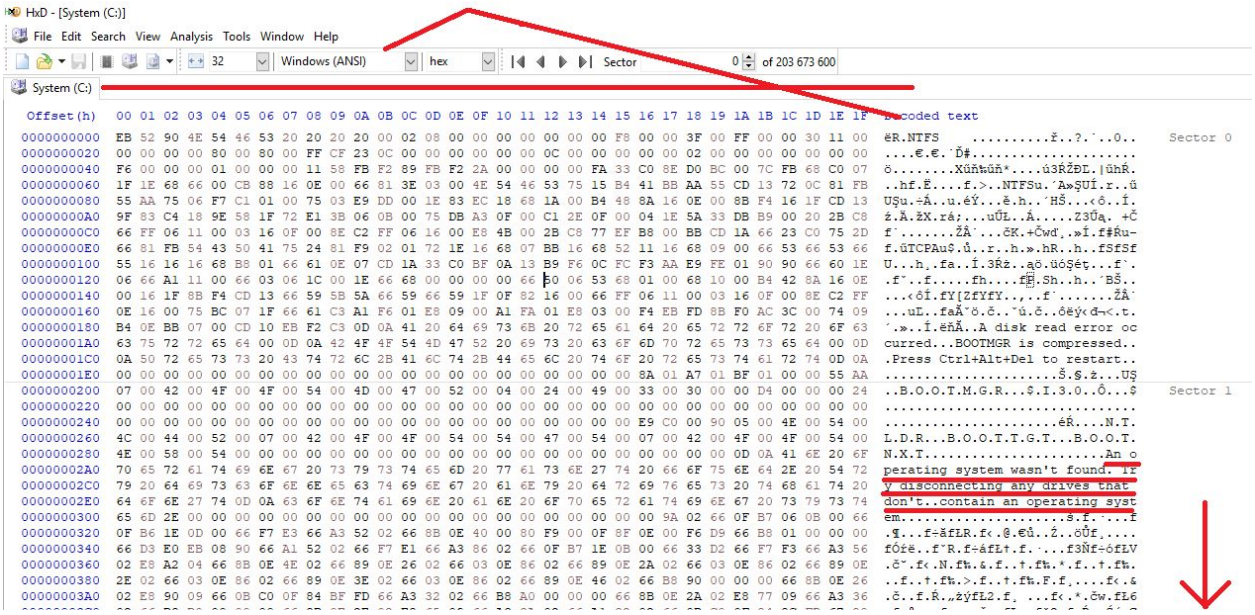
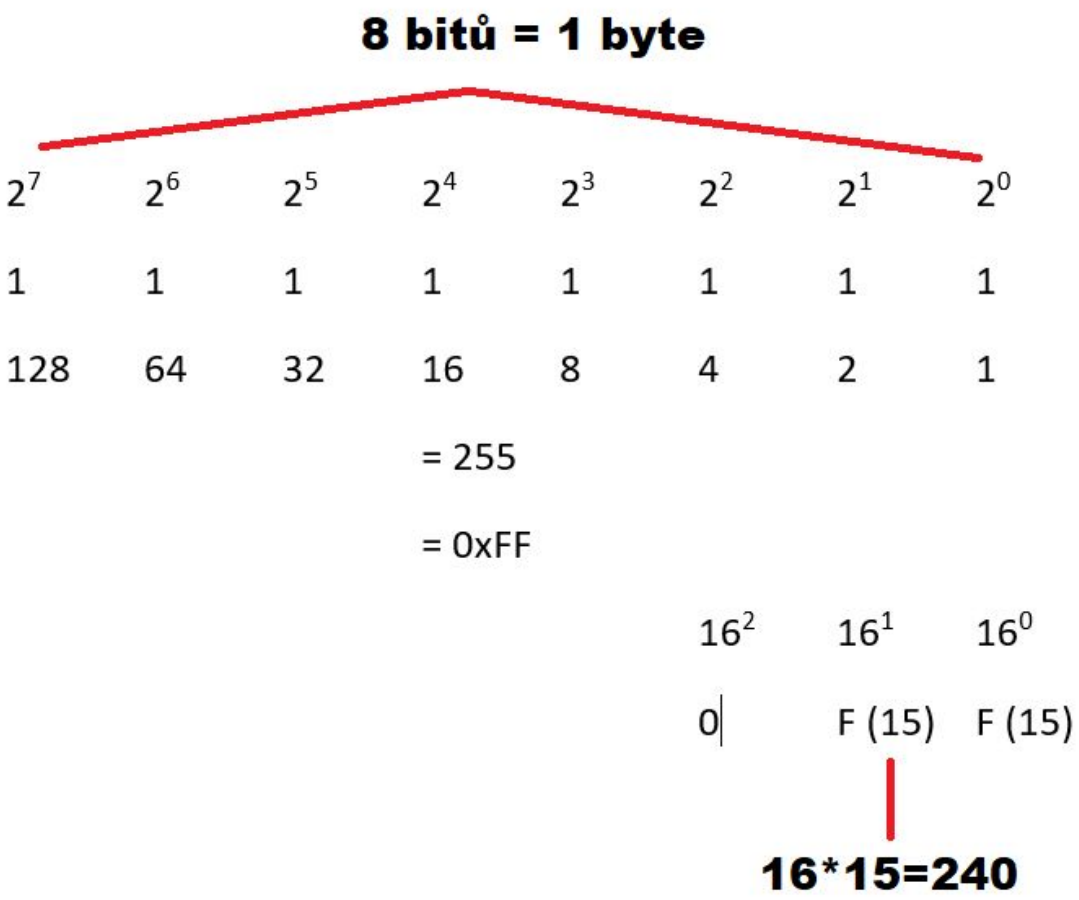
SOUBORY A FORMÁTY

