Notebook

ggT:

Euklidischer Algorithmus

```
private long smallggT(long a, long b){
        if(b == 0)
            return a;
        else
            return smallggT(b, a%b);
}
```

Steinscher Algorithmus

```
private long smallStein(long a, long b){
        int k = 0;
        long t = 0;

        while(a%2==0 && b%2==0){
            a /= 2;
            b /= 2;
            k += 1;
        }

        if(a%2!=0)
            t = -b;
        else
            t = a;

        while(t != 0){
            while(t%2==0)
                    t /= 2;
            if(t>0)
                    a = t;
            else
                    b = -t;
            t = a - b;
        }

        return a*(long)Math.pow(2, k);
}
```

Sortieren:

Quicksort (2D-Array)

```
public static void sort(int[][] a, int start, int end) {
        int[] tmp;
        int i = start;
        int j = end;
        int x = a[(start + end) / 2][0];

        do {
          // Vom Startpunkt aus einen Wert suchen der >= x ist
          while (a[i][0] < x) {
                i++;
          }
          // Vom Endpunkt aus einen Wert suchen der <= x ist
          while (a[j][0] > x) {
                j--;
          }

          // a[i] und a[j] tauschen
          if (i <= j) {
                tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
                i++;
                j--;
          }
        } while (i <= j);

        if (start < j) {
          sort(a, start, j);
        }

        if (i < end) {
          sort(a, i, end);
        }
}
```

Dynamische Programmierung:

Counting Change

```
int[] coin = new int[]{50, 25, 10, 5, 1};                    // Verschiedene Beträge
// Betrag als Integerwert in cent
int amount;
// Kombinationsmöglichkeiten für die verschiedenen Beträge
long[] nway = new long[30001];
nway[0] = 1;
// Sämtliche Kombinationen für alle Beträge von 0 cent bis 30000 cent berechnen
for(i=0; i<coin.length; i++){
        c = coin[i];
        for(j = c; j<=30000; j++)
                nway[j] += nway[j-c];
        }
```

Graphen:

Prim

```
for (i = 0; i < roads; i++) {
        coordinates[i][3] = i - 1;
        coordinates[i][4] = i + 1;
}
head = 0;
nodes[0] = true;

// Alle Knoten durchlaufen
for (j = junctions - 1; j > 0; j--) {
        i = head;
        while (i < roads) {
           // Steht weder der Anfangs- noch der Endknoten der Straße auf true wird mit dem nächsten Wert weitergemacht
           if (!nodes[coordinates[i][0]] && !nodes[coordinates[i][1]]) {
                i = coordinates[i][4];
           } // Steht mindestens einer der Beiden Knoten der aktuellen Straße auf true
           else {
                // Ist der Vorgängerwert -1, bekommt die nachfolgende Straße auch -1 als Vorgängerwert
                if (coordinates[i][3] == -1) {
                   coordinates[coordinates[i][4]][3] = coordinates[i][3];
                } // Ansonsten bekommt die vorherige Straße den nachfolgerwert als nächste Straße
                // und doie nachfolgende Straße die Vorgängerstraße als vorherige Straße
                else {
                   coordinates[coordinates[i][3]][4] = coordinates[i][4];
                   coordinates[coordinates[i][4]][3] = coordinates[i][3];
                }
                // Ist nur einer der beiden Knotenpunke der Straße auf true wird der andere Knotenpunkt auch auf true gesetzt und die Länge
                // der Straße von der Gesamtlänge des Straßennetzes abgezogen, d.h. die Straße kann ohne Beleuchtung auskommen.
                if ((nodes[coordinates[i][0]] && !nodes[coordinates[i][1]]) || (nodes[coordinates[i][1]] && !nodes[coordinates[i][0]])) {
                   nodes[nodes[coordinates[i][0]] ? coordinates[i][1] : coordinates[i][0]] = true;
                   length -= coordinates[i][2];
                   // Ist der Vorgängerwert -1 wird der Nachfolgerwert als Startpunkt für die while-Schleife gesetzt, ansonsten wird mit
                   // dem alten Startwert begonnen
                   if (coordinates[i][3] == -1) {
                        head = coordinates[i][4];
                   }
                   // Schleife unterbrechen
                   break;
                }
                i = coordinates[i][4];
           }
        }
}
```

Kruskal

```
// Summe der Abstände
sum = 0;

// Array mit den Abständen durchlaufen
for(int i=0; i<node.length; i++){
        // Sind die Werte an den beiden Positionen unterschiedlich wird
        // der erste Wert gesucht und durch den zweiten ersetzt sowie der Abstand zur Summe addiert,
        // bis nur noch immer der selbe Wert in coordinates[][2] steht, dann wurde
        // der kürzeste Weg gefunden.
        if(coordinates[(int)node[i][0]][2]!=coordinates[(int)node[i][1]][2]){
                temp=coordinates[(int)node[i][0]][2];
                // Array mit den Koordinaten durchlaufen
                for(int j=0; j<coordinates.length ;j++)
                        if(coordinates[j][2]==temp)
                                coordinates[j][2]=coordinates[(int)node[i][1]][2];
                        // Wert dazuaddieren
                        sum+=Math.sqrt(node[i][2]);
        }
}
```

DFS

```
ways = new int[aliens][aliens];

for (i = 0; i < aliens; i++) {

        for (j = 0; j < y; j++) {
           for (k = 0; k < x; k++) {
                field2[j][k] = max;
           }
        }

        begin = end = 0;
        queue = new int[max][3];
        queue[0][0] = coordinates[i][0];
        queue[0][1] = coordinates[i][1];
        queue[0][2] = 0;
        end++;

        field2[queue[0][0]][queue[0][1]] = 0;

        while (end > begin) {
           int r = queue[begin][0];
           int c = queue[begin][1];
           int cost = queue[begin++][2] + 1;

           if (field[r][c] == 'S' || field[r][c] == 'A') {
                for (j = 0; j < aliens; j++) {
                   if (coordinates[j][0] == r && coordinates[j][1] == c) {
                        ways[i][j] = cost - 1;
                   }
                }
           }

           if (field2[r + 1][c] > cost && field[r + 1][c] != '#') {
                field2[r + 1][c] = cost;
                queue[end][0] = r+1;
                queue[end][1] = c;
                queue[end++][2] = cost;
           }
           if (field2[r - 1][c] > cost && field[r - 1][c] != '#') {
                field2[r - 1][c] = cost;
                queue[end][0] = r-1;
                queue[end][1] = c;
                queue[end++][2] =cost;
           }
           if (field2[r][c + 1] > cost && field[r][c + 1] != '#') {
                field2[r][c + 1] = cost;
                queue[end][0] = r;
                queue[end][1] = c +1;
                queue[end++][2] = cost;
           }
           if (field2[r][c - 1] > cost && field[r][c - 1] != '#') {
                field2[r][c - 1] = cost;
                queue[end][0] = r;
                queue[end][1] = c-1;
                queue[end++][2] = cost;
           }

        }


}
```