Contents

Chapter 1

Base concepts

1.1 Density

$$\frac{\text{edges of graph g}}{\text{num edges complete graph with n nodes}}$$

1.2 BFS

- Each node has three states: unvisited, visited, explored
- When a node is visited all non-visited neighbours are added into a FIFO queque, then the node is set as explored
- First, the starting node is explored
- While the queque is not empty:
  1. Pick the first node in the queque
  2. Visit the node
- Each time a node is inserted into the queue, we also associate to this node its parent
- Property of BFS in unweighted graphs: the nodes at level i in the BFS tree are also the nodes at distance i in the graph
- When the graph is undirected it also allow us to know if the graph is connected (check the bfs tree contains all the nodes)
- You can know also the connected components of the graph doing a BFS from the non visited nodes while all are not explored
- Time complexity: each edge is considered two times: one when an extreme is visited and another when the other extreme is visited

1.3 DFS

- Same of BFS but instead of a FIFO queque use a LIFO queque or recursion
- Property of DFS: the dfs not include only the edges that link a node x to its ancestor
- With the latter property we can use the DFS to check if a graph g contains a cycle

1.4 Terms

1.4.1 Subgraph

Subset of nodes of a graph g and all edges that links these nodes

1.4.2 Connected components

Maximal subgraph contained in the graph g where cannot be inserted more nodes

1.4.3 Graph connected

A graph is connected if exists only one connected component

1.4.4 Strongly connected components (direct graph)

A direct graph is strongly connected contains only one maximal subgraph where exists a path from x to y *and* from y to x

Chapter 2

Algorithm for SCC in a direct graph

The problem is that when we perform the DFSs we can visit different connected components.

2.1 DAG

A DAG is a Directed Acyclic Graph. To check if a graph is a DAG with a DFS (see above).

2.2 Topologicals ordering

With a DAG you can sort the nodes in a *topological ordering* with some alters to the DFS algorithm:

- We denote as s(x) the time when a node x is visited and as f(x) the time when the invocation of dfs on the node ends
- When a node is visited the time indicator is increased and the the invocation ends the *time* counter is increased then assigned as finished time
- At the end we can sort the nodes by finishing time
- If all nodes are not reached, remove all the reached nodes from the graph and then start a new DFS while all nodes are not removed

We demonstrate the following property: given a DAG, if exists an edge between a pair of nodes x and y $(x \to y)$ then $f(x) > f(y)$:
There are two possible option when the DFS is invoked with argument x:

1. y is explored or y is not visited, otherwise is an ancestor edge but this not respect the hypotheses that g is a DAG. In the first case y is yet explored so it have assigned the finished time while x not so is surely more than f(y).

2. In the second case x is visited but the finished time is assigned after all the neighbours are explored (included y) so we return to the latter case

2.3 Component Graph

A component graph is a graph where the nodes are the SCC (strongly connected components) of g and exists an edge between c_1 and c_2 if exists one of the two options:

- exists an edge from a node in c_1 to c_2
- exists an edge from a node in c_2 to c_1

Any component graph is a DAG: for absurd if exists a bidirectional edge between c_1 and c_2 so the two nodes are not maximal so c_1 and c_2 should be an unique node

2.4 Graph transpose

The transpose G^T of a graph G is a graph where the edge directions are inverted. E.G. $(a, b) \in G \rightarrow (b, a) \in G^T$

2.5 Kosaraju-Sharir algorithm

Base intuition: if we start the DFS in the last SCC of component graph (the SCC with only entering edge) then the DFS tree will include only the nodes of the SCC. Then remove the explored nodes from the graph and restart the DFS until all nodes are removed.
Problem: We don't know the SCCs, that is what we want to find!
Let's give some definitions:

- $f(c)$: largest finished time of a node in a connected component c
- $s(c)$: smallest finished time of a node in a connected component c

Let's prove the following assert: given c_1 and c_2 two connected components such as exists the edge (u, v) where $u \in c_1$ and $v \in c_2$, then $f(c_1) > f(c_2)$.

2.5.1 Demonstration

We distinguish two cases:

- $s(c_1) < s(c_2)$: Given x the first node explored in c_1 then the DFS will find the edge (u, v) where $u \in c_1$ and $v \in c_2$ that continue the DFS in c_2. So it's obvious that for, how works the DFS and how are assigned the finishing times, the finishing time are first assigned to c_2 because the recursive calls through nodes in c_2 ends until reaches the node v that continue the assignment of the finishing time in c_1.
  In conclusion we can infer the following dis-equation:

$$f(x) = f(c_1) > f(u) > f(v) = f(c_2)$$

- $s(c_1) > s(c_2)$: In this case we start in c_2 but we can infer that not exists a path from c_2 to c_1 because the component graph is a DAG and exists for hypothesis the edge (u, v) where $u \in c_1$ and $v \in c_2$. So we can assert that the current DFS will not explore c_1 but a successive DFS will explore c_1 so $f(c_1) > f(c_2)$

So we have demonstrate the latter assert.
From this proposition we can infer that if we take the node with the largest finished time we are surely in a SCC with only outgoing edges in the component graph! So if we do a DFS in the transpose graph we explore only the nodes in its connected component.
In conclusion the Kosaraju-Sharir algorithm is the following:

1. Do DFS until all nodes have finishing time
2. Transpose the graph
3. Do DFS starting from the node with largest finishing. This is a connected component
4. Remove all node explored
5. Go to 3 until the graph have at least one node

