HIGH PERFORMANCE COMPUTING PROJECT
KRUSKAL'S ALGORITHM USING MPI IN C

CED15I038
MOHIT AGARWAL

INTRODUCTION

The parallel Kruskal's Algorithm makes multiple trees in parallel, creating a forest, where each vertex is its own separate tree. The given graph is first shorted according to their weights. Union-Find Algorithm is used for combining two trees and adding them to the forest, if the weight chosen is minimum. Find operation is used to determine which tree a particular vertex is in, and Union operation is used to merge two trees.

The format of the input file was number of vertexes and then number of edges, following that in the next row where the vertex, vertex (→the edge) and the weight of the edge.

Steps taken in the following program to calculate the MST (Mining Spanning Tree), from a given graph are:

1. First the file is read and converted into a graph using structure in C.
2. Than the graph is sorted using merge sort according to the weight in increasing order.
3. Now every edge is checked if the edges satisfy the condition for Mining Spanning Tree and if does, then edges its respective vertexes are merge together.
4. Merging is performed by sending the edge of one process to another and terminating the process which had sent the edge.
5. After going to through all the edges in the list, the MST is established.

The MPI functions used in the program are:

1. MPI_Send() and MPI_Recv()
2. MPI_comm_Rank() and MPI_comm_Size()
3. MPI_Bcast() – Broadcasting the graph to all the processes.
4. MPI_Scatter() – To scatter the graph for parallel execution.

