نام و نام خانوادگی دانشجو: محمدامین احمدی رشته :کامپیوتر شماره دانشجویی: 9912358001

نام استاد : خانم خدابندلو

موضوع : پروژه ی فاینال ساختمان داده

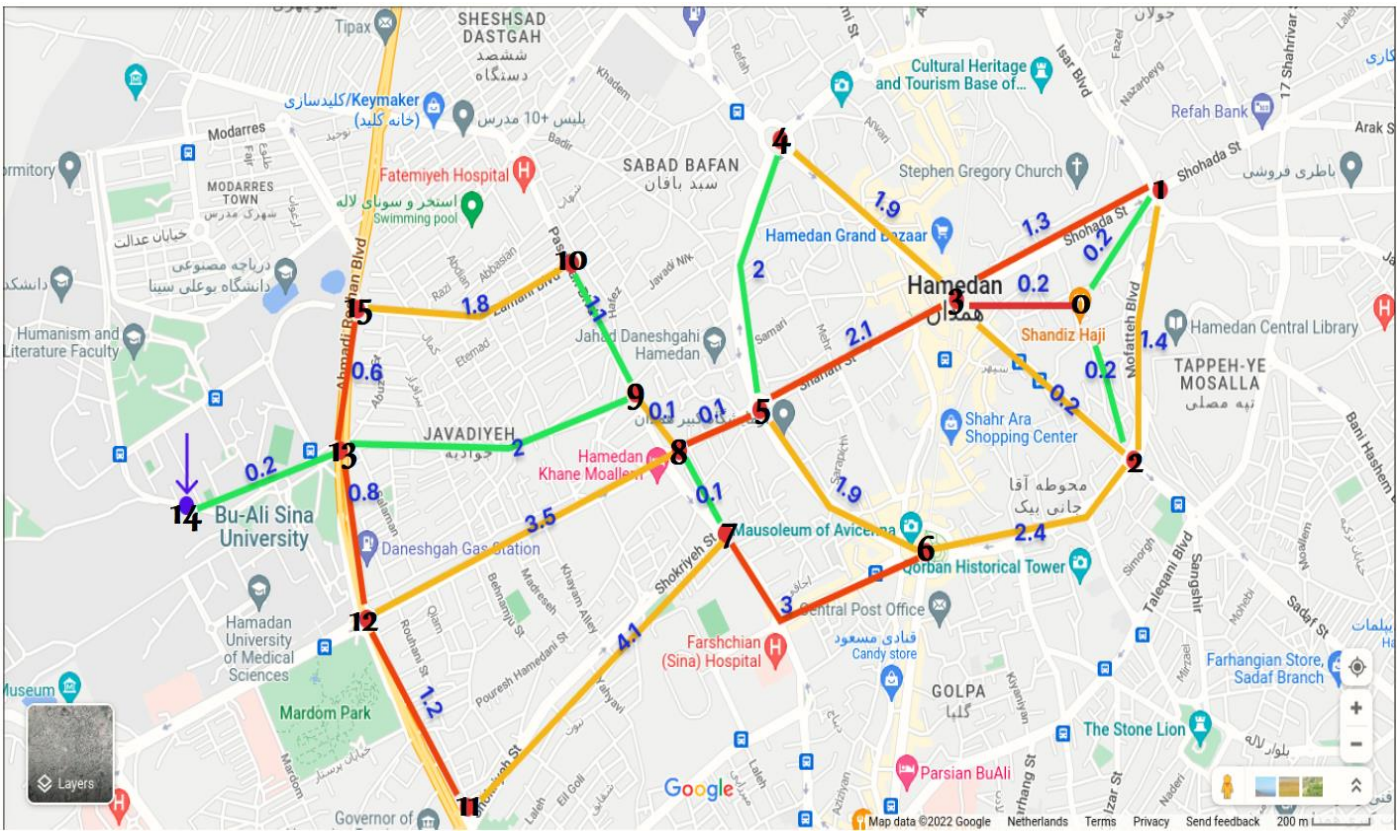

بسم الله الرحمن الرحيم

فهرست مطالب

شرح پروژه

این پروژه با زبان پایتون پیاده سازی شده. دو کلاس دارد و قسمت Drive کد مربوط به اطلاعات گراف ما از قبیل :

شماره ی هر گره و وزن یال بین دو گره مشخص می شود. هدف از این پروژه کمک به پیک برای رساندن محموله از رستوران به خوابگاه است.