Complexity: $O(m)$

2.6 CC in social networks

2.6.1 Giant component

A giant component is a (strongly) connected component that contains a constant fraction of the nodes

2.6.2 Giant components in SN

The giant components is SN are very commons.
In the web directed graph a common phenomenon is the *bowtie phenomenon*: it says that we can classify the nodes in the following groups:

- SCC: Nodes contained in a big giant strongly connected component
- IN: the set of nodes that can only reach the SCC
- OUT: the set of nodes that can be reached from the SCC
- *tendrils*: nodes that can be reached only from IN but are not part of SCC and nodes that can reach OUT but they are not part of SCC
- *disconnected*: nodes that can't reach the SCC also if the graph is undirected

2.7 Shortest path

In the case of undirected edges, the shortest path between two nodes x and y is the minimum numbers of edges needed to reach y from x. It can be computed performing a BFS starting from x.
There are two variants of this problem:

- SSSP (Single Source Shortest Path) that consists to find all the shortest path from a specific source
- APSP (All Pairs Shortest Path) that consists in find all the shortest path between all the pairs of nodes

Chapter 3

Dijkstra's algorithm

The Dijkstra's algorithm solves the SSSP problem in the case of weighted edges when the weights are positive. It is a *greedy* algorithm, so it take the best decision only considering the current state and not the overall procedure.
At the beginning choose a starting node s and then initialize a distance array $dist$ with $dist[s] = 0$ and $dist[x] = \infty \; \forall x \neq s$
The greedy step computed by Dijkstra is the following:

1. choose a node x not yet picked and that has the minimum $dist[x]$
2. for each neighbour $y \in N(x)$ set $dist[y] = \min(dist[y],\ dist[x] + d(x, y))$

3.1 Demonstration of the correctness of Dijkstra's algorithm

To demonstrate the Dijkstra's algorithm we have to prove that given x the node chosen at the step i then $dist[x]$ must be the minimum from s to x.
Let's introduce some terms: A_i is the set of nodes picked at the step i and $d_i(x)$ the shortest path only through the nodes in A_i.
First, we prove by induction that $dist[x]$ at the step i is equal to $d_i(x)$:

- *base case (i = 1)* this case is simple: the shortest path from s to the neighbours y is simply $d(s, y) = d_1(y)$
- *inductive step* $(k \to k + 1)$ Let's suppose that the proof is valid until k so we have that $dist[x] = d_k(n) \; \forall n \in A_k$. Given x the node picked at the step k+1 and $\pi_{k+1}(y)$ the shortest path from s to y through the nodes in $A_{k+1} = A_k \cup \{x\}$, for each edge (x, y) we have the two following cases about y:
  - $x \notin \pi_{k+1}(y)$ it means that x is not in the shortest path from s to y not pass through x so we have that $\pi_k(y) = \pi_{k+1}(y)$
  - $x \in \pi_{k+1}(y)$ it means that the shortest path to y pass through x so we have that $d_{k+1}(y) = d_k(x) + w(x, y)$

Now we show that *if x is the node chosen at the step i, a shortest path from s to x pass only through* A_i. We demonstrate for absurd, so given y the first node

not in A_i that is in the shortest path of x from s we can infer the following propositions:

- Since Dijkstra choose the node with minimum distance, $d_i(x) = dist[x] \leq dist[y] = d_i(y)$
- $d(s, y) < d(s, x)$ because all weights are positive and y is in the shortest path of x
- $d(s, x) < d_i(x)$ since we are assuming that a shortest path from s to x not pass through nodes in A_i
- $d(s, y) = d_i(y)$ since y is the first node outside of A_i (see above demonstration)

With the latter propositions we have an absurd because:
$d(s, y) < d(s, x) < d_i(x) \leq d_i(y) = d(s, y)$

3.2 Complexity of Dijkstra's algorithm

The complexity depends on the data structure used to represent *dist*.
There are three types of time which complexity of Dijkstra's algorithm depends on:

- t_i: The time to initialize *dist*
- t_m Time to find the minimum in *dist*
- t_u Time to update a value in *dist*

Suppose that *dist* is an array, we have that:

- $t_i = O(n)$ (Initialize n values)
- $t_m = O(n)$ (Sequential search)
- $t_u = O(1)$ (Set a value in an array is immediate)

so the time complexity of Dijkstra's algorithm is

$$O(t_i + nt_m + mt_u) = O(n + n \cdot n + m) = O(n^2)$$

But if *dist* is an heap we have:

- $t_i = O(n \log(n))$ (Initialize n values and sort by heap property)
- $t_m = O(\log(n))$ (Thanks to heap property)
- $t_u = O(\log(n))$ (Check heap property)

in this case the time complexity of Dijkstra's algorithm is

$$O(t_i + nt_m + mt_u) = O(n \log(n) + n \log(n) + m \log(n)) = O((n + m) \log(n))$$

which is good if the graph is sparse

Chapter 4

Floyd-Warshall algorithm

This algorithm solve the APSP problem with edges having negative edges, but that not contains a negative cycle because if it exists the shortest path should pass through the cycle infinitely.

4.1 Dynamic programming technique

The dynamic programming technique is used in the Floyd-Warshall algorithm and consists of saving in a data structure the results of sub-problems of a generic algorithm to use them when a call with the same input happens.

