《最优化技术》实验报告

一、实验目的

理解动态规划算法的基本思想，并应用于求解实际问题。

二、实验项目内容

请利用动态规划解决以下两个问题：

1) 爬楼梯问题。自行输入 n 阶楼梯，每次可以爬 1 或 2 个台阶，请编程判断有多少种不同方法可以爬完楼梯。

2) 判定买卖股票的最佳时机。自行给定一个数组，它的第 i 个元素是一支给定股票第 i 天价格。设计一个算法来计算你所能获取的最大利润。注意你不能在买入股票前卖出股票，不能同一天进行多次交易。

例如：

输入： [7,1,3,5,4,6]

输出：最大获利：6

对以上结果的解释：第二天买入股票，第四天卖出股票，则获利为 5-1=4 元，第五天买入股票，第六天卖出股票，获得 2 元，总共获得 6 元。

注意：所有程序请用 python 语言实现。只提交本电子文档，注意本文件末尾的文件命名要求；源程序一节请用代码备注的方式说明你的算法和思路；实验结果一节需要提供测试结果截图并给出结果分析。

三、实验过程或算法（源程序）

1）爬楼梯问题

（分别使用递归和动态规划两种方法对比时间复杂度以比较优劣）

```
'''爬楼梯问题：
题目:输入 n 阶楼梯,每次可以爬 1 或 2 个台阶,用动态规划判断有多少种不同方法可以爬完楼梯
法一:首先想到递归，climbStairs(n)=climbStairs(n-1)+climbStairs(n-2)
    但相当费时，记录程序执行时间，与法二动态规划进行对比
'''
import time
def climbStairs(n):
    if(n==1):       return 1
    elif(n==2):  return 2
    else:
        c1=climbStairs(n-1)     #爬 1 层
```

报告创建时间：2022.4.25

```
        c2=climbStairs(n-2)     #爬 2 层
        return c1+c2

n=int(input())

time_start = time.time()        #开始时间
solution=climbStairs(n)
time_end = time.time()          #结束时间
time_c= time_end - time_start

print("Solution:",solution)
print("Time of Recursion:",time_c)
```

```
'''爬楼梯问题:
题目:输入 n 阶楼梯,每次可以爬 1 或 2 个台阶,用动态规划判断有多少种不同方法可以爬完楼梯
法二:动态规划,dp[i]=dp[i-1]+dp[i-2]
'''
import time

def climbStairs(n):
    Method=[0 for i in range(n+1)]
    Method[1]=1
    Method[2]=2
    if(n==1):       return 1
    elif(n==2):  return 2
    else:
        for i in range(3,n+1):
            Method[i]=Method[i-1]+Method[i-2]
        return Method[n]

n=int(input())
time_start = time.time()
solution=climbStairs(n)
time_end = time.time()
time_c= time_end - time_start

print("Solution:",solution)
print("Time of DP:",time_c)
```

2）买卖股票

```
'''买卖股票:
题目:判定买卖股票的最佳时机。自行给定一个数组,它的第 i 个元素是一支给定股票第 i 天价格。设计一个算法来计算你所能获取的最大利润。注意你不能在买入股票前卖出股票,不能同一天进行多次交易。
输入样例:
[7,1,3,5,4,6]
```

```
输出样例:
6
思路:股票买卖显然需要先买再卖,每天的状态只有手里有股票和无股票两种,设
    dp[i][0]表示第 i 天手里没有股票可得的最大收益,dp[i][1]表示第 i 天手里有
    股票可得的最大收益(手中股票不能折现,第一次买入后手中收益为负)。每天的收
    益均由前一天决定,若当天手中有股票,则可能今天买入或前一天手中有股票;若
    当天手中没有股票,则可能前一天卖出或者前一天手中没股票。显然有递推关系
    式:
        dp[i][0]=max{dp[i-1][0],dp[i-1][1]+price[i]}    #前一天手里没
股票或今天卖了股票
        dp[i][1]=max{dp[i-1][1],dp[i-1][0]-price[i]}    #前一天手中有
股票或今天买入
'''
price=[]
def Stock(n):
    dp=[[0 for col in range(2)] for row in range(n)]
    dp[0][0]=0
    dp[0][1]=-1*price[0]
    for i in range(1,n):
        dp[i][0]=max(dp[i-1][0],dp[i-1][1]+price[i])    #前一天手里没
股票或今天卖了股票
        dp[i][1]=max(dp[i-1][1],dp[i-1][0]-price[i])    #前一天手中有
股票或今天买入
    return dp[n-1][0]                                   #最后一天必须卖
出

n=int(input())

for i in range(n):
    p=int(input())
    price.append(p)
profit=Stock(n)

print("profit:",profit)
```

