Algorithm

<Branch and Bound 0-1Knapsack>

Signature:

Signature:

Signature:

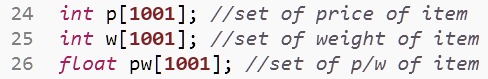
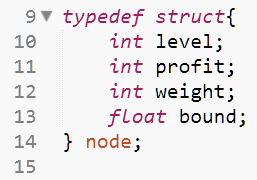
Participant No.1: 2017011976 박경리

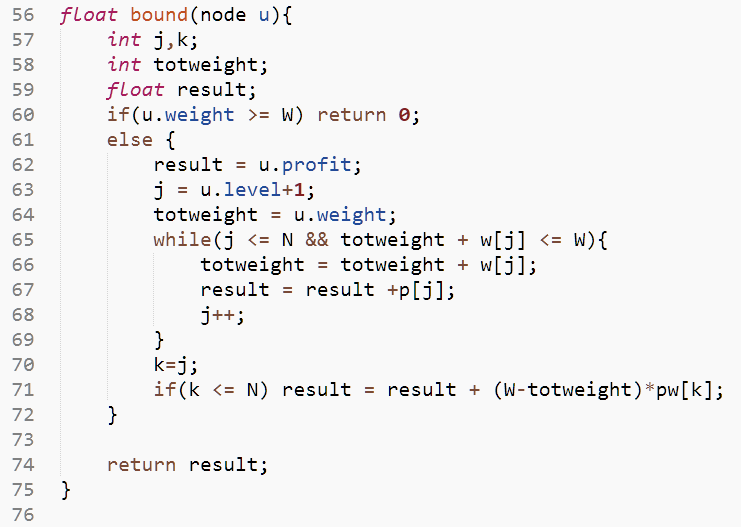
Participant No.2: 2017012033 배수연

Participant No.3: 2017012197 여채린

**1. Implement the branch-and-bound algorithm that solves the 0/1 knapsack problem. You need to implement 3 different versions: depth-first-search, breadth-first-search, and, best-first-search.**