4.2 The algorithm

Let's define the following terms:

- $A_0 = \emptyset$ and $A_i = \{1 \dots i\}$
- $\pi_i(x, y)$ the shortest path from x to y through the nodes contained in A_i (if x cannot reach y through A_i then $\pi_i(x, y) = \infty$)
- $d_i(x, y)$ the length of $\pi_i(x, y)$

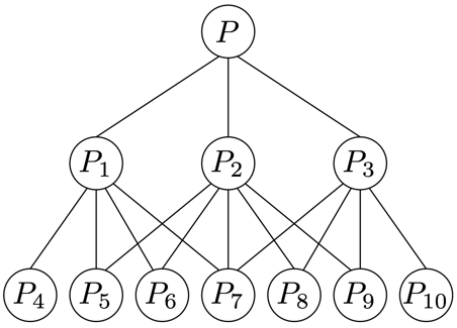

Figure 4.1: Problem tree representation when it's useful the dynamic programming technique

- Note that for $i = 0$ we have $d_0(x, y) = w(x, y)$ if exists an edge between x and y else ∞

For $1 \leq i \leq n$ we can calculate $d_i(x, y)$ by distinguish between the two following cases:

- $d_i(x, y)$ pass through A_i but not by i: in this case $d_i(x, y) = d_{i-1}(x, y)$
- $d_i(x, y)$ pass through A_i and by i: in this case $d_i(x, y) = d_{i-1}(x, i) + d_{i-1}(i, y)$

4.2.1 Complexity

Space

This algorithm need a tri-dimensional array *dist* where the first dimension is used to memorize each step and the other two as matrix of the distance so the space complexity is $O(n^3)$ but it can save some space memorizing only two matrix: the matrix of distances at step i and $i - 1$, then the space complexity is $O(n^2)$

Time

The time complexity is $O(n^3)$ because for n times the algorithm must update the distance matrix ($O(n^2)$)

Chapter 5

Six degree of separation

5.1 The experiment

An experiment conducted by *Stanley Milgram* says that, in 1990, the average distance between two persons in 6. He give to 296 people a letter that must be sent to the recipient and they can do two things:

- if it know the recipient: send directly the letter
- if it don't know the recipient: send the letter to the most probable person that maybe know the recipient

64 of 296 letters are delivered to the recipient and the average distance between the sender and the recipient is measured to be 6

5.2 Small world hypothesis

The experiment of *Stanley Milgram* aims to verify that is called in SNA the *small world hypothesis*, that means calculate the average shortest path across a network. In a social network, the average shortest path means calculate APSP (All Pair Shortest Path) then computationally is very often prohibitive (at least $O(n^2)$ and in the worst case $O(n^3)$).
To avoid the high cost, we calculate an approximation of average shortest path using a *sample* of nodes, but we need to measure the error of this approximation in order to use it rightfully.

5.3 Approximation of average shortest path

5.3.1 Markov's inequality

The Markov's inequality is:

$$Pr\left[X \geq t\right] \leq \frac{\mathbb{E}\left[X\right]}{t}$$

The demonstration is simple:

$$\begin{aligned}\mathbb{E}[X] &= \sum_u u \cdot P(X = u) \\ &\geq \sum_{u \geq t} u \cdot P(X = u) \\ &\geq t \cdot \sum_{u \geq t} P(X = u) = t \cdot P(X \geq t)\end{aligned}$$

5.3.2 The Chernoff's bounding method

The Chernoff?s bounding method says that if X is a random variable and $\forall t > 0$ we can assert:

$$P(X \geq t) \leq \min_{s>0} e^{-st}\, \mathbb{E}\left[e^{sX}\right]$$

Using the Markov's inequality we can demonstrate the Chernoff?s bounding method:

$$\begin{aligned}P(X \geq t) &= P(sX \geq st) \\ &= P(e^{sX} \geq e^{st}) \leq e^{-st}\, \mathbb{E}\left[e^{sX}\right]\end{aligned}$$

5.3.3 Assert to demonstrate

If X is a random variable such that $\mathbb{E}[X] = 0$ and $a \leq X \leq b$, for any $s > 0$ we have that:

$$\mathbb{E}\left[e^{sX}\right] \leq e^{\dfrac{s^2(b-a)^2}{8}}$$

Since e^{sx} is convex we have that:

$$e^{sX} \leq \frac{X-a}{b-a} e^{sb} + \frac{b-X}{b-a} e^{sa}$$

For hypothesis $\mathbb{E}[X] = 0$ so we have that:

$$\begin{aligned}\mathbb{E}\left[e^{sX}\right] &\leq \mathbb{E}\left[\frac{X-a}{b-a} e^{sb} + \frac{b-X}{b-a} e^{sa}\right] \\ &= -\frac{a}{b-a} e^{sb} + \frac{b}{b-a} e^{sa}\end{aligned}$$

Now we define $p = \dfrac{b}{b-a}$ and $u = s(b-a)$, then we rewrite the latter formula in a log scale with the new terms:

$$\begin{aligned}\log\left(\frac{b}{b-a} e^{sa} - \frac{a}{b-a} e^{sb}\right) &= sa + \log\left(p + (1-p)e^{s(b-a)}\right) \\ &= (p-1)u + \log(p + (1-p)e^u) = \varphi(u)\end{aligned}$$

We apply the Taylor theorem to $\varphi(u)$:

$$\varphi(u) = \varphi(0) + \varphi'(0)u + \frac{1}{2}\varphi''(\xi)u^2$$

Since $\varphi(0)$ and $\varphi'(0)$ are 0 and $\varphi''(\xi) \leq \frac{1}{4}$ we have that:

$$\begin{aligned}\varphi(u) &= \frac{1}{2}\varphi''(\xi)u^2 \\ &\leq \frac{u^2}{8} = \frac{s^2(b-a)^2}{8}\end{aligned}$$