Data jsou bity

- V důsledku jsou všechna data bity - 0 a 1.
- Celý disk lze ukázat jako obrovský řetěz 0 a 1
    - V tom řetězu pak některé kombinace 0 a 1 budou sloužit jako oddělovače mezi jedním a druhým souborem (v reálu je to asi ještě trochu složitější, ale už ne o moc) => takže máme třeba řetěz
        - 00110001110101110101000011010110010011100011000 10
    - Oddělovač je řekněme “**111000”**
    - **00110001110101110101000011010110010011100011000 10**
- Hexeditor: https://mh-nexus.de/en/hxd/



- Když už na něco takového koukáme, tak na to většinou nekoukáme jako na serii 0 a 1, protože to je úplně nečitelné, ale spíše jako na byty (8 bitů), které lze pěkně zapsat dvojmístným hexem (viz obrázek níže a vysvětlení výše)



Názvy souborů, mazání souborů a obnova dat, přípony

- Disk má nějakou kapacitu, takže je jasně dána ta délka řetězce
- Jednotlivé soubory jsou tedy ty určité kousky toho řetězce (viz výše)
- Když nějaká data dáme do “koše”, tak jen změníme umístění do této speciální “složky” a vlastně “nemažeme”.
- Když pak koš vysypeme a nebo rovnou mažeme soubory (shift + delete), pak se děje to, že se na disku uloží informace, že ten určitý řetězec 0 a 1 je volný k přepsání nějakými novými daty. => dokud ale nezačnu na disk něco dalšího zapisovat, nejsou stále smazaná
- ⇒ obnova disku: takto pak dochází k obnovám disku. Speciální program nebo i člověk se pokusí identifikovat data, která chceme obnovit. Pokud jsme je něčím nepřemazali, pak je lze zase zaktivnit. Jakmile začne docházet k jejich přepisování, je soubor poničen a s největší pravděpodobností už data nepůjde obnovit, protože nikdo netuší, které 0 a 1 jsou které. Expert dokáže obnovit, co se dá, protože rozumí, jak jsou různá data poskládána a dokáže je třeba obnovit alespoň částečně (třeba kriminálka). Zvláště text je dobře obnovitelný, protože tam je víceméně každý byte určité písmeno.

