Jakub Radoszewski — Treść zadania
Michał Pilipczuk, Jakub Radoszewski — Opracowanie
Michał Pilipczuk — Program

OI, Etap I, 22.10–19.11.2007

Plakatowanie

Wszystkie budynki we wschodniej części Bajtogrodu zostały zbudowane zgodnie z zasadami starego bajtobudownictwa: stoją one jeden przy drugim (nie ma między nimi przerw). Razem tworzą bardzo długą ścianę budynków o zróżnicowanej wysokości, ciągnącą się ze wschodu na zachód.

Burmistrz Bajtogrodu, Bajtazar, postanowił, że ścianę budynków należy od północnej strony pokryć plakatami. Bajtazar zastanawia się, jaką minimalną liczbą plakatów można pokryć całą północną ścianę budynków. Plakaty powinny mieć kształt prostokątów o bokach pionowych i poziomych. Plakaty nie mogą zachodzić na siebie, natomiast mogą stykać się brzegami. Każdy plakat musi w całości przylegać do ścian pewnych budynków i cała powierzchnia północnych ścian budynków musi być pokryta plakatami.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opisy budynków,*
- *wyznaczy minimalną liczbę plakatów potrzebnych do całkowitego pokrycia ich północnych ścian,*
- *wypisze wynik na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n ($1 \leqslant n \leqslant 250\,000$), oznaczającą liczbę budynków stojących w rzędzie. Kolejne n wierszy zawiera po dwie liczby całkowite d_i i w_i ($1 \leqslant d_i, w_i \leqslant 1\,000\,000\,000$), oddzielone pojedynczym odstępem i oznaczające długość i wysokość i-tego budynku w rzędzie.

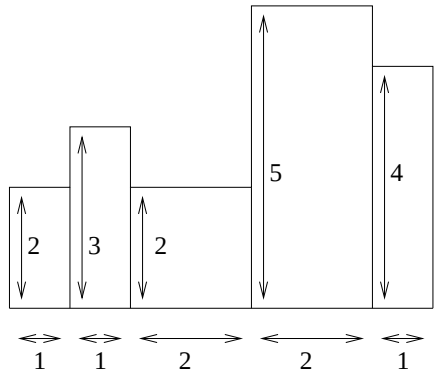
Wyjście

Pierwszy i jedyny wiersz wyjścia powinien zawierać jedną liczbę całkowitą — minimalną liczbę prostokątnych plakatów, którymi można całkowicie pokryć północne ściany budynków.

Przykład

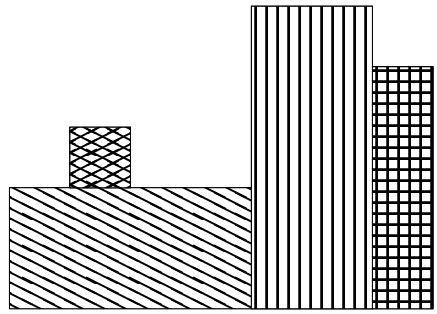
Dla danych wejściowych:

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



poprawnym wynikiem jest:

```
4
```



Na rysunkach została przedstawiona sama północna ściana rzędu budynków. Drugi z rysunków przedstawia przykładowe pokrycie ściany czterema plakatami.

Rozwiązanie

Dolne ograniczenie i uproszczenia

Na początek spróbujmy określić, ile co najmniej plakatów jest potrzebnych do pokrycia całej północnej ściany budynków, czyli jakie jest *dolne ograniczenie* na ich liczbę. Potem pokażemy algorytm, który wyznacza plakatowanie wykorzystujące dokładnie tyle plakatów.

Zauważmy, że północna ściana każdego budynku musi zostać w całości pokryta, więc każdemu budynkowi możemy jednoznacznie przyporządkować plakat, który pokrywa jego *lewy górny róg*. W takim przyporządkowaniu jeden plakat może odpowiadać dwóm różnym budynkom tylko wtedy, gdy są one tej samej wysokości i między nimi nie stoi żaden budynek niższy od nich. Wśród budynków, którym jest przyporządkowany ten sam plakat, możemy wyróżnić jeden — stojący najbardziej na lewo, czyli na zachód. Dokładniej określa to następująca definicja.

Definicja 1. Budynek nazywamy *redundantnym*, jeśli istnieje budynek położony na lewo od niego i takiej samej wysokości jak on, taki że pomiędzy tymi budynkami nie ma żadnego niższego od nich budynku. Jeżeli ten warunek dla danego budynku nie zachodzi, to nazywamy go *nieredundantnym*.

