Week 4 programming assignment

Task: It is necessary to develop and implement in C++ a local search heuristics for solving the maximum clique problem.

Advices:

Please use Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Variable Neighborhood Search (VNS), Tabu Search (TS) and Large Neighborhood Search (LNS) approaches to get the best results.

For example in our implementation we use a multi-start approach from GRASP each time starting with a fast randomized heuristic which returns a completely random solution (at each step we choose a random candidate uniformly). Though such a solution has a low quality it allows local search trying different regions of a solution space. We make about 5000 iterations.

In the local search we use VNS approach and make move, swap1-1 and swap1-2 steps. To prevent cycling during swap1-1 steps we apply TS approach: use short tabu lists for added and removed vertices and also choose vertices to swap randomly.

To provide more diversification swap1-2 vertices are also chosen in a random way. Moves are performed each time we get a non-empty set of candidates after swap1-1 or swap1-2 step.

We restart the local search when we are in the local optimum for all neighborhoods (move, swap1-1, swap1-2) or when we have already performed a certain number of steps without improving the current solution.

Input: text file in DIMACS format containing an undirected graph

Output: the size k of the found clique in this graph in the 1-st line and then k vertices of this clique in the 2-nd line.

Example:

Input file:

```
c c – comments, p – problem, edge – “edge” format, e - edge
c first line: p edge <number of vertices> <number of edges>
c next lines – graph edges: e <vertex 1> <vertex 2>
c WARNING: edges can be duplicated in these lines: (1,2), (2,1)
p edge 5 7
e 1 2
e 2 1
e 2 3
e 3 4
e 4 5
e 5 1
e 5 2
```

Output:

```
3
1  2  5
```

Here vertices 1, 2, 5 form a clique of size 3.

Input files to test the program should be taken from DIMACS challenge graphs:

http://iridia.ulb.ac.be/~fmascia/maximum_clique/DIMACS-benchmark or

https://hse.ru/data/2021/09/16/1471878190/max_clique_txt.7z

The program should be tested on the following files. The size of the maximum clique is specified in brackets. Your heuristics will not always find the exact maximum:

Easy: C125.9.clq (34), johnson8-2-4 (4), johnson16-2-4 (8), MANN_a9 (16), keller4 (11), haming8-4 (16)

Moderate: brock200_1 (21), brock200_2 (12), brock200_3 (15), brock200_4 (17), gen200_p0.9_44 (44), gen200_p0.9_55 (55), MANN_a27 (126), p_hat1000-1 (10), p_hat1000-2 (46), p_hat300-3 (36), p_hat500-3 (50), sanr200_0.9 (42), sanr400_0.7 (21)

Hard: brock400_1 (27), brock400_2 (29), brock400_3 (31), brock400_4 (33), san1000 (15), p_hat1500-1 (12)

Computational results

Please compile a release version of your program with maximal optimizations turned on to obtain good computational results. Together with the program code it is necessary to provide the following table with your computational results including the computational time and the found solution:

Instance	Time, sec	Clique size	Clique vertices
C125.9.clq	10.232	30	4, 6, 8, 13, …
…	0.001	8	3, …

Operating systems: it is possible to provide a C++ program code for Windows, Linux or MacOs. Other operating systems and programming languages are not allowed.

Evaluation:

First evaluation:

1. Maximum number of points is 10.

2. Incorrectly working programs get no more than 3 points. Incorrect programs are those which do not pass Check() test or can fail with "core dumped" or other errors shown by an operating system. Programs with hard-coded solutions or some other "tricks" are also considered incorrect.

Two things are the most important:

1. A quality of solutions (first priority): the larger is the found clique – the better
2. An efficiency of your program (second priority): the faster is your program – the better

Here are the computational times (for release version – with O3 optimizations) and clique sizes, which get the maximal mark of 10 points:

Instance	Time, sec	Clique size
brock200_1.clq	≤ 10	= 21
brock200_2.clq	≤ 10	= 12
brock200_3.clq	≤ 10	= 15
brock200_4.clq	≤ 10	= 17
brock400_1.clq	≤ 40	≥ 25
brock400_2.clq	≤ 40	≥ 25
brock400_3.clq	≤ 40	≥ 30
brock400_4.clq	≤ 40	≥ 25
C125.9.clq	≤ 10	= 34
gen200_p0.9_44.clq	≤ 20	= 44
gen200_p0.9_55.clq	≤ 20	= 55
hamming8-4.clq	≤ 15	= 16
johnson16-2-4.clq	≤ 3	= 8
johnson8-2-4.clq	≤ 1	= 4
keller4.clq	≤ 10	= 11
MANN_a27.clq	≤ 500	= 126
MANN_a9.clq	≤ 10	= 16
p_hat1000-1.clq	≤ 500	= 10
p_hat1000-2.clq	≤ 500	= 46
p_hat1500-1.clq	≤ 1500	≥ 11

p_hat300-3.clq	≤ 35	= 36
p_hat500-3.clq	≤ 100	= 50
san1000.clq	≤ 1500	≥ 10
sanr200_0.9.clq	≤ 20	= 42
sanr400_0.7.clq	≤ 40	= 21

Here are approximate results, which get 6 points:

Instance	Time, sec	Clique size
brock200_1.clq	~ 20	~ 21
brock200_2.clq	~ 20	~ 12
brock200_3.clq	~ 20	~ 15
brock200_4.clq	~ 20	~ 17
brock400_1.clq	~ 100	~ 24
brock400_2.clq	~ 100	~ 24
brock400_3.clq	~ 100	~ 25
brock400_4.clq	~ 100	~ 25
C125.9.clq	~ 20	~ 34
gen200_p0.9_44.clq	~ 50	~ 40
gen200_p0.9_55.clq	~ 50	~ 50
hamming8-4.clq	~ 30	~ 16
johnson16-2-4.clq	~ 10	~ 8
johnson8-2-4.clq	~ 5	~ 4
keller4.clq	~ 20	~ 11
MANN_a27.clq	~ 1000	~ 124
MANN_a9.clq	~ 20	~ 16
p_hat1000-1.clq	~ 1000	~ 10
p_hat1000-2.clq	~ 1000	~ 43
p_hat1500-1.clq	~ 3000	~ 11
p_hat300-3.clq	~ 100	~ 36
p_hat500-3.clq	~ 200	~ 46
san1000.clq	~ 3000	~ 9

sanr200_0.9.clq	~ 50	~ 39
sanr400_0.7.clq	~ 100	~20

Correct but slower programs or programs returning smaller cliques get 4-5 points. Better and faster programs get 7-10 points.