باسمه تعالی

پروژه پایانی

درس ساختمان داده‌ها

مدرس: دکتر احمدعلی آبین

همکاران این شماره: امیرحسین صدر

بهار ۱۴۰۴

۱ پیش مطالعه

یکی از مسائل مهم مطرح شده در جهان، از ابتدا تاکنون، مساله **بهینه‌سازی** می‌باشد. این مساله از آن جهت حائز اهمیت است که می‌توان بسیاری از مسائل علوم دیگر از جمله مسائل دنیای واقعی را مبتنی بر آن مدل سازی کرد و سپس اقدام به حل آنها نمود.

سیستم توصیه‌گر را سیستمی می‌نامیم که ورودی آن تعداد بسیار زیادی نمونه از یک جنس (مثلا کتاب!) با ویژگی‌ها مختلف و خروجی آن نمونه یا نمونه‌هایی است که براساس ارزش یا ارزش‌های کاربر (مثلا قیمت کمتر!)، نزدیک‌ترین به سلیقه او می‌باشد.

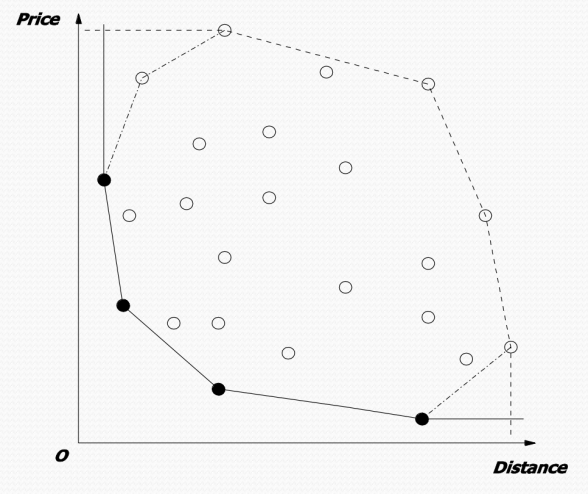
فرض کنید یک سامانه رزرو هتل طراحی کرده‌ایم و می‌خواهیم با هر بار باز شدن سامانه، براساس ارزش‌هایی که کاربر از قبل آنها را مشخص کرده یا مبتنی بر اطلاعات (رزروهای) گذشته که آنها را به خاطر داریم، بهترین گزینه‌های موجود برای رزرو را به او پیشنهاد بدهیم. در این صورت نیاز به یک سیستم توصیه‌گر داریم.

حال سوال اینجاست که برمبنای چه مکانیزم و توابعی خروجی سیستم توصیه‌گر آماده می‌شود. یک روش برای این منظور حل مساله بهینه‌سازی می‌باشد. بدین صورت که تمام داده‌های ورودی سیستم را در قالب یک ساختار مشخص، مثلا پایگاه داده ذخیره می‌کنیم. همانطور که می‌دانید یک سیستم پایگاه داده شامل جداولی از داده است به طوری که هر سطر از جدول را یک رکورد (Record) می‌گوییم. هر رکورد دارای مقادیر مختلفی به ازای ستون‌های مختلف جدول می‌باشد به طوری که هر ستون یک پارامتر یا ویژگی (Feature)، یا فیلد (Field) یا بعد (Dimension)، یا ابژه (Objective) را نمایندگی می‌کند. در کاربردهای واقعی مانند سیستم توصیه‌گر رکوردهای پایگاه داده به مرور اضافه و تکمیل می‌گردند. فراهم بودن داده مناسب از الزامات اولیه عملکرد صحیح سیستم توصیه‌گر می‌باشد.

