

# گزارش تمرین سری دوم درس برنامه نویسی پیشرفته

محمد صالح گشانی - شماره دانشجویی : ۹۴۲۳۰۹۲

۱۷ اسفند ۱۳۹۷

در کد های ارسالی سعی شد تا جای ممکن با کامنت گذاری بخش هایی از کد که ممکن است نا مفهوم باشد ، توضیح داده شده شود .

## ۱ سوال اول

این سوال شامل سه بخش است که در هر سه بخش باید به صورت تصادفی تعدادی عدد بین ۰ تا ۱۰۰ را تولید کرده و از آن ها استفاده می کردیم برای این کار لازم بود دو کتابخانه ی `stdlib.h` و `time.h` را در برنامه ی خود `include` کنیم . سپس در تابع خود باید دستور `(rand(unsigned time(NULL))` را وارد می کردیم که این دستور باعث می شود که در هر بار اجرای برنامه اعداد تصادفی متفاوتی تولید شود . در آخر هم برای تولید اعداد تصادفی باید مقدار `rand () % ۱۰۱` را در متغیر مورد نظر قرار می دادیم .

همچنین در هر سه بخش از تابع `destructor` نوشته شد که در این تابع متغیر هایی که `new` شده بودند `delete` شدند .

در ادامه به هر سه بخش ، جداگانه به طور مفصل پرداخته خواهد شد .

### ۱.۱ قسمت الف

در این بخش ابتدا کلاس به نام `map` را تعریف کردیم و چهار متغیر را برای آن تعریف کردیم : ۱ - متغیر `n` که این متغیر برای ساخت نقشه ی  $n \times n$  استفاده می شود ۲ - متغیر `obstacle` که این متغیر در واقع همان ارتفاع هر ناحیه است که با توجه به اینکه نقشه ی ما دو بعدی است یک آرایه ی دینامیک دو بعدی با استفاده از `pointer to obstacle` برای استفاده کردیم ۳ - متغیر `route` که از این متغیر هنگام رسم در خروجی (برای نمایش مسیر از ابتدا به انتها) استفاده شد به این ترتیب که اگر مسیر ما از یک ناحیه رد می شد `route[i][j]` مربوط به آن ناحیه برابر با ۱ و در غیر اینصورت برابر با ۰ قرار داده می شد . این متغیر هم همانند متغیر `obstacle` توسط آرایه دینامیک دو بعدی ساخته شد ۴ - متغیر `list` که از این متغیر برای محاسبه ی فاصله ی بین ابتدا و انتها استفاده شد به این ترتیب که در هر مرحله که ناحیه ی بعدی انتخاب می شد ارتفاع آن ناحیه را درون این متغیر ریخته و با ارتفاع ناحیه فعلی که در همین متغیر ذخیره شده است مقایسه کرده و اختلاف را به فاصله ی فعلی اضافه می کردیم . این متغیر هم با استفاده از آرایه دینامیک ساخته شد با این تفاوت که این متغیر تک بعدی است و از `pointer` استفاده شد .

در ادامه در تابع `constructor` این برنامه که تنها متغیر `n` را به عنوان ورودی می گرفت متغیر های `obstacle` و `route` و `list` را با استفاده از دستور `new` ساختار دهی و سپس با استفاده از حلقه های `for` مقدار دهی اولیه کردیم . سپس تابع `showMap` را با استفاده از دو حلقه `for` تو در تو نوشتیم . در تابع `findRoute` با توجه به این که فقط امکان پایین و یا راست آمدن را داشتیم تعداد `n-۱` عدد عمل پایین آمدن و تعداد `n-۱` عدد عمل راست آمدن نیاز بود

پس یک حلقه for به شرط  $i < 2(n - 1)$  نوشتیم. در این حلقه فاصله ی هر ناحیه را با ناحیه های پایین و راست آن مقایسه کرده و هر کدام که اختلاف کمتری با ناحیه ی فعلی داشتند را به عنوان ناحیه ی بعد انتخاب کردیم و با اطمینان حاصل کردن از این که به گوشه ی پایین و یا گوشه ی سمت راست نرسیده ایم (در صورت این که یکی از دو حالت گفته شده پیش می آمد تنها به طور مستقیم مسیر را ادامه داده و دیگر فاصله با همسایه ها را چک نمی کردیم) به مرحله ی بعد می رفتیم. در آخر هم در تابع showRoute با استفاده از متغیر route مسیر نهایی را نمایش می دادیم.

