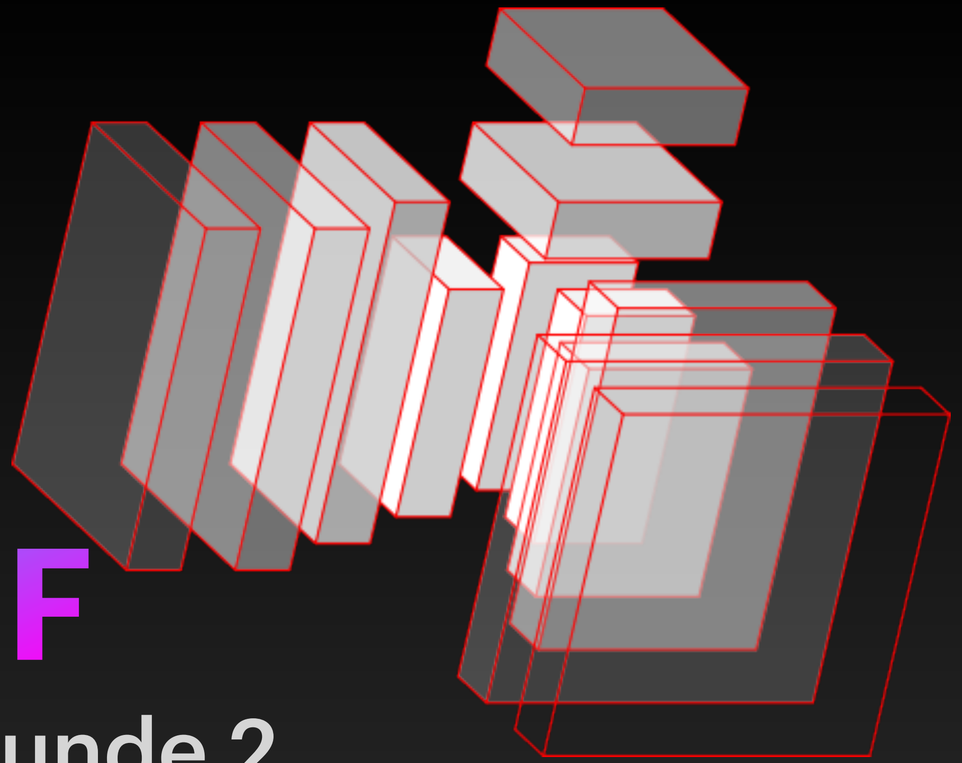
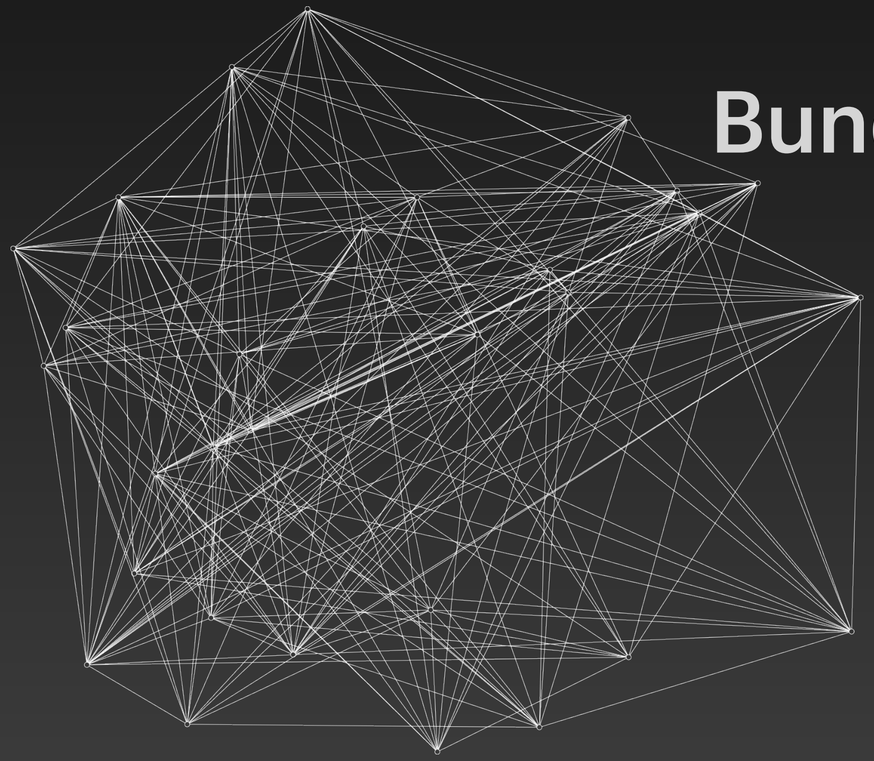