اگر مساله داده را حل شده فرض کنیم، باید برمبنای یک (Single-Objective) یا چند (Multi-Objective) پارامتر، بهینه‌سازی را انجام داد. مجددا سامانه رزرو هتل را در نظر بگیرید. فرض کنید اطلاعات تعداد قابل توجهی هتل در قالب ریکوردهای مختلف شامل مقادیر متفاوت به ازای چندین پارامتر در مجموعه داده (پایگاه داده) موجود است و کاربر می‌خواهد برمبنای دو ویژگی -نزدیکی به دریا- و -قیمت- هتل مدنظر خود را برای رزرو پیدا کند. در واقع این دو ویژگی بیانگر ارزش‌های کاربر برای انتخاب هستند. بنابراین باید مساله بهینه‌سازی را بر مبنای این دو پارامتر حل کرد. اما چگونه؟

شکل ۱ نقاط مختلف موجود داده را نشان می‌دهد. محور x بیانگر مقدار بعد نزدیکی به دریا و محور y بیانگر مقدار بعد قیمت است.

این نکته در حل این مساله حائز اهمیت است که ما نیازمند یک یا چند ساختمان داده برای ذخیره‌سازی داده‌های مورد نیازمان هستیم تا سپس بتوانیم الگوریتم بهینه‌سازی خود را بر روی/به کمک آنها اجرا کنیم. مثلا اگر بخواهیم مبتنی بر روش‌های مرتب‌سازی این کار را انجام دهیم نیازمند یک یا چند آرایه هستیم (البته دقت شود در پایگاه‌داده‌ها ساختار ذخیره‌سازی به صورت یکپارچه و مشخص می‌باشد – در ادامه مشخص می‌شود که داده‌ها به صورت فایل و متنی موجود هستند و بنابراین باید ساختمان داده مناسبی برای کار با داده‌ها انتخاب و یا طراحی شود).



شکل ۱: مدلسازی مجموعه داده بر اساس دو ویژگی قیمت و فاصله از دریا (نقاط مشکی نقاط skyline هستند)

مسائل بهینه‌سازی با توجه به جنس مساله و راه حلشان به دسته‌های مختلفی تقسیم‌بندی می‌شود. در یک فضای نقاط (داده) D بعدی (D = 2 مطابق شکل ۱)، به نقاطی Skyline می‌گوییم که نسبت به سایر نقاط، حداقل در یک بعد مقدارشان از مقدار متناظر در همان بعد بدتر نباشد. با توجه به تعریف می‌توان گفت ما با دسته‌ای از مسائل تحت عنوان Skyline Computation سروکار داریم. در واقع اگر ما نقاط یا داده‌های Skyline را در مجموعه داده‌مان پیدا کنیم همان بهینه است که خروجی سیستم می‌باشد.

فرض کنید مجموعه نقاط T با N بعد وجود دارد. مقدار t_1 باید حداقل با t_2 در $N - 1$ بعد برابر و در یک بعد بهتر از t_2 باشد تا بگوییم t_1، t_2 را Dominate می‌کند. می‌توان نقاط Skyline را این گونه تعریف کرد: نقاطی هستند که توسط هیچ یک از نقاط دیگر Dominate نمی‌شوند.

Algorithm • `dominates()` in reference

```
1: procedure dominates(dataset, a: index, b: index)
2:   flag ← 0;
3:   for d = 0 to dimensions − 1 do
4:     if dataset[a][d] > dataset[b][d] then
5:       return 0
6:     else if dataset[a][d] < dataset[b][d] then
7:       flag ← 1;
8:   return flag
```

الگوریتم ۰: Domination

دو نکته در حل این مساله حائز اهمیت است. نکته اول این است که بهتر و بدتر براساس جنس ویژگی تعریف می‌شود. مثلا اگر ویژگی مورد بررسی کیفیت باشد آنوقت هر چه کیفیت بیشتر باشد بهتر است. اما در خصوص ویژگی قیمت برعکس می‌باشد. نکته دوم این است که سیستم مورد بررسی ما static نیست. و در Step های زمانی مختلف ممکن است داده‌هایی از مجموعه داده ما خارج شوند و یا به آن اضافه شوند (Continous Skyline). مثلا سیستم رزرو هتل نیز چنین ویژگی را داراست.