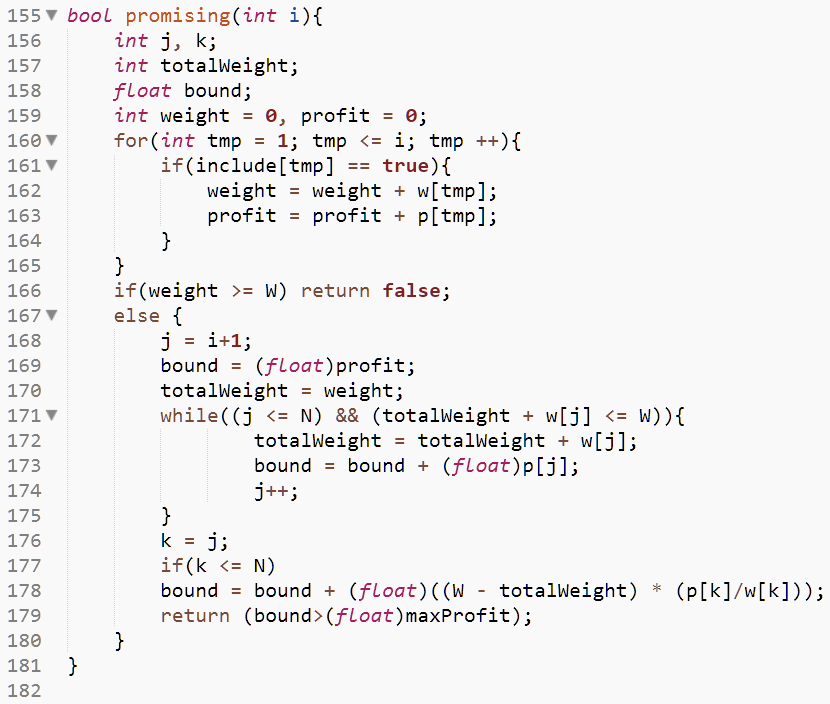
<Basic>



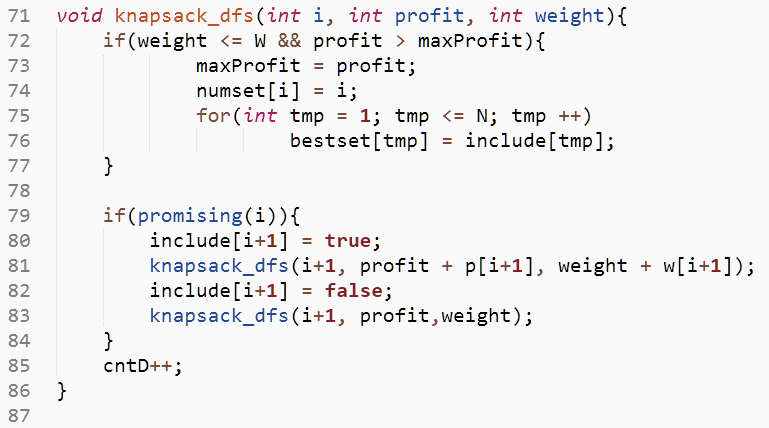


<Depth-First-Search>

-Check promissing

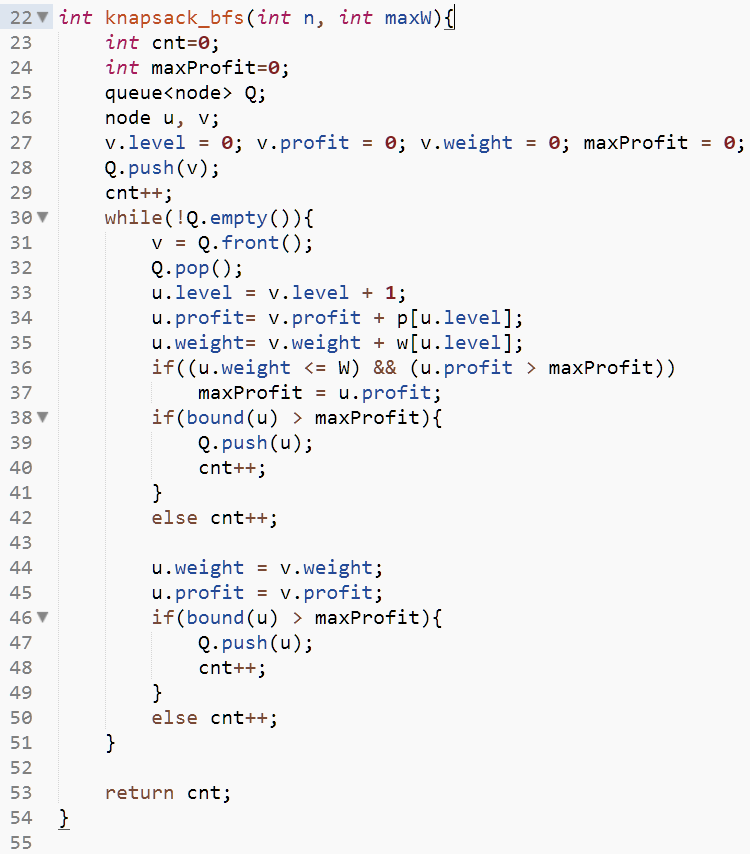


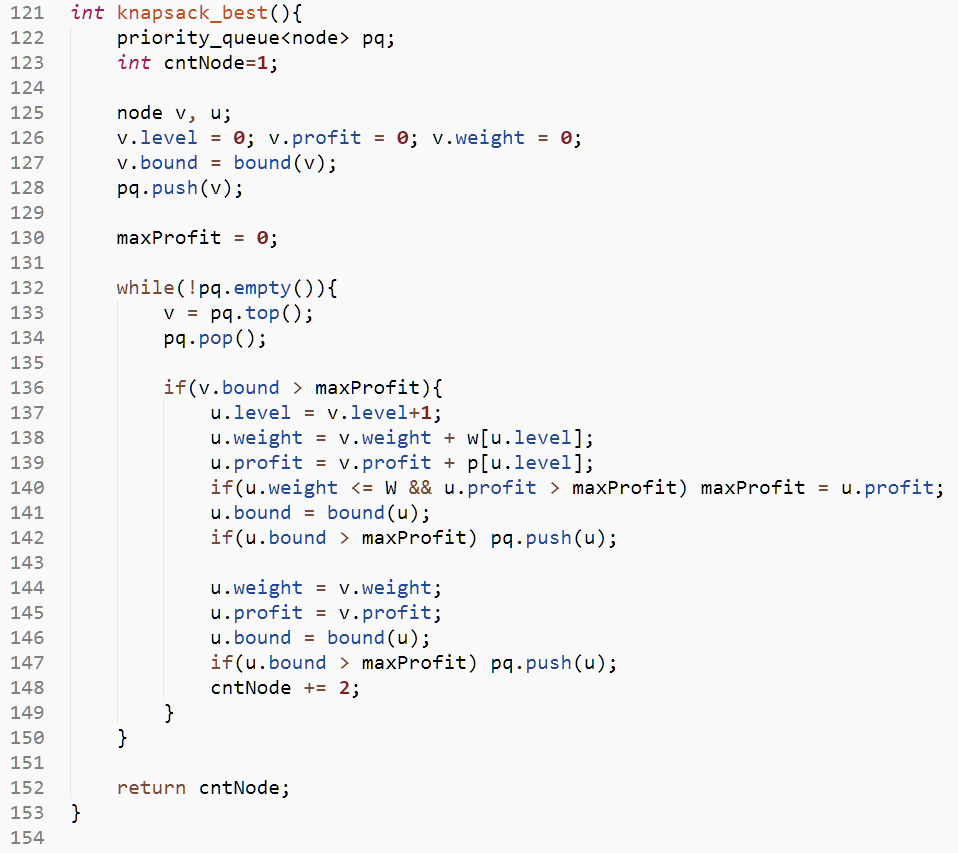
-knapsack



//cntD : count number of visited node

<Breadth-First-Search>

<Best-First-search>



**2-3. Check the number of nodes that you need to visit until finding the solution node for the Example 