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)