در این پروژه از شما می‌خواهیم یک راه حل بهینه و SOTA برای این مساله را که در این مقاله ارائه شده را پیاده‌سازی کنید. در این مقاله یک ساختمان داده (درخت) جدید به همراه الگوریتم‌های آن تحت عنوان BJR-Tree معرفی شده است. راه حل (الگوریتم) مذکور در بخش ۳ و ۴ مقاله تشریح شده. در بخش‌های بعدی پروژه این الگوریتم (به ترتیب معادل بخش ۳.۱، ۳.۲ – ۳.۳ و ۴ از مقاله) به صورت خلاصه تشریح شده است. در بخش ۵ پروژه که دارای نمره امتیازی بسیار بالایی می‌باشد از شما می‌خواهیم در صورتی که راه حلی نوآورانه با مرتبه‌زمانی کمتر از این الگوریتم برای این مساله در نظر دارید معرفی نمایید. دقت شود برای فهم بهتر مساله و راه حل توصیه می‌شود مقاله به صورت دقیق‌تر خوانده شود. همچنین مجموعه داده مورد نیاز در این لینک قرار گرفته است و قابل دانلود می‌باشد. ویژگی‌های این مجموعه داده در جدول ۱ آماده است.

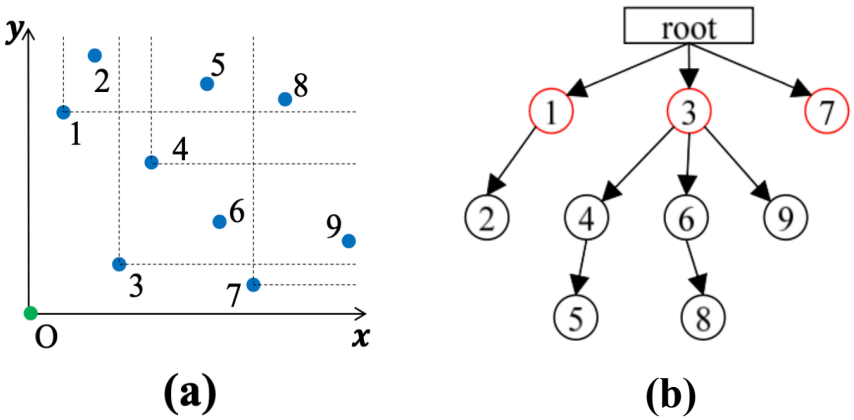
TABLE I. TEST DATASET CHARACTERISTICS

Name	Entries	Dimensions	Time Steps	Max. Skyline Elem.	Appx. Input Size	Appx. Output Size
small	1,000	4	109	34	8 KB	15 KB
medium	50,000	5	10,998	310	500 KB	14 MB
large	800,000	7	40,998	2008	11 MB	329 MB

جدول ۱: مشخصات مجموعه داده

۲ صورت پروژه (بخش اول - اجباری)

در این بخش قصد داریم بخش ۳.۱ از مقاله که به معرفی الگوریتم‌های Injection و Ejection (که الگوریتم Injection بالانس درخت را حفظ نمی‌کند) درخت می‌پردازد را پیاده سازی کنیم (در واقع حرف B در نام الگوریتم به بحث Balancing اشاره دارد که در این بخش مورد بررسی نیست).



شکل ۲: نمای کلی درخت متناظر (b) با فضای دو بعدی (a)

فرزندان ریشه درخت نقاط Skyline می‌باشند. موقعیت سایر نقاط در درخت از شکل ۲ قابل استنباط است. تاکنون به صورت شهودی نحوه ساخت درخت را دیدیم. در ادامه الگوریتم Injection و Ejection به صورت دقیق آورده شده اند.

