Dokumentacja

Sebastian Pietras

1 Temat projektu

Projekt nr 8: Chemikalia

Zakłady chemiczne produkują n różnych substancji, z których każda składowana jest w jednym z dwóch magazynów. Ze względu na niebezpieczne opary, które mogą ze sobą reagować, niektóre substancje nie mogą znajdować się w tym samym magazynie. Napisać algorytm, który w bezpieczny sposób (o ile to możliwe) rozmieszcza każdą z substancji, mając na wejściu dane o parach, które nie mogą być przechowywane razem.

Przykładowe dane wejściowe:

```
1 4
2 4
3 5
```

Jedno z poprawnych rozwiązań:

```
1 2 3
4 5
```

2 Uruchomienie

Istnieją trzy tryby wykonania:

1. Rozwiązanie problemu ze standardowego wejścia

   ```
   python3 program.py m1 n << i
   ```

   - n - liczba substancji
   - i - restrykcje

2. Rozwiązanie wygenerowanego problemu

   Rozwiązywalnego:

   ```
   python3 program.py m2 solvable [-d] [-f] n
   ```

   Nierozwiązywalnego:

   ```
   python3 program.py m2 unsolvable [-d] n
   ```

   - n - liczba substancji
   - d - zagęszczenie restrykcji (ułamek maksymalnych)
   - f - liczba substancji w jednym z magazynów

3. Pomiar i testowanie rozwiązania dla rosnących rozmiarów problemów

   ```
   python3 program.py m3
   [-ins] [-r] [-v] n iterations step density
   ```

   - n - początkowa liczba substancji
   - iterations - ile razy zwiększyć rozmiar problemu
   - step - o ile zwiększyć ilość substancji w każdej iteracji
   - density - zagęszczenie restrykcji
   - ins - ile problemów wygenerować w każdej iteracji
   - r - ile razy powtarzać rozwiązanie każdego problemu
   - v - 0 nic nie wypisuje, 1 wypisuje nagłówek z podstawowymi informacjami, 2 wypisuje informacje po każdej iteracji

3 Założenia o danych wejściowych

Wejściem powinny być niepowtarzające się pary liczb odpowiadających substancjom, które nie mogą być ze sobą w magazynie, oddzielone spacją. Kolejne pary powinny być oddzielone znakiem nowej linii.

Na przykład:

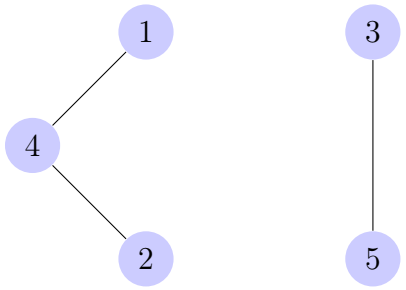
```
1 4
2 4
3 5
```

Liczby odpowiadające substancjom powinny być naturalne i nie przekraczać liczby wszystkich substancji. Jedyne dopuszczalne znaki na wejściu to: cyfry, spacja i znak nowej linii.

4 Rozwiązanie

Znając dane wejściowe, stwórzmy graf o n wierzchołkach. Będzie on reprezentowany listą sąsiedztwa. To znaczy, że powstanie struktura, która dla każdego wierzchołka będzie pamiętała zbiór wierzchołków z nim połączonych. Połączmy krawędzią te wierzchołki, które zostały wymienione na wejściu.

Dla przykładowych danych wejściowych otrzymamy następujący graf:



Graf jest reprezentowany przez strukturę:

```
1: {4}
2: {4}
3: {5}
4: {1, 2}
5: {3}
```

Aby stworzyć tę strukturę wystarczy utworzyć zbiory dla każdego wierzchołka i dla każdej krawędzi z wejścia dodawać odpowiedni wierzchołek do odpowiedniego zbioru.

