Algorytmy z nawrotami

Algorytmy z nawrotami często bazują na systematycznym przejrzeniu wszystkich podzbiorów pewnego zbioru. Można to robić na przykład posługując się następującym schematem wypisywania wszystkich podzbiorów zbioru $\{0, 1, 2, \ldots, n-1\}$:

```
1: S ← ∅
2: GenerateSubsets(0)
3: procedure GenerateSubsets(k ∈ ℕ)
4:     Wypisz zbiór S
5:     for m = k, k + 1, ..., n − 1 do
6:         Dodaj m do zbioru S
7:         GenerateSubsets(m + 1)
8:         Usuń m ze zbioru S
```

W momencie wywołania procedury GenerateSubsets(k) (dla $k > 0$) największym elementem zawartym w zbiorze S jest $k - 1$. Procedura wypisuje wszystkie nadzbiory S (włącznie z S) powstałe przez dodanie do S pewnej liczby elementów ze zbioru $\{k, k+1, \ldots, n-1\}$; m należy interpretować jako najmniejszy z tych elementów. Łatwo zauważyć, że przy takim przeglądzie każdy zbiór zostanie wypisany dokładnie raz.

Zauważmy, że w powyższym kodzie istotny jest zakres pętli for. Jeżeli byłby on szerszy, mogłoby to prowadzić, zależnie od konkretnej implementacji, do przeglądania tego samego zbioru wielokrotnie lub do nieskończonej pętli, podczas gdy zawężenie go skutkowałoby pominięciem niektórych zbiorów.

Innym typowym zadaniem jest przejrzenie wszystkich permutacji zadanego zbioru – poniżej pseudokod generujący wszystkie permutacje zbioru $\{0, 1, 2, \ldots, n - 1\}$.

```
1: Oznacz wszystie liczby 0, 1, ..., n − 1 jako nieużyte
2: GeneratePermutations(0)
3: procedure GeneratePermutations(k ∈ ℕ)
4:     if k == n then
5:         Wypisz permutację
6:         return
7:     for m = 0, 1, ..., n − 1 do
8:         if m nieużyte then
9:             oznacz m jako użyte
10:            permutacja[k] = m
11:            GeneratePermutations(k + 1)
12:            oznacz m jako nieużyte
```

Wywołanie GeneratePermutations(k) wypisuje wszystkie permutacje, które zaczynają się od k pierwszych symboli (o indeksach $0, 1, \ldots, k - 1$) w tablicy permutacja; łatwo zauważyć, że podany kod wypisze każdą permutację dokladnie raz.

Istotnym elementem powyższego kodu jest oznaczanie wybranego elementu m jako użytego przed wywołaniem rekurencyjnym oraz oznaczenie go jako nieużytego po powrocie z rekurencji. W ten sposób dbamy o to, aby zapisana informacja o użytych elementach była zgodna z tym, co znajduje się w pierwszych k elementach tablicy permutacja; nieprawidłowa realizacja tego elementu jest częstym błędem.

Optymalizacje

Często już na wczesnych poziomach rekursji potrafimy stwierdzić, że aktualne (częściowe) rozwiązanie jest niepoprawne i nie ma potrzeby wykonywać dalszych wywołań rekurencyjnych. Przekłada się to na znacznie przyspieszenie działania algorytmu. Jako przykład rozważmy następujący problem.

Problem kliki
Dane: Graf G
Szukane: Najliczniejsza klika w G

O klice w grafie możemy myśleć jako o podzbiorze zbioru wierzchołków, zatem naturalnym pomysłem jest bazowanie na algorytmie przeglądającym wszystkie takie podzbiory. Zauważmy jednak, że możemy poczynić przynajmniej dwa usprawnienia:

- Jeśli rozpatrywany wierzchołek v nie sąsiaduje ze wszystkimi wierzchołkami w aktualnym zbiorze S, możemy pominąć dalsze wywołania rekurencyjne odpowiadające zbiorom zawierającym v,
- Jeśli rozmiar zbioru S powiększony o liczbę wierzchołków nieodrzuconych przez powyższe kryterium jest nie większy niż rozmiar najwiekszej znalezionej do tej pory kliki, możemy natychmiast wyjść z danego poziomu rekurencji (bo nie mamy szansy na lepsze rozwiązanie).