Algorithm 1 New node injection

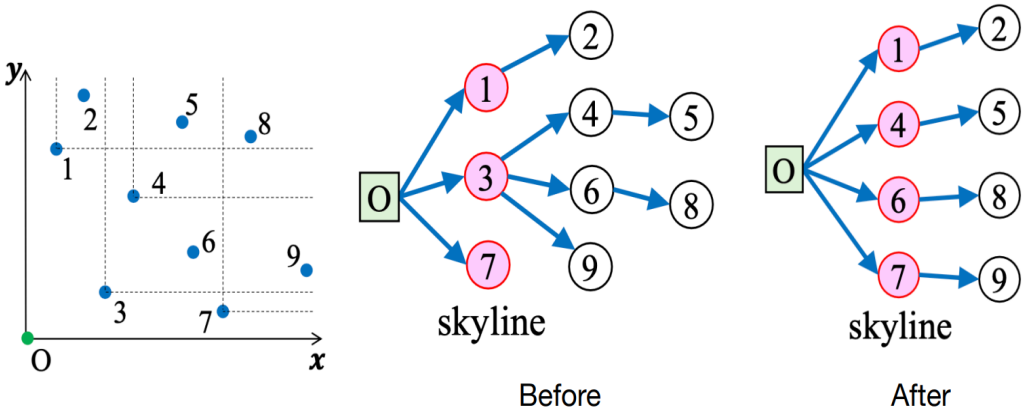
```
1:  procedure inject(parent, e: new node)
2:  children ← children of parent;
3:  for all c ∈ children do
4:      if dominates(c, e) then
5:          inject(c, e);
6:          return
7:  add e to parent as a child;
8:  for all c ∈ children do
9:      if dominates(e, c) then
10:         move c to e as a child;
```

الگوریتم ۲: الگوریتم Injection نود به درخت

Algorithm 2 Node ejection

```
1:  procedure eject(e: node)
2:  parent ← parent of e;
3:  children ← children of e;
4:  remove e from parent;
5:  for all c ∈ children do
6:      inject(parent, c);
```

الگوریتم ۳: الگوریتم Ejection نود از درخت



شکل ۳: مثالی از Ejeciton

در شکل ۳ مثالی از Ejection (نود ۳) آورده شده است.

۳ صورت پروژه (بخش دوم - اجباری)

در این بخش می‌خواهیم بالانس درخت را هنگام injection به کمک تکنیک Lazy Evaluation حفظ نماییم. به الگوریتم ۳ توجه نمایید. در شکل ۴ تاثیر تکنیک مذکور به وضوح قابل مشاهده است.

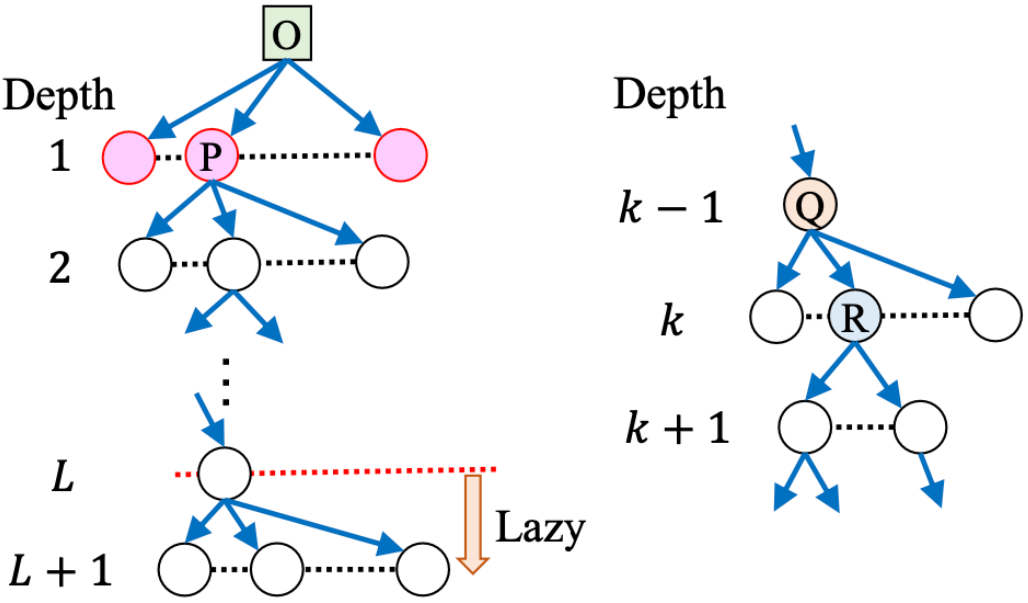
Algorithm 3 Injection (lazy evaluation and tree balancing)

```
1: d: the depth at which a vertex is injected lazily
2: procedure inject(r: root, v: new vertex)
3: if r = O or Depth(r) < d then
4:     C ← children of r;
5:     g ← +∞;
6:     t ← null;
7:     for all c ∈ C do
8:         if c dominates v and Desc(c) < g then
9:             t ← c;
10:            g ← Desc(c);
11:    if t ≠ null then
12:        inject(t, v);
13:        return
14: set v to r's child;
15: if r = O or Depth(r) < d then
16:    for all c ∈ C do
17:        if v dominates c then
18:            move c to v's child;
```

