Dynamiczne kodowanie słownikowe LZSS

Realizacja trzeciego etapu projektu

Domański Piotr, Pietrowcew Jakub, Skórka Kornel

2022-01-30

Implementacja algorytmu LZSS

Projekt został zaimplementowany w języku C++. Składa się z dwóch oddzielnych programów: kodera `coder_lzss` oraz dekodera `decode_lzss`. Oba programy posiadają jedynie interfejs konsolowy i są sterowane za pomocą parametrów ich wywołania.

Koder LZSS

Wywołanie

Argumentami wywołania kodera są:

- nazwa pliku wejściowego (kodowanego)
- nazwa pliku wyjściowego
- rozmiar słownika wyrażony w bajtach
- rozmiar bufora frazy wyrażony w bajtach

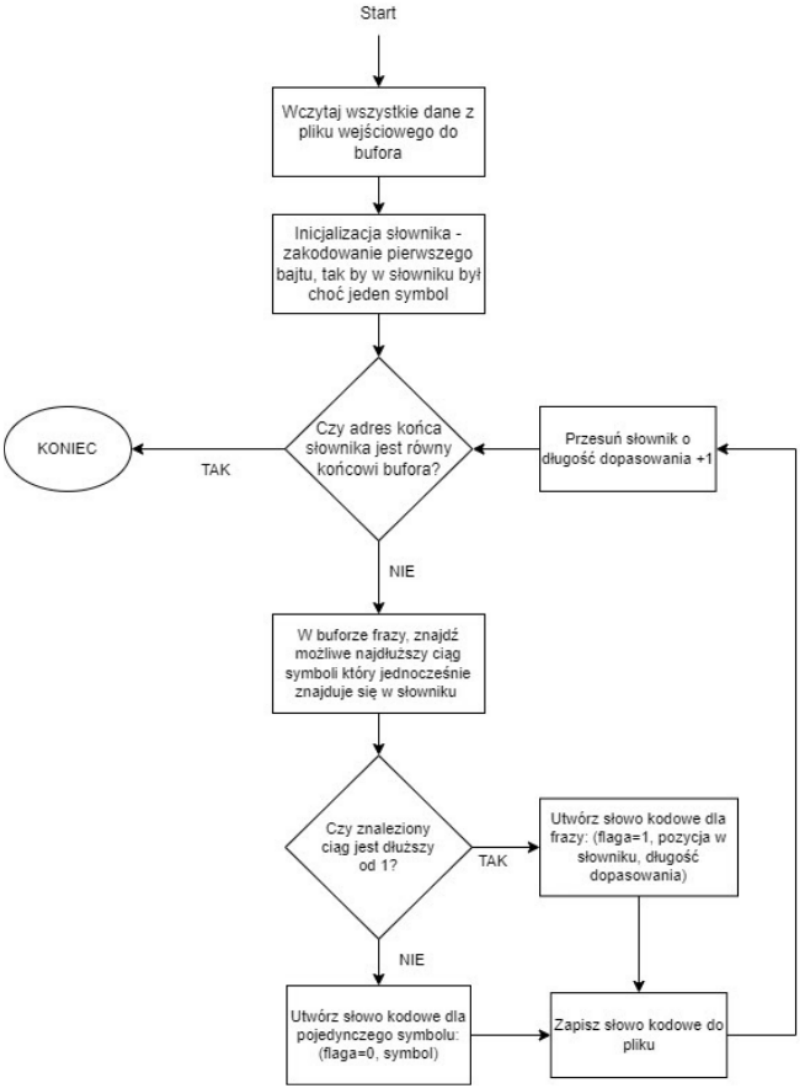
Rezultatem poprawnego wykonania programu są dwa pliki. Pierwszy plik o nazwie zgodnej z argumentem wywołania zwierające zakodowane dane. Drugi natomiast zawiera statystyki związane z procesem kodowania, a jego nazwa tworzona jest jako: nazwa_zakodowanego_stats.txt. Zawiera on poniższe dane:

- informację o wielkości słownika
- informację o wielkości bufora
- informację o czasie kodowania
- wielkość pliku przed kompresją
- wielkość pliku po kompresji
- stopień kompresji
- liczbę słów kodowych
- średnią długość bitową

Kluczowe szczegóły implementacji algorytmu kodera

Cała zawartość pliku, który ma zostać zakodowany jest na początku wczytywana do bufora “wejściowego” o rozmiarze równym wielkości pliku. Takie podejście jest wystarczające biorąc pod uwagę niewielkie rozmiary plików wejściowych oraz ilość pamięci dostępnej we współczesnych komputerach. Jednakże lepszym podejściem byłoby wczytywanie do bufora tylko części pliku wejściowego, tak aby przy rozmiarach plików wejściowych rzędu setek MB nie alokować, takich ilości pamięci.

Słownik jest implementowany jako wskaźnik, który informuje, gdzie znajduje się w danym momencie początek słownika w buforze “wejściowym”. Poza wskaźnikiem przechowujemy bieżącą długość słownika. Długość słownika jest nie większa niż maksymalna długość słownika podana jako parametr wywołania - na początku działania algorytmu, po inicjalizacji słownika jego długość wynosi jeden i rośnie do maksymalnej dopuszczalnej wartości wraz z wykonywaniem algorytmu.