5.3.4 The Hoeffding's inequality

Given k independent random variables $X_1 \dots X_k$ such that $a_i \leq X_i \leq b_i$ and $X = \sum_i X_i$ for any $t > 0$ we have that:

$$P(X - \mathbb{E}[X] \geq T) \leq e^{\frac{-2t^2}{\sum_{i=1}^{k}(b_i - a_i)^2}}$$

If we apply the *Chernoff's bounding method* to $X - \mathbb{E}[X]$ we have that

$$\begin{aligned}P(X - \mathbb{E}[X] \geq t) &\leq \min_{s>0} e^{-st}\,\mathbb{E}\left[e^{s(X-\mathbb{E}[X])}\right] \\ &= \min_{s>0} e^{-st} \prod_{i=0}^{k} \mathbb{E}\left[e^{s(X_i-\mathbb{E}[X_i])}\right] \\ &= \min_{s>0} e^{-st} \prod_{i=0}^{k} e^{\frac{s^2(b_i-a_i)^2}{8}} \\ &= \min_{s>0} e^{-st} e^{\frac{s^2\sum_i(b_i-a_i)^2}{8}} \\ &= \min_{s>0} e^{-st+\frac{s^2\sum_i(b_i-a_i)^2}{8}}\end{aligned}$$

In conclusion we have that $P(X - \mathbb{E}[X] \geq t) \leq \min_{s>0} e^{-st+\frac{s^2\sum_i(b_i-a_i)^2}{8}}$. Now we want to minimize the exponent of e for the Hoeffding's inequality, so we derive $-st + \frac{s^2\sum_i(b_i-a_i)^2}{8}$ by s and we obtain a minimum for $s =$

$\frac{4t}{\sum_{i=1}^{k}(b_i-a_i)^2}$. If we substitute we have:

$$
\begin{aligned}
P(X-\mathbb{E}[X]\geq t) &\leq \min_{s>0} e^{-st+\frac{s^2\sum_i(b_i-a_i)^2}{8}} \\
&= e^{-\frac{4t}{\sum_{i=1}^{k}(b_i-a_i)^2}\cdot t+\frac{\left(\frac{4t}{\sum_{i=1}^{k}(b_i-a_i)^2}\right)^2\sum_i(b_i-a_i)^2}{8}} \\
&= e^{-\frac{4t^2}{\sum_{i=1}^{k}(b_i-a_i)^2}+\frac{\frac{(4t)^2}{\sum_{i=1}^{k}(b_i-a_i)^2}}{8}} \\
&= e^{\frac{-4t^2\cdot 8+\frac{(4t)^2}{\sum_{i=1}^{k}(b_i-a_i)^2}\cdot\sum_{i=1}^{k}(b_i-a_i)^2}{\sum_{i=1}^{k}(b_i-a_i)^2\cdot 8}} \\
&= e^{\frac{(4t)^2(-2+1)}{\sum_{i=1}^{k}(b_i-a_i)^2\cdot 8}} \\
&= e^{-\frac{16t^2}{\sum_{i=1}^{k}(b_i-a_i)^2\cdot 8}} = e^{-\frac{2t^2}{\sum_{i=1}^{k}(b_i-a_i)^2}}
\end{aligned}
$$

The Hoeffding's inequality works also when random variables are in the form $-X_1 \ldots - X_k$ because

$$
\begin{aligned}
P(X-\mathbb{E}[X]\leq -t) &\leq e^{-\frac{2t^2}{\sum_{i=1}^{k}(b_i-a_i)^2}} \\
P(|X-\mathbb{E}[X]|\geq t) &\leq 2e^{-\frac{2t^2}{\sum_{i=1}^{k}(b_i-a_i)^2}}
\end{aligned}
$$

5.4 Finally the approximation

We can define grthe average shortest path at distance h as

$$N_h = \frac{|\{(u,v)\in V\times V : d(u,v)=h\}|}{n(n-1)}$$

instead of V if we sample from V a subset $U=\{u_1\ldots u_k\}\subseteq V$ we have

$$N_h(U) = \frac{|\{(u,v)\in U\times V : d(u,v)=h\}|}{|U|(n-1)} = \frac{\sum_{i=0}^{k} N_h(\{u_i\})}{k}$$

The expected value of $N_h(\{u_i\})$ is

$$\begin{aligned}\mathbb{E}\left[N_h(\{u_i\})\right] &= \frac{1}{n}\sum_{v\in V} N_h(\{v\}) \\ &= N_h(V) = N_h\end{aligned}$$

then the expected value of set $U \subseteq V$ is

$$\begin{aligned}\mathbb{E}\left[N_h(U)\right] &= \mathbb{E}\left[\frac{\sum_{i=1}^{k} N_h(\{u_i\})}{k}\right] \\ &= \frac{\sum_{i=1}^{k} \mathbb{E}\left[N_h(\{u_i\}]\right)}{k} \\ &= \frac{\sum_{i=1}^{k} N_h}{k} \\ &= \frac{kN_h}{k} = N_h\end{aligned}$$

Because N_h is a ratio, we can assert that $0 \leq N_h \leq 1$ so $\forall\ i\ N_h(\{u_i\})$ we can define $a_i = 0$ and $b_i = 1$. So we can apply the Hoeffding's inequality to N_h to obtain this approximation:

$$P\left(\left|\frac{\sum_{i=1}^{k} N_h(\{u_i\})}{k} - N_h\right| \geq t\right) \leq 2e^{-2t/k}$$

