Algorithms Homework 4: Minimum Set Cover

Thomas Lent

Algorithm

This program implements a backtracking algorithm for finding a minimum set cover of given sets. The program recursively chooses different combinations of sets until the minimum set cover has been found. The following checks are made in this order to reduce the number of set combinations that need to be calculated:

- If a set is a subset of another set, remove the subset from consideration. There is no reason to consider a set that gives the same elements as a larger set. Selecting a subset of another set covers fewer elements for the same cost in the set cover. When considering whether or not a set is a subset of another set we only consider the elements in each set that are still uncovered.
- If an uncovered element is only found in a single set, we select that set as it is necessary in any valid set cover.
- If neither of the above cases is true, we must consider selecting or not selecting a set. Because every set is either in the minimum cover or not this considers every possibility for any sets that we cannot eliminate with one of the two cases above.
    - We always consider the set with the most uncovered elements remaining because it is most likely to be in the minimum set cover and to cause the above cases to be true for future sets.
    - For the minimum set cover to be exact, we must also consider the set cover if we do not choose this set. We will choose whichever set cover is smaller out of the set cover containing this set and the set cover not containing this set.

Source Code

bitset.h:

```h
#include <stdlib.h>

#define TYPE unsigned int

typedef struct {
    size_t capacity;
    size_t size;
    size_t bit_array_count;
    TYPE *bit_arrays;
} BitSet;

BitSet *bitset_init(size_t capacity);
int bitset_add(BitSet *bitset, int value);
int bitset_remove(BitSet *bitset, int value);
int bitset_contains(BitSet *bitset, int value);
int bitset_is_subset(BitSet *superset, BitSet *subset);
int bitset_is_empty(BitSet *bitset);
BitSet *bitset_difference(BitSet *a, BitSet *b);
BitSet *bitset_union(BitSet *a, BitSet *b);
BitSet *bitset_copy(BitSet *bitset);
void bitset_print(BitSet *bitset);
void bitset_free(BitSet *bitset);
```

bitset.c:

```c
#include <math.h>
#include <string.h>
#include <stdio.h>
#include "bitset.h"

size_t ARRAY_BITS = sizeof(TYPE) * 8;

BitSet *bitset_init(size_t capacity) {
    BitSet *bitset = malloc(sizeof(BitSet));
    if (!bitset)
        return NULL;
    size_t necessary_arrays = 1 + capacity / ARRAY_BITS;
    bitset->bit_arrays = calloc(necessary_arrays, sizeof(TYPE));
    bitset->bit_array_count = necessary_arrays;
    bitset->capacity = capacity;
    bitset->size = 0;
    return bitset;
}

int bitset_add(BitSet *bitset, int value) {
    if (!bitset || value > bitset->capacity)
        return 0;
    size_t array = value / ARRAY_BITS;
    int position = value % ARRAY_BITS;
    TYPE bit = 1;
    bitset->bit_arrays[array] |= (bit << position);
    ++bitset->size;
```

```
    return 1;
}

int bitset_remove(BitSet *bitset, int value) {
    if (!bitset || value > bitset->capacity)
        return 0;
    size_t array = value / ARRAY_BITS;
    int position = value % ARRAY_BITS;
    TYPE bit = 1;
    bitset->bit_arrays[array] &= ~(bit << position);
    --bitset->size;
    return 1;
}

int bitset_contains(BitSet *bitset, int value) {
    if (!bitset || value > bitset->capacity || bitset_is_empty(bitset))
        return 0;
    size_t array = value / ARRAY_BITS;
    int position = value % ARRAY_BITS;
    TYPE bit = 1;
    TYPE bit_array = bitset->bit_arrays[array];
    return bit_array != 0 && (bit_array & (bit << position)) != 0;
}

int bitset_is_subset(BitSet *superset, BitSet *subset) {
    if (!superset || !subset || superset->capacity != subset->capacity
            || subset->size > superset->size)
        return 0;
    for (size_t i = 0; i < superset->bit_array_count; i++) {
        TYPE superset_array = superset->bit_arrays[i];
        if ((superset_array | subset->bit_arrays[i]) != superset_array)
            return 0;
    }
    return 1;
}

int bitset_is_empty(BitSet *bitset) {
    if (!bitset)
        return 1;
    return bitset->size == 0;
}

BitSet *bitset_difference(BitSet *a, BitSet *b) {
    if (!a || !b || a->capacity != b->capacity)
        return NULL;
    BitSet *difference_bitset = bitset_init(a->capacity);
    if (!difference_bitset)
        return NULL;
    size_t total_bit_count = 0;
    for (size_t i = 0; i < difference_bitset->bit_array_count; i++) {
        TYPE a_array = a->bit_arrays[i];
        TYPE b_array = b->bit_arrays[i];
        TYPE diff_array = a_array & ~b_array;
        difference_bitset->bit_arrays[i] = diff_array;
        TYPE bit_array = diff_array;
        TYPE bit_count;
        for (bit_count = 0; bit_array; bit_count++) {
            bit_array &= bit_array - 1;
        }
        total_bit_count += bit_count;
```

```
    }
    difference_bitset->size = total_bit_count;
    return difference_bitset;
}

BitSet *bitset_union(BitSet *a, BitSet *b) {
    if (!a || !b || a->capacity != b->capacity)
        return NULL;
    BitSet *union_bitset = bitset_init(a->capacity);
    if (!union_bitset)
        return NULL;
    size_t total_bit_count = 0;
    for (size_t i = 0; i < union_bitset->bit_array_count; i++) {
        TYPE a_array = a->bit_arrays[i];
        TYPE b_array = b->bit_arrays[i];
        union_bitset->bit_arrays[i] = a_array | b_array;
        TYPE bit_array = union_bitset->bit_arrays[i];
        TYPE bit_count;
        for (bit_count = 0; bit_array; bit_count++) {
            bit_array &= bit_array - 1;
        }
        total_bit_count += bit_count;
    }
    union_bitset->size = total_bit_count;
    return union_bitset;
}

BitSet *bitset_copy(BitSet *bitset) {
    if (!bitset)
        return NULL;
    BitSet *new_bitset = bitset_init(bitset->capacity);
    if (!new_bitset)
        return NULL;
    new_bitset->size = bitset->size;
    memcpy(new_bitset->bit_arrays, bitset->bit_arrays,
        sizeof(TYPE) * (bitset->bit_array_count));
    return new_bitset;
}

void bitset_print(BitSet *bitset) {
    if (bitset) {
        printf("[");
        int first_print = 1;
        for (size_t i = 0; i <= bitset->capacity; i++) {
            if (bitset_contains(bitset, i)) {
                if (first_print) {
                    printf("%zu", i);
                    first_print = 0;
                } else
                    printf(", %zu", i);
            }
        }
        printf("]\n");
    }
}

void bitset_free(BitSet *bitset) {
    free(bitset->bit_arrays);
    free(bitset);
}
```

set.h:

```c
#include <stdlib.h>
#include "bitset.h"

typedef struct {
    unsigned int set_number;
    BitSet *elements;
} Set;

Set *set_init(unsigned int set_number, size_t capacity, BitSet *elements);
Set *set_difference(Set *a, Set *b);
Set *set_union(Set *a, Set *b);
Set *set_copy(Set *set);
void set_print(Set *set);
void set_free(Set *set);
```

