

Assignment #C: bfs & dp

Updated 1436 GMT+8 Nov 25, 2025

2025 fall, Complied by 韩旭 元培学院

说明:

- 1) 请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora <https://typoraio.cn>，或者用word）。AC或者没有AC，都请标上每个题目大致花费时间。
- 2) 提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、“作业评论”区有上传的md或者doc附件。
- 3) 如果不能在截止前提交作业，请写明原因。

1. 题目

sy321迷宫最短路径

bfs, <https://sunnywhy.com/sfbj/8/2/321>

用时: 20min

思路:

这道题是经典的bfs走迷宫的变式，在普通版本的基础上需要对bfs找出的最短路径进行记录。这里仍然需要注意普通版本的一些事项，比如保护圈、`inq`集合记录已经进入队列的点等。这里面使用了`prev`矩阵来记录路径，思路类似并查集，即在队列中某节点出队列后，将其记录为下一步能到达的所有节点的父节点。这样，最后到达(n,m)点返回后，在`prev`矩阵中逐渐上溯到(1,1)再反转（用`path.reverse()`函数）即可得到相应路径。

代码:

```
from collections import deque

n,m=map(int,input().split())
maze=[[-1]*(m+2)]+[[[-1]+list(map(int,input().split()))+[-1] for _ in range(n)]+[[[-1]]*(m+2)]]
dx=[-1,1,0,0]
dy=[0,0,-1,1]

def bfs(x,y,n,m,maze):
    queue=deque([(x,y)])
    inq={(x,y)}
    prev=[[False]*(m+2) for _ in range(n+2)]

    while queue:
        cur_x,cur_y=queue.popleft()
```

```

if cur_x==n and cur_y==m:
    return prev
for i in range(4):
    next_x=cur_x+dx[i]
    next_y=cur_y+dy[i]
    if maze[next_x][next_y]==0 and (next_x,next_y) not in inq:
        queue.append((next_x,next_y))
        inq.add((next_x,next_y))
        prev[next_x][next_y]=(cur_x,cur_y)

prev=bfs(1,1,n,m,maze)
path=[(n,m)]
tmp_x,tmp_y=n,m
while tmp_x!=1 or tmp_y!=1:
    path.append(prev[tmp_x][tmp_y])
    tmp_x,tmp_y=prev[tmp_x][tmp_y]
path.reverse()

for x,y in path:
    print(x,y)

```

代码运行截图

题目
题解

代码书写
Python

迷宫最短路径

通过数 1646 提交数 3330 难度 中等 显示标签 ⭐

题目描述

现有一个 $n \times m$ 大小的迷宫，其中 1 表示不可通过的墙壁，0 表示平地。每次移动只能向上下左右移动一格，且只能移动到平地上。假设左上角坐标是(1,1)，行数增加的方向为 x 增长的方向，列数增加的方向为 y 增长的方向，求从迷宫左上角到右下角的最少步数的路径。

输入描述

第一行两个整数 n, m ($2 \leq n \leq 100, 2 \leq m \leq 100$)，分别表示迷宫的行数和列数；接下来 n 行，每行 m 个整数（值为 0 或 1），表示迷宫。

输出描述

从左上角的坐标开始，输出若干行（每行两个整数，表示一个坐标），直到右下角的坐标。
数据保证最少步数的路径存在且唯一。

样例1

输入 复制	3 3	0 1 0
-------	-----	-------

收起面板
运行
提交

```

1  from collections import deque
2
3  n,m=map(int,input().split())
4  maze=[[-1]*(m+2)]+[[[-1]+list(map(int,input().split()))+[-1] for _ in range(n+2)]]
5  dx=[-1,1,0,0]
6  dy=[0,0,-1,1]
7
8  def bfs(x,y,n,m,maze):
9      queue=deque([(x,y)])
10     inq={(x,y)}
11     prev=[[False]*(m+2) for _ in range(n+2)]
12
13     while queue:
14         cur_x,cur_y=queue.popleft()
15         if cur_x==n and cur_y==m:
16             return prev
17         for i in range(4):
18             next_x=cur_x+dx[i]
19             next_y=cur_y+dy[i]
20             if maze[next_x][next_y]==0 and (next_x,next_y) not in inq:
21                 queue.append((next_x,next_y))
22                 inq.add((next_x,next_y))
23                 prev[next_x][next_y]=(cur_x,cur_y)

