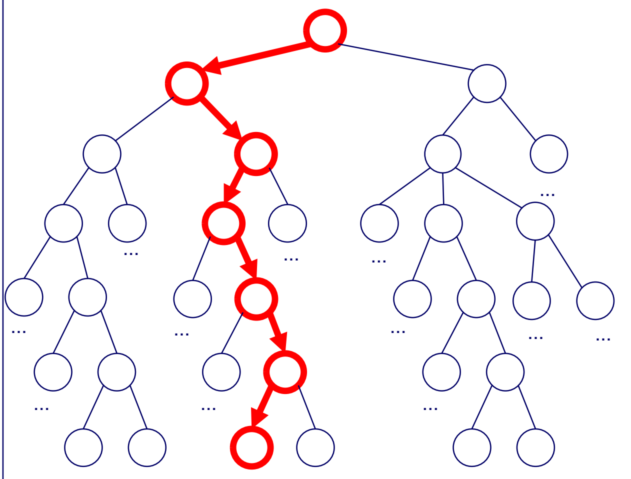
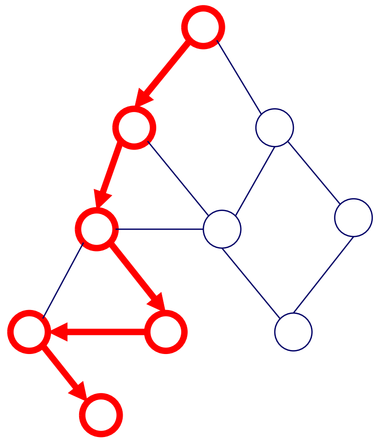
Infi puzzel op Adevent of Code 2019
Mijn leerpad/zoektocht naar een algemene oplossing [C++17]

> Maak een **tree** met alle routes permutaties
> Traverse alle branches, set van energie



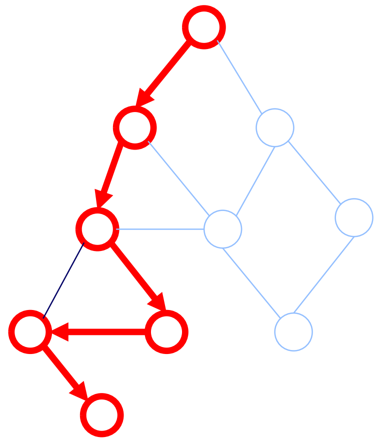
Depth first search
(Recursie of Stack)

> Maak **complete graph** (generieke class)
> Pas daarop Dijkstra toe



Breadth First Search
(Queue)

> **Dijkstra efficient**: exploreer alleen het relevante deel van de graph "as you go"



Dijkstra (P-BFS)
(Priority Queue & Set & Map)

```
Struc Node
{
    std::pair<int,int> pos;
    std::vector<Node*> nextFlats;
    int totalEnergyUsed;
};
```

```
Struct Arc; // Forward declaration
Struc Node
{
    std::string name;
    std::pair<int,int> pos;
    std::set<Arc*> arcs;
}
struct Arc
{
    Node* start, finish;
    int cost;
};
```

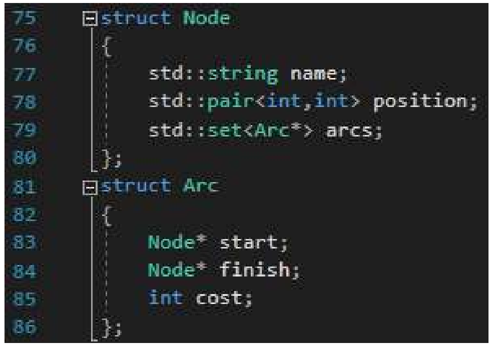
O(2^N) Processortijd*: 2800 ms (debug), ~30ms (release)
Flats: 29, Nodes: 52.280, loopCount = ~277.000

O(N^2) Processortijd*: ~25 ms (debug), ~15ms (release)
Flats: 29, Vertices: 29, Edges: 51, loopCount = ~800