Rysunek 1: Schemat działania

Słowa kodowe w postaci bitów są tworzone poprzez wykonywanie operacji bitowych (alternatywy i przesunięć) na zmiennej typu unsigned int o rozmiarze 32 bitów - odbywa się to w metodzie WriteBits klasy BitBuf. Po zapełnieniu 32 bitów zmiennej typu unsigned int jest ona umieszczana w buforze _puiBuf będącym polem prywatnym klasy BitBuf. Bufor ten jest cyklicznie zapisywany do pliku wyjściowego. Pewnym problem okazała się sytuacja, gdy na koniec kodowania 32 bitowa zmienna typu unsigned int była tylko częściowo wypełniona bitami związanymi ze słowem kodowym i to w taki sposób, iż nie były to pełne bajty (do pliku najmniej możemy zapisać 1 bajt) tylko np. 10 z 32 bitów. W takim przypadku bity pozostałe do pełnego bajtu ustawiane były na wartość jeden. Takie uzupełnienie jedynkami dla było jednocześnie sposobem na poinformowaniu dekodera o końcu sekwencji kodowej. Ponieważ dekoder napotykając uzupełniające jedynki będzie w pierwszej kolejności postępował tak jakby dekodował słowo kodowe dla frazy (odczyta bit flagi = 1), ale bitów będzie mniej niż dla typowego słowa kodowego frazy, bądź w skrajnym przypadku odczyta że dopasowanie o maksymalnej długości zaczyna się na końcu słownika, co jest naturalnie niemożliwe.

Dekoder LZSS

Wywołanie

Argumentami wywołania dekodera są:

- zakodowany plik
- wskazanie na docelowy plik ze zdekodowanymi danymi
- ewentualne wskazanie na plik ze statystykami

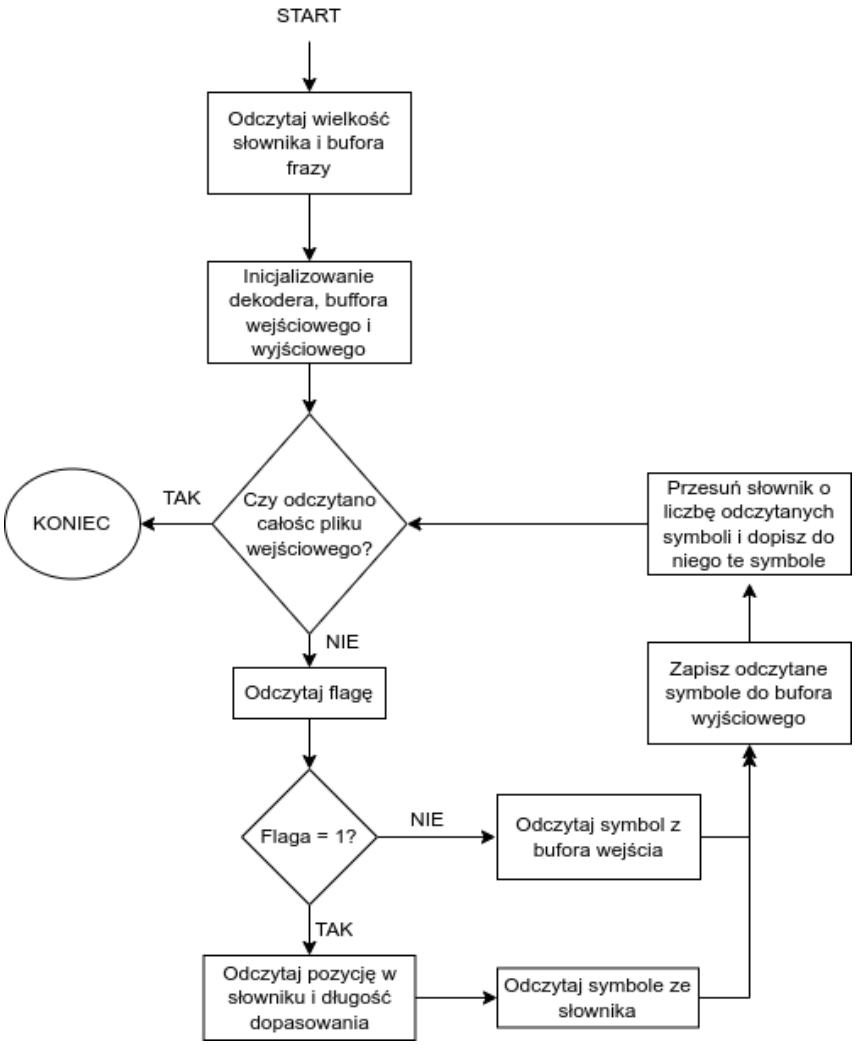
Rezultatem poprawnego wywołania dekodera jest odkodowanie danych do wskazanego pliku docelowego. Dodatkowo, jeżeli został podany plik ze statystykami zostanie do niego dopisania informacja o czasie dekodowania.

Kluczowe szczegóły implementacji algorytmu dekodera

W przeciwieństwie do kodera, dekoder nie wczytuje pliku wejściowego w całości do pamięci. Zamiast tego zaimplementowano specjalny buffer bitowy - BitBuf, który w pamięci przechowuje tylko aktualnie analizowane 4 bajty (32 bitów) pliku wejściowego. BitBuf umożliwia czytanie kolejnych bitów wejścia za pomocą operacji bitowych - alternatywy i przesunięć na zmiennej unsigned int (co wynika z rozmiaru analizowanych danych wejściowych, 4 bajty). Implementacja BitBuf zapewnia ciągłe wczytywanie kolejnych porcji danych wejściowych i dostarcza informacji o odczytaniu ostatniego słowa kodowego sygnalizującą ukończenie dekodowania danych.

Słownik został zaimplementowany jako tablica elementów typu char o określonej wartości i przechowuje jedynie aktualną zawartość słownika. Jest on aktualizowany i przesuwany w każdej iteracji algorytmu.

