Sprawozdanie

Zadanie 4: Mapowanie metodą częściowego trawienia

I. Opis algorytmu

a) Wczytanie instancji

Wczytywanie danych z pliku następuje poprzez wywołanie odpowiedniej funkcji. Do zadeklarowanego globalnie wektora, który będzie przechowywał multizbiór, wczytywany jest element po elemencie aż do momentu, w którym kończy się plik.

b) Sprawdzanie liczby cięć za pomocą symbolu Newtona

W pętli *while*, która wykonuje się do momentu, aż zmienna *qualifiedForMapping* ma wartość *false* i liczba cięć *k* jest mniejsza od rozmiaru całego multizbioru, najpierw zwiększana jest wartość zmiennej *k* (która przed rozpoczęciem pętli została zadeklarowana z wartością 0), a następnie wywoływana jest funkcja sprawdzająca, czy po podłożeniu danej wartości zmiennej *k* do symbolu Newtona wynik działania będzie równy rozmiarowi całego multizbioru. Symbol Newtona podany w treści zadania został rozpisany w następujący sposób:

$$\binom{k+2}{2} = \frac{(k+2)!}{2\,k!}$$

W funkcji najpierw liczona jest za pomocą pętli *for* silnia dla *k+2* w taki sposób, że zmiennej *factorial*, (której przed rozpoczęciem pętli została przypisana wartość 1) przypisywana jest wartość *factorial* pomnożonego przez iterowane w pętli *i*, przy czym iteracja *i* w pętli zaczyna się od wartości 1, a pętla będzie się wykonywać do momentu, w którym *i* przekroczy wartość *k+2*. W analogiczny sposób liczona jest silnia dla samego *k*. Następnie liczona jest wartość symbolu Newtona zgodnie z rozpisanym powyżej wzorem. W przypadku, gdy wartość ta będzie równa rozmiarowi całego multizbioru, to zmiennej *qualifiedForMapping* przypisywana jest wartość *true*. Funkcja kończy swoje działanie, a pętla *while* jest przerywana. W przypadku, gdy nie istnieje liczba cięć przekładająca się na liczność multizbioru użytkownik zostaje o tym powiadomiony odpowiednim komunikatem, a następnie program kończy swoje działanie.

c) Konstruowanie mapy

Gdy multizbiór jest uznany za możliwy do zmapowania zostają znalezione dwa największe elementy w nim występujące. Ich różnica jest pierwszym elementem dodanym do globalnie zadeklarowanych dwóch wektorów: jeden z nich przechowuje elementy mapy, a drugi - elementy multizbioru, które zostały już użyte. Zmiennej *maxInd* oznaczającej maksymalny poziom poszukiwań (a przy okazji też i rozmiar całej mapy) przypisywana jest wartość *k+1*. Deklarowana jest również zmienna *mapExists* o wartości *false*. Zostaje uruchomiony zegar, a po tym następuje pierwsze wywołanie funkcji *search*, do której przekazywane są numer aktualnie przeszukiwanego poziomu, czyli 1, a także zmienna *mapExists*.

Funkcja *search* zaczyna swoje działanie od sprawdzenia, czy aktualnie przeszukiwany poziom (czyli zmienna *ind*) jest równy zmiennej *maxInd*. Jeżeli tak, to do jednego kontenera typu *set* przekazywane są wszystkie elementy multizbioru, a do drugiego - wszystkie elementy wektora z użytymi elementami. Własności kontenera *set* pozwalają na ułożenie elementów w kolejności rosnącej, efektem czego po dodaniu wszystkich elementów są one od razu posortowane. Tak powstałe kontenery są ze sobą porównywane i jeżeli ich zawartość jest taka sama, to mapa zostaje uznana za znalezioną. Następuje wypisanie rezultatu, a funkcja kończy działanie. Użycie kontenerów typu *set* dla wektorów z multizbiorem i z użytymi elementami stanowi dodatkowe sprawdzenie, czy wszystkie elementy w mapie zostały prawidłowo dodane - w przypadku, gdy mapa zostanie źle skonstruowana, wektor z elementami użytymi powstałym na jej podstawie będzie różnił się zawartością od wektora z elementami multizbioru. W takim przypadku mapa poszukiwana jest dalej.

