CS29003 Algorithms Laboratory

Assignment No: 3

Last date of submission: 29–January–2019

Greedy Algorithms

Greedy algorithms are often used to obtain near-optimal solutions to many optimization problems that appear to be very difficult to solve. An exhaustive search requires exponential time. We use several greedy heuristics to solve the problem in polynomial time. These greedy heuristics perhaps do not guarantee optimal solutions, but we are happy with solutions that are close to optimal.

Ms. Kunde from Greedland wants to buy n items from foreign countries. At the beginning of the story, the prices of these items are $p_0, p_1, p_2, \ldots, p_{n-1}$ (positive floating-point numbers). The Customs Department of Greedland does not allow anybody to buy multiple foreign items in a month, so Ms. Kunde plans to buy her n items in n consecutive months (starting from the current month). The economy of Greedland is very poor, and all foreign prices suffer from appreciation rates. For the items of Ms. Kunde's choice, the appreciation rates are $r_0, r_1, r_2, \ldots, r_{n-1}$ (floating-point numbers > 1), that is, in Month j, the cost of Item i is $p_i r_i^j$. Ms. Kunde wants to find the best possible sequence of buying her n items so that her total payment is as small as possible. In particular, if she buys Item i in Month j_i, her total payment equals

$$C = C(j_0, j_1, j_2, \ldots, j_{n-1}) = p_0 r_0^{j_0} + p_1 r_1^{j_1} + p_2 r_2^{j_2} + \cdots + p_{n-1} r_{n-1}^{j_{n-1}}.$$

Here, $(j_0, j_1, j_2, \ldots, j_{n-1})$ is a permutation of $(0, 1, 2, \ldots, n-1)$, and the cost C is to be minimized over all of the $n!$ permutations of $(0, 1, 2, \ldots, n-1)$. Clearly, a naive exhaustive search cannot be used except for only very small values of n. Greedy algorithms help Ms. Kunde in this context. If all her items are from the same foreign country so that $r_0 = r_1 = r_2 = \cdots = r_{n-1}$, then the greedy strategy *costliest first* clearly works. Another special case is that all the initial item prices are equal, that is, $p_0 = p_1 = p_2 = \cdots = p_{n-1}$. In this case, the greedy strategy *largest appreciation rate first* works. In the general cases, neither of these strategies is guaranteed to give optimal solutions.

Part 1: Greedy strategy *costliest first*

Implement a function `grdsearch1()` that buys the costliest item in each month. In Month j, buy the item i if its current cost $p_i r_i^j$ is maximum among all items remaining to be bought. Here, j should be chosen in the order $j = 0, 1, 2, \ldots, n-1$.

Part 2: Greedy strategy *cheapest last*

Choose the item to buy in Month j in the sequence $j = n-1, n-2, \ldots, 2, 1, 0$. For each j, choose that item i for which the cost $p_i r_i^j$ is minimum among all the items that are not yet decided to be bought later (in months $j+1$ through $n-1$). Write a function `grdsearch2()` to implement this greedy strategy.

Part 3: Greedy strategy *maximum price increase first*

A decision on which item to buy in Month j can also be taken as follows, for $j = 0, 1, 2, \ldots, n-2$. Calculate the price hikes for the items remaining to be bought as $p_i(r_i^{j+1} - r_i^j)$. Choose that item for which the price hike is largest. In Month $n-1$, there is only one item remaining to be bought, so buy it. Write a function `grdsearch3()` to implement this part.

Part 4: Exhaustive search with pruning

Now that you have several greedy heuristics in your bag, the obvious question is how good they are. To quantify the goodness of a greedy heuristic, a well-known method is to determine its approximation ratio ρ. One uses mathematical tools to prove that if OPT is the optimal solution of a minimization problem, the greedy algorithm does not supply solutions GRD poorer than $\rho \times \text{OPT}$. In such proofs, we do not need to *know* the value of OPT; it instead suffices to bound the ratio as $\frac{\text{GRD}}{\text{OPT}} \leqslant \rho$. It is not clear to us how such a bound can be established for the current problem.

