CS 6385.0W2 - Algorithmic Aspects of Telecommunication Networks - Su19

Assignment 5

Submitted By: ***Yash Pradhan (ypp170130)***

Problem Statement:

Create an implementation of the Nagamochi-Ibaraki algorithm (see in the lecture notes) for finding a minimum cut in an undirected graph, and experiment with it.

Explain how your implementation of the algorithm works. Provide pseudo code for the description, with sufficient comments to make it readable and understandable by a human.

Write a computer program that implements the algorithm. You may use C/C++ or java for programming.

Run the program on randomly generated examples. Let the number of nodes be fixed at n = 25, while the number m of edges will vary between 50 and 550, increasing in steps of 5. Once a value of m is selected, the program creates a graph with n = 25 nodes and m edges.

The actual edges are selected randomly among all possible ones, with parallel edges allowed, but self-loops are excluded.

Solution:

Nagamochi-Ibaraki Algorithm for finding a min-cut in an undirected graph.

The source code resides at src/ypp170130, instructions to compile and run the code are given the src/readme file, the code has detailed commenting to explain its working.

This report underlines the algorithm, the experiments when I ran the code as required, with n = 25 and edges between 50 and 550, with steps of 5, allowing parallel edges but not self-loops, and my understanding and screenshot of the execution.

Implementation Details:

I have used the adjacency matrix representation for representing the undirected graph, merging parallel edges on the go. I have defined utility functions mainly for computing the min-cut of the graph, for getting an arbitrary vertex, merging two vertices, computing the s-t-phase cut as well as for random generation of the graph as stated. The programming language used is Java.

Nagamochi Ibaraki Algorithm.

- MIN-CUT (G, w):

```
    w(minCut) ← ∞
    while |G.V| > 1 :
        s-t-phasecut ← MIN-CUT-PHASE (G, w)
        // is this better?
        if w(s-t-phasecut) < w(min Cut):
            min Cut ← s-t-phasecut
        merge (G, s, t)
    return minCut
```

Run Time:

$O(mn + n^2 \log(n))$

- MIN-CUT-PHASE(G, w):

```
    a ← G.getArbitaryVertex()
    A ← {a}
    while A ≠ V
        v ← A.getMostTightVertex()
        A ← A ∪ {v}
    // s and t are vertex last added
    return cut(A - t, t)
```

- Merge (G, s, t)

```
    G ← G\{s,t} ∪ {st}        // contract st
    ∀ v ∈ V, v ≠ {st}
        w(st, v) = w(s, v) + w(t, v)
    return
```

Screenshot of Execution:

Execution for the example given in Slides

```
Run:  Nagamochilbaraki
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
#Vertices: 6; #Edges: 7
------------------------------------
{0, 6,  0,  0,  0,  0,  },
{6, 0,  8,  3,  0,  0,  },
{0, 8,  0,  0,  1,  0,  },
{0, 3,  0,  0,  20, 5,  },
{0, 0,  1,  20, 0,  2,  },
{0, 0,  0,  5,  2,  0,  },
------------------------------------
Min Cut: 4
0: IN_CUT, 1: IN_CUT, 2: SECOND_LAST, 3: LAST, 4: DELETED, 5: DELETED,

Process finished with exit code 0
```

