```
for (i=0; i < n; i++)  used[i]= 0;
for (end= n-1; end ≥ 0; end--)
{
    max_pos= -1;   max= -1;

    for (u=0; u < n ; u++)
    {
        if (!used[u])
        {
            deg=0;
            for (i=0;  < n; i++) deg+= A[u][i];
            if (deg > max) {max_pos= u; max= deg;}
        }
    }
    p[end]= max_pos; used[max_pos]=1;
}
```

A is an adjacency matrix. How much time does this algorithm take to sort the vertices by degree? How could you do it faster?

For the big problems, an $O(n^3)$ sorting algorithm similar to this took 31 seconds on the 2187 vertex football graph. We cannot afford to be inefficient when we only have a short time (in this case one second) to find a dominating set for each graph.

I tested your heuristics using one second per graph.

If the file had k graphs I set a total limit of k+1 seconds for the cpu time.

in_football_1.txt has 6 graphs:

```
limit cputime 7
a.out 1 0 < ../../in_football_1.txt > of1
```

When I marked your creative approaches, I edited some of the programs so they would conform to specs.

I am not going to edit any future submissions.

Please test your programs very carefully before you submit them.

Use the testing scripts I gave you and test on all the input graphs.

A sample pseudo-random number generator:

```
static unsigned int seed = 5323;
unsigned int rand()
{
     seed = (8253729 * seed + 2396403);
     return seed  % 32768;
}
```

If you generate some random numbers starting with a given seed, then you start again with the same seed it will generate the same sequence of random numbers.

http://www.learncpp.com/cpp-tutorial/59-random-number-generation/

The timer clock tics very slowly compared to how fast our programs run.
Here for example are some counts for the first football problem:

```
Number of iterations per tick:      6845
Number of iterations per tick:      5117
Number of iterations per tick:      5874
Number of iterations per tick:      4918
```

If you reseed the random number generator with the time at each iteration, for each block you will have the same random permutation.

Here is the number of times I consider different permutations
with and without reseeding on in_football_1.txt

```
Graph   1: number of iterations 380098 with srand 60
Graph   2: number of iterations  63275 with srand 60
Graph   3: number of iterations  11473 with srand 60
Graph   4: number of iterations   1917 with srand 60
Graph   5: number of iterations    290 with srand 60
Graph   6: number of iterations     30 with srand 30
```

It is even worse if you reseed it with the same constant each trial as it means every permutation will be the same.

Student heuristics performed badly compared to mine when you did this even when I gave you twice as much time as I had.

For future submissions:

DO NOT call srand.

Test your program to see how much better your solutions are if you do not use srand.

There is a second problem with using srand. If you have a bug it is nice if it is reproducible. The clock probably will tick in between the times you run the program. This will change the permutations the program will run on and could mean that you had a bug but cannot reproduce the computation that was buggy.

If you want to choose a seed with a pseudo-random number generator, you should do it only once at the beginning of your computation. But for this class, do not use srand.

Using dynamic memory allocation:

The one program using dynamic memory allocation (new) was “Killed” on the C_{80} fullerenes and the queen graphs because it ran out of memory.

Java does automatic garbage collection. In C/C++ if you ask for memory without freeing it when you are done you will eventually run out of memory.

Using NMAX will avoid problems with memory leaks.

