最短経路系(最長経路の場合、距離を-1倍する)

ベルマンフォード法

1. def BellmanFord(edges,V,source):
2. INF = float("inf")
3. dist = [INF for \_ in range(N)]
4. dist[source] = 0
6. for i in range(V):
7. for now,fol,d in edges:
8. if dist[now] != INF and dist[fol] > dist[now]+d:
9. dist[fol] = dist[now]+d
10. if i == V-1 and fol == N-1:
11. return "inf"
12. return dist[-1]

閉路を探索する場合、fol == N-1がいらない

ゴールまでに閉路が存在するかしないかの探索の場合、fol == N-1が必要

ダイクストラ法

import heapq

import sys

input = sys.stdin.readline

def main():

N,M = map(int,input().split())

INF = 10\*\*10

root = [[INF for \_ in range(N)] for \_ in range(N)]

for \_ in range(M):

a,b,c = map(int,input().split())

root[a-1][b-1] = c

root[b-1][a-1] = c

dist = [INF for \_ in range(N)]

prev = [-1 for \_ in range(N)]　　　＃通過ルート復元用

que = [(0,0)]

dist[0] = 0

while que:

dif,now = heapq.heappop(que)

for following in range(N):

if (root[now][following] != INF and dist[following] > dist[now] + root[now][following]):

dist[following] = dist[now] + root[now][following]

heapq.heappush(que,(dist[following],following))

prev[following] = now

print(dist)

goal = N-1

go = [goal+1]

while (goal != -1):

go.append(prev[goal]+1)

goal = prev[goal]

go.reverse()

print(\*go[1:])

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. def dij(s):
2. dist = [INF for \_ in range(N)]
3. use = [False for \_ in range(N)]
4. dist[s] = 0
5. q = [(0,s)]
6. heapq.heapify(q)
7. while q:
8. d,now = heapq.heappop(q)
9. if use[now]:
10. continue
11. use[now] = True
12. for w,fol in edge[now]:
13. if dist[now]+w < dist[fol]:
14. dist[fol] = dist[now]+w
15. heapq.heappush(q,(dist[fol],fol))
16. return dist

ワーシャルフロイド法

#頂点数N個に対してO（N＾3）で全頂点の最短距離を求めることが可能

from scipy.sparse.csgraph import floyd\_warshall を使うと速い

import sys

input = sys.stdin.readline

def main():

N,M = map(int,input().split())

INF =float(‘INF’)　 #←適宜変える

l = [[INF for \_ in range(N)] for \_ in range(N)]

for i in range(M):

a,b,c = map(int,input().split())

l[a-1][b-1] = c

l[b-1][a-1] = c

for i in range(N):

l[i][i] = 0

for i in range(N): #通過点

for j in range(N): #出発点

for k in range(N): #到達点

if (l[j][k] > l[j][i]+l[i][k]):

l[j][k] = l[j][i]+l[i][k]

print(l)

if \_\_name\_\_ == "\_\_main\_\_":

main()

numpyを利用したfloyd\_warshall

1. import sys
2. input = sys.stdin.readline
3. import numpy as np
4. from scipy.sparse.csgraph import floyd\_warshall
6. def main():
7. N,M = map(int,input().split())
8. edge = np.zeros((N,N),dtype=int)
9. for \_ in range(M):
10. a,b,c = map(int,input().split())
11. edge[a-1,b-1] = c
12. edge[b-1,a-1] = c
14. G = floyd\_warshall(edge)

最小全域木(プリム法)

import heapq

import sys

input = sys.stdin.readline

def main():

N,M = map(int,input().split())

edge = [[] for \_ in range(N)]

for \_ in range(M):

a,b,c = map(int,input().split())

edge[a-1].append((c,b-1))

edge[b-1].append((c,a-1))

go = [True for \_ in range(N)]

go[0] = False

queue = edge[0]

ans = 0

while queue:

cost,following = heapq.heappop(queue)

if not go[following]:

continue

go[following] = False

for e in edge[following]:

if go[e[1]]:

heapq.heappush(queue,e)

ans += cost

print(ans)

if \_\_name\_\_ == "\_\_main\_\_":

main()