6.1 in the textbook. You should show me the numbers in depth-first, breadth-first, and best-first-searches, respectively.**

<Data Set >

- Size = 4

- Max Weight = 16

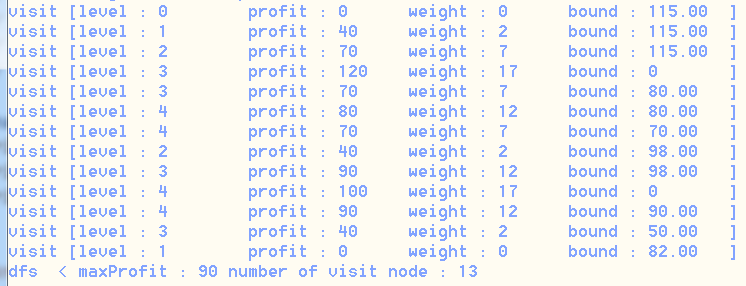
- Price set = {40, 30, 50, 10}

- Weight set = {2, 5, 10, 5}

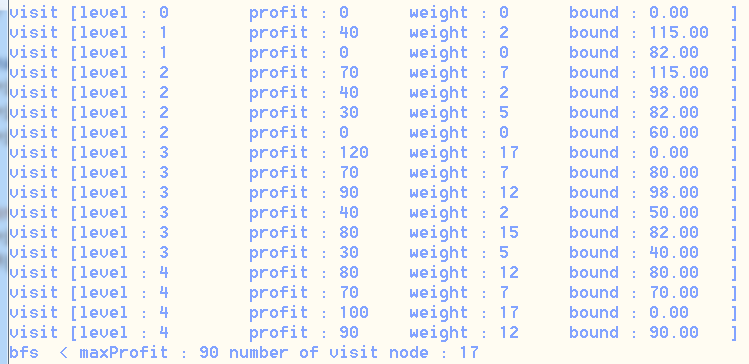
<Output>

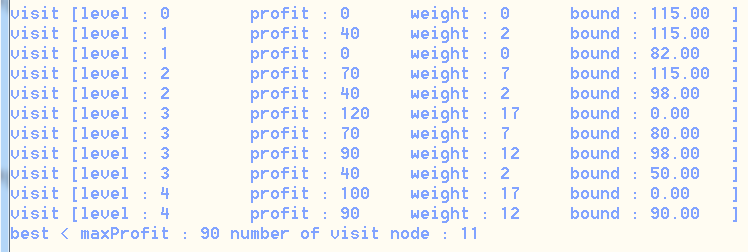
// Print the information of the nodes in the visited order