Po implementacji tych usprawnień algorytm będzie wyglądał następująco:

```
1: S ← ∅
2: bestS ← ∅
3: MaxCliqueRec(0)
4: procedure MaxCliqueRec(k ∈ ℕ)
5:     C ← zbiór wierzchołków z zakresu {k, k+1, ..., n−1} sąsiadujących z każdym wierzchołkiem z S
6:     if |C| + |S| ≤ |bestS| then
7:         return
8:     else
9:         if |S| > |bestS| then
10:            bestS = S
11:    for m ∈ C do
12:        Dodaj m do zbioru S
13:        MaxCliqueRec(m + 1)
14:        Usuń m ze zbioru S
```

Zadanie: Najwieksza klika, izomorfizm grafów

Zadanie polega na uzupełnieniu dwóch metod

```
int MaxClique(this Graph g, out int[] clique)
bool IsomorpchismTest(this Graph<int> g, Graph<int> h, out int[] map)
```

zdefiniowanych w załączonym pliku.

Uwagi

- Izomorfizm powinien uwzględniać wagi krawędzi.

Punktacja

- Największa klika
  - 1 pkt – wszystkie testy poprawne
  - 0.5 pkt – zwykłe testy poprawne, w testach wydajności dozwolony wynik „Timeout” („Fail” niedozwolony)
- Izomorfizm grafów
  - 1.5 pkt – wszystkie testy poprawne
  - 0.5 pkt – zwykłe testy poprawne, w testach wydajności dozwolony „Timeout” („Fail” niedozwolony)

Zadanie: Kampania wyborcza

Zbliżają się wybory prezydenckie. Pan Marchewa, jeden z kandydatów ze znanej partii Altruistycznego Stowarzyszenia Demokratów, celem zwiększenia poparcia wyborców zdecydował się jeździć po kraju i zorganizować spotkania z wyborcami w poszczególnych miastach. Panu Marchewie udało się zdobyć odrobinę partyjnych pieniędzy, ale nie na tyle dużo, żeby zorganizować spotkania z wyborcami ze wszystkich miast w państwie. Pan Marchewa chce spotkać się z jak największą liczbą mieszkańców zostawiając w kieszeni jak najwięcej pieniędzy. Musi on zdecydować które miasta odwiedzi, rozpoczynając od stolicy, w której aktualnie się znajduje.

Założenia:

1. Każda podróż między dwoma miastami kosztuje.
2. Całkowity koszt podróży nie może przekroczyć budżetu.
3. Każde miasto można odwiedzić maksymalnie raz (może pojawić sie na liście wynikowej maksymalnie raz).
4. Każde miasto może mieć różną liczbę mieszkańców.
5. Po odwiedzeniu ostatniego miasta na swojej ścieżce pan Marchewa wraca do stolicy (to połączenie musi istnieć oraz musi wystarczyć na nie budżetu).
6. Jeżeli istnieje kilka ścieżek skutkujących identyczną liczbą odwiedzonych mieszkańców, należy zwrócić najtańszą z nich. Jeżeli jest kilka ścieżek o minimalnym koszcie można zwrócić dowolną.

Część 1 (1 punkt)

Zadanie: znaleźć ciąg miast, dla których koszt podróży zmieści się w budżecie, a suma mieszkańców odwiedzonych miast będzie maksymalna. Jeśli dla kilku tras suma mieszkańców miast jest jest taka sama wybieramy tą o niższym koszcie. Pierwszym miastem musi być stolica (do stolicy musimy też wrócić, ale nie umieszczamy jej po raz drugi w odpowiedzi).

Przykład:
Klika K_3 (wierzchołki 0, 1, 2)
Koszty podróży między miastami: stałe, równe 1
Liczby mieszkańców: 100, 200, 300
Dostępny budżet: 5
Stolica to miasto o numerze 0
Wynik: (0, true), (1, true), (2, true)

Czyli pan Marchewa zacznie w stolicy (0) i zorganizuje tam spotkanie, po czym pojedzie do miasta 1 za 1 pieniędzy, zorganizuje tam spotkanie, a później pojedzie do miasta 2 za 1 pieniędzy i tam również zorganizuje spotkanie. Na końcu wróci do stolicy za 1 pieniędzy.