- Přípony: nejsou nic příliš speciálního, jsou to jen písmena za tečkou a windows má v sobě uloženo, v jakém programu se má daná přípona primárně otevírat. Když vím, v čem to mám otevřít tak to žádnou příponu nepotřebuje (ukázat)
    - Film Marťan, kde poslali “obrázkový soubor. jpg”, ale nedařilo se jej otevřít a zjistilo se, že jde vlastně o textový soubor

Formát(y) obecně

- Pojem “formát” je klíčový
    - Značí jednak způsob/logiku, jak jsou ty 0 a 1 poskládány, aby se v nich dal zachovat ten obsah, který chceme
    - Z opačné strany je to domluva mezi těmi chaotickými 0/1 a programem, který se je snaží otevřít/číst.
- V konečném důsledku si formát může vymyslet každý z nás:
    - řekneme, že třeba prvních 100 0/1 značí nadpis, dalších 500 text a posledních 200 podpis… => když to nějaký program bude ochotný číst, máme formát
    - Jiný příklad:
        - první byte určí počet kostiček v řádku a následovat budou 3 byty definující barvu v R-G-B formátu
        - ⇒ program, který bude ochotný to číst, nám vykreslí linku v barvě, kterou jsme si nadefinovali a o délce, kterou jsme určili v prvním bytu
        - ⇐ pro kohokoli, kdo formát nezná to budou jen 0/1 či řada 4 bytů (délka + barva)
- Některé formáty jsou hodně jednoduché, jiné jsou tak komplikované, že není v lidských silách z nich nic smysluplného pochopit (řídí se však logikou, takže pokud by tím ten člověk strávil třeba celý život, pak byl schopen zrekonstruovat třeba obrázek…)
    - Viz dále: textové (prosté) formáty jsou ještě lidsky pochopitelné, většina dalších už ne

Kategorie souborů

- Mnou navržená kategorizace (= nejde o nějaké oficiální dělení takto)
    - ⇒ detailněji k jednotlivým typům dále mini-kapitolky, zde jen výčet
    - a) Textové soubory - prosté a méně prosté (txt, xml, csv, rtf, html)
    - b) Databazove soubory (sqlite, ms access...)
    - c) Archivy, balíčky (zip a jine)
    - d) Základní mediální soubory (bmp, wav)
    - e) Programové soubory (windows: spustitelny exe, dll)
    - x) Ostatní… cokoliv dalšího, co je nějaký program schopný interpretovat (png, flac, kra…)

a-1) prosté textové formáty a formáty složené z prostého textu (txt, csv, xml/json)

- "Prostý text" (plain text) je zavedený pojem
    - Má se tím na mysli řetězec ASCII/UTF znaků bez dalšího formátování ve smyslu velikost textu, odsazení paragrafu, barva písma, síla textu (bold), zarovnání na řádku…
- Tím prostým textovým formátem se myslí to, že se jedná vlastně jen o řetězec znaků v ANSI/UTF (nová řádka je také znak, který je interpretovaný jako nová řádka)
- Když naruším obsah souboru, třeba uberu prvních ½ 0/1 (bytů), tak tím nenaruším čitelnost zbytku obsahu, ale naruším tím logiku souboru jako celku
    - Jako když roztrhnu knihu a budu číst pouze rozuzlení, není jasná zápletka, expozice postav, prostředí…
    - Když odmažu hlavičku nějakého formuláře, tak nevím, co to je za formulář…
    - Stejně tak ITácké formáty jako CSV, XML… když takto naruším, budou sice lidsky stále čitelné, ale naruší se struktura a ani pro nás, ani pro nějaký program, který by to chtěl číst, to nebude dávat moc smysl ⇐ v tomto smyslu je pak soubor také porušen (= corrupt file)
- CSV, XML/JSON
    - Slouží pro strukturovaný přenos dat
    - Představit strukturu těchto formátů

a-2) rich text format (RTF) ~ HTML

- Prostý textový formát + to "základní" formátování (skoro všechno, co by nás při prvotním výčtu napadlo)
    - RTF jako koncept a jako konkrétní formát
    - Takto formátovaný text lze zapsat třeba také v jiném formátu, např. "HTML", což je formát užívaný pro vytváření webových "stránek" (stránka jako stránka ne?), přičemž se jedná ještě o lidsky čitelnější formát
        - Ukázat příklad z takového html
        - Přestože html soubor je očekávaný webovým prohlížečem, umí jej otevřít také právě třeba MS Word, protože se právě jedná také víceméně o formátovaný strukturovaný text
        - **[trynna impress everyone by quickly throwing together some html...]**
        - **Teď nezačínat dále o html, css, ničem takovým… samostatné téma**
- RTF jako formát
    - Ukázat otevřený jako prostý text: lze to číst, ale už to není úplně lidské...