Use

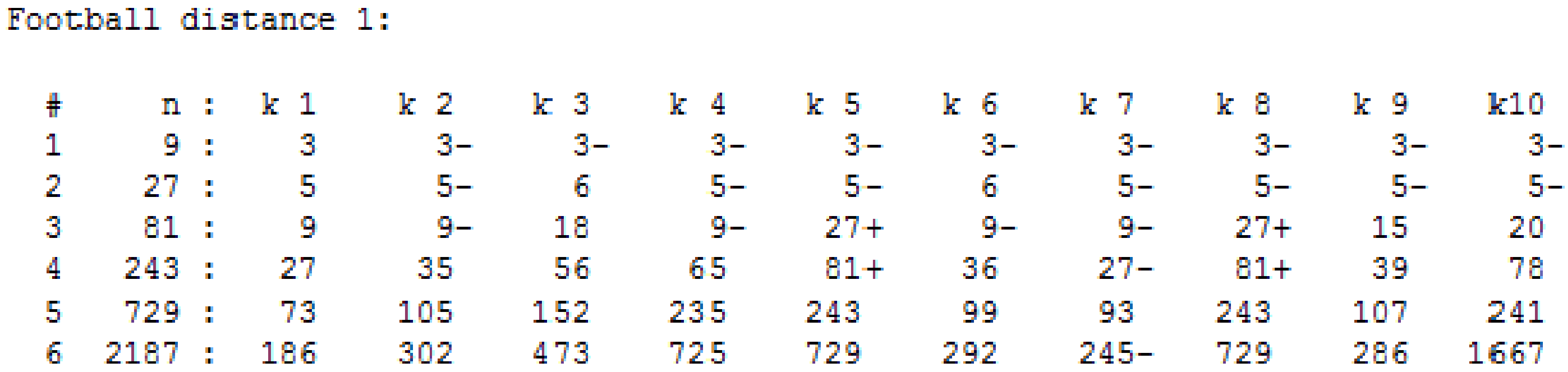
```
fflush(stdout);
```

each time you print some results.

If you do this and you run out of time you will get credit for the problems you managed to solve.

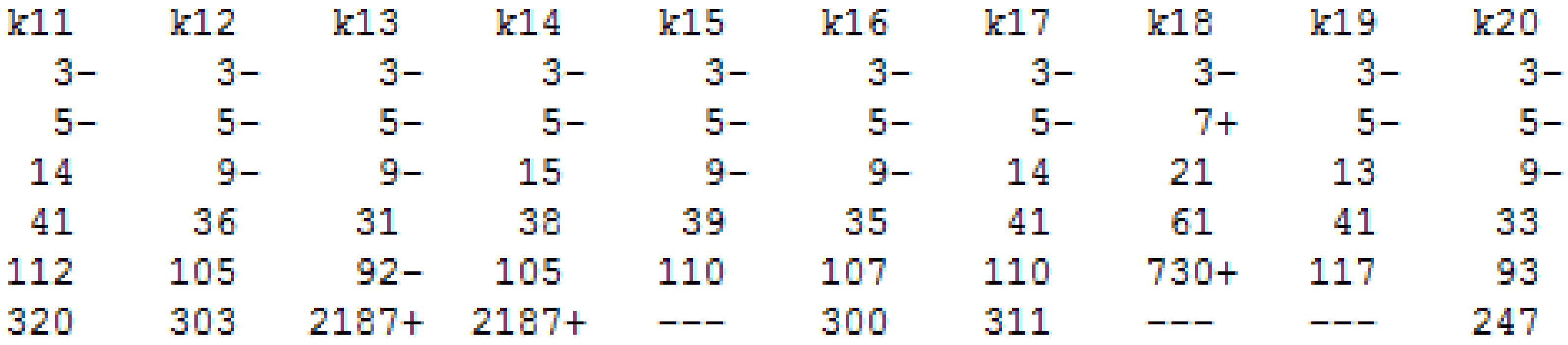
Otherwise, since the output is buffered, some of the problem results may not be printed.

Your student number kxx will be on your feedback sheet. k1= best known solution. k2= my implementation of Alg. 1 (random permutation greedy). You should be able to match or beat that.

