数据结构与算法

Data Structure and Algorithm

XVIII. 图论 II

授课人：Kevin Feng

翻译　：梁少华

Review

回顾

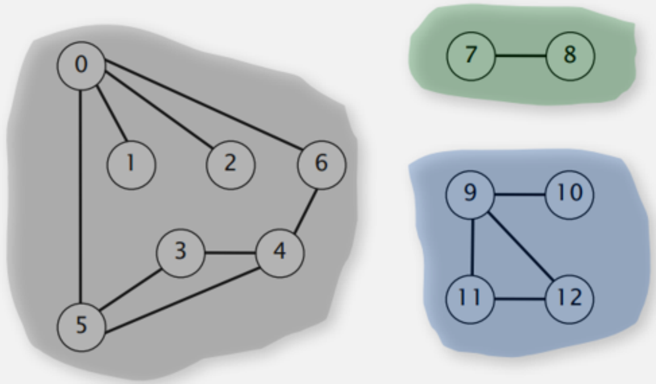
- 数据结构与算法
- 数学回顾
- 数组
- 数组列表
- 搜索和排列
- 递归与迭代
- 二进制搜索
- 分而治之
- 链接列表
- 散列表
- 树
- 堆
- 图论-BFS,DFS

概述

- 连通分支
- 单路径最短路径
  - Dijkstra 算法
  - Bellman-Ford 算法
  - A* 算法
- 生成树

连通图

- 在无向图中，若从顶点v到顶点w之间存在路径，则顶点v和w是连通的
- 在有向图中，连接顶点v和顶点w的路径中所有的变都必须同向。
- 如果图中任意两点都是连通的，那么图被称作连通图。



3 connected components

v	id[]
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	2
10	2
11	2
12	2

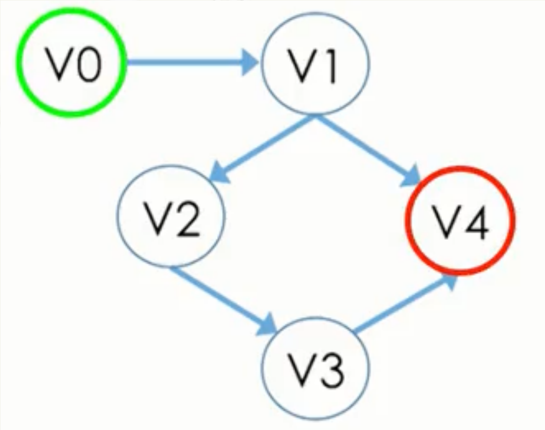
- Goal: Partition vertices into connected components

Connected components

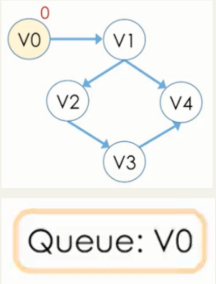
Initialize all vertices v as unmarked.

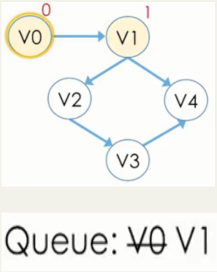
For each unmarked vertex v, run DFS to identify all vertices discovered as part of the same component.

