# 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 ، تا ۱۰۰ را تولید کرده و include از آن ها استفاده می کردیم برای این کار لازم بود دو کتابخانه ی stdlib.h و time.h را در برنامه ی خود کنیم . سپس در تابع خود باید دستور (srand(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 را (rand () ۲۰۱ % () prand 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 .

#### 1.1 قسمت الف

در این بخش ابتدا کلاس به نام map را تعریف کردیم و چهار متغیر را برای آن تعریف کردیم : ۱ \_ متغیر  $n \ge 1$  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 ناحیه pointer برای obstacle استفاده کردیم n = 1 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 n = 1 متغیر عالم برابر با ۱ و در غیر اینصورت برابر با ۱ قرار داده می شد . این متغیر هم همانند متغیر عاصفه تو ساخته شد n = 1 متغیر 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 شد .

در ادامه در تابع constructor این برنامه که تنها متغیر n را به عنوان ورودی می گرفت متغیر های obstacel 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 ۱ - n عدد عمل پایین آمدن و تعداد ۱ - n عدد عمل راست آمدن نیاز بود

پس یک حلقه for به شرط  $i < \mathsf{Y}(n-1)$  نوشتیم . در این حلقه فاصله ی هر ناحیه را با ناحیه های پایین و راست  $i < \mathsf{Y}(n-1)$  نوشتیم 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 مستفیم 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 = 1 قابل مشاهده است

شكل ١: نمونه حل سوال ١ قسمت الف

#### ۲.۱ قسمت ب

این بخش تماما 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 (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 راست عمل می کند و باعث می شود که ما زودتر از (n-1) تکرار به مقصد برسیم .

در شکل ۲یک نمونه از برنامه برای n = 9 قابل مشاهده است

#### ٣.١ قسمت ج

alt و num دو بخش قبل تفاوت داشت. در این بخش در تعریف کلاس map دو متغیر جدید num و علاوه برا متغیر های اشاره شده در دو بخش قبل تعریف شدند. متغیر num برای ذخیره تعداد حالت های مختلفی که علاوه برا متغیر و راست رفتن از ابتدا به انتها رسید استفاده شد. متغیر alt هم که از نوع \*\*rah بود به عنوان آرایه می شد با پایین و راست رفتن از ابتدا به انتها رسید استفاده شد به عنوان مثال در حالت n=0 یک از حالت های قابل قبول ی دینامیک برای ذخیره حالت های مختلف استفاده شد به عنوان مثال در حالت n=0 یک از حالت های در آرایه به صورت "rrdrddrd" 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 در مقدار دهی به متغیر num به آن نیاز بود .

شكل ٢: نمونه حل سوال ١ قسمت ب

در constructor کلاس در این بخش همانند بخش الف یک ورودی گرفته می شد . سپس همانند بخش الف متغیر alt متغیر های obstacle و ... ساختار دهی و مقدار دهی اولیه شدند . در ادامه با محاسبه ی متغیر num متغیر های ساختار دهی شد و سراغ مقدار دهی آن رفتیم . در مرحله مقدار دهی باید تمام جایگشت های ممکن r و p را می ساختیم به همین منظور جایگشت ها را به دو بخش تقسیم کرده که یکی از "...ddd..." شروع کرده و نصف دیگر از ....ddd شروع می کرد . در ادامه برای مثال در نیمه ی اول r اول را در هر تکرار با یکی از p ها عوض می کردیم تا جایگشت جدیدی ساخته شود سپس زمانی که p اول با تمام p ها جا به جا شد حال همین کار را با p دوم انجام می دادیم تا به آخرین p برسیم و سپس همین روند را با نیمه ی دوم و p 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 یا توجه به کوتاه 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 d را و d مقدار دهی کردیم .

در شکل  $\pi$  یک نمونه از برنامه برای n=1 قابل مشاهده است

## ٢ 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 و Vec.h ، Vec.cpp دو کلاس به نام های libVec و counter که ورودی عدد به نام های libVec و libVec را ساخته و در هر کدام از این کلاس ها یک تابع به نام libVec تابع طبیعی N را گرفته و در خروجی مجموع اعداد N-1 را می دهد ، تعریف کردیم . در کلاس libVec استفاده می کرد . counter 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

```
For the design of the design o
```

شكل ٣: نمونه حل سوال ١ قسمت ج

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

ذكر اين نكته لازم است كه در صورت سوال خواسته شده بود كه سوال با استفاده از pointer to member function حل شود ولى طى مكاتبه با استاد ايشان گفتند كه روشى كه در اين سوال پيش گرفتيم (فرستادن يك شى از كلاس به تابع runTime و اجرا كردن تابع counter از طريق شى) نيز براى حل اين سوال قابل قبول است .