Przykład 1. W teście przykładowym z treści zadania trzeci budynek od lewej jest redundantny, a pozostałe są nieredundantne.

Łatwo zauważyć, że każdym dwóm budynkom nieredundantnym muszą być przyporządkowane różne plakaty, co pozwala nam sformułować następujący wniosek.

Obserwacja 1. *Liczba budynków nieredundantnych* jest *dolnym ograniczeniem na liczbę plakatów* użytych w dowolnym plakatowaniu.

Warto także poczynić kolejne obserwacje, które pozwalają lepiej określić, na czym polega trudność problemu.

Obserwacja 2. Jeśli wszystkie budynki w Bajtogrodzie są różnej wysokości, to wszystkie one są nieredundantne i do ich pokrycia potrzeba n plakatów! Takie pokrycie można łatwo skonstruować, naklejając na każdy budynek osobny plakat. Widać więc, że problem stanowią jedynie budynki tej samej wysokości.

Obserwacja 3. Szerokości budynków nie mają znaczenia dla rozwiązania, więc możemy założyć, że szerokość każdego budynku wynosi 1.

Algorytm

W rozwiązaniu wzorcowym będziemy przeglądać budynki od lewej do prawej. Za każdym razem, gdy napotkamy budynek nieredundantny, dodamy do rozwiązania nowy plakat rozpoczynający się w jego górnym, lewym narożniku. Rozmiar plakatu ustalimy tak, by stworzyć plakatowanie pokrywające wszystkie budynki. W szczególności, górny fragment każdego redundantnego budynku zostanie pokryty za pomocą plakatu, rozpoczynającego się na ostatnim budynku nieredundantnym o tej samej wysokości, położonym na lewo od niego. Założony cel osiągniemy, jeśli będziemy przestrzegać następujących kryteriów. Dla budynku nieredundantnego b i przyporządkowanego mu plakatu p_b:

(i) lewy, górny róg plakatu p_b będzie przyklejony dokładnie w lewym, górnym rogu budynku b;

(ii) dolny brzeg plakatu p_b będzie umieszczony najniżej, jak tylko się da — tak aby plakat nie nachodził na żaden z wcześniej umieszczonych plakatów (a zatem dolna krawędź świeżo przyklejonego plakatu będzie od dołu dotykać górnej krawędzi pewnego innego plakatu $p_{b'}$, dla budynku b' stojącego na lewo od b, albo ziemi);

(iii) prawy brzeg plakatu p_b będzie umieszczony najdalej, jak tylko się da — tak, aby plakat nie wystawał poza ścianę żadnego budynku; to oznacza, że p_b zostanie zakończony tuż przed rozpoczęciem pierwszego budynku b'' niższego od b i położonego na prawo od b.

Skonstruujemy teraz algorytm wyznaczający plakatowanie zgodne z kryteriami (i) – (iii). Plakaty, które będziemy przyklejać, będą „prawostronnie otwarte", czyli w momencie dodawania ich do rozwiązania nie będziemy zastanawiać się nad tym, gdzie dany plakat powinien mieć swój prawy brzeg. Określimy to w jednym z dalszych kroków algorytmu,

gdy napotkamy budynek, na którym plakat ten powinien zakończyć się zgodnie z kryterium (iii). W każdym momencie, wszystkie rozpoczęte i jeszcze nie zakończone plakaty będziemy trzymać na *stosie S*, uporządkowane od najwyżej do najniżej położonych. Zadbamy przy tym, by *każdy plakat znajdujący się na stosie stykał się dolnym brzegiem z górnym brzegiem pewnego innego plakatu ze stosu albo z ziemią*. Warunek ten będzie *niezmiennikiem* naszej procedury. Jego przestrzeganie zagwarantuje, że plakaty znajdujące się na stosie będą całkowicie pokrywały aktualnie rozważaną pozycję w rzędzie budynków, a na zakończenie algorytmu na ścianie nie pozostaną żadne nieoplakatowane „dziury".