Kruskal's Algorithm in C using MPI

```
// C standard header files

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stdio.h>
```

```c
#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <mpi.h>


const int UNSET_ELEMENT = -1;


typedef struct Set {

	int elements;

	int* canonicalElements;

	int* rank;

} Set;


typedef struct WeightedGraph {

	int edges;

	int vertices;

	int* edgeList;

} WeightedGraph;



//initialize and allocate memory for the members of the graph

void newWeightedGraph(WeightedGraph* graph, const int vertices, const int edges) {

	graph->edges = edges;

	graph->vertices = vertices;

	graph->edgeList = (int*) calloc(edges * 3, sizeof(int));

}


// read a previously generated maze file and store it in the graph

void readGraphFile(WeightedGraph* graph, const char inputFileName[]) {

	// open the file

	FILE* inputFile;
```

```
	const char* inputMode = "r";
	inputFile = fopen(inputFileName, inputMode);
	if (inputFile == NULL) {
		fprintf(stderr, "Couldn't open input file, exiting!\n");
		exit(EXIT_FAILURE);
	}

	int fscanfResult;

	// first line contains number of vertices and edges
	int vertices = 0;
	int edges = 0;
	fscanfResult = fscanf(inputFile, "%d %d", &vertices, &edges);
	newWeightedGraph(graph, vertices, edges);

	int from;
	int to;
	int weight;
	int i;
	for (i = 0; i < edges; i++) {
		fscanfResult = fscanf(inputFile, "%d %d %d", &from, &to, &weight);
		graph->edgeList[i * 3] = from;
		graph->edgeList[i * 3 + 1] = to;
		graph->edgeList[i * 3 + 2] = weight;

		if (fscanfResult == EOF) {
			fprintf(stderr,"Something went wrong during reading of graph file,
exiting!\n");
			fclose(inputFile);
			exit(EXIT_FAILURE);
		}
```

```
	}

	fclose(inputFile);
}

// print all edges of the graph in "from to weight" format
void printWeightedGraph(const WeightedGraph* graph) {
	printf("------------------------------------------------\n");
	int i, j;
	for (i = 0; i < graph->edges; i++) {
		for (j = 0; j < 3; j++) {
			printf("%d\t", graph->edgeList[i * 3 + j]);
		}
		printf("\n");
	}
	printf("------------------------------------------------\n");
}

void newSet(Set* set, const int elements) {
	set->elements = elements;
	set->canonicalElements = (int*) malloc(elements * sizeof(int));
	memset(set->canonicalElements, UNSET_ELEMENT, elements * sizeof(int));
	set->rank = (int*) calloc(elements, sizeof(int));
}

//return the canonical element of a vertex with path compression
int findSet(const Set* set, const int vertex) {
	if (set->canonicalElements[vertex] == UNSET_ELEMENT) {
		return vertex;
	}
	else {
```

```
        set->canonicalElements[vertex] = findSet(set,set->canonicalElements[vertex]);
        return set->canonicalElements[vertex];
    }
}

// merge the set of parent1 and parent2 with union by rank
void unionSet(Set* set, const int parent1, const int parent2) {
    int root1 = findSet(set, parent1);
    int root2 = findSet(set, parent2);

    if (root1 == root2) {
        return;
    }
    else if (set->rank[root1] < set->rank[root2]) {
        set->canonicalElements[root1] = root2;
    }
    else if (set->rank[root1] > set->rank[root2]) {
        set->canonicalElements[root2] = root1;
    }
    else {
        set->canonicalElements[root1] = root2;
        set->rank[root2] = set->rank[root1] + 1;
    }
}

// copy an edge
void copyEdge(int* to, int* from) {
    memcpy(to, from, 3 * sizeof(int));
}

// scatter the edge list of a graph
```

```
void scatterEdgeList(int* edgeList, int* edgeListPart, const int elements,int* elementsPart) {
	int rank;
	int size;
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);

	MPI_Scatter(edgeList, *elementsPart * 3, MPI_INT, edgeListPart,		*elementsPart * 3,
MPI_INT, 0, MPI_COMM_WORLD);

	if (rank == size - 1 && elements % *elementsPart != 0) {
		// number of elements and processes isn't divisible without remainder
		*elementsPart = elements % *elementsPart;
	}

	if (elements / 2 + 1 < size && elements != size) {
		if (rank == 0) {
			fprintf(stderr, "Unsupported size/process combination, exiting!\n");
		}
		MPI_Finalize();
		exit(EXIT_FAILURE);
	}
}

// cleanup set data
void deleteSet(Set* set) {
	free(set->canonicalElements);
	free(set->rank);
}

//cleanup graph data
void deleteWeightedGraph(WeightedGraph* graph) {
```

```
	free(graph->edgeList);
}

// merge sorted lists, start and end are inclusive
void merge(int* edgeList, const int start, const int end, const int pivot) {
	int length = end - start + 1;
	int* working = (int*) malloc(length * 3 * sizeof(int));

	// copy first part
	memcpy(working, &edgeList[start * 3],(pivot - start + 1) * 3 * sizeof(int));

	// copy second part reverse to simpify merge
	int workingEnd = end + pivot - start + 1;
	int i;
	for (i = pivot + 1; i <= end; i++) {
		copyEdge(&working[(workingEnd - i) * 3],&edgeList[i * 3]);
	}

	int left = 0;
	int right = end - start;
	int k;
	for (k = start; k <= end; k++) {
		if (working[right * 3 + 2]< working[left * 3 + 2]) {
			copyEdge(&edgeList[k * 3],&working[right * 3]);