W sumie spotka się z 600 mieszkańcami. Cena tej ścieżki wynosi 3.

Punktacja:

- Testy poprawnościowe: 0.5 punktu,
- Testy wydajnościowe: 0.5 punktu.

Część 2 (1 punkt)

W związku z nową ustawą o Surowym Okresie Przedwyborczym, miasta wymagają od kandydatów opłaty za zorganizowanie spotkania z wyborcami. Każde miasto ustala swoją opłatę.

Pan Marchewa modyfikuje zatem swoją strategię i może wybrać w których odwiedzanych przez siebie miastach zorganizuje spotkania wyborcze, a w których nie. W stolicy, z której startuje, może również wybrać czy organizuje spotkanie, czy nie.

Zadanie: znaleźć ciąg miast, które powinien odwiedzić pan Marchewa oraz zdecydować w których z nich powinien on zorganizować spotkania tak, aby spotkać się z jak największą liczbą mieszkańców, a jednocześnie zmieścić w budżecie. Jeśli dla kilku tras suma mieszkańców miast w których zorganizowano spotkania jest jest taka sama wybieramy tą o niższym koszcie (mieszkańców miast, przez które przejeżdzamy, ale nie zorganizowano w nich spotkania nie uwzględniamy w sumowaniu spotkanych mieszkańców). Pierwszym miastem musi być stolica (do stolicy musimy też wrócić - nie umieszczamy jej po raz drugi w odpowiedzi), ale niekoniecznie musi w niej być spotkanie.

Przykład:
Klika K3 (wierzchołki 0, 1, 2)
Koszty podróży między miastami: stałe, równe 1
Liczby mieszkańców: 100, 200, 300
Koszty organizacji spotkań: 1, 1, 1

Dostępny budżet: 5
Stolica to miasto o numerze 0
Wynik: (0, false), (1, true), (2, true)

Czyli pan Marchewa zacznie w stolicy (0), ale nie zorganizuje tam spotkania, po czym pojedzie do miasta 1 za 1 pieniędzy, zorganizuje tam spotkanie za 1 pieniędzy, a później pojedzie do miasta 2 za 1 pieniędzy i tam również zorganizuje spotkanie za 1 pieniędzy. Na końcu wróci do stolicy za 1 pieniędzy.

W sumie spotka się z 500 mieszkańcami. Cena tej ścieżki wynosi 5 (koszt transportu między miastami jest równy w sumie 3, a koszt organizacji spotkań w miastach 1 i 2 wynosi w sumie 2).

Punktacja:

- Testy poprawnościowe: 0.5 punktu
- Testy wydajnościowe: 0.5 punktu

Wejście

- cities – mapa państwa przekazana w postaci nieskierowanego grafu, którego wierzchołkami są miasta, a krawędziami drogi między miastami. Waga krawędzi równa jest kosztowi podróży między miastami, które łączy. Wagi krawędzi zawsze są nieujemne.
- citiesPopulation – tablica zawierająca liczby mieszkańców w każdym z miast. citiesPopulation[i] jest liczbą mieszkańców w mieście i. Wartości zawsze są nieujemne.
- meetingCosts – tablica zawierająca koszt zorganizowania spotkania z mieszkańcami w każdym z miast. meetingCosts[i] jest kosztem zorganizowania spotkania w mieście i. Wartości zawsze są nieujemne. Dla części 1 koszt zorganizowania spotkań jest równy 0.
- budget – dostępny budżet, zawsze wartość nieujemna.
- capitalCity – numer wierzchołka odpowiadającego stolicy – miastu startowemu

Nie można modyfikować danych wejściowych.

Wyjście

- Maksymalna liczba mieszkańców, z którymi uda się spotkać panu Marchewie
- out (int, bool)[] path – tablica opisująca kolejne miasta odwiedzane przez pana Marchewę. Pierwszy element krotki to numer miasta, a drugi decyduje czy w danym mieście ma być zorganizowane spotkanie.

  Pierwszym miastem zawsze musi być stolica (w której można, ale nie trzeba organizować spotkania).

  Jeżeli pan Marchewa nie będzie wyjeżdżał ze stolicy, to ta lista powinna zawierać jedynie wpis dla stolicy. Nie wystąpią wtedy żadne koszty związane z podróżą.

  W części 1 drugi element krotki musi być zawsze równy true.