Oznaczmy kolejne budynki opisane w danych zadania $b_1, b_2, \ldots, b_n$. Rozważmy teraz krok algorytmu, w którym przeanalizowaliśmy już $i-1$ budynków i aktualnie rozważamy budynek b_i o wysokości w_i. Zgodnie z niezmiennikiem algorytmu, stos S zawiera w tym momencie ciąg plakatów $p_1, \ldots, p_j$ takich, że p_1 styka się z ziemią, p_2 styka się dolnym brzegiem z górnym brzegiem p_1, itd. aż do p_j, który od dołu styka się z p_{j-1}. Aby zachować kryterium (iii), musimy w tym momencie zakończyć wszystkie otwarte plakaty sięgające wysokości większej niż w_i — usuwamy je więc ze stosu. Po tej operacji na stosie pozostają plakaty $p_1, \ldots, p_k$, dla pewnego $k \leqslant j$ (jeśli usuniemy ze stosu wszystkie plakaty, czyli $S = \emptyset$, to dla uproszczenia możemy przyjąć, że p_k oznacza poziom zerowy, czyli ziemię). Spełniają one oczywiście niezmiennik algorytmu, skoro spełniały go plakaty $p_1, \ldots, p_j$.

Niech $w \leqslant w_i$ oznacza wysokość, do jakiej sięga najwyższy pozostały na stosie plakat p_k. Zauważmy, że jeżeli budynek b_i jest redundantny, to musi zachodzić $w = w_i$. Faktycznie, niech b_l będzie najbliższym na lewo od b_i budynkiem nieredundantnym o wysokości $w_l = w_i$. Wówczas między b_l a b_i z definicji nie ma żadnego budynku o wysokości mniejszej niż w_i, czyli plakat rozpoczęty w lewym górnym rogu b_l nie został jeszcze prawostronnie zakończony. Ponieważ górny brzeg tego plakatu jest położony na wysokości w_i, to musi to być plakat p_k o wysokości $w = w_i$. To oznacza, że budynek b_i możemy pokryć, przedłużając w prawo o d_i wszystkie plakaty z S i nie dodając do rozwiązania żadnego nowego plakatu.

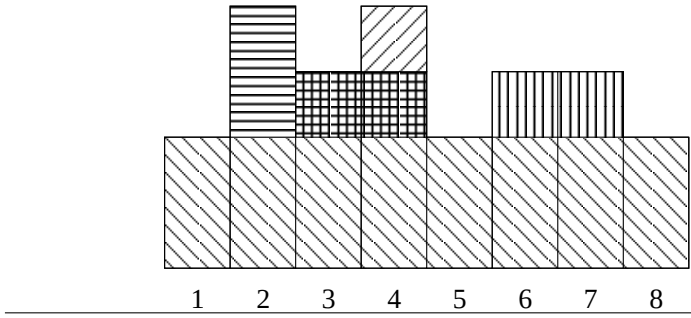
W przypadku, gdy budynek b_i jest nieredundantny, zachodzi natomiast nierówność $w < w_i$. Łatwo to wykazać, przeprowadzając rozumowanie analogiczne jak poprzednio. Z budynkiem b_i związemy więc plakat p, rozpoczynający się w jego lewym górnym rogu (zgodnie z kryterium (i)) i stykający się dolnym brzegiem z plakatem p_k (zgodnie z kryterium (ii)). Plakat p pozostawimy w tym momencie prawostronnie otwarty, tzn. na razie nie będzie nas interesować, kiedy i gdzie on się zakończy. Umieścimy go natomiast na szczycie stosu S, który od tej chwili zawiera kolejno plakaty: $p_1, \ldots, p_k, p$, a zatem wciąż spełnia niezmiennik algorytmu. Co więcej, przedłużenie wszystkich plakatów z S w prawo o szerokość budynku b_i spowoduje całkowite pokrycie ściany tego budynku.

W ten sposób opisaliśmy i-ty krok algorytmu, w którym uzupełniamy plakatowanie zgodnie z kryteriami (i) – (iii), a po jego wykonaniu budynek b_i jest całkowicie oplakatowany i ponadto zachowany jest niezmiennik algorytmu. Prosty dowód indukcyjny pozwala wykazać, że wykonanie kolejnych kroków algorytmu dla $i = 1, 2, \ldots, n$ doprowadza do konstrukcji poprawnego plakatowania o liczności równej liczbie nieredundantnych budynków w ciągu.