در شکل ۱ یک نمونه از برنامه برای  $n = 5$  قابل مشاهده است

[illegible]

شکل ۱: نمونه حل سوال ۱ قسمت الف

## ۲.۱ قسمت ب

این بخش تماماً شبیه به بخش الف بود و تنها تفاوت آن در تابع `findRoute` بود که در این تابع علاوه بر فاصله ی ناحیه فعلی با همسایه های پایین و راستی ، فاصله ی این ناحیه با ناحیه ی چپ پایین هم محاسبه شده و در حلقه ی مربوط این فاصله هم منظور می شد . همچنین در این حلقه یک شرط هم وجود داشت که اگر زودتر از  $2(n - 1)$  تکرار با مقصد رسیدیم دیگر الگوریتم را ادامه ندهد منطق این شرط هم در اینجاست که حرکت قطری همانند ترکیب یک حرکت به سمت پایین و یک حرکت به سمت راست عمل می کند و باعث می شود که ما زودتر از  $2(n - 1)$  تکرار به مقصد برسیم .

در شکل ۲ یک نمونه از برنامه برای  $n = 6$  قابل مشاهده است

### ۳.۱ قسمت ج

این بخش اما کمی با دو بخش قبل تفاوت داشت. در این بخش در تعریف کلاس map دو متغیر جدید num و alt علاوه بر متغیرهای اشاره شده در دو بخش قبل تعریف شدند. متغیر num برای ذخیره تعداد حالت های مختلفی که می شد با پایین و راست رفتن از ابتدا به انتها رسید استفاده شد. متغیر alt هم که از نوع char\*\* بود به عنوان آرایه ی دینامیک برای ذخیره حالت های مختلف استفاده شد به عنوان مثال در حالت n = 5 یک از حالت های قابل قبول به صورت "rrdrddrd" و یک حالت قابل قبول دیگر به صورت "drdrddrr" است که تمامی این حالت ها در آرایه دینامیک alt ذخیره شده و در تابع findRoute از آنها برای تشخیص کوتاه ترین مسیر استفاده شد. همچنین در این بخش یک تابع به نام fact تعریف شد که برای محاسبه ی فاکتوریل از آن استفاده می شد که در مقدار دهی به متغیر num به آن نیاز بود.

```

C:\Windows\system32\cmd.exe
15 15 95 32 98 66
37 64 44 17 6 40
32 98 2 68 57 39
36 89 84 3 76 22
58 4 32 3 82 80
15 59 28 52 97 59
The Distance is : 86
*
*
*
*
*
*
Press any key to continue . . .

```

شکل ۲: نمونه حل سوال ۱ قسمت ب

در constructor کلاس در این بخش همانند بخش الف می شد. سپس همانند بخش الف متغیرهای obstacle و ... ساختار دهی و مقدار دهی اولیه شدند. در ادامه با محاسبه ی متغیر num متغیر alt هم ساختار دهی شد و سراغ مقدار دهی آن رفتیم. در مرحله مقدار دهی باید تمام جایگشت های ممکن r و d را می ساختیم به همین منظور جایگشت ها را به دو بخش تقسیم کرده که یکی از "rrr...ddd..." شروع کرده و نصف دیگر از "ddd...rrr..." شروع می کرد. در ادامه برای مثال در نیمه ی اول r اول را در هر تکرار با یکی از d ها عوض می کردیم تا جایگشت جدیدی ساخته شود سپس زمانی که r اول با تمام d ها جا به جا شد حال همین کار را با r دوم انجام می دادیم تا به آخرین r برسیم و سپس همین روند را با نیمه ی دوم d ها طی می کردیم. در تابع findRoute در یک حلقه با توجه به جایگشت های موجود در متغیر alt (اگر در alt حرف r را دیده پایین رفته و اگر حرف d را دیده راست می رفتیم) مسیر های ممکن را طی کرده و به همان روش بخش الف و ب فاصله ها را محاسبه کرده و در یک آرایه ی دینامیک به نام D فاصله ی هر جایگشت را ذخیره کردیم. سپس در یک حلقه ی for کوتاه ترین فاصله را پیدا کرده و شماره آن را ذخیره کردیم. در آخر هم در یک حلقه با توجه به کوتاه ترین مسیر خانه های متغیر route را ۰ و ۱ مقدار دهی کردیم. در شکل ۳ یک نمونه از برنامه برای  $n = 4$  قابل مشاهده است.