b) databázové soubory

- Jen úplný základ k “databázi”, **jinak samostatné téma**
  - Pojem “tabulka”
    - = každý zná i obsahově, žádnou babičku “tabulka” úplně nepřekvapí
    - prostě seznam sloupců určující “co” + řádky s hodnotami/daty
    - řádek (jednotlivé buňky) spolu při normálním použití souvisí jako údaje k té jedné samé věci (není to teda mřížka s náhodně naházenými hodnotami)
    - *[Řádek by ideálně měl být uvozen tzv. “Primární klíčem”, unikátním identifikátorem řádku*
      - *V běžném použití mimo IT svět o něčem takovém člověk neuvažuje ale i tak to implicitně vlastně používá ⇐ pokud se chci s někým dorozumět “jaký řádek má na mysli”, dost možná mu odpovím třeba “čtvrtý od shora” = to lze zjednodušeně chápat jako tento identifikátor]*
  - Pojem “databáze”
    - = dnes běžně užívaný, ale smysl už většinou spíše nejasný; babička bude nejspíš mimo
    - V důsledku je to sice daleko komplexnější záležitost, ale v základu je to hodně prosté a často to stačí i ITákovi programátorovi takhle ⇒ databáze je pojem pro *soubor* tabulek, přičemž ty tabulky mohou být na sobě buď zcela nezávislé a nebo spolu mohou nějak souviset (tabulka zaměstnanců, tabulka měsíčních výplat těchto zaměstnanců…)
- Základní logika je sice stejná, ale mám více druhů databází, od různých firem
  - Stejně jako textový soubor je také jedna logika, tak ale mohu mít .docx, .rtf, .odt (OpenOffice), .odf (LibreOffice) atd… a navzájem spolu ne vždy komunikují
  - Tak mám také různé databázové systémy: MySQL, Oracle (SIS), MS SQL Server, SQLite
- SQLite
  - je velice často používaný databázový soubor a ukládá-li nějaký program nějaké uživatelem definované hodnoty, je celkem pravděpodobné, že to bude právě SQLite soubor
  - takový lze otevřít například volným nástrojem “DB Browser for SQLite”
  - Pokud by někdo nechtěl pracovat s Excelovými tabulkami, ale chtěl by s daty nějak komplexněji pracovat, může si takto vytvořit svůj vlastní databázový soubor
    - **[Vytvořit si příklad, který by ladil s tím SQL z první části?]**
  - Zkusit na PC najít nějaké náhodné databázové soubory (.db) a zkusit je otevřít (Skype byl třeba povolnej)

c) Archivy (zatajené archivy)

- Asi dvojí funkce: 1) jednak je občas součástí “sbalení” do archivu také nějaká komprese dat (**ke kompresi krátce dále**) a 2) asi ještě důležitěji: sloučí více souborů do jednoho souboru takovým způsobem, že je jasné, z čeho se skládá a lze to zase “rozbalit”
- Přiznané, komprimované archivy: .zip, .rar, 7z (*7-zip doporučený nástroj)*
- Zatajené archivy:
    - Některé soubory, u kterých by nás to na první pohled vůbec nenapadlo, jsou v důsledku také archivy a to dokonce rozbalitelné klasickými nástroji (extrahovat zip)
    - Příklady: docx, Krita, Photoshop soubor třeba a spousta dalších (většinou různé “projekty”)
    - programy toho vyuzivaji, aby měly pekne oddělená data a zároveň, aby uživatel neviděl složku, kterou by měl tendenci modifikovat a tím narušit strukturu

d) Základní mediální soubory

- Také základní, ale už dost komplikované na to, aby je člověk jen tak přečetl.
- Vyčlenil jsem je z kategorie “x) Ostatní” proto, že přeci jen s určitou snahou je ještě reálné jim porozumět

Příklad BMP

- *[V přírodě nejsou žádné barvy, jen odlišné odrážení spektra bílého světla]*
- Víceméně stejně jako WAV, jen je to ještě pochopitelnější
- U WAV se za sebou skládá síla tlaku vzuchu/prohnutí membrány reproduktoru = data; zde se za sebou skládají barvy pixelů = data
- Dokumentace: http://www.ece.ualberta.ca/~elliott/ee552/studentAppNotes/2003_w/misc/bmp_file_format/bmp_file_format.htm