Dekoder trzyma otwarty strumień wyjścia do pliku wyjściowego i w każdej iteracji algorytmu zapisuje do niego dane odczytane z kolejnego słowa kodowego.



Rysunek 2: Schemat działania

Dane testowe

W pierwszej kolejności, w celu sprawdzenia poprawności działania przygotowanego algorytmu upewniono się, czy pliki przed kompresją są identyczne z plikami otrzymanymi po przejściu procesu kompresji i dekompresji. Oczywiście takie sprawdzenie jest możliwe tylko dla algorytmów kompresji bezstratnej, do której zalicza się kodowanie słownikowe LZSS.

Rozmiar plików testowych podlegających kodowaniu wynosił 262 182 bajty dla obrazów i 262 159 bajty dla rozkładów.

Testy dla: słownika = 4096 i bufora frazy = 32 bajty

Nazwa pliku	Czas kodowania [ms]	Czas dekodowania [ms]	Rozmiar po kodowaniu [bajty]	Wsp. kompresji	Liczba słów kodowych	Średnia długość bitowa
barbara.pgm	1466	2383	280 576	0.93	178 892	8.56
boat.pgm	1343	2068	276 096	0.95	156 878	8.42
chronometer.pgm	1004	1494	185 344	1.41	111 105	5.66
geometr_05.pgm	913	648	103 552	2.53	46 405	3.16
geometr_099.pgm	1939	3090	294 656	0.89	233 412	8.99
geometr_09.pgm	1201	1601	250 752	1.05	119 755	7.65
laplace_10.pgm	1366	2073	285 824	0.92	153 096	8.72
laplace_20.pgm	1592	2559	293 376	0.89	191 261	8.95
laplace_30.pgm	1782	2868	294 400	0.89	215 992	8.98
lena.pgm	1271	2091	271 872	0.96	155 514	8.30
mandril.pgm	1458	2388	284 928	0.92	175 942	8.69
normal_10.pgm	1275	1858	282 112	0.93	138 604	8.61
normal_30.pgm	1691	2754	294 144	0.89	205 102	8.98
normal_50.pgm	1900	3189	294 656	0.89	234 429	8.99
peppers.pgm	777	1233	181 888	1.44	89 507	5.55
uniform.pgm	1978	3289	294 784	0.89	247 271	9.00

Testy dla: słownika = 2048 i bufora frazy = 32 bajty

Nazwa pliku	Czas kodowania [ms]	Czas dekodowania [ms]	Rozmiar po kodowaniu [bajty]	Wsp. kompresji	Liczba słów kodowych	Średnia długość bitowa
barbara.pgm	800	1334	278 272	0.94	197 098	8.49
boat.pgm	764	1162	272 640	0.96	173 158	8.32
chronometer.pgm	541	844	187 776	1.40	119 927	5.73
geometr_05.pgm	518	343	106 496	2.46	50 716	3.25
geometr_099.pgm	1017	1660	292 736	0.90	245 988	8.93
geometr_09.pgm	654	884	252 032	1.04	131 888	7.69
laplace_10.pgm	766	1136	278 656	0.94	169 910	8.50
laplace_20.pgm	890	1409	287 872	0.91	212 567	8.78
laplace_30.pgm	966	1564	291 072	0.90	233 684	8.88
lena.pgm	718	1175	269 568	0.97	175 611	8.23
mandril.pgm	818	1334	281 472	0.93	198 631	8.59
normal_10.pgm	705	1030	274 816	0.95	153 420	8.39
normal_30.pgm	940	1510	290 048	0.90	226 446	8.85
normal_50.pgm	997	1649	292 864	0.90	246 781	8.94
peppers.pgm	434	681	186 496	1.41	100 234	5.69
uniform.pgm	1040	1699	293 888	0.89	254 422	8.97

Testy dla: słownika = 1024 i bufora frazy = 32 bajty

Nazwa pliku	Czas kodowania [ms]	Czas dekodowania [ms]	Rozmiar po kodowaniu [bajty]	Wsp. kompresji	Liczba słów kodowych	Średnia długość bitowa
barbara.pgm	450	754	278 272	0.94	214 740	8.49
boat.pgm	436	650	271 232	0.97	191 998	8.28
chronometer.pgm	293	444	191 488	1.37	129 351	5.84
geometr_05.pgm	264	202	110 080	2.38	55 990	3.36
geometr_099.pgm	524	866	292 608	0.90	253 471	8.93
geometr_09.pgm	359	496	251 264	1.04	146 449	7.67
laplace_10.pgm	424	646	274 560	0.95	190 566	8.38
laplace_20.pgm	489	783	286 720	0.91	230 924	8.75

Nazwa pliku	Czas kodowania [ms]	Czas dekodowania [ms]	Rozmiar po kodowaniu [bajty]	Wsp. kompresji	Liczba słów kodowych	Średnia długość bitowa
laplace_30.pgm	512	832	290 688	0.90	245 861	8.87
lena.pgm	406	671	269 952	0.97	197 288	8.24
mandril.pgm	453	744	281 088	0.93	219 278	8.58
normal_10.pgm	404	599	269 696	0.97	174 914	8.23
normal_30.pgm	502	823	289 664	0.91	241 842	8.84
normal_50.pgm	517	860	292 864	0.90	254 081	8.94
peppers.pgm	252	397	193 536	1.35	114 893	5.91
uniform.pgm	518	868	293 888	0.89	258 146	8.97