## ۲ سوال دوم

برای حل این سوال ابتدا در دو دسته فایل های مجزا به نام های Vec.h ، Vec.cpp و Arr.h ، Arr.cpp دو کلاس به نام های libVec و libArr را ساخته و در هر کدام از این کلاس ها یک تابع به نام counter که ورودی عدد طبیعی N را گرفته و در خروجی مجموع اعداد ۰ تا  $N - 1$  را می دهد، تعریف کردیم. در کلاس libVec تابع counter از vector استفاده کرده و در کلاس libArr این تابع از آرایه ی دینامیک استفاده می کرد. در تابع main در یک تابع به نام runTime زمان اجرای هر کدام از این توابع را به ازای N های مختلف اندازه گیری کردیم. به این صورت که با استفاده از template ورودی تابع runTime را یک شی از کلاسی دلخواه قرار داده و در این تابع، تابع counter شی مورد نظر را روی عدد N (که آن را هم در ورودی تابع runTime گرفته ایم) اجرا کرده و زمان اجرای آن تابع را اندازه گیری می کردیم. برای این کار از کتابخانه ی ctime و دستور

```

C:\Windows\system32\cmd.exe
47  40  44  35
47  61  10  34
43  43  57  69
43  76  42  33

The distance for each route is listed below :
1. 72
2. 120
3. 120
4. 118
5. 154
6. 154
7. 152
8. 140
9. 140
10. 138
11. 82
12. 44
13. 46
14. 110
15. 72
16. 74
17. 124
18. 85
19. 88
20. 82

The shortest distance is : 44

*
*
*
Press any key to continue . . .

```

شکل ۳: نمونه حل سوال ۱ قسمت ج

`std::clock()` استفاده کردیم . به این صورت که ابتدا دستور `std::clock()` را اجرا کرده و زمان شروع را در یک متغیر ذخیره می کردیم سپس تابع `counter` کلاس مورد نظر را با ورودی  $N$  اجرا کرده و در نهایت با اجرای دوباره ی دستور `std::clock()` و کم کردن زمان بین دوبار اجرای این دستور زمان تقریبی اجرای تابع `counter` در کلاس مورد نظر را اندازه گیری کردیم .

ذکر این نکته لازم است که در صورت سوال خواسته شده بود که سوال با استفاده از `pointer to member function` حل شود ولی طی مکاتبه با استاد ایشان گفتند که روشی که در این سوال پیش گرفتیم (فرستادن یک شی از کلاس به تابع `runTime` و اجرا کردن تابع `counter` از طریق شی) نیز برای حل این سوال قابل قبول است .

در شکل ۴ و ۵ نمونه هایی از اجرای برنامه قابل مشاهده است . (دلیل قرار دادن ۲ شکل این است که زمان اجرا در محیط لینوکس دقیق تر نمایش داده شد و همچنین در محیط لینوکس زمان اجرای تابع هر دو کلاس کمتر از زمان اجرا در ویندوز بود)

```

C:\Windows\system32\cmd.exe
1000 clocks per second
The time needed to execute counter in libArr for N = 10 is : 0 milliseconds
The time needed to execute counter in libVec for N = 10 is : 0 milliseconds

The time needed to execute counter in libArr for N = 100 is : 0 milliseconds
The time needed to execute counter in libVec for N = 100 is : 0 milliseconds

The time needed to execute counter in libArr for N = 1000 is : 0 milliseconds
The time needed to execute counter in libVec for N = 1000 is : 0 milliseconds

The time needed to execute counter in libArr for N = 10000 is : 0 milliseconds
The time needed to execute counter in libVec for N = 10000 is : 8 milliseconds

The time needed to execute counter in libArr for N = 100000 is : 1 milliseconds
The time needed to execute counter in libVec for N = 100000 is : 87 milliseconds

The time needed to execute counter in libArr for N = 1000000 is : 4 milliseconds
The time needed to execute counter in libVec for N = 1000000 is : 799 milliseconds

The time needed to execute counter in libArr for N = 10000000 is : 43 milliseconds
The time needed to execute counter in libVec for N = 10000000 is : 8144 milliseconds

Press any key to continue . . .