Jeżeli indeks obecnie przeszukiwanego poziomu jest mniejszy od wartości zmiennej *maxInd*, to w pętli *for* następuje sprawdzanie, czy kolejne elementy z multizbioru nadają się jako elementy mapy. W tym celu najpierw zliczana jest liczba wystąpień w multizbiorze obecnie sprawdzanego elementu, a następnie liczba wystąpień tego samego elementu w wektorze z elementami użytymi. Ma miejsce sprawdzenie, czy liczba wystąpień elementu w multizbiorze jest mniejsza od liczby wystąpień elementu w wektorze z elementami użytymi. Sprawdzane jest także, czy obecnie sprawdzany element z multizbioru jest równy wartości zmiennej *el_checked*, która przy wywołaniu funkcji *search* przyjmuje wartość 0, jednak w przypadku, gdy element z multizbioru nie może zostać do mapy, to zmienna ta przyjmuje jego wartość. Jeśli jeden z tych warunków jest spełniony, to pętla jest

kontynuowana. W innym wypadku element może zostać uznany za potencjalnego kandydata do mapy, dlatego zostaje do niej dodany. Przed dodaniem takiego elementu do wektora z elementami użytymi zostaje zapisany rozmiar tego wektora. Rozmiar ten zostanie użyty w przypadku, gdy obecnie sprawdzany element nie spełni warunków do zostania elementem mapy. W ten sposób z wektora z użytymi elementami zostaną poprawnie usunięte wszystkie elementy, które zostały dodane przy sprawdzaniu tego konkretnego elementu.

Po dodaniu obecnie sprawdzanego elementu do obu wektorów następuje zliczanie sum. Zmiennej *sum* przypisywana jest wartość obecnie sprawdzanego elementu, a następnie w pętli *for* ma miejsce przechodzenie przez każdy element wektora z mapą od końca zaczynając od elementu przedostatniego. Pętla ta zaczyna się od dodania do zmiennej *sum* wartości obecnie iterowanego elementu z mapy. Następnie zliczana jest liczba wystąpień obecnie sprawdzanej sumy w multizbiorze oraz w wektorze z użytymi elementami w sposób analogiczny do tego, który został opisany wyżej. Jeżeli liczba wystąpień danej sumy w multizbiorze jest większa od liczby wystąpień tej sumy w wektorze z elementami użytymi, to suma ta zostaje dodana do wektora z elementami użytymi, a pętla wykonuje następną iterację w celu sprawdzenia kolejnej sumy. W innym wypadku zmiennej *next* (która wskazuje na to, czy warto kontynuować bieżące rozwiązanie) przypisywana jest wartość *true*. Z wektora z elementami użytymi zostają usunięte wszystkie dodane elementy w tej iteracji pierwszej pętli *for*, a następnie druga pętla *for* (obliczająca sumy) zostaje przerwana.

Po zakończeniu pętli *for* obliczającej sumy następuje sprawdzenie, czy wartość zmiennej *next* ma wartość *false* - jeżeli tak, to ma miejsce obliczanie odległości do prawego końca mapy: od wartości największego elementu w mapie zostaje odjęta ostatnia obliczona suma (czyli suma wszystkich obecnie znajdujących się elementów w wektorze z mapą). Następnie ma miejsce sprawdzenie, czy tak obliczona odległość znajduje się w multizbiorze i w wektorze z elementami użytymi. Dzieje się to w taki sam sposób, co poprzednio, czyli poprzez zliczanie wystąpień tej odległości w obu wektorach. W przypadku, gdy liczba wystąpień odległości w multizbiorze jest większa od liczby wystąpień w wektorze z elementami użytymi, to następuje wywołanie funkcji *search*, do której przekazywana jest wartość *ind+1*. Kiedy jednak poszukiwany jest ostatni element mapy to nie spełni on takiego warunku. Z tego powodu dodane są inne warunki, które będzie spełniał tylko ostatni element z mapy. Sprawdzane jest zatem, czy

wartość zmiennej *ind* jest równa wartości *maxInd-1*, czy odległość od prawego końca mapy jest równa 0 oraz czy ostatnia obliczona suma jest równa wartości największego elementu w mapie. W przypadku, gdy te wszystkie warunki są spełnione, to tak samo wywoływana jest funkcja *search*, do której przekazywana jest zmienna *ind+1*. Kiedy żaden z tych warunków nie jest spełniony, to element nie powinien zostać dodany do mapy i nie warto kontynuować tego rozwiązania. Dlatego też wartość obecnie sprawdzanego elementu zostaje przypisana do wcześniej wspomnianej zmiennej *el_checked*, a on sam zostaje usunięty z mapy, ma także miejsce usunięcie z wektora z elementami użytymi wszystkich elementów, które zostały dodane w tej iteracji funkcji *for.* Następuje sprawdzenie kolejnego elementu z multizbioru.

