ДВАДЕСЕТ И ВТОРА УЧЕНИЧЕСКА СЕКЦИЯ УК'22

Хилбертообразни криви в задачи за заявки в интервали

Атанас Димитров Димитров
СМГ „Паисий Хилендарски“, град София

Научен ръководител:
Радослав Стоянов Димитров
Oxford

Абстракт

Намирането на ефикасни алгоритми за отговарянето на заявки в множество от данни е важно както в практическата, така и в теоретичната сфера на компютърните науки.

Статията представя метод за поддържане на интервални заявки при съществуването на обновления, като това се случва чрез предварително познаване на всички данни - както множеството, така и заявките и обновленията. Използван е терминът черна кутия, за да се изложи генерално решение, независещо от въпросите, на които се търси отговор.

Разгледана е връзката между минимални по дължина хамилтонови пътища в квадратни решетки и задачата. Хилбертовите криви са разгледани като ефикасни и практически удобни аппроксимации. Показани са изследвания сравняващи метода с други, познати в литературата, такива.

Допълнително генерализиране е направено чрез въвеждането на семейството на Хилбертообразните запълващи-пространството криви. Предложени са евристични алгоритми за намирането им.

Abstract

Having efficient algorithms for answering queries in a given data set proves helpful in both the practical and theoretical spheres of computer science, as it often arises as a subroutine of many other algorithms.

The following paper aims to present a novel method for handling range queries and updates in a given data set in an offline manner. The approach uses the notion of a black box as a layer of abstraction, which helps make the problem more generalizable.

We have examined the connection between minimum length hamiltonian paths in multidimensional grids and the given problem. The Hilbert curves were looked at as an efficient and practical approximation. Benchmarks were conducted to measure and compare the efficiency of the approach against other known methods.

Further attempts at generalizing the solution were made, as we defined the Hilbertian family of space-filling curves - an extension to the Hilbert curve and provided heuristic algorithms for finding them.

1 Въведение

1.1 Дефиниция на задачата

В статията са разгледани алгоритми, които отговарят на въпроси, свързани с предварително зададен масив от данни. Формално казано, имаме зададен масив с размер N и със стойности a_i за $\forall\ 0 \leq i < N$. Освен това имаме последователност от заявки и обновления. С U ще означим нареденото множеството от обновления, всяко характеризирано от (ind, val) - променяме стойността на масива на позиция ind на val. С Q ще означим нареденото множество от заявки, всяка от които ще означим с тройката (l, r, u) - тя търси отговор на някакъв въпрос за интервала $l \leq i \leq r$, като се имат предвид първите u обновления. Друго валидно означение на елементите на заявките е l_i, r_i, u_i за $0 \leq i < |Q|$, а за обновленията ще въведем означенията ind_i, val_i и old_i (това е стойността на позиция ind_i преди обновлението) за $0 \leq i < |U|$.

1.2 Значението на ефикасните алгоритми за отговаряне на заявки

В света около нас често се налага разглеждането на големи количества от данни. Така се стига до заключението, че за пълноценно използване на технологиите са наложителни ефикасни алгоритми за справянето с динамични промени и въпроси в данните.

Друго полезно приложение на алгоритмите, отговарящи на завки, е като помощни при различни задачи свързани например с динамичното програмиране или алчните алгоритми.

1.3 Използване на черна кутия

Черната кутия е структура от данни, която отговаря на множество, в което със сложност O(T) могат да се добавят елементи, да се изваждат елементи и да се намира отговор на даден въпрос за множеството (максимум; сума; минимум; най-малко число, което не се среща в него; най-често срещаното число в него). Това е начин да се направи абстрактен въпроса в дадена задача, за да може да се изложи генерален метод. В статията, която следва, е разгледан въпросът на такава, за която всички от гореизброените операции са с една и съща сложност - O(T), където T е функция, зависеща от броя на елементите в началното

множеството.

1.4 Познати алгоритми

1.4.1 Наивен алгоритъм

Наивният алгоритъм се състои в отговаряне на всяка заявка, сортирайки ги от такава с най-малко u, минавайки по всички елементи a_i в интервала (l, r) и слагайки ги в черната кутия. При обновления на елементи в масива единственото, което трябва да се направи, е да се промени стойността в паметта. При анализ сложността на алгоритъма е $O(\sum_{i=0}^{|Q|-1}(r_i - l_i) * T + |U|)$. Тъй като всяка заявка може да е $(1, n)$, следва, че финалната сложност е $O(N * |Q| * T + |U|)$.

1.4.2 Специализиран алгоритъм за някои задачи

Нека разгледаме въпроси, които означаваме с функцията $f : \mathbb{R}^{m+1} \to \mathbb{R}$, за която е вярно, че $f(S_1 \cup S_2) = f(\{f(S_1), f(S_2)\})$. В такива случаи можем да използваме сегментно дърво [4]. В тях сложността е по-добра от тази на наивния алгоритъм - $O(logN)$ на заявка и $O(N)$ за предварително построяване на дървото, следователно общата сложност е $O(|Q| * logN + |U| * logN + N)$.

1.4.3 Разделяне на ленти

Разделянето на ленти е алгоритъм, възползващ се от съществуването на черна кутия. Той подрежда заявките в няколко групи в зависимост от стойностите им на l и u. По-кокретно, той избира константа $BLOCK$, след това групира в "ленти"всички заявки a, b, за които $\lfloor \frac{l_a}{BLOCK} \rfloor = \lfloor \frac{l_b}{BLOCK} \rfloor$ и $\lfloor \frac{u_a}{BLOCK} \rfloor = \lfloor \frac{u_b}{BLOCK} \rfloor$. Отговарянето на заявките във всяка "лента"се случва по ред на нарастващо r, използвайки алгоритъм, подобен на 2, за придвижването между 2 последователни заявки. Сложността може да бъде изчислена като $O(d * |Q|^{\frac{d-1}{d}} * N)$, ако бъде избрана добра стойност на константата $BLOCK$.