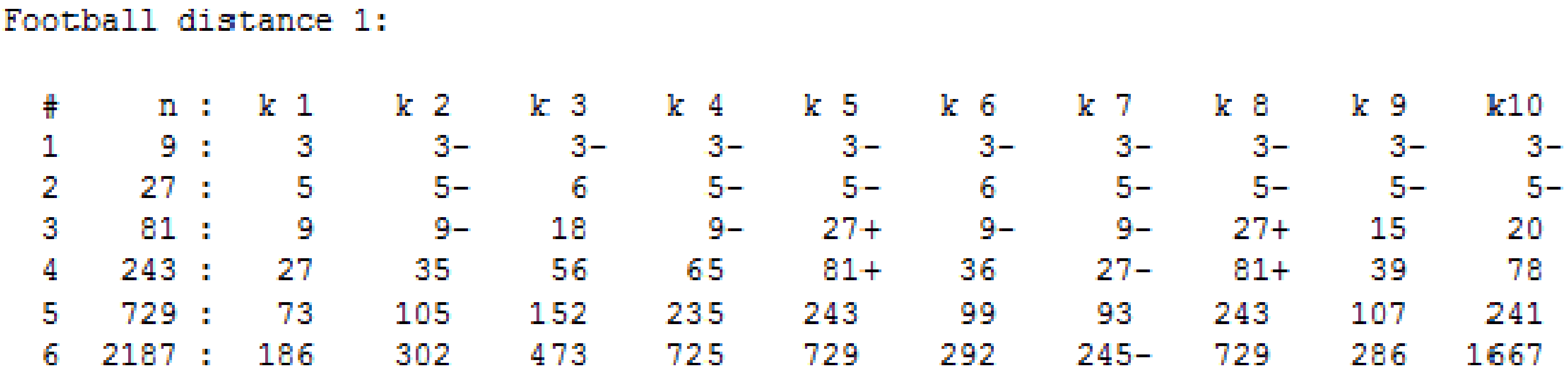


```
Football distance 1:

  #      n :   k 1      k 2     k 3     k 4     k 5     k 6     k 7     k 8     k 9     k10
  1      9 :     3        3-      3-      3-      3-      3-      3-      3-      3-      3-
  2     27 :     5        5-      6       5-      5-      6       5-      5-      5-      5-
  3     81 :     9        9-     18       9-     27+      9-      9-     27+     15      20
  4    243 :    27       35      56      65      81+     36      27-     81+     39      78
  5    729 :    73      105     152     235     243      99      93     243     107     241
  6   2187 :   186      302     473     725     729     292     245-    729     286    1667
```



```
 k11      k12      k13      k14      k15      k16      k17      k18      k19      k20
   3-       3-       3-       3-       3-       3-       3-       3-       3-       3-
   5-       5-       5-       5-       5-       5-       5-       7+       5-       5-
  14        9-       9-      15        9-       9-      14       21       13        9-
  41       36       31       38       39       35       41       61       41       33
 112      105       92-     105      110      107      110      730+     117       93
 320      303     2187+    2187+     ---      300      311      ---      ---      247
```



```
Football distance 1:

  #      n :  k 1    k 2    k 3    k 4    k 5    k 6    k 7    k 8    k 9    k10
  1      9 :    3     3-     3-     3-     3-     3-     3-     3-     3-     3-
  2     27 :    5     5-      6     5-     5-      6     5-     5-     5-     5-
  3     81 :    9     9-     18     9-    27+     9-     9-    27+     15     20
  4    243 :   27     35     56     65    81+     36    27-    81+     39     78
  5    729 :   73    105    152    235    243     99     93    243    107    241
  6   2187 :  186    302    473    725    729    292   245-    729    286   1667
```

Your score for each problem is:
n if your program fails to find a solution to a problem of size n.

