Overview of this lecture

- Organizational
  - Feedback and results from Exercise Sheet 5 (Web app)

- Contraction Hierarchies (CHs)
  - Yet another (clever) algorithm for fast route planning
  - Basic idea: far away from source / target only use "important" roads (think of highways)
  - This lecture: outline + the central "contraction" procedure
  - Next lecture: missing details, so that you know how to build a route planner based on CH
  - Exercise Sheet 6: implement the central contraction method (that will be the basic building block of the CH pre-processing)
Summary / excerpts

- Fun exercise / interesting to see how web apps work
- Nice to see our algorithms in action / that it really works
- Server side was relatively straightforward
  - though some used the opportunity for further improvements
- Client side was not hard, but quite a lot of new stuff
  - code provided was (of course) very useful
  - though one said it made thing too easy
- Typical time investment 4-6 hours / student
Let's have a look at a few demos

- One with comparison to Google API
  - Observation: both routes reasonable, but often different
  - Reason: Google seems to penalize certain turns
- One on Baden-Württemberg  (not Baden-Württenberg)
  - Observation: Query time independent of $\text{dist}(s, t)$
  - Reason: Heuristic function computed for all nodes
Bidirectional Dijkstra 1/4

- Basic idea
  - "Simultaneously" search from both source and target
  - Stop when the search spaces "meet"
  - This reduces the search space only by a factor of $\sim 2$
  - However: bi-directional search is an important ingredient in many of the more sophisticated algorithms ... like CH
**Bidirectional Dijkstra 2/4**

- **Implementation**
  - **Interleave** the two Dijkstra computations as follows
    - in each step, one iteration from the Dijkstra where the smallest key in the PQ is smaller
    - alternatively, maintain a joint priority queue, where each item in the PQ knows to which Dijkstra it belongs
  - **Stop** when settling a node from one Dijkstra that is already settled in the other Dijkstra
    - that node is not necessarily on the SP ...
  - The cost of the shortest path is then
    \[
    \min \{ \text{dist}_s[u] + \text{dist}_t[u] : \text{for all } u \text{ visited in both Dijkstras} \} \]
Counterexample

- ... where the first node that is settled in both searches does not lie on the shortest path
Correctness proof

- Let $D = \text{dist}(s, t)$, the cost of the SP from $s$ to $t$
- Let $u$ be the first node settled in both Dijkstras
- If both dist labels of $u$ are exactly $D/2$, we are done
- If not, one of the dist labels must be $> D/2$
- Hence all nodes with $\text{dist} \leq D/2$ have already been settled
- Let $v_s$ and $v_t$ on a shortest path from $s$ to $t$ such that
  \[ \text{dist}(s, v_s) \leq D/2 \quad \text{and} \quad \text{dist}(v_t, t) \leq D/2 \]
- Then $v_s$ has already been settled in the Dijkstra from $s$, and the relaxation has set $\text{dist}_s[v_t] = \text{dist}(s, v_t)$
- Same for $v_t$, hence $\text{dist}_s[v_t] + \text{dist}_t[v_t] = \text{dist}(s,t)$
Hierarchical Approaches  1/4

- Basic intuition
  - "Far away" from the source and target, consider only "important" roads ... the further away, the more important
  - Let's look at the shortest path of some random queries on Google Maps, typically:
    - close to source and target: mainly white (residential) roads
    - a bit further away: mainly yellow (national) roads
    - even further away: mainly orange (motorway) roads
  - But also note that this is not always true
Hierarchical Approaches  2/4

- This intuition leads to the following heuristic
  - Indeed consider the types / colors from the road, with an order between them, e.g. white < yellow < orange
  - Have a radius for each color > white: \( r_{\text{yellow}}, r_{\text{orange}} \)
  - Run a bi-directional Dijkstra, with the following constraints
    - at distance \( \geq r_{\text{yellow}} \) from source and target, consider only roads of type \( \geq \text{yellow} \)
    - at distance \( \geq r_{\text{orange}} \) from source and target, consider only roads of type \( \geq \text{orange} \)
  - **Note:** this does not necessarily find the shortest path
  - Still, heuristics of this kind were employed in navigation devices for a long time ... since no better algo's were known
Hierarchical Approaches  3/4