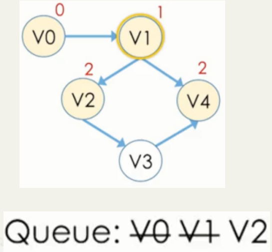
最短路径



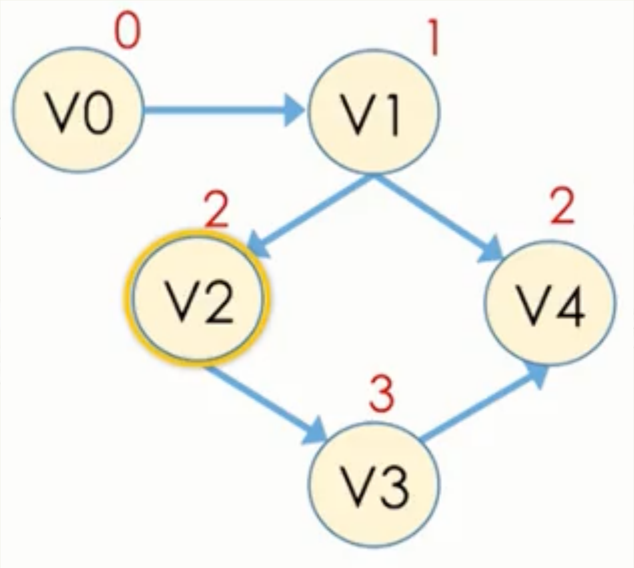
- 从 V0 到 V4
- 让我们试试 BFS





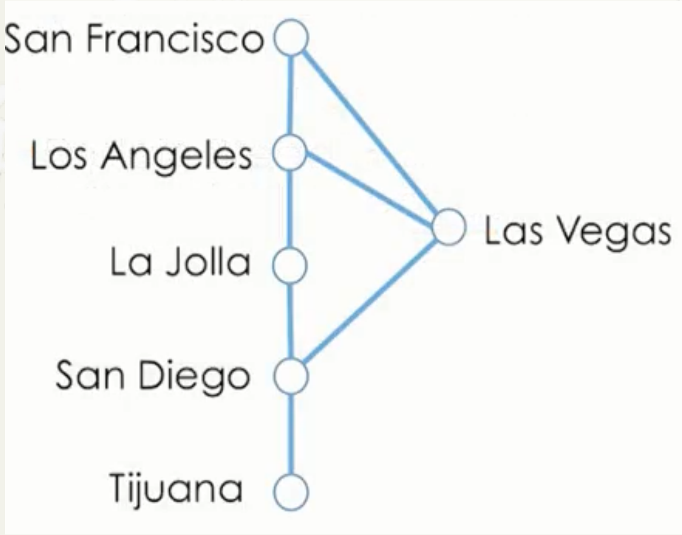


- 发现 V4
- 我们 *完成了吗*?

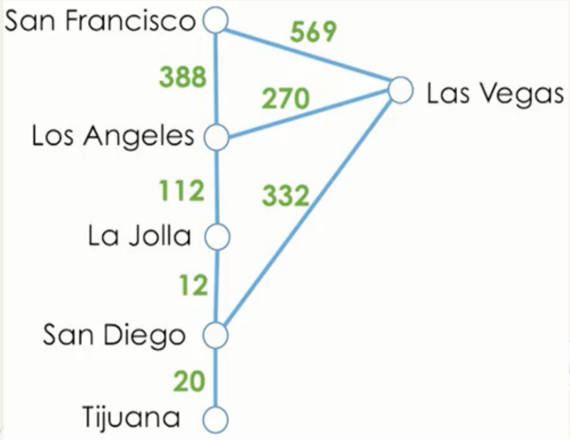


- BFS是否适用于所有图表?

