Advanced Computer Algorithms

Final project report

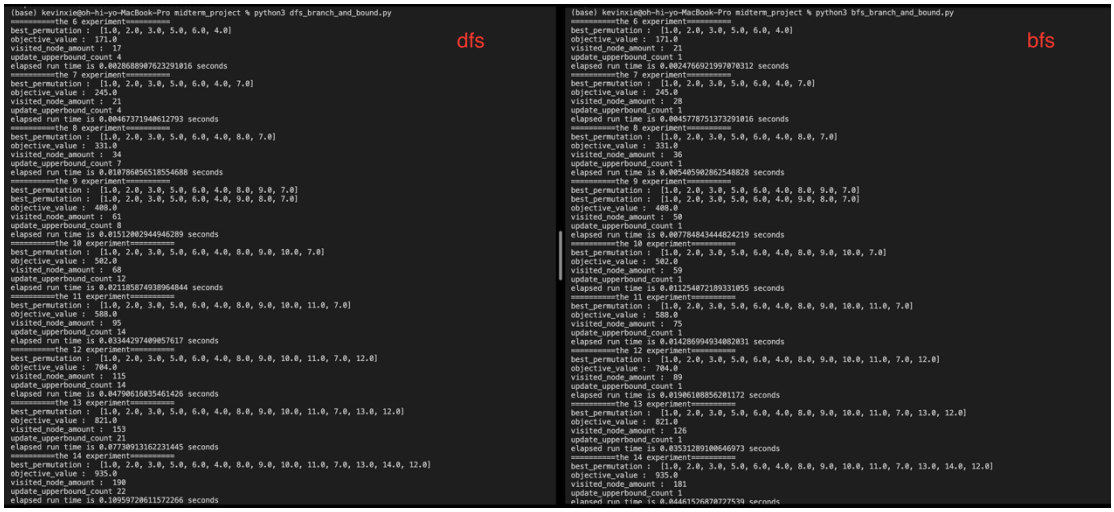
組員：309706029　謝敦凱

309706022　黃靖

309706033　康佑誠

1. 結果比較

6~14 個 job



15 個 job

20~50 個 job

Bfs:

```
(base) kevinxie@oh-hi-yo-MacBook-Pro midterm_project % python3 bfs_branch_and_bound.py
==========the 20 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0]
objective_value :  1810.0
visited_node_amount :  385
update_upperbound_count 1
elapsed run time is 0.1542677879333496 seconds
==========the 25 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 21.0,
24.0]
objective_value :  2789.0
visited_node_amount :  685
update_upperbound_count 1
elapsed run time is 0.33257389068603516 seconds
==========the 30 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 29.0, 24.0]
objective_value :  3964.0
visited_node_amount :  7462
update_upperbound_count 1
elapsed run time is 4.440095901489258 seconds
==========the 35 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 34.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 34.0, 33.0, 35.0, 24.0]
objective_value :  5444.0
visited_node_amount :  12954
update_upperbound_count 1
elapsed run time is 9.373600006103516 seconds
==========the 40 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 34.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 34.0, 33.0, 35.0, 24.0]
objective_value :  6945.0
visited_node_amount :  38105
update_upperbound_count 1
elapsed run time is 34.05823898315430 seconds
```

```
==========the 45 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 43.0, 44.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 44.0, 43.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 43.0, 44.0, 45.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 44.0, 43.0, 45.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 43.0, 44.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 44.0, 43.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 43.0, 44.0, 45.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 44.0, 43.0, 45.0, 33.0, 35.0, 24.0]
objective_value :  8571.0
visited_node_amount :  650056
update_upperbound_count 1
elapsed run time is 596.6961300373077 seconds
```

50 job

```
objective_value :  10663.0
visited_node_amount :  834585
update_upperbound_count 1
elapsed run time is 3853.6936490535736 seconds
```