set.c:

```c
#include "set.h"
#include <stdio.h>

Set *set_init(unsigned int set_number, size_t capacity, BitSet *elements) {
    Set *set = malloc(sizeof(Set));
    if (!set)
        return NULL;
    set->set_number = set_number;
    if (!elements)
        set->elements = bitset_init(capacity);
    else
        set->elements = elements;
    if (!set->elements)
        return NULL;
    return set;
}

Set *set_difference(Set *a, Set *b) {
    if (!a || !b)
        return NULL;
    BitSet *set_difference = bitset_difference(a->elements, b->elements);
    if (!set_difference)
        return NULL;
    return set_init(a->set_number, a->elements->capacity, set_difference);
}

Set *set_union(Set *a, Set *b) {
    if (!a || !b)
        return NULL;
    BitSet *set_union = bitset_union(a->elements, b->elements);
    if (!set_union)
        return NULL;
    return set_init(a->set_number, a->elements->capacity, set_union);
}

Set *set_copy(Set *set) {
    if (!set)
```

```
        return NULL;
    BitSet *new_bitset = bitset_copy(set->elements);
    if (!new_bitset)
        return NULL;
    Set *new_set = set_init(set->set_number, set->elements->capacity, new_bitset);
    if (!new_set) {
        bitset_free(new_bitset);
        return NULL;
    }
    return new_set;
}

void set_print(Set *set) {
    if (set) {
        printf("Set #%d: ", set->set_number);
        bitset_print(set->elements);
    }
}

void set_free(Set *set) {
    bitset_free(set->elements);
    free(set);
}
```

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include "vector.h"
#include "set.h"

static const char *usage = "Usage: %s input_filename\n";
static const int max_line_length = 1000;

void usage_error(const char *errorMessage, const char *argv[]) {
    fprintf(stderr, "%s\n\n", errorMessage);
    fprintf(stderr, usage, argv[0]);
}

int read_data_from_file(const char *filename, Vector *sets,
        size_t *element_count, size_t *set_count) {
    if (!filename || !sets || !element_count || !set_count)
        return 0;
    FILE *fp;
    if ((fp = fopen(filename, "r"))) {
        int success = 1;
        if (success && !(fscanf(fp, "%zu\n", element_count)
                && fscanf(fp, "%zu\n", set_count))) {
            success = 0;
        }
```

```
        char line[max_line_length];
        for (unsigned int i = 0; i < *set_count && success; i++) {
            Set *set = set_init(i + 1, *element_count, NULL);
            if (!set)
                success = 0;
            if (!fgets(line, max_line_length, fp))
                success = 0;
            const char *delimeters = " \n";
            char *elementStr = strtok(line, delimeters);
            while (elementStr && success) {
                int element = atoi(elementStr);
                if(!bitset_add(set->elements, element))
                    success = 0;
                elementStr = strtok(NULL, delimeters);
            }
            if (!vector_append(sets, set))
                success = 0;
        }
        fclose(fp);
        return success;
    }
    return 0;
}

void print_set_cover(BitSet *set_cover, Vector *original_sets, float run_time) {
    printf("Found minimum set cover containing %zu sets", set_cover->size);
    if (run_time)
        printf(" in %.6f seconds", run_time);
    printf(".\n");
    printf("Included sets: ");
    bitset_print(set_cover);
    for (size_t i = 0; i < original_sets->size; i++) {
        Set *set = vector_index(original_sets, i);
        if (bitset_contains(set_cover, set->set_number)) {
            set_print(set);
        }
    }
}

Vector *copy_sets(Vector *sets) {
    Vector *new_vector = vector_init(sets->capacity);
    for (size_t i = 0; i < sets->size; i++) {
        Set *set = vector_index(sets, i);
        if (!set)
            return NULL;
        Set *new_set = set_copy(set);
        if (!new_set)
            return NULL;
        if (!vector_append(new_vector, new_set)) {
```

```
            set_free(new_set);
            return NULL;
        }
    }
    return new_vector;
}

void free_sets(Vector *sets) {
    for (size_t i = 0; i < sets->size; i++) {
        set_free(vector_index(sets, i));
    }
    vector_free(sets);
}

int select_set(Vector *sets, Set *selected_set) {
    for (size_t i = 0; i < sets->size; i++) {
        Set *set = vector_index(sets, i);
        Set *diff = set_difference(set, selected_set);
        if (!diff)
            return 0;
        if (!bitset_is_empty(diff->elements)) {
            if (!vector_set(sets, i, diff))
                return 0;
        } else {
            Set *removed_set = vector_remove(sets, i);
            if (!removed_set)
                return 0;
            set_free(removed_set);
            set_free(diff);
        }
        set_free(set);
    }
    return 1;
}

BitSet *minimum_set_cover(Vector *sets, BitSet *covered_elements,
                          size_t set_count, size_t element_count) {
    if (!sets || !covered_elements)
        return NULL;
    if (vector_is_empty(sets))
        return bitset_init(set_count);
    // Prune all sets that are subsets of another set
    for (size_t i = 0; i < sets->size; i++) {
        for (size_t j = 0; j < sets->size; j++) {
            if (i != j) {
                Set *a = vector_index(sets, i);
                Set *b = vector_index(sets, j);
                if (!a || !b) {
                    return NULL;
```

```
                }
                if (bitset_is_subset(a->elements, b->elements)) {
                    Vector *new_sets = copy_sets(sets);
                    if (!new_sets)
                        return NULL;
                    Set *removed_set = vector_remove(new_sets, j);
                    if (!removed_set) {
                        free_sets(new_sets);
                        return NULL;
                    }
                    set_free(removed_set);
                    BitSet *msc = minimum_set_cover(new_sets, covered_elements,
                        set_count, element_count);
                    free_sets(new_sets);
                    return msc;
                }
            }
        }
    }
    // Select any sets which contain an element not found in any other set
    for (unsigned int i = 1; i < element_count + 1; i++) {
        if (!bitset_contains(covered_elements, i)) {
            int containing_sets = 0;
            size_t last_containing_index = 0;
            for (size_t index = 0; index < sets->size; index++) {
                Set *set = vector_index(sets, index);
                if (bitset_contains(set->elements, i)) {
                    last_containing_index = index;
                    ++containing_sets;
                }
                if (containing_sets > 1)
                    break;
            }
            if (containing_sets == 0) { // an uncovered element is not in any set
                return NULL;
            }
            if (containing_sets == 1) {
                Set *containing_set = vector_index(sets, last_containing_index);
                BitSet *new_covered_elements = bitset_union(covered_elements,
                    containing_set->elements);
                if (!new_covered_elements) {
                    return NULL;
                }
                Vector *new_sets = copy_sets(sets);
                if (!new_sets) {
                    return NULL;
                }
                Set *removed_set = vector_remove(new_sets, last_containing_index);
                if (!removed_set) {
```

```
                    free_sets(new_sets);
                    return NULL;
                }
                set_free(removed_set);
                if (!select_set(new_sets, containing_set)) {
                    return NULL;
                }
                BitSet *msc = minimum_set_cover(new_sets, new_covered_elements,
                    set_count, element_count);
                free_sets(new_sets);
                bitset_free(new_covered_elements);
                if (!msc) {
                    return NULL;
                }
                if (!bitset_add(msc, containing_set->set_number)) {
                    return NULL;
                }
                return msc;
            }
        }
    }
    // Test both selecting and not selecting the largest remaining set. Choose
    // the option that creates a smaller cover.
    size_t largest_set_index = 0;
    Set *largest_set = vector_index(sets, largest_set_index);
    Set *cursor;
    for (size_t i = 1; i < sets->size; i++) {
        cursor = vector_index(sets, i);
        if (cursor->elements->size > largest_set->elements->size) {
            largest_set_index = i;
            largest_set = cursor;
        }
    }
    // MSC without largest set
    Vector *new_sets_without_largest = copy_sets(sets);
    if (!new_sets_without_largest)
        return NULL;
    Set *removed_set = vector_remove(new_sets_without_largest, largest_set_index);
    if (!removed_set) {
        free_sets(new_sets_without_largest);
        return NULL;
    }
    set_free(removed_set);
    BitSet *msc_without_largest = minimum_set_cover(new_sets_without_largest,
        covered_elements, set_count, element_count);
    free_sets(new_sets_without_largest);
    if (!msc_without_largest)
        return NULL;
    // MSC with largest set
```