```

测试输入
提交结果
历史提交

完美通过 查看题解

100% 数据通过测试 详情
运行时长: 0 ms

sy324多终点迷宫问题

bfs, <https://sunnywhy.com/sfbj/8/2/324>

用时：10min

思路：

这道题也是经典的bfs走迷宫的变式，在普通版本的基础上需要对到达每个点的最短路径进行记录。由于本题输出的是路径长度，所以保存到队列里面的元组要包含目前的步数。bfs的特性保证了只要完整地跑完这个算法，就可以遍历迷宫中所有可达的点，且到达每个点的时候都是最短路径。因此只需要在每个节点出队列的时候，将对应的步长记录在 `ans` 矩阵中即可，没有被记录的节点默认为-1（不可达到）。

代码：

```
from collections import deque

n,m=map(int,input().split())
maze=[[-1]* (m+2)]+[[[-1]+list(map(int,input().split()))+[-1] for _ in range(n)]+[[[-1]]*(m+2)]]
dx=[-1,1,0,0]
dy=[0,0,-1,1]
ans=[[-1]* (m+2) for _ in range(n+2)]

def bfs(x,y,maze):
    queue=deque([(0,x,y)])
    inq={(x,y)}
    while queue:
        step,cur_x,cur_y=queue.popleft()
        ans[cur_x][cur_y]=step
        for i in range(4):
            next_x=cur_x+dx[i]
            next_y=cur_y+dy[i]
            if maze[next_x][next_y]==0 and (next_x,next_y) not in inq:
                queue.append((step+1,next_x,next_y))
                inq.add((next_x,next_y))

prev=bfs(1,1,maze)
for i in range(1,n+1):
    print(" ".join(map(str,ans[i][1:m+1])))
```

代码运行截图

题目 题解

接下来 n 行，每行 m 个整数（值为 0 或 1），表示迷宫。

输出描述

输出 n 行 m 列个整数，表示从左上角到迷宫中每个位置需要的最小步数。如果无法到达，那么输出 -1。注意，整数之间用空格隔开，行末不允许有多余的空格。

样例1

输入 复制

```
3 3  
0 0 0  
1 0 0  
0 1 0
```

输出 复制

```
0 1 2  
-1 2 3  
-1 -1 4
```

解释

假设左上角坐标是(1,1)，行数增加的方向为 x 增长的方向，列数增加的方向为 y 增长的方向。

可以得到从左上角到所有点的前进路线：(1, 1)=>(2, 1)=>(2, 2)或(1, 3)=>(2, 3)=>(3, 3)。

左下角的三个位置无法到达。

代码书写

Python

```
3 n,m=map(int,input().split())  
4 maze=[[-1]*(m+2)]+[[[-1]+list(map(int,input().split()))+[-1] for  
5 dx=[-1,1,0,0]  
6 dy=[0,0,-1,1]  
7 ans=[[-1]*(m+2) for _ in range(n+2)]  
8  
9 def bfs(x,y,maze):  
10    queue=deque([(0,x,y)])  
11    inq={(x,y)}  
12    while queue:  
13        step,cur_x,cur_y=queue.popleft()  
14        ans[cur_x][cur_y]=step  
15        for i in range(4):  
16            next_x=cur_x+dx[i]  
17            next_y=cur_y+dy[i]  
18            if maze[next_x][next_y]==0 and (next_x,next_y) not in  
19                inq:  
20                queue.append((step+1,next_x,next_y))  
21                inq.add((next_x,next_y))  
22    prev=bfs(1,1,maze)  
23    for i in range(1,n+1):  
24        print(" ".join(map(str,ans[i][1:m+1])))
```

测试输入

提交结果

历史提交

查看题解

100% 数据通过测试

详情

运行时长: 0 ms

收起面板

运行

提交

M02945: 拦截导弹

dp, greedy <http://cs101.openjudge.cn/pctbook/M02945>

用时: 25min

思路

这道题等价于在导弹序列中找到单调递减（非严格）的最长子序列。我们按照子序列结尾进行dp， $dp[i]$ 表示以序号 i 的导弹结尾的子序列中，最长的单调递减子序列。那么对于 $dp[i+1]$ ，我们只需要遍历第0个到第 $i-1$ 个导弹，然后找到其中不比第 i 个导弹小的导弹序号 s ，再在 $1+dp[s]$ 中选取最大值即可。