Execution output for a graph with n = 25 and m = 50

```
Run:  Nagamochilbaraki
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
#Vertices: 25; #Edges: 44
------------------------------------
{0, 0,  0,  0,  9,  0,  0,  0,  0,  43, 0,  0,  11, 0,  0,  0,  0,  0,  0,  0,  0,  0,  30, 0,  0,  },
{0, 0,  0,  0,  0,  16, 0,  0,  32, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  25, 0,  0,  0,  },
{0, 0,  0,  0,  0,  0,  21, 0,  0,  0,  0,  0,  0,  0,  26, 0,  0,  0,  0,  0,  7,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  29, 0,  0,  0,  0,  0,  0,  17, 0,  0,  0,  0,  16, 0,  0,  },
{9, 0,  0,  0,  0,  0,  0,  0,  5,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  },
{0, 16, 0,  0,  0,  0,  0,  0,  60, 0,  0,  0,  0,  0,  35, 1,  0,  32, 0,  0,  0,  0,  13, 0,  9,  },
{0, 0,  21, 0,  0,  0,  0,  0,  0,  0,  0,  0,  11, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  32, },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  30, 0,  0,  0,  0,  12, 0,  0,  },
{0, 32, 0,  0,  5,  60, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  14, 0,  0,  0,  0,  0,  },
{43,    0,  0,  0,  0,  0,  0,  0,  0,  0,  2,  10, 0,  0,  0,  0,  0,  0,  19, 0,  0,  0,  0,  28, 20, },
{0, 0,  0,  29, 0,  0,  0,  0,  0,  2,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  10, 0,  0,  0,  0,  20, 7,  0,  0,  0,  48, 0,  0,  25, 0,  26, },
{11,    0,  0,  0,  0,  0,  11, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  17, 0,  0,  0,  0,  0,  0,  },
{0, 0,  26, 0,  0,  35, 0,  0,  0,  0,  0,  20, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  1,  0,  0,  0,  0,  0,  7,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  3,  0,  26, },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  25, 0,  0,  0,  0,  0,  17, 0,  },
{0, 0,  0,  17, 0,  32, 0,  30, 0,  0,  0,  0,  0,  0,  0,  0,  25, 0,  0,  20, 0,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  19, 1,  0,  0,  17, 0,  0,  0,  0,  0,  0,  36, 0,  0,  24, 0,  },
{0, 0,  0,  0,  0,  0,  0,  0,  14, 0,  0,  48, 0,  0,  0,  0,  0,  20, 0,  0,  0,  0,  0,  0,  0,  },
{0, 0,  7,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  36, 0,  0,  0,  0,  0,  0,  },
{0, 25, 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  },
{30,    0,  0,  16, 0,  13, 0,  12, 0,  0,  0,  25, 0,  0,  0,  3,  0,  0,  0,  0,  0,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  0,  0,  0,  0,  28, 0,  0,  0,  0,  0,  0,  17, 0,  24, 0,  0,  0,  0,  0,  0,  },
{0, 0,  0,  0,  0,  9,  32, 0,  0,  20, 0,  26, 0,  0,  0,  26, 0,  0,  0,  0,  0,  0,  0,  0,  0,  },
------------------------------------
Min Cut: 14
```

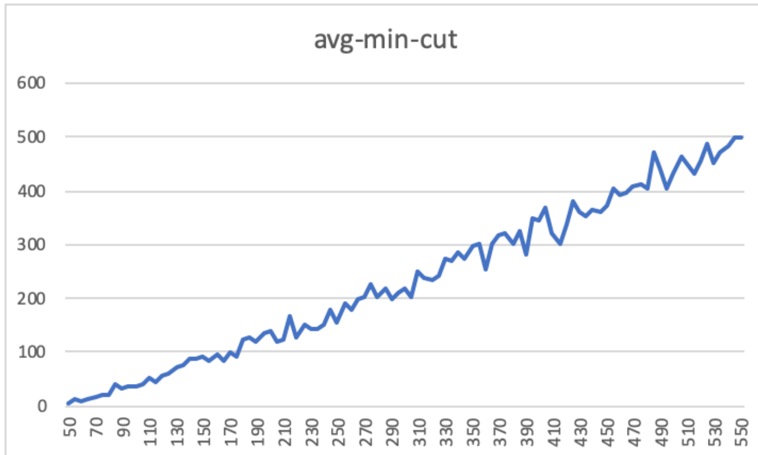
As you can see in output, it might be case where there are parallel edges generated, so in order to capture that, we combine parallel edges on the go and the count represents the number of edges in the resultant graph.

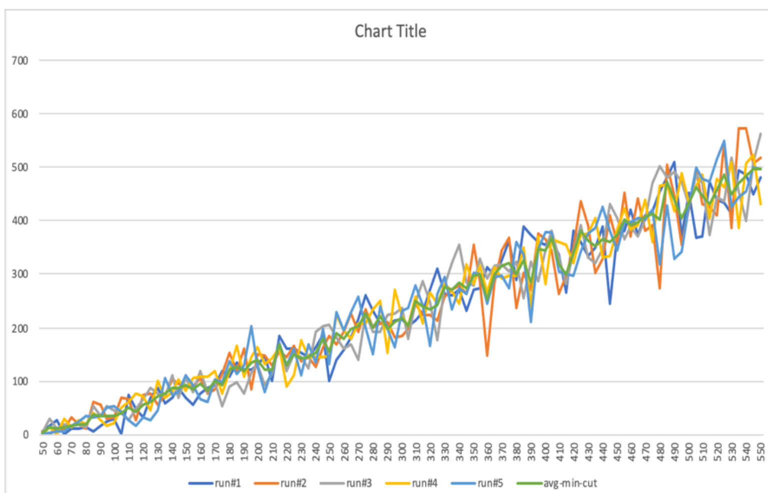
Experiments: I ran a few runs, with n = 25, and edges from 50 to 550, step size 5.