クラスカル法

import sys

input = sys.stdin.readline

def main():

N,M = map(int,input().split())

way = []

for \_ in range(M):

a,b,c = map(int,input().split())

way.append((c,a-1,b-1))

I = [i for i in range(N)]

rank = [1 for \_ in range(N)]

def root(x):

if x == I[x]:

return x

I[x] = root(I[x])

return I[x]

def unite(x,y): #uniteでは親と親同士をつなげていく

px,py = root(x),root(y)

rx,ry = rank[px],rank[py]

if rx > ry:

I[py] = px #小さい方をつける

else:

I[px] = py

if rx == ry:

rank[py] += 1

ans = 0

way.sort()

for i in range(M):

print(I)

nx,ny = root(way[i][1]),root(way[i][2])

print(way[i],(nx,ny))

if nx != ny:

ans += way[i][0]

unite(way[i][1],way[i][2])

print(I)

print(ans)

if \_\_name\_\_ == "\_\_main\_\_":

main()

グラフ接続・切断

＃グラフを切断するより後ろからつなげていって逆に出力がやりやすい

N, M = map(int, input().split())

I = [[i, 1] for i in range(N)]　＃左が親、右が属する頂点の数

B = [0] \* M

for i in range(M):

B[M - 1 - i] = [int(x) - 1 for x in input().split()]

def root(a):

if I[a][0] == a:

return a

else:

I[a][0] = root(I[a][0]) ＃親を決める

return I[a][0]

def uni(a,b):

ra = root(a)

rb = root(b)

na = I[ra][1]

nb = I[rb][1]

if ra == rb:

return 0

if ra < rb:

I[rb][0] = ra

I[ra][1] += I[rb][1]

else:

I[ra][0] = rb

I[rb][1] += I[ra][1]

return na\*nb

NC = [0]\*M

NC[-1] = N\*(N-1)//2　＃すべて接続すると頂点すべて行き来可能

for i in range(M-1):

a,b = B[i]

NC[M-2-i] = NC[M-1-i]-uni(a,b)

print(I)

for nc in NC:

print(nc)

Union-Find

1. I = [[i,1] for i in range(N)]
3. def root(a):
4. if I[a][0] == a:
5. return a
6. else:
7. I[a][0] = root(I[a][0])
8. return I[a][0]
10. def unite(a,b):
11. ra = root(a)
12. rb = root(b)
13. na = I[ra][1]
14. nb = I[rb][1]
15. if ra < rb:
16. I[rb][0] = ra
17. I[ra][1] += I[rb][1]
18. else:
19. I[ra][0] = rb
20. I[rb][1] += I[ra][1]

グラフ接続

import sys

input = sys.stdin.readline

def main():

N,M = map(int,input().split())

road = [[] for \_ in range(N)]

for i in range(M):

a,b = map(int,input().split())

road[a-1].append(b-1)

road[b-1].append(a-1)

go = [False for \_ in range(N)]

go[0] = True

queue = [0]

while queue:

now = queue.pop()

for i in road[now]:

if not go[i]:

go[i] = True #←通過していない地点は通過後にappend

queue.append(i)

if all(go):

print("POSSIBLE")

else:

print("IMPOSSIBLE")

if \_\_name\_\_ == "\_\_main\_\_":

main()