If we choose the sample size $k = \frac{2}{\alpha}t^2\dfrac{1}{\ln(n)}$ we obtain the following bound

$$\begin{aligned}2e^{\frac{-2t^2\ln(n)}{\frac{2}{\alpha}t^2}} &= 2e^{-\alpha\cdot\ln(n)} \\ &= 2e^{\ln(n^{-\alpha})} \\ &= 2n^{-\alpha}\end{aligned}$$

Then the upper bound is dependent from α (often $\alpha = 2$)

Chapter 6

Compute the diameter

6.1 Definition of diameter

The diameter is the maximum distance between two vertices

6.2 A simple approach

In the general case the best approach is calculate APSP (All Pairs Shortest Path) then find the maximum distance, that has a time complexity of $O(n^{2.38})$ where the 2.38 derive from some optimization on matrix calculation(see below) for dense graph while $O(mn)$ for sparse graph. The problem with this simple approach is that is not usable in real-world graphs because contains millions of nodes and edges.

6.3 Lower bound for time complexity of diameter computation

6.3.1 SETH Hypothesis

The Strong Exponential Time Hypothesis (SETH) says that there not exists an algorithm that solves k-SAT in less than $O((2-\epsilon)^n)$ where $\epsilon > 0$

6.3.2 Reduction between two problems

Given two problems A and B, the relative sets of instances of the problem I_A and I_B and the solutions set $A(x)$ and $B(x)$ of a instance x we can say that A is reducible to B if exists two functions f and g where given $x \in I_A$, $x' = f(x) \in I_B$, $y' = B(x')$ and $g(y', x) \in A(x)$

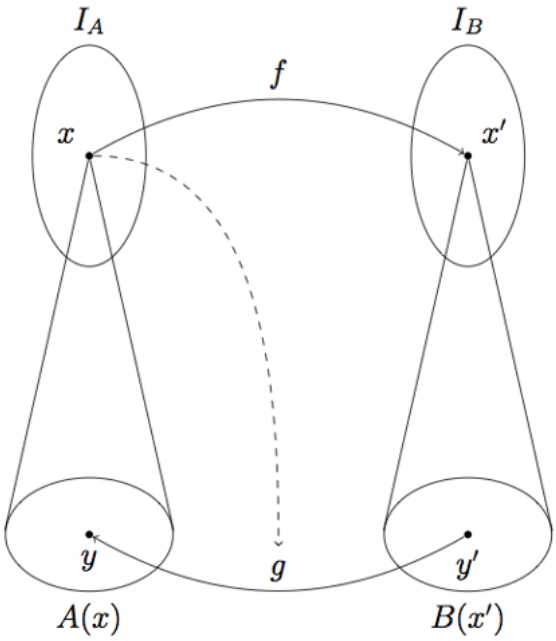

Figure 6.1: Graphical representation of problem reduction

6.3.3 K-SAT*

A variant of the k-sat problem is the $K - SAT^*$ where between them change only the input: The $K - SAT^*$ receive two sets of assignments X and Y of size $n = 2^{\frac{m}{2}}$ to respectively the first half and second half of variables where m is the number of variables and a set of clauses C.
In order to find an assignment that satisfy C we combine each first-half assignment of X to Y, then try to check if applied to the set of clauses returns *true*.
So the complexity of the algorithm is virtually $O(n^2)$ on the input size, but if we substitute n in function of m (the number of variables) we obtain $O((2^{\frac{m}{2}})^2) = O(2^m)$. Also this problem cannot have the complexity $O(n^{2-\varepsilon})$ unless SETH is false. Remember that SAT clauses are in CNF form that means:
$(X_1 \vee X_2 \vee \cdots \vee X_n) \wedge (Y_1 \vee X_2 \vee \cdots \vee X_n) \wedge (X_1 \vee Y_2 \vee \cdots \vee X_n) \wedge (Y_1 \vee Y_2 \vee \cdots \vee X_n) \wedge \cdots \wedge (Y_1 \vee Y_2 \vee \cdots \vee Y_n)$.
To resume in words, in CNF variables in the clause are concatenated with OR operator while the clauses are concatenated with AND operator

6.3.4 Disjoint set problem

Given a set of sets C, the solution is 1 when exists two sets $A, B \in C$ such that $A \cap B = \emptyset$. The complexity of this algorithm is $O(|C|^2)$.

6.3.5 Reduction from K-SAT* to disjoint set

Given the set of all clauses C and X,Y respectively the assignment to the first and second half of variables, we define the collection $S = S_1 \cup S_2$ as follow:

$$S_1 := \{\{t_1\} \cup \{c : x \in X \not\models c \ \forall c \in C\}\}$$

$$S_2 := \{\{t_2\} \cup \{c : y \in Y \not\models c \ \forall c \in C\}\}$$

t_1 and t_2 are only tokens to avoid the empty intersection between set from the same assignment.
To resume, if exists a good assignment in $K - SAT^*$ then in disjoint sets should exists an empty intersection between two sets, one in S_1 and another in S_2.
To explain the reduction from $K - SAT^*$ to disjoin set, a good assignment should satisfy all the clauses in K-SAT* and this thing is transposed in disjoint set with the intersection operator: given $s_1 \in S_1$ and $s_2 \in S_2$ if $s_1 \cap s_2 \neq \emptyset$ then exists a clause not satisfied both from X and Y so the clause if false, then is not a good assignment while a good assignment should satisfy all the clauses so it should not have any clause in the intersection.

