جست‌وجوی ترکیبیاتی و الگوریتم تولید کد گریِ لایه‌هایِ میانی *

امید یعقوبی

دانشکده علوم ریاضی، دانشگاه صنعتی شریف

۱۸ فروردین ۱۴۰۰

چکیده

تولید اشیا ترکیبیاتی به روشی الگوریتمیک و با کمترین هزینه محاسباتی همواره مورد توجه محققان علوم کامپیوتر بوده است. در کتاب *The Art of Computer Programming* جلد ۴ از دونالد کنوث [۱] به معرفی برخی از الگوریتمهای کلاسیک در این زمینه پرداخته است. همچنین وی متوجه شد که برای برخی از اشیا با ویژگیهای خاص می توان الگوریتمهای سریع تری معرفی کرد. به طور خاص می توان به حدس معروف *Middle levels conjecture* و ارتباط آن با تولید اشیا ترکیبیاتی اشاره کرد که تمرکز اصلی این ارائه می باشد. این حدس ادعا دارد که برای هر زیرگرافِ $(2n+1)$-cube که توسط زیرمجموعه هایی که دقیقا شامل (n) یا $(n+1)$ عضو از مجموعه مادر می باشند، به شرط $n \geq 1$، القا شده باشد، یک سیکل همیلتونی وجود دارد. امکان تولید همه ی اشیاء $(n+1, n+1)$-combination به شکلی که نمایش باینری هر شی با شی بعدی تنها در یک بیت تفاوت داشته باشد توسط *Havel*، *Buck* و *Wiedmann* در سال ۱۹۸۰ حدس زده شده بود [۲]. این حدس توسط *Mütze* و همکاران در سالهای اخیر اثبات و مورد استفاده قرار گرفته است. در جلسه ی روز چهارشنبه به برخی از یافته های ایشان در مقالات [۳] [۴] برای تولید تمام شی های ترکیبیاتی از یک نوع به شکلی که نمایش باینری هر شی با شی بعدی تنها در یک بیت اختلاف داشته باشد پرداخته شد.

کلمات کلیدی: Gray code,Hamilton Cycle,Hypercube,Middle levels conjecture,Lattice

۱ معرفی

۱.۱ اشیا ترکیبیاتی

ترکیبیات به گفته‌ی کنوث مطالعه‌ی الگوهای متفاوتی است که چند شی گسسته می‌توانند داشته باشند [۱]. در نتیجه، یک شی ترکیبیاتی بازنمایشی از یک الگویِ متمایز است که توسط چند شی گسسته ساخته می‌شود. شاید هیچ چیز مانند مثال روشن کننده نباشد. یک *درخت دودویی*، یک *زیرمجموعه*، یک *جایگشت*، همه و همه شی ترکیبیاتی هستند. مثلا $\{2,3\}$ یک 2-*subset* از مجموعه مادر $\{1,2,3,4\}$ بوده و $(2,3)$ یک 2-*tuple*، گفتنی است که این دو شیِ متمایزند، چراکه در یک الگو ترتیب مهم و در دیگری بی‌اهمیت است. همچنین در اینجا شی $\{1,3\}$ با آن‌که از $\{2,3\}$ متمایز است ولی هر دو نمونه‌ای از یک کلاس ترکیبیاتی با نام 2-*subset* می باشند.

---

*Combinatorial Searching and an algorithm for middle levels Gray codes

۲.۱ بازنمایِ [1] شی ترکیبیاتی

بازنمایی اشیا در کلاس‌های متمایز، به روشنی متفاوت است. برای نمونه ممکن است در بازنمایی یک گراف،سه بازنمایِ *ماتریس مجاورت*، *ماتریس برخورد* و بازنمایِ غیرمستقیمِ مجموعه‌ی دوتایی‌ها مناسب باشد. ولی این بازنمایی‌ها در نمایش یک 3-*subset* دست‌کم در نگاه اول ناکارا به نظر می‌رسد. الگوریتمها و روشهای محاسبه نیز تا حد زیادی به متد استفاده شده در بازنماییِ آن شی‌ها وابسته‌اند.

بازنمایِ مهم در k-combination رشته‌ی باینری [2] است. در این بازنمایی برای هر شی درون مجموعه یک اندیس در نظر گرفته می‌شود. برای نمونه ترتیب *lexicographic* را در نظر داشته باشید. حال روشن است که رشته‌ی 0110 در مجموعه‌ی مادر $\{1,2,3,4\}$ نمایش $\{2,3\}$ که یک 2-subset است می‌باشد. همچنین از آن‌جا که معمولا اشیا در مجموعه‌ی مادر فاصله‌های منظمی از هم ندارند، در برخی الگوریتمها زیرآرایه‌ای شامل k اندیس از عضوهای مجموعه‌ی مادر با اندازه‌ی n، می‌تواند بازنمایِ یک k-combination باشد. برای نمونه اینجا $[1,2]$ نمایشی متفاوت از 0110 می‌باشد.

۳.۱ (s,t)-combination

یک انتخاب tتایی از n شی را ما به عنوان یک t-combination از مجموعه‌ای n عضوی می‌شناسیم. ولی کنوث به تقارنی اشاره دارد که بین انتخاب شده‌ها و انتخاب نشده‌ها مشهود است. گویی دسته‌ی انتخاب نشده‌ها نیز به همان اندازه‌ی انتخاب شده‌ها مهم‌اند. این تقارن همان برابری $\binom{n}{t}$ با $\binom{n}{n-t}$ می‌باشد. چراکه یک شی از کلاس t-combination مجموعه‌ی مادر را به دو پاره‌ی انتخاب شده‌ها و انتخاب نشده‌ها پارتیشن می‌کند. حال از این پس یک t-combination از مجموعه‌ای n عضوی را یک $(n-t,t)$-combination می‌خوانیم.

مثال ۱.۱. *مجموعه‌ی مادر زیر را در نظر بگیرید:*

$$v = \{\diamondsuit_1\clubsuit_1\clubsuit_2\spadesuit_1\heartsuit_1\diamondsuit_2\clubsuit_3\spadesuit_2\}$$

حال q یک 3-combination است، که مجموعه‌ی مادر را دوپاره می‌کند:

$$q = \{\heartsuit_1\spadesuit_2\clubsuit_2\} = 00101001$$
$$\{\underbrace{\{\heartsuit_1\spadesuit_2\clubsuit_2\}}_{Selected=t},\underbrace{\{\diamondsuit_1\clubsuit_1\spadesuit_1\diamondsuit_2\clubsuit_3\}}_{Not\ Selected=s}\}$$

پاره‌ی انتخاب شده‌ها یا t اندازه‌ی 3 و پاره‌ی انتخاب نشده‌ها یا s دارای اندازه‌ی 5 است. در نتیجه q یک $(5,3)$-combination از v است.