逆元リスト、二項係数 N = 10＾5が0.19sec程度

1. import sys
2. input = sys.stdin.buffer.readline
3. import copy
5. def main():
6. N,K = map(int,input().split())
7. MOD = 10\*\*9+7
9. fac = [0 for \_ in range(2\*10\*\*5+1)]
10. fac[0],fac[1] = 1,1
11. inv = copy.deepcopy(fac)
12. invfac = copy.deepcopy(fac)
14. for i in range(2,2\*10\*\*5+1):
15. fac[i] = (fac[i-1]\*i)%MOD
16. inv[i] = MOD-(MOD//i)\*inv[MOD%i]%MOD
17. invfac[i] = (invfac[i-1]\*inv[i])%MOD #←逆元リスト
19. def coef(x,y): #←二項係数
20. num = (((fac[x]\*invfac[y])%MOD)\*invfac[x-y]%MOD)
21. return num
22. MOD = 998244353
23. fac = [0 for \_ in range(N+1)]
24. fac[0],fac[1] = 1,1
25. invfac = copy.deepcopy(fac)
27. for i in range(2,N+1):
28. fac[i] = (fac[i-1]\*i)%MOD
30. invfac[-1] = pow(fac[-1],MOD-2,MOD)
31. for i in range(N,0,-1):
32. invfac[i-1] = (invfac[i]\*i)%MOD
34. def coef(x,y):
35. num = (((fac[x]\*invfac[y])%MOD)\*invfac[x-y]%MOD)
36. return num

こっちのほうが早い

1. fac = [0 for \_ in range(N+1)]
2. fac[0],fac[1] = 1,1
4. for i in range(2,N+1):
5. fac[i] = (fac[i-1]\*i)%MOD
7. def comb(x,y):
8. return (fac[x+y]\*pow(fac[x],MOD-2,MOD)\*pow(fac[y],MOD-2,MOD))%MOD

もっとはやい

素数テーブル

1. def prime\_list(n):
2. prime = [True for \_ in range(n+1)]
3. prime[0] = False
4. prime[1] = False
5. for i in range(2,int(n\*\*0.5)+1):
6. if not prime[i]:
7. continue
8. for j in range(i\*2,n+1,i):
9. prime[j] = False
10. return [i for i in range(n+1) if prime[i]]

二次元累積和(numpy)

1. def main():
2. N = int(input())
3. a = np.array([[0 for \_ in range(N+1)]] + [[0] + list(map(int,input().split())) for \_ in range(N)])
4. cum = np.cumsum(np.cumsum(a,axis=0),axis=1)
6. make = np.zeros(N\*N+1,dtype=int)
7. for x in range(1, N+1):
8. for y in range(1, N+1):
9. val = cum[x:, y:] - cum[x:, :-y] - cum[:-x, y:] + cum[:-x, :-y]
10. make[x\*y] = max(make[x\*y],val.max())
12. np.maximum.accumulate(make,out=make)
13. q = int(input())
14. for \_ in range(q):
15. print(make[int(input())])

dp (use numpy)

1. import sys
2. input = sys.stdin.buffer.readline
3. from operator import itemgetter
4. import numpy as np
6. def main():
7. N,T = map(int,input().split())
8. food = []
9. for \_ in range(N):
10. a,b = map(int,input().split())
11. food.append((a,b))
13. dp = np.zeros(T,dtype=int)
14. food.sort(key = itemgetter(0))
16. ans = 0
17. for a,b in food:
18. ans = max(ans,dp[-1]+b)
19. dp[a:] = np.maximum(dp[a:],dp[:-a]+b)
21. print(ans)
23. if \_\_name\_\_ == "\_\_main\_\_":
24. main()

numpyの場合、最大値の更新をするときはnp.maximum(a,b)を用いる

LCS(最長共通部分列)

1. def lcs(x,y):
2. lx,ly = len(x),len(y)
3. dp = [[0 for \_ in range(ly+1)] for \_ in range(lx+1)]
4. for i in range(lx):
5. for j in range(ly):
6. if x[i] == y[j]:
7. p = 1
8. else:
9. p = 0
10. dp[i+1][j+1] = max(dp[i][j]+p,dp[i][j+1],dp[i+1][j])
11. return dp[-1][-1]

連続した部分のLCSの場合、dpよりs[i:j] in s[j:]のほうが早い

1. def main():
2. N = int(input())
3. s = input()
4. a,i,j = 0,0,1
6. while j < N:
7. if s[i:j] in s[j:]:
8. a = max(a,j-i)
9. j += 1
10. else:
11. i += 1
12. if i == j:
13. j += 1