6.3.6 From disjoint set to diameter computation

Given in input a set of sets C and the variables X we can build a clique of the size of X, then add a node for each subset in C and add an edge between the node $c_i \in C$ and x_j if x_j appears in the set c_i.
Then we can interpreter the diameter between c_i and c_j in the following mode :

- $diameter(c_i, c_j) = 2 \rightarrow$ exists an intersection between c_i and c_j
- $diameter(c_i, c_j) = 3 \rightarrow$ not exists an intersection between c_i and c_j

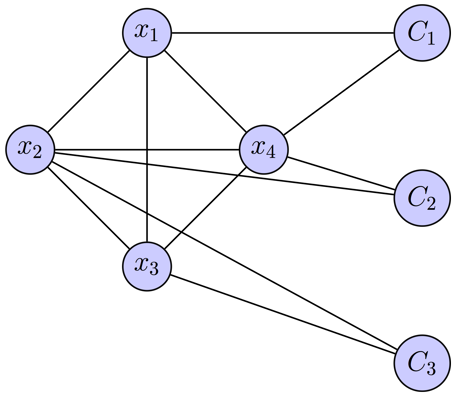

Figure 6.2: Graphical representation of reduction from disjoint sets to diameter computation

6.3.7 Complexity of diameter computation

With the above reductions we have demonstrated that the complexity of diameter computation cannot be in the form $O(n^{2-\varepsilon})$ unless SETH is false, so the complexity of diameter cannot be lower than $O(n^2)$

6.4 Heuristic for computing the diameter

6.4.1 BFS and diameter

The height of BFS tree is a lower bound for computing the diameter. We can use this value also as approximation of the diameter doing many BFS from a random vertex. Below we present two methods to approximate the diameter with BFS. This approximations works well in various types of graphs (including social networks) but in other types not, for example road networks.

6.4.2 2-SWEEP

1. Pick a random vertex r
2. Do a BFS from r
3. Pick x, one of the farther vertexes from r
4. Do a BFS from x and return the height of the BFS tree as diameter

Complexity: $2 \cdot m$

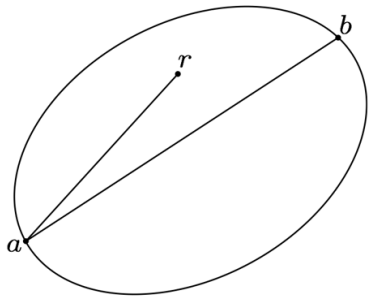

Figure 6.3: 2-SWEEP graphical representation

6.4.3 4-SWEEP

1. Do a 2-SWEEP
2. Pick one of the middle vertexes in the longest path of the 2-SWEEP
3. Do a 2-SWEEP from that vertex

Complexity: $4 \cdot m$

6.5 Exact heuristic of diameter

6.5.1 Eccentricity

The eccentricity of a vertex $ecc(v)$ is the maximum distance between v and any other node.. In formula it is $ecc(v) = \max_{u \in V} d(u, v)$
We can express the diameter in terms of eccentricity: it is the maximum eccentricity among all nodes, or in formula $diameter(G) = \max_{v \in V} ecc(v)$

6.5.2 How heuristic works and complexity

The intent of this exact heuristic is to find the maximum eccentricity among all nodes in order to return the diameter but the worst case doesn't change: it is always $O(nm)$ so the reduction from $K - SAT^*$ is valid but in the mean case this heuristic should be very effective in terms of time cost. For this purpose we use the concept of eccentricity, some definitions and a demonstration.

6.5.3 Some terms

- $F(u) = \{v|d(u,v) = ecc(u)\}$ so the set of nodes at maximum distance from u
- $F_i(u) = \{v|d(u,v) = i\}$ so the nodes at distance i from u (e.g. $F_{ecc(u)(u)=F(u)}$ and $F_1(u) = Neightbours(u)$)
- $B_i(u) = \max_{z \in F_i(u)} ecc(z)$ so the maximum eccentricity between nodes at distance i from u

6.5.4 To the upper bound

Let's demonstrate the following assert: For any $1 \leq i \leq ecc(u)$ and $1 \leq k < i$ and for any $x \in F_{i-k)}(u)$ such that $ecc(x) > 2(i-1)$ there exists $y \in F_j(u)$ such that $d(x,y) = ecc(x)$ with $j \geq i$. Let's first demonstrate that $j \geq i$:
First, we can do some observations:

- for $x \in F_i(u)$ or $y \in F_i(u)$ we have that $d(x,y) < B_i(u)$ because $d(x,y) \leq \min\{ecc(x), ecc(y)\} \leq B_i(u)$. Remember that $B_i(u) = max_{x \in F_i(u)} ecc(x)$
- for any $1 \leq i,j \leq ecc(u)$ and $\forall x \in F_i(u), y \in F_j(u)$ we have $d(x,y) \leq i + j \leq 2\max\{i,j\}$

