piątek, 22 stycznia 2016

Wiktor Adamski

Zadanie dodatkowe na pracownię SO – pamięć stosowa

„Jeżeli zabałaganione biurko jest oznaką zabałaganionego umysłu, oznaką czego jest puste biurko? ”

Albert Einstein

Treść zadania

Oprogramuj podsystem zarządzania pamięcią zorganizowaną w n stosów. Początkowo stosy są puste. W trakcie działania systemu stosy przyrastają (lub maleją) w różnym tempie. W odniesieniu do danego stosu zamówienia mogą być różnej wielkości. Jeśli wyrażone w bajtach zamówienie na pamięć dla któregoś ze stosów nie może być zrealizowane, ponieważ blok wolnej pamięci powyżej stosu jest za mały, należy wykonać procedurę reorganizacji całej pamięci, kierując się zasadą przydziałów proporcjonalnych.

Doprecyzowanie zadania

W niniejszej symulacji rolę pamięci pełni tablica zmiennych typu `Int32`, także w rzeczywistości pamięć jest podzielona na indeksowane bloki po 4 bajty. Każdy ze stosów zawiera informacje o indeksie początkowym, rozmiarze, liczbie pozostałych bloków pamięci, oraz tempie przyrostu danych w stosie.

Reorganizacja pamięci

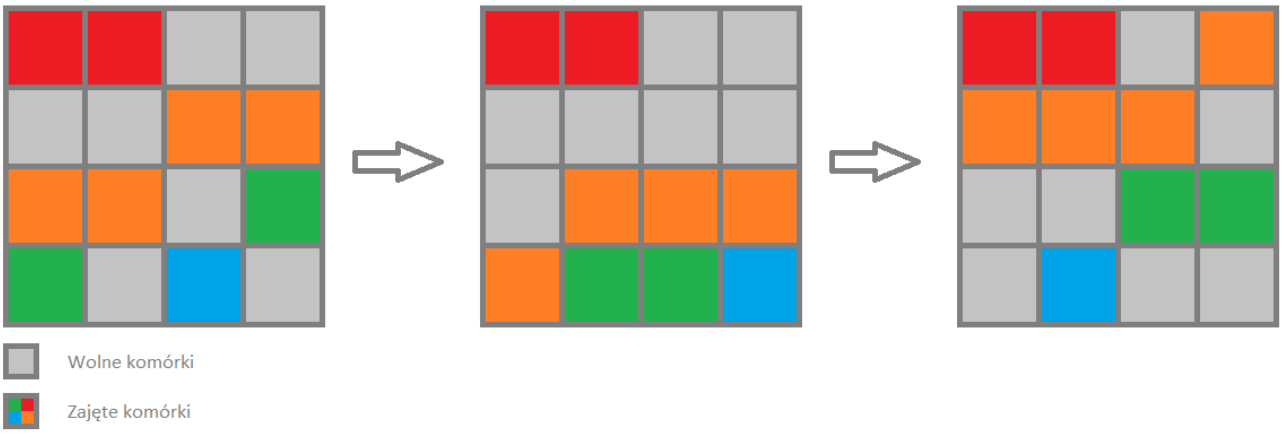
Największym problemem zadania jest opracowanie algorytmu, który przy zgłoszeniu przez stos braku miejsca tak poprzesuwa dane w pamięci aby stosy mogły dalej rosnąć. W tym celu dla każdego stosu obliczamy wartość `nowaPojemnosc` według poniższego wzoru.

```
nowaPojemnosc = (Int) (przyrost * wolnaPamiec / calkowityPrzyrost)
```

Gdzie `calkowityPrzyrost` jest sumą przyrostów ze wszystkich stosów. Może się zdarzyć, że w wyniku rzutowania na typ `Int`, nie wszystkie komórki pamięci zostaną przydzielone. Ich liczba zostanie zapamiętana i uwzględniona przy następnej reorganizacji. `nowaPojemnosc` dla stosu który zgłosił brak miejsca jest odpowiednio większa, aby zaspokoić wcześniejsze zgłoszenie. Jeśli w wyniku obliczania nowych pojemności przekroczymy pojemność pamięci, zgłoszony zostanie błąd krytyczny. W następnym kroku przechodzimy do faktycznego przegrupowania danych. Ponieważ pierwszy stos zawsze zaczyna się od komórki nr 0, nie musimy go uwzględniać w algorytmie przenoszenia.