四、实验结果及分析和（或）源程序调试过程

1. 结果分析：

1）爬楼梯问题

使用递归和动态规划分别求解，下给定几组不同规模的输入数据 n 对比两种方法的复杂度。（每组样例的第一个结果为递归实现，第二个为动态规划）

样例 1（n<10）：

```
5
Solution: 8
Time of Recursion: 0.0
5
Solution: 8
Time of DP: 0.0
```

样例 2（n<20）：

```
12
Solution: 233
Time of Recursion: 0.0
12
Solution: 233
Time of DP: 0.0
```

样例 3（n<30）：

```
24
Solution: 75025
Time of Recursion: 0.00801396369934082
24
Solution: 75025
Time of DP: 0.0
```

样例 4（n<50）：

（接近递归法极限）

```
48
Solution: 7778742049
Time of Recursion: 781.016063451767
48
Solution: 7778742049
Time of DP: 0.0
```

样例 5（n<100）：

（递归法崩溃，动态规划仍然良好）

```
500
Solution: 225591516161936330872512695036072072046011324913758190588638866418474627738686883405015987052796968498626
Time of DP: 0.0
```

样例 6（n~10^5）：

（远远超出递归法能力范围，动态规划依旧快速有效）

```
100005
Solution: 46608576297572097173629557820989278761631568314316724318800114049562953123413846656204286204554467660077588087234403849548948498127515268185351441869243068424167567502147400071013287760640035525560369407318388684370218413712408042720876731992499025057171927458890604130457001367743243283147416202680771610013840119830645179100610493900310179765857652418901335745903947038006424973336291385980783541488463310258176047399924611990537700323158041402705217265377861780726256003694334169735821395990955258881837468342668916839962709797481816275580856964011762175276198626978361690159370521452521203263888762430036335905825362966504662474918687172317350025083994891652275264729626398706564507895840510042247532857692255553029554301290202978731297110745707373465400917529067562978904642738692549734304434610773042350946963434910637887611639603183428631170681169748144535778309484603121588636829961964673025419861741501296187433726775784740201600361216587978811981021579830276950831659688670083995085786362698732723644244260212317290415966693599307828872190282631815794365825250008012071995800725795039649440936523489603048396902224446684319545768994664657716863244637696612565418807079334253856955375830514699020194722111935539301339923937470430651846845984427456410412245603791491507501122581955953663318317733468853643888597402989988933888222901244019087872339978002912835307668003397328854294353512941505937173941089003416318390121025112858285048737346165979144403591141222707135603578706366484795361088857017241589767074944853775153569925852591927018398648168055050147463861322121935028181814599504238815133273720959862693864634079100020331039162641549949739024499180418135455751768608896438204507819153929453293935202670294355230961701904878797611053564928735230106420465277195184571762541605853948083916468538915043157626248673978752021307418448259037253720053543990037556527072633096504843726823905320391463183159253397274372633368070703706876626203280658443707728149871467363499073337533806173202012456359225266450267973867218946132304569432501454378989561751604117757055865097773675314554502211870426836412689612645508329844427823969273292320500709258022379168494778610
```

```
4440980363099580282427331347120944039853445780070731888843872044049434942382347309533689640647407864113083897
8242874164848194825092543461552608988283435540457572519543184967895679518226467157889209618865315579788577059
7875187374276776777675773854052399641209351077220421021787512119747425474702424049840409958062598277337728602
1622440154716208949893221647281029952633077871191186544105677168194012977026398989953652425712169876317160199 2
5778739794505689555614667833339295693270874596457727400785318529822497369576897512865978825849902871691817054
2110923954066824838150791025666973426218736415622302556341689321145358034139700455751292242109076028711142480
0943413564789641940454877432379109424183182242969590968116565075631910258875696010186411970311117496347550452
9461825457461234537859477883768342249255611954121993296806573023746283489714611803039986485216819860650882940