-DFS



-BFS



-BEST

**4. Do the same thing for the example in Exercises #1 in page 281. Again, you need to show me the numbers of three versions, respectively.**

<Data Set>

- Size = 5

- Max Weight = 13

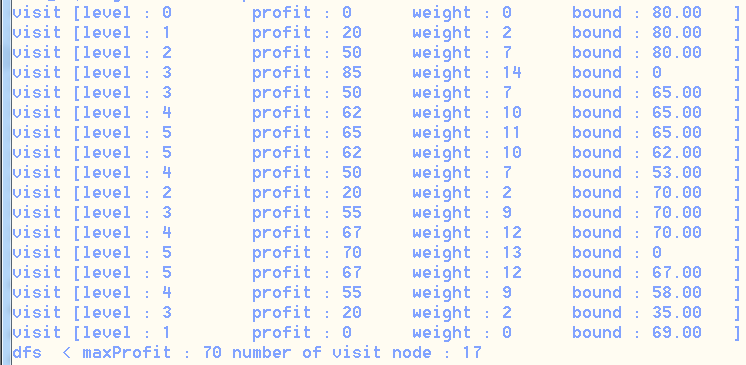
- Price set = {20, 30, 35, 12, 3}

- Weight set = {2, 5, 7, 3, 1}

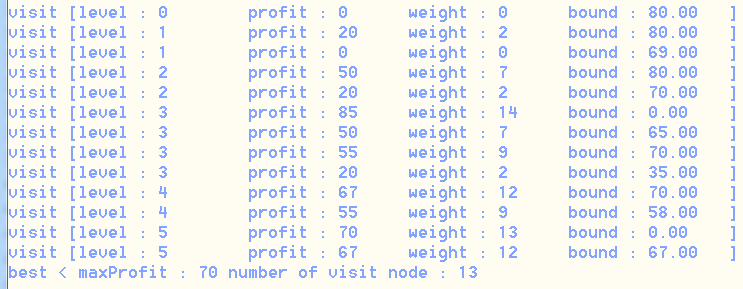
<Output>

// Print the information of the nodes in the visited order

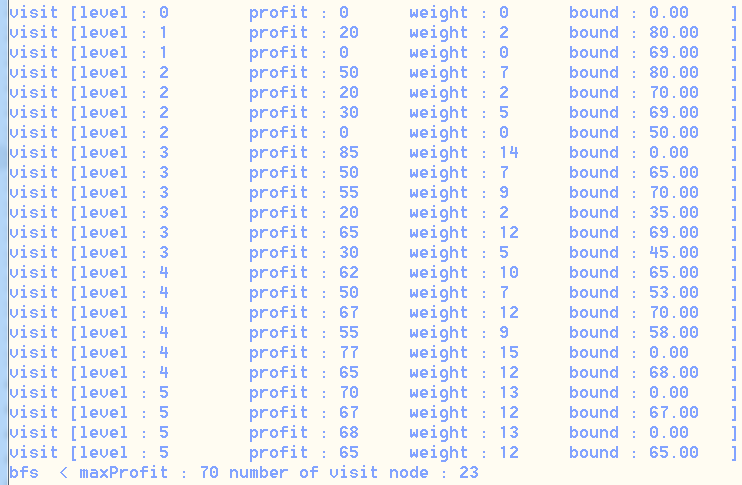
-DFS



-BFS



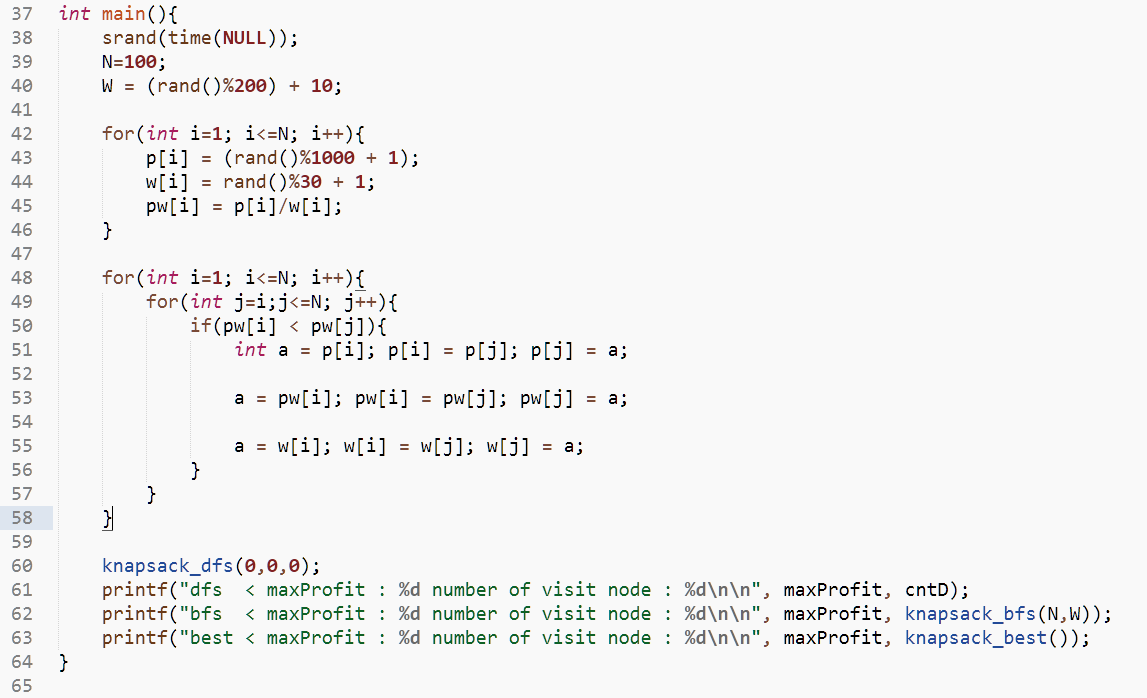
-BEST



**5. Make your own sets of problems and do the same experiments as many as you can. Try your best to come up with some interesting numbers that might impress me.**

CASE I) Max weight, profit set and weight set are random

<used code>



<The number of items is 100, run the code 10 times.>

**텍스트이(가) 표시된 사진



자동 생성된 설명 텍스트이(가) 표시된 사진



자동 생성된 설명**

<The number of items is 500, run the code 10 times.>

텍스트, 신문이(가) 표시된 사진



자동 생성된 설명 텍스트, 신문이(가) 표시된 사진



자동 생성된 설명

텍스트이(가) 표시된 사진



자동 생성된 설명사람, 텍스트이(가) 표시된 사진



자동 생성된 설명<The number of items is 800, run the code 10 times.>

텍스트이(가) 표시된 사진



자동 생성된 설명

<RESULT>

When there is no weight limit, since the weight limit value is random, the average number of nodes can not be obtained. However, the number of node visits increases in the order of best-first-search, depth-first-search, and breadth-first-search.

**CASE II) Max weight is fixed at 45, profit and weight are random**

<The number of items is 100, run the code 10 times.>

**텍스트, 신문이(가) 표시된 사진



자동 생성된 설명텍스트이(가) 표시된 사진



자동 생성된 설명**

****

<The number of items is 500, run the code 10 times.>

**텍스트, 신문이(가) 표시된 사진



자동 생성된 설명텍스트이(가) 표시된 사진



자동 생성된 설명**



<The number of items is 800, run the code 10 times.>

병, 실외이(가) 표시된 사진



자동 생성된 설명병, 텍스트이(가) 표시된 사진



자동 생성된 설명도로이(가) 표시된 사진



자동 생성된 설명

<RESULT>

When there is no weight limit, since the weight limit value is fixed, the average number of nodes can be obtained.