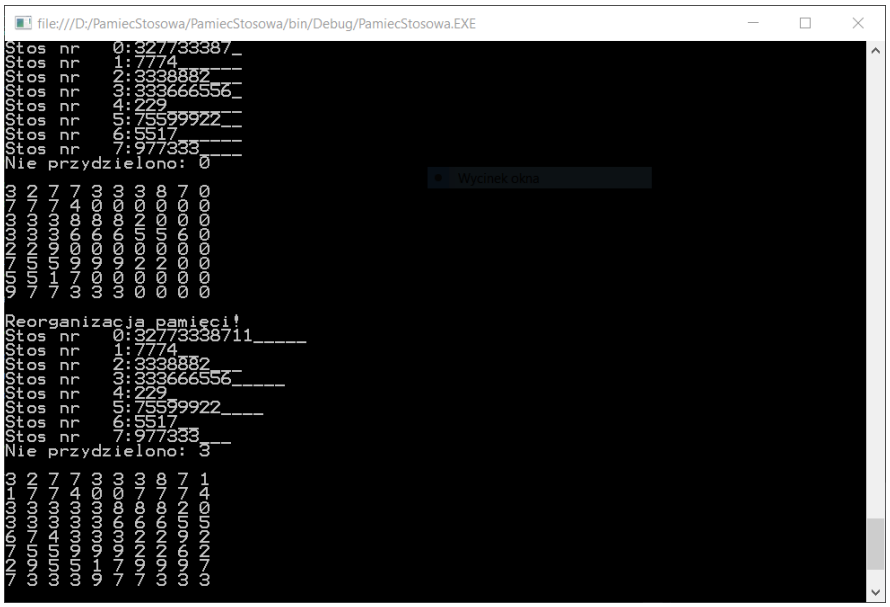
1. Zaczynając od ostatniego stosu, przenieś zawartość zajętych komórek na koniec pamięci, upakowując je bez wolnych miejsc między nimi.
2. Zaczynając od drugiego stosu przenieś zawartość tylu komórek z końca ile wynosił poprzedni rozmiar na nową lokalizację stosu.

Postępując w ten sposób nie dopuszczamy do sytuacji, w której zawartość jednego stosu jest zapisana przez stos wcześniejszy, który musiał zostać przesunięty w kierunku końca pamięci. Przy każdej reorganizacji wartość `przyrost` zostaje pomnożona przez 0.8, aby nowo dodane dane miały większy wpływ na określanie przydziału miejsca przy następnym wykonaniu algorytmu.



Ilustracja działania algorytmu przesuwania danych

Działanie programu



Na powyższym zrzucie ekranu widać, jak przed działaniem algorytmu wszystkie stosy miały równą wielkość. Po wykonaniu algorytmu, w wolnych komórkach pamięci znajdują się pozostałości po przenoszeniu, jednakże dzięki wskaźnikom stosów, rzeczywiste dane nie uległy zniekształceniu (znak `_` oznacza wolną komórkę przydzieloną do danego stosu). Można zauważyć również, że w wyniku nowego przydziału pozostały 3 komórki nienależące do żadnego stosu. W tej wersji algorytmu nie będą one modyfikowane do momentu następnego przemieszczania stosów, ale można łatwo zmodyfikować kod, tak aby np. zostały przydzielone za każdym razem do ostatniego stosu.

Uwagi praktyczne

Tak jak w przypadku poprzedniego zadania, zamieszczam obok kodu źródłowego, skompilowany program. Po uruchomieniu wyświetli się komunikat na temat stanu początkowego pamięci i nastąpi oczekiwanie na naciśnięcie przycisku. Naciśnięcie klawisza `P` spowoduje wyświetlenie zawartości pamięci. Klawisz `X` odpowiada za zakończenie pracy programu. Naciśnięcie pozostałych klawiszy sprawi, że do pamięci zostanie dodana losowo wygenerowana informacja i ponownie wyświetli się komunikat o zawartości. Jeśli nastąpi zapełnienie pamięci, zostanie wyświetlony stosowny komunikat i program zakończy działanie.