[1] Representation
[2] Bitstring

۴.۱ الگوریتمهای ترکیبیاتی

الگوریتمهای ترکیبیاتی [۳] به سه دسته‌ی زیر تقسیم می‌شوند:

1. **Enumeration**
2. **Random generation**
3. **Exhaustive generation**

در دسته‌ی اول، آن‌چه برای ما مهم است تعداد اشیا ترکیبیاتی ممکن است. برای مثال می دانیم که تعداد (s,t)-combination برابر $\binom{t}{s+t}$ می‌باشد.

در دسته‌ی دوم می خواهیم چند شی را به تصادف از میان اشیا یک کلاس ترکیبیاتی انتخاب کنیم برای مثال پرتاب دو سکه، تولید یک شی 2-tuple از مجموعه‌ی مادر $\{H,T\}$ است.

در نهایت در دسته‌ی آخر می‌خواهیم تمام اشیا ترکیبیاتی یک کلاس ترکیبیاتی را دقیقا یک‌بار بسازیم. در اینجا برخلاف نوع اول نه تعداد، بلکه خود این اشیا هستند که مورد اهمیت‌اند. به این الگوریتم‌ها همچنین جست‌وجوی ترکیبیاتی [۴] نیز می‌گویند، چراکه در بیشتر موارد هدف از تولید همه‌ی اشیا از یک کلاس جست‌وجوی یک شی با شرایطی خاص است. مهمترین الگوریتم‌ها در این دسته قرار می‌گیرند. همچنین تمرکز ما نیز در اینجا روی این دسته‌ی مهم از الگوریتم‌ها محدود می‌شود.

۲ الگوریتمهای عمومی

۱.۲ Lexicographic Generation

آنچه به فراوان در الگوریتم‌های ترکیبیاتی [۵] استفاده می‌شود. محاسبه‌ی اشیا بر مبنایِ ترتیب *Lexicographic* است. در اینجا می‌خواهیم یک الگوریتم عمومی برای تولید (s,t)-combination معرفی کنیم که برای تمامی حالات اشیا مورد نیاز را تولید می‌کند. این الگوریتم در [1, p. 358] مورد بررسی قرار گرفته است. در این الگوریتم بازنمایِ یک (s,t)-combination به شکل اندیسهای موجود در شی ترکیبیاتی نسبت به مجموعه‌ی مادر است *(از راست به چپ و شروع از صفر)* که در یک لیست اندیس‌دار *(آرایه)* نگه‌داری می‌شوند. برای نمونه شی $[0,1,2]$ از مجموعه مادر $\{1,2,3,4,5,6\}$ همان شی 000111 یا زیرمجموعه‌ی $\{1,2,3\}$ می‌باشد. که یک سه‌ـانتخاب از این مجموعه شش‌تایی است. در ابتدا به این اشاره کنیم که منظور از ترتیب *Lexicographic* ترتیب روی رشته‌ی باینری [۶] می‌باشد. اگر زین پس قرارداد کنیم که به (s,t)-combination به اختصار (s,t)-comb گوییم. این الگوریتم در ابتدا آرایه *(با نام C)* را با کمترین عدد ممکنه در بازنمایش باینری مقداردهی اولیه [۷] می‌کند. روشن است که برای این‌کار می‌بایست همه‌ی tعدد 1 در سمت راست نمایش بیتی فشرده شوند و به آرامی در هر مرحله به سمت چپ حرکت کنند. *(برای توضیحات بیشتر به صفحه ۳۵۷ مرجع[۱] مراجعه کنید)* در این الگوریتم در مکان $t+1$ از آرایه (از یک شروع کنیم) و مکان $t+2$ از آرایه، به ترتیب دو مقدار n که اندازه‌ی مجموعه‌ی

[3] Combinatorial Algorithms
[4] Combinatorial Searching
[5] Combinatorial Algorithms
[6] Binary String
[7] Initialize

مادر است و 0 به عنوان اشیای نگهبان [۸]، برای تشخیص شرایط مرزی نگهداری می‌شوند.

سپس در هر تکرار حلقه روی j که از 1 شروع می‌شود. در حلقه‌ی داخلی $while$ تا زمانی که شرط $C_j + 1 = C_{j+1}$ برقرار است C_j با مقدار $j - 1$ سِت می‌شود. اگر این شرط برقرار نبود در اولین دستور بیرون از بلاک $while$ متغیر C_j یک واحد افزایش میابد. این کار تا رسیدن به شرایط مرزی $j > t$ ادامه پیدا می‌کند و در هربار اجرا، پیش از شروع بلاک $while$ مقدار j با 1 سِت می‌شود. روند اجرای این الگوریتم را برای تولید $\mathrm{comb}-(2,2)$ از $\{1,2,3,4\}$ در ادامه می‌بینیم. (شبه کد این الگوریتم در صفحه‌ی ۳۵۸ مرجع [۱] می‌باشد)

مثال ۱.۲. ***اجرای الگوریتم عمومی برای تولید*** ***comb***-$(2,2)$ ***از مجموعه مادر*** $\{1,2,3,4\}$ **:**
در ابتدا برای آرایه‌ی C مقداردهی اولیه‌ی زیر را داریم:

$$C = [0,1,4,0]$$

که نمایشی از 0011 است. حال برای C_1 و C_2 شرط $C_2 = C_1 + 1$ برقرار است. پس C_1 با مقدار 0 سِت می‌شود. ولی شرط $C_3 = C_2 + 1$ برقرار نیست، چراکه $C_3 = 4$ در صورتی که $C_2 = 1$ پس از بلاک $while$ خارج می‌شویم و مقدار C_2 با مقدار $C_2 + 1$ یعنی 2 سِت می شود. حال برای C داریم:

$$C = [0,2,4,0]$$

که به درستی نمایشی از 0101 است. در تکرار بعدی ابتدا دوباره j با 1 مقدار دهی می‌شود. در ابتدای بلاک $while$ شرط $C_2 = C_1 + 1$ برقرار نیست. چراکه $C_1 = 0$ ولی $C_2 = 2$ پس از بلاک خارج شده و C_1 با مقدار $C_1 + 1$ یعنی 1 مقداردهی می‌شود. برای C داریم