**CASE III) Our own special dataset.**

<Data set : Size=100 / Max Weight = 2000>

- Price set = { 648, 42, 908, 737, 208, 663, 770, 602, 976, 728, 170, 958, 492, 779, 864,

204, 587, 457, 268, 842, 849, 601, 535, 4, 650, 9, 464, 382, 591, 404, 611,

373, 340, 571, 740, 953, 895, 676, 727, 312, 117, 563, 652, 82, 956, 594,

942, 737, 479, 191, 659, 741, 303, 623, 360, 94, 913, 150, 342, 775, 496,

373, 985, 739, 921, 514, 570, 601, 339, 472, 845, 125, 824, 567, 108, 980,

59, 755, 481, 653, 946, 687, 99, 77, 896, 389, 985, 653, 398, 192, 653, 111,

537, 761, 568, 752, 247, 962, 71, 723

}

- Weight set = { 788, 81, 663, 427, 691, 868, 3, 104, 25, 615, 441, 788, 333, 972, 957, 147,

69, 37, 892, 788, 733, 923, 951, 498, 851, 199, 150, 531, 88, 46, 905, 188, 698, 911, 235, 84, 64, 951, 367, 203, 222, 63, 807, 97, 245, 847, 91, 911, 124, 508, 974, 331, 228, 621, 191, 953, 923, 424, 296, 188, 290, 223, 848, 337, 1, 751, 875, 467, 606, 838, 335, 569, 289, 181, 291, 437, 26, 828, 276, 681, 894, 626, 679, 39, 237, 525, 380, 28, 478, 506, 178, 689, 108, 931, 844, 73, 391, 703, 949, 427

}

- Output



<Data set : Size = 100 / Max Weight = 1000>

- Price set = { 200, 473, 486, 89, 497, 302, 88, 15, 9, 50, 176, 325, 446, 143, 71, 327, 76,

289, 174, 431, 275, 78, 337, 160, 434, 112, 349, 78, 338, 239, 284, 147,

206, 417, 27, 85, 210, 137, 191, 358, 187, 287, 45, 128, 310, 412, 221, 283,

132, 384, 27, 236, 108, 276, 457, 53, 99, 358, 456, 399, 228, 247, 382, 359,

436, 32, 323, 463, 80, 58, 177, 107, 147, 363, 410, 36, 66, 432, 172, 216,

34, 97, 419, 6, 249, 480, 392, 89, 195, 484, 297, 413, 178, 146, 418, 252,

100, 424, 214, 278

}

- Weight set = { 322, 45, 346, 270, 249, 344, 318, 346, 410, 343, 417, 387, 345, 281, 32,

437, 61, 176, 168, 495, 420, 148, 335, 61, 108, 2, 168, 456, 434, 445, 72,

498, 347, 405, 436, 498, 481, 346, 63, 125, 52, 84, 272, 439, 357, 94, 446,

89, 145, 219, 281, 40, 59, 436, 186, 64, 422, 347, 162, 496, 266, 355, 108,

258, 167, 389, 1, 143, 247, 35, 4, 87, 446, 287, 25, 246, 496, 294, 77, 345,

238, 39, 206, 495, 151, 187, 224, 322, 179, 165, 328, 13, 56, 190, 41, 202,

243, 340, 451, 109

}

- Output



<Data set : Size = 500 / Max Weight = 1000>

- Price set = { 661, 535, 3, 166, 792, 232, 902, 132, 161, 785, 77, 348, 815, 834, 439, 195,