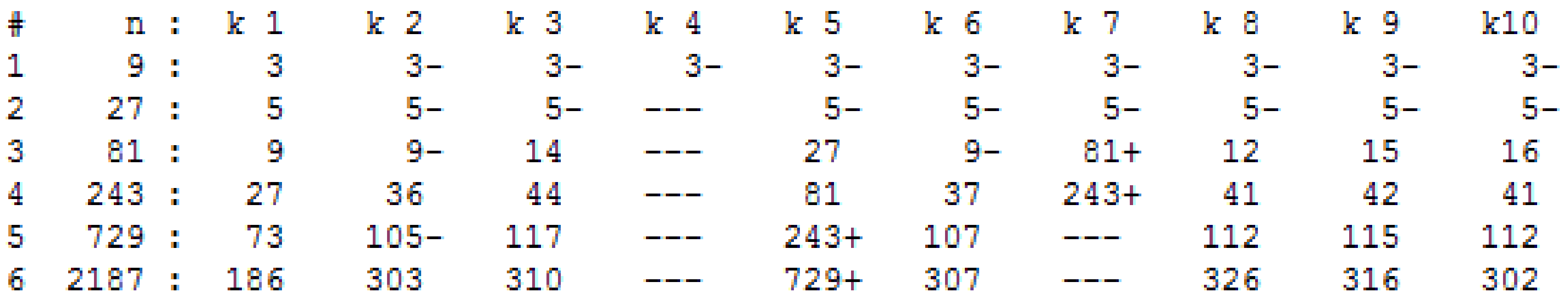
Otherwise it is your solution minus the k 1 solution.

Smaller scores are better.

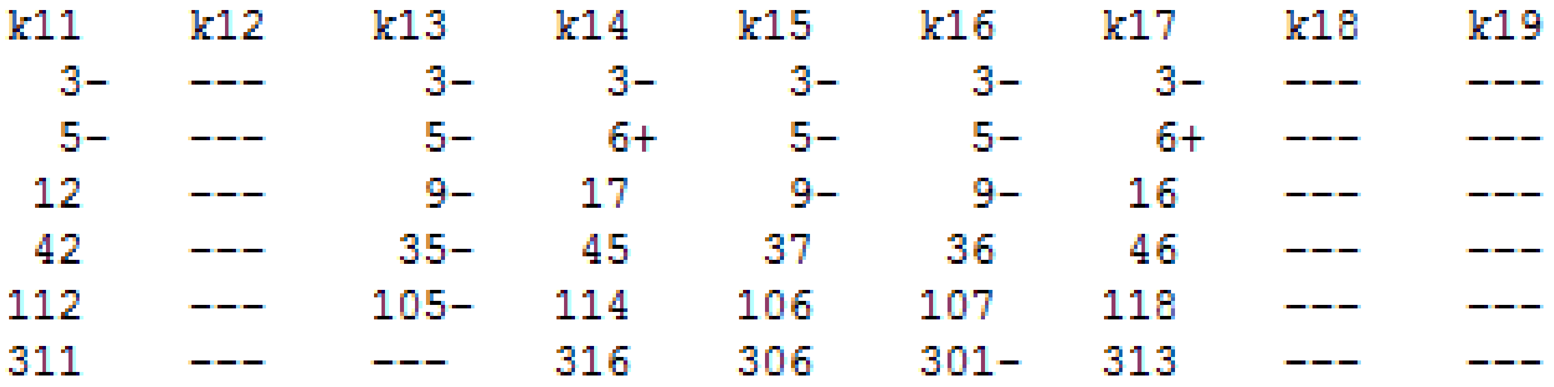
```
Rank  1: Student  7 with score  79
Rank  2: Student 20 with score  87
Rank  3: Student  6 with score 142
Rank  4: Student  9 with score 152
Rank  5: Student  2 with score 156
Rank  5: Student 16 with score 156
==================================
Rank  7: Student 12 with score 158
Rank  8: Student 17 with score 181
Rank  9: Student 11 with score 192
Rank 10: Student  3 with score 405
Rank 11: Student  4 with score 739
Rank 12: Student  5 with score 785
Rank 12: Student  8 with score 785
Rank 14: Student 10 with score 1711
Rank 15: Student 13 with score 2024
Rank 16: Student 14 with score 2050
Rank 17: Student 15 with score 2236
Rank 18: Student 19 with score 2249
Rank 19: Student 18 with score 2892
```

Student 2 is my implementation of Algorithm 1 (the random permutation greedy algorithm).