Poniżej przedstawiamy pseudokod opisanego algorytmu. Zmienna S oznacza w nim stos plakatów, które reprezentujemy za pomocą wysokości ich górnego brzegu. Na stosie możemy wykonywać podstawowe operacje: wstawienie elementu na szczyt (operacja *push*), odczytanie elementu ze szczytu (operacja *top*), usunięcie elementu ze szczytu (operacja *pop*) oraz sprawdzenie, czy stos jest pusty.

```
1:  S := ∅;
2:  p := 0; { liczba użytych plakatów }
3:  for i := 1 to n do
4:  begin
5:    while S ≠ ∅ and S.top() > w_i do
6:      S.pop();
7:    if S = ∅ or S.top() < w_i then
8:    begin
9:      { analizowany budynek jest nieredundantny }
10:     S.push(w_i);
11:     p := p + 1;
12:   end;
13: end;
14: return p;
```

Przykład 2. Na rys. 1 przedstawione jest znalezione przez opisany algorytm pokrycie rzędu budynków o wysokościach: 2, 4, 3, 4, 2, 3, 3, 2 za pomocą pięciu plakatów.



Rys. 1: Budynki nieredundantne na rysunku to budynki o numerach: 1, 2, 3, 4 i 6. W ich lewych górnych rogach przyklejone są lewe górne rogi pięciu plakatów. Pozostałe budynki (o numerach: 5, 7 i 8) są redundantne, a ich górne fragmenty są pokryte plakatami przyporządkowanymi budynkom nieredundantnym, odpowiednio 1, 6, 1.

Złożoność czasowa algorytmu

W jednym kroku pętli **for** (wiersze 3–13) łączna liczba wykonanych operacji może być nawet rzędu $O(n)$, ze względu na liczbę iteracji pętli **while** z wierszy 5–6. Daje to oszacowanie złożoności czasowej algorytmu równe $O(n^2)$. Przekonamy się jednak, że jest to szacowanie bardzo zawyżone, zliczając nieco dokładniej liczbę operacji wykonywanych w algorytmie (metodą zwaną *analizą kosztu zamortyzowanego*[1]).

Otóż okazuje się, że sumaryczna liczba obrotów pętli **while**, we wszystkich iteracjach pętli **for**, jest nie większa niż n. Faktycznie, w instrukcji **while** wykonujemy operację usunięcia plakatu ze stosu S. Operację tę możemy wykonać tylko n razy w całym algorytmie,

[1] Więcej informacji na temat analizy kosztu zamortyzowanego można znaleźć w [20]

bo najwyżej tyle plakatów zużyjemy, a każdy z nich tylko raz trafi na stos (operacja w wierszu 10) i tylko raz może zostać z niego zdjęty.

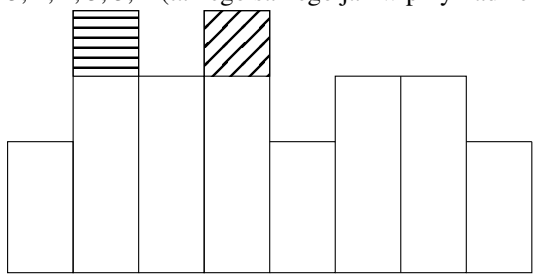
Wszystkie wymagane operacje na stosie potrafimy wykonywać w czasie stałym. Dodatkowo łączny koszt czasowy wszystkich operacji algorytmu wykonywanych poza pętlą **while** jest nie większy niż $O(n)$. Wynika stąd, że złożoność czasowa całego algorytmu wynosi $O(n)$. Opisane rozwiązanie zostało zaimplementowane w plikach `pla.c`, `pla1.pas` oraz `pla6.java`.

Inne rozwiązania

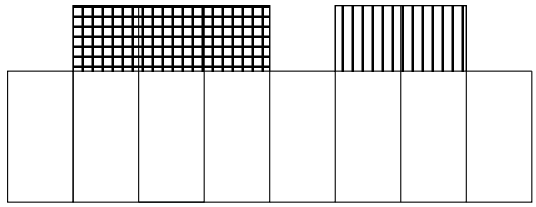
Metoda pokazana w algorytmie wzorcowym nie jest jedyna — istnieje wiele innych sposobów plakatowania, pozwalających uzyskać maksymalną liczbę punktów za zadanie. Naszkicujemy je poniżej, pomijając szczegółowe dowody poprawności. We wspomnianych rozwiązaniach kolejność przeglądania budynków jest inna niż w rozwiązaniu wzorcowym, a mianowicie: od najwyższych bądź od najniższych.

Plakatowanie „od góry". Na początku zajmujemy się budynkami najwyższymi — załóżmy, że każdy z nich ma wysokość h. Jeśli takie budynki stoją obok siebie, to „sklejamy" je w jeden. Na górną część każdego z najwyższych budynków nalepiamy nowy plakat. Plakat kończy się od dołu na wysokości wyższego z dwóch sąsiadów rozważanego budynku. Następnie „obcinamy" zalepione plakatami fragmenty najwyższych budynków i pozostaje nam do oplakatowania ciąg niższych budynków, z którym postępujemy analogicznie.

