Some information about task PALIN

A good method to solve this problem is to determine the length L
of a longest common subsequence (maximal matching) for the input and
its reverse. The answer then is N - L. An alternative approach is
to match a prefix of the string with the reverse of a postfix.

The length of a longest common subsequence can be determined
by dynamic programming. A triangular table can be constructed,
of which only two rows need to be stored. The complexity
is then O(N) space and O(N^2) time.

Note that constructing a witness (indicating where which characters
have to be inserted to make a palindrome) is computationally more
involved and is not asked.

For special inputs, other simpler methods may apply.

The 10 test cases have the following characteristics:

Case #	N	D	A	Kind of data
1	62	62	61	Each allowed character exactly once
2	4960	62	4801	80 repetitions of case #1
3	5000	1	0	'9'^5000
4	5000	2	2500	'A'^2500 ++ 'z'^2500
5	5000	2	1	'PC'^2500
6	100	48	79	Random
7	3	2	1	'FFT'
8	5000	2	919	Random with only characters 'O' and 'K'
9	4999	10	2628	Random with only digits
10	4999	20	88	'W'^4999 randomly perturbed in few places

where
  N = length of the string (input)
  D = number of distinct characters in input string
  A = correct answer

Some information about task CAR

The final state of the parking center can easily be determined by sorting the input list of car types. This can be done in linear time (O(M+N)). O(N^2) and N*log N methods may be too slow for larger N.

A greedy approach to construct successive rounds will work, since in each round it can be guaranteed that at least W-1 cars are put into their final position. Hence, the number of rounds needed by this greedy algorithm is D/(W-1) rounded up, where D is the number of displaced cars in the initial state for the input. In general, finding the minimum number of rounds is NP-complete. Even reducing the number of rounds for the greedy algorithm by just one round is, in general, as hard as finding the minimum (compare to bin packing: you need to find cycles that add up in length to W, for otherwise one worker is not doing useful work).

The greedy algorithm can be implemented in O(N+M) space and O(N) time. However, a more naive implementation may use O(N) space and O(N^2) time. The test data has been designed to distinghuish linear solutions from less efficient ones.

The 10 test cases have the following characteristics:

#	N	M	W	Q	D	Min	Max	Kind of data
1	5	5	2	5	0	1	1	Sorted
2	12	10	5	3	10	1	2	Manually designed
3	30	30	6	6	30	1	1	Manually designed
4	300	50	12	28	293	1	14	Random
5	500	50	27	20	485	3	16	Random
6	20000	50	5	5000	20000	400	400	4000 2-cycles, 4000 3-cycles
7	20000	50	49	417	20000	400	400	400 50-cycles
8	20000	50	2	20000	19595	368	439	Random
9	20000	50	10	2223	443	393	406	Sorted, randomly perturbed
10	20000	50	50	409	19087	345	449	Random

where
- N = number of cars (input)
- M = number of car types (input)
- W = number of workers (input)
- Q = N / (W-1) rounded up
- D = number of displaced cars
- Min = minimum number of cars in a type, over all types
- Max = maximum number of cars in a type, over all types

Task MEDIAN

The 10 test cases for MEDIAN have been designed to detect performance differences as exhibited by 16 different algorithms (also see below):

- OPE = Onion Peeling Eliminiation
- LISF = Linear Insertion Sort Using Full List
- LISH = Linear Insertion Sort Using Half List
- LISZ = Linear Insertion Sort Using Zoom List
- BISF = Binary Insertion Sort Using Full List
- BISH = Binary Insertion Sort Using Half List
- BISZ = Binary Insertion Sort Using Zoom List
- TISF = Ternary Insertion Sort Using Full List
- TISH = Ternary Insertion Sort Using Half List
- TISZ = Ternary Insertion Sort Using Zoom List
- TPFS = Ternary Partioning Find Using Straddled Pivots
- TPFF = Ternary Partioning Find Using First Pivots
- TPFP = Ternary Partioning Find Using Proportional Pivots
- TPFR = Ternary Partioning Find Using Random Pivots
- SLSB = Sorted List of Sorted Buckets
- HTSB = Heap-like Tree of Sorted Buckets

The next table shows how many calls each algorithm made for each test case solved within the bound of 7777 calls. The rightmost column shows the score.

Case #	1	2	3	4	5	6	7	8	9	10	
N	5	177	577	975	1087	1267	1357	1415	1415	1499	
Cat	M	R	N	R	R	R	R	R	A	R	Pts
Alg											
OPE	4	7744									20
LISF	4	4062	619								30
LISH	3	2590	598								30
LISZ	3	2160	598								30
BISF	4	861	4175	7051							40
BISH	5	843	4108	6803							40
BISZ	4	730	3621	6269	7078						50
TISF	3	712	2918	5415	6143	7376					60
TISH	3	669	2707	5349	6011	7103	7642				70
TISZ	3	609	2537	4889	5540	6641	7191	7511	7572		90
TPFS	3	517		1525	2842	3257	3531	2231		3218	80
TPFF	4	395		2205	2378	3635	3601	2663		2493	80
TPFP	5	331	848	3512	1705	2291	3093	2860	2863	2985	100