代码:

```
k=int(input())  
missile=list(map(int,input().split()))  
dp=[0]*k  
for i in range(k):  
    if i==0:  
        dp[i]=1  
        continue  
    tmp=1  
    for s in range(i-1,-1,-1):
```

```

if missile[s] >= missile[i]:
    tmp=max(tmp,1+dp[s])
dp[i]=tmp

ans=0
for i in range(k):
    ans=max(ans,dp[i])
print(ans)

```

代码运行截图

#51103297提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```

k=int(input())
missile=list(map(int,input().split()))
dp=[0]*k
for i in range(k):
    if i==0:
        dp[i]=1
        continue
    tmp=1
    for s in range(i-1,-1,-1):
        if missile[s] >= missile[i]:
            tmp=max(tmp,1+dp[s])
    dp[i]=tmp

ans=0
for i in range(k):
    ans=max(ans,dp[i])
print(ans)

```

基本信息

#: 51103297
 题目: M02945
 提交人: 22n2200017737
 内存: 3652kB
 时间: 25ms
 语言: Python3
 提交时间: 2025-12-02 18:07:10

©2002-2022 POJ 京ICP备20010980号-1

English 帮助 关于

189A. Cut Ribbon

brute force/dp, 1300, <https://codeforces.com/problemset/problem/189/A>

用时: 10min

思路:

这道题根据ribbon的长度确定状态进行dp。对于某个长度*i*, 我们要找出其最多能按照a,b,c切成多少段。那么就遍历a,b,c, 以a为例, 考虑去掉a之后长度为*i-a*的ribbon能否按a,b,c切分, 以及如果能, 能够切多少段。所以这里的判断条件比较重要, 如果要更新 `dp[i]`, 要么是 `i-a==0`, 即切掉a后就完成了, 这时候更新为1; 要么是 `i-a>0` 但是 `dp[i-a]>0`, 即切掉a之后剩下的部分能够按照a,b,c切分, 这时候更新为 `dp[i-a]+1`。遍历a,b,c, 选出对应更新的 `dp[i]` 中的最大值即可。这里默认值设置成 `-float('inf')` 可能更好, 相当于区分清楚了“无法按a,b,c切分”和“0段”的情况, if条件更简单。

代码：

```
n,a,b,c=map(int,input().split())
dp=[0]*(n+1)
MIN=min(a,b,c)
for i in range(n+1):
    if i<MIN:
        continue
    if i==MIN:
        dp[i]=1
    for j in [a,b,c]:
        if (i-j>0 and dp[i-j]>0) or (i==j):
            dp[i]=max(dp[i],dp[i-j]+1)
print(dp[n])
```

代码运行截图

#	When	Who	Problem	Lang	Verdict	Time	Memory
351609531	Dec/02/2025 22:21 UTC+8	xuhanecon	189A - Cut Ribbon	Python 3	Accepted	62 ms	100 KB

M01384: Piggy-Bank

dp, <http://cs101.openjudge.cn/practice/01384/>

用时：25min

思路

这道题的整体思路和Cut Ribbon很像，但是Cut Ribbon是取最大值，所以默认值设置为0（或者`-float('inf')`更好），这里面是算最小值，所以默认值设置为`float('inf')`。注意这里面用coins数组整体存储values和weights的方式可以在循环时直接对coin数组循环避免使用索引，从而提高速度。

代码：

```
T=int(input())
for _ in range(T):
    E,F=map(int,input().split())
    W=F-E
    N=int(input())
    coins=[]
    for i in range(N):
        coins.append(tuple(map(int,input().split())))
    dp=[float('inf')]*(W+1)
    dp[0]=0
    for i in range(W+1):
        for v,w in coins:
```

```

if i-w>=0:
    dp[i]=min(dp[i],v+dp[i-w])
if dp[w]<float('inf'):
    print(f"The minimum amount of money in the piggy-bank is {dp[w]}")
else:
    print("This is impossible.")

```

代码运行截图

#51108344提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```

T=int(input())
for _ in range(T):
    E,F=map(int,input().split())
    W=F-E
    N=int(input())
    coins=[]
    for i in range(N):
        coins.append(tuple(map(int,input().split())))
    dp=[float('inf')]*(W+1)
    dp[0]=0
    for i in range(W+1):
        for v,w in coins:
            if i-w>=0:
                dp[i]=min(dp[i],v+dp[i-w])
    if dp[W]<float('inf'):
        print(f"The minimum amount of money in the piggy-bank is {dp[W]}")
    else:
        print("This is impossible.")