Każda krawędź reprezentuje ograniczenie substancji. To znaczy, że wierzchołki połączone ze sobą krawędzią nie mogą być ze sobą w magazynie. Jeżeli uda nam się udowodnić, że graf jest dwudzielny i wyznaczyć podział wierzchołków na dwa podzbiory to w ten sposób rozwiążemy problem. Te dwa podzbiory będą reprezentować rozłożenie substancji w dwóch magazynach, ponieważ wtedy połączone będą ze sobą jedynie wierzchołki z dwóch różnych podzbiorów, czyli wszystkie restrykcje zostaną spełnione.

Aby sprawdzić dwudzielność i wyznaczyć podział spróbujmy pokolorować wierzchołki grafu na dwa kolory. Możemy to zrobić za pomocą algorytmu BFS (przeszukiwanie wszerz).

Zauważmy, że nasz graf nie musi być spójny, może mieć wiele składowych. W szczególności może na przykład nie mieć żadnych krawędzi. Algorytm BFS za jednym razem jest w stanie przeszukać jedynie jedną spójną składową grafu. Z tego powodu trzeba pamiętać, które wierzchołki zostały już przeszukane. Wtedy pomijając te wierzchołki trzeba wykonywać algorytm tak długo, aż nie przeszuka wszystkich.

Działanie algorytmu BFS oparte jest na kolejce. Pobieramy wierzchołek z kolejki i dodajemy do kolejki wszystkich sąsiadów, którzy nie są jeszcze pokolorowani. Od razu kolorujemy ich innym kolorem niż kolor pobranego wierzchołka. W ten sposób wszystkie wierzchołki w kolejce są pokolorowane (z tym, że na początku trzeba pokolorować wierzchołek startowy algorytmu i wrzucić go do kolejki). Jeśli któryś z sąsiadów pobranego wierzchołka jest pokolorowany na ten sam kolor co wierzchołek, to znaczy że nie można pokolorować grafu na dwa kolory i nie można rozdzielić substancji na dwa magazyny. Możemy w tym przypadku zwrócić wadliwą krawędź i dotychczasowe rozłożenie na magazyny. Algorytm powtarzamy dopóki kolejka nie będzie pusta. Na końcu zwracamy zbiory wierzchołków pokolorowanych na te same kolory i dodajemy jeden z nich do jednego magazynu a drugi do drugiego.

Zbiory kolorów zaimplementowane są jako set (tablica hashująca) a kolejka jako deque (kolejka dwukierunkowa), co pozwala potrzebnym operacjom mieć średni koszt jednostkowy.

Przykład:

Graf dla przypomnienia:

```
1: {4}
2: {4}
3: {5}
4: {1, 2}
5: {3}
```

Prześledźmy po kolei kroki algorytmu:

Rozłożenia na magazyny początkowe (puste):

```
magazyn_a = {}
magazyn_b = {}
```

Żadne wierzchołki nie są jeszcze pokolorowane, bierzemy któryś z nich, na przykład (1), i uruchamiamy na nim algorytm kolorujący BFS:

Kolorujemy pierwszy wierzchołek (obojętnie na jaki kolor, niech będzie 1) i wrzucamy do kolejki:

```
kolejka = [1]
kolor1 = {1}
kolor2 = {}
```

Kolejka nie jest pusta, bierzemy wierzchołek z kolejki (1). Żaden z sąsiadów nie jest pokolorowany. Kolorujemy sąsiadów na kolor inny niż kolor wierzchołka (wiemy, który to kolor, bo wiemy czy wierzchołek znajduje się w zbiorze kolor1 czy kolor2) i dodajemy ich do kolejki:

```
kolejka = [4]
kolor1 = {1}
kolor2 = {4}
```

Kolejka nie jest pusta, bierzemy wierzchołek z kolejki (4). Sąsiad (1) jest już pokolorowany (znajduje się w zbiorze kolor1), więc go nie rozpatrujemy. Kolorujemy sąsiadów na kolor inny niż kolor wierzchołka i dodajemy ich do kolejki:

```
kolejka = [2]
kolor1 = {1, 2}
kolor2 = {4}
```

Kolejka nie jest pusta, bierzemy wierzchołek z kolejki (2). Wszyscy sąsiedzi są pokolorowani. Przechodzimy dalej:

```
kolejka = []
kolor1 = {1, 2}
kolor2 = {4}
```

Kolejka jest pusta, zwracamy zbiory kolor1 i kolor2.