Algorithm 1: no editing done to fix bugs.
Input file football_1.txt

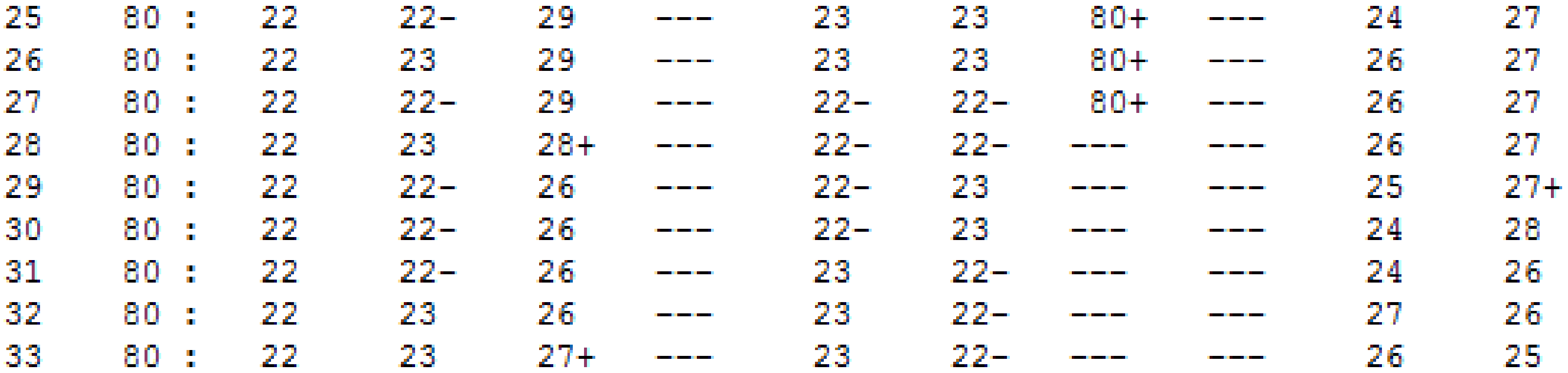


```
#      n :  k 1    k 2    k 3    k 4    k 5    k 6    k 7    k 8    k 9    k10
1      9 :    3     3-     3-     3-     3-     3-     3-     3-     3-     3-
2     27 :    5     5-     5-    ---     5-     5-     5-     5-     5-     5-
3     81 :    9     9-    14     ---    27      9-    81+    12     15     16
4    243 :   27    36     44     ---    81     37    243+    41     42     41
5    729 :   73   105-   117     ---   243+   107    ---    112    115    112
6   2187 :  186   303    310     ---   729+   307    ---    326    316    302
```