```
    BitSet *new_covered_elements = bitset_union(covered_elements,
        largest_set->elements);
    if (!new_covered_elements) {
        bitset_free(msc_without_largest);
        return NULL;
    }
    Vector *new_sets_with_largest = copy_sets(sets);
    if (!new_sets_with_largest) {
        bitset_free(msc_without_largest);
        return NULL;
    }
    removed_set = vector_remove(new_sets_with_largest, largest_set_index);
    if (!removed_set) {
        free_sets(new_sets_with_largest);
        bitset_free(msc_without_largest);
        return NULL;
    }
    set_free(removed_set);
    if (!select_set(new_sets_with_largest, largest_set)) {
        free_sets(new_sets_with_largest);
        bitset_free(msc_without_largest);
        return NULL;
    }
    BitSet *msc_with_largest = minimum_set_cover(new_sets_with_largest,
        new_covered_elements, set_count, element_count);
    free_sets(new_sets_with_largest);
    bitset_free(new_covered_elements);
    if (!msc_with_largest) {
        bitset_free(msc_without_largest);
        return NULL;
    }
    if (!bitset_add(msc_with_largest, largest_set->set_number)) {
        bitset_free(msc_without_largest);
        return NULL;
    }
    // Compare the 2 MSCs and choose the one with fewer sets.
    if (msc_with_largest->size < msc_without_largest->size){
        bitset_free(msc_without_largest);
        return msc_with_largest;
    }
    bitset_free(msc_with_largest);
    return msc_without_largest;
}

int main(int argc, const char *argv[]) {
    struct timeval start, end;
    if (argc < 2) {
        usage_error("Missing required filename argument.", argv);
        return 1;
```

```
    } else if (argc > 2) {
        usage_error("Too many arguments supplied.", argv);
        return 1;
    } else {
        const char *filename = argv[1];
        size_t element_count;
        size_t set_count;
        Vector *sets = vector_init(16);
        if (!sets)
            return 1;
        if (!read_data_from_file(filename, sets, &element_count, &set_count)) {
            printf("Error reading file.\n");
            vector_free(sets);
            return 1;
        }
        Vector *original_sets = copy_sets(sets);
        BitSet *covered_elements = bitset_init(element_count);
        if (!covered_elements || !original_sets) {
            printf("Memory error.\n");
            free_sets(sets);
            if (covered_elements)
                bitset_free(covered_elements);
            if (original_sets)
                free_sets(original_sets);
            return 1;
        }
        gettimeofday(&start, NULL);
        BitSet *set_cover = minimum_set_cover(sets, covered_elements,
            set_count, element_count);
        gettimeofday(&end, NULL);
        float run_time = 0;
        run_time = (float) (end.tv_usec - start.tv_usec) / 1000000 +
            (float) (end.tv_sec - start.tv_sec);
        bitset_free(covered_elements);
        free_sets(sets);
        if (!set_cover) {
            printf("Error finding minimum set cover.\n");
            free_sets(original_sets);
            return 1;
        }
        print_set_cover(set_cover, original_sets, run_time);
        bitset_free(set_cover);
        free_sets(original_sets);
    }
    return 0;
}
```

Outputs

s-X-12-6

```
Found minimum set cover containing 3 sets in 0.000007 seconds.
Included sets: [3, 4, 5]
Set #3: [1, 4, 7, 10]
Set #4: [2, 5, 7, 8, 11]
Set #5: [3, 6, 9, 12]
```

s-rg-8-10

```
Found minimum set cover containing 4 sets in 0.000011 seconds.
Included sets: [2, 4, 5, 8]
Set #2: [2, 3]
Set #4: [1, 3, 4]
Set #5: [5, 6, 7]
Set #8: [5, 8]
```

s-rg-31-15

```
Found minimum set cover containing 9 sets in 0.000085 seconds.
Included sets: [3, 4, 5, 6, 10, 11, 12, 14, 15]
Set #3: [1, 11, 12, 13]
Set #4: [4, 14, 15, 16, 17, 18]
Set #5: [11, 19]
Set #6: [5, 20]
Set #10: [6, 16, 20, 24]
Set #11: [2, 7, 25, 27, 28, 29]
Set #12: [8, 12, 21, 28, 30]
Set #14: [9, 18, 22, 26, 31]
Set #15: [3, 10, 23, 31]
```

s-rg-40-20

```
Found minimum set cover containing 10 sets in 0.000117 seconds.
Included sets: [1, 6, 7, 8, 9, 10, 12, 13, 18, 19]
Set #1: [1, 2, 3]
Set #6: [4, 17, 18, 19, 20]
Set #7: [7, 17, 21, 22]
Set #8: [1, 11, 23]
Set #9: [8, 12, 15, 24, 25, 26, 27]
Set #10: [28, 29]
```

```
Set #12: [13, 16, 18, 30, 32]
Set #13: [5, 9, 24, 33, 34, 35, 36, 37]
Set #18: [10, 26, 32, 38, 40]
Set #19: [6, 14, 29, 31, 36, 39]
```

s-k-20-30

```
Found minimum set cover containing 6 sets in 0.000148 seconds.
Included sets: [2, 3, 5, 6, 10, 20]
Set #2: [3, 7, 14, 18, 20]
Set #3: [1, 2, 5, 9, 16]
Set #5: [4, 6, 8, 15, 18]
Set #6: [4, 12, 13, 16, 17]
Set #10: [3, 11, 13, 17, 19]
Set #20: [10]
```

s-k-20-35

```
Found minimum set cover containing 6 sets in 0.000398 seconds.
Included sets: [3, 5, 6, 9, 11, 13]
Set #3: [6, 8, 11, 13, 17]
Set #5: [2, 5, 8, 12, 20]
Set #6: [1, 9, 14, 16, 17]
Set #9: [4, 11, 12, 15, 17]
Set #11: [7, 8, 10, 11, 19]
Set #13: [1, 3, 8, 18, 20]
```

s-k-30-50