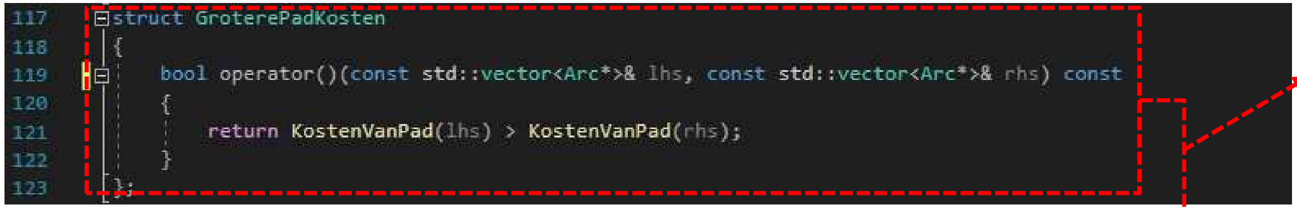
O(ElogV) Processortijd*: <5 ms (debug), <1 ms (release)
Flats: 29, Vertices: 28, Edges: 36, loopCount = ~500

* (i3-4160,3.6GHz)

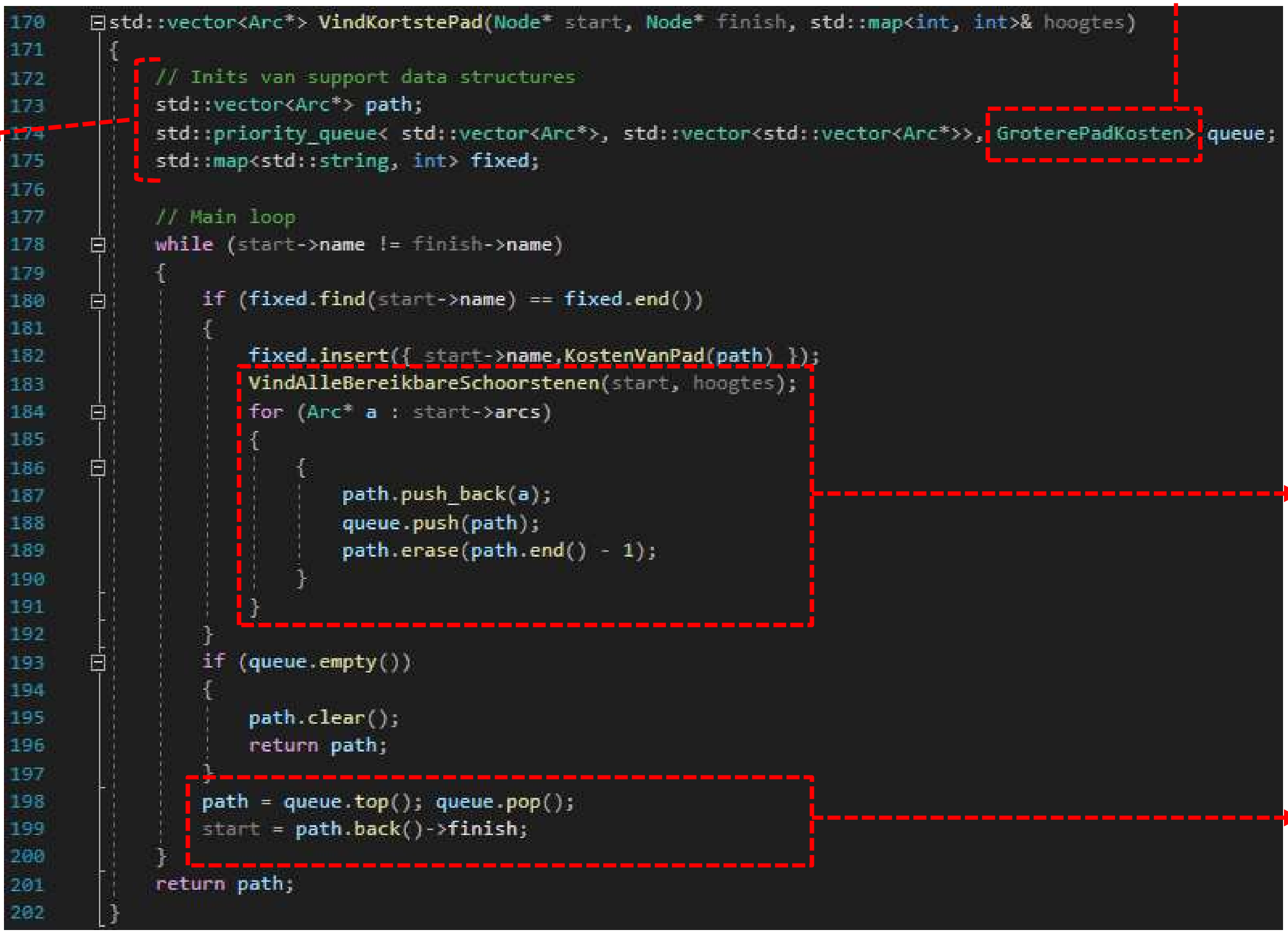
De schoonheid van Dijkstra’s algoritme [C++17]



```
struct Node
{
    std::string name;
    std::pair<int,int> position;
    std::set<Arc*> arcs;
};
struct Arc
{
    Node* start;
    Node* finish;
    int cost;
};
```



```
struct GroterePadKosten
{
    bool operator()(const std::vector<Arc*>& lhs, const std::vector<Arc*>& rhs) const
    {
        return KostenVanPad(lhs) > KostenVanPad(rhs);
    }
};
```



```
std::vector<Arc*> VindKortstePad(Node* start, Node* finish, std::map<int, int>& hoogtes)
{
    // Inits van support data structures
    std::vector<Arc*> path;
    std::priority_queue< std::vector<Arc*>, std::vector<std::vector<Arc*>>, GroterePadKosten> queue;
    std::map<std::string, int> fixed;

    // Main loop
    while (start->name != finish->name)
    {
        if (fixed.find(start->name) == fixed.end())
        {
            fixed.insert({ start->name,KostenVanPad(path) });
            VindAlleBereikbareSchoorstenen(start, hoogtes);
            for (Arc* a : start->arcs)
            {
                {
                    path.push_back(a);
                    queue.push(path);
                    path.erase(path.end() - 1);
                }
            }
        }
        if (queue.empty())
        {
            path.clear();
            return path;
        }
        path = queue.top(); queue.pop();
        start = path.back()->finish;
    }
    return path;
}
```