Dfs:

```
(base) kevinxie@oh-hi-yo-MacBook-Pro midterm_project % python3 dfs_branch_and_bound.py
==========the 20 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0]
objective_value :  1810.0
visited_node_amount :  566
update_upperbound_count 36
elapsed run time is 0.7497751712799072 seconds
==========the 25 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 21.0,
24.0]
objective_value :  2789.0
visited_node_amount :  926
update_upperbound_count 41
elapsed run time is 2.766064167022705 seconds
==========the 30 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 29.0, 24.0]
objective_value :  3964.0
visited_node_amount :  7189
update_upperbound_count 103
elapsed run time is 37.190752029418945 seconds
==========the 35 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 34.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 34.0, 33.0, 35.0, 24.0]
objective_value :  5444.0
visited_node_amount :  12130
update_upperbound_count 150
elapsed run time is 93.85372805595398 seconds
==========the 40 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 34.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 33.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 40.0, 34.0, 33.0, 35.0, 24.0]
objective_value :  6945.0
visited_node_amount :  25421
update_upperbound_count 234
elapsed run time is 313.0979940891266 seconds
```

```
==========the 45 experiment==========
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 43.0, 44.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 44.0, 43.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 43.0, 44.0, 45.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 30.0, 31.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 44.0, 43.0, 45.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 33.0, 43.0, 44.0, 45.0, 34.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 43.0, 44.0, 45.0, 33.0, 35.0, 24.0]
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0, 17.0, 16.0, 18.0, 19.0, 20.0, 22.0, 23.0, 25.0, 26.0,
27.0, 21.0, 28.0, 31.0, 30.0, 32.0, 29.0, 37.0, 38.0, 36.0, 39.0, 41.0, 40.0, 42.0, 34.0, 44.0, 43.0, 45.0, 33.0, 35.0, 24.0]
objective_value :  8571.0
visited_node_amount :  835216
update_upperbound_count 438
elapsed run time is 7663.467152118683 seconds
```

50 job

```
objective_value :  10663.0
visited_node_amount :  863332
update_upperbound_count 578
elapsed run time is 13418.605674028397 seconds
```

討論：

當 node 數多的時候，DFS 明顯比 BFS 花更多時間，也是因為拜訪節點數的巨大差距所導致的。

節點數的差距來自 BFS 跟 DFS 的終止條件不同，在這 BFS 的運作是當 heap 內還有 lowerbound<=upperbound 才會繼續做下去，又因為 SRPT 把起始時間改變之後估計時間會多出很多，因為起始時間改成固定工作的最後一個完工時間，所以整個 SRPT 比起起始時間等於 0 開始估算的時間大上許多，會讓 lowerbound 很快就超越 upperbound 了，這個改進造成拜訪的節點數大幅減少。