2 Релация на проблема с този за минимален по дължина хамилтонов път

2.1 Заявки в статичен масив

Нека се спрем на случая, в който $|U| = 0$, т.е. няма обновления. След това при разглеждане на процедурата, предложена в 1.4.1, можем да направим следната оптимизация - вместо всеки път да започваме да пълним структурата от данни наново, ще подредим заявките по произволен ред и ще опишем начин, по който ако вече структурата съдържа елементите отговарящи на интервала (l_i, r_i), можем да преминем в състояние, такова че структурата да съдържа елементите в интервала (l_{i+1}, r_{i+1}). В псевдокод 1 е показан код, отговарящ на описаната задача. Сложността можем да изчислим като $O(\sum_{i=2}^{N}(|l_i - l_{i-1}| + |r_i - r_{i-1}|))$. Тъй като може да се получи ситуация, в която се редуват заявки $(1, N)$, $(N/2, N/2)$, то сложността отново е от порядък $O(|Q| * N * T)$.

Нека разгледаме претеглен граф $G = (V, E)$, където $|V| = N$, а в E има ребро ab с дължина c между всяко $0 < a, b < |Q|$, където $c = |l_a - l_b| + |r_a - r_b|$. Така трансформираме задачата по подреждане в такава за минимален по дължина манхатанов хамилтонов път. Понеже този проблем е известен като NP-сложен, можем единствено да мислим в посока евристики за апроксимации [2].

Algorithm 1: Преместване на (l, r) в (l', r')

Data: T черната кутия
Data: (l, r) интервал, съдържанието на който е във T
Data: (l', r') интервал, чието съдържание ще бъде в T след изпълнението на алгоритъма
Result: T съдържащо елементите на (l', r')

$pl \leftarrow l$;
$pr \leftarrow r$;
while $pl < l'$ **do**
  // Изважда всички елементи в интервала $l \leq i < l'$
  remove(T, a_{pl});
  $pl \leftarrow pl + 1$;
end
while $pl > l'$ **do**
  // Добавя всички елементи в интервала $l' \leq i < l$
  $pl \leftarrow pl - 1$;
  add(T, a_{pl});
end
while $pr < r'$ **do**
  // Добавя всички елементи в интервала $r < i \leq r'$
  $pr \leftarrow pr + 1$;
  add(T, a_{pr});
end
while $pr > r'$ **do**
  // Изважда всички елементи в интервала $r' < i \leq r$
  remove(T, a_{pr});
  $pr \leftarrow pr - 1$;
end

2.2 Заявки в променящ се масив

Псевдокод 2 обяснява метод, с който може да се разшири решението в случай че $|Q| \neq 0$. По аналог на доказателството в секция 2.1, тук ще търсим минимален манхатанов хамилтонов път, но между точките в $\mathbb{N}^3$.

Algorithm 2: Преместване на (l, r, u) в (l', r', u')

Data: T черната кутия
Data: (l, r, u) интервал, съдържанието на който е във T
Data: (l', r', u') интервал, чието съдържание ще бъде в T след изпълнението на алгоритъма
Result: T съдържащо елементите на (l', r', u')

$pl \leftarrow l$;
$pr \leftarrow r$;
while $pl < l'$ **do**
  // Изважда всички елементи в интервала $l \leq i < l'$
  remove(T, a_{pl});
  $pl \leftarrow pl + 1$;
end
while $pl > l'$ **do**
  // Добавя всички елементи в интервала $l' \leq i < l$
  $pl \leftarrow pl - 1$;
  add(T, a_{pl});
end
while $pr < r'$ **do**
  // Добавя всички елементи в интервала $r < i \leq r'$
  $pr \leftarrow pr + 1$;
  add(T, a_{pr});
end
while $pr > r'$ **do**
  // Изважда всички елементи в интервала $r' < i \leq r$
  remove(T, a_{pr});
  $pr \leftarrow pr - 1$;
end
while $pu < u'$ **do**
  // Прилага всички обновления в интервала $u < i \leq u'$
  if $l \leq ind_{pu} \leq r$ **then**
    remove(T, $a_{ind_{pu}}$);
    add(T, val_{pu});
  end
  $a_{ind_{pu}} \leftarrow val_{pu}$;
  $pu \leftarrow pu + 1$
end
while $pu > u'$ **do**
  // Връща всички обновления в интервала $u' < i \leq u$
  if $l \leq ind_{pu} \leq r$ **then**
    remove(T, $a_{ind_{pu}}$);
    add(T, old_{pu});
  end
  $a_{ind_{pu}} \leftarrow old_{pu}$;
  $pu \leftarrow pu - 1$
end

3 Използване на запълващи-пространството криви

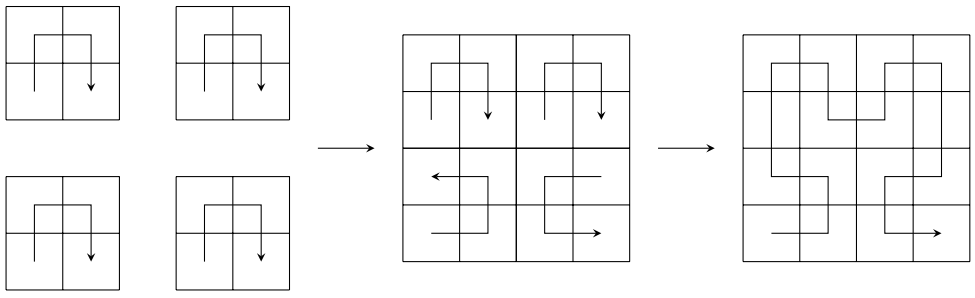
Тъй като по определение запълващи-простра34нството криви минават през всички клетки на $N \times N$ решетка, то може да подредим всички заявки по реда, по който кривата ги обхожда.