作为地图的地理地图



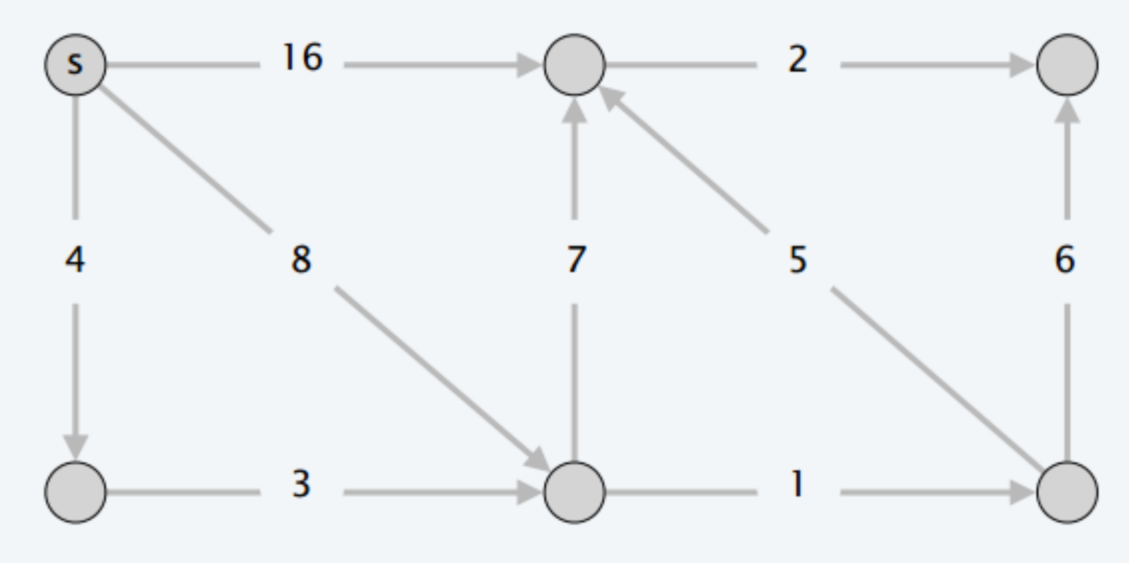
- 图中缺少了什么?
  - 距离
- 还有什么?
  - 速度限制
  - 交通
  - 登机交叉延误



- BFS能够找到从圣地亚哥到旧金山的最短路径?
- BFS不考虑边缘权重，只考虑边缘数量

单源最短路径问题

- 给定一个图 *G* = (*V*, *E*)和一个V中的“源”顶点*u*， V中的一个“目标”顶点*v*，找到从*u*到*v*的最小代价路径.
- 给定图*G* = (*V*, *E*) 和V中的“源”顶点*s*， 求出从*s*到V中每个顶点的最小代价路径.



Dijkstra’s 算法

- 回顾 BFS
  - 如何跟踪下一步搜索的位置?
  - 使用队列
- Dijkstra’s 算法
  - 贪婪
  - 使用优先级队列
  - 列表中添加元素{元素，优先级}，并从另一端删除最高优先级项
  - 入队→添加一个{元素，优先级}
  - 队列→删除最高优先级的元素
  - 优先级队列通常使用“堆”来实现，并可以优先考虑低值（Min-Heap）或大值（Max-Heap）