6891353757084581621578345998036243850114403361679671403888307525142342643377541347335867702725196732472313204
1222526948683338722567553181862743527156917879225436002710181023893645315741636439713147042727555848470006100
4665103940163029122145074860842941890409353800607892867551005910044335942740068757758191564319442777206620255
1449841978238449663976765281862139339660057144472485354157095605659616538534644578196907464330328479219740569
1957968555084115269957502883746204598580717852082992148980945787090984480211845173328663277635068373285732288
4504285455661997994080023705143742451847113180178451201117542770275964118660971618476838919722326920244016132
5426329836494752152234376554856482802222813268056077652828510749173675130098598161634698982042040008941575458378
8803251122853543154633003323434061248332150789046132415162308272605131378424289087650212104894630226666709127
4341611532232932580558927845063513728372616607161047402677934835856461296404007490705802759306995465924186988
2175928414513940396265567278823242514798831924640560254280763327970460984312783381406720666395413537850159851
5746758004007300972414336298598315960210592002764767361815127510207009746177917147354337645433615935609720534
2701588579219365066012679625784434029805066351292498817111256433270413299651089962367036172167551197484736553
4977885314248146221488204291942688440467069779308905677133509673677605681526276281261864067716560520496724920
6383466177286471317459809567494092410042133020539352758887139230793523764338362108267597547285625985987972672
5225994693957332448765424482004146142889514600798376812674310154380649159135902167026286305035980030165474952
6912641160088342941351551605357083030306876625734534993514747712578127327248546157659454181335345292322158477
4068065479945789153766191558100483186227835233969985535876143805511190039834619153995615544926451555912258402
2205253396292398640709234544405967998310436302160466058123771175439584369368704871381952169258314079706047556
7357378506778789612287733494768451601719373803798957181633256184658564694508138636553249682870437029649461640
6357910996909985461870073848101802652468786504139205167656976474267169917782290231319752277874617976753975430
4285749567643891840626434567720887795254329672093617901733626842799314628540400016664754220134070470455519553 44
1733309730143848502621561344004274552345680286939822616459324120979288479235233100770651283514864648993132685
8850869542228455954614642758537978959286202940391461701720914134781649835587344290436483969015516447322498357
7169952739775842896066369828307320665301256209411372186116446430103642305593427149018804024958510524543964790
6619094761219534332014110552551481480878919971628706143248512095382048874865450483528157733641067860096298817
8342942180494158806959986836984302550645667764683707561159548928653729077145397772305616670111367494715069467
4817276045170186899785256890860402045714416911871744633593422499985145010906668543660642157066840853366103538 7
7613578486271104449488375408454321885737992545368827184052384909808063051858814998946768411544712125006831410
4536715520548624644869222490868425055224470028382287366596599640371510246669169822259420072299273107813441958 9
2716618137111638311779195042307151243774419835692142161428557186268666999878376915395442292377180172810882088
9716676198815384470787729129379020752020935472168530044037556568532565535102789114095552835446032499471734999
6328539787169992954531505839246735609874705624871314642182194952239085243757148470929449247349959769802536072
8899398445814067960965925041602733900288898238220956685562764686658572751885213457202637766871097587172534219 2
0672249799868722748437578300590051786616324829880782406222783406225595021866372493922664256288291748801434930 7