طبق این تصویر من نقاط موردنظر برای مسیریابی را شماره گذاری کرده ام تا بتوان درک بهتری از مسیر داشت.

مبدا ما یعنی رستوران با برچسب 0 و مقصد که خوابگاه است با برچسب 14مشخص شده.

مقدمه

در این پروژه من نقشه ی راه رسیدن محموله از رستوران به دانشگاه را همانند گرافی در نظر گرفته ام. برای پیدا کردن کوتاه ترین مسیر نیز از الگوریتم Dijkstra استفاده شده که می توان گفت بهترین الگوریتم برای پیدا کردن کوتاه ترین مسیر در گراف های با وزن مثبت است.

همچنین از ماژول heapq هم استفاده شده. این heap یک نوع ساختار درختی است که مشخص می کند که هر گره والدی که وجود دارد باید مقدارش کمتر یا مساوی از فرزندانش باشد.به مثال زیر توجه کنید:

```
1   from heapq import heapify, heappop, heappush
2   n = []
3   heappush(n,4)
4   heappush(n,5)
5   heappush(n,3)
6   heappush(n,2)
7   heappush(n,8)
8
9   print(heappop(n))
10  print(n)
11
```

```
JUPYTER   DEBUG CONSOLE   PROBLEMS 15   OUTPUT   TERMINAL

[Running] python -u "c:\Users\ahmadi\Desktop\python_
2
[3, 5, 4, 8]
```

علت استفاده از این ساختار پیاده سازی الگوریتم حریصانه در Dijkstra است که با عث می شود ما به مجموعه ی دیگری برای ثبت و ذخیره ی این که چه گرهی ملاقات شده و چه گرهی ملاقات نشده، احتیاج نداشته باشیم .

باتشکر از دوست خوبم آقای رستمی که در پیاده سازی تابع Masir و یافتن راه حلی برای چاپ بهترین مسیر به من کمک کرد.

الگوریتم Dijkstra

الگوریتم دایجسترا(Dijkstra's Algorithm) الگوریتمی است که برای پیدا کردن کوتاه‌ترین مسیر بین دو گره راس در گراف به کار می‌رود. این گراف، ممکن است نشان‌گر شبکه جاده‌ها یا موارد دیگری باشد. الگوریتم دایجسترا دارای انواع گوناگونی است. الگوریتم اصلی، کوتاه‌ترین مسیر بین دو گره را پیدا می‌کند؛ اما نوع متداول‌تر این الگوریتم، یک گره یکتا را به عنوان گره مبدا (آغازین) در نظر می‌گیرد و کوتاه‌ترین مسیر از مبدا به دیگر گره‌ها در گراف را با ساختن درخت کوتاه‌ترین مسیر پیدا می‌کند .