```
Found minimum set cover containing 9 sets in 0.000480 seconds.
Included sets: [1, 2, 5, 8, 10, 11, 17, 18, 25]
Set #1: [8, 12, 15, 17, 21]
Set #2: [8, 10, 14, 22, 30]
Set #5: [7, 10, 16, 17, 29]
Set #8: [1, 2, 9, 12, 23]
Set #10: [3, 6, 11, 12, 20]
Set #11: [4, 22, 25, 26, 27]
Set #17: [18, 19, 24, 28, 29]
Set #18: [2, 13, 18, 19, 29]
Set #25: [5]
```

s-k-30-55

Found minimum set cover containing 9 sets in 0.002003 seconds.
Included sets: [2, 5, 12, 16, 18, 19, 23, 24, 51]
Set #2: [6, 7, 13, 14, 27]
Set #5: [3, 6, 10, 13, 19]
Set #12: [2, 4, 15, 20, 25]
Set #16: [9, 12, 17, 22, 29]
Set #18: [18, 21, 22, 24, 25]
Set #19: [2, 5, 14, 16, 19]
Set #23: [2, 11, 18, 23, 29]
Set #24: [1, 8, 20, 28, 30]
Set #51: [26]

s-k-40-60

Found minimum set cover containing 14 sets in 0.000636 seconds.
Included sets: [2, 3, 4, 5, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Set #2: [6, 29, 35, 37, 39]
Set #3: [13, 18, 29, 32, 33]
Set #4: [9, 11, 15, 16, 24]
Set #5: [10, 14, 29, 30, 39]
Set #11: [8, 9, 32, 38, 40]
Set #12: [3, 13, 15, 25, 34]
Set #13: [9, 11, 17, 27, 29]
Set #14: [3, 5, 7, 12, 30]
Set #15: [13, 19, 34, 39, 40]
Set #16: [2, 9, 11, 26, 35]
Set #17: [1, 4, 13, 29, 40]
Set #18: [8, 20, 21, 31, 36]
Set #19: [2, 22, 23, 33, 39]
Set #20: [20, 22, 28, 32, 36]

s-rg-109-35

Found minimum set cover containing 22 sets in 0.002017 seconds.
Included sets: [3, 4, 5, 6, 7, 11, 12, 14, 15, 16, 18, 19, 20, 22, 23, 24, 25, 26, 28, 29, 32, 35]
Set #3: [1, 7, 8, 9, 10, 11, 12, 13]
Set #4: [2, 4, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Set #5: [14, 25, 26, 27, 28, 29, 30, 31, 32]
Set #6: [33, 34, 35, 36]
Set #7: [37, 38, 39, 40]
Set #11: [5, 27, 46, 47, 48, 49, 50, 51]
Set #12: [7, 52, 53, 54, 55, 56, 57, 58]

Set #14: [46, 61, 62, 63, 64]
Set #15: [17, 43, 47, 65, 66, 67, 68, 69, 70]
Set #16: [59, 71, 72, 73]
Set #18: [34, 41, 74, 79, 80, 81]
Set #19: [18, 30, 45]
Set #20: [35, 38, 48, 66, 82, 83, 84, 85, 86, 87]
Set #22: [67, 75, 91]
Set #23: [20, 76, 92, 93]
Set #24: [36, 61, 77, 79, 83, 94, 95, 96]
Set #25: [8, 31, 52, 60, 72, 84, 92, 97, 98]
Set #26: [9, 21, 53, 62, 85, 99, 100, 101]
Set #28: [49, 54, 63, 73, 88, 103, 104, 105, 106]
Set #29: [11, 42, 55, 89, 97, 103, 107]
Set #32: [6, 12, 40, 44, 57, 80, 90, 93, 102, 108]
Set #35: [3, 13, 70, 78, 87, 106, 109]

s-rg-118-30

Found minimum set cover containing 20 sets in 0.002075 seconds.
Included sets: [1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 19, 24, 25, 26, 28]
Set #1: [1, 2, 3, 4, 5, 6, 7, 8]
Set #3: [1, 15, 16, 17, 18, 19, 20, 21]
Set #4: [22, 23, 24, 25, 26, 27, 28, 29]
Set #5: [2, 9, 15, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
Set #6: [30, 41, 42, 43, 44, 45, 46, 47]
Set #7: [3, 31, 48, 49, 50]
Set #8: [4, 10, 51, 52, 53, 54, 55, 56, 57, 58]
Set #9: [32, 59, 60, 61, 62, 63, 64, 65, 66]
Set #10: [33, 51, 59, 67, 68, 69, 70, 71]
Set #12: [23, 35, 72, 78, 79, 80]
Set #14: [24, 48, 62, 67, 73, 86, 87, 88, 89, 90]
Set #15: [5, 37, 42, 54, 63, 74, 81, 91, 92, 93]
Set #16: [11, 25, 43, 75, 82, 91, 94, 95, 96, 97]
Set #17: [68, 98, 99, 100, 101]
Set #18: [26, 38, 55, 69, 83, 94, 102, 103, 104]
Set #19: [17, 39, 44, 64, 84, 98, 105, 106, 107, 108]
Set #24: [12, 79, 99, 107, 109, 112, 113]
Set #25: [20, 27, 70, 85, 112, 114, 115, 116]
Set #26: [7, 13, 40, 50, 57, 76, 80, 100, 111, 114, 117, 118]
Set #28: [14, 21, 28, 77, 93, 108, 110, 113, 115]

s-rg-63-25

Found minimum set cover containing 16 sets in 0.000628 seconds.
Included sets: [3, 4, 6, 9, 10, 12, 13, 14, 15, 16, 17, 19, 20, 22, 23, 25]

Set #3: [1, 14, 15, 16, 17, 18]
Set #4: [6, 19, 20, 21, 22]
Set #6: [23, 27, 28, 29]
Set #9: [2, 7, 24, 37, 38, 39, 40, 41, 42]
Set #10: [8, 14, 20, 43, 44]
Set #12: [3, 33, 37, 45, 46]
Set #13: [4, 9, 15, 47]
Set #14: [10, 48, 49]
Set #15: [5, 16, 34, 38, 48]
Set #16: [30, 50, 51, 52, 53]
Set #17: [31, 35, 44, 47]
Set #19: [40, 54, 57, 58, 59]
Set #20: [22, 41, 55, 57, 60, 61, 62]
Set #22: [11, 25, 32, 51, 60, 63]
Set #23: [12, 42, 56, 61]
Set #25: [13, 18, 26, 36, 49, 53, 59, 63]

s-k-35-65

Found minimum set cover containing 10 sets in 0.007174 seconds.
Included sets: [13, 15, 16, 17, 21, 22, 23, 24, 27, 30]
Set #13: [1, 14, 20, 24, 31]
Set #15: [6, 18, 28, 29, 33]
Set #16: [5, 24, 25, 29, 35]
Set #17: [10, 11, 16, 30, 32]
Set #21: [2, 12, 19, 22, 35]
Set #22: [3, 7, 24, 26, 33]
Set #23: [9, 13, 18, 31, 34]
Set #24: [1, 7, 8, 15, 23]
Set #27: [4, 7, 12, 16, 27]
Set #30: [15, 17, 21, 22, 29]

s-rg-155-40

Found minimum set cover containing 27 sets in 0.008868 seconds.
Included sets: [3, 5, 6, 7, 9, 12, 13, 15, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 35, 36, 38, 39, 40]
Set #3: [13, 14, 15, 16, 17, 18, 19, 20]
Set #5: [1, 27, 28, 29, 30, 31, 32, 33, 34, 35]
Set #6: [8, 36, 37, 38, 39, 40, 41, 42, 43, 44]
Set #7: [21, 27, 45, 46, 47, 48, 49, 50, 51]
Set #9: [56, 57, 58, 59, 60, 61, 62]
Set #12: [2, 13, 37, 45, 75, 76, 77, 78, 79, 80]
Set #13: [38, 63, 75, 81, 82, 83, 84, 85, 86, 87]
Set #15: [29, 52, 64, 97, 98]

Set #17: [3, 39, 65, 77, 101, 102, 103, 104]
Set #18: [4, 9, 48, 105]
Set #19: [66, 88, 99, 106, 107, 108, 109, 110, 111]
Set #21: [5, 30, 49, 56, 71, 106, 114, 115, 116, 117, 118]
Set #22: [15, 53, 89, 107, 119, 120, 121, 122]
Set #23: [10, 31, 90, 101, 119, 123, 124, 125, 126]
Set #24: [16, 40, 72, 82, 120, 127, 128, 129, 130, 131, 132]
Set #25: [41, 57, 91, 98, 112, 127, 133, 134, 135]
Set #26: [92, 102, 114, 136, 137, 138]
Set #27: [17, 32, 58, 67, 78, 93, 103, 121, 123, 139, 140]
Set #29: [11, 22, 68, 84, 100, 108, 141, 143, 144]
Set #30: [12, 23, 104, 122, 128, 134, 139, 145]
Set #31: [6, 24, 69, 79, 85, 109, 129, 140, 143, 146, 147]
Set #32: [42, 54, 60, 70, 94, 124, 137, 148, 149]
Set #35: [50, 61, 73, 86, 125, 130, 147, 152]
Set #36: [43, 74, 95, 115, 145, 148, 153, 154]
Set #38: [19, 25, 33, 80, 87, 132, 142, 149, 150, 155]
Set #39: [7, 20, 26, 34, 44, 55, 96, 111, 117, 151, 155]
Set #40: [35, 113, 118, 126]

s-rg-197-45

Found minimum set cover containing 30 sets in 0.012716 seconds.
Included sets: [1, 2, 3, 6, 7, 8, 10, 11, 12, 13, 14, 16, 18, 20, 21, 22, 25, 28, 29, 30, 32, 35, 37, 38, 39, 40, 41, 42, 44, 45]
Set #1: [1, 2, 3, 4, 5, 6]
Set #2: [7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Set #3: [1, 7, 21, 22, 23, 24, 25, 26]
Set #6: [37, 38, 39, 40, 41, 42]
Set #7: [21, 27, 43, 44, 45, 46, 47, 48, 49, 50]
Set #8: [51, 52, 53, 54, 55, 56, 57]
Set #10: [2, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73]
Set #11: [74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84]
Set #12: [9, 37, 74, 85, 86, 87, 88, 89, 90, 91, 92, 93]
Set #13: [10, 22, 28, 85, 94, 95, 96, 97, 98, 99]
Set #14: [32, 100, 101, 102, 103, 104, 105]
Set #16: [33, 58, 65, 86, 111, 112]
Set #18: [44, 94, 113, 117, 118, 119, 120, 121, 122, 123]
Set #20: [23, 88, 95, 124, 128, 129, 130, 131, 132, 133, 134, 135]
Set #21: [13, 52, 59, 114, 136, 137, 138, 139]
Set #22: [14, 29, 34, 39, 66, 75, 89, 100, 106, 140, 141, 142, 143, 144, 145, 146, 147, 148]
Set #25: [16, 30, 96, 101, 149, 159, 160, 161, 162, 163]
Set #28: [17, 25, 41, 46, 60, 125, 130, 155, 164, 172, 173]
Set #29: [47, 67, 78, 115, 150, 174, 175, 176]
Set #30: [31, 68, 98, 107, 131, 151, 167, 177, 178, 179]
Set #32: [92, 99, 116, 118, 156, 159, 165, 168, 180]

Set #35: [81, 119, 169, 172, 177, 181]
Set #37: [50, 102, 108, 126, 143, 178, 182]
Set #38: [18, 109, 121, 144, 152, 161, 188]
Set #39: [19, 42, 93, 103, 112, 145, 153, 157, 162, 173, 175, 179, 188, 189]
Set #40: [20, 35, 133, 146, 170, 190, 191]
Set #41: [36, 56, 61, 82, 127, 184, 189, 192, 193, 194]
Set #42: [71, 83, 104, 147, 158, 176, 185, 187, 190, 195, 196]
Set #44: [6, 62, 73, 123, 139, 154, 166, 193, 197]
Set #45: [57, 63, 110, 135, 148, 171, 183, 186, 191, 194, 196, 197]

s-rg-245-50

Found minimum set cover containing 35 sets in 0.040700 seconds.
Included sets: [1, 2, 4, 5, 7, 10, 11, 13, 14, 15, 18, 19, 20, 21, 23, 25, 26, 27, 29, 31, 32, 33, 34, 35, 38, 39, 40, 41, 42, 43, 45, 4
6, 47, 49, 50]
Set #1: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Set #2: [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]
Set #4: [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]
Set #5: [1, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52]