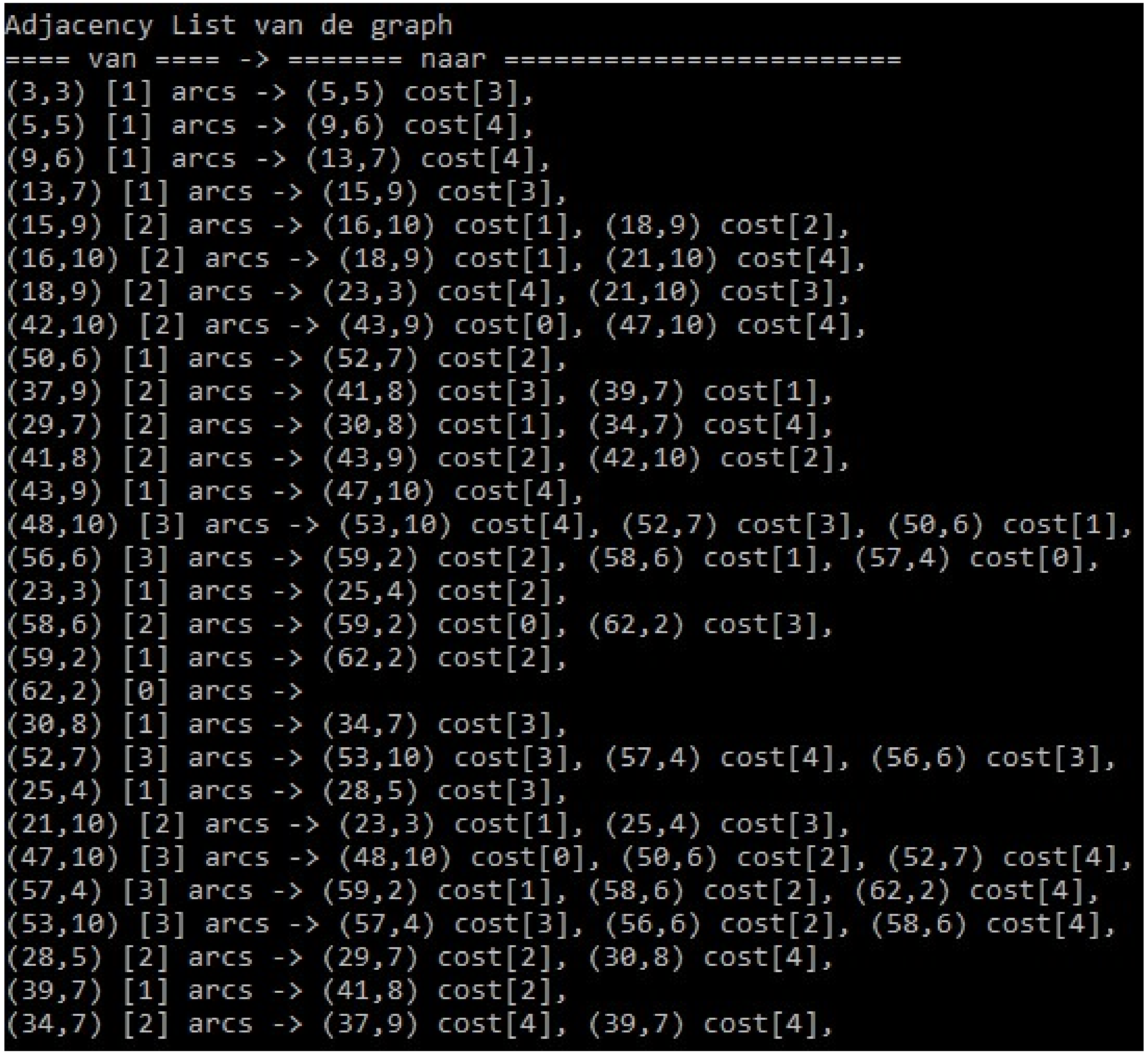
Comparison Function Object

De drie kern containers

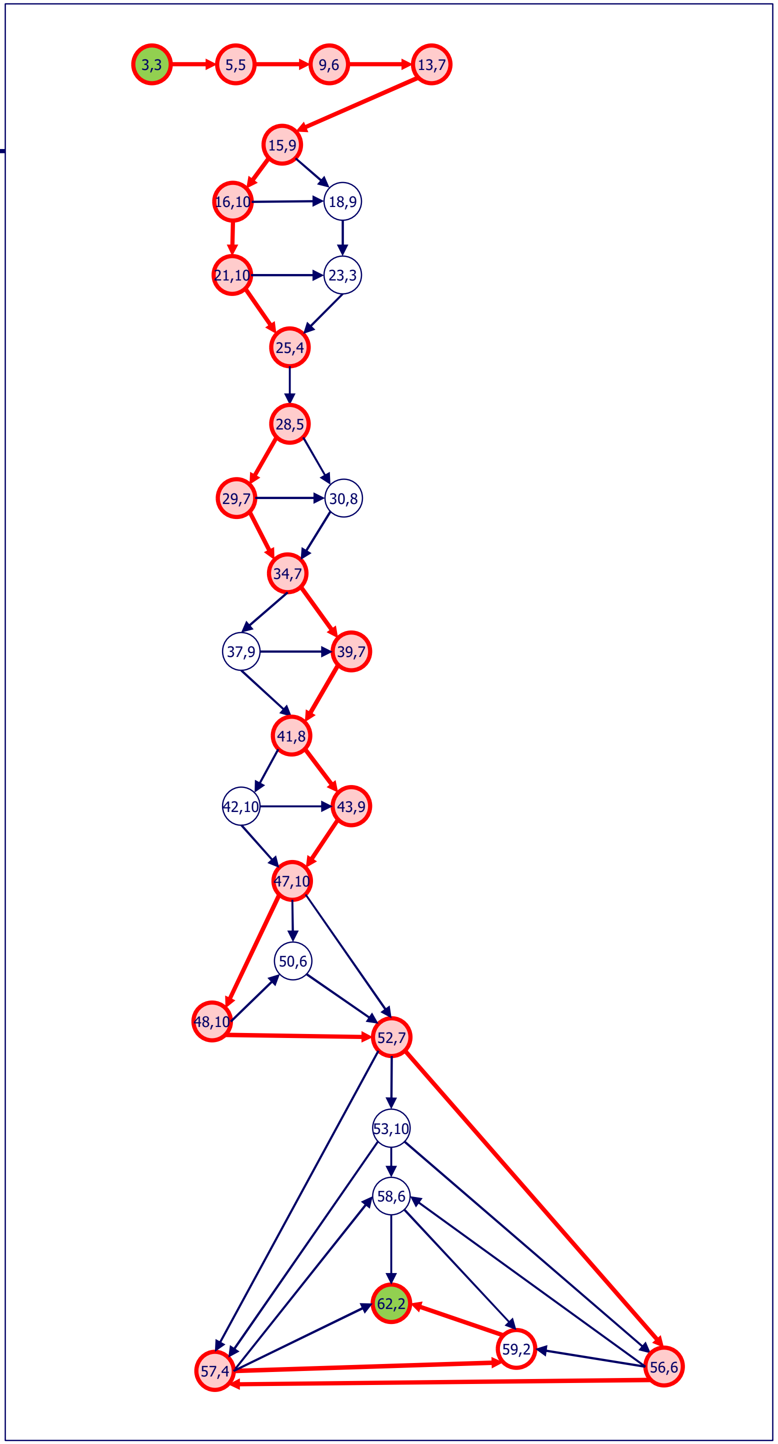
Exploreer alle nieuwe nodes…

Maar altijd zó dat je kortste paden eerst evalueert

Hoe zag “mijn” graph er echt uit?



```
Adjacency List van de graph
==== van ==== -> ======= naar =======================
(3,3) [1] arcs -> (5,5) cost[3],
(5,5) [1] arcs -> (9,6) cost[4],
(9,6) [1] arcs -> (13,7) cost[4],
(13,7) [1] arcs -> (15,9) cost[3],
(15,9) [2] arcs -> (16,10) cost[1], (18,9) cost[2],
(16,10) [2] arcs -> (18,9) cost[1], (21,10) cost[4],
(18,9) [2] arcs -> (23,3) cost[4], (21,10) cost[3],
(42,10) [2] arcs -> (43,9) cost[0], (47,10) cost[4],
(50,6) [1] arcs -> (52,7) cost[2],
(37,9) [2] arcs -> (41,8) cost[3], (39,7) cost[1],
(29,7) [2] arcs -> (30,8) cost[1], (34,7) cost[4],
(41,8) [2] arcs -> (43,9) cost[2], (42,10) cost[2],
(43,9) [1] arcs -> (47,10) cost[4],