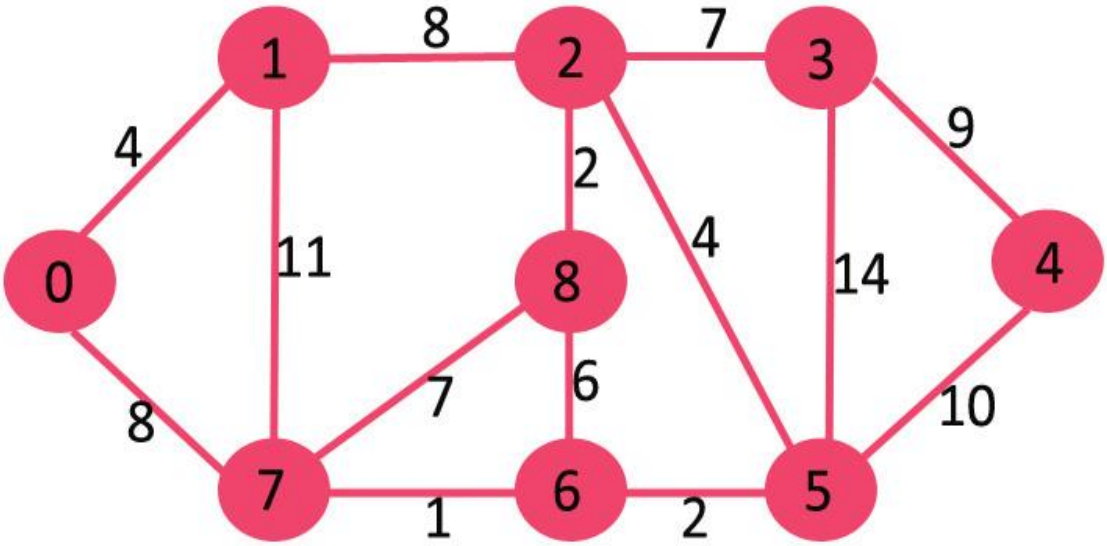
برای یک گره مبدا داده شده، الگوریتم، کوتاه ترین مسیر بین آن گره و دیگر گره‌ها را پیدا می‌کند. همچنین، الگوریتم دایجسترا برای پیدا کردن کوتاه ترین مسیر از یک گره یکتا به گره مقصد یکتای دیگری به کار می‌رود؛ برای انجام این کار، الگوریتم هنگامی که کوتاه ترین مسیر از مبدا به مقصد را پیدا کند، متوقف می‌شود. برای مثال، اگر گره‌های گراف نشان‌گر مناطق شهری و هزینه ی یال ها حاصلضرب ترافیک در مسافت بین دو منطقه باشد که با خیابان های مستقیم به هم متصل شده‌اند، از الگوریتم دایجسترا می‌توان برای پیدا کردن کوتاه‌ترین راه بین یک مبدا و مقصد استفاده کرد.

الگوریتم دایجسترا از برچسب‌هایی استفاده می‌کند که اعداد صحیح یا حقیقی مثبت هستند. جالب توجه است که الگوریتم دایجسترا می‌تواند برای استفاده از برچسب‌های تعریف شده به هر شکلی، تعمیم پیدا کند. چنین تعمیمی، «تعمیم الگوریتم کوتاه‌ترین مسیر دایجسترا» نامیده می‌شود.

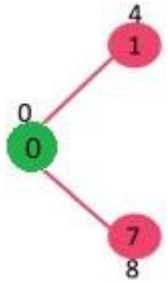
یافتن کوتاه‌ترین مسیر:

فرض می‌شود که یک گراف به همراه یک راس مبدا داده شده و هدف پیدا کردن کوتاه‌ترین مسیر به همه راس‌های موجود در گراف مذکور است. در الگوریتم دایجسترا درخت کوتاه‌ترین مسیر با استفاده از مبدا داده شده به عنوان ریشه، ساخته می‌شود. در هر مرحله از الگوریتم، راسی پیدا می‌شود که در مجموعه دیگر (مجموعه راس‌های در نظر گرفته نشده) قرار دارد و دارای کمترین فاصله از ریشه است. در ادامه، گام‌های مورد استفاده در الگوریتم دایجسترا به منظور یافتن کوتاه‌ترین مسیر از یک راس مبدا مجرد به دیگر راس‌ها در گراف داده شده به صورت مشروح بیان شده‌اند.

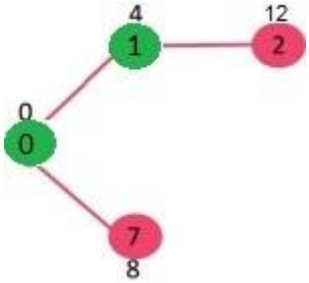
1. ساخت مجموعه Shortest Path Tree Set(sptset) که به دنبال راس‌های قرار گرفته در درخت کوتاه‌ترین مسیر می‌گردد؛ یعنی، راسی که حداقل فاصله آن از مبدا محاسبه و نهایی شده است. به طور مقدماتی، این مجموعه خالی است.
2. تخصیص یک مقدار فاصله به همه راس‌ها در گراف ورودی. مقداردهی اولیه همه مقادیر فاصله‌ها به عنوان INFINITE. تخصیص مقدار فاصله صفر به راس مبدا که موجب می‌شود این راس در ابتدا انتخاب شود.
3. تا هنگامی که مجموعه Shortest Path شامل همه راس‌ها نشده است، اقدامات زیر انجام می‌شود:
   - راس u انتخاب می‌شود که در sptSet نیست و دارای حداقل مقدار فاصله است.
   - uدر sptSet قرار می‌گیرد.
   - مقدار فاصله از همه راس‌های مجاور u به روزرسانی می‌شود. برای به روزرسانی مقادیر فاصله، این عمل در همه راس‌های مجاور تکرار انجام می‌شود. برای هر راس مجاورv ، اگر مجموع فاصله u (از منبع) و وزن یال u-v کمتر از مقدار فاصله v باشد، مقدار فاصله از v به روز رسانی می‌شود.
   - برای درک بهتر موضوع، مثال زیر مورد بررسی قرار خواهد گرفت.