			right--;
		} else {
			copyEdge(&edgeList[k * 3],&working[left * 3]);
			left++;
		}
	}
```

```
	// clean up
	free(working);
}

//sort the edge list using merge sort, start and end are inclusive
void mergeSort(int* edgeList, const int start, const int end) {
	if (start != end) {
		// recursively divide the list in two parts and sort them
		int pivot = (start + end) / 2;
		mergeSort(edgeList, start, pivot);
		mergeSort(edgeList, pivot + 1, end);

		merge(edgeList, start, end, pivot);
	}
}

// sort the edges of the graph in parallel with mergesort in parallel
void sort(WeightedGraph* graph) {
	int rank;
	int size;
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	MPI_Status status;

	bool parallel = size != 1;

	// send number of elements
	int elements;
	if (rank == 0) {
		elements = graph->edges;
		MPI_Bcast(&elements, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
	}
	else {
		MPI_Bcast(&elements, 1, MPI_INT, 0, MPI_COMM_WORLD);
	}

	// scatter the edges to sort
	int elementsPart = (elements + size - 1) / size;
	int* edgeListPart = (int*) malloc(elementsPart * 3 * sizeof(int));
	if (parallel) {
		scatterEdgeList(graph->edgeList, edgeListPart, elements, &elementsPart);
	} else {
		edgeListPart = graph->edgeList;
	}

	// sort the part
	mergeSort(edgeListPart, 0, elementsPart - 1);

	if (parallel) {
		// merge all parts
		int from;
		int to;
		int elementsRecieved;
		int step;
		for (step = 1; step < size; step *= 2) {
			if (rank % (2 * step) == 0) {
				from = rank + step;
				if (from < size) {
					MPI_Recv(&elementsRecieved, 1, MPI_INT, from,
0,MPI_COMM_WORLD, &status);
					edgeListPart = realloc(edgeListPart,(elementsPart +
elementsRecieved) * 3* sizeof(int));
```

```
                    MPI_Recv(&edgeListPart[elementsPart *
3],elementsRecieved * 3,MPI_INT, from, 0, MPI_COMM_WORLD, &status);
                    merge(edgeListPart, 0, elementsPart + elementsRecieved -
1,      elementsPart - 1);
                    elementsPart += elementsRecieved;
                }
            }
            else if (rank % step == 0) {
                to = rank - step;
                MPI_Send(&elementsPart, 1, MPI_INT, to, 0, MPI_COMM_WORLD);
                MPI_Send(edgeListPart, elementsPart * 3, MPI_INT, to,0,
    MPI_COMM_WORLD);
            }
        }

        // edgeListPart is the new edgeList of the graph, cleanup other memory
        if (rank == 0) {
            free(graph->edgeList);
            graph->edgeList = edgeListPart;
        } else {
            free(edgeListPart);
        }
    } else {
        graph->edgeList = edgeListPart;
    }
}

// find a MST of the graph using Kruskal's algorithm
void mstKruskal(WeightedGraph* graph, WeightedGraph* mst) {
    // create needed data structures
    Set* set = &(Set ) { .elements = 0, .canonicalElements = NULL, .rank =NULL };
    newSet(set, graph->vertices);
```

```
	int rank;
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);

	// sort the edges of the graph
	sort(graph);

	if (rank == 0) {
		// add edges to the MST
		int currentEdge = 0;
		int edgesMST;
		for (edgesMST = 0;edgesMST < graph->vertices - 1 || currentEdge < graph->edges;) {
			// check for loops if edge would be inserted
			int canonicalElementFrom = findSet(set,graph->edgeList[currentEdge * 3]);
			int canonicalElementTo = findSet(set,graph->edgeList[currentEdge * 3 + 1]);
			if (canonicalElementFrom != canonicalElementTo) {
				// add edge to MST
				copyEdge(&mst->edgeList[edgesMST * 3],&graph-
>edgeList[currentEdge * 3]);
				unionSet(set, canonicalElementFrom, canonicalElementTo);
				edgesMST++;
			}
			currentEdge++;
		}
	}

	// clean up
	deleteSet(set);
}

// main program
```

```
int main(int argc, char* argv[]) {
	// MPI variables and initialization
	int rank;
	int size;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Comm_size(MPI_COMM_WORLD, &size);

	// graph Variables
	WeightedGraph* graph = &(WeightedGraph ) { .edges = 0, .vertices = 0,.edgeList = NULL };
	WeightedGraph* mst = &(WeightedGraph ) { .edges = 0, .vertices = 0,	.edgeList = NULL };

	if (rank == 0) {

		// read the maze file and store it in the graph
		readGraphFile(graph, argv[1]);

		// print the edges of the read graph
		printf("Original Graph:\n");
		printWeightedGraph(graph);

		newWeightedGraph(mst, graph->vertices, graph->vertices - 1);
	}

	double start = MPI_Wtime();
	// use Kruskal's algorithm
	mstKruskal(graph, mst);



	if (rank == 0) {
```

```
        // print the edges of the MST
        printf("Minimum Spanning Tree (Kruskal):\n");
        printWeightedGraph(mst);

        unsigned long weightMST = 0;
        int i;
        for (i = 0; i < mst->edges; i++) {
            weightMST += mst->edgeList[i * 3 + 2];
        }
        printf("MST weight: %lu\n", weightMST);

        // cleanup
        deleteWeightedGraph(graph);
        deleteWeightedGraph(mst);

        printf("Time elapsed: %f s\n", MPI_Wtime() - start);
    }

    MPI_Finalize();

    return EXIT_SUCCESS;
}
```