Set #7: [58, 59, 60, 61, 62, 63, 64]
Set #10: [22, 33, 78, 79, 80, 81]
Set #11: [82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92]
Set #13: [65, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107]
Set #14: [23, 53, 58, 82, 97, 108, 109, 110, 111, 112, 113]
Set #15: [59, 71, 114, 115, 116, 117, 118, 119, 120, 121, 122]
Set #18: [34, 45, 66, 98, 123, 137, 138, 139, 140, 141, 142]
Set #19: [84, 143, 144, 145, 146, 147, 148]
Set #20: [54, 60, 67, 115, 130, 149, 150, 151, 152, 153]
Set #21: [72, 93, 116, 124, 131, 154, 155, 156, 157, 158]
Set #23: [47, 73, 117, 159, 163, 164, 165, 166, 167]
Set #25: [12, 62, 74, 94, 99, 110, 173, 174, 175, 176, 177, 178]
Set #26: [13, 35, 118, 149, 173, 179, 180, 181, 182, 183, 184, 185]
Set #27: [48, 79, 125, 132, 137, 143, 160, 163, 179, 186, 187, 188]
Set #29: [14, 24, 101, 111, 168, 194, 195, 196]
Set #31: [85, 133, 164, 189, 201, 202, 203]
Set #32: [7, 25, 134, 138, 175, 181, 204, 205, 206, 207]
Set #33: [37, 169, 176, 190, 194, 208, 209, 210]
Set #34: [49, 86, 126, 161, 177, 182, 208, 211, 212, 213]
Set #35: [15, 63, 95, 154, 165, 197, 204, 214, 215, 216, 217]
Set #38: [17, 55, 68, 75, 89, 146, 151, 178, 183, 191, 201, 212, 224, 225, 226]
Set #39: [9, 26, 64, 90, 112, 156, 170, 202, 227, 228]
Set #40: [18, 27, 127, 171, 215, 218, 222, 229, 230, 231]
Set #41: [28, 76, 91, 139, 157, 166, 195, 213, 216, 219, 227, 232]
Set #42: [29, 69, 96, 140, 158, 220, 233, 234]
Set #43: [19, 56, 92, 104, 120, 141, 198, 205, 224, 228, 233, 235, 236, 237]

Set #45: [20, 38, 77, 121, 162, 187, 203, 206, 210, 230, 238, 239, 240]
Set #46: [39, 52, 57, 128, 135, 184, 192, 236, 239, 241, 242]
Set #47: [30, 221, 223, 241, 243]
Set #49: [107, 129, 148, 153, 167, 188, 193, 196, 199, 217, 226, 243, 244]
Set #50: [41, 70, 122, 136, 172, 200, 231, 232, 237, 240, 242, 245]

s-k-40-80

Found minimum set cover containing 9 sets in 0.185711 seconds.
Included sets: [8, 13, 21, 29, 30, 31, 33, 36, 40]
Set #8: [8, 9, 17, 18, 37, 40]
Set #13: [3, 4, 11, 23, 29, 34]
Set #21: [8, 10, 12, 19, 27, 36]
Set #29: [14, 16, 21, 31, 32, 35]
Set #30: [1, 15, 21, 30, 33, 40]
Set #31: [5, 6, 13, 21, 25, 34]
Set #33: [2, 7, 13, 20, 22, 28]
Set #36: [3, 8, 18, 22, 24, 38]
Set #40: [10, 19, 25, 26, 34, 39]

s-k-50-95

Found minimum set cover containing 12 sets in 0.325660 seconds.
Included sets: [1, 3, 19, 21, 27, 32, 35, 36, 38, 39, 44, 56]
Set #1: [2, 13, 15, 18, 37, 50]
Set #3: [7, 21, 23, 28, 35, 50]
Set #19: [7, 42, 43, 44, 47, 49]
Set #21: [9, 13, 24, 26, 32, 45]
Set #27: [8, 33, 35, 43, 47, 48]
Set #32: [1, 6, 16, 32, 36, 50]
Set #35: [3, 4, 14, 18, 26, 39]
Set #36: [5, 17, 22, 27, 31, 39]
Set #38: [18, 20, 29, 30, 32, 40]
Set #39: [10, 12, 31, 34, 38, 46]
Set #44: [6, 7, 19, 20, 25, 41]
Set #56: [11]

s-rg-413-75

Found minimum set cover containing 52 sets in 0.614918 seconds.
Included sets: [1, 3, 4, 5, 8, 9, 11, 12, 13, 15, 17, 18, 21, 23, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 38, 40, 41, 43, 44, 46, 47, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 66, 67, 68, 69, 71, 73]

Set #1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
Set #3: [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
Set #4: [1, 12, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56]
Set #5: [24, 57, 58, 59, 60, 61, 62, 63, 64]
Set #8: [25, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92]
Set #9: [41, 72, 93, 94, 95, 96, 97, 98, 99, 100]
Set #11: [2, 57, 101, 109, 110, 111, 112]
Set #12: [3, 27, 102, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122]
Set #13: [73, 123, 124, 125, 126, 127, 128]
Set #15: [4, 13, 42, 82, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152]
Set #17: [29, 74, 153, 162, 163, 164, 165, 166, 167, 168]
Set #18: [14, 30, 95, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178]
Set #21: [142, 179, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209]
Set #23: [15, 47, 103, 129, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223]
Set #25: [65, 154, 228, 229, 230, 231, 232, 233]
Set #26: [31, 84, 104, 196, 213, 228, 234, 235, 236, 237, 238, 239, 240]
Set #27: [16, 143, 169, 214, 241, 242, 243, 244, 245, 246, 247]
Set #28: [66, 125, 144, 187, 234, 241, 248, 249, 250, 251, 252, 253, 254, 255]
Set #29: [58, 67, 85, 97, 188, 210, 248, 256, 257, 258, 259, 260, 261, 262, 263, 264]
Set #30: [126, 170, 180, 215, 265, 266, 267, 268, 269, 270, 271, 272]
Set #31: [130, 224, 256, 273, 274, 275, 276, 277, 278, 279, 280, 281]
Set #32: [86, 155, 163, 211, 282, 283, 284, 285, 286, 287, 288, 289]
Set #33: [17, 48, 156, 197, 257, 290, 291, 292, 293]
Set #35: [18, 198, 299, 300, 301, 302, 303, 304, 305]
Set #36: [59, 68, 131, 172, 189, 225, 290, 306, 307, 308, 309, 310]
Set #38: [132, 157, 173, 190, 216, 299, 316, 317, 318, 319]
Set #40: [32, 181, 191, 229, 311, 325, 326, 327]
Set #41: [19, 49, 133, 174, 199, 217, 242, 266, 328, 329, 330, 331, 332, 333]
Set #43: [20, 320, 334, 338, 339, 340, 341, 342, 343, 344]
Set #44: [75, 201, 218, 275, 329, 345, 346, 347, 348, 349]
Set #46: [6, 105, 134, 158, 202, 276, 294, 307, 339, 350, 354, 355, 356]
Set #47: [21, 60, 76, 110, 146, 165, 268, 295, 302, 321, 325, 340, 357, 358, 359, 360, 361, 362]
Set #49: [52, 135, 230, 243, 250, 261, 291, 304, 312, 322, 370, 371, 372, 373, 374]
Set #50: [53, 114, 147, 182, 192, 204, 231, 251, 284, 296, 357, 370, 375, 376, 377]
Set #51: [7, 77, 166, 212, 232, 316, 354, 371, 378, 379, 380]
Set #52: [78, 111, 148, 236, 285, 297, 330, 342, 381, 382]
Set #53: [34, 106, 115, 149, 167, 193, 252, 278, 363, 375, 383]
Set #54: [8, 98, 136, 159, 183, 313, 317, 323, 331, 384, 385]
Set #56: [35, 69, 116, 137, 205, 220, 305, 308, 345, 378, 387]
Set #57: [61, 70, 117, 286, 351, 383, 386, 388, 389, 390, 391]
Set #58: [36, 79, 118, 151, 233, 237, 245, 279, 392, 393, 394, 395]
Set #59: [89, 175, 184, 194, 226, 238, 364, 376, 396, 397, 398]
Set #60: [22, 90, 99, 112, 138, 262, 298, 346, 352, 365, 384, 399, 400]
Set #61: [71, 287, 314, 366, 379, 392, 401, 402, 403]
Set #62: [54, 263, 335, 347, 372, 387, 404]
Set #63: [80, 100, 119, 139, 206, 264, 343, 388, 393, 396, 405]
Set #66: [11, 55, 62, 240, 246, 270, 289, 324, 336, 373, 381, 390, 406, 409, 410]

Set #67: [37, 107, 253, 292, 337, 344, 400, 401, 409, 411]
Set #68: [23, 108, 121, 168, 185, 222, 254, 326, 367, 402]
Set #69: [81, 122, 207, 271, 280, 353, 360, 368, 408, 412]
Set #71: [92, 160, 186, 223, 255, 310, 319, 410, 412, 413]
Set #73: [161, 208, 227, 315, 327, 332, 349, 369, 382, 397, 407]

s-k-150-225

Found minimum set cover containing 36 sets in 1.157804 seconds.
Included sets: [2, 4, 5, 7, 12, 15, 18, 20, 23, 25, 31, 34, 35, 36, 37, 40, 41, 47, 53, 57, 58, 59, 62, 63, 68, 69, 70, 71, 72, 73, 75, 91, 134, 144, 183, 187]
Set #2: [34, 46, 75, 76, 81, 94, 125, 131]
Set #4: [2, 18, 38, 40, 45, 47, 76, 105]
Set #5: [33, 52, 67, 86, 89, 99, 115, 131]
Set #7: [3, 9, 93, 106, 110, 123, 127, 148]
Set #12: [50, 51, 82, 85, 91, 103, 130, 142]
Set #15: [4, 54, 60, 71, 102, 114, 116, 123]
Set #18: [6, 55, 92, 100, 104, 117, 125, 139]
Set #20: [11, 36, 48, 58, 61, 87, 90, 92]
Set #23: [23, 31, 41, 49, 62, 84, 97, 107]
Set #25: [1, 14, 29, 39, 71, 96, 136, 145]
Set #31: [5, 22, 35, 41, 54, 63, 105, 126]
Set #34: [20, 29, 54, 77, 82, 88, 105, 124]
Set #35: [31, 85, 91, 93, 101, 102, 138, 143]
Set #36: [2, 83, 90, 103, 122, 134, 141, 149]
Set #37: [1, 7, 36, 79, 82, 86, 91, 106]
Set #40: [10, 37, 75, 84, 93, 106, 139, 147]
Set #41: [6, 29, 49, 53, 66, 67, 94, 140]
Set #47: [2, 20, 21, 22, 24, 70, 78, 90]
Set #53: [21, 22, 28, 47, 77, 128, 129, 136]
Set #57: [4, 15, 21, 60, 77, 106, 109, 146]
Set #58: [33, 34, 45, 55, 73, 87, 134, 135]
Set #59: [3, 13, 30, 51, 56, 57, 80, 113]
Set #62: [8, 12, 32, 62, 63, 106, 132, 134]
Set #63: [25, 27, 43, 47, 71, 110, 120, 136]
Set #68: [17, 19, 22, 90, 91, 93, 106, 130]
Set #69: [26, 33, 72, 80, 93, 110, 133, 139]
Set #70: [4, 44, 71, 91, 118, 121, 137, 138]
Set #71: [13, 42, 56, 65, 74, 95, 109, 111]
Set #72: [30, 53, 60, 64, 85, 100, 144, 150]
Set #73: [45, 51, 68, 76, 85, 98, 105, 127]
Set #75: [18, 29, 87, 100, 110, 119, 122, 126]
Set #91: [16]
Set #134: [59]
Set #144: [69]
Set #183: [108]

Set #187: [112]

s-k-50-100

Found minimum set cover containing 9 sets in 4.041318 seconds.