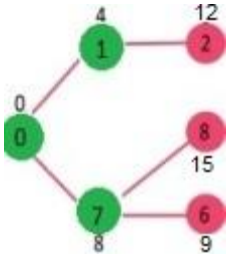
مجموعه sptSet در ابتدا خالی است و فاصله تخصیص پیدا کرده به راس‌ها برابر با { صفر, INF, INF, INF, INF, INF, INF, INF} هستند که در آن INF نشان‌گر بی‌نهایت (Infinite) است. اکنون، باید راسی که دارای کم‌ترین مقدار فاصله است انتخاب شود. راس ۰ انتخاب می‌شود و در sptSet قرار می‌گیرد. بنابراین، sptSet به صورت {0} می‌شود. پس از قرار دادن ۰ درsptSet ، مقدار فاصله‌ها از راس‌های مجاور آن به روز رسانی می‌شوند. راس‌های مجاور ۰، راس‌های ۱ و ۷ هستند. مقدار فاصله برای ۱ و ۷، برابر با ۴ و ۸ است. زیرگراف زیر، راس‌ها و مقدار فاصله آن‌ها را نشان می‌دهد. راس‌های موجود در spt به رنگ سبز نمایش داده شده‌اند.



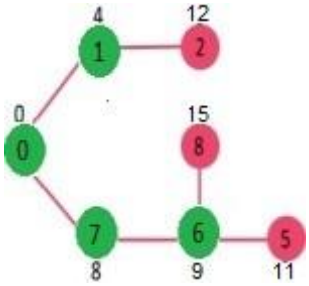
راسی که حداقل فاصله را از مبدا دارد و تاکنون انتخاب نشده است، یعنی در sptSET قرار ندارد، انتخاب می‌شود. راس ۱ انتخاب و به sptSet اضافه می‌شود. بنابراین، اکنون sptSet به صورت {۰ ,۱} خواهد بود. مقدار فاصله راس‌های مجاور ۱ به روز رسانی می‌شود. مقدار فاصله از راس ۲ برابر با ۱۲ خواهد بود.



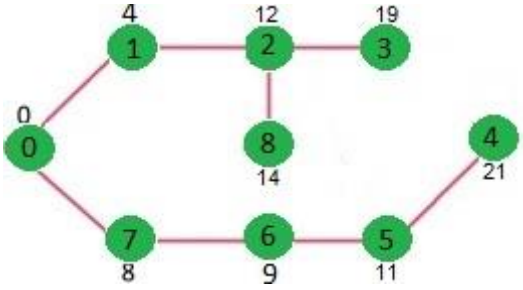
راسی با کمترین مقدار فاصله که در حال حاضر در spt قرار ندارد باید انتخاب شود. راس ۷ انتخاب می‌شود. بنابراین، sptSet اکنون به صورت {۷ , 1 , ۰} خواهد بود. مقدار فاصله از راس‌های مجاور ۷ محاسبه می‌شود. مقدار فاصله از راس ۶ و ۸ متناهی است (به ترتیب، ۱۵ و ۹).



راسی با حداقل مقدار فاصله که در spt نیز قرار ندارد باید انتخاب شود. راس ۶ انتخاب می‌شود.
بنابراین، sptSet اکنون برابر با {۰, ۱, ۷, ۶} است. مقدار فاصله‌ها از راس‌های مجاور ۶ باید به روز رسانی شود. مقدار فاصله برای راس‌های ۵ و ۸ به روز رسانی می‌شود.



مراحل بیان شده تا جایی تکرار می‌شوند که sptSet شامل همه راس‌های گراف داده شده نباشد. در نهایت، درخت کوتاه‌ترین مسیر (SPT) زیر حاصل می‌شود.