Забележка: Аналогични разсъждения могат да се приложат и в по-високите измерения.

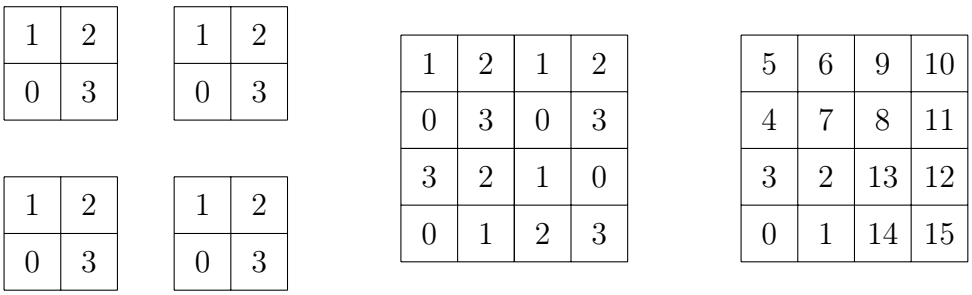
3.1 Крива на Хилберт

3.1.1 Конструкция

Кривата на Хилберт е крива минаваща през всички клетки на таблица $N \times N$ за $N = 2^k$. Тя е създадена от 4 криви на Хилберт за таблици $N' \times N'$ за $N' = 2^{k-1}$, като са направени завъртания на 2 от тях съответно по часовниковата и обратно на часовниковата стрелка. На фигура 1 е показан процесът на долепяне на четири 4×4 таблици в една 16×16. В статията се разглежда реда, по който клетките биват обходени от кривата. На фигура 2 е показан визуално начина на образуване на тази номерация. След завъртанията и долепянията във всяка подрешетка оригиналната подредба се запазва, единственото което се променя е общата подредба. Аналогично могат да се направят криви на Хилберт за по-високи [1] степени. В генералния случай за d-измерна крива в хиперкубична решетка с размер на страните $N = 2^{dk}$ можем да долепим 2^d по-малки такива с размер на страните $N' = 2^{d(k-1)}$.



Фигура 1: Образуване на крива запълваща 16×16 решетка от четири 4×4 решетки.



Фигура 2: Номерация на пътя на кривата

3.1.2 Свойства за локалност

Ще докажем, че дължината на манхатанов хамилтонов път измежду $|Q|$ точки, които са подредени по реда на кривата на Хилберт, в d-измерна решетка във формата на хиперкуб с размер на страните N е $O(d * |Q|^{\frac{(d-1)}{d}} * N)$. За улеснение следващото доказателство използва $N = 2^k$ и $|Q| = 2^{dk'}$, където d е броя измерения, т.е. ако $|Q|$ или N не спазват тези условия, ще ги увеличаваме докато това не се случи. Това няма да промени сложност на алгоритъма.

Доказателство: *Нека разделим решетката на d-измерни хиперкубични подрешетки с размер на страната $2^{k-k'}$. Първо ще разгледаме разстоянието, което пътя трябва да измине да обиколи всички подрешетки. Броя на подрешетките е $(2^{k'})^d = 2^{dk'}$. Предвижването между две такива е от порядък $O(d*2^{k-k'})$. Така сложността за минаване през всички е $O(2^{k-k'} * d * 2^{dk'}) = O(d * 2^k * (2^{k'})^{d-1}) = O(d*|Q|^{\frac{(d-1)}{d}}*N)$. Остава да се разгледа единствено допълнителното разстояние, което пътя трябва да премине, за да обиколи всички точки във всяка подрешетка. Поради рекурсивната природа на кривата всички точки от подрешетката ще бъдата последователни в подреждането. Следователно горна граница на това разстояние е максималното разстояние в една подрешетка умножено по броя точки: $O(d * |Q| * 2^{k-k'}) = O(d * |Q|^{\frac{(d-1)}{d}} * N)$. С това доказваме твърдението за порядъка на общата дължина на пътя.*

4 Изследвания

4.1 Черната кутия

Задачата за отговаряне на въпроси, свързани със сума в интервал, е избрана като подходяща за сравнение. Тъй като обратната операция на събирането

е изваждането, можем да изведем лесен псевдокод на черната кутия, който работи със сложност $O(1)$. Чрез избор на проста задача за отговор можем да избегнем случайния елемент.

Algorithm 3: Добавяне към черната кутия

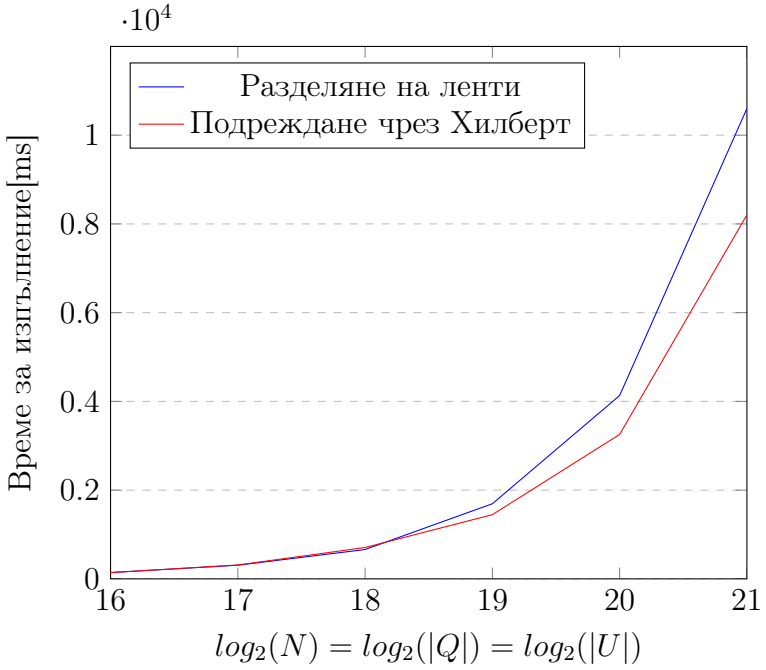
Data: T "черната кутия"
Data: x стойност, която трябва да бъде добавена
Result: T съдържащо информация за променената сума
$T.sum \leftarrow T.sum + x$