$$C = [1,2,4,0]$$

که به درستی نمایشی از 0110 است. در مرحله‌ی بعدی شرط $C_2 = C_1 + 1$ برقرار است. پس C_1 با 0 سِت می‌شود ولی $C_3 = C_2 + 1$ برقرار نیست. چراکه $C_2 = 2$ ولی $C_3 = 4$ پس از بلاک خارج می‌شویم و C_2 را با $C_2 + 1$ یعنی 3 مقداردهی می‌کنیم. در این تکرار به درستی شی 1001 تولید می‌شود. در دو تکرار بعدی هر بار C_1 یک واحد افزایش میابد. در اولین تکرار از 0 به 1 و در دومین بار از 1 به 2 و این دو تکرار به درستی دو شی 1010 و 1100 را تولید می کنند. در تکرار نهایی ابتدا شرط $C_2 = C_1 + 1$ بررسی می شود که درست است و $C_1 = 0$ خواهد شد، در تکرار بعدی $C_3 = C_2 + 1$ به درستی برقرار است پس $C_2 = 1$ خواهد شد. حال به نگهبان می‌رسیم. مقدار j به 3 رسیده است که از t بزرگتر است. پس به درستی شرط پایانی $j > t$ را تشخیص دادیم و از اجرا خارج می‌شویم. همچنین به درستی 6 شی $0011, 0101, 0110, 1001, 1010, 1100$ را تولید کردیم.

این الگوریتم نمونه‌ی سریع‌تری هم دارد که در تمرینهای همان بخش از منبع [۱] آمده است. ولی به راحتی می‌توان دید که زمان اجرای این الگوریتم به ازای هر شی به مقدار t وابسته است. همچنین فاصله

[8] Sentinel

همینگ [9] بین اشیا نیز به طور میانگین بالا و وابسته به t و n بوده که این پیچیدگی محاسبه‌ای [10] آن را بالا می‌برد.

۲.۲ روش ناکارای بازگشتی

در این روش بر روی اشیا انتخاب شده بازگشت می‌زنیم. به بیانی سوال بازگشتیِ *"آیا شی c_i در انتخابهایمان هست؟"* را در هر مرحله از بازگشت پاسخ می‌دهیم:

$$C(m,n) = \begin{cases} c_i \in \text{selected} \rightarrow b_i = 1, C(m-1, n-1) \\ c_i \notin \text{selected} \rightarrow b_i = 0, C(m, n-1) \\ \\ \text{Basis Conditions: } m = n \ , \ m = 0 \ , \ n = 0 \end{cases}$$

این روش با توجه به پتانسیل بالا بودن تعداد اشیا ترکیبیاتی ناکارآمد است. چراکه تصور کنید شی kام تولید شده است. در این هنگام تمامی اشیا بعد از k به صورت نیمه‌کاره در درخت بازگشت ساخته شده‌اند و مموری اشغال کرده‌اند. همچنین در واقعیت مموری ما بینهایت نیست، پس این برنامه به احتمال زیاد برای اعداد نسبتا بزرگ خطای سرریز بافر [11] خواهد داد. چراکه آدرسهای برگشت این توابع به علت عمق بالای بازگشت استک را نابود خواهند کرد. شاید بتوان با استفاده از روشهایی این راه‌حل را بهتر ساخت ولی همچنان زمان وابسته به t و میانگین فاصله همینگ وابسته به t و n و همچنین پیچیدگی در بهبود روش بازگشتی، آن را از رده‌ی الگوریتمهای کاربردی به کل خارج می‌کند.

۳ کد گریِ لایه‌های میانی [12]

۱.۳ کمترین فاصله‌ی همینگ در شی‌های ترکیبیاتی

در الگوریتم‌های تولید سریع اشیا ترکیبیاتی، آنچه ایده‌آل است، فاصله بسیار پایین همینگ بین یک شی و شی بعدیش است. ما به دنبال فاصله همینگ [13] 1 یعنی کد گری [14] هستیم. به طور کلی کد گری درست در تولید اشیا ترکیبیاتی، کمترین تغییرات در لیست تولید شده‌ها را به ما خواهد داد. همچنین در تولید این اشیا به دنبال آنیم که لیستمان سیکلیک [15] باشد، تا به انتها رسیدن تولید نیز راحت‌تر قابل تشخیص شود.

در اینجا فاصله همینگ 2 نیز می‌تواند به نوعی بهینه حساب شود. دلیل آن روشن است: زمانی که در یک (s,t)-comb ما یک بیت با مقدار 1 را با بیتی با مقدار 0 تعویض می‌کنیم، کمترین تغییرات ممکن را داشته‌ایم. برای مثال کد زیر همه‌ی $(2,2)$-comb ها را با فاصله همینگ 2 و به صورت سیکلیک

[9] Hamming Distance
[10] Arithmetic Complexity
[11] Stack Overflow
[12] Middle levels Gray codes
[13] Hamming Distance
[14] Gray code
[15] Cyclic

تولید می‌کند، برای تاکید، زیر دو بیتی که هربار تبادل می‌شوند خط کشیده شده است:

مثال ۱.۳. ***تولید همه‌ی*** $(2,2)$-***comb*** ***از مجموعه‌ی*** $\{1,2,3,4\}$ ***با فاصله‌ی همینگ*** 2 ***:***

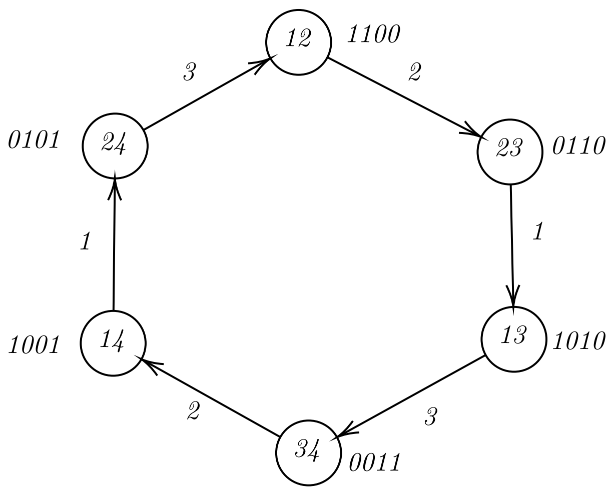
$$\to 1\underline{10}0 \to 10\underline{10} \to \underline{10}01 \to 0\underline{10}1 \to 0\underline{0}1\underline{1} \to \underline{0}1\underline{1}0 \to$$

ولی اگر موقعیت یکی از بیتهایی که در آن تبادل [۱۶] صورت می‌گیرد را ثابت نگه داریم، می‌توانیم آن موقعیت را از نمایشمان حذف کنیم، چرا که می‌توان به راحتی متوجه شد که اگر این بیت که مقدار آن را ثابت گرفتیم هربار یک طرف از مبادله باشد، آن‌گاه یک در میان 0 و 1 خواهد شد، پس می‌دانیم که زمانی که تعداد یکهایمان $t-1$ است مقدار آن بیت حذف شده برابر 1 و زمانی که تعداد یکها یا همان وزن [۱۷] برابر t است مقدار آن برابر 0 است. در هر بار اجرا نیز با دانستن زوج یا فرد بودن اندیس شی ترکیبیاتی تولید شده (این که این شی چندمین شی تولید شده است) می‌توانیم مقدار بیت حذف شده را محاسبه کنیم. [۳] در زیربخش بعدی به معرفی یک نمونه از این نمایش گری برای (s,t)-comb خواهیم پرداخت.