```

基本信息

#: 51108344
 题目: 01384
 提交人: 22n2200017737
 内存: 54348kB
 时间: 676ms
 语言: PyPy3
 提交时间: 2025-12-02 23:21:03

©2002-2022 POJ 京ICP备20010980号-1

English 帮助 关于

M02766: 最大子矩阵

dp, kadane, <http://cs101.openjudge.cn/pctbook/M02766>

用时: 20min

思路

首先遍历子矩阵的最上方行和最下方行 (`top` 和 `bottom`)，然后把各列和压缩成 `col_sum`，再使用Kadane算法即可。这里要特别注意计算 `col_sum` 的时候，不需要对每一对 `top` 和 `bottom` 都从头开始计算，而是使用类似前缀和的方式，在遍历 `bottom` 的时候依次在 `col_sum` 上把当前行的值进行累加，就可以得到当前从 `top` 到 `bottom` 的各列和。另外需要注意这个题目的输入比较奇怪，第一，需要限定输入的数字个数在 `n**2` 以内，然后用 `while` 循环进行不定行输入；第二，需要用一个列表生成式把 `nums` 切片成 `matrix`。

代码:

```

n=int(input())
nums=[]
while len(nums)<n**2:
    nums+=list(map(int,input().split()))
matrix=[nums[i*n:(i+1)*n] for i in range(n)]

def kadane(s):
    cur_max=total_max=s[0]
    for x in s[1:]:
        cur_max=max(x,cur_max+x)
        total_max=max(total_max,cur_max)
    return total_max

ans=0
for top in range(n):
    col_sum=[0]*n
    for bottom in range(top,n):
        for j in range(n):
            col_sum[j]+=matrix[bottom][j]
        ans=max(ans,kadane(col_sum))
print(ans)

```

代码运行截图

#51207246提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: Accepted

源代码

```

n=int(input())
nums=[]
while len(nums)<n**2:
    nums+=list(map(int,input().split()))
matrix=[nums[i*n:(i+1)*n] for i in range(n)]

def kadane(s):
    cur_max=total_max=s[0]
    for x in s[1:]:
        cur_max=max(x,cur_max+x)
        total_max=max(total_max,cur_max)
    return total_max

ans=0
for top in range(n):
    col_sum=[0]*n
    for bottom in range(top,n):
        for j in range(n):
            col_sum[j]+=matrix[bottom][j]
        ans=max(ans,kadane(col_sum))
print(ans)

```

基本信息

#: 51207246
 题目: M02766
 提交人: 22n2200017737
 内存: 3804kB
 时间: 245ms
 语言: Python3
 提交时间: 2025-12-09 16:11:46

2. 学习总结和收获

本次作业包括 BFS 与 DP 两个部分。

在BFS部分，通过两道迷宫题，我更加熟悉了BFS的模版写法，同时也认识到BFS在进入队列的元素、需要记录的内容等方面具有一定的**灵活性**，这方面和前面练习的DFS（比如全排列的各种变式）很像，比如所需要的结果有时候在进入队列的元素里面递推地记录，有时候又可以在一个全局变量或数据结构中记录。尤其在第一题中，通过维护 `prev` 矩阵实现最短路径恢复，思路和**并查集**的想法很像，是一种**用链表构造图关系**的方法。与此同时，我也关注到了BFS中的常见技巧，例如保护圈、`inq` 集合/数组去重等。

在DP部分，我发现很多dp的题型都可以归结为某些经典的例题，比如说拦截导弹问题实际上是 **LIS 的递减版**，关键仍然在于围绕“以 i 结尾”的子序列建立局部最优。Cut Ribbon 与 Piggy Bank 本质上都是**完全背包问题**，其核心就是允许同一物品重复使用，并且在更新 dp 时只需考虑 $dp[i - w]$ 是否可行。但它们分别是取最大值与取最小值的版本，从而需要不同的默认值与判断逻辑（如 `float('inf')`）。