Included sets: [1, 5, 9, 10, 11, 25, 31, 33, 37]
Set #1: [13, 16, 19, 25, 26, 30, 46]
Set #5: [4, 8, 15, 29, 31, 44, 45]
Set #9: [3, 10, 15, 17, 38, 47, 49]
Set #10: [9, 13, 14, 21, 24, 27, 41]
Set #11: [2, 10, 12, 21, 22, 35, 37]
Set #25: [6, 14, 17, 21, 23, 33, 43]
Set #31: [5, 11, 36, 38, 40, 42, 48]
Set #33: [1, 16, 18, 19, 27, 28, 39]
Set #37: [7, 9, 20, 32, 34, 37, 50]

s-rg-733-100

Found minimum set cover containing 76 sets in 20.496086 seconds.
Included sets: [1, 2, 3, 4, 7, 8, 9, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 35, 36, 37, 38, 39, 40, 42, 43, 44, 45, 46, 48, 51, 52, 54, 57, 58, 60, 61, 62, 63, 64, 65, 66, 67, 68, 70, 72, 73, 75, 76, 78, 79, 80, 83, 84, 85, 86, 87, 88, 89, 90, 91, 93, 94, 95, 96, 97, 98, 99]
Set #1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
Set #2: [14, 15, 16, 17, 18, 19, 20, 21, 22]
Set #3: [23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]
Set #4: [40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55]
Set #7: [2, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89]
Set #8: [3, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101]
Set #9: [102, 103, 104, 105, 106, 107, 108, 109, 110]
Set #11: [4, 102, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142]
Set #12: [143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154]
Set #13: [90, 111, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169]
Set #15: [14, 65, 112, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195 196, 197, 198, 199, 200, 201, 202]
Set #16: [203, 204, 205, 206, 207, 208, 209, 210, 211]
Set #17: [23, 41, 113, 203, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225 226, 227, 228]
Set #18: [56, 181, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241]
Set #19: [24, 114, 182, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254]
Set #20: [42, 155, 170, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264]
Set #21: [43, 66, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277]
Set #22: [204, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289]
Set #24: [57, 115, 156, 171, 183, 278, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 30 310]
Set #25: [25, 58, 116, 172, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321]

Set #26: [26, 44, 59, 255, 265, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334]
Set #27: [6, 67, 173, 230, 242, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345]
Set #28: [15, 157, 231, 266, 279, 311, 346, 347, 348, 349, 350, 351, 352, 353, 354]
Set #29: [117, 143, 184, 205, 346, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368]
Set #30: [243, 267, 290, 355, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378]
Set #31: [27, 185, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391]
Set #35: [68, 92, 215, 280, 417, 418, 419, 420, 421, 422, 423]
Set #36: [28, 69, 129, 186, 256, 268, 335, 424, 425, 426, 427]
Set #37: [216, 348, 428, 429, 430, 431, 432, 433, 434, 435, 436]
Set #38: [70, 232, 269, 281, 400, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449]
Set #39: [7, 187, 217, 244, 299, 312, 336, 450, 451, 452, 453, 454, 455, 456, 457, 458]
Set #40: [118, 159, 300, 337, 357, 459, 460, 461, 462, 463, 464, 465, 466]
Set #42: [17, 119, 219, 323, 460, 478, 479, 480, 481, 482, 483, 484, 485]
Set #43: [120, 130, 188, 291, 301, 314, 428, 467, 486, 487, 488, 489, 490, 491, 492]
Set #44: [29, 174, 220, 358, 429, 437, 493, 494, 495, 496, 497, 498, 499, 500]
Set #45: [8, 30, 81, 104, 160, 221, 233, 292, 338, 359, 370, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515]
Set #46: [245, 257, 339, 392, 478, 501, 516, 517, 518, 519, 520]
Set #48: [9, 31, 175, 246, 258, 302, 371, 417, 452, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538]
Set #51: [82, 425, 468, 539, 556, 557, 558, 559, 560, 561, 562]
Set #52: [46, 94, 282, 393, 528, 563, 564, 565, 566, 567, 568, 569, 570, 571]
Set #54: [33, 84, 146, 315, 326, 401, 469, 479, 504, 540, 579, 580, 581, 582, 583]
Set #57: [34, 60, 96, 247, 273, 418, 431, 462, 521, 541, 572, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606]
Set #58: [47, 71, 147, 191, 316, 381, 463, 522, 565, 584, 592, 607, 608, 609, 610, 611, 612]
Set #60: [19, 105, 304, 432, 585, 593, 607, 617, 618, 619, 620, 621, 622]
Set #61: [148, 193, 260, 293, 394, 420, 613, 623]
Set #62: [97, 106, 131, 162, 176, 194, 395, 410, 453, 556, 573, 624, 625, 626, 627, 628, 629, 63
631, 632]
Set #63: [48, 61, 163, 177, 195, 284, 340, 440, 507, 531, 633, 634, 635]
Set #64: [261, 294, 305, 341, 350, 382, 470, 493, 542, 550, 557, 597, 624, 636, 637, 638, 639, 640, 641]
Set #65: [49, 98, 396, 402, 411, 441, 464, 481, 598, 642, 643, 644, 645, 646, 647]
Set #66: [50, 196, 383, 403, 532, 551, 586, 642, 648, 649, 650, 651, 652]
Set #67: [72, 99, 328, 508, 533, 558, 648, 653, 654, 655]
Set #68: [62, 132, 164, 285, 442, 517, 574, 587, 599, 656, 657, 658, 659, 660, 661]
Set #70: [121, 133, 197, 236, 249, 329, 351, 404, 443, 454, 471, 482, 579, 636, 666, 667, 668, 669]
Set #72: [149, 224, 262, 274, 342, 361, 397, 472, 489, 534, 625, 649, 656, 673, 674, 675]
Set #73: [20, 209, 263, 286, 295, 362, 405, 495, 510, 523, 552, 618, 657, 676, 677]
Set #75: [352, 373, 406, 444, 497, 626, 658, 670, 678, 681, 682, 683, 684, 685, 686, 687, 688]
Set #76: [11, 21, 331, 353, 363, 374, 385, 398, 455, 518, 567, 667, 673, 679, 689, 690, 691, 692
Set #78: [122, 225, 296, 387, 446, 456, 634, 659, 690, 695, 696, 697, 698]
Set #79: [251, 308, 412, 447, 473, 519, 535, 560, 575, 581, 644, 662, 674, 699, 700]

Set #80: [73, 86, 210, 237, 297, 354, 421, 484, 543, 568, 582, 614, 645, 676, 681, 701, 702, 703
704, 705]
Set #83: [108, 136, 178, 407, 457, 603, 627, 692, 701, 712, 713, 714, 715]
Set #84: [12, 52, 123, 137, 376, 389, 408, 413, 512, 544, 553, 604, 646, 671, 683, 716]
Set #85: [151, 167, 226, 343, 365, 409, 490, 524, 561, 594, 628, 695, 706]
Set #86: [22, 53, 152, 168, 333, 448, 536, 684, 707, 710, 717, 718, 719]
Set #87: [88, 179, 276, 426, 498, 595, 605, 623, 660, 712, 720]
Set #88: [63, 74, 124, 153, 240, 366, 414, 422, 545, 554, 588, 639, 696, 702, 721]
Set #89: [101, 138, 252, 367, 390, 525, 537, 546, 589, 609, 663, 672, 675, 685, 699, 708, 713,
722, 723]
Set #90: [54, 89, 139, 241, 377, 474, 576, 590, 651, 654, 693, 703, 717]
Set #91: [125, 227, 423, 475, 491, 526, 555, 610, 615, 620, 629, 664, 677, 704, 724, 725, 726]
Set #93: [38, 75, 368, 465, 538, 547, 661, 700, 709, 715, 716, 725, 727, 728]
Set #94: [76, 126, 200, 253, 320, 344, 399, 435, 476, 513, 635, 718, 721, 729]
Set #95: [64, 140, 154, 180, 228, 527, 548, 611, 640, 652, 680, 687]
Set #96: [39, 127, 141, 309, 415, 449, 499, 569, 577, 612, 668, 697, 705, 723, 727, 730, 731]
Set #97: [77, 289, 378, 416, 477, 578, 621, 630, 665, 726, 732]
Set #98: [110, 169, 201, 334, 458, 500, 514, 549, 570, 591, 596, 622, 631, 647, 655, 698, 729,
730, 732]
Set #99: [78, 202, 254, 310, 436, 492, 562, 606, 616, 669, 694, 711, 728, 733]

s-k-200-300

Found minimum set cover containing 35 sets in 82.366257 seconds.
Included sets: [3, 5, 6, 7, 15, 21, 24, 28, 29, 30, 32, 33, 34, 39, 40, 41, 42, 50, 51, 52, 53, 54, 55,
56, 61, 63, 64, 66, 74, 76, 77, 89, 90, 100, 204]
Set #3: [23, 31, 63, 97, 102, 103, 107, 167, 175, 195]
Set #5: [34, 76, 108, 116, 124, 127, 149, 151, 173, 196]
Set #6: [30, 55, 61, 66, 73, 110, 117, 161, 185, 198]
Set #7: [22, 39, 41, 42, 46, 58, 167, 177, 181, 196]
Set #15: [5, 7, 8, 84, 88, 128, 136, 138, 177, 193]
Set #21: [17, 48, 70, 87, 93, 97, 118, 156, 158, 192]
Set #24: [8, 28, 64, 72, 89, 108, 171, 174, 183, 190]
Set #28: [17, 25, 53, 67, 105, 107, 108, 134, 151, 185]
Set #29: [21, 47, 91, 106, 111, 117, 122, 136, 172, 195]
Set #30: [10, 33, 56, 97, 98, 117, 120, 137, 140, 168]
Set #32: [27, 52, 55, 68, 80, 88, 95, 115, 118, 149]
Set #33: [16, 25, 34, 63, 87, 90, 100, 126, 128, 191]
Set #34: [5, 9, 26, 27, 80, 123, 139, 143, 150, 181]
Set #39: [27, 56, 65, 66, 125, 145, 152, 156, 162, 199]
Set #40: [12, 15, 22, 25, 29, 50, 161, 164, 178, 182]
Set #41: [20, 30, 35, 49, 56, 79, 82, 105, 144, 194]
Set #42: [6, 19, 54, 64, 85, 89, 118, 137, 179, 181]
Set #50: [44, 86, 110, 126, 131, 150, 153, 187, 191, 197]
Set #51: [2, 11, 44, 48, 60, 92, 121, 122, 130, 179]
Set #52: [2, 3, 39, 46, 57, 69, 78, 85, 132, 155]