۲.۳ Star Transposition

اگر بیت اول را در تبادل ثابت فرض کنیم، در هر مرحله یکی از بیتهایی که مقداری مکمل بیت اول دارد با آن جابه‌جا خواهد شد. در مثال زیر اندیس گذاری از چپ بوده و مقدار اندیس اول 0 می باشد. در هر تولید شی یک تبادل به شکل $(0,k)$ داریم که k اندیس بیتی است که با اولین بیت جابه جا می‌شود و رنجی در $[1,n-1]$ دارد. این روش، یعنی روشی که در هر مرحله یکی از مکمل‌های بیت اول را با بیت اول جابه‌جا می‌کنیم، *Star Transposition* نام دارد. مثال زیر یک نمونه از تولید سیکلیک $(2,2)$-comb برای مجموعه $\{1,2,3,4\}$ به وسیله‌ی *Star Transposition* است.

مثال ۲.۳. ***تولید سیکلیک*** $(2,2)$-***comb*** ***های*** $\{1,2,3,4\}$ ***با*** ***Star Transposition*** ***:***



برای کوته‌نویسی و به علت تک‌رقمی بودن اعضای مجموعه‌ی مادر، ویرگول را به عنوان جداکننده حذف کردیم و می‌دانیم که 12 *در این‌جا همان زیرمجموعه‌ی* $\{1,2\}$ *با رشته‌ی باینری* 1100 *می‌باشد. در این*

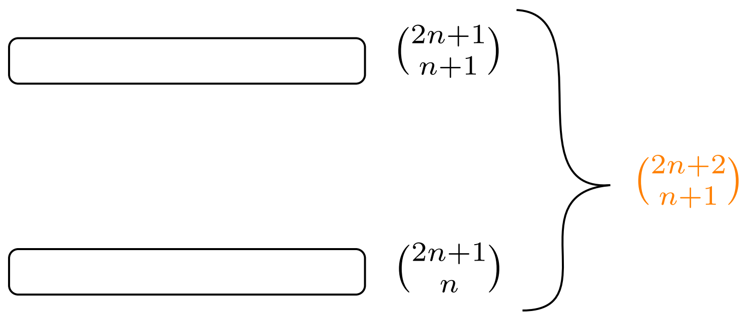
[16] Exchange
[17] Weight

مثال روی هر یال برچسب اندیسی خورده است که در Star Transposition بیت موقعیت 0 با آن بیت جابه‌جا شده است. برای مثال از نود 12 به 23 با جابه‌جایی بیت موقعیت 0 و 2 رسیده‌ایم. این تبادل با نوتیشن $(0,2)$ نمایش داده می‌شود و از آنجا که در Star Transposition بیت اول همیشه 0 است، نگه داری یک عدد به ازای هر تبادل کافیست. اگر این اعداد را به شکل یک رشته در نظر بگیریم، مثلا در اینجا $(2\ 1\ 3\ 2\ 1\ 3)$، یک Flip sequence داریم. این نام به این علت است که در هر مرحله گویی یکی از موقعیتهای غیر 0 را Flip می‌کنیم. اگر Flip sequence را به تقلید از منبع [۳] با α معرفی کنیم. آنگاه اولین تبادل همان $(0,\alpha_0)$ است. به همین ترتیب آخرین تبادل نیز همان $(0,\alpha_{\binom{2n+2}{n+1}-1})$ که در اینجا $(0,\alpha_{\binom{4}{2}-1}) = (0,\alpha_5) = (0,3)$ است خواهد بود. پس با داشتن t و n و Flip sequence می‌توانیم همه‌ی $(n+1,n+1)$-comb را با کد گری تولید کنیم.

۳.۳ تقارن در لایه‌های میانی

به لایه‌های با وزن n و $n+1$ در Q_{2n+1} لایه‌های میانی [۱۸] می‌گویند.

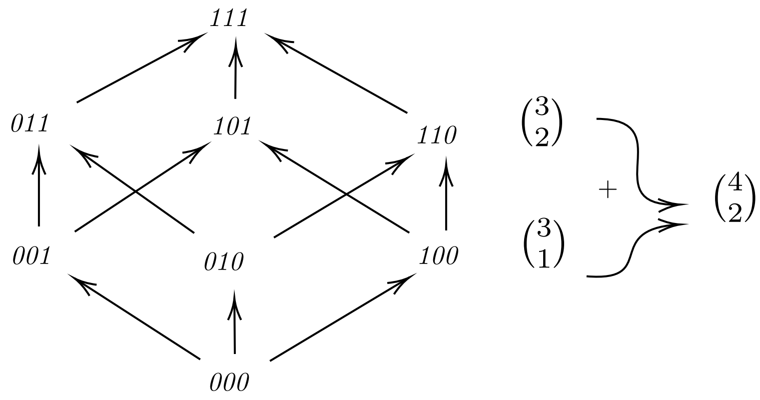
می‌توان مشاهده کرد که در لایه‌ی با وزن n در Q_{2n+1} از آنجا که دقیقا n عدد 1 داریم و بقیه‌ی $2n+1$ عدد 0 هستند. تعداد نودهای این لایه به روشنی $\binom{2n+1}{n}$ است. همچنین تعداد اعضای این لایه با لایه‌ی بالایی‌اش یکسان است، چراکه در لایه‌ی بالایی، یعنی لایه‌ی با وزن $n+1$ دقیقا به همان تعداد که 1 داشتیم اینجا 0 داریم و برعکس. در لایه‌ی با وزن $n+1$ دقیقا $n+1$ عدد 1 داریم و بقیه‌ی $2n+1$ عدد 0 هستند. پس تعداد نودهای این لایه نیز به روشنی $\binom{2n+1}{n} = \binom{2n+1}{n+1}$ است.