```

شکل ۴: نمونه ی حل سوال ۲ در محیط ویندوز

```
garch@ubuntu: ~/Desktop/HW2/2
File Edit View Search Terminal Help
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$ make
make: 'main' is up to date.
garch@ubuntu:~/Desktop/HW2/2$ make clean
rm -f main main.o Arr.o Vec.o
garch@ubuntu:~/Desktop/HW2/2$ make
g++ -std=c++17 -Wall -c main.cpp
g++ -std=c++17 -Wall -c Arr.cpp
g++ -std=c++17 -Wall -c Vec.cpp
g++ -std=c++17 main.o Arr.o Vec.o -o main
garch@ubuntu:~/Desktop/HW2/2$ ./main
1000000 clocks per second
The time needed to execute counter in libArr for N = 10000000 is : 44.415 milise
conds
The time needed to execute counter in libVec for N = 10000000 is : 217.372 milis
econds
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$
garch@ubuntu:~/Desktop/HW2/2$
```

شکل ۵: نمونه ی حل سوال ۲ در محیط لینوکس

### ۳ سوال سوم

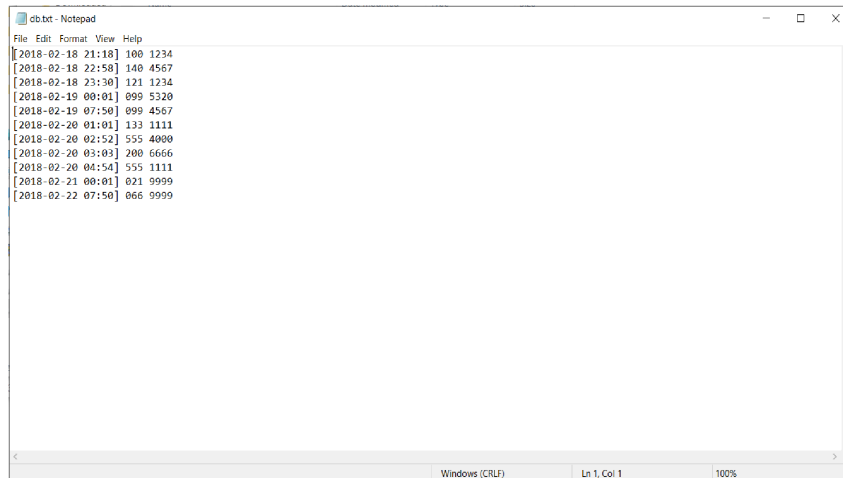
برای حل این سوال تابعی به نام database نوشتیم و در این تابع استخراج اطلاعات از پایگاه داده ی قبلی را انجام داده و پارامترهای پایگاه داده جدید را محاسبه کردیم . روند کار به این صورت بود که در تابع اصلی پس از خواندن اطلاعات از روی فایل ، اندازه ی هر ورودی و نقاط مختلف در هر ورودی که اعداد مختلف (تاریخ و شماره ی محصول و خریدار) شروع می شدند را استخراج کرده و سپس تمامی این اطلاعات را به همراه ورودی و تعدادی آرایه برداری را به صورت reference به تابع دادیم .

در تابع database ابتدا تاریخ ها و شماره های مختلف را در بردارهای جداگانه ریخته و سپس با یک حلقه for تعداد تاریخ های مختلف را به دست آوردیم . سپس در حلقه ی آخر برنامه که بخش اصلی کد در آنجا بود به استخراج شماره ها پرداختیم . روش این کار به این صورت بود که در دو حلقه for تو در تو با توجه به تعداد خرید های هر تاریخ در بردارهایی که که شماره های محصولات و خریداران را ذخیره کرده بودیم به جستجو پرداخته و تعداد شماره های متمایز را استخراج کردیم . با توجه به این که تعداد ورودی ها برای هر تاریخ متفاوت بود و همچنین تعداد شماره های محصولات خریداری شده و یا خریداران نیز تفاوت می کرد لازم شد که از بردارهای دو بعدی استفاده کنیم تا بر حسب نیاز با استفاده از دستور pushback() طول بردار را افزایش بدهیم و لازم نباشد که تعداد ستون های هر سطر ماتریس ساخته شده با هم برابر باشد . در یک بردار دو بعدی نحوه ی استفاده از دستور pushback به این صورت است که ابتدا در متغیر اصلی یک بردار تک بعدی را pushback کرده سپس در هر یک از این بردارهای تک بعدی بر حسب نیاز اطلاعات مورد نیاز را به طور معمول (در آرایه های تک بعدی) pushback می کردیم .