برای پیاده‌سازی الگوریتم بالا، از آرایه بولین sptSet[] برای ارائه مجموعه‌ای از راس‌های قرار گرفته در SPT استفاده می‌شود. اگر مقدار sptSet[v] «درست» (True) باشد، راس v در SPT قرار می‌گیرد، در غیر این صورت، یعنی اگر sptSet[v] «غلط» (False) باشد، راس v در SPT قرار نمی‌گیرد. آرایه dist[] برای ذخیره‌سازی کوتاه‌ترین مقدار فاصله از همه راس‌ها مورد استفاده قرار می‌گیرد..

شبه کد:

Dijkstra (G, w, s)

```
1.  INITIALIZE-SINGLE-SOURCE(V, s)
2.  S ← ∅
3.  Q ← V[G]
4.  while Q ≠ ∅
5.      do u ← EXTRACT-MIN(Q)
6.          S ← S ∪ {u}
7.          for each vertex v ∈ Adj[u]
8.              do RELAX(u, v, w)
9.              Update Q (DECREASE_KEY)
```

class Gereh

کلاس گره شامل دو تابع است که هریک از این توابع به شکل خاصی به بهبود برنامه کمک میکنند که در ادامه به آنها میپردازیم:

1) کانستراکتور: این تابع دو تا از مهم ترین مؤلفه ها را برای ما ذخیره می کند:

الف)شماره ی هر راس گراف:

```
self.vertexNum
```

ب)فاصله ی آن راس تا مبدا:

```
self.SourceDist
```

پروتوتایپ کانستراکتور که شامل یک مقداردهی پیش فرض برای فاصله ی راس تا منبع است:

```
def __init__(self, vertex, weight=0):
```

2) مجیک فانکشن Less than که برای مقایسه ی دو شی از گره به کار می رود با استفاده از مقایسه کردن فاصله آن راس ها تا مبدا:

```
def __lt__(self, other):
    return self.SourceDist < other.SourceDist
```

نکته: وجود این مجیک فانکشن برای ذخیره سازی گره ها در هیپ ضروری است که در ادامه بیشتر در مورد آن میخوانیم.

class Graph

در این کلاس درواقع به ایجاد یک ماتریس مجانبی به کمک کلاس Gereh می پردازیم.

تابع عضو کلاس:

کانستراکتور:پروتوتایپ آن شامل دو پارامتر می شود :

الف)edges : یک لیست است شامل گره های متوالی و اندازه یال بین آن.

ب) n : که تعداد راس های این گراف است.

def __init__(self, edges, n):

با استفاده از دستور self.adjList = [[] for _ in range(n)] یک ماتریس مجانبی خالی برای ذخیره ی مشخصات راس ها و یال های بین آن ها ایجاد می کنیم و در حلقه ی for با استفاده ازدستور:

self.adjList[mabdae].append((maghsad, andaze))

-این ماتریس مجانبی را مقداردهی می کنیم.در این حالت شماره ی هر ستون می شود راس مبدا و شماره ی هر سطر می شود راس مقصد و عددی که در این سطر و ستون قرار می گیرد می شود اندازه ی یال از راس مبدا به راس مقصد.

تابع Masir

تابع بازگشتی Masir برای چاپ مسیر طی شده به کار می رود و پروتوتایپ آن شامل سه پارامتر می شود :

def Masir(jad, i, Rah):

الف)jad : یک لیست از راس های پدر است که از قبل توسط تابع DijkstraDP(در ادامه به آن میپردازیم) و به تابع Masir فرستاده می شود.

ب)i : شماره ی راس مقصد ما است یعنی راس 14.

ج)Rah : در ابتدا یک لیست خالی است.

دستور Masir(jad, jad[i], Rah) که درون بلاک شرط قرار دارد باعث به وجود آمدن تابع بازگشتی می شود و این عمل بازگشتی و ایجاد پشته تا جایی ادامه پیدا میکند که به پدر راس مبدأ(راس صفر) که برابر با 1- است برسد .پس از آن چون i = -1 می شود شرط برقرار نیست و دستور Rah.append(i) از درونی ترین پشته شروع به اجرا میکند و پشته ها یکی پس از دیگری از i = 0 تا i = 14 این دستور را اجرا میکنند و و لیست خالی Rah را از مسیر طی شده پر می کنند.