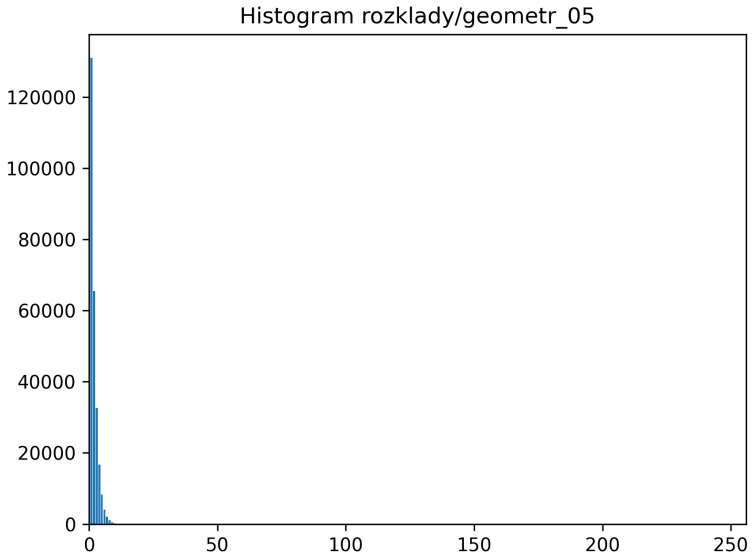
Testy dla: słownika = 2048 i bufora frazy = 64 bajty

Nazwa pliku	Czas kodowania [ms]	Czas dekodowania [ms]	Rozmiar po kodowaniu [bajty]	Wsp. kompresji	Liczba słów kodowych	Średnia długość bitowa
barbara.pgm	808	1361	285 440	0.92	197 098	8.71
boat.pgm	746	1164	282 496	0.93	173 158	8.62
chronometer.pgm	539	837	193 792	1.35	119 624	5.91
geometr_05.pgm	486	346	112 768	2.32	50 716	3.44
geometr_099.pgm	1007	1688	294 784	0.89	245 988	9.00
geometr_09.pgm	656	888	264 960	0.99	131 888	8.09
laplace_10.pgm	755	1142	289 664	0.91	169 910	8.84
laplace_20.pgm	926	1442	293 888	0.89	212 567	8.97
laplace_30.pgm	962	1562	294 528	0.89	233 684	8.99
lena.pgm	720	1209	278 400	0.94	175 571	8.49
mandril.pgm	815	1343	288 768	0.91	198 631	8.81
normal_10.pgm	702	1061	287 616	0.91	153 420	8.78
normal_30.pgm	965	1503	294 528	0.89	226 446	8.99
normal_50.pgm	995	1680	294 784	0.89	246 781	9.00
peppers.pgm	449	692	195 712	1.34	100 234	5.97
uniform.pgm	1049	1712	294 784	0.89	254 422	9.00

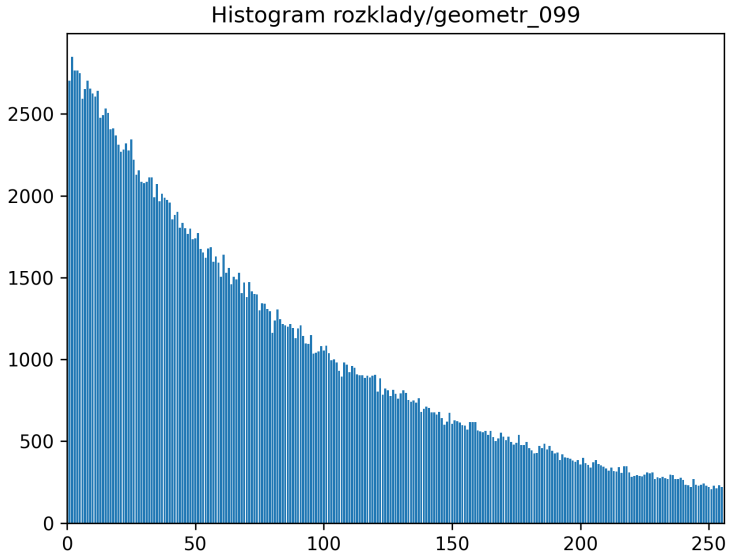
Histogram i entropia danych wejściowych

Program został przetestowany przy użyciu sztucznie wygenerowanych ciągów danych tekstowych oraz obrazów. Do testów zostały wykorzystane pliki o rozkładzie równomiernym, normalnym oraz Laplace. Poniżej znajdują się histogramy tych plików.

