az út nem célravezető. Ugyanis az Hadamard mátrixot épp azért használjuk, mert, szemléletesen szólva, elkeni az állapotvektort, így az azzal való szorzás csak az $O(n)$ csomópontú becslést használhatnánk. Ugyan a cikkben levő becslésnél ez is jobb lenne, mivel ott csak a polinomiális kapcsolat bizonyítása volt a cél, az előző kettő módszerhez képest nagyon rossz eredményt kapnánk.

Tehát az eredmények egy táblázatban összefoglalva:

| módszer | inicializálás | szorzás |
|---|---|---|
| eredeti | $O(n)$ | $O(\|A\|^{16} n^{14})$ |
| $(H_n R_n H_n V_f) \cdot \|x\rangle$ | $O(n^{22})$ | $O(n^2)$ |
| $(H_n R_n H_n) \cdot (V_f) \cdot \|x\rangle$ | $O(n^{10})$ | $2 \cdot O(n^2)$ |

1. táblázat. Lépésszámok egy keresett elem esetén

Mivel az egész iterációt $R = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor$-szer kell elvégezni, ahol jelen esetben $M = 1$ a keresett elemek száma és $N$ a bemeneti adatok száma, azaz $n = \log(N)$. Ezért az iterációk száma $n$-ben exponenciális. Vagyis az inicializációban levő lépések száma összességében elhanyagolható az egész algoritmushoz képest, sokkal inkább az iterációban résztvevő feladatok lépésszámát éri meg csökkenteni. Emiatt úgy tűnik ideálisabb, ha az egész $H_n R_n H_n V_f$ szorzatot előre kiszámoljuk.

## 5.2. Több jó elem esetén

Abban az esetben, ha több jó érték is található az adathalmazban, a számítások kis mértékben módosulnak. Mivel $H_n R_n H_n$ nem függ a bemenettől, csak $n$-től, ezért amit arról megállapítottunk, nem fog változni. Ellenben $V_f$ az előző konstrukcióval nem állítható elő $O(n)$ méretben, ugyanis akár $\frac{2^n}{2}$ jó elem is lehet. Ezért itt bevezetjük az eredeti cikkben is használt $|A|$ jelölést.

Az első felbontásban, ahol a négy mátrixot szorozzuk össze, $H_n R_n H_n$ továbbra is $O(n)$ méretű, ezt szorozzuk meg az $O(|A|)$ méretű $V_f$ mátrixszal. Ez $O((n\,|A|)^2)$ lépés. Így az iteráció egy lépése is ilyen nagyságrendben lesz. Az inicializálás lépésszáma ebben az esetben pedig $O((((n \cdot n)^2 \cdot n)^2 \cdot |A|)^2) = O(n^{10}\,|A|^2)$ lesz.

A másik módszerre is hasonlóan számolhatjuk a lépésszámokat. Ezeket a következő táblázat tartalmazza.

| módszer | inicializálás | szorzás |
|---|---|---|
| eredeti | $O(n)$ | $O(\|A\|^{16} n^{14})$ |
| $(H_n R_n H_n V_f) \cdot \|x\rangle$ | $O(n^{10}\,\|A\|^2)$ | $O(n^2\,\|A\|^2)$ |
| $(H_n R_n H_n) \cdot (V_f) \cdot \|x\rangle$ | $O(n^4\,\|A\|^2)$ | $O(n^2\,\|A\|^4)$ |

2. táblázat. Lépésszámok több keresett elem esetén

Ebben az esetben is úgy tűnik, hogy az optimális eset az, ha az összes mátrixot összeszorozzuk előre, és csak egy mátrixszal szorozzuk az állapotvektort az iteráció lépéseiben.

# 6. További lehetőségek

Sajnos a programot nem sikerült megszereznem a cikkíróktól, de érdekes lenne megnézni, hogy az implementációban hogy számol a program. Elvégzi-e a mátrixok előre összeszorzását, vagy mindegyik szorzást külön-külön végzi el.

A programban természetesen lehetőség van egy kvantumállapoton mérést végezni. Ezt a cikk alapján egy $M_m$ operátorral való szorzással ($M_m |\psi\rangle$), majd a kapott vektor normálásával végzi el. Itt egy optimalizálási lehetőség lehet az, hogy a normáláshoz a [6] cikkben leírtakkal ellentétben nem számolja ki a $\langle\psi| M_m^H M_m |\psi\rangle$ szorzatot (ahol $M_m^H$ az $M_m$ mátrix adjungáltja, vagy más néven a konjugált transzponáljta), hanem megszámolja a gráf egyszeri bejárásával, hogy melyik csomópont hányszor szerepel, és ez alapján számolja ki a vektor normáját.

# Hivatkozások


[1] Bahar, R. I., Frohm, E. A., Gaona, C. M., Hachtel, G. D., Macii, E., Pardo, A., and Somenzi, F. Algebraic decision diagrams and their applications, 1993.

[2] Feynman, R. E. Simulating physics with computers. *International Journal of Theoretical Physics 21*, 6 (June 1982), 467–488.

[3] Grover, L. K. A fast quantum mechanical algorithm for database search, 1996, arXiv:quant-ph/9605043v3.

[4] Hirvensalo, M. *Quantum Computing*. Natural Computing Series. Springer, 2004.

[5] Shor, P. W. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1994), SFCS '94, IEEE Computer Society, pp. 124–134.

[6] Viamontes, G. F., Markov, I. L., and Hayes, J. P. Improving gate-level simulation of quantum circuits, 2003, arXiv:quant-ph/0309060v2.


A QuIDD adatstruktúrát használó QuIDDPro program honlapja:
http://vlsicad.eecs.umich.edu/Quantum/qp/