- **Highway Hierarchies (HHs)**
  - **Compute** a level for each **arc**
  - Along with a **radius** for each level: \( r_1, r_2, r_3, \ldots \)
  - Similarly as for the heuristic, run bi-directional Dijkstra
    - constraint now: at distance \( \geq r_i \) from the source and target, consider only arcs of level \( \geq i \)
  - This was first made precise in an ESA 2005 paper by Schultes and Sanders (KIT, Karlsruhe) ... see references
  - **Note:** the basic idea is simple, but the (implementation) details are quite intricate, in particular:
    - hard to get the implementation error-free in practice
Hierarchical Approaches  4/4

- **Contraction Hierarchies (CHs)**
  - **Compute** a level for each node
  - At query time again do a bidirectional Dijkstra
    - in the Dijkstra from the source consider only arcs $u,v$ where $\text{level}(v) > \text{level}(u)$ ... so called **upwards** graph
    - in the Dijkstra from the target, consider only arcs $v,u$ with $\text{level}(v) > \text{level}(u)$ ... so called **downwards** graph
  - Intuitively, this is like a "continuous" version of highway hierarchies ... and significantly easier to implement
  - We will look at CH in more detail now ...
Contraction of a single node

- This is the basic building block of the CH precomputation
- **Idea:** take out a node, and add all necessary arcs such that all SP distances in the remaining graphs are preserved
- Formally, a node \( v \) is **contracted** as follows
  - Let \( \{u_1, \ldots, u_l\} \) be the incoming arcs, i.e. \((u_i, v) \in E\)
  - Let \( \{w_1, \ldots, w_k\} \) be the outgoing arcs, i.e. \((v, w_j) \in E\)
  - For each pair \( \{u_i, w_j\} \), if \((u_i, v, w_j)\) is the **only** shortest path from \( u_i \) to \( w_j \), add the **shortcut** arc \((u_i, w_j)\)
  - Then **remove** \( v \) and its adjacent arcs from the graph
Example for contraction of a single node

- Shortcut must be added
- Shortcut not absolutely necessary, but OK to add it... see later slide
Contraction of all nodes in the graph

- Let $u_1, \ldots, u_n$ be an arbitrary order of the nodes
- We will see that CH is correct for any order, but more efficient for some orders than for others ... next lecture
- Let $G = G_0$ be the initial graph
- Let $G_i$ be the graph obtained from $G_{i-1}$ by contracting $u_i$ that is, without $u_i$ and adjacent arcs and with shortcuts
  - in particular therefore, $G_i$ has $n - i$ nodes
- In the end, let $G^* = \text{the original graph with all nodes and arcs and all shortcuts}$ from any of the $G_1, G_2, \ldots$
- In the implementation, we can work on one and the same graph data structure throughout the algorithm ... later slide
Example for contraction of all nodes in a graph
Given $G^* = (V, E^*)$ and a source $s$ and a target $t$

- Define the upwards graph $G^* \uparrow = (V, \{(u, v) \in E^* : v > u\})$
- Define the downwards graph $G^* \downarrow = (V, \{(u, v) \in E^* : v < u\})$
- Do a full Dijkstra computation from $s$ \textbf{forwards} in $G^* \uparrow$
- Do a full Dijkstra computation from $t$ \textbf{backwards} in $G^* \downarrow$
- Let $I$ be the set of nodes settled in \textbf{both} Dijkstras
- Take $\text{dist}(s, t) = \min \{\text{dist}(s, v) + \text{dist}(v, t) : v \in I\}$
- Is this correct and if yes why? ... next lecture
- In the implementation, we need not construct $G^* \uparrow$ and $G^* \downarrow$ explicitly, we can just work on $G^*$ ... next lecture
- In symm. graphs backw. on $G^* \downarrow = \text{forw.}$ on $G^* \uparrow$ ... next lecture
Example query on our example graph from before
How to determine when a shortcut is needed?