二分グラフ

1. def bipartite\_graph(edge,N,M):
2. dist = [-1 for \_ in range(N)]
3. col = [0 for \_ in range(N)]
4. dist[0],col[0] = 0,1
5. q = deque([(0,1)])
6. while q:
7. now,nc = q.popleft()
8. for fol in edge[now]:
9. if col[fol] == 0:
10. col[fol] = -1\*nc
11. q.append((fol,-1\*nc))
12. elif col[fol] == nc:
13. return "This graph is not bipartite\_graph!"
14. return "This graph is bipartite\_graph"
15. dist = [-1 for \_ in range(N)]
16. col = [0 for \_ in range(N)]
17. dist[0],col[0] = 0,1
18. q = deque([(0,1)])
19. D = 0
20. while q:
21. now,nc = q.popleft()
22. for i in range(N):
23. if edge[now][i] == "1":
24. fol = i
25. if col[fol] == 0:
26. col[fol] = -1\*nc
27. q.append((fol,-1\*nc))
28. elif col[fol] == nc:
29. print(-1)
30. exit()

木dp

一番深いところからdpをしたいので再帰

1. dp = [[1,1] for \_ in range(N)]
2. def col(now,prev):
3. for fol in edge[now]:
4. if fol == prev:
5. continue
6. col(fol,now)
7. dp[now][0] \*= (dp[fol][0]+dp[fol][1])
8. dp[now][0] %= MOD
9. dp[now][1] \*= dp[fol][0]
10. dp[now][1] %= MOD
11. return

転倒数(Binary Indexed Tree)

def Binary\_Indexed\_Tree(a):

res = [0 for \_ in range(N)]

BIT = [0 for \_ in range(N+1)]

def BIT\_query(index):

res\_sum = 0

while index>0:

res\_sum += BIT[index]

index -= index&(-index)

return res\_sum

def BIT\_update(index,x):

while index<=N:

BIT[index] += x

index += index&(-index)

return

b = sorted(a)

for i,e in enumerate(a):

a[i] = bisect.bisect\_left(b,e)

Ai = [None for \_ in range(N)]

for i,e in enumerate(a):

Ai[e] = i

for i,e in enumerate(Ai):

res[i] = i-BIT\_query(e+1)

BIT\_update(e+1,1)

return res

a = list(map(int,input().split()))

print(Binary\_Indexed\_Tree(a))

bit DP(N!は大きいがは小さい時)

集合(すでに使った、すでに通過した等)をbitで保持する

1. dp = [1] + [0]\*(1<<N)
3. for i in range(N):
4. use = [0]\*((1<<N)+1)
5. for s in range(1<<N):
6. if dp[s] == 0:
7. continue
8. for j in range(N):
9. jb = 1<<j
10. if not (s&jb) and a[i][j]:
11. use[s^jb] += dp[s]
12. dp = use

Segment Tree

1. def segfunc(x,y): #segfunc(max,min,etc…)
2. return max(x,y)
4. def init(init\_val): #配列の初期化
5. for i in range(N+1):
6. seg[i+num-1] = init\_val[i]
7. for i in range(num-2,-1,-1):
8. seg[i] = max(seg[2\*i+1],seg[2\*i+2])
10. def update(k,x): #update
11. k += num-1
12. seg[k] = x
13. while k:
14. k = (k-1)//2
15. seg[k] = segfunc(seg[2\*k+1],seg[2\*k+2])
17. def query(p,q): #return ans
18. if q <= p:
19. return ide\_ele
20. p += num-1
21. q += num-2
22. ret = ide\_ele
23. while q-p>1:
24. if p&1 == 0:
25. ret = segfunc(ret,seg[p])
26. if q&1 == 1:
27. ret = segfunc(ret,seg[q])
28. q -= 1
29. p = p//2
30. q = (q-1)//2
31. ret = segfunc(segfunc(ret,seg[p]),seg[q])
32. return ret
34. ide\_ele = 0
35. num = 2\*\*N.bit\_length()
36. seg = [ide\_ele]\*2\*num