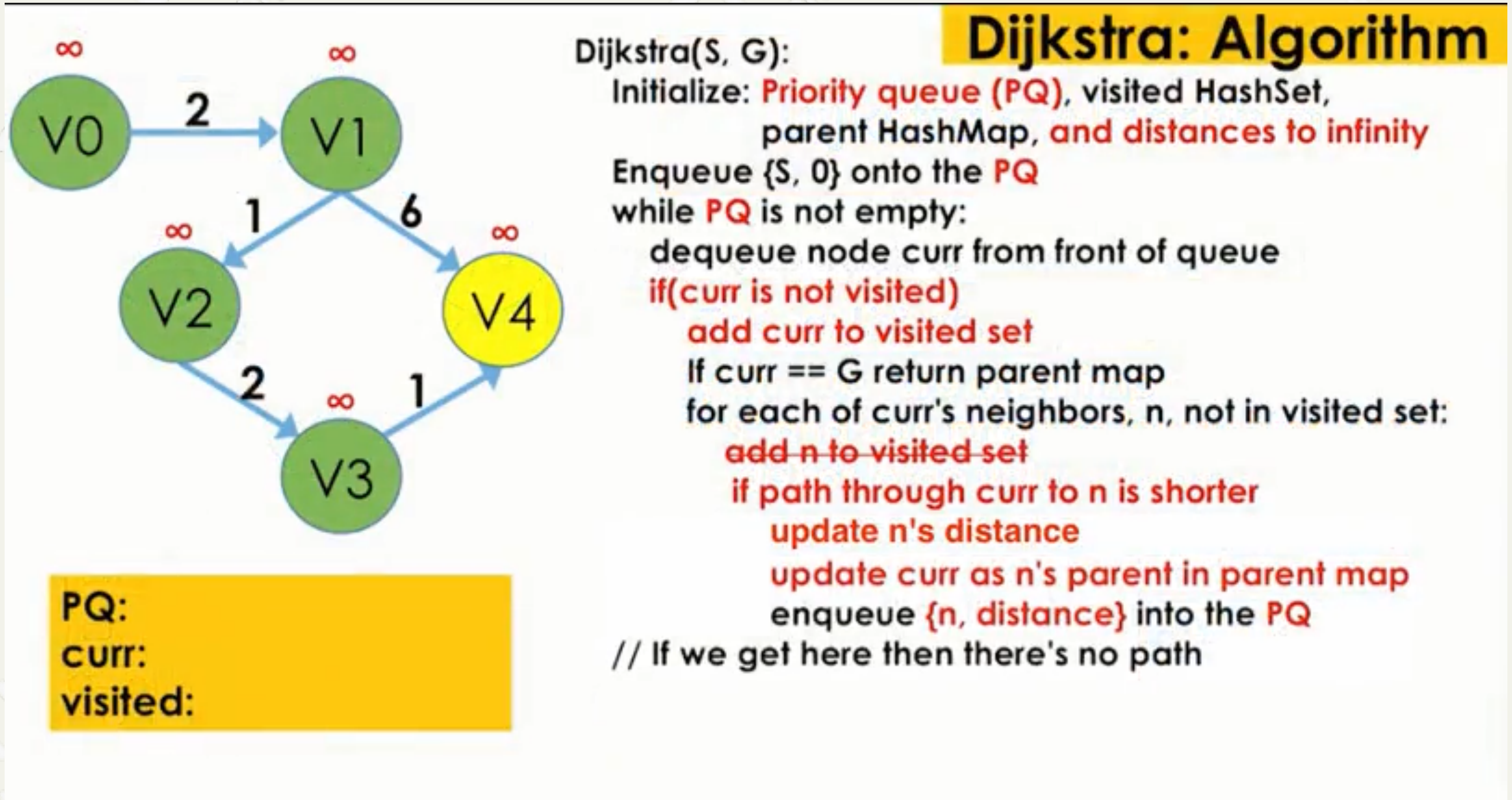
粗略的想法

- Maintain an estimate $d[v]$ of the length $\delta(s, v)$ of the shortest path for each vertex v
- Always $d[v] \geq \delta(s, v)$ and $d[v]$ equals the length of a known path
  - ($d[v]= \infty$ if we have no paths so far)
- Initially $d[s]=0$ and all the other $d[v]$ values are set to ∞. The algorithm will then process the vertices one by one in some order.
  - The processed vertex's estimate will be validated as being real shortest distance, i.e. $d[v] = \delta(s, v)$
- Here "processing a vertex u" means finding new paths and updating $d[v]$ for all $v \in Adj[u]$ if necessary. The process by which an estimate is updated is called **relaxation**.
- When all vertices have been processed,
  - $d[v]= \delta(s, v)$ **for all** v
- ***Question 1:*** How does the algorithm find new paths and do the relaxation?
- ***Question 2:*** In which order does the algorithm process the vertices one by one?

粗略的答案

- ***Question 1:*** How does the algorithm find new paths and do the relaxation?
- ***Answer:*** Finding new paths. When processing a vertex u , the algorithm will examine all vertices $v \in Adj[u]$. For each vertex $v \in Adj[u]$, a new path from s to v is found (path from s to u + new edge).
  - We use Greedy algorithm. For each vertex $v \in Adj[u]$. The next vertex processed is always a vertex $v \in Adj[u]$ for which $d[u]$ is minimum
  - that is, we take the unprocessed vertex that is closest (by our estimate) to
- ***Question 2:*** In which order does the algorithm process the vertices one by one?
- ***Answer:*** Relaxation. If the length of the new path s to v is shorter than $d[v]$, then update $d[v]$ to the length of this new path.