```
Set #53: [13, 45, 65, 71, 87, 115, 127, 131, 151, 193]
Set #54: [1, 14, 21, 28, 35, 59, 72, 105, 116, 155]
Set #55: [5, 22, 37, 62, 94, 118, 127, 142, 156, 170]
Set #56: [32, 38, 44, 50, 62, 79, 103, 129, 169, 173]
Set #61: [4, 7, 40, 43, 79, 86, 121, 147, 148, 189]
Set #63: [46, 47, 56, 58, 83, 96, 109, 164, 178, 188]
Set #64: [10, 15, 26, 64, 87, 101, 114, 159, 186, 196]
Set #66: [39, 56, 99, 113, 114, 146, 157, 158, 166, 200]
Set #74: [6, 17, 24, 50, 51, 77, 86, 164, 166, 199]
Set #76: [18, 71, 75, 108, 112, 115, 160, 165, 178, 184]
Set #77: [12, 48, 64, 74, 76, 81, 113, 155, 163, 196]
Set #89: [18, 37, 46, 98, 100, 109, 119, 135, 164, 193]
Set #90: [50, 105, 126, 130, 141, 144, 150, 154, 180, 194]
Set #100: [18, 36, 63, 77, 91, 103, 133, 150, 176, 192]
Set #204: [104]
```

s-k-100-175