تابع DijkstraDP

DijkstraDP برای محاسبه ی مسیر طی شده و تعیین پدر هر راس به کار می رود و پروتوتایپ آن شامل سه پارامتر می شود :

def DijkstraDP(graph, source, n):

الف) graph : یک شی از کلاس Graphاست که در واقع یک ماتریس مجاورتی از شماره ی راس و اندازه یال بین آن ها است.

ب) source : گره مقصد یاهمان گره صفر است.

ج) n : مربوط به تعداد راس موجود در گراف ما می شود که 16 تا است.

دستور heappush(Relax, Gereh(source)) راس شماره ی 0 را با مکانیزم min_heap وارد لیست خالی Relax می کند.

با استفاده از دستور dist = [infinity] * n ابتدا تمامی فاصله ی راس ها از گره صفر را برار با بینهایت(یک عدد بزرگ) قرار می دهیم.

با توجه به اینکه کوچک ترن گره ما گره صفر است و جد تمامی گره ها محسوب می شود و همچنین شرط if i >= 0 از تابع بازگشتی Masir ، در ابتدا لیست pedar را به تعداد 16 گره گراف با 5- پر می کنیم.

اعمال حلقه ی While

به طور خلاصه در حلقه ی while دو اتفاق می افتد.اول اینکه پدر هر راس مشخص می شود و دوم، هر راس ریلکس می شود.

دستور while Relax به این معنی است که تکرار این حلقه تا زمانی که لیست Relax خالی شود ادامه پیدا می کند.

وظیفه ی خالی کردن این لیست با دستور heappop است که بعنوان ورودی لیست Relax را می گیرد و با استفاده از مکانیزم min_heap کوچک ترین گره(یعنی گرهی که کمترین فاصله را با راس مبدا دارد) را از لیست خارج کرده و در Node میریزد. این عمل باعث می شود که در طی مکانیزم Relaxation به گره تکراری بر نخوریم.

حلقه ی for : درون این حلقه ما عمل ریلکس کردن را به ازای تعداد همسایه های ملاقات نشده گره u تکرار می کنیم.

شرط (dist[u] + weight) < dist[v] برسی میکند که اگر فاصله ی گره v از گره مبدا بیشتر از فاصله ی گره u تا مبدا به علاوه ی طول یال تا v باشد در آن صورت سه عمل مهم را انجام میدهد:اول از همه گره v را ریلکس می

کند و فاصله اش تا مبدا را به روزرسانی می کند و دوم، پدر گره v از 5- به u تغییر می کند. در سومین گام گره v با دستور heappush(Relax, Gereh(v, dist[v])) و با مکانیزم min_heap وارد لیست گره های ریلکس شده می شود.

زمان آن فرا رسیده که علاوه بر طول مسییر، خود مسیر هم مشخص شود.پس ما تابع Masir را در دل تابع DijkstraDP فرا خوانی کنیم.

این تابع سه پارامتر دارد:

اول: پارامتر pedar که لیستی است که توسط حلقه ی while و حلقه ی for درونش تدارک دیده شده وشامل پدر هر یک از این 16 گره است.

دوم: پارامتر i که همان مقصد مورد نظر ما یعنی خوابگاه غدیر است که با عدد 14 مقدار دهی شده است.

سوم: پارامتر Rah که یک لیست خالی است و باید توسط تابع Masir از نقاط صحیح پر شود.

در انتهای تابع DijkstraDP مسیری که باید طی شود و طول آن نمایش داده می شود.

Driver code

این بخش با :'__main__' == __name__ if از بقیه تمیز داده شده. هر ماژول پایتون دارای __name__ تعریف شده است و اگر "__main__" باشد، به این معنی است که ماژول به صورت مستقل توسط کاربر اجرا می شود. اگر این اسکریپت را به عنوان یک ماژول در اسکریپت دیگری وارد کنید، __name__ به نام اسکریپت تنظیم می شود.

در متغیر YallHa در هر آرایه که با پرانتز باز و و بسته مشخص می شود از چپ به راست گره مبدا و گره مقصد و طول مسیر (اندازه ضرب در ترافیک) وارد شده است.

با دستور n = 16 تعداد گره های مووجود در گراف را مشخص می کنیم.

graph یک object از کلاس Graph است که در کنسراکتور آن از دو متغیر YallHa و n استفاده می شود.

با فراخوانی تابع DijkstraDP برنامه ی ما کامل می شود.

در پناه خدا

شاد و پیروز باشید