Dodajemy zwrócone zbiory do magazynów:

```
magazyn_a = {1, 2}
magazyn_b = {4}
```

Do rozpatrzenia pozostaje zbiór wierzchołków:

```
{3, 5}
```

Bierzemy któryś z nich, na przykład (3), i ponownie uruchamiamy wyżej opisany algorytm. Zostanie zwrócone:

```
kolor1 = {3}
kolor2 = {5}
```

Dodajemy zwrócone zbiory do magazynów:

```
magazyn_a = {1, 2, 3}
magazyn_b = {4, 5}
```

Nie pozostały żadne wierzchołki do rozpatrzenia, możemy zakończyć działanie algorytmu, zwracając rozłożenia na magazyny.

5 Szacowana złożoność

Najpierw spróbujmy oszacować kres górny, używając algorytmu brutalnego, żeby mieć jakiś punkt odniesienia.

Możemy wygenerować wszystkie możliwe rozłożenia i sprawdzać czy dane rozłożenie jest prawidłowe. Wszystkich rozłożeń jest 2^n (wariacje z powtórzeniami). Następnie dla każdego rozłożenia trzeba sprawdzić, czy spełnione są wszystkie restrykcje podane na wejściu (przyjmijmy, że jest ich m). Ostatecznie algorytm ten ma złożoność $O(2^n m)$. Możemy też zauważyć, że maksymalnie restrykcji na wejściu będzie n^2, a maksymalna liczba restrykcji, które da się spełnić wynosi $\frac{n^2}{4}$. Daje to złożoność $O(2^n n^2)$.

Natomiast nasz algorytm dzieli się na dwa etapy: budowa grafu i kolorowanie. Przy budowie grafu trzeba najpierw utworzyć zbiory dla każdego wierzchołka, a potem iterować po wszystkich restrykcjach i obie liczby z pary dodać do odpowiedniego zbioru. Zbiory są zaimplementowane jako tablice hashujące, więc otrzymujemy średnią złożoność $O(n + m)$ i pesymistyczną $O(n + mn)$. Kolorowanie to algorytm BFS z dodanym sprawdzaniem kolorów. Musimy przeszukać wszystkie wierzchołki w grafie (których jest n) oraz dla każdego wierzchołka jego krawędzie, co po zsumowaniu da nam $2m$ rozpatrzonych krawędzi. Dla każdej krawędzi będziemy musieli sprawdzić czy jeden z wierzchołków należy do zbioru kolorów. Zbiory kolorów są zaimplementowane jako tablica hashująca, a kolejka wierzchołków jako kolejka dwukierunkowa, co daje nam średnią złożoność $O(n + m)$ i pesymistyczną $O(n + mn)$.

Składając te dwie części razem dostaniemy średnią złożoność $O(n + m)$ i pesymistyczną $O(n + nm)$. Przy maksymalnej ilości par dających rozwiązanie ($m = \frac{n^2}{4}$) dostaniemy średnią złożoność $O(n^2)$ i pesymistyczną $O(n^3)$. W każdym wypadku złożoność naszego algorytmu jest lepsza niż brutalnego.

Warto dodać, że w rzeczywistości ilość substancji jest mniejsza niż modulo funkcji hashującej, więc kolizje w tablicach hashujących w tym przypadku praktycznie w ogóle nie wystąpią. Z tego powodu jako praktyczną złożoność trzeba wskazać średnią złożoność $O(n + m)$.