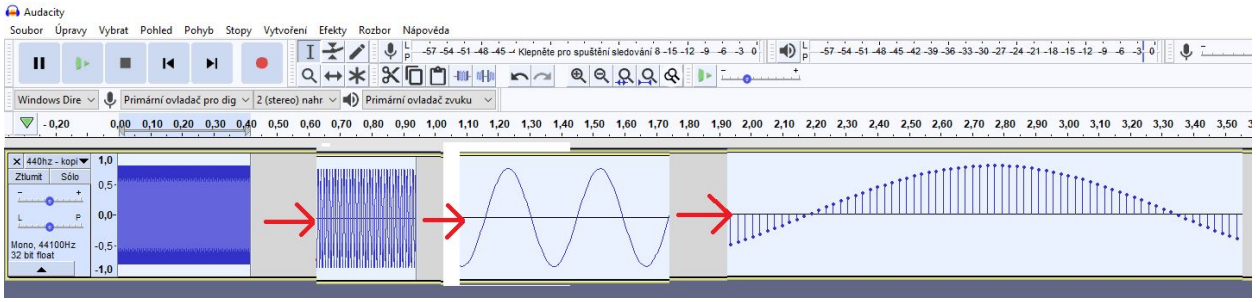
[obrázek: ilustrace dokumentace na skutečném příkladu obrázku BMP o rozměru 4*4 pixelů]

Below is a more detailed table of the contents of each of these structures.

Name	Size	Offset	Description
Header	14 bytes		Windows Structure: BITMAPFILEHEADER
Signature	2 bytes	0000h	'BM'
FileSize	4 bytes	0002h	File size in bytes
reserved	4 bytes	0006h	unused (=0)
DataOffset	4 bytes	000Ah	Offset from beginning of file to the beginning of the bitmap data
InfoHeader	40 bytes		Windows Structure: BITMAPINFOHEADER
Size	4 bytes = 40 bytes	000Eh	Size of InfoHeader =40
Width	4 bytes	0012h	Horizontal width of bitmap in pixels
Height	4 bytes	0016h	Vertical height of bitmap in pixels
Planes	2 bytes	001Ah	Number of Planes (=1)
Bits Per Pixel	2 bytes	001Ch	Bits per Pixel used to store palette entry information. This also identifies in an indirect way the number of possible colors. Possible values are: 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 24 = 24bit RGB. NumColors = 16M
Compression	4 bytes	001Eh	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding
ImageSize	4 bytes	0022h	(compressed) Size of Image It is valid to set this =0 if Compression = 0
XpixelsPerM	4 bytes	0026h	horizontal resolution: Pixels/meter
YpixelsPerM	4 bytes	002Ah	vertical resolution: Pixels/meter
Colors Used	4 bytes	002Eh	Number of actually used colors. For a 8-bit / pixel bitmap this will be 100h or 256.
Important Colors	4 bytes	0032h	Number of important colors 0 = all
ColorTable	4 * NumColors bytes	0036h	present only if Info.BitsPerPixel less than 8 colors should be ordered by importance
Red	1 byte		Red intensity
Green	1 byte		Green intensity
Blue	1 byte		Blue intensity
reserved	1 byte		unused (=0)
repeated NumColors times			
Pixel Data	InfoHeader.ImageSize bytes		The image data

Příklad WAV

- Základ logiky formátu wav - přímá souvislost s podstatou toho, co je to zvuk a mechanikou záznamu a přehrávání
- *[v přírodě nejsou žádné zvuky, jen smršťování a rozpínání látek]*



- Formát je otevřený, má dokumentaci, každý jej může pochopit, když bude chtít a může si třeba napsat nějaký program, který takový formát bude generovat: http://soundfile.sapp.org/doc/WaveFormat/

e) Programové soubory (exe, dll, jar, app)