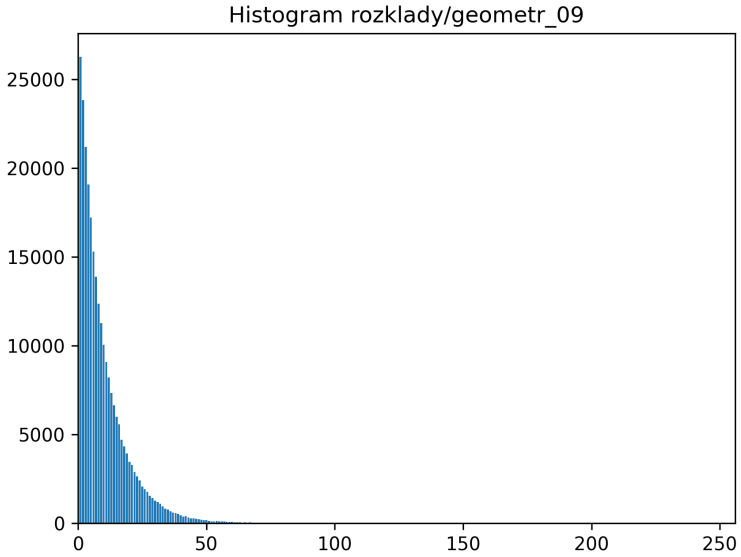
Entropia rozkładów



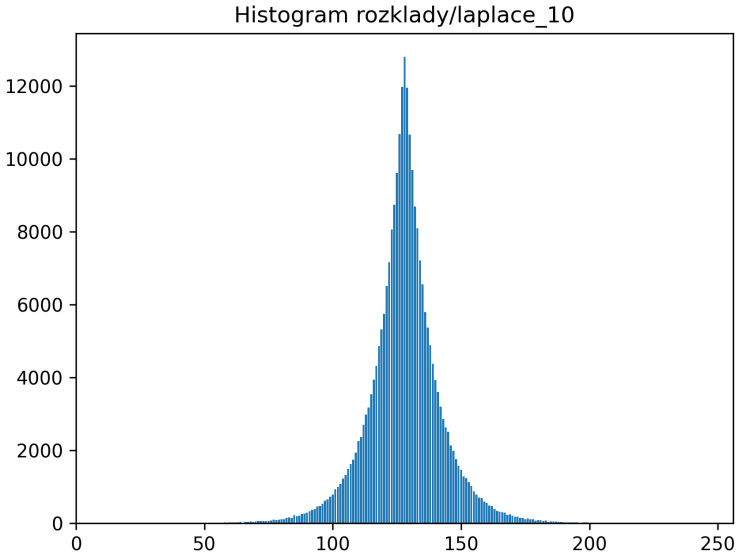
Rysunek 3: geometr_05.pgm - histogram.