از اینها نتیجه می‌شود که لایه‌های میانی دقیقا $2.\binom{2n+1}{n}$ نود دارند. همچنین می‌دانیم که تمام این نودها باهم متمایزند. تقارن قابل توجه اینجاست که این تعداد با تعداد اشیا ترکیبیاتی $(n+1,n+1)$-comb برابر است. تعداد این اشیا به روشنی $\binom{2n+2}{n+1}$ است و خیلی سخت نیست که ببینیم $\binom{2n+2}{n+1} = 2.\binom{2n+1}{n}$ در نتیجه، تناظر یک‌به‌یک و پوشایی بین اشیای $(n+1,n+1)$-comb و لایه‌های میانی Q_{2n+1} وجود دارد. این تناظر به شکلی است که فاصله همینگ هر دو عضو متناظر در لایه‌های میانی Q_{2n+1} دقیقا 1 است *(توجه شود که تنها برخی از این Bijection ها خاصیت فاصله‌ی همینگ 1 را دارند.)* . پس پیدا کردن این تناظر به ما اجازه می‌دهد که به کد گری‌ای دست پیدا کنیم که می‌تواند بازنمایِ اشیا ترکیبیاتی از کلاس $(n+1,n+1)$-comb باشد.

[18] Middle levels

مثال ۳.۳. ***برابری تعداد اشیا $(n+1, n+1)$-comb با تعداد نودهای لایه‌های میانی Q_{2n+1} به ازای $n=1$:***



۴.۳ کد گریِ لایه های میانی [19]

اگر از یکی از نودهای لایه‌های میانی شروع کنیم و بتوانیم درون سیکلی همیلتونی در این دولایه حرکت کنیم و به نقطه آغازین برگردیم، راسهایی که تا به حال آن‌ها را ملاقات کردیم، همان کد گری لایه‌های میانی می‌باشد. توجه شود که بین دو راس در Q_{2n+1} تنها در صورتی یال وجود دارد که این دو تنها در یک بیت اختلاف داشته باشند. با توجه به زیربخش قبل می‌دانیم که این سیکل یک تناظر یک‌به‌یک و پوشا با اعضای $(n+1, n+1)$-comb است. از آن‌جا که این کد یک کد با فاصله همینگ 1 است، با توجه به زیربخش ۲ از همین بخش، یک دنباله‌ی تبادل [20] به شکل (k, c_i) وجود دار که در آن k ثابت است. در حقیقت اینجا k همان عنصر اول با اندیس 0 است. در نتیجه این کد گری به ما یک *Flip sequence* می‌دهد که با *Star Transposition* می‌توانیم اشیا $(n+1, n+1)$-comb را تولید کنیم. برای ساخت این *Bijection* کافیست روی هر یال در سیکل همیلتونی لایه‌های میانی، موقعیت بیتی را قرار دهیم که در نود بعدی *Flip* شده است. مثلا اگر از 100 به 110 رفته ایم، اگر از 1 و از چپ شروع کنیم، یال مربوطه با مقدار 2 برچسب‌گذاری می‌شود، چون موقعیت بیت 2 از 0 به 1 تغییر کرده. از آن‌جا که در لایه‌ی با وزن n تعداد 1ها بجای $n+1$ برابر n است. اگر این لایه را با یک 1 الحاق کنیم، اول اینکه طول هر نود به درستی برابر $2n+2$ خواهد شد. دوم اینکه تعداد بیتهای با مقدار 1 نیز دقیقا به درستی همان $n+1$ خواهد بود. به همین ترتیب لایه‌ی بالایی را با یک 0 الحاق کنیم تا بازنمایِ باینری اشیا $(n+1, n+1)$-comb را داشته باشیم. توجه شود که بین هیچ نودی در یک لایه‌ی یکسان یالی وجود ندارد و همچنین تمام نودها در این دو لایه متمایز و با طول $2n+1$ بوده‌اند. در نتیجه الحاق 1 به یک لایه و الحاق 0 به لایه‌ی دیگر متمایز بودن آن‌ها را کماکان حفظ می‌کند. در مثال بعدی یک نمونه از این تناظر را به ازای $n=1$ می‌بینیم.

---
[19] Middle levels Gray codes
[20] Exchange Sequence

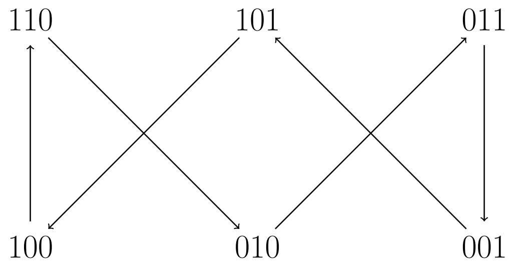


Figure 1: Hamilton cycle in Middle level of 3-cube

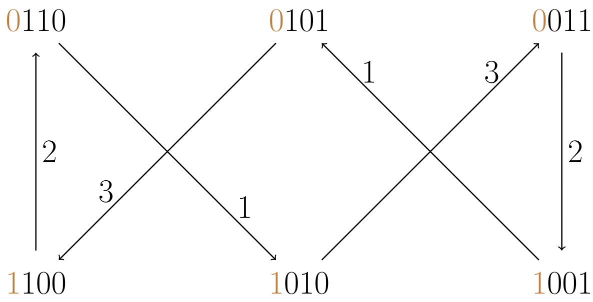


Figure 2: Star Transposition for $(2, 2)$*-combination !!!*

مثال ۴.۳. همانطور که مشاهده می‌شود، نمایش باینری اشیا $(2, 2)$-*comb* را توسط الحاق 0 به لایه‌ی با وزن $n+1$ و الحاق 1 به لایه‌ی با وزن n به دست آوردیم. همچنین با کنار هم قرار دادن برچسب یالها به *Flip Sequence* می‌رسیم:

Exchange List: $(0, 2), (0, 1), (0, 3), (0, 2), (0, 1), (0, 3)$

$\alpha = (2 \quad 1 \quad 3 \quad 2 \quad 1 \quad 3)$ *(Flip sequence)*

Middle levels gray code: $(011, 001, 101, 100, 110, 010)$

۴ قضیه‌ی لایه‌های میانی [۲۱]

۱.۴ حدس لایه‌های میانی [۲۲]

این حدس توسط *Havel*، *Buck* و *Wiedemann* در اوایل دهه‌ی ۸۰ زده شد. *Buck* و *Wiedemann* این حدس را اولین بار در [۲] عمومی کردند. ادعای آن‌ها این بود که برای هر *Hypercube* با ابعاد فرد بزرگترمساوی 1 ، سیکل همیلتونی در لایه‌های میانی وجود دارد.