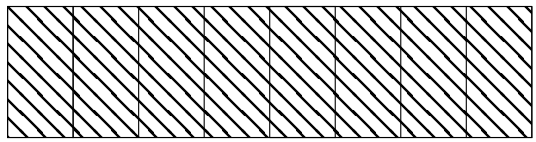
Przykład 3. Rozważmy kolejne kroki plakatowania „od góry" dla ciągu budynków o wysokościach 2, 4, 3, 4, 2, 3, 3, 2 (takiego samego jak w przykładzie 2).



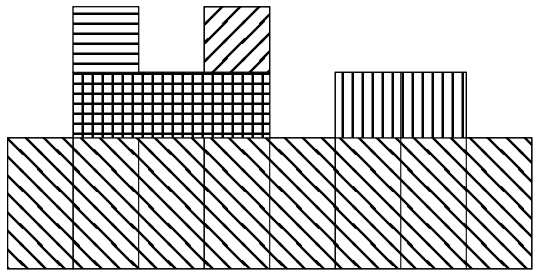
Rys. 2: Najwyższe budynki mają wysokość $h = 4$ i nie sąsiadują ze sobą, więc nie trzeba ich sklejać. Na ich górne części naklejamy dwa plakaty, a następnie obniżamy każdy z budynków do wysokości wyższego sąsiada, czyli $\max(2,3) = 3$.



Rys. 3: Po redukcji mamy rząd budynków, w którym najwyższe mają wysokość $h' = 3$. Tworzą one dwie grupy sąsiadujących budynków jednakowej wysokości — każdą z nich traktujemy jak jeden budynek. Na górną część każdej grupy naklejamy jeden plakat, redukując tym samym problem do wysokości 2.



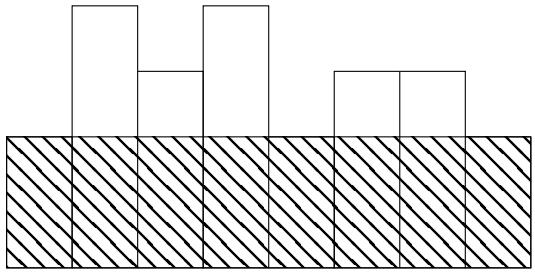
Rys. 4: Na koniec pozostaje nam rząd budynków o wysokości $h'' = 2$. Pokrywamy go jednym plakatem, co kończy działanie algorytmu.



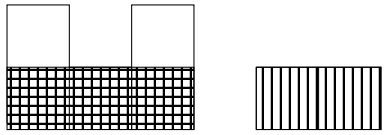
Rys. 5: Ostateczna postać rozwiązania skonstruowanego przez opisany algorytm, czyli plakatowania o liczności 5. Zauważmy, że uzyskane plakatowanie jest inne niż znalezione przez rozwiązanie wzorcowe, ale wykorzystuje tyle samo plakatów.

Plakatowanie „od ziemi". Optymalne rozwiązanie można także uzyskać, przeglądając budynki od najniższych do najwyższych. Rozpoczynamy od naklejenia tuż nad ziemią plakatu o wysokości równej wysokości najniższego budynku i szerokości równej sumarycznej szerokości wszystkich budynków. Następnie „ucinamy" oplakatowane części wszystkich budynków. W ten sposób dostajemy jedną bądź klika grup budynków o mniejszej wysokości niż wyjściowa. Każdy z uzyskanych podproblemów rozwiązujemy w analogiczny sposób, ponownie zaklejając jednym plakatem o szerokości równej szerokości całej grupy dolne części budynków.

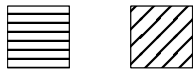
Przykład 4. Rozważmy kolejne kroki plakatowania „od ziemi" dla ciągu budynków o wysokościach 2, 4, 3, 4, 2, 3, 3, 2 (takiego samego jak w obydwu dotychczasowych przykładach).