Algorithm 4: Изваждане от черната кутия

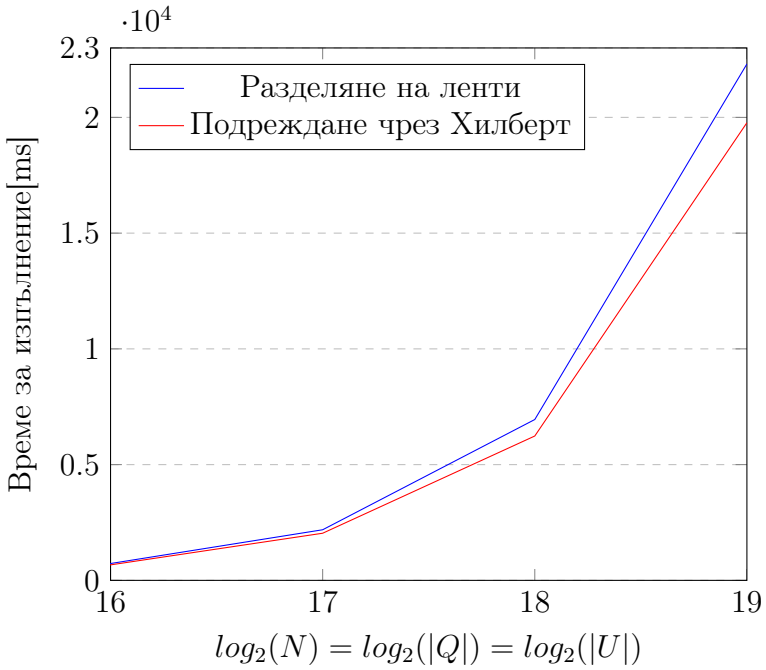
Data: T "черната кутия"
Data: x стойност, която трябва да бъде извадена
Result: T съдържащо информация за променената сума
$T.sum \leftarrow T.sum - x$

4.2 Сравнение с разделянето на ленти

Поради сходството между двата метода е добре да се направи сравнение между тях. На графики 3 и 4 са разгледани времената за изпълнение на заявки с и без обновления. Вижда се как предложеният метод е с 20% по-добър от разделянето на ленти.



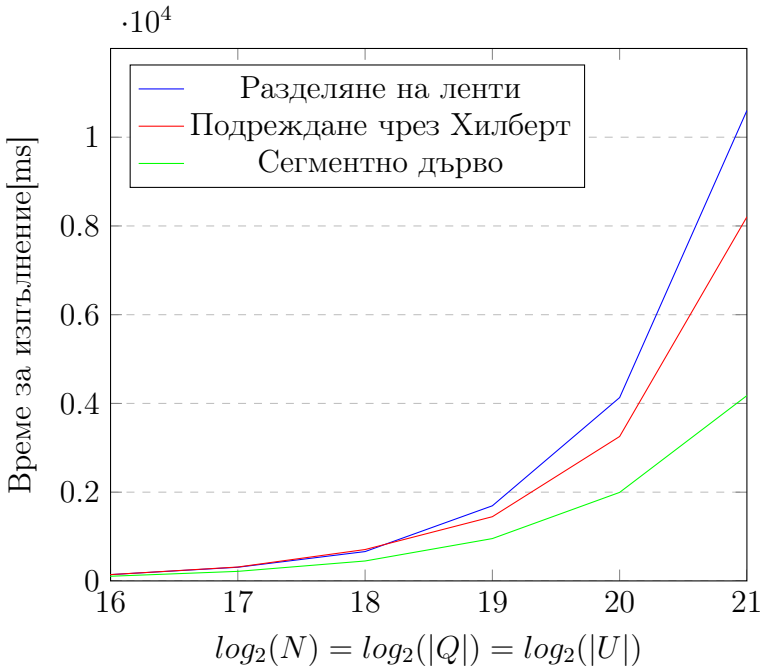
Фигура 3: Измерване без обновления



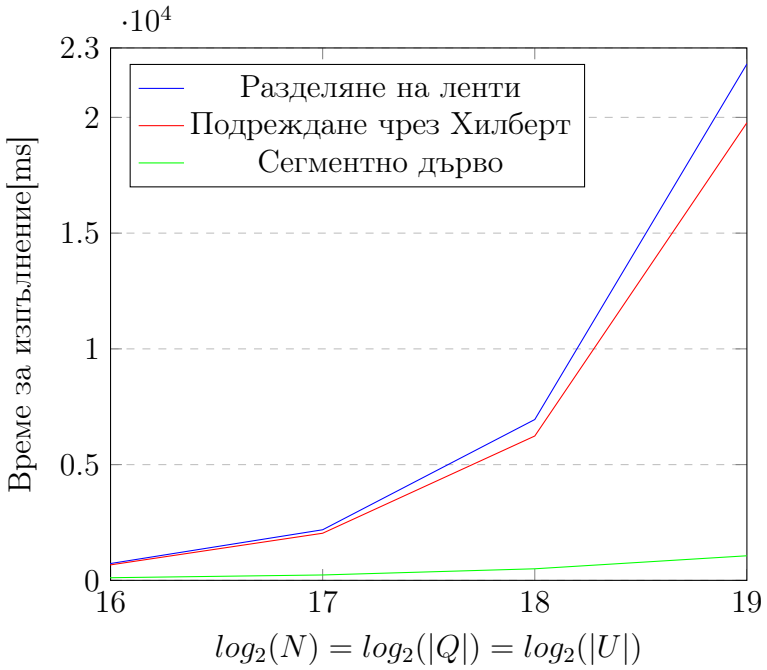
Фигура 4: Измерване с обновления

4.3 Сравнение със сегментно дърво

В измерванията е добавен алгоритъма на сегментото дърво. Фигура 5 разглежда случая, когато няма обновления, а Фигура 6, когато има.