از جمله تلاشهایی که در روند اثبات نهایی بسیار مفید بود، می‌توان به تلاش برای پیدا کردن یک پارتیشن از *Matching* هایی اشاره کرد که روی هم می‌توانند اجزای سیکل همیلتونیه لایه‌های میانی را تشکیل دهند. این تلاش اولین بار چهار سال بعد از انتشار مقاله، توسط *Kierstead* در مقاله [۵] انجام شد ولی متاسفانه *Matching* خاصی که او روی آن نام *Lexical Matching* گذاشته بود، نمی‌توانست همه‌ی سیکل همیلتونی لایه‌های میانی را تشکیل دهد. بعدها در سال ۲۰۰۸ دقیقا ۴ سال پیش از آنکه *Mütze* این حدس را اثبات کند، این روش توسط *Kelkar* در مقاله‌ی [۶] استفاده شد تا توسط *Lexical Matching* های معرفی شده در سال ۱۹۸۸ بتواند یک *2-factor* از لایه‌های میانی محاسبه کند. او تا حد زیادی به اثبات نزدیک شده بود، در نهایت این *Mütze* بود که در سال ۲۰۱۲ و ۲۰۱۴ یک اثبات برای وجود سیکل همیلتونی در لایه‌های میانی ارائه داد [۷] [۸] . او نیز با تعمیم *Lexical Matching* توانست پارتیشنی روی هم کامل [۲۳] از مسیرهایی ارائه دهد که روش ساختشان بسیار به *Lexical Matching* ها نزدیک است. این پارتیشن در نهایت با وصل کردن چند راس بهم، یک *2-factor* از لایه‌های میانی در Q_{2n+1} به ما می‌دهد. درست مانند آن چیزی که *Kelkar* در فصل سوم بخش دوم [۶] به آن پرداخته است.

این اثبات، یعنی اثبات *Mütze*، تکنیکالیتی‌های فراوانی داشت که بعدتر در [۹] روند اثبات به گفته‌ی خود *Mütze* کوتاه‌تر و ساده فهم‌تر شد (؟).

آنچه *Mütze* در اثبات وجود سیکل همیلتونی در لایه‌های میانی در [۹] ارائه داد، اثبات با ساخت [۲۴] می‌باشد که از همان روش نیز برای ارائه الگوریتم استخراج کد گری لایه‌های میانی [۲۵] در مقاله‌ای که در همان سال منتشر شد استفاده کرد [۴]. این دو مقاله که در یکی به اثبات و در دیگری به الگوریتم پرداخته شده است، تا میزان زیادی باهم اشتراک دارند. در اینجا از هر دوی این مقالات کمک گرفته شده.

۲.۴ پیچیدگی زمانی ثابت به ازای هر شی [۲۶]

توجه شود که کمترین هزینه‌ی مصرفی برای تولید همه‌ی $(n+1, n+1)$-comb ها، دست‌کم برابر $\binom{2n+2}{n+1}$ است. چراکه این اندازه‌ی خروجی است و برنامه نمی‌تواند از اندازه‌ی خروجی سریعتر مساله را حل کند. از این رو منظور ما از پیچیدگی ثابت [۲۷] در واقع پیچیدگی ثابت به ازای هر شی است. به این معنی که هزینه‌ی تولید هر شی به طور میانگین برابر مقداری ثابت باشد. در نهایت نیز اگر بتوانیم به پیچیدگی ثابت به ازای هر شی برسیم، کل اشیا در زمان $c.\binom{2n+2}{n+1}$ که c مقداری ثابت است، حل می‌شود.

بد نیست که منظورمان را از پیچیدگی دقیقتر بیان کنیم. منظور از پیچیدگی در این نوشتار، پیچیدگی محاسبه‌ای [۲۸] می‌باشد. در تعریف این پیچیدگی دو فاکتور اصلی تاثیرگذار، یک عمق *DAG* و دو اندازه

[21] Middle levels theorem
[22] Middle levels conjecture
[23] Collectively Exhaustive
[24] Proof by construction
[25] Middle levels Gray codes
[26] Constant time per object
[27] Constant
[28] Arithmetic Complexity

مدار [۲۹] می‌باشد. لازم به ذکر است که منظور از اندازه اینجا، درجه‌ی ورودی بر مدارهای محاسبه پایه‌ای داخل DAG است.

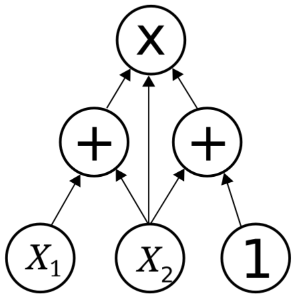


Figure 3: Size and Depth, *Size*=6 , *Depth*=2

۳.۴ اثبات و الگوریتم

۱.۳.۴ نمای از بالایِ اثبات

اثبات به این نحو است که ابتدا یک زیرگراف از لایه‌های میانی با نام H_n0 استخراج می‌شود. این زیرگراف شامل راس‌هایی است که بیت آخرشان مقدار 0 دارد. حال روی این زیرگراف یک پارتیشن از مسیرها پیدا می شود. سپس به علت تقارنی که بین H_n0 و H_n1 وجود دارد. از روی مسیرهای محاسبه شده برای H_n0 که همه‌ی H_n0 را پوشش می‌دهند و باهم اشتراکی ندارند، مسیرهای H_n1 را به دست می‌آوریم. از آنجا که ساخت مسیرهای H_n0 توسط یک تابع بازگشتی که روی راس‌های شروع عمل می‌کند محاسبه می‌شود، ما از پیش راس‌های شروع‌مان را انتخاب کرده بودیم. به ازای هر یک از راس‌های شروع یک مسیر در H_n0 یک راس پایانی وجود دارد که آن‌هم قابل محاسبه است. حال که H_n0 را به چند مسیر پارتیشن کردیم و برای هر مسیر نیز شروع و پایان را می‌دانیم، و همچنین می‌دانیم که به ازای هر مسیر در H_n0 یک مسیر متقارن در H_n1 وجود دارد، که آن هم قابل محاسبه است. حال با تابعی که بین H_n0 و H_n1 تناظر را برقرار می‌کند، راس‌های شروع و پایان این دو بخش را به هم متصل می‌کنیم. به این ترتیب یک 2-*factor* از لایه‌های میانی به دست آوردیم. و می‌دانیم که از هر 2-*factor* در گراف، دست‌کم یک سیکل همیلتونی قابل بیرون کشیدن است.