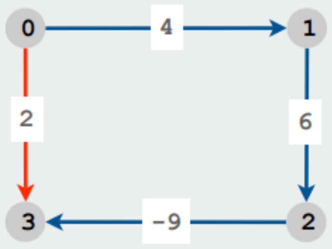
Dijkstra’s 算法



Dijkstra’s 挑战

- 负数会怎么样？

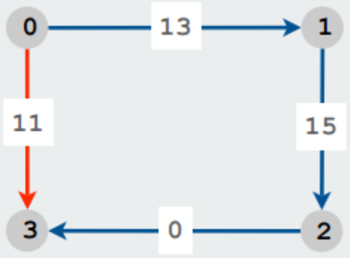
Dijkstra. Doesn’t work with negative edge weights.



Dijkstra selects vertex 3 immediately after 0.
But shortest path from 0 to 3 is 0→1→2→3.

- 我们试试

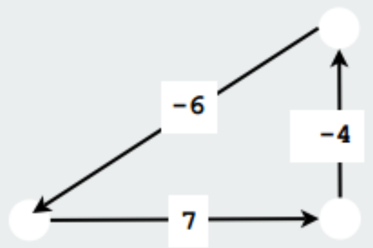
Re-weighting. Adding a constant to every edge weight also doesn’t work.



Adding 9 to each edge changes the shortest path because it adds 9 to each segment, wrong thing to do for paths with many segments.

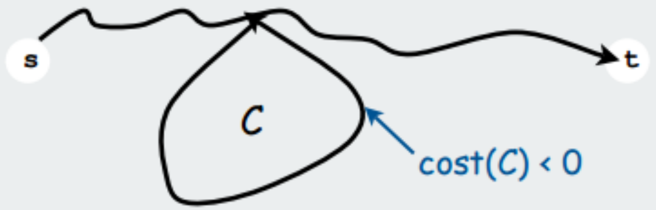
负循环

Negative cycle. Directed cycle whose sum of edge weights is negative.



Observations.

- If negative cycle C on path from `s` to `t`, then shortest path can be made arbitrarily negative by spinning around cycle
- There exists a shortest `s-t` path that is simple.



Worse news: need a different problem

Bellman-Ford 算法

Initialize distTo[s] = 0 and distTo[v] = ∞ for all other vertices.

Repeat V times:

- Relax each edge.

```
for (int i = 0; i < G.V(); i++)
   for (int v = 0; v < G.V(); v++)
      for (DirectedEdge e : G.adj(v))
         relax(e);
```

pass i (relax each edge)

- *时间复杂度?*
- *暴力解法*
- *O (EV)*

Bellman-Ford 改进

Observation. If `distTo[v]` does not change during pass `i`, no need to relax any edge pointing from `v` in pass `i+1`.

FIFO implementation. Maintain queue of vertices whose `distTo[]` changed.

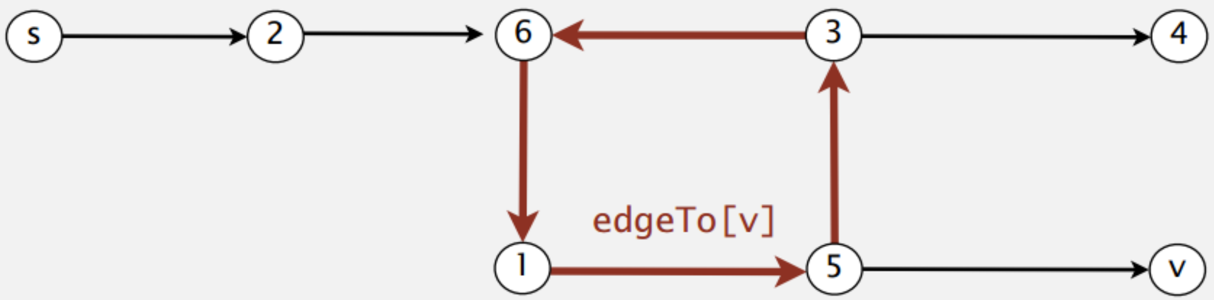
be careful to keep at most one copy of each vertex on queue (why?)

Overall effect.

- The running time is still proportional to $E \times V$ in worst case.
- But much faster than that in practice.

寻找负循环

Observation. If there is a negative cycle, Bellman-Ford gets stuck in loop, updating `distTo[]` and `edgeTo[]` entries of vertices in the cycle.



Proposition. If any vertex `v` is updated in pass V, there exists a negative cycle (and can trace back `edgeTo[v]` entries to find it).

In practice. Check for negative cycles more frequently.

负循环应用

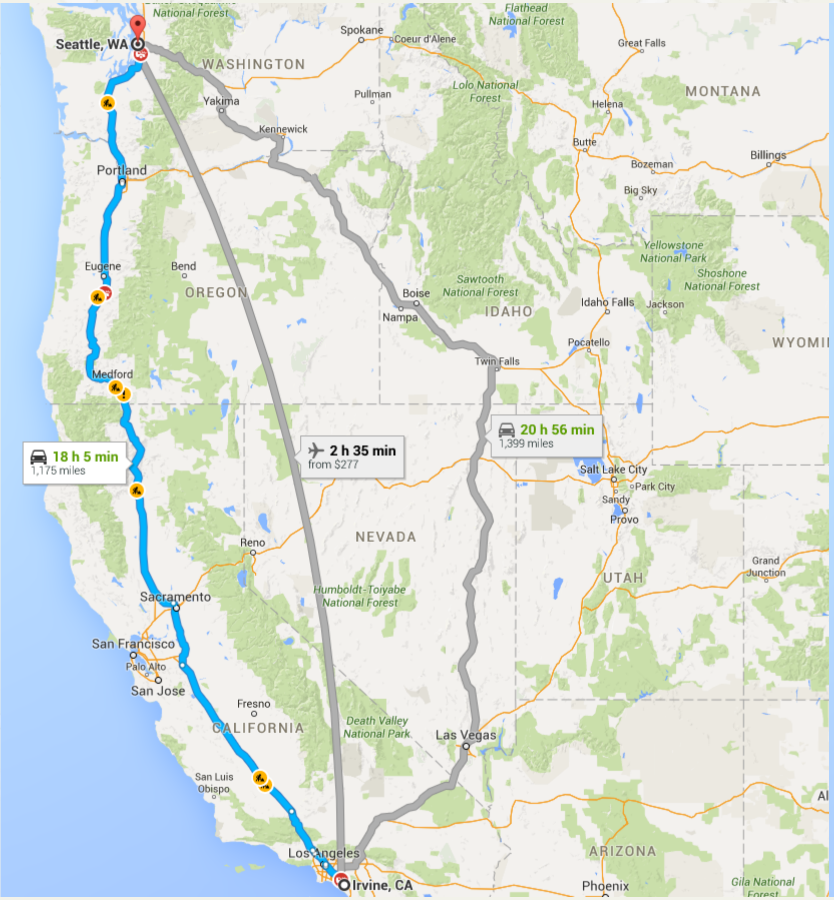
Problem. Given table of exchange rates, is there an arbitrage opportunity?

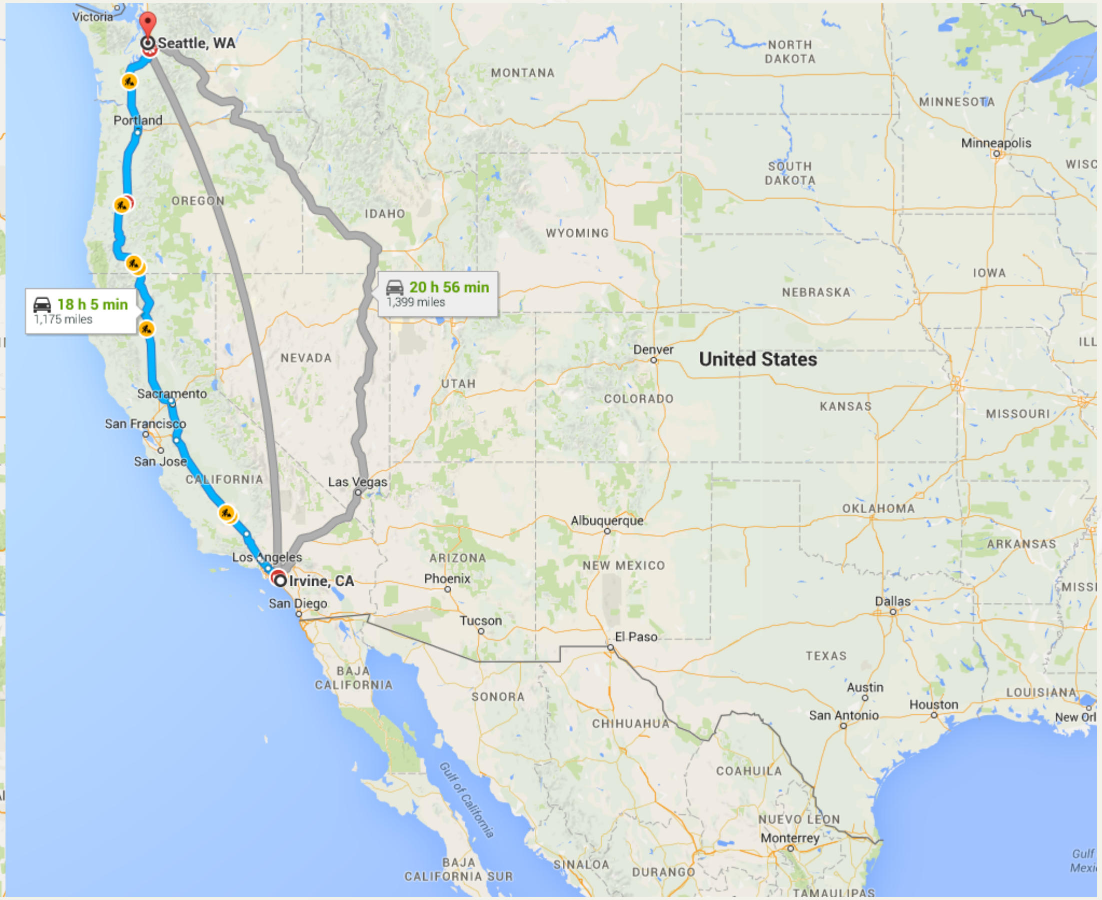
	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.011
EUR	1.350	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.943	0.698	0.620	1	0.953
CAD	0.995	0.732	0.650	1.049	1

Ex. \$1,000 $\Rightarrow$ 741 Euros $\Rightarrow$ 1,012.206 Canadian dollars $\Rightarrow$ \$1,007.14497.

$1000 \times 0.741 \times 1.366 \times 0.995 = 1007.14497$

Dijkstra 局限性





- Dijkstra is BFS
- 并可能到达凤凰城，达拉斯，丹佛和盐湖城
- Dijkstra只考虑距离信号源的距离

A* 算法

- Dijkstra’s 算法
  - 是基于优先级队列排序
  - G(n):从起始顶点到顶点n的距离
- 我们也应该考虑距离目标
- A* 算法
  - G(n):从起始顶点到顶点n的距离
  - H(n):从顶点n到目标顶点的启发式估计成本
  - F(n) = g(n) + h(n)
  - Dijkstra可以看作h(n) = 0的特例
  - 保证找到最短路径如果估计永远不是高估
  - 在前面的例子中,
  - 低估：使用直线距离
  - 这很容易计算，我们有经度和纬度
  - 只是改变优先功能

数据结构与算法

Data Structure and Algorithm

XVIII. 图论II

结束

授课人：Kevin Feng

翻译　：梁少华