در نهایت هم با استفاده از تابع size() روی بردارهایی که به صورت reference به تابع database فرستاده بودیم اطلاعات لازم را در فایل خروجی نوشتیم .

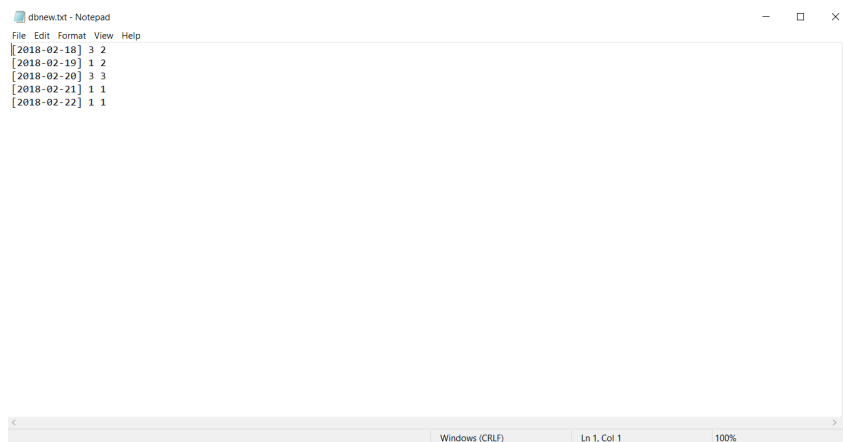
ذکر این نکته لازم است که با توجه به این که فایل db.txt در فایل تمرین قرار داده نشده بود یک فایل db.txt دلخواه را ساخته و عملیات رو روی آن انجام دادیم .

در در شکل ۶ و ۷ می توانید یک نمونه فایل db.txt ساخته شده و خروجی برنامه فایل dbnew.txt را مشاهده کنید .



```
File Edit Format View Help
[2018-02-18 21:18] 100 1234
[2018-02-18 22:58] 140 4567
[2018-02-18 23:30] 121 1234
[2018-02-19 00:01] 099 5320
[2018-02-19 07:50] 099 4567
[2018-02-20 01:01] 133 1111
[2018-02-20 02:52] 555 4000
[2018-02-20 03:03] 200 6666
[2018-02-20 04:54] 555 1111
[2018-02-21 00:01] 021 9999
[2018-02-22 07:50] 066 9999
Windows (CRLF) Ln 1, Col 1 100%
```

شکل ۶: فایل db ساخته شده



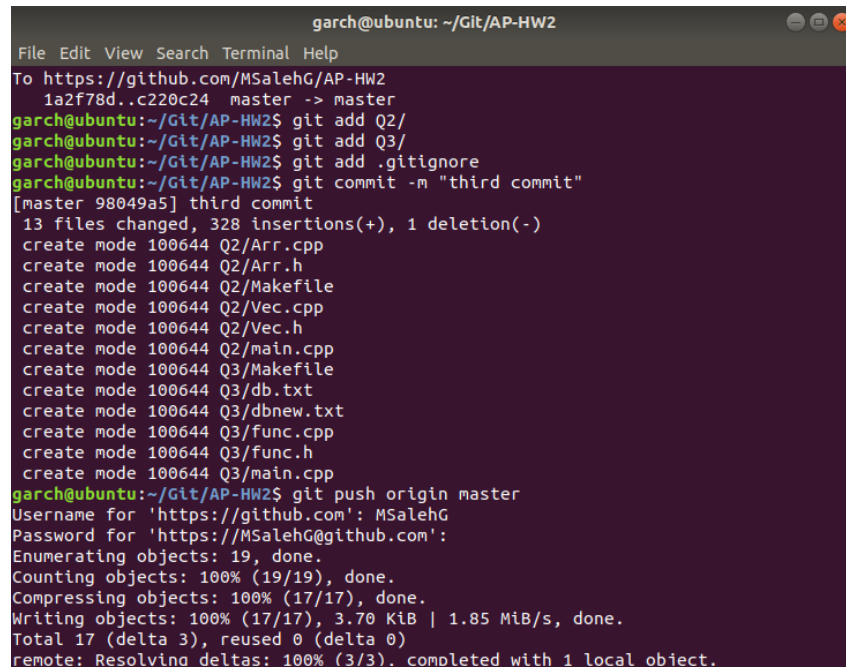
```
File Edit Format View Help
[2018-02-18] 3 2
[2018-02-19] 1 2
[2018-02-20] 3 3
[2018-02-21] 1 1
[2018-02-22] 1 1
Windows (CRLF) Ln 1, Col 1 100%
```