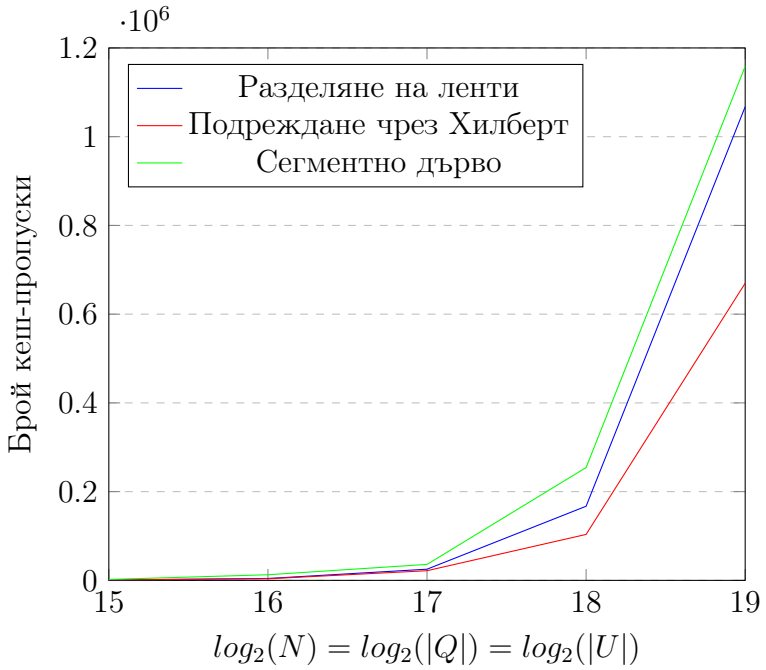
Фигура 5: Измерване без обновления



Фигура 6: Измерване с обновления

Спирайки се на фигура 5 е неочаквано, че сегментното дърво се справя едва 2 пъти по-добре от алгоритмите, служещи си с черна кутия (Разглеждайки сложносностите $O(\frac{|Q|*logN}{d*|Q|^{\frac{(d-1)}{d}}*N}) \approx O(\frac{logN}{d*|Q|^{\frac{1}{2}}}) \approx \frac{21}{2*1000} \approx \frac{1}{100}$ е теоретично-предполагаемото отношение). При фигура 6 отново наблюдаваме, че отношението е много по-малко, отколкото е теоретично-предполагаемото. По аналогични разсъждения за триизмерна крива, очакваното отношение е $O(\frac{|Q|*logN}{d*|Q|^{\frac{(d-1)}{d}}*N}) \approx O(\frac{logN}{d*|Q|^{\frac{2}{3}}}) \approx \frac{21}{3*9000} \approx \frac{1}{2000}$, а на практика то е $\frac{1}{20}$.

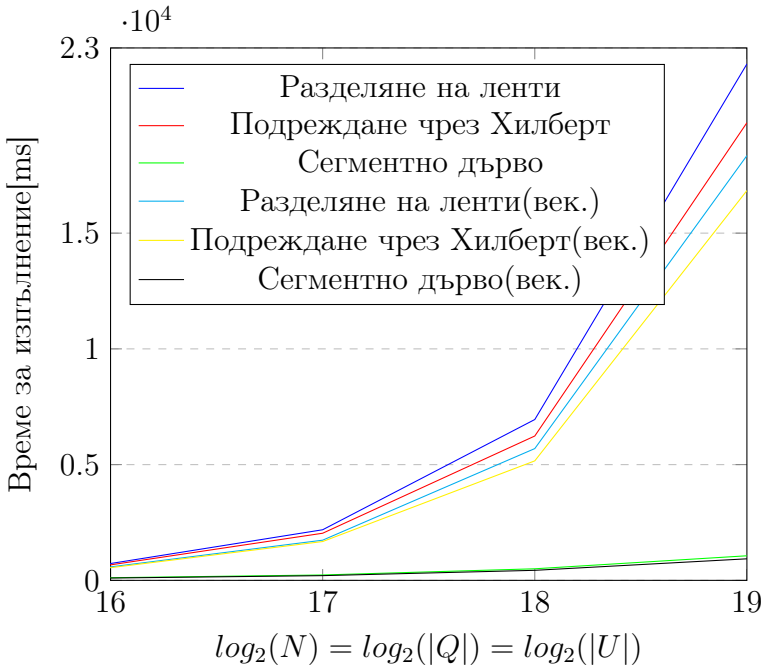
Нека да разгледаме други статистики като брой кеш-пропуски, които са важни при разглеждането на ефикасността на алгоритмите. На графика 7 са показани броят пропуски на L3 кеша за задачата, в която са включени обновления.



Фигура 7: Измерване на броя кеш-пропуски

Тук ясно се вижда причината за неочакваната скорост. Можем чрез асимптотични разсъждения да разгледаме броя кеш-пропуски. При сегментното дърво заради начина на разположение на паметта, процесорът не може да разбере следващото докосване в паметта, заради което почти на всяка итерация се получава такъв пропуск, следователно броят пропуски е от порядък $O(|Q| * logN + |U| * logN)$. Различното при сортирането чрез Хилберт е, че там заради последователността на докосваната памет можем да изчислим порядъка на броя пропуски като $O(\frac{d*|Q|\frac{(d-1)}{d}*N}{B})$, където B е броя на елементите, които се побират в една кеш линия. Комбинирайки тези разсъждения с ниската стойност на $d \leq 3$ в случая се получава това несъответствие между теоритичното време за изпълнение и практическото. Важно е да се отбележи, че този алгоритъм се представя ефективно дори без да "знае"за стойността на B. Заради това можем да заключим, че е кеш-безразличен(cache-oblivious[3]).