- **Recall:** when contracting node $v$, we need to insert the shortcut arc $(u, w)$, if and only if $(u, v) \in E$ and $(v, w) \in E$ and $(u, v, w)$ is the only shortest path from $u$ to $w$
- As before, $\{u_i\} =$ incoming arcs and $\{w_j\} =$ outgoing arcs
- Perform a Dijkstra **for each** $u_i$ in the graph **without** $v$
- Let $D_{ij} = \text{cost}(u_i, v) + \text{cost}(v, w_j)$ ... cost of path via $v$
- In the Dijkstra from $u_i$
  - ... stop when node with cost $> \max_j D_{ij}$ is settled
  - ... add shortcut $(u_i, w_j)$ if and only if $\text{dist}[w_j] > D_{ij}$
Correctness of this routine

- Assume there is a SP from $u_i$ to $w_j$ that does not pass through $v$
  - then the cost of that SP is $\leq D_{ij}$ and the Dijkstra from $u_i$ just described will not stop before it has found it
  - then $\text{dist}[w_j] \leq D_{ij}$ and indeed no shortcut is added
- Beware: there might be a SP through $v$ with cost $< D_{ij}$
  - that looks like a problem, because this might be shorter than the SP in the graph without $v$
  - and we might not add a shortcut although we should
  - But such a path will then contain $(u_i', v, w_j')$
  - And this will be taken care of by the Dijkstra from $u_i'$
Heuristic improvement

- For each Dijkstra computation (from each of the $u_i$), put a limit on the size of the search space (#nodes settled)
  - With this heuristic, we may fail to find a shortest path from $u_i$ to $w_j$ that does not use $v$, and thus insert the shortcut $(u_i, w_j)$ unnecessarily
  - But unnecessary shortcuts do not harm correctness, only performance (if there are too many of them)
  - So there is a trade-off: if the heuristic saves a lot of time in the precomputation at the cost of only a few unnecessary shortcuts, than it is worth it
- Various additional heuristics in the paper ... see references
How to add shortcuts / remove contracted nodes?

- If you implemented the adjacency lists with an `Array<Array<Arc>>`, adding arcs is straightforward.
- But make sure that either your Dijkstra implementation does not have a problem with the same arc existing twice ... or that you avoid adding an already existing arc.
- Removing nodes / arcs from the graph is more cumbersome, but luckily there is no need to do that.
- Instead, you can just ignore the respective nodes / arcs.
- In the precomputation, when contracting $u_i$, simply ignore all nodes $u_1,\ldots u_{i-1}$ and their adjacent arcs.
- You can use `Arc::arcFlag` for that ... see code suggestion.
The Dijkstra searches for each contraction
- ... should take only very little time (<< 1 millisecond)
  - for the full CH algorithm, we have to do one per node
- To achieve that, pay attention to the following
  - make sure that the Dijkstra search spaces are small
    ... see the three slides on "Shortcuts"
  - requires two trivial extensions of DijsktrasAlgorithm class
    ... see code design suggestion linked on Wiki
  - avoid resetting the dist value for every node ... this
    would take $\Theta(n)$ time for each (tiny) Dijkstra
  - instead only reset the dist values of nodes that were
    visited in the previous Dijkstra (visitedNodes array)
References

- **Highway Hierarchies**
  - Engineering Highway Hierarchies
  - Highway Hierarchies Hasten Exact Shortest Path Queries
    Dominik Schultes and Peter Sanders, ESA 2005 & 2006
    http://algo2.iti.uka.de/schultes/hwy/esa06HwyHierarchies.pdf
    http://algo2.iti.uka.de/schultes/hwy/esaHwyHierarchies.pdf

- **Contraction Hierarchies**
  - Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks
    Geisberger, Sanders, Schultes, Delling, WEA 2008
    http://algo2.iti.uka.de/schultes/hwy/contract.pdf