Outputs of various Graphs

The above program was ran on the graph having 1000 vertexes and 25000 edges, and the MST produced is as follows:

```
-----------------------------------------------
Minimum Spanning Tree (Kruskal):
-----------------------------------------------
172     80      1
426     918     1
```

236	364	1
571	269	2
482	329	2
190	942	4
50	347	4
599	401	5
657	437	5
826	93	5
628	497	6
22	684	6
16	818	6
774	76	7
368	425	7
930	376	7
164	268	7
315	510	7
764	71	7
157	431	8
975	165	8
603	991	8
437	663	8
871	595	9
709	35	10
291	197	10
659	701	10
257	860	10
97	472	10
289	276	10
99	178	10
119	710	11
973	579	11

931	86	12
15	627	13
716	430	13
958	642	13
577	645	13
786	120	14
850	311	14
397	35	15
151	120	15
674	968	15
854	733	15
533	826	16
282	640	17
912	565	17
306	329	18
848	721	20
506	871	20
211	694	20
356	908	20
836	959	21
779	335	21
495	472	21
548	904	22
748	842	22
332	649	23
928	412	23
109	843	23
887	155	23
787	677	23
74	998	23
172	709	24

112	799	24
675	151	24
329	123	25
787	144	25
733	787	25
116	152	26
260	722	27
212	812	27
577	779	27
328	757	28
345	368	28
472	121	28
447	377	28
72	317	28
17	407	29
70	290	29
482	591	29
133	952	30
563	534	30
105	540	30
544	898	31
362	86	31
267	604	31
202	548	31
636	5	32
618	249	32
547	786	32
737	412	33
442	344	33
953	407	33
180	644	33

776	206	34
640	83	34
971	221	35
259	260	35
990	427	35
766	552	36
809	930	36
800	872	36
780	179	36
369	547	37
959	56	37
300	120	37
834	706	37
390	511	38
993	288	39
957	313	39
213	8	39
484	27	40
294	311	40
882	514	41
9	731	41
202	71	41
1	532	41
905	316	42
939	805	42
577	383	43
858	948	43
762	571	44
344	406	44
140	351	45
758	223	46

865	71	46
499	569	46
39	768	47
989	303	47
413	30	47
454	642	48
265	914	49
579	241	50
169	864	51
415	118	51
83	329	51
888	441	52
973	662	52
404	52	53
404	385	53
87	615	53
420	116	54
920	870	54
181	119	56
997	369	56
881	700	56
280	568	57
755	657	57
36	511	58
895	948	58
540	401	58
174	427	58
980	193	58
809	729	59
651	84	59
65	322	60

208	460	60
202	36	61
622	595	61
763	114	61
255	916	62
46	490	62
247	172	62
96	529	62
959	837	62
802	426	63
820	293	63
870	939	63
140	784	64
762	549	64
226	706	65
513	944	65
3	963	65
428	178	65
919	181	66
910	347	66
905	227	67
46	385	68
402	632	69
812	161	69
447	953	69
14	182	69
682	599	70
553	597	71
644	185	71
510	821	71
334	808	71

649	900	72
381	642	72
585	249	73
184	384	73
9	311	74
337	618	74
162	581	74
270	622	75
14	606	75
785	48	75
100	593	75
619	325	75
212	160	75
945	277	75
334	84	75
293	505	75
54	305	76
82	197	76
19	753	76
467	163	77
251	211	77
302	442	78
808	321	78
240	79	79
17	809	79
598	952	80
841	812	80
940	901	80
942	780	80
31	738	80
872	727	81