В езици като C++ съществува опция за компилиране на кода, възползвайки се от по-дългите регистри, наречена векторизация. Представена е графика 8, на която е сравнено времето за изпълнение на кодовете без и с тази опция. Тук отново заради последователността на паметта се наблюдава по-голямо забързване разделянето на ленти и сортирането чрез Хилберт($\approx 15\%$), отколкото при сегментното дърво($\approx 7\%$) .



Фигура 8: Измерване на времето при използвана и неизползвана векторизация

5 Хилбертообразни криви

5.1 Проблемът с оригиналната крива

Разглеждайки случая, в който размерът на оригиналния масив е малко по-голям от степен на двойката, $N = 2^k + \epsilon$, кривата ще бъде конструирана в решетка с размер на страната близък до $N' = 2^{k+1}$. Това е възможно място за подобрение - трябва да се конструират криви, чиито страни не са степени на двойките.

Дефиниция 1. *Хилбертообразна $H(N, d)$ крива ще наричаме път, минаващ през клетките на d-измерна хиперкубична решетка със страна N, която започва от клетка с координати $(0, \underbrace{0, \cdots, 0}_{d-1})$ и завършва в $(N-1, \underbrace{0, \cdots, 0}_{d-1})$. Начало ще наричаме съответно първата клетка, а край - последната.*

5.2 Решение на случая d=2

Теорема 1.1. *Ако съществуват $H(n, 2)$ и $H(m, 2)$, то можем да конструираме $H(nm, 2)$.*

Доказателство: *(Съответстващите фигури разглеждат примера $n = 3$, $m = 2$.)*

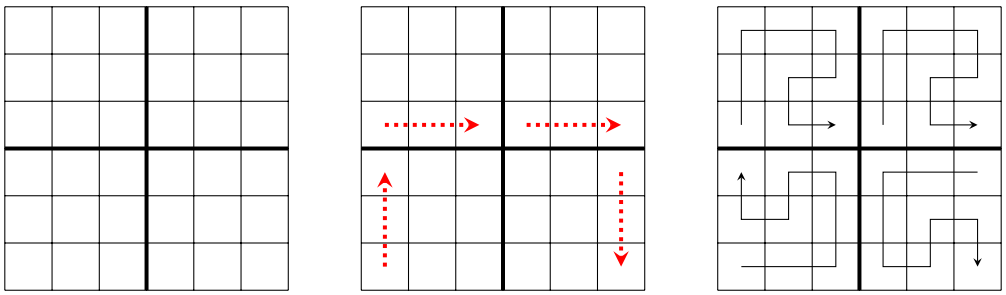
Ще разделим $nm \times nm$ решетката на непокриващи се $n \times n$ подрешетки като в 9а. Ще съставим път, който минава по всяка подрешетка последователно, следвайки пътя, образуван от $H(m, 2)$. Остава да определим ориентацията на пътя във всяка подрешетка, която ще бъде обиколена от $H(n, 2)$. За улеснение ще разглеждаме само началото и края на всяка такава. В доказателството обозначаваме горния ляв ъгъл и долния десен ъгъл като нечетни, а горния десен и долния ляв - четни. Ще представим конструктивно доказателството, разгледано в няколко стъпки. Конструкцията ще е такава, че ако от една подрешетка трябва да преминем в съседна, то краят на първата такава ще бъде съседен с началото на втората.

Стъпки:

1) Ще разгледаме ориентацията на първата подрешетка (това е долната лява такава). Ако след нея трябва да преминем вдясно, тя ще бъде ориентирана по начин 10а, ако не - по начин 10б. Така краят на подрешетката ще бъде съседен на следващата такава. Началото на първата подрешетка се пада в четна клетка, а краят в нечетна.

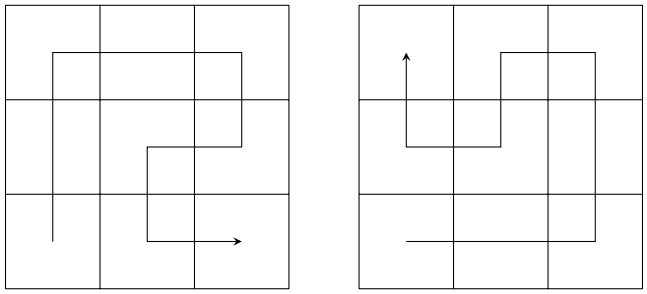
2) Ще ориентираме всички без последната подрешетка. Допускаме, че сме обходили първите $k-1$ и края на $(k-1)$-ата е нечетен и съседен на k-тата подрешетка в реда на обхождането. Началото на тази k-тата $H(n, 2)$ трябва да поставим в ъгъла, съседен на края на предходната подрешетка. Той ще бъде четен. Знаем, че от четно начало можем да ориентираме $H(n, 2)$, така че краят ѝ да е в който и да е от двата нечетни ъгъла, следователно е възможно да нагласим продължаването в следващата подрешетка в обхождането.

3) В края на конструкцията е нагласянето на последната подрешетка. От конструкцията на останалите подрешетки можем да заключим, че началото ѝ ще бъде четно. С това е ясно, че пътят в последната подрешетка може да се ориентира, така че краят на цялата $H(nm, 2)$ да е в долния десен ъгъл на решетката.