TPFR	4	372	1778	2201	2507	2981	3377	3987	3279	3540	100
SLSB	4	491	1954	3242	3605	4258	4578	4824	4149	5147	100
HTSB	4	508	2218	3184	3902	4517	4862	5074	4389	5354	100

The 10 test cases belong to 4 categories:
  M = Manually designed
  R = Randomly generated
  N = Nearly sorted
  A = Alternating outside-to-inside (1 3 5 ... 6 4 2)

Here is a similar table showing the number of calls for cases where the algorithm FAILS (does not stay within the bound). When the number of calls exceeds 9999, only an approximate value in "scientific notation" is given, where >XeY means that the number of calls exceeds X*10^Y, but does not exceed (X+1)*10^Y. One extra column has been added on the right. It indicates whether for N = 1499 and under worst-case conditions (W), the algorithm stays within the bound of 7777 (shown as <=) or not (shown as >). The library, however, is not able to create such worst-case conditions dynamically.

Case #	1	2	3	4	5	6	7	8	9	10	
N	5	177	577	975	1087	1267	1357	1415	1415	1499	1499
Cat	M	R	N	R	R	R	R	R	A	R	W
Alg											
OPE			>8e4	>2e5	>2e5	>4e5	>4e5	>4e5	>4e5	>5e5	>
LISF				>1e5	>1e5	>1e5	>2e5	>2e5	>2e5	>2e5	>
LISH				>7e4	>8e4	>1e5	>1e5	>1e5	>1e5	>1e5	>
LISZ				>5e4	>7e4	>1e5	>1e5	>1e5	>6e4	>1e5	>
BISF					7791	9532	>1e4	>1e4	>1e4	>1e4	>
BISH					7811	9414	>1e4	>1e4	>1e4	>1e4	>
BISZ						8537	9299	9803	>1e4	>1e4	>
TISF							7981	8386	8339	8993	>
TISH								7980	7946	8519	>
TISZ										8032	>
TPFS			>7e4						>1e4		>
TPFF			>5e4						>1e4		>
TPFP											>
TPFR											>
SLSB											<=

```
----   + ----  ----  ----  ----  ----  ----  ----  ----  ----  ---- + ---
HTSB   |                                                           |  <=
----   + ----  ----  ----  ----  ----  ----  ----  ----  ----  ---- + ---
```

Information about the algorithms

OPE: Repeatedly eliminate the two extremes (min and max strength).
This takes (N-1)^2 / 4 calls.

All Insertion Sort methods: Maintain a sorted list of objects investigated so far (sorted modulo up or down) and repeatedly insert a next object. The location to insert can be found by linear, binary, or ternary search. Linear insertion is quadratic in both worst and average cases, and linear in best cases. Binary and ternary insertion have N*log N complexity (ternary has smaller constant factor).

Instead of maintaining the Full list (LISF, BISF, TISF) containing all the objects in the end, it is enough to limit the list to contain no more than half the number of objects (Half List: LISH, BISH, TISH). Reason: after having considered (N+1)/2 objects, the element at the end of the list cannot be the median, because more than (N-1)/2 objects are stronger/weaker than this object.

In fact, both extremes in the sorted list can be eliminated once (N+1)/2 objects have investigated (Zoom List: LISZ, BISZ, TISZ). That way, the list increases in length during the first half, and descreases in length during the second half, until only one candidate remains (which then must be the median); it zooms out and then in on the median.

All Partitioning Find methods: Compare to median selection by partitioning (as in QuickSort, discarding the segment that is known not to contain the median). Only partitioning into three parts (based on choosing two pivot objects) have been considered. In general these methods are quadratic in worst case, but linear in average and best case. There are various ways to choose the pivots: one at each end (Straddled: TPFS), both at one end (First: TPFF), at one third and two thirds in the list (Proportional: TPFP), and Random (TPFR). For TPFS and TPFF, the sorted input is bad, but for TPFP and TPFR it is (very) good. TPFR has no specific worst case inputs. Worst case input for TPFP depends on details of rounding when choosing the proportional pivots.

SLSB: Maintains buckets of at most K objects (for some K; K=8 is a good choice). Each bucket is sorted (with respect to the order of two reference objects), and the list of buckets is sorted on the minimum of the buckets (w.r.t. the same reference objects). Compared to insertion sort into a list of single objects (Full, Half, or Zoom) this saves calls (over 2000 in the worst case of the task), because only a partial order instead of a total order is constructed. You can calculate the number of calls in the worst case for N=1499, and it is just below 7777. Average case behavior is better than worst case.

HTSB: This takes the idea of SLSB one step further by maintaining

the buckets in a heap-like leaftree. The data structure is more complicated, and this method is not needed to obtain a perfect score. It shows that more advanced data structures can do even better, not only on average but also in the worst case. Note that the advantage is not visible for "small" N (such as 1499) on random cases.

Some information about task WALLS

A straightforward solution can be based on the dual graph
of the planar graph representing the map of towns and
connecting walls. The dual graph is obtained by viewing
the areas as nodes that are connected when they share a wall
(this can be a multigraph).