1221935841472641958516339745547252886054206978683139075280451011254391391329029512626613796617607430274174908
8963723199950474608323577318037058165000948762308962692236226459656950871994930181931025752326293870363882185 5
3746469264454030006012565411698949757379470925625818001454248710901175959970121971273345356456262887329047549
2059812842787870257808065442887740102172209473624928417973260854679914045333866602974756297335405037070697156
5507237743425694798306879902909610945569248279497624335637365115604090388804456593325715606745246563717893 18
8981935292095794712715858097801492152698839066064351212989858311679501051466547947009599638419185629599823912
9487760217327296996645491426115026399252223050318587765014011169168833750171646897321389697568896739398340416
3264457285658983752609883190669754360658722723674529910055457047961802212540298864302195840296978906938008348
7618014976895331747425596984964264239789496601240967054745950625439184987029525158453484156410681087782319945
1295604980913488607052930923828000665233132869496041449623565111195798630676249897372936230542210347674440905
7587788532770641859735831168662043157827396276051929579610586911722891218922859060464193194345452089796326738
2315140839916454845465285894609625816424804144572814375317269867216602371441564867946398518773838135462631263
8715021457626256604421498672689922037581189782669720929851520043336389581897697080761078677361290939377916116
2863123008981060444546443373854784384587322355416424408889079481775900355963713561860651230081957925776362139 7
4177444253679642265761809750509171038956777913163867744614104397684640829005752043638561098866740667398565399
8415459284292542809417293873999825237550145094518521285635786347086517513713001606130630295579671597174475782
2003972604536562732043480172072776993434181798609439247931026469242916117370029366763983246770167409812624184
1663846430214286868896345640857942267688193458411341412386024226156875376940834905875047333301991331454833944
5246920742365616167987042289392669762905419442965565972867201631928412412817836498244917125560891966224918607
1305418614721019688405350200039317355377241493904399242515482083705134055341284246839272273133276260084184203
4660545257313115947918947038651804895179985587809224916747988654533856744991877319405787786616330018753276229
7044169961295892629938651479028665686211691766299081932985033508846552448232713809864257545002056779376796084
7748395707455127252868905694909950205626775117943303617286764584276196269320710425764671817579528156410463614
3168889224520158611376110894599714732366361831172777988920427556294258429030410722338908230242497186197507934
1755661616499956279713728464122372961327213352163898975444006568773963395633979254452700536689305357871842350
0456189049404434893233354052698579657341482453960556502250534931069616939499339564519416961761228053374702443
0155524561972811250229982506945870227409067706099754310313158874710846580062233059950384324091239983783681881
0904240382847917065834181133819545837963826486455565618349884434451303179334718270566824471245198959504540006
8418973617631046574437215741075921433947879573643239430295564677263730451305834155987967606113196561051177386
6798319900043384032579517622732849720167833732130363612628928794160637815818619658984664396144546242683228874
3715580189170506178902047287554178409372168167608194067336077557362433258386153748257644713712957886909441872
9008592550113226978801632003071640745446199569867708730970185851515370802281025148034439255499747119929504617
0822643650731178043906621133073961175681280329778256836880162497642774803658255100148777659225477226088611107
0488712524010747143808350104330958507041911196080151768424782385947050983941353380148725284167413258633 38
Time of DP: 0.21677875518798828
```