۲.۳.۴ تناظر بین H_n0 و H_n1

ما روی گراف القا شده توسط راس‌هایی که با 0 تمام می‌شوند مسیرهایی جدا از هم و روی هم کامل را به دست می‌آوریم چرا که به ازای هر مسیر در H_n0 یک مسیر معادل در H_n1 وجود دارد. لازم به ذکر است که پس از انتخاب این راسها 0 آخر آنها را حذف می‌کنیم، و هر راسی از رشته‌ای باینری به طول $2n + 1$ به رشته‌ای باینری به طول $2n$ تبدیل می‌شود. این تناظر از روی تناظر خردتری که بین راس‌های H_n0 و H_n1 در Q_{2n+1} وجود دارد به دست می‌آید. برای هر $v_i \in H_n0$ راس متناظر $\overline{rev}(x)1 \in H_n1$ وجود دارد. تابع $\overline{rev}(x)$ ابتدا آرگومان خود را که رشته‌ای باینری است *Reverse* می‌کند، سپس مکمل آن را حساب می‌کند. به این ترتیب برای راس $100 \in Q_3$ راس متناظر $101 \in Q_3$ وجود دارد.

[29] Size of a circuit

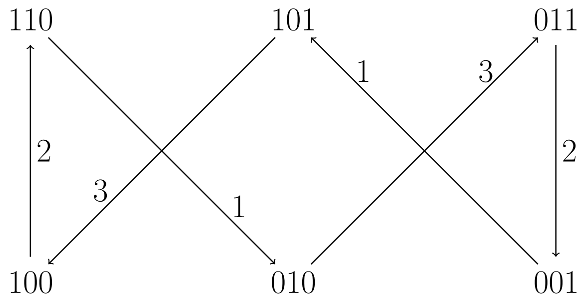


Figure 4: HamCycle in Q_3

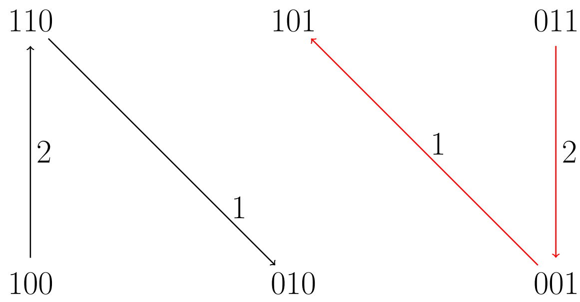


Figure 5: HamPath(H_n0) $\Leftrightarrow$ HamPath(H_n1)

مثال ۱.۴.   مسیر همیلتونی متناظر در H_n1:

سیکل همیلتونی Figure 4 را در نظر بگیرید. مشخص است که راسهای H_n0 در اینجا سه راس $110, 100, 010$ هستند. در Figure 5 مسیر همیلتونی ایجاد شده در H_n0 از این سیکل همیلتونی به رنگ مشکی دیده می‌شود. دشوار نیست که ببینیم به ازای هر یال (v_i0, v_j0) یک یال $(\overline{rev}(v_j)1, \overline{rev}(v_i)1)$ وجود دارد. مثلا برای $(100, 110)$ یال متناظر $(001, 101)$ وجود دارد و برای $(110, 010)$ نیز $(011, 001)$ وجود دارد. حال اگر انتهای مسیر همیلتونیه H_n0 را که 010 است در نظر بگیریم و آن را به متناظر خود که شروع مسیر همیلتونی متناظر در H_n1 است وصل کنیم. پس راس 010 را به متناظر خود یعنی 011 وصل کرده‌ایم و به همین ترتیب اگر انتهای مسیر همیلتونیه H_n1 را که 101 است به متناظر خود یعنی 100 وصل کنیم، آنگاه یک 2-factor از Q_3 به دست آوردیم، که بر حسب کوچکیِ این Hypercube اتفاقا سیکل همیلتونی هم هست (Figure 6).

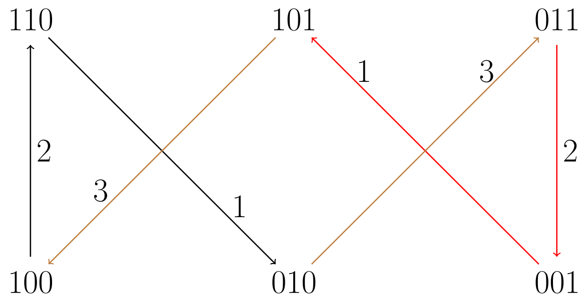


Figure 6: 2-factor in Q_3

۳.۳.۴ رشته های بیتی و مسیرهای دیک [30]

پارتیشن H_n0 به چند مسیر مجزا به این شکل صورت می گیرد که رشته‌های *Dyck* زیرمجموعه‌ی H_n به عنوان راس ابتدایی در مسیر انتخاب می‌شوند. سپس به وسیله‌ی تابع δ به صورت بازگشتی از روی هر یک از این رشته‌ها یک *Flip Sequence* به دست می‌آوریم. به وسیله‌ی این *Flip Sequence* مسیر مجزای شروع شده از این رشته‌ی *Dyck* به دست می‌آید. توجه شود که در اینجا ما روی H_n کار می‌کنیم که 0 آخر از هر راس در آن حذف شده است. در نتیجه در H_n همه‌ی رشته‌های به طول $2n$ با وزن n و $n+1$ را داریم و می‌دانیم که $D_n \subset H_n$ چراکه اینجا H_n همان $B_{2n,n} \bigcup B_{2n,n+1}$ است. منظورمان از D_n کلمات *Dyck* به طول n و منظورمان از $B_{2n,n}$ نیز رشته‌های باینری به طول $2n$ با وزن n می‌باشد.

می‌دانیم که برای هر رشته‌ی *Dyck* یک مسیر متناظر *Dyck Path* داریم، که در آن از نقطه $(0,0)$ در لاتیس $\mathbb{Z}_2$ شروع کرده و در هرقدم اگر بیت متناظر 1 بود $(+1,+1)$ داریم و اگر بیت متناظر 0 بود $(+1,-1)$. از آن‌جا این رشته ها *Dyck* هستند، پس n عدد 1 و n عدد 0 داریم، و دشوار نیست که ببینیم با توجه به خاصیت پیشوندی که در هر رشته‌ی *Dyck* وجود دارد، به این شکل که در همه‌ی پیشوندهای [31] رشته حداقل به اندازه‌ی 0 ها بیت 1 داریم، در اینجا نیز هیچ‌گاه از محور x در لاتیس پایین‌تر نمی‌رویم (y هیچ وقت منفی نمی‌شود). در نهایت نیز در $y = 0$ متوقف خواهیم شد، چراکه پرانتزها بالانس هستند.