#edges	run#1	run#2	run#3	run#4	run#5	avg-min-cut
50	7	6	2	0	0	3
55	16	0	30	15	4	13
60	26	12	11	0	5	10.8
65	0	10	20	30	6	13.2
70	10	31	16	13	16	17.2
75	10	18	20	26	23	19.4
80	15	12	13	20	34	18.8
85	6	62	53	41	31	38.6
90	17	55	36	28	34	34
95	24	29	54	17	51	35
100	29	32	44	22	52	35.8
105	0	70	39	50	43	40.4
110	73	67	27	62	27	51.2
115	48	27	47	76	17	43
120	31	74	65	72	31	54.6
125	68	78	87	44	28	61
130	86	55	77	100	44	72.4
135	59	75	72	69	105	76
140	70	103	112	79	73	87.4
145	86	95	69	103	80	86.6
150	70	89	106	83	111	91.8
155	56	91	80	106	93	85.2
160	76	105	118	107	65	94.2
165	87	77	79	109	61	82.6
170	93	85	98	120	103	99.8
175	119	114	52	78	98	92.2
180	108	154	91	120	137	122
185	135	120	98	167	113	126.6
190	121	162	78	109	128	119.6
195	122	85	117	144	204	134.4
200	128	149	133	164	126	140
205	148	147	93	133	80	120.2
210	100	130	113	143	122	121.6
215	184	155	171	163	171	168.8
220	161	146	120	89	129	129
225	161	167	151	112	160	150.2
230	152	136	144	176	110	143.6
235	144	146	123	144	169	145.2
240	163	126	193	145	134	152.2
245	187	164	204	146	198	179.8
250	101	184	205	148	133	154.2
255	140	169	182	223	230	188.8
260	158	191	161	195	194	179.8
265	182	226	168	179	229	196.8
270	217	191	141	207	258	202.8
275	260	234	226	209	201	226
280	232	199	191	233	149	200.8
285	209	208	193	250	240	220
290	206	211	225	153	197	198.4
295	214	183	227	270	163	211.4
300	215	185	238	222	232	218.4
305	204	197	180	203	236	204
310	213	246	248	258	279	248.8
315	228	225	288	208	243	238.4
320	270	225	247	265	167	234.8
325	310	214	176	246	264	242
330	263	259	278	281	294	275
335	261	272	322	268	233	271.2
340	271	278	354	246	273	284.4
345	231	264	285	319	263	272.4
350	272	356	278	282	302	298
355	273	288	329	315	301	301.2
360	314	147	292	265	246	252.8
365	298	284	316	312	295	301
370	325	345	319	292	298	315.8
375	362	368	305	298	273	321.2
380	289	238	324	296	359	301.2
385	388	302	256	349	337	326.4
390	374	270	323	239	210	283.2
395	359	377	288	370	346	348
400	354	365	344	281	378	344.4
405	366	345	382	366	378	367.4
410	336	264	332	360	305	319.4
415	267	296	282	354	299	299.6
420	381	351	343	321	298	338.8
425	360	437	393	374	341	381
430	333	392	331	377	376	361.8
435	349	302	322	405	386	352.8
440	390	329	346	332	426	364.6
445	246	409	431	335	381	360.4
450	374	357	406	384	344	373
455	381	451	365	422	396	403
460	421	372	390	385	396	392.8
465	373	442	371	393	406	397
470	410	382	402	439	402	407
475	418	392	471	361	420	412.4
480	455	273	503	466	318	403
485	481	504	481	467	428	472.2
490	511	446	491	418	330	439.2
495	370	356	472	489	342	405.8
500	453	423	429	434	429	433.6
505	369	496	486	466	500	463.4
510	372	431	470	487	479	447.8
515	473	427	373	405	474	430.4
520	438	409	443	478	515	456.6
525	433	547	437	463	548	485.6
530	413	386	518	509	425	450.2
535	493	574	454	387	444	470.4
540	485	574	399	506	454	483.6
545	449	506	509	524	499	497.4
550	481	517	563	432	497	498

The edge weights are assigned uniformly at random in range [1, 32].

The following graph shows #edges vs mincut averaged over 5 runs.





Interesting Visualizations!

This demonstrates how the robustness of our network increase as graph has more edges