(а) (б) (в)

Фигура 9: Конструктивният процес на образуване на $H(6, 2)$



(а) (б)

Фигура 10: Случаи за първата подрешетка

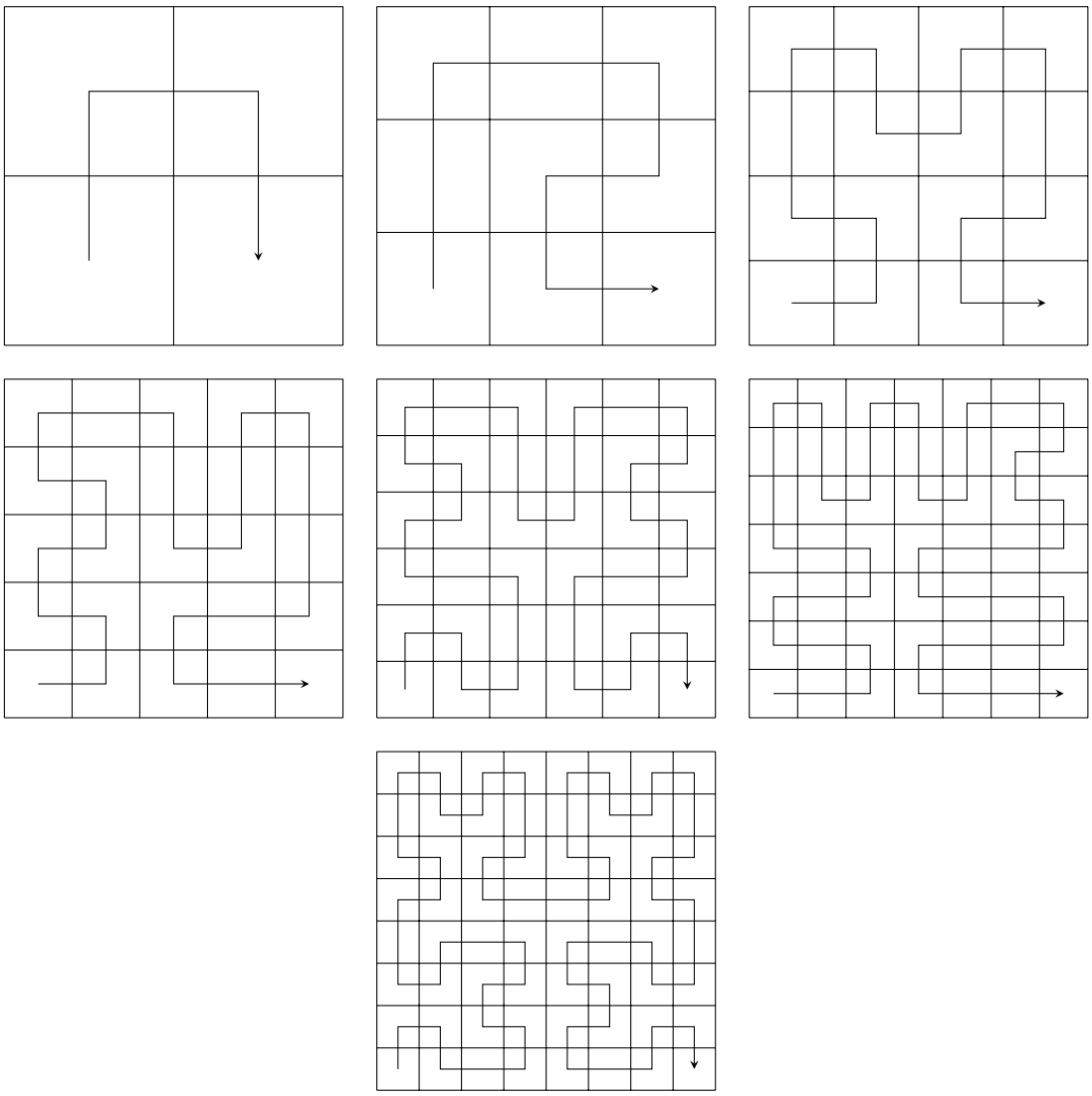
5.3 Пълно изчерпване за откриване на някои H(n, 2)

Предложен е алгоритъм, който е установен като експериментално ефективен за намиране на "добри"$H(n, 2)$. Ще изчислим фитнеса на една хилбертообразна крива като разгледаме набор от множества, съставени от случайни точки. След това всяко такова множество ще бъде подредено в реда, по който кривата обхожда точките в него, и ще намерим дължината на пътя, минаващ през всички такива, образуван чрез това обхождане. Псевдокод 5 отговаря на описанието на тази фитнес функция.

Algorithm 5: Определяне на фитнес на решение

Data: *curve* $H(n, 2)$, която трябва да се оцени
Result: $fitness$ - Число, определящо фитнес на кривата, колкото по-малко е то, толкова по-добре се справя кривата на практика

$fitness \leftarrow 0$;
for *constant number of iterations* **do**
    `// points е инициализирана със случаен брой координати на клетки в решетка` $N \times N$
    $points \leftarrow randompoints(n, n)$;
    `// Сортира точките в реда на обхождане от curve`
    $sort(points, curve)$;
    `// Добавя дължината на пътя образуван от точките`
    $fitness \leftarrow fitness + length(points)$
end



Фигура 11: $H(N, 2), N \leq 8$

За намирането на самите $H(n, 2)$ е използвано пълно dfs-изчерпване. Използвани са някои евристики, за да се забърза изпълнението на програмата. До тях е стигнато, разглеждайки естеството на $H(n, d)$, в частност ако кривата мине през клетката, която трябва да е последна преди обхождането на останалите, то тя няма как да се върне в нея в края. Друго невалидно продължение се случва, когато кривата не може да стигне до края без да мине през себе си. На фигура 11 са представени намерените криви за $3, 4, 5, 6, 7, 8$. Важно е да се отбележи, че кривите за $2, 4$ и 8 са точно кривите на Хилберт. Конструкцията на $H(6, 2)$ е подобна на такава получена чрез теоремата от 5.2, по-големи решетки могат да се получат прилагайки я неколкократно.

