ETH zürich

Übung 12

Datenstrukturen und Algorithmen, D-MATH, ETH Zurich

Programm von heute

Feedback letzte Übung

MaxFlow

C++ Threads

Zwei Quiz-Fragen

1. Feedback letzte Übung

Aufgabe Union-Find

```
class UnionFind{
    std::vector<size_t> parents_;
public:
    UnionFind(size_t size) : parents_(size, size) { };

    size_t find(size_t index){
        while(parents_[index] != parents_.size())
            index = parents_[index];
        return index;
    }

    void unite(size_t a, size_t b){
        parents_[find(a)] = b;
    }
};
```

Union-Find with Map (Aufgabe Kruskal)

```
class UnionFind {
  private:
    std::unordered_map<NodeP,NodeP> parent;
    std::unordered_map<NodeP,unsigned> depth;
  public:
    void MakeSet(NodeP n){
      parent[n] = n; depth[n] = 0;
    }
    NodeP Find(NodeP n){
      while (parent[n] != n){
        n = parent[n];
      }
      return n;
    }
```

Optimierendes Union

```
bool Union(NodeP l, NodeP r){
  l = Find(l);
  r = Find(r);
  if (l == r){
    return false;
  } else {
    if (depth[l] < depth[r])
      std::swap(l,r);
    parent[r] = l;
    if (depth[l] == depth[r])
      depth[l]++;
    return true;
  }
}
```

Alternative: optimierendes Find

```
NodeP Find(NodeP n){
  NodeP root = n;
  while (parent[root] != root){
    root = parent[root];
  }
  while (parent[n] != root){
    auto next = parent[n];
    parent[n] = root;
    n = next;
  }
  return root;
}
```

kein **depth** nötig

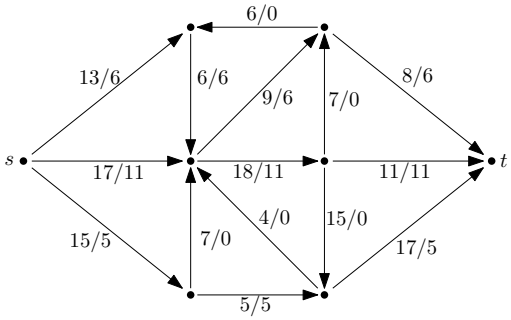
Kruskal

```
std::vector<Edge> result;
std::sort(edges.begin(),edges.end(),
  [](const Edge& l, const Edge& r) {return l.length < r.length;}
);

UnionFind uf;
for (auto n: nodes)
  uf.MakeSet(n);

for (const auto& e: edges){
  if (uf.Union(e.source, e.target)){
    result.push_back(e);
  }
}
return result;
```

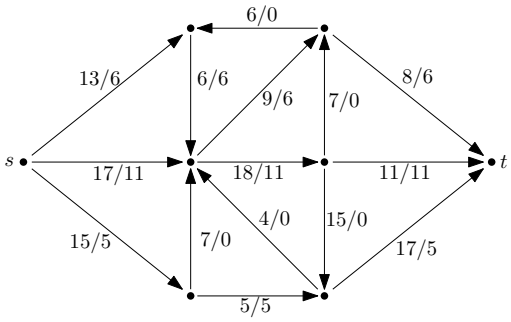
TSP

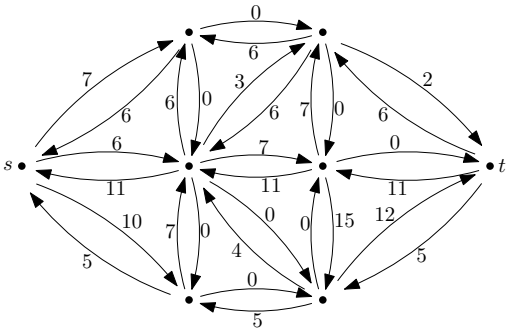
```
std::vector<Edge> Graph::TSP(){
  std::vector<Edge> mst = Kruskal();
  std::unordered_map<NodeP,std::vector<Edge>> adj;
  for (const auto& e: mst){
    adj[e.source].push_back(e);
    adj[e.target].emplace_back(e.target,e.source);
  }
  return DFS(adj,mst[0].source);
}
```

Aufgabe Manual Max-Flow



Aufgabe Manual Max-Flow





Travelling Salesperson Problem

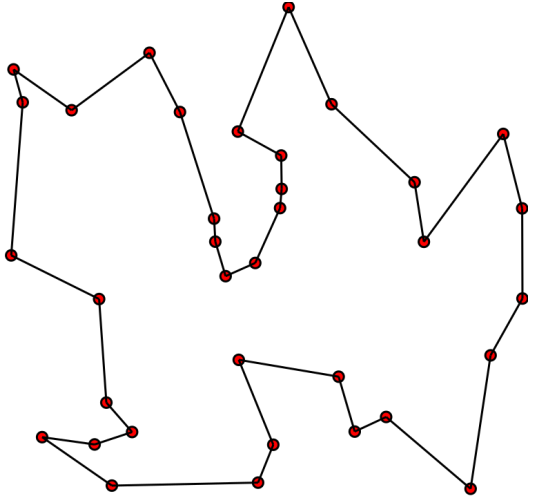
Problem

Gegeben ist eine Karte und eine Liste von Städten. Welches ist die kürzeste Route, die jede Stadt einmal besucht und in die ursprüngliche Stadt zurückkehrt?

Mathematical model

Auf einem ungerichteten, gewichteten Graph G ist nach dem Kreis gesucht, welcher jede der Knoten von G genau einmal enthält und die kleinste Kantensumme aufweist.

Travelling Salesperson Problem



Travelling Salesperson Problem

- Es ist kein Polynomialzeitalgorithmus zum Lösen des Problems bekannt.
- Es gibt verschiedene heuristische Algorithmen. Diese liefern oft nicht die optimale Lösung.

Travelling Salesperson Problem

- Der heuristische Algorithmus, den Sie auf CodeExpert implementieren sollen (*The Travelling Student*) benutzt einen Minimalen Spannbaum:
  1. Berechne den Minimalen Spannbaum M
  2. Mache eine Tiefensuche auf M
- Der Algorithmus ist eine 2-Approximation. Das bedeutet, dass er eine Lösung liefert, die maximal 2 mal die Kosten einer optimalen Lösung aufweist.
- Der Algorithmus geht von einem vollständigen Graphen $G = (V, E, c)$ aus, auf dem die Dreiechsungleichung gilt:
  $c(v, w) \leq c(v, x) + c(x, w) \; \forall \; v, w, x \in V$

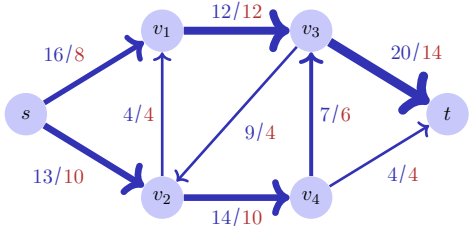
2. MaxFlow

Fluss

Ein **Fluss** $f : V \times V \to \mathbb{R}$ erfüllt folgende Bedingungen:

- **Kapazitätsbeschränkung**:
  Für alle $u, v \in V$: $f(u, v) \leq c(u, v)$.
- **Schiefsymmetrie**:
  Für alle $u, v \in V$: $f(u, v) = -f(v, u)$.
- **Flusserhaltung**:
  Für alle $u \in V \setminus \{s, t\}$:

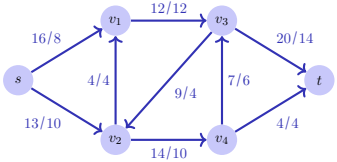
$$\sum_{v \in V} f(u, v) = 0.$$

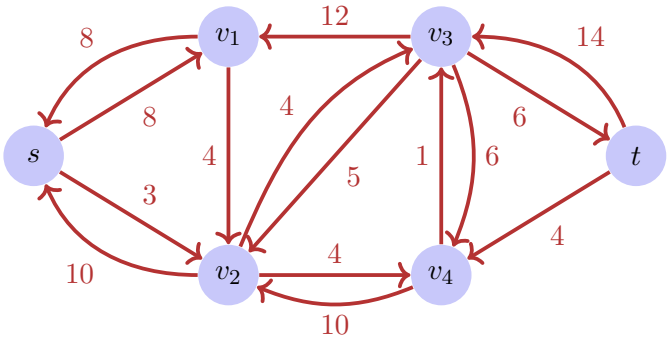


Wert w des Flusses:
$|f| = \sum_{v \in V} f(s, v)$.
Hier $|f| = 18$.

Restnetzwerk

Restnetzwerk G_f gegeben durch alle Kanten mit Restkapazität:





Restnetzwerke haben dieselben Eigenschaften wie Flussnetzwerke, ausser dass antiparallele Kanten zugelassen sind.

Erweiterungspfade

Erweiterungspfad p: einfacher Pfad von s nach t im Restnetzwerk G_f.
Restkapazität $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ Kante in } p\}$

Max-Flow Min-Cut Theorem

Theorem 1

Wenn f ein Fluss in einem Flussnetzwerk $G = (V, E, c)$ mit Quelle s und Senke t is, dann sind folgende Aussagen äquivalent:

1. *f ist ein maximaler Fluss in G*
2. *Das Restnetzwerk G_f enthält keine Erweiterungspfade*
3. *Es gilt $|f| = c(S, T)$ für einen Schnitt (S, T) von G.*

Algorithmus Ford-Fulkerson(G, s, t)

Input: Flussnetzwerk $G = (V, E, c)$
Output: Maximaler Fluss f.

for $(u, v) \in E$ **do**
    $f(u, v) \leftarrow 0$
while Existiert Pfad $p : s \rightsquigarrow t$ im Restnetzwerk G_f **do**
    $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$
    foreach $(u, v) \in p$ **do**
        if $(u, v) \in E$ **then**
            $f(u, v) \leftarrow f(u, v) + c_f(p)$
        else
            $f(v, u) \leftarrow f(u, v) - c_f(p)$

Edmonds-Karp Algorithmus

Wähle in der Ford-Fulkerson-Methode zum Finden eines Pfades in G_f jeweils einen Erweiterungspfad kürzester Länge (z.B. durch Breitensuche).

Theorem 2

Wenn der Edmonds-Karp Algorithmus auf ein ganzzahliges Flussnetzwerk $G = (V, E)$ mit Quelle s und Senke t angewendet wird, dann ist die Gesamtanzahl der durch den Algorithmus angewendete Flusserhöhungen in $\mathcal{O}(|V| \cdot |E|)$

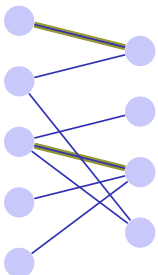
$\Rightarrow$ *Gesamte asymptotische Laufzeit: $\mathcal{O}(|V| \cdot |E|^2)$*

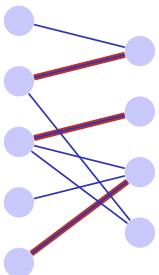
Anwendung: Maximales bipartites Matching

Gegeben: bipartiter ungerichteter Graph $G = (V, E)$.

Matching M: $M \subseteq E$ so dass $|\{m \in M : v \in m\}| \leq 1$ für alle $v \in V$.

Maximales Matching M: Matching M, so dass $|M| \geq |M'|$ für jedes Matching M'.

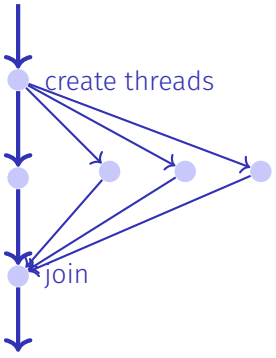




3. C++ Threads

C++11 Threads

```
void hello(int id){
  std::cout << "hello from " << id << "\n";
}

int main(){
  std::vector<std::thread> tv(3);
  int id = 0;
  for (auto & t:tv)
    t = std::thread(hello, ++id);
  std::cout << "hello from main \n";
  for (auto & t:tv)
    t.join();
  return 0;
}
```



Nichtdeterministische Ausführung!

Eine Ausführung:

```
hello from main
hello from 2
hello from 1
hello from 0
```

Andere Ausführung:

```
hello from 1
hello from main
hello from 0
hello from 2
```

Andere Ausführung:

```
hello from main
hello from 0
hello from hello from 1
2
```

Technische Details I

- Beim Erstellen von Threads werden auch Referenzparameter kopiert, ausser man gibt explizit std::ref bei der Konstruktion an.

Technische Details I

- Beim Erstellen von Threads werden auch Referenzparameter kopiert, ausser man gibt explizit std::ref bei der Konstruktion an.

```
void calc( std::vector<int>& very_long_vector ){
  // doing funky stuff with very_long_vector
}
int main(){
  std::vector<int> v( 1000000000 );
  std::thread t1( calc, v );          // bad idea, v is copied
  // here v is unchanged
  std::thread t2( calc, std::ref(v) ); // good idea, v is not copied
  // here v is modified
  std::thread t2( [&v]{calc(v)}; } ); // also good idea
  // here v is modified
  // ...
```

Technische Details II

- Threads können nicht kopiert werden.

Technische Details II

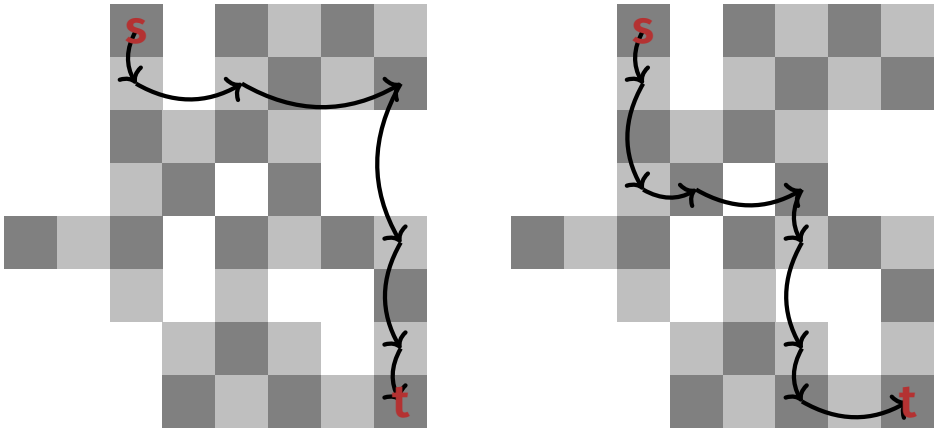
- Threads können nicht kopiert werden.

```
{
  std::thread t1(hello);
  std::thread t2;
  t2 = t1; // compiler error
  t1.join();
}
{
  std::thread t1(hello);
  std::thread t2;
  t2 = std::move(t1); // ok
  t2.join();
}
```

4. Zwei Quiz-Fragen

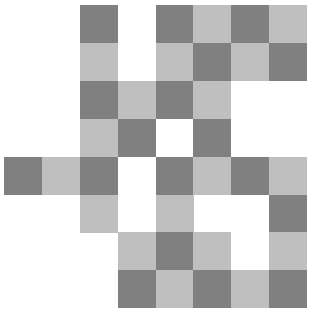
[Exam 2018.01], Aufgaben 4 und 5

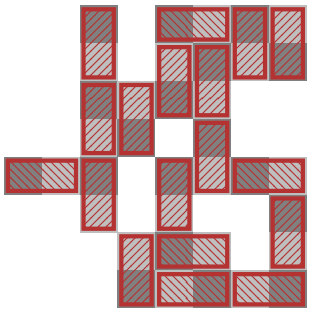
Kürzeste Wege Frage



Wichtigste Frage: Was ist der dazugehörige Zustandsraum?

Max Flow Question





Wichtigste Frage: Wie bildet man das auf ein Max-Flow (Matching) Problem ab?