Po zakończeniu wywoływania wszystkich funkcji *search* ma miejsce zatrzymanie zegara i zostaje obliczony czas, w którym zostało znalezione rozwiązanie. Jeżeli udało się znaleźć mapę, to czas ten zostaje wypisany na ekranie.

II. Zawartość wygenerowanych instancji

instance1.txt
2 4 4 5 6 6 8 9 10 10 11 12 14 14 18 18 19 20 23 24 29

instance2.txt
2 4 6 10 17 23 26 27 27 28 29 32 33 37 39 50 54 55 56 65 82

instance3.txt
5 10 19 27 27 32 33 46 48 58 60 75 79 93 103 108 120 123 125 130 133 135 135 151 154 156 166 178 183 183 193 202 212 226 253 258 259 286 286 291 305 313 318 332 337

instance4.txt
20 22 30 43 52 66 85 93 95 99 105 109 113 115 135 142 145 165 180 194 198 200 208 220 237 250 279 293 299 303 315 322 342 345 388 392 408 414 435 444 457 487 501 523 553

instance5.txt
2 2 3 5 5 8 8 10 10 12 13 13 14 15 15 15 16 17 17 17 18 19 19 21 21 22 25 27 29 31 32 32 32 33 34 34 35 37 40 46 47 47 48 48 50 51 53 55 56 63 64 65 65 66 67 68 69 72 77 79 80 80 82 82 85 85 87 87 90 95 97 97 99 100 102 104 112 114

instance6.txt
2 2 3 4 4 5 7 7 9 11 12 14 15 16 18 18 20 21 21 22 23 25 26 27 30 30 32 34 35 36 38 39 39 39 40 41 42 42 44 45 46 46 47 48 49 49 51 51 51 53 53 55 56 57 59 60 63 67 69 71 74 80 81 83 85 85 86 87 90 93 95 97 102 106 110 125 132 136

instance7.txt
10 10 13 13 14 15 15 16 18 19 21 23 25 26 27 29 31 32 33 34 38 39 39 41 42 43 44 45 49 49 52 52 53 57 61 62 62 63 66 68 72 73 76 76 77 78 82 83 85 88 91 92 93 94 95 96 99 101 101 105 110 111 114 114 115 115 117 118 124 125 126 128 129 131 132 133 134 138 139 144 145 146 146 153 154 156 158 159 160 164 167 167 171 172 175 177 177 187 190 192 193 202 206 208 211 219 221 229 232 244 245 247 255 260 263 270 270 285 289 304

instance8.txt
2 3 3 5 6 6 8 9 10 12 15 15 17 17 18 18 20 20 21 23 26 26 27 27 28 28 29 32 32 35 38 38 39 43 44 45 46 47 49 49 49 51 52 53 54 55 55 57 58 59 59 60 64 70 76 76 77 77 77 78 81 83 84 86 86 87 94 96 98 103 103 104 104 104 106 109 112 113 115 121 123 124 126 129 130 131 132 133 135 136 136 139 141 141 147 151 153 156 159 161 162 163 164 167 174 176 179 179 180 180 182 182 185 188 190 194 200 218 233 239

III. Wyniki przeprowadzonych testów

W przypadku instancji wygenerowanych własnoręcznie, **Czas 1** oznacza czas znalezienie rozwiązania dla instancji posortowanej rosnąco, natomiast **Czas 2** to czas dla instancji, w której kolejność elementów była losowa.

Instancja: instance1.txt
Mapa: 5, 4, 2, 8, 4, 6
Czas 1: 0,01s
Czas 2: 0,01s

Instancja: instance2.txt
Mapa: 17, 10, 23, 4, 2, 26
Czas 1: 0,01s
Czas 2: 0,01s

Instancja: instance3.txt
Mapa: 5, 27, 93, 10, 48, 75, 33, 27, 19
Czas 1: 0,01s
Czas 2: 0,03s

Instancja: instance4.txt
Mapa: 30, 22, 93, 20, 85, 95, 99, 43, 66
Czas 1: 0,01s
Czas 2: 0,02s

Instancja: instance5.txt
Mapa: 2, 12, 3, 2, 8, 8, 13, 19, 15, 17, 5, 10
Czas 1: 0,09s
Czas 2: 3,32s

Instancja: instance6.txt
Mapa: 4, 7, 15, 20, 3, 2, 2, 14, 18, 12, 9, 30
Czas 1: 0,02s
Czas 2: 0,21s