714	552	81
338	128	82
765	151	82
369	281	83
596	439	83
643	941	84
107	213	85
934	887	85
784	614	85
996	183	86
277	370	86
305	817	86
283	402	87
944	273	87
362	862	87
650	162	87
114	92	87
771	253	89
460	4	89
428	586	90
502	939	90
844	780	90
204	804	90
800	278	91
799	337	91
894	710	92
433	883	92
451	771	93
84	832	94
845	46	94
890	533	95

422	854	95
68	980	95
439	766	95
118	474	96
804	580	97
241	731	97
174	862	99
93	451	99
939	133	99
927	938	100
171	359	100
70	654	100
959	263	100
952	360	101
501	73	101
787	281	101
794	915	101
257	342	101
609	255	101
376	16	102
733	452	103
889	893	103
846	977	103
410	154	103
985	850	104
878	864	104
725	227	105
81	286	105
585	639	106
917	200	107
426	675	108

728	508	108
328	582	108
687	368	108
99	793	108
681	421	110
750	234	110
565	108	110
377	66	111
981	162	111
521	919	112
493	186	113
154	718	114
414	616	114
509	789	114
357	961	114
43	207	114
565	83	115
962	956	115
919	90	116
688	11	116
514	572	117
862	161	118
274	941	119
748	36	119
513	24	120
674	581	120
852	70	121
184	87	121
278	608	122
136	698	122
763	688	123

605	59	123
401	390	124
468	703	124
869	882	124
880	995	124
917	973	125
809	876	125
481	243	125
263	233	125
993	174	126
674	372	126
18	111	126
345	645	127
68	168	127
77	219	127
993	969	128
2	478	129
211	8	129
193	617	130
249	383	130
788	419	130
24	35	130
14	516	132
965	646	132
704	557	132
711	133	133
260	75	133
975	47	134
850	183	135
53	715	135
682	586	136

284	303	136
645	297	136
573	219	136
638	537	137
769	698	137
216	423	138
418	384	138
651	67	138
741	609	138
287	956	139
866	708	140
724	840	141
437	545	143
586	708	143
609	409	143
236	11	143
688	405	144
182	162	144
266	320	146
349	786	147
888	703	147
513	409	147
501	297	148
145	625	148
130	217	148
983	442	149
153	305	149
938	441	149
917	186	149
447	301	150
347	254	150

137	782	150
495	610	150
805	574	150
12	633	151
926	866	152
35	815	152
400	294	154
778	856	154
183	133	155
904	591	156
904	562	157
734	845	157
317	950	157
186	734	157
23	10	158
998	108	158
999	617	158
207	564	158
725	454	158
91	752	159
44	427	160
582	848	161
867	580	161
716	450	163
410	669	163
775	765	164
184	803	164
825	360	164
634	153	165
504	408	165
521	942	165

408	442	165
281	57	165
570	152	165
452	796	166
119	675	166
814	651	166
628	424	166
773	225	166
103	633	167
271	634	167
710	61	167
507	460	168
999	956	168
627	467	169
373	107	169
517	473	170
554	191	171
892	377	171
595	189	172
975	740	172
771	536	172
552	514	173
756	506	173
508	400	173
438	782	174
744	730	174
927	360	175
564	115	175
545	452	176
976	266	176
905	737	176

621	246	177
171	599	178
21	64	178
982	139	178
664	371	179
943	803	179
227	376	179
200	402	180
381	755	180
900	971	181
383	696	181
452	501	182
684	475	183
311	282	183
763	741	183
491	637	183
203	921	184
214	677	184
682	700	184
888	411	184
861	918	185
65	502	185
538	349	186
963	785	187
307	591	188
207	386	188
423	629	188
13	0	189
480	323	190
978	270	190
493	459	190