With this observations we can demonstrate that $j \geq i$. Suppose for absurd that $i > j$, $x \in F_{i-k}(u)$, $ecc(x) > 2(i-1)$ and $y_x \in F_j(u)$ at distance $ecc(x)$ from x we have that:

$$2(i-1) < ecc(x) = d(x,y_x) \leq 2\max\{i-k, j\} \leq 2\max\{i-k, i-1\} = 2(i-1)$$

which is a contradiction so $j \geq i$.
In words, it means that if a node has a eccentricity more than $2(i-1)$ and stay above the level i then a node at maximum distance from x stay below x in the *BFS tree* of u.
Now let's define lb as the maximum eccentricity below the level i or in then we have an upper bound on $x \in F_i(u)$ that is $ecc(x) \leq \max\{lb, 2(i-1)\}$. We can have two cases about $ecc(x)$ that are:

- $ecc(x) \leq 2(i-1) \rightarrow$ is true that $ecc(x) \leq \max\{lb, 2(i-1)\}$
- $ecc(x) > 2(i-1) \rightarrow$ in this case exists a node y below x that has eccentricity greatest or equal than x. In the latter assert we have demonstrate that if a node has eccentricity more than *2(i-1)* above the level i then exists a node y below the level i such that $d(x,y) = ecc(x)$. This imply that the eccentricity of y is at minimum $d(x,y)$ but can be also more so we can assert that $ecc(y) \geq ecc(x)$. In conclusion $ecc(x) \leq ecc(y) \leq lb$, so the dis-equation $ecc(x) \leq \max 2(i-1), lb$ is respected

6.5.5 The algorithm

With the upper bound $\max\{2(i-1), lb\}$ we can assert that if $M > 2(i-1)$ below the level i doesn't exists a node with eccentricity more than lb so we can return lb. We can schematize the algorithm in the following steps:

1. Do a BFS from a node u
2. Set $i = ecc(u)$ and $M = B_i(u)$
3. If $M > 2(i-1)$ return M; else set $i = i-1$ and $M = \max\{M, B_i(u)\}$ then repeat this step while not return a value

Chapter 7

Power law phenomenon

7.1 The power law

Initially the simplest distribution assumed for the degree distribution on a social network is a *Normal Gaussian*. But it is verified that the distribution of the degree is proportional to $f(k) = \frac{1}{k^\beta}$ with β close to 2. A function $f(k)$ that decrease as increase k is called *power law*

7.1.1 Difference between a normal distribution and a power law

The normal distribution is a representation of a balanced phenomenon, where in the mean there is the maximum frequency and in the slopes decrease the frequency.
In the power law instead is a representation of a imbalanced phenomenon where, with small values of x we have high frequency and with high values x we have a small frequency. Some example of power law phenomenon are the richness, number of relations in a social network.

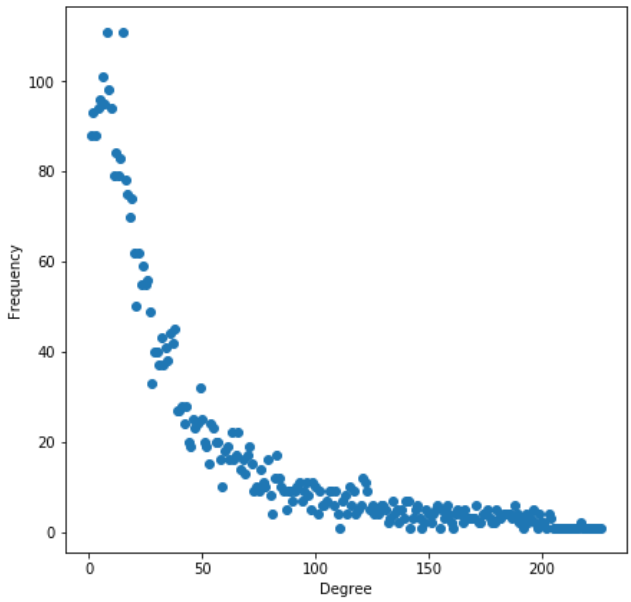

Figure 7.1: Degree distribution on a Facebook social network

7.1.2 Check if a degree distribution is a power law

In order to check if a degree distribution is a power law it must be in the form $f(k) = \alpha \cdot \frac{1}{k^{\beta}}$. We can do a linear regression in log-log scale of $f(k)$ that is

$$\log(f(k)) = \log\left(\frac{\alpha}{k^{\beta}}\right) = \log(\alpha) - \beta \cdot \log(k)$$

and should be a straight line to represent a power law

7.2 Generative model of social network based on degree distributions

In this section we study some generative models based on degree distribution of real graph. The generative models aim to estimate the $p(x)$, the distribution of data that in this case is a social network

7.2.1 Erdös-Gilbert-Rényi model

This model is very simple, in fact it consider the probability p that exists an edge between two nodes is independent to any other.
In statistical terms we can consider the distribution edge as a $Bernoulli(k; p)$.
In this case we define the graph as $G(n, p)$ where n is the number of nodes and p the probability that exists an edge.

The expected number of edges is:

$$\mathbb{E}\left[Edges\right] = \sum_{1 \leq x \leq y \leq n} (1 \cdot p + 0 \cdot (1-p)) = \sum_{1 \leq x \leq y \leq n} p = \binom{n}{2} p = \frac{n(n-1)}{2} p$$

Since in a indirect graph an edge is present in the two extremes of the edge the expected degree of a node is $2 \cdot \frac{n(n-1)}{2} p$.
The probability that a node have k neighbours is a binomial distribution so we can express as $Bin(k; n, p) = \binom{n}{k} \cdot p^k \cdot (1-p)$.
This model for n and k sufficiently big approximate to a normal distribution

7.2.2 Chung-lu model

In this model the probability that exists an edge between two nodes is proportional to the expected degree of the pair of nodes.