در شکل ۴ و ۵ نمونه هایی آز 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)

```
The time needed to execute counter in libArr for N = 1000 is : 0 miliseconds
The time needed to execute counter in libArr for N = 100 is : 0 miliseconds
The time needed to execute counter in libVec for N = 100 is : 0 miliseconds
The time needed to execute counter in libArr for N = 100 is : 0 miliseconds
The time needed to execute counter in libArr for N = 1000 is : 0 miliseconds
The time needed to execute counter in libArr for N = 1000 is : 0 miliseconds
The time needed to execute counter in libArr for N = 10000 is : 0 miliseconds
The time needed to execute counter in libArr for N = 10000 is : 0 miliseconds
The time needed to execute counter in libArr for N = 10000 is : 8 miliseconds
The time needed to execute counter in libArr for N = 100000 is : 8 miliseconds
The time needed to execute counter in libArr for N = 100000 is : 87 miliseconds
The time needed to execute counter in libArr for N = 100000 is : 87 miliseconds
The time needed to execute counter in libArr for N = 1000000 is : 87 miliseconds
The time needed to execute counter in libArr for N = 1000000 is : 87 miliseconds
The time needed to execute counter in libArr for N = 1000000 is : 83 miliseconds
The time needed to execute counter in libArr for N = 10000000 is : 83 miliseconds
The time needed to execute counter in libArr for N = 10000000 is : 83 miliseconds
The time needed to execute counter in libArr for N = 100000000 is : 83 miliseconds
The time needed to execute counter in libArr for N = 100000000 is : 83 miliseconds
The time needed to execute counter in libArr for N = 100000000 is : 83 miliseconds
```

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
 m -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
 ++ -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
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$
arch@ubuntu:~/Desktop/HW2/2$
 arch@ubuntu:~/Desktop/HW2/2$
  rch@ubuntu:~/Desktop/HW2/2$
```

شكل ۵: نمونه ى حل سوال ۲ در محيط لينوكس

### ٣ 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 پارامتر های 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 بردار های 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 بردار هایی که که شماره های محصولات و خریداران را ذخیره کرده بودیم به جستجو پرداخته و تعداد شماره های متعداد 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 بردار های 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 ماتریس ساخته شده با هم برابر باشد . در یک بردار دو بعدی نحوه ی استفاده از دستور pushback به این بردار های تک بعدی و pushback (در آرایه های 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 بردار هایی که به صورت reference به تابع database فرستاده بودیم اطلاعات لازم را در فایل خروجی نوشتیم .

فكر اين نكته لأزم است كه با توجه به اين كه فايل db.txt در فايل تمرين قرار داده نشده بود يك فايل db.txt در فايل تمرين قرار داده نشده بود يك فايل cdb.txt دلخواه را ساخته و عمليات رو روى آن انجام داديم .

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

شكل 6: فايل db ساخته شده



شكل ٧: فايل dbnew خروجي برنامه

## ۴ بارگذاری در github

برای بارگذاری کد ها در gitignore 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 remote - ۷ و بعد از آن y- git remote در یک - فایل در بخش در ادامه فایل های پوشه های برنامه های نوشته شده را در بخش در ادامه فایل های پوشه های برنامه های نوشته شده را در بخش در ادامه فایل های پوشه ها را اضافه کردیم . سپس با دستور git push origin master و با استفاده از دستور git و 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 امورد نظر در فایل را ویرایش کرده سپس دوباره آنرا ba سپس commit و در نهایت این و و فایل مورد نظر در به رود که فایل را ویرایش کرده سپس دوباره آنرا ba سپس commit و در نهایت این و و فایل مورد نظر در repository خود به روز رسانی می شد .

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

```
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/Wec.cpp
create mode 100644 Q2/Vec.h
create mode 100644 Q2/Vec.h
create mode 100644 Q3/Makefile
create mode 100644 Q3/Makefile
create mode 100644 Q3/db.txt
create mode 100644 Q3/db.cxt
create mode 100644 Q3/db.cxt
create mode 100644 Q3/func.cpp
create mode 100644 Q3/func.cpp
create mode 100644 Q3/func.b
create mode 100644 Q3/func.b
create mode 100644 Q3/func.b
create mode 100645 Q3/func.b
create mode 100645 Q3/func.b
create mode 100645 Q3/func.b
create mode 100645 Q3/func.h
create mode 100646 Q3/func.h
create mode 100647 Q3/func.h
create mode 100647 Q3/func.h
create mode 100648 Q3/func.h
create mode 100649 Q3/func.h
create mode 100644 Q3/func.h
create mo
```

شکل ۸: دستورات اجرا شده برای بارگذاری در git

همیچنین با کلیک روی لینک زیر می توانید به repository ساخته شده در git بروید . https://github.com/MSalehG/AP-HW۲