رشته‌های دیگری نیز وجود دارند که بسیار شبیه به رشته‌های *Dyck* هستند که آنها را با D_n^- نشان می‌دهیم. این رشته‌ها، رشته‌های به طول $2n$ هستند که در آنها دقیقا به ازای یک پیشوند تعداد 1ها بیشتر از 0هاست. اگر پیشوندی را که در آن تعداد 1 ها بیشتر از 0 شده را در نظر بگیریم، در صورتی می‌توان گفت که تنها همین یک پیشوند این خاصیت را نقض کرده که کارکتر بعدی‌اش حتما 1 باشد. به این ترتیب برای $x \in D_n^-$ یک شکل $x = u01v$ وجود دارد که برای حفظ خاصیت در آن باید u و v هر دو رشته‌های *Dyck* باشند. با تکان دادن این 01 میان u و v های مختلف گویی همان کاری را می‌کنیم که در شمارش رشته‌های *Dyck* انجام دادیم. زمانی که $x \in D_n$ پس $x = (u)v$ و حرکت پرانتزها به ازای هر u و v دقیقا مانند حرکت 01 در این رشته به ازای هر u و v است.

در H_n هر $x \in D_n$ راس آغازین یک مسیر مجزا و $x' \in D_n^-$ که در آن x' راس متناظر x است (با توجه به تناظر یک به یک و پوشای بین D_n و D_n^-) راس پایانی آن مسیر می‌باشد. مسیر بین این دو راس نیز توسط تابع δ به صورت بازگشتی و با *Flip* کردن متوالی به دست می‌آید.

در ادامه برای تولید مسیرهای مجزا از یک رشته *Dyck* ، از *Decomposition* رشته‌ها استفاده می‌کنیم.

[30] Bitstring and Dyck path

[31] Prefixes

۴.۳.۴ Canonical decomposition

برای $x \in D_n$ مسیر مجزا به این شکل حساب می‌شود که در هر مرحله به صورت بازگشتی x را *Decompose* می‌کنیم. این *Decomposition* توسط تابع δ انجام می‌شود، به این شکل که رشته‌ی x ابتدا به شکل $1u0v$ شکسته می‌شود. حال اول 1 را *Flip* می‌کنیم سپس 0 را *Flip* می‌کنیم و پس از آن تابع بازگشتی دیگری را با نام δ' روی u اجرا می‌کنیم و به دلایل ترکیبیاتی از v صرف نظر می‌کنیم. حال در u که خود یک رشته‌ی *Dyck* است به صورت بازگشتی *Decomposition* را انجام می‌دهیم، دوباره u به شکل $u = 1u'0v'$ شکسته می‌شود. این بار برای هر دوی u' و v' عمل *Decomposition* را انجام می‌دهیم. ولی این *Decomposition* با کمی تغییرات انجام می‌شود، به این شکل که، ابتدا بیت 0 را *Flip* می‌کنیم، سپس بیت 1 را *Flip* می‌کنیم و سپس u' را *Decompose* می‌کنیم، سپس یک بیت عقب‌تر از 0 را *Flip* می‌کنیم و دوباره 1 را *Flip* می‌کنیم، در نهایت هم v' را *Decompose* می‌کنیم.

این نحوه‌ی تولید *Flip Sequence* توسط دو تابع δ و δ' به شکلی که به ازای هر رشته‌ی *Dyck* یک مسیر با راس‌های منحصر به فرد بدون اشتراک راس با مسیرهای متناظر دیگر رشته‌های D_n تولید کند و همچنین همه‌ی $B_{2n,n} \bigcup B_{2n,n+1}$ را پوشش دهد، دلایل پیچیده‌ی ترکیبیاتی دارد که برای جزییات آن می‌توانید به [۷] [۵] [۶] [۸] رجوع کنید.

۵.۳.۴ محاسبه‌ی 2-factor از گراف

در نهایت 2-factor گراف لایه‌های میانی Q_{2n+1} به این شکل محاسبه می‌شود:

$$\mathcal{C} = \mathcal{P}_n 0 \ \ \bigcup \ \ \overline{rev}(\mathcal{P}_n)1 \ \ \bigcup \ \ M'_n$$

$$M'_n := \{(x0, x1) | x \in D_n \ \ \bigcup \ \ D_n^-\}$$

که در آن $\mathcal{P}_n$ مسیرهایی است که از رشته‌های *Dyck* زیرمجموعه H_n توسط تابع δ حساب کردیم و $\overline{rev}(\mathcal{P}_n)$ مسیرهای متقارن متناظرشان است. در انتهای هر یک نیز 0 و 1 که حذف شده بود الحاق کردیم. در نهایت M'_n یالهای بین راس‌های آغازین و پایانی در $\mathcal{P}_n 0$ و راس‌های آغازین و پایانی در $\overline{rev}(\mathcal{P}_n)1$ است. با اتصال این دو پارتیشن مسیرهای مجزا موفق شدیم که کل گراف لایه‌های میانی را با چند سیکل بدون اشتراک راس پوشش دهیم. به دست آوردن یک سیکل از روی یک 2-factor هم بسیار بدیهی و ساده است.

References

[1] D. Knuth, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. No. pt. 1, Pearson Education, 2014. 1, 3, 4

[2] M. Buck and D. Wiedemann, "Gray codes with restricted density," *Discrete Mathematics*, vol. 48, no. 2-3, pp. 163–171, 1984. 1, 10

[3] A. Merino, O. Mička, and T. Mütze, "On a combinatorial generation problem of knuth," in *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 735–743, SIAM, 2021. 1, 6, 7

[4] T. Mütze and J. Nummenpalo, "A constant-time algorithm for middle levels gray codes," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2238–2253, SIAM, 2017. 1, 10

[5] H. Kierstead and W. Trotter, "Explicit matchings in the middle levels of the boolean lattice," *Order*, vol. 5, no. 2, pp. 163–171, 1988. 10, 14

[6] A. Kelkar, *A study of the subgraphs and the conjecture of the middle two layers graph using modular matchings*. Arizona State University, 2008. 10, 14

[7] T. Mütze and F. Weber, "Construction of 2-factors in the middle layer of the discrete cube," *Journal of Combinatorial Theory, Series A*, vol. 119, no. 8, pp. 1832–1855, 2012. 10, 14

[8] T. Mütze, "Proof of the middle levels conjecture," *Proceedings of the London Mathematical Society*, vol. 112, no. 4, pp. 677–713, 2016. 10, 14

[9] P. Gregor, T. Mütze, and J. Nummenpalo, "A short proof of the middle levels theorem," *arXiv preprint arXiv:1710.08249*, 2017. 10