BLL BWINF

Bundeswettbewerb Runde 2



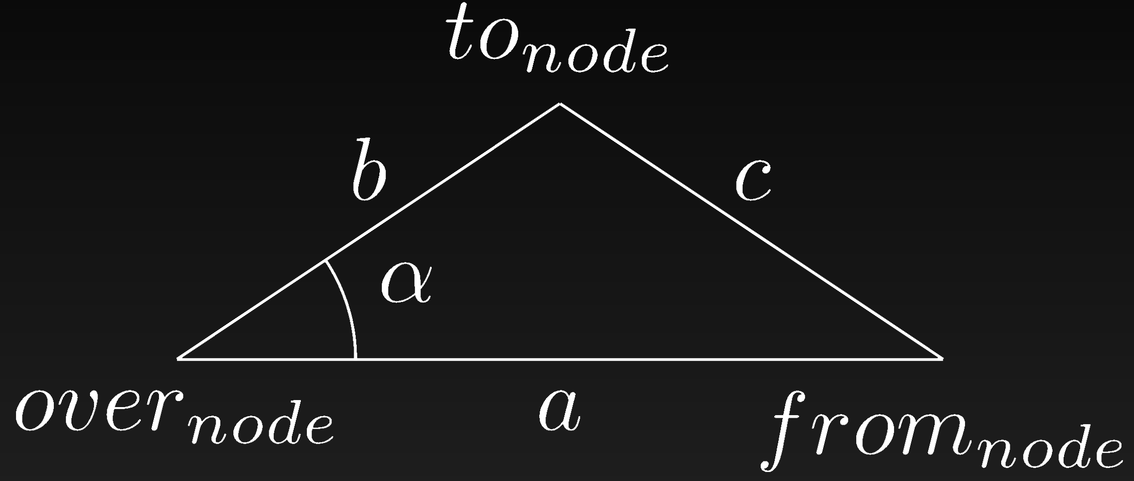
Lennart Protte 26.05.23

Weniger krumme Touren

Aufgabe 1

Lösungsansatz
Verfahren & Algorithmen

- Greedy-Algorithms
- Hamiltionwegproblem
- Nearest-Neighbour-Heuristik
- koordinaten$[i] \notin$ route $\wedge$ ($|$route$| < 2 \vee$ winkel ≥ 90)



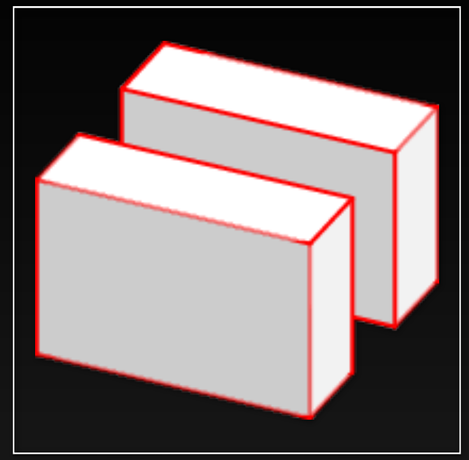
```cpp
bool solve(vector<pair<double, double> > &route, vector<pair<double, double> > &coordinates) {
    if (!route.empty()) { //Wenn die Route nicht leer ist, sortiere nach dem nächsten Knoten
        const auto &last : const pair<...> & = route.back();
        sort( first: coordinates.begin(), last: coordinates.end(),
              comp: [last](const auto &lhs : const pair<...> &, const auto &rhs : const pair<...> & ) -> bool {
                  return sqrt(pow((last.first - lhs.first), 2.0) + (pow((last.second - lhs.second), 2.0)))
                         < sqrt(pow((last.first - rhs.first), 2.0) + (pow((last.second - rhs.second), 2.0)));
              });
    }
    for (int i = 0; i < coordinates.size(); i++) { //Für jeden Knoten
        //Wenn dieser Knoten bereits in der Lösungsmenge existiert, überspringe diesen
        if (std::find( first: route.begin(), last: route.end(), value: coordinates[i]) != route.end()) {
            continue;
        }
        double angle = -1;
        if (route.size() >= 2) {
            angle = cross_angle( from_node: route[route.size() - 2], over_node: route.back(), to_node: coordinates[i]);
        }
        if (route.empty() ||
            (std::find( first: route.begin(), last: route.end(), value: coordinates[i]) == route.end() &&
             (route.size() < 2 || angle >= 90))
                ) {
            route.push_back(coordinates[i]); //Füge den Knoten hinzu
            if (route.size() == coordinates.size()) { //Wenn alle Knoten in der Lösungsmenge sind
                return true;
            }
            if (solve( &: route, &: coordinates)) { //Wenn es eine Lösung mit der aktuellen Route gibt
                return true;
            } else {
                route.pop_back();
            }
        }
    }
    return false; //Wenn es mit der aktuellen Route keine Lösung geben kann
}
```