702, 974, 775, 704, 450, 206, 822, 740, 997, 664, 577, 41, 59, 415, 42, 848,

952, 960, 953, 905, 348, 455, 409, 133, 162, 626, 803, 941, 228, 256, 886,

143, 788, 543, 165, 140, 208, 182, 201, 183, 507, 163, 567, 300, 96, 110,

478, 254, 748, 625, 970, 745, 229, 826, 917, 504, 352, 595, 504, 166, 508,

168, 97, 524, 273, 682, 422, 670, 224, 433, 603, 653, 634, 564, 878, 208,

102, 148, 345, 653, 462, 199, 223, 790, 741, 928, 459, 71, 850, 527, 135,

379, 416, 754, 390, 332, 902, 802, 228, 471, 742, 779, 63, 727, 558, 650,

785, 125, 52, 971, 930, 336, 821, 903, 606, 45, 66, 642, 870, 471, 375, 25,

40, 649, 860, 900, 576, 977, 244, 428, 329, 39, 351, 454, 309, 48, 168, 364,

307, 543, 561, 221, 978, 361, 832, 542, 455, 468, 696, 654, 274, 141, 82,

371, 570, 115, 343, 126, 392, 188, 527, 5, 826, 684, 882, 500, 144, 667,

540, 45, 166, 869, 559, 905, 861, 314, 246, 177, 106, 568, 681, 770, 98,

560, 178, 320, 461, 471, 923, 382, 510, 468, 293, 493, 426, 679, 555, 611,

322, 366, 999, 428, 693, 843, 790, 423, 818, 598, 723, 325, 530, 475, 679,

731, 809, 926, 477, 970, 630, 672, 894, 187, 3, 574, 963, 797, 320, 531,

759, 445, 524, 918, 743, 362, 779, 391, 404, 578, 328, 330, 999, 295, 600,

452, 376, 539, 711, 454, 217, 11, 551, 701, 13, 156, 483, 477, 194, 954,

902, 14, 251, 673, 497, 655, 918, 254, 672, 468, 431, 199, 664, 366, 296,

111, 498, 74, 485, 481, 487, 20, 65, 919, 780, 671, 690, 422, 391, 890, 902,

11, 892, 752, 799, 365, 437, 836, 375, 529, 138, 728, 78, 220, 504, 565,

628, 709, 697, 649, 14, 694, 920, 877, 878, 215, 632, 224, 209, 53, 36, 901,

473, 910, 658, 498, 644, 406, 255, 737, 764, 696, 1000, 270, 559, 3, 707,

772, 688, 887, 737, 864, 346, 990, 340, 892, 779, 469, 255, 559, 966, 858,

871, 784, 486, 633, 290, 815, 523, 435, 911, 253, 785, 882, 970, 161, 514,

3, 552, 940, 73, 313, 858, 436, 497, 708, 82, 161, 955, 171, 539, 664, 911,

184, 783, 50, 185, 996, 648, 292, 534, 245, 265, 155, 279, 436, 911, 900,

931, 10, 413, 729, 177, 742, 969, 11, 880, 947, 37, 318, 91, 583, 615, 44,

772, 450, 438, 886, 309, 264, 93, 338, 374, 225, 593, 937, 923, 124, 378,

124, 701, 522, 281, 431, 165, 226, 396, 519, 592, 144, 967, 575, 844, 821,

898, 557, 99, 143, 723, 136, 687, 52, 809, 37, 189, 80, 842, 532, 696, 703,

857, 613, 38, 818, 60, 456, 3, 715, 134, 34, 364, 937, 838, 486, 511, 298,

95, 696, 397, 289, 286, 678, 31, 65, 732, 888

}

- Weight set = { 380, 474, 833, 176, 779, 268, 503, 941, 735, 486, 28, 977, 121, 901, 571,