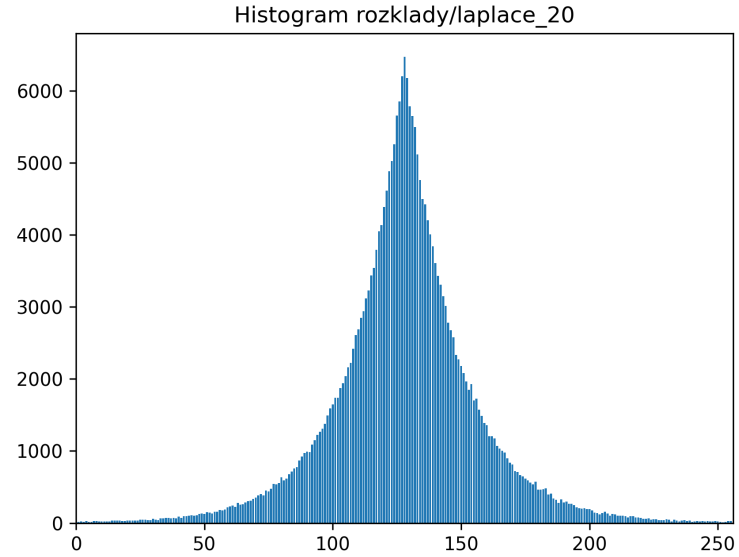
Rysunek 4: geometr_099.pgm - histogram



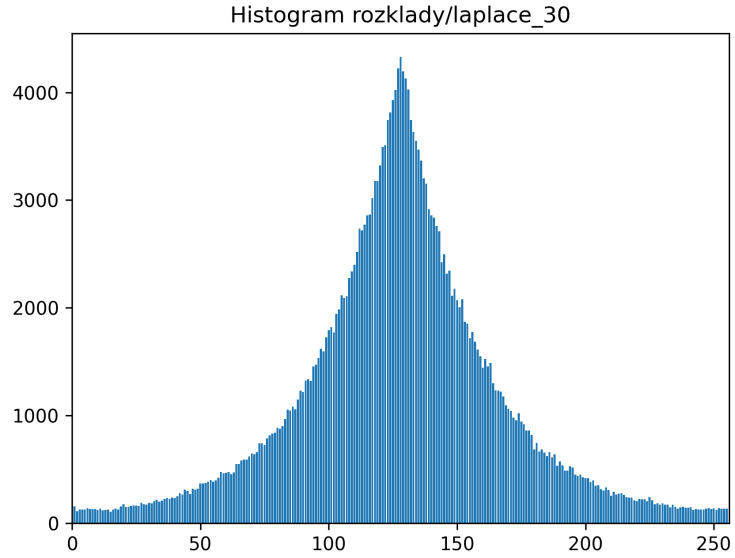
Rysunek 5: geometr_09.pgm - histogram



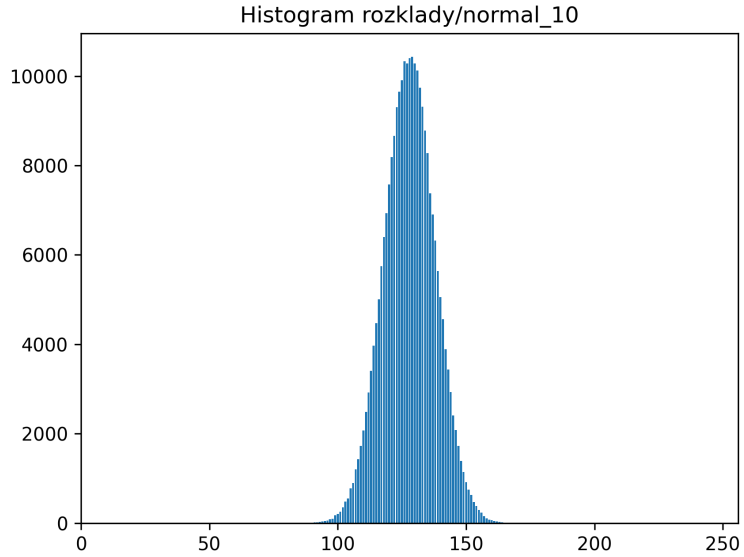
Rysunek 6: laplace_10.pgm - histogram



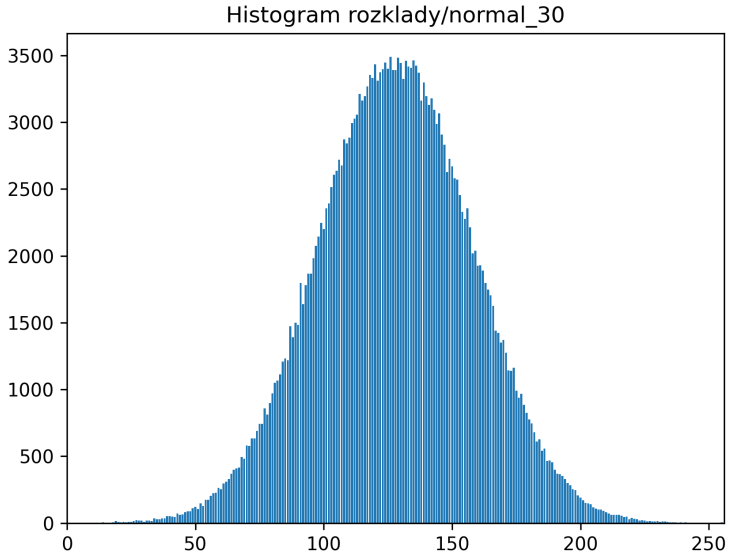
Rysunek 7: laplace_20.pgm - histogram



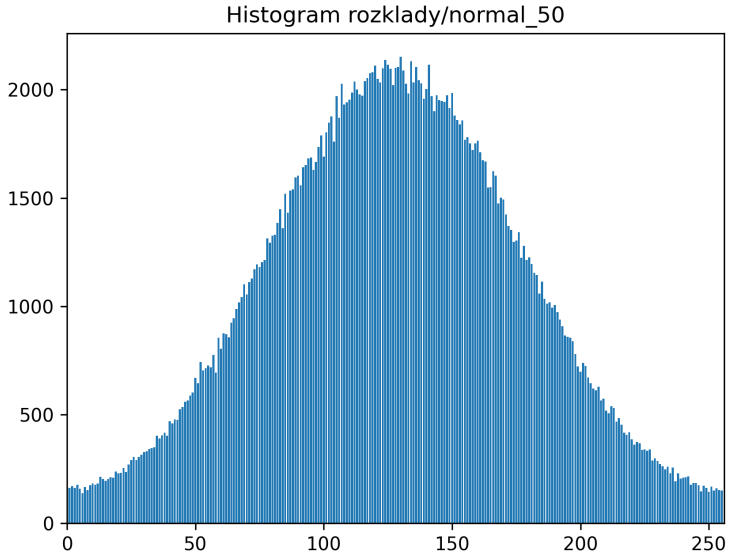
Rysunek 8: laplace_30.pgm - histogram



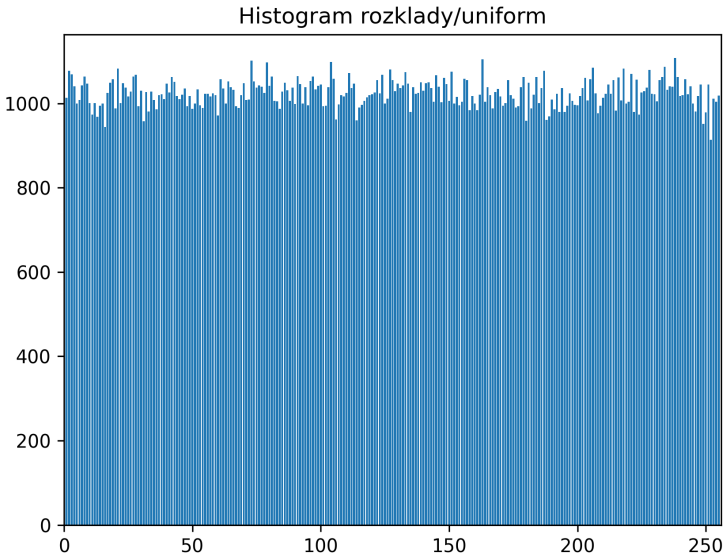
Rysunek 9: normal_10.pgm - histogram



Rysunek 10: normal_30.pgm - histogram

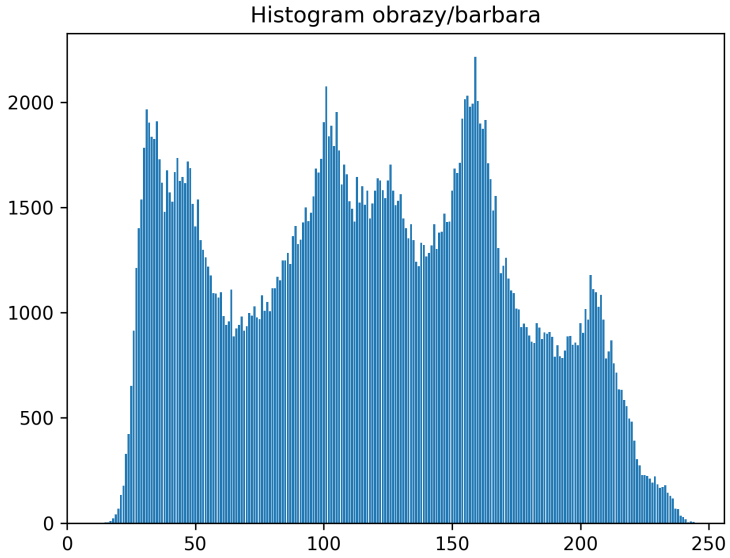


Rysunek 11: normal_50.pgm - histogram

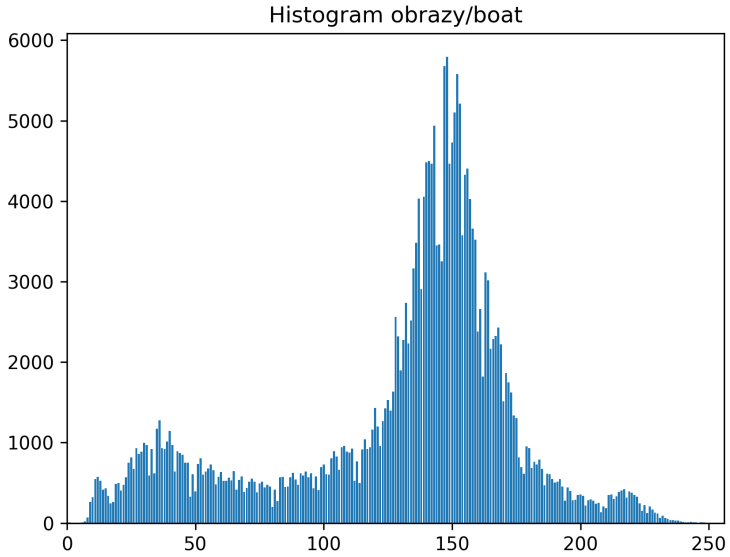


Rysunek 12: uniform.pgm - histogram

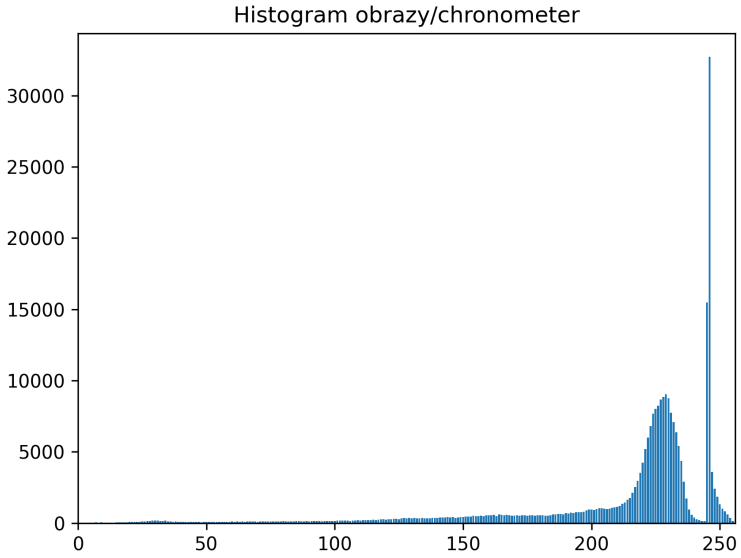
Entropia obrazów



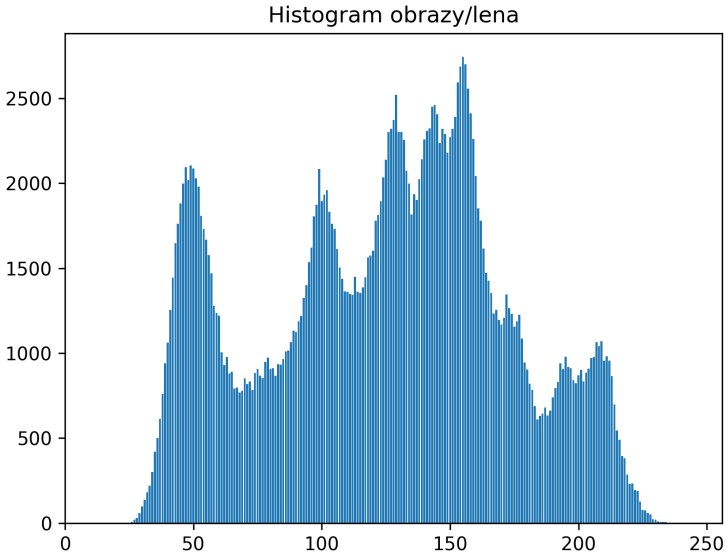
Rysunek 13: barbara.pgm - histogram



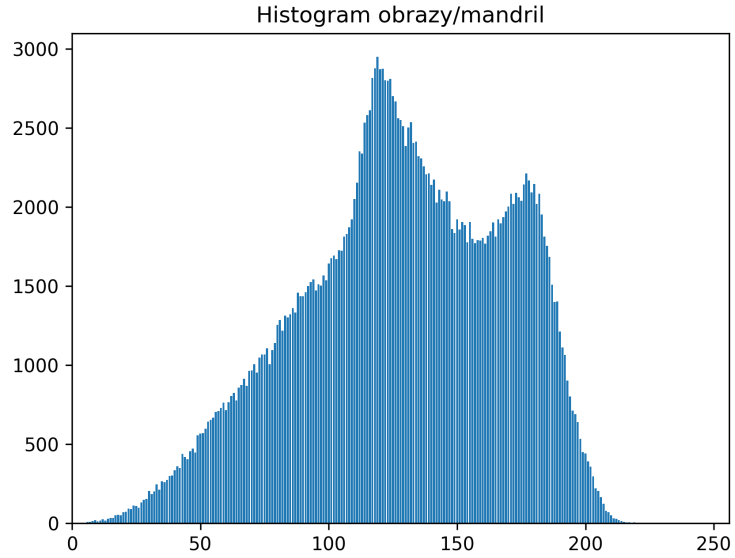
Rysunek 14: boat.pgm - histogram



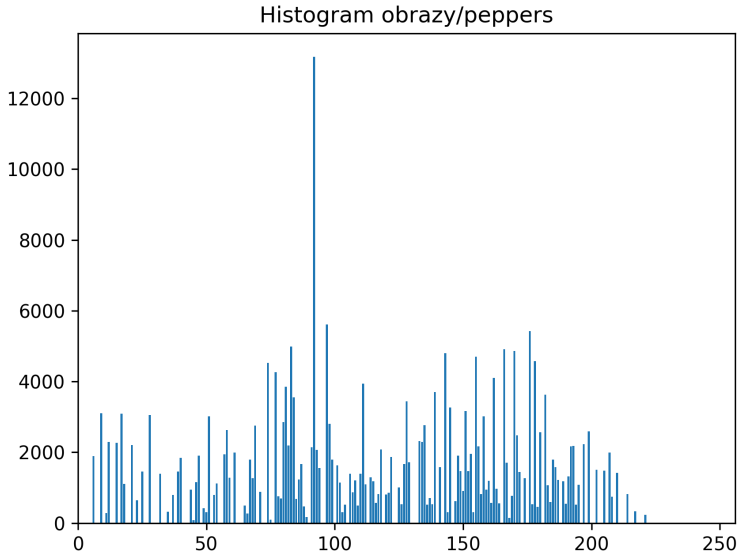
Rysunek 15: chronometer.pgm - histogram



Rysunek 16: lena.pgm - histogram



Rysunek 17: mandril.pgm - histogram



Rysunek 18: peppers.pgm - histogram

Entropia źródła blokowego rzędu 2 i 3

Tabela Entropii danych wejściowych

Plik	Entropia	Entropia Źródła blokowego 2 rzędu	Entropia Źródła blokowego 3 rzędu
barbara.pgm	7.632119011	8.830805669	11.44049797
boat.pgm	7.191370218	8.305931861	11.11675819
chronometer.pgm	6.113328353	6.649469668	8.413736565
lena.pgm	7.445061353	8.260271592	10.9359003
mandril.pgm	7.292548775	8.781243037	11.53755631
peppers.pgm	6.762425168	6.935477559	8.894681322