圖示：

Bfs 15 個 jobs 的更新 upperbound 過程

```
(base) kevinxie@oh-hi-yo-MacBook-Pro final_project % python3 bfs_branch_and_bound.py
updated current_job_process_time 1051.0 lowerbound_now 2327.0 upperbound 9223372036854775807
updated current_job_process_time 1051.0 lowerbound_now 2128.0 upperbound 9223372036854775807
updated current_job_process_time 1051.0 lowerbound_now 1972.0 upperbound 9223372036854775807
updated current_job_process_time 1051.0 lowerbound_now 1815.0 upperbound 9223372036854775807
updated current_job_process_time 1051.0 lowerbound_now 1676.0 upperbound 9223372036854775807
updated current_job_process_time 1059.0 lowerbound_now 1517.0 upperbound 9223372036854775807
updated current_job_process_time 1059.0 lowerbound_now 1396.0 upperbound 9223372036854775807
updated current_job_process_time 1061.0 lowerbound_now 1324.0 upperbound 9223372036854775807
updated current_job_process_time 1063.0 lowerbound_now 1520.0 upperbound 9223372036854775807
updated current_job_process_time 1063.0 lowerbound_now 1242.0 upperbound 9223372036854775807
updated current_job_process_time 1063.0 lowerbound_now 1193.0 upperbound 9223372036854775807
updated current_job_process_time 1065.0 lowerbound_now 1130.0 upperbound 9223372036854775807
updated current_job_process_time 1065.0 lowerbound_now 1644.0 upperbound 9223372036854775807
updated current_job_process_time 1066.0 lowerbound_now 1531.0 upperbound 9223372036854775807
updated current_job_process_time 1067.0 lowerbound_now 1431.0 upperbound 9223372036854775807
updated current_job_process_time 1067.0 lowerbound_now 1100.0 upperbound 9223372036854775807
updated current_job_process_time 1067.0 lowerbound_now 1963.0 upperbound 9223372036854775807
updated current_job_process_time 1067.0 lowerbound_now 1821.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1684.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1396.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1327.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1308.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1255.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1198.0 upperbound 9223372036854775807
updated current_job_process_time 1068.0 lowerbound_now 1135.0 upperbound 9223372036854775807
updated current_job_process_time 1069.0 lowerbound_now 1320.0 upperbound 9223372036854775807
updated current_job_process_time 1069.0 lowerbound_now 1081.0 upperbound 9223372036854775807
updated current_job_process_time 1069.0 lowerbound_now 1069.0 upperbound 1069.0
updated current_job_process_time 1070.0 lowerbound_now 1528.0 upperbound 1069.0
best_permutation :  [1.0, 2.0, 3.0, 5.0, 6.0, 4.0, 8.0, 9.0, 10.0, 11.0, 7.0, 13.0, 14.0, 12.0, 15.0]
objective_value :  1069.0
visited_node_amount :  233
update_upperbound_count 1
elapsed run time is 0.06168794631958008 seconds
```

可以跟上面的 DFS 拜訪節點數比較，DFS:260, BFS:233，差距已經出來了，

而 job 數越高，拜訪的節點數會差更多。

DFS 會走這麼多節點的原因是：

DFS 繼續執行下去的條件設為 lowerbound <=upperbound，這跟 BFS 一樣，但 BFS 多了一個 heap 來維護全域最佳的 lowerbound，所以很快可以找到最佳的 upperbound，相比之下，DFS 就只能慢慢確認每個點，所以才會走比較多點。

實作細節：

1. DFS

這裡 dfs 有兩種，以下分別介紹：

```
dfs1(arr, l, r):
    # arr[0] : job list
    # arr[1] : job arrival time
    # arr[2] : job process time
    # arr[3] : completion time of each job
    if (l == r):   #  代表排到底了
        才去確認要不要更新 upperbound
    else: #排列組合
        # 介紹最重要的觀念
            for i in range(l, r+1):
                #l 行跟 i 行交換，代表 l 行以前都是固定的，以後都是不固
            定的
                swap_cols(arr, l, i)
                fixed_job = arr[:, :l + 1]    # 所以固定的 job 取到 l 行
                unfixed_job = arr[:, l + 1:]  # 不固定的 job 取 l+1 之後的行
```

```
                if upperbound >= lowerbound_now:    # 可以繼續做下去
                    dfs1(arr, l+1, r)
                swap_cols(arr, l, i)


dfs2(arr, all_seq, walked_seq):
    #介紹最重要的觀念
    leftest = 0  # 用來確認目前的排列是不是在點下一個 level 的最左邊
    for i in all_seq:
        # 只有每個分支的最左邊不需要取出 walked_seq 的最後一個 job，其他都要取出來，不然 walked job 會一直包含該分支最左邊的 job
        if leftest > 0:
            walked_seq = walked_seq[:len(walked_seq) - 1] + [i]
        else:
            walked_seq = walked_seq + [i]

        leftest += 1
        remain_seq = [x for x in total_job if x not in walked_seq] #拿到還沒排好的 job
```

重要的 function:

(1) calculate_fixed_time(arr): # 負責排列好的 job 的時間計算

return sum(job finish time), last_job_finish_time

(2) srpt_version1(arr, last_job_finish_time):

arr[0] : job list

arr[1] : job arrival time

arr[2] : job process time

arr[3] : completion time of each job

last_job_finish_time : the last job finish time

step1: 先把 arrival time <= current_time 的 push 進去 heap queue

step2: 確認一下目前是不是所以的 job 都 push 進去 heap，若是，就可以直接進行 srpt，做完後直接 return sum(job finish time)。

step3:若還有 job 還沒進入 heap，那就從之前紀錄的 start index 開始一個一個看 job，若 job 的 arrival time <= current time，就先 push 進去 heap，之後再做細部的運算，反之，job 的 arrival time > current time 的話，先把 idle time 算出來，再去做細部運算，當 idle time 處理完了，目前的 current time == arrival time，這時 job 才能 push 進去 heap。

補充：細部運算的概念是看下個 job 跟目前的 job 的抵達時間差了多少，這中間的時間差就是可以利用的時間(can_use_time)，然後我會取

得 queue 內最小的 process_time(命名為 current_job_process_time)

while can_use_time >= current_job_process_time:

就會把該 job pop 出來，因為他一定可以完工，然後取得他的 process_time(命名為 min_time)，再將 current_time 加上 min_time，can_use_time 減去 min_time， 然後把該 job 的 finish_time 記錄為 current_time，如果做到後面 queue 內沒有東西了，就 break，把 idle time 加上去 current_time，若 queue 還有東西，那就把最短剩餘工作時間的 job 的 process_time 減去剩下的 can_use_time，以上是細節的介紹。

step4:當每個 job 都走完了，都 push 入 heap，但 heap 內最後還是有東西，代表還沒排完，就把 heap 內的東西依序 pop 掉，然後時間的運算就是把 current_time 加上目前 pop 出去的 job 的剩餘工作時間，當有 job 被從 heap 內 pop 出去，代表他能完工，所以 current time 更新完後，就會記錄在該 job 的 completed time 的位置，重複以上動作，直到 queue 內沒東西就結束了。

(3) srpt_version2(arr, last_job_finish_time):

這個實作方法跟 version1（自己寫的）有點不同，他是去看 current_time 每次加 1 的時候 queue 內最小完工時間的 job 的 process time 是否為 0，為 0 代表他做完了，若不為 0 就把 process time 減 1。

i = 0 # record job index

while completed_job_amount != job_amount:

用完工的 job 數量當終止條件。

while i < job_amount and arrival_time <= current_time:

這裡是要看有沒有 job 的 arrival_time <= current_time，如果有就 push 進去 heap，那因為我是一個一個看 job，所以用 i 來紀錄目前走到哪個 job 的 index;

if len(process_queue) != 0 and process_queue[0][0] == 0:

這裡就是在看 queue 是不是全空了，如果還沒，且有 job 的 process_time == 0，那他就完成了，即用 current_time 紀錄他在目前的時間完工的，然後完工的 job 數量+1，這時 objective_value 加上 current_time。

if len(process_queue) != 0:

如果只符合這個條件，就讓 process_queue 最短的剩餘時間的 job 的 process_time 減 1

做完上面的動作後，current_time+=1，往下一個時刻去做以上的動作。

2.BFS

```
bfs(arr):
    # arr[0] : job list
    # arr[1] : job arrival time
    # arr[2] : job process time
    # arr[3] : completion time of each job
    # return lowerbound_count
    step1: 產生初始資料進去 heap，每個 job 當頭去估算 lowerbound，然後再 push[lowerbound, fixed_job]進去 heap。
    step2: while (current_job_process_time <= upperbound):
            #當 heap 內沒有更好的 lowerbound，就不繼續做下去了
                fixed_job = hq.heappop(queue)[1]
                #取得每一個 branch 的 fixed, unfixed job
                for i in total_job_list:
                    if i not in fixed_job:
                        walked_job = fixed_job + [i]
                        remain_job = [x for x in total_job_list if x not in
```

walked_job]

如果走到樹葉節點了，才去確認要不要更新 upperbound，

若還沒走到，直接把目前的[lowerbound, fixed_job]push 進去 heap