(48,10) [3] arcs -> (53,10) cost[4], (52,7) cost[3], (50,6) cost[1],
(56,6) [3] arcs -> (59,2) cost[2], (58,6) cost[1], (57,4) cost[0],
(23,3) [1] arcs -> (25,4) cost[2],
(58,6) [2] arcs -> (59,2) cost[0], (62,2) cost[3],
(59,2) [1] arcs -> (62,2) cost[2],
(62,2) [0] arcs ->
(30,8) [1] arcs -> (34,7) cost[3],
(52,7) [3] arcs -> (53,10) cost[3], (57,4) cost[4], (56,6) cost[3],
(25,4) [1] arcs -> (28,5) cost[3],
(21,10) [2] arcs -> (23,3) cost[1], (25,4) cost[3],
(47,10) [3] arcs -> (48,10) cost[0], (50,6) cost[2], (52,7) cost[4],
(57,4) [3] arcs -> (59,2) cost[1], (58,6) cost[2], (62,2) cost[4],
(53,10) [3] arcs -> (57,4) cost[3], (56,6) cost[2], (58,6) cost[4],
(28,5) [2] arcs -> (29,7) cost[2], (30,8) cost[4],
(39,7) [1] arcs -> (41,8) cost[2],
(34,7) [2] arcs -> (37,9) cost[4], (39,7) cost[4],
```


3,3
5,5
9,6
13,7
15,9
16,10
18,9
21,10
23,3
25,4
28,5
29,7
30,8
34,7
37,9
39,7
41,8
42,10
43,9
47,10
50,6
48,10
52,7
53,10
58,6
62,2
59,2
57,4
56,6