包含：

1. DFS
2. BFS
3. BFS轉DFS（跑到層數為 (總job數的一半)/2 時，轉DFS）

設定：

1. 以 Sum of c 的值做比較
2. 用SRPT的值，當還沒有長到tree最底部前，節點的估計值
3. 加入branch and bound 條件：（以bfs的code說明，dfs的大同小異）
   1) 若有n個已經排定的job，job i 接著排在第n+1的位置，
      當 其餘未排的job（不含job i ）中，可找到一個job j滿足，
      job j的release time< job i的release time，且
      cmax（第n+1個位置排job j）<cmax（第n+1個位置排job i）
      → 第n+1個位置放job i 的branch不往下長

```
'''condition 1'''
skip=False
for j in newToSet:
        if i[2]>j[2] and cmaxOfFixed>cmax(currentJob.fixed+[j]):
            #若目前head比toSet中的某節點晚到，且head+原fixed運作的時間 比 head+某點 的cmax慢
            skip=True
            break
if skip:
    continue
```

   - currentJob.fixed：已經排好的job（假設有n個）
   - job i：下一個要排的job（放在第n+1個）
   - job j：除了已經排好的job和job i，其他未排定的job
   - cmaxOfFixed： currentJob.fixed和job i 的cmax值

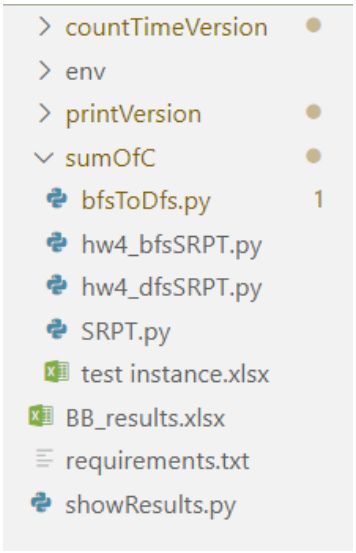
   2) 當已經長出的層數>=總job數的一半時，做判斷：
      若此時cmax（已確定排序的jobs）> 所有未排job中最大的release time
      →將job排序固定為：
      已排序的jobs + （未排序的jobs依processing time小到大排列），計算其sumOfC值

```
'''condition 2'''
if len(copyList)<=len(copyList+newFixed)/2 :
    temp=[[-job[2],job]for job in newToSet]
    hq.heapify(temp)
    if cmaxOfFixed>=-temp[0][0]:
        copyList.sort(key = lambda x: int(x[1]))
        currList=newFixed+copyList
        newValue=sumOfC(currList)
        if newValue<upperBound:
            record=currList
            upperBound=newValue
            updateUB+=1
        continue
```

   - copyList ：已固定順序的job
   - temp：用python 內建的heapq套件，依未排序job的released time建heap，取出最大released time

程式說明

1. 檔案：



- sumOfC module資料夾： //有完整的作業code
  內有三種B&B的程式碼，以及測試data的檔
  （以sum of c計算）
- showResults.py檔：
  若沒有error就繼續跑更多筆的資料做B&B ，直到三種B&B都遇到error則停止。
  將B&B結果，以及有error出現時的error message儲存到excel xlsx檔輸出（BB_results.xlsx）。
- requirements.txt：環境安裝了那些python package以及其版本。
- BB_results：以sumOfC資料夾中的code運行的結果（以sumOfC計算）。

- 註：countTimeVersion、printVersion資料夾中的是以cmax計算的B&B，其中printVersion會把B&B過程中做branch或delete branch的過程印出。

2. 執行showResults.py檔，會將運作結果印到BB_results.xlsx檔中。

3. sumOfC module資料夾：
   1) SRPT.py：
      - 做SRPT。
      - Job的格式 [‘job name ‘, processing time, released time] 。eg.[‘B’ ,8,0]
      - Input：固定好的job list、未固定的job list、以固定的job之cmax

        ```
        def SRPT(fixed,jobs,cmaxOfFixed):
        ```

      - Output：可中斷的最佳sum of c 時間
      - SRPT的heap：從index 為1開始排 //第0個擺0不會用到

        ```
        class MyHeap:
            minHeap = [0]
        ```

   2) hw4_dfsSRPT.py：DFS的B&B。用遞迴來做
      - `def dfs(toSet,fixed):`
        Input：[[toSet未固定的job list], [fixed已固定的job list]]
      - `def runDfs(number):`
        從測試資料中抓幾筆出來跑
      - `def dfsForBFS(toSet,fixed):`
        給bfs轉dfs叫的function，其output：

        ```
        return {'record':record, 'value':upperBound, 'nodeVisited':nodeVisited,'updateUB':updateUB}
        ```

   3) hw4_bfsSRPT.py： BFS的B&B
      - Priority queue的heap： `class MyHeap():`
      - Heap中node的att.：固定的job list、未固定的jobs、value : SRPT或sumOfC值

        ```
        class heapNode:
            fixed=[]
            toSet=[]
            value=0
        ```

   4) bfsToDfs.py：先做bfs，當固定job的層數達到一半的總job數時，轉成dfs

Excel檔之結果

Number of jobs	DFS_sumOfC	DFS_elapse time	DFS_nodeVisited	DFS UB_update	BFS_sumOfC	BFS elapse time	BFS_nodeVisited	BFS UB_update	bfsToDfs_su
5	129	0	12	1	129	0	12	1	129
6	171	0	30	3	171	0.000988722	30	3	171
7	245	0.000999451	57	6	245	0.000989437	57	6	245
8	331	0.000999689	108	10	331	0.000999212	103	7	331
9	408	0.000986338	188	17	408	0.001998663	171	9	408
10	502	0.00199914	310	24	502	0.001998425	265	11	502
11	588	0.002999067	530	35	588	0.003998518	457	14	588
12	704	0.007995605	1096	47	704	0.011974335	907	16	704
13	821	0.018990278	2580	67	821	0.024986267	2051	18	821
14	935	0.141451359	11512	88	935	0.170146704	9963	20	935
15	1069	0.070541143	22820	110	1069	0.087458134	14059	22	1069
16	1207	0.157909155	36318	132	1207	0.194869757	23709	23	1207
17	1343	0.209877968	58434	162	1343	0.252841473	36673	24	1343
18	1487	0.326430082	90752	199	1487	0.386425257	56579	26	1487
19	1644	0.664605379	147640	236	1644	0.782173634	94851	27	1644
20	1810	1.104408264	246766	273	1810	1.287429571	157125	28	1810
21	1994	1.657086849	394826	310	1994	1.879209042	245901	29	1994
22	2176	2.908415556	627040	348	2176	3.270186663	392603	30	2176
23	2372	5.103441954	1009372	386	2372	5.747971058	633839	32	2372
24	2595	6.298059225	1534194	424	2595	7.041495085	923493	34	2595
25	2789	10.20599127	2245458	464	2789	11.21866536	1357297	37	2789
26	3016	12.27351546	3184232	504	3016	13.44427848	1875395	40	3016
27	3223	37.92559028	5104172	548	3223	41.6150918	3303571	43	3223
28	3447	326.3667107	18015524	597	3447	356.8316684	14814959	46	3447
29	3707	745.0161326	55887285	668	3707	822.9827476	41293385	49	3707
30	3964	1899.788256	147578700	749	3964	2089.409964	106632188	51	3964
31	4229	2234.104987	289703630	829	4229	2432.295787	183661330	53	4229
32	4489	6461.690627	570114233	913	4489	6981.17794	387300863	58	4489