شکل ۷: فایل dbnew خروجی برنامه

## ۴ بارگذاری در github

برای بارگذاری کد ها در github ابتدا بعد از ساختن اکانت repository را ساختیم و در همان ابتدا فایل gitignore را با تنظیمات C++ در قسمت آپشن های repository تشکیل دادیم (با این کار در فایل gitignore. فایل ها با پسوند های نامطلوب که در برنامه نویسی C++ ساخته می شوند خود به خود اضافه شدند و در ادامه ما تنها اسم main را در این فایل اضافه کردیم ). سپس در محیط لینوکس دستور git init را اجرا کرده و سپس با دستور "git clone https://github.com/MSalehG/AP-HW۲" و بعد از آن git remote -v در یک di-rectory clone شده اضافه کرده و با استفاده از دستور "git add" پوشه های برنامه های نوشته شده را در بخش clone شده اضافه کردیم . در ادامه فایل های پوشه های برنامه های نوشته شده را در بخش clone شده اضافه کرده و با استفاده از دستور "git add" پوشه ها را اضافه کردیم . سپس با دستور "git commit -m" برای بارگذاری آماده کردیم . در نهایت هم با دستور git push origin master فایل ها را فرستادیم . لازم به ذکر است که اگر نیاز به تغییرات در هر یک از فایل های commit شده بود تنها نیاز بود که فایل را ویرایش کرده سپس دوباره آنرا add سپس commit و در نهایت push کنیم و فایل مورد نظر در repository خود به روز رسانی می شد .

در شکل ۸ می توانید مراحل انجام کار را مشاهده کنید .



```
garch@ubuntu: ~/Git/AP-HW2
File Edit View Search Terminal Help
To https://github.com/MSalehG/AP-HW2
1a2f78d..c220c24 master -> master
garch@ubuntu:~/Git/AP-HW2$ git add Q2/
garch@ubuntu:~/Git/AP-HW2$ git add Q3/
garch@ubuntu:~/Git/AP-HW2$ git add .gitignore
garch@ubuntu:~/Git/AP-HW2$ git commit -m "third commit"
[master 98049a5] third commit
13 files changed, 328 insertions(+), 1 deletion(-)
create mode 100644 Q2/Arr.cpp
create mode 100644 Q2/Arr.h
create mode 100644 Q2/Makefile
create mode 100644 Q2/Vec.cpp
create mode 100644 Q2/Vec.h
create mode 100644 Q2/main.cpp
create mode 100644 Q3/Makefile
create mode 100644 Q3/db.txt
create mode 100644 Q3/dbnew.txt
create mode 100644 Q3/func.cpp
create mode 100644 Q3/func.h
create mode 100644 Q3/main.cpp
garch@ubuntu:~/Git/AP-HW2$ git push origin master
Username for 'https://github.com': MSalehG
Password for 'https://MSalehG@github.com':
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Compressing objects: 100% (17/17), done.
Writing objects: 100% (17/17), 3.70 KiB | 1.85 MiB/s, done.
Total 17 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
```

شکل ۸: دستورات اجرا شده برای بارگذاری در git

همچنین با کلیک روی لینک زیر می توانید به repository ساخته شده در git بروید .  
<https://github.com/MSalehG/AP-HW2>