We follow a different approach here. We compute the optimal solution. The exhaustive search algorithm minimizes Ms. Kunde's payment over all of the $n!$ allowed buying sequences. Such an exhaustive search can supply the output in reasonable time, only for small values of n (like $n \leqslant 12$ for the current problem). There are many ways in which all permutations of $0, 1, 2, \ldots, n-1$ can be generated. The following pseudocode is one such method. It initializes the permutation array $P = (0, 1, 2, \ldots, n-1)$ outside the outermost call. The outermost call is on $k = 0$.

```
genpermutations ( int P[], int n, int k )
{
   int i;

   if (k == n) {
      A permutation is generated, process it, and return.
   }
   for (i=k; i<n; ++i) {
      Swap P[k] with P[i].
      Make a recursive call genpermutations(P,n,k+1).
      Swap P[k] with P[i].
   }
}
```

A *pruning heuristic* can practically cut down the total number of calls of this function. The function keeps on processing each permutation as soon as it is fully generated. Maintain the minimum of the payments over all permutations generated so far. The recursive call in the above code does not alter the elements of P at indices $0, 1, 2, \ldots, k$. If the buying of the first $k+1$ items as suggested by P already incurs a cost larger than (or equal to) the minimum payment discovered so far, there is no point exploring further down the recursion tree. This strategy may prune many branches of the tree, leading to noticeable practical speedup.

Implement a function **`exhsearch()`** to implement this exhaustive search with the pruning heuristic. Only for small values of n, the function would actually make the search. Otherwise, it would simply notify that n is too large to be comfortably handled.

The *main*() function

- The user supplies n, the initial prices $p_0, p_1, p_2, \ldots, p_{n-1}$, and the appreciation rates $r_0, r_1, r_2, \ldots, r_{n-1}$.
- One by one, call the functions implementing the three greedy algorithms described above. Report the solutions and the corresponding payment amounts of Ms. Kunde.
- Call **`exhsearch()`** to compute and print an optimal solution (provided that $n \leqslant 12$).