geometr_05.pgm	2.000995001	2.773701012	4.158251694
geometr_09.pgm	4.693658432	6.497106357	9.585845249
geometr_099.pgm	7.65440713	9.537849618	12.35606984
laplace_10.pgm	5.767144375	7.959112291	11.30569869
laplace_20.pgm	6.756458638	9.09743682	12.16752892
laplace_30.pgm	7.287118849	9.46102916	12.32849581
normal_10.pgm	5.368684963	7.432366604	10.8641293
normal_30.pgm	6.95431406	9.432597371	12.3270292
normal_50.pgm	7.649568445	9.667618415	12.39125859
uniform.pgm	7.999317791	9.672492641	12.39279973

Obserwacje

Wśród obrazów najmniejszą entropią cechowały się pliki chronometer i peppers, natomiast wśród danych losowych geometr_05 następnie geometr_09, normal_10 i laplace_10.

Największy wpływ na wyniki kompresji miał rodzaj danych testowych oraz ich entropia:

- Obrazy o najniższej wartości entropii pozwalały na uzyskanie mniejszego rozmiaru pliku (chronometer.pgm, peppers.pgm), a te o wyższej wartości nie doświadczały zmian w rozmiarze.
- W przypadku danych losowych sytuacja miała się podobnie – plik o najmniejszej entropii (geometr_05.pgm) dawał największy stopień kompresji – wynika to też z tego, że plik geometr_05.pgm cechuje się najmniejszą ilością różnych znaków.
- Widać tu więc potwierdzenie teorii stojącej za kodowanie LZSS – powtarzające się dane są tokenizowane (o ile token nie zajmuje więcej niż zastępowane przez niego dane) dzięki czemu widać zależność: im większa ilość powtarzających się danych w pliku, tym wyższy stopień kompresji.

Dla każdego pliku wejściowego uzyskana **średnia bitowa jest większa od entropii**, wynika to z bezstratności kompresji LZSS.