134	524	191
399	345	192
709	589	193
924	896	193
942	203	194
125	431	194
366	395	195
870	348	196
692	407	198
342	378	198
446	526	198
231	791	198
678	543	198
916	695	199
486	634	200
317	209	200
753	40	200
762	611	200
452	260	200
782	509	201
984	58	201
18	895	201
739	968	201
597	624	202
51	641	202
102	634	202
715	384	203
285	708	203
27	518	203
970	150	204
511	923	204

645	429	204
70	841	204
422	484	205
920	560	205
269	693	206
269	265	206
165	125	206
518	614	206
462	254	206
301	635	206
616	867	206
510	462	207
257	953	207
970	336	207
884	499	207
783	172	207
406	743	208
767	5	208
723	892	208
632	126	209
531	275	210
255	798	210
160	378	211
528	43	211
120	198	211
437	877	211
581	978	212
327	662	212
414	618	213
264	624	214
863	334	214

608	303	215
973	257	215
225	878	215
104	746	216
686	126	216
611	400	217
999	226	217
852	846	217
29	742	218
759	397	218
982	560	219
731	20	220
121	119	220
946	729	221
668	345	222
746	754	222
219	12	222
201	514	222
407	338	223
925	503	223
365	475	223
323	325	224
619	833	224
883	990	224
155	752	224
789	944	224
530	567	224
95	920	224
496	629	225
409	522	226
292	397	228

546	113	229
585	669	229
386	907	230
464	736	231
999	255	231
341	196	231
819	259	232
913	729	232
304	297	232
332	506	233
665	814	233
264	380	233
15	178	234
207	466	235
73	751	235
970	811	235
20	405	236
187	453	236
915	486	237
546	578	238
97	89	239
492	711	239
911	858	239
817	763	239
710	558	239
497	904	240
454	156	240
262	931	241
196	927	241
791	878	243
955	373	244

768	161	244
489	324	244
923	535	245
825	736	246
531	293	246
877	896	246
940	353	247
713	485	247
239	801	248
192	43	248
206	882	249
662	267	250
304	851	250
716	245	252
286	663	252
436	935	253
907	723	254
165	643	254
626	999	255
153	208	255
263	345	255
803	226	255
656	203	255
814	116	256
847	279	256
479	992	256
390	856	256
477	283	256
162	11	257
512	21	257
203	979	258

901	699	258
910	478	259
515	615	259
404	880	260
250	865	260
725	450	261
130	926	261
112	531	261
735	761	262
145	274	263
324	354	263
175	493	264
125	509	264
171	541	264
927	485	264
584	520	265
307	785	266
943	829	267
850	106	267
536	132	267
402	638	267
567	21	267
780	637	268
605	430	268
10	178	269
462	362	272
787	806	272
279	327	272
882	238	273
759	569	273
204	738	274

196	218	275
925	917	275
534	701	275
981	29	275
77	695	276
41	756	277
2	67	277
548	434	278
389	965	278
745	375	279
443	344	280
41	155	281
701	559	281
750	864	282
702	410	282
752	563	283
798	530	284
843	879	285
358	882	286
448	398	286
203	481	287
909	741	289
978	769	291
449	777	291
637	380	292
443	335	293
900	448	293
897	503	294
827	821	294
468	272	295
717	469	296

4	235	299
788	982	299
688	583	299
732	876	299
283	194	300
231	719	301
209	618	301
436	254	301
759	828	302
261	619	303
466	584	303
807	467	303
633	666	304
858	977	304
701	215	305
922	468	305
683	653	306
474	777	306
350	355	306
592	203	307
374	494	308
216	746	309
631	458	309
367	462	310
188	868	310
588	979	311
231	379	312
393	154	313
163	557	314
958	224	315
568	527	315

750	137	317
937	473	318
658	977	319
841	476	319
38	334	319
551	194	320
346	374	320
458	860	321
524	187	322
727	668	322
991	723	323
984	646	323
292	382	326
676	456	327
45	456	328
726	152	328
641	706	330
95	833	330
778	30	331
473	8	332
654	201	332
735	440	332
717	827	332
577	699	333
356	639	333
430	960	333
529	904	334
947	470	335
138	654	337
253	5	337
100	403	338