6 Generacja danych testowych

Generator na wyjściu poda wylosowane pary substancji, które nie mogą być przechowywane ze sobą w magazynie. Na wejściu można przekazać mu następujące parametry:

- *type* - Czy rozwiązanie da się znaleźć czy nie. Innymi słowy, czy graf ma być dwudzielny. Generowanie nierozwiązywalnych problemów nie będzie ciekawe, ale przyda się do oceny poprawności. (*solvable*, *unsolvable*)
- n - Liczba substancji. (jeśli *solvable*: $n \geq 1$, w przeciwnym wypadku $n \geq 3$)
- f (*opcjonalne*) - W przypadku wybrania *solvable*, określa ile substancji ma być w jednym z magazynów. Jeśli brak, losowane. W przypadku *unsolvable* ignorowane. ($0 \leq f \leq n$)
- d (*opcjonalne*) - Określa, jaka część możliwych krawędzi w grafie ma być wygenerowana. Na przykład dla $d = 1.0$ wygenerowane zostaną wszystkie możliwe krawędzie (czyli $f(n-f)$ lub $\frac{n(n-1)}{2}$ krawędzi, w zależności od ustawionego *type*). Jeśli brak, losowane. ($0.0 \leq d \leq 1.0$)

Jeśli *unsolvable*:

Utwórz listę wierzchołków od 1 do n. Żeby zapewnić, że graf nie będzie dwudzielny, wybierz losowo trzy wierzchołki i utwórz krawędzie między nimi. Utwórz listę krawędzi jako wszystkich możliwych kombinacji bez powtórzeń par wierzchołków wykluczając wyżej utworzone krawędzie. Będzie ich $e = \frac{n(n-1)}{2} - 3$. Z nich losowo wybierz d krawędzi (d to ułamek, więc ściślej mówiąc wybierz $[de]$ krawędzi).

Jeśli *solvable*:

Utwórz listę wierzchołków od 1 do n. Losowo podziel je na dwie listy, gdzie w pierwszej będzie f wierzchołków, a w drugiej $n - f$ wierzchołków. Utwórz listę krawędzi jako listę wszystkich możliwych krawędzi pomiędzy jedną a drugą listą (iloczyn kartezjański). Będzie ich $e = f(n-f)$. Z nich losowo wybierz d krawędzi (d to ułamek, więc ściślej mówiąc wybierz $[de]$ krawędzi).

W obu przypadkach zwróć wybrane krawędzie.

7 Pomiary czasu wykonania i porównanie ze złożonością

Aby sprawdzić, czy teoretyczna złożoność zgadza się z rzeczywistością, zmierzmy czas wykonania algorytmu dla rosnących rozmiarów problemów.

W celu lepszej analizy sprowadźmy problem dwuwymiarowy do kilku szczególnych problemów jednowymiarowych. W naszym przypadku ustalmy liczbę restrykcji dla różnych wartości parametru d (zagęszczenie restrykcji):

- 0.0 - brak restrykcji
- 0.5 - połowa wszystkich możliwych spełnialnych restrykcji
- 1.0 - wszystkie możliwe spełnialne restrykcje

Wiedząc, że maksymalnych restrykcji może być $m = \frac{n^2}{4}$ złożoność zależna od ilości substancji i zagęszczenia wyniesie $O(n + d\frac{n^2}{4})$. Zatem w szczególności złożoność jest $O(n)$ dla $d = 0.0$ i $O(n^2)$ dla $d = 1.0$.

W każdej iteracji wygenerujmy pewną ilość problemów o danym rozmiarze, rozwiążmy je pewną ilość razy i zapiszmy średni czas wykonania algorytmu. W kolejnych iteracjach zwiększajmy rozmiar problemu.

Po wykonaniu wszystkich pomiarów możemy porównać wyniki z oszacowaniem teoretycznym. Dla każdego punktu pomiarowego obliczmy dopasowanie zgodnie ze wzorem $q = \frac{t(n)}{cT(n)}$, gdzie $c = \frac{t(n_{mediana})}{T(n_{mediana})}$, $t(n)$ jest zmierzonym czasem wykonania, a $T(n)$ teoretyczną złożonością.

Zakładamy, że rzeczywisty czas wykonania będzie złożonością pomnożoną przez pewien czynnik. c próbuje zniwelować ten czynnik.

Jeśli q będzie w okolicy 1 to znaczy, że teoretyczna złożoność pokrywa się z rzeczywistością. Jeśli q będzie rosnące to znaczy, że wystąpiło niedoszacowanie, a jeśli q będzie malejące przeszacowanie.

Poniżej zaprezentowane są wyniki pomiarów i porównań dla 20 iteracji, 10 instancji problemu w każdej iteracji i 10 powtórzeń rozwiązania każdej instancji.