W przypadku przygotowanej implementacji czasy kodowania i dekodowania są dosyć zbliżone, choć według teorii dekoder powinien wykonywać się szybciej, gdy nie musi przeszukiwać słownika tak jak ma to miejsce w koderze, gdzie szukamy najdłuższej frazy. W naszym przypadku zbliżone czasy wynikają z faktu, że mierzony czas wykonania kodera nie uwzględnia czasu poświęconego na odczytanie całego pliku wejściowego. Natomiast dekoder na bieżąco odczytuje kolejne porcje danych z pliku wejściowego i po ich zdekodowaniu są one zapisywane do pliku wyjściowego, stąd też operacje te są uwzględnione w czasie wykonania. Należy zatem zakładać, iż operacje wejścia/ wyjścia zajmując najwięcej czasu podczas pracy dekodera. W przypadku dekodera można powiedzieć, że poświęcamy więcej czasu na czytanie i zapis, ale za to podczas wykonania zajmujemy mniej pamięci. Odwrotna sytuacja jest w koderze, gdzie wczytując cały plik do pamięci podręcznej zyskujemy na czasie wykonania poprzez eliminację częstych operacji odczytu z dysku.

Ocena efektywności algorytmu do kodowania obrazów naturalnych.

Jak wykazały testy algorytmu LZSS pomimo używania różnych rozmiarów słownika i bufora frazy, dla większości danych testowych, które stanowiły obrazy pliki wyjściowe były większe od plików wejściowych. Można zatem stwierdzić, iż kodowanie LZSS nie jest najlepszym rozwiązaniem do kodowania obrazów naturalnych. Prawdopodobnie algorytm LZSS lepiej sprawdza się przy danych tekstowych, w których to zwykle powtarzają się pewne sekwencje słów.