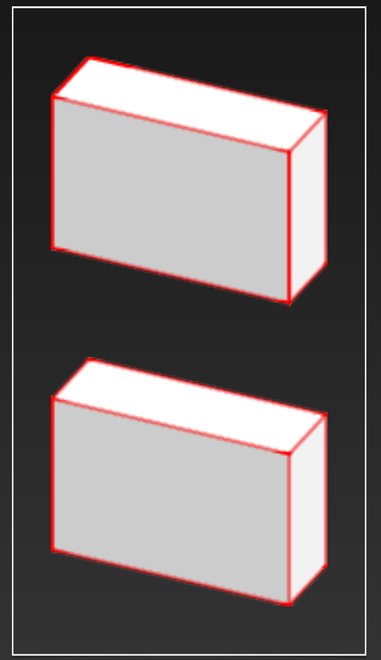
Alles Käse

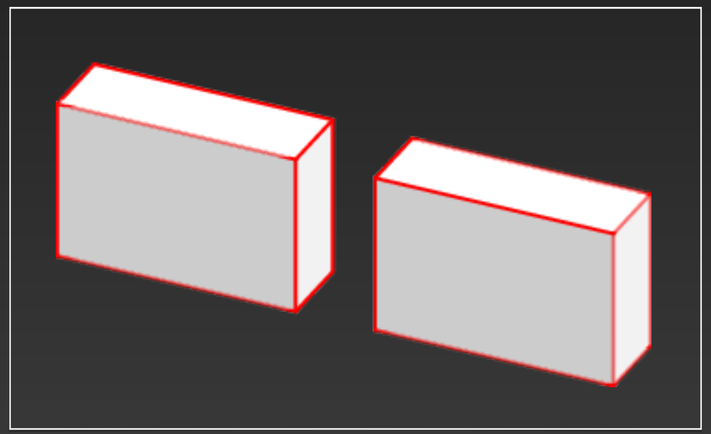
Aufgabe 2

Lösungsansatz
Verfahren & Algorithmen

- Greedy-Algorithmus
- Kombinationsproblem
- Quader zerschneiden





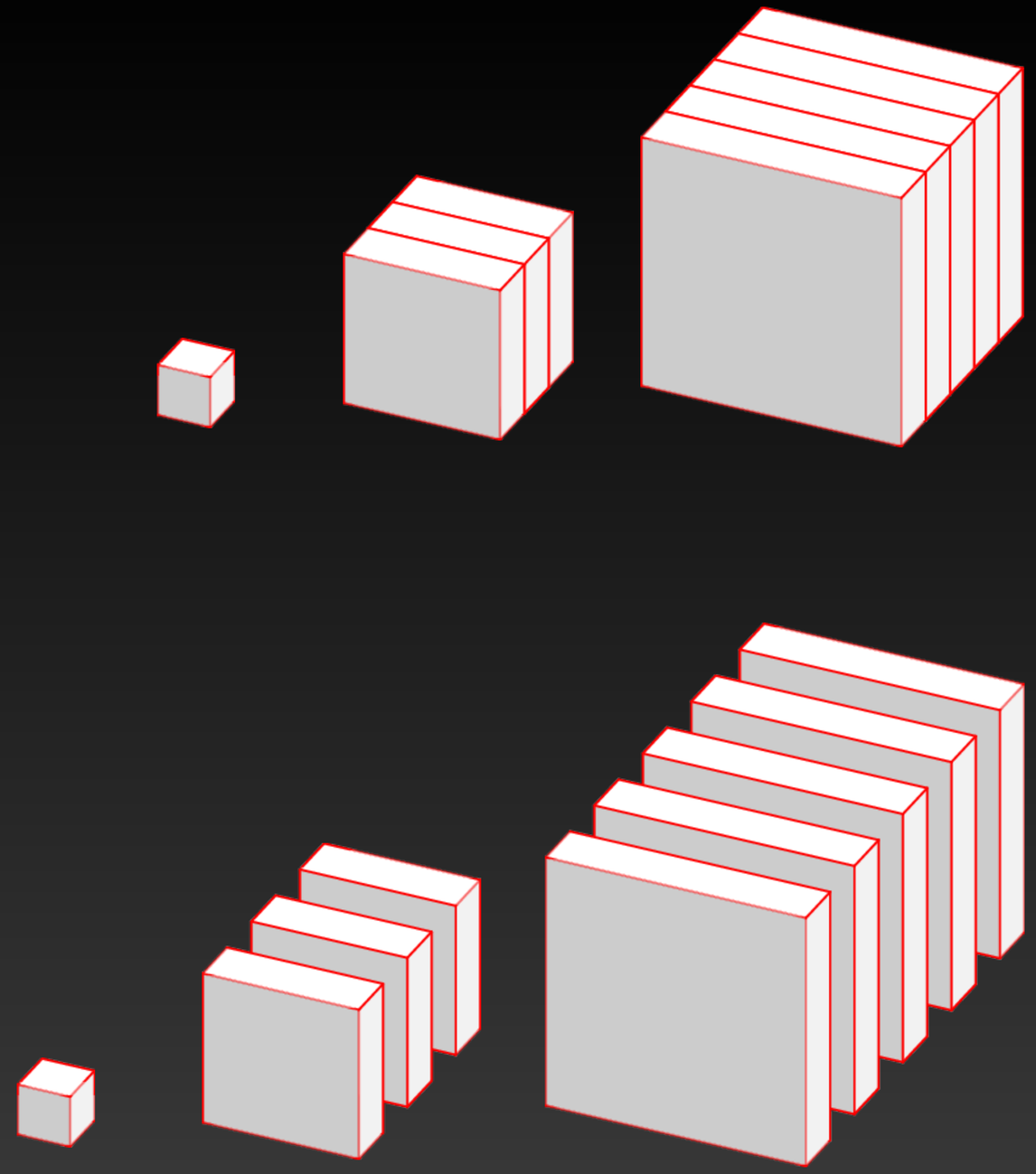


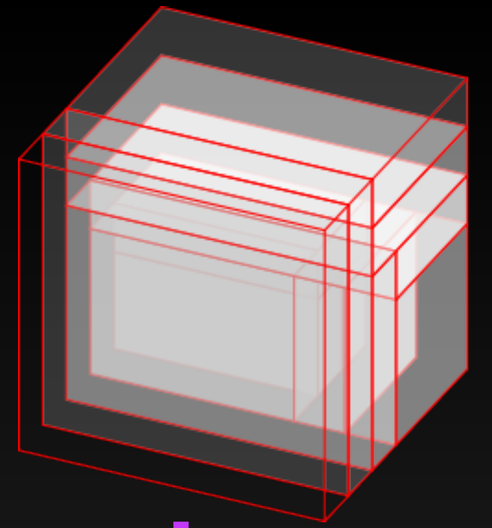
```cpp
bool calculate_square(int length, int height, int depth, std::vector<std::pair<Slice, Dimension>> &order, std::vector<Slice> &slices) {

    //Wenn mindestens eine der Seiten auf null ist (daher das Volumen des Quaders null ist)
    if (length == 0 || height == 0 || depth == 0) {
        return true;
    }
    std::vector<Slice> removed_slices;
    //Für jede noch nicht verwendete Schiebe
    for (auto it :iterator<Slice *> = slices.begin(); it != slices.end(); ++it) {
        Dimension dimension = can_remove_slice(length, height, depth, slice: *it);
        //Wenn die aktuelle Scheibe abgeschnitten werden kann
        if (dimension != INVALID) {
            //Aktualisiere die Maße des Quaders
            int new_length = length - (dimension == SIDE ? 1 : 0);
            int new_height = height - (dimension == TOP ? 1 : 0);
            int new_depth = depth - (dimension == FRONT ? 1 : 0);
            //Füge die Scheibe zur Lösungsmenge hinzu
            order.emplace_back( t1: *it, t2: dimension);
            removed_slices.push_back(*it);
            slices.erase( position: it);
            //Wenn es eine Lösung mit der aktuellen Scheibe gibt
            if (calculate_square( length: new_length, height: new_height, depth: new_depth, &: order, &: slices)) {
                return true;
            } else {
                //Entferne die aktuelle Scheibe von der Lösungsmenge
                order.pop_back();
                for (auto &slice :Slice & : removed_slices) {
                    slices.push_back(slice);
                }
                removed_slices.clear();
            }
        }
    }
    //Wenn keine der noch nicht betrachteten Scheiben verwendet werden kann
    return false;
}
```

Erweiterung
Lösungsidee & Umsetzung

- n-Quader aus n-Scheiben
- Kombinationsproblem

$$Kombination = \begin{Bmatrix} \{1 & 1 & 1\} \\ \{3 & 3 & 3\} \\ \{5 & 5 & 5\} \end{Bmatrix}$$





Computer science is no more about computers than astronomy is about telescopes

- Edsger W. Dijkstra