Traversing an edge in the dual graph corresponds to crossing
a wall, hence minimizing wall crossings corresponds to
selecting shortest paths in the dual graph.

A brute force approach tries each area (factor M) as a meeting
area and determines the best routes for each member (factor L)
by trying each starting area (factor <=N). Applying
Warshall's all-pairs shortest path algorithm on the dual graph,
provides all the information needed. This yields an
O(N^3+M*L*N) algorithm to solve the problem. The time limit is
chosen such that this solution is acceptable, even though the
complexity can be reduced by selecting a different algorithm
for determining all relevant distances.

The 10 test cases WALLS#.IN have the following characteristics:

#	M	N	L	W	WAn	WAx	ATn	ATx	Answer	Kind of data
1	6	10	1	14	3	9	2	5	0	1* Manual, one member
2	5	5	5	8	3	4	2	4	1	4 Manual, member in every town
3	10	10	3	18	3	7	2	6	2	3 The example
4	50	98	20	146	3	10	2	8	35	34 Random, medium size
5	100	218	2	316	3	14	2	9	2	20* Random, many areas, few members
6	200	231	30	429	3	7	2	14	94	67 Random, large size
7	180	208	20	386	3	32	2	17	110	84 Random, large size
8	100	225	10	323	3	10	2	7	33	23 Random, medium size
9	200	247	30	445	3	14	2	16	51	99* Random, members close together
10	157	182	30	337	4	50	2	4	39	156 12x13 grid

where
- M = number of areas (input)
- N = number of towns (input)
- L = number of members (input)
- W = number of walls
- WAn = minimum number of walls around an area
- WAx = maximum number of walls around an area
- ATn = minimum number of areas (walls) touching a town
- ATx = maximum number of areas (walls) touching a town
- Answer = minimum crossing-sum, optimal meeting area (* if not unique)

Some information about task POST

The exact solution can be obtained by dynamic programming
based on a 2-dimensional table. The entire table must be
stored (O(P*V) space), and each entry can be computed in
O(V) time worst case. This yields an algorithm of O(P*V^2)
time complexity.

Many approximate solutions based on various heuristics exist
(spread post offices evenly or based on gap size between villages,
local search, simulated annealing, genetic programming, ...).
Because of the rules for partial credit these programs can
also score some points in some cases.

The 10 test cases POST#.IN have the following characteristics:

#	V	P	X1	XV	Gn	Gx	Answer	Kind of data
1	5	1	1	5	1	1	6	Manual, one post office
2	10	2	1	10	1	1	12	Manual, small
3	10	5	1	50	1	22	9	The example
4	284	30	56	9985	1	209	18394	Random, large
5	290	55	7	9897	1	162	24780	Random, large
6	300	30	44	9249	1	202	18153	Random, largest
7	300	1	1	9996	1	3700	1428420	Random, longint answer
8	100	9	92	996	1	37	2134	Random, medium size
9	259	15	2	9899	1	1701	3595	Random, villages in 10 clusters
10	19	3	1	6765	1	2584	5026	Villages at Fibonacci coordinates

where
  V = number of villages (input)
  P = number of post offices (input)
  X1 = smallest village X coordinate
  XV = greatest village X coordinate
  Gn = minimum gap between neighboring villages
  Gx = maximum gap between neighboring villages
  Answer = sum of distances for optimal post office locations

Some information about task BLOCK

A lower bound on the number of blocks in a decomposition is V/4
rounded up. Finding a minimal decomposition by simple backtracking
is slow because there are so many solutions with small blocks.
The technique of branch-and-bound can be used to reduce the running
time drastically. When a partial decomposition with T blocks
has been obtained, a lower bound for the number of blocks in
the complete decomposition is T + W/4 rounded up, where W is the
volume remaining to be decomposed. Thus, the recursion can be
stopped when T + W/4 exceeds the minimum obtained so far.

Each block type consists of at most 24 rotated blocks (the actual
number depends on the symmetries of the block). These rotations
can be precomputed (based on the rotation group of the cube,
which can be generated by two permutations). The translations
can be done on-the-fly during backtracking.

It can be expected that programs for this task are somewhat
longer than for the other tasks.

#	V	M	1	2	3	4	5	6	Kind of data
1	4	1	.	4	.	.	.	.	Single 2x2 block
2	22	6	4	2	10	6	.	.	Manual, a "chair"
3	19	6	6	7	4	.	.	2	Manual, a "tree"
4	18	5	6	3	6	3	.	.	The example (a "horse")
5	30	8	2	28	.	.	.	.	Manual, medium size
6	30	8	.	20	8	2	.	.	Manual, medium size
7	50	13	10	25	6	7	2	.	Manual, largest size
8	33	12	11	17	1	4	.	.	Manual, large size
9	37	13	13	13	11	.	.	.	Manual, large size
10	28	11	12	5	10	1	.	.	Manual, medium size

where
  V = volume (input)
  M = minimum number of blocks in decomposition
  1 = number of unit cubes with 1 neighbors (. means 0)
  2,3,4,5,6 = same for indicated number of neighbors