Tablica 1: Porównanie dla $d = 0.0$

n	**t(n)[ms]**	**q(n)**
100	0.16838	1.20479
200	0.41265	1.47629
300	0.44059	1.05082
400	0.59449	1.06341
500	0.70756	1.01253
600	0.83351	0.99398
700	1.01610	1.03861
800	1.12776	1.00866
900	1.24362	0.98870
1000	1.39227	0.99618
1100	1.54269	1.00347
1200	1.67930	1.00130
1300	1.82335	1.00356
1400	1.92018	0.98136
1500	2.04197	0.97404
1600	2.18393	0.97664
1700	2.39190	1.00673
1800	2.49459	0.99161
1900	2.70256	1.01774
2000	2.78311	0.99567

Tablica 2: Porównanie dla $d = 0.5$

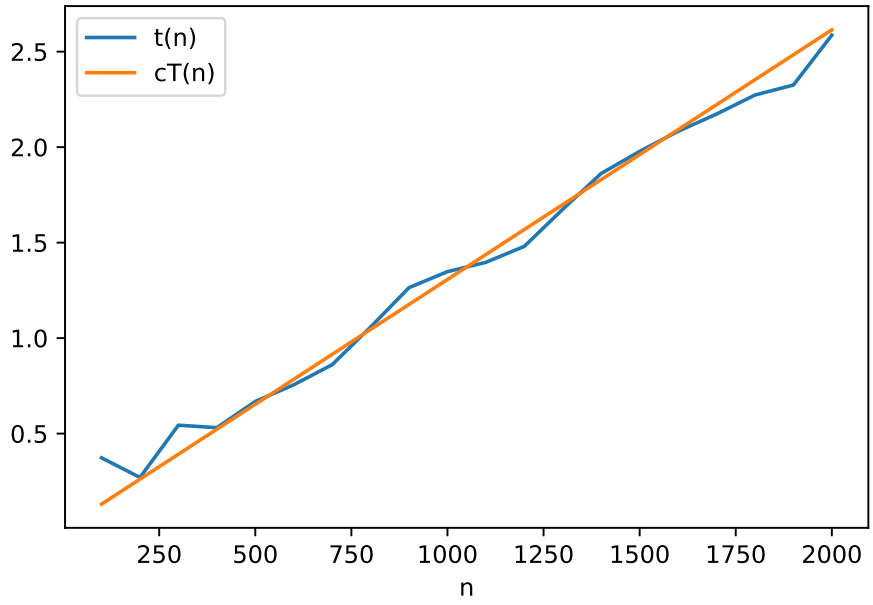
n	**t(n)[ms]**	**q(n)**
100	1.04453	1.09233
200	2.73720	0.74314
300	6.89849	0.84321
400	13.97587	0.96719
500	19.63995	0.87330
600	28.95386	0.89641
700	39.97777	0.91105
800	55.76846	0.97441
900	69.89178	0.96595

n	**t(n)[ms]**	**q(n)**
1000	83.80573	0.93901
1100	112.91415	1.04634
1200	121.64114	0.94774
1300	160.36617	1.06516
1400	189.08795	1.08340
1500	211.61423	1.05659
1600	241.21625	1.05890
1700	271.00275	1.05412
1800	305.12615	1.05892
1900	335.50506	1.04526
2000	416.70929	1.17191