86	166	338
72	811	338
110	715	339
951	186	339
553	256	339
850	135	340
542	389	341
349	859	343
367	373	343
139	600	343
672	396	344
84	899	344
37	569	345
222	835	346
432	459	346
455	456	347
476	180	348
309	567	348
775	240	349
502	440	349
146	973	350
30	648	350
289	651	352
226	838	352
657	216	353
78	836	354
336	113	355
279	449	355
124	483	356
786	244	357
666	158	358

229	796	358
177	62	358
23	898	358
714	691	359
393	40	360
855	269	360
801	850	361
616	195	361
321	230	363
45	247	364
537	823	365
688	308	367
159	164	367
416	767	368
197	518	370
34	686	371
721	13	372
313	515	373
613	201	374
446	244	374
509	623	374
749	911	375
417	246	375
252	7	375
407	664	375
972	339	376
90	621	377
109	246	378
972	857	379
519	608	383
242	862	386

642	312	386
899	124	387
932	315	391
993	612	393
632	587	393
403	295	393
465	991	394
518	590	394
131	416	394
933	682	395
16	350	396
275	906	396
419	543	398
795	473	398
705	240	398
220	970	399
387	736	399
126	463	401
630	155	401
435	53	402
389	716	402
974	61	405
690	839	405
922	228	406
69	240	407
988	139	407
417	767	410
794	967	411
222	762	411
457	721	412
575	521	413

848	714	415
839	170	416
42	188	416
43	550	417
364	268	417
853	410	417
592	330	417
232	799	417
554	127	417
712	114	417
28	683	417
331	901	418
976	380	418
661	408	418
893	640	419
28	31	419
824	753	420
271	62	420
839	986	421
295	469	422
499	318	423
253	142	424
788	744	424
966	189	431
792	386	431
454	690	431
167	793	432
943	26	432
210	237	432
2	370	433
561	463	433

388	86	435
393	85	437
563	210	439
830	946	439
527	314	444
824	655	445
511	681	447
797	626	447
677	148	451
357	999	456
51	319	456
587	987	457
724	555	458
396	887	459
207	129	461
101	441	462
324	1	462
972	105	464
224	117	464
42	579	466
770	266	466
76	712	471
749	333	471
413	32	472
577	724	474
463	141	477
903	809	478
252	722	480
461	604	482
644	566	482
929	516	484

424	620	488
345	280	489
707	452	489
462	601	489
143	728	490
441	670	493
962	689	494
487	798	496
765	992	497
176	407	502
94	488	505
394	967	512
55	653	512
475	103	513
556	533	513
772	803	514
392	145	514
623	810	517
874	407	524
525	656	525
873	378	528
299	541	531
998	375	538
566	697	542
725	813	542
871	199	543
319	849	544
346	479	552
470	589	553
490	524	555
737	445	557

679	400	558
532	916	561
332	607	563
471	796	570
127	844	571
652	796	576
214	790	583
964	965	584
647	384	585
667	261	585
816	910	587
737	298	591
999	122	604
673	28	610
9	747	611
464	98	612
996	498	613
857	488	615
2	173	622
758	842	630
78	63	632
674	954	634
456	6	637
671	169	638
894	539	640
484	395	641
49	161	643
66	760	657
147	291	660
732	602	673
352	303	675

205	234	676
361	887	681
509	949	681
25	212	683
522	902	687
580	248	689
326	266	694
350	685	694
945	680	717
994	11	722
594	465	728
82	660	734
355	149	740
720	11	745
391	547	747
760	444	773
728	886	786
60	319	803
331	363	804
516	885	810
875	665	814
126	258	838
310	344	860
88	726	866
148	781	887
348	523	888
794	500	930
130	296	938
868	340	946
831	458	947
495	576	980

936	552	987
880	822	1159
891	624	1191
343	224	1210
648	33	1302

MST weight: 241238

Time elapsed: 0.007906 s