分析：根据对比，当 n<30 时，两种方法耗时大致相同，几乎都为 0；当 n>40 后，两种方法优劣开始显现：递归方法耗时明显大大增加，而动态规划耗时情况比较乐观；当 n>100 后，递归方法直接崩溃，而动态规划仍然相当高效。

因为**递归方法**在计算结果时存在大量**重复子问题**，即计算 f(n)时会计算 f(n-1)和 f(n-2)，计算 f(n-1)时会计算 f(n-2)和 f(n-3)，计算 f(n-2)时会再次计算一遍 f(n-2)。而**动态规划**方法将所有**子问题的结果记录在列表中**，不会出现重复计算问题。由此可见，递归重复计算了大量重复问题，因此当 n 较大时耗时较长；而动态规划记录重复子问题，

甚至 n~10^5 也可以快速计算出答案。

2）买卖股票

股票买卖显然需要先买再卖，每天的状态只有手里有股票和无股票两种，设 dp[i][0]表示第 i 天手里没有股票可得的最大收益，dp[i][1]表示第 i 天手里有股票可得的最大收益(手中股票不能折现，第一次买入后手中收益为负)。每天的收益均由前一天决定，若**当天手中有股票**，则可能**今天买入**或**前一天手中有股票**；若**当天手中没有股票**，则可能**今天卖出或者前一天手中没股票**。显然有递推关系式：

dp[i][0]=max{dp[i-1][0],dp[i-1][1]+price[i]}

dp[i][1]=max{dp[i-1][1],dp[i-1][0]-price[i]}

下给出几组不同样例，测试结果如下：

样例 1（可以交易两次）：

```
6
7
1
3
5
4
6
profit: 6
```

样例 2（只能交易一次，**手里不能同时握有 2 支股票**）：

```
5
7
4
2
5
7
profit: 5
```

样例 3（不交易，因为**股价一直跌**）：

```
3
4
2
1
profit: 0
```

样例 4（多次交易）：

```
6
2
4
1
6
5
7
profit: 9
```

对于样例 1，第 2 天买第 4 天卖，第 5 天买第 6 天卖，可以赚 5-1+6-4=6；对于样例 2，手中不能同时握有两只股票（即第 2 天买第 4 天卖，第 3 天买第 5 天卖），只能

第 3 天买第 5 天卖赚 7-2=5；对于样例 3，因为股价一直跌，所以不买卖，收益为 0；对于样例 4，第 1 天买第 2 天卖，第 3 天买第 4 天卖，第 5 天买第 6 天卖，可以赚 4-2+6-1+7-5=9。

2. 程序调试：

1）对于爬楼梯问题，分别用递归和动态规划求解，对比不同数据量下的执行时间。N<30 情况下执行时间近似相同；N 接近 50 时递归法执行相当缓慢，动态规划仍然很快；N 接近 100 时递归算法崩溃，动态规划仍然不受影响，甚至 **N 达到 10^5 规模动态规划仍然可以在相当短的时间内计算出结果**

2）对于买卖股票问题，一开始没有理清买卖之间的关系，误将当天价格与前一天弄错

3）其实**买卖股票问题不仅可以用动态规划，还可以用贪心算法**，每次只要赚钱就交易，具体代码见附录，也可以得到正确答案。相当于取局部最优，并且小的局部最优可以合并成大的大问题的最优，因此算法有效。

五、附录

贪心算法解决买卖股票问题：

```
'''买卖股票的最佳时机(允许无限次操作)
题目:给定一个数组,它的第 i 个元素是一支给定股票第 i 天的价格
    允许完成无数笔交易,设计一个算法来计算你所能获取的最大利润。
注意：你不能在买入股票前卖出股票。
输入样例:[1,2,3,5,2]
输出:4
思路:比较相邻的值,只要赚就买(同刘大师买股票)
'''
price=[]
n=int(input())
for i in range(n):
    p=int(input())
    price.append(p)
profit=0
for i in range(n-1):
    if(price[i+1]>price[i]):
        profit+=price[i+1]-price[i]
print("Max profit:",profit)
```

运行结果：

```
6
7
1
3
5
4
6
Max profit: 6
```

```
5
7
4
2
5
7
Max profit: 5
```

```
3
4
2
1
Max profit: 0
```