Sample output

```
n = 10

Initial prices
14      189     154     37      103     73      111     13      72      126

Appreciation rates
2.316   1.536   1.563   1.436   1.979   1.928   1.933   2.339   2.176   1.220

+++ Greedy search 1
    [ 0  1     189.000000]    [ 4  8    1614.239836]    [ 8  3     669.017138]
    [ 1  2     240.702000]    [ 5  5    1944.724153]    [ 9  9     754.412753]
    [ 2  6     414.750279]    [ 6  0    2160.525348]
    [ 3  4     798.315584]    [ 7  7    4979.146504]
--- Total cost = 13764.833595

+++ Greedy search 2
    [ 0  6     111.000000]    [ 4  5    1008.674353]    [ 8  3     669.017138]
    [ 1  4     203.837000]    [ 5  0     932.869321]    [ 9  9     754.412753]
    [ 2  8     340.918272]    [ 6  7    2128.750109]
    [ 3  1     684.913066]    [ 7  2    3509.405618]
--- Total cost = 10343.797630

+++ Greedy search 3
    [ 0  6     111.000000]    [ 4  1    1052.026469]    [ 8  3     669.017138]
    [ 1  4     203.837000]    [ 5  0     932.869321]    [ 9  9     754.412753]
    [ 2  8     340.918272]    [ 6  7    2128.750109]
    [ 3  5     523.171345]    [ 7  2    3509.405618]
--- Total cost = 10225.408025
```

```
+++ Exhaustive search
    [ 0  6     111.000000]    [ 4  0     402.793317]    [ 8  3     669.017138]
    [ 1  4     203.837000]    [ 5  7     910.111205]    [ 9  9     754.412753]
    [ 2  8     340.918272]    [ 6  1    2482.041841]
    [ 3  5     523.171345]    [ 7  2    3509.405618]
--- Total cost = 9906.708489

------------------------------------------------------------------------------------------

n = 32

Initial prices
187     96      21      11      48      51      101     125     32      167
127     91      131     151     150     71      22      135     32      89
111     52      27      99      195     96      67      86      135     122
137     17

Appreciation rates
1.135   1.476   1.644   1.873   1.313   1.941   1.645   2.024   1.967   1.946
2.002   1.294   1.254   1.542   1.697   2.379   1.190   1.176   1.619   2.361
1.364   1.775   2.268   1.522   2.157   1.773   2.181   2.152   1.730   1.418
1.986   2.460

+++ Greedy search 1
    [ 0 24     195.000000]    [11  5   75136.518067]    [22 23 1020369.686972]
    [ 1  9     324.982000]    [12  8  107350.061445]    [23  1  743459.739588]
    [ 2 30     540.354852]    [13 28  167854.546557]    [24 29  532830.861290]
    [ 3 19    1171.326497]    [14 25  291201.448855]    [25 20  260424.738531]
    [ 4 15    2274.239074]    [15 14  418137.283862]    [26 11   74017.187192]
    [ 5  7    4245.829536]    [16 21  504867.089960]    [27  4   74883.966754]
    [ 6 27    8541.852213]    [17  6  477685.119475]    [28 12   74063.173586]
    [ 7 10   16370.133946]    [18  3  885263.743736]    [29 17   14864.356030]
    [ 8 26   34302.054082]    [19 13  565692.014163]    [30  0    8350.595602]
    [ 9 31   56087.345382]    [20 18  489876.891513]    [31 16    4834.799672]
    [10 22   97228.652388]    [21  2  718054.440111]
--- Total cost = 7730500.032929

+++ Greedy search 2
    [ 0 24     195.000000]    [11  5   75136.518067]    [22 23 1020369.686972]
    [ 1  9     324.982000]    [12  8  107350.061445]    [23  1  743459.739588]
    [ 2 19     496.114569]    [13 25  164242.215936]    [24 29  532830.861290]
    [ 3 15     955.964302]    [14 28  290388.365543]    [25 20  260424.738531]
    [ 4  7    2097.741865]    [15 14  418137.283862]    [26 11   74017.187192]
    [ 5 27    3969.262181]    [16 21  504867.089960]    [27  4   74883.966754]
    [ 6 30    8406.128646]    [17  3  472644.817798]    [28 12   74063.173586]
    [ 7 26   15727.672665]    [18  6  785792.021536]    [29 17   14864.356030]
    [ 8 10   32773.008159]    [19 13  565692.014163]    [30  0    8350.595602]
    [ 9 31   56087.345382]    [20 18  489876.891513]    [31 16    4834.799672]
    [10 22   97228.652388]    [21  2  718054.440111]
--- Total cost = 7618542.697307

+++ Greedy search 3
    [ 0 24     195.000000]    [11  5   75136.518067]    [22 23 1020369.686972]
    [ 1  9     324.982000]    [12  8  107350.061445]    [23  1  743459.739588]
    [ 2 19     496.114569]    [13 25  164242.215936]    [24 29  532830.861290]
    [ 3 15     955.964302]    [14 28  290388.365543]    [25 20  260424.738531]
    [ 4  7    2097.741865]    [15 14  418137.283862]    [26 11   74017.187192]
    [ 5 27    3969.262181]    [16 21  504867.089960]    [27  4   74883.966754]
    [ 6 26    7211.220846]    [17  3  472644.817798]    [28 12   74063.173586]
    [ 7 30   16694.571492]    [18  6  785792.021536]    [29 17   14864.356030]
    [ 8 31   22799.733895]    [19 13  565692.014163]    [30  0    8350.595602]
    [ 9 10   65611.562334]    [20 18  489876.891513]    [31 16    4834.799672]
    [10 22   97228.652388]    [21  2  718054.440111]
--- Total cost = 7617865.631022

+++ Exhaustive search
*** n is too large, skipping exhaustive search
```

Submit a single C/C++ source file. Do not use global/static variables.