الگوریتم ۳: الگوریتم injection به همراه Lazy Evaluation

٤ صورت پروژه (بخش سوم - امتیازی)

هدف از این بخش پیاده‌سازی بخش ۴ مقاله و تکنیک ND-Cache است که ۳۰ درصد نمره امتیازی به همراه دارد. مطالعه جزئیات این بخش برعهده علاقه‌مندان است.



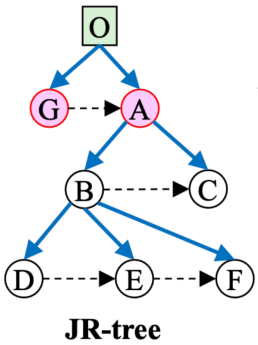
شکل ۴: تاثیر تکنیک Lazy Evaluation

۵ صورت پروژه (بخش چهارم – سوپرامتیازی)

هدف این بخش ارائه الگوریتمی برای Injection و Ejection است که به لحاظ مرتبه زمانی بهتر از مرتبه زمانی BJR-Tree که در بخش ۵ مقاله آمده است باشد. الگوریتم‌ها باید در گزارش آورده شوند و مرتبه زمانی ادعا شده اثبات شود. پیاده‌سازی نیاز نیست.

۶ نکات تکمیلی

- پیاده‌سازی فقط به زبان C و یا C++ مجاز می‌باشد.
- بخشی از نمره پروژه به انجام تست صحیح بر روی مجموعه داده‌های ارائه شده است. مکانیزم تست باید شفاف باشد.
- نحوه ذخیره‌سازی داده‌های مساله در حافظه می‌تواند در نوع خود چالش برانگیز باشد و مورد سوال و ارزیابی قرار خواهد گرفت. یک روش در شکل ۵ آمده است.



Contents of tree-structure memory

Address	Parent	Next	Head child
A	Root	Null	B
B	A	C	D
C	A	Null	Null
D	B	E	Null
E	B	F	Null
F	B	Null	Null
G	Root	A	Null

شکل ۵

۷ نکات مربوط به ارسال پاسخ

۱. هر گونه سؤال از پروژه را در گروه دیسکاشن و به صورت عمومی مطرح کنید تا برابری برقرار باشد.

۲. پروژه باید در کوئرا به این صورت آپلود شود که یک فایل zip شامل گزارش و یک پوشه شامل فایل‌های کد باشد. نیازی به قرار دادن دیتاست‌ها در فایل نیست.

۲. در صورت مشاهده هر گونه تقلب، نمره صفر برای پروژه در نظر گرفته خواهد شد.

۳. فرمت نامگذاری پروژه به صورت DS-FP-[Student ID]-[Student Name] باشد.

موفق باشید