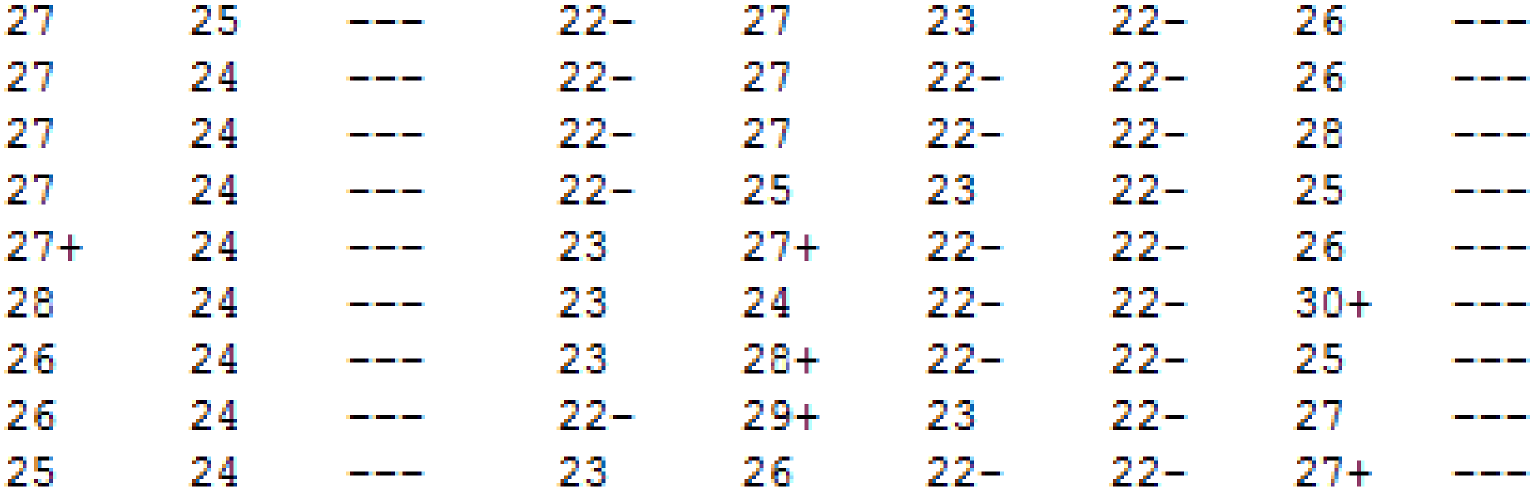


```
k11     k12     k13     k14     k15     k16     k17     k18     k19
  3-    ---       3-      3-      3-      3-      3-    ---     ---
  5-    ---       5-      6+      5-      5-      6+    ---     ---
 12     ---       9-    17        9-      9-    16      ---     ---
 42     ---     35-     45      37      36      46      ---     ---
112     ---    105-    114     106     107     118      ---     ---
311     ---     ---    316     306     301-    313      ---     ---
```

Tail end of C8- results for Algorithm 1:



```
25    80 :   22    22-   29    ---    23    23    80+   ---    24    27
26    80 :   22    23    29    ---    23    23    80+   ---    26    27
27    80 :   22    22-   29    ---    22-   22-   80+   ---    26    27
28    80 :   22    23    28+   ---    22-   22-   ---   ---    26    27
29    80 :   22    22-   26    ---    22-   23    ---   ---    25    27+
30    80 :   22    22-   26    ---    22-   23    ---   ---    24    28
31    80 :   22    22-   26    ---    23    22-   ---   ---    24    26
32    80 :   22    23    26    ---    23    22-   ---   ---    27    26
33    80 :   22    23    27+   ---    23    22-   ---   ---    26    25
```



```
27     25     ---     22-    27     23     22-    26     ---
27     24     ---     22-    27     22-    22-    26     ---
27     24     ---     22-    27     22-    22-    28     ---
27     24     ---     22-    25     23     22-    25     ---
27+    24     ---     23     27+    22-    22-    26     ---
28     24     ---     23     24     22-    22-    30+    ---
26     24     ---     23     28+    22-    22-    25     ---
26     24     ---     22-    29+    23     22-    27     ---
25     24     ---     23     26     22-    22-    27+    ---
```

```
Rank  1: Student 16 with score  34
Rank  2: Student 13 with score  40
Rank  3: Student  2 with score  42
==============================================
Rank  4: Student  6 with score  46
Rank  4: Student 15 with score  46
Rank  6: Student  5 with score  55
Rank  7: Student 11 with score 111
Rank  8: Student  9 with score 124
Rank  9: Student 10 with score 173
Rank 10: Student  3 with score 174
Rank 11: Student 17 with score 177
Rank 12: Student 14 with score 182
Rank 13: Student  7 with score 2079
Rank 14: Student  4 with score 2565
Rank 15: Student  8 with score 2640
Rank 15: Student 12 with score 2640
Rank 15: Student 18 with score 2640
```