5.4 Комбиниране на методите

Чрез комбиниране на оптималните $H(n, 2)$ за $n < 9$, използвайки алгоритъма от теорема 1.1, можем да създадем $H(n', 2)$ за по-големи стойности на n. Направените изследвания, върху така получени $H(3 * 2^n, 2)$, $H(5 * 2^n, 2)$ и $H(7 * 2^n, 2)$, показват съответно 12%, 9% и 7% средно по-къси пътища в сравнение с разширяването на таблицата до такава със страна 2^k и използване на кривата на Хилберт в нея.

6 Други задачи, решими с метода

Можем да разгледаме задачата, в която въпросът се отнася единствено до елементите в интервала, чиято стойност е между x и y (x и y са специфични за всяка заявка). Тук черна кутия със сложност $O(1)$ може да бъде съставена и ще разгледаме минимален хамилтонов път между точките (l, r, u, x, y). Методът, описан в статията постига сложност $O(|Q|^{\frac{4}{5}} * max(N, |U|, A))$, където A е максималната стойност на x и y. Важно е да се отбележи, че чрез структури от данни като рядко сегментно дърво(sparse segment tree), можем да достигнем до решение със сложност $O(|Q| * log^3 N + |U| * log^3 N)$, но отново ще се наблюдава голяма константа при това решение. Вижда се приложимостта върху други задачи, в които съществува $O((|Q| + |U|) * log^k N)$, но с по-ниска константа.

Методът е приложим на много различни типове въпроси, за които не съществува полилогаритмично решение. Примери за такива са намиране на mex (функция важна в безпристрастните (impartial) игри), брой инверсии в интервала и други. Това, което го прави добър, е ниската константа, която може да се осигури заради последователността на докосваните елементи в паметта.

7 По-нататъчно развитие

7.1 Разширяване на метода

Тъй като в тази статия е разгледана черна кутия с равна сложност на добавяне и изваждане, то логично продължение е разглеждане на такава със сложност $O(A)$ за добавяне на елемент, $O(R)$ изваждане на елемент и $O(M)$ за модифициране на такъв вътре в нея (този тип операции не е разглеждан в настоящата статия, но би представлявал интерес за бъдещото развитие).

7.2 Различни криви

Някои структури от данни поддържат по-бързо едновременно добавяне на елементи или изваждане на такива, т.е. може да се разгледат различни криви, възползващи се от този факт.

7.3 Използване на по-добри апроксимиращи алгоритми в задачата за минимален хамилтонов път

Тъй като методът се опира на задачата за минимален манхатанов хамилтонов път, то като продължение може да се разгледа използването на други апроксимиращи алгоритми. Пример за такъв е 2-апроксимиращият, който използва минимално покриващо дърво в графа. Минимално покриващо дърво на граф, образуван измежду точки в двуизмерното пространство, има познато полиномиално решение, следователно е добра посока за разглеждане и сравнение.

7.4 Използване на машинно обучение за откриване на Хилбертообразни криви

Тъй като алгоритъмът за откриване на Хилбертообразни криви в двуизмерна решетка, предложен в тази статия, използва 20 минути за създаване на $H(8, 2)$, то следва, че методът не е приложим за по-големи решетки и измерения. Възможни посоки за откриване на $H(N, d)$ са пълно изчерпване с подреждане на съседните стейтове чрез евристика или невронна мрежа. Друг подход за задачата е използването на 2-opt или други генетични алгоритми, възползвайки се от функцията, определяща фитнеса на решението.

8 Заключение

В настоящата статия е разгледан начин, по който могат да бъдат решавани въпроси, свързани със заявки в масив от данни. Изложеният метод е около 20% по-бърз от най-добрия генерален познат такъв. Разгледано е също и представянето му срещу сегментното дърво и са открити причините, заради които порядъкът на отношенията е по-малък от очаквания. Въведени са Хилбертообразни криви, които помагат за справянето със случая $N \neq 2^k$ за $k \in \mathbb{Z}_+$. Представени са и разширения на метода за задачи, имащи познати решения със сложност $O(|Q| * log^k N)$, които могат да се решат чрез криви на Хилберт от по-високи измерения. Разгледани са начини за изчисляване на $H(N, 2)$ за N.

9 Забележки по реализацията на проекта

Статията е писана в онлайн средата за разработка на latex Overleaf. Кодът, използван за сравнения на алгоритмите, е писан на C++ и компилиран с g++. За измерването на характеристики на изпълнението на методите е използвана програмата perf-stat. Всеки метод бива пуснат няколко (10) пъти за всички различни стойности на параметрите и е взето средното аритметично от изпълненията. Написани са и набор от помощни програми като тази, генерираща latex код за изображенията на кривите, и такава, която намира Хилбертообразни криви. Всеки код, описан в статията, е авторски и е публикуван в github страницата на проекта. Машината, на която са извършени всички измервания, е с процесор Intel(R) Core(TM) i7-6500U CPU @ 2.50Ghz, архитектурата му е x86_64, размерите на кешовете му са съответно L1i-64KiB, L1d-64KiB, L2-512KiB и L3-4MiB.

10 Благодарности

Благодаря на научния си ръководител Радослав Димитров, както за помощта с избирането на тема, така и заради факта, че винаги беше готов да помогне и да поднесе начин как проектът да бъде подобрен, на Иван Тончев за вдъхновението да се занимавам с компютърните науки и на Цветелина Илиева за пробния прочит.

Литература

[1] Hui Liu et al. “Encoding and Decoding Algorithms for Arbitrary Dimensional Hilbert Order”. B: (2016). URL: arxiv.org/abs/1601.01274.

[2] “An alternative sorting order for Mo’s algorithm”. B: (2018). URL: codeforces.com/blog/entry/61203.

[3] Harald Prokop. “Cache-Oblivious Algorithms”. Massachusetts Institute of Technology, 1999. URL: supertech.csail.mit.edu/papers/Prokop99.pdf.

[4] “The stabbing problem”. B: (1999). URL: cs.nthu.edu.tw/~wkhon/algo08-tutorials/tutorial-stabbing.pdf.