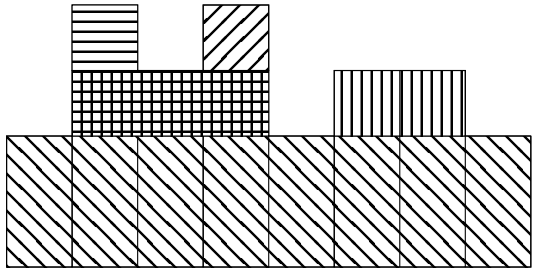
Rys. 6: Najniższy budynek ma wysokość 2. Przyklejamy zatem na dole na całej szerokości ściany plakat o wysokości 2. Potem usuwamy całą pokrytą powierzchnię, co powoduje powstanie dwóch grup budynków, które odtąd będziemy rozważać osobno.



Rys. 7: W każdej z grup najniższy budynek ma wysokość 1. Naklejamy zatem na całej szerokości każdej z grup plakat o tej wysokości. W ten sposób druga grupa jest całkowicie pokryta, natomiast z pierwszej powstają dwa samotnie stojące budynki o wysokości 1.



Rys. 8: Pokrycie dwóch samotnie stojących budynków wymaga oczywiście zużycia dwóch plakatów. Kończy to działanie algorytmu.



Rys. 9: Ostateczna postać rozwiązania skonstruowanego przez opisany algorytm to plakatowanie o liczności 5. Zauważmy, że jest ono takie samo, jak w przypadku poprzedniej metody.

W każdej z przedstawionych metod potrzebna jest odpowiednia struktura danych pozwalająca łatwo wyznaczać kolejne plakaty (czyli najwyższe bądź najwyższe budynki) w kolejnych podproblemach:

- w plakatowaniu „od ziemi” możemy wykorzystać statyczne drzewo przedziałowe; otrzymujemy w ten sposób algorytm o złożoności czasowej $O(n \log n)$, zastosowany w programach `pla2.cpp` i `pla7.java`;
- w plakatowaniu „od ziemi” możemy także wykorzystać statyczne drzewo licznikowe; otrzymujemy w ten sposób algorytm o złożoności czasowej $O(n \log n)$, zastosowany w programach `pla3.cpp` i `pla8.java`;
- w plakatowaniu „od góry” możemy wykorzystać listy wskaźnikowe; pozwala to znaleźć rozwiązanie w czasie $O(n)$ plus czas sortowania n budynków; algorytm jest zaimplementowany w plikach `pla4.cpp` i `pla9.java`;
- w plakatowaniu „od góry” możemy wykorzystać także strukturę danych Find-Union (patrz na przykład [15] lub [20]); pozwala to osiągnąć złożoność czasową $O(n \log^* n)$ plus czas sortowania budynków względem wysokości; implementacja tego rozwiązania znajduje się w plikach `pla5.cpp` i `pla10.java`.

Obie opisane metody mogą być także zaimplementowane prościej, bez odpowiednich struktur danych, co daje rozwiązania o złożoności czasowej $O(n^2)$. Ich implementacje można znaleźć w plikach `plas1.cpp-plas4.cpp`. Pozwalały one uzyskać na zawodach 40% punktów.

Testy

Zadanie było sprawdzane na dziesięciu grupach testów. Testy wydajnościowe były generowane tak, żeby sprawiały jak najwięcej problemów nawet najbardziej pomysłowym rozwiązaniom o złożoności $O(n^2)$.

Nazwa	n	Opis
pla1a.in	60	mały test poprawnościowy
pla1b.in	126	mały test wydajnościowy
pla1c.in	1	przypadek brzegowy — jeden budynek
pla2a.in	1 180	mały test poprawnościowy
pla2b.in	1 152	mały test wydajnościowy
pla2c.in	804	przypadek brzegowy — prawie wszystkie budynki równej wysokości
pla3a.in	1 990	mały test poprawnościowy
pla3b.in	2 445	mały test wydajnościowy
pla3c.in	1 708	przypadek brzegowy — prawie wszystkie budynki różnej wysokości
pla4a.in	3 941	średni test poprawnościowy
pla4b.in	3 951	średni test wydajnościowy

Nazwa	n	Opis
pla5a.in	145 941	średni test poprawnościowy
pla5b.in	163 330	średni test wydajnościowy
pla6a.in	236 836	duży test poprawnościowy
pla6b.in	201 424	duży test wydajnościowy
pla7a.in	246 755	duży test poprawnościowy
pla7b.in	224 023	duży test wydajnościowy
pla8a.in	240 497	duży test poprawnościowy
pla8b.in	250 000	duży test wydajnościowy
pla9a.in	243 763	duży test poprawnościowy
pla9b.in	230 031	duży test wydajnościowy
pla10a.in	247 951	duży test poprawnościowy
pla10b.in	249 089	duży test wydajnościowy