```
Found minimum set cover containing 19 sets in 386.585480 seconds.
Included sets: [3, 16, 24, 26, 29, 31, 32, 39, 44, 46, 48, 53, 56, 61, 66, 68, 71, 74, 94]
Set #3: [20, 22, 37, 43, 66, 84, 90, 98]
Set #16: [6, 11, 16, 35, 48, 53, 93, 96]
Set #24: [25, 50, 51, 52, 54, 65, 89, 97]
Set #26: [2, 17, 38, 55, 68, 72, 90, 93]
Set #29: [8, 22, 23, 25, 40, 46, 85, 92]
Set #31: [12, 16, 28, 45, 58, 61, 79, 91]
Set #32: [7, 8, 12, 14, 21, 42, 45, 87]
Set #39: [5, 7, 21, 26, 31, 75, 80, 88]
Set #44: [3, 4, 18, 31, 71, 74, 76, 99]
Set #46: [9, 14, 18, 57, 61, 73, 77, 83]
Set #48: [34, 39, 62, 64, 65, 68, 81, 93]
Set #53: [5, 32, 47, 67, 68, 78, 90, 100]
Set #56: [4, 15, 27, 28, 41, 49, 56, 94]
Set #61: [29, 31, 32, 33, 49, 60, 86, 92]
Set #66: [4, 8, 11, 44, 63, 70, 82, 88]
Set #68: [7, 24, 33, 50, 51, 69, 80, 91]
Set #71: [1, 10, 15, 32, 36, 65, 80, 95]
Set #74: [13, 22, 30, 31, 32, 50, 59, 75]
Set #94: [19]
```

s-k-150-250

The program takes too long.