923, 144, 367, 526, 461, 72, 315, 405, 57, 812, 702, 613, 650, 402, 346, 986, 479, 421, 852, 959, 784, 874, 577, 900, 561, 13, 698, 720, 242, 356, 274, 292, 985, 498, 83, 771, 565, 656, 995, 675, 572, 439, 790, 225, 4, 18, 609, 988, 427, 366, 950, 344, 34, 753, 40, 160, 299, 912, 111, 386, 828, 227, 633, 911, 486, 170, 986, 115, 423, 308, 278, 624, 757, 302, 278, 486, 64, 805, 966, 659, 570, 105, 870, 434, 709, 435, 1, 31, 206, 238, 143, 733, 481, 591, 299, 744, 848, 744, 72, 906, 832, 215, 974, 755, 437, 331, 531, 883, 732, 767, 663, 499, 758, 982, 22, 915, 133, 166, 9, 27, 899, 14, 835, 977, 908, 420, 28, 299, 652, 717, 329, 915, 112, 458, 380, 657, 362, 489, 846, 143, 217, 180, 377, 246, 445, 335, 968, 394, 617, 95, 900, 838, 195, 952, 920, 118, 108, 867, 731, 93, 336, 289, 414, 6, 607, 545, 660, 710, 478, 622, 649, 481, 209, 562, 987, 133, 947, 271, 674, 320, 117, 43, 487, 112, 86, 209, 571, 368, 851, 982, 2, 534, 206, 372, 713, 357, 273, 51, 770, 871, 189, 677, 73, 219, 468, 780, 356, 557, 143, 351, 612, 271, 338, 190, 143, 928, 741, 605, 404, 936, 733, 225, 913, 49, 824, 818, 3, 679, 75, 248, 38, 17, 460, 859, 872, 127, 237, 729, 763, 160, 587, 333, 958, 39, 632, 735, 656, 570, 220, 599, 332, 528, 180, 895, 892, 520, 50, 65, 718, 726, 928, 856, 583, 671, 970, 922, 258, 149, 18, 411, 618, 86, 942, 252, 53, 346, 326, 887, 104, 498, 45, 265, 276, 201, 252, 827, 819, 984, 995, 55, 89, 594, 362, 307, 337, 661, 274, 374, 851, 929, 486, 853, 13, 254, 945, 493, 768, 194, 461, 960, 825, 446, 756, 582, 682, 410, 536, 236, 45, 946, 772, 556, 583, 331, 9, 687, 699, 667, 255, 819, 390, 824, 660, 919, 396, 805, 946, 887, 537, 756, 29, 651, 53, 910, 324, 292, 203, 256, 373, 266, 592, 982, 5, 613, 726, 567, 390, 181, 333, 156, 292, 923, 163, 155, 994, 56, 490, 213, 169, 929, 240, 842, 240, 856, 196, 732, 136, 477, 663, 219, 719, 524, 304, 45, 220, 989, 93, 764, 635, 981, 600, 565, 763, 842, 492, 961, 113, 648, 358, 507, 165, 440, 5, 397, 884, 381, 57, 693, 249, 3, 809, 546, 106, 988, 531, 41, 621, 147, 272, 729, 996, 349, 599, 504, 539, 304, 671, 657, 290, 190, 931, 437, 158, 652, 443, 656, 900, 92, 226, 486, 122, 165, 847, 161, 12, 260, 835, 605, 125, 506, 35, 128, 299, 580, 764, 445, 498, 612, 76, 742, 642, 956, 395, 271, 885, 772, 281, 319, 180, 489, 739, 308, 950, 797, 507, 538, 349, 520, 497, 735, 770, 427, 87, 128, 88

}

- Output****

**6. Estimate the performance of best-first-search version using Monte Carlo simulation.**

At First, We made this code :



But this is Monte\_Carlo Estimate for DFS, not Best-First-Search.

We think, the selection between the promising children occurs when the item at that level is not selected.

So, we made this code :

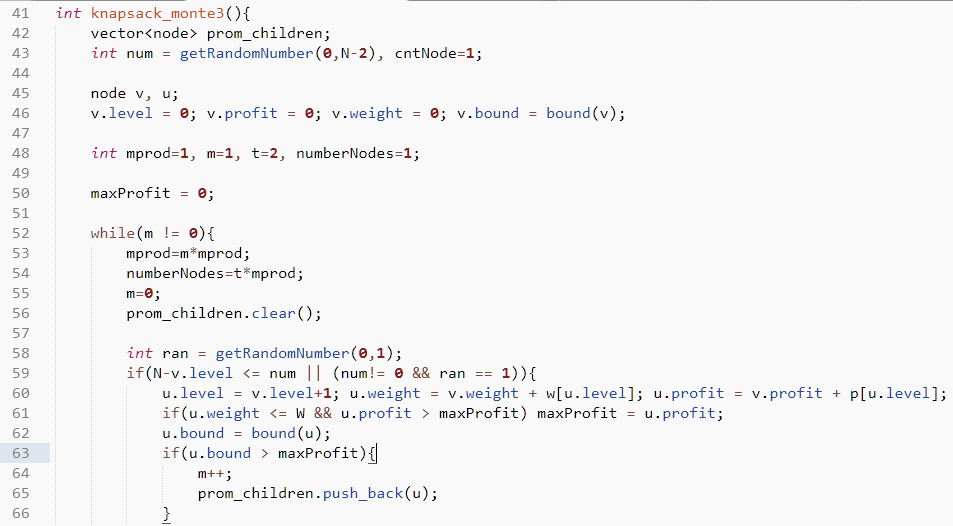
// It is the code that performs the best search by randomly selecting each item and the number of items that it does not have.



But this code did not give us the results we expected.

Next, we supplemented the code :

//It is a code that randomly determines a child at a randomly determined branch point. All items at levels other than the branch point are selected.



However, our random number generator did not work properly and could not produce a proper result.