- Initially we set $\mathbf{d} = (d_1 \dots d_n)$, the expected degree of each node.
- The probability that exists an edge between the node i and j is

$$p(E_{ij}) = \frac{d_i d_j}{\sum_k d_k}$$

If the degree distribution $\mathbf{d}$ is a power law, then the graph generated is a power law

7.2.3 Barabasi-Albert model

The previous models are static, in fact they have a fixed number of nodes.
This model is dynamic so it mean that doesn't require the number of nodes a priori but they can be added always.
With this dynamism try to give an explain to creation of a social network at difference of the previous models.
Every time a node is inserted an edge is created with the *preferential attachment* system.

Preferential attachment system

Every time a node j is added, a direct edge is created with the following two possibilities:

- with probability p choose a random node $i < j$
- with probability $(1-p)$ choose a node with probability proportional to the in-degree of the node

The latter option is called *rich-get-richer rule*. In fact if you have a high number of relation you are more likely to make more.

Degree distribution

When a node j is created, the probability that exists the edge $(j, i))$ with $j > i$ is:

$$\begin{aligned} P(E_{ji}) &= \frac{1}{j-1} \cdot p + \frac{G_i(j-1)}{\sum_{h \leq (j-1)} G_h(j-1)} \cdot (1-p) \\ &= \frac{1}{j-1} \cdot p + \frac{G_i(j-1)}{j-1} \cdot (1-p) \end{aligned}$$

where $G_i(j-1)$ is the in-degree of the node i at the time $j-1$

Deterministic $G_j(t)$

We want to find an approximation of $G_j(t)$ that give deterministically the in-degree of a node l at time t. Let us denote as $g_l(t)$ the approximation of $G_l(t)$. The increase of in-degree can be expressed from a differential equation:

$$\begin{aligned} &\frac{d(g_l(t))}{dt} = \frac{p}{t} + \frac{g_l(t) \cdot (1-p)}{t} = \frac{p + g_l(t)(1-p)}{t} \\ &\frac{d(g_l(t))}{dt} \cdot \frac{1}{p + g_l(t)(1-p)} = \frac{1}{t} \\ &\int \frac{d(g_l(t))}{dt} \cdot \frac{1}{p + g_l(t)(1-p)} dt = \int \frac{1}{t} dt \\ &g_l(t) \cdot \frac{\ln(p + g_l(t)(1-p))}{1-p} + c' = \ln(t) + c'' \\ &g_l(t) \cdot \ln(p + g_l(t)(1-p)) = \ln(t) \cdot (1-p) + c \\ &e^{\ln(p + g_l(t)(1-p))} = e^{\ln(t) \cdot (1-p) + c} \\ &e^{\ln(p + g_l(t)(1-p))} = e^{\ln(t) \cdot (1-p)} \cdot e^c \\ &(p + g_l(t)(1-p)) = t^{(1-p)} \cdot e^c \\ &g_l(t) = \frac{t^{1-p} \cdot e^c - p}{1-p} \end{aligned}$$

When we insert the node l at time l we have that $g_l(l) = 0$ so we have

$$\begin{aligned} 0 &= \frac{l^{1-p} \cdot e^c - p}{1-p} \\ \frac{p}{1-p} &= \frac{l^{1-p} \cdot e^c}{1-p} \\ e^c &= \frac{p}{l^{1-p}} \end{aligned}$$

if we substitute the latter expression in $g_l(t)$ we have

$$\begin{aligned} g_l(t) &= \frac{t^{1-p} \cdot \frac{p}{l^{1-p}} - p}{1-p} \\ g_l(t) &= p \cdot \left(\left(\frac{t}{l} \right)^{(1-p)} - 1 \right) \cdot \frac{1}{1-p} \end{aligned}$$

We can use the latter expression to estimate the number of nodes that at time t have degree at least k. So we write

$$\left(\left(\frac{t}{l}\right)^{(1-p)} - 1\right) \cdot \frac{p}{1-p} \geq k$$

Then we pone respect to l:

$$\left(\frac{t}{l}\right)^{(1-p)} \geq k \cdot \frac{1-p}{p} + 1$$

$$\frac{t}{l} \geq \left(k \cdot \frac{1-p}{p} + 1\right)^{\frac{1}{1-p}}$$

Note that the next expression is the fraction of nodes at time t that have at least in-degree k:

$$\frac{l}{t} \leq \left(k \cdot \frac{1-p}{p} + 1\right)^{-\frac{1}{1-p}}$$

$$l \leq t \cdot \left(k \cdot \frac{1-p}{p} + 1\right)^{-\frac{1}{1-p}}$$

To have the nodes that have *exactly k in-degree* at *time t* we must take the opposite of derivative (why?) of $\left(k \cdot \frac{1-p}{p} + 1\right)^{-\frac{1}{1-p}}$ respect to k.
The result is

$$\frac{1}{1-p}\left(k\frac{1-p}{p} + 1\right)^{-\left(1+\frac{1}{1-p}\right)}$$

We can infer from the latter expression that the distribution of in-degree nodes is a power law with exponent $\beta = 1 + \frac{1}{1-p}$.
From

- when $p \simeq 1$ we exponent disappear then we tend to Erdös-Gilbert-Rényi model.
- when $p \simeq 0$ we apply almost ever the Preferential Attachment System, so the new nodes will more probably link to others with high in-degree, that is very similar to what happen with Chung-lu model.
- The distribution disappear when $p \propto k^{-2}$