Instancja: instance7.txt
Mapa: 15, 26, 42, 10, 21, 18, 14, 13, 16, 33, 39, 13, 10, 15, 19
Czas 1: 0,09s
Czas 2: 5,65s

Instancja: instance8.txt
Mapa: 6, 15, 28, 10, 17, 28, 32, 17, 9, 18, 2, 3, 3, 12, 39
Czas 1: 0,01s
Czas 2: 5,42s

Instancje z treści zadania

Instancja: ins-PDP-11a-asc.txt
Mapa: 4, 6, 5, 8, 3, 9, 5, 2, 4, 7, 8, 6
Czas znalezienia rozwiązania: 0,21s

Instancja: ins-PDP-12a-asc.txt
Mapa: 4, 6, 5, 8, 3, 9, 5, 2, 4, 7, 8, 6, 6
Czas znalezienia rozwiązania: 0,39s

Instancja: ins-PDP-13a-asc.txt
Mapa: 3, 6, 6, 8, 7, 4, 2, 5, 9, 3, 8, 5, 6, 4
Czas znalezienia rozwiązania: 0,62s

Instancja: ins-PDP-14a-asc.txt
Mapa: 4, 6, 5, 8, 3, 9, 5, 2, 4, 7, 8, 6, 6, 3, 5
Czas znalezienia rozwiązania: 5,87s

Instancja: ins-PDP-11b-asc.txt
Mapa: 38, 74, 27, 66, 42, 15, 89, 47, 35, 13, 12, 54
Czas znalezienia rozwiązania: 0,02s

Instancja: ins-PDP-11b-desc.txt
Mapa: 38, 74, 27, 66, 42, 15, 89, 47, 35, 13, 12, 54
Czas znalezienia rozwiązania: 0,18s
Instancja: ins-PDP-14b-asc.txt
Mapa: 54, 12, 13, 35, 47, 89, 15, 42, 66, 27, 74, 38, 25, 57, 79
Czas znalezienia rozwiązania: 0,02s

Instancja: ins-PDP-14b-desc.txt
Mapa: 54, 12, 13, 35, 47, 89, 15, 42, 66, 27, 74, 38, 25, 57, 79
Czas znalezienia rozwiązania: 1,94s

IV. Wnioski

Podczas testowania programu zarówno na instancjach zamieszczonych w zadaniu, jak i na tych wygenerowanych własnoręcznie, udało mi się zauważyć kilka własności. Jedną z nich jest fakt, że w przypadku instancji, w których kolejność elementów jest losowa program działa znacznie dłużej w porównaniu do instancji, w których elementy są posortowane (rosnąco lub malejąco). Z kolei podczas testowania instancji *ins-PDP-14a-asc.txt* zauważyłam, że w przeciwieństwie do innych posortowanych instancji czas wyszukiwania tego rozwiązania jest znacznie dłuższy. Po porównaniu instancji *ins-PDP-14a-asc.txt* z innymi instancjami doszłam do wniosku, że tak długi czas wyszukiwania może być spowodowany tym, że w *ins-PDP-14a-asc.txt* istnieje bardzo dużo powtarzających się elementów. Kolejną rzeczą, która się potwierdziła, a którą podejrzewałam jeszcze przed wykonaniem jakichkolwiek testów, jest fakt, że im więcej cięć w mapie, a co za tym idzie - im dłuższa instancja, tym dłuższy będzie czas działania algorytmu.

Jednak najważniejszą rzeczą, którą udało mi się zauważyć jeszcze podczas pisania kodu i testowania działania kolejnych funkcji mojego programu, jest fakt jak bardzo istotnym okazało się dodanie wspomnianej w opisie algorytmu zmiennej *el_checked*. Przed umieszczeniem jej w kodzie rozwiązania do wszystkich instancji wyszukiwały się w czasie znacznie dłuższym. Największą różnicę zauważyłam przy testowaniu wcześniej już wspomnianej instancji *ins-PDP-14a-asc.txt* - przed dodaniem zmiennej *el_checked* rozwiązanie do tej instancji wykonywało się w czasie dłuższym niż 5000s, co przekłada się na około 1,5 godziny. Po dodaniu zmiennej *el_checked* czas ten skrócił się do niecałych 6s, czyli ponad 800 razy krócej. Po zauważeniu tak znacznej różnicy doszłam do wniosku, że aby skrócić czas działania algorytmu istotne jest w pewnym sensie "naprowadzenie" go w kierunku prawidłowego rozwiązania poprzez eliminowanie takich poszczególnych rozwiązań, które on sam uznał za nieprawidłowe.