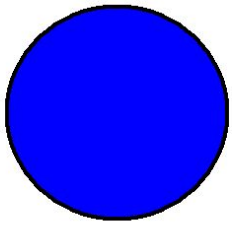
- exe a dll
    - exe i dll jsou formáty specifické pro Windows
    - exe je “executable”
        - “executes” soubor nějakých instrukcí (= program)
    - dll je “dynamic link **library**”
        - nakonec skoro to samé, co exe, ale není to přímo spustitelné, protože to nemá žádný výchozí bod, od kterého by se nějakým způsobem měly instrukce vykonávat. Je to pouze soubor instrukcí, knihovna, čekající až je nějaký program (exe) použije
    - Analogie EXE a DLL?
        - Hračka “chodící robot” je EXE = stojí a nic nedělá než ho “spustím” a on začne chodit, jak má naprogramováno
        - “Laser gun” pro “chodícího robota” sama nic nedělá, nelze ji samotnou nijak spustit, aby začala mířit a střílet ⇒ ale pokud ji dám robotovi do ruky, tak v určité chvíli “ji použije” a začne mířit a střílet...
    - ⇒ když programátor (windows, .NET) píše kód, pak se tento lidsky čitelný text ve výsledku “kompiluje” a vytváří tím “exe” nebo “dll” soubory (programátorův kód v případě dll úplně stejný, odlišnost je jen v tom “výchozím bodu”)
        - Z pouhého textového souboru (.cs) se stává nečitelný binární soubor ve speciálním komplikovaném formátu (exe/dll)
        - Ne každý kód se kompiluje, některé jazyky pracují přímo s lidsky napsaným kódem (php, html, javascript)
    - V novějších (víc jak 20let asi) Windows je nainstalovaný tzv. .NET framework. To je prostě soubor novějších windows knihoven (taky dll), které programátor používá a program s nimi počítá
- jar
    - Podobně jako většina exe počítá, že bude ve windows nějaká aktuálnější verze .NET framework, tak .jar soubory, které jsou také “executables”, počítají, že bude na počítači nainstalovaná tzv. Java
    - Java je také takový framework + je to i název programovacího jazyku, ve kterém se pro tento framework píše kód (existuje také jazyk, který se jmenuje “javascript” a ten s tímto nemá vůbec nic společného, i když se jmenuje podobně)
- App
    - Soubory s příponou .app jsou ekvivalentem .exe na Macu (Mac OS)

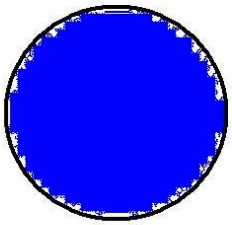
x) Ostatní

- Cokoliv jiného, většinou dost komplikované, umějí to otevřít konkrétní programy (někdy prostupnost .PSD, jindy pouze pro daný program… dohoda)

Vývoj formátů, velikosti souborů, komprese

- Formáty procházely vývojem od základních, kdy se jen za sebou skládaly informace, po komplikovane, kde dochází k co největší kompresi dat bez ztraty informaci
- Příklad:
  - “ahoj ahoj ahoj ahoj ahoj ahoj ” a “7*ahoj-”
  - Také viz odkaz dokumentace BMP formátu výše a část o kompresi, kde se popisuje prakticky shodný princip
- Ztrátové a bezztrátové komprese
  - Některé formáty jsou takové, že pro úsporu objemu data ukládají způsobem, kdy se některá data ztrácejí, zjednoduší = ztrátové formáty
    - bmp > jpg (podobně barevné pixely vyprumeruje a ulozi si je jako shodnou informaci)
    - wav > mp3
  - Jiné, chytřejší, novější, formáty jsou schopné data uložit hodně efektivně a přitom neztrácet vůbec žádná data
    - Bmp > png
    - Wav > ogg, flac, ape
  - ⇒ příklad výše “ahoj ahoj” by byla bezztrátová komprese. Ztrátová typicky jpg **[ilustrovat příkladem “kolečko” jako na příkladu z netu]**





Quick Lesson About Saving in JPG vs. PNG

• The top circle was saved in PNG, then reopened and filled in. As you can see, it fills in nicely.

• The bottom was saved in JPG and reopened the same way, and filled in. It, however did not fill in nicely.

The reason for this, is when a picture is saved in JPG format using a program like Microsoft Paint, the pictures quality decreases and causes a minor shading.

To do this, JPEG relies on discrete cosine transform (DCT). While the math behind it is complicated, this compression algorithm takes a look at the entire image, determines which pixels in the image are similar enough to the ones around it, and merges the pixels together in tiles (groups of pixels that have the same value).

This method is extremely efficient but comes at the cost of throwing away information you can't get back. JPEG images (with a few exceptions mentioned below) are lossy, which means after the image is saved, the data that was lost can't be recovered.