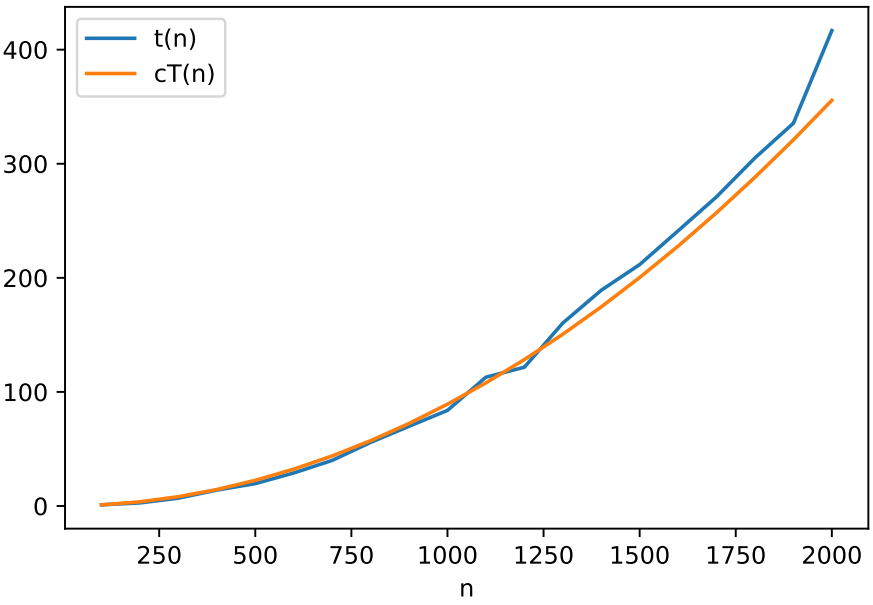
Tablica 3: Porównanie dla $d = 1.0$

n	**t(n)[ms]**	**q(n)**
100	1.03666	0.55402
200	5.32157	0.72494
300	12.26002	0.74717
400	22.49534	0.77371
500	36.72622	0.81003
600	57.35082	0.87958
700	90.39763	1.01956
800	116.10453	1.00329
900	146.56760	1.00127
1000	179.78674	0.99529
1100	218.44444	0.99978
1200	261.45906	1.00582
1300	319.91725	1.04892
1400	364.04820	1.02941
1500	415.30729	1.02319
1600	473.77625	1.02606
1700	537.35481	1.03102
1800	602.07213	1.03054
1900	679.87886	1.04457
2000	762.98143	1.05806

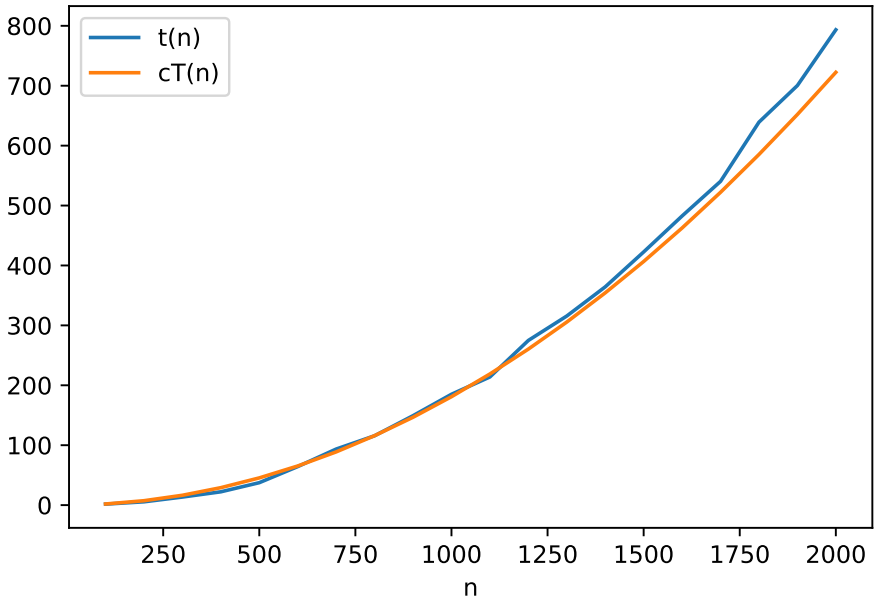
Rysunek 1: Porównanie w formie graficznej dla $d = 0.0$



Rysunek 2: Porównanie w formie graficznej dla $d = 0.5$



Rysunek 3: Porównanie w formie graficznej dla $d = 1.0$



Z otrzymanych wyników można wywnioskować, że oszacowana złożoność jednowymiarowa $O(n + d\frac{n^2}{4})$ oraz dwuwymiarowa $O(n + m)$ są zgodne z rzeczywistością.