matching and flow:A matching in a graph is a subset of the edges $M \subseteq E$ of the graph such that each vertex in the graph is incident on at most one edge in M.A maximum matching in a graph is a matching with the largest number of edges among all matchings in the graph. A matching M is called maximal if there is no edge e in the graph such that $M \cup e$ is also a matching. Every maximum matching is also maximal M-augmenting path: Let M be a matching. A path P is called M-augmenting if

- the edges in P are alternatingly in M and $\bar{M}, (\bar{M} = E(G) \backslash M$
- the first and last edges of P are not in M, and
- the first and last vertices, i.e, the endpoints of P are not in M

Augmenting a matching M with an M-augmenting path. Let P be the M-augmenting path. To get a larger matching M'

- If $e \in P \backslash M$, add it to M'
- If $e \in P \cap M$, don't add it to M'
- Add every edge $e \in M \backslash P$ to M'

Claim: M' is a matching Claim: M' is larger than M Lemma: M is maximum $\iff \nexists$ M-augmenting path

Bipartite graphs are those whose vertices can be partitioned into two sets such that no edge of the graph has both its end-points in the same set. Auxillary graph: To create an auxillary graph, direct the edges of the bipartite graph as follows:

- If $e \in M$, direct it from right to left.
- If $e \notin M$, direct it from left to right.

(If we were to start with an unmatched vertex on the right, we would reverse the directions listed above.)

Flow and Cuts Residual Network: Let f be a flow in the network $G = (V, A)$ with capacities c. Define the residual network G_f as: For every arc (v, w) in A:

- If $f(v, w) < c(v, w)$, then (v, w) is a forward arc in G_f of residual capacity $c_f(v, w) = c(v, w) - f(v, w)$
- If $f(v, w) > 0$, then (w, v) is a backward arc in G_f of residual capacity $c_f(w, v) = f(v, w)$.

```
Ford-Fulkerson Algorithm
1: procedure MaxFlow(G, c)
2: f : f(v, w) = 0 for all (v, w) in A
3: Construct residual network Gf
4: while There is a path P from s to t
in Gf do
5: x = min{cf (v, w) | (v, w) in P}
6: for (v, w) in P do
7: if (v, w) is forward arc then
8: f(v, w) = f(v, w) + x
9: else
10: f(v, w) = f(v, w) - x
11: return f
```

Cut:

- $S \cup T = V$
- $S \cap T = \emptyset$
- $s \in S$ and $t \in T$

capacity of a cut: $C(S,T) = \sum_{v \in S, w \in T : (v,w) \in A} c(v, w)$ the flow of a cut is defined likewise.

Lemma: For every s-t cut $(S, T), |f| \leq f(S, T)$

Theorem 3.3, Ford-Fulkerson: The maximum value of flow in a network G is equal to the capacity of a cut in the smallest capacity.

```
Edmonds-Karp Algorithm
1: procedure MaxFlow(G, c)
2: f : f(v, w) = 0 for all (v, w)
3: Construct residual network Gf
4: Find the shortest path P from s
to t in Gf using BFS
5: if P exists then
6: x = min{cf (v, w) | (v, w) in P}
7: for (v, w) in P do
8: if (v, w) is forward arc then
9: f(v, w) = f(v, w) + x
10: else
11: f(v, w) = f(v, w) - x
```

Theorem 3.5 Edmonds-Karp algorithm finds a maximum flow in a given flow network in $O(n \cdot m^2)$ time. m is the amount of edges. Claim 3.6: In every iteration of flow augmentation, the length of the augmenting path monotonically increases. Claim 3.7: Each edge $e \in E(G)$ becomes critical at most $n/2$ times

Amortized Analysis Aggregate method: Averag cost Accounting method: Prove that $\sum_i^n a_i - \sum_i^n t_i \geq 0$ for all n. Potential function: Phi that maps the configuration of the data structure to real number. $Phi_1 > \Phi_0$ for all i and $a_i = t_i + \Phi_i - \Phi_{i-1}$

Fibonacci heaps Heap: insert: $O(\log(n))$, find-min: $O(1)$, Extract-min $O(\log(n))$.

order of a node: Amount of childeren, order of a tree: order of root, Insert(H,k): $O(1)$, Union(H1,H2): $O(1)$,

```
Decrease-Key(heap H, node x, key k)
Change key value of the node x into k
if k is smaller than key x's parent p
  Cut x from p
  Cascading-Cut(H, p)
Update min(H)

Cascading-Cut(heap H, node p)
if p is not marked then
  Mark p
else
  Cut p from its parent q, unmark p
  cas(H, q)
```

Decrease-Key can take $O(m) = \max$ height of tree time.

```
Extract-Min(heap H)
Delete the min node y from H
for each child z of y do
  The subtree rooted on z becomes a
  new tree in H, unmark z
  Consolidation(H)
Update min(H)

Consolidation(heap H)
for i = 0 to Max-degree(H) do
  Pair the trees with order and make
  the one with the larger root-key value
  a new child of the other one
```

Extract-min takes $O(\text{Max-degree} + t)$ time and t is the number of trees.

```
Delete(heap H, node x)
Decrease-Key(H, x, -infty)
Extract-Min(H)
```

amortized Decrease-key is $O(1)$, Extract-min is $O(\text{Max-Degree})$.

Minimum Spanning Trees: In the minimum spanning trees problem, we are given a weighted undirected graph $G = (V, E, w)$. The goal is to find a subset $E' \subseteq E$ of edges that connects all vertices in V such that the total weight of the edges in E' is minimized.

lemma If the weights are unique the minimum spanning trees are also unique. A light edge for a component C is an edge (u, v) such that there is exactly one endpoint in C and has the minimum weight.

```
Boruvka's algorithm
while F is not a spanning tree do
  Add all light edges
```

Boruvka is $O(|E| \log |V|)$

```
Prim's algorithm
Start at a singleton T
Repeatedly adding the light edge of T
to F
```

$O(|E|\cdot\log|E|) = O(|E|log|V|)$ and using fibbonacci heaps $O(|E| + |V|\log|V|)$

Kruskal(G):

```
Scan all edges by increasing weight
if an edge is light for some
component, add it to F
```

Total time complexity is dominated by time of sorting the edges

Approximation Algorithms

approximation ratio: $\max_I\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)}\}$ (minimization) and $\max_I\{\frac{\mathrm{ALG}(I)}{\mathrm{OPT}(I)}\}$ (maximization) it is an α-approximation if the approximation ratio is smaller than α.

String Matching:

Naïef

```
NaiveMatch(A,P)
  for k:=0 to n-m
    if Checkmatch(A,k,P)
      output k
  return
Checkmatch(A,k,P)
  for i :=0 to m-1
    if A[k+i] \neq P[i]
      return false
  return true
```

looptijd $O(nm)$ en gemmiddelde looptijd is $O(1)$.

Rabin-Karp

```
RK3(A,P)
  ph := String2Int(P) % q
  sh := String2Int(A[0...m-1]) % q
  for k := 0 to n-m
    if (ph == sh)
      output k
    if k < n-m
      sh := A[k+m] + sh × 26 -
        A[k] × 26^m % q
  return
```

Gerandomiseerde algoritmes:

Las Vegas: Antwoord altijd correct maar rekentijd is niet te voorspellen

Monte Carlo: Antwoord niet altijd correct maar rekentijd is te voorspellen Las Vegas kan Monte Carlo worden en Las Vegas kan Monte Carlo worden mits er gecontroleerd kan worden of het antwoord correct is.

Geometrische verdeling: $P(X